Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2006-030 April 4, 2006

Robust Execution of Bipedal Walking Tasks From Biomechanical Principles

Andreas Hofmann

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Robust Execution of Bipedal Walking Tasks From Biomechanical Principles

by

Andreas G. Hofmann

B.S., E.E.C.S, Massachusetts Institute of Technology, 1982
M.E., Electrical Engineering, Rennselaer Polytechnic Institute, 1985

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy
at the
Massachusetts Institute of Technology
January, 2006

© 2006 Massachusetts Institute of Technology. All rights reserved.

Author………………………………………………………………………………………

Department of Electrical Engineering and Computer Science
November 28, 2005

Certified by…………………………………………………………………………………

Brian C. Williams
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by…………………………………………………………………………………

Steven G. Massaquoi
Assistant Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by…………………………………………………………………………………...

Arthur C. Smith
Chairman, Committee on Graduate Students
Department of Electrical Engineering and Computer Science

Robust Execution of Bipedal Walking Tasks From Biomechanical Principles

Andreas Hofmann

Submitted to the Department of Electrical Engineering and Computer Science on December 1, 2005, in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Effective use of robots in unstructured environments requires that they have sufficient autonomy and agility to execute task-level commands successfully. A challenging example of such a robot is a bipedal walking machine. Such a robot should be able to walk to a particular location within a particular time, while observing foot placement constraints, and avoiding a fall, if this is physically possible. Although stable walking machines have been built, the problem of task-level control, where the tasks have stringent state-space and temporal requirements, and where significant disturbances may occur, has not been studied extensively.

This thesis addresses this problem through three objectives. The first is to devise a plan specification where task requirements are expressed in a qualitative form that provides for execution flexibility. The second is to develop a task-level executive that accepts such a plan, and outputs a sequence of control actions that result in successful plan execution. The third is to provide this executive with disturbance handling ability.

Development of such an executive is challenging because the biped is highly nonlinear and has limited actuation due to its limited base of support. We address these challenges with three key innovations. To address the nonlinearity, we develop a *dynamic virtual model controller* to linearize the biped, and thus, provide an abstracted biped that is easier to control. The controller is model-based, but uses a sliding control technique to compensate for model inaccuracy. To address the under-actuation, our system generates *flow tubes*, which define valid operating regions in the abstracted biped. The flow tubes represent sets of state trajectories that take into account dynamic limitations due to under-actuation, and also satisfy plan requirements. The executive keeps trajectories in the flow tubes by adjusting a small number of control parameters for key state variables in the abstracted biped, such as center of mass. Additionally, our system uses a novel strategy that employs angular momentum to enhance translational controllability of the system's center of mass.

We evaluate our approach using a high-fidelity biped simulation. Tests include walking with foot-placement constraints, kicking a soccer ball, and disturbance recovery.

Thesis Supervisor: Brian C. Williams
Title: Associate Professor of Aeronautics and Astronautics

Thesis Supervisor: Steven G. Massaquoi
Title: Assistant Professor of Electrical Engineering and Computer Science

Acknowledgements

Over the past five years, I have had the fortunate opportunity to pursue an investigation of a topic that has interested me my entire professional life: control of robotic bipeds in unstructured environments. This exciting area of research draws from a number of different fields including robotics, bipedal locomotion, task-level plan execution, nonlinear control, and biomechanics. I have learned new skills in each of these fields, and have combined and integrated them in novel ways in order to accomplish my research objectives. The results of this effort are described in this thesis.

Writing a Ph.D. thesis is an intensely personal effort, in which, the outcome of the investigations is not clear at the outset. It requires patience, and acceptance that not all experiments will turn out as expected. Doubts associated with the uncertainties of open-ended research must be confronted. During this effort, I have experienced the despair of taking a wrong turn in my research, and the intense thrill that accompanies a success after a series of wrong turns. I have also found that, while this is a personal effort, it is also a collaboration between the student, and many others that have an interest in the thesis, and in the student's success. I would like to thank the following people who helped me in this process.

I would like to first thank the members of my thesis committee: Brian Williams, for teaching me about flexible plan execution systems, for his passion for good writing, and for funding me, even though it wasn't clear, at first, how our research interests would coincide; Steve Massaquoi, for teaching me about the biological basis of movement, for his passion for knowledge, and his ability to focus on gaining knowledge amidst the distractions of academic life; and Hugh Herr, for teaching me about biomechanics, and for using his entrepreneurial skills and personal life experiences to build a large organization involved in human locomotion research. I thank Gill Pratt, my original advisor, who asked me to come back to MIT for graduate school, and who has stayed with me throughout this process, for his depth of understanding and wise counsel; and Jovan Popovic, for his inspiring research on space-time optimizations, and his advice about giving presentations.

Besides the members of my thesis committee, there are many others at MIT who have helped me with the work described in this thesis. I would like to thank Randy Davis, for his advice about the graduate school process, and for the interesting and probing discussions about my work, Marko Popovic, with whom I collaborated on a number of papers, Bruce Deffenbaugh, for our lunch time discussions about MIT, sailing, and life, and Edward Fredkin, my undergraduate advisor, for his continued support, and for our interesting discussions on the topic of tightrope walking. Others who have helped me include Jerry Pratt, who showed me how to use the MIT Leg Lab's simulations and other technology, Russ Tedrake, with whom I've had many discussions regarding legged locomotion, Thomas Leaute, who gave me numerous comments on my thesis, and John Stedl, who's own thesis gave me a better understanding of temporally flexible plan execution. I would also like to thank Martin Sachenbacher, Paul Robertson, Marilyn Pierce, and Marcia Davidson for their advice and support.

Besides people directly connected to MIT, there have been many others who have helped me during my time in graduate school. I would like to thank my mother, Rosemarie, for encouraging me to go back to graduate school, for her tireless support throughout this time, and for her understanding when I was having difficulties. I thank my father, Arno, who always gave me wise and insightful counsel, even though he was far away, and my brother, Markus, for his philosophical perspective. I wish him luck with his own graduate studies in religion. I would like to thank Ann, for being supportive of my academic efforts, even during our divorce. I thank my son, Christopher, for his interest in my activities, and for his wry sense of humor, and my daughter, Francie, for her maturity and responsible advice on all matters.

I thank Verena, for her companionship, and for coming into my life, and bringing me happiness, when I least expected it. I would like to thank Nadya for showing me how art and science can be combined in creative and elegant ways, and for her confidence in me. I thank Cindy for our political discussions, and look forward to many more.

Contents

1 Introduction…12
1.1 Motivation…16
1.1.1 Demand…17
1.1.2 Technology Drivers…18
1.2 Problem Statement…19
1.2.1 Specification of Task Goals through Qualitative State Plan…20
1.2.2 Execution of Qualitative State Plan…24
1.3 Challenge…26
1.3.1 Nonlinearity, High Dimensionality, and Tight Coupling…26
1.3.2 Dynamic and Actuation Limits…27
1.3.3 Inherent Sensitivity to Balance Disturbances…28
1.4 Approach and Innovations…29
1.4.1 Dynamic Virtual Model Controller…31
1.4.2 Hybrid Task-level Executive and Flow Tube Trajectories…34
1.4.3 Balance Enhancement by Generating Angular Momentum…39
1.4.4 Summary of Benefits…42
1.5 Experiments…43
1.6 Roadmap…44

2 Background…45
2.1 Control of Walking Bipeds…45
2.1.1 The ZMP Control Method…46
2.1.2 Stability Analysis and Control Design using Poincare Return Maps…50
2.1.3 Joint Trajectory Planning Methods…51
2.1.4 Virtual Model Control Methods…53
2.2 Plan Execution for Hybrid Systems…57
2.2.1 Plan Execution for Discrete State Systems…59
2.2.2 Execution of Temporally Flexible Plans in Discrete Activity Systems…60
2.2.3 Model-based Plan Executives for Hybrid Systems…67
2.2.4 Plan Compilation using Flow Tubes…69
2.3 Biomechanical Analysis…74
2.4 Summary of Limitations of Previous Work…75

3. Biomechanical Analysis of Balance Requirements and Constraints…77
3.1 Clues from Human Walking Trials…82
3.1.1 Motivation for human walking trials: determination of the tightness of angular momentum conservation…82
3.1.2 Human Walking Trial Data Collection and Analysis…84
3.1.2 Results on Conservation of Angular Momentum and Relation between CM and ZMP…86
3.1.4 Prediction of Horizontal Forces…89

3.1.5 Non-conservation of Angular Momentum and the
Zero Torque Center of Pressure……………………………………………92
3.2 Enhancing Balance Control
Through Use of Non-Contact Limb Movement……………………………...96
3.2.1 Simplified 2-link Model……………………………………………………97
3.2.2 PD Controller for the Simplified 2-link Model…………………………102
3.3 Disturbance Metrics and Classification……………………………………...109
3.3.1 FRI Constraint…………………………………………………………110
3.3.2 Balance Control Inputs and Outputs……………………………………114
3.3.3 Disturbance Metrics……………………………………………………117
3.3.4 Definition of Loss of Balance Control…………………………………118
3.3.5 Disturbance Classification……………………………………………...119
3.3.6 Disturbance Handling…………………………………………………..121

4 Hybrid Task-Level Executive…………………………………………………..123
4.1 Overview of Problem Solved by Hybrid Executive…………………………124
4.2 Hybrid Executive Approach…………………………………………………126
4.2.1 Relation to Activity Plan Execution……………………………………127
4.2.2 Efficient Plan Execution through Compilation…………………………129
4.2.3 Summary of Key Innovations…………………………………………..132
4.2.4 Roadmap………………………………………………………………..134
4.3 Linear Virtual Element Abstraction…………………………………………135
4.4 Qualitative State Plan………………………………………………………...139
4.4.1 Qualitative State Plan Definition………………………………………145
4.4.2 Problem Solved by The Hybrid Executive……………………………..147

5 Qualitative Control Plan………………………………………………………..148
5.1 Requirements of the Qualitative Control Plan………………………………148
5.1.1 Flow Tube Representation
Must Include Only Feasible Trajectories………………………………149
5.1.2 Flow Tube Must Represent Goal Region Explicitly……………………152
5.1.3 Flow Tube Goal Region is Subset of Successor's Initial Region………153
5.1.4 Flow Tube Must Represent Initial Region Explicitly…………………..155
5.1.5 Requirements for Representations for Flexible Durations……………...155
5.1.6 Requirements to Support Dispatcher Efficiency………………………..160
5.1.7 Requirements for Temporal Constraint Representation………………...160
5.2 Challenges for Qualitative Control Plan Representation……………………161
5.3 Qualitative Control Plan Approach…………………………………………..162
5.3.1 Flow Tube Representation Using Goal Region and Duration…………162
5.3.2 Flow Tube Representation Including Rectangular Initial Region……...163
5.3.3 Flow Tube Representation for Flexible Duration………………………164
5.3.4 Example Flow Tubes for QSP…………………………………………165
5.4 Qualitative Control Plan Definition…………………………………………167
5.4.1 Structure of a QCP……………………………………………………...167
5.4.2 Correct QCP for a QSP…………………………………………………169

5.4.3 Controllable and Temporal Dispatchability of a QCP……………………170
5.4.4 Successful Execution of a QSP using a Correct QCP…………………..176
5.4.5 Disturbance Definitions……………………………………………...178

6 Hybrid Dispatcher……………………………………………………………186
6.1 Dispatcher Requirements……………………………………………………186
6.2 Dispatcher Approach………………………………………………………...189
6.2.1 Initialization………………………………………………………………189
6.2.2 Monitoring……………………………………………………………...190
6.2.3 Transition…………………………………………………………………193
6.3 Hybrid Dispatcher Algorithm………………………………………………..193
6.3.1 Dispatcher Initialization and Execution Window Propagation…………194
6.3.2 Dispatch Event and Initialize Event……………………………………197
6.3.3 SetControl……………………………………………………………………198
6.3.4 Monitor……………………………………………………………………203
6.3.5 Transition…………………………………………………………………206
6.3.6 Example Execution………………………………………………………..207
6.3.7 Algorithm Complexity Analysis………………………………………..216

7 Plan Compiler…………………………………………………………………..218
7.1 Plan Compiler Problem………………………………………………………219
7.2 Flow Tube Computation for Single Activity using
Two-spike Control Law……………………………………………………...220
7.2.1 Flow Tube Approximation Parameters…………………………………220
7.2.2 Two-spike Control Law………………………………………………...221
7.2.3 Trajectories Representing Duration Bounds……………………………222
7.2.4 GFT and GST for Two Spike Control Law……………………………224
7.2.5 Optimality of Initial Region Defined by
GFT and GST for Two Spike Control Law……………………………229
7.2.6 Trade-off Between Initial Region Size and Controllable Duration…….233
7.3 Flow Tube Computation for Single Activity using PD Control Law………..235
7.3.1 Similarity of Two-Spike and PD Control Laws………………………..236
7.3.2 GFT and GST for PD Control Law……………………………………237
7.3.3 Actuation Constraints for PD Control Law……………………………243
7.4 Plan Compiler Algorithm……………………………………………………245
7.4.1 Satisfying Controllability Requirements………………………………245
7.4.2 Satisfying Temporal Dispatchability Requirements……………………251

8 Dynamic Virtual Model Controller……………………………………………257
8.1 Detailed Humanoid Simulation………………………………………………260
8.2 Closed-Loop Control Rule Representation and Derivation…………………265
8.2.1 Feedback Linearization of the Biped Plant……………………………..267
8.2.2 Multivariable Optimal Controller………………………………………276
8.2.3 Sliding Control Framework……………………………………………279
8.3 Results…………………………………………………………………………285
8.3.1 Forward Disturbance on Level Ground………………………………...286

8.3.2 Lateral Disturbance on Level Ground……………………………………..287
8.3.3 Forward Disturbance on Podium………………………………………289
8.3.4 Lateral Disturbance on Podium………………………………………...290
8.3.5 Adjusting Movement Preferences………………………………………292
8.3.6 Effect of Omitting Joint Limit Constraints……………………………..293
8.4 Discussion…………………………………………………………………293

9 Results……………………………………………………………………..296
9.1 Medium Speed Walking on Firm, Level Terrain……………………………297
9.1.1 Input QSP………………………………………………………………298
9.1.2 QCP……………………………………………………………………..301
9.1.3 Medium Speed Walking Execution……………………………………301
9.2 Slow and Fast Walking on Firm, Level Terrain……………………………..307
9.3 Lateral Push Disturbances……………………………………………………310
9.4 Irregular Foot Placement……………………………………………………..315
9.5 Kicking a Soccer Ball…………………………………………………………..317
9.6 Disturbance Recovery Using the Moment Strategy…………………………317
9.7 Walking on Soft or Slippery Ground…………………………………………...323
9.8 Completeness of Flow Tube Approximation………………………………...326

10 Discussion and Future Work……………………………………………………328
10.1 Completeness of Flow Tube Approximation………………………………...328
10.1.1 Multiple Initial Regions for Flexible-Duration Flow Tubes……………329
10.1.2 Initial Region Representation…………………………………………..333
10.2 Incremental Adjustment of Flow Tubes……………………………………..335
10.3 Learning…………………………………………………………………………...340
10.4 Detailed Comparison with Trial Data………………………………………..343
10.5 Biological Models……………………………………………………………344
10.6 Implementation on a Real Biped……………………………………………..345
10.7 Conclusion………………………………………………………………………...346

Bibliography………………………………………………………………………348

Appendix A – Homogeneous Transforms……………………………………………..353
1 Translation Transformations………………………………………………………354
2 Rotation Transformations…………………………………………………………355
3 General Translation and Rotation Transformations…………………………….356

Appendix B – Jacobian Computation………………………………………………..357
1 Differential Relationships and Computation of Jacobian………………………357
2 Simple Manipulator Jacobian……………………………………………………..361

Appendix C – Computation of Rotational Part of Jacobian and Hessian………………366
1 Rotational Part of Jacobian………………………………………………………..366
1.1 Orientation Representation Conventions……………………………………367
1.2 Conversion Between Angular Velocity Representations……………………368

1.3 Rotational Part of Jacobian In Terms of Angle Derivatives…………………369
2 Rotational Part of Hessian…………………………………………………...369
2.1 Spatial Acceleration Computations……………………………………..369

Appendix D – Introduction to Sliding Control…………………………………………374
1 Motivation and Background…………………………………………………374
2 Sliding Surfaces…………………………………………………………………...374
2.2 Intuitive Basis of Sliding Control…………………………………………374
2.3 Controlling s……………………………………………………………………377

Appendix E - Balance Recovery Through Stepping……………………………………381
1.1 Virtual Leg Model……………………………………………………………382
1.2 Stability Analysis for Fixed Leg Length Stepping……………………………..383
1.3 Model with Extendable Legs…………………………………………………...390
1.4 Model with Foot……………………………………………………………..396
1.5 What is the best foot placement?………………………………………………399

Appendix F – Proofs of Lemmas and Theorems………………………………………400

1 Introduction

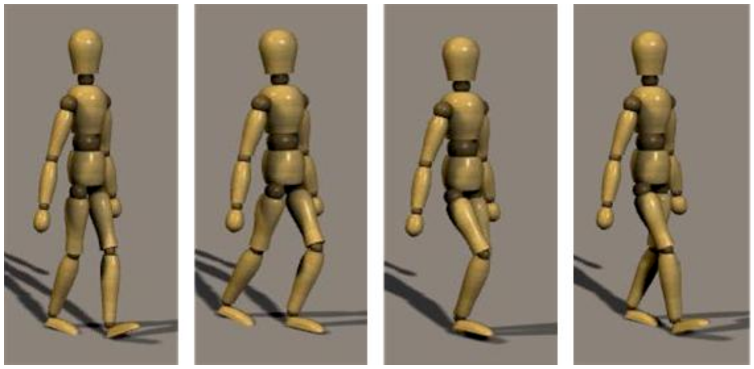
Effective use of autonomous robots in unstructured human environments requires that they have sufficient autonomy to perform useful tasks independently, have sufficient size, strength, and speed to accomplish these tasks in a timely manner, and that they operate robustly and safely in the presence of disturbances. These requirements are more challenging than the ones for today's factory robots, which are stationary, work in very restricted environments, and have very limited autonomy.

A particularly challenging example of an autonomous robot in an unstructured environment is a bipedal walking machine, as shown in Fig. 1.1a. An example task for such a system is to walk to a moving soccer ball and kick it, as shown in Fig. 1.1b. Stepping movement must be synchronized with ball movement so that the kick happens when the ball is close enough. More generally, such tasks require that the biped be in the right location at an acceptable time. This implies spatial and temporal constraints for such tasks. There are also important dynamic balance constraints that limit the kinds of movements the biped may make without falling down.

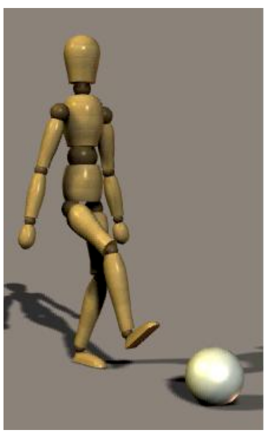
If the system encounters a force disturbance while performing a task, it will have to compensate in some way in order to satisfy these constraints. The disturbance may cause a delay, allowing another player to kick the ball, or it may interfere with movement synchronization. For example, a trip, shown in Fig. 1.1c causes disruption of synchronization between the stepping foot, and the overall forward movement of the system's *center of mass*.

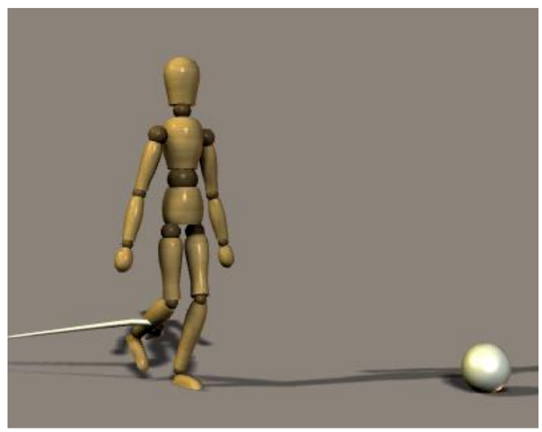
A second example task is walking on a constrained foot path, such as stones across a brook, as shown in Fig. 1.2a, or on a balance beam as shown in Fig. 1.2b. As with the soccer ball example, this task has spatial, temporal, and dynamic constraints, but in this case, the spatial constraints are more stringent; the biped must reach its goal using foot placements that are precisely constrained.

Fig. 1.3 shows a biped walking over blocks that constrain foot placement in a similar manner. When foot placement is constrained, the stepping pattern can't be changed arbitrarily to compensate for a disturbance. For example, if a lateral push disturbance occurs, rather than stepping the leg out to the side, other compensating techniques, such as angular movement of the body and swing leg must be used, as shown in Fig. 1.4.



a.





b. c.

Fig. 1.1 – a. Walking biped, b. kicking soccer ball, c. trip disturbance

In these examples, and others like them, the key challenge is to move a complex, dynamic system to the right place, at the right time, despite actuation limits, and despite disturbances. The system should be able to recover from disturbances such as slips, trips, pushes, and ground contact instability due to soft terrain, even when foot placement is constrained.

Accomplishing such challenging tasks requires sophisticated planning and control. Traditional AI-based approaches for such problems use a three-tier architecture involving a planner, an executive, and a skills library, as shown in Fig. 1.5a. The planner generates a task-level plan, for example, a navigation plan for a wheeled robot. The plan may have temporal constraints on achieving waypoints, as shown in the diagram. These take into consideration the velocity limits of the robot. The executive monitors execution, and invokes primitives from the skills library. The planner and executive work together to compensate for disturbances and temporal uncertainty. These systems assume that acceleration limits and detailed dynamics can be ignored, or can be abstracted away in the temporal constraints. For example, if a task for a wheeled robot takes between 20 and 30 seconds, as shown in Fig. 1.5a, and if the wheeled robot can accelerate from a stop to maximum speed in less than 1 second, then the acceleration limit can be ignored. It is adequate for the planner and executive to consider only velocity limits, and assume that accelerations happen instantaneously. The temporal constraint of 20 to 30 seconds shown in Fig. 1.5a is based on such a velocity limit.

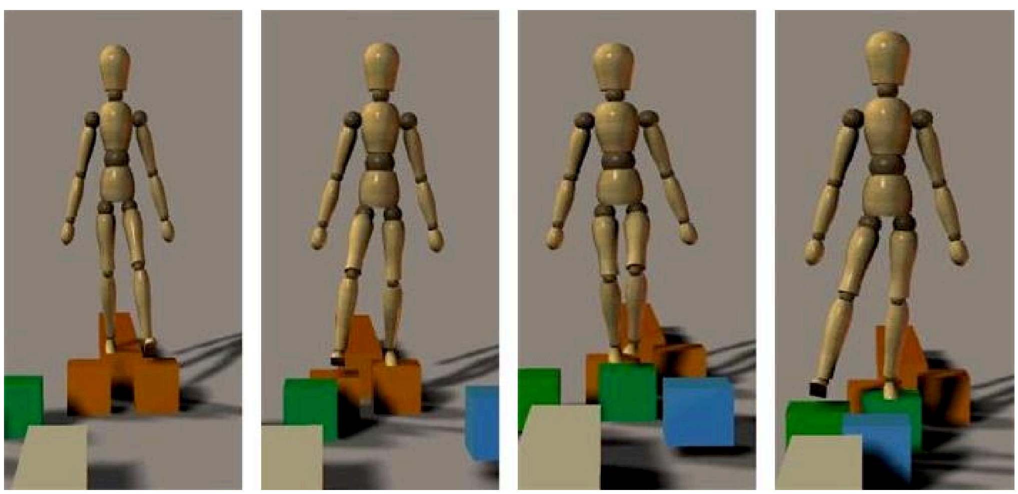


Fig. 1.3 – Dynamic walking with foot placement constraints.

For a biped, acceleration limits cannot be ignored. A biped achieves acceleration of its center of mass through force against the ground. Because the biped's contact with the ground is not firm, there is a limit to this acceleration. This limit must be taken into consideration for tasks requiring agility, like kicking a soccer ball, or walking over difficult terrain quickly. For such tasks, the time delays resulting from acceleration limits are significant compared with the overall time of executing the tasks. For example, if the biped starts in a fully stopped state, and has to get to a location 2 meters away in 2 seconds in order to kick a soccer ball, then if the acceleration limit is 0.5 m/s/s, and the

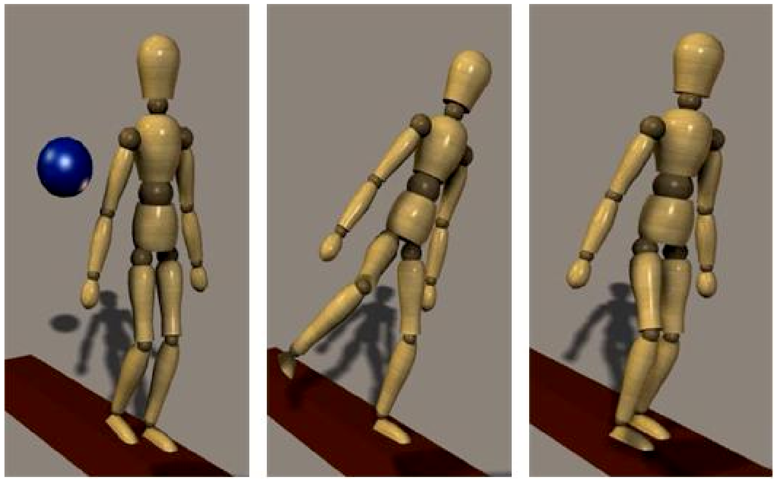
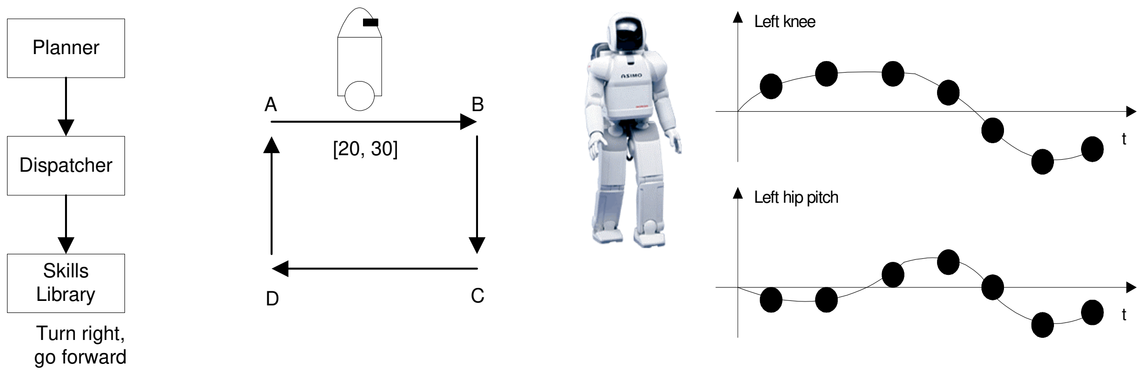


Fig. 1.4 – Compensating for lateral push disturbance using angular movement of torso and swing leg.

velocity limit is 1 m/s, the time delays due to the acceleration limit are more important than those due to the velocity limit in determining whether the task can be performed successfully. Since, for constant acceleration $distance = 0.5 \times acceleration \times time^2$, if the biped moves with the maximum acceleration of 0.5 m/s/s, it will just barely be able to traverse the required distance of 2 meters in 2 seconds. Note that with this maximum acceleration, the maximum velocity is achieved only at the very end of the interval, so the maximum velocity limit is not a determining factor for task success. Thus, for this type of bipedal walking task, detailed dynamics cannot be ignored.

Previous approaches to control of robots like Asimo [Hirai et al., 1998] do take dynamics into account. These approaches generate detailed joint trajectories offline using dynamic optimization algorithms that observe dynamic limitations, as shown in Fig. 1.5b. These trajectories are then tracked using simple high-impedance PD control laws. However, this approach is not very robust to disturbances, since it depends on close tracking of the reference trajectories. If a disturbance occurs, tracking error can easily become too large due to actuation limits related to imperfect ground contact, and the system can lose its balance [Pratt and Tedrake, 2005].



a. b.

Fig. 1.5 – a. Traditional AI-based planning and control architecture; b. detailed dynamic optimization of joint trajectories.

In this thesis, we address the class of problems that require movement of a dynamic bipedal system according to stringent state-space and temporal requirements, despite actuation limits and disturbances. This class of problems, of which soccer ball kicking and agile traversal of difficult terrain are examples, is inadequately addressed by the three-tier AI-based approach and by the high-impedance robotic control approaches. What is needed is a system that combines the task-level plan execution and robustness provided by the three-tier approach, with the sensitivity to dynamics provided by the robotic control approach. We present such a system in this thesis.

We next discuss, in Section 1.1, reasons for studying this class of problems. This is followed by a more detailed statement of the problem being solved, in Section 1.2. Section 1.3 presents challenges to solving this problem. We then discuss how we address these challenges in Section 1.4, and summarize key innovations of our approach in Section 1.5. In Section 1.6, we introduce experiments used to validate our approach. We conclude this introductory chapter with a roadmap for the rest of the thesis, in Section 1.7.

1.1 Motivation

Investigation of control of walking machines in unstructured environments is motivated by both the anticipated demand for such machines and recent technology advances that make them possible.

1.1.1 Demand

The retirement-age population in America is growing dramatically. As shown in Fig. 1.6, the number of people over 65 is expected to grow by almost 70 million over the next 30 years.

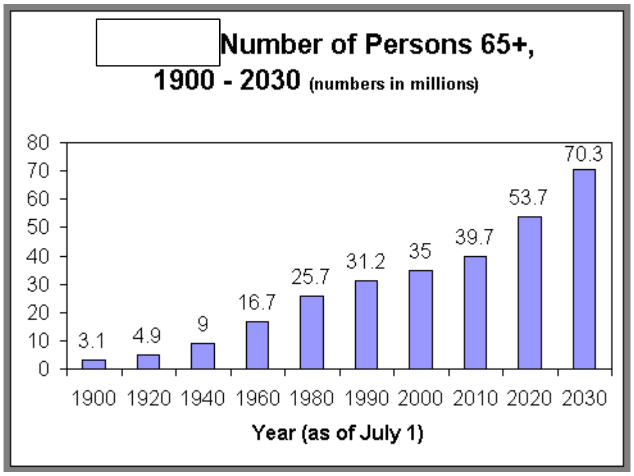


Fig. 1.6 - Growth of retirement-age population in America

(Source: Administration on Aging, http://www.aoa.dhhs.gov/aoa/stats/profile/2.html)

A significant percentage of the current retirement-age population is disabled. In the 65 – 75 age group, approximately 50% have some disability. In the over 75 age group, this percentage goes up to over 70%. Of the people with disabilities, approximately 49% have disabilities related to arthritis, and approximately 18% have orthopedic impairments. Such disabilities have an important limiting effect on these people's daily activities. It is reasonable to expect that these disability percentages will not decline precipitously over the next 30 years. This, combined with the increased retirement-age population trend indicates that over the next few decades, there will be a significant number of people that will have difficulty performing ordinary, daily activities.

The trends are similar in other countries in developed regions of the world. In Japan, this looming problem has been the motivation for much of that country's significant efforts in humanoid robotics, exoskeletons, and other assistive devices; a primary goal of these efforts is to develop robots that can operate in unstructured, domestic environments, and can provide assistance to disabled, elderly people.

Bipedal configurations have unique characteristics that provide significant advantages and disadvantages over quadruped or wheeled robots. Because bipeds have only two legs, their support base is naturally constrained, allowing them to operate in environments where support base space is limited. However, humanoid bipeds have a high center of mass. The combination of limited support base and high center of mass presents a challenge in terms of balance control in that such a system is inherently less stable than a quadrupedal or four-wheeled configuration.

Balance control is essential both for autonomous legged assistive robots, and for a variety of assistive devices, including powered exoskeletons that provide locomotion to the disabled. For such systems, preventing a fall is of paramount importance. An autonomous robot that falls may damage itself, or may hurt a human in its environment. In the case of an exoskeleton, a fall implies that the human wearer of the exoskeleton has fallen. Thus, a bipedal walking machine should avoid falling, if at all physically possible, even if it encounters a significant disturbance.

1.1.2 Technology Drivers

A number of recent advances in technology, when combined appropriately, will enable development of walking machines suitable for the kinds of applications described in the previous section. One important advance is autonomous task-level planning and control systems that can execute temporally and spatially flexible plans [Kim, 2001; Walcott, 2003]. Such systems support task-level commands, and therefore, allow for a high degree of autonomy. They are able to guarantee successful execution even when there is uncertainty due to the possibility of disturbances, as long as this uncertainty is bounded [Stedl, 2004]. A second important advance is the development of sophisticated nonlinear control algorithms, capable of linearizing and simplifying control of complex nonlinear plants, such as bipedal walking machines. Two particularly useful techniques, in this context, are feedback linearization and sliding control [Slotine, 1991].

A third important advance is the development of a new class of actuators that are compliant to disturbances, and thus are more suitable for use in unstructured environments than traditional motors. This class of actuators, called *series-elastic* actuators [Pratt and Williamson, 1995] utilize an elastic component between the motor and the load, resulting in some reduction in bandwidth, but vastly improved impedance control and disturbance response characteristics. A fourth important advance is an improved understanding of how humans balance [Popovic et al., 2004a]. This improvement in understanding has come about by a combination of analysis of human motion data, and analysis of biomechanical balance models. Finally, the continuing increases in raw computing power make it increasingly feasible to use advanced planning and control algorithms in real-time applications.

1.2 Problem Statement

We seek to develop a robust plan execution system capable of guiding a robotic biped through a series of walking task goals, in the presence of disturbances. The system must understand commands at the *task level*; it must take as input a high-level specification of where it should be, and by what time, and then automatically figure out the details of how to move to accomplish these goals. It should also be able to automatically detect whether a task that it is given can be accomplished in the allotted time, and should warn the human operator when this is not the case. If a disturbance occurs during execution of the task, the system should attempt to compensate in order to avoid a fall, and should still try to complete the task on time.

To develop such a system, we seek answers to the following questions.

- How should walking task goals be expressed?
- How do these goals interact?
- What are the fundamental requirements for stability and for achieving these goals?
- What kinds of disturbances may occur while executing walking tasks?
- How do these disturbances interfere with the fundamental requirements for stability and goal achievement?
- What fundamental balance strategies can bipeds use?
- How should these balance strategies be combined?

Another important general consideration for autonomous bipeds is that they move in a manner that is safe to surrounding people, and to the environment. This will ultimately have to be addressed, but, it is not the topic of this thesis. Nevertheless, the effective management of the control problems addressed herein provide a valuable foundation for future safe bipedal locomotion control.

We now introduce how walking task goals are represented in a plan, and then pose the problem as one of executing such a plan successfully.

1.2.1 Specification of Task Goals through Qualitative State Plan

There are two basic kinds of task goals: state space and temporal. We specify state space goals as requirements on values of key position and velocity variables that summarize the state of the system. The goals are expressed as constraints that require these values to be within particular desired regions of state-space. Key variables for state space goals include the system's center of mass position and velocity, and foot placement positions. For example, the goal that the biped be at a particular location, such as in front of a soccer ball, or at the end of a path, is conveniently expressed as a requirement that the center of mass be in a region that defines an appropriate vicinity of the location. The requirement that the biped must walk along a constrained path, or that the feet must be in an appropriate position for kicking a soccer ball is expressed using constraints on foot placement positions.

Temporal constraints arise from two fundamentally different sources: actuation and dynamic limitations of the biped itself, and externally imposed goals on task completion times. Dynamic limitations arise from the fact that the biped is a complex, articulated mechanism, where movement is achieved by applying torques to the joints, which accelerate the segments in the mechanism. Acceleration is limited by segment inertias, limitations of the joint actuators, and by the fact that the support base on the ground is limited.

Externally imposed temporal goals are useful for specifying that the system be at a goal location at an acceptable time. The two kinds of temporal constraints are often in conflict since externally imposed goals typically specify upper bounds on task completion times, and dynamic limitations imply lower bounds, that is, a minimum time needed for the physical system to perform the task. The system must check that externally imposed temporal constraints are reasonable; that they are consistent with the temporal limitations arising from the biped's dynamics.

Fig. 1.6 shows a specification of such state-space and temporal requirements, expressed as a sequence of *qualitative states*, which forms a *qualitative state plan*. A qualitative state is a region of state space in which all states have a uniform property with respect to the task at hand [Williams, 1984]. For a biped, qualitative states are defined by foot ground contact state. In the first qualitative state, QS1, the biped is in a *double-support* state, where both feet are in contact with the ground, with the left foot being in front of the right. In QS2, the biped is in a *single-support* state, with the left foot in contact with the ground, and the right foot taking a step. In this state, the left foot is called the *stance* or *support* foot, and the right is called the *swing* foot. QS3 is double-support with the right foot in front, QS4 is right single support, and QS5 is a repeat of the first state. Thus, the sequence of qualitative states forms a complete walking gait cycle.

Each qualitative state may specify ranges defining valid operating regions for particular state variables. The foot placement position constraints shown in Fig. 1.6 are examples of such operating region constraints. Each qualitative state may also specify ranges defining goal regions that particular state variables must attain. The center of mass region specification for the last state in Fig. 1.6 is an example of such a goal region constraint. Such operating and goal region constraints are continuous. Thus, a qualitative state is hybrid in that it is defined by continuous state regions, like allowable regions for the center of mass position, as well as by discrete state, like which feet are in contact with the ground.

Transitions from one qualitative state to another are defined by *events*. For example, the transition from double to single support is defined by a *toe-off* event, which is the point where the swing foot lifts off the ground. The transition from single to double support is defined by a *heel-strike* event, which is the point where the swing foot touches the ground after taking a step.

Events represent temporal boundaries that can be restricted by temporal constraints. For example, the temporal constraint in Fig. 1.6 imposes a lower and upper bound on time between the toe-off event of the first qualitative state, and the toe-off event of the last. Since the last qualitative state is a repeat of the first, this temporal constraint defines an allowable time range for completion of a gait cycle.

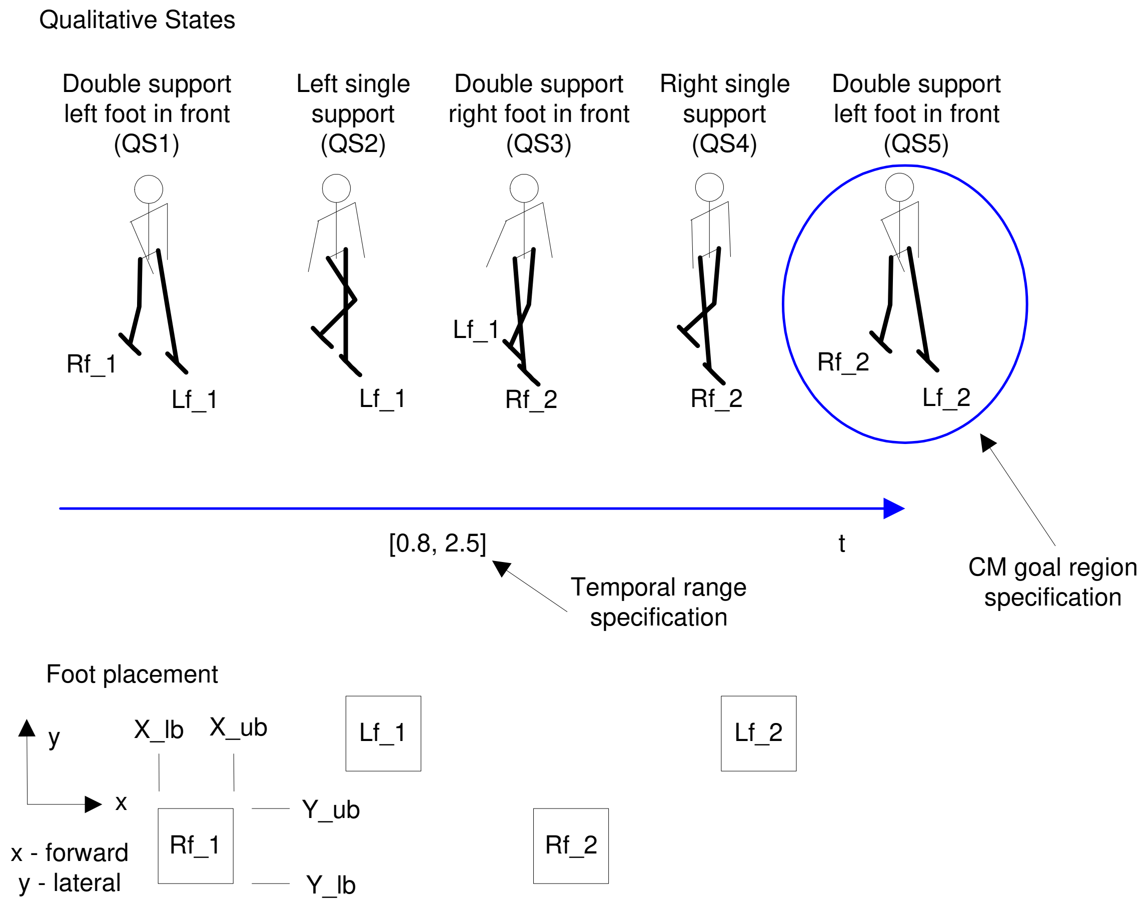


Fig. 1.6 - State plan specifies qualitative poses (single support, double support), but does not specify details (joint angles, velocities). Spatial goal region specified in terms of range of allowable locations for center of mass (CM). Temporal constraint specified as lower bound (0.8 sec.) and upper bound (2.5 sec.) on time to achieve spatial goal region.

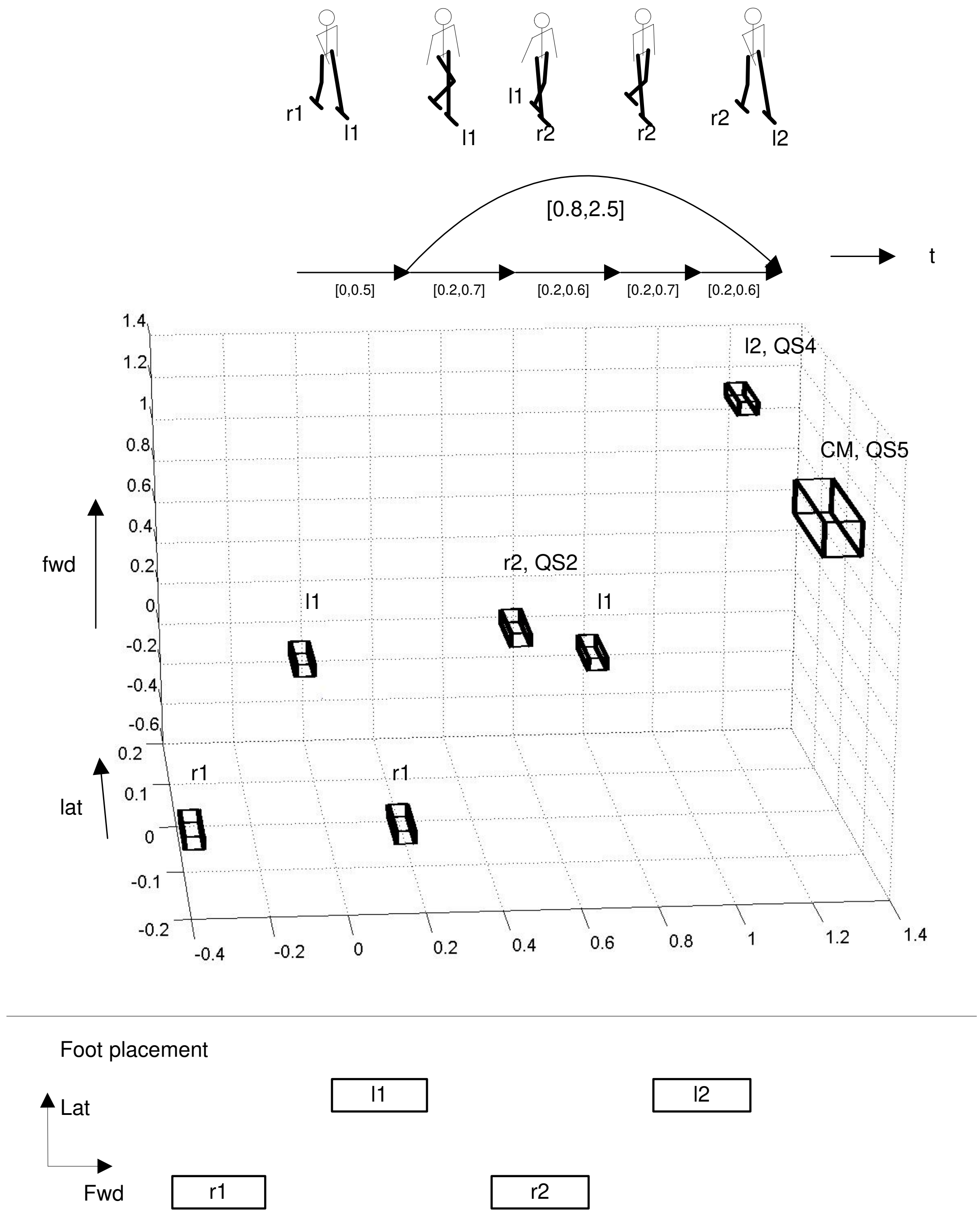


Fig. 1.7 – Goal regions for qualitative state plan from Fig. 1.6. Region r2 represents the goal for the right (stepping) foot, for QS2. Region l2 represents the goal for the left (stepping) foot, for QS4. Region CM represents the goal for the center of mass, for QS5.

Goal regions for qualitative states define requirements for transition from one qualitative state to the next. If the biped is not in a required goal region for a qualitative state when the transition event occurs, then the plan execution has failed. Fig. 1.7 shows a three-dimensional view of forward and lateral position goal regions vs. time for the qualitative state plan shown in Fig. 1.6. For example, in QS2, the left foot is the stance foot, and the right foot is stepping. Region r2 represents the position and temporal goal for the right foot as it completes its step. Similarly, region l2 represents the position and temporal goal for the left foot as it completes its step. Region CM represents the position and temporal goal for the center of mass at the end of QS5, just before right toe-off.

Use of qualitative states with goal regions and temporal constraints is a more natural and convenient way to specify requirements than a detailed set of poses, expressed in terms of joint angles, or a detailed set of control actions, expressed in terms of joint torques. The flexibility inherent in the specification in Figs. 1.6 and 1.7 gives the system room to adjust to disturbances that may occur during execution, while still allowing it to execute the plan successfully.

1.2.2 Execution of Qualitative State Plan

The problem to be solved can now be stated in the following way. Given a qualitative state plan, and given a biped to be controlled, generate a sequence of control actions that result in state trajectories for the biped that satisfy the requirements of the plan. Thus, the generated state trajectories must pass through the goal regions at acceptable times, as shown in Fig. 1.8. The goal regions can be thought of as "hoops" through which the state trajectories must pass.

In generating such control actions, the executive must take into account dynamic and actuation limitations of the biped. These impose additional requirements on the state trajectories, in addition to the ones imposed by the qualitative state plan. The executive must compute these additional requirements automatically, for any given qualitative state plan.

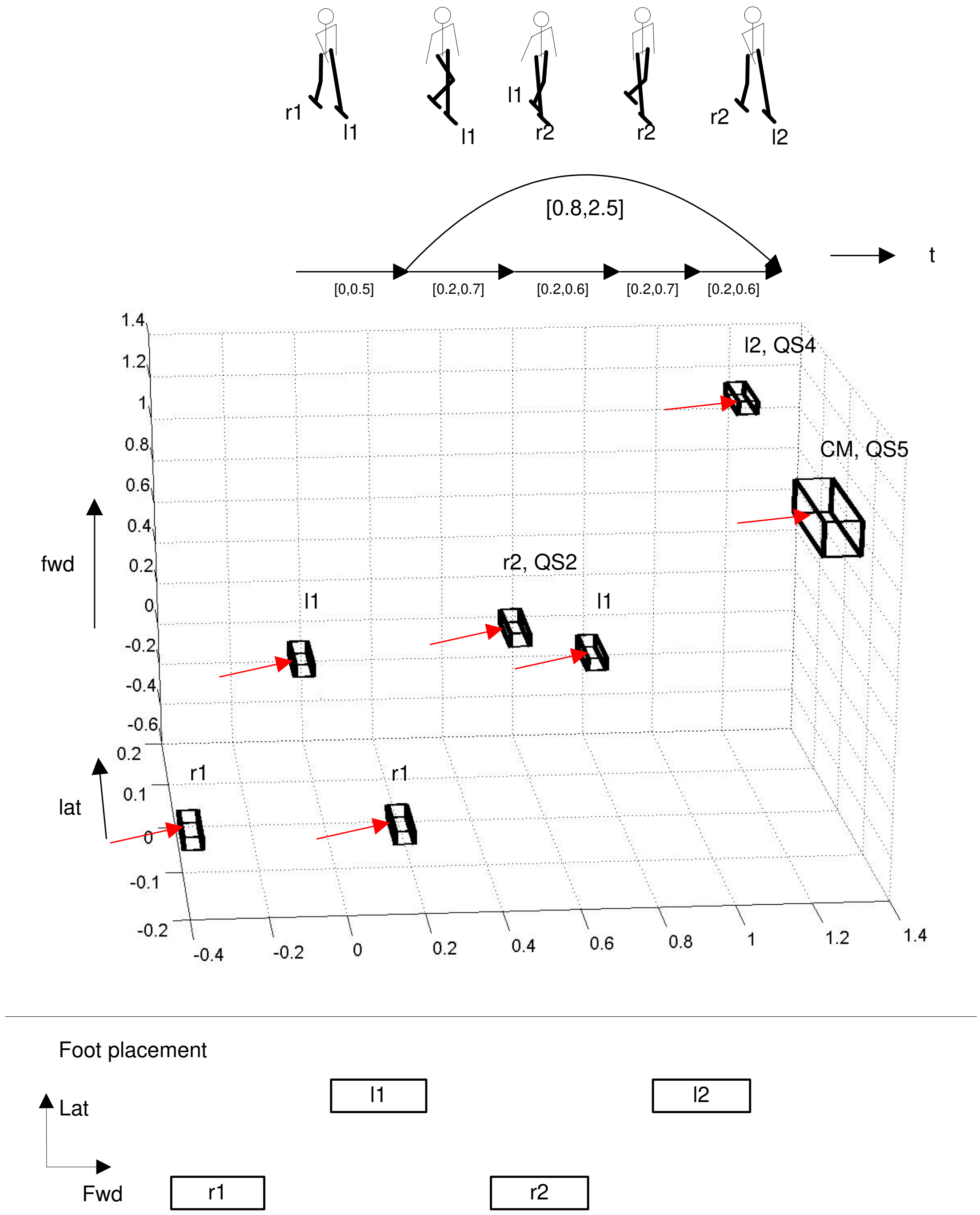


Fig. 1.8 – State trajectories generated by control actions must pass through goal regions at an acceptable time.

Given that the additional requirements can be computed automatically, it is then necessary to investigate how disturbances of various types cause failure to meet the requirements. This leads to an understanding of how disturbances like pushes, slips, and trips cause falls.

To cope with disturbances, we must investigate strategies that humans use for balance, and generalize these to all bipeds with limited support base and high center of mass. In order to combine these balance strategies appropriately, we must combine our understanding of how disturbances interfere with the requirements for successful plan execution, with our understanding of how various balance strategies can be used to restore fulfillment of these requirements.

To summarize, our goal is to develop a plan execution system that, when given a qualitative state plan, and a biped with sufficient size and sufficient actuator strength and speed, computes control actions for the biped such that the plan is executed successfully. The system should generate appropriate compensating control actions in response to disturbances so that the biped does not fall, and so that plan goals are still achieved, if this is physically possible. The system should check whether the desired task specification is feasible, and warn the human operator if it is not.

1.3 Challenges

There are three key challenges to solving the problem stated in the previous section. First, a biped is a high-dimensional, highly-nonlinear, tightly coupled system, so computing control actions that achieve a desired state is a challenging problem. Second, a biped is under-actuated and has significant inertia, so future state evolution is coupled to current state through dynamics that limit acceleration, and the executive must consider how current state and actions may limit achievement of future desired state. Third, a biped has a high center of mass and limited support base on the ground, and is therefore very sensitive to balance disturbances. We next discuss each of these challenges in more detail.

1.3.1 Nonlinearity, High Dimensionality, and Tight Coupling

Bipeds have multiple articulated joints, and therefore, have a large number of degrees of freedom. For example, the biped model used in this thesis has 12 actuated joints, and thus 12 actuated degrees of freedom, along with and an additional 6 un-actuated degrees of freedom representing translational and rotational movement of the torso with respect to the ground. This results in a system with 18 degrees of freedom, 36 state variables, and 12 control input variables. Furthermore, movement dynamics are highly nonlinear and tightly coupled, so computing control actions that achieve a desired state is a challenging problem. A standard dynamic programming approach [Bertsekas, 2005] to such a problem is not possible due to the size of the state space. For example, if a discretization of 10 increments were used for each state space dimension and for each control input, the policy table would have to have 10^{36} entries for each time increment, with each such entry providing 12 control input values.

1.3.2 Dynamic and Actuation Limits

A biped has significant inertia, and limited ability to generate force against the ground. This results in limits on the biped's ability to accelerate its center of mass. Therefore, the executive must take into consideration the coupling between current and future state; it must consider how current state and actions may limit future state evolution, through dynamics that limit acceleration. Thus, the executive must consider a finite horizon of time into the future over which the biped's state evolves.

As explained previously, a standard dynamic programming approach is not feasible. A breadth-first search approach at execution time is also infeasible. For example, a time discretization of 0.05 seconds with a 1 second horizon results in 20 time increments over the entire horizon. Given a discretization of 10 increments for each control input, this results in an exponential expansion of 10^{12} nodes in the search tree for each time increment, resulting in $10^{12\times20}$ nodes at the deepest level of the search.

Acceleration limits due to the biped's dynamics also, potentially, are in conflict with temporal constraints imposed by the qualitative state plan. In particular, the temporal constraints imposed by the qualitative state plan typically impose a maximum time limit on task completion, and thus, encourage fast movement. These constraints are potentially in conflict with dynamic constraints of the system, which limit accelerations due to inertias and actuation limits, and thus, impose minimum time limits on task completion. The executive must balance these competing constraints, and must also detect when the qualitative state plan is infeasible due to unreasonable temporal constraints.

1.3.3 Inherent Sensitivity to Balance Disturbances

The previous two challenges make control of bipedal walking machines difficult even under nominal circumstances. Operation in unstructured environments requires, further, that the control system be able to handle force disturbances of a variety of magnitudes. If such disturbances are severe, the required compensating actions may be fairly complex. The control system must generate these actions in real time in order to avoid a fall.

The extreme sensitivity of a biped to balance disturbances is due to its high center of mass and limited base of support. Because the contact surface of the feet with the ground is limited, particularly in single support, the feet may slip or roll if inappropriate actuation forces are used. To avoid this, lateral force exerted by the feet against the ground must be limited, but this also limits acceleration of the center of mass, and thus, the ability to control its position when there are disturbances. Thus, unlike manipulators, walking machines are under-actuated because they do not have a firm base of support, and therefore, are very sensitive to balance disturbances. They require careful management in order to achieve robust behavior.

A further complicating factor is due to the inherent nature of walking tasks. Bipeds operate in a sequence of discrete modes defined by contact of the feet with the ground. These are the qualitative states described previously. At transitions between these qualitative states, the base of support changes discontinuously. Thus, at toe-off, the base of support is instantly reduced because the biped transitions from double to single support. At heel-strike, the base of support is instantly enlarged because the biped transitions from single to double support. These discontinuous changes in support base imply discontinuous changes in actuation limits. The executive must take these discontinuous changes in actuation limits into account when generating control actions and projecting evolution of state trajectories over the future time horizon.

The alternation between single and double support qualitative states results in a stable limit cycle, for normal dynamic walking. Note, however, that each of the qualitative states does not have a stable equilibrium point. Thus walking is an inherently unstable process, where the system is constantly in a state where it is about to fall. Qualitative state transitions simply defer the fall. This is a direct consequence of the actuation constraints, and the fact that the biped has significant momentum during fast walking.

This means, for example, that the biped cannot stop instantly, in the middle of a gait cycle. Rather, the system must first slow down to exit the limit cycle, coming to a rest in a standing position, a qualitative state that does have a stable equilibrium point. Thus, for such walking plans, the executive must guide the system through a sequence of inherently unstable states to get to a goal state that is stable.

We next discuss our approach to addressing the above described challenges.

1.4 Approach and Innovations

We seek to guide a bipedal walking machine through a sequence of qualitative states so that it achieves a specified task-level goal. This is in contrast to systems that generate walking motions to achieve a stable limit cycle, or that play back very detailed joint trajectories.

To address the difficulty of determining the effect of control actions on biped state, we use a model-based approach, where a model of the biped is used to predict this effect. Thus, we use a *model-based executive* [Williams and Nayak, 1997; Leaute, 2005] to interpret plan goals, monitor biped state, and compute joint torque inputs for the biped, as shown in Fig. 1.9. The executive computes a sequence of joint torques for the biped that results in achievement of each successive qualitative state goal in the sequence, as shown in Fig. 1.10. To keep the biped from falling, the executive uses balance strategies used by humans, which are applicable to bipeds in general.

We address the challenges described in Section 1.3 with three key innovations. To address the first challenge (nonlinearity, high dimensionality, and tight coupling), we linearize and decouple the biped system into a set of independent, linear, single-input single-output second-order systems, resulting in an abstraction of the biped that is easier to control. We accomplish this through a novel controller called a *dynamic virtual model controller* [Hofmann, et. al., 2004], which is introduced in more detail in Section 1.4.1. The linearization and decoupling provided by this controller allows points on the biped to be controlled directly, in a manner similar to the way that a puppeteer controls a marionette.

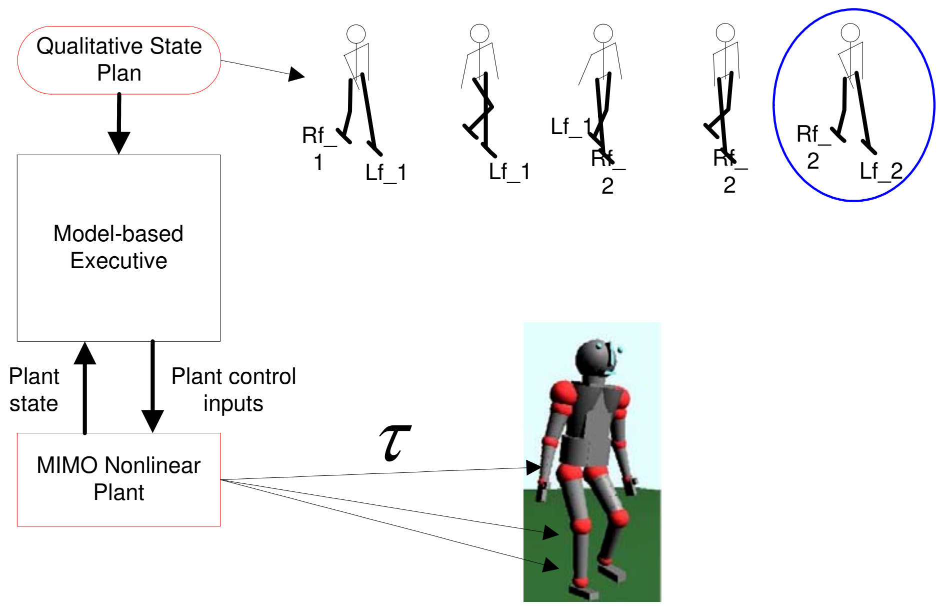


Fig. 1.9 – A model-based executive computes a sequence of joint torques for the biped that results in the achievement of the successive qualitative state goals.

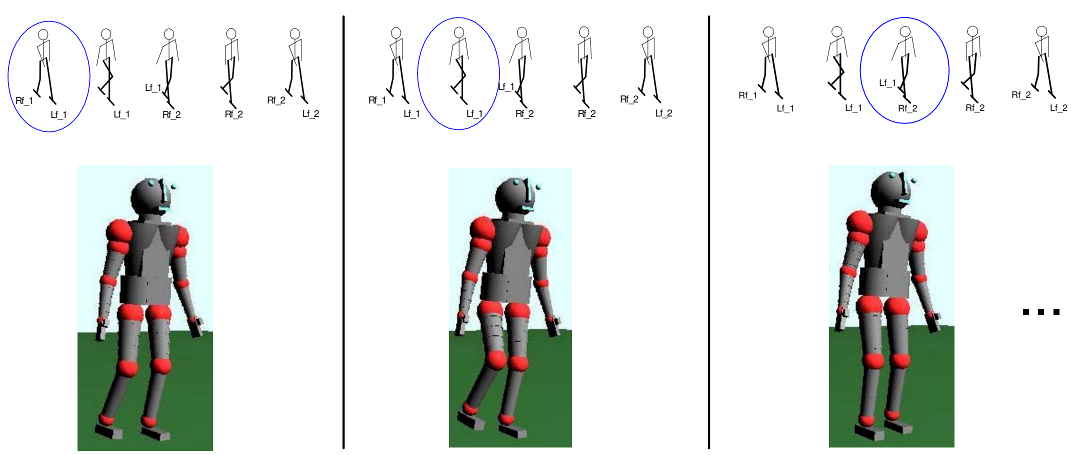


Fig. 1.10 – Execution of a sequence in the qualitative state plan.

To address the second and third challenges (actuation limits and sensitivity to balance disturbances), our executive generates *flow tubes* that define valid operating regions for the state variables and control parameters in the abstracted biped. The flow tubes represent bundles of state trajectories that take into account dynamic limitations due to under-actuation, and also satisfy plan requirements. Off-line generation of these flow tubes represents a pruning of infeasible trajectories, so that the on-line executive can focus on executing the plan by using only trajectories in the flow tubes.

Finally, to address the third challenge, our system uses a novel strategy that employs angular momentum to increase the horizontal force that can be applied to the system's center of mass, and thus, to enhance its balance controllability. This strategy is particularly useful for tasks where foot placement is constrained. We now describe these three innovations in more detail.

1.4.1 Dynamic Virtual Model Controller

Our dynamic virtual model controller is similar, in concept, to a virtual model controller [Pratt et al., 1997]. For both types of controllers, the goal is to provide an abstraction whereby a complex, articulated mechanism is controlled by virtual spring-damper elements attached at key reaction points on the mechanism, as shown in Fig. 1.11.

The key difference between the capabilities of these two types of controllers is that our dynamic virtual model controller takes dynamics into account, while a virtual model controller does not. A virtual model controller uses a Jacobian transformation to translate the desired forces at the reaction points, specified by the virtual elements, into joint torques that produce these forces. This works well for static or slow-moving mechanisms, but can break down as movements become faster because the controller does not take into account the dynamics of the system. Therefore, movement of the reaction point is not necessarily in line with the desired virtual force. In contrast, our dynamic virtual model controller uses a dynamic model to account for the biped's dynamics. This results in a linear system, where reaction points move as if they were simple linear second order systems, controlled by the virtual elements, as shown in Fig. 1.12.

We call the linear system provided by the controller a linear virtual element abstraction. The controller performs a coordinate transformation so that the state variables in this abstraction are workspace state variables, such as center of mass forward position and velocity, which must be controlled to balance the biped, rather than joint space state variables, such as right hip pitch.

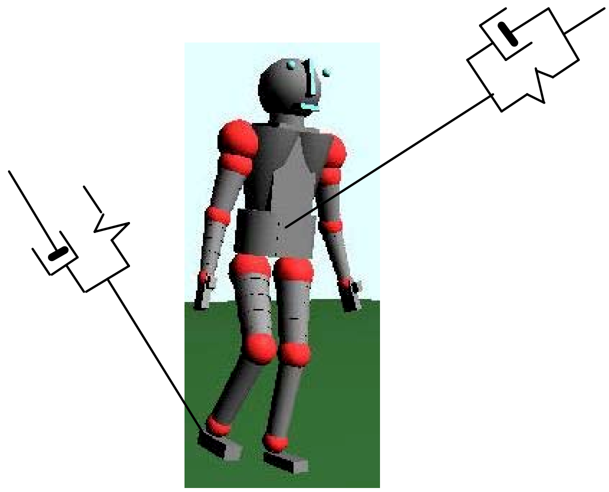


Fig. 1.11 – Virtual model control uses virtual spring-damper elements attached to reaction points allowing the mechanism to be controlled as if it were a puppet.

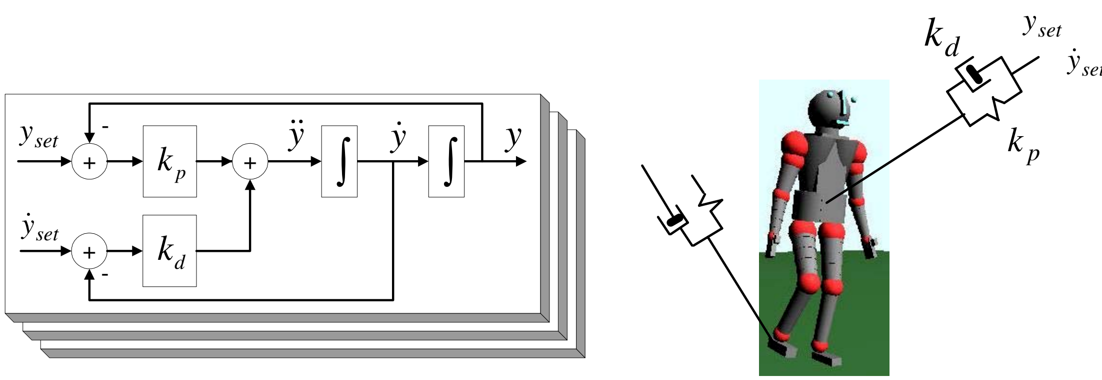


Fig. 1.12 – A dynamic virtual model controller provides a linear abstraction so that the reaction points move as if they were independent, linear second-order systems, controlled by the virtual elements.

Because the dynamic virtual model controller is model-based, the problem of model inaccuracy must be addressed. In order to compensate for this model error, we use a sliding control technique [Slotine, 1991].

The dynamic virtual model controller is a multivariable controller; it tries to achieve multiple goals simultaneously. Sometimes this is not possible. For example, if a situation occurs where desired movement is limited due to actuation constraints, the system becomes over-constrained and some goals must be deferred until the situation improves. To address this problem, our controller incorporates a goal prioritization algorithm that automatically sacrifices lower-priority goals when the system becomes over-constrained in this way. For example, the system may temporarily sacrifice goals of maintaining upright posture in order to achieve balance goals.

The linear virtual element abstraction provided by the controller allows for simple, intuitive specification of desired behavior. Fulfilling such specifications may require sophisticated balancing movements. Consider, for example, the problem of balancing on one foot on a narrow podium as shown in Fig. 1.13. Due to the restricted area of support provided by the podium, even a small disturbance may require complex corrective action involving rotation of the body, and movement of the non-contact leg. Our dynamic virtual model controller automatically computes these coordinated body and limb movements in response to only a single setpoint directive: that the center of mass position be directly above the center of the podium, and that its velocity be 0. In generating these movements, the controller takes into account the current state of the system, but does not try to predict future state explicitly. In particular, it does not require search or dynamic optimization over a future time horizon to choose the best course of action.

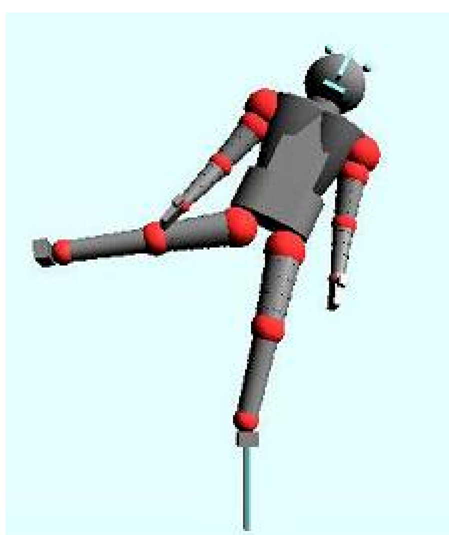


Fig. 1.13 – Balancing on a narrow podium.

Thus, the dynamic virtual model controller is a powerful tool. Note, however, that its range of operation is limited to a particular qualitative state. In Fig. 1.13, it is assumed that the biped will be in left single support, and the center of mass setpoint is a simple constant. To perform walking tasks, setpoint and gain parameters for the controller must be sequenced appropriately, in order to achieve transition through a sequence of qualitative states.

1.4.2 Hybrid Task-level Executive and Flow Tube Trajectories

Due to the dynamic and actuation limits discussed previously, the model-based executive must consider the future evolution of the biped's state over successive qualitative state goals. Thus, to achieve correct plan execution, the model-based executive must generate a control trajectory that satisfies all future goal region and temporal constraints specified in the qualitative state plan. We accomplish this by leveraging the linear virtual element abstraction in a two-part architecture, consisting of a *hybrid task-level executive*, and a dynamic virtual model controller, as shown in Fig. 1.14.

The hybrid executive controls the biped indirectly, by setting control parameters for the dynamic virtual model controller, rather than directly, by generating joint torques for the biped. Thus, it leverages the linear, decoupled abstraction provided by the dynamic virtual model controller so that it need only consider the evolution of independent linear systems, rather than a tightly coupled high dimensional nonlinear system. This allows the biped to be controlled as if it were a puppet, moving in response to movement of the virtual elements.

Now, consider that actuation constraints limit the speed of movement of the virtual elements. Thus, the hybrid executive must plan movement of the virtual elements carefully, taking into consideration consequences of current control actions for achievement of future plan goals.

In order to project the feasible future evolution of the biped's state, the hybrid executive computes *flow tubes* that define valid operating regions in terms of the linear virtual element abstraction. The flow tubes represent bundles of state trajectories that satisfy plan requirements, and also take into account dynamics and actuation limitations.

Fig. 1.15 shows flow tubes for the center of mass for the qualitative state plan in Figs. 1.6 and 1.7. The flow tubes observe the discontinuous changes in actuation constraints due to ground contact events. Once the flow tubes have been computed, the hybrid executive executes the plan by adjusting control parameters in the linear virtual element abstraction in order to keep trajectories within the tubes.

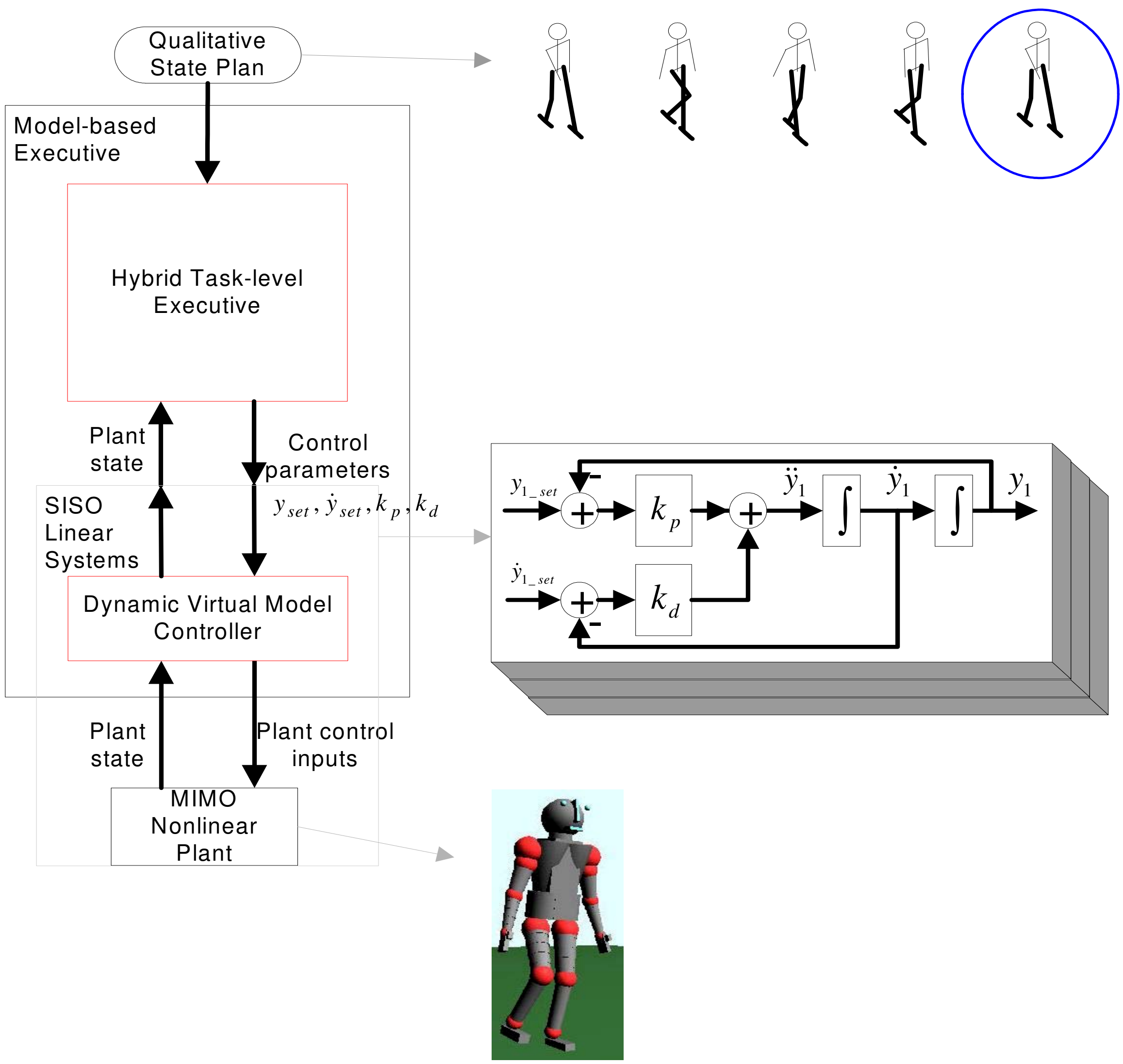


Fig. 1.14 – The model-based executive consists of a hybrid task-level executive and a dynamic virtual model controller. The hybrid executive controls the biped by adjusting control parameters of the linear virtual element abstraction provided by the controller. The hybrid executive sets control parameters to guide state variables through successive goal regions, while satisfying timing and balance constraints.

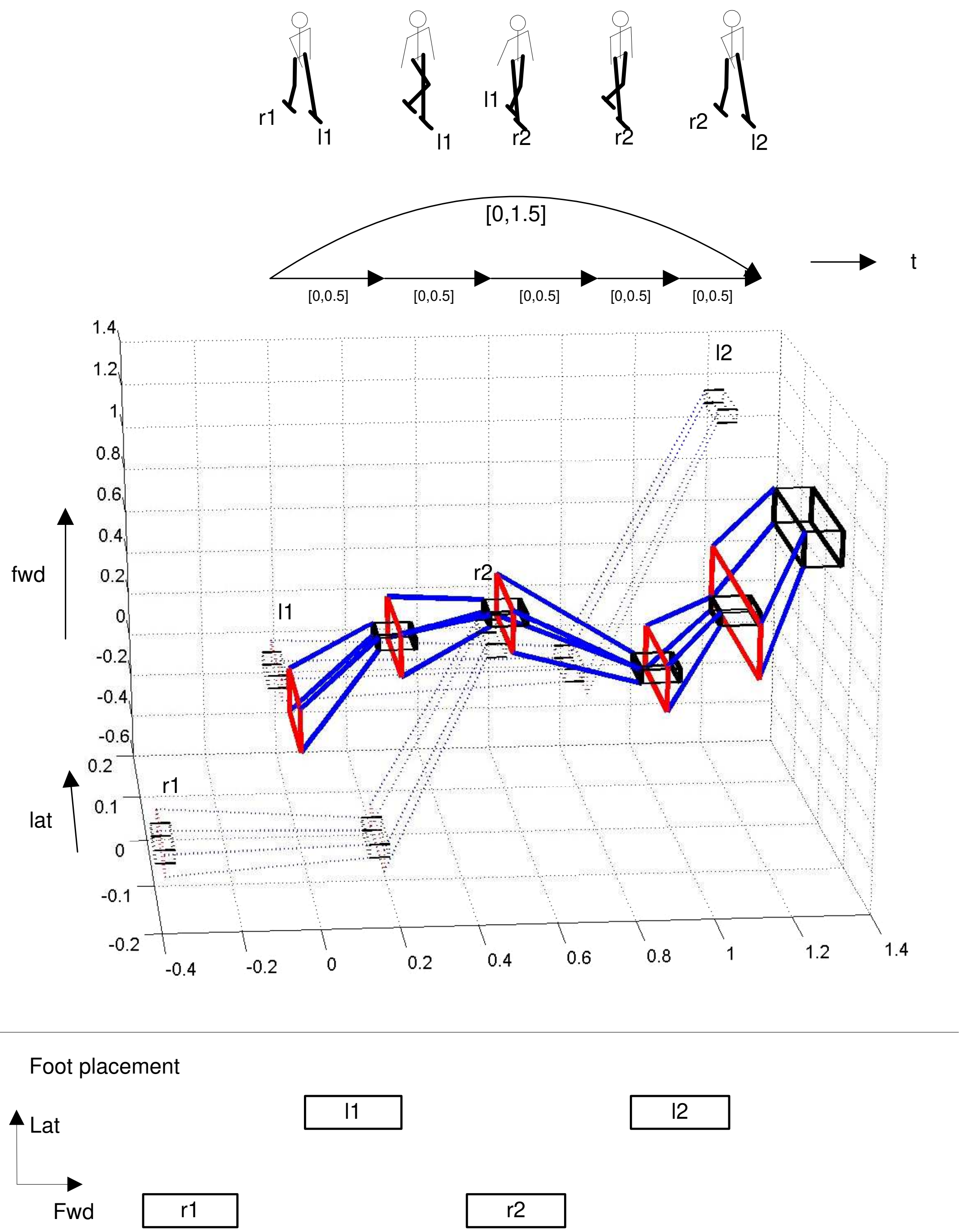


Fig. 1.15 – The qualitative control plan contains flow tubes that define permissible operating regions in state space. As long as state trajectories remain within the flow tubes, the plan will execute successfully. Flow tubes for center of mass are shown, with initial regions in red, goal regions in black, and tubes in blue. Flow tubes for left and right foot position are shown using dotted lines.

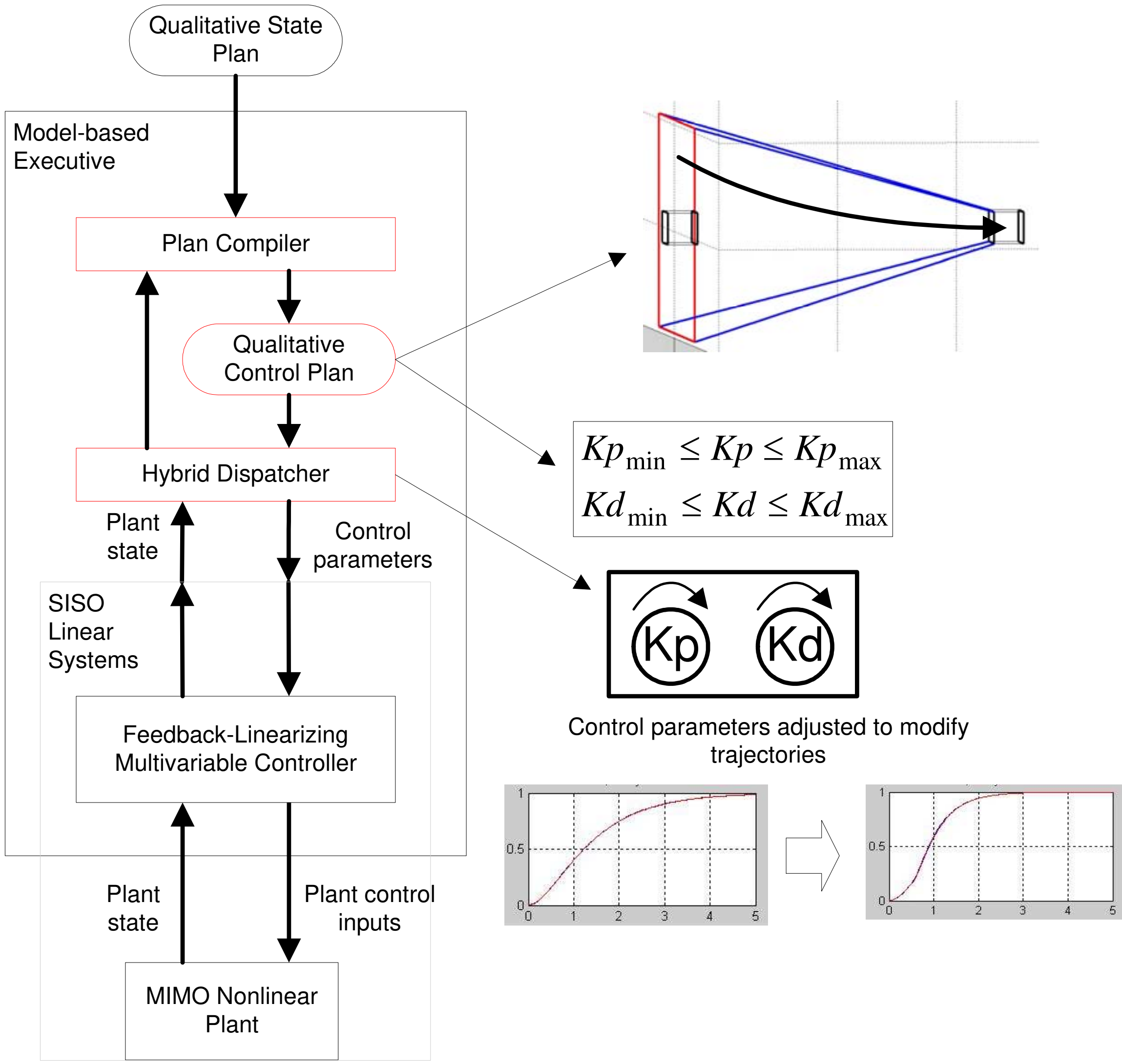


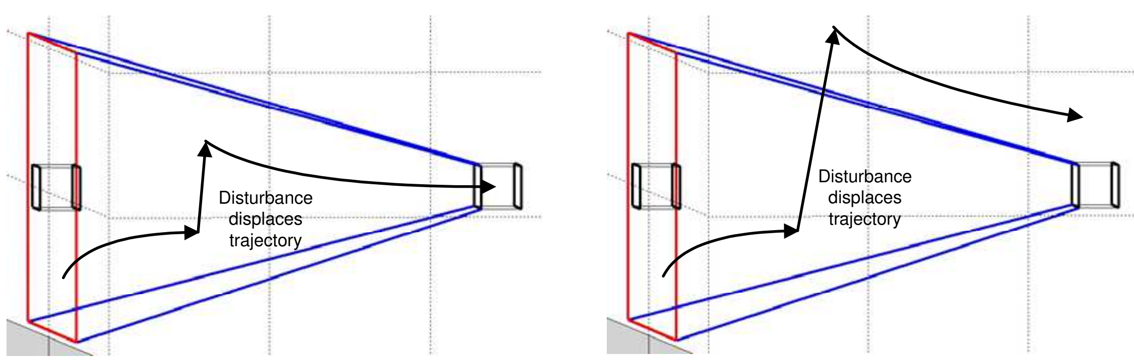
Fig. 1.16 – The plan compiler generates a qualitative control plan for the qualitative state plan. The qualitative control plan contains flow tubes representing state trajectories that satisfy the state plan requirements. A control plan also contains bounds on control parameters. The hybrid dispatcher interprets the qualitative control plan by adjusting control parameters within these bounds to keep the state trajectories inside the flow tubes, and to adjust arrival time in goal regions.

Because computation of flow tubes is time consuming, and because the hybrid executive must run in real time, we perform this step off-line, as a compilation. Thus, the hybrid executive consists of two components: a *plan compiler*, and a *hybrid dispatcher*, as shown in Fig. 1.16.

The plan compiler outputs a *qualitative control plan*, which contains the flow tubes for all state variables in the linear virtual element abstraction. The qualitative control plan also contains feasible ranges for control parameters for this abstraction. These ranges help guide the adjustment of control parameters during execution.

The hybrid dispatcher monitors the state of the linear virtual element abstraction, adjusting control parameters based on the specifications in the qualitative control plan to keep trajectories in their flow tubes. At the start of a plan execution, the dispatcher sets control parameters for each linear system to nominal values that correspond to a feasible trajectory within the flow tube. If a disturbance occurs, the dispatcher may adjust these parameters, but only within the permissible ranges specified in the control plan.

The dispatcher monitors plan execution by monitoring the linear virtual element abstraction's state relative to the plan. In this way, it checks whether each trajectory is in its tube. If a disturbance occurs, the dispatcher attempts to adjust the SISO control parameter settings in order to compensate, so that the trajectory remains inside the tube, as shown in Fig. 1.16a. If the disturbance has pushed the trajectory outside its tube, as shown in Fig. 1.16b, then the dispatcher aborts, indicating to a higher-level planner that plan execution has failed.



a. b.

Fig. 1.16 – a. If a disturbance is not too large, the trajectory remains inside the tube. This means that the dispatcher will be able to adjust control parameters so that the trajectory reaches the goal at an acceptable time. b. If a disturbance is too large, it pushes the trajectory outside its tube, and the plan fails.

The flow tube concept has been used previously to characterize the qualitative boundaries of the state space associated with mechanical devices, and to synthesize controllers for each qualitative region [Bradley and Zhao, 1993]. However, the exhaustive nature of this analysis limited the approach to relatively simple devices with small state spaces. The flow tube approach has also been used for autonomous helicopter control [Frazzoli, 2001]. However, this was a low-dimensional problem, compared with biped control. Furthermore, this application did not allow for execution-time control parameter adjustment, which limits the size of the flow tubes. Also, the helicopter application did not allow flexible temporal constraints. For these reasons, this approach used for helicopter control is unsuitable for task-level control of bipeds.

1.4.3 Balance Enhancement by Generating Angular Momentum

To address the extreme sensitivity of a biped to balance disturbances, the model-based executive automatically integrates different balance strategies, which are based on balance strategies that humans use. Humans use three basic balance control strategies: 1) zero-moment, which uses torque exerted by the stance ankle to generate a force on the center of mass, 2) step adjustment, which changes the base of support, and 3) moment, which uses rotational movement of the torso and non-contact limbs to generate angular momentum about the center of mass. All of these strategies seek to control the horizontal (forward and lateral) position of the system's center of mass (CM) by changing the horizontal component of the ground reaction force. This is the forward and lateral force exerted by the feet against the ground, and is the only external force acting on the system, so an appropriate ground reaction force will accelerate the CM in the desired direction.

Previous implementations of bipedal walking machines [Hirai et al., 1998; Yamaguchi et al., 1996; Kagami, 2001] have relied extensively on the first of these, but have made limited use of the second, and have almost completely ignored the third. This is a problem because there are common situations where each is needed. In particular, the third strategy is useful for tasks where foot placement is constrained. This thesis provides a comprehensive approach that combines use of the first and third strategies. This approach is also compatible with use of the second strategy, step adjustment, although this is not the focus of this thesis.

In the zero-moment strategy, shown in Fig. 1.17a, torque is exerted at the stance ankle joint in order to shift the *zero-moment point* (ZMP). The ZMP is the point on the bottom of the foot through which the ground reaction force vector passes. It represents an average of all points of contact with the ground, and is also called the *center of pressure*. Shifting the ZMP by exerting ankle joint torque changes the lateral component of the ground reaction force, as shown in Fig. 1.17b, and therefore, can be used to exert a beneficial lateral force on the CM to try to control it. For example, if the CM in Fig. 1.17b is moving to the right, the lateral component of the ground reaction force, which is acting to the left, can be used to slow it. If the goal is to balance on one leg, the lateral component can be used to stop the CM movement, if its initial momentum isn't too large. Once the CM has stopped, the ZMP is shifted to be directly under the CM to achieve balance.

Note that with this strategy, the ground reaction force vector points directly from the ZMP to the CM, so that no moment is generated about the CM, hence, the name zero-moment strategy. Note also that the ZMP is confined to the boundaries of the support base, which, in this case, is the boundary of the stance foot, as shown in Fig. 1.17b. This limits the horizontal CM force that can be exerted using this strategy.

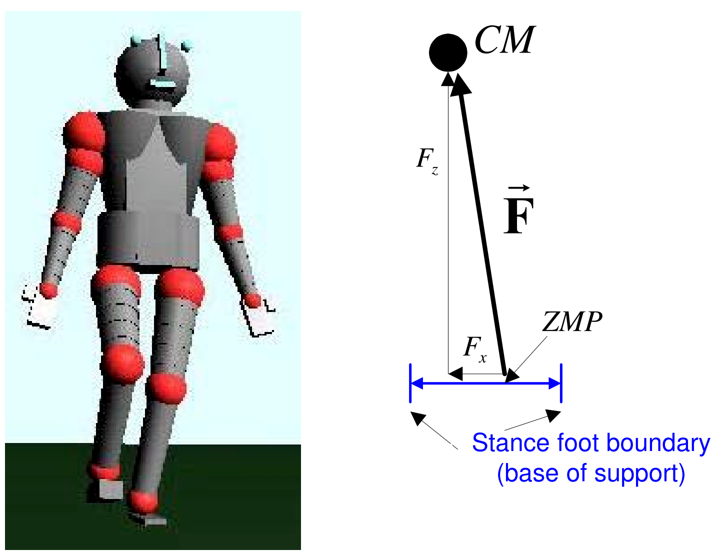


Fig. 1.17 – a. Stance ankle torque strategy (left), b. Relation between CM and ZMP

In the second strategy, shown in Fig. 1.18, a step of appropriate length and direction is taken in order to extend or change the support base. This enlarges or changes the region where the ZMP can be moved, thus allowing for a horizontal force on the CM that is different from the ones possible with the original support base. For example, suppose that the CM in Fig. 1.18 is moving to the left. Because the left-most edge of the original support base is to the right of the CM, a lateral force on the CM acting towards the right cannot be exerted. The leftward CM velocity cannot be reduced with this support base. By taking a step, the support base is extended as shown in Fig. 1.18. The ZMP can now be moved to the left of the CM, so that a lateral force acting towards the right can be exerted.

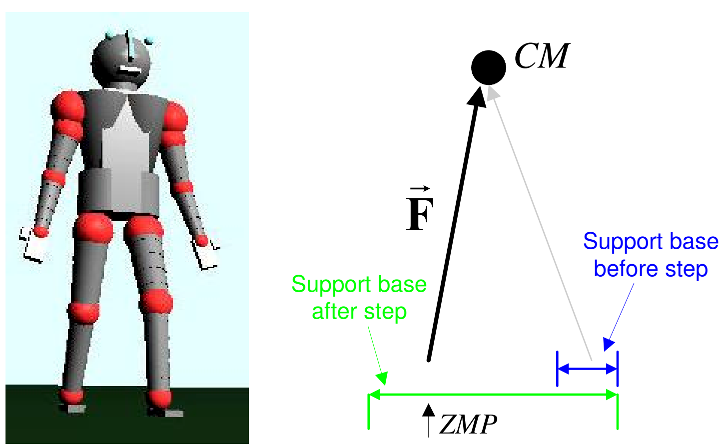
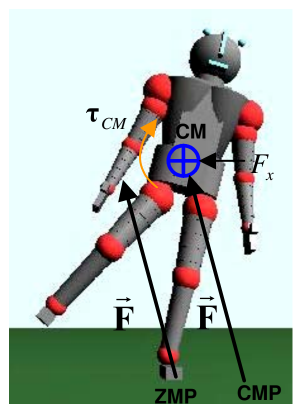


Fig. 1.8 – Stepping to change the support base.

The third strategy, the moment strategy, is used when the first strategy is inadequate, and the second strategy cannot be used, due to constraints on where the stepping foot can be placed. This occurs, for example, in tightrope or balance beam walking, as shown in Fig. 1.19a, but it can also occur in more general situations when foot placement is constrained for some reason. Appropriate rotational movement of non-contact limbs and torso allows for greater horizontal force than is possible with the zero-moment strategy alone. This is because such rotational movement generates a torque about the CM, resulting in *spin* angular momentum about the CM, as shown in Fig. 1.19b. By conservation of angular momentum, this generates a beneficial *orbital* angular momentum of the center of mass about the ZMP. This enhances the effective torque of the ankle, while allowing the foot to remain flat on the ground, which enhances the horizontal restoring force that can be exerted on the CM, as shown in Fig. 1.19b. Another way to think of this is that it increases the effective size of the support polygon, allowing the ZMP to, effectively, move to a point outside the actual support polygon. We call this point the *centroidal moment point* (CMP), as shown in Fig. 1.9b.

Through coordinated action of the hybrid executive and dynamic virtual model controller, our model-based executive automatically employs the moment strategy when the zero-moment strategy is inadequate. This makes the biped more robust to disturbances when traversing terrain with strict foot placement constraints.



a. b.

Fig. 1.19 – a. Tightrope and balance beam walking require rotational movement of the torso and non-contact limbs to maintain balance. b. Angular momentum about the CM allows the CMP to be moved beyond the limits of the support base.

1.4.4 Summary of Benefits

The key benefit of our approach is that unlike current bipedal walking systems, which emphasize stable walking with a regular gait pattern, our system supports robust, execution of walking tasks. Such tasks may require irregular foot placement, in order to traverse difficult terrain, and may require dynamic, agile movement, in order to meet time constraints. Our model-based executive automatically generates feasible state trajectories for the biped that satisfy specified task goals, and computes control actions that achieve these trajectories, compensating for disturbances if necessary.

1.5 Experiments

We demonstrate our approach using a high-fidelity biped simulation serving as the plant. This plant model captures the essential morphological features of the human lower body relevant for standing, balancing, and walking. The model is structurally realistic, with segment lengths and mass distributions defined to match those of a single human test subject [Hofmann, et al., 2002]. The biomimetic nature of the model makes it suitable for comparing its trajectory results with those from human walking trials.

We perform a number of experiments in two categories to validate our approach: stationary balance, and walking. The stationary balance experiments involve maintaining balance only, without the additional goal of moving to a goal location. The walking experiments involve maintaining balance, while at the same time attempting to reach a goal location within a specified period of time.

Stationary balance experiments involve balancing on one leg, and responding to push disturbances of various magnitudes applied to the torso. Lateral and forward direction disturbances are tested. These experiments demonstrate our executive's ability to automatically employ the spin torque balance strategy when necessary.

Walking experiments include nominal walking on firm ground at different speeds. In these experiments, our executive achieves stable dynamic walking at a variety of speeds. For these tests, performance is evaluated in terms of dynamic walking ability and achieved speed. Preliminary comparisons with human walking trial data are also made. These show close agreement of center of mass trajectories.

We also perform a number of experiments involving walking on soft and slippery ground. Stable walking is achieved, even when the ground is so soft that the feet sink into it by as much as 4 cm.

Experiments involving lateral push disturbances while walking with restricted foot placement are also performed. These experiments demonstrate the executive's ability to use a combination of stance ankle torque and spin torque balance strategies, and to synchronize these with forward movement requirements.

Trip disturbance recovery is also tested. These experiments demonstrate the executive's ability to automatically adjust control parameter settings in order to achieve synchronization goals, and thereby, to avoid a fall. The experiments show that significant adjustment of control parameters is needed to recover from this type of disturbance, and that the ability to perform such an adjustment at execution time is critical. This ability distinguishes our approach from traditional ones that use fixed parameters.

To evaluate the system's ability to observe externally specified temporal constraints, we include tests involving the biped kicking a moving soccer ball. To evaluate the system's ability to observe irregular foot placement constraints while walking dynamically, we include tests where the biped has to traverse a path with irregular footholds, as in Fig. 1.3.

For the disturbance tests, performance is evaluated in terms of the system's ability to compensate and execute the plan successfully despite the disturbance. Successful plan execution requires fall avoidance.

1.6 Roadmap

The next chapter provides an overview of related work, with emphasis on research upon which this thesis is based. Chapter 3 describes the three balancing strategies in more detail, using a combination of analysis of human walking data, and biomechanical analysis. Chapter 4 introduces the model-based executive, and defines the qualitative state plan and SISO abstraction. Chapter 5 defines the qualitative control plan. Chapter 6 describes the plan compiler component of the hybrid executive, and Chapter 7, the dispatcher. Chapter 8 describes the multivariable controller that provides the SISO abstraction. Chapter 9 presents results, and Chapter 10, conclusions and future work.

2 Background

This Chapter summarizes previous, related work, beginning, in Section 2.1, with a discussion of artificial walking bipeds, and algorithms that have been used to control them. This is followed, in Section 2.2, by a review of planning and control for *hybrid* systems, that is, systems that have continuous and discrete constraints, and that have spatial and temporal constraints. We review this work because the techniques used can be extended in order to control walking bipeds. This is followed, in Section 2.3, by a review of relevant biomechanical analysis studies. We review this work because it is sensible, when designing controllers for artificial bipeds, to analyze the performance of a successful case of a biped, occurring in nature (a human being). Finally, in Section 2.4, we summarize limitations of this previous work, and introduce how we extend some of this work in this thesis.

2.1 Control of Walking Bipeds

Control of balance for legged robots has been studied extensively. Among the first successful hardware implementations are Raibert's hopping robots [Raibert, 1986]. These robots used pneumatic legs, and were not humanoid, but they did achieve locomotion. More recently, a number of humanoid robots capable of walking have been developed. These include the Honda P3 and Asimo robots [Hirai, 1997, 1998], the Sony SDR [Yamaguchi, 1999], and Tokyo University's H6 [Kagami, 2001]. Asimo and SDR are shown in Fig. 2.1

In this section, we review a number of popular algorithms for control of such devices. In section 2.1.1, we discuss the *ZMP control method*, which is used to control the Honda and Sony robots. In section 2.1.2, we discuss Poincare return map methods, which have been used to analyze performance of passive walkers, and to automatically design controllers for actuated bipeds in order that they achieve stable limit cycle walking. In section 2.1.3, we discuss methods for planning detailed joint trajectories that use high-fidelity models of the system's dynamics. In section 2.1.4, we describe virtual model control methods, which have a number of attractive features compared with the other methods, including simplicity of design, and robustness to disturbances.

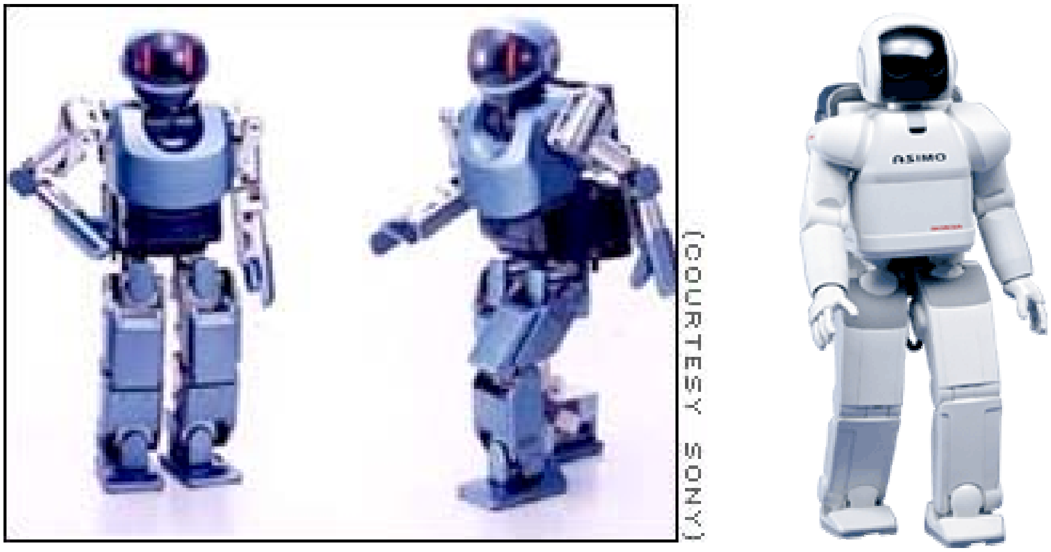


Fig. 2.1 – Sony SDR (left), and Honda Asimo (right)

2.1.1 The ZMP Control Method

The Honda and Sony robots achieve balance control while walking by tracking a desired center of pressure point on the bottoms of the feet. This results in stable walking, as long as disturbances are not significant.

This method of control is called the *Zero Moment Point* (ZMP) control method, after the point on the ground that is tracked. This point is called the zero moment point because it is the location on the ground where the net moment generated from the ground reaction forces is zero [Vukobratovic and Juricic, 1969]. The ZMP can be computed analytically, based on the state and acceleration of the robot's articulated links and joints. It is equivalent to the *center of pressure* (CP), which is the point on the ground through which the ground reaction force acts (see discussion in Chapter 3). The CP can be measured directly using a suitable set of force sensors at the bottoms of the robot's feet. Although the ZMP and CP are equivalent [Goswami, 1999; Pratt and Tedrake, 2005], the term ZMP is commonly used to refer to the point computed from the robot's state and acceleration, whereas the term CP is commonly used to refer to the equivalent point measured using force sensors. In order to avoid confusion, in this thesis, we use the term ZMP, exclusively, to refer to this point.

For control of bipeds, the method for computing the ZMP analytically from the robot's joint trajectories is used to plan these trajectories. The key assumption, with this method, is that the supporting foot or feet are always flat on the ground, and that the ground is level. As long as this is the case, trajectories can be planned as if the robot were a manipulator that is firmly attached to the ground. The key issue here is that, for a biped, the foot is not, generally, in perfect contact with the ground. The foot can easily slip or roll if inappropriate forces are exerted. For a biped, if the ZMP lies at the edge of the support base, the supporting foot or feet may begin to roll [Goswami, 1999]. The ZMP control method solves this problem by ensuring that, for the trajectories generated, the ZMP stays strictly inside the support polygon on the ground, which is defined by the support foot or feet (see further discussion Chapter 3). Thus, by keeping the ZMP strictly inside the support polygon, and therefore, the supporting foot or feet flat, the ZMP control method allows the system to assume that the biped is firmly attached to the ground, just like a stationary manipulator. This assumption is made for the entire duration of a single or double support gait phase.

Because detailed trajectory planning is computationally intensive, it is typically performed offline. These detailed trajectories are then played back, using high impedance (stiff) controllers that closely track the precomputed reference joint trajectories. Deviations between the precomputed and actual ZMP are used to modify these reference trajectories [Yamaguchi et al., 1996; Hirai et al., 1998].

There are a number of problems with the ZMP control method, and with its associated assumptions. First, much of the popularity of the ZMP method is based on the belief that keeping the ZMP strictly inside the support polygon is a sufficient criterion for preventing a fall [Arakawa and Fukuda, 1997; Nishiwaki et al., 1999]. Under this assumption, "if the ZMP is inside the convex hull of contact points between a robot and the ground, the robot will not fall." [Nishiwaki et al., 2001]. This simply is not true! It is possible for the robot to fall down, even if its ZMP remains in the center of its support polygon, as shown in Fig. 2.2. As another example, if all the joints of the robot are made limp, it will collapse, but the feet may well stay flat, at least well beyond the point of recovery.

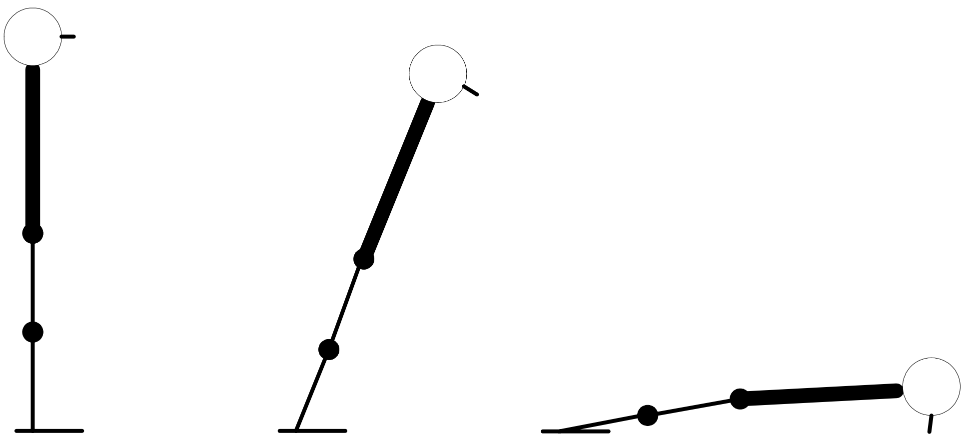


Fig. 2.2 – This sequence shows a biped from the side. The body pitches forward about the ankle joint. At all times, ankle pitch torque is adjusted to ensure that the foot is flat on the ground, and therefore, that the ZMP stays inside the foot. This does not prevent sequences such as the one shown. In the middle pose, the robot's center of mass is well beyond the point where balance can be recovered, yet the foot remains flat on the ground, so the ZMP stays inside the foot's boundaries. In the last pose, the robot has fallen. Its nose is in contact with the ground (ouch), yet the foot is still flat, and the ZMP is within the foot's boundaries.

A more accurate statement about ZMP control is that if the ZMP stays within the support polygon, then the trajectories generated by this method are dynamically feasible, and can be accurately tracked using a high-gain tracking controller. This approach leads to a number of problems, however. First, use of high-impedance position control to track predetermined, detailed, reference trajectories results in a lack of compliance and robustness to force disturbances. The tracking controller will try to follow the predetermined trajectory no matter what, even if the situation requires a completely different response, such as modifying stepping foot placement, or using non-contact limb movement (see discussion in Chapter 3). Humans, on the other hand, are compliant to force disturbances in that they yield, when necessary, and are robust in that they can take complex compensating actions.

Achieving human-like performance using the ZMP method would require it to either generate, or somehow find, a new reference trajectory, quickly, when a significant disturbance occurs. Since generation of such reference trajectories, using the ZMP method, is computationally expensive, and since a very large number of such trajectories is needed to cover a wide range of disturbances, achieving human-like compliance and robustness is an unsolved problem for this method. Thus, although the ZMP control method achieves stable walking on level terrain, it is brittle in that its robustness to force disturbances, and performance on rough, uneven terrain, is poor.

Another problem with the ZMP method is that its requirement that the supporting foot or feet remain flat on the ground at all times is overly conservative. This results in a flat-footed, short-step walking style that is less dynamic than that of humans. During normal walking, humans do not obey the ZMP requirement. For example, just before taking a step, humans push off with the toe of the stepping foot. During this toe-off movement, the heel lifts, and the foot pivots about the ball of the foot; the foot does not remain flat on the ground.

Finally, there is a fundamental problem with the ZMP method that only becomes clear when one considers the true inputs and outputs of the balance control problem, as explained further in Section 3.3. Consider, first, the problem of driving a car at a constant speed. To accomplish this, the driver monitors the speed as shown on the speedometer, and adjusts the gas pedal position accordingly. Gas pedal position represents an acceleration input to the system, and the velocity is the output being controlled, as shown in Fig. 2.3. For this problem, and control problems in general, desired behavior is specified in terms of the output to be controlled not the input. Thus, for the problem of driving a car at a desired speed, the desired behavior is specified as this speed, not as a gas pedal position. In fact, it would be awkward and difficult for a driver to maintain a constant speed, just based on the gas pedal position, without looking at the speedometer.

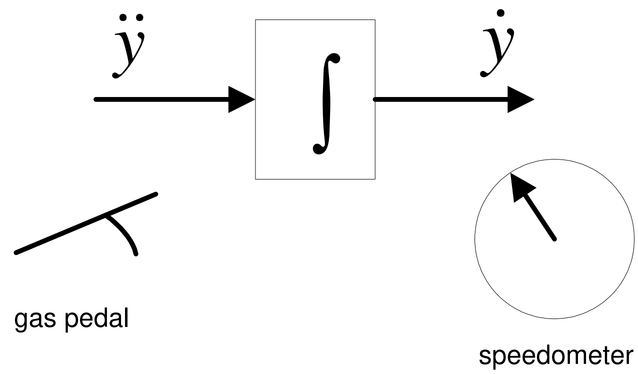


Fig. 2.3 – The control input for the automobile speed control problem is the gas pedal position, which is proportional to acceleration. The output sensor is the speedometer, which is proportional to velocity.

Monitoring just the gas pedal position, and not the speedometer represents tracking a control input, rather than an output. In general, tracking of an input reference trajectory rather than an output is an open-loop approach, and is avoided in most control systems. Open-loop approaches are susceptible to errors and instability because they do not monitor the variable that is really of interest. Yet, the ZMP method is an open-loop approach! As we will explain in Chapter 3, the ZMP is directly related to the horizontal force applied to the CM, and is thus, an acceleration input to the system. The outputs that we desire to control, for the purpose of balance control, are the CM position and velocity. Because the ZMP method tracks the ZMP, an input, rather than the CM, an output, the ZMP method is, fundamentally, an open-loop control approach.

2.1.2 Stability Analysis and Control Design using Poincare Return Maps

Poincare return maps are a useful technique for analyzing periodic systems. With this technique, the system is assumed to have a stable limit cycle. A fixed point is chosen at some point in this cycle. Small deviations from the cycle follow the linear relation

$$\mathbf{x}_{n+1} = \mathbf{K}\mathbf{x}_n \qquad (2.1)$$

where $\mathbf{x}_n$ is the vector of deviations from the fixed point, $\mathbf{K}$ is a linear return map, and $\mathbf{x}_{n+1}$ is the vector of deviations in the following cycle. If the absolute value of the eigenvalues of $\mathbf{K}$ are all less than one, then the limit cycle is stable. Poincare maps are commonly used in the analysis of passive dynamic bipeds [McGeer, 1990; Goswami et al., 1996; Collins et al., 2001]. They have also been used to analyze learning algorithms for a passive dynamic biped with ankle actuation [Tedrake, 2005], and for automated control system design of an actuated planar biped [Westerveldt, 2005].

Poincare return maps have a serious limitation; they are only applicable for systems with periodic behavior. Thus, they cannot be used for the kinds of locomotion tasks introduced in Chapter 1. There is nothing periodic about walking across unevenly spaced stones, or kicking a moving soccer ball. Thus, Poincare return maps cannot be used for the applications of interest in this thesis.

2.1.3 Joint Trajectory Planning Methods

Trajectory planning algorithms attempt to generate optimal reference trajectories, which, if followed, are guaranteed to be feasible. The trajectory generating aspect of the ZMP method, described in Section 2.1.1, is a special case of a joint trajectory planning method. In this section, we describe a general class of such methods, including ones that do not suffer from the ZMP method's limitations.

The methods described here do not address the problem of control; their purpose is to generate realistic, feasible reference trajectories. These methods are used, primarily, in graphics applications. In such applications, the trajectories are simply played back in a deterministic, open-loop fashion, so there is no need for a tracking controller.

A type of open-loop dynamic optimization called space-time constraints [Popovic and Witkin, 1999] has been used to generate human motions for animation, as shown in Fig. 2.4. With this type of algorithm, each trajectory is represented by a spline with a finite set of control points, as shown in Fig. 2.5. The control points are adjusted by a sequential quadratic programming (SQP) optimizer so that constraints are satisfied, and a cost function is minimized. The first and last control points in a trajectory are fixed to correspond to boundary conditions. For animation applications, this algorithm is used to adapt libraries of motion capture data to new movements. The motion capture data provides the initial animation key frame poses, which are then adjusted to poses representing new desired movements. The poses specify the control point boundary conditions. Laws of physics, and human motion preferences are encoded as constraints and costs to be minimized in the problem formulation. This ensures that the control point adjustments made by the SQP optimizer result in motion trajectories that look realistic.

Fig. 2.4 - Human motion animation using space-time constraints.

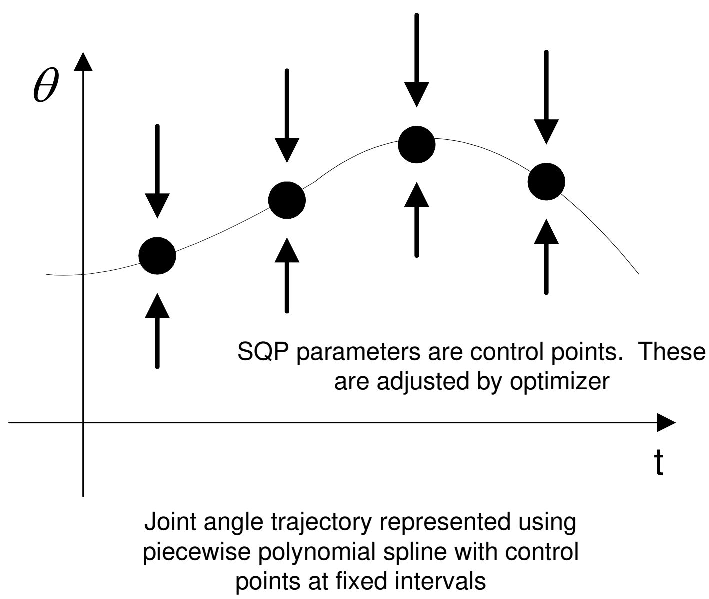


Fig. 2.5 – Control point adjustment by SQP optimizer.

This approach has also been used in biological studies to investigate human walking [Anderson and Pandy, 2001]. In this study, the time horizon used was half of a gait cycle (a single step), so the boundary conditions represented the beginning and the completion of a step. The model used included detailed muscle dynamics, and the optimization formulation used a very simple cost function: minimization of muscular energy expenditure. This generated results that were reasonably biomimetic, based on comparison with data from human trials. However, it is not clear how much of the result was due to boundary conditions and constraints, and how much was due to the goal of minimizing muscle energy.

Neither the animation nor the human walking applications of the space-time constraints algorithm involve real time control; they are both off-line applications. These are open-loop trajectory generation systems; they do not monitor plant state, and they do not provide closed-loop control. Therefore, they do not deal with disturbances. Theoretically, this type of algorithm could be extended to deal with disturbances by adding a controller that attempts to follow the generated reference trajectories. However, as discussed in Section 2.1.1, a controller that closely tracks detailed joint trajectories has limited robustness to disturbances.

2.1.4 Virtual Model Control Methods

Advances in hybrid position/force, or, *impedance* control [Hogan, 1985] have addressed problems of robustness in the presence of disturbances, and performance in unstructured environments. Impedance control monitors the relationship between force and position, rather than controlling these separately, and therefore, provides a unified approach for both free motions and motions made during contact with the environment. Advantages of impedance control approaches are compliance and robustness no force disturbances, and simplicity, because they don't require modeling of plant dynamics, and don't require solution of inverse kinematics problems.

The *virtual model control* (VMC) algorithm [Pratt et al., 1997], implements a kind of impedance control, and so, inherits the associated advantages. This algorithm uses "virtual" spring and damper elements attached to points on the mechanism that implement simple force control based on linear feedback of position and velocity error:

$$F_x = k_p\left(x_{des} - x\right) - k_d\dot{x} \tag{2.2}$$

The virtual elements do not actually exist, but the system behaves as if the joint torques were 0, and the mechanism were being moved, like a puppet, by these elements. The required leg joint torques needed to achieve the force specified by the virtual elements is computed using a Jacobian-based static force computation that has been used extensively in robot manipulators [Paul, 1981; Craig, 1989]. VMC is a powerful multi-variable control abstraction because it maps multiple output goals, such as a 6-element spatial vector for desired body force, to multiple required inputs, an n-element torque vector, where n is the number of joints in all legs in contact with the ground.

This control algorithm has been used to control planar bipeds [Pratt, et. al., 1997], as shown in Fig. 2.6, and hexapods [Pratt, et. al., 1996]. Both of these implementations compute an overall desired force at a point on the robot's body, and then allocate contribution of force to each leg that is on the ground.

Fig. 2.6 – Planar biped using the VMC control algorithm.

The main problem with this algorithm is that it makes the simplifying assumption that the system is static. It does produce the desired force at a reaction point, but due to the complex nonlinear dynamics of the system, this does not mean that this point will move in the direction of the force. In other words, eq. 2.2 is not a linear control law that specifies acceleration. This problem can be addressed by increasing the gains in eq. 2.2, but this is only useful up to a point. Making the gains too high can cause large forces leading to system instability. Furthermore, use of overly high gains is undesirable because this leads to very rigid, non-compliant control.

One approach to solving this problem, without increasing gains, is to use feedback linearization, an algorithm that takes dynamics into account. Feedback linearization [Slotine and Li, 1991] is a powerful technique for linearizing a nonlinear system, and decoupling the variables to be controlled.

A typical robotic plant consisting of articulated links with actuated joints is commanded by a torque vector input. The torques are applied to the respective joints, and the plant moves according to its dynamics, resulting in a new joint state [Featherstone, 1987]. The problem is that the robot plant is typically highly nonlinear, so it is difficult to know how a set of torque inputs will cause the system to move. Furthermore, it is desirable to control the plant in terms of workspace coordinate quantities rather than joint coordinate quantities. For example, a workspace quantity like lateral position of center of mass is of more interest in balance control than a joint space quantity like left knee joint position.

Feedback linearization solves this problem using a sophisticated geometric transformation that makes the overall system appear linear to the outside world. The transformation also changes the system inputs to be ones of interest. Finally, these inputs, and the corresponding outputs, are decoupled so that the entire multivariable plant appears, to the outside, as a set of single-input single-output (SISO) 2^{nd}-order linear systems. This is extremely beneficial because techniques for controlling such simple systems are well developed.

For a robot plant, feedback linearization can be used to convert desired output variable accelerations, $\ddot{\mathbf{y}}$, into corresponding joint torques, $\boldsymbol{\tau}$, as shown in Fig. 2.7. An output transformation, $\mathbf{h}$, converts from robot joint state $(\mathbf{q},\dot{\mathbf{q}})$ back to workspace state

$(\mathbf{y}, \dot{\mathbf{y}})$. If we draw a black box around the diagram in Fig. 2.7, the nonlinear tightly-coupled MIMO robot plant appears, to the outside world, to be a set of decoupled SISO linear 2nd-order systems.

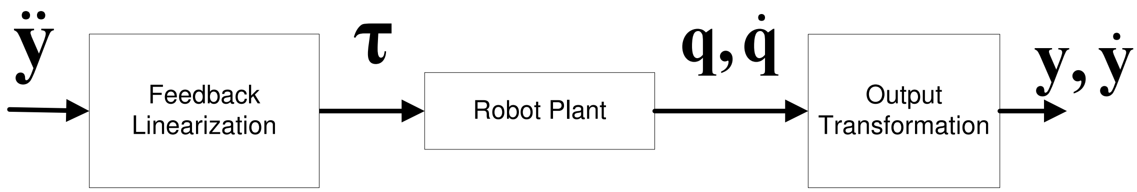


Fig. 2.7 – Feedback linearization for a robot plant.

Each of these SISO systems can be controlled independently using simple linear control rules, as shown in Fig. 2.8. Thus, feedback linearization provides a powerful control abstraction.

Feedback linearization is a useful technique, but the linearization can be subverted if there are constraints on state variables or inputs. This is because the system, which is typically fully constrained by the feedback linearization, becomes over-constrained when additional constraints are added. This is a common problem in the control of multivariable systems; the controller must make a compromise and prioritize goals. Thus, when additional constraints are active, and the system becomes over-constrained, the controller has to sacrifice (hopefully temporarily) the goals of some controlled variables in favor of the goals of other, more important ones. The notion that some variables are more important than others is a basic concept in multivariable control.

The *whole-body control* method combines feedback linearization with this kind of prioritization in order to control a humanoid form that has multiple movement goals [Sentis and Khatib, 2004]. With this method, goals are prioritized, and only the most important ones are included, the rest being ignored so that the system will remain feasible. A shortcoming of this method, and of feedback linearization approaches in general, is that they are susceptible to model error.

A similar approach has been used for balance control using non-contact limb movement [Hofmann et al., 2004]. In this work, infeasibilities are avoided by using a quadratic programming (QP) optimizer with a problem formulation that incorporates slack variables for each desired acceleration input. Slacks are minimized, with relative costs of the slack variables being used to express relative importance of each element. This mechanism provides the "safety" valve that avoids infeasibilities. This approach also uses a sliding control algorithm [Slotine and Li, 1991] to mitigate the effects of model error. The *Dynamic Virtual Model Controller*, described in Chapter 8, is based on this approach.

2.2 Plan Execution for Hybrid Systems

As described in Chapter 1, example locomotion tasks are walking over unevenly spaced stones within a maximum period of time, or kicking a moving soccer ball. Such tasks are specified using a plan that expresses task goals. In this section, we review previous work on systems that execute such plans. Although these previously developed systems do not address the problem of executing plans for bipedal locomotion tasks, they are relevant to this thesis in that they provide useful background for the general problem of plan execution, and because we extend some of their approaches and techniques for use in bipedal locomotion plan execution.

The system being controlled according to a plan is called a *plant*. Thus, for bipedal locomotion plan execution, the plant is the biped. Other examples of plants include autonomous aircraft, autonomous rovers, and chemical processes.

Complex plants such as bipeds and aircraft are difficult to control because they are highly nonlinear, high-dimensional, and tightly coupled. This makes it difficult to compute control actions that lead the plant to a desired state. To solve this problem, *model-based executives* [Williams and Nayak, 1997; Leaute, 2005; Effinger et al., 2005] use a plant model to predict the effect of control actions on plant state, and thus, to simplify computation of control actions. As introduced in Chapter 1, we use such a model-based executive to control the biped.

We are particularly interested in plan execution for *hybrid* plants. The state of a simple *continuous plant*, like an inverted pendulum [Kailath, 1980], is represented by a set of state variables that have continuous values. A plan for such a system is just a reference trajectory, or a fixed setpoint, expressed in terms of these state variables. Real plants are often more complicated in that they have significant nonlinearities and discontinuities that require their overall operating region to be sub-divided into smaller operating modes that are managed by different control rules. We call such plants *hybrid* because their behavior is specified by a combination of discrete and continuous variables. Such plants, of which walking bipeds are an example, require a more sophisticated approach to planning and control than the simple reference trajectories for purely continuous plants.

The discrete modes in a hybrid plant arise from discontinuities in the plant itself, and/or because the plant is so nonlinear, that very different control settings have to be used in different operating regions. For a walking biped, plant discontinuities occur when the biped transitions between single support modes, where one foot is on the ground, and double support modes, where both feet are on the ground. As explained further in Chapter 4, when a biped begins to take a step, it lifts the stepping leg, resulting in a transition from a double support mode to a single support mode. When the stepping leg strikes the ground, the biped transitions from single support back to double support.

The different kinds of support modes represent qualitatively different operating regions for the plant, each with its own set of constraints and control requirements. Discrete variables are used to indicate whether a foot is in contact with the ground, and to represent the discrete modes.

The presence of discrete modes complicates the planning and control problem. In a hybrid system, plans are not just reference trajectories as for continuous systems. They must also specify how discrete state should evolve. This corresponds to understanding how the system should transition from one mode to another. Additionally, within each mode, the plan must specify how the continuous state should evolve, using reference trajectories and related control information. Additionally, a plan may specify temporal restrictions on when mode transitions can occur.

The problems of discrete mode transition and temporal constraint satisfaction have been studied, extensively, for discrete state systems. A discrete state system is one that has discrete state variables, but no continuous ones. Thus, the details of continuous planning and control can either be ignored or abstracted away, depending on the particular application. Before investigating planning and control for hybrid systems, it is useful to review planning and execution techniques that have been developed for discrete systems. In particular, it is useful to review the key concepts and attractive properties of these systems, and then investigate how these concepts can be extended for hybrid systems.

2.2.1 Plan Execution for Discrete State Systems

Plans for discrete state systems are represented using a network of activities specifying temporal bounds and constraints on discrete state, as shown in Fig. 2.8.

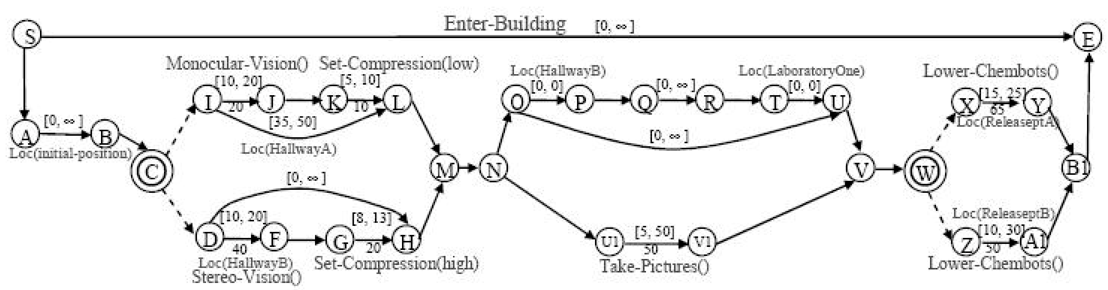


Fig. 2.8 – Discrete state plan for urban rescue mission [Walcott, 2004].

Discrete state constraints are used to represent preconditions that must be satisfied for an activity to begin, and post conditions that must be satisfied for an activity to end. These conditions may be functions of discrete variables associated with the activity, and also discrete variables associated with related activities. Operating conditions that must be true while the activity is executing can also be specified, but these are functions only of variables associated with the activity. Thus, a weak coupling between activities is assumed. In particular, two activities can be executed at the same time as long as their pre and post conditions do not collide, and their temporal constraints are satisfied. Thus, the plan specifies how activities should be synchronized based on pre and post conditions, and on temporal constraints, but otherwise, the activities can be executed independently.

For such discrete state systems, plans of the type shown in Fig. 2.8 are generated by a search-based planner [Effinger et al., 2005]. The planner guarantees that the plan is feasible, even in the presence of bounded temporal uncertainty. The plans are executed by a *dispatcher*, which monitors plant state, and schedules activity transitions. In particular, for any activity that is executing, the dispatcher decides when that activity should end by checking if its post conditions and temporal bounds are satisfied. The dispatcher also decides when an activity should begin based on whether its preconditions are satisfied.

The problem of scheduling activities consistently with temporal constraints has been studied extensively for discrete activity systems, which are similar to discrete state systems in that they have activities and temporal constraints, but are different in that they do not represent discrete state. We now review how discrete activity systems process and execute temporally flexible plans, in order to investigate whether these techniques can be extended for use with hybrid plants.

2.2.2 Execution of Temporally Flexible Plans in Discrete Activity Systems

The problem of efficient plan execution has been studied extensively for discrete activity plan execution systems. A discrete activity plan consists of activities, events, and temporal constraints. Activities connect events in that an activity has a start event and a finish event. Temporal constraints specify a lower and upper temporal bound on time between two events. The job of the execution system is to efficiently decide times for the events, and thereby, schedule execution of the activities, such that the event times are consistent with the temporal constraints of the plan.

A challenge to efficient execution of activity plans is that the temporal constraints stated explicitly in the plan may imply further temporal constraints on activities that the executive must observe in order to ensure temporal consistency. For example, suppose a plan involves driving from Boston to New York, and then on to Washington D. C. Suppose the plan specifies that the drive from Boston to New York should take 5 hours, and that the overall trip should take, at most, 9 hours. This implies that the New York to Washington drive should take, at most, 4 hours.

Computing these implicit bounds at execution time is inefficient; the solution is to compute them offline, before any execution begins. Thus, for activity plans, execution efficiency is achieved by compiling the plan into a dispatchable form that makes the tightest, that is, most restrictive, temporal bounds explicit [Muscettola, 1998]. Such a dispatchable plan is then executed by a *dispatcher*. Because all temporal bounds are explicit in the dispatchable plan, the dispatcher can execute the plan directly, and doesn't have to worry about deducing implicit bounds at execution time.

Thus, a typical activity plan executive consists of two components: a compiler and a dispatcher. The compiler converts the plan into a dispatchable form, to be executed by the dispatcher. The dispatcher updates the explicit bounds in the dispatchable plan if disturbances occur that require further tightening of subsequent activity durations.

We now review how such an activity plan compiler and dispatcher work. To begin, we introduce the concept of a *Simple Temporal Network* (STN) [Dechter, 1991], which is frequently used to represent the temporal constraints of an activity plan. We then review requirements for dispatchability, in terms of a plan's STN, and algorithms that convert plans into dispatchable form.

A *Simple Temporal Network* (STN) is a directed graph that represents the temporal constraints of a plan. The nodes of the graph represent events, and the arcs between the nodes represent temporal bounds on the duration between the events. An example STN is shown in Fig. 2.9a. An STN has an equivalent representation, called a distance graph [Dechter, 1991]. A distance graph allows shortest path algorithms to be used to determine implicit constraints of an STN, and for checking the STN's consistency. Fig. 2.9b shows the distance graph corresponding to the STN in Fig. 2.9a. Implicit constraints are derived by computing shortest paths in the distance graph, for example, as in the Floyd-Warshall all pairs shortest path (APSP) algorithm. Fig. 2.9c shows the APSP graph corresponding to the distance graph of Fig. 2.9b. The APSP graph exposes the implicit constraints between A and C.

The consistency of an STN is also checked using the APSP graph. If the APSP graph contains no negative cycles, then the STN is temporally consistent [Dechter, 1991]. Negative cycles are detected by checking for negative distances on the diagonals of the tabular form of the APSP graph. More efficient algorithms also exist for testing consistency using a *single source shortest path* (SSSP) approach.

A dispatcher for a plan constrained by an STN selects the execution time of events on the fly and then deduces the effect of this decision on the feasible execution times of future events through a one step local propagation. To do this, the dispatcher maintains an *execution window* for unexecuted events. An execution window consists of lower and upper bounds, which represent the range of feasible execution times for the event. When an event is executed, the local propagation updates the execution window of unexecuted events.

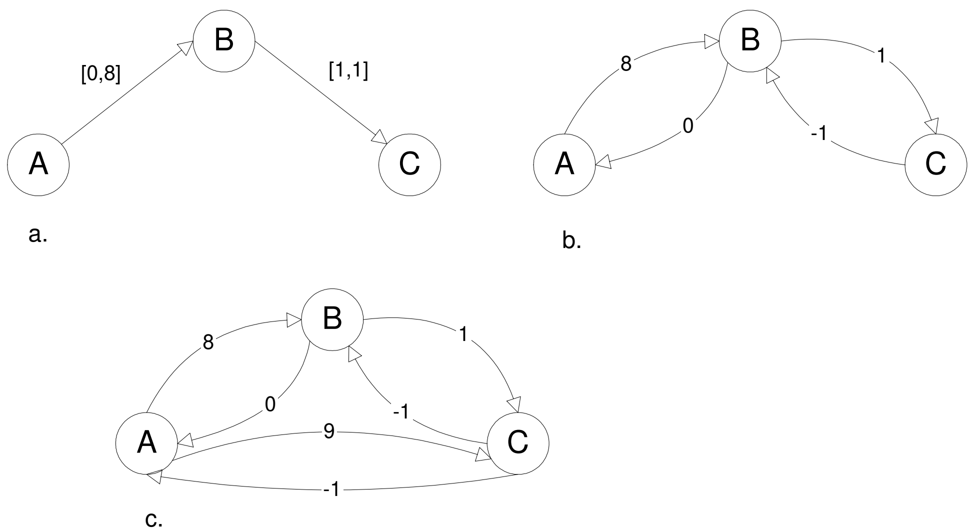


Fig. 2.9 a. STN; b. corresponding distance graph; c. corresponding APSP graph

Fig. 2.10 (see also [Stedl, 2004]) shows an example of execution of a plan constrained by an STN. The initial execution windows, at T=0 (a.), are computed based on the edges in the distance graph. Let's assume that A is the first event executed and that it is executed at T=0. The upper bound on the execution windows is then determined by computing the shortest path from A to every other event. This is a Single-Source Shortest Path (SSSP) computation. Similarly, the lower bound on the execution windows is determined by computing the shortest path from every event to A. This is a Single-Destination Shortest Path (SDSP) computation. These two computations result in an execution window of [1, 10] for event B, and [6, 20] for event C.

After event A is executed, the dispatcher considers execution of events that become enabled by execution of event A. An event, X, is enabled only if all events that must precede X have been executed. Events that must precede X are found by following negative outgoing edges from X. For example, suppose there is a negative outgoing edge from X to event Y. This negative edge represents a lower bound on the amount of time that X must happen after Y. Thus, Y must precede X. In the example of Fig. 2.10, B becomes enabled after A is executed because there is a negative outgoing edge from B to A. Thus, when an event (A) is executed, newly enabled events (B) can be found by following negative incoming edges to the executed event.

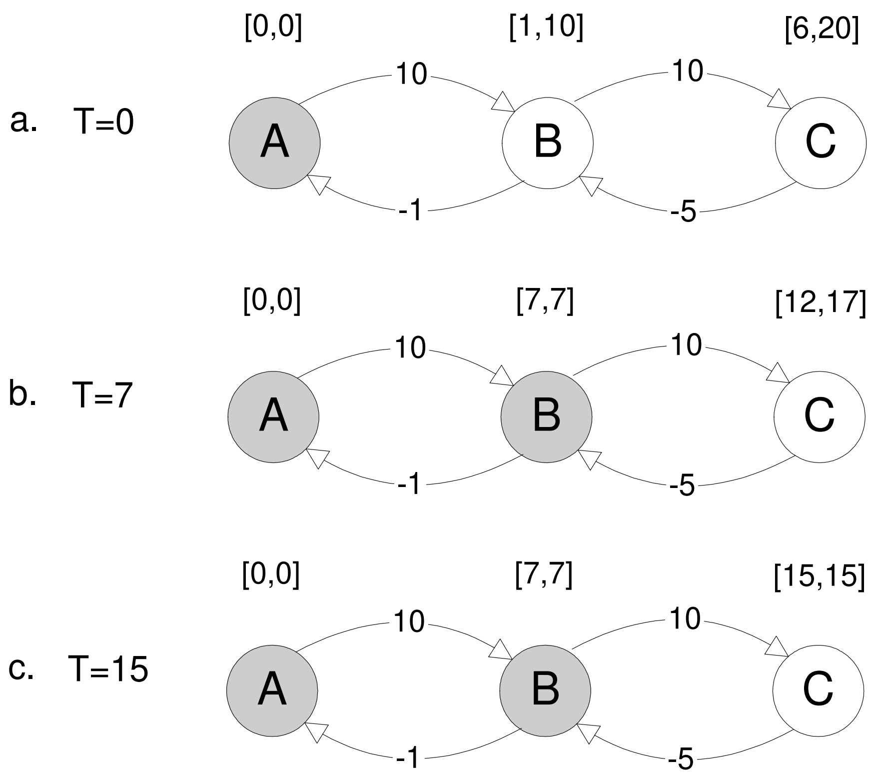


Fig. 2.10 – Plan execution using local propagation; a) event A is executed at T=0, b) event B is executed at T=7, c) event C is executed at T=15

After event A is executed at T=0, time advances. The newly enabled event, B, becomes *alive* when the current time is within its execution window. When an event is both alive and enabled, the dispatcher is free to execute it. In the example of Fig. 2.10, the dispatcher decides to execute event B at time T=7. The effect of this execution is propagated to event C, resulting in a tightening of C's execution window. Propagation is accomplished using the SSSP and SDSP computations, just as these are used for computing initial windows, but beginning with the newly executed event as the reference point. After event B is executed, event C becomes enabled. It becomes alive when T=12. The dispatcher decides to execute this event at T=15.

Pseudocode for this dispatching algorithm is shown in Fig. 2.11. Note that the input to this algorithm is a *dispatchable graph*. A graph is dispatchable if it always results in consistent event executions when input to the dispatching algorithm. Specifically, a graph is dispatchable if local propagation to future events can be used, where upper bounds only need to be propagated along outgoing non-negative edges, and lower bounds only need to be propagated along incoming negative edges.

```
Function STN_DISPATCHING(G)
Input:  a dispatchable distance graph G
Effects:  dynamically schedules each event in G

1.  A = {start_event}   // First event to execute
    current_time = 0
    S = { }

2.  Compute initial execution windows for all events in G

3.  Choose an event, EV, in A, such that current_time is within EV's execution
window

4.   Set EV's execution time to current_time and add EV to S

5.  Propagate time of execution to EV's immediate neighbors in G

6.    Add newly enabled events (events with negative edges going to EV) to A

7.   Increment current_time until it has advanced to some time between
        min{lower_bound(EV)  : EV in A and
        min(upper_bound(EV)  : EV in A

8.  If all events in S, then done, else go to 3
```

Fig. 2.11 – Dispatching algorithm pseudocode [Stedl, 2004]

An STN can be converted into an equivalent dispatchable graph by first computing the associated distance graph, and then computing the associated APSP graph [Muscettola, 1998]. While the APSP graph is dispatchable, it may have a large number of redundant edges that result in unnecessary propagation. Such redundant edges can be removed without adversely affecting the ability of the dispatcher to dynamically execute the network. The following *triangle rule* [Muscettola, 1998] is used to detect redundant edges. Given three events: A, B, and C,

(1) A non-negative edge AC is redundant if $|AB| + |BC| = |AC|$

(2) A negative edge AC is redundant if $|AB| + |BC| = |AC|$

The reasoning behind this triangle rule is as follows. First, note that $|AC|$ would never be greater than $|AB| + |BC|$ since this would violate the properties of an APSP graph. Second, if $|AB| + |BC| = |AC|$, then the individual constraints AB and BC combine so that their effect on constraining the time between A and C is identical to that of the constraint AC. Now, if $|AC|$ is less than $|AB| + |BC|$, then constraint AC cannot be removed because it allows for a shorter path from A to C than the one through B.

The algorithm given in [Muscettola, 1998] first marks redundant edges, and then removes these from the graph. This results in a *minimal dispatchable graph*.

Another problem that has been studied extensively for discrete activity plan execution systems, besides efficient plan execution, is the problem of scheduling for uncertainty in activity execution times. Such uncertainty may be due to external disturbances to processes that are being controlled, or to the fact that some events are completely outside the control of the system. For example, the activity of driving a car involves a process being controlled; the driver attempts to keep the car on the road by using the steering wheel, gas pedal, and brake. Bumps on the road may slow the progress of the car. This represents a disturbance that causes a temporal delay in the execution of the activity. On the other hand, an activity that requires waiting for rain to stop depends on an event that cannot be controlled.

Temporal adjustments arise from the need to compensate for temporal disturbances. The possibility of such disturbances implies temporal uncertainty in the execution time of an activity. Some discrete state plan execution systems represent such uncertainty explicitly when bounds on the uncertainty are known. Thus, in such discrete state plan execution systems, the temporal plan contains some events whose execution time can be controlled, within some range, and ones whose execution time is uncertain, with the uncertainty bounded by some range [Stedl, 2004; Morris et al., 2001].

For example, suppose that a plan requires a car and a sailboat to leave Boston for Provincetown at the same time, and then to meet there at some later time. The duration of the sail is uncertain, but the uncertainty is bounded to be between 6 and 12 hours. Likewise, the duration of the drive is uncertain, but is known to be between 3 and 4 hours. The car is guaranteed to arrive in Provincetown before the sailboat regardless of the actual duration outcomes of each trip. Therefore, the car will have to wait. Suppose that the car is able to wait anywhere between 0 hours and 10 hours for the sailboat to arrive. Presumably, the occupants of the car lose patience waiting longer than 10 hours.

In the worst case for waiting, the car arrives in Provincetown after 3 hours, and the sailboat takes 12. However, even though the car arrives 9 hours earlier, synchronization is still assured because the car can wait 10 hours for the boat. This plan is said to be *dynamically controllable* [Morris et al., 2001]; regardless of the actual duration outcomes of the uncertain activities, the plan is guaranteed to succeed. A key to this success is the car's ability to wait long enough. The large controllable duration of the wait activity is used to compensate for the uncertainty in the other activities. A second key to success is the ability to decide dynamically the amount of time to wait in Provincetown. Suppose that the car is willing to wait no longer than 5 hours. In this case, the plan is no longer dynamically controllable; success cannot be guaranteed. If the sailboat takes too long, the plan will fail.

Determination of dynamic controllability requires that uncertain durations be represented explicitly. This is accomplished using a *Simple Temporal Network with Uncertainty* (STNU), which is an extension of an STN that allows some links to have uncertain duration, where the uncertainty is bounded. STNU's can be used in applications where good bounds on uncertain activities are known, and where it is necessary, at the beginning of plan execution, to be completely sure that the plan will succeed. However, it is not applicable to problems like biped locomotion, where the range of temporal disturbances can be large, but their occurrence is relatively rare. Consider, for example, the problem of kicking a moving soccer ball. Suppose that the biped must take 4 steps forward to do this, and that the soccer ball is moving in a direction perpendicular to the direction of the biped's walking. Thus, the biped must arrive in a location near the soccer ball at the right time in order to kick it. If the soccer ball is moving quickly, the range of times that the biped must be in this location is very limited, hence kicking a soccer ball in this way requires skilled timing.

Suppose now that a disturbance, such as a push or a trip may occur during any step that the biped takes. Such a disturbance may cause a delay. The biped might be able to compensate for this delay by taking subsequent steps more quickly, and might then still be able to kick the soccer ball. Note that, theoretically, such a disturbance could occur during each step that the biped takes. Incorporating this uncertainty explicitly into the plan for every step the biped takes is overly conservative; it represents the worst case. Dynamic controllability requires that this worst case be overcome, somehow. Due to actuation limits, the biped is not able to overcome this worst case scenario, and dynamic controllability cannot be guaranteed. Of course, for this application, this does not mean that the biped shouldn't try to kick the soccer ball. For this type of application, use of an STNU is not appropriate, because it does not adequately represent the probability of a disturbance, or of the success of a plan.

A more appropriate approach for the biped application is to begin by assuming that a disturbance will not occur, and then, if it does occur, to deal with it reactively. Therefore, we use STN's rather than STNU's, and use the approach of [Muscettola, 1998] described previously. This allows for fast determination that a plan cannot succeed, if a disturbance is too large. It does not require a detailed model of temporal uncertainty in the execution. It deals with disturbances one at a time, making the best decision based on the available information, but it doesn't try to anticipate and compensate for all possible future disturbances that may occur.

The temporal constraint processing methods discussed in this section are an important component of our biped task executive. Use of these methods, for the biped application, is discussed in more detail in Chapters 6 and 7. However, because these methods are intended for discrete activity systems, they do not offer a complete solution for the hybrid systems, such as bipeds, that we are interested in. Therefore, we next review previous work on plan executives for hybrid systems.

2.2.3 Model-based Plan Executives for Hybrid Systems

A model-based approach has been used, recently, for path planning and control of multiple un-manned air vehicles [Leaute, 2005]. This air vehicle application uses a receding horizon *model-predictive control* (MPC) algorithm, run at regular intervals, to generate optimal control input trajectories. Desired behavior is specified using a *qualitative state plan*, which expresses goals in terms of regions in state space and temporal constraints. The problem is then formulated as a *mixed-integer linear program* (MILP). This formulation incorporates the temporal constraints, state space goal region and obstacle constraints, and simplified vehicle dynamics. The formulation is passed to an MILP solver, which produces the optimal control trajectories.

In this application, the air vehicle models are continuous. However, the overall system is hybrid because the obstacles represent discontinuities in the region of operation. This makes the problem formulation disjunctive; it implies that there are discrete choices in the path planning for the air vehicles. This disjunctive formulation is the reason that an MILP solver is required, rather than an LP solver, which is much faster.

At regular intervals, the formulation is updated, to reflect the current vehicle and environment state, and the MILP solver is run to generate a new set of control trajectories. The frequency with which this update can be performed is limited by the size of the MILP being solved. This is a function of the number of air vehicles, the number and complexity of obstacles, the size of the time horizon, and the time discretization used in the MILP formulation.

This limitation has important implications for real-time performance. If updates cannot be performed frequently enough, the system may become unstable. The reason is that between updates, the program follows the control trajectories produced by the most recent update. If a disturbance occurs between updates, it won't be accounted for until the next update. When the next update is performed, it begins with the disturbed state and generates a new control trajectory to attempt to bring the system back under control. If the disturbance is significant, and the next update doesn't happen for a long time, then the system may reach a state of disturbance from which it cannot recover.

For this reason, this type of model-predictive approach has traditionally been used only for applications whose real-time dynamics are slow enough for the algorithm to be run frequently enough to compensate for anticipated disturbances. One such type of application is chemical process control [Garcia and Prett, 1986; Cutler and Ramaker, 1979; Richalet et al., 1978]. Chemical processes change slowly enough that the update interval, for a model-predictive control algorithm, can be several seconds, or even, several minutes. At the same time, due to the high volume of materials in industrial chemical processes, small percentage changes in optimality can have significant economic benefits. These factors have led to extensive use of MPC for control of industrial chemical processes, over the past two decades.

As computers become faster, MPC will be used in an increasing number of applications, like control of autonomous air vehicles and ships. However, the performance requirements for agile robotic systems, like bipeds, are still several orders of magnitude beyond what is currently possible with a standard MPC approach.

For such applications, a promising approach, that combines the model-based advantages of MPC with classical control techniques, is to perform an off-line analysis of the plant's operating region, and thereby, to determine sets of feasible state trajectories and control laws that lead to plan success. With this approach, a partial compilation is performed that identifies such sets of feasible trajectories, or *flow tubes*, and provides guidance to the dispatcher, so that it can keep the plant within such a tube at runtime. We use the flow tube approach, as introduced in Chapter 1, and described further in Chapters 5 – 7. We next review, in section 2.2.4, previous work on flow tube analysis.

2.2.4 Plan Compilation using Flow Tubes

The problem of simplifying control by analyzing a state space offline, and dividing it into separate regions, each with a dedicated, automatically synthesized controller, so that runtime trajectories remain feasible, has been studied extensively in the qualitative control community.

One recent example of this approach is Qualitative Heterogeneous Control (QHC) [Kuipers and Ramamoorthy, 2001]. This technique synthesizes global behaviors for nonlinear dynamical systems by separating concerns of qualitative correctness from ones of quantitative optimization. Qualitative constraints are used to partially define state space region boundaries and controller requirements. These qualitative constraints are used to guide a quantitative optimization, which defines the regions and control parameters precisely.

In QHC, state space division is initially specified qualitatively, using a *transition graph*. The imprecision of the qualitative specification leaves flexibility for multiple quantitative (precise) choices for the actual region boundaries, which are then computed numerically. In QHC , the numerical computation is performed using a *Sequential Quadratic Programming* (SQP) optimization algorithm.

A key feature of QHC is the emphasis on getting from one region to another, without being too concerned about the precise way in which this happens. This is in contrast to many traditional control design techniques [Slotine and Li, 1991], whose emphasis is almost exclusively on precise tracking of a reference trajectory.

So far, QHC has been used exclusively for low-dimensional "textbook" nonlinear control problems, such as getting a pendulum, or the classic pendulum on cart to swing up and balance. While the approach works very well for such small problems, it is not clear that it would scale up to a high-dimensional problem such as biped locomotion control.

A second key limitation is that QHC does not incorporate any notion of temporal constraints. In QHC, qualitative region descriptions are given in terms of equilibrium points and orbits. The goal is, typically, to attain a global equilibrium point, at some time, by transitioning through an appropriate sequence of regions containing attractors and repelors. In contrast, our biped application requires plan execution, where the qualitative region description is simply to be in the right region at the right time.

Flow tubes have been used, in a system called MAPS, to characterize phase spaces of nonlinear dynamical systems [Bradley and Zhao, 1993]. These flow tubes are similar, in concept, to the above-mentioned activity tubes of a QCP in that they represent a set of trajectories with common characteristics that connect two regions. The MAPS system exhaustively derives all tubes of a phase space using polyhedral approximations. The tubes are formed into a graph that can be searched to find paths from one region to another. The flow graph derivation also supports related systems that perform controller synthesis by modifying control parameters to modify the phase space. Intersections of the flow graph of the modified phase space with the original phase space are found to determine points at which control parameters should be switched, in order to move a trajectory from one flow tube into another.

MAPS and associated systems have a number of significant limitations that make them unsuitable for bipedal locomotion applications. First, the emphasis of MAPS is to automatically derive a qualitative description of a state space. In our application, this qualitative description is provided, partially, as a input. The exhaustive nature of the analysis performed by MAPS limits it to low-dimensional problems; it does not scale up well to high-dimensional problems like control of bipeds. Second, as with QHC, MAPS does not incorporate temporal constraints. Its emphasis is on finding stability regions and equilibrium points, rather then on traversing regions within a limited time range.

Another system that incorporates the concepts of flow tubes, state space discretization, and controller synthesis for each state space region is the *Maneuver Automaton* [Frazzoli, 2001]. A maneuver automaton is a control framework for planning the motion of underactuated mechanical systems, such as helicopters. The motion plan is built by interconnecting appropriately selected trajectory primitives, to move from an initial state to a goal state.

As with flow tube and finite state machine approaches, a key concept behind the maneuver automaton is a discretization that reduces the computational complexity of motion planning for a constrained, nonlinear, high dimensional system, but preserves most of the flexibility of the original system with respect to optimal control. This discretization is not, however, a discretization of state, control inputs, or time, as is often done in dynamic programming problems. Rather, it is a discretization that restricts the feasible nominal system trajectories to the relatively small set of trajectories that can be obtained by the interconnection of a small set of primitive trajectories, as shown in Fig. 2.12.

As with the flow tube approach, trajectory primitives are designed to keep the system state flowing through a tube in state space. Two kinds of trajectory primitives are used: trim trajectories and maneuver trajectories. Trim trajectories are characterized by a constant control setting, and a constant velocity vector. The simplicity of these trajectories is that they are parameterized solely by start state and duration. Thus, an example of a trim trajectory is flying a helicopter in a straight line, or in a steady, constant turn. A trim trajectory can be followed for an arbitrary amount of time; it is a steady state condition. This provides significant flexibility of control; the system state can be changed an arbitrary amount, in a particular direction, while maintaining stability.

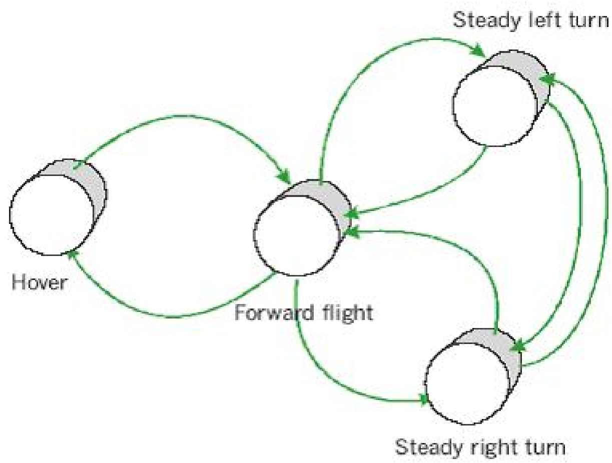


Fig. 2.12 – A simple maneuver automaton for a helicopter.

For a typical nonlinear system, trim trajectories are not enough. A dynamic system cannot transition instantaneously from one trim trajectory to another. There is an intervening point where there is some acceleration and non-constant control input, thus violating the definition of a trim trajectory. To address this, the concept of a maneuver trajectory is used. A maneuver trajectory is defined as a finite time transition between two trim trajectories. It is characterized by a specific time interval, a control action during that interval, and a state function describing the expected time evolution of system state during the interval resulting from the control action. Because a maneuver is characterized by a fixed time interval and control input during that interval, it results in a fixed change in state. Note that because maneuvers imply acceleration, they must observe the dynamic and actuation limits of the plant.

A maneuver automaton makes two key assumptions: 1) that trim trajectories can be used, and 2) that the continuous state of the plant is low dimensional. The first assumption implies that the plant can be in a steady state for significant periods of a task execution. The second assumption is necessary because a maneuver automaton uses a dynamic programming approach for its motion planning, which requires low dimensionality.

These two assumptions make a maneuver automaton unsuitable for bipedal locomotion. As mentioned previously, a key characteristic of walking bipeds is that they are constantly in a state of imbalance, and require constant corrective action. Strictly speaking, there are no relative equilibria for walking. Thus, trim trajectories cannot be used. This leaves maneuver trajectories. The problem with the completely fixed maneuver trajectories used in the maneuver automaton is that they are too inflexible. They don't provide enough capability to traverse from most initial states to most goal states, and therefore, result in a system that is uncontrollable. The second assumption (low dimensionality) is also a significant problem. The helicopter models used in the maneuver automaton typically have on the order of four state variables. In contrast, a walking robot model has 18 or more degrees of freedom, resulting in 36 or more state variables.

As stated previously in the introduction to Section 2.2, bipedal locomotion systems are hybrid due to the discontinuities that occur when transitioning between single and double support modes. As with the QHC and Maneuver Automaton approaches, many previous bipedal locomotion systems have addressed the challenges associated with the hybrid nature of the problem by dividing state space according to these qualitatively different support modes, and using a separate, dedicated controller for each mode. When the system transitions to a new mode, the executive automatically switches in a new dedicated controller for the new mode.

In most such applications, the state space division and controller synthesis is performed manually, based on experimentation with simulations and real robots, and on manual analysis [Raibert, 1986; Pratt et al., 1997]. Algorithms that perform the state space division and controller synthesis automatically have been developed recently [Hu 2001; Westerveldt, 2004], but they assume simplified planar models and regularity in the gait cycle. These assumptions allow for use of Poincare' return map analysis techniques, as discussed in Section 2.1.2. However, as discussed in Section 2.1.2, these assumptions cannot be made for the problem we address here. First, we seek to control a 3-D biped, which is much more difficult than a planar biped. The return map techniques do not scale well to such a larger problem. Second, we are concerned not with achieving a stable limit cycle, but rather, with successful plan execution. This is an important distinction. Successful plan execution may require observance of foot placement, temporal, and other state space constraints necessary for achieving locomotion tasks like traversing difficult terrain or kicking a soccer ball. Such requirements are not addressed by return map techniques; their emphasis is on achievement of a repetitive cycle.

2.3 Biomechanical Analysis

Human posture and balance have been studied extensively from a biological perspective [Nashner, 1981; Nashner and McCollum, 1985; and Rietdyk, et. al., 1999]. These studies describe the relation of muscle synergies to body balance forces. This is analogous to the computation performed by the VMC algorithm in that it involves coordinated control action of multiple muscles to produce multiple joint torques that result in appropriate restoring force on the center of mass. Kuo and Zajac (1992) derive a mathematically rigorous control system for postural response to perturbations of erect stance, again in the sagittal plane. This study addresses the issue of which synergies are most optimal in terms of force efficiency. More recently, a biologically realistic model of parts of the human cerebellum and neuromuscular control system has been used to achieve balance recovery from force disturbances to a planar model of a standing human [Jo and Massaquoi, 2004].

The Nashner and McCollum study, as well as work by Allum (1985, 1992, and 1993), describes particular multiple muscle synergies or strategies used to respond to postural perturbations, in the sagittal plane. In this literature, these strategies are referred to as the "ankle" strategy and the "hip" strategy. With the ankle strategy, balance is restored using torques generated at the ankle joint, and there is no bending at the hip. With the hip strategy, balance is restored by combined use of ankle torque, and by bending at the hip. Biological experiments show that humans prefer using the ankle strategy first, if postural disturbances are small, and resort to the hip strategy only if disturbances are large. These biological studies have observed these strategies in humans, but have not explained the underlying biomechanical causes for this behavior. Some of these studies have speculated that this behavior is a matter of preference, or is due to energy efficiency considerations.

As we explain in Chapter 3, and in the balance controller implementation of Chapter 8, these behaviors are due to biomechanical necessity. The ankle strategy corresponds to the zero-moment strategy introduced in Chapter 1. The hip strategy corresponds to the moment strategy. Thus, as explained in Chapter 3, but not in any of the biological studies, the hip strategy is used to generate angular momentum about the center of mass, in order to generate a horizontal restoring force on the center of mass.

Biological studies support the overall approach used in the VMC algorithm: that of tracking a high-level reaction frame point trajectory using spring and damper elements to implement a relatively simple feedback control system. The equilibrium point hypothesis, first proposed by Feldman (1966), and extended by Bizzi et al. (1992), and McIntyre and Bizzi (1993), is based on the idea that the neuromuscular system exhibits position-dependent properties that tend to restore the limbs to a commanded equilibrium posture. Central commands generate a sequence of such equilibrium positions, and the spring-like properties of limbs tend to drive them along such trajectories.

2.4 Summary of Limitations of Previous Work

None of the previously developed systems described above address the problem of task-level control of bipeds, with automated handling of disturbances, where the tasks are temporally and spatially demanding, and therefore, require significant agility. We now summarize the limitations of these previously developed systems for this type of problem, and discuss how we extend some of the techniques used in these systems in order to solve this type of problem.

As described in Section 2.1, the ZMP method, and the detailed joint trajectory planning methods are not robust to significant disturbances. Additionally, the ZMP method does not support the use of moment balance strategies of the type introduced in Chapter 1 and discussed further in Chapter 3. Also, the ZMP method is overly conservative in its requirement that the support foot or feet be flat on the ground. Finally, the ZMP method is, fundamentally, an open-loop control method, as explained in Section 2.1.1. Analysis using Poincare return maps is limited to applications where gait is periodic, and where the goal is to achieve a stable limit cycle. This is not our goal here; we are concerned with successful execution of locomotion tasks, where stepping patterns and timing may be uneven, as explained previously. Therefore, Poincare return map analysis cannot be used for the types of problems considered in this thesis.

Virtual model control methods are promising, but are limited because they do not take dynamics into account. We solve this problem by using a dynamic virtual model controller, as introduced in Chapter 1 (see Fig. 1.14). Our dynamic virtual model controller has the same goal as the virtual model controller: allowing the biped to be controlled using virtual elements, as if it were a puppet. However, the dynamic virtual model controller, described further in Chapter 8, has superior performance because it takes into account the biped's dynamics.

The linearization and decoupling provided by this controller also allow adaptation of the flow tube techniques discussed in Section 2.2. As described in Section 2.2.4, previously developed systems that perform flow tube analysis, such as MAPS, are limited to plants with low dimensionality. The linearization and decoupling provided by the dynamic virtual model controller allows for use of a very simple flow tube representation, described in Chapters 5, 6, and 7. This results in a loosely coupled system that has properties similar to the discrete state systems described in Section 2.2.1. For such systems, multiple, parallel activities can be executed independently as long as their pre and post conditions, and their temporal constraints, are satisfied. These properties allow us to leverage plan execution techniques used for the discrete state systems, particularly, the temporal processing algorithms described in Section 2.2.2.

3. Biomechanical Analysis of Balance Requirements and Constraints

Balance control is essential for performing walking tasks robustly. Balance control requires the ability to adjust the biped's linear and angular momentum. Due to conservation of momentum laws, such adjustment can only be achieved through force interaction with the environment. For a biped, this force interaction is comprised of gravity and the *ground reaction force*, the net force exerted by the ground against the biped. This chapter presents an analysis of physical constraints and requirements for balancing. This leads to a simple, comprehensive model of balance control that specifies coordination of control actions that adjust the ground reaction force, and therefore, the momentum of the biped.

Similar models have been used previously in a number of gait planning algorithms [Kajita et al., 2001; Yokoi et al., 2001; Sugihara et al., 2002; Nishiwaki et al., 2002]. These models, as well as ours, regulate a biped's linear and angular momentum. The key difference is that our model is able to purposely sacrifice angular momentum control goals in order to achieve linear control goals when both cannot be met. Additionally, our novel contributions for this model consist of a biological validation of the model against human walking trial data [Popovic, et al. 2004a], a description of the three bipedal balance strategies introduced in Section 1.4.3 in terms of this model, and a characterization of disturbances in terms of their effects on this model.

A model of balance control that is simple is extremely useful for achieving efficient planning and control, which is necessary for real-time operation. In such a model, we seek a level of abstraction that captures the essential requirements of balance control, without the complex details of individual joint motions. Thus, the plan compiler component of the model-based executive, introduced in Chapter 1, uses this model to generate the qualitative control plan. The multivariable controller component then generates the detailed joint motions based on the control plan.

To derive this simplified model, we make use of a number of physical points that summarize the system's balance state. These points are the *center of mass* (CM), the *zero-moment point* (ZMP) [Vukobratovic and Juricic, 1969], and the *centroidal-moment* *point* (CMP) [Popovic et al., 2005]. As we will discuss in more detail, the ZMP is a point on the ground that represents the combined force interaction of all ground contact points. The CMP is the point on the ground from which the ground reaction force would have to emanate if it were to produce no torque about the CM.

We define the biped's support base as the smallest convex polygon that includes all points where the feet are in contact with the ground. This is a standard concept for bipedal walking [Hirai, 1997]. When in *single support*, that is, where one foot, the *stance* foot, is on the ground and the other is stepping, the support base is the outline of the part of the stance foot that is in contact with the ground. When in *double support*, that is, where both feet are on the ground, the base of support is the convex polygon that includes all points where the two feet are in contact with the ground.

The ground reaction force vector, $\mathbf{f}_{gr}$, is then defined as the integral, over the base of support, of the incremental ground reaction forces emanating from each point of contact with the ground. This is expressed as

$$\mathbf{f}_{gr} = \iint_{B.O.S} \mathbf{f}_{gr}(x, y)\,dxdy$$

where $\mathbf{f}_{gr}(x, y)$ is the incremental force at point x,y on the ground, and B.O.S refers to the base of support region.

The CM is the weighted mean of the positions of all points in the system, where the weight applied to each point is the point's mass. Thus, for a discrete distribution of masses m_i located at positions $\mathbf{r}_i$, the position of the center of mass is given by

$$CM = \frac{\sum_i m_i \mathbf{r}_i}{\sum_i m_i}.$$

A bipedal mechanism consists of a set of articulated links, each of which is a rigid body with mass m_i. Each rigid body has its own CM at a point $\mathbf{r}_i$. Thus, the above definition applies to bipedal mechanisms.

The CM represents the effective mass of the system, concentrated at a single point. This is valuable because it allows us to simplify the balance control problem by reducing the problem to keeping the CM in the right place at the right time. Furthermore, the control dynamics of this point is expressed, simply, by Newton's law, $\mathbf{f}_{gr} = ma$, where, in this case, m is the total mass of the system, and a is the resulting acceleration of the CM.

The ZMP [Vukobratovic and Juricic, 1969] also is a point that represents a combination of distributed points. It is defined as the point on the ground, where the total moment generated due to gravity and inertia is 0 [Takanishi et al., 1985]. This point has been shown to be the same as the *center of pressure* [Goswami, 1999], which is the point on the ground where the ground reaction force acts. Because the base of support is defined by the convex polygon of points in contact with the ground, and because the ZMP represents the average force contribution of these points, the ZMP is always inside the biped's base of support [Goswami, 1999].

The CMP is the point on the ground, not necessarily within the support base, from which the observed net ground reaction force vector would have to act in order to generate no torque about the CM. As will be shown, the relationship between the CM and CMP then indicates the specific effect that the net ground reaction force has on CM translation. Because the observed net ground reaction force always operates at the ZMP which is within the support base, whenever the net ground reaction force generates no torque about the CM, then the ZMP and CMP coincide. If the net ground reaction force generates torque, however, then the CMP and ZMP differ in location, and, in particular, the CMP may be outside the support base. As we will see, sometimes it is useful to have the net ground reaction force produce a torque about the CM. In this case the CMP can be displaced from the ZMP which reflects the increased ability of the net ground reaction force to affect translation of the CM. Viewing CM translation control in terms of CMP displacement turns out to be a useful simplifying technique.

This capability of producing torque about the CM comes at an expense, however. While translational controllability of the CM is improved, angular stability about the CM is sacrificed. Thus, for example, the torso may deviate from its upright posture. In many situations, such a sacrifice is worthwhile if the angular instability is bounded and temporary. For example, a tightrope walker will tolerate temporary angular instability if this means that he won't fall off the tightrope.

A model of balance control, where requirements for balance are expressed in terms of CM, ZMP, CMP, and the support base is extremely useful for planning and control, due to its simplicity. Balance control is then reduced to a problem of adjusting the base of support, adjusting the ZMP within the base of support, and, if necessary, performing motions that generate angular momentum, so that the CMP can be moved, temporarily, outside the base of support, in order to exert additional compensating force on the CM.

Our analysis shows that the details of joint movement are determined, to a large extent, by the physical requirements, constraints, and goals of the task to be performed, and by the morphology of the biped itself. For example, biomechanical observations of normal human walking have shown that angular momentum remains small throughout the gait cycle. Thus, regulating angular momentum appears to be an important goal for humans during normal walking. This makes sense; large oscillations in angular momentum, due, for example, to exaggerated tilting of the body forward or back, or side to side, results in significant wasted energy.

This regulation of angular momentum results in simple relations between the CM and ZMP points, and simple methods for predicting horizontal center of mass forces, which can be applied to planning and control. This result also leads to an important question: are there situations during walking or balancing when angular momentum is not conserved? Consider the case where foot placement is constrained. Such constraints are due to the combination of environmental restrictions on where feet can be placed, and constraints due to the morphology of the biped. An example of a morphological constraint is when the robot's leg is too short to reach a particular foot placement. When foot placement is constrained, it may be impossible to adjust the support base so that the ZMP can be moved to a point required for CM controllability. In such cases, it becomes necessary to move the CMP outside the support base by generating appropriate angular momentum about the CM. The way in which this angular momentum is generated depends on a combination of factors including joint position, velocity, and acceleration limits, posture requirements, and, possibly, limits due to obstacles in the environment.

The first section in this chapter, Section 3.1, Clues from Human Walking Trials, describes a series of experiments involving human test subjects performing normal walking tasks. We describe our observations regarding regulation of angular momentum during these tasks, and relations between CM and ZMP that are consistent with these observations. The section concludes with an introduction of the CMP, and a description of how it can be used to enhance balance control. This enhancement of balance control is the *moment* strategy, introduced in Section 1.4.3, and is a key novel contribution of our work.

The second section, Section 3.2, Enhancing Balance Control Through Use of Non-Contact Limb Movement, provides a detailed analysis of movements used to generate moment about the CM, in order to enhance horizontal controllability of the CM. The analysis provides details about how non-contact limb movement is used to generate this moment, and analyzes limits of this movement, and associated limits on the degree to which CM control can be enhanced.

The last section, Section 3.3, Disturbance Metrics and Classification, provides a classification of different types of disturbances, and characterizes them in terms of their effect on the previously introduced balance model. This section also discusses strategies for handling the disturbances, and metrics that help classify the severity and type of disturbance. These metrics are also extremely important for efficient planning and control of walking tasks. In particular, they are used by the model-based executive to evaluate the stability of the biped during execution of walking tasks.

Appendix E, Balance Recovery Through Stepping, analyzes how the support base should be adjusted, by stepping, in order to fulfill balance requirements.

3.1 Clues from Human Walking Trials

Observation of normal human walking yields important clues and guidelines for how balance is maintained. To this end, we performed a series of human walking trials to determine underlying principles of balance control that could be applied to bipedal walking machines. In particular, the study focused on the observation of angular momentum during different phases of a gait cycle.

A key result from this study is that angular momentum is tightly regulated during normal walking. At the end of this section, we also investigate whether there are situations during walking or balancing when angular momentum is not conserved. This question is addressed in more detail in section 3.2, but here, we introduce the concept, and formally define the CMP, which is useful for investigating this question.

The next sub-section introduces the concept of angular momentum conservation during normal human walking, and discusses reasons for studying it. The following sub-section describes how the walking trials were performed, and how data was collected and analyzed. The subsequent sub-section describes important rules extracted from careful analysis of the data. These include relations between center of mass and center of pressure, and methods for predicting horizontal center of mass forces. Finally, we show how these methods can be used for prediction and planning, and use the CMP to introduce situations where the rules are broken.

3.1.1 Motivation for human walking trials: determination of the tightness of angular momentum conservation

It is a fundamental law of nature that the angular momentum of a body about its center of mass (CM) is conserved in the absence of external forces. For example, a brick tumbling in space will continue to tumble with the same angular momentum until some external force acts on it. This is expressed as:

$$\mathbf{L}_{CM} = \mathbf{k} \qquad (3.1)$$

$$\frac{d\mathbf{L}_{CM}}{dt} = 0$$

where $\mathbf{L}_{CM}$ is the angular momentum about CM, and $\mathbf{k}$ is a constant vector. Conversely, a non-zero torque, $\boldsymbol{\tau}_{CM}$, about CM, due to external forces, implies a non-constant angular momentum:

$$\boldsymbol{\tau}_{CM} = \frac{d\mathbf{L}_{CM}}{dt} \tag{3.2}$$

Eqs. 3.1 and 3.2 apply to both rigid bodies, such as bricks, and to articulated bodies, such as humans or bipedal walking machines, consisting of multiple segments such as torso, upper leg, and lower leg. Thus, just as is the case for the above-mentioned brick, a human, or humanoid robot tumbling in space will continue to tumble at constant angular momentum until acted on by an external force.

For a rigid body, angular momentum is related to velocity by

$$\mathbf{L}_{CM} = \mathbf{I}\boldsymbol{\omega} \tag{3.3}$$

where $\mathbf{I}$ is a constant inertia matrix, and $\boldsymbol{\omega}$ is the angular velocity vector. In the absence of external torques, this angular momentum is conserved. For an articulated body, although the angular momentum of individual segments may not be constant, the angular momentum of the entire system about its center of mass is conserved, in the absence of external torques. For an articulated system, $\mathbf{L}_{CM}$ is related to segment velocities by

$$\mathbf{L}_{CM} = \sum_i \left(\mathbf{I}_i\boldsymbol{\omega}_i + m_i\,\mathbf{v}_i \times \left(\mathbf{G}_i - \mathbf{G}\right)\right) \tag{3.4}$$

where i indicates the segment, m_i is the mass of the segment, $\mathbf{I}_i$ is the constant inertia matrix of the segment, $\boldsymbol{\omega}_i$ and $\mathbf{v}_i$ are the angular and linear velocities of the segment, $\mathbf{G}_i$ is the CM position of the segment, and $\mathbf{G}$ is the CM position of the entire articulated system, as shown in Fig. 3.1. The positions $\mathbf{G}$ and $\mathbf{G}_i$ are expressed in a global coordinate frame, with origin at point $\mathbf{o}$.

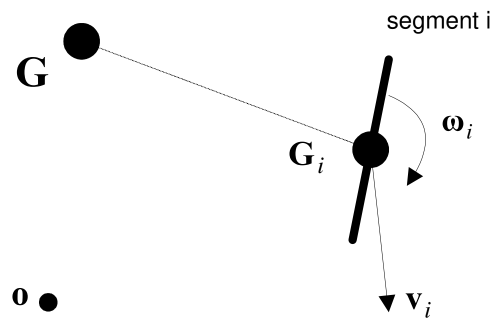


Fig. 3.1 – Articulated body segment.

For a walking biped, external forces are generated by contact of the feet with the ground. Therefore, Eq. 3.2 applies, and a non-zero torque may be generated about the CM. However, when averaged over a long enough time, such as the time needed to take multiple steps, $\mathbf{L}_{CM}$ has to be 0, or the body would tip over. Thus, even though the instantaneous CM torque may be nonzero, as in Eq. 3.2, the torque averaged over time has to be 0, as in Eq. 3.1. The key question is how small the time interval is over which Eq. 3.1 is accurate. In other words, we have to consider how tightly angular momentum is regulated during human walking. The human walking trial study was performed to answer this question. The answer is important because it reveals basic balance control properties that are useful for control of bipedal walking machines.

The next section describes how the walking trials were performed, and how data was collected and analyzed. The subsequent section describes some important rules extracted from careful analysis of the data. These rules summarize properties of conservation of angular momentum about the center of mass during normal walking. Finally, we show how these rules can be used for prediction and planning, and situations where the rules are broken.

3.1.2 Human Walking Trial Data Collection and Analysis

A 104 Kg, male test subject was used to collect position and force data for self-selected slow, medium, and fast walking speeds [Popovic, et al. 2004a]. Trajectory data was collected using a Vicon motion capture system [Vicon, 2002a.]. Infrared-reflecting markers were attached to appropriate points on the legs, pelvis, and torso of the test subject. The Vicon system then combined the inputs from 12 separate infrared cameras to generate three-dimensional motion trajectories for the markers. The error of this system is typically less than one millimeter. The Vicon system, using Bodybuilder software [Vicon, 2002b.], then automatically computed joint center positions based on marker position and morphological measurements taken on the test subject. In addition to the motion trajectories, two force plates [AMTI, 2001], one for each foot, were used to measure ground-reaction force. The error of this system is typically less than one tenth of a Newton. Sampling frequencies for motion and force data collection were 120 Hz and 1080 Hz, respectively. Matlab interpolation functions [Matlab, 2004a.] were used to filter the force data to make time intervals between data points consistent with the motion data time intervals. The time interval used for both human and simulation data was 0.0012 seconds [Hofmann et al., 2002].

A morphologically realistic model consisting of 16 links and 32 degrees of freedom was used to calculate angular momentum [Popovic, et al. 2004a]. Model segment dimensions and inertias were carefully computed to match those of the test subject. Kinematic trial data segment positions and velocities were applied to the model through Eq. 3.4 to compute $\mathbf{L}_{CM}$. A quantity we call the *effective angular velocity*, $\boldsymbol{\omega}_{eff}$, was then computed using

$$\boldsymbol{\omega}_{eff} = \mathbf{I}_{eff}^{-1}\mathbf{L}_{CM} \tag{3.5}$$

where $\mathbf{I}_{eff}$ is the effective or *whole body* inertia tensor about the CM. This inertia is a non-constant function of segment position. Thus, it was computed by applying segment position trial data to the model over time. Integrating $\boldsymbol{\omega}_{eff}$ yields a quantity we call the *effective angle*, $\boldsymbol{\theta}_{eff}$:

$$\boldsymbol{\theta}_{eff} = \int \boldsymbol{\omega}_{eff}\, dt \tag{3.6}$$

This quantity gives a good indication of angular stability. Because $\boldsymbol{\theta}_{eff}$ is the integral of $\boldsymbol{\omega}_{eff}$, if $\boldsymbol{\theta}_{eff}$ remains small throughout the gait cycle, this is an indication that angular momentum is tightly regulated.

3.1.3 Results on Conservation of Angular Momentum and Relation between CM and ZMP

The results of this analysis show that maximum excursions for $\boldsymbol{\theta}_{eff}$ remain small throughout the gait cycle. The angular excursion in the sagittal, transverse, and coronal planes was less than 1, 2, and 0.2 degrees, respectively. This shows that angular momentum is tightly conserved during normal walking.

There has been some debate about whether this tight conservation is due to direct control of angular momentum by the human central nervous system, or whether this property emerges naturally due to other factors. In retrospect, it is not surprising that conservation is tight during normal walking, given that most of the inertia is in the torso, and the nervous system is undoubtedly acting to keep the torso and the head at a relatively constant, upright orientation. The legs move primarily in the sagittal plane, forward and backward, but the momentum of one tends to cancel the momentum of the other since the swing leg moves forward as the stance leg moves back.

Regardless of the reason that angular momentum is tightly conserved, we can make use of this observation in the design of controllers for bipedal walking machines, in order to simplify control of normal walking. It is important to note, however, that, while this property is true for normal walking, it is not always true. Summersaults, spinning about the vertical axis, cartwheels, walking around a corner, bowing at the waist, and a variety of common athletic and dance maneuvers all violate this property. Furthermore, as we will see in Section 3.1.5, there are important balance situations where angular momentum should not be tightly conserved.

We now discuss the force relation between CM and ZMP for the case where angular momentum is tightly conserved. We use this in our evaluation of the biological test data. Furthermore, a simple relation that expresses how CM is accelerated as a function of ZMP position is useful for planning and plan compilation.

If we assume that the only external force acting on the system is the force exerted against the ground, then the ZMP represents the point at which all ground reaction forces act, as discussed previously. This point is also called the zero-moment point (ZMP) in the robotics literature [Vukobratovic and Juricic, 1969]. This point can be used to express the torque about the CM:

$$\boldsymbol{\tau}_{CM} = (\mathbf{r}_{ZMP} - \mathbf{r}_{CM}) \times \mathbf{f}_{gr} \tag{3.7}$$

where $\mathbf{r}_{ZMP}$ is position of the ZMP, $\mathbf{r}_{CM}$ is position of the CM, and $\mathbf{f}_{gr}$ is the ground reaction force. Assuming perfect angular momentum conservation, we set $\boldsymbol{\tau}_{CM}$ to 0 and define $\mathbf{r}_{CZ} = (\mathbf{r}_{ZMP} - \mathbf{r}_{CM})$, so Eq. 3.7 becomes

$$\mathbf{0} = \mathbf{r}_{CZ} \times \mathbf{f}_{gr} \tag{3.8}$$

Solving for the horizontal components of $\mathbf{f}_{gr}$ yields

$$f_x = r_x \frac{f_z}{r_z} \qquad f_y = r_y \frac{f_z}{r_z} \tag{3.9}$$

where r_x, r_y, and r_z are the forward, lateral, and vertical components of $\mathbf{r}_{CZ}$, respectively, and f_x, f_y, and f_z are the corresponding components of $\mathbf{f}_{gr}$. Eq. 3.9 can be expressed as

$$f_x = k\, r_x \qquad f_y = k\, r_y \tag{3.10}$$

where $k = f_z / r_z = -f_z / Z_{CM}$ can be regarded as a non-constant vertical spring stiffness. Eq. 3.10 is a simple relation that provides CM force as a function of the difference between

CM and ZMP horizontal positions. This is useful for computing CM flow tubes based on ZMP restrictions due to foot placement.

Another way to view Eqs. 3.8 - 3.10 is to note that if $\boldsymbol{\tau}_{CM}$ is 0, then $\mathbf{f}_{gr}$ points from the ZMP position directly towards the CM position, as shown in Fig. 3.2.

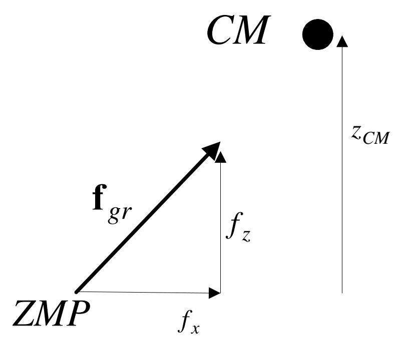


Fig. 3.2 – If angular momentum is perfectly conserved, the ground reaction force vector points from the CP directly toward the CM.

Finally, note that Eq. 3.10 can be expressed as a relation between horizontal CM, ZMP, and horizontal CM acceleration through a second order differential equation:

$$F_x = m\,\ddot{x}_{CM} = k\delta_x = k(x_{ZMP} - x_{CM}) \qquad (3.11)$$
$$F_y = m\,\ddot{y}_{CM} = k\delta_y = k(y_{ZMP} - y_{CM})$$

or

$$\ddot{x}_{CM} = \frac{k}{m}(x_{ZMP} - x_{CM}) \qquad (3.12)$$
$$\ddot{y}_{CM} = \frac{k}{m}(y_{ZMP} - y_{CM})$$

where m is total mass. Eq. 3.12 provides an expanding, spring-like, relation between horizontal CM acceleration, and the difference between horizontal ZMP and CM positions. As stated previously, this is useful for computing CM flow tube limits for a particular foot placement. This is because the ZMP is restricted to be within the support polygon defined by the foot placement, and Eq. 3.12 relates CM movement to ZMP position.

3.1.4 Prediction of Horizontal Forces

It should now be possible to use Eqs. 3.10 – 3.12 to make a variety of trajectory predictions. Eq. 3.10 is based on Eq. 3.8, which makes the assumption that spin angular momentum is perfectly conserved. We begin by validating this assumption. This can be accomplished by using the equation to predict horizontal forces, based on measured CM and ZMP trajectories from the trial data, and then comparing the force predictions with measured force trajectories.

Fig. 3.3 shows the result of this validation. The predicted force trajectory, the thick red line, is in good agreement with the measured force trajectory, the thin blue line, and is within a standard deviation across 7 walking trials. In Fig. 3.3, the horizontal axis represents 0% to 50% of the gait cycle, spanning from the middle of a single support phase to the middle of the next single support phase of the opposite leg.

This result is encouraging in that it validates the assumption behind Eq. 3.10, but it is not, by itself, very useful for flow tube computation. For such computation, we need a relation between CM movement and ZMP. Eq. 3.12 provides such a relation, and since it is based on Eq. 3.10, the validation of Eq. 3.10 suggests that Eq. 3.12 can be validated in a similar way.

Consider Eq. 3.12 re-arranged as follows:

$$\ddot{x}_{CM} + \frac{k}{m} x_{CM} = \frac{k}{m} x_{ZMP} \qquad (3.13)$$

$$\ddot{y}_{CM} + \frac{k}{m} y_{CM} = \frac{k}{m} y_{ZMP}$$

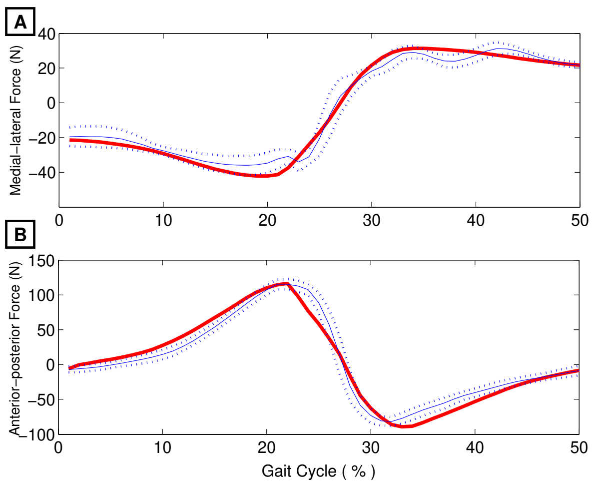
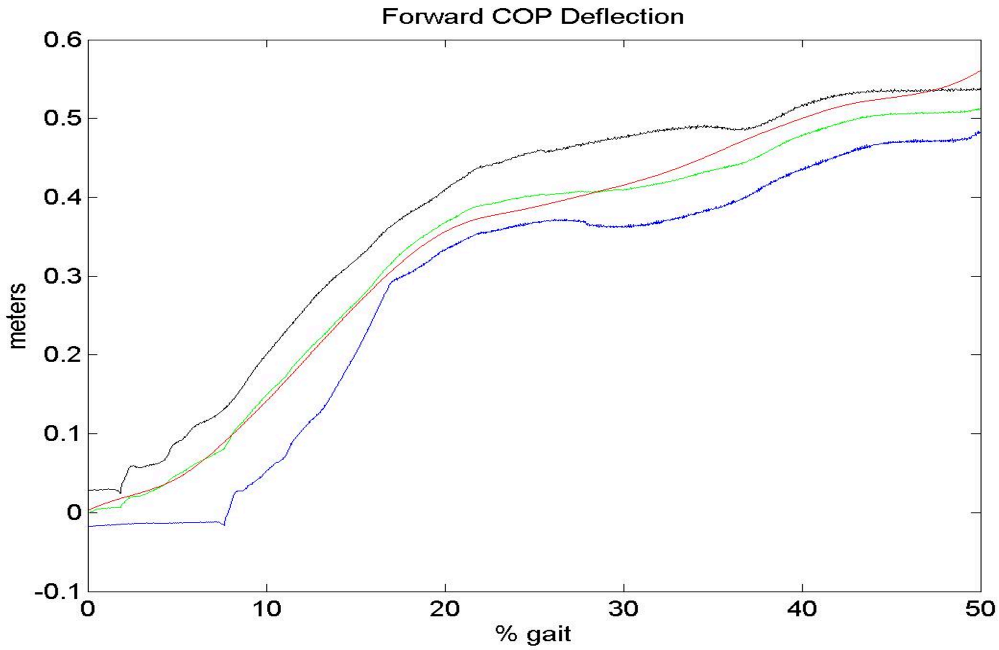


Fig. 3.3 – a. Lateral force prediction, b. forward force prediction.

Prediction is shown as a thick red line. Average measured value over 7 trials is shown as a thin blue line. Standard deviation bounds for trials are shown as dotted lines.

This shows horizontal CM as the output of two 2nd-order differential equations, where horizontal ZMP is the input. Unfortunately, these equations are not linear, because k is not a constant. However, it is worth investigating whether a linearization is possible, by assuming k to be constant. A linear form of this relationship would further simplify computation of flow tubes.

A very interesting predictive test is to begin with trajectories for x_{CM} and y_{CM}, differentiate twice to get $\ddot{x}_{CM}$ and $\ddot{y}_{CM}$, and then compute x_{ZMP} and y_{ZMP} using Eq. 3.13 and an empirically determined constant value for k. This constant can be determined by averaging the true value for k over the entire gait cycle for several trial gait cycles. Fig. 3.4 shows the ZMP predictions from this test.



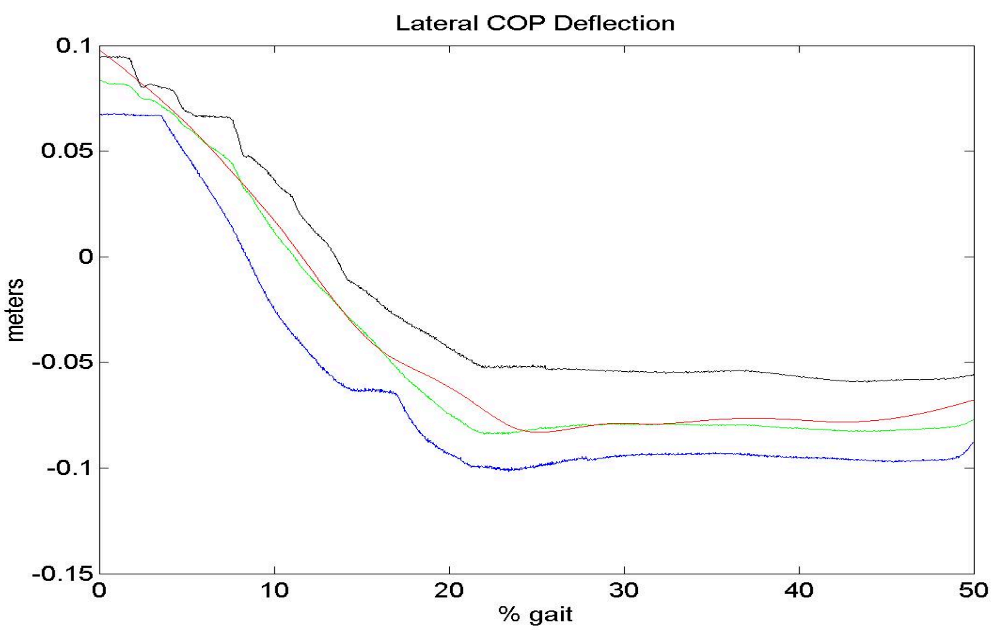


Fig. 3.4 – Forward and lateral ZMP (COP) prediction. Predictions are in red, the average over 7 trials is in green, and standard deviation bounds are in black and blue. The close agreement between the model prediction and the trial data validates Eq. 3.13, with k being constant.

These results show good agreement between predicted and average values. This indicates that the constant k assumption, which results in a simple linear differential relation between horizontal components of CM and ZMP (Eq. 3.13), is valid.

3.1.5 Non-conservation of Angular Momentum and the Zero Torque Center of Pressure

The previous discussion shows that tight regulation of angular momentum is a useful property for control during normal walking. However, it is not guaranteed for all balancing tasks, and it is important to investigate situations where this property does not hold.

Fig. 3.2, above, demonstrated that, when $\boldsymbol{\tau}_{CM}$ is 0, the ground reaction force vector points to the CM. Conversely, if this vector does not point to the CM, then a non-zero torque is generated about the CM, as shown below in Fig. 3.5.

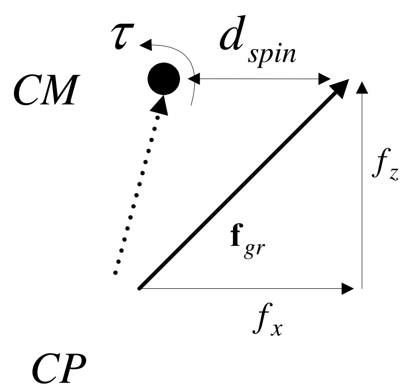


Fig. 3.5 – When $\mathbf{f}_{gr}$ does not point towards the CM, a torque is generated about the CM.

The moment arm for vertical component of the ground reaction force is d_{spin}. The introduction of this non-zero torque is generally not beneficial since it interferes with a control goal of upright orientation. Note, however, that it is beneficial in the sense that the horizontal component of the force is potentially larger for this case than for the case shown in Fig. 3.2, where torque about the CM is 0. As shown in Fig. 3.5, F_x is greater than it would be if $\vec{\mathbf{F}}$ pointed directly at the CM, as it does in Fig. 3.2. This is important because controlling horizontal movement of CM is more important for maintaining balance than controlling orientation, as long as disturbances to orientation are temporary and bounded. Recall that the ZMP is constrained to be inside the base of support. If $\vec{\mathbf{F}}$ is required to always point directly at the CM, then the maximum horizontal force that can be exerted is constrained, because $\vec{\mathbf{F}}$ begins at the ZMP, and the ZMP must be inside the base of support. Allowing $\vec{\mathbf{F}}$ to point away from the CM provides a way to overcome this limit.

Another way to overcome the ZMP limit, without sacrificing angular stability, is to increase the polygon of support, by taking a step. This is not always possible, however. If foot placement is constrained, as when walking on a tightrope or balance beam, it may not be possible to extend the support base in the desired way. In such cases, a temporary sacrifice of angular stability to gain greater horizontal force on the CM, as shown in Fig. 3.5, is well worth it, in order to avoid a fall off the tightrope. The important requirement, if this course of action is chosen, is that the disturbance to angular stability be temporary and bounded. Otherwise, biped segments involved in the angular disturbance, such as the torso, will deviate significantly from their nominal upright posture, and will ultimately exceed their limits; there is a limit to how far a human can bend at the waist.

A useful quantity for representing this situation, where a spin torque is generated about the CM, is the *Centroidal Moment Point* (CMP) [Popovic, et al. 2005]. The CMP is the point, not necessarily inside the base of support, where the ZMP would have to be, in order for the ground reaction force vector to pass through the CM, as shown in Fig. 3.6. The distance between the ZMP and the CMP is the moment arm, d_{spin}. This distance represents the additional horizontal force that is exerted due to the fact that the CMP is further from the CM than the ZMP. This moment arm also causes a disturbance to the nominal orientation.

Use of the CMP is demonstrated in Fig. 3.7, which depicts recovery from a lateral disturbance. This sequence shows an initial disturbance that pushes the system to the right. To compensate, the system takes control actions involving rotation of the body and swing leg, that move its CMP to the right, creating a lateral compensating force to the left. Because the disturbance is significant, the CMP moves beyond the edge of the support polygon, and thus, it does not coincide with the ZMP. This compensating action corresponds to a clockwise torque about the CM, which is manifested by clockwise rotation of the torso and right leg.

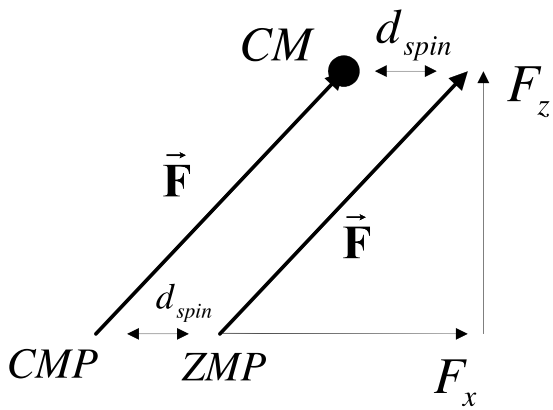


Fig. 3.6 – The CMP is the point where the ZMP would have to be in order for the ground reaction force vector to pass through the CM.

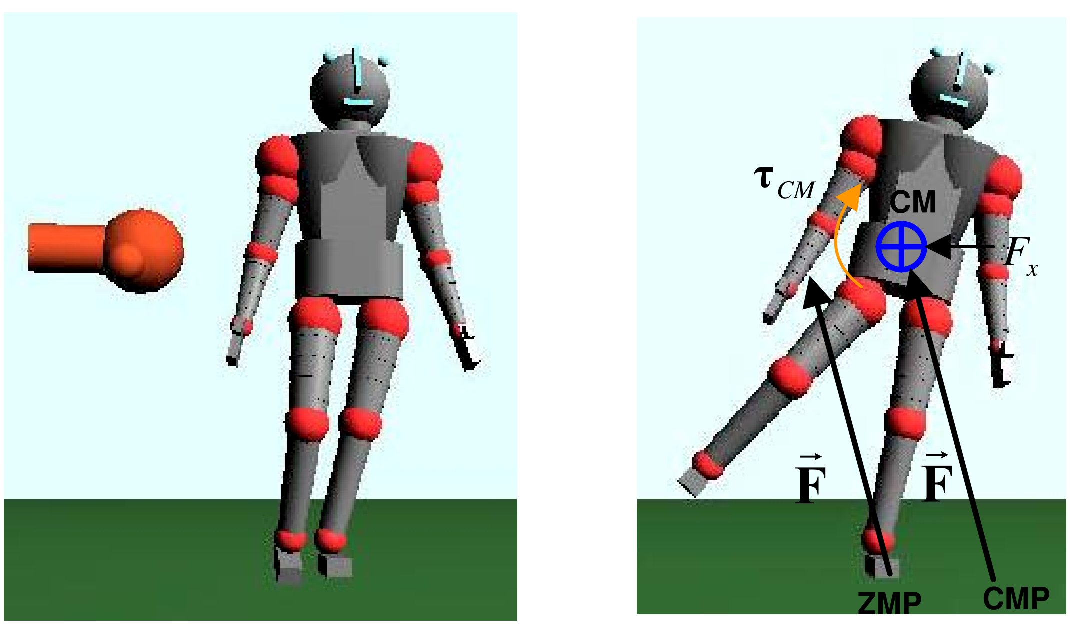


Fig. 3.7 – Recovery from lateral disturbance using CMP.

The next section discusses requirements for keeping the disturbance to angular stability temporary and bounded. These are essential requirements for our control scheme, because, as mentioned previously, there is a limit to how much orientation of the torso and other biped segments can deviate from their nominal orientations.

3.2 Enhancing Balance Control Through Use of Non-Contact Limb Movement

The previous section introduced the idea of moving the CMP outside the support polygon, in order to provide enhanced lateral force control on the CM. This involves generation of a torque about the CM. This section provides details about how non-contact limb movement is used to generate this torque, and analyzes limits on this movement, and associated limits on the degree to which CM control can be enhanced.

Before proceeding to the control of high-dimensional humanoid models, it is useful to first investigate simplified models. This is because stability limits are much easier to compute for simplified models than for high-dimensional humanoid ones. Analysis of simplified models allows for the derivation of conservative limits that can, successfully, be applied to the humanoid models. The study of simplified models also simplifies the derivation of simple, direct control laws that can then be extended for use in the full models.

One of the most important constraints in bipedal walking is that, in single support, the stance foot should remain flat on the ground, except at heel strike and toe-off. Unlike a robot manipulator that is attached firmly to a base, the stance foot is not firmly attached to the ground. There is no guarantee, from the mechanism's structure, that the foot will not roll or otherwise lose contact with the ground when it isn't supposed to. Therefore, the stance leg must be controlled carefully, in order to prevent this. In particular, ankle torques must be limited so that the foot does not roll unexpectedly.

One way to ensure the level foot constraint is to ensure that the *Foot Rotation Indicator* (FRI) point [Goswami, 1999] remains within the polygon of support. As we will see, this constraint greatly restricts direct controllability of the CM; it makes the system underactuated. Underactuated systems are, in general, difficult to control, because they are characterized by a scarceness of equilibrium points. The limits on control action require carefully integrated planning and control, and in many cases, such as walking, no actual equilibrium point is ever achieved. Instead, the system strives to achieve *limit cycle stability* by cycling through a sequence of regions in state space, none of which, themselves, contain equilibrium points.

The model discussed below is very simple compared to a full humanoid model. Nevertheless, this simplified model captures the essence of the FRI constraint, and can, therefore, be used to analyze control tradeoffs in a multivariable control context.

3.2.1 Simplified 2-link Model

Consider the simplified two-link 2-D model shown in Fig. 3.8.

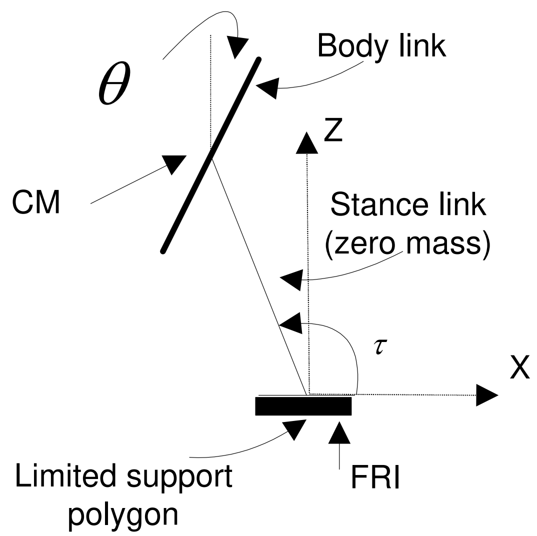


Fig. 3.8 – Simplified Model.

The model consists of two links: a stance link, representing the stance leg, which is assumed to have zero mass, and a body link, representing the upper body, head, arms and swing leg, lumped together. The body link in this model is symmetric about its joint with the stance link, so the CM of the system is always located at this joint. The base of support is limited in its length. The model has actuators at the stance link-ground joint, and at the body link-stance link joint.

The FRI equation (see Eq. 3.3.1 in Section 3.3) for this model is

$$(x_{CM} - x_{FRI})M_1 g = (x_{CM} - x_{FRI})M_1 \ddot{z}_{CM} - M_1 \ddot{x}_{CM} z_{CM} - I\ddot{\theta} \qquad (3.2.1)$$

where x_{FRI} is the horizontal position of the FRI point, x_{CM} and z_{CM} are the horizontal and vertical CM position, M_1 is the mass of the body link, and I is its inertia. If the angle between the stance link and the vertical axis is small, or if movements are relatively slow, then $\ddot{z}_{CM}$ can be assumed to be 0, and Eq. (3.2.1) becomes

$$(x_{CM} - x_{FRI})M_1 g = -M_1 \ddot{x}_{CM} z_{CM} - I\ddot{\theta} \qquad (3.2.2)$$

or

$$-x_{FRI} M_1 g = -x_{CM} M_1 g - M_1 \ddot{x}_{CM} z_{CM} - I\ddot{\theta} \qquad (3.2.2a)$$

The left hand side term is the stance ankle torque:

$$\tau = -x_{FRI} M_1 g \qquad (3.2.3)$$

The first two terms on the right-hand side are the orbital torque, which is the torque of the CM about the origin:

$$\tau_{orbital} = -x_{CM} M_1 g - M_1 \ddot{x}_{COM} z_{COM} \qquad (3.2.4)$$

This is the rate of change of orbital angular momentum of the CM. The third term on the right-hand side of Eq. 3.2.2a is the spin torque, which is the torque about the CM

$$\tau_{spin} = -I\ddot{\theta} \qquad (3.2.5)$$

This is the rate of change of the spin angular momentum about the CM. Eq. 3.2.1 can then be rewritten as

$$\tau = \tau_{orbital} + \tau_{spin} \qquad (3.2.6)$$

which shows the tradeoff between orbital and spin terms. In particular, the orbital torque, which is the torque of the CM about the origin, and therefore, produces a horizontal force on the CM, is generated by a combination of stance ankle torque, and spin torque, which results from angular movement of segments about the CM. Note that if there is no actuation at the stance ankle, then orbital and spin components must balance, as would be expected from conservation of angular momentum. Now, suppose that the support polygon extends from the origin in both directions along the x axis by an amount $x_{\text{supp\_bound}}$. To prevent the foot from rolling, the FRI position must stay within this bound:

$$\left|x_{FRI}\right| \le \left|x_{\text{supp\_bound}}\right| \tag{3.2.7}$$

This imposes a limit on the stance ankle torque

$$\left|\tau\right| \le \left|x_{\text{supp\_bound}}\right| M_1\, g = \tau_{\max} \tag{3.2.8}$$

The most important variable to control in order to maintain balance is x_{CM}. Let's suppose that we use an input-output linearization [Slotine and Li, 1991] to linearize and decouple the system so that the state vector is $\left[x_{CM},\ \ \dot{x}_{CM},\ \ \theta,\ \ \dot{\theta}\right]$, where $[x_{CM},\ \ \dot{x}_{CM}]$ is decoupled from $\left[\theta,\ \ \dot{\theta}\right]$. Suppose that $\ddot{x}_{CM}$ is computed based on a simple PD control law. Then, the trajectory for $\tau_{orbital}$ is known, assuming the system is properly linearized, and x_{CM} follows the trajectory for a simple decoupled linear second-order system.

The bound of Eq. 3.2.8 divides the control state space into two different regions, according to whether the stance ankle torque is at its limit. We analyze these regions to determine stability of the system from any initial condition.

We define the first region as

$$-\tau_{\max} \le \tau_{orbital} \le \tau_{\max} \tag{3.2.9}$$

In this region, from Eq. 3.26, the bounds on τ_{spin} are

$$-\tau_{\max} + \tau_{orbital} \le \tau_{spin} \le \tau_{\max} - \tau_{orbital} \tag{3.2.10}$$

Note that in this case, τ_{spin} can be set to 0, and there will be no ankle roll, due to Eq. 3.2.9. The second region is defined by

$$\tau_{orbital} > \tau_{\max} \text{ (positive case)}$$

$$\text{or } \tau_{orbital} < -\tau_{\max} \text{ (negative case)} \tag{3.2.11}$$

We assume that for this case, the ankle torque is pegged at

$$\tau = \tau_{\max} \text{ (positive case)} \tag{3.2.12}$$

or $\tau = -\tau_{\max}$ (negative case)

Then, from Eq. 3.26, the minimum allowable absolute value for τ_{spin} is given by

$$\tau_{spin} = \tau_{\max} - \tau_{orbital} \text{ (positive case)} \tag{3.2.13}$$

or $$\tau_{spin} = -\tau_{\max} - \tau_{orbital} \text{ (negative case)}$$

These region equations make it possible to predict whether the system will be stable from any given initial condition. Let's begin with the case of Eq. 3.2.13. Assuming a simple PD control law, with position setpoint x_{set}, position gain k_p, and damping gain k_d, the general solution for linearized x_{CM} motion is

$$\begin{aligned} x_{CM} &= e^{\alpha t}\left(K_1 \cos(\beta t) + K_2 \sin(\beta t)\right) + x_{set} \\ \dot{x}_{CM} &= e^{\alpha t}\left(\beta\left(-K_1 \sin(\beta t) + K_2 \cos(\beta t)\right) + \alpha\left(K_1 \cos(\beta t) + K_2 \sin(\beta t)\right)\right) \end{aligned} \tag{3.2.14}$$

where

$$\begin{aligned} K_1 &= x_{CM}(0) - x_{set} \\ K_2 &= -\left(\alpha K_1 - \dot{x}_{CM}(0)\right)/\beta \\ \alpha &= \frac{-k_d}{2} \\ \beta &= \frac{\sqrt{4k_p - {k_d}^2}}{2} \end{aligned} \tag{3.2.15}$$

We assume that the system is under-damped, hence, β is always real.

Suppose we choose a position setpoint $x_{set} = 0$, to reflect the desire to stabilize the CM over the origin. Then, for an initial condition $x_{CM}(0)$, $\dot{x}_{CM}(0)$, and for particular settings for k_p and k_d, Eq. 3.2.14 provides an analytic solution for the CM trajectory.

The PD control law equation provides an analytic solution for CM acceleration as a function of position and velocity:

$$\ddot{x}_{CM} = -k_d \dot{x}_{CM} - k_p x_{CM} \tag{3.2.16}$$

Eq. 3.2.4 can then be used to compute $\tau_{orbital}$. For a given value for $x_{\text{supp\_bound}}$, Eq. 3.2.8 can be used to compute $\tau_{\max}$. Then, from Eq. 3.2.13, τ_{spin} is computed, and from Eq. 3.2.5, $\ddot{\theta}$. Integrating this gives trajectories for θ and $\dot{\theta}$.

Constraints on θ and $\dot{\theta}$ can be used to express maximum bounds on body angle and angular velocity. The trajectories for θ and $\dot{\theta}$ can be checked against these bounds to ensure that the system remains within feasible operating regions. The gap between the initial values for $\theta, \dot{\theta}$ and the bounds on $\theta, \dot{\theta}$ is a "reservoir" of τ_{spin} that can be used to assist τ (Eq. 3.2.13). This reservoir is limited. Its size depends on the initial values for $\theta, \dot{\theta}$ and the bounds on $\theta, \dot{\theta}$.

To summarize, the simplified model allows for a simple check that will determine whether the system will be stable, given any initial condition.

3.2.2 PD Controller for the Simplified 2-link Model

The previous section introduced the simplified 2-link model, and proposed the use of a PD controller, along with an appropriate input-output linearization, to control it. This section elaborates on use of a PD controller, and reveals a number of important, issues related to the spin reservoirs introduced in the previous section. In particular, we use the simplified model to compute conservative limits on the size of the spin reservoir. These limits represent maximum limits on CMP position, which are important for determining the extent to which the moment balance strategy can be used in a particular situation. Knowledge of these limits is also important for flow tube compilation.

The simplified 2-link model has 2 degrees of freedom. To balance this system, it is necessary to control translational position of the CM. Additionally, we wish to maintain an upright body orientation, if possible. Thus, the outputs to control are x_{CM} and θ. Therefore, the state vector is: $\left[x_{CM},\quad \dot{x}_{CM},\quad \theta,\quad \dot{\theta}\right]$.

Stabilizing this system requires getting the outputs to their nominal values, which are 0 in this case, and ensuring that the norm of the state vector remains bounded. In this case, stabilization is achieved by getting all elements of the state vector to 0.

We assume, as in the previous section, that the system is appropriately linearized and decoupled, and that the primary control output is x_{CM}, with $\ddot{x}_{CM}$ computed based on a simple PD control law, as in Eq. 3.2.16. Because x_{CM} is the primary output, its behavior can be assumed to be linear; it will follow the PD control law, so its trajectory will be as specified in Eq. 3.2.14. Its acceleration, $\ddot{x}_{CM}$, will be as specified in Eq. 3.2.16, and therefore, $\tau_{orbital}$ is completely determined, by Eq. 3.2.4.

Given this value for $\tau_{orbital}$, the next step is to find control laws for τ and τ_{spin} that satisfy Eq. 3.2.6. To accomplish this, it is useful to separate τ_{spin} into two parts by introducing a slack variable.

$$\tau_{spin} = \tau_{spin\_des} + \tau_{spin\_slack} \qquad 3.2.17$$

Here, τ_{spin_des} is computed by a PD control law based on θ and $\dot{\theta}$ (τ_{spin} is directly related to $\ddot{\theta}$ through Eq. 3.2.5). The slack, τ_{spin_slack}, represents extra torque so that Eq. 3.2.6 is always satisfied. Eq. 3.2.6 can then be written as

$$\tau_{orbital} + \tau_{spin\_des} = \tau - \tau_{spin\_slack} \qquad 3.2.18$$

The following control law makes $\left|\tau_{spin\_slack}\right|$ as small as possible, while enforcing the restriction on maximum ankle torque.

If

$$\left|\tau_{orbital} + \tau_{spin\_des}\right| \le \tau_{\max}$$

then

$$\tau = \tau_{orbital} + \tau_{spin\_des}$$

$$\tau_{spin\_slack} = 0$$

else

$$\tau = \tau_{\max}$$

$$\tau_{spin\_slack} = \tau - \tau_{orbital} - \tau_{spin\_des}$$

Because x_{CM} behaves linearly, it is stable, as long as appropriate parameters are used for the PD controller. To prove stability for the overall system, it is necessary to show that non-zero values of τ_{spin_slack} are temporary and bounded. More precisely, it is necessary to show that non-zero values of τ_{spin} and its integrals, and therefore, non-zero values of $\ddot{\theta}$, $\dot{\theta}$, and θ are transient and bounded.

For the "then" case in the above control law, τ_{spin_slack} is 0, so τ_{spin} is just τ_{spin_des}. As long as appropriate PD control parameters are used for τ_{spin_des}, τ_{spin} and its integrals will be stable. The "else" case in the control law is the interesting one. Unfortunately, it is not possible to obtain an analytic solution for θ, as was the case for x_{CM} and $\tau_{orbital}$. This is because the input acceleration is $\ddot{\theta}$ which is linearly related to τ_{spin} through Eq. 3.2.5, and, from the control law, τ_{spin} is a function of $\tau_{orbital}$, which is a fairly complex curve, not a constant. Therefore, a numerical integration must be used to compute trajectories for θ, $\dot{\theta}$, and $\ddot{\theta}$, and for τ_{spin}. Nevertheless, because the model is so simple, it is possible to perform a complete numerical analysis that leads to general conclusions about stability limits.

Fig. 3.9 shows x_{CM} and $\dot{x}_{CM}$ trajectories resulting from an initial velocity of 0, and an initial position deviation of 0.1 m. Normalized values of $M_1 = 1$ and stance leg length = 1 m were used, and $x_{\text{supp\_bound}}$ was set to 0.05 m. Fig. 3.10 shows the associated $\tau_{orbital}$ trajectory. As can be seen, $\tau_{orbital}$ is initially larger than $\tau_{\max}$, so the "else" case of the control law must be used, and τ_{spin_slack} will not be 0 (τ_{spin} will not have its desired value).

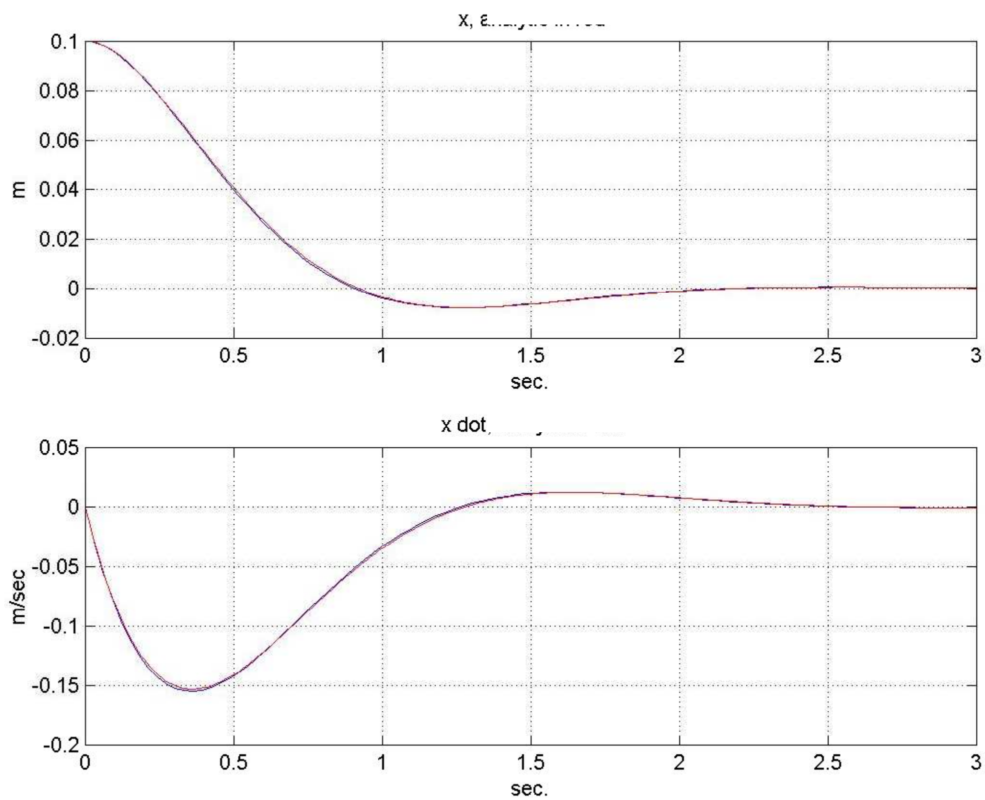


Fig. 3.9 – Example x_{CM}, $\dot{x}_{CM}$ trajectories.

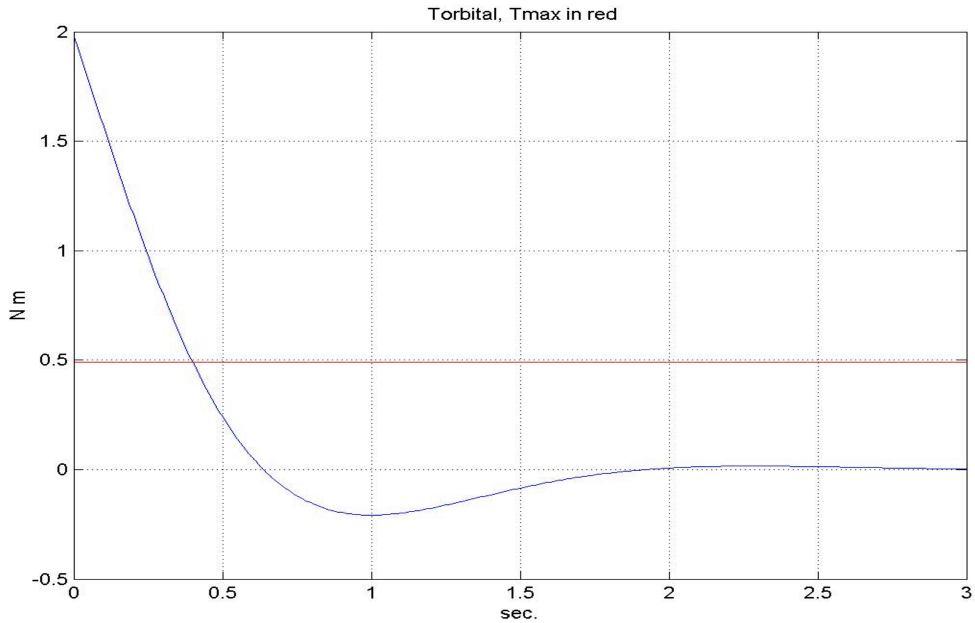


Fig. 3.10 - $\tau_{orbital}$ trajectory for the example x_{CM} trajectory.

Fig. 3.11 shows trajectories for θ and $\dot{\theta}$ resulting from application of the control law.

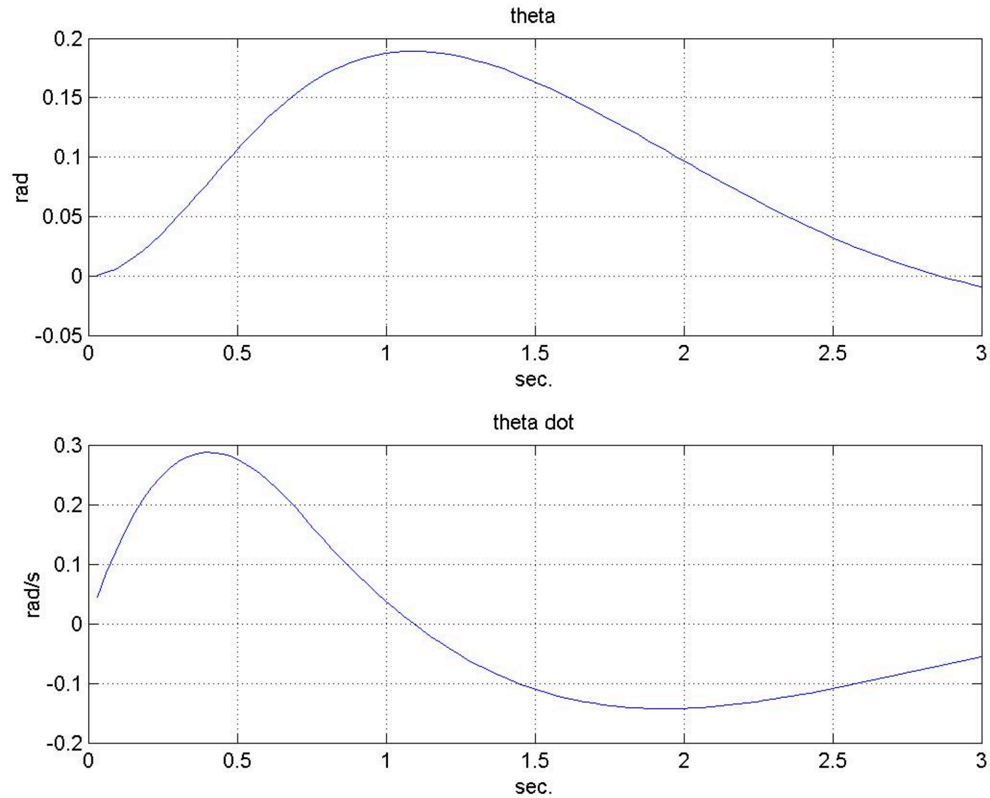


Fig. 3.11 – Trajectories for θ and $\dot{\theta}$.

As can be seen, θ is less than 0.2 radians, or, less than about 12 degrees. A reasonable limit on body rotation is 90 degrees, if we assume that a typical human would be unwilling or unable to rotate their torso more than this about the pitch or roll axes. Therefore, the θ trajectory shown in Fig. 3.11 is well within this limit. Similarly, the maximum value of $\dot{\theta}$ is less than 0.3 radians per second, or, less than about 18 degrees per second. This is also well within the limits of a typical human.

This example gives insight about how spin torque can be used to achieve overall stability, but it is just one example for a specific initial condition. For a more complete analysis we used an optimization algorithm to determine the maximum stable initial x_{CM} position for a range of initial θ positions, with the system beginning at rest. Fig. 3.12 shows the stability boundary in the $x_{CM}(0)$ - $\theta(0)$ plane; the region below the curve is stable.

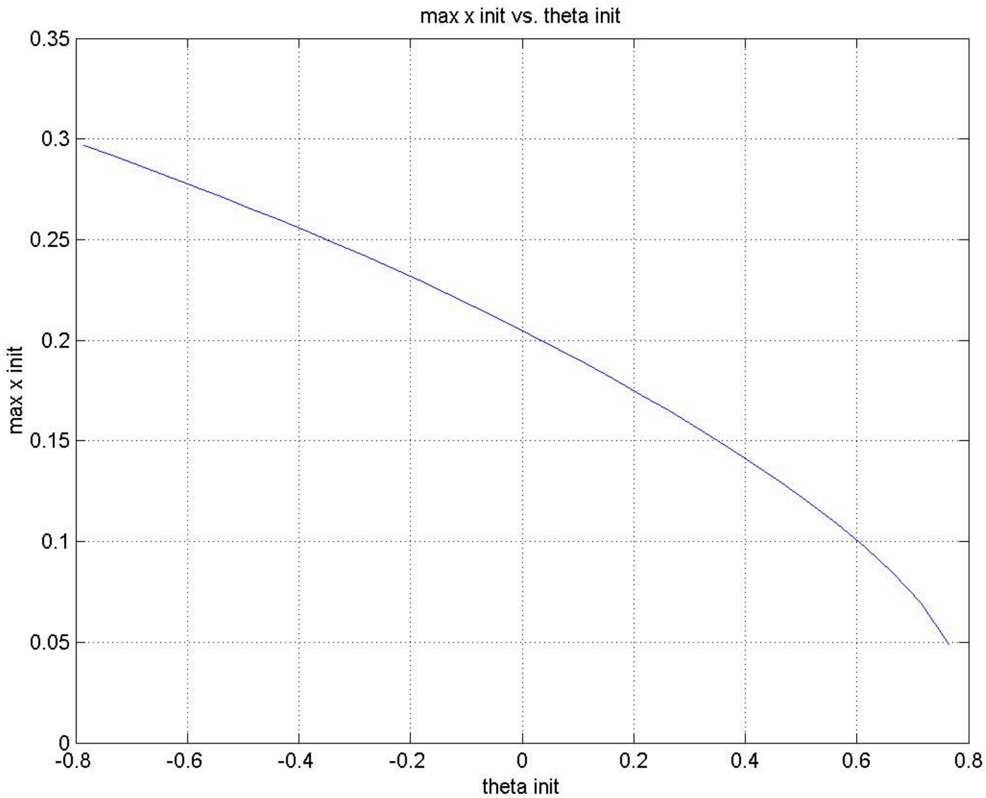


Fig. 3.12 – Maximum initial CM deflection vs. initial θ

A similar analysis was performed to determine maximum initial CM position for a range of initial CM velocities, with initial θ and $\dot{\theta}$ being 0. Fig. 3.13 shows the stability boundary in the $x_{CM}(0)$ - $\dot{x}_{CM}(0)$ plane; again, the region below the curve is stable.

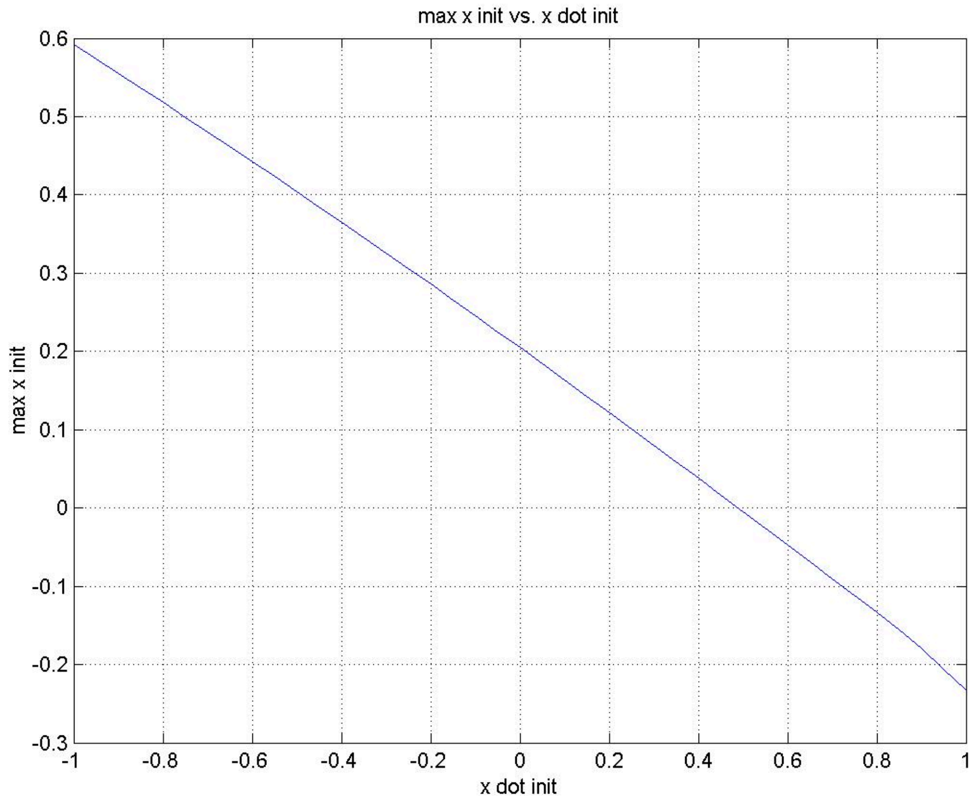


Fig. 3.13 – Maximum initial CM deflection vs. initial CM velocity

These results show that the use of spin torque in a balance control law can have a significant beneficial effect on the ability to control balance. Although these results are for a simplified model, they provide intuition about balance control requirements for a full biped model, and suggest a control approach for a full biped. An implementation of an advanced version of this approach, for a full biped, is described in Chapter 8. The simplified model results also provide approximate bounds on spin reservoirs, as depicted in Figs. 3.12 and 3.13. This is useful for efficient planning and plan compilation.

In the next section, we build on these results by defining a comprehensive balance control model, a classification of disturbances in terms of this model, and a set of metrics indicating the degree to which the biped is disturbed.

3.3 Disturbance Metrics and Classification

A comprehensive approach to bipedal walking in unstructured environments requires an understanding of the different types of disturbances that may be encountered, and the different strategies for dealing with them. The previous sections introduced the concept of conservation of spin angular momentum during normal walking, and non-conservation of this quantity as a balance strategy, as well as a preliminary control system that can select either mode as appropriate. These represent two of the balance strategies introduced in Chapter 1. A discussion of the stepping strategy is provided in Appendix 3.1.

Choosing the right combination of strategies for a particular disturbance requires a model that can clearly map from disturbances to strategies. This model should include metrics that help classify the severity and type of disturbance, and that provide a good summary of the overall balance state of the biped. Such metrics are important for efficient planning and plan compilation of walking tasks.

To gain insight into the key requirements for balance control, and for the sake of computational efficiency during plan compilation, we seek a model that adequately captures balance control requirements in terms of a minimal set of input and output values. In the following discussion, we show that by choosing the right abstractions, balance control is achieved by controlling a small set of key outputs using a small set of inputs. This allows us to classify bipedal walking into a limited set of basic behaviors, and to classify disturbances into a limited set of types according to how they affect the model. This allows for the model to be used to select the right combination of the three fundamental balancing strategies introduced in Chapter 1.

The next section begins with a discussion of the FRI constraint [Goswami, 1999], a key constraint in balance control problems. This constraint can be used to show clearly the relation between the key balance control input and output values. This is followed by a description of the set of disturbance metrics. This leads to a definition of what it means to fall down, that is, loss of balance control, in terms of these metrics. After this discussion, a classification of basic disturbance types is presented in terms of the key inputs, outputs, and disturbance metrics. Finally, disturbance handling strategies are described.

3.3.1 FRI Constraint

In order to understand balance control inputs and outputs, it is useful to first review the FRI constraint, a key constraint involved in balance control. A typical stationary manipulator is firmly bolted to the ground, and all of its joints are actuated; it is thus a fully actuated system. Unlike such manipulators, walking bipeds are not firmly bolted to the ground. The foot can roll, for example, if the torque exerted by the ankle is too large, because the support base is limited in size. This limits the moment arm of the edge of the foot, and thus, the counter-acting torque created by the contact of the edge of the foot with the ground, which is required to keep the foot flat.

. The contact of the foot with the ground can be thought of as a semi-underactuated joint; it is not fully actuated, because of the limited support base size, but it can support a certain finite maximum torque because the support polygon does have some finite size.

A useful quantity for determining whether the foot will roll is the Foot Rotation Indicator (FRI), [Goswami, 1999]. The FRI is the point on the foot/ground contact surface where the ground reaction force would have to act to keep the foot from rolling. If the FRI is inside the support polygon, then the foot remains flat on the ground. If the FRI is outside the support polygon, then the foot will roll. In Fig. 3.14, the FRI, indicated by the point F, is in front of the stance foot limit, indicating that the foot is about to roll on its forward edge.

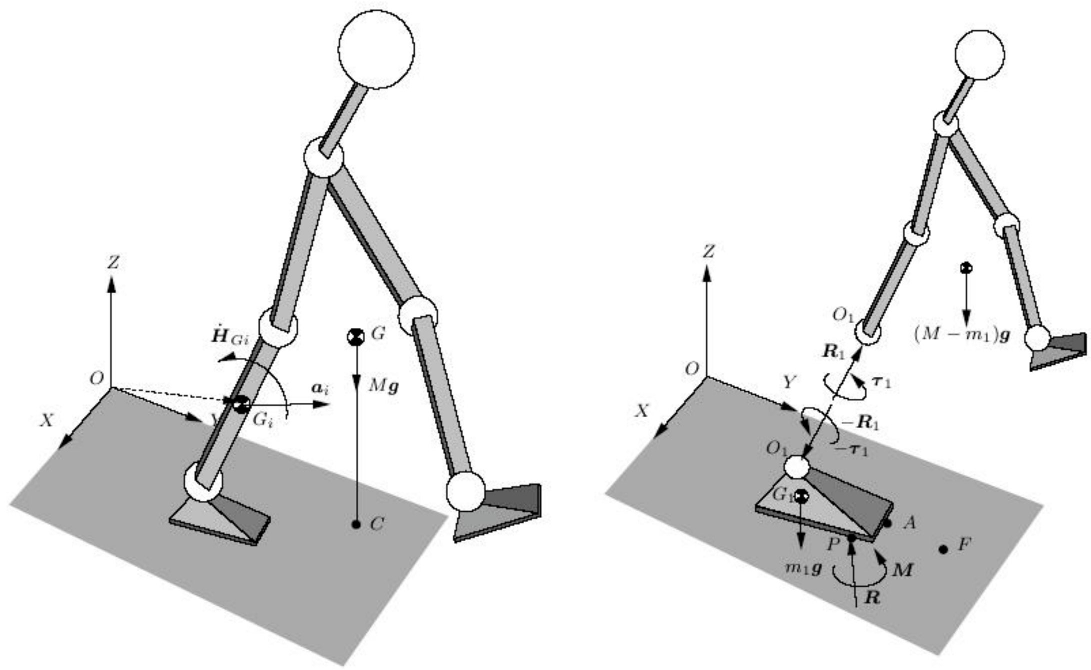


Fig. 3.14 – Foot rotation indicator (from [Goswami, 1999])

In this diagram, quantities are

F - the FRI point

G - the CM position

M - mass of the full system

G_i - CM of link i

m_i - mass of link i

$\dot{H}_i$ - torque about link i

a_i - linear acceleration of link i

The FRI appears in the following torque balance equation [Goswami, 1999].

$$\sum_{i=2}^{n} FG_i \times m_i g = \sum_{i=2}^{n} \dot{H}_i + \sum_{i=2}^{n} FG_i \times m_i a_i \qquad (3.3.1)$$

This states that the sum of the torques about the FRI due to gravitational acceleration of each link CM, which is the term on the left hand side of Eq. 3.3.1, and linear acceleration, which is the second term on the right hand side, is equal to the sum of the torques about each link. This can be expressed, in 2D, as

$$\sum_{i=2}^{n} FG_{xi} f_{zi} = \sum_{i=2}^{n} \dot{H}_i + \sum_{i=2}^{n} FG_{zi} f_{xi} \tag{3.3.2}$$

where

$$f_{xi} = m_i a_{xi}$$

$$f_{zi} = m_i \left(g - a_{zi} \right)$$

The equation for the y-z plane is similar.

We would like to transform this torque balance equation into a form that separates out the orbital, spin, and ankle torque components, as in Eq. 3.2.6, so that the control concepts from the previous section can be applied. To accomplish this, note first that the vector from the FRI to the CM of any link can be broken into the sum of two vectors, from the FRI to the system CM, and from the system CM to the link CM.

$$FG_i = FG + GG_i \tag{3.3.3}$$

Substituting into Eq. 3.3.2 yields

$$\sum_{i=2}^{n} GG_{xi} f_{zi} + FG_x f_z = \sum_{i=2}^{n} \dot{H}_i + \sum_{i=2}^{n} GG_{zi} f_{xi} + FG_z f_x \tag{3.3.4}$$

Now, if we introduce an origin point, O, located at a point on the bottom of the foot directly below the ankle joint, we can break the vector from the FRI to the CM into the sum of a vector from the FRI to the origin, and a vector from the origin to the CM.

$$FG = FO + OG \tag{3.3.5}$$

$$FG_x = FO_x + OG_x$$

$$FG_z = FO_z + OG_z = OG_z$$

Substituting into Eq. 3.3.4 yields

$$-OF_x f_z + OG_x f_z - OG_z f_x = \sum_{i=2}^{n} \dot{H}_i + \sum_{i=2}^{n} GG_{zi} f_{xi} + \sum_{i=2}^{n} GG_{xi} f_{zi} \qquad (3.3.6)$$

This is of the desired form

$$\tau_{stance\_ankle} - \tau_{orbital} = \tau_{spin} \qquad (3.3.7)$$

where

$$\tau_{stance\_ankle} = -OF_x f_z ,$$
$$\tau_{orbital} = OG_z f_x - OG_x f_z ,$$
$$\tau_{spin} = \sum_{i=2}^{n} \dot{H}_i + \sum_{i=2}^{n} GG_{zi} f_{xi} + \sum_{i=2}^{n} GG_{xi} f_{zi} \text{ , and}$$

$$\tau_{stance\_ankle} - \tau_{orbital} = \tau_{spin}$$

The orbital torque, $\tau_{orbital}$, is the torque of the system CM about the origin. The spin torque, τ_{spin}, is the torque about the system CM. The ankle torque, τ_{stance_ankle}, is the torque exerted by the ankle joint.

The vector OF is constrained by the support polygon size, so that F, the FRI point, is inside the support polygon, and hence, the foot does not roll. This, in turn, constrains τ_{stance_ankle}. The orbital torque, $\tau_{orbital}$, results in translational movement of the CM. As can be seen from Eq. 3.3.7, spin torque, τ_{spin}, can be used to assist the limited τ_{stance_ankle} in order to provide sufficient $\tau_{orbital}$. However, there are limits to how large τ_{spin} can be, and how long it can be used, as was discussed in the previous section.

Now that we have discussed the FRI constraint, we begin our discussion of balance control inputs and outputs.

3.3.2 Balance Control Inputs and Outputs

Horizontal CM position is the key value to be controlled to maintain balance. Loss of control of this value corresponds to loss of control of the system, that is, falling down. As long as this output can be controlled, and as long as angular disturbance state remains within required bounds, the system will remain stable. A more precise definition of loss of balance control will be given subsequently, in section 3.3.4.

Horizontal acceleration of the CM can be related to $\tau_{orbital}$ using an input-output linearization technique, as will be discussed in detail in Chapter 8. Thus, from Eq. 3.3.7, horizontal CM can be controlled by stance ankle torque and spin torque, as shown in Fig. 3.15. This is a concise way to view the balance control problem.

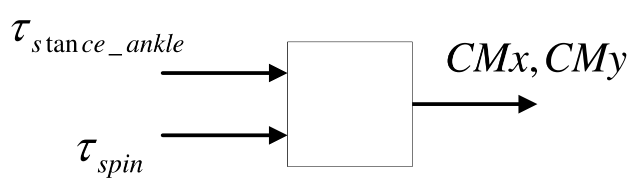


Fig. 3.15 – Balance control inputs and outputs

Let's assume, for the moment, that τ_{spin} is 0, which is the nominal walking case. The stance ankle torque is generated as a result of contact with the ground. This effect can be represented by a ground contact force, acting at the ZMP. Let's also assume, for the moment, that the FRI point is inside the foot support polygon, so the ZMP is the same point as the FRI, and the CMP. From Eq. 3.3.7,

$$\tau_{stance\_ankle} = -OF_x f_z = OG_z f_x - OG_x f_z = \tau_{orbital} \qquad (3.3.8)$$

A similar relation can be written for the y direction. Furthermore, because τ_{spin} is 0, the ground reaction force vector points from the ZMP to the CM, and the ratio between horizontal and vertical ground reaction force is constrained. Thus, instead of using τ_{stance_ankle} as an input, as in Fig. 3.15, the inputs can be represented as a vertical ground reaction force, and the ZMP position, instead of τ_{stance_ankle}, as shown in Fig. 3.16.

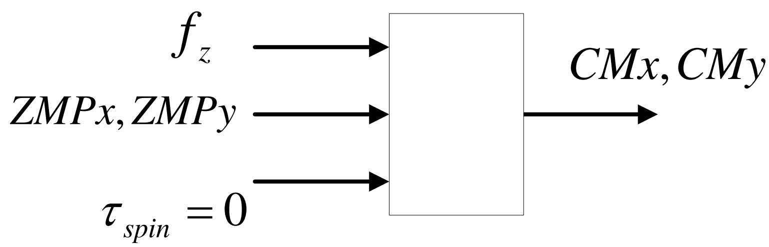


Fig. 3.16 – Balance control model in terms of ZMP

Now, let's suppose that the desired acceleration for CM is high, so that a high value is desired for $\tau_{orbital}$. Let's suppose this value is higher than the maximum allowed by the FRI constraint; the limit on OF_x in Eq. 3.3.8 due to the limited size of the support polygon. Thus, Eq. 3.3.8 no longer holds; τ_{spin} can no longer be set to 0. Eq. 3.3.7, which allows non-zero τ_{spin}, must be used. The FRI is against the edge of the support polygon. The FRI remains inside the support polygon so that the foot remains flat on the ground. In this situation, the FRI coincides with the ZMP. Because τ_{spin} is not zero, the ground reaction force vector from the ZMP no longer points to the CM, as shown previously in Fig. 3.5. The distance d_{spin} in Fig. 3.5 is the distance by which the ground reaction force vector misses the CM; it is the moment arm for the spin torque. The spin torque is then

$$\tau_{spin} = d_{spin} f_z \qquad (3.3.9)$$

This situation can also be represented using the CMP, as shown previously in Fig. 3.6. The ZMP is the same as the CMP when the CMP is inside the support polygon. In this case, d_{spin} (and τ_{spin}) are 0. When the CMP is outside the support polygon, the ZMP is at the edge of the support polygon closest to the CMP, as shown in Fig. 3.17.

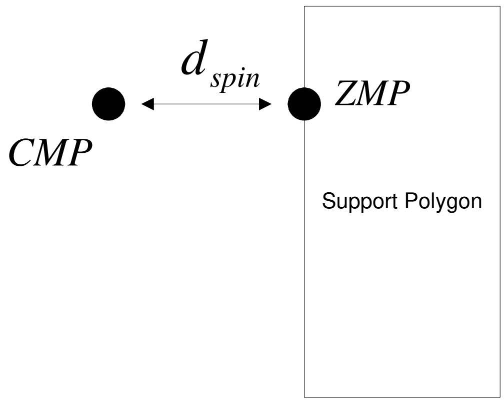


Fig. 3.17 – CMP, ZMP and support polygon

Thus, ZMP can always be computed from CMP, as long as the position and shape of the support polygon is known. The previous control diagram of Fig. 3.16 can now be expressed using the CMP to allow for a non-zero τ_{spin}, as shown in Fig. 3.18.

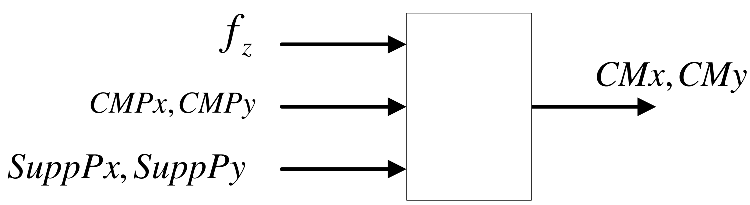


Fig. 3.18 - Balance control model in terms of CMP

The support polygon, indicated by $SuppPx, SuppPy$, is an input to this model, but it is constant for the duration of a particular foot placement configuration. The vertical force, f_z, is used only to compute τ_{spin}. For typical planning and stability analysis purposes, it can be estimated by a worst-case, highest value constant for walking motions. Therefore, the model shown in Fig. 3.18 really has only two inputs, forward and lateral CMP position, which are used to control the two outputs, forward and lateral CM position.

This simple model is useful for planning feasible CM trajectories, and is used in the plan compiler component of the hybrid executive, discussed in Chapters 4 – 7. The three balance strategies introduced in Chapter 1 are explained easily in terms of this model. For the ankle torque strategy, the CMP is inside the support polygon. For the spin torque strategy, it is outside. The stepping strategy represents a change in the support polygon.

3.3.3 Disturbance Metrics

As explained in the previous sections, the quantity τ_{spin} can play an important role in achieving desired orbital torque, through Eq. 3.3.7, the torque balance equation, and therefore, desired horizontal CM position through an appropriate linearization. However, there are limits to the magnitude and duration of use of non-zero τ_{spin}, as was explained previously in Section 3.2 in our discussion of spin reservoirs.

To understand these limits better, first consider that τ_{spin} is an input quantity, related to acceleration, as discussed in the previous section. The integral of this is the spin angular momentum:

$$L_{spin} = \int \tau_{spin} dt \qquad (3.3.9)$$

This is a quantity related to angular velocity, and also to inertia. Integrating this gives a new quantity, which we call the spin disturbance level:

$$D_{spin} = \int L_{spin} dt \qquad (3.3.10)$$

This is a quantity related to angular position, and also to inertia. The spin disturbance level is related to a quantity called the effective angle [Popovic, 2004b].

One can think of τ_{spin}, L_{spin}, and D_{spin} as being a second-order linear system, where the position-like quantity, D_{spin}, is being controlled, via a PD control law, to be 0. The cases where τ_{spin} has to be non-zero, or more precisely, a value other than the one called for by the PD control law, can be viewed as a temporary disturbance. This is OK as long as D_{spin} and L_{spin} do not become too large.

Keeping D_{spin} within bounds is an important control goal. However, as explained previously, the primary goal is to control lateral CM position. As explained previously, this is related to $\tau_{orbital}$. Thus, an approach similar to the one that derived D_{spin} can be taken for the orbital component.

$\tau_{orbital}$ is an input quantity, related to acceleration. The integral of this is the orbital angular momentum:

$$L_{orbital} = \int \tau_{orbital} dt \qquad (3.4.11)$$

Integrating again gives a new quantity, which we call the orbital disturbance level:

$$D_{orbital} = \int L_{orbital} dt \qquad (3.4.12)$$

As with the spin case, one can think of $\tau_{orbital}$, $L_{orbital}$, and $D_{orbital}$ as being a second-order linear system, where the position-like quantity, $D_{orbital}$, is being controlled.

For stationary balancing, the setpoint for $D_{orbital}$ is nominally 0. For walking, it has to be non-zero. However, the extent to which $D_{orbital}$ is non-zero is also a measure of instability. As with D_{spin}, there are bounds on maximum possible values for $D_{orbital}$. This will be explored in more detail in the next section.

Thus, a weighted sum of $D_{orbital}$, $L_{orbital}$, D_{spin}, and L_{spin} provides a good summary of the disturbance state of the system, with respect to the current foot placement configuration.

3.3.4 Definition of Loss of Balance Control

The ability of the system to recover balance is a function of $D_{orbital}$, $L_{orbital}$, D_{spin}, and L_{spin}. If these values are within appropriate bounds, as determined by this *balance recovery function*, the system will recover. Otherwise, it will fall down.

The balance recovery function is dependent on the morphology of the system, and on the control system, and it is difficult to compute analytically. Numerical techniques can be used, with simplified models, to compute this function, as discussed in Section 3.2. Our approach here is to use the simplified models to get approximate, conservative bounds that can be used for planning motion for the full plant.

The goal of the control system is to control $D_{orbital}$, $L_{orbital}$, D_{spin}, and L_{spin} so that the system is well within the bounds of the balance recovery function. Knowledge of the bounds is useful when a fall is inevitable; if the bounds are exceeded, the system will know that it is about to fall and can take mitigating action, like putting the hands out.

3.3.5 Disturbance Classification

Having derived the balance control model shown in Fig. 3.18, we are now in a position to classify disturbances in terms of their effect on this model. This will be useful for obtaining a mapping to disturbance handling strategies. Before presenting a classification of disturbances, we review the basic kinds of locomotion activity for bipeds.

There is a limited set of basic types of bipedal locomotion activity:

- Standing
    - Single support
    - Double support
- Walking
    - Single support
    - Double support
- Running
    - Single support
    - Aerial

Running is beyond the scope of this study; the focus here is on standing and walking. Therefore, the following discussion of disturbances pertains, primarily, to standing and walking activities.

Disturbances can be classified in the following way.

Push – a force with a particular direction and magnitude exerted on a particular segment of the articulated linkage walking mechanism, and at a particular point of contact on that segment. Examples include a push on the body, or a push on the swing leg while walking, which may result in a trip.

This can be represented as a direct disturbance to the control inputs $\tau_{orbital}$ and τ_{spin}. It, therefore, directly affects the disturbance metrics $D_{orbital}$, $L_{orbital}$, D_{spin}, and L_{spin}.

Trip – special case of a push, relevant only for walking and running activities. In this case, the push is exerted on the swing leg. The trip, if severe enough, may delay the timing of the change in support polygon, by delaying the stepping action. In terms of the balance control inputs and outputs presented previously, this corresponds to a disturbance , or timing restriction, on ZMP and CMP.

Slip – a horizontal force exerted at the bottom of a stance foot resulting in horizontal translation of the foot. In terms of the balance control inputs and outputs presented previously, this corresponds to an uncontrolled change in the support polygon, and thus, on ZMP and CMP.

Stance foot roll – a torque about the ankle resulting in rolling or pitching of the foot. This is acceptable during the toe-off phase of walking, but is unacceptable otherwise. It corresponds to the FRI leaving the support polygon. When the foot begins to roll, it means that the support capability of the corresponding leg is diminished, or disappears altogether. In terms of the balance control inputs and outputs presented previously, this corresponds to an uncontrolled change in the support polygon, and thus, on ZMP and CMP.

These disturbances all involve a force or torque exerted at some point on the mechanism. A separate class of disturbances to the normal locomotion modes is based on position restrictions rather than force. These can be classified in the following way.

Foot placement restriction – restriction on where the stance foot can be placed during walking or running. This corresponds to a restriction on the support polygon, and thus, on ZMP and CMP.

Obstacle avoidance – restriction on motion path of a particular segment or segments of the articulated linkage walking mechanism.

These can be thought of as disturbances, even though they don't involve unanticipated contact, because they require activity that deviates from the normal locomotion modes. They are necessarily anticipated, though the time of anticipation may be short. They are important because they represent motion constraints that can have a significant influence on choice of disturbance handling strategies. For example, a push disturbance on unrestricted terrain that is easily handled by a step may require more complex action of non-contact limbs if the position of the step is restricted. If a tightrope walker is pushed, he does not have the luxury of stepping off the tightrope, but rather, must use coordinated activity of non-contact limbs to restore balance. Similarly, a football player trying to stay in bounds after a collision will use non-contact limbs in a similar way, because his stepping area is restricted.

It is also useful to consider the degree to which a disturbance can be anticipated. A football player that sees an opponent on a collision course will know, with a high degree of certainty, that a collision will occur. On the other hand, a person walking on ice or on rough terrain has some degree of anticipation that a disturbance is likely to occur, but is not certain when. Such a person is likely to modify gait and joint stiffness in anticipation of the event in order to minimize its effects.

3.3.6 Disturbance Handling

The combination of different kinds of locomotion activity and disturbances, including, multiple disturbances, leads to a large set of possible situations that must be handled. Nevertheless, it is clear that humans can easily and quickly select appropriate disturbance handling strategies, for a wide variety of disturbance situations. As introduced in Chapter 1, humans select some combination of the following three strategies.

Ground reaction torques – torques are exerted with respect to the support polygon to influence position of the CM. The amount of torque that can be exerted is limited by the extent of the support polygon, and is, therefore, relatively small when in single support, or in double support when the feet are close together. This is sometimes called the "ankle strategy" in the biomechanics literature [Nashner, 1982; Jo and Massaquoi, 2004].

Non-contact limb movement – non-contact segments of the articulated linkage, such as the torso, arms, and swing leg when in single support are moved to generate a spin torque about the CM in order to generate a beneficial force on the CM, as described previously. This is sometimes called the "hip strategy" in the biomechanics literature [Nashner, 1982; Jo and Massaquoi, 2004].

Step – change in foot placement to change the support polygon in a beneficial way.

The previous section discussed how disturbances of various types can be mapped to the balance control inputs and outputs introduced earlier. This allows for the following approach to disturbance handling. First, the effect of the disturbance is represented in terms of the balance control inputs and outputs of the model in Fig. 3.18, and the disturbance metrics. The appropriate disturbance handling strategy is then computed based on this model and these metrics. This approach utilizes the model's abstraction and simplifies control in that there is no explicit mapping from one particular disturbance type to one particular disturbance handling strategy.

4 Hybrid Task-Level Executive

As introduced in Chapter 1, we seek to guide the bipedal walking machine so that it accomplishes a specified locomotion task, such as walking at a specified speed, walking on a set of irregularly placed stones, or walking to a soccer ball in time to kick it. Recall from Chapter 1 that these tasks are specified in terms of flexible state and temporal goals, which are assembled into a qualitative state plan. The qualitative state plan is executed by a *model-based executive*, which generates control inputs for the biped such that the plan goals are satisfied, as shown in Figs. 1.9 and 1.10.

In this chapter, we describe the *hybrid task-level executive* component of our model-based executive. As introduced in Chapter 1, the hybrid task-level executive takes a qualitative state plan as input, and attempts to execute this plan successfully, even if there are significant disturbances. The hybrid executive does not generate control actions for the biped directly. Rather, it controls an abstraction of the biped, called a linear virtual element abstraction, which is provided by the dynamic virtual model controller, as shown in Fig. 1.14. As discussed in Section 1.4.1, this abstraction simplifies the job of the hybrid executive by linearizing and decoupling the biped plant, making it appear to be a set of independent, linear systems.

We begin this chapter by introducing the problem solved by the hybrid executive (Section 4.1). We then outline our approach, by introducing the major components of the executive, and by summarizing innovations (Section 4.2). Formal definitions of the linear virtual element abstraction and qualitative state plan, are provided in Sections 4.3 and 4.4, respectively. We conclude Section 4.4 with a formal definition of the problem solved by the executive.

Subsequent chapters provide implementation details of the hybrid executive. Chapter 5 describes a *qualitative control plan*, which is generated from the qualitative state plan, in order to support efficient, robust execution. Chapters 6 and 7 describe the two major components of the executive: the *plan compiler* and the *hybrid dispatcher*. The plan compiler generates a qualitative control plan from a qualitative state plan. The dispatcher executes the qualitative control plan by dynamically scheduling plan activities, and by executing these activities through the continuous adjustment of control parameters.

Details of how the linear virtual element abstraction is computed by the dynamic virtual model controller are provided in Chapter 8.

4.1 Overview of Problem Solved by Hybrid Executive

This section provides an intuitive introduction to the problem solved by the hybrid executive, emphasizing requirements and challenges. We discuss what it means for a plan to be executed successfully. From this, we derive requirements for the hybrid executive, and discuss its challenges for guiding a walking biped through a qualitative state plan successfully.

Given a qualitative state plan as input, and the linear virtual element abstraction, provided by the dynamic virtual model controller, the job of the hybrid executive is to guide the biped through the sequence of qualitative states in the plan, by adjusting control parameters in the abstraction.

The flexibility of the temporal and state-space specifications in the qualitative state plan means that there are many possible ways available to the executive that achieve the plan goals; there are many state trajectories for the biped that satisfy plan requirements. This flexibility allows the hybrid executive to consider multiple possible control parameter sequences and to choose the most appropriate one given the situation. For example, suppose that the biped begins in a nominal state. The hybrid executive begins to issue control parameter commands that result in nominal state trajectories that lead to satisfaction of all the plan's goals. If a disturbance occurs, the situation changes, and a new set of state trajectories is required in order to achieve the plan in this disturbed situation. This new set of state trajectories must begin from the disturbed state, and must still satisfy all of the plan's goals. Achieving these new state trajectories may require the hybrid executive to deviate from the original nominal control parameter sequence.

Searching the space of control parameter values, in order to achieve an acceptable state trajectory, is difficult. As discussed in Chapter 1, this is due to two key challenges. First, movement dynamics are relatively high-dimensional, highly nonlinear and tightly coupled, so computing control actions that achieve a desired state is a challenging problem, as discussed in Section 1.3.1. Second, a biped has limits on its ability to accelerate its center of mass. Therefore, the executive must consider how current state and actions may limit future state evolution, as discussed in Section 1.3.2.

Our model-based executive separates these problems and addresses them individually. As discussed in Section 1.4, we use the dynamic virtual model controller component of the model-based executive to address the first of these key challenges, and the hybrid executive to address the second. The linear virtual element abstraction provided by the dynamic virtual model controller dramatically simplifies the hybrid executive's job. Instead of searching the space of control inputs for all joints of the biped, and projecting resulting state trajectories for the high-dimensional, highly nonlinear and tightly coupled biped plant, the hybrid executive searches a much smaller space. Specifically, it searches the space of a few control parameters for each simple linear 2^{nd}-order system.

Although much simpler than a search over the full plant state space, this search is still challenging in that each SISO system may have state space region and temporal constraints specified in the qualitative state plan, and may have dynamic actuation constraints that must be satisfied. In particular, although the linear virtual element abstraction results in simple linear decoupled systems that are seemingly independent, they are still loosely coupled, due to temporal constraints in the qualitative state plan. Furthermore, the actuation constraints make the problem especially challenging, even with use of the SISO systems, in that the future consequences of current actions must be carefully considered to ensure that all plan goals are met.

Thus, the problem solved by the hybrid executive is to determine, for each SISO system, the set of state trajectories that satisfy the state space and temporal constraints specified in the qualitative state plan, and the dynamic constraints of the plant. We call such trajectory sets *flow tubes*. The hybrid executive computes approximations of these tubes, and executes the plan by keeping the state trajectories for each SISO system within its approximated tube. The hybrid executive must accomplish this by adjusting control parameters in the dynamic virtual model controller. The hybrid executive must perform these adjustments quickly enough to control the biped in real time.

If a disturbance occurs that is large enough that the biped has no hope of achieving the plan goals, the hybrid executive must notify a higher-level re-planning function, in order to generate a new plan that will succeed for the new situation. Such a re-planning function is beyond the scope of this thesis, but it would be an important cognitive component in a fully operational system [Kim et al., 2001].

4.2 Hybrid Executive Approach

This section introduces the major components of the hybrid executive, and discusses how key design choices and innovations address the above-described challenges.

In order to satisfy performance requirements, our hybrid executive uses a partial compilation approach, where flow tubes are computed off-line by a *plan compiler*. The flow tubes represent sets of feasible trajectories that satisfy the plan. Use of these flow tubes allows the execution-time component of the hybrid executive to focus on exploring a much smaller fraction of the state space than would be necessary without the flow tubes. The plan compiler outputs the flow tubes as a *qualitative control plan*, which is executed by a hybrid dispatcher, as shown in Fig. 1.16.

This compiler/dispatcher architecture is similar to ones used by activity plan executives [Muscettola et al., 1998; Morris et al., 2001]. An activity plan contains activities with temporal constraints, just like a qualitative state plan. However, unlike a qualitative state plan, an activity plan does not include constraints on state variables. Thus, an activity plan dispatcher schedules start and finish times for activities directly, to satisfy temporal constraints, but does not have to consider state variable conditions to check whether an activity can be executed, or whether it has completed.

Before discussing our compilation approach, we review work on activity plan compilation and execution, discuss similarities and differences between activity plan and qualitative state plan execution, and discuss aspects of the previous activity plan work that we leverage in our hybrid executive. We then discuss our approach, and how it can be viewed as an extension of these techniques. In particular, whereas the activity plan compilers perform a tightening of temporal bounds, our plan compiler performs a tightening of state-space, as well as temporal bounds. For example, the plan compiler may significantly tighten the state-space region for allowable movement of the biped's center of mass, thus providing important guidance to the execution-time component.

4.2.1 Relation to Activity Plan Execution

Although the linear virtual element abstraction decouples the interaction between state variables of the SISO controllers, they are still coupled through the state plan temporal constraints. Thus, although the linear virtual element abstraction dramatically simplifies the control problem, the SISO systems are not completely independent in that the temporal constraints must be observed. A key feature of our hybrid executive is the way in which it manages this temporal coupling.

The temporal coupling is similar to that for activity plan execution [Muscettola et al., 1998; Morris et al., 2001; Stedl, 2004]. Thus, it is worthwhile investigating to what extent techniques for these systems can be applied to the present problem. To do this, we must analyze the similarities and differences between activity plan execution, and qualitative state plan execution.

An activity plan has temporal constraints, just like a qualitative state plan. However, unlike a qualitative state plan, an activity plan does not deal with state variables, and has no state variable constraints. Thus, in an activity plan, it is assumed that an activity can always be executed, regardless of any system state, as long as its start and finish event times are consistent with the temporal constraints of the plan.

In contrast, a qualitative state plan contains constraints for continuous state variables. Values of these state variables do not change instantaneously, but rather, continuously over time according to differential equations. Such dynamic equations represent constraints on the time evolution of the state variables. In particular, they represent constraints relating accelerations, control input limitations, and time needed to reach goal regions. Thus, the SISO systems are inertial, and hence, state variables are not directly set, but rather, are moved continuously by adjusting control parameters. Furthermore, due to limitations on control inputs, controllability is often very limited, and the system may be underactuated. In such cases, activity goal regions may not contain equilibrium points, so there is a limited time that the controller can keep state variables "waiting" in such goal regions. For example, in dynamic walking, it is not possible to instantly stop forward movement in the middle of a step. The stepping foot must move out in front, or the biped will fall. To summarize, whereas an activity plan executive is allowed to schedule activities without regard to system state, a qualitative state plan executive must consider this state, which changes continuously, and which may be difficult to control due to actuation limits.

An important consequence of this is that, for a qualitative state plan, an activity is executed by shaping a continuous state trajectory. The trajectory is shaped by setting the goal region, which establishes the trajectory heading, and by setting the associated SISO system's control parameters, which defines the shape of the trajectory and its arrival time in the goal region.

Both activity plans and qualitative state plans have continuous temporal variables, and constraints on these variables. In both cases temporal constraints have to be satisfied in order for the plan to execute successfully. However, in the case of activity plan execution, the executive is allowed to arbitrarily choose activity start times and durations, as long as these are consistent with the temporal constraints. In contrast, for qualitative state plan execution, the executive controls activity start times and durations indirectly, by manipulating the SISO abstraction's control parameters. Furthermore, some start times and durations that may be allowed by the temporal constraints may not, in fact, be achievable due to the dynamic limitations.

Both activity plan and qualitative state plan executives must deal with disturbances that occur during plan execution. For activity plan execution, a disturbance is represented as an unexpected change in the duration of an activity. The executive compensates for this unexpected duration change by adjusting durations of subsequent activities, in order to ensure that the plan temporal constraints are still satisfied [Morris et al., 2001; Stedl, 2004].

An extensive set of algorithms has been developed for efficient activity plan execution [Muscettola et al., 1998; Morris et al., 2001; Stedl, 2004]. Due to the similarities with activity plan execution, particularly the common requirements for temporal consistency, many elements of these techniques can be leveraged for qualitative state plan execution. One very effective technique that can be used for both types of plan execution is to compile the state plan into a *dispatchable* form that can be executed directly [Muscettola et al., 1998]. This form makes the tightest temporal bounds explicit so that the plan can be executed directly, without runtime derivation of implicit constraints. We use this technique in our plan compiler as well.

4.2.2 Efficient Plan Execution through Compilation

We now describe our approach to efficient execution of a qualitative state plan. We begin by continuing the above discussion of explicit and implicit temporal bounds, and extend the method to explicit and implicit state-space bounds, resulting in the flow tube representation.

The explicit temporal bounds in an activity plan may imply further implicit bounds on activities that the executive must satisfy in order to ensure temporal consistency. For example, suppose a plan involves driving from Boston to New York, and then on to Washington D. C. Suppose the plan specifies that the drive from Boston to New York should take 5 hours, and that the overall trip should take at most 9 hours. This implies that the New York to Washington drive should take at most 4 hours.

Computing these implicit bounds at execution time is inefficient; the solution is to compute them offline, before any execution begins. Thus, for activity plans, execution efficiency is achieved by compiling the plan into a dispatchable form that makes the tightest, that is, most restrictive, temporal bounds explicit [Muscettola, 1998]. Such a dispatchable plan is then executed by a *dispatcher*. Because all temporal constraints are explicit in the dispatchable plan, the dispatcher can execute the plan directly, and doesn't have to worry about deducing implicit constraints at execution time.

Thus, a typical activity plan executive consists of two components: a compiler and a dispatcher. The compiler converts the plan into a dispatchable form, to be executed by the dispatcher. The dispatcher updates the explicit bounds in the dispatchable plan if disturbances occur that require further tightening of subsequent activity durations.

Our qualitative state plan compiler performs temporal bound tightening similar to that performed by compilers for discrete state plans. In addition, our compiler takes into account temporal constraints arising, indirectly, from dynamic limitations on the state variables, and combines these with ones specified explicitly in the state plan.

Furthermore, because a qualitative state plan contains state space as well as temporal constraints, the compiler performs a bound tightening of state space constraints, analogous to the bound tightening performed on temporal constraints. As with the temporal constraints, this tightening makes implicit spatial constraints explicit, so that the plan can be executed directly. For example, suppose that the qualitative state plan explicitly specifies a goal region for the center of mass, but does not specify any such constraint for the intermediate qualitative states leading up to the goal. Although there are no explicit state space bounds on the center of mass for the intermediate qualitative states, there are certainly bounds due to the dynamics of the mechanism. For example, the biped cannot reach the goal region in one second if the center of mass during the previous qualitative state is 1000 miles away! The plan compiler computes finite bounds for all state variables, for each activity, so that state space bounds implied through the dynamics are all made explicit. Additionally, the plan compiler computes corresponding bounds on control parameters. The bounds on state variables and control parameters prune infeasible trajectories, and thus, ensure feasible state plan execution, as long as the dispatcher observes these bounds. To summarize, the benefits of compilation of the qualitative state plan are: 1) to constrain the feasible region of control parameters that the dispatcher must consider, and 2) to simplify execution monitoring by identifying unsafe states with respect to successful plan execution.

The plan compiler outputs a *qualitative control plan* (QCP), which is similar in form to the input qualitative state plan, but which contains the additional tightened state space, temporal, and control parameter bounds that result from compilation, and are exploited for efficient execution. The state space bounds define flow tubes, which define permissible operating regions in state space. Fig 1.15 shows example flow tubes for the center of mass of the biped. If a trajectory begins in a tube's initial region and stays inside the tube, it will reach the goal region at an acceptable time.

Given a QCP, the job of the dispatcher is to keep each state and control parameter trajectory within its respective tube. Consider the tube shown in Fig. 4.1. If a trajectory begins in the initial region of the tube, the dispatcher chooses control parameter settings, within the bounds specified in the control plan, so that the trajectory is guaranteed to remain in the tube if there are no further disturbances. If the trajectory remains in the tube, it will reach the goal region within the time constraints of the plan.

The dispatcher monitors plan execution by monitoring the SISO abstraction's state. In this way, it checks whether each trajectory is in its tube. When goals are met for each state variable, the dispatcher transitions to the next qualitative state. If a disturbance occurs, the dispatcher compensates by attempting to adjust the SISO control parameter settings, within the bounds specified in the control plan, in order to keep the trajectory inside its tube so that the goal can still be achieved, as shown in Fig. 4.2.

If the disturbance has pushed the trajectory outside its tube, as shown in Fig. 4.3, then the dispatcher aborts, indicating to a higher-level planner that plan execution has failed.

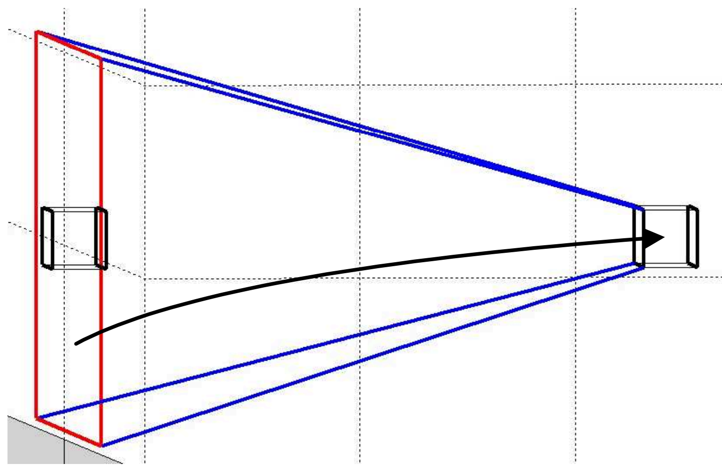


Fig. 4.1 – A flow tube defines a permissible initial region, shown in red, a goal region, shown in black, and an operating tube, shown in blue, that connects the initial and goal regions. If a trajectory, like the trajectory shown in black, begins in the initial region and stays in the tube, it will reach the goal region at an acceptable time.

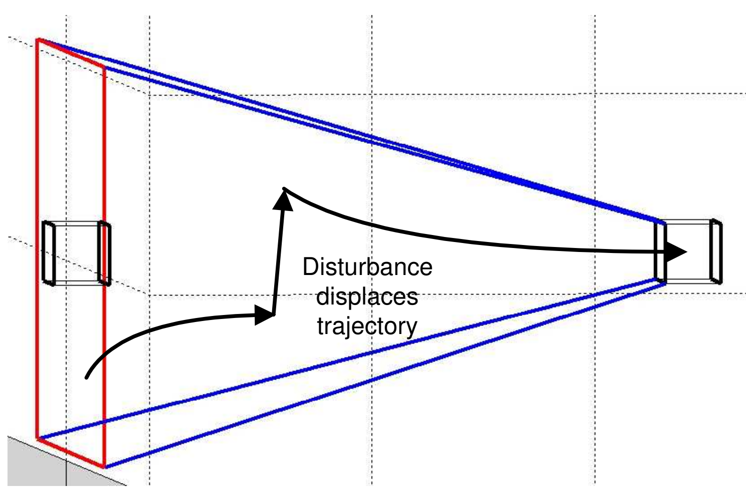


Fig. 4.2 – A disturbance displaces a trajectory in state space. If the disturbance is

small enough, the trajectory remains inside the tube. This implies that the dispatcher will be able to successfully adjust control parameters, so that the trajectory remains in the tube, and reaches the goal at an acceptable time.

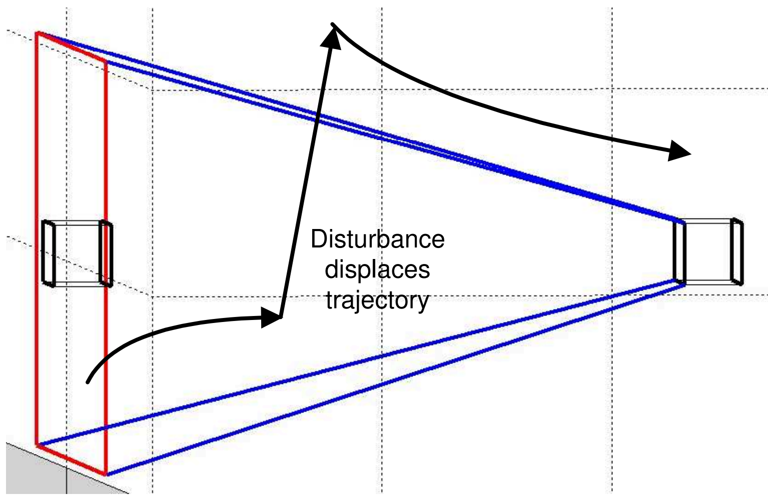


Fig. 4.3 – A disturbance pushes a trajectory outside its tube. This implies that the dispatcher will not be able to adjust control parameters to achieve the goal; the plan fails.

Note that an important role of the dispatcher is one of synchronization. The decoupling provided by the SISO abstraction results in a set of linear systems that appear to be independent. However, they are loosely coupled through temporal constraints, hence arrival in their respective goal regions must be synchronized appropriately to satisfy these constraints. By adjusting control parameters, the hybrid dispatcher accelerates or decelerates an SISO system's trajectory to its goal region, as shown in Fig. 1.16. This type of adjustment allows the hybrid dispatcher to synchronize goal region arrival time so that temporal constraints are satisfied.

4.2.3 Summary of Key Innovations

A key feature of our approach is that the input plan is specified at the task level, using qualitative specifications. We call these specifications qualitative because they are expressed in terms meaningful to the task, and because they are partial and flexible. Plan flexibility supports robustness to disturbances by providing a range of paths, rather than just one path, to plan execution success.

Because the input plan specification does not take into consideration limitations due to plant dynamics or actuation constraints, we use a novel compilation process that prunes invalid trajectories from the input plan specification, resulting in a set of *flow tubes* that represent valid paths to plan success. Thus, the plan is executed successfully if the executive keeps the system's trajectories within these tubes. In generating the flow tubes, the compilation process performs a fully automated synthesis of controllers dedicated to successful execution of the input plan.

The linear virtual element abstraction allows us to apply and generalize concepts previously developed for activity plan execution systems, yielding a hybrid dispatcher that deals with spatial, as well as temporal constraints. Activity plan execution systems observe temporal constraints, but they do not represent continuous spatial constraints, and they ignore the continuous dynamics of the underlying plant. This works perfectly well for many applications, but it is not appropriate for agile, under-actuated, dynamic systems like bipeds, where movement is fast and controllability is limited. The lack of equilibrium points in such systems means that state is constantly changing, and the executive cannot assume the system will wait in a particular state at the end of an activity. Therefore, we extend the techniques used for activity plan execution systems so that they can be used for execution of qualitative state plans.

The decoupling provided by the dynamic virtual model supports definition of plan success in terms of synchronized presence of key high-level state variables, like center of mass, or swing foot position, in goal regions at key points in the gait cycle. For example, if the center of mass moves too far forward before the swing foot moves out, the biped will fall down. Compilation of the qualitative state plan into a qualitative control plan results in a precise specification of operating regions and synchronization requirements that result in successful execution of the plan. This also allows disturbances to be characterized in terms of how they disrupt this synchronization. For example, a trip can be characterized as a delay of forward movement of the swing foot, so that its synchronization with forward movement of the center of mass is disrupted. Furthermore, the operating regions and synchronization requirements in the control plan make clear what has to be done to regain synchronization, in order to recover from a disturbance. For example, to avoid a fall when a trip occurs, the swing foot must be made to move forward faster, in order to make up for the delay, or the forward movement of the center of mass must be slowed. A combination of such corrective actions could also be used to regain synchronization and plan success.

4.2.4 Roadmap

The rest of this chapter provides formal definitions. Section 4.3 provides definitions for the SISO abstraction, and Section 4.4 provides definitions for the qualitative state plan. Because the qualitative state plan and the SISO system state are the inputs to the hybrid executive, and the SISO control parameter settings are the output, these definitions serve as a formal input/output specification of the problem solved by the hybrid executive.

The following three chapters provide technical details of the hybrid executive. Chapter 5 describes a *qualitative control plan*, the compiled from of the qualitative state plan that supports efficient robust execution. Chapters 6 and 7 describe the hybrid dispatcher and the plan compiler. The dynamic virtual model controller is described in Chapter 8.

4.3 Linear Virtual Element Abstraction

Recall from Chapter 1 that the hybrid task-level executive does not control the biped directly, but rather, a linearized abstraction called a *linear virtual element abstraction*, which is easier to control than the actual biped. This abstraction is provided by the dynamic virtual model controller, which is described in detail in Chapter 8. In this section, we describe the linear virtual element abstraction without getting into the details of how it is implemented.

Computing control inputs that achieve a goal state for strongly coupled nonlinear plants is challenging, because the effect of the inputs on plant state is difficult to determine, and because we want to express desired behavior in terms of abstract variables, like *center of mass* (CM) position, rather than joint state space variables, like left knee joint angle. Computing control inputs for the linear virtual element abstraction is much easier, due to the transformation of the tightly-coupled nonlinear plant into a set of seemingly independent, linear, 2nd-order *single-input single-output* (SISO) systems. The effect of a control input on an SISO system is easy to compute analytically, as the solution to a linear 2nd-order differential equation. Furthermore, the SISO abstraction is also a state transformation from the directly actuated joint state representation, where the state vector consists of joint angles and velocities, to a more convenient *workspace* state representation, where the state vector consists of values relevant to balance control, such as center of mass position and velocity, and body orientation and angular velocity. The qualitative state plan state space constraints are expressed in terms of these workspace state variables.

To see how the linear virtual element abstraction operates in more detail, suppose the row vector $\left[\mathbf{x}^T, \dot{\mathbf{x}}^T\right]$ is the position/velocity state vector in the directly actuated joint state representation, and $\left[\mathbf{y}^T, \dot{\mathbf{y}}^T\right]$ is the corresponding position/velocity state vector in the workspace state representation. Elements of $\mathbf{x}$ include joint angle positions, such as left knee joint angle and right hip roll angle. Elements of $\mathbf{y}$ include forward and lateral CM position and swing foot position. A geometric transform, $\mathbf{h}$, is used by the controller to convert from the directly actuated to the workspace state representation.

$$\left[\mathbf{y}^T, \dot{\mathbf{y}}^T\right]^T = \mathbf{h}\left[\mathbf{x}^T, \dot{\mathbf{x}}^T\right]^T \tag{4.1}$$

Suppose the state plan specifies a particular change to the workspace state. The executive performs this change by specifying to the SISO abstraction an acceleration vector, $\ddot{\mathbf{y}}$, for the workspace position state variables. The SISO abstraction implements these changes using a multivariable controller that uses a feedback linearization approach [Slotine and Li, 1991] in order to convert the desired accelerations into a set of joint torques, $\boldsymbol{\tau}$, applied directly to the plant, that achieve the accelerations, as shown in Fig. 4.11. Application of these torques results in a new joint state, $\lfloor \mathbf{x}^T, \dot{\mathbf{x}}^T \rfloor$. The multivariable controller then uses the transformation, $\mathbf{h}$, to convert from directly actuated to workspace state.

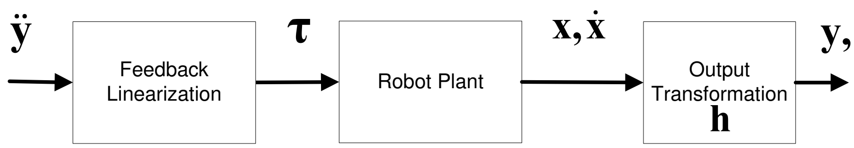


Fig. 4.11 – Feedback linearization transformation

If we draw a black box around the series of transforms in Fig. 4.11, the *multiple-input multiple-output* (MIMO) nonlinear plant appears to be a set of decoupled SISO linear 2nd-order systems, as shown in Fig. 4.12. Each element, $\mathbf{y}_i$ of position vector $\mathbf{y}$, can be viewed as the output of one of the SISO systems, with the corresponding acceleration element, $\ddot{\mathbf{y}}_i$, being the input. Each SISO system can be controlled by a simple linear control law, such as the proportional-differential (PD) law shown in Fig. 4.12. A linear control law is adequate because the plant is linearized.

More precisely, the linear virtual element abstraction is defined as a set of SISO systems, each of which is defined as follows.

Definition 4.1 (SISO System): A single-input single-output (SISO) system is a tuple, $\left\langle y_{set}, \dot{y}_{set}, k_p, k_d \right\rangle$, where y_{set} and $\dot{y}_{set}$ are position and velocity setpoints, and k_p and k_d are proportional and differential gains. The *state* of an SISO system is given by the tuple $\left\langle y, \dot{y} \right\rangle$, which gives the position and velocity of the second-order system, as shown in Fig. 4.12

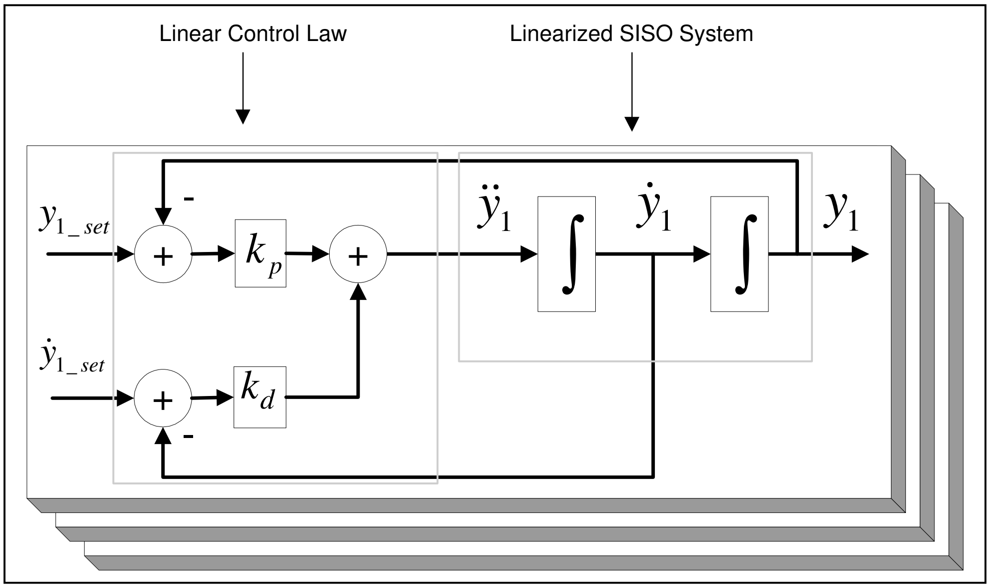


Fig. 4.12 – SISO abstraction

The hybrid executive controls each SISO system by adjusting its setpoints and gains. Given an initial state and control parameter setting, the state trajectory for each SISO system is defined by a linear, second-order, differential equation, hence the state trajectory at any time can be computed analytically. The model-based executive exploits this analytic solution in order to compute control parameters that result in achievement of state plan goals. In addition, to compensate for disturbances, the executive adjusts the control parameters, in order to speed up or slow down the rate at which the state trajectory approaches its current setpoint.

We now describe the analytic solution. The PD control law is of the form

$$\ddot{y} = k_p\left(y_{set} - y\right) + k_d\left(\dot{y}_{set} - \dot{y}\right) \tag{4.1}$$

or

$$\ddot{y} + k_d\,\dot{y} + k_p\,y = k_p\,y_{set} + k_d\,\dot{y}_{set} \tag{4.2}$$

This is a linear, second-order differential equation with a constant right-hand side. The solution to this is

$$y = e^{\alpha t}\left(K_1 \cos \beta t + iK_2 \sin \beta t\right) + \frac{u}{c} \tag{4.3}$$

$$\dot{y} = e^{\alpha t}\left(\beta\left(-K_1 \sin \beta t + iK_2 \cos \beta t\right) + \alpha\left(K_1 \cos \beta t + iK_2 \sin \beta t\right)\right)$$

where

$$\alpha = \frac{-b}{2a}$$

$$K_1 = y(0) - \frac{u}{c} \qquad \beta = \frac{-i\sqrt{b^2 - 4ac}}{2a}$$

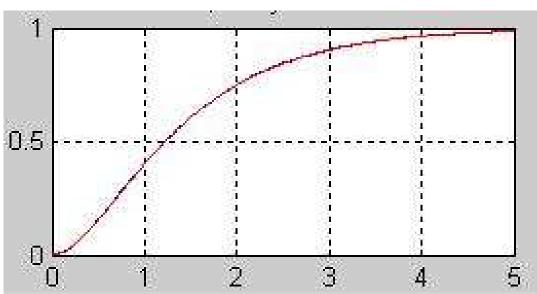
$$K_2 = \frac{i}{\beta}\left(\alpha K_1 - \dot{y}(0)\right) \qquad a = 1$$

$$b = k_d$$

$$c = k_p$$

$$u = k_p\, y_{set} + k_d\, \dot{y}_{set}$$

Fig. 4.13a shows an example of such a solution trajectory, for initial condition $y = 0, \dot{y} = 0$, and with setpoints $y_{set} = 1, \dot{y}_{set} = 0$. By adjusting gains k_p and k_d, the goal position can be achieved more quickly, as shown in Fig. 4.13b, for example, in response to a negative disturbance.



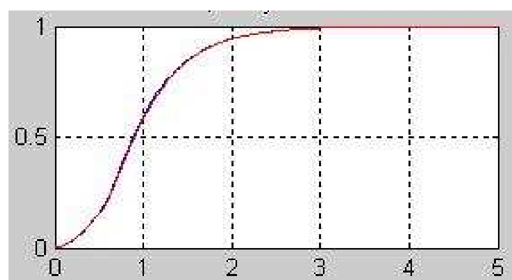


Fig 4.13 - a. SISO PD solution trajectory (left), b. faster response due to adjusted gains

The analytic solution given in Eq. 4.3 is of the form

$$y = f_1\left(t_s, t_f, y(t_s), \dot{y}(t_s), y_{set}, \dot{y}_{set}, k_p, k_d\right) \tag{4.4}$$

$$\dot{y} = f_2\left(t_s, t_f, y(t_s), \dot{y}(t_s), y_{set}, \dot{y}_{set}, k_p, k_d\right)$$

where t_s and t_f are the start and finish times of the trajectory. This analytic solution is leveraged by the hybrid executive both for efficient plan compilation and adaptive control. As described in the previous section, the plan compiler takes into account temporal constraints arising from dynamic limitations, and combines these with ones specified explicitly in the state plan. To accomplish this, the plan compiler uses Eq. 4.4, in order to compute the temporal constraints that are imposed due to dynamics; Eq. 4.4

represents the relation between control inputs, and time needed to reach goal regions. The executive also uses Eq. 4.4, to find control settings that achieve the goal region at the right time, and to adjust these settings when a disturbance occurs.

Trajectories for SISO systems, such as the ones shown in Fig. 4.13, are used to define successful plan execution. Therefore, we now provide a precise definition for an SISO trajectory.

Definition 4.2 (SISO Plant Trajectory): Given an SISO system, S, a plant trajectory of S, $traj(S)$, is a function of time, $y(t)$, that satisfies Eq. 4.4, where the control parameters in Eq. 4.4 are given by S.

Note that this definition requires that the trajectory conform to the dynamics of the SISO system, that is, to Eq. 4.4. Thus, given that $y(t_s)$ and $\dot{y}(t_s)$ are the initial position and velocity conditions for the trajectory, then the rest of the trajectory, $y(t)$, $\dot{y}(t)$ is given by Eq. 4.4.

4.4 Qualitative State Plan

In this section, we begin with an informal description of a qualitative state plan, and provide an example of such a plan. We follow this with a formal definition of a qualitative state plan, and of the problem solved by the hybrid executive.

For most practical applications, a precise specification of state and temporal goals is not necessary. Rather, a loose specification, in terms of state space regions and temporal ranges, is preferable in that it admits a wider set of possible solutions. This may be exploited, for example, to improve optimality or to adapt to disturbances. An example state space goal is for the biped's center of mass position to be within a particular region. An example temporal goal is that this state space goal be achieved after 5 seconds, but before 6. We exploit this flexibility in the goal specification to make handling of disturbances easier; sufficient goal flexibility allows the executive to achieve plan success, even if there is an unforeseen disturbance.

Reaching a goal location may require the biped to take a sequence of steps. Such steps represent transitions through a sequence of fundamentally different states, defined by which feet are in contact with the ground. Thus, a stepping sequence consists of alternating between double support phases, where both feet are on the ground, and single support phases, where one foot (the stance foot) is in contact with the ground, and the other foot (the swing foot) is taking the step. These phases represent qualitatively different system states, with correspondingly different behaviors.

Analysis of locomotion in terms of qualitative behavior has a rich history. In the late 19th century, Ewearde Muybridge performed a series of photographic studies of animal and human locomotion. These photographs were later compiled into a book [Muybridge, 1955]. These studies were performed before the advent of motion picture technology. However, by using a stop-action photography technique, Muybridge was able to show, clearly, different phases of gait cycles for various animals. For example, Muybridge's photographs revealed the different gait patterns used by horses, such as trotting and galloping. Muybridge also photographed human locomotion, such as the walking sequence shown in Fig. 4.14.

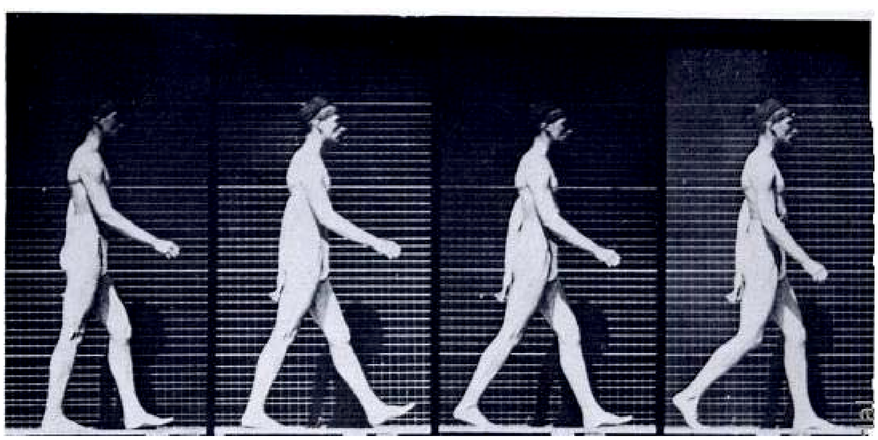


Fig. 4.14 – Human walking sequence. The second photograph shows left heel strike. The fourth photograph shows right toe-off.

Bipedal gait patterns for humans are simpler than quadrupedal gait patterns. The basic cycle for walking is an alternation between single and double support qualitative states, as discussed previously. In Fig. 4.14, the first photograph on the left in the sequence shows a single support qualitative state, with the right leg being the stance leg, and the left leg being the stepping leg. The second photograph shows heel-strike of the left foot, which represents a transition from a single support to a double support qualitative state. The third photograph shows a double support qualitative state, with both feet on the ground. The fourth photograph shows toe-off of the right foot, which represents a transition from a double support to a single support qualitative state. Since Muybridge, these sorts of qualitative behavior descriptions have been used extensively for analysis and control of bipedal walking [Pratt et al., 1997].

We define a *qualitative state* as an abstract constraint on desired position, velocity, and temporal behavior of the biped. A qualitative state indicates which feet are on the ground, and includes constraints on foot position. It may also include state space constraints on quantities like the biped's center of mass, and temporal constraints specifying time ranges by which the state space goals must be achieved. Thus, a qualitative state is a loose, partial specification of desired behavior for a portion of a walking gait cycle. A sequence of qualitative states represents intermediate goals that lead to the final overall task goal, as shown in Fig. 4.1. Such a sequence forms a qualitative state plan.

For example, a plan for a biped divides the walking cycle into a sequence of qualitative states representing single and double support gait phases. Such a plan is shown in Fig. 4.14. In this plan, the first qualitative state represents double support with the left foot in front, the second, left single support, the third, double support with the right foot in front, and the fourth, right single support. The fifth qualitative state repeats the first, but is one gait cycle forward.

A qualitative state plan has a set of *activities* representing constraints on desired state evolution of workspace state variables. Activities are indicated by horizontal arrows in Fig. 4.14, and are arranged in rows corresponding to their associated state variables. In Fig. 4.14, the activities *left foot ground 1* and *left foot step 1* are for the left foot, *right foot ground 1*, *right foot step 1*, and *right foot ground 2* are for the right foot, and CM1 – 4 are for the center of mass.

Every activity starts and ends with an *event*, represented by a circle in Fig. 4.14. Events represent the beginning of an activity, and also, simultaneously, the end of the previous activity. Note that this implies an important temporal constraint; activities are not allowed to “wait” to start after the end of the previous activity; they must start at the previous activity’s end. Events in this plan relate to behavior of the stepping foot. Thus, a *toe-off* event represents the stepping foot lifting off the ground, and a *heel-strike* event represents the stepping foot landing on the ground. These events are so named because, during normal walking, the last point of contact when the stepping foot lifts off the ground is near the toe, and the first point of contact when the stepping foot lands is near the heel, as shown in Fig. 4.2.

Events define the boundaries of qualitative states. Thus, the right toe-off event defines the end of the first qualitative state (double support), and the beginning of the second qualitative state (left single support). Similarly, the right heel-strike event defines the end of the second qualitative state and the beginning of the third; the left toe-off event defines the end of the third and the beginning of the fourth, and the left heel-strike event defines the end of the fourth and the beginning of the fifth.

The qualitative state plan in Fig. 4.14 has a temporal constraint between the start and finish events (between the beginnings of the first and fifth qualitative states). This constraint specifies a lower and upper bound, $[lb, ub]$, on the time between these events. Such temporal constraints are useful for specifying bounds on tasks consisting of sequences of qualitative states. The temporal constraint in Fig. 4.14 is a constraint on the time to complete the gait cycle, and thus, can be used to specify walking speed.

In addition to temporal constraints, qualitative state plans include state space constraints. These are associated with activities, and are specified as rectangles in position/velocity state space. Such rectangles can be used to specify required initial and goal regions, as shown in Fig. 4.15. If an initial region is specified for an activity, then the trajectory must be within this initial region, in order for the activity to begin. For the goal region, the position/velocity rectangle is stretched over a time interval to form a rectangular parallelepiped (box) in position/velocity/time space. This expresses the requirement that the trajectory be in the goal position/velocity rectangle within this time interval, in order for the activity to finish successfully. In Fig. 4.14, the goal region constraint $CM \in R_1$ represents the requirement that the CM trajectory must be in region R_1 for the CM movement activity to finish successfully.

In addition to rectangular initial and goal regions, an activity may also have operating region constraints that specify valid regions in state-space where the trajectory must be over the entire duration of the activity. These are of the form $g(y_i, \dot{y}_i) \le 0$, and they may be linear or nonlinear. Such constraints are used to express actuation limits. For example, CM movement in the plan of Fig. 4.14 is represented by four separate activities: CM1 – CM4. Only CM4 has a goal region. However, each of these activities have different operating regions. This is due to the discontinuous changes in the base of support resulting from the foot contact events; the base of support in double support is very different from the one in single support. Thus, for CM1, the base of support is the polygon defined by r1 and l1 in Fig. 4.14. For CM2, it is the polygon defined by l1 only.

As described in Chapters 1 and 3, the base of support has a strong effect on the maximum force that can be exerted on the CM. This is why these operation constraints must be defined; they represent actuation limits for the CM activities. This is why CM movement in the plan of Fig. 4.14 is represented by four activities instead of only one.

There are several benefits to using a qualitative state plan to specify desired behavior. The fact that the qualitative state plan specifies a sequence of desired states that the plant should be in, rather than a sequence of commands, allows the generator of the plan to focus on goals to be achieved, rather than on their means of achievement. This is a significant convenience over approaches that require detailed command sequences to be input explicitly. The fact that the state space region and temporal constraints are partial, because they are not, necessarily, specified for every activity, and the fact that they specify ranges rather than points gives the plan flexibility that we exploit to handle disturbances.

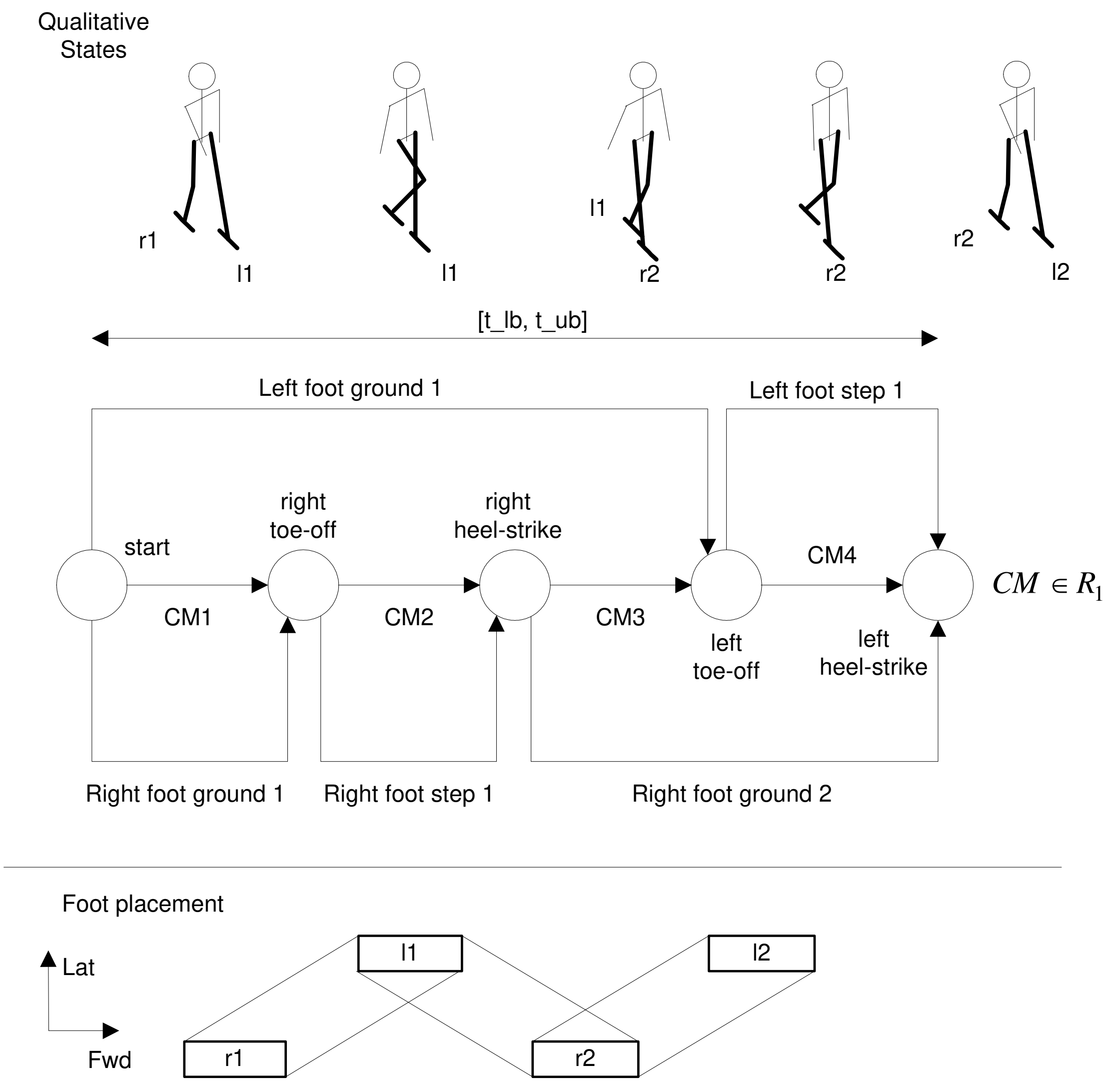


Fig. 4.14 – Example qualitative state plan for walking gait cycle. Circles represent events, and horizontal arrows between events represent activities. Activities may have associated state space constraints, such as the goal region constraint $CM \in R_1$, which specifies a goal for CM position and velocity. Foot placement constraints are indicated at the bottom; for example, rectangle r1 represents constraints on the first right foot position on the ground, and rectangle l1 on the first left foot position. The lines between the rectangles define the polygon of support when in double support.

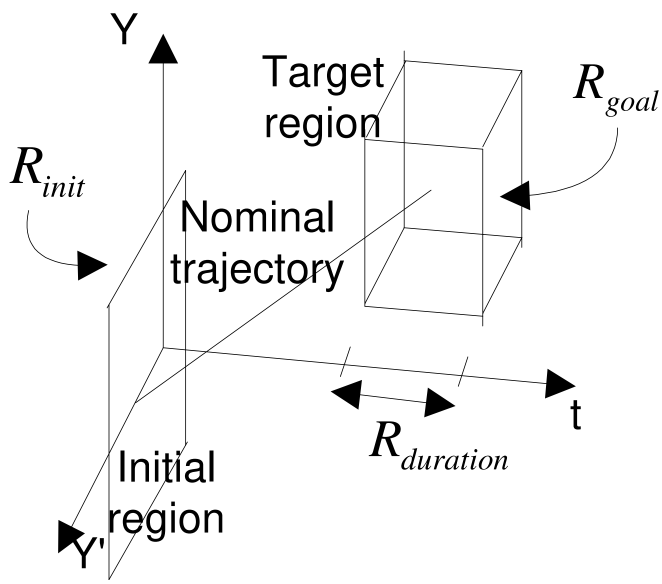


Fig. 4.15 – Initial and goal regions for an activity

4.4.1 Qualitative State Plan Definition

At this point, we are ready to formally define the problem solved by the hybrid executive in terms of its inputs and outputs. The inputs are the qualitative state plan, and the SISO system state. The outputs are SISO system control parameters. The previous discussion provided an intuitive, example-driven description of a qualitative state plan. We now proceed to a more formal definition, which specifies the valid syntactic structure.

Definition 4.3 (QSP): A qualitative state plan (QSP) is a tuple $\langle E, A, TC \rangle$, where E is a set of events (Def. 4.6), A is a set of activities, (Def. 4.4), and TC is a set of externally imposed temporal constraints on the start and finish times of the activities (Def. 4.5).

For example, the QSP of Fig. 4.14 has six activities and one temporal constraint.

Definition 4.4 (Activity): An activity is a tuple $\left\langle ev_s, ev_f, R_{op}, RS_{init}, RS_{goal}, S, A_{next} \right\rangle$, where

ev_s is an event, (Definition 4.6), representing the start of the activity,

ev_f is an event representing its finish,

R_{op} is a set of state-space operational constraints that must hold for the duration of the activity,

RS_{init} is a state-space region constraint that must hold for the activity to begin,

RS_{goal} is a state-space region constraint that must hold for the activity to finish,

S is an SISO system (Definition 4.1), that is associated with the activity, and

A_{next} is an optional successor activity.

Each element of R_{op} is a constraint of the form $g(y, \dot{y}) \leq 0$, where y and $\dot{y}$ are the position and velocity state variables of S. RS_{init} is a tuple $\langle y_{\min}, y_{\max}, \dot{y}_{\min}, \dot{y}_{\max} \rangle$, which defines a rectangular region in the position-velocity state space of the associated SISO system. RS_{goal} is a tuple similar to RS_{init}. The finish time of an activity coincides with the start of its successor, if one exists: $t(ev_s(A_{next})) = t(ev_f)$.

The rectangular regions in RS_{init} and RS_{goal} represent explicitly specified bounds on the state of the SISO system. Note that these bounds are optional in that lower bounds may be negative infinity, and upper bounds may be infinity.

As shown in the example QSP of Fig. 4.14, the activities are arranged in rows, via the A_{next} links. All activities in such a row share the same SISO system, S. In addition, as specified in Definition 4.4, the finish time of an activity coincides with the start of its successor. Thus, transition from an activity to its successor is immediate upon completion of the activity; as stated before, no waiting is allowed. Fig. 4.15 shows examples of initial and goal regions.

Definition 4.5 (Temporal Constraint): A temporal constraint is a tuple $\langle ev_1, ev_2, l, u \rangle$, where ev_1 and ev_2 are events (Def. 4.6), and l and u represent lower and upper bounds on the time between these events, where $l \in \Re \cup \{-\infty\}$, $u \in \Re \cup \{\infty\}$ such that $l \leq t(ev_2) - t(ev_1) \leq u$.

In the QSP of Fig. 4.14, the temporal constraint restricts the time between the start and finish events. Events are used to represent start and finish times of an activity, as in

Definition 4.4, and can be constrained by temporal constraints, as in Definition 4.5. An event is defined in the following way.

Definition 4.6 (Event): An event, ev, represents a point in time. For a schedule, T (Def. 4.8), the specific time of ev, is given by $T(ev)$.

Definitions 4.1, for an SISO system, and 4.3, for a QSP, define the input to the hybrid executive. Having formally defined this input, we are now in a position to define the problem solved by the executive in terms of a successful execution of a QSP.

4.4.2 Problem Solved by The Hybrid Executive

The problem solved by the hybrid executive is successful execution of a qualitative state plan. Successful execution can be expressed in terms of the previous definitions for SISO systems and QSP's, by defining satisfaction of a QSP by a set of SISO plant trajectories and a schedule. This is similar to the definition used in a recently developed system for controlling cooperative air vehicles [Leaute, 2005].

Definition 4.7 (Satisfaction of a QSP): Given a qualitative state plan, qsp (Def. 4.3), a set, Y, of SISO plant trajectories (Def. 4.2), and a schedule, T (Def. 4.8), then qsp is satisfied by $\langle Y,T \rangle$ if T is consistent with qsp (Def. 4.8), and $\langle Y,T \rangle$ satisfies all activities in qsp (Def. 4.9).

Definition 4.8 (Consistent Schedule): Given a qualitative state plan, qsp (Def. 4.3), a *schedule*, T, is an assignment of a specific time, to each event of qsp. T is consistent with qsp if it satisfies all temporal constraints in qsp, that is, for each temporal constraint, $tc \in TC(qsp)$, if $ev_1 = ev_1(tc), ev_2 = ev_2(tc)$, then a schedule assigns $t(ev_1) = T_1, t(ev_2) = T_2$ such that $l(tc) \le T_2 - T_1 \le u(tc)$.

Definition 4.9 (Satisfaction of an activity): Given an SISO system, S (Def. 4.1), a plant trajectory, $y(t)$, for S (Def. 4.2), and a schedule, T (Def. 4.8), then an activity, a, of S (Def. 4.4) is satisfied by $y(t)$ and T if the following conditions hold:

1) $y(t)$ must satisfy the initial and goal region state space constraints of a. Let $t_s = T(ev_s(a))$ be the start time of a under schedule T, and $t_f = T(ev_f(a))$ be the finish time. Then $y(t)$ satisfies the initial and goal region constraints if $\langle y(t_s), \dot{y}(t_s) \rangle \in RS_{init}(a)$ and $\langle y(t_f), \dot{y}(t_f) \rangle \in RS_{goal}(a)$. The membership of a trajectory point in a region, is defined as $y_{\min}(RS) \le y \le y_{\max}(RS) \wedge \dot{y}_{\min}(RS) \le \dot{y} \le \dot{y}_{\max}(RS)$.

2) $y(t)$ must satisfy the operating region state space constraints of a. That is, for each operating region constraint, $g \in R_{op}(a)$, it must be the case that $g(y(t), \dot{y}(t)) \le 0, \forall t : t_s \le t \le t_f$.

Definition 4.7 ensures satisfaction of a QSP by ensuring that the state trajectories are consistent with the state space and temporal constraints of the QSP, and also are consistent with the plant dynamics. Consistency with temporal constraints is ensured through Definition 4.8. Consistency with state space constraints is ensured through Definition 4.9. Consistency with plant dynamics is ensured through Definition 4.2, which requires the state trajectories to be consistent with the SISO plant dynamics.

We conclude this chapter by defining the problem solved by the hybrid executive in terms of the previous definitions for QSP satisfaction.

Definition 4.10 (Problem solved by the hybrid executive): Given a qualitative state plan, qsp, a set of SISO systems, serving as an abstract biped plant, a current time, t_c, and an estimate, $\hat{\mathbf{y}}$, of the biped plant's current state vector, the problem solved by the hybrid executive is to find a sequence of control parameter settings for each SISO system such that qsp is satisfied, according to Definition 4.7. If this is not possible, the executive must abort and signal that plan execution has failed.

We do not address the problem of state estimation in this thesis, and we assume that $\hat{\mathbf{y}}$ is an accurate estimate of the true biped plant state, $\mathbf{y}$. Ideally, the evolution of this plant state matches the evolution of corresponding state variables in the set of SISO systems, which serves as a plant model. However, due to disturbances, and to inaccuracies in the dynamic virtual model controller, this will generally not be the case. Thus, a trajectory of the true plant state, $y(t)$, will generally not satisfy Definition 4.2.

This could be solved by including additional error terms in Eq. 4.4. However, getting the true plant state trajectory to satisfy Definition 4.2 is not the ultimate goal; we are interested in successful execution of the QSP. Therefore, we take the approach of beginning from the current state, $\mathbf{y}$, and assuming that there will be no further disturbances, and that there are no inaccuracies in the dynamic virtual model controller. This is not true, but it is the best approach possible, given the information at the current time. Therefore, the hybrid executive generates a sequence of control parameter settings and projects *future* state trajectories that are consistent with the plant dynamics, according to Definition 4.2. Even though the actual plant trajectories will not match these exactly, they are the best control choices given the most recently available state information. As disturbances occur, and estimated state information is updated, the hybrid executive updates its control parameter settings, if necessary, to compensate. The topic of disturbances is addressed in more detail in the next chapter.

This completes our definition of the hybrid executive's inputs, and the problem it solves. In the next chapter, we define the qualitative control plan, a key intermediate result generated by the hybrid executive.

5 Qualitative Control Plan

Recall that the challenge of execution is that, in order to handle disturbances, the hybrid executive must generate state and control parameter trajectories in real time. However, as discussed in Sections 1.3.1 and 1.3.2, searching the space of possible trajectories for one that is consistent with all plan requirements, and with the plant dynamics, is intractable, due to the dimensionality of the state space, and the coupling between current and future state. To achieve tractability, we construct, at compile time, a *qualitative control plan* (QCP), which uses flow tubes to represent all trajectories that satisfy the QSP and the plant dynamics. The size of the search space for each flow tube is dramatically smaller than the original space. Using the QCP, the executive achieves efficiency by selecting an appropriate trajectory, within each flow tube, that begins at the current system state. Because such a trajectory is within a flow tube, it is guaranteed to lead the system to achieving the plan goals. The executive performs this selection by appropriately adjusting the control parameters for the SISO system associated with each flow tube.

As introduced in Chapter 1, the QCP is generated, offline, by a *plan compiler*, and is executed by a *hybrid dispatcher*. These components comprise the hybrid executive, as shown in Figs. 1.14 and 1.16.

In this chapter, we discuss requirements for the QCP and introduce our approach to achieving these requirements. We conclude the chapter with a formal definition of a QCP, which serves as an output specification for the plan compiler, and an input specification for the hybrid dispatcher. In Chapter 6, we describe the hybrid dispatcher, and how it executes a QCP. In Chapter 7, we describe how the plan compiler automatically generates a QCP from a QSP, taking into account dynamic limitations of the plant.

5.1 Requirements of the Qualitative Control Plan

In order to understand requirements of the QCP, we first review requirements for successful QSP execution, as defined in Section 4.4.2, and then consider what the hybrid executive must know in order to perform this execution efficiently. This leads to requirements for an appropriate representation of feasible trajectories, such that the hybrid dispatcher is able to search this representation efficiently at run time in order to select an appropriate trajectory.

In order to execute a QSP successfully, the hybrid executive must find a consistent schedule and trajectory set, as specified in Definitions 4.7 – 4.9. The temporal and state space constraints explicitly specified in the QSP, and implicit in the plant, restrict the feasible trajectory set. Deducing these restrictions at runtime is not tractable due to the extensive search required, as discussed in Sections 1.3.1 and 1.3.2. To solve this problem, we generate, at compile time, flow tubes that represent the feasible trajectory sets.

Given the decision to use a partial compilation approach, we must now design the QCP, the compiled representation that contains the flow tubes. In order to do this, we must first consider requirements for representing flow tubes of feasible trajectories.

5.1.1 Flow Tube Representation Must Include Only Feasible Trajectories

Due to the interaction of the constraints explicitly specified in the QSP, and the constraints due to plant dynamics, the set of feasible trajectories has a complex geometry. Therefore, any tractable flow tube representation will be an approximation of the feasible set.

There are two basic approaches for such an approximation: *internal* and *external* [Kurzhanski and Varaiya, 1999, 2005; Casagrande et al., 2004]. An internal, or *under-approximation* includes only feasible trajectories. Such an approximation excludes all infeasible trajectories, and it may also exclude some feasible ones. It is called internal because it is completely inside the true flow tube. An external, or *over-approximation* includes all feasible trajectories. Such an approximation may also include some infeasible trajectories. It is called external because it completely surrounds the true flow tube.

An external approximation has the advantage that it does not exclude any valid solutions; it is complete. However, because it may include invalid solutions, there is some finite probability that decisions based on this representation will lead to execution failure. The internal approximation is more conservative; any solution from this approximation is guaranteed to be valid. This has the advantage that it provides the ability to guarantee successful execution, or to guarantee safe operation [Bhatia and Frazzoli, 2004]. However, it has the disadvantage that be it may exclude valid solutions; it is incomplete.

In choosing an approach for our flow tube representation, we must consider execution requirements for the hybrid dispatcher. A key requirement is speed; the dispatcher must be able to perform its calculations for executing a QCP in real time in order to control a real biped. Therefore, simplifying dispatcher computation is a high priority in our design. One way to accomplish this simplification is to use a flow tube representation upon which the dispatcher can completely rely. In particular, if the dispatcher chooses a trajectory that is feasible according to the flow tube representation, it should not have to perform an additional check at runtime to verify that it really is feasible. Such a calculation could be costly, if it involves searching a lengthy sequence of activities that lead to a plan goal.

For this reason, we require our flow tube representation to include only feasible trajectories; we use an internal approximation. Thus, the representation may include a subset of all feasible trajectories, but it may not include a superset. This requirement provides the guarantee that any trajectory selected by the dispatcher from a flow tube will succeed, as long as there are no further disturbances to the system.

For example, consider a very simple example QSP; one with a single activity, and a single temporal constraint that requires the duration of the activity (the time between the start and finish events) to be a fixed interval, D. Suppose, also, that the activity has a goal region, R_G, with finite bounds, and that it may also have operating region constraints (see Def. 4.4). The set of feasible trajectories that satisfy such a plan, and the plant dynamics of the associated SISO system, are depicted in Fig. 5.1.

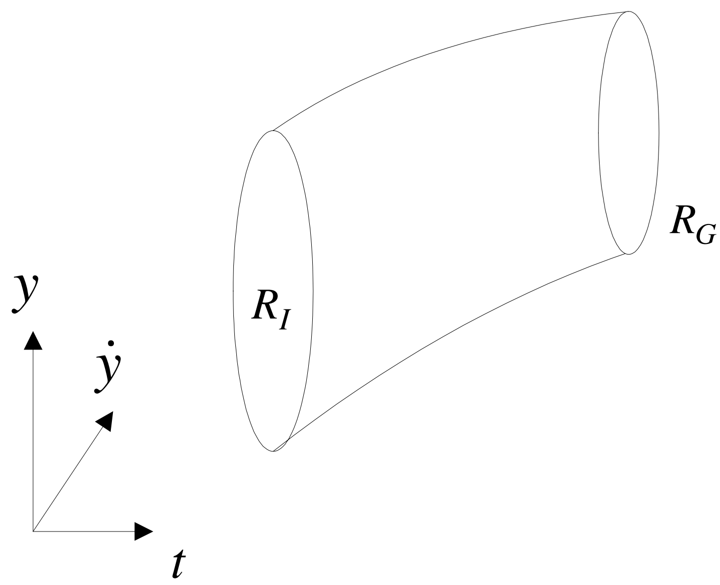


Fig. 5.1 – Flow tube of feasible trajectories that reach region R_G after duration D.

The region, R_I, at the start of the tube in Fig.5.1, is the set of trajectory states at time 0. Any trajectory beginning in this region will reach the goal region, R_G, after duration D. We call this region at time 0 the *initial cross section* of the flow tube, because it is the cross section, in the position-velocity plane, of the flow tube at time 0.

Any flow tube representation that consists of a subset of the feasible trajectories in the full set shown in Fig. 5.1 satisfies the above-stated primary requirement. Thus, a representation that consists of all feasible trajectories satisfies the requirement. Alternatively, the representation shown in Fig. 5.2, which consists of a subset of the feasible trajectories, also satisfies the requirement.

The requirement allows the dispatcher to always make a satisfactory decision based on the current state information. Thus, the dispatcher must select a trajectory that is consistent with the current state, and that is in the flow tube. If this is possible, execution will succeed as long as there are no further disturbances.

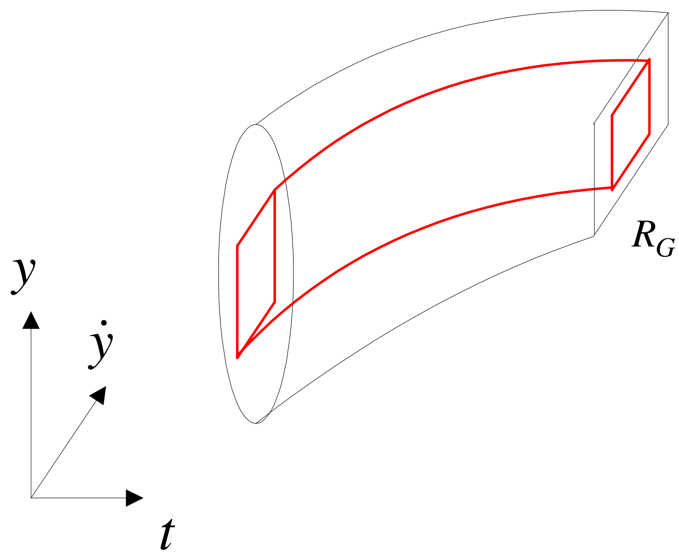


Fig. 5.2 – Flow tube representation, shown in red, which consists of a subset of all feasible trajectories shown in Fig. 5.1.

The requirement that the flow tube representation contain a subset of the feasible trajectories is useful, but it is, potentially, overly conservative, if the representation contains only a very small subset. As shown in Figs. 4.2 and 4.3, the tighter the flow tube representation, the smaller the allowable disturbance. Therefore, to maximize robustness, we require that our QCP representation include as many of the feasible trajectories as possible.

5.1.2 Flow Tube Must Represent Goal Region Explicitly

Given that a flow tube of feasible trajectories is determined by a goal region and a duration, D, as shown in Fig. 5.1, we further require that our flow tube representation include an explicit representation of the goal region. This representation is a cross section, in the position-velocity plane, of the flow tube, similar to the initial cross section, but at a time D after the initial time. Given such a cross section for the goal region, any other cross section of the flow tube, corresponding to any time between the initial and final time, can be computed using Eq. 4.4.

5.1.3 Flow Tube Goal Region is Subset of Successor's Initial Region

To understand further requirements for a QCP, we next consider more interesting cases of QSP's. For example, consider a QSP with two activities; A_1 and A_2, where A_2 is the successor to A_1, as defined in Definition 4.4. The QSP has temporal constraints that constrain the duration of A_1 to be D_1, and the duration of A_2 to be D_2. Suppose, also, that A_1 has finite goal region R_{G1} and that A_2 has finite goal region R_{G2}. Both A_1 and A_2 may have operating region constraints.

The set of feasible trajectories for this plan are depicted in Fig. 5.3. Each such trajectory can be divided into two segments; one for A_1, and one for A_2. This is because, as stated in Definition 4.9, any trajectory that satisfies an activity has a trajectory segment determined by the activity's start and finish time. Thus, each feasible trajectory can be divided into trajectory segments, each of which is associated with an activity through the activity's start and finish events. The set of feasible trajectory segments for a particular activity then form the flow tube for the activity, as shown in Fig. 5.3.

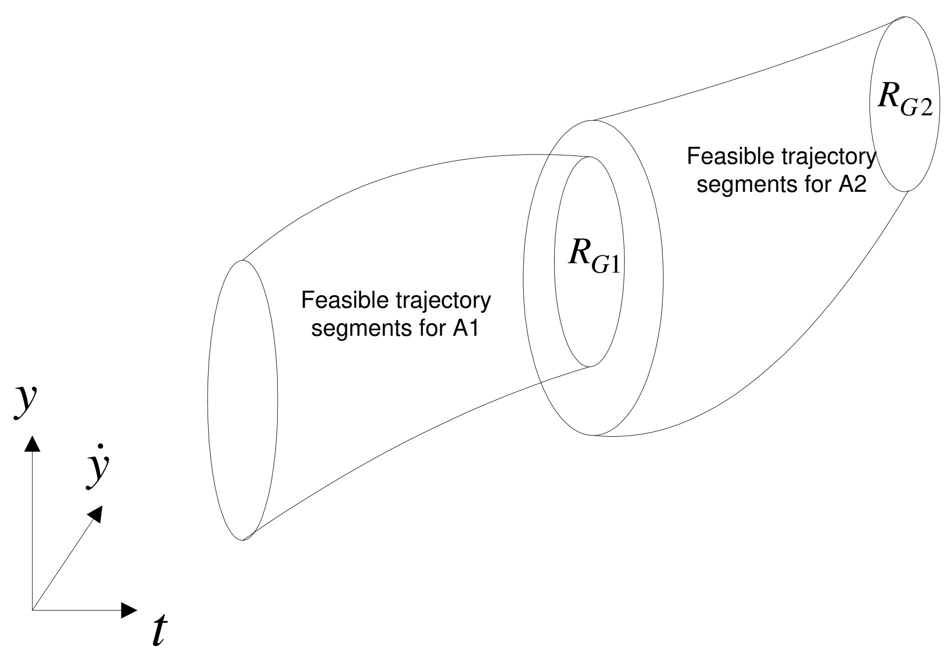


Fig. 5.3 – Feasible trajectory segments for A1 and A2.

Because A_2 is the successor of A_1, any feasible trajectory segment for A_1 must be part of a trajectory that has a feasible trajectory segment for A_2. That is, the final state of such a trajectory segment for A_1 must coincide with the initial state of a trajectory segment for A_2. Therefore, it is a requirement that the goal region for the flow tube for A_1 be a subset of the initial cross section of the flow tube for A_2. More generally, the goal region of a flow tube for an activity must be a subset of the initial cross section of the flow tube of the activity's successor activity.

Note that this requirement may imply a tightening of goal regions specified for activities in the QSP. Consider, for example, the case where R_{G1}, the goal region for A_1 specified in the QSP, is not a subset of the initial cross section of the flow tube for A_2. In this case, the goal region for the flow tube of A_1 is not R_{G1}, but rather, the intersection of R_{G1} with the initial cross section of the flow tube for A_2, as shown in Fig. 5.4.

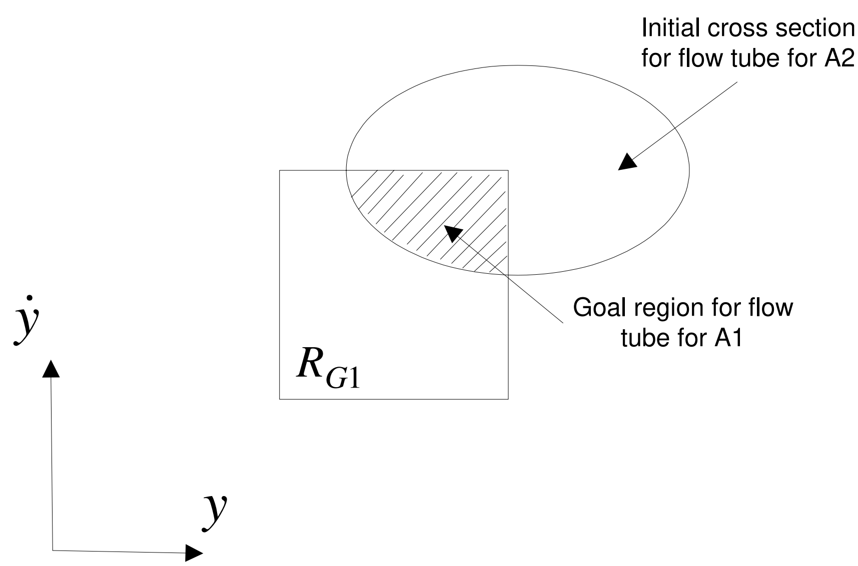


Fig. 5.4 – The goal region of the flow tube for an activity, A1, must be a subset of the initial cross section of the flow tube for the successor activity, A2.

The requirement that the goal region of a flow tube for an activity must be a subset of the initial cross section of the flow tube of the activity's successor activity provides a significant convenience to the dispatcher in that it allows the dispatcher to always make decisions locally, based on the flow tube for the current activity. Thus, if the dispatcher is executing activity A_1 and finds a trajectory segment in the flow tube for A_1 that is consistent with the current state, it knows that the trajectory will not only lead to the goal region of A_1, but also, that this trajectory segment will have a continuation in the flow tubes of all successor activities. This has the essential property that, when executing an activity, the dispatcher need only search the flow tube of that activity, and not the flow tubes of future successor activities.

5.1.4 Flow Tube Must Represent Initial Region Explicitly

In order to support the requirement that the goal region of a flow tube for an activity must be a subset of the initial cross section of the flow tube of the activity's successor activity, we must provide a simple means of checking that a goal region is a subset of an initial region. In order to support this requirement, we further require that, in our flow tube representation, the initial cross section be represented explicitly. This requirement, along with the one stated in Section 5.1.2, specifies that our flow tube representation include explicit representations for both initial and goal region cross sections. These explicit representations must be such that it is easy to check that a goal region is a subset of an initial region.

5.1.5 Requirements for Representations for Flexible Durations

So far, in our discussion of requirements, we have considered example QSP's where activity duration was fixed by temporal constraints. We now consider cases where activity duration is flexible, that is, it is constrained to an interval, not a point. This is important because it is not always possible to fix the schedule of events, and therefore the durations of activities, at compile time. For example, consider the QSP shown in Fig. 5.5. This QSP has two activities; A_1 and B_1, and a temporal constraint that requires that they finish at the same time.

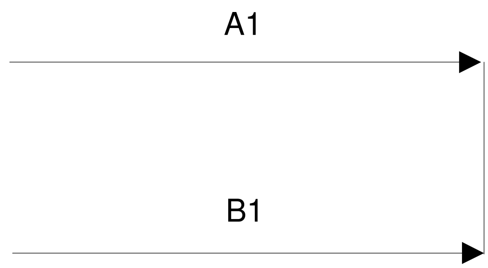


Fig. 5.6 – A QSP with two activities. The vertical bar indicates a temporal constraint that the activities finish at the same time.

If both activities were required to have some fixed duration, D, then the dispatcher would schedule both to start at the same time. Suppose, however, that the start of A_1 is delayed, due to a disturbance. To compensate for such a disturbance, the dispatcher would have to reduce the duration of A_1 and/or increase the duration of B_1. This is possible only if the activities have some flexibility in their duration.

Let's assume, for the moment, that no duration bounds are imposed by the QSP on A_1 and B_1. In this case, the temporal bounds are implied by the dynamic limitations. How can we determine these temporal bounds, and what is an appropriate flow tube representation for this situation?

To address these questions, consider an activity with a goal region, R_G. Fig. 5.1 shows a flow tube for such a region, for a fixed duration D. Suppose, now, that we let D vary over a range such that $l \le D \le u$. This can be represented by a set of flow tubes, one for each value of D, as shown in Fig. 5.7.

Consider, now, the initial cross sections of these tubes. The union of these cross sections, shown in Fig. 5.8a, represents an initial region from which R_G can be achieved in either duration D_1, D_2, or D_3, depending on where in this initial region the trajectory begins. In contrast, the intersection of these cross sections, shown in Fig. 5.8b, represents an initial region from which R_G can be achieved in *any* duration, D_1, D_2, or D_3.

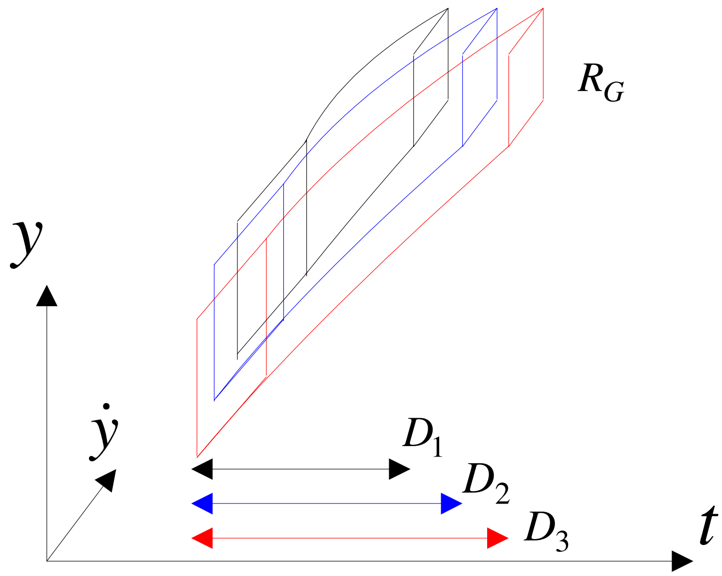


Fig. 5.7 – Flow tube set for variable duration. The longest flow tube, shown in red, reaches the goal region, R_G, after duration D_3. The second-longest flow tube, shown in blue, reaches R_G after duration D_2. The shortest flow tube, shown in black, reaches R_G after duration D_1.

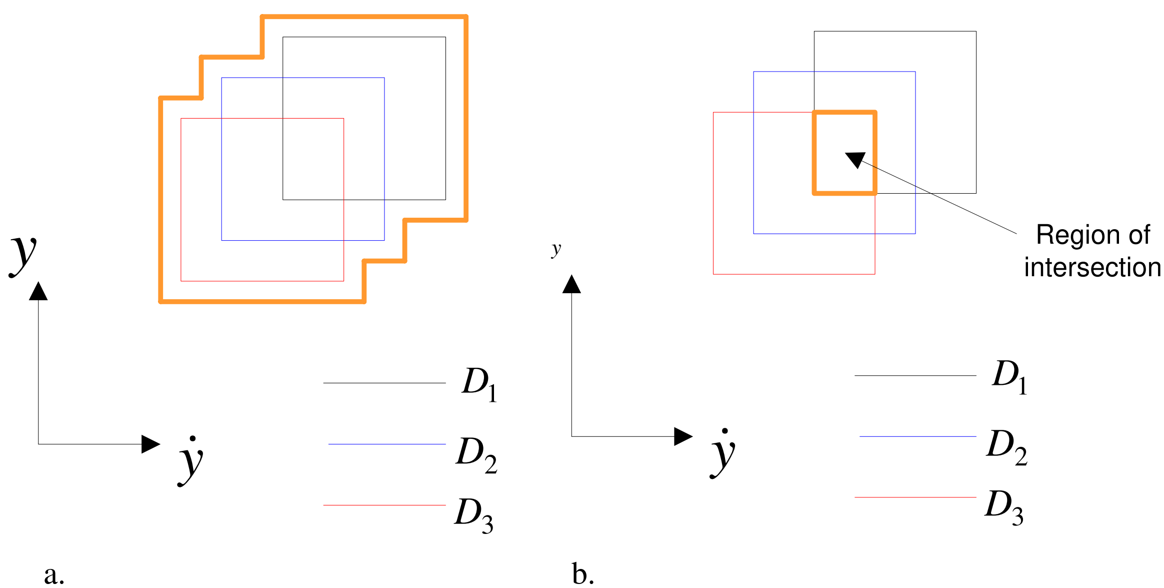


Fig. 5.8 – a. The union of initial cross sections represents an initial region from which R_G can be achieved in one of the durations, D_1, D_2, or D_3. b. The intersection represents an initial region from which R_G can be achieved in any of the durations, D_1, D_2, or D_3.

The initial region of Fig. 5.8b gives the dispatcher more control over duration than the initial region shown in Fig. 5.8a. Note, however, that the initial region of Fig. 5.8b is also smaller, because it is an intersection rather than a union. This illustrates an important trade-off between state-space and temporal controllability; as the desired controllable temporal range, $[l,u]$, increases, the initial region from which trajectories must start becomes smaller. This is because the set of durations, $l \leq D \leq u$, and the corresponding set of flow tubes becomes larger, so the intersection of the initial regions of these tubes becomes smaller, as shown in Fig. 5.8b.

Flow tube sets can also be used to determine temporal constraints implied by plant dynamics and state-space constraints. Fig. 5.9 shows an initial region, similar to the one in Fig. 5.8a, consisting of the union of initial regions of flow tubes of duration D, where $l \leq D \leq u$. Within this region is a second, smaller region, representing an initial region constraint for the activity, specified in the QSP. If this smaller region excludes some of the initial regions of flow tubes in the larger region, then the range of durations possible with trajectories originating from the smaller region is a subset of $[l,u]$.

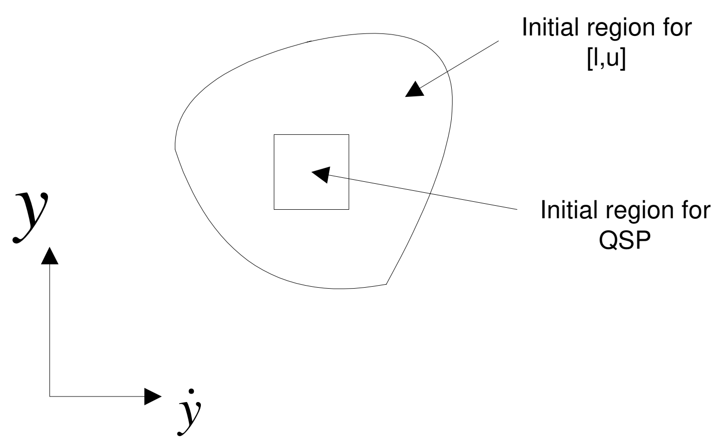


Fig. 5.9 – The larger region is the union of initial regions of flow tubes of duration D, where $l \leq D \leq u$. The smaller, rectangular region is specified as the initial region for the activity by the QSP. If this smaller region excludes some of the flow tubes of the larger region, then the range of possible durations of trajectories originating from the smaller region is a subset of $[l,u]$. Thus, a state-space region constraint specified in the QSP, combined with plant dynamic limitations, implies temporal constraints.

We now return to the example of Fig. 5.6, and consider how the dispatcher can use flow tube sets to make scheduling decisions for activities A_1 and B_1. Suppose that both activities are scheduled to begin at the same time. Due to the temporal constraint requiring simultaneous finish, the dispatcher must find trajectories for A_1 and B_1 such that their durations are the same.

Two approaches are possible. In the first, the dispatcher uses the union of initial flow tube regions, as shown in Fig. 5.8a. Suppose that this union region implies a set of possible durations, D, where $l \le D \le u$. Thus, if a trajectory begins in this region, then the dispatcher is guaranteed to find a trajectory that arrives in the goal region at some time between l and u. Suppose, for example, that the union of initial flow tube regions for A_1 results in a temporal range, $[l,u]$, of possible durations. Suppose, also, that this union, for B_1 results in the same temporal range. Thus, the dispatcher is guaranteed to find feasible trajectories for A_1 and B_1 that arrive at their respective goal regions at some time between l and u. However, there is no guarantee that the time found for A_1 will be the same as the one found for B1. Thus, the intersection of feasible durations for A_1 and B_1 may be null, in which case, A_1 and B_1 have different durations, and the temporal constraint requiring simultaneous finish is violated.

In the second approach, the dispatcher uses the intersection of initial flow tube regions, as shown in Fig. 5.8b. Such an intersection region implies a set of controllable durations; if a trajectory begins in such an intersection region, then the dispatcher is able to arbitrarily control the time of arrival in the goal region, within a range $[l,u]$. Suppose, for example, that the intersection of initial flow tube regions for A_1 results in a temporal range, $[l,u]$, of controllable durations. Suppose, also, that this intersection, for B_1, results in the same temporal range. Because the durations for A_1 and B_1 are fully controllable within this range, the dispatcher is able to arbitrarily choose any duration, D, between l and u, and is guaranteed to find trajectories for A_1 and B_1 that begin in the respective intersection regions for A_1 and B_1, and that have duration D.

Thus, the second approach is superior because it provides compile-time guarantees for successful execution; in the previous example, execution is guaranteed to succeed if the trajectories begin in the respective intersection regions for A_1 and B_1, and if there are no further disturbances. Therefore, we require that the dispatcher use this second approach, and that the QCP provide a suitable representation of intersection regions to support this. For each activity of a QCP, the intersection region must provide sufficient temporal controllability to satisfy temporal constraints, and must be of sufficient size to satisfy state space constraints. Further, for each activity of a QCP, in order to enhance robustness to disturbances, we wish to maximize both the temporal controllable range, and the size of the intersection region. As explained previously, in the discussion of Fig. 5.8, this involves a trade-off between these two goals. We discuss how we resolve this trade-off in Chapter 7.

5.1.6 Requirements to Support Dispatcher Efficiency

In order to support efficient execution, the flow tube representation of feasible trajectories must enable the dispatcher to quickly find a feasible trajectory that matches the current state. Finding this trajectory implies determining a set of control parameters that produce the trajectory. In particular, the dispatcher must be able to: 1) quickly determine whether the current state, and the current control parameter settings result in a trajectory that is within the flow tube, and 2) if this is not the case, quickly determine whether the control parameters can be adjusted to correspond to a trajectory in the tube, and what the adjustment should be.

5.1.7 Requirements for Temporal Constraint Representation

In Chapter 2, we introduced compilation approaches for discrete activity systems, and discussed how a compilation of temporal constraints into a minimum dispatchable graph supports efficient execution [Muscettola, 1998]. Our dispatcher uses a similar approach to deal with the temporal aspect of our problem. Therefore, we require that temporal constraints in a QCP be in this minimum dispatchable form. This allows our dispatcher to use a one-step temporal constraint propagation algorithm, similar to the one used by Muscettola.

Note that the temporal constraints represented in the minimum dispatchable graph must include those specified in the QSP, and those implied by the dynamic limitations of the plant. Temporal constraints due to dynamic limitations are determined using flow tube sets, as described previously in Section 5.1.5.

5.2 Challenges for Qualitative Control Plan Representation

In order to design an appropriate representation for a flow tube, we must consider the geometry of the tube. A flow tube containing the full set of feasible trajectories can have a complex geometry, where cross sections are arbitrary curved regions in the position/velocity phase plane, as shown in Figs. 5.1, and 5.3. This complex geometry presents a significant challenge. Computing a full, explicit representation for all cross sections, for every instant in time, is computationally intractable; such a computation would take a long time, and would consume a large amount of memory. More important than the compile-time computational cost is the fact that this would require a data representation for large sets of geometric points, which, in itself, would interfere with execution efficiency requirements, particularly the ones stated in Section 5.1.6. Therefore, we seek a simpler representation that includes a significant subset of the feasible trajectories in the full tube, as shown in Fig. 5.2, and that satisfies all the requirements stated previously.

A second challenge, besides representation of individual flow tubes, is representation of feasible trajectories for activities with flexible duration. As introduced in Section 5.1.5, flow tube sets can be used to represent feasible trajectories for an activity with flexible duration. Thus, the feasible trajectories of an activity with flexible duration D, where $l \leq D \leq u$, can be represented by a set of flow tubes, all ending in the same goal region, and each having a different value of D. Since D can vary continuously over the range $[l,u]$, the number of flow tubes in this set is infinite, and a representation that includes each one explicitly is intractable. Therefore, we investigate more compact representations, including discretizations of time, and representations that explicitly represent the goal region, and the initial intersection region, but not all the individual initial cross sections of all tubes.

5.3 Qualitative Control Plan Approach

In this section, we develop our QCP representation by discussing our approach to satisfying the requirements and challenges presented in the previous sections. The purpose of this section is to provide the intuitions behind our approach; formal definitions for a QCP are provided in Section 5.4.

5.3.1 Flow Tube Representation Using Goal Region and Duration

To address the first challenge described in Section 5.2, that is, the complex geometry of the flow tubes, we use a simplified representation. Consider, a representation consisting of a rectangular goal region specification and a duration. Thus, the goal region is represented by a tuple, $\langle y_{g\min}, y_{g\max}, \dot{y}_{g\min}, \dot{y}_{g\max} \rangle$. This representation is consistent with the specification of goal regions for activities in a QSP, which have the same rectangular form, as defined in Definition 4.4.

This representation satisfies the requirement stated in Section 5.1.1 because feasible trajectories are fully specified if the goal region and the duration are known. Recall that the feasible trajectories are those that reach the goal region after the specified duration, taking into account initial and operating region constraints and constraints due to plant dynamics. Also, this representation satisfies the requirement stated in Section 5.1.2 because the goal region is represented explicitly by the tuple that defines the rectangle.

Consider now the requirements stated in Section 5.1.6. These offer a critical test of whether this simple representation is explicit enough to support efficient execution. The first of these requirements is that the dispatcher must be able to quickly determine whether the current state, and the current control parameter set result in a trajectory that is within the flow tube. Although cross sections of the tube are not represented explicitly, this requirement is satisfied easily. Recall that the tube is defined as the set of feasible trajectories that achieve the goal region in the required duration. To check whether the current state and control parameter set result in a trajectory that achieves the goal region in the required duration, the dispatcher uses a simple prediction, based on Eq. 4.4. Because the goal region and duration are specified explicitly, this test is accomplished by simply evaluating Eq. 4.4 with the current state and control parameters, to determine the predicted state at the required time. If this predicted state falls within the goal region, then the current state is in the tube. Details of this computation are provided in Chapter 6.

The second requirement of Section 5.1.6 is that if the current state and control parameter set result in a trajectory that is not in the tube, then the dispatcher must be able to quickly determine whether the control parameters can be adjusted to correspond to a trajectory in the tube, and what the adjustment should be. This requirement is also satisfied. As described in more detail in Chapter 6, because the number of control parameters is small, and because the prediction provided by Eq. 4.4 is efficient, we can use a simple optimization algorithm to adjust the control parameters such that the resulting trajectory achieves the goal region at the desired time. If this optimization succeeds, then the current state is on a feasible trajectory, otherwise, it is not.

The key to satisfying these two requirements is the fact that it is easy, due to Eq. 4.4, to predict future trajectory state, and it is easy to check whether this predicted state is in the goal region, because this region is represented explicitly. This is why the simple flow tube representation, consisting of only a goal region and a duration is adequate, and why a fully explicit specification of every cross section of the flow tube, for all times in the duration, is not necessary.

5.3.2 Flow Tube Representation Including Rectangular Initial Region

The requirement stated in Section 5.1.4 requires an explicit representation of the initial region. To satisfy this requirement, we use a rectangular representation for the initial region, just as we do for the goal region. This satisfies the requirement stated in Section 5.1.3; it makes it easy to check if the goal region of a predecessor activity flow tube is a subset of the initial region, as shown in Fig. 5.10.

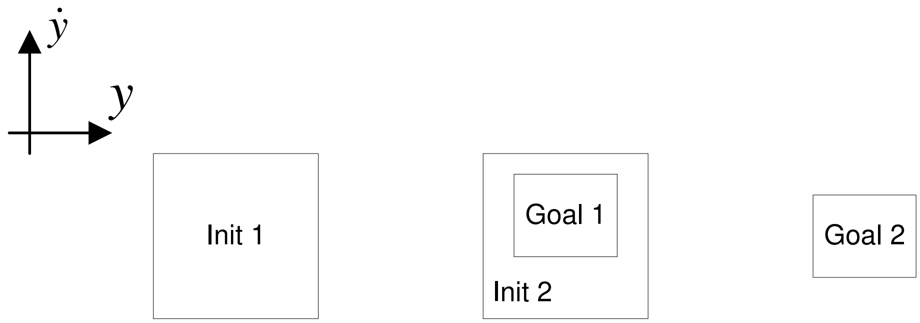


Fig. 5.10 – Sequence of QCP activity flow tube regions. The goal region for activity 1 fits completely inside the initial region for activity 2. Therefore, if the trajectory begins in initial region 1, it will reach goal region 2 at an acceptable time if there are no further disturbances.

To satisfy the requirement stated in Section 5.1.1, that the flow tube representation only include feasible trajectories, we require that the rectangular initial region be a subset of the initial cross section of the full flow tube containing all feasible trajectories. This results in a representation that includes a subset of all feasible trajectories, as shown in Fig. 5.2. Thus, some feasible trajectories are lost for the sake of simplicity of representation. To minimize this loss, we make the initial rectangular region be as large as possible, while still fitting inside the initial cross section of the full flow tube.

Rectangular initial and goal regions have been used in many legged locomotion applications [Pratt et al., 1997; Raibert, 1986]. For these systems, unlike our system, the regions were derived manually. However, rectangular regions were used in these applications for the same reasons that we use them: the representation is simple, yet it provides adequate functionality in that it includes enough feasible trajectories to achieve robust walking.

5.3.3 Flow Tube Representation for Flexible Duration

As introduced in Section 5.1.5, flow tube sets can be used to represent feasible trajectories for an activity with flexible duration. However, as stated in Section 5.2, a full set representation is intractable because the set is infinite. Therefore, we investigate more compact representations of this set.

One approach is to discretize time, using an appropriate increment, Δt. We then include in the set only flow tubes with duration that is a multiple of this increment. With such a discretization, the set becomes finite. As discussed in Chapter 6, the dispatcher operates at a discrete time interval. If the dispatcher time increment is also Δt, then the dispatcher will only perform updates at multiples of Δt, and will only have to consider flow tubes with durations that are multiples of Δt. Therefore, a representation using a finites set of flow tubes, with durations that are multiples of Δt, satisfies the requirements of Section 5.1.5.

A second approach is to use a single rectangular initial region, and a single rectangular goal region, to represent the entire set of flow tubes. Using a single rectangular goal region is easy because all tubes in the set have the same rectangular goal region. The initial region is a rectangular region that is a subset of the intersection region described in Section 5.1.5. Thus, if the system is in a state that falls in this initial region, then the dispatcher can arbitrarily decide any duration between l and u, as discussed in Section 5.1.5.

We use the second approach, as explained in more detail in Section 5.4, because of its simplicity. Chapter 10 provides a discussion of how the first approach might be used.

5.3.4 Example Flow Tubes for QSP

In the QSP of Fig. 4.14, activities CM1 – CM4 represent movement of the center of mass. Activity CM4 has a goal region specified, but CM1 – CM3 do not. CM1 – CM4 have different operating region constraints due to changes in the support base resulting from ground contact events.

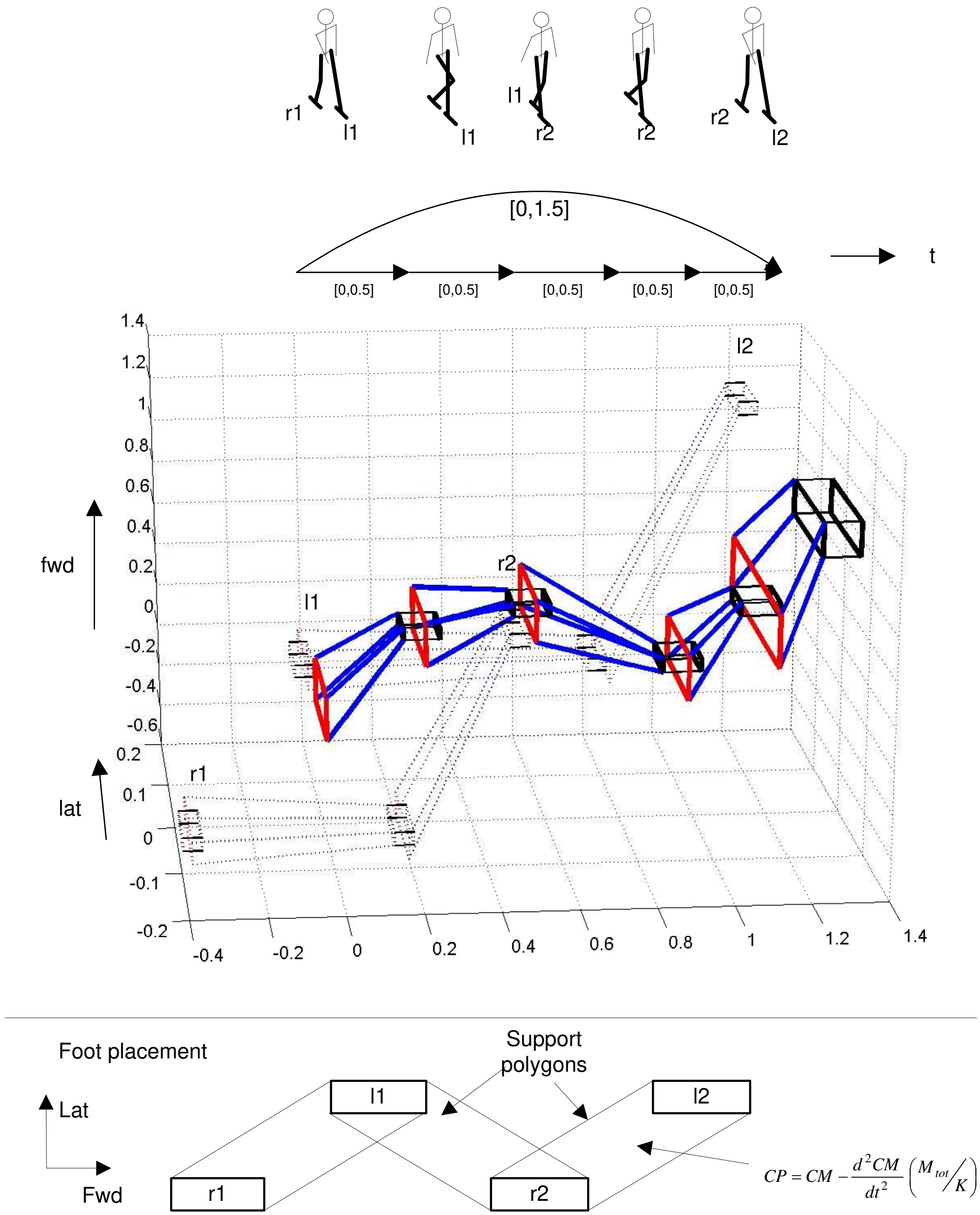


Fig. 5.11 – Flow tubes for CM activities are shown, with initial regions in red, goal regions in black, and tubes in blue. The first four flow tubes, starting from the left, correspond to CM1 – CM4 in Fig. 4.14. Flow tubes for left and right foot position are shown using dotted lines.

To compute flow tubes for these activities, the plan compiler uses the goal region for CM4 as the goal region for this activity's flow tube, but must compute goal and initial regions for the other activities CM1 – CM3, as shown in Fig. 5.11. This figure shows initial regions of flow tubes in red and goal regions in black. Flow tubes between the initial and goal regions are also depicted, in blue, even though these are not actually computed by the plan compiler, but rather, are detected at execution time by the dispatcher, as discussed previously.

5.4 Qualitative Control Plan Definition

The previous discussion of qualitative control plan requirements, and our approach to meeting these requirements, provides an intuitive description of what a QCP is. We now formally define a QCP. This includes a definition of its structure, and also controllability properties it must have in order to be correct with respect to a QSP. We use these properties to define what it means for a QCP to be executed successfully. In particular, we present a theorem that explains how successful execution of a QSP is achieved using a corresponding controllable QCP. Additionally, we define disturbances that may occur during execution of a QCP, and show how the need to handle such disturbances motivates the controllability properties.

5.4.1 Structure of a QCP

A QCP contains all the information needed to control the virtual element abstraction, and to monitor its status with respect to state region and temporal bounds. It has a structure similar to that of a QSP, but augments it with flow tubes on state and control parameters. Recall that the QSP specifies goals that the plant must achieve over time in terms of regions of state space to be achieved at specified time intervals. The QCP specifies feasible state variable trajectories for achieving these goals, using flow tubes that connect the goal regions.

Definition 5.1 (QCP): A qualitative control plan (QCP) is a triple $\langle E, CA, TC \rangle$, where E is a set of events (Def. 4.6), TC is a set of temporal constraints on the events (Def. 4.5), and CA is a set of control activities (Def. 5.2). Each event is either a start event or finish event of a control activity.

Note that Definition 5.1, for a QCP, is identical to Definition 4.3, for a QSP, except that, where a QSP contains activities (Def. 4.4), a QCP contains control activities (Def. 5.2). Thus, Fig. 4.14 shows the structure of both a QSP, and its corresponding QCP, except that for the QCP, the activities are replaced by control activities.

A *control activity* includes the information of the corresponding activity in the QSP, augmented with flow tubes bounding the activity's state variable and corresponding control parameters.

Definition 5.2 (Control activity): A control activity is a tuple $\langle A, CP, l, u, R_{init}, R_{goal} \rangle$, where A is an activity, Def. 4.4, CP contains constraints on the activity's control parameters, and $[l,u]$ represents a temporal bound on the activity's duration due to dynamic limitations of $S(A)$, the SISO system associated with the activity (Def. 4.4). CP is a tuple $\langle y_{set}, \dot{y}_{set}, kp_{\min}, kp_{\max}, kd_{\min}, kd_{\max} \rangle$, denoting limits on control parameters. All elements of this tuple are scalar real values, where y_{set} and $\dot{y}_{set}$ are position and velocity setpoints for $S(A)$'s control law, $kp_{\min}$ and $kp_{\max}$ are bounds on the proportional gain, and $kd_{\min}$ and $kd_{\max}$ are bounds on the damping gain (Def. 4.1). The regions R_{init} and R_{goal} must be subsets of the corresponding regions in A: $R_{init} \subseteq RS_{init}(A)$, and $R_{goal} \subseteq RS_{goal}(A)$.

As we will see, in Definition 5.7, the regions R_{init} and R_{goal} will be used to define a flow tube for the activity such that any trajectory beginning in R_{init} can be made to reach R_{goal} through appropriate control parameter settings, if there is no further disturbance. The time between start in R_{init} and arrival in R_{goal} can be controlled to be any time in the range $[l,u]$, by adjusting the control parameters.

Recall, from Definition 4.1, that each SISO system is characterized by the tuple $\langle y_{set}, \dot{y}_{set}, k_p, k_d \rangle$. The control parameter information of a control activity provides a nominal, fixed setting for the position and velocity setpoints, $y_{set}, \dot{y}_{set}$, of the SISO system associated with the activity, and a range of settings for its k_p and k_d gains, such that the operation constraints, specified by $R_{op}(A)$ are satisfied. The dispatcher is then free to adjust these gains within this range.

For example, the SISO system trajectory shown in Fig. 4.13a results from initial conditions $y = 0, \dot{y} = 0$, setpoints $y_{set} = 1, \dot{y}_{set} = 0$, and gains $k_p = 2, k_d = 3$. By adjusting these gains to $k_p = 8, k_d = 6$, the trajectory reaches the setpoint more quickly, as shown in Fig. 4.13b. Thus, if CP is $\langle y_{set} = 1, \dot{y}_{set} = 0, kp_{\min} = 2, kp_{\max} = 8, kd_{\min} = 3, kd_{\max} = 6 \rangle$, then the dispatcher is free to make this adjustment.

This concludes our formal definition of a QCP. In the next section, we define a correct QCP for a QSP. We follow this, in Section 5.4.3, with definitions of controllability properties that support the definition of a correct QCP. Section 5.4.4 defines successful execution of a QSP using a correct QCP. Section 5.4.5 defines disturbances that may occur during this execution, and discusses how controllability properties support the handling of these disturbances.

5.4.2 Correct QCP for a QSP

In order to specify correct functioning of the plan compiler, it is necessary to define a correct output of the compiler, given a valid input. For the plan compiler, the input is a valid QSP (Def. 4.3), and the output must be a correct QCP for the QSP.

A correct QCP for a QSP is defined in terms of the structure of the QSP, and in terms of the controllability and temporal dispatchability properties. As discussed previously, a QCP for a QSP has the same structure as the QSP; it has the same events and temporal constraints, but the activities of the QSP are replaced with corresponding control activities in the QCP. The control activities have all the information of the QSP activities, augmented with flow tube information (Def. 5.2).

A QCP is *controllable* if its feasible trajectories begin in the initial state space regions specified in the QSP, end in the goal regions specified in the QSP, and satisfy the temporal constraints. The property of controllability of a QCP is defined, in more detail, in Section 5.4.3. A QCP is temporally dispatchable if the effects of event scheduling at execution time can be correctly updated using a one-step, local propagation algorithm.

This property is also defined in Section 5.4.3, and is analogous to that in [Muscettola, 1998].

Definition 5.3 (Correct QCP for a QSP): Given a qualitative state plan, qsp (Def. 4.3), a correct qualitative control plan, qcp, for qsp, has a structure, as defined in Definition 5.1, where the events and temporal constraints are identical to the ones in qsp. Furthermore, for each activity, a, in qsp, qcp has a corresponding control activity, ca, such that $A(ca) = a$ (see Def. 5.2). Furthermore, qcp is controllable, as defined in Definition 5.8, and is temporally dispatchable, as defined in Definition 5.9.

In the next section, we formally define the properties of controllability and temporal dispatchability of a QCP.

5.4.3 Controllable and Temporal Dispatchability of a QCP

To support the definition of controllability of a QCP, we provide a number of definitions related to flow tubes and resulting controllability properties. These definitions also specify requirements for computation for the R_{init} and R_{goal} regions of a control activity, and thus, for computation of flow tubes, one of the key tasks of the plan compiler.

Recall that a flow tube for an activity with a particular desired duration is the set of trajectories such that actuation and dynamic limits of the plant are observed, and such that the goal region is achieved after this duration. We define such a flow tube as a *fixed-duration tube*. We begin with fixed duration tubes, and then extend this concept to flow tubes with flexible durations.

Definition 5.4 (Fixed-duration tube): A fixed-duration tube is a mapping $TUBE : CA, D \rightarrow TRAJ$, where CA is a control activity (Def. 5.2), D is a scalar real value representing a duration for the activity's execution, and $TRAJ$ is a trajectory set. Let s be the SISO system associated with the control activity ($s = S(A(CA))$) (see Def. 5.2). A trajectory, $traj$, is an element of the tube ($traj \in TRAJ = TUBE(CA, D)$) if

1) it reaches the control activity's goal region at the end of the duration; that is, if $(y(D), \dot{y}(D))(traj) \in R_{goal}(CA)$,

2) it observes the activity's actuation limits; if $\forall g \in R_{op}(A(EA)): g(y(t), \dot{y}(t)(traj)) \le 0, \forall t : 0 \le t \le D$, and

3) it is consistent with the plant dynamics; that is, if $traj$ is consistent with s, (Def. 4.2).

As a simple example of a fixed-duration tube, consider the case where the goal region is $R_{goal} = \langle y_{\min} = 2, y_{\max} = 3, \dot{y}_{\min} = 1, \dot{y}_{\max} = 2 \rangle$, as shown in Fig. 5.12c, $D = 2$, and the limits specified in CP are such that $kp_{\min} = kp_{\max} = 0$, and $kd_{\min} = kd_{\max} = 0$. The severe limit on these parameters means that the control input acceleration, $\ddot{y}$, of the SISO system is always 0! Thus, this is a simple, somewhat degenerate example, but it is useful for illustrating a few important properties of fixed-duration tubes. Fig. 5.11 shows cross sections of the tube, for $\langle y, \dot{y} \rangle$, at t = 0, 1, and 2.

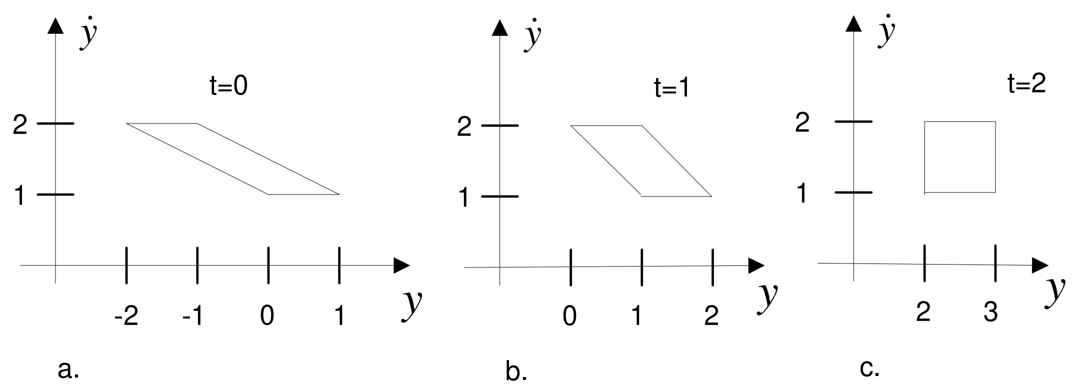


Fig. 5.12 – Cross sections of example fixed-duration tube at t = 0, 1, 2

Suppose that non-zero cross sections of the tube exist for all time points between 0 and D. From Lemma 5.1, if a trajectory is in the tube at any point in time between 0 and D, and if there are no disturbances after that time, then the trajectory is guaranteed to reach the goal region at time D. For example, in Fig. 5.12, suppose a trajectory is at $y = 0, \dot{y} = 1$ at time t=0. Thus, it is in the cross section shown in Fig. 5.12a. If there are no further disturbances, then the trajectory is guaranteed to reach the goal region (Fig. 5.12c) at time t=2, as shown in Fig. 5.13. Note that this is true even though there is no control action for this tube; the parameters are set so that the acceleration, $\ddot{y}$, of the SISO system is always 0. It is a property of the open-loop dynamics of the SISO system.

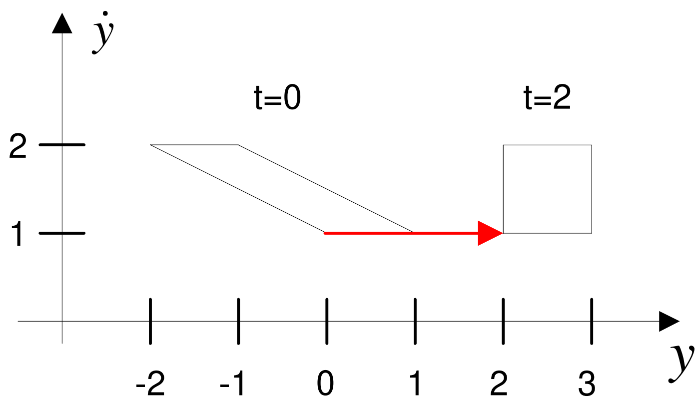


Fig. 5.13 – Trajectory beginning in tube at t=0 reaches goal at t=2 if there are no disturbances.

Continuing with this example, at t=1, the trajectory shown in Fig. 5.13 is at $y = 1, \dot{y} = 1$; it is in the cross section shown in Fig. 5.12b. Suppose that a disturbance occurs at this time. We model such a disturbance as an acceleration spike input to the SISO system. An acceleration spike with area A will cause a step change of A in the trajectory's velocity, because velocity is the integral of acceleration. For example, an acceleration spike with area 0.5 at t=1 pushes the trajectory to $y = 1, \dot{y} = 1.5$, as shown in Fig. 5.14a. This point is still within the tube; it is within the cross section shown in Fig. 5.12b. Therefore, the trajectory still reaches the goal at t=2, despite the disturbance. Suppose now that the acceleration spike has area 2 instead of 0.5. In this case, the disturbance pushes the trajectory to $y = 1, \dot{y} = 3$, which is outside the cross section shown in Fig. 5.12b. The trajectory is therefore not inside the goal region at t=2, as shown in Fig. 5.14b.

This simple example serves to illustrate important properties of fixed-duration tubes, even though there is no control action. For activities where control action is allowed, the tube becomes more complex, geometrically, and includes a larger set of feasible trajectories.

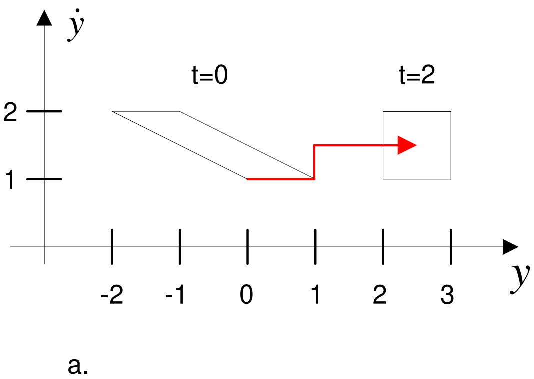


Fig. 5.14 – a. for a disturbance at t=1 with area 0.5, trajectory stays in tube; b. for a disturbance at t=1 with area 2, trajectory is pushed out of tube

We have previously introduced, in Fig. 5.12, the concept of a cross section of a fixed-duration tube. We now define this concept formally, and use it to express a relation between a control activity's initial region and its controllable tube set. We then use this relation to define controllability of a control activity.

Definition 5.5 (Cross section of a fixed-duration tube): Given a fixed-duration tube, $tube = TUBE(CA, D)$, and a time t within the duration ($0 \le t \le D$), a cross section, $\sec = SEC(tube, t)$, of this tube at time t, is a region in the position-velocity plane such that every point in this region is a point on a trajectory of the tube, where the point on the trajectory is at time t. Thus, if $traj \in tube$, then $\langle y(t), \dot{y}(t) \rangle (traj) \in \sec$.

Examples of such a cross section are shown in Fig. 5.12. The initial cross section is the cross section at time $t = 0$. This is defined as $init\sec = SEC(tube, 0)$. Fig. 5.12a shows an example of such an initial cross section.

Definitions 5.4 and 5.5 are used in Lemma 5.1, which provides a guarantee of goal region arrival time.

Lemma 5.1: Given a fixed-duration tube, $tube = TUBE(CA, D)$ (Def. 5.4), and an associated SISO system $s = S(A(CA))$ (Def. 5.2), if the state, $\langle y(t_i), \dot{y}(t_i) \rangle$, of s at time t_i is on a trajectory that is in $tube$, and t_i is between 0 and D, that is, $0 \le t_i \le D$, and if there are no disturbances after this time, that is, during $t_i \le t \le D$, then the state of s is guaranteed to reach the goal region of CA at time D. A state, $\langle y(t_i), \dot{y}(t_i) \rangle$, is in $tube$ at t_i if the trajectory position and velocity at t_i are in the tube's cross section at t_i; $\langle y(t_i), \dot{y}(t_i) \rangle \in SEC(tube, t_i)$ (Def. 5.5).

Therefore, as a special case of this, if an SISO state begins in the initial cross section of a fixed-duration tube, and if there are no further disturbances, then the trajectory will reach the goal region after the desired duration. We now formalize this property by defining controllability of an activity with respect to an initial region and a duration.

Definition 5.6 (Spatial and temporal controllability of a control activity with respect to an initial region and a duration): Let CA be a control activity, D a duration, and R_0 a region in the position-velocity plane. The associated fixed-duration tube is then $tube = TUBE(CA, D)$, and the initial cross section is $init\sec = SEC(tube, 0)$ (Def. 5.5). If the region R_0 is a subset of $init\sec$ ($R_0 \subseteq init\sec$), then the activity is said to be spatially and temporally controllable with respect to R_0 and D.

This implies, through Lemma 5.1, that a state for the control activity will reach the activity's goal region at time D if the state begins in R_0, and if there are no further disturbances. We now utilize this concept to define controllability of an activity.

Definition 5.7 (Controllability of a control activity): Let CA be a control activity. The *controllable tube set* of CA is the set of all fixed duration tubes of CA that contain trajectories that reach the goal region of CA in the temporal range $[l(CA), u(CA)]$. Thus, it is the set of tubes $TUBES_{controllable} = \bigcup_{D} \{TUBE(CA, D) \mid l(CA) \le D \le u(CA)\}$. The intersection of the initial cross sections (Def. 5.5) of the tubes in this set is called the *controllable initial* *region* of the activity; that is, $INITSEC_{controllable} = \bigcap_{tube} SEC(tube,0)$, $tube \in TUBES_{controllable}$. If the rectangular initial region of the control activity is a subset of this set ($R_{init}(CA) \subseteq INITSEC_{controllable}$), then the activity is said to be controllable.

Definition 5.7 is used in Lemma 5.2, which provides a guarantee of goal region arrival time.

Lemma 5.2: Given a controllable control activity, CA (Def. 5.7), if the state for CA is in $R_{init}(CA)$, then a control setting exists that causes the state to reach the activity's goal region, $R_{goal}(CA)$, at any desired time within the range $[l(CA), u(CA)]$, if there are no further disturbances during execution of CA.

Note that the fact that $INITSEC_{controllable}$ is defined as an intersection, rather than a union, of initial cross sections is crucial. Recall that definition as an intersection ensures temporal controllability over the entire range $[l,u]$; if a state is in the intersection, it can be controlled to reach the goal region at any duration in $[l,u]$. Definition as a union would allow for a bigger initial range, but would introduce temporal uncertainty. Such a definition would guarantee arrival in the goal region at some time in the range $[l,u]$, but the exact time in this range would not be controllable. We will return to this distinction later, when we discuss temporal networks and temporal dispatchability.

We now define controllability of a QCP in terms of controllability of its constituent control activities.

Definition 5.8 (Controllability of a QCP): A qualitative control plan, qcp (Def. 5.1), is said to be controllable if

1) all control activities in qcp are controllable, as defined in Definition 5.7,
2) the temporal bounds, $[l,u]$, of all control activities in qcp are consistent with the plan's temporal constraints $TC(qcp)$,
3) the goal regions of all control activities in qcp are subsets of the initial regions of their successors.

This last condition is expressed as

$$\forall ca \in CA(qcp): \exists ca_{next} = A_{next}(A(ca)) \wedge R_{goal}(ca) \subseteq R_{init}(ca_{next})$$

The third condition in Definition 5.8 ensures an unbroken chain of controllable transitions from each activity to its successor, as shown in Fig. 5.10.

As we will show in Section 5.4.4, a controllable QCP can be guaranteed to execute successfully, if all trajectories begin in the initial regions of the plan's initial activities, and if there are no disturbances. No such guarantee can be made if a QCP is not controllable, as defined in Definition 5.8. This is why we require such controllability as one of the properties of a correct QCP for a QSP, as defined in Definition 5.3.

The other key property that Definition 5.3 specifies for a correct QCP is temporal dispatchability. As discussed previously in Section 5.1.7, we require this property in order to support efficient one-step temporal constraint propagation in our dispatcher. We now define this property of a QCP.

Definition 5.9 (Temporal Dispatchability of a QCP): A qualitative control plan, qcp (Def. 5.1), is said to be temporally dispatchable if the distance graph generated from the temporal constraints $TC(qcp)$, and the temporal bounds, $[l,u]$, of all control activities in qcp, is in minimal dispatchable form [Muscettola, 1998].

5.4.4 Successful Execution of a QSP using a Correct QCP

In this section, we discuss guarantees for successful execution provided by a correct QCP for a QSP. Such a successful execution generates trajectories that are consistent with the QSP upon which the QCP is based. Therefore, such an execution represents a solution to the hybrid execution problem, as defined in Definition 4.10.

Our discussion centers on two theorems, one for successful execution of a control activity, and one for successful execution of a QCP. These theorems build on the definitions and lemmas from Section 5.4.3. Both theorems are based on the concept that, if the system begins in an appropriate initial region, and if there are no further disturbances, beyond the previous ones represented in its current state, then successful execution can be guaranteed.

Theorem 5.1 (Successful execution of a controllable control activity): Let CA be a controllable control activity, and s, the SISO system associated with CA ($s = S(A(CA))$, (Def. 5.2). If the state of CA is in $R_{init}(CA)$, and if there are no further disturbances during execution of CA, then there exists a constant control parameter setting $\langle y_{set}, \dot{y}_{set} kp, kd \rangle$ which, when applied to s (Def. 4.1), results in a trajectory $y(t)$, and a duration, D, consistent with a schedule T, such that:

1) the activity in the QSP corresponding to CA, $A(CA)$, is satisfied by $y(t)$ and T, as defined by Definition 4.9
2) D, is within the temporal bounds of CA ($l(CA) \leq D \leq u(CA)$).

This follows from Lemma 5.2, and from Definitions 5.7, 5.4, and 4.2 (see proof in Appendix F). Note that Theorem 5.1 states that the successful trajectory, $y(t)$, is achieved using constant control parameter settings, as long as there are no further disturbances. If there are disturbances, then the dispatcher may have to adjust the control gain parameter settings to compensate. This results in a sequence of gain settings of the form $(kp(t_0), kd(t_0); \ldots; kp(t_i), kd(t_i))$. If the trajectory resulting from this gain sequence stays in the activity's tube, then the activity is still executed successfully.

Theorem 5.1 guarantees successful execution of an activity, in isolation from other activities and events, and from temporal constraints that relate such events. This is not sufficient; we need to make sure, not only that all activities execute successfully in isolation, but also that their start and finish event times are all temporally consistent with the temporal constraints specified in the QSP. Therefore, we use Theorem 5.2, stated below, to provide execution guarantees for a correct QCP, as a whole.

Theorem 5.2 (Successful execution of a correct QCP for a QSP): Let qsp be a qualitative state plan, and qcp, a correct qualitative control plan for qsp. If for each initial activity, CA, in qcp, the state associated with CA is in $R_{init}(CA)$, and if there are no further disturbances, then there exists a schedule, T, and there exist constant control parameter settings for each activity, resulting in trajectory set Y, of SISO plant trajectories (Def. 4.2), such that Y and T satisfy qsp according to Definition 4.7. The set of initial activities is the set of activities with no predecessor.

This follows from Theorem 5.1, Def. 5.8 (see proof in Appendix F). Initial activities are ones with no predecessor.

We can easily make Theorem 5.2 more general by stating it recursively. Consider a correct QCP that is being executed, and that the current time is t_{curr}, where t_{curr} is not, necessarily 0, the initial time. Suppose that execution up to t_{curr} has resulted in a trajectory set Y_{curr}, and a schedule of events, T_{curr}, that satisfy qsp, according to Definition 4.7, through time t_{curr}. Then, Theorem 5.2 applies for completion of successful execution, if we consider the set of initial activities to be the union of the set of activities that are successors to activities currently being executed, and the set of activities that have not yet started, and that do not have predecessors.

Theorem 5.2 provides guarantees for successful execution of a QSP provided by a correct QCP for the QSP. Such an execution represents a solution to the hybrid execution problem, as defined in Definition 4.10. Thus, we require use of a correct QCP in order to provide guarantees that this problem will be solved.

5.4.5 Disturbance Definitions

The previous definitions and theorems describe controllability and successful execution in terms of trajectories staying within the bounds of flow tubes. This implies bounds on disturbances without explicitly modeling such disturbances.

In this section, we provide an explicit model of disturbances, and provide definitions for different kinds of disturbances. This is useful for two reasons. First, an understanding of how disturbances adversely affect successful execution, as defined in Section 5.4.4, leads to requirements for the hybrid dispatcher algorithm. By understanding the adverse effects, we are able to specify compensating actions that the hybrid dispatcher must take. Second, an understanding of disturbances in terms of the previously developed controllability properties of a QCP motivates these properties, and justifies desirable characteristics of a QCP that the plan compiler is attempting to optimize. For example, the plan compiler maximizes both the controllable temporal range, and the size of the intersection region for all activities, as discussed in Section 5.1.5.

Definition 5.10 (Disturbance to SISO system): A disturbance to an SISO system is an impulse [Wolfram, 2005] that is added to the acceleration input of the SISO system. The impulse has infinite magnitude, 0 duration, and area A. The area defines the magnitude of the impulse in that the integral of the impulse is a step function that changes by A.

Impulses have long been used to model disturbances [Kailath, 1980]. For linear systems, superposition can be used to model any disturbance signal as a sequence of impulses.

Because the disturbance is an acceleration spike, it has an instant effect on an SISO system's trajectory. This is because velocity is the integral of acceleration, so a disturbance spike at time t results in a velocity step at time t. Disturbances are applied to an SISO system as shown in Fig. 5.15.

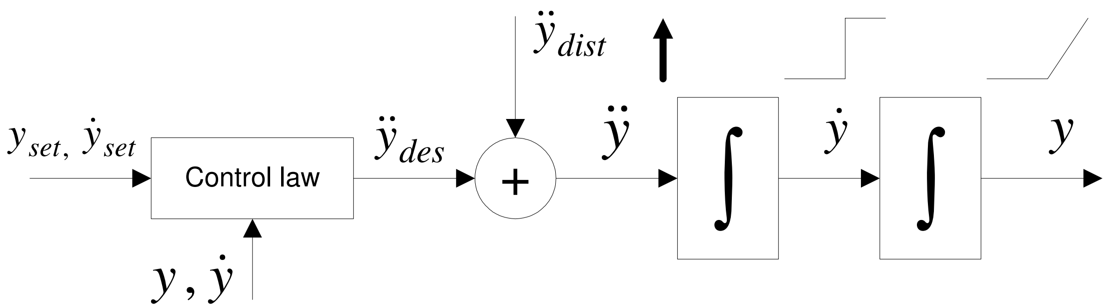


Fig. 5.15 – Application of disturbance spike to SISO system

Due to the linearization of the SISO system, we can model the effects of disturbances using superposition, hence, the disturbance acceleration spike is added to the desired acceleration produced by the control law. For example, if the control law acceleration is 0, and a disturbance acceleration spike at time t_1 has area = 1, then the velocity is increased by 1 at t_1. This is shown in Fig. 5.16.

One way to view this approach to disturbance modeling is that it models disturbances by their effect on trajectories in state space. This is a common technique in control theory [Kailath, 1980]. The deviation of the trajectory from the nominal one represents the cumulative effect of previous disturbances.

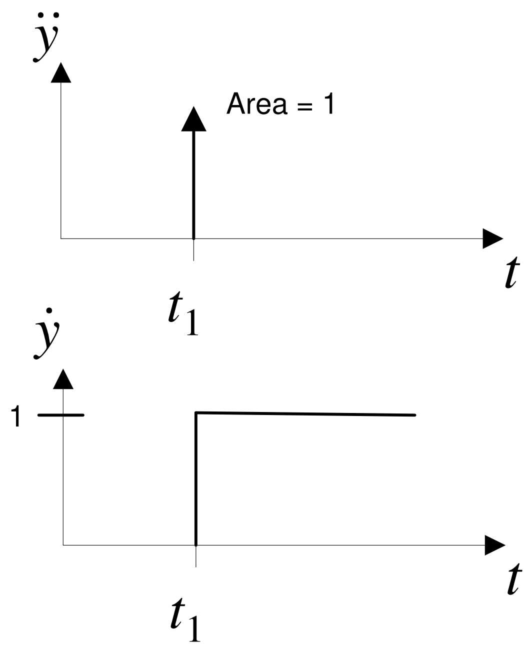


Fig. 5.16 – Effect of acceleration spike on velocity

We are interested in how disturbances affect state with respect to region boundaries. For example, as discussed in Theorem 5.1, one requirement for successful execution of a controllable activity is that the trajectory begins in the activity's initial region. Thus, an important concept is whether a disturbance pushes a trajectory out of such a region.

Definition 5.11 (Disturbance bounded by a region): A disturbance to an SISO trajectory (Def. 4.2) is said to be bounded by a position-velocity region, R, if the trajectory position and velocity after applying the disturbance is inside R.

For example, the disturbance in Fig. 5.14a, at t=1, is bounded by the tube cross section region in Fig. 5.12b. The disturbance in Fig. 5.14b is not.

The concept of a disturbance bounded by a region is crucial for successful execution of activities. Theorem 5.1 states that one requirement for successful execution of a control activity is that the trajectory associated with the activity be within the activity's initial region at the start of the activity's execution. Disturbances prior to the start of such an activity cause the trajectory to deviate from its nominal course. If this deviation is bounded, as defined by Definition 5.11, then the requirement for Theorem 5.1 is satisfied. For this reason, we wish to make initial regions of activities in a QCP as large as possible; maximizing the area of these regions maximizes robustness to disturbances.

So far, we have assumed, in our discussion of successful execution guarantees, that no disturbances occur after the start of a controllable activity. We now consider the case where disturbances do occur during a controllable activity's execution. As before, we model such a disturbance as a deviation in the SISO system's trajectory.

If a disturbance occurs during execution of the activity, there are three possible outcomes. To understand these outcomes, consider the implications of Lemma 5.2 for execution. Lemma 5.2 implies that, at the start of execution of a controllable control activity, CA, the dispatcher is able to arbitrarily select a goal region arrival time within the range $[l(CA), u(CA)]$. In doing this, the dispatcher is choosing one of the tubes in the controllable tube set (Def. 5.7). After activity execution has started, this choice has been made, and the dispatcher is attempting to keep the activity's trajectory within the particular chosen tube. Any disturbance that occurs at this time must be considered with respect to this tube.

For the first possible outcome, the disturbance is small enough that the trajectory stays within its current tube. We call such a disturbance spatially and temporally controllable.

Definition 5.12 (Spatially and temporally controllable disturbance): Given a controllable activity, CA, being executed with goal duration D_{goal}, the corresponding tube is $tube = TUBE(CA, D_{goal})$ (Def. 5.4). The associated SISO system is $s = S(A(CA))$. A disturbance that occurs during execution of this activity is spatially and temporally controllable if it does not cause the position/velocity trajectory for s (Def. 4.2) to be outside of any cross-section of $tube$ (Def. 5.5).

From Lemma 5.1, if a spatially and temporally controllable disturbance occurs, and if there are no further disturbances, the goal is still reached after duration D_{goal}.

The second possible outcome of a disturbance during activity execution is that the trajectory is pushed outside its current tube, and into a different one in the controllable tube set of CA (Def. 5.7). We call such a disturbance spatially controllable.

Definition 5.13 (Spatially controllable disturbance): A disturbance that occurs during execution of a controllable activity, CA, with goal duration D_{goal}, is spatially controllable if it pushes the trajectory out of $TUBE(CA, D_{goal})$, and into another tube, $TUBE(CA, D_{goal} + D_{\Delta})$, where $l(CA) \leq D_{goal} + D_{\Delta} \leq u(CA)$.

The tube $TUBE(CA, D_{goal} + D_{\Delta})$ is still a member of the controllable tube set of CA, so, from Lemma 5.1, if there are no further disturbances, the goal can still be reached, but at duration $D_{goal} + D_{\Delta}$ instead of duration D_{goal}.

The third possible outcome of a disturbance during activity execution is that the trajectory is pushed out of all tubes in the controllable tube set of CA. We call such a disturbance uncontrollable.

Definition 5.15 (Uncontrollable disturbance): A disturbance that occurs during execution of a controllable control activity, CA, is uncontrollable if it pushes the trajectory out of all tubes in the controllable tube set of CA.

For an uncontrollable disturbance, successful execution cannot be guaranteed, even if there are no further disturbances.

A spatially and temporally controllable disturbance can be handled locally, for the activity, by the dispatcher. An uncontrollable disturbance represents a plan failure; the dispatcher must abort and request a new plan. The interesting case here is the spatially controllable disturbance. In this case, the dispatcher is able to make the activity's trajectory reach the goal region, but at a time other than the one originally intended. If, however, it can also appropriately adjust the durations of other activities whose completion must be synchronized with that of the disturbed activity, it will still be able to execute the QCP successfully, as long as there are no further disturbances.

In order to preserve maximum temporal flexibility for this case, it is desirable that the temporal range, $[l, u]$, of control activities, be as large as possible. Thus, an important goal of the plan compiler is to maximize these ranges. As stated previously, in order to maximize robustness to spatially and temporally controllable disturbances, it is also desirable to maximize the initial regions of control activities. As we will see in Chapter 7, the dual goals of maximizing control activity initial regions, and maximizing their temporal ranges, are often at odds, and thus, present a challenge for the plan compiler.

6 Hybrid Dispatcher

The hybrid dispatcher executes a QCP output by the plan compiler, as introduced previously in Sections 1.4.2 and 4.2.2. In this chapter, we review requirements for the dispatcher, discuss our approach to fulfilling these requirements, and develop the dispatcher algorithm. We present results from a number of plan executions in Chapter 9.

6.1 Dispatcher Requirements

To execute a correct QCP for a QSP, the dispatcher must successfully execute each control activity in the QCP. The dispatcher accomplishes this by setting control parameters for each control activity such that the associated trajectory reaches the activity's goal region at an acceptable time, thereby, indirectly scheduling start and finish events so that they are consistent with the temporal constraints of the QCP. This results in a schedule, T, and a trajectory set, Y, that satisfy the QSP, according to Definition 4.7.

To execute a control activity successfully, the dispatcher must guide the trajectory associated with each activity into its goal region within the time specified by the bound on the activity's duration, as shown in Fig. 6.1. The dispatcher guides the trajectory by adjusting control parameters, as specified in Theorem 5.1. Thus, the dispatcher guides the plant indirectly, by adjusting control law parameters in the SISO abstractions, rather than by directly issuing joint torque commands to the biped, as shown in Fig. 1.16. By adjusting control parameters, the dispatcher keeps the trajectory in the flow tube of its current activity, and guides the trajectory from the activity's initial region to its goal region, as shown in Fig. 6.1. Also, by adjusting control parameters, the hybrid dispatcher accelerates or decelerates a trajectory (Fig. 4.13), allowing it to adjust the time that a trajectory is in its goal region. As the dispatcher guides its state trajectory from the initial to the goal region of an activity, the dispatcher must fulfill two key requirements. First, it must ensure that the trajectory reaches the goal region, and second, it must ensure that the trajectory is in the goal region at an acceptable time. Thus, the dispatcher is a time varying control program that attempts to ensure successful execution of the QCP by ongoing recalibration of the decoupled SISO systems, based on the predicted trajectory given the current state and settings.

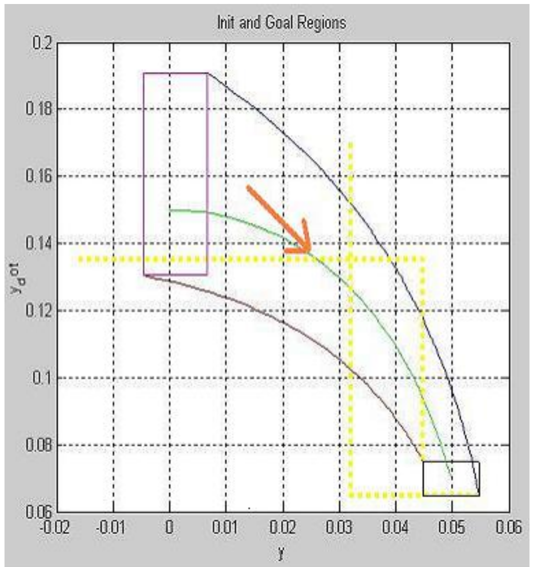


Fig. 6.1 – Dispatcher guides the trajectory through the flow tube from the initial to the goal region.

The ability to control the time that an activity's trajectory is in its goal region is important because completion of the activity may require synchronization with the completion of another activity. Consider, for example the QCP shown in Fig. 6.2. This QCP is a correct QCP for the QSP of Fig. 4.14. Note that the activity and event structure of the QCP is identical to that of the QSP, as required by Definition 5.3. The only difference is that Fig. 4.14 shows forward and lateral components of CM lumped together into common activities, for the sake of simplicity and brevity, whereas in Fig. 6.2, the forward and lateral components are broken out into separate activities, corresponding to separate SISO systems. In Fig. 6.2, many activities end at the same event. For example, the activities CM_Fwd_1, CM_Lat_1, and Right foot ground 1 all end at the event called right toe-off. Similarly, the activities CM_Fwd_2, CM_Lat_2, and Right foot step 1 all end at the event called right heel-strike. When a set of activities all end at the same event, they must all finish at the same time. This implies that, for any activity in such a set to complete successfully, its trajectory must be in its goal region at the same time that the other activities trajectories are in theirs. Therefore, it is important for the dispatcher to be able to control the time that an activity's trajectory is in its goal region.

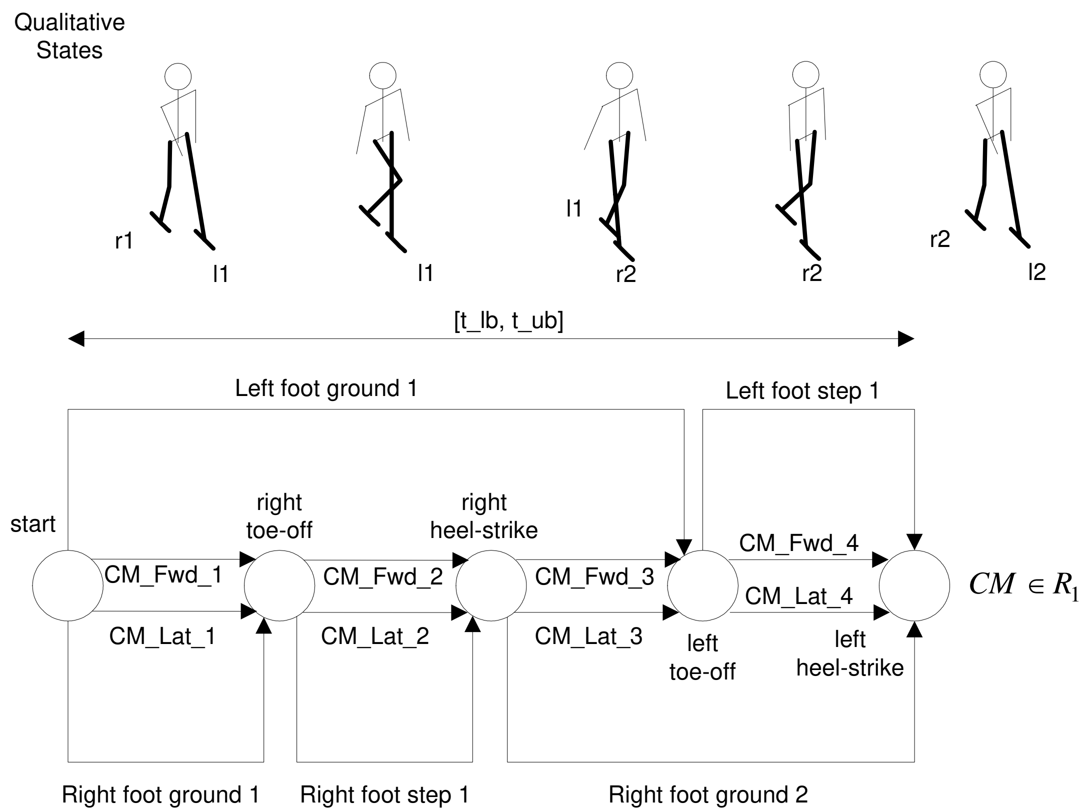


Fig. 6.2 – QCP for QSP of Fig. 4.14. Circles represent events, and horizontal arrows between events represent activities. Activities ending at the same event must be synchronized so that they finish at the same time. For example, the activities CM_Fwd_1, CM_Lat_1, and Right foot ground 1 all end at the event right toe-off. Therefore, these activities must finish at the same time.

Events such as activity completion may be constrained by temporal constraints in the QCP. As explained previously in Section 2.2, when such an event occurs, the effects of this occurrence must be propagated by the dispatcher, via the temporal constraints, to other events. This propagation may result in a tightening of the execution windows of subsequent events. For example, in Fig. 2.10, the occurrence of event B at T = 7 tightens the execution window of event C, due to the distance graph arcs between B and C. Similarly, occurrence of events in Fig. 6.2, such as right toe-off, may result in a tightening of the execution windows of subsequent events through temporal constraints, such as the $[l,u]$ bounds on activities like CM_Fwd_2, CM_Lat_2, and Right foot step 1.

The dispatcher must be able to deal appropriately with unforeseen disturbances that may occur during plan execution. As explained in Section 5.4.5, when a disturbance occurs, there are three possible outcomes. In the first such outcome, where the disturbance is spatially and temporally controllable (Def. 5.12), the disturbance is small enough that the dispatcher does not have to change the scheduled duration of the activity, which was decided at the time the activity begins executing. In this case, the dispatcher may, or may not have to change control parameter settings in order to keep the trajectory on track towards being in the activity's goal region at the scheduled time. In the second outcome, where the disturbance is spatially controllable (Def. 5.13), the disturbance is large enough that a change in the activity's originally scheduled duration is necessary. This may also necessitate a change in the duration of other activities, whose completion must be synchronized with that of the disturbed activity. In the third outcome, the disturbance is so large that no adjustment of control parameters is able to compensate for it. In this case, there is no way to guarantee successful execution of the QCP, even if there are no further disturbances. The dispatcher must recognize this immediately after the disturbance and abort execution, notifying a higher-level control authority that the current plan execution has failed and that a new plan is needed.

Besides executing a QCP according to the requirements stated thus far, the dispatcher must be efficient enough that this plan execution can be accomplished in real time.

6.2 Dispatcher Approach

To fulfill the above-stated requirements, the dispatcher performs three key functions in executing a control activity: initialization, monitoring, and transition. Initialization is performed at the start of an activity's execution, monitoring is performed continuously during the activity's execution, and transition is performed at the finish of the activity's execution.

6.2.1 Initialization

The initialization function is run at the beginning of each execution of a control activity. Assuming that all trajectories begin in the initial regions of their control activity, the dispatcher chooses a goal duration for the control activity that is consistent with its execution window, and sets control parameters for the control activity such that the state trajectory is predicted to be in the activity's goal region at the goal time.

For example, consider initialization for the QCP shown in Fig. 6.2. The first event in this QCP is the event called "start". Suppose the dispatcher schedules this event to occur at time 0, when execution of the QCP begins. Four activities, CM_Fwd_1, CM_Lat_1, Left foot ground 1, and Right foot ground 1 share this event as their start events. Thus, the dispatcher performs the activity initialization function for each of these four activities upon start of execution of the QCP, as shown in Fig. 6.3. For each of these activities, the initialization function chooses a goal duration, and control parameter settings.

The control parameter settings are such that, if there are no further disturbances, they will not have to be adjusted, as specified in Theorem 5.1; the control parameters are set such that the state trajectory is predicted to be in the goal region at the goal time. To check that a trajectory will be in its goal region at the desired time, the dispatcher uses an efficient prediction algorithm, as will be explained shortly.

6.2.2 Monitoring

After initializing an activity, the dispatcher begins monitoring execution of that activity. To monitor execution, the dispatcher continually checks whether the state trajectory remains in its flow tube, and hence, is on track to be in the goal region at the goal time. If this is not the case, it attempts to correct this situation by adjusting control parameters. If this is unsuccessful, the dispatcher aborts plan execution and requests a new plan from a higher control authority. As part of the monitoring function, the dispatcher also continually checks whether a control activity's completion conditions are satisfied. Thus, it checks whether the state trajectory is in the activity's goal region, and whether the state trajectories of other activities whose completion must be synchronized are in their activity's goal regions. If all completion conditions for a control activity are satisfied, the dispatcher switches to the transition function.

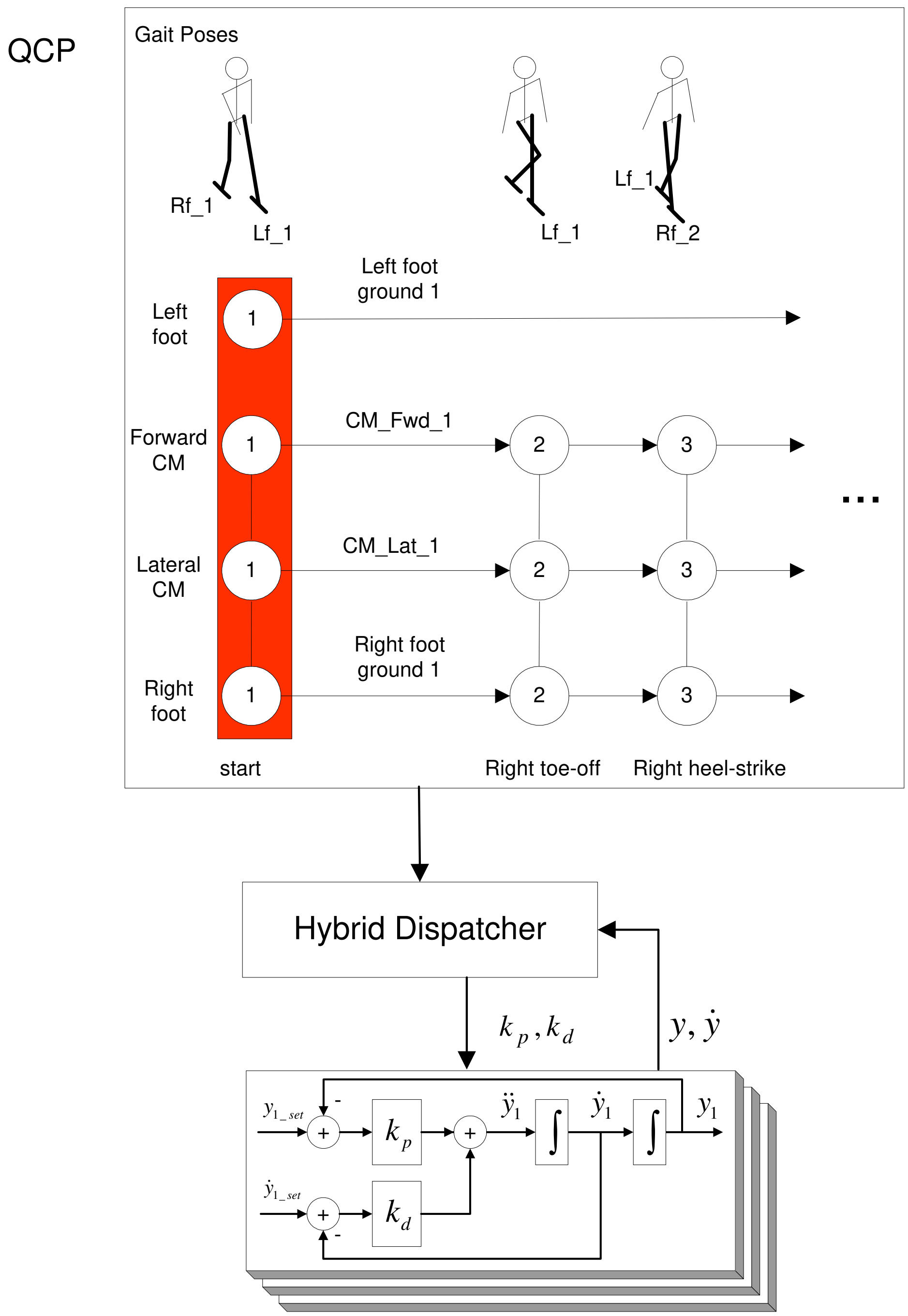


Fig. 6.3 – Initialization of a set of activities beginning at the start event. The dispatcher chooses optimal settings for the control parameter gains, within the bounds specified by the QCP.

For example, after the initialization shown in Fig. 6.3, the dispatcher begins monitoring the four activities, CM_Fwd_1, CM_Lat_1, Left foot ground 1, and Right foot ground 1, as shown in Fig. 6.4a.

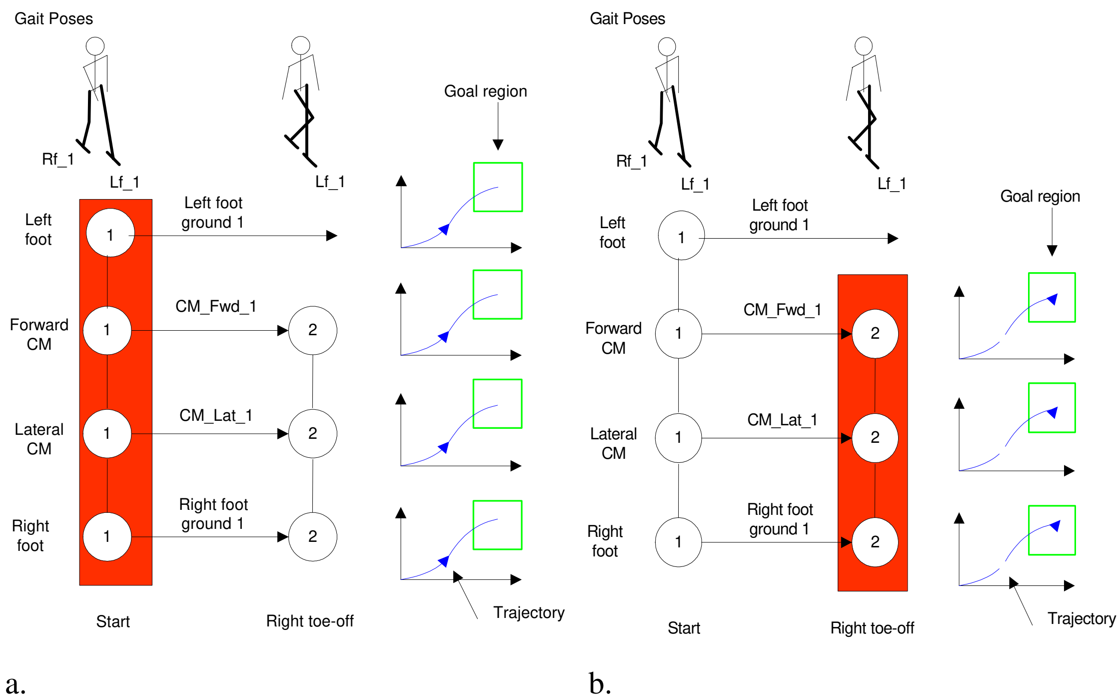


Fig. 6.4 – a. The dispatcher monitors progress of each SISO system trajectory towards its goal. b. It transitions to successor activities when all trajectories for activities that must be synchronized are in their respective goal regions.

Recall, from our discussion in Section 5.1.6, that a key requirement for monitoring is that the dispatcher be able to quickly determine whether a trajectory is in its flow tube, and hence, whether it will be in its goal region at the desired time. To accomplish this, the dispatcher uses the same efficient prediction algorithm as the one used in initialization. If the prediction shows that a trajectory will not achieve its goal, then the prediction algorithm is combined with an efficient execution-time search in order to determine whether a control parameter adjustment can be made that will achieve the goal. The prediction and search algorithms will be explained in more detail shortly.

In order to check for activity completion, the dispatcher forms a set of all currently executing activities that must finish at the next scheduled event. In Fig. 6.4a, the next event after start is Right toe-off. Of the four activities that begin at the start event, three (CM_Fwd_1, CM_Lat_1, and Right foot ground 1) finish at the Right toe-off event. At the time when the trajectories for all three of these activities are in their goal regions, the dispatcher transitions all three, as shown in Fig. 6.4b.

6.2.3 Transition

If the control activity has a successor, the transition function invokes the initialization function for this new activity. As part of this transition, the dispatcher notes the time of the transition event and propagates this through the temporal constraints. In the example shown in Fig. 6.4b, the Right toe-off event is marked as having occurred at the transition time, and the consequences of this event occurring at this time are propagated, via temporal constraints, in order to appropriately tighten execution windows of future events.

6.3 Hybrid Dispatcher Algorithm

The previous sections discussed requirements for the dispatcher, and our approach to addressing these requirements. In this section, we present pseudocode for the dispatcher algorithm, and describe its execution in detail for part of an example walking task.

The top-level dispatcher function, Dispatch, is shown in Fig. 6.5. This function takes a QCP as its argument and begins to execute it, beginning with the first event in the QCP. Dispatch calls the function Initialize (line 2), to perform one-time initialization for execution of the QCP. It then enters a loop (lines 3 – 7), calling DispatchEvent to execute events. DispatchEvent performs the three key functions for activity execution (initialize, monitor, and execute), described in Section 6.2.

The loop in Dispatch exits normally after the last event has been executed. The function returns true in this case in order to indicate successful execution of the plan. Non-local exits out of this loop are also possible due to aborts caused by plan failure. Such aborts are used if a sub-function detects that plan execution has become infeasible for some reason. These aborts are analogous to *throw* statements in Java.

```
successful? = Dispatch(qcp)
    // Dispatch takes a qualitative control plan, qcp, and begins to execute it,
    // beginning with the first event in qcp.  After initialization, it enters
    // a loop to dispatch events.  When the last event has occurred, execution
    // terminates.  Note that sub-functions may cause an abort, signaling error,
    // if they determine that plan execution has failed.
    start_event = GetFirstEvent(qcp);
    Initialize(qcp, start_event);
    current_event = GetNextEvent(qcp, start_event);
    while (current_event != NULL)
        // Execute until the next event.
        current_event = DispatchEvent(qcp, current_event);
    return true;    // to indicate plan success
```

Fig. 6.5 – Pseudocode for Dispatch.

6.3.1 Dispatcher Initialization and Execution Window Propagation

Pseudocode for the function Initialize is shown in Fig. 6.6. This function initializes execution windows for all events using an approach similar to the one described in Chapter 2 [Muscettola, 1998]. It sets the execution window for the first event, based on the current time (line 2). It then propagates this window to subsequent events using the function PropagateExecutionWindow (line 3).

PropagateExecutionWindow, shown in Fig. 6.6, takes a QCP and an event as its arguments. It begins (lines 1 – 7) by initializing the flags, LowerBoundUpdated and UpperBoundUpdated, for each event. These flags are used in the function to indicate whether the execution window for an event has been updated (see Fig. 6.7). PropagateExecutionWindow then initializes a queue of events whose execution windows are to be propagated (lines 9 and 10). It then iterates while the queue is not empty. For each iteration, it first pops an event, event1, off the queue. It then iterates over the positive outgoing arcs of event1 in the minimum dispatchable graph of the QCP. For each such arc, it updates the upper bound of execution windows of events at the output of these arcs, and adds these events to the queue for further propagation (lines 14 - 19). A similar iteration is performed for negative incoming arcs (lines 20 - 25).

```
Initialize(qcp, start_event)
    // This initializes the execution window for each event..
    start_time = GetCurrentTime();
    SetExecutionWindow(start_event, start_time, start_time);
    PropagateExecutionWindow(qcp, start_event);
```

```
PropagateExecutionWindow(qcp, event)
    //   PropagateExecutionWindow efficiently propagates changes in
    //  the execution window of event to other events, using the minimum
    // dispatchable graph of the qcp.

    for each Event, event1, in qcp {
       // Indicate that bounds of event1 have to be updated.
        ClearLowerBoundUpdated(event1);
        ClearUpperBoundUpdated(event1); }
    // Indicate that bounds for event have already been updated.
    SetLowerBoundUpdated(event);
    SetUpperBoundUpdated(event);

    queue = CreateEmptyQueue();  // Create queue of events to be propagated.
    AddElementToQueue(queue, event);
    while (not(QueueEmpty(queue))) {
      event1 = Pop(queue);
      l, u = GetExecutionWindow(event1);
      for each positive outgoing arc A1 of event1 {  // Propagate upper bounds
        event2 = GetOutputEvent(A1);
        if (not (UpperBoundUpdated(event2)) {
          SetExecutionWindowU(event2, u + dist(A1));
          SetUpperBoundUpdated(event2);
          AddElementToQueue(queue, event2); }}
      for each negative incoming arc A1 of event1 {  // Propagate lower bounds
        event2 = GetInputEvent(A1);
        if (not (LowerBoundUpdated(event2)) {
          SetExecutionWindowL(event2, l – dist(A1));
          SetLowerBoundUpdated(event2);
          AddElementToQueue(queue, event2); }}
```

Fig. 6.6 – Pseudocode for Initialize and PropagateExecutionWindow.

ExecutionWindow – a tuple, [l, u], indicating the valid range of event times

LowerBoundUpdated – a flag indicating that the lower bound of the event's execution window has been updated (used in PropagateExecutionWindow).

UpperBoundUpdated – a flag indicating that the upper bound of the event's execution window has been updated (used in PropagateExecutionWindow).

Fig. 6.7 - Runtime data structures associated with events of a QCP. These are implemented using a suitable hash table mechanism that associates the data structure with the event, and provides for efficient access.

For example, Fig. 6.8a shows the minimum dispatchable graph corresponding to the QCP shown in Fig. 6.2. Arc distances are computed by the plan compiler based on temporal constraints in the QCP, as described in Chapter 7. These include the temporal constraints specified explicitly in the QSP, as well as the temporal constraints due to dynamic limitations of the activities, as represented by activity flow tube sets (see Definition 5.7 and Lemma 5.2). Fig. 6.8b shows the event execution windows after Initialize has run, assuming start_time (line 1 of Initialize) is 0.

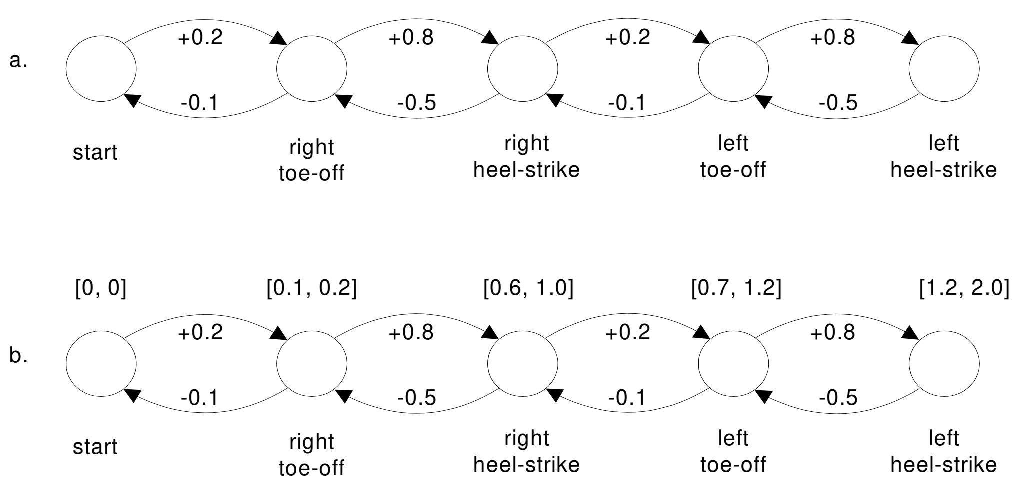


Fig. 6.8 – a. Minimum dispatchable graph for QCP of Fig. 6.2. b. Event execution windows after setting start event time to 0 and propagating.

Supporting functions for PropagateExecutionWindow are shown in Fig. 6.9.

CreateEmptyQueue() // Create an empty queue.
AddElementToQueue(queue, element) // Add element to end of queue.
element = Pop(queue) // Pop first element off queue.
boolean = QueueEmpty(queue) // Returns true if queue is empty, false otherwise.
l, u = GetExecutionWindow(event); // Get lower and upper bounds of window.
SetExecutionWindow(event, l, u); // Set lower and upper bounds of window.
SetExecutionWindowL(event, l); // Set lower bound of window.
SetExecutionWindowU(event, u); // Set upper bound of window.
event = GetOutputEvent(arc); // Get output event of arc of dispatchable graph.
event = GetInputEvent(arc); // Get input event of arc of dispatchable graph.

Fig. 6.9 – Support functions for PropagateExecutionWindow.

6.3.2 Dispatch Event and Initialize Event

The function DispatchEvent, shown in Fig. 6.10, is the main function for executing control activities ending in a common event. DispatchEvent takes a QCP, and an event as its arguments. For example, for the QCP shown in Fig. 6.2, the first event is the event called right toe-off, so DispatchEvent is called with this event first.

DispatchEvent begins by calling InitializeEvent, also shown in Fig. 6.10, which performs the initialization discussed in Section 6.2; it computes a goal event time, and associated control parameter settings such that trajectories for each activity ending at the event will be in their goal regions at the goal transition time, if there are no further disturbances. This computation is performed by the function SetControl, which is called from line 1 of InitializeEvent.

After calling InitializeEvent, DispatchEvent enters a loop (lines 2 - 4) in which it continually calls Monitor, which performs the monitor function discussed previously. Monitor returns a flag indicating whether the system is ready to transition to the next event. When this flag is true, execution breaks out of the monitor loop and the function

Transition is called (line 5 of DispatchEvent). For example, after InitializeEvent is called for the event right toe-off, Monitor is called with this event. When the goal regions for the activities CM_Fwd_1 and CM_Lat_1 are reached, Monitor returns true for this event.

```
next_event = DispatchEvent(qcp, current_event)
    // DispatchEvent comprises the three main dispatching functions:
    // initialize, monitor, and transition.
    1.  goal_event_time = InitializeEvent(qcp, current_event);
    2.  while (not make_transition)
    3.      make_transition, goal_event_time
    4.              = Monitor(qcp, current_event, goal_event_time);
    5.  return (next_event = Transition(qcp, current_event));
```

```
goal_event_time = InitializeEvent(qcp, current_event)
    // InitializeEvent determines a goal time for current_event, and sets control
    // parameters for activities that end at current_event.
    1.  return(goal_event_time = SetControl(qcp, current_event));
```

Fig. 6.10 – Pseudocode for DispatchEvent and InitializeEvent.

6.3.3 SetControl

Pseudocode for SetControl is shown in Fig. 6.11. This function takes a QCP and an event as arguments, computes control parameter settings for all activities ending at the event, and computes a goal time for the event's occurrence. In order to do this, SetControl begins by looking up the execution window, [l,u], for the event (line 1). It then begins a search for a feasible time for the occurrence of the event. This search begins with goal_event_time at the midpoint between l and u, and then proceeds upwards, by increments of delta_t until u is reached (lines 1 - 3). Delta_t is chosen to be an appropriately small increment. A value of 0.05 seconds works well for the bipedal locomotion application. We use the heuristic of beginning the search at the midpoint between l and u since this affords the greatest slack between the bounds.

For each iteration, the algorithm calls SetControlForEventTime with goal_event_time as the argument (line 5). If SetControlForEventTime returns feasible for any of these iterations, the algorithm returns the corresponding goal_event_time immediately, since a feasible solution with goal_event_time as the predicted time of the event has been found. If none of these attempts are successful, the algorithm tries a second iteration, setting goal_event_time to the midpoint of the execution window minus delta_t, and then iterating downwards to 1 (lines 7 – 9). If none of these attempts are successful, the algorithm aborts, indicating plan execution failure. This abort is thrown completely out of the Dispatch function, and must be caught by a higher-level control authority that is capable of issuing a new plan.

The function SetControlForEventTime, shown in Fig. 6.11, is called by SetControl, and takes a QCP, an event, and goal_event_time as an argument, and iterates over each executable activity that ends at the event. For each iteration, it calls SetControlForActivity with the activity and goal_event_time as arguments. SetControlForActivity attempts to find control parameters such that the trajectory for the activity is in the goal region at goal_event_time. If SetControlForActivity fails to do this for any activity, SetControlForEventTime returns false. Otherwise, it returns true.

Continuing the previous example, SetControl calls SetControlForEventTime with the right toe-off event, and with candidate times that are within the execution window for the right toe-off event. This execution window is [0.1, 0.2], as shown in Fig. 6.8b. SetControlForEventTime then calls SetControlForActivity for the activities CM_Fwd_1, and CM_Lat_1, in order to find control parameters such that the trajectories for these activities are in their respective goal regions at the same time.

Note that as long as all trajectories in the set of activities that end at event begin in the initial regions of their associated activities, SetControl is guaranteed to find a goal event time and associated control parameters. This is guaranteed due to the fact that the QCP is controllable, according to Definition 5.8.

```
goal_event_time = SetControl(qcp, current_event)
    // SetControl searches for a feasible time for occurrence of current_event.
    //  This time must be within the execution window.  The search is performed
    // in increments of delta_t, a globally defined parameter.
    1.   l, u = GetExecutionWindow(current_event);
    2.   mid = (l + u) / 2;
    3.   for goal_event_time = mid to u by delta_t {
    4.      // Search from midpoint to upper limit
    5.     feasible? = SetControlForEventTime(qcp, current_event, goal_event_time);
    6.     if (feasible?) return goal_event_time; }   // If feasible, return this time.
    7.   for goal_event_time = mid – delta_t downto l by delta_t {
    8.      // Search from midpoint to lower limit
    9.      feasible? = SetControlForEventTime(qcp, current_event, goal_event_time);
    10.   if (feasible?) return goal_event_time; }   // If feasible, return this time.
    11. abort "plan execution failure, unable to compute control parameters for
         execution window"
```

```
feasible? = SetControlForEventTime(qcp, current_event, goal_event_time)
    // SetControlForEventTime searches for feasible control parameter settings
    // for all activities ending at current_event.
    1.   activity_set = GetInputActivities(qcp, current_event);
    2.   for each ControlActivity, a1, in activity_set {
    3.      activity_feasible? = SetControlForActivity(a1, goal_event_time);
    4.      if (not activity_feasible?) return false; }
    5.   return true;
```

Fig. 6.11 – Pseudocode for SetControl and SetControlForEventTime.

Pseudocode for SetControlForActivity is shown in Fig. 6.12. This function takes a control activity and goal_event_time as its arguments. It first retrieves the SISO system, s1, associated with the activity (line 1), and then calls FindControlParams to compute appropriate control parameters for the activity. If this computation is successful, it applies the computed control parameters to s1 and returns true (lines 4 and 5). Otherwise, it returns false (line 7).

```
feasible? = SetControlForActivity(a1, goal_event_time)
    // SetControlForActivity searches for feasible control parameter settings
    // for a control activity so that it reaches its goal at goal_event_time.
    s1 = GetSISO(a1);
    feasible?, params = FindControlParameters(a1, s1, goal_event_time);
    if (feasible?) {
      ApplyControlParams(params, s1);
      return true; }
    else
      return false;
```

Fig. 6.12 – Pseudocode for SetControlForActivity.

FindControlParameters, shown in Fig. 6.13, takes an activity, an SISO system and goal_event_time, and computes control parameters for the activity such that the SISO system's trajectory ends in the activity's goal region at goal_event_time. FindControlParams begins by retrieving the activity's goal region, the current time, and the current state of the SISO system (lines 1 – 3). It then calls FormulateControlQP, also shown in Fig. 6.13, to formulate a quadratic programming problem for computing the control parameters. This quadratic program is solved by calling SolveQP, a quadratic program solver based on the Matlab function quadprog [Matlab, b.].

As shown in Fig. 6.13, the parameters being optimized in the QP formulation are the predicted state of the trajectory at goal_event_time, and the control parameters. The control parameters are of the form $\langle y_{set}, \dot{y}_{set}, k_p, k_d \rangle$ (see Def. 4.1). The equality constraint is Eq. 4.4, the analytic solution to a linear second-order differential equation, which relates future state to current state. The inequality constraints require the predicted state to be in the goal region, and the cost function biases this predicted state towards the center of the goal region.

```
feasible?, params = FindControlParameters(a1, s1, goal_event_time)
    // FindControlParameters formulates a quadratic program and solves it to find
    // control parameters that the activity's trajectory reaches the goal region at
    // goal_event_time.
    1. Rgoal = a1.Rgoal;   // Get activity goal region
    2. current_time = GetCurrentTime();
    3. y, y' = GetCurrentState(s1);
    4. qp_formulation =
    5.    FormulateControlQP(Rgoal, y, y', current_time, goal_event_time);
    6. return (feasible?, params = SolveQP(qp_formulation);
```

qp_formulation = FormulateControlQP($R_{goal}, y, \dot{y}, t_s, t_f$)

// This generates the following formulation

Parameters to optimize: y_{pred} , $\dot{y}_{pred}$, $params$

Equality constraints:

$$y_{pred} = f_1(t_s, t_f, y, \dot{y}, params)$$
$$\dot{y}_{pred} = f_2(t_s, t_f, y, \dot{y}, params)$$

(from Eq. 4.4)

Inequality constraints:

$$y_{\min}(R_{goal}) \leq y_{pred} \leq y_{\max}(R_{goal})$$
$$\dot{y}_{\min}(R_{goal}) \leq \dot{y}_{pred} \leq \dot{y}_{\max}(R_{goal})$$

(trajectory prediction must be within goal region)

Cost function

$$y_{goal} = (y_{\min}(R_{goal}) + y_{\max}(R_{goal}))/2$$
$$\dot{y}_{goal} = (\dot{y}_{\min}(R_{goal}) + \dot{y}_{\max}(R_{goal}))/2$$
$$cost = (y_{goal} - y_{pred})^2 + (\dot{y}_{goal} - \dot{y}_{pred})^2$$

Fig. 6.13 – Pseudo code and formulation for FindControlParameters.

6.3.4 Monitor

Whereas InitializeEvent is called at the start of DispatchEvent, and Transition is called at the finish, the function Monitor is executed continually, through the loop in DispatchEvent. Monitor performs the monitoring function described in Section 6.2. Pseudocode for Monitor is shown in Fig. 6.14.

The function takes a QCP, an event, and goal_event_time as its arguments. The algorithm begins by setting two flags: all_in_goal, and all_on_target to true (lines 1 and 2). The flag all_in_goal indicates whether all trajectories are in their goal region at the right time so that a transition may occur. The flag all_on_target indicates whether all trajectories are on target to get to their goal regions at the right time. The algorithm then iterates over each control activity that ends at the event, and for each one, calls CheckProgress (lines 3 – 6). CheckProgress returns two flags: in_goal_region?, and on_target?. The flag in_goal_region? indicates whether the activity's trajectory is in the goal region. The flag on_target? indicates that the activity's trajectory is on track to being in the goal region at goal_event_time. If in_goal_region? is false, then all_in_goal is set to false (lines 7 and 8), indicating that a transition cannot be made yet. If not_on_target? is true, then all_on_target is false, and the algorithm breaks out of the loop, by iterating over control activities of the current event (lines 9 – 12). After the loop finishes, the algorithm checks the flags all_in_goal and all_on_target. If both are false, then a control parameter adjustment is necessary, and SetControl is called, just as in InitializeEpoch, to select a new goal transition time and to compute appropriate control parameters (lines 14 and 15). If SetControl fails to make such an adjustment, it aborts and the algorithm exits, indicating plan execution failure. If SetControl succeeds, Monitor concludes by calling CheckTransition. CheckTransition returns the flag make_transition, which is true if all_in_goal is true, and if it is OK to make a transition at the current time. This flag is then returned as the value of Monitor.

```
make_transition, goal_event_time = Monitor(qcp, current_event, goal_event_time)
    // Monitor tracks the progress of activities that end with current_event.
    // It may make adjustments to control parameters, and checks whether transition
    // conditions are satisfied.
    all_in_goal = true;    // Flag that indicates all activities are in their goal.
    all_on_target = true;   // Flag that indicates all activities are on track to goal.
    activity_set = GetInputActivities(qcp, current_event);
    for each ControlActivity, a1, in activity_set {
        in_goal_region?, on_target? =
                    CheckProgress(a1, goal_event_time);
        if (not in_goal_region?) {
            all_in_goal = false;
            if (not on_target?) {
                // If any activity is not on target, break and attempt adjust.
                all_on_target = false;
                break;
            }
        }
    }
    if ((not all_in_goal) and (not all_on_target)) {
            goal_event_time = SetControl(qcp, current_event); }
    make_transition = CheckTransition(all_in_goal, goal_event_time, current_event);
    return make_transition, goal_event_time;
```

Fig. 6.14 – Pseudocode for Monitor.

CheckProgress takes a control activity and goal_event_time as its argument and returns two Boolean values: in_goal?, and on_target?, as shown in the pseudocode of Fig. 6.15. It first checks whether the trajectory associated with the activity is currently within the goal region rectangle, by calling InGoal (lines 1 - 3). If it is, it returns with in_goal? true. Next, if the trajectory is not currently in the goal region, the function computes a prediction for the trajectory state at goal_event_time, by calling PredictTrajectory (line 10). If this state is within the goal region rectangle, then the on_target? return value will be true, otherwise, false.

```
in_goal?, on_target? = CheckProgress(a1, goal_event_time)
    // CheckProgress first checks if the trajectory of activity a1 is in the goal region.
    // If not, it checks whether it is on track to reach the goal region at goal_event_time.
    s1 = GetSISO(a1);   // Get the SISO system for a1.
    y, y' = GetCurrentState(s1);
    in_goal? = InGoal(a1, y, y');
    if (in_goal?) {
      on_target? = true;
      return(in_goal?, on_target?); }
    current_time = GetCurrentTime();

    // Predict where the trajectory will be at goal_event_time.
    y_pred, y'_pred = PredictTrajectory(a1.Rgoal, y, y',
                                            current_time, goal_event_time);
    on_target? = InGoal(a1, y_pred, y'_pred);
    return(in_goal?, on_target?);;
```

```
in_goal? = InGoal(a1, y, y')
    // InGoal checks whether the y, y' trajectory state is in activity
    // a1's goal region.
    Rgoal = a1.Rgoal;   // Get activity goal region
    if ((y_min(Rgoal) <= y <= y_max(Rgoal)) and
        (y'_min(Rgoal) <= y' <= y'_max(Rgoal)))
      return true;
    else
      return false;
```

y_{pred}, $\dot{y}_{pred}$ = PredictTrajectory($R_{goal}, y, \dot{y}, t_s, t_f$)

// PredictTrajectory predicts future trajectory from current state using Eq. 4.4.

$y_{pred} = f_1(t_s, t_f, y, \dot{y}, params)$ // From Eq. 4.4.

$\dot{y}_{pred} = f_2(t_s, t_f, y, \dot{y}, params)$

return(y_{pred} , $\dot{y}_{pred}$);

Fig. 6.15 – Pseudocode for CheckProgress and sub-functions.

Estimation of the state at goal_event_time is by the function PredictTrajectory, shown in Fig. 6.15. As in FormulateControlQP (Fig. 6.13), PredictTrajectory uses Eq. 4.4 to provide an analytic solution for position and velocity as a function of time.

CheckTransition, shown in Fig. 6.16, takes a flag, all_in_goal, a time, and an event as its arguments, and returns the flag make_transition?, a boolean value indicating whether it is OK to transition to the next event. Pseudocode for this function is shown in Fig. 6.16. If all_in_goal is false, then CheckTransition returns false immediately. Otherwise, it checks whether the time is within the event's execution window. If it is, it returns true, otherwise, false (lines 2 – 6).

```
make_transition? = CheckTransition(all_in_goal, current_time, current_event)
    // CheckTransition checks the all_in_goal flag, and if this is true, returns true
    // if the current time is within the execution window of the current_event.
    1.   if (not all_in_goal) return false;
    2.  l, u = GetExecutionWindow(current_event);
    3.  if (l <= current_time <= u)
    4.    return true;  // in execution window
    5.  else
    6.    return false;
```

Fig. 6.16 – Pseudocode for CheckTransition.

6.3.5 Transition

Transition is called at the end of DispatchEvent, in order to complete execution of the event, and transition to the next one. Pseudocode for this function is shown in Fig. 6.17. Transition takes a QCP and an event as arguments. It sets the execution window for the event to the current time and then propagates this window (lines 1 - 3). It then retrieves the next event from the QCP.

Note that, because events are fully ordered in the QCP, it is always possible to retrieve the next event by following the positive outgoing arcs of the minimum dispatchable graph from the current event, and selecting the one with the smallest distance. The function GetNextEvent uses this approach; it returns either the next event, or NULL, if there are no further events.

```
next_event = Transition(qcp, current_event)
   // Transition switches execution to the next event after current_event.
   1. current_time = GetCurrentTime();
   2. SetExecutionWindow(current_event, current_time, current_time);
   3. PropagateExecutionWindow(qcp, current_event);
   4. return(next_event = GetNextEvent(qcp, current_event));
```

Fig. 6.17 – Pseudocode for InitializeEvent, Monitor, and Transition.

Fig. 6.18 shows supporting functions used in InitializeEvent, Monitor, and Transition. These include functions for retrieving arcs and events from the dispatchable graph, for obtaining the current time, and for obtaining state of an SISO system.

```
current_time = GetCurrentTime()   // Get the current time.
y, y' = GetCurrentState(s1)   // Get the current position and velocity of an SISO system.
event = GetFirstEvent(qcp);  // Get the first event in the QCP.
event = GetNextEvent(qcp, current_event);   // Get the next event after the current one.
activity_set = GetInputActivities(qcp, event);  // Get activities that end at event.
activity_set = GetOutputActivities(qcp, event);  // Get activities that start at event.
```

Fig. 6.18 – Support functions.

6.3.6 Example Execution

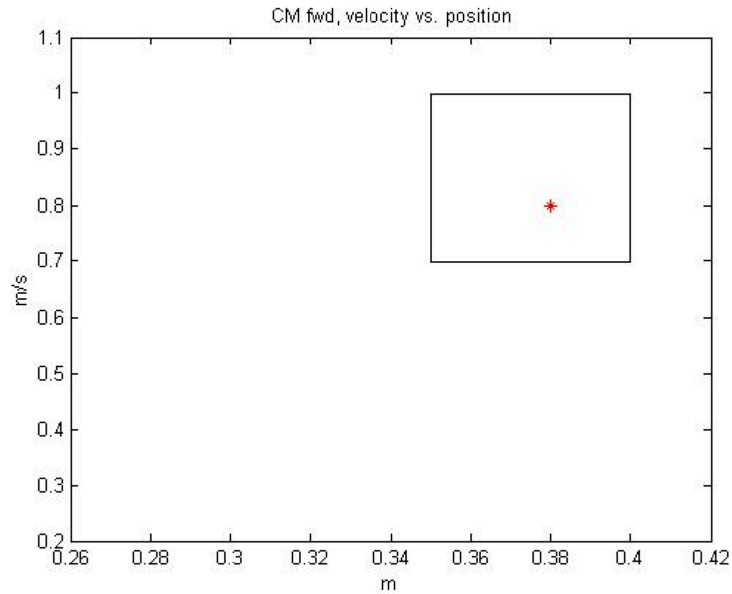
In order to provide a better understanding of the pseudocode presented in the previous sections, we now describe an example execution of a portion of the QCP shown in Fig. 6.2. As described previously, the top-level function, Dispatch (Fig. 6.5) obtains the first event, start, in the QCP, and calls Initialize with this event. Initialize sets the execution window of this event to the current time, and propagates the effects of this using PropagateExecutionWindow (Fig. 6.6). If we assume that the current time at Initialize is 0, then the execution windows for all events are as shown in Fig. 6.8b after this propagation.

After Initialize, Dispatch obtains the next event, right toe-off, and enters the loop that calls DispatchEvent (lines 3 – 6, Fig. 6.5). For the first iteration of this loop, DispatchEvent is called with the event right toe-off. DispatchEvent (Fig. 6.10) calls InitializeEvent, with right toe-off, in order to choose a goal time for the event, and to set control parameters that achieve this time (line 1). InitializeEvent calls SetControl (Fig. 6.11), with right toe-off as the event.

SetControl searches for a feasible goal time by calling SetControlForEventTime with the right toe-off event, and with candidate times that are within the execution window for the right toe-off event (lines 3-10). For the right toe-off event, this execution window is [0.1, 0.2], as shown in Fig. 6.8b.

SetControlForEventTime checks whether a candidate time is feasible, by forming the set of activities that end at the event (line 1), and calling SetControlForActivity for each one (lines 2 – 4). If all activities are feasible for this time, SetControlForEventTime returns true. The set of activities that end at the right toe-off event includes the activities CM_Fwd_1 and CM_Lat_1. We will focus our example execution discussion on these two because the other two activities that end at the right toe-off event (left foot ground 1 and right foot ground 1) are not very interesting; for these, the feet are motionless, and on the ground. For the activities CM_Fwd_1 and CM_Lat_1, SetControlForEventTime finds a feasible time of 0.15 for the goal event time. For this time, the predicted states of the SISO systems associated with these activities, as computed in FindControlParameters, are as shown in Fig. 6.19. Note that these predictions are within the required goal regions. This implies that the control parameters, computed by FindControlParameters, will achieve all required goal regions at the goal event time, given that there are no further disturbances.

After InitializeEvent, DispatchEvent begins calling Monitor continuously, with right toe-off event as the event, and goal event time of 0.15 (lines 2 – 4). If there are no significant disturbances, CheckProgress will always return true for on_target (lines 5 and 6). When the goal regions for CM_Fwd_1 and CM_Lat_1 are reached, CheckProgress will return true for in_goal? for all activities. Trajectories for CM_Fwd_1 and CM_Lat_1 are shown in Fig. 6.20.



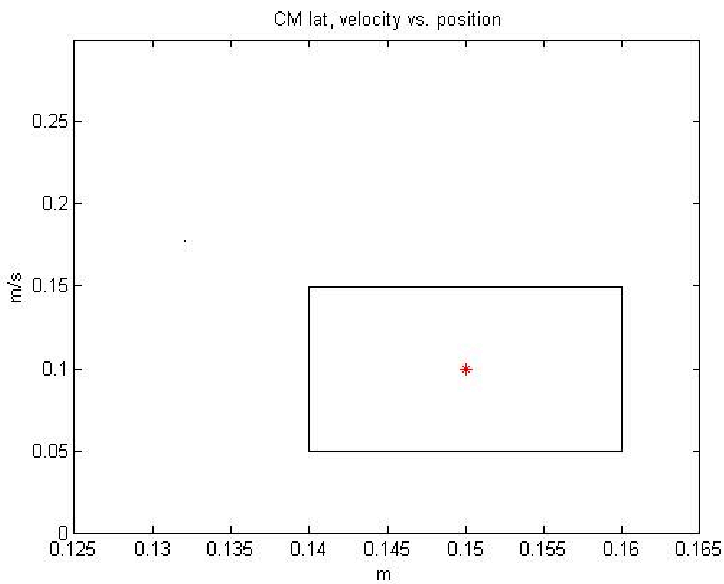


Fig. 6.19 – SISO system state predictions for CM forward and lateral components, for activities CM_Fwd_1 and CM_Lat_1.

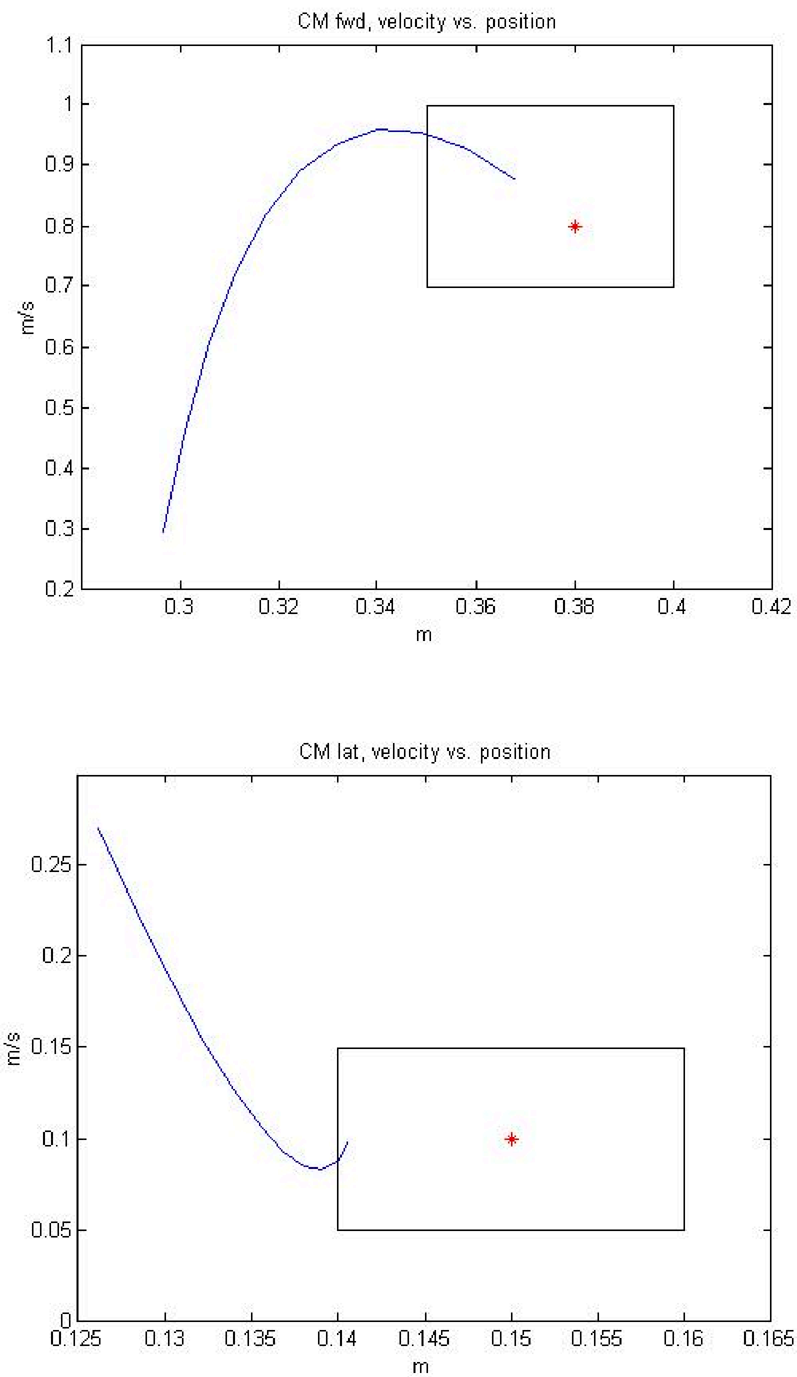


Fig. 6.20 – SISO system state trajectories for CM forward and lateral components, for activities CM_Fwd_1 and CM_Lat_1. Original predictions (see also Fig. 6.19) are shown with a red star.

When all_in_goal is true in Monitor, it calls CheckTransition (line 18) to make sure that the current time is within the event's execution window. Note that the current time need not exactly match the goal time for the event computed by InitializeEvent. Suppose that the current time when all_in_goal becomes true is 0.145 seconds. This is slightly less than the goal time of 0.15, but is within the execution window of [0.1, 0.2] (see Fig. 6.8b). Therefore CheckTransition returns true, and Monitor returns true for MakeTransition. DispatchEvent then calls Transition with right toe-off as the event (line 5).

As with Initialize, Transition sets the execution window of the event (right toe-off in this case) to the current time, and then propagates the effect of this on future propagation windows. Since the time is now 0.145, the execution windows change from those shown in Fig. 6.8b, to the ones shown in Fig. 6.21.

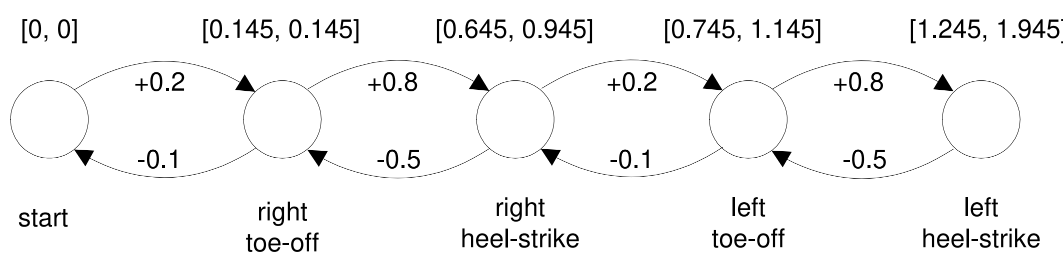


Fig. 6.21 – a. Execution windows after right toe-off event occurs at 0.145 seconds.

The last thing Transition does is to retrieve the next event after right toe-off. This event is right heel-strike, and is returned to DispatchEvent, which returns it to Dispatch. Dispatch then performs the next iteration of the event loop (lines 4 – 6), and calls DispatchEvent with right heel-strike.

Execution for this event, if disturbances are not significant, is similar to that of the previous event. Predictions and actual CM trajectories are shown in Fig. 6.22. These correspond to the activities CM_Fwd_2 and CM_Lat_2 in Fig. 6.2. These activities end at right heel-strike. A third activity that ends at right heel-strike is right foot step 1. This activity represents forward movement of the stepping foot, so its trajectory, shown in Fig. 6.23, is of interest also.

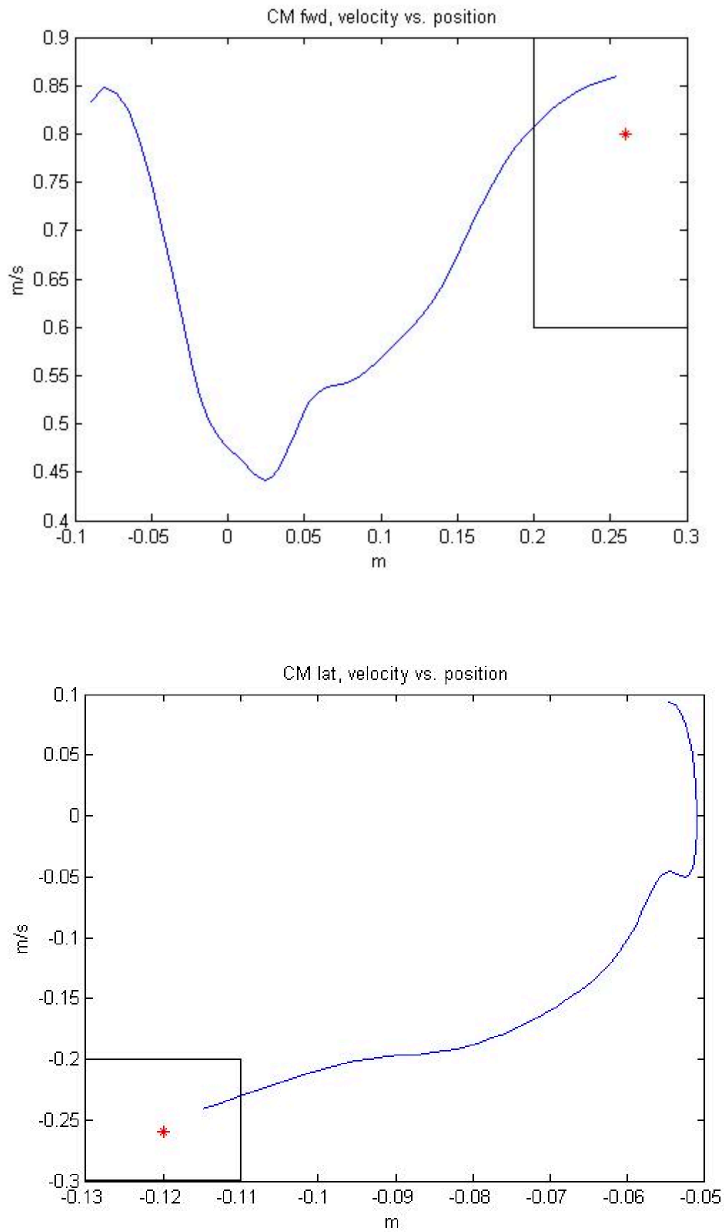


Fig. 6.22 – SISO system state trajectories for CM forward and lateral components, for activities CM_Fwd_1 and CM_Lat_1. Original predictions are shown with a red star.

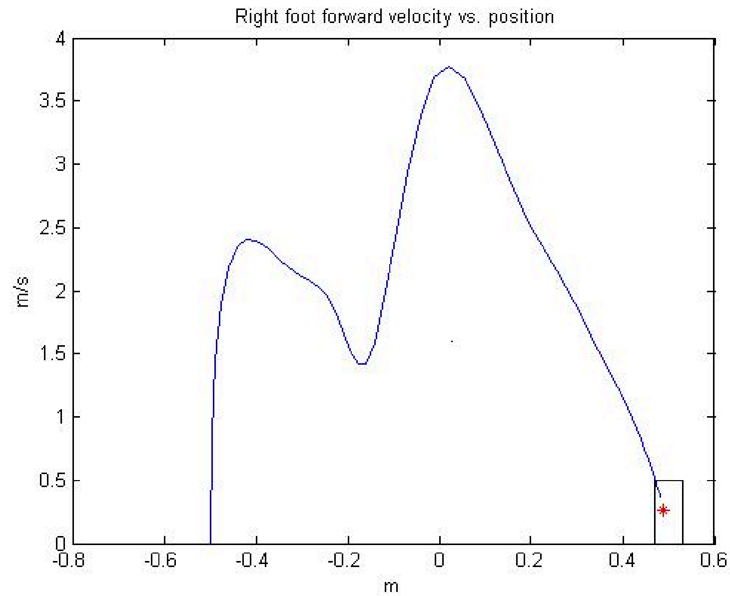


Fig. 6.23 – SISO system state trajectory for right (stepping) foot forward movement, for activity right foot step 1.

Now, let's suppose that a trip occurs as the foot is stepping forward. The trip is a disturbance that impedes the forward progress of the foot, temporarily, so that the trajectory misses the goal region at the desired time, if control parameters are not changed, as shown in Fig. 6.24. After the disturbance occurs, CheckProgress detects, via its prediction, that the goal region will be missed. It returns false for on_target, causing Monitor to call SetControl, in order to adjust parameters (lines 16 and 17). This results in the spring constant for the SISO system for the forward movement of the stepping foot to be increased from 110 to 205, and in a trajectory that reaches the goal region at the desired time, as shown in Fig. 6.25.

Motion sequences for the biped for the uncompensated and compensated cases are shown in Fig. 6.26. As can be seen from these sequences, not compensating for the trip disturbance by increasing the spring constant, as described above, leads to a fall.

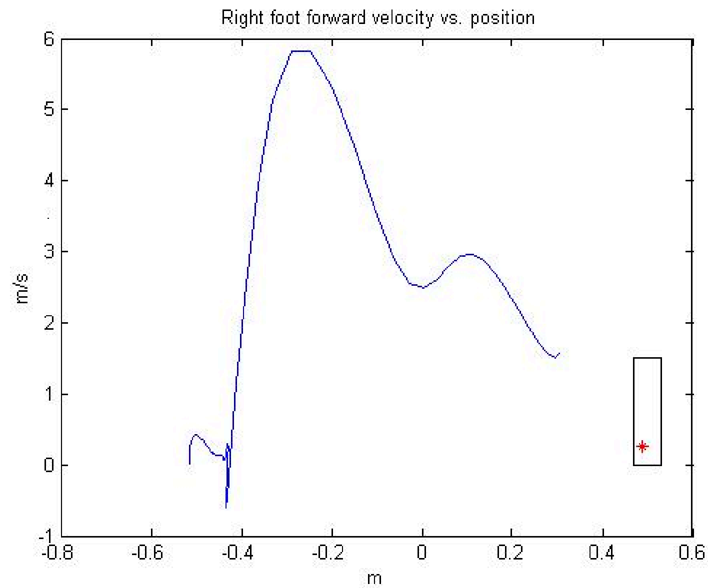


Fig. 6.24 – SISO system state trajectory for right (stepping) foot forward movement, for activity right foot step 1, with trip disturbance. The disturbance impedes the forward progress of the stepping foot, causing it to miss its goal region at the desired time.

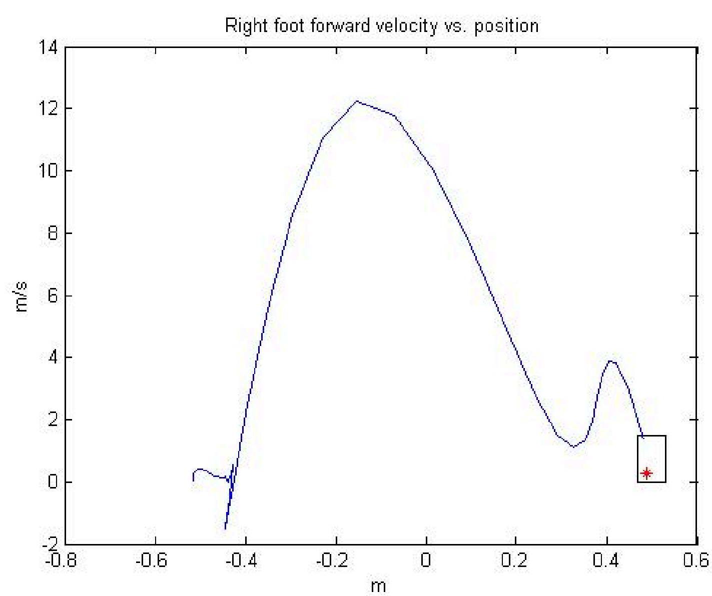
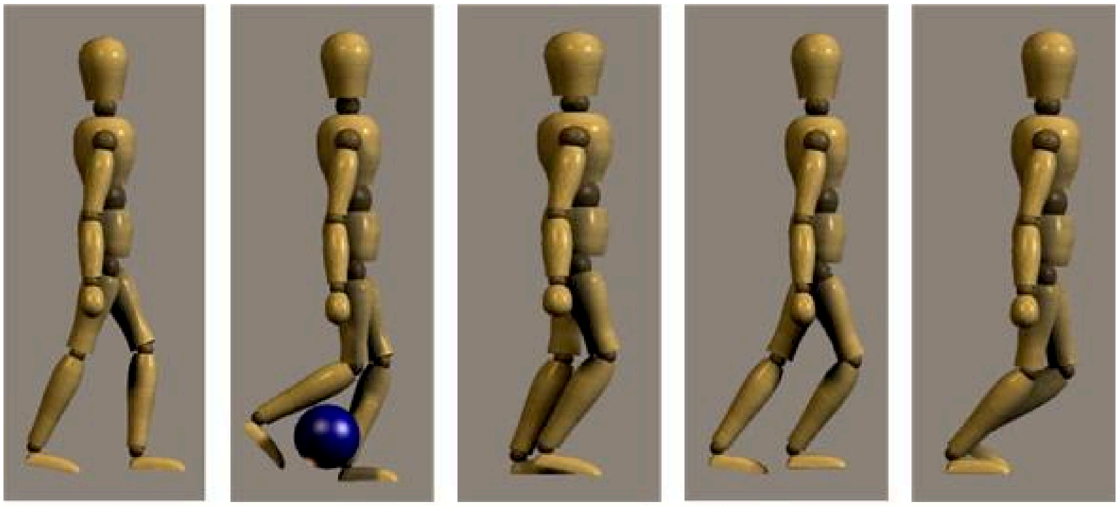
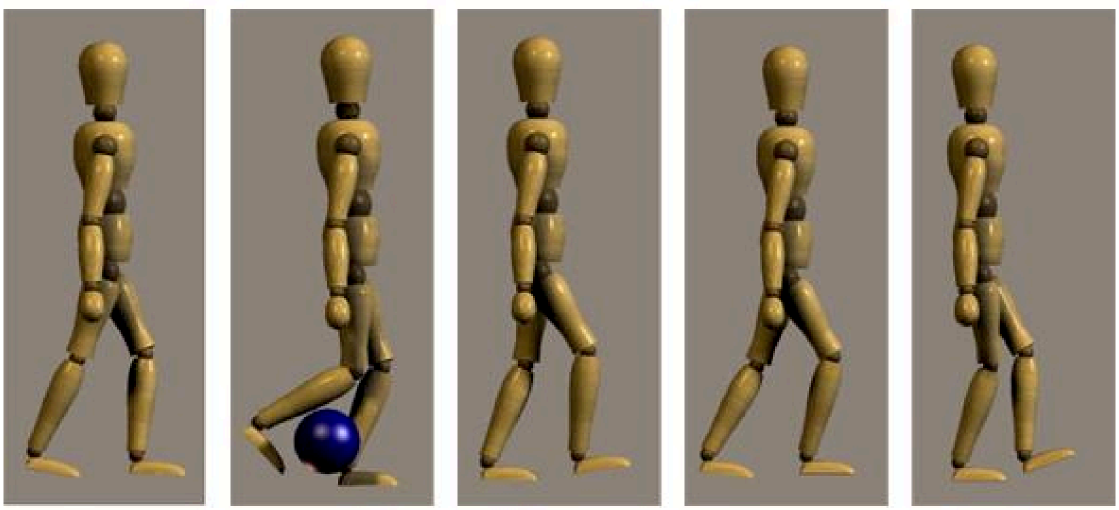


Fig. 6.25 – Stepping foot trajectory with trip disturbance and compensation for disturbance. The disturbance slows progress of the foot, but the compensation speeds it up so that the goal region is achieved at the desired time.



a. Fall due to trip



b. Trip recovery

Fig. 6.26 – Trip disturbance; a) the biped falls when the dispatcher does not adjust control parameters; b) fall is avoided through dispatcher adjustment of parameters

6.3.7 Algorithm Complexity Analysis

To control an actual biped, the dispatcher algorithm must run in real time. In order to ensure that this is possible, we now analyze the complexity of SetControl, which is the most computationally intensive part of the dispatcher algorithm.

In our analysis, we assume that an active-set algorithm [Luenberger, 1989] is used to solve the QP of Fig. 6.13. The inner-most loop of such an algorithm involves matrix multiplication, where the matrices are square and the number of rows and columns is the same as the number of parameters. Therefore, if n is the number of parameters, then the complexity of the inner-most loop is $O n^3$ [Cormen, 2000]. This loop is executed a number of times that is proportional to the number of inequality constraints. Given that there are two inequality constraints and two parameters being optimized, the overall complexity is proportional to 16 floating-point operations. The proportionality constant is based on the number of matrix multiplications in the active set method, which is less than 10. Therefore, solution of the QP of Fig. 6.13 takes less than 160 floating-point operations.

The QP problem is solved for each activity ending at a particular event (lines 2 – 4 of SetControlForEventTime, Fig. 6.11), and for each goal event time in the time search performed by SetControl. If we let m be the maximum number of activities ending at an event, and n_t, the maximum number of times searched by SetControl, then the number of QP problems solved is $O(m n_t)$. If we assume, based on the previous example execution discussion, that there are no more than 3 activities that end at an event, and no more than 10 discrete times being searched, then the number of QP problems solved by SetControl is less than 30. Combining this with the previous analysis of QP solution complexity implies that the computational load of SetControl related to QP solution is less than 30 x 160 = 4800 floating point operations.

Given that today's ordinary PC's run at several Gigahertz, and are capable of Giga-FLOP performance, it is reasonable to use an estimate of 1 nanosecond for the time needed to perform one floating-point operation. This indicates that the QP-related solution time of SetControl is less than 5 microseconds. Now, SetControl is called at most once for each iteration of Monitor. In our testing, we set this iteration to occur at an interval of 50 milliseconds. Therefore, the worst-case estimate for SetControl is well within the limits needed for robust real-time control.

As we discuss in Chapter 10, there are also a number of ways to improve performance of the dispatcher algorithm. Although not needed for this application, such improvements could be useful for systems with more activities, and a larger range of times to search.

In this chapter, we have provided a detailed description of the dispatcher algorithm. This algorithm executes a QCP by keeping trajectories associated with activities within the flow tubes for those activities. The algorithm does this by adjusting control parameters, and goal region arrival times within the temporal bounds specified in the QCP. Thus, the dispatcher is a time varying control program that attempts to ensure successful execution of the QCP by ongoing recalibration of the decoupled SISO systems, based on the predicted trajectory given the current state and settings.

This concludes our discussion of the dispatcher, and its execution of QCP's. In the next Chapter, we describe the plan compiler, and how it generates a QCP from a QSP.

7 Plan Compiler

The purpose of the plan compiler is to generate a qualitative control plan (QCP) from a qualitative state plan (QSP), as described in Chapter 1. This Chapter describes how the plan compiler accomplishes this, and how it fulfills the requirements defined for the QCP in Chapter 5.

We begin, in Section 7.1, with a definition of the problem solved by the plan compiler. We follow this, in Section 7.2, with a discussion of computation of flow tubes for a single activity. Recall, from Definition 5.2, that we use a flow tube approximation consisting of rectangular initial and goal regions, and a controllable duration. A key goal of Section 7.2 is to derive a set of *analytic relations* between the parameters of this approximation. These relations are then used, in Section 7.4, as part of a complete problem formulation for computation of all flow tubes in the QCP. In Section 7.2, in order to simplify computation, we base our discussion on a simple control law consisting of two acceleration spikes, where an acceleration spike represents a step change in velocity. Use of such a simple control law makes it easy to compute state trajectories resulting from the control action. Although the control law is very simple, this discussion provides intuition about the flow tube computation problem, especially, how plant dynamics and actuation limits determine important characteristics of the flow tube.

In Section 7.3, we extend this discussion to a more general control law called a *proportional-differential* (PD) control law. This control law is generally applicable to a large number of problems, including control of bipeds. We also discuss specializations of this control law that are useful for controlling a biped's center of mass movement.

In Section 7.4, we use the analytic relations between flow tube parameters, developed in Section 7.3, to formulate a problem for computing flow tubes for all activities in the QCP. The problem is formulated as an optimization problem and is solved using a nonlinear programming algorithm. This, along with an algorithm that transforms temporal constraints into dispatchable form, results in a plan compiler that produces a correct QCP for a QSP, according to Definition 5.3. In Chapter 9, we present example qualitative control plans produced by the plan compiler.

7.1 Plan Compiler Problem

The plan compiler takes, as input, a QSP, as specified by Definition 4.3. It produces a correct QCP for the QSP, as specified by Definition 5.3. Additionally, it strives to generate a QCP that maximizes the robustness goals described in Sections 5.1.5 and 5.4.5.

Definition 7.1 (Plan Compiler Problem): Given an input QSP, as specified in Definition 4.3, the plan compiler generates a corresponding correct QCP as specified in Definition 5.3. Additionally, the plan compiler maximizes robustness by maximizing the initial regions and temporal durations of control activities in the QCP (see Definitions 5.1, 5.2, and 5.7).

The dual goals of maximizing initial regions and temporal durations of control activities were introduced in Section 5.1.5. A discussion of how this improves robustness was provided in Section 5.4.5.

As specified in Definition 5.3, a QCP has the same activity, event and temporal constraint structure as the input QSP, except that activities in the QCP are converted into control activities in the QCP. Thus, the first task of the plan compiler is to copy the QSP activities, events, and temporal constraints into the QCP, and to convert the activities into control activities (see Definition 5.2). This task is a straightforward copy operation, and is described in more detail in Section 7.4. The remaining tasks of the plan compiler are then to compute the flow tubes for all activities, and to compute the dispatchable graph that satisfies Definition 5.9.

Computation of activity flow tubes involves computing, for each control activity, the R_{init} and R_{goal} regions, the l and u duration bounds, and the CP control parameter constraints (see Definitions 5.2, 5.7, and 5.8). As suggested by Definition 7.1, this is a constrained optimization problem. This type of problem is formulated as a set of parameters to be optimized, a set of equality and inequality constraints that relate the parameters, and a cost function of the parameters. In this case, the parameters are the R_{init} and R_{goal} regions, and the l and u duration bounds. The constraints result from the controllability requirements, specified in Definition 5.8. The optimization goals are to maximize the $[l,u]$ duration range, and the R_{init} regions of the control activities.

A key aspect of this problem is the way that the initial and goal regions, the duration bounds, and the control parameter ranges for control activities are related by dynamic constraints of the plant. In Sections 7.2 and 7.3, we derive these constraints for a single activity. We also derive a cost function for the optimization goals. In Section 7.4, we use these constraints and cost function as part of an overall constrained optimization problem formulation, for computing the flow tubes of all activities in the QCP.

7.2 Flow Tube Computation for Single Activity using Two-spike Control Law

To begin our discussion of the plan compiler, we focus, in this section, on deriving a set of analytic relations between the parameters of our flow tube approximation. In this section, we restrict ourselves to using a simple, two-spike control law, in order to gain intuition about the problem. In Section 7.3, we extend this analysis to the more practical PD control law.

7.2.1 Flow Tube Approximation Parameters

As stated in Section 7.1, computation of a flow tube approximation for a control activity involves computing the R_{init} and R_{goal} regions, the l and u duration bounds, and the CP control parameter ranges of the activity. The region R_{init} is a rectangle defined by the tuple $\langle y_{init\_\min}, y_{init\_\max}, \dot{y}_{init\_\min}, \dot{y}_{init\_\max} \rangle$. Similarly, the region R_{goal} is a rectangle defined by the tuple $\langle y_{goal\_\min}, y_{goal\_\max}, \dot{y}_{goal\_\min}, \dot{y}_{goal\_\max} \rangle$.

As discussed in Section 5.3.3, our approximation of the controllable tube set (Def. 5.7) is based on an initial rectangular region that is a subset of the intersection region described in Section 5.1.5. If the system is in a state that falls in this initial region, then the dispatcher can arbitrarily decide any duration between l and u, as discussed in Section 5.1.5.

Computation of the $[l,u]$ bounds is important because algorithms for temporally flexible plan execution, such as the hybrid dispatcher described in Chapter 6, require a specification of the controllable durations of activities. This is in addition to temporal coordination constraints between activities, which are specified explicitly in the QSP. Prior work (see Chapter 2) assumes that these activity durations are somehow externally provided. For our system, the controllable durations are derivable from a control activity's controllable tube set (Def. 5.7), which is a function of the dynamic limitations of the system. Thus, although activity durations can be specified explicitly in the QSP, such specifications are made for the purpose of satisfying a task goal, not to take dynamics into account. By computing the controllable tube sets for activities, and by deriving the associated controllable durations, our compiler automatically generates temporal constraints, not included in the QSP, which take dynamic limitations into account.

7.2.2 Two-spike Control Law

As specified in Definition 5.4, a fixed-duration flow tube is constrained by the closed-loop dynamic limitations of the plant (see also Definitions 4.1 and 4.2). This implies that our approximation of a controllable tube set (Definition 5.7), using R_{init}, R_{goal}, and $[l,u]$, is also constrained by these dynamic limitations. The closed-loop dynamics are a function of the control law, which may incorporate actuation constraints.

In order to simplify our discussion, we restrict our analysis, in this section, to a very simple control law. This control law has two acceleration spikes, one at the beginning of an activity's execution, and one at the end, as shown in Fig. 7.1. The spikes (Def. 5.10) must be in opposite directions and have finite area. For example, the first spike shown in Fig. 7.1 a is positive, and results in a positive step change in velocity, as shown in Fig. 7.1b. After this spike, the trajectory continues at constant velocity, until the negative spike occurs, which causes a negative step change in velocity, as shown in Fig. 7.1b.

Definition 7.2 (Two-spike control law): Given a control activity, CA (Def. 5.2), executed with duration d, a *two-spike control law* is a control action with two acceleration spikes (Def. 5.10), where the first spike occurs at the start of the activity, and the second at the end. The spikes have opposite direction, and have a limit, $\max A$, on their area, representing an actuation limit. The spikes are applied to the SISO system, $S(A(CA))$ associated with CA (see Definitions 5.2 and 4.1).

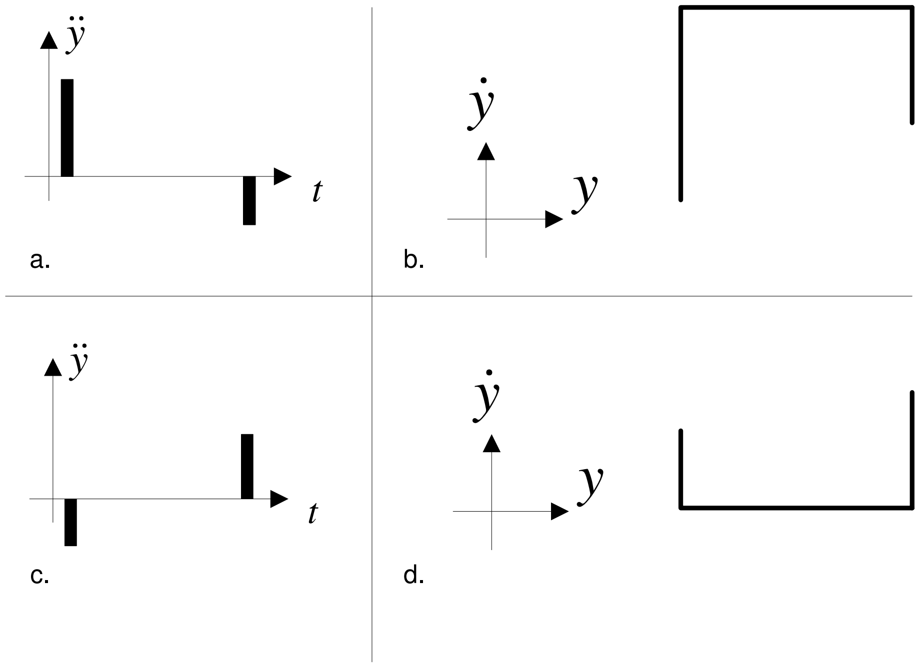


Fig. 7.1 – Two-spike control input. a. A positive spike followed by a negative spike results in the phase-plane trajectory shown in b. c. A negative spike followed by a positive spike results in the trajectory shown in d.

Acceleration spikes in opposite directions imply that velocity does not change monotonically. However, we assume that position changes monotonically, which implies that the velocity does not change sign.

7.2.3 Trajectories Representing Duration Bounds

In order to determine the relation between controllable duration, and initial and goal regions of an activity, we investigate trajectories from initial to goal regions, and their associated durations. We first define the concept of a controllable duration bound, relating this to previous controllability definitions in Chapter 5. We then derive trajectories that correspond to these bounds.

Definition 7.3 (Controllable Duration Bound): Let CA be a control activity (Def. 5.2), with rectangular initial and goal regions, $R_{init}(CA)$ and $R_{goal}(CA)$. A duration, $d \in \Re$, is said to be *controllable* with respect to CA, if $R_{goal}(CA)$ can be reached from any starting point in $R_{init}(CA)$ with a trajectory of duration d, through appropriate setting of control parameters, and assuming no disturbances. A duration bound $[l,u]$ is said to be controllable with respect to CA, if every duration, d, in this bound ($l \le d \le u$) is a controllable duration.

If a control activity, CA, is spatially and temporally controllable with respect to $R_{init}(CA)$ and d, as defined by Definition 5.6, then d is a controllable duration for CA, according to Definition 7.3. If a control activity, CA, is controllable according to Definition 5.7, then, by Lemma 5.2, $[l(CA), u(CA)]$ is a controllable duration bound for CA.

Next, to determine the relation between initial and goal regions, and bounds on the controllable duration, we consider two trajectories that correspond to the lower and upper bounds. The guaranteed fastest trajectory (GFT) of a control activity is the trajectory, within the approximation of the controllable tube set, with the minimum time that can be guaranteed. This time corresponds to l in $[l,u]$. Similarly, the guaranteed slowest trajectory (GST) corresponds to u, the maximum time that can be guaranteed.

Definition 7.4 (Guaranteed fastest and slowest trajectories): Given a control activity, CA (Def. 5.2), with an approximation, $tube_{rect}$, of a controllable tube set (Def. 5.7), represented by R_{init}, R_{goal}, and $[l,u]$, the guaranteed fastest trajectory (GFT) is the trajectory, within $tube_{rect}$, with the minimum time that can be guaranteed for getting from any point in R_{init} to some point in R_{goal}. Thus, it is the trajectory, $traj_{GFT} \in tube_{rect}$ such that $\forall traj \in tube_{rect}\ duration(traj) \ge duration(traj_{GFT})$. The guaranteed slowest trajectory (GST) is the trajectory, within $tube_{rect}$, with the maximum time that can be guaranteed for getting from any point in R_{init} to some point in R_{goal}. Thus, it is the trajectory, $traj_{GST} \in tube_{rect}$ such that $\forall traj \in tube_{rect}\ duration(traj) \le duration(traj_{GST})$.

The durations of the GFT and GST correspond to the lower and upper bounds of a controllable duration for the control activity. Thus, given a control activity, CA, with controllable duration bound $[l,u]$, then $l = duration(traj_{GFT})$ and $u = duration(traj_{GST})$.

7.2.4 GFT and GST for Two Spike Control Law

As suggested by Definition 7.4, the GFT and GST can be used to derive the relation that we seek between initial and goal regions, and controllable duration. The GFT and GST depend on the particular control law used. To develop intuition, we start by determining the GFT and GST for the two spike control law (Def. 7.2).

Consider the initial and goal region rectangles shown in Fig. 7.2. This figure shows four points: A, B, C, and D, on the corners of the regions; these will prove important for our subsequent discussion of trajectories associated with controllable duration bounds. Point A is the point in the initial region with maximum position and velocity. It is, thus, the closest and fastest departure point from the initial region to the goal. Point B is the point in the initial region with minimum position and velocity. It is, thus, the farthest and slowest departure point from the initial region to the goal. Point D is the point in the goal region with minimum position and maximum velocity. It is, thus, the closest and fastest entry point to the goal from the initial region. Point C is the point in the goal region with maximum position and minimum velocity. It is, thus, the farthest and slowest entry point to the goal from the initial region.

Definition 7.5 (Initial and Goal Region Points): Given a control activity, CA (Def. 5.2), with rectangular initial and goal regions, $R_{init}(CA)$ and $R_{goal}(CA)$, we define the following points in these regions (see also Fig. 7.2).

A – the maximum position and velocity corner of the initial region,

B – the minimum position and velocity corner of the initial region,

C – the maximum position and minimum velocity corner of the goal region,

D – the minimum position and maximum velocity corner of the goal region.

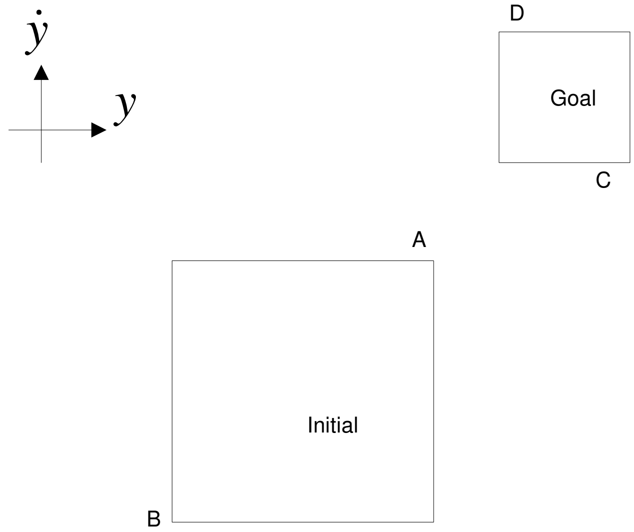


Fig. 7.2 – Point A is the minimum position and velocity corner, and point B is the maximum position and velocity corner of the initial region. Point C is the maximum position and minimum velocity corner, and point D is the minimum position and maximum velocity corner of point D.

We now determine the GFT and GST in terms of points A, B, C, and D, for a two spike control law. This will give us a set of relations between initial region, goal region, and controllable duration.

Recall, from Fig. 7.1, that there are two basic cases of the two-spike control law. For the first case, shown in Fig. 7.1 a. and b., the first spike is positive, and the second is negative. The first spike accelerates the trajectory to a higher velocity, and the second decelerates it. This *push-pull* action is used when we wish to cover a distance quickly, that is, more quickly than if there were no control action, and just the initial velocity were used. The stronger the control action (the greater the area of the spkes), the higher will be the maximum velocity achieved, and the shorter will be the time needed to reach the goal position.

Now, for the GFT, we wish to reach the goal rectangle from any point in the initial rectangle as quickly as possible. Therefore, we wish to use the strongest possible push-pull control action for the GFT; we would like to accelerate as much as possible, then decelerate as much as necessary, in order to be in the velocity limits of the goal region.

Now that we know the control action for the GFT, the only remaining question is where this trajectory should begin and end. Because the GFT is guaranteed to be the fastest trajectory from any point in the initial region (Def. 7.4), we must consider the worst-case starting point. This is point B, because it is the minimum velocity point in the initial region that is furthest from the goal. Hence, from any point in the initial region, we are guaranteed to get to the goal region at least as quickly as we can if we start from point B. In order to understand where the GFT should end, consider that Definition 7.4 requires that it end at some point in the goal region; any point in the goal region is acceptable. Therefore, we may consider the best-case ending point. This is point D, because it is the maximum velocity point in the goal region that is nearest to the goal. Hence, from any particular point in the initial region, we are guaranteed to get to point D at least as quickly as any other point in the goal region.

The GFT is shown in Fig. 7.3. It begins at point B, ends at point D, and uses a push-pull control action, for the reasons stated above.

The second case of the two-spike control law is shown in Fig. 7.1 c. and d. In this case, the first spike is negative, and the second is positive. The first spike decelerates the trajectory to a lower velocity, and the second accelerates it. This *pull-push* action is used when we wish to cover a distance slowly, that is, more slowly than if there were no control action, and just the initial velocity were used. For the GST, we wish to reach the goal rectangle from any point in the initial rectangle as slowly as possible. Therefore, we wish to use the strongest possible pull-push control action for the GST; we would like to decelerate as much as possible, then accelerate as much as necessary, in order to be in the velocity limits of the goal region.

As with the GFT, we determine start and end points for the GST by considering the requirements stated in Definition 7.4. Because the GST is guaranteed to be the slowest trajectory from any point in the initial region, the worst-case starting point is point A, because it is the maximum velocity point in the initial region that is closest to the goal. Hence, from any point in the initial region, we are guaranteed to get to the goal region at least as slowly as we can if we start from point B. The best-case end point for the GST is point C, because it is the minimum velocity point in the goal region that is furthest from the goal. Hence, from any particular point in the initial region, we are guaranteed to get to point C at least as slowly as any other point in the goal region.

The GST is shown in Fig. 7.3. It begins at point A, ends at point C, and uses a pull-push control action, for the reasons stated above. The constraints on the GFT and GST, discussed above, are stated more formally in Theorem 7.1.

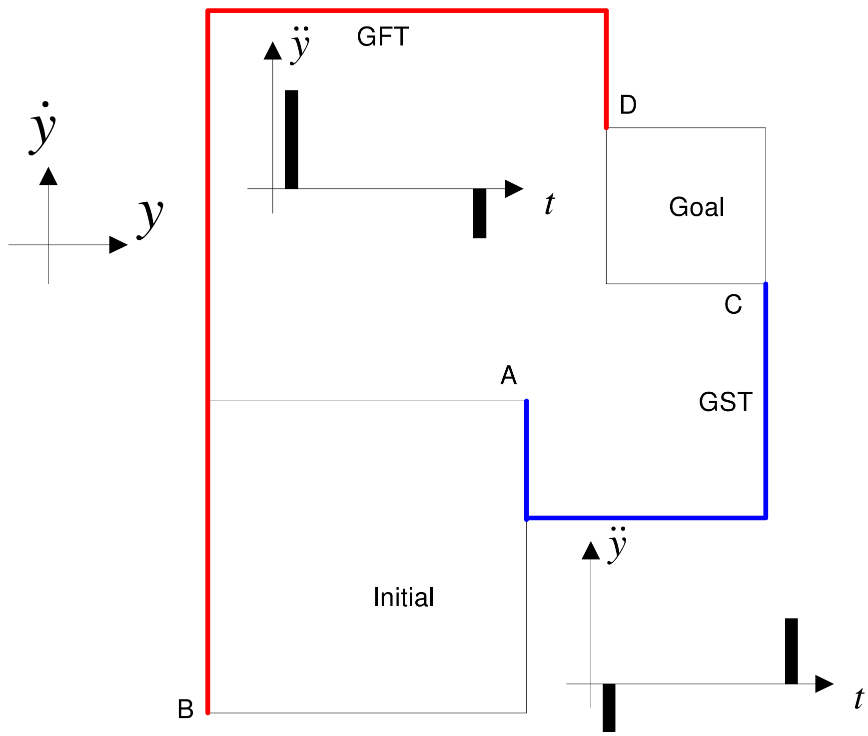


Fig. 7.3 – GFT, GST for two-spike control input

Theorem 7.1 (GFT and GST for a two spike control law): Let CA be a control activity (Def. 5.2), with controllable duration bound $[l,u]$, and regions R_{init} and R_{goal}, with points for these regions A, B, C, and D, as specified in Definition 7.5. For a two-spike control law, constraints on the GFT and GST are then specified as:

$$\dot{y}(D) = \dot{y}(B) + \Delta v_1 + \Delta v_2 \qquad \text{(GFT)}$$

$$y(D) = y(B) + (\dot{y}(B) + \Delta v_1)l$$

$$\dot{y}(C) = \dot{y}(A) + \Delta v_3 + \Delta v_4 \qquad \text{(GST)}$$

$$y(C) = y(A) + (\dot{y}(A) + \Delta v_3)u$$

where Δv_1 and Δv_2 are the areas of the first and second spikes for the GFT, and Δv_3 and Δv_4 are the areas of the first and second spikes for the GST. If the actuation bound on the two-spike control law is $\max A$ (Def. 7.2), then the spikes are limited by the following inequality constraints:

$$-\max A \leq \Delta v_1 \leq \max A$$

$$-\max A \leq \Delta v_2 \leq \max A$$

$$-\max A \leq \Delta v_3 \leq \max A$$

$$-\max A \leq \Delta v_4 \leq \max A$$

Additionally, to ensure that the initial region, the goal region, and the controllable duration, are not empty, we require that

$$l \leq u$$

$$y(A) \geq y(B)$$
$$\dot{y}(A) \geq \dot{y}(B)$$
$$y(C) \geq y(D)$$
$$\dot{y}(C) \leq \dot{y}(D)$$

Theorem 7.1 provides the relation we seek between initial region, goal region, and controllable duration. Note that there are 14 parameters in this relation; the four parameters $y(A), \dot{y}(A), y(B), \dot{y}(B)$ define the initial region, the four parameters $y(C), \dot{y}(C), y(D), \dot{y}(D)$ define the goal region, the four parameters $\Delta v_1, \Delta v_2, \Delta v_3, \Delta v_4$ define the control action, and the two parameters l and u define the controllable duration. Theorem 7.1 specifies four equality constraints and nine inequality constraints. Since there are more parameters than constraints, the relation specified by Theorem 7.1 is under-constrained, and there are multiple solutions that satisfy the constraints.

Some of this ambiguity may be resolved through further constraints specified in the QSP, or through interaction with other activities in the QCP, as discussed in Section 7.4. Any remaining ambiguity is resolved by using a cost function. Recall from Definition 7.1 that we wish to maximize robustness by maximizing the initial regions and controllable durations of control activities in the QCP. The goal of maximizing controllable duration is expressed using the following term in the cost function:

$$-w_{cd}(u-l)$$

The weighting factor, w_{cd} is used to prioritize this goal relative to others. Since cost is to be minimized, the negative sign encourages maximizing the difference between u and l. The goal of maximizing the initial region area is expressed using the following cost function term:

$$-w_{ir}\left(y(A)-y(B)\right)\left(\dot{y}(A)-\dot{y}(B)\right)$$

The weighting factor, w_{ir} is used to prioritize this goal. The negative sign encourages maximizing the area.

As introduced in Section 5.1.5, the goal of maximizing the initial region competes with the goal of maximizing controllable duration. The weighting factors w_{cd} and w_{cd} are used to control the trade-off between these goals. We will return to this cost function, and the values that we use for these weighting factors, shortly.

Fig. 7.3 shows an example GFT and GST. For the GFT, the first spike results in a positive velocity step from $\dot{y}(B)$ to $\dot{y}(B)+\max A$. The second spike results in a negative velocity step from $\dot{y}(B)+\max A$ to $\dot{y}(D)$. For the GST, the second spike results in a positive velocity step from $\dot{y}(C)-\max A$ to $\dot{y}(C)$. The first spike results in a negative velocity step from $\dot{y}(A)$ to $\dot{y}(C)-\max A$. Thus, for the GFT, position increases at maximum possible velocity for the entire duration $0 \le t \le l$. For the GST, position increases at minimum possible velocity for the entire duration $0 \le t \le u$.

7.2.5 Optimality of Initial Region Defined by GFT and GST for Two Spike Control Law

Theorem 7.1 provides a relation between the rectangular initial and goal regions, and the controllable duration of a flow tube approximation, based on the GFT and GST trajectories. In this subsection, we investigate properties of the initial region, R_{init}, specified by this relation, for a given goal region, R_{goal}, and controllable duration, $[l,u]$.

Recall that Definition 5.7 and Lemma 5.2 define $INITSEC_{controllable}$ to be the initial cross section region from which a given goal region can be reached at any time in a given controllable duration $[l,u]$. In this subsection, we compare R_{init}, the rectangular initial region specified by Theorem 7.1, with $INITSEC_{controllable}$, the true initial cross section region. As stated in Definition 5.7 and Lemma 5.2, if $R_{init} \subseteq INITSEC_{controllable}$, then a trajectory that begins in R_{init} can be controlled to reach R_{goal} at any desired time within the range $[l,u]$, if there are no further disturbances. Besides being a subset, we would like R_{init} to be as large a subset as possible.

We begin by considering that, for a given R_{goal} and $[l,u]$, and for a given initial position, the GFT and GST specified by Theorem 7.1 represent extreme points of initial velocity, when combined with the above-stated cost function term that maximizes the area of R_{init}.

Lemma 7.1 (Initial velocities of GFT and GST are Extreme): Let R_{goal} be a rectangular goal region, $[l,u]$, a controllable duration, and $y(A)$ and $y(B)$, initial positions that specify the maximum and minimum positions of a rectangular initial region, R_{init}. For any particular set of values for R_{goal}, l, u, $y(A)$, and $y(B)$, if we apply the relation stated in Theorem 7.1 and maximize the area of R_{init}, then the resulting initial velocities $\dot{y}(A)$ and $\dot{y}(B)$, for the GST and GFT, respectively, are extreme. Specifically, $\dot{y}(A)$ is the maximum possible velocity for the given $y(A)$, and $\dot{y}(B)$ is the minimum possible velocity for the given $y(B)$.

Because the relation specified by Theorem 7.1 is under-constrained, there are multiple possible GFT and GST combinations for a particular R_{goal} and $[l,u]$ range. However, Lemma 7.1 states that if we use a cost function term to maximize the area of Rinit, then, for a given initial position, $y(B)$, for the GFT, the corresponding initial velocity, $\dot{y}(B)$, will be the minimum feasible one. Similarly, for a given initial position, $y(A)$, for the GST, the corresponding initial velocity, $\dot{y}(A)$, will be the maximum feasible one. We can now use this to expand a set of GFT and GST points, corresponding to a set of initial $y(A)$ and $y(B)$ positions, as shown in Fig. 7.4.

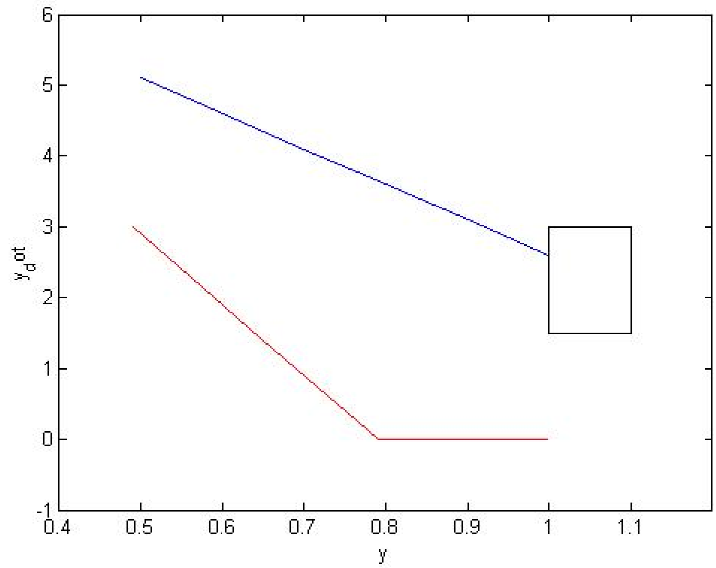


Fig. 7.4 – Initial GFT points, in red, and initial GST points, in blue, using Lemma 7.1. The goal region, R_{goal}, is shown in black. Its position ranges from 1 to 1.1, and its velocity ranges from 1.5 to 3. Controllable duration bounds are l = 0.1 and u = 0.2. The actuation limit is $\max A$ = 2. The region between the set of initial GFT points, and the set of initial GST points is $INITSEC_{controllable}$.

Because the velocities for the initial GFT points are the minimum ones possible, and those for the initial GST points are the maximum ones possible, the region between the set of initial GFT points, and the set of initial GST points is $INITSEC_{controllable}$. It is the true, un-approximated, initial cross section specified in Definition 5.7.

Lemma 7.2 (GFT and GST initial points determine controllable initial region): The set of initial GFT and GST points specified in Lemma 7.1 determines the boundaries of $INITSEC_{controllable}$ (Def. 5.7). Thus, for some point, P, $P \in INITSEC_{controllable}$ iff

$$\dot{y}_{GFT}(P) \leq \dot{y}(P) \leq \dot{y}_{GST}(P)$$

In Fig. 7.4, each pair of GFT and GST points defines a rectangular initial region, R_{init}, as specified by Theorem 7.1 and Lemma 7.1. An example set of such rectangles is shown in Fig. 7.5.

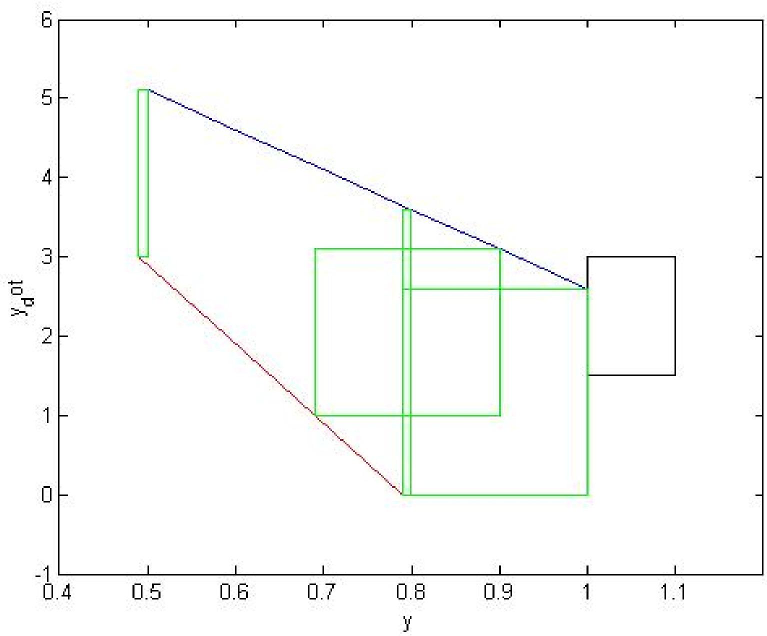


Fig. 7.5 – Example initial region rectangles, in green, for the $INITSEC_{controllable}$ region shown in Fig. 7.4.

Note that each of these example rectangles is a subset of $INITSEC_{controllable}$. If every R_{init} specified in this way is a subset of $INITSEC_{controllable}$, then each R_{init} satisfies Definition 5.7, and from Lemma 5.2, any trajectory that begins in R_{init} can be controlled to reach R_{goal} at any desired time within the range $[l,u]$, if there are no further disturbances.

Theorem 7.2 (R_{init} **for a two spike control law):** The rectangular initial region, R_{init}, specified by any pair of initial GFT and GST points specified in Lemma 7.1 is a subset of $INITSEC_{controllable}$ $(R_{init} \subseteq INITSEC_{controllable})$. Furthermore, R_{init} is maximal in that the velocity $\dot{y}(A)$ for point A is the maximum possible velocity for position $y(A)$ of point A.

Similarly, the velocity $\dot{y}(B)$ for point B is the minimum possible velocity for position $y(B)$ of point B.

In our flow tube approximation (Def. 5.2), we are allowed only a single initial rectangle, R_{init}; we must choose one from the multiple possible rectangles specified in Theorem 7.2. In order to maximize robustness (Def. 7.1), we choose the rectangle with the largest area that is consistent with other constraints (see also Section 7.4).

7.2.6 Trade-off Between Initial Region Size and Controllable Duration

Fig. 7.4, and Fig. 7.6a below, show $INITSEC_{controllable}$ for a controllable duration of l = 0.1 and u = 0.2, and with goal region and actuation limit parameters as specified in Fig. 7.4. Fig. 7.6b shows $INITSEC_{controllable}$ for the same parameters, except that the controllable duration is l = 0.2, and u = 0.2.

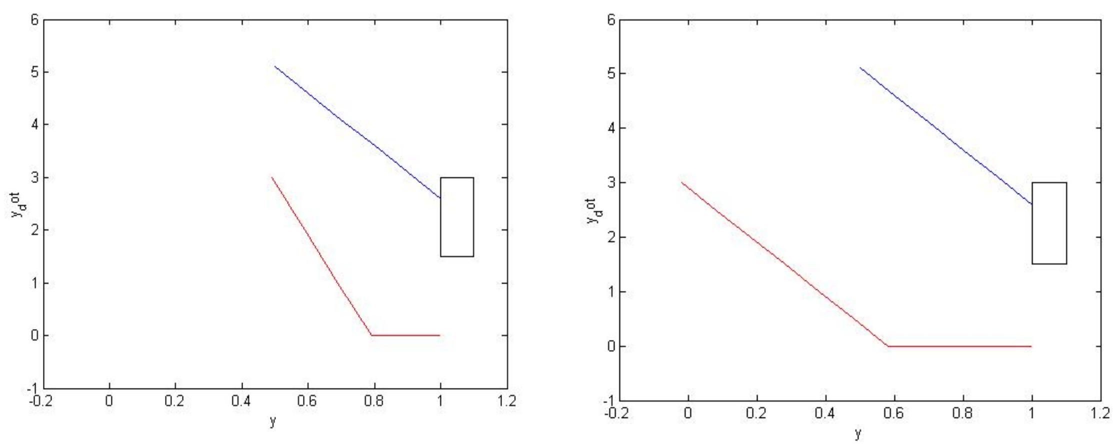


Fig. 7.6a. - $INITSEC_{controllable}$ for l = 0.1 and u = 0.2. b. - $INITSEC_{controllable}$ for l = 0.2 and u = 0.2.

The initial region shown in Fig. 7.6b is bigger; because the lower bound has increased from 0.1 to 0.2, the GFT trajectories have more time to get to the goal region. This results in a shift of the initial GFT positions to include smaller initial positions. For example, the minimum feasible initial position in Fig. 7.6a is 0.5, and the minimum in Fig. 7.6b is 0.

This illustrates the trade-off between initial region size and controllable duration, which was explained in Section 5.1.5. We use the cost function weighting factors, w_{ir} and w_{cd}, which were introduced previously, to resolve this trade-off. Through experimentation (see also Chapter 9), we have empirically determined that a value of 50 for w_{ir} and 1 for w_{cd} works well for a variety of walking tasks. This 50 to 1 ratio appropriately balances the goals of maximizing initial region size and controllable duration, according to the cost function terms presented in Section 7.2.4. For example, if the initial region position range is 0.01 m, and the velocity range is 0.1 m/s, then the area of the initial region is 0.001. With the 50:1 ratio, a controllable duration of 0.05 seconds balances the cost function terms. Similarly, an initial region with area 0.01 balances with a controllable duration of 0.5 seconds. These areas and controllable durations are typical of walking task activities. For example, as presented in Chapter 9, during normal walking, the lateral center of mass position fluctuates by only about 0.1 m. Therefore, an initial region that has a position range of 0.02 m represents a capture range that is 20% of the entire range of motion. This provides significant robustness to significant lateral force disturbances. Similarly, for normal walking, a stepping activity takes about 0.2 seconds, a controllable duration of 0.05 seconds represents a 25% temporal adjustment capability.

During compilation of a full plan, the trade-off is resolved, in many cases, by additional constraints from the QSP, or from other activities. For example, a QSP constraint may explicitly specify the initial region. In this case, the maximum controllable duration is directly determined from the relation in Theorem 7.1.

To summarize, in Section 7.2, we have provided a set of constraints, in Theorem 7.1, that relate the initial region, goal region, and controllable duration of our flow tube approximation, for the two-spike control law. By combining these constraints with a cost function that maximizes initial region size, we guarantee, through Theorem 7.2, that the initial region rectangle of our flow tube approximation is a maximal subset of $INITSEC_{controllable}$, the true initial region. As discussed in Section 7.2.6, an appropriate weighting of cost function terms balances the goal of maximizing the area of $INITSEC_{controllable}$ with the goal of maximizing the controllable duration. In Section 7.3, we extend this discussion to the more general *proportional-differential* (PD) control law.

7.3 Flow Tube Computation for Single Activity using PD Control Law

Due to its simplicity, the two-spike control law, discussed in Section 7.2, is useful for providing insight into the problem of computing a flow tube approximation for a control activity. However, a two-spike control law is not practical for control of the biped, or for many similar applications, because it concentrates all the acceleration into a very short time period. A more generally applicable control law is the PD control law [Ogata, 1982], which adjusts acceleration continuously based on a position and velocity goal. Thus, for control of the biped, we do not use acceleration spike control inputs, but rather, the PD control laws provided by the SISO abstraction (Def. 4.1).

In this section, we extend concepts developed for the two-spike control law to the more generally useful PD control law. Def. 7.3, for controllable duration bound, and Def. 7.4, for the general concept of the GFT and GST, are independent of control law, and therefore, are valid for this section. In order to extend the concepts developed for the two-spike control law to the PD control law, we first show, in Section 7.3.1, how these control laws are similar, as long as we continue to assume that position changes monotonically.

As in Section 7.2, we represent a flow tube approximation using the rectangular initial region, goal region, and controllable duration bounds of a control activity (Def. 5.7). In order to compute this approximation, we seek to establish relations between these regions and bounds, except that in this section, we use a PD control law, rather than a two-spike control law. To this end, in Section 7.3.2, we adapt Theorem 7.1 for a PD control law, resulting in a set of constraints that relate the parameters of the flow tube approximation. As with the two-spike control law, the resulting system of parameters and constraints is under-constrained, resulting in multiple possible solutions. As with the two-spike control law, we resolve this ambiguity using a cost function, and extend Lemma 7.1, 7.2, and Theorem 7.2 to show that rectangular initial regions computed in this way are maximal subsets of $INITSEC_{controllable}$.

In Section 7.3.2, we defer detailed discussion of actuation constraints, and assume that the control parameters used for the PD control law are valid, in that they satisfy any actuation constraints. Detailed discussion of the nature of these actuation constraints is then covered in Section 7.3.3.

7.3.1 Similarity of Two-Spike and PD Control Laws

Recall that a two-spike control law (Def. 7.2) has two acceleration spikes, in opposite directions. For example, as shown in Fig. 7.1 a. and b., the first spike is positive, and the second is negative, resulting in a push-pull action. Position, velocity, and acceleration trajectories for such a control law are shown in Fig. 7.7a.

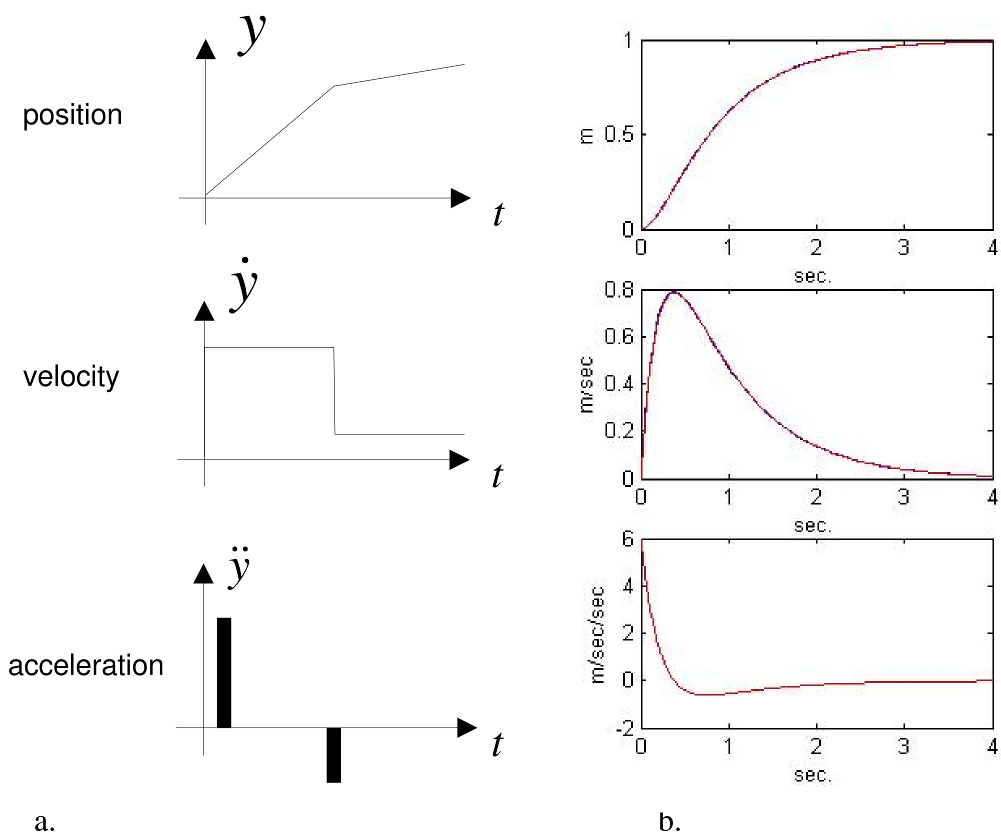


Fig. 7.7 – a. Position, velocity, and acceleration trajectories for two-spike control law; b. the corresponding trajectories for a PD control law, with

$y_{set} = 1, \dot{y}_{set} = 0, k_p = 6, k_d = 6, y(0) = 0, \dot{y}(0) = 0$.

Recall, from Eq. 4.1, that a PD control law specifies acceleration as a function of position and velocity according to

$$\ddot{y} = k_p\left(y_{set} - y\right) + k_d\left(\dot{y}_{set} - \dot{y}\right) \tag{7.1}$$

Position, velocity, and acceleration trajectories for such a control law are shown in Fig. 7.7b. Notice the similarity to the plots of Fig. 7.7a. For both sets of plots, the accelerations are characterized by a period of positive acceleration, followed by a period of negative acceleration. We assume in this section, as we did in Section 7.2, that position changes monotonically. Therefore velocity doesn't change sign. Hence, for both sets of velocity plots in Fig. 7.7a. and b., velocity increases sharply at first, and then decreases, but it never goes negative. For both sets of position plots, position increases sharply, at first, and then increases at a slower rate.

This suggests a similarity in the behavior of the two control laws, and that the concepts developed for the two-spike control law in Section 7.2 can be extended to the PD control law. We begin this extension, in Section 7.3.2, by extending Theorem 7.1 in this way.

7.3.2 GFT and GST for PD Control Law

For the PD control law, as with the two spike control law, we seek to find GFT and GST trajectories that provide a relation between a control activity's initial region, goal region, and controllable duration bound. Valid trajectories for the PD control law (see Defs. 4.1 and 4.2) are specified by Eq. 4.3, which provides an analytic relation between start state, finish state, start time, finish time, and control parameter settings. This equation is restated here.

$$y = e^{\alpha t}\left(K_1 \cos \beta t + iK_2 \sin \beta t\right) + \frac{u}{c} \tag{7.2}$$

$$\dot{y} = e^{\alpha t}\left(\beta\left(-K_1 \sin \beta t + iK_2 \cos \beta t\right) + \alpha\left(K_1 \cos \beta t + iK_2 \sin \beta t\right)\right)$$

where

$$K_1 = y(0) - \frac{u}{c} \qquad \alpha = \frac{-b}{2a}$$

$$K_2 = \frac{i}{\beta}\left(\alpha K_1 - \dot{y}(0)\right) \qquad \beta = \frac{-i\sqrt{b^2 - 4ac}}{2a}$$

$$a = 1$$
$$b = k_d$$
$$c = k_p$$
$$u = k_p\, y_{set} + k_d\, \dot{y}_{set}$$

This equation is of the form

$$\begin{aligned} y(t_2) &= f_1\left(y(t_1), \dot{y}(t_1), y_{set}, \dot{y}_{set}, kp, kd, t_1, t_2\right) \\ \dot{y}(t_2) &= f_2\left(y(t_1), \dot{y}(t_1), y_{set}, \dot{y}_{set}, kp, kd, t_1, t_2\right) \end{aligned} \tag{7.3}$$

We now adapt Theorem 7.1 so that it applies to a PD control law. To do this, we retain the use of the points A, B, C, and D. Due to the fact that position changes monotonically, the GFT still goes from point B to point D, and the GST still goes from point A to point C. However, we replace the GFT and GST position and velocity trajectory equations in Theorem 7.1 with ones based on Eq. 7.3.

Theorem 7.3 (GFT and GST for a PD control law): Let CA be a control activity (Def. 5.2), with controllable duration bound $[l, u]$, and regions R_{init} and R_{goal}, with points for these regions A, B, C, and D, as defined in Definition 7.5. The acceleration input to the SISO system of CA is computed according to a PD control law, using the SISO system's control parameters (Definitions 4.1 and 4.2). If there exists a control parameter setting that results in a trajectory from B to D that is a member of the controllable tube set (Def. 5.7), then there exists a control parameter setting that results in the GFT for CA, and this GFT begins at B and ends at D. Similarly, if there exists a control parameter setting that results in a trajectory from A to C that is a member of the controllable tube set (Def. 5.7), then there exists a control parameter setting that results in the GST for CA, and this GST begins at A and ends at C. The trajectory equations for the GFT and GST are then specified as

$$
\begin{aligned}
y(D) &= f_1\left(y(B), \dot{y}(B), \langle y_{set}, \dot{y}_{set}, kp, kd \rangle (GFT), 0, l\right) \\
\dot{y}(D) &= f_2\left(y(B), \dot{y}(B), \langle y_{set}, \dot{y}_{set}, kp, kd \rangle (GFT), 0, l\right)
\end{aligned}
\qquad \text{(GFT)}
$$

$$
\begin{aligned}
y(C) &= f_1\left(y(A), \dot{y}(A), \langle y_{set}, \dot{y}_{set}, kp, kd \rangle (GST), 0, u\right) \\
\dot{y}(C) &= f_2\left(y(A), \dot{y}(A), \langle y_{set}, \dot{y}_{set}, kp, kd \rangle (GST), 0, u\right)
\end{aligned}
\qquad \text{(GST)}
$$

To ensure that the initial region, the goal region, and the controllable duration, are not empty, we require that

$$
\begin{aligned}
& l \le u \\
& y(A) \ge y(B) \\
& \dot{y}(A) \ge \dot{y}(B) \\
& y(C) \ge y(D) \\
& \dot{y}(C) \le \dot{y}(D)
\end{aligned}
$$

Note that whereas Theorem 7.1 explicitly states actuation constraints in terms of $\max A$, the actuation bound on the two-spike control law, Theorem 7.3 specifies actuation constraints indirectly, by requiring any GFT or GST to be feasible, that is, to be within the controllable tube set (Def. 5.7). This allows us to defer discussion of actuation constraint details for the PD control law to Section 7.3.3.

An example GFT and GST trajectory, using the relation in Theorem 7.3, is shown in Fig. 7.8.

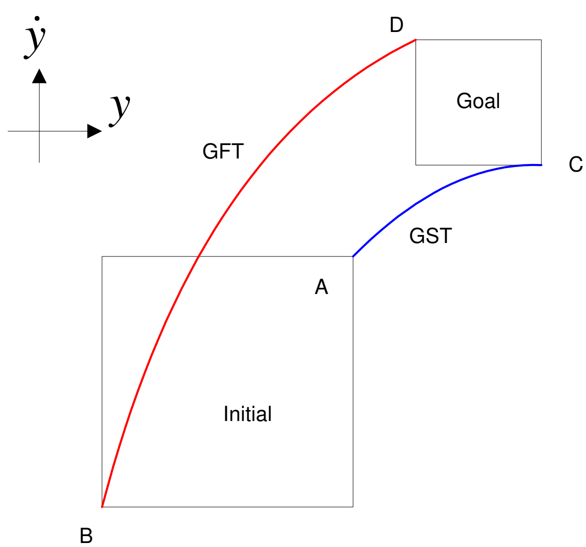


Fig. 7.8 – Example GFT and GST trajectories for PD control law.

As with Theorem 7.1, the relation specified in Theorem 7.3 represents an under-constrained system. To resolve this ambiguity, we use the same cost function terms used for the two-spike control law. Thus, Lemma 7.3 is an extension of Lemma 7.1 for the PD control law.

Lemma 7.3 (Initial velocities of GFT and GST are extreme for PD control law): Let R_{goal} be a rectangular goal region, $[l,u]$, a controllable duration, and $y(A)$ and $y(B)$, initial positions that specify the maximum and minimum positions of a rectangular initial region, R_{init}. For any particular set of values for R_{goal}, l, u, $y(A)$, and $y(B)$, if we apply the relation stated in Theorem 7.3 and maximize the area of R_{init}, then the resulting initial velocities $\dot{y}(A)$ and $\dot{y}(B)$, for the GST and GFT, respectively, are extreme. Specifically, $\dot{y}(A)$ is the maximum possible velocity for the given $y(A)$, and $\dot{y}(B)$ is the minimum possible velocity for the given $y(B)$.

As with the two-spike control law, we can use Lemma 7.3 to expand a set of GFT and GST points, corresponding to a set of initial $y(A)$ and $y(B)$ positions, as shown in Fig. 7.9. Also, as with the two-spike control law, because the initial GFT and GST points are extreme, the region between the set of initial GFT points, and the set of initial GST points is $INITSEC_{controllable}$.

Lemma 7.4 (GFT and GST initial points determine controllable initial region for PD control law): The set of initial GFT and GST points specified in Lemma 7.3 determines the boundaries of $INITSEC_{controllable}$ (Def. 5.7). Thus, for some point, P, $P \in INITSEC_{controllable}$ iff

$$\dot{y}_{GFT}(P) \leq \dot{y}(P) \leq \dot{y}_{GST}(P)$$

As with the two-spike control law, each pair of GFT and GST points in Fig. 7.9 defines a rectangular initial region, R_{init}, as specified by Theorem 7.3 and Lemma 7.3. An example set of such rectangles is shown in Fig. 7.10.

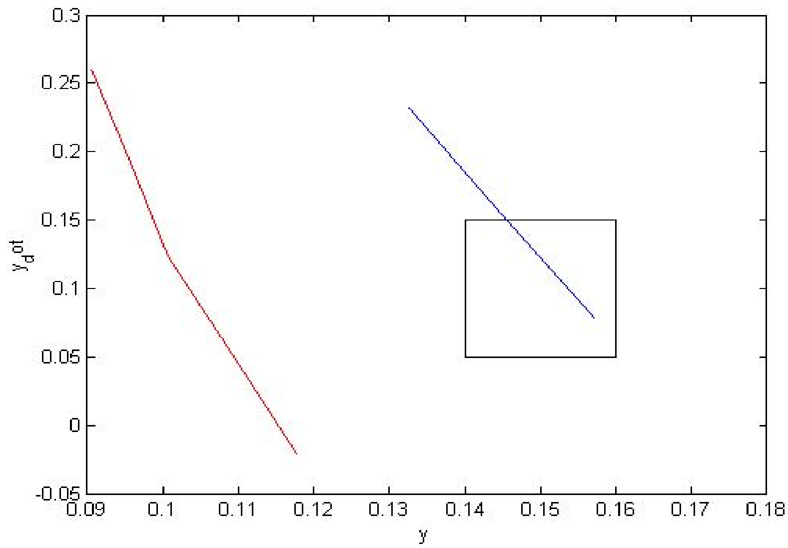


Fig. 7.9 - Initial GFT points, in red, and initial GST points, in blue, using Lemma 7.3. The goal region, R_{goal}, is shown in black. Its position ranges from 0.14 to 0.16, and its velocity ranges from 0.05 to 0.15. Controllable duration bounds are l = 0.2 and u = 0.25. The ZMP range is from –0.05 to 0.25 (see Section 7.3.3). The region between the set of initial GFT points, and the set of initial GST points is $INITSEC_{controllable}$.

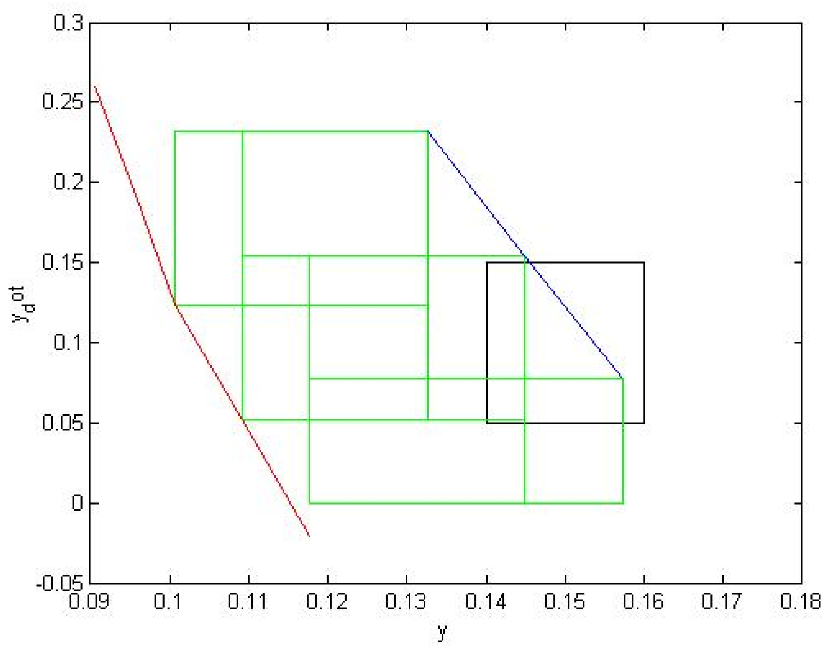


Fig. 7.10 – Example initial region rectangles, in green, for the $INITSEC_{controllable}$ region shown in Fig. 7.9.

As with the two-spike control law, each of these example rectangles is a subset of $INITSEC_{controllable}$.

Theorem 7.4 (R_{init} for a PD control law): The rectangular initial region, R_{init}, specified by any pair of initial GFT and GST points specified in Lemma 7.3 is a subset of $INITSEC_{controllable}$ $(R_{init} \subseteq INITSEC_{controllable})$. Furthermore, R_{init} is maximal in that the velocity $\dot{y}(A)$ for point A is the maximum possible velocity for position $y(A)$ of point A. Similarly, the velocity $\dot{y}(B)$ for point B is the minimum possible velocity for position $y(B)$ of point B.

As with the two-spike control rule, we choose the rectangle with the largest area that is consistent with other constraints (see also Section 7.4). Also, as with the two-spike control law, we use the cost function weighting factors, w_{ir} and w_{cd}, to resolve the trade-off between initial region size and controllable duration.

To summarize, Theorem 7.4 guarantees that R_{init} satisfies Definition 5.7. Therefore, from Lemma 5.2, any trajectory that begins in R_{init} can be controlled to reach R_{goal} at any desired time within the range $[l,u]$, if there are no further disturbances. Furthermore, because R_{init} is maximal, the goal of maximizing robustness by maximizing R_{init} is achieved.

Theorem 7.3 requires that the control parameters be set so that the resulting GFT and GST trajectories are in the controllable tube set (Def. 5.7). This implies that these trajectories are feasible. In particular, it implies that they observe actuation constraints. However, unlike Theorem 7.1, for the two-spike control law, Theorem 7.3 does not explicitly specify the actuation constraints. We have deferred discussion of these constraints in order to simplify the presentation of key concepts in Theorem 7.3. In the next section, we discuss details of the actuation constraints, and how control parameters should be set to satisfy them.

7.3.3 Actuation Constraints for PD Control Law

In computing a rectilinear flow tube approximation, it is necessary to account for limitations imposed by actuation constraints. Actuation constraints limit the set of valid control parameters, specified in Theorem 7.3, and thus, limit the GFT and GST. This, in turn, constrains the relation between the initial and goal regions, and the controllable duration bounds, of the rectilinear flow tube approximation for a control activity.

Therefore, in this section, we discuss two important classes of actuation constraints, and analyze how these constraints limit the set of valid control parameters. For the PD control law, the acceleration input of an SISO system (Def. 4.1) is computed according to Eq. 4.1. The first class of actuation constraint that we consider is a constant limit on this acceleration.

$$\ddot{y} \leq \ddot{y}_{\max} \tag{7.4}$$

This type of actuation constraint is useful for a wide range of applications. For control of the biped, we use this type of actuation constraint to limit the acceleration of the swing leg.

Combining Eq. 7.4 with the PD control law (Eq. 7.1) eliminates the acceleration term, resulting in a relation that constrains the SISO system state.

$$k_p\left(y_{set} - y\right) + k_d\left(\dot{y}_{set} - \dot{y}\right) \leq \ddot{y}_{\max} \tag{7.5}$$

Note that this is of the form $g(y, \dot{y}) \leq 0$, so it is specified as an R_{op} constraint of an activity (Def. 4.4). The control parameters for the GFT and GST, $\langle y_{set}, \dot{y}_{set}, kp, kd \rangle(GFT)$ and $\langle y_{set}, \dot{y}_{set}, kp, kd \rangle(GST)$, must satisfy the equality constraints in Theorem 7.3, and also, the inequality constraint of Eq. 7.5.

The second class of actuation constraint that we consider is a constant limit on the control parameter y_{set}. This is useful for control of the biped's CM. As discussed in Chapter 3, the key actuation limit for balance control is the limit on horizontal CM acceleration due to the limited size of the support base, which limits where the ZMP can be placed. The ZMP is limited to be within the support base defined by the convex polygon surrounding the foot or feet that are on the ground. As was discussed previously in Chapter 3, horizontal CM acceleration is closely approximated by a linear relation between horizontal CM position and ZMP.

$$\ddot{y} = K(y - ZMP) \tag{7.6}$$

This is of the form of the PD control law of Eq. 7.1, where y_{set} in Eq. 7.1 is ZMP in Eq. 7.6, k_p in Eq. 7.1 is $-K$ in Eq. 7.6, and k_d in Eq. 7.1 is 0. Eq. 7.6 can be broken up into its forward and lateral components, corresponding to the separate SISO systems forward and lateral components of horizontal CM movement.

$$\begin{aligned} \ddot{y}_1 &= K(y_1 - ZMP_{fwd}) \\ \ddot{y}_2 &= K(y_2 - ZMP_{lat}) \end{aligned} \tag{7.7}$$

Here, index 1 refers to the SISO system for forward CM movement, and index 2 refers to the one for lateral CM movement. The qualitative state plan specifies foot placements that constrain ZMP. For example, in Fig. 4.14, during execution of CM2, which represents left single support, the ZMP is constrained to be inside the bounds of l1.

$$\begin{aligned} Fwd_{lb}(l1) \le ZMP_{fwd} \le Fwd_{ub}(l1) \\ Lat_{lb}(l1) \le ZMP_{lat} \le Lat_{ub}(l1) \end{aligned} \tag{7.8}$$

As with the first class of actuation constraint, the control parameters for the GFT and GST, $\langle y_{set}, \dot{y}_{set}, kp, kd \rangle (GFT)$ and $\langle y_{set}, \dot{y}_{set}, kp, kd \rangle (GST)$, must satisfy the equality constraints of Theorem 7.3, and the actuation constraint expressed by Eq. 7.8. Eq. 7.8 is a simple, constant constraint on one of these parameters, the position setpoint, y_{set}.

Section 7.3 has provided constraints that must be observed, for the PD control law, when computing the flow tube approximations, along with cost function terms useful for this computation. In the next section, we show how these constraints and cost function terms, are used in the implementation of the plan compiler.

7.4 Plan Compiler Algorithm

As stated in Definition 7.1, the Plan Compiler computes a correct QCP for an input QSP, where a correct QCP for a QSP was defined in Definition 5.3. Definition 5.3 requires that the QCP be controllable, (Def. 5.8), and that it be temporally dispatchable, (Def. 5.9).

In this section, we summarize our plan compiler algorithm for computing a QCP according to Definition 7.1. We begin, in Section 7.4.1, by discussing how the plan compiler algorithm satisfies the controllability requirements in Definition 5.8. In particular, we discuss how we compute the rectilinear flow tube approximations for each activity, using the single-activity constraints specified in Theorem 7.3, as well as additional constraints, such as temporal constraints, which synchronize multiple activities. In Section 7.4.2, we describe how the compiler transforms all temporal constraints in to a temporally dispatchable form, as defined in Definition 5.9.

7.4.1 Satisfying Controllability Requirements

Definition 5.8 states three requirements for controllability of a QCP. The first requires that all control activities in the QCP be controllable, according to Definition 5.7. This requirement pertains to individual activities, independent of other activities. Therefore, it is addressed by the constraints from Section 7.3; controllability of an individual activity is governed by the constraints specified in Theorem 7.3. The second requirement states that all temporal constraints in the QCP must be consistent. This includes the temporal constraints from the QSP, and the controllable duration bounds of all control activities. The third requirement is that the goal regions of all control activities must be subsets of the initial regions of their successors. Additionally, as stated in Definition 7.1, we seek that the QCP maximize robustness by maximizing the initial regions and temporal durations of its control activities (see Defs. 5.1, 5.2, and 5.7).

To construct a QCP, we reformulate these requirements as a constrained optimization problem. In particular, we formulate this problem as a *nonlinear program* (NLP) and use a *sequential quadratic programming* (SQP) optimizer to solve it. The NLP formulation is described in terms of the parameters to be optimized, constraints, and the cost function.

Parameters being optimized

Parameters being optimized are specified in the qualitative control plan's control activities and events. For each control activity, the parameters to optimize are the initial and goal state space rectangular regions, the temporal bounds, and the control parameters (Def. 5.2):

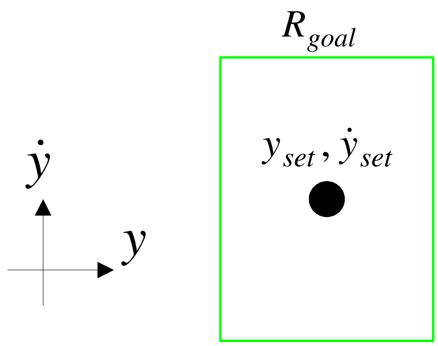
$$y_{\min}(R_{init}), y_{\max}(R_{init}), \dot{y}_{\min}(R_{init}), \dot{y}_{\max}(R_{init}),$$
$$y_{\min}(R_{goal}), y_{\max}(R_{goal}), \dot{y}_{\min}(R_{goal}), \dot{y}_{\max}(R_{goal}),$$
$$l, u, y_{set\_\min}, y_{set\_\max}, \dot{y}_{set\_\min}, \dot{y}_{set\_\max}, kp_{\min}, kp_{\max}, kd_{\min}, kd_{\max}$$

For each event, the parameters to be optimized are the execution window bounds, $[l,u]$, of the event. Note that the control activity start and finish times are not an output of the plan compiler. Rather, the plan compiler computes bounds on these values. The actual start and finish times are determined at runtime by the dispatcher, as described in Chapter 6.

Constraints for Activity Controllability

Constraints for activitiy controllability, the first requirement in Definition 5.8, were presented in Section 7.3. Recall that these constraints relate controllable duration with initial and goal regions, and with control parameter actuation limits. These constraints include the equality constraints for the GFT and GST trajectories, specified in Theorem 7.3. For swing leg activities, such as the activity right foot step 1 in Fig. 4.14, actuation constraints are expressed as a constraint on maximum acceleration, given in Eq. 7.4, resulting in constraints on PD control parameters of the form shown in Eq. 7.5. For this type of activity, we further constrain the position and velocity setpoints, y_{set} and $\dot{y}_{set}$ to be at the center of the goal region using equality constraints, as shown in Fig. 7.11. These setpoints become the goal trajectory point for the dispatcher, within the goal region. By centering these setpoints, we maximize the distance to the goal region boundaries, and thus, maximize robustness to disturbances.

.

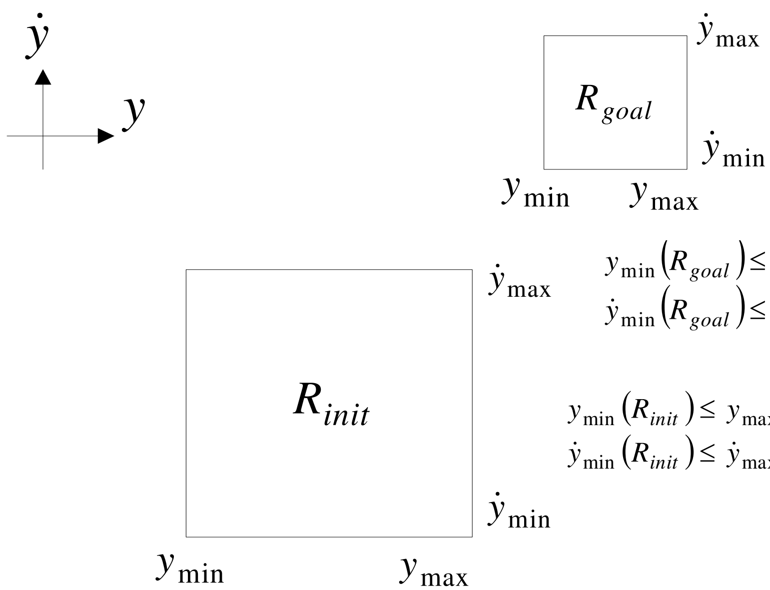


$$y_{set} = \left(y_{\min}\left(R_{goal}\right) + y_{\max}\left(R_{goal}\right)\right)/2$$
$$\dot{y}_{set} = \left(\dot{y}_{\min}\left(R_{goal}\right) + \dot{y}_{\max}\left(R_{goal}\right)\right)/2$$

Fig. 7.11 – Goal setpoints are at the center of the goal rectangle for the swing leg activities.

For CM activities, such as CM1 of Fig. 4.14, actuation constraints are expressed as constraints on the position setpoints, given in Eqs. 7.7 and 7.8. For this type of activity, we set k_d and $\dot{y}_{set}$ to 0, and k_p to $-K$, so that the PD control law is of the form given in Eq. 7.7. The position setpoint is constrained to be within the bounds of the ZMP (Eq. 7.8).

The individual initial and goal regions of control activities also have associated constraints. A simple set of inequality constraints is used to ensure that the initial and goal rectangles have non-negative position and velocity ranges, as shown in Fig. 7.12.



$$y_{\min}\left(R_{goal}\right) \leq y_{\max}\left(R_{goal}\right)$$
$$\dot{y}_{\min}\left(R_{goal}\right) \leq \dot{y}_{\max}\left(R_{goal}\right)$$

$$y_{\min}\left(R_{init}\right) \leq y_{\max}\left(R_{init}\right)$$
$$\dot{y}_{\min}\left(R_{init}\right) \leq \dot{y}_{\max}\left(R_{init}\right)$$

Fig. 7.12 – Inequality constraints for region rectangle existence.

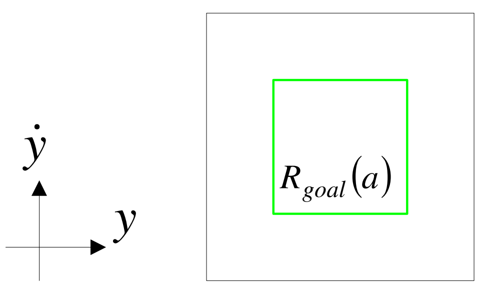
Inequality constraints on individual initial and goal regions are also used to represent region constraints specified explicitly in the QSP (see, Def. 4.4 and Def. 5.2).

$$
\begin{aligned}
&y_{\min}(RS_{init}) \le y_{\min}(R_{init}),\ y_{\max}(R_{init}) \le y_{\max}(RS_{init}),\\
&\dot{y}_{\min}(RS_{init}) \le \dot{y}_{\min}(R_{init}),\ \dot{y}_{\max}(R_{init}) \le \dot{y}_{\max}(RS_{init}),\\
&y_{\min}(RS_{goal}) \le y_{\min}(R_{goal}),\ y_{\max}(R_{goal}) \le y_{\max}(RS_{goal}),\\
&\dot{y}_{\min}(RS_{goal}) \le \dot{y}_{\min}(R_{goal}),\ \dot{y}_{\max}(R_{goal}) \le \dot{y}_{\max}(RS_{goal}),
\end{aligned}
\tag{7.9}
$$

As discussed previously in Chapter 4, upper bounds specified in the QSP may be positive infinity, and lower bounds may be negative infinity. For such cases, the corresponding constraints, of the form of Eq. 7.9, become inactive.

Constraints Relating Activities to Successors

The third requirement stated in Definition 5.8 is that the goal regions of all control activities must be subsets of the initial regions of their successors. Let a be a control activity, and a_{next} its successor. The goal region subset requirement is then expressed as shown in Fig. 7.13.



$R_{init}(a_{next})$

$$
\begin{aligned}
&y_{\min}(R_{init}(a_{next})) < y_{\min}(R_{goal}(a))\\
&\dot{y}_{\min}(R_{init}(a_{next})) < \dot{y}_{\min}(R_{goal}(a))\\
&y_{\max}(R_{init}(a_{next})) > y_{\max}(R_{goal}(a))\\
&\dot{y}_{\max}(R_{init}(a_{next})) > \dot{y}_{\max}(R_{goal}(a))
\end{aligned}
$$

Fig. 7.13 – Inequality constraints to ensure that the goal region of a control activity, a, is within the initial region of its successor, a_{next}.

Temporal Constraints

The second requirement stated in Definition 5.8 states that all temporal constraints in the QCP must be consistent. This includes the temporal constraints from the QSP, and the controllable duration bounds of all control activities.

The temporal constraints in the QCP restrict the execution windows of events in the QCP, as introduced in Section 6.1. Consider a QCP, such as the one shown in Fig. 6.2. If we set the execution window of the start event to [0, 0], then the execution windows of other events can be computed based on the temporal constraints, as shown in Fig. 6.8b.

In order for the QCP to be executable, all event execution windows must be feasible, that is, for each event, ev.

$$l(ev) \leq u(ev) \tag{7.10}$$

To complete this analysis, we must consider the relationship between the execution windows of activities ending on a common event. The lower bound of the execution window is the maximum lower bound consistent with all temporal constraints ending at the event. These include temporal constraints specified explicitly in the QSP, and temporal constraints due to activity dynamics (the controllable duration limits of activities). This is expressed in the following way, for each event, ev. Let CA_{ev} be the set of controllable activities ending at ev. Thus, given a control activity, ca, then $ca \in CA_{ev}$ if $ev_f(A(ca)) = ev$ (see Defs. 4.4 and 5.2). Also, let TC_{ev} be the set of QSP temporal constraints with finish event ev. Thus, given a temporal constraint, tc, then $tc \in TC_{ev}$ if $ev_2(tc) = ev$ (see Def. 4.5). Let $\max_{ca}$ be the maximum lower bound of the execution window due to activity controllable duration limits.

$$\max_{ca} = \max\{\forall ca \in CA_{ev}\ of\ (l(ev_s(A(ca))) + l(ca))\} \tag{7.11}$$

Let $\max_{tc}$ be the maximum lower bound of the execution window due to explicitly specified temporal constraints.

$$\max_{tc} = \max\{\forall tc \in TC_{ev} \text{ of } (l(ev_1(tc)) + l(tc))\} \quad (7.12)$$

Then, the lower bound of the execution window is

$$l(ev) = \max(\max_{ca}, \max_{tc}) \quad (7.13)$$

Similarly, the upper bound of the execution window is the minimum upper bound propagated by all temporal constraints ending at the event. Thus,

$$\min_{ca} = \min\{\forall ca \in CA_{ev} \text{ of } (u(ev_s(A(ca))) + u(ca))\} \quad (7.14)$$
$$\min_{tc} = \min\{\forall tc \in TC_{ev} \text{ of } (u(ev_1(tc)) + u(tc))\}$$
$$u(ev) = \min(\min_{ca}, \min_{tc})$$

Note that the constraints of Eqs. 7.13 and 7.14 are those propagated through by the dispatcher in the function PropagateExecutionWindow (Fig. 6.6).

For example, in the QCP of Fig. 6.2., CA_{ev} for the event right heel strike is the set of activities {CM_Fwd_2, CM_Lat_2, right foot step 1}. TC_{ev} for this event is empty. The execution window bounds on the right heel strike event are constrained by the execution window bounds of the event right toe off, and the duration bounds of the activities in CA_{ev}, according to Eqs. 7.11 and 7.12. For the event left heel strike, CA_{ev} is the set of activities {CM_Fwd_4, CM_Lat_4, left foot step 1}, and TC_{ev} consists of the temporal constraint with bounds [t_lb, t_ub], shown in Fig. 6.2.

Cost Function

Given that the above constraints are satisfied, we wish to maximize robustness by maximizing the initial regions and controllable durations of control activities in the QCP, as described in Definition 7.1. In our NLP formulation, we specify this desire to maximize these values through the cost function terms described in Sections 7.2.4 and 7.2.5. As discussed in Section 7.2.5, we use the cost function weighting factors, w_{ir} and

w_{cd} to resolve the trade-off between the competing goals of maximizing initial regions and maximizing controllable durations.

This concludes our summary of the NLP formulation that satisfies controllability requirements of Definition 5.8. The NLP is passed to an SQP optimizer, as described previously. The SQP optimizer produces values for the parameters to be optimized, which were listed above. All necessary parameters for control activities in the QCP are computed in this way. Examples of resulting initial and goal regions, and controllable durations, are provided in Chapter 9, for different walking speeds.

Satisfying the controllability requirements of Definition 5.8 is a key part of fulfilling the requirements for QCP compilation, as specified in Definitions 7.1 and 5.3. The remaining task is to produce a minimum dispatchable graph of the temporal constraints, as required by Definition 5.9 (see Def. 5.3). In the next Section, we discuss how the plan compiler takes the QCP computed from the NLP formulation, and generates the required minimal dispatchable graph. This graph is used by the dispatcher to simplify scheduling of events, as described in Chapter 6.

7.4.2 Satisfying Temporal Dispatchability Requirements

Definition 5.9 requires that the distance graph generated from the QCP's temporal constraints be in minimal dispatchable form [Muscettola, 1998]. Consider a QCP, qcp, whose parameters have been computed by the SQP optimizer, according to the NLP formulation described in Section 7.4.1. The temporal constraints of qcp are $TC(qcp)$, plus the set of duration bounds, $[l,u]$, of all control activities in qcp, where the temporal constraints, $TC(qcp)$, are explicitly stated in the QSP, and the duration bounds are computed according to the NLP formulation.

We use the algorithm described in Section 2.2.2 to convert the STN formed by the temporal constraints into an equivalent dispatchable graph. The three major steps for this algorithm are shown in Fig. 7.14.

ComputeDispatchableGraph(qcp)
// This computes the minimal dispatchable graph for the temporal
// constraints in qcp.
1. Compute distance graph.
2. Compute APSP graph using the Floyd-Warshall algorithm.
3. Remove redundant edges using triangle rules.

Fig. 7.14 – Steps for computing minimal dispatchable graph.

As discussed in Section 2.2.2, the following *triangle rule* [Muscettola, 1998a] is used to detect redundant edges. Given three events: A, B, and C,

(1) A non-negative edge AC is redundant if |AB| + |BC| = |AC|

(2) A negative edge AC is redundant if |AB| + |BC| = |AC|

We now consider an example to summarize how the algorithm in Fig. 7.14 computes a minimal dispatchable graph. In particular, we examine how this graph changes according to how demanding the QSP's explicitly specified temporal constraints are. Recall the example QCP of Fig. 6.2, shown again below in Fig. 7.15. We now use this example QCP in order to highlight the interaction between the temporal constraints explicitly specified in the QSP, and the duration bounds imposed by dynamic limitations. In particular, we show how this interaction affects the minimal dispatchable graph computed by the algorithm of Fig. 7.14.

The example QCP provides a walking plan for two steps; a right step, followed by a left step. The QCP has one explicitly specified temporal constraint, which specifies a time range, [t_lb, t_ub], during which these steps must be taken. Additionally, each of the activities in the QCP (CM_Fwd_1 and CM_Lat_1, for example) has duration bounds imposed by dynamic limitations. These duration bounds are the controllable durations of the flow tube approximations, as was discussed previously in Section 7.3.

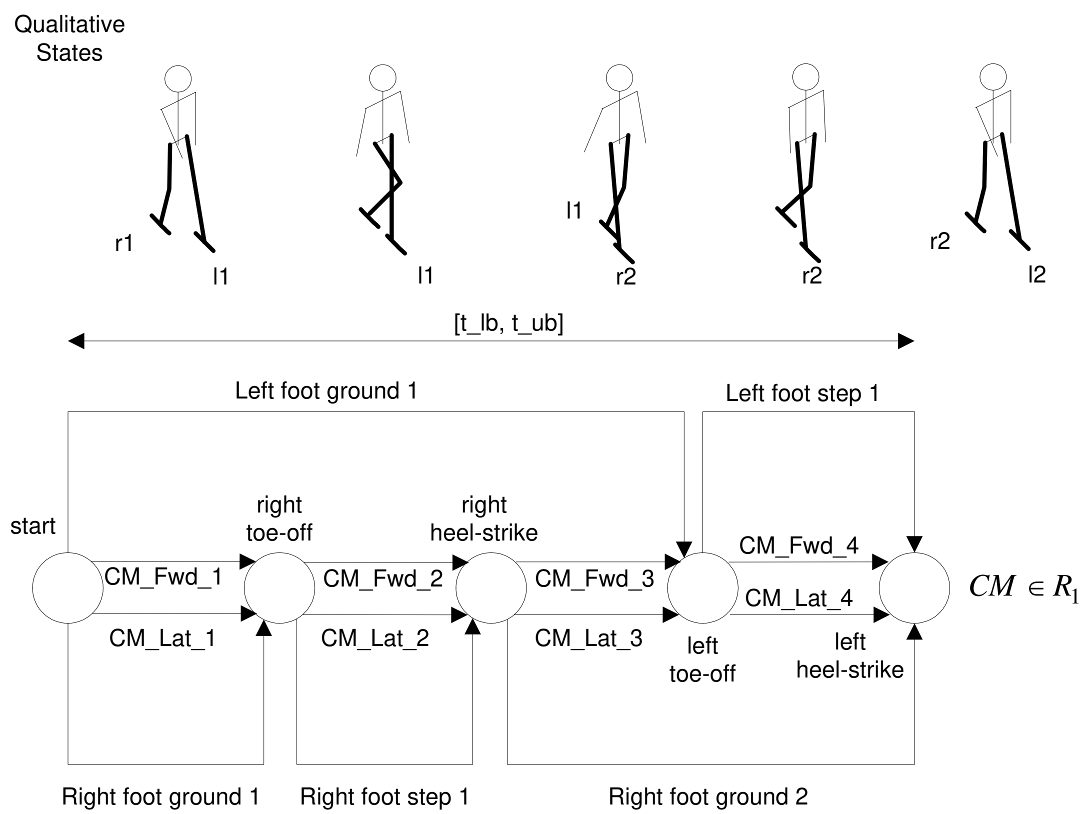


Fig. 7.15 – QCP from Fig. 6.2. Circles represent events, and horizontal arrows between events represent activities. Activities ending at the same event must be synchronized so that they finish at the same time. For example, the activities CM_Fwd_1, CM_Lat_1, and Right foot ground 1 all end at the event right toe-off. Therefore, these activities must finish at the same time.

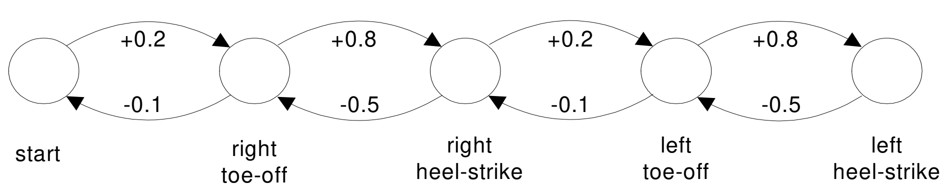


Fig. 7.16 – Minimal dispatchable graph (same as Fig. 6.8).

In order to understand how the minimal dispatchable graph is affected by the interaction between temporal constraints, we consider first the minimal dispatchable graph for the duration bounds only, and then analyze how the graph changes when the explicitly specified constraint is added. Suppose that the minimal dispatchable graph corresponding to the duration bounds only is as shown in Fig. 7.16. In the following discussion, we analyze the effect, on this minimal dispatchable graph, of adding the [t_lb, t_ub] explicitly specified temporal constraint.

Suppose, for example, that timing isn't critical for this walking task, and that the QSP's explicitly specified temporal constraint allows a broad temporal range for completion of the two steps; [1, 5], for example. If we add the arcs for this explicit QSP constraint to the minimal dispatchable graph of Fig. 7.16, we obtain the graph shown in Fig. 7.17. Note, however, that these arcs are not included in the APSP form of this graph. The path from start to left heel-strike going through each event has distance 2.0, which is shorter than the distance, 5.0, of the arc going directly from start to left heel-strike. Therefore, this direct arc is not part of the APSP graph. Similarly, the path from left heel-strike to start going through each event has distance –1.2, which is less than the distance, -1.0, of the direct arc from left heel-strike to start. Therefore, this direct arc is not part of the APSP graph. The Floyd-Warshall APSP algorithm, which is used as part of the algorithm of Fig. 7.14, will therefore not include these direct arcs in the APSP graph that it computes. Thus, in this case, the minimal dispatchable graph resulting from including the explicitly specified temporal constraint is exactly the same as the graph that results from omitting it. In both cases, the minimal dispatchable graph is as shown in Fig. 7.16. In this example, because the explicitly specified temporal constraint is not restrictive, the minimal dispatchable graph is determined solely by the duration bounds.

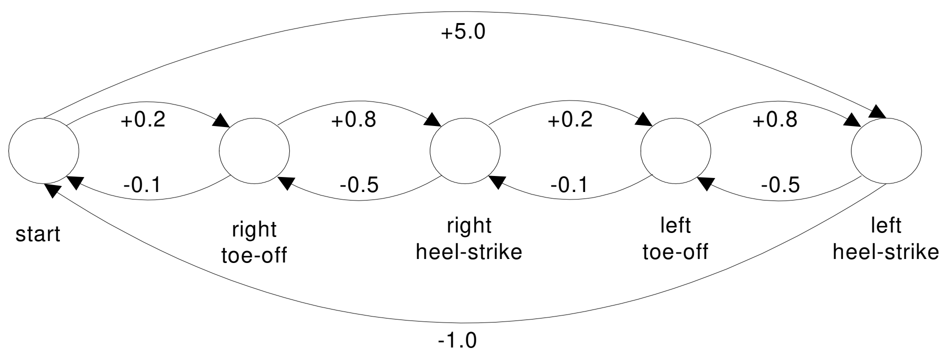


Fig. 7.17 – Minimal dispatchable graph, from Fig. 7.16, with addition of arcs for the explicitly specified temporal constraint.

Now, suppose that the QCP of Fig. 7.15 requires more precise timing. Suppose, for example, that the two steps are taken in order to kick a soccer ball, as in the example introduced in Chapter 1. The soccer ball is moving quickly, so the steps must be performed so as to move the biped to the goal location at a precise time. Therefore, suppose that the explicitly specified temporal constraint is [1.6, 1.6]. The minimal dispatchable graph is now as shown in Fig. 7.18. The direct arc from s (start) to lhs (left heel-strike) is necessary, because it has a distance of 1.6, which is less than 2.0, the distance from s to lhs going through each event. Further, the direct arc from s to lto (left toe-off) is also necessary, because it has a distance of 1.1, which is less than 1.2, the distance from s to lto going through events rto and rhs. The distance of 1.1 for this direct arc is achieved by traversing the arc of distance 1.6 from s to lhs, followed by the arc of –0.5 from lhs to lto. Similarly, the direct arcs from lhs to s and lhs to rto are also necessary, because they specify a minimum distance. The Floyd-Warshall algorithm therefore includes these arcs in the APSP graph that it computes.

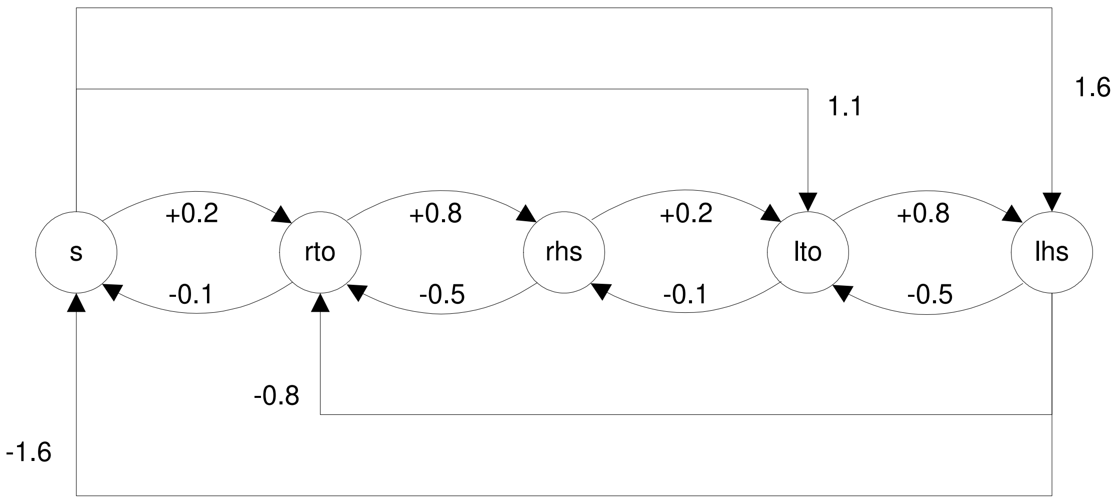


Fig. 7.18 – Minimal dispatchable graph for explicit temporal constraint of [1.6, 1.6].

The initial execution windows for lto and lhs, respectively, are now [0.8, 1.1], and [1.6, 1.6]. These are much tighter than the corresponding initial execution windows in Fig. 6.8b. Note that the execution windows for rto and rhs are the same in both graphs.

The examples of Fig. 7.17 and 7.18 illustrate the interaction between explicitly specified temporal constraints, and activity duration bounds due to dynamic limitations. When the explicitly specified temporal constraints are demanding, they squeeze the execution windows that are based only on the activity duration bounds. In such cases, these constraints must be included in the minimal dispatchable graph, as in Fig. 7.18. When the explicitly specified temporal constraints are less demanding, they can become redundant with the duration bounds due to dynamic limitations, and therefore, can be omitted from the minimal dispatchable graph, as in Fig. 7.17.

This concludes our discussion of the plan compiler. Example outputs of the plan compiler are presented in Chapter 9. This also concludes our discussion of the hybrid executive. Recall, from Chapter 1, that our model-based executive consists of two major components: the hybrid executive, and the dynamic virtual model controller (Fig. 1.14). Thus, having completed our discussion of the hybrid executive, we next present, in Chapter 8, a detailed description of the dynamic virtual model controller.

8 Dynamic Virtual Model Controller

This chapter presents the design of the dynamic virtual model controller, the component of the model-based executive that interacts directly with the biped, as shown in Fig. 1.14. The controller provides the linear virtual element abstraction, described in Section 4.3. This abstracted biped is easier to control than the actual one. Thus, the dynamic virtual model controller simplifies the job of the hybrid executive component of the model-based executive by providing this abstraction (see Fig. 1.14).

The primary purpose of the dynamic virtual model controller is to provide the hybrid executive with a simple way to control the biped's forward and lateral center of mass (CM) position. This allows the hybrid executive to maintain the system's balance, and to move the biped forward during walking, by specifying desired CM movement. Additionally, the controller must provide the hybrid executive with a simple way to control movement of the stepping foot, during walking, and to maintain the upright posture of the torso.

As introduced in Section 1.4.1, our goal is to provide an abstracted biped that is controlled by the hybrid executive, like a puppet, using virtual linear spring-damper elements. These virtual elements are attached at key reaction points, like the center of mass, and the stepping foot, as shown in Fig. 8.1. The hybrid executive can then assume that the motion of the reaction points will be linear, according to the virtual element parameters that it sets. This greatly simplifies the planning and control functions of the hybrid executive because it does not have to be concerned with the nonlinear dynamics of the actual biped, or with computing joint actuator torques. The hybrid executive lets the dynamic virtual model controller worry about these details.

The job of the dynamic virtual model controller is challenging because, while desired behavior is specified, by the hybrid executive, in terms of abstract variables like CM position, the actual biped must be controlled in terms of joint state variables like left knee joint position. Furthermore, computing joint torque control inputs for a multivariable, highly nonlinear, and tightly coupled system, such as the humanoid biped shown in Fig. 8.1 is challenging because the effect of these inputs on joint position and velocity state is a function of the complex nonlinear, coupled dynamics of the system.

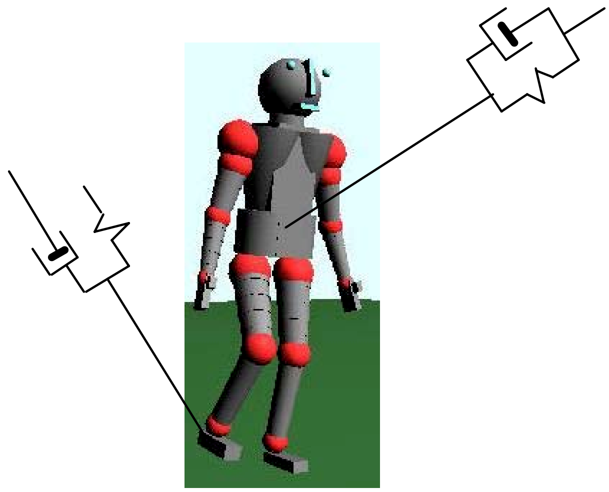


Fig. 8.1 – Virtual linear spring-damper elements, attached to reaction points, allow the mechanism to be controlled as if it were a puppet.

The virtual element control approach is in contrast to previous commonly used approaches [Hirai et al., 1997] in which detailed reference trajectories are generated for all the joints, and then high-impedance PD controllers are used to closely track these reference trajectories. These high-impedance control approaches have achieved walking, but they are not robust to significant disturbances, and they are not compliant, making them unsuitable for operation in unstructured environments, where unforeseen collisions may occur [Pratt and Tedrake, 2005]. The virtual element approach allows for low-impedance control of the reaction points. This provides compliant, robust control of quantities relevant to locomotion tasks, like CM position.

Our use of virtual elements is similar, in concept, to the one used in a virtual model controller [Pratt et al., 1997]. An important difference is that our dynamic virtual model controller takes dynamics into account, while a virtual model controller does not. A virtual model controller uses a Jacobian transformation to translate the desired forces at the reaction points, specified by the virtual elements, into joint torques that produce these forces. This works well for static or slow-moving mechanisms, but can break down as movements become faster because the controller does not take into account the dynamics of the system. Therefore, movement of the reaction point is not necessarily in line with the desired virtual force. In contrast, our dynamic virtual model controller uses a dynamic model to account for the biped's dynamics. This results in a linear system, where reaction points move as if they were simple linear second order systems, controlled by the virtual elements, as shown in Fig. 1.12.

In order to address the previously mentioned challenges related to computing joint torques for the nonlinear biped, our controller performs three key functions. First, it uses a model-based input-output linearization algorithm [Slotine and Li, 1991] to linearize the plant. Second, the controller decouples the plant state variables so that they appear to be independent. Third, the controller performs a geometric transform from joint space to workspace coordinates in order to make state variables relevant to balance control, such as center of mass position, directly controllable, as the state variables of a simple linear system.

Our controller is based on an input-output linearization approach, but is augmented with a Lagrangian relaxation technique to accommodate actuation constraints. This is important because the dynamic virtual model controller is a multivariable controller; it tries to achieve multiple goals simultaneously. Sometimes this is not possible; actuation constraints may cause the overall system to become over-constrained, in which case, some goals must be deferred. To address this problem, our controller incorporates a goal prioritization algorithm based on Lagrangian relaxation that automatically sacrifices lower-priority goals when the system becomes over-constrained in this way. For example, the system may temporarily sacrifice goals of maintaining upright posture in order to achieve balance goals.

Because the dynamic virtual model controller is model-based, the problem of model inaccuracy must be addressed. We compensate for this model error by incorporating a sliding control algorithm [Slotine and Li, 1991].

By incorporating the FRI constraint, in order to keep the stance feet from rolling (see Sections 3.2 – 3.4), and by utilizing its goal prioritization algorithm, our controller automatically generates angular momentum about the CM in order to enhance translational controllability of the CM, as described in Section 3.2. This allows the biped to recover balance without taking a step, as shown in Fig. 8.2, and 1.13. This is important when walking on difficult terrain, where foot placement is constrained.

Fig. 8.2 – Balance recovery from lateral disturbance using spin angular momentum.

The linearization and goal prioritization approach of our controller is similar, in concept, to the recently developed whole-body control algorithm [Khatib et al., 2004]. However, the whole-body controller relies heavily on an accurate model; it does not account for model inaccuracy. Our controller accounts for this inaccuracy using the sliding control approach. Furthermore, the whole-body control algorithms implemented thus far do not generate angular momentum to enhance balance control, as our controller does.

To summarize, the dynamic virtual model controller automatically coordinates movement of contact and non-contact segments in order to achieve linear behavior of the reaction points, as specified by the virtual element setpoints and gains. It generates these movements based only on the virtual element information, and the current state of the biped. In particular, it does not require a dynamic optimization that projects trajectory state over a future horizon, and it does not require use of pre-computed trajectories.

We test our controller with a morphologically realistic, 3-dimensional, 18 degree-of-freedom humanoid simulation, serving as the plant. This simulation is described in the next section. A detailed description of the multivariable controller derivation follows, in section 8.2. We conclude with test results, and a discussion of these results, in sections 8.3 and 8.4.

8.1 Detailed Humanoid Simulation

The controller was tested using a high-fidelity, humanoid simulation, serving as the plant to be controlled. The overall test configuration, showing controller and plant simulation, is given in Fig. 8.3.

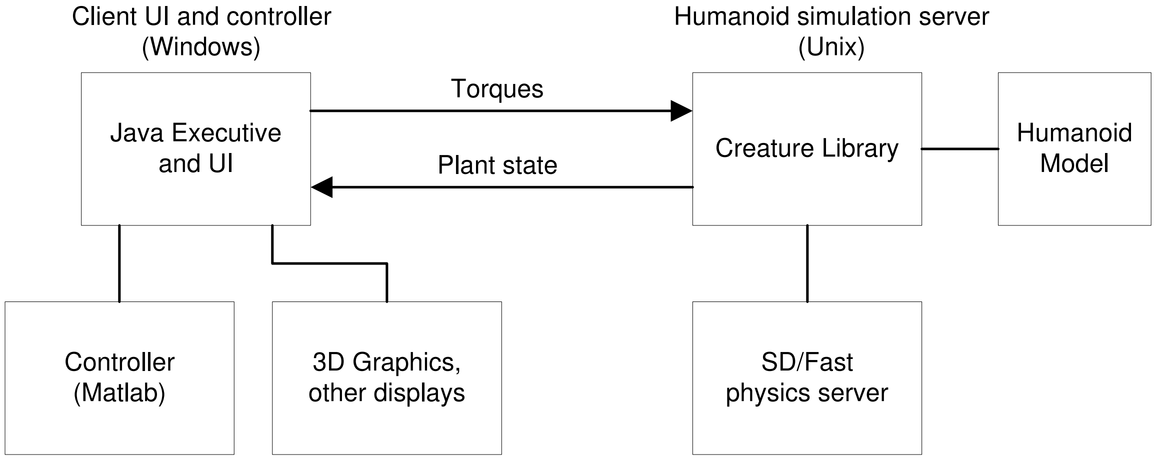


Fig. 8.3 – Test configuration

The humanoid model is compiled by Creature Library, a program developed at the MIT Leg Lab [Ringrose, 1997], into a form understandable to the physics server. Creature Library also computes ground reaction forces based on plant state, and passes these, along with input control torques, to the physics server. The physics server is based on a commercial product, SD/Fast [SD/Fast ref.]. The physics server computes dynamics of motion. In particular, it computes accelerations given control torques and ground reaction forces, and integrates these accelerations to update simulated plant state. The plant state is output via Creature Library. This simulation configuration has been validated extensively using robotic hardware, to ensure that it generates accurate motion trajectories. This configuration is a standard in that it is used at a number of labs involved with legged robot research, including the Leg Lab at MIT, the Robotics lab at CMU, and Boston Dynamics.

The Java executive interprets user commands to start and stop tests, and invokes the controller in order to compute updated control torques based on updated plant state received from the plant simulation. The Java executive uses a variety of displays, including 3D graphics, to show plant state.

The humanoid plant model, shown in Figure 8.4, is three-dimensional with 12 internal (controlled) and 6 external (un-controlled) degrees of freedom. The 6 external degrees of freedom correspond to the position and orientation of the trunk of the body. The 12 internal degrees of freedom correspond to joints (6 in each leg) that can exert torques. Each leg was modeled with a ball-and socket hip joint (3 degrees of freedom), a pin knee joint (one degree of freedom), and a saddle-type ankle joint (two degrees of freedom). Here the saddle joint architecture allows for ankle plantar/dorsiflexion motions and ankle inversion/eversion. The upper body (head, arms and torso), upper leg and lower leg were modeled with cylindrical shapes, and the feet were modeled with rectangular blocks.

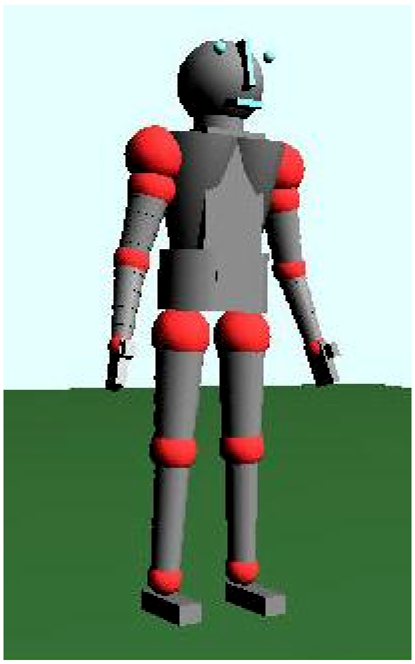


Fig. 8.4 – Humanoid model

The total mass was divided among the segments according to morphological data from the literature [Clauser et al., 1969; Brown, 1987]. The overall mass of the model was set equal to the mass of the test subject mentioned in the previous chapter (104 Kg), in order to allow for comparison with biological trajectory data. Mass proportions are listed in Table 8.1. The dimensions of each model segment were obtained by considering morphological data that describe average human proportions [Tilley and Dreyfuss, 1993; Winters, 1990], along with motion capture data, used to derive segment lengths, and finally, direct measurements on the test subject. Length parameters are listed in Table 8.2.

Body Segment	% of total mass	Total mass
Foot	1.5	1.56 kg
Lower leg	4.3	4.48 kg
Upper leg	10.3	10.73 kg
Upper body	67.8	70.65 kg

Table 8.1: Model segment masses and percentages of total body mass (104 Kg) are listed for the foot, leg and body of the model.

Upper body length	0.636 m
Upper body radius	0.183 m
Upper leg length	0.465 m
Upper leg radius	0.083 m
Lower leg length	0.480 m
Lower leg radius	0.053 m
Hip spacing	0.25 m

Table 8.2: Model segment lengths.

In the simulation, the ground was modeled using a nonlinear spring-damper system at four points per stance leg, located at each corner of the rectangular foot (see, also, Section 9.7 for a more detailed description of this ground contact model). Spring and damper coefficients were defined for x, y, and z directions, where x and y are horizontal directions, and z is vertical. Coefficients are listed in Table 8.3. Ground stiffness was first set so that the feet only penetrated the ground by a small amount in standing (~5 mm). Increasing damping from zero, then minimized oscillations between the ground and foot. The position of the contact points with respect to the ground were computed from the state variables. Thus, the application of the spring and damper constants to produce ground reaction forces on the contact points is a straightforward calculation, as described further in Section 9.7.

k_x	k_y	k_z	b_x	b_y	b_z
2,000,000 N/m	2,000,000 N/m	2,000,000 N/m	400 N/m/s	400 N/m/s	400 N/m/s

Table 8.3: Listed are ground stiffness and damping values in x, y and z directions.

8.2 Closed-Loop Control Rule Representation and Derivation

The linearization and decoupling provided by the dynamic virtual model controller transforms the tightly-coupled, nonlinear biped into a set of seemingly independent, linear, SISO (single input single output) systems. These systems form the linear virtual element abstraction, defined formally in Section 4.3. In this section, we derive the transformations used by the controller, and explain, in detail, how they are used, and how the controller works.

Recall, from Section 4.3, that a geometric transform, $\mathbf{h}$, is used to convert from the joint state to the workspace state representation, according to

$$\left[\mathbf{y}^T, \dot{\mathbf{y}}^T\right]^T = \mathbf{h}\left[\mathbf{x}^T, \dot{\mathbf{x}}^T\right]^T \qquad 8.1$$

where $\left\lfloor \mathbf{x}^T, \dot{\mathbf{x}}^T \right\rfloor$ is the joint state vector, and $\left\lfloor \mathbf{y}^T, \dot{\mathbf{y}}^T \right\rfloor$ is the workspace state vector. Elements of $\mathbf{x}$ include joint angle positions, such as left knee joint angle, and elements of $\mathbf{y}$ include forward and lateral CM position. The controller uses a feedback linearizing transformation to convert desired workspace variable accelerations, $\ddot{\mathbf{y}}$, into corresponding joint torques, $\boldsymbol{\tau}$, as shown in Fig. 8.5. Application of these torques results in a new joint state, $\left\lfloor \mathbf{x}^T, \dot{\mathbf{x}}^T \right\rfloor$. The multivariable controller then uses the transformation, $\mathbf{h}$, to convert from joint to workspace state.

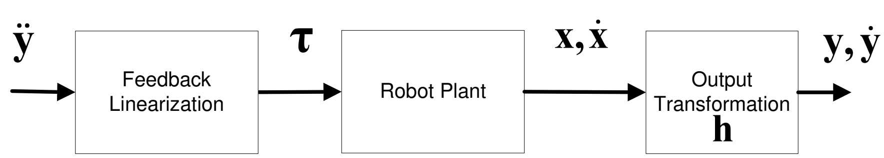


Fig. 8.5 – Feedback linearization and output transformation

If we draw a black box around the series of transforms in Fig. 8.5, the *multiple-input multiple-output* (MIMO) nonlinear plant appears to be a set of decoupled SISO linear 2nd-order systems, as shown in Fig. 8.6. Each element, $\mathbf{y}_i$ of position vector $\mathbf{y}$, can be viewed as the output of one of the SISO systems, with the corresponding acceleration element, $\ddot{\mathbf{y}}_i$, being the input. Each SISO system can be controlled by a simple linear control law, such as the proportional-differential (PD) law shown in Fig. 8.6. The set of SISO systems, with associated linear control laws, forms the linear virtual element abstraction. The solution trajectory for each SISO system is defined by a linear second-order differential equation, so the trajectory value at any time can be computed analytically.

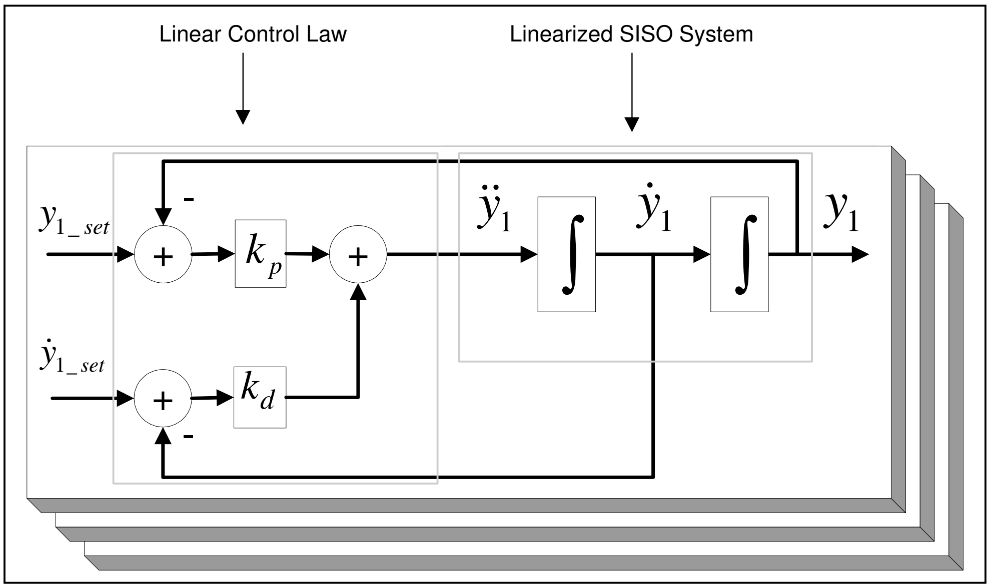


Fig. 8.6 – Linear virtual element abstraction consisting of a set of SISO systems with associated linear control laws.

We now derive the transformations used by the controller, and explain the detailed workings of the controller. Recall that the controller incorporates three key features: feedback linearization, goal prioritization, and sliding control. The feedback linearization component decouples and linearizes the dynamics of the plant, as shown in Fig. 8.5. The goal prioritization component uses an optimization algorithm to observe constraints such as joint ranges, maximum joint torques, and the restriction, essential to balance, that the foot rotation indicator (see Chapter 3) reside within the support polygon. Finally, the sliding control component compensates for modeling inaccuracies. These three components are now described in more detail, in the next three subsections, respectively.

8.2.1 Feedback Linearization of the Biped Plant

In order to describe the biped plant linearization, we begin by discussing the structure of the plant dynamics. We then discuss how these dynamics can be linearized, what the outputs of interest are, and how the overall system can be linearized with respect to these outputs.

Plant Dynamics

The dynamics for the plant are expressed in the following standard form [Craig, 1989]:

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} \qquad 8.2$$

where $\mathbf{q}$ is a vector of joint angles, $\boldsymbol{\tau}$ is a vector of joint torques, which are the control input to the plant, $\mathbf{H}(\mathbf{q})$ is a matrix of inertial terms, $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$ is a matrix of velocity-related terms, and $\mathbf{g}(\mathbf{q})$ is a vector of gravitational terms. Eq. 8.2 gives the plant inverse dynamics; it gives the control input, $\boldsymbol{\tau}$, that is needed to achieve a particular joint acceleration, $\ddot{\mathbf{q}}$, given a current joint state, $[\mathbf{q},\dot{\mathbf{q}}]^T$. Because $\mathbf{H}(\mathbf{q})$ is always invertible [Slotine and Li, 1991], the plant forward dynamics can always be obtained by multiplying both sides by $\mathbf{H}(\mathbf{q})^{-1}$ to get

$$\begin{aligned}\ddot{\mathbf{q}} &= \mathbf{H}(\mathbf{q})^{-1}(\boldsymbol{\tau} - \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) - \mathbf{g}(\mathbf{q})) \\ &= \mathbf{H}(\mathbf{q})^{-1}(-\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) - \mathbf{g}(\mathbf{q})) + \mathbf{H}(\mathbf{q})^{-1}\boldsymbol{\tau}\end{aligned} \qquad 8.3$$

The forward dynamics give joint acceleration for a particular control input, given a current joint state.

Linearization of Plant Dynamics

The forward dynamics (Eq. 8.3) are of the form

$$\ddot{\mathbf{q}} = \mathbf{f}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{b}(\mathbf{q},\dot{\mathbf{q}})\mathbf{u} \qquad 8.4$$

where

$$
\begin{aligned}
&\mathbf{u} = \boldsymbol{\tau} \\
&\mathbf{b} = \mathbf{H}(\mathbf{q})^{-1} \\
&\mathbf{f} = \mathbf{H}(\mathbf{q})^{-1}(-\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) - \mathbf{g}(\mathbf{q}))
\end{aligned}
\qquad 8.5
$$

This system can be perfectly linearized by the control input

$$\mathbf{u} = \mathbf{b}^{-1}(\mathbf{v} - \mathbf{f}) \qquad 8.6$$

where $\mathbf{v}$ is the desired value for $\ddot{\mathbf{q}}$, and $\mathbf{b}$ is invertible. Combining this with Eq. 8.4 yields the desired linearization.

$$\ddot{\mathbf{q}} = \mathbf{v} \qquad 8.7$$

Substituting values from Eq. 8.5 into 8.6 yields the control law

$$\mathbf{H}(\mathbf{q})\mathbf{v} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} \qquad 8.8$$

Thus, the system is exactly linearized, and completely decoupled into a set of SISO systems. This technique is sometimes called "computed torque", "inverse dynamics" or "feedforward" control in the robotics literature [Paul, 1981]. This linearization provides a system in "controllability" canonical form [Kailath, 1978].

$$[\dot{\mathbf{q}}, \ddot{\mathbf{q}}]^T = \mathbf{A}[\mathbf{q}, \dot{\mathbf{q}}]^T + \mathbf{B}\mathbf{v} \qquad 8.9$$

where

$$
\begin{aligned}
&\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \\
&\mathbf{B} = [0_1 \quad \ldots \quad 0_n \quad 1_1 \quad \ldots \quad 1_n]^T \\
&\mathbf{v} = \ddot{\mathbf{q}}_{des}
\end{aligned}
\qquad 8.10
$$

and $\mathbf{A}_{11}$, $\mathbf{A}_{21}$, $\mathbf{A}_{22}$ are n x n zero matrices, and $\mathbf{A}_{12}$ is an n x n identity matrix (n is the number of degrees of freedom).

This linearization is straightforward, due to the structure of the plant dynamics. However, the problem is not solved, because the goals specified in the qualitative state plan are not in terms of joint state, but rather outputs derived from plant state. These outputs, such as CM position, are nonlinear functions of plant state.

Plant Outputs and Input-Output Linearization

Plant outputs can be expressed as a nonlinear function of plant joint state, using the transform, $\mathbf{h}$, which was introduced previously.

$$\mathbf{y} = \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) \qquad 8.11$$

A linear mapping between workspace accelerations, $\ddot{\mathbf{y}}$, and joint accelerations, $\ddot{\mathbf{q}}$, is obtained by computing the second derivative of $\mathbf{y}$.

$$\ddot{\mathbf{y}} = \frac{\partial^2 \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}})}{\partial (\mathbf{q}, \dot{\mathbf{q}})^2} \qquad 8.12$$

resulting in an equation of the form

$$\ddot{\mathbf{y}} = \mathbf{\Psi}\ddot{\mathbf{q}} + \mathbf{\Psi}_{const} \qquad 8.13$$

It is assumed that this linear system can be solved for $\ddot{\mathbf{q}}$ given $\ddot{\mathbf{y}}$, at least for the region of state space in which the controller is operating.

Now, suppose we add a linear controller, like the one in Fig. 8.6, that computes a control input of the form

$$\ddot{\mathbf{y}}_{des} = \mathbf{f}_{controller}(\mathbf{y}_{des}, \mathbf{y}, \dot{\mathbf{y}}_{des}, \dot{\mathbf{y}}) \qquad 8.14$$

The mapping in Eq. 8.13 is then used to convert to desired joint accelerations:

$$\ddot{\mathbf{q}}_{des} = \mathbf{\Psi}^{-1}\left(\ddot{\mathbf{y}}_{des} - \mathbf{\Psi}_{const}\right) \qquad 8.15$$

These are then substituted into Eq. 8.6, with $\mathbf{v} = \ddot{\mathbf{q}}_{des}$, in order to get the desired control torques.

The result is a a two-stage linearization, where setpoints are specified in terms of the desired output variables, as shown in the Fig. 8.7. In this two-stage linearization, output variable accelerations are converted to joint accelerations by the first linearization, and then joint accelerations are converted to torques by the second, inverse dynamics, linearization.

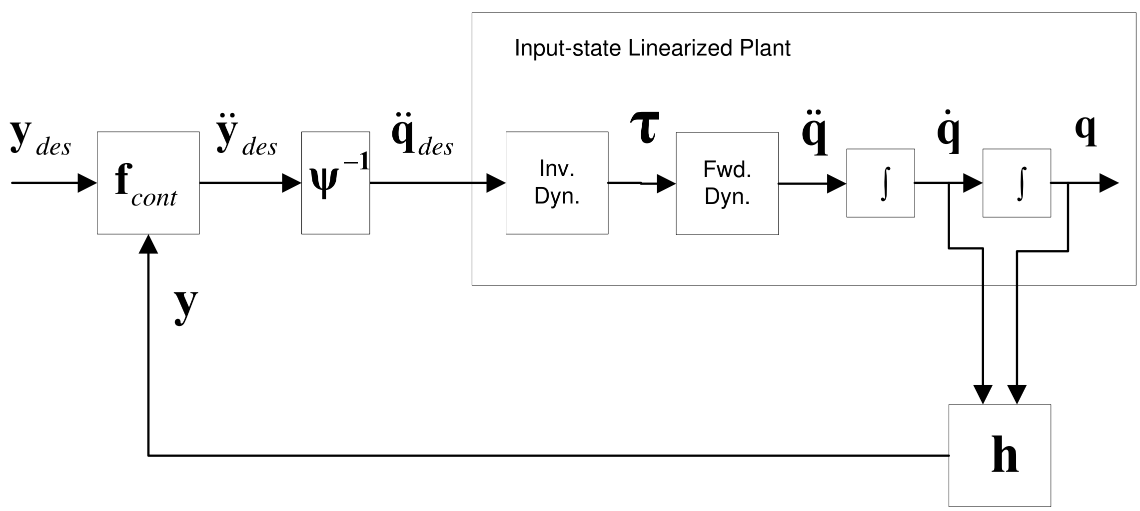


Fig. 8.7 - Two-stage Linearization

Computation of Input-Output Linearization

Computation of this linearization depends on the details of the **h** transformation. An important class of such transformations maps from plant joint state to specific reaction points on the mechanism, such as the CM position. This type of transformation is specified using homogeneous kinematic transforms (see Appendix A). They are functions of angle position only, not angular velocity. Thus, Eq. 8.11 simplifies to

$$\mathbf{y} = \mathbf{h}(\mathbf{q}) \qquad 8.16$$

Differentiating the first element of $\mathbf{y}$ yields

$$\dot{y}_1 = \frac{\partial h_1}{\partial(\mathbf{q})}\dot{\mathbf{q}} \qquad 8.17$$

Differentiating this again yields

$$\ddot{y}_1 = \frac{\partial\left[\frac{\partial h_1}{\partial \mathbf{q}}\dot{\mathbf{q}}\right]}{\partial \mathbf{q}}\dot{\mathbf{q}} = \frac{\partial[\dot{y}_1]}{\partial \mathbf{q}}\dot{\mathbf{q}} \qquad 8.18$$

$$= \sum_{i=1}^{n}\left(\frac{\partial h_1}{\partial q_i}\ddot{q}_i + \sum_{j=1}^{n}\frac{\partial^2 h_1}{\partial q_i \partial q_i}\dot{q}_i\dot{q}_j\right)$$

which is of the form of Eq. 8.13. The $\frac{\partial h_1}{\partial q_i}$ terms correspond to elements of the Jacobian, which are computed using the algorithm given in Appendix B. The $\frac{\partial^2 h_1}{\partial q_i^{\,2}}$ terms correspond to the Hessian. Computation of these terms is more complicated, and requires review of aspects of Jacobian computation.

To accomplish this, it is best to start with a simple example. Consider the planar two-link manipulator presented in Appendix B. The $\mathbf{h}$ transform for this is

$$\mathbf{h} = {}_0\mathbf{T}_2 = \mathbf{A}_1\mathbf{A}_2 \qquad 8.19$$

This homogeneous transform (see Appendix A) gives the position of the end point of the second link as a function of joint angles, as in Eq. 8.16. This end point of the second link is the end-effector of the manipulator, and is considered to be a reaction point in this example. Thus, the x-y-z position of this reaction point, in global coordinates, is given by

$$\mathbf{r}_{ef} = {}_{\mathbf{0}}\mathbf{T}_{\mathbf{2}} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad 8.20$$

Differentiating Eq. 8.20 with respect to the first joint angle yields a column of the Jacobian:

$$\frac{\partial \mathbf{r}_{ef}}{\partial \theta_1} = \frac{\partial \left({}_{\mathbf{0}}\mathbf{T}_{\mathbf{2}} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right)}{\partial \theta_1} = \frac{\partial \left({}_{\mathbf{0}}\mathbf{T}_{\mathbf{2}} \right)}{\partial \theta_1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad 8.21$$

Differentiating again yields the desired Hessian component:

$$\frac{\partial^2 \mathbf{r}_{ef}}{\partial \theta_1^{\,2}} = \frac{\partial^2 \left({}_{\mathbf{0}}\mathbf{T}_{\mathbf{2}} \right)}{\partial \theta_1^{\,2}} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad 8.22$$

The derivation in Appendix B provides a general way to compute Jacobians and Hessians. Eq. 33 in Appendix B is used to compute the column of the Jacobian corresponding to θ_1. Using eq. 16 of Appendix A, these values can be put into matrix form, representing a differential transform.

$$^{T_n}\Delta_1 = \begin{bmatrix} 0 & -^{T_n}\delta_{1z} & ^{T_n}\delta_{1y} & ^{T_n}d_{1x} \\ ^{T_n}\delta_{1z} & 0 & -^{T_n}\delta_{1x} & ^{T_n}d_{1y} \\ -^{T_n}\delta_{1y} & ^{T_n}\delta_{1x} & 0 & ^{T_n}d_{1z} \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad 8.23$$

Now, using Eq. 28 of Appendix B,

$$\frac{\partial\left({}_0\mathbf{T}_2\right)}{\partial\theta_1} = {}_0\mathbf{T}_2{}^{T_2}\Delta_1 \qquad 8.24$$

This is the Jacobian column for θ_1. Differentiating this yields an element of the Hessian:

$$\frac{\partial^2\left({}_0\mathbf{T}_2\right)}{\partial\theta_1{}^2} = \frac{\partial\left(\frac{\partial\left({}_0\mathbf{T}_2\right)}{\partial\theta_1}\right)}{\partial\theta_1} = \frac{\partial\left(\left({}_0\mathbf{T}_2\right)^{T_2}\Delta_1\right)}{\partial\theta_1} = \frac{\partial\left({}_0\mathbf{T}_2\right)}{\partial\theta_1}{}^{T_2}\Delta_1 = {}_0\mathbf{T}_2{}^{T_2}\Delta_1{}^{T_2}\Delta_1 \qquad 8.25$$

Similarly,

$$\frac{\partial^2\left({}_0\mathbf{T}_2\right)}{\partial\theta_1\partial\theta_2} = {}_0\mathbf{T}_2{}^{T_2}\Delta_1{}^{T_2}\Delta_2 \qquad 8.26$$

More generally, for any homogeneous transform, ${}_0\mathbf{T}_i$, the Hessian is computed by

$$\frac{\partial^2\left({}_0\mathbf{T}_i\right)}{\partial\theta_j\partial\theta_k} = {}_0\mathbf{T}_i{}^{T_i}\Delta_j{}^{T_i}\Delta_k \qquad 8.27$$

To summarize, for $\mathbf{h}$ transforms expressed using homogeneous transforms, Eqs. 8.24 and 8.27 provide a way to compute the corresponding Jacobian and Hessian matrices. These are used in the terms of Eq. 8.18, which is of the form of Eq. 8.13. This form is needed in order to accomplish the linearization, as described previously.

There are additional complexities related to computation of the rotational part of the Jacobians and Hessians. These complexities are discussed in Appendix C.

Plant Outputs for Humanoid Model

To accomplish the linearization for our humanoid model, we used $\mathbf{h}$ transforms expressed as homogeneous transforms, and chose the following outputs to be elements of the $\mathbf{y}$ vector.

- forward CM position
- lateral CM position
- stance knee joint angle
- torso roll angle
- torso pitch angle
- torso yaw angle
- forward swing foot position
- lateral swing foot position
- swing knee joint angle
- swing foot roll
- swing foot pitch
- swing hip joint yaw angle

The forward and lateral CM position are important variables to control for balancing, as explained in Chapter 3, so it makes sense to include these in the output vector. Similarly, swing foot placement determines the shape of the support polygon, and is therefore also crucial for balance control. Thus, it makes sense to include swing foot forward and lateral position in the output vector. It is desirable to maintain an upright torso position, so torso orientation should also be included in this vector. Note that vertical CM and swing foot position are not included in the output vector, in order to avoid singularities that may occur with these quantities. Instead, stance and swing knee joint angles are controlled, and vertical CM and swing foot positions emerge from these. This is done to avoid singularity problems, a well-known difficulty with feedback linearization control.

With this choice of outputs, the system given by Eq. 8.13 is square, because there are 12 inputs (the torques to the 12 joints), and there are 12 outputs. The output functions for all of these outputs are given by homogeneous kinematic transforms, which are functions of joint angles.

We now provide details of computation of the CM forward position output function and associated linearization. Computation of the other output functions is similar. In the subsequent discussion, the following notation will be used to represent transforms to reaction points:

$_{0}T_{RPi}$ - Transform from reaction point i to origin coordinates
$^{T_{RPi}}\Delta_{j}$ - Jacobian column differential transformation from joint j

The origin is at the base of the stance foot. The output function for forward (x direction) CM position is the average of the forward CM positions of each link in the mechanism, weighted by the corresponding link masses. There are 7 links in the mechanism: two feet, two lower legs, two upper legs, and one torso. Therefore, the first output vector element (forward CM position) is

$$y_1 = \frac{1}{m_{tot}} \sum_{i=1}^{7} m_i CM_{i\_x} \qquad 8.28$$

where m_i is the mass of link i, and CM_{i_x} is the forward position of its CM. The link CM points are reaction points, and are specified using homogeneous transforms, as in the two link manipulator example discussed previously. Jacobian and Hessian terms are computed as described previously.

Consider, for example, the torso CM reaction point. The x position is specified using the transform

$$x_{torso\_rp} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} {}_0T_{torso\_rp} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = h_{torso\_rp\_x}(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) \qquad 8.29$$

Note that torso position and orientation are a function of the first six joint angles of the stance leg: stance ankle roll and pitch, stance knee pitch, and stance hip pitch, roll, and yaw. Derivatives of this output are computed as described in the previous section. Jacobian and Hessian terms are

$$\frac{\partial h_{torso\_rp\_x}}{\partial \theta_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \frac{\partial \left({}_0T_{torso\_rp} \right)}{\partial \theta_i} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad 8.30$$

$$\frac{\partial^2 h_{torso\_rp\_x}}{\partial\theta_i\partial\theta_j} = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}\frac{\partial^2\left({}_0T_{torso\_rp}\right)}{\partial\theta_i\partial\theta_j}\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

where, from Eqs. 8.24 and 8.27,

$$\frac{\partial\left({}_0\mathbf{T}_{torso\_rp}\right)}{\partial\theta_i} = {}_0\mathbf{T}_{torso\_rp}\,{}^{T_{torso\_rp}}\Delta_i$$

$$\frac{\partial^2\left({}_0\mathbf{T}_{torso\_rp}\right)}{\partial\theta_i\partial\theta_j} = {}_0\mathbf{T}_{torso\_rp}\,{}^{T_{torso\_rp}}\Delta_i\,{}^{T_{torso\_rp}}\Delta_j$$

Jacobian and Hessian terms for other reaction point positions are computed in a similar similar way.

This concludes our discussion of the two-stage linearization shown in Fig. 8.7, and of its computation for the humanoid biped. In the next subsection, we discuss the goal prioritization component of the dynamic virtual model controller.

8.2.2 Multivariable Optimal Controller

The input-output linearization described in the previous section results in the overall system shown earlier in Fig. 8.7. To the outer controller, $\mathbf{f}_{cont}$, the rest of the system appears to be completely linearized and decoupled. Thus, simple control techniques for SISO linear systems, such as pole placement, can be used. The outer controller implements control laws of the form of Eq. 8.14.

Next, consider what happens when bounds on plant inputs due to saturation limits are added. In particular, consider bounds that ensure that the FRI (see Chapter 3) remains within the support polygon. This constraint is required to ensure that the stance feet remain flat on the ground and do not roll. If the controller does not take these bounds into consideration, it could generate values for $\ddot{\mathbf{y}}_{des}$ that cause the bounds to be violated.

To avoid this type of infeasibility, slack variables, $\ddot{\mathbf{y}}_{slack}$, are introduced for each element of $\ddot{\mathbf{y}}_{des}$, so that the new controller output, $\ddot{\mathbf{y}}_{cont\_out}$, is

$$\ddot{\mathbf{y}}_{des} = \ddot{\mathbf{y}}_{cont\_out} + \ddot{\mathbf{y}}_{slack} \qquad 8.31$$

Use of these slack variables provides flexibility in that $\ddot{\mathbf{y}}_{des}$ conforms to the controller's linear control law, without regard to the actuation bounds, while $\ddot{\mathbf{y}}_{cont\_out}$, the true output of the controller, does obey actuation bounds. The goal of the overall control system is then to minimize $\ddot{\mathbf{y}}_{slack}$, taking into account the relative importance of each element.

This minimization is accomplished by formulating the control problem as a quadratic program (QP), and then using a QP optimizer to solve it. The relative importance of the slack variables is expressed in the cost function for the QP. Slack variables associated with important outputs are given higher cost than slack variables for less important outputs. This causes the optimizer to prioritize goals by minimizing the slack variables for the most important outputs first, and therefore, setting $\ddot{\mathrm{y}}_{cont\_out}$ to be as close as possible to $\ddot{\mathrm{y}}_{des}$, in Eq. 8.31, for these outputs. For example, slack variables associated with the CM position output are given higher cost than those associated with torso orientation.

The variables in this formulation, and their associated constraints, are as follows.

$\ddot{\mathbf{y}}_{des}$ - desired output accelerations, determined by the controller $\mathbf{f}_{cont}$ as a function of current $\mathbf{y}$, $\dot{\mathbf{y}}$, and $\mathbf{y}_{des}$. These are a fixed input to the QP.

$\ddot{\mathbf{y}}_{slack}$ - output slack variables, minimized according to cost weightings.

$\ddot{\mathbf{y}}_{cont\_out}$ - the true controller output, satisfying all constraints. This is linearly constrained by Eq. 8.31.

$\ddot{\mathbf{q}}$ - joint accelerations, a linear function of $\ddot{\mathbf{y}}_{cont\_out}$ using $\mathbf{\Psi}^{-1}$. Note that $\mathbf{\Psi}$ is computed as a function of current state and is therefore fixed in this optimization. It is assumed to be invertible.

$\mathbf{RP\ddot{x}}$ - the reaction point acceleration in the x direction for each biped link. This is a linear function of $\mathbf{\Psi}_{RPx}$, which, like $\mathbf{\Psi}$, is computed from the Current state, and is therefore fixed in this optimization. $\mathbf{\Psi}_{RPx}$ is just like $\mathbf{\Psi}$ except that the output functions are the reaction point x positions of each link.

$\mathbf{RP\ddot{y}}$ - the reaction point acceleration in the y direction for each link, similar to $\mathbf{RP\ddot{x}}$.

$\mathbf{RP\ddot{z}}$ - the reaction point acceleration in z direction for each link, similar to $\mathbf{RP\ddot{x}}$.

$\dot{\boldsymbol{\omega}}_x$ - the x component of the angular acceleration of each link. This is a linear function of $\mathbf{\Psi}_{alpha}$, which, like $\mathbf{\Psi}$, is computed from current state, and is therefore fixed in this optimization.

$\dot{\boldsymbol{\omega}}_y$ - y component of angular acceleration of each link, similar to $\dot{\boldsymbol{\omega}}_x$

$\boldsymbol{\tau}$ - the joint torques for the biped plant. This is linear function of $\ddot{\mathbf{q}}_{des}$ using the inverse dynamics transformation given by Eq. 8.8. Note that this transformation is computed as a function of current state and is therefore fixed in this optimization.

The linear equality constraints between the variables are:

$\ddot{\mathbf{y}}_{des} = \ddot{\mathbf{y}}_{cont\_out} + \ddot{\mathbf{y}}_{slack}$ (from Eq. 8.31),

$\ddot{\mathbf{y}}_{cont\_out} = \boldsymbol{\Psi}\ddot{\mathbf{q}} + \boldsymbol{\Psi}_{const}$ (from Eq. 8.13),

$\mathbf{RP}\ddot{\mathbf{x}} = \boldsymbol{\Psi}_{RPx}\ddot{\mathbf{q}} + \boldsymbol{\Psi}_{RPx\_const}$ (similar to Eq. 8.13),

$\mathbf{RP}\ddot{\mathbf{y}} = \boldsymbol{\Psi}_{RPy}\ddot{\mathbf{q}} + \boldsymbol{\Psi}_{RPy\_const}$ (similar to Eq. 8.13),

$\mathbf{RP}\ddot{\mathbf{z}} = \boldsymbol{\Psi}_{RPz}\ddot{\mathbf{q}} + \boldsymbol{\Psi}_{RPz\_const}$ (similar to Eq. 8.13),

$\dot{\omega}_x = \boldsymbol{\Psi}_{alpha\_x}\ddot{\mathbf{q}} + \boldsymbol{\Psi}_{alpha\_x\_const}$ (similar to Eq. 8.13),

$\dot{\omega}_y = \boldsymbol{\Psi}_{alpha\_y}\ddot{\mathbf{q}} + \boldsymbol{\Psi}_{alpha\_y\_const}$ (similar to Eq. 8.13),

$\mathbf{H}(\mathbf{q})\mathbf{v} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}$ (similar to Eq. 8.8).

The reaction point and angular acceleration variables are necessary because these are terms in the FRI inequality constraint. This inequality constraint is very important; it ensures that the FRI remains within the support polygon, which, as described in Chapter 3, ensures that the support feet remain flat on the ground. The FRI point is given by the following equations (see Section 3.4).

$$FRI_x = \frac{\sum_{i=2}^{7} m_i RPx_i\left(RP\ddot{z}_i + g\right) - \sum_{i=2}^{7} m_i RPz_i RP\ddot{x}_i - \sum_{i=2}^{7} H\dot{y}_i}{\sum_{i=2}^{7} m_i\left(RP\ddot{z}_i + g\right)} \qquad 8.32$$

$$FRI_y = \frac{\sum_{i=2}^{7} m_i RPy_i\left(RP\ddot{z}_i + g\right) - \sum_{i=2}^{7} m_i RPz_i RP\ddot{y}_i + \sum_{i=2}^{7} H\dot{x}_i}{\sum_{i=2}^{7} m_i\left(RP\ddot{z}_i + g\right)}$$

$$H\dot{x}_i = I_{Gi}\,\dot{\omega}_{xi}$$
$$H\dot{y}_i = I_{Gi}\,\dot{\omega}_{yi}$$

Eq. 8.32 is transformed into a set of linear inequality constraints by replacing FRIx and FRIy with min and max terms, reflecting the bounds, so that these become constants:

$$FRI_{x\_max}\sum_{i=1}^{12} m_i g - \sum_{i=1}^{12} m_i RPx_i g \geq$$
$$-FRI_{x\_max}\sum_{i=1}^{12} m_i RP\ddot{z}_i + \sum_{i=1}^{12} m_i RPx_i RP\ddot{z}_i - \sum_{i=1}^{12} m_i RPz_i RP\ddot{x}_i - \sum_{i=1}^{12} I_{yGi}\dot{\omega}_{yi} \qquad 8.33$$

$$FRI_{x\_min}\sum_{i=1}^{12} m_i g - \sum_{i=1}^{12} m_i RPx_i g \leq$$
$$-FRI_{x\_min}\sum_{i=1}^{12} m_i RP\ddot{z}_i + \sum_{i=1}^{12} m_i RPx_i RP\ddot{z}_i - \sum_{i=1}^{12} m_i RPz_i RP\ddot{x}_i - \sum_{i=1}^{12} I_{yGi}\dot{\omega}_{y\_i}$$

$$FRI_{y\_max}\sum_{i=1}^{12} m_i g - \sum_{i=1}^{12} m_i RPy_i g \geq$$
$$-FRI_{y\_max}\sum_{i=1}^{12} m_i RP\ddot{z}_i + \sum_{i=1}^{12} m_i RPy_i RP\ddot{z}_i - \sum_{i=1}^{12} m_i RPz_i RP\ddot{y}_i + \sum_{i=1}^{12} I_{xGi}\dot{\omega}_{x\_i}$$
$$FRI_{y\_min}\sum_{i=1}^{12} m_i g - \sum_{i=1}^{12} m_i RPy_i g \leq$$
$$-FRI_{y\_min}\sum_{i=1}^{12} m_i RP\ddot{z}_i + \sum_{i=1}^{12} m_i RPy_i RP\ddot{z}_i - \sum_{i=1}^{12} m_i RPz_i RP\ddot{y}_i + \sum_{i=1}^{12} I_{xGi}\dot{\omega}_{x\_i}$$

The left sides of these inequalities are all constants with respect to the optimization formulation. The right sides are all linear in the variables being optimized.

This QP formulation is solved using a QP optimizer, in order to compute the torque vector, $\boldsymbol{\tau}$, that minimizes the slacks, and therefore, achieves the most important goals.

8.2.3 Sliding Control Framework

Feedback linearization is a powerful technique for computing control actions for systems with nonlinear dynamics, but it can be insufficient for real plants because it assumes a perfect plant model. The sliding control algorithm, described in Appendix D, addresses this problem using a two-part structure. The first part is the nominal part; it assumes the model is perfect, and issues control commands using a feedback linearization based on this model. For this part, we employ the linearization and goal prioritization components described in Sections 8.2.1 and 8.2.2. The second part contains additional corrective control terms used to compensate for model inaccuracy.

We now discuss how we incorporate this second part of sliding control into the dynamic virtual model controller. For our controller, the nominal or feed-forward control input to the plant is $\boldsymbol{\tau}$, which is the joint torque vector output by the inverse dynamics block in Fig. 8.7. The corrective control terms are feedback torques, $\boldsymbol{\tau}_{fb}$, which are combined with the feed-forward torques to get the new, combined plant input torque $\boldsymbol{\tau}_{plant}$.

$$\boldsymbol{\tau}_{plant} = \boldsymbol{\tau} + \boldsymbol{\tau}_{fb} \tag{8.34}$$

Note that the corrective control terms must be applied directly to the torques, the actual inputs to the plant, in order to bypass the kinematic and inverse dynamics models, and any associated inaccuracies in these models (see Fig. 8.7). For this study, the inverse dynamics block in Fig. 8.7 used a slightly simplified model compared with the one used in the forward dynamics plant simulation, hence some model inaccuracies were introduced, just as would be the case with an actual plant.

As discussed in Appendix D, the corrective control terms are of the form

$$\boldsymbol{\tau}_{fb} = -\boldsymbol{\lambda}\dot{\tilde{\mathbf{q}}} - \mathbf{k}\,\mathrm{sgn}(\mathbf{s}) \tag{8.35}$$

where $\tilde{\mathbf{q}}$ is the tracking error, defined as the difference between the actual and nominal joint angles

$$\tilde{\mathbf{q}} = \mathbf{q} - \mathbf{q}_{nom} \tag{8.36}$$

and $\mathbf{q}_{nom}$ is computed by integrating $\ddot{\mathbf{q}}_{des}$ in Fig. 8.7. The constants in the diagonal matrix $\boldsymbol{\lambda}$ control convergence, while on the sliding surface (see Appendix D). The vector $\mathbf{s}$ is the distance from the sliding surface, defined as

$$\mathbf{s} = \dot{\tilde{\mathbf{q}}} + \boldsymbol{\lambda}\tilde{\mathbf{q}} \tag{8.37}$$

The constants in the diagonal matrix $\mathbf{k}$ are made large enough to account for model uncertainty [Slotine and Li, 1991].

Fig. 8.8 shows the overall control architecture, including the sliding controller feedback terms.

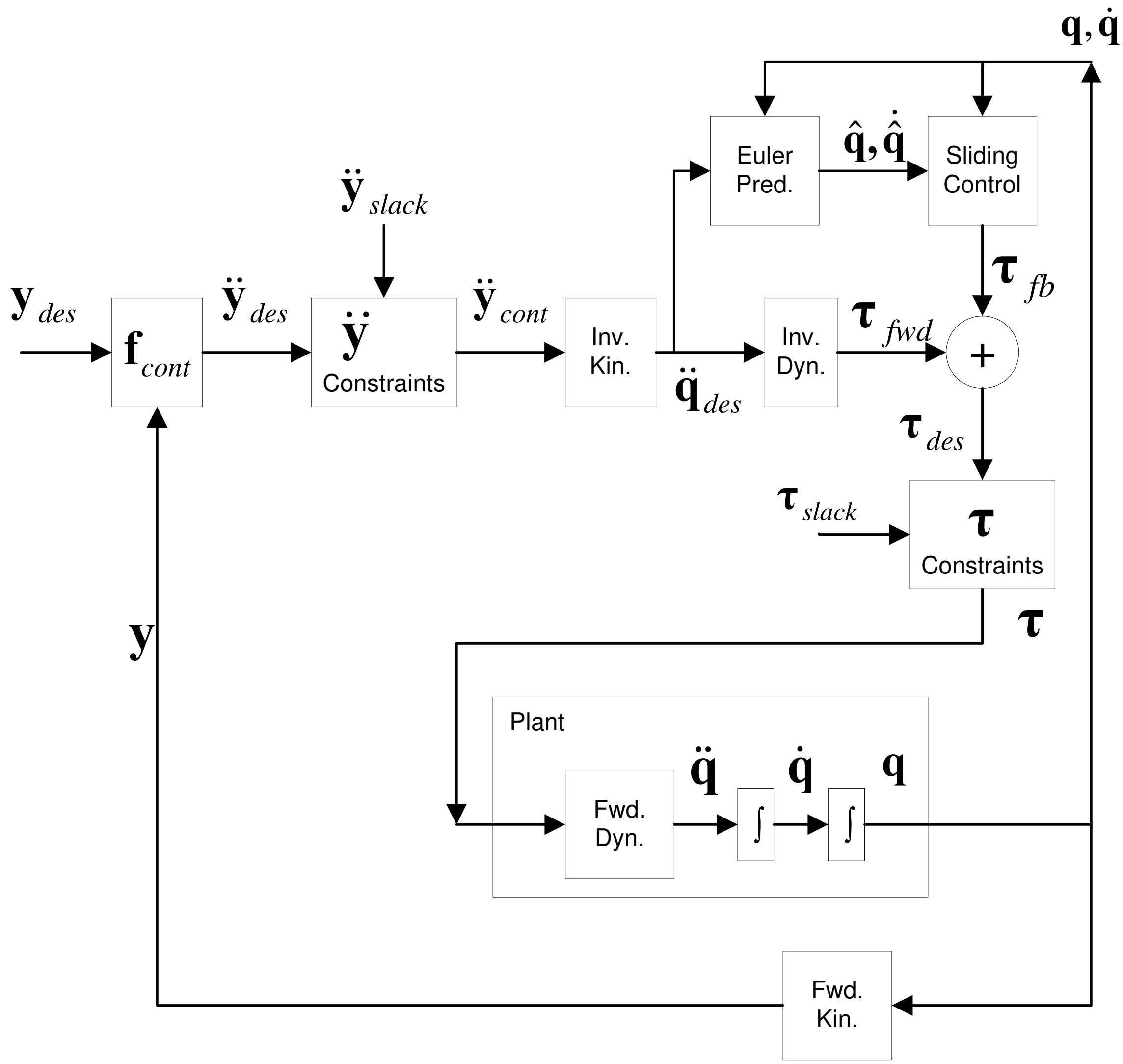


Fig. 8.8 – Overall controller architecture including sliding controller

The major extension for sliding conrol, from Fig. 8.7, is the feedback torque computation mechanism, which is based on the above described sliding control law.

The sections of the controller leading to the computation of joint acceleration vector, $\ddot{\mathbf{q}}_{des}$, are the same as before. This vector is then used to compute a prediction for joint state:

$$\dot{\hat{\mathbf{q}}}(k+1) = \dot{\mathbf{q}}(k) + \ddot{\mathbf{q}}_{des}\,\Delta t$$
$$\hat{\mathbf{q}}(k+1) = \mathbf{q}(k) + \dot{\hat{\mathbf{q}}}(k+1)\Delta t \qquad 8.38$$

, where k is the time increment index, and Δt is the time increment. Use of a simple Euler integration here is justified, since Δt is relatively small (0.01 sec) compared with the overall system dynamics, and since the prediction is reset periodically to conform to the actual value from the biped simulation. The tracking error terms are then computed from these predictions.

$$\tilde{\mathbf{q}} = \mathbf{q} - \hat{\mathbf{q}} \qquad (8.2.42)$$
$$\dot{\tilde{\mathbf{q}}} = \mathbf{q} - \dot{\hat{\mathbf{q}}}$$

These terms are then used in the sliding control law of Eq. 8.35 to compute the feedback torque.

As shown in Fig. 8.8, the desired torque is then

$$\boldsymbol{\tau}_{des} = \boldsymbol{\tau}_{fwd} + \boldsymbol{\tau}_{fb}$$

Now, the torque that is input to the plant must be such that the ZMP remains within the support polygon. Thus, the ZMP constraint must be asserted. Not that this requires knowledge of ground reaction force, which is computed from inverse dynamics. Slack variables, which are minimized in the optimization formulation, are used to allow for the discrepancy between desired and actual torques:

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{des} + \boldsymbol{\tau}_{slack}$$

The full optimization formulation, including the sliding control component, is now summarized. The variables in the formulation are:

$\ddot{\mathbf{y}}_{des}$ - desired output accelerations,
$\ddot{\mathbf{y}}_{slack}$ - output acceleration slack variables,

$\ddot{\mathbf{y}}_{cont}$ - output acceleration control output,
$\ddot{\mathbf{q}}_{des}$ - joint accelerations,
$\ddot{\mathbf{x}}_{RPi}$ - x-direction acceleration of reaction point i,
$\ddot{\mathbf{y}}_{RPi}$ - y-direction acceleration of reaction point i,
$\ddot{\mathbf{z}}_{RPi}$ - z-direction acceleration of reaction point i,
$\dot{\boldsymbol{\omega}}_{RPi}$ - angular acceleration of reaction point i,
$\boldsymbol{\tau}_{fwd}$ - feedforward (computed) torque,
$\boldsymbol{\tau}_{des}$ - desired torque,
$\boldsymbol{\tau}_{slack}$ - torque slacks,
$\boldsymbol{\tau}$ - actual torque input to plant,
$\mathbf{F}_{COM}$ - Force on CM (also known as ground reaction force).

Equality constraints are:

$$\ddot{\mathbf{y}}_{cont} + \ddot{\mathbf{y}}_{slack} = \ddot{\mathbf{y}}_{des}$$

where $\ddot{\mathbf{y}}_{des}$ is computed outside the optimization based on current state and a PD control law,

$$\ddot{\mathbf{y}}_{cont} - \boldsymbol{\Psi}\ddot{\mathbf{q}}_{des} = \boldsymbol{\Psi}_{const}$$

where $\boldsymbol{\Psi}, \boldsymbol{\Psi}_{const}$ are computed as part of the kinematics computations,

$$\begin{bmatrix} \ddot{\mathbf{x}}_{RPi} \\ \ddot{\mathbf{y}}_{RPi} \\ \ddot{\mathbf{z}}_{RPi} \end{bmatrix} - \boldsymbol{\Psi}_{RPi}\ddot{\mathbf{q}}_{des} = \boldsymbol{\Psi}_{RPi\_const}$$

where $\boldsymbol{\Psi}_{RPi}, \boldsymbol{\Psi}_{RPi\_const}$ are computed as part of the kinematics computation,

$$\dot{\boldsymbol{\omega}}_{RPi} - \boldsymbol{\Psi}_{RPi\_alpha}\ddot{\mathbf{q}}_{des} = \boldsymbol{\Psi}_{RPi\_alpha\_const}$$

where $\boldsymbol{\Psi}_{RPi\_alpha}, \boldsymbol{\Psi}_{RPi\_alpha\_const}$ are computed as part of the kinematics computations,

$$\mathbf{H}\ddot{\mathbf{q}}_{des} - \boldsymbol{\tau}_{fwd} = -\mathbf{C}$$

where $\mathbf{H}, \mathbf{C}$ are computed by the inverse dynamics algorithms

$$\boldsymbol{\tau}_{des} = \boldsymbol{\tau}_{fwd} + \boldsymbol{\tau}_{fb}$$

where $\boldsymbol{\tau}_{fb}$ is computed outside the optimization according to the above equations

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{des} + \boldsymbol{\tau}_{slack}$$

As with the other slack variables, $\boldsymbol{\tau}_{slack}$ is penalized in the cost function so as to minimize it. The ground reaction force is given by the equality constraint

$$\mathbf{F}_{CM} = \sum_i m_i \begin{bmatrix} \ddot{x}_{RPi} \\ \ddot{y}_{RPi} \\ \ddot{z}_{RPi} \end{bmatrix}$$

Inequality constraints in the formulation are: the FRI inequality constraints, described previously (Eq. 8.33), and stance ankle torque limits to keep the ZMP within support polygon bounds. The stance ankle torque limits are:

$$-\tau_{ap} + foot\_height\, F_{COM\_x} - foot\_length\_back\, F_{COM\_z} \leq 0$$

$$\tau_{ap} - foot\_height\, F_{COM\_x} - foot\_length\_front\, F_{COM\_z} \leq 0$$

$$\tau_{ar} + foot\_height\, F_{COM\_y} - foot\_half\_width\, F_{COM\_z} \leq 0$$

$$-\tau_{ar} - foot\_height\, F_{COM\_y} - foot\_half\_width\, F_{COM\_z} \leq 0$$

where

τ_{ap} is ankle pitch torque,
τ_{ar} is ankle roll torque,
foot_height is distance from the ankle straight down to the bottom of the foot
foot_half_width is half the width of the foot
foot_length_front is the distance from the ankle, projected to the ground, to the front of the foot
foot_length_back is the distance from the ankle, projected to the ground, to the back of the foot

This concludes our description of the three components of the dynamic virtual model controller: feedback linearization, goal prioritization, and sliding control. The formulation provided in this section produces a control torque vector that is input to the biped's joints, in order to achieve, as closely as possible, the desired acceleration vector, $\ddot{\mathbf{y}}_{des}$. The next section presents test results for this controller.

8.3 Results

The ability to balance on one leg is an important prerequisite for walking, expecially when foot placement is constrained. Therefore, a series of experiments was performed to evaluate the controller's ability to stabilize the biped in single-support mode, that is, standing on one leg.

Balance recovery was tested by initializing the biped in a motionless position, but with the horizontal position of the center of mass (CM) outside the support polygon, defined by the stance foot. For such an initial condition, stance ankle torque alone is insufficient for restoring balance. The maximum stance ankle torque that can be exerted without having the foot roll places the FRI point (see Chapter 3) at the edge of the support polygon, but not beyond it. Since the CM is beyond this point, this is insufficient for generating an appropriate corrective horizontal component of the ground reaction force, as explained in Chapter 3. The biped is sufficiently out of balance that it becomes necessary to perform dynamic movement of non-contact segments in order to generate spin torque about the CM. As explained in Chapter 3, this augments the horizontal ground reaction force provided by the stance ankle torque, by moving the CMP outside the edge of the support polygon. This action can help the system restore balance, by bringing the horizontal position of the CM back to the center of the support polygon, but it also causes a disturbance in the angular stability (upright posture) of the biped. The controller, therefore, must judiciously sacrifice angular stability temporarily, in order to bring the CM back under control, after which, it corrects for the angular disturbance.

The initial condition used here for testing results in an instability that is similar to the one that occurs when the system is pushed near its CM. Thus, it is a good indicator of how the system will perform when disturbed in this way.

Experimental results are now presented for four cases. The first two are for a forward and lateral disturbance, with the biped standing on one leg on level ground. The second two involve similar disturbances, but in these tests, the biped is standing on a narrow podium that further restricts the support polygon. This is similar to the situation encountered when balancing on a tightrope or balance beam.

8.3.1 Forward Disturbance on Level Ground

Fig. 8.9 shows the system's response to a forward disturbance, that is, where the forward position of the CM starts beyond the forward limit of the support polygon, while standing on level ground. The counter-clockwise rotation of the upper body and right (non-stance) leg results in spin angular momentum about the CM. By conservation of angular momentum, this results in an orbital angular momentum of the CM about the support point, which augments the angular momentum produced by stance ankle torque. This pushes the CM backwards. Fig. 8.10 shows the FRI (in blue), and the CMP (in red) during this maneuver.

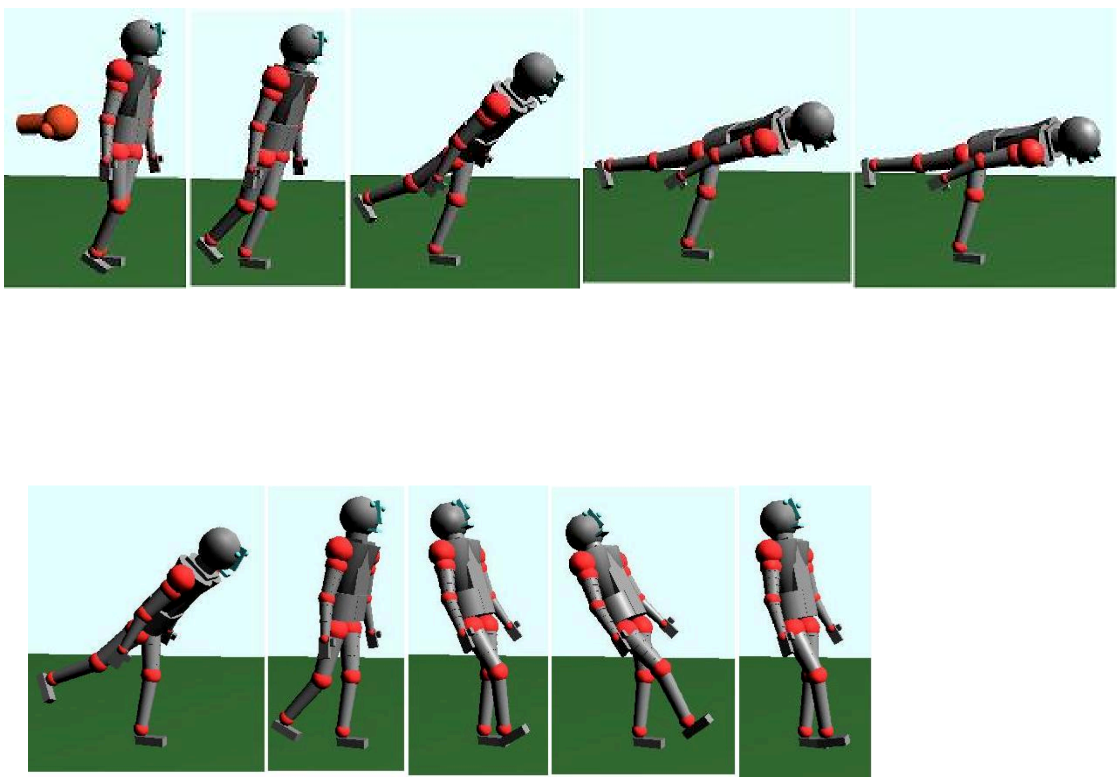


Fig. 8.9 – Motion sequence of biped for forward disturbance while standing on left leg on level ground

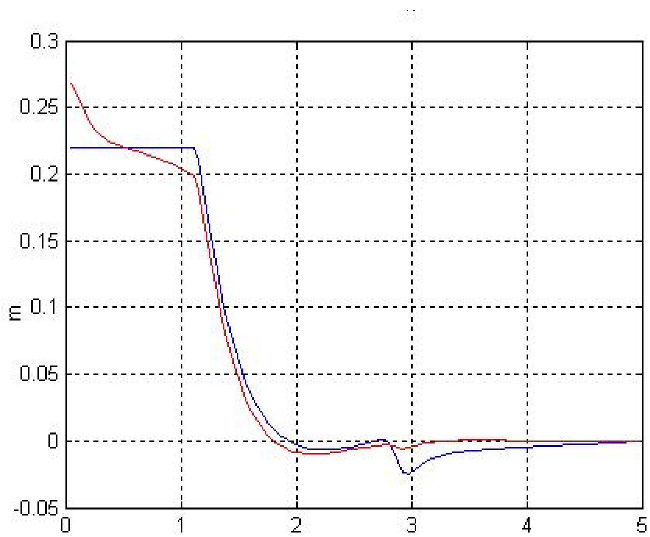


Fig. 8.10 – Forward direction FRI in blue, CMP in red

The forward limit of the support base is at 0.22 meters. As can be seen from Fig. 8.10, the FRI stays within this limit, so that the foot does not roll, but the CMP starts outside of it. The angular movement of the upper body and non-stance leg is what causes the CMP to leave the bounds of the support base. This provides enough equivalent horizontal ground reaction force to bring the horizontal position of the CM back to the center of the support base. After the CM is under control, the system restores its upright position. This is indicated in the motion sequence in Fig. 8.9, and also, in the plot in Fig. 8.10. Note that after about 1 second, the FRI is no longer pegged at the limit. This is an indication that the CM is under control, and that the controller can now turn its attention to correcting the angular disturbance.

8.3.2 Lateral Disturbance on Level Ground

Fig. 8.11 shows the system's response to a lateral disturbance, that is, where the lateral position of the CM starts beyond the left-most limit of the system's support polygon, while standing on level ground. Similarly to the forward disturbance case, the counter-clockwise rotation of the upper body and right (non-stance) leg results in spin angular momentum about the CM. By conservation of angular momentum, this results in an orbital angular momentum of the CM about the support point, which augments the angular momentum produced by stance ankle torque. This pushes the CM toward the system's right.

Fig. 8.12 shows the FRI (in blue), and the CMP (in red) during this maneuver. The left-most limit of the support base is at 0.05 meters. As can be seen from Fig. 8.3.4, the FRI stays within this limit, so that the foot does not roll, but the CMP starts outside of it. Similarly to the forward disturbance case, the angular movement of the upper body and non-stance leg provides enough equivalent horizontal ground reaction force to bring the horizontal position of the CM back to the center of the support base. Note that after about 0.8 seconds, the ZMP is no longer pegged at the limit. This is an indication that the CM is under control, and that the controller can turn its attention to correcting the angular disturbance.

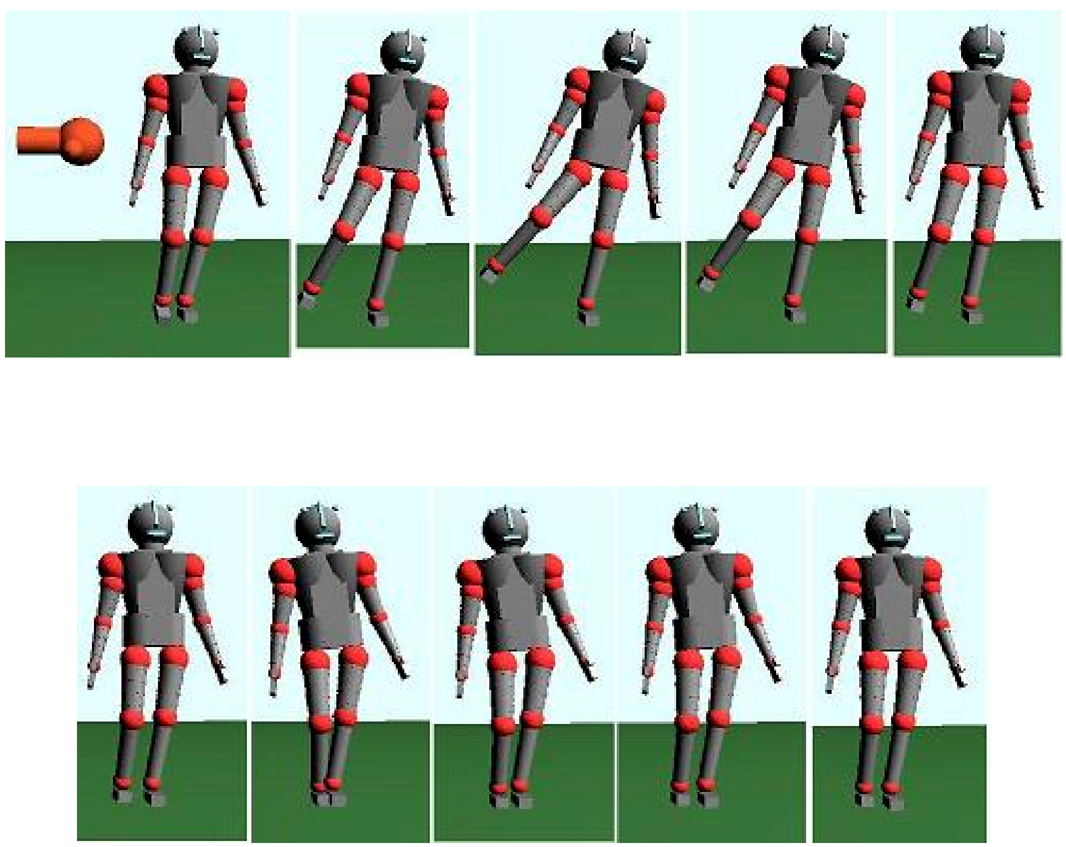


Fig. 8.11 – Motion sequence of biped for lateral disturbance while standing on left leg

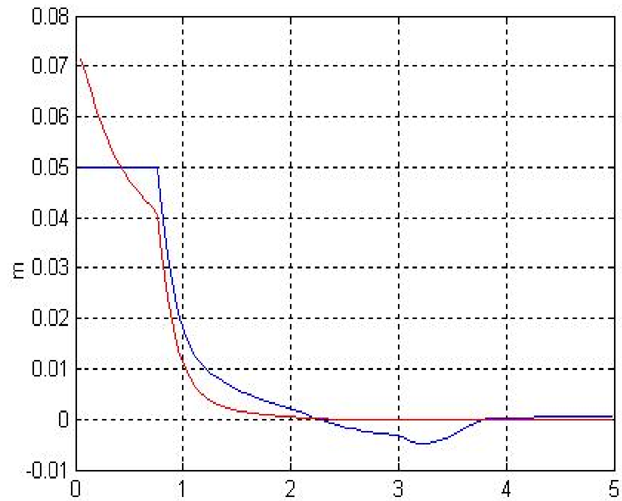


Fig. 8.12 – Lateral (leftward) direction ZMP in blue, CMP in red

8.3.3 Forward Disturbance on Podium

Fig. 8.13 shows the system's response to a forward disturbance while standing on a podium with limited area.

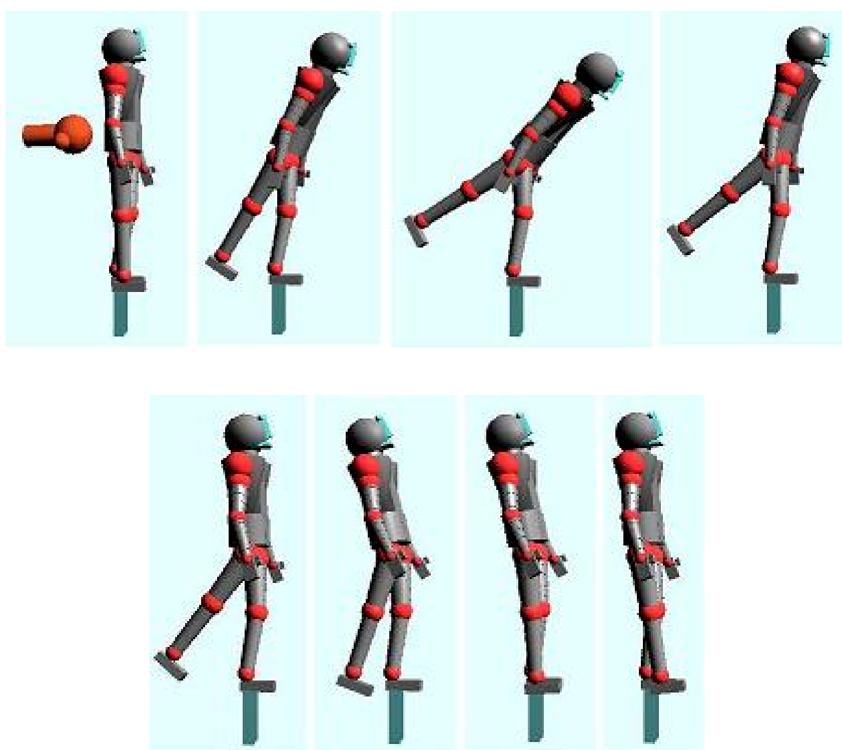


Fig. 8.13 - Motion sequence of biped for forward disturbance while standing on a podium

In this test, the podium is only 10 cm wide, which is much less than the 22 cm length of the foot. Therefore, the polygon of support is much reduced compared with the level ground test in Section 8.3.1. As with the level ground case, the rotation of the upper body and right (non-stance) leg results in spin angular momentum that helps push the CM backwards.

Because the support base is limited, the role of angular momentum-induced restorative force, relative to the stance ankle torque-induced restorative force, is more significant than for the case where the biped is standing on level ground. This is indicated in Fig. 8.14. The forward limit of the support base is at 0.05 m, but the CMP starts at 0.08. As a percentage of support base size, this is a much more significant difference than in Fig. 8.10. This indicates that the force provided by stance ankle torque, when standing on the podium, is proportionately smaller than for the level ground case.

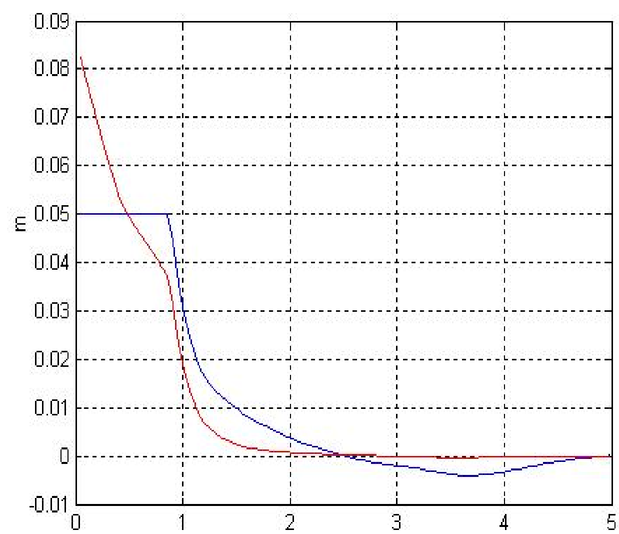


Fig. 8.14 – Forward direction ZMP in blue, CMP in red

8.3.4 Lateral Disturbance on Podium

Fig. 8.15 shows the system's response to a lateral disturbance, while standing on a podium with limited area. In this test, the podium is only 2.5 cm wide, which is much less than the 10 cm width of the foot. Therefore, the polygon of support is much reduced compared with the level ground test in Section 8.3.2. As with the level ground case, the rotation of the upper body and right (non-stance) leg results in spin angular momentum that helps push the CM to the biped's right. As with the case of the forward disturbance on the podium, because the support base is limited, due to the limited area of the podium, the role of angular momentum-induced restorative force, relative to the stance ankle torque-induced restorative force, is more significant than for the case where the biped is standing on level ground. This is indicated in Fig. 8.16. The left-most limit of the support base is at 0.0125 m, but the CMP starts at 0.025, more than twice as far away. As a percentage of support base size, this is a much more significant difference than in Fig. 8.3.4. This indicates that the force provided by stance ankle torque, when standing on the podium, is proportionately smaller than for the level ground case.

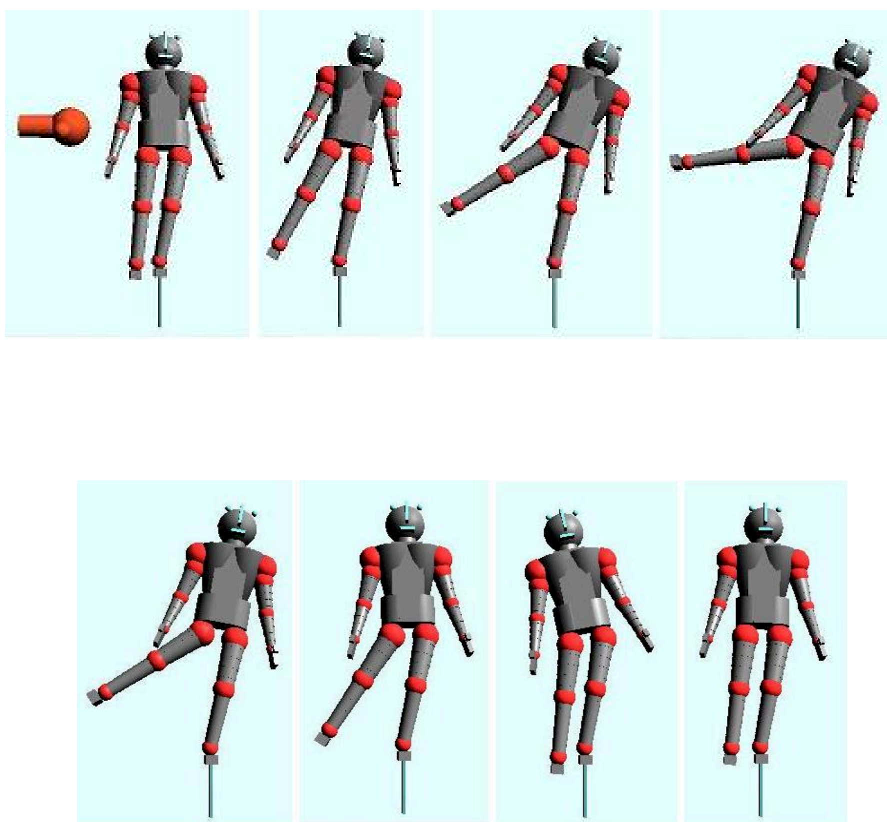


Fig. 8.15 - Motion sequence of biped for lateral disturbance while standing on podium

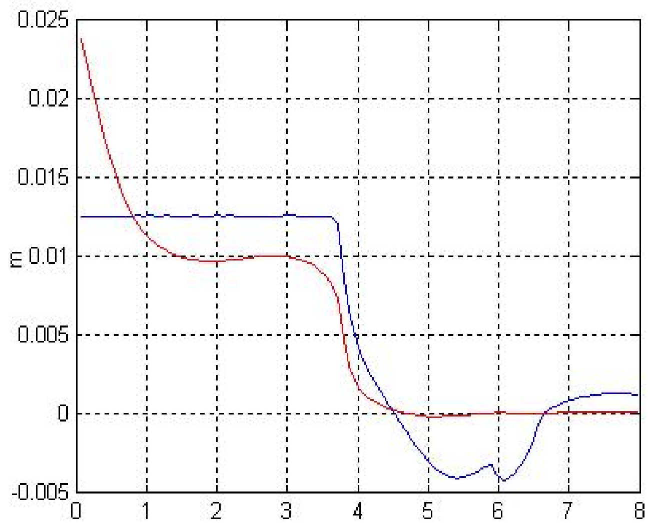


Fig. 8.16 – Lateral direction ZMP in blue, CMP in red

8.3.5 Adjusting Movement Preferences

The previous results show that spin angular momentum generation can have a significant favorable effect on balance stability. However, there are several different ways that this effect can be achieved. For example, a very flexible biped might favor use of the non-contact leg rather than the upper body, as shown in Fig. 8.17.

This is a very different movement from those shown in Figs. 8.11 and 8.15. It is achieved by adjusting the cost function weights in the controller's QP formulation; by setting the slack costs for body orientation to be high, the system maintains body orientation as a high priority goal. The burden then falls on the non-contact leg to supply the necessary angular momentum.

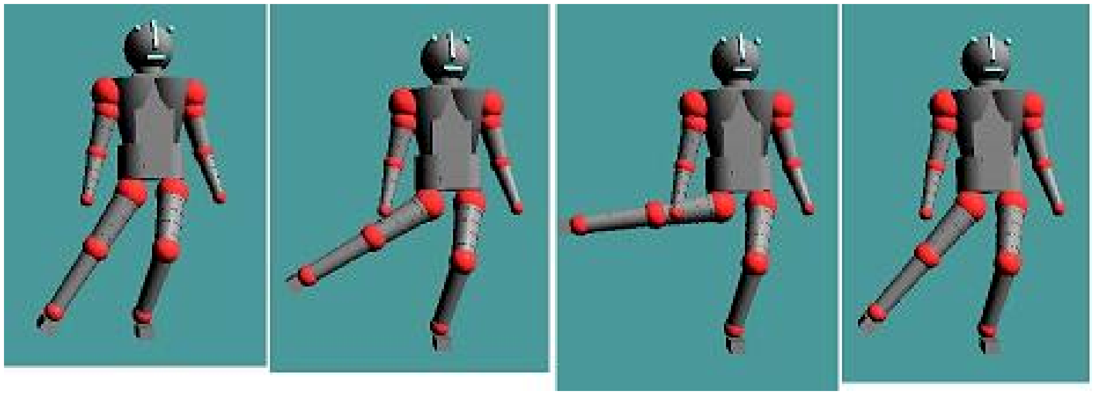


Fig. 8.17 - Motion sequence of biped for lateral disturbance while keeping body upright

8.3.6 Effect of Omitting Joint Limit Constraints

The controller's QP formulation includes constraints that enforce joint limits. These ensure, for example, that the knee can't bend backwards, or that the leg can't spin 360 degrees around the hip socket. Omitting these constraints results in interesting behavior, as shown in Fig. 8.18. Without the constraints, the controller finds a solution that generates the required angular momentum, but in a way that is not physically possible. Thus, the constraints are important.

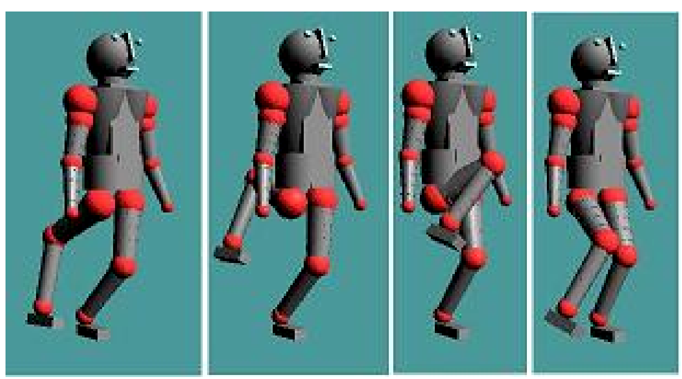


Fig. 8.18 - Motion sequence for forward disturbance, no joint limits

8.4 Discussion

The results show that the controller's use of non-contact segments is crucial for balance recovery in situations where stance ankle torque is insufficient, and stepping is restricted. This is especially true in situations like the podium balance experiments, where the support base is even smaller than the bottom of the stance foot.

The controller allows balance requirements to be specified concisely in terms of workspace state variables, like CM, which summarize balance state. For the balance experiments described in the previous section, the primary goal was expressed as a CM horizontal position setpoint over the support polygon, and a velocity setpoint of 0. The secondary goal of upright body orientation was expressed as 0 position and velocity setpoints on body roll, pitch, and yaw. This is a much more convenient way of specifying behavior than attempting to express balance requirements in terms of the native joint state variables of the system.

Besides the setpoints, PD gain and slack cost parameters can be adjusted to tune performance. For example, changing slack costs to favor rotation of the non-contact leg over rotation of the body resulted in the behavior described in Section 8.3.5. Changing PD gain parameters changes the trajectory shape of linearized variables such as CM as they approach their setpoints. For example, by increasing the PD proportional gain, the trajectory can be made to approach its position setpoint more quickly.

Despite the fact that the balance requirement specification is simple, the controller automatically produces very sophisticated and behavior, as shown in the motion sequences of the previous section. What is particularly striking about this approach is that it generates this behavior based solely on current state information; there is no look-ahead, no dynamic optimization over some future horizon to generate detailed trajectories, and there is no use of stored trajectories.

Another key difference between this controller, and previous commonly used controllers, such as the ones used for Asimo, is that this controller does not track detailed joint reference trajectories. Instead, it tracks workspace state variables, like CM, that summarize desired balance behavior, and then automatically figures out what the joints should do. This is similar, in concept, to the virtual model control (VMC) algorithm [Pratt et al., 1997], which uses a Jacobian transformation to compute torques that achieve a desired force at a reaction point. However, unlike the VMC algorithm, this controller takes dynamics into account. This is essential for producing the appropriate non-contact rotational movements, shown in the previous motion sequences, which are crucial for restoring balance.

9 Results

In Chapter 4, we described how task goals are specified using a qualitative state plan (QSP). Chapter 4 also described the linear virtual element abstraction, an abstracted biped that is easier to control than the actual one. This abstraction is provided by the dynamic virtual model controller, described in Chapter 8. The QSP, and the state of the abstracted biped are input to the hybrid executive, as shown in Fig. 1.14. The hybrid executive outputs control parameters to the abstracted biped in order to execute the plan. Within the hybrid executive, the QSP is compiled into a qualitative control plan (QCP) by the plan compiler, and the QCP is executed by the hybrid dispatcher, as shown in Fig. 1.16. Chapter 5 describes the QCP, Chapter 6, the hybrid dispatcher, and Chapter 7, the plan compiler.

We now present test results of execution of a variety of qualitative control plans by the dispatcher. These tests exercise nominal walking at different speeds, walking with disturbances, and walking with foot placement and temporal constraints. The tests are designed to evaluate the system's performance with respect to the thesis goals. Recall from our problem statement in Section 1.2 that our goal is a robust plan execution system capable of guiding a robotic biped through a series of walking task goals, in the presence of disturbances. The system must take a high-level task specification input (the QSP), and then automatically generate control actions that accomplish these tasks. If a disturbance occurs, the system should generate compensating control actions so that plan goals are achieved, despite the disturbance. If this is not possible, the system must issue a warning.

We begin with a set of nominal walking tests, without disturbances, in order to exercise the system's ability to understand task goals and translate them into appropriate control actions. We then introduce a set of tests that measure the system's ability to deal with lateral push disturbances. These tests are important because, due to the narrowness of the support base, the system is more sensitive to lateral push disturbances than to forward ones. We perform these lateral disturbance tests first without using the moment strategy (see Section 1.4.3) and then with this strategy, in order to evaluate its usefulness for balance recovery. Besides lateral push disturbance tests, we also perform tests to evaluate the system's ability to recover from trips.

In order to exercise the system's ability to execute plans with arbitrary state-space constraints, we perform tests with foot placement constraints that are different from those for normal walking, include walking on a balance beam, where foot placement is constrained in the lateral direction, and walking that requires an irregular stepping pattern in order to achieve very restrictive foot placement constraints. The latter is useful for walking or climbing on difficult terrain, where stable footholds are scarce.

Tests with temporal constraints exercise the system's ability to synchronize its movements with external timing requirements. An example of such a test is kicking a moving soccer ball. The biped must be at the right location at the right time in order to kick the ball.

A special kind of disturbance occurs when the biped walks on ground that is not firm. To evaluate the system's performance on such terrain, we perform a series of tests with the biped walking on soft and slippery ground. This tests the system's ability to maintain control over the biped's center of mass, even when the supporting feet are moving significantly from the nominal positions they have when walking on firm ground.

We conclude this chapter with a series of tests that measure the completeness of the flow tube approximation described in Chapter 5. Whereas the preceding tests are intended to evaluate the adequacy of our approximation, the tests that measure completeness are intended to discover opportunities for improvement (see also Chapter 10).

The experimental setup of these tests is the same as that described in Section 8.1.

9.1 Medium Speed Walking on Firm, Level Terrain

In this section, we provide results for a test involving walking at medium speed on firm level terrain. We begin with a presentation of the input QSP, followed by the QCP produced from the QSP by the plan compiler. We then present results of executing this QCP.

9.1.1 Input QSP

The QSP used for this test is identical to the one shown in Fig. 4.14. Multiple walking cycles are achieved by executing the QCP associated with the input QSP repeatedly.

Foot placement constraints are given in Table 9.1. These foot placement constraints are used to express state-space constraints in the input QSP. In Table 9.1, R1 refers to the first right foot placement, L1 to the first left foot placement, R2 to the second right foot placement, and L2 to the second left foot placement. The suffix _fwd refers to the forward direction, and _lat refers to the lateral direction, which points from right to left. Thus, the first row in Table 9.1 gives minimum and maximum values for R1_fwd, the forward position of the first right foot placement. These foot placements are also shown in Fig. 9.1.

Foot placement variable	**min**	**max**
R1_fwd	-0.01	0.01
R1_lat	-0.11	-0.09
L1_fwd	0.46	0.5
L1_lat	0.09	0.11
R2_fwd	0.96	1.0
R2_lat	-0.11	-0.09
L2_fwd	1.46	1.5
L2_lat	0.09	0.11

Table 9.1 – Foot placement constraints for medium speed nominal walking.

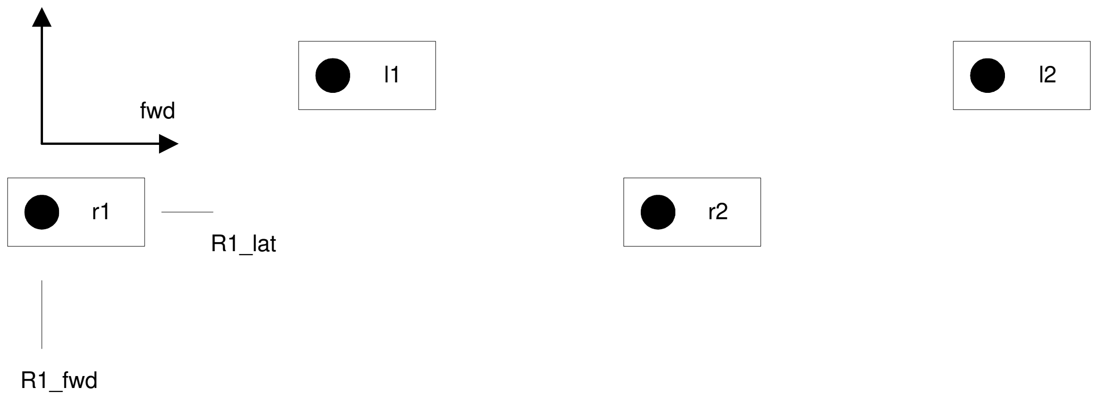


Fig. 9.1 – Foot placement sequence.

Explicitly specified temporal constraints are shown in Table 9.2. There is only one such constraint; it specifies an allowable duration between the events start and left heel strike (see Fig. 4.14). Because this is a constraint between the first and last event, it constrains the time for one walking cycle.

start event	**finish event**	**l**	**u**
start	left heel strike	1.2	1.8

Table 9.2 – Temporal constraints for the QSP for medium speed nominal walking.

State-space constraints for each activity in the input QSP are shown in Table 9.3. The activity structure shown in Fig. 4.14 is simplified in that forward and lateral components are shown as a single activity. In the actual QSP, they are separate. Thus, the activities CM1 (forward) and CM1 (lateral), shown in Table 9.3, are shown as a single activity, CM1, in Fig. 4.14.

Note that the goal region for CM4 (lateral) is the same as the initial region for CM1 (lateral). The initial region for CM4 (forward) is the same as the initial region for CM1 (forward), except that it is offset forward in position. This correspondence between the goal region of the last activity, and the initial region of the first, allows the plan to be executed repeatedly to achieve a sequence of walking steps.

Activity	Variable	Constraints
CM1 (forward)	Forward CM	Initial Region: 0.2 < pos. < 0.3, 0.6 < vel. < 0.9 Operating: min(R1_fwd) < pos_set < max(L1_fwd)
CM1 (lateral)	Lateral CM	Initial Region: 0.0 < pos. < 0.03, 0.15 < vel. < 0.3 Operating: min(R1_lat) < pos_set < max(L1_lat)
CM2 (forward)	Forward CM	Operating: min(L1_fwd) < pos_set < max(L1_fwd)
CM2 (lateral)	Lateral CM	Operating: min(L1_lat) < pos_set < max(L1_lat)
Rf step 1 (fwd)	Forward rf	Goal Region: 0.95 < pos < 1.0, 0 < vel < 0.1
Rf step 1 (lat)	Lateral rf	Goal Region: -0.11 < pos < -0.09, -0.05 < vel < 0.05
CM3 (forward)	Forward CM	Operating: min(L1_fwd) < pos_set < max(R2_fwd)
CM3 (lateral)	Lateral CM	Operating min(L1_lat) < pos_set < max(R2_lat)
CM4 (forward)	Forward CM	Goal Region: 1.2 < pos. < 1.3, 0.6 < vel. < 0.9 Operating: min(R2_fwd) < pos_set < max(R2_fwd)
CM4 (lateral)	Lateral CM	Goal Region: 0.0 < pos < 0.03, 0.15 < vel. < 0.3 Operating: min(R2_fwd) < pos_set < max(R2_fwd)
Lf step 1 (fwd)	Forward lf	Goal Region: 1.45 < pos < 1.5, 0 < vel < 0.1
Lf step1 (lat)	Lateral lf	Goal Region: 0.09 < pos < 0.11, -0.05 < vel < 0.05

Table 9.3 – State space constraints for the QSP for medium speed nominal walking.

9.1.2 QCP

In this subsection, we present the QCP corresponding to the QSP presented in the previous section. Table 9.4 shows the flow tube approximations of the QCP, computed by the plan compiler. Initial regions, goal regions, and duration ranges are provided for each activity.

Note that the QCP has rectangular goal regions for all activities. In particular, note that these regions have been computed for activities CM1, CM2, and CM3, even though the QSP does not specify goal regions for these activities. The flow tube approximations for these activities, including the rectangular goal and initial regions, are computed according to the algorithms described in Sections 7.3 and 7.4. The approximations take into account the foot placement constraints, which, in turn, represent actuation limits on horizontal force that can be applied to the center of mass, as described in Section 7.3.

9.1.3 Medium Speed Walking Execution

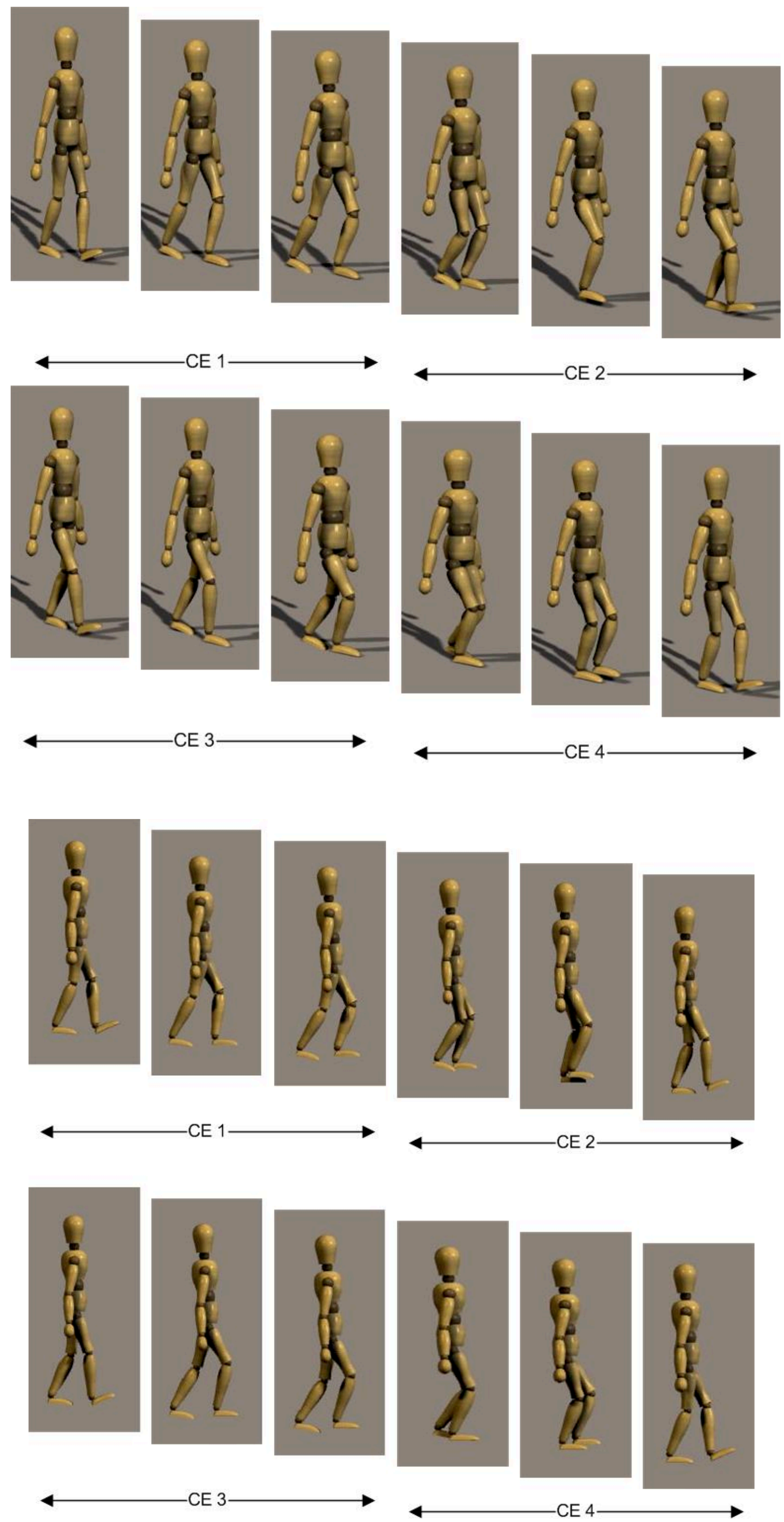
Fig. 9.2 shows the biped motion sequence for nominal walking at a medium speed of 0.73 m/s. The figure shows several frames corresponding to each activity CM1 – CM4. As required by the QSP of Fig. 4.14, activity CM1 corresponds to a double support qualitative state, with the left foot in front. CM2 corresponds to a single support qualitative state, where the left foot is the supporting foot, and the right foot is stepping. CM3 corresponds to double support with the right foot in front. CM4 corresponds to single support, where the right foot is the supporting foot, and the left foot is stepping.

The first frame for CM1 shows left heel-strike. The third frame of CM1 shows right toe-off, representing the transition to single support. The third frame of CM2 shows right heel-strike, representing the transition to double support. Similarly, the third frame of CM3 shows left toe-off, and the third frame of CM4 shows left heel strike.

Fig. 9.3 shows the same sequence, but from a front view.

Activity	Initial region				Goal region				Duration	
	pos		vel		pos		vel			
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
CM1 (fwd)	0.2	0.3	0.6	0.9	0.35	0.45	0.6	1.0	0.1	0.2
CM1 (lat)	0.0	0.03	0.15	0.3	0.04	0.06	0.05	0.15	0.1	0.2
CM2 (fwd)	0.35	0.45	0.6	1.0	0.7	0.8	0.6	0.9	0.5	0.8
CM2 (lat)	0.04	0.06	0.05	0.15	-.03	0.0	-0.3	-.18	0.5	0.8
Rf step 1 (fwd)	-0.1	0.1	-0.1	0.1	0.95	1.0	0	0.1	0.5	0.8
Rf step 1 (lat)	-0.11	-0.09	-0.05	0.05	-.11	-0.09	-0.05	0.05	0.5	0.8
CM3 (fwd)	0.7	0.8	0.6	0.9	1.05	1.15	0.6	1.0	0.1	0.2
CM3 (lat)	-0.03	0.0	-0.3	-0.15	-.06	-.04	-0.15	-0.05	0.1	0.2
CM4 (fwd)	0.85	0.95	0.6	1.0	1.2	1.3	0.6	0.9	0.5	0.8
CM4 (lat)	-0.06	-0.04	-0.15	-0.05	0	0.03	0.18	0.3	0.5	0.8
Lf step 1 (fwd)	0.4	0.6	-0.1	0.1	1.45	1.5	0	0.1	0.5	0.8
Lf step 1 (lat)	0.09	0.11	-0.05	0.05	0.09	0.11	-0.05	0.05	0.5	0.8

Table 9.4 – QCP for medium speed nominal walking.



a.

b.

Fig. 9.2 – Walking at speed of 0.73 m/s; a) side-front view, b) side view

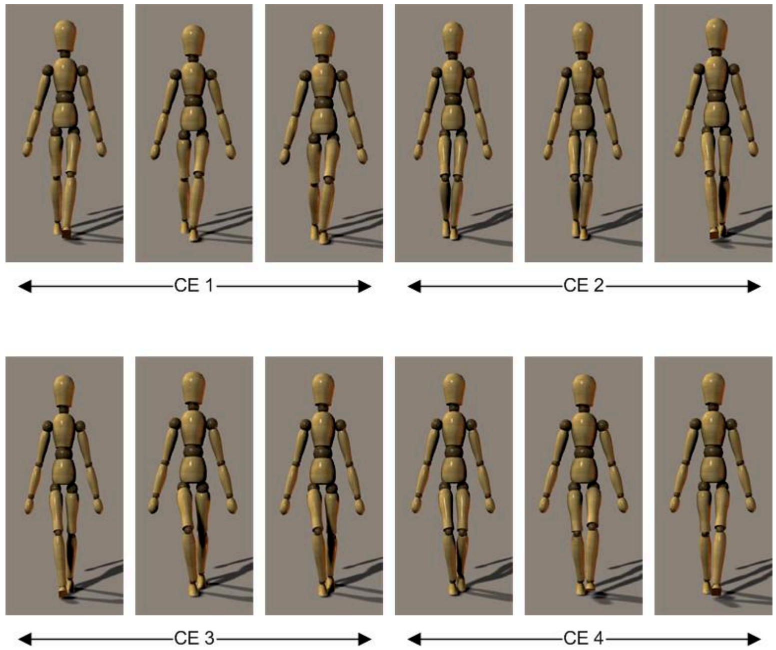
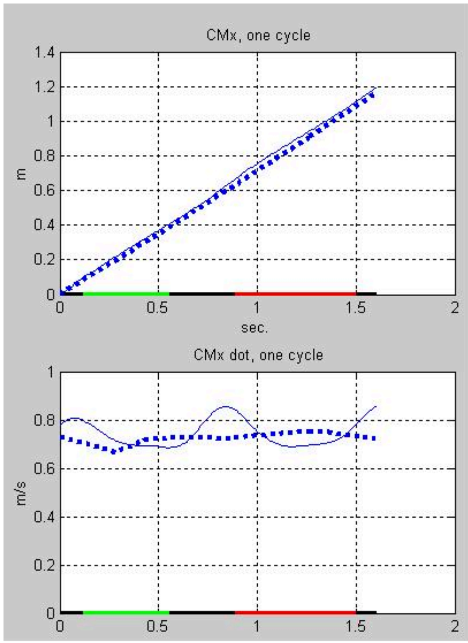


Fig. 9.3 – Walking at 0.73 m/s, front view

Fig. 9.4 shows forward and lateral CM trajectories for medium walking, at 0.73 m/s. The dotted lines show trajectories produced by the biped model. The solid lines show representative trajectories from a human motion capture trial (see Chapter 3 for a description of how this data was collected). These are shown to give an indication of level of biomimetic performance. The position trajectories produced by the model match those from the motion capture trial closely. For the lateral CM position trajectory, shown in Fig. 9.4b, the maximum deviation is less than 2 mm, which is less than 3% of the overall 8 cm peak to peak range of this trajectory over one walking cycle. The lateral velocity trajectories also match closely, but there is some deviation in the forward velocity trajectories. This deviation does not effect overall performance; the discrepancies cancel out so that the average deviation, over the cycle, is zero, as indicated by the fact that the forward CM position trajectories match closely. This is an indication that a variety of velocity trajectories result in correct behavior. For this test, correct behavior is walking at a particular average speed without falling down. Further testing will be needed to uncover the reason for the discrepancy, and whether one trajectory is preferable to the other. A particularly interesting test would be to see which of the two trajectories is more energy efficient.

Fig. 9.5 shows CM and center of pressure (CP) trajectories, also for walking at 0.73 m/s. While the CM trajectories match closely, there is some deviation in the CP trajectories. This is not surprising because CP is related to the horizontal ground reaction force component, as described in Chapter 3. This force input is proportional to the horizontal acceleration of the CM, which is the second derivative of the CM position. Therefore, a small deviation in the CM position trajectory can correspond to a large one in the CP trajectory. As with the velocity deviation in Fig. 9.4a, this is an indication that a variety of force inputs yields similar CM trajectories.

Fig. 9.6 shows desired lateral CM acceleration and acceleration slack of the dynamic virtual model controller (see Chapter 8). Recall, from Chapter 8, that a non-zero acceleration slack is proportional to the error between the desired acceleration, and the actual acceleration achieved. As can be seen from this plot, acceleration slack is almost 0 at all times, so the actual acceleration is very close to the desired. This indicates that the controller is able to control lateral CM movement according to the linear control law, while observing actuation limits. Therefore, the lateral CM moves according to the linear prediction of its flow tube, and little or no adjustment of the control parameters computed when an activity begins should be necessary. This is confirmed by the fact that within the dispatcher's Monitor loop (see Chapter 6), adjustment of control parameters from their initial settings, computed at the start of an activity, is required less than 10% of the time, for this test.

a. 

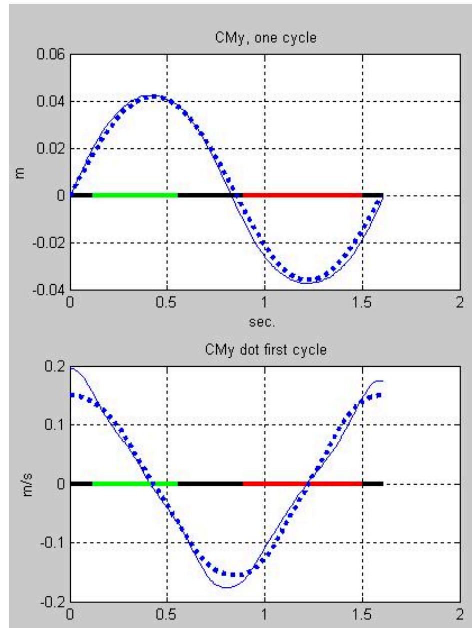
b. 

Fig. 9.4 - Center of mass trajectories for nominal walking at 0.73 m/s; a) forward component; b) lateral component; upper plots show position vs. time, lower plots show velocity; dotted line shows trajectory from biped model; solid line is representative trajectory from human motion capture trial; horizontal bar indicates support state, with red indicating left single support, green indicating right single support, and black indicating double support.

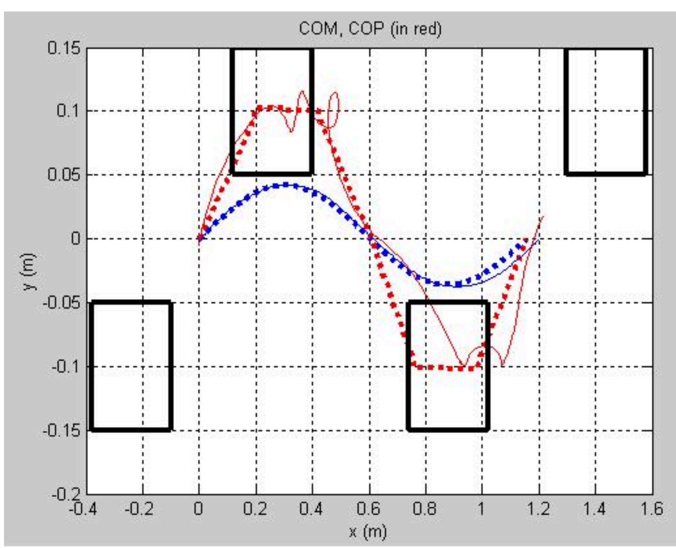


Fig. 9.5 – Center of mass (blue) and center of pressure (red) trajectories for nominal walking at 0.73 m/s; dotted lines show trajectories from biped model; solid lines are

representative trajectories from human motion capture trial; black rectangles show foot placement positions; vertical axis represents lateral movement, horizontal axis, forward.

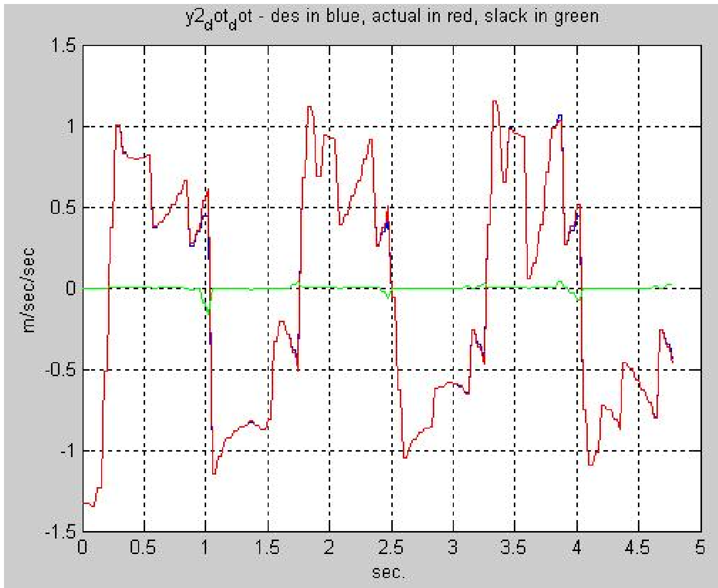


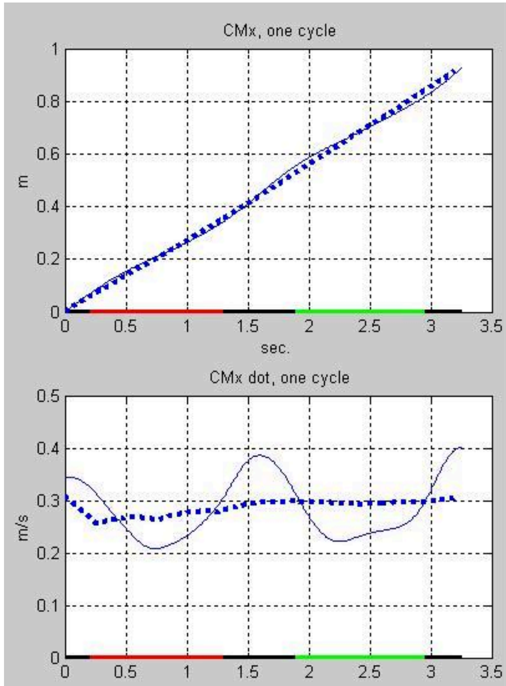
Fig. 9.6 – Desired lateral CM acceleration (red) and associated slack (green), for Nominal walking at 0.73 m/s. Three walking cycles are shown.

9.2 Slow and Fast Walking on Firm, Level Terrain

In order to further validate the capabilities of our system, we performed a series of walking tests at a slow and fast walking speed, to augment the medium walking speed tests. Like the medium walking speed tests, these tests were performed using firm, level terrain.

For slow and fast walking, a QSP identical to the one for medium speed walking was used, but the temporal constraint was adjusted to control speed for the cycle. Fig. 9.7, which is similar to Fig. 9.4, shows forward and lateral CM trajectories for slow walking, at 0.31 m/s. Fig. 9.8 shows CM and CP trajectories, also for slow walking. Fig. 9.9 shows forward and lateral CM trajectories for fast walking, at 1.15 m/s, and Fig. 9.10 shows corresponding CM and CP trajectories.

As with the plots for medium speed walking, the dotted lines show trajectories produced by the biped model, and the solid lines show representative trajectories from a human motion capture trial. As with medium speed walking, there is some deviation in

a. 

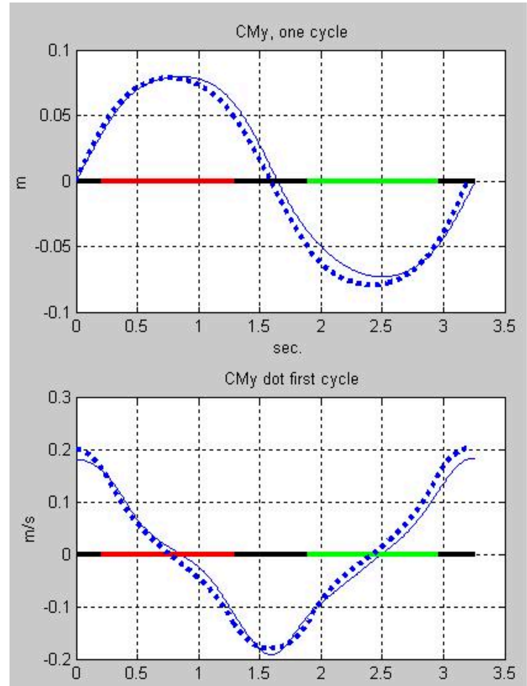
b. 

Fig. 9.7 - Center of mass trajectories, vs. time, for nominal walking at 0.31 m/s; a) forward component; b) lateral component; upper plots show position vs. time, lower plots show velocity

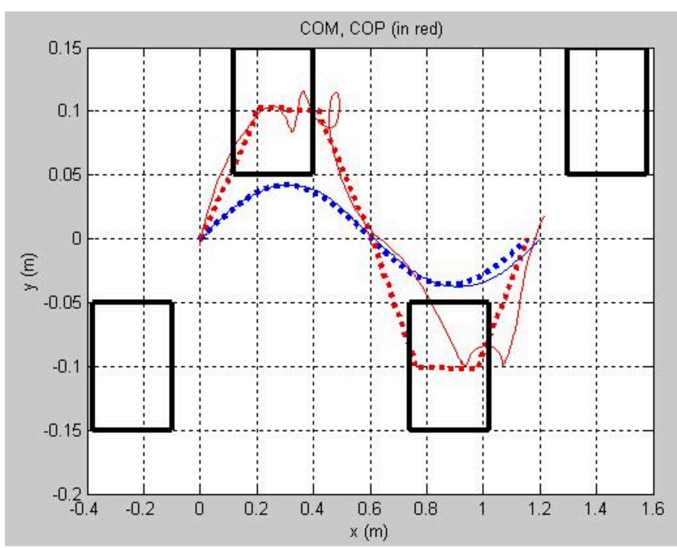
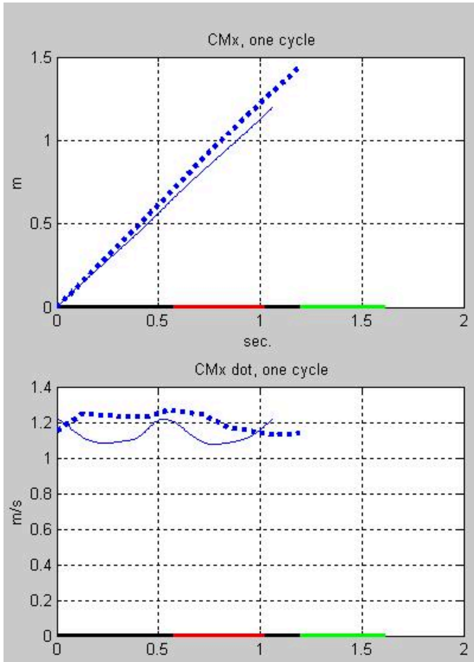


Fig. 9.8 – Center of mass (blue) and center of pressure (red) trajectories for nominal walking at 0.31 m/s; vertical axis represents lateral movement, horizontal axis, forward

a.



b.

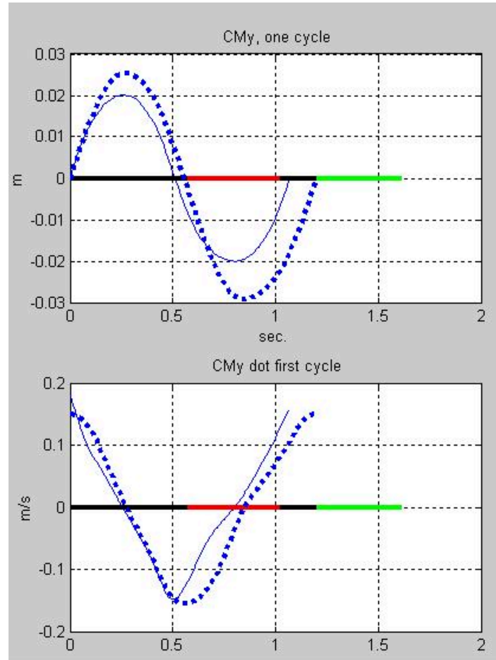


Fig. 9.9 - Center of mass (blue) and center of pressure (red) trajectories for nominal walking at 1.15 m/s; upper plots show position vs. time, lower plots show velocity

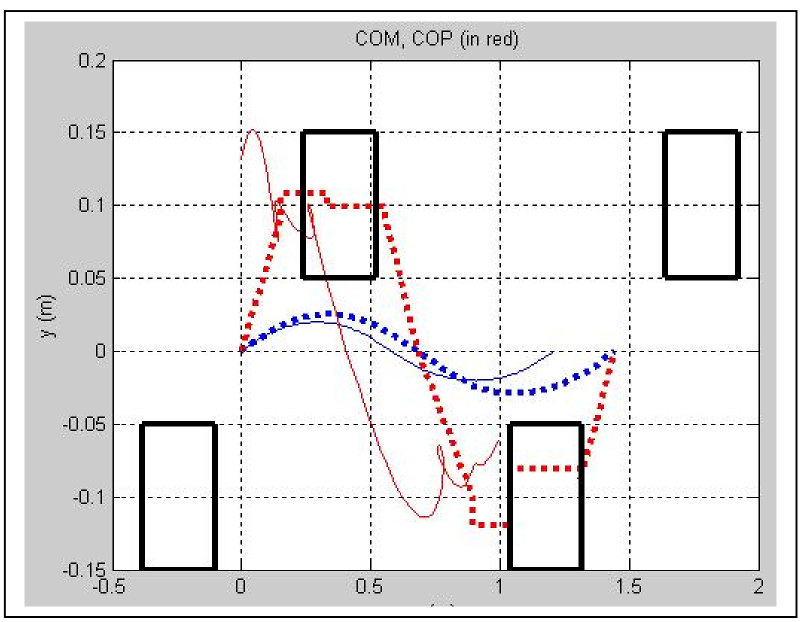


Fig. 9.10 – Center of mass (blue) and center of pressure (red) trajectories for nominal walking at 1.15 m/s; vertical axis represents lateral movement, horizontal axis, forward

the forward velocity trajectories, which averages to zero over the walking cycle, so the forward CM position trajectories match closely.

In Figs. 9.9 and 9.10, the model's trajectory lags slightly behind the human trial trajectory. This is because, for this test, the walking speed of the model was not exactly the same as that in the human trial.

As can be seen by comparing the position plots in Figs. 9.4b, 9.7b, and 9.9b, the peak-to-peak amplitude in the lateral CM position trajectory decreases as speed increases. For slow walking (Fig. 9.7b), the peak-to-peak amplitude is 0.16m. For medium walking (Fig. 9.4b), it is 0.08m, and for fast walking (Fig. 9.9b), it is 0.055m. Thus, the CM sways less from side to side as speed increases. This makes sense, intuitively, because forward and lateral CM movement must be synchronized with stepping, and at faster walking speeds, there is less time to shift weight from one supporting leg to the other, when taking a step.

This reduction in swaying also corresponds to a greater reliance on dynamic balancing when walking faster. The CM trajectory is further from the supporting feet for fast walking than for slow walking, as can be seen by comparing Fig. 9.10 (fast walking) with Fig. 9.8 (slow walking). Given that the supporting feet are 0.2 m apart for both walking speeds, the reduced peak-to-peak lateral CM amplitude when walking faster corresponds to a motion where the ground projection of the CM is further from the support base. This motion is less statically stable; the system cannot stop suddenly, in single support, and balance itself, because the CM is not over the support base. The system relies on the stepping foot being placed in the right position at the right time. Thus, the biped is constantly in an unstable state, in that there are no equilibrium points in the fast walking cycle. However, by constantly updating the base of support appropriately, it achieves limit-cycle stability by deferring falling indefinitely. Of course, to avoid a fall, the biped first slows down, and then comes to a stop in double-support, where there is an equilibrium point.

9.3 Lateral Push Disturbances

In order to validate the robustness provided by the region and duration flow tube approximations in the QCP of Table 9.4, we performed a series of walking tests, at medium speed on firm level terrain, as in Section 9.1, but with random lateral disturbances applied throughout the walking cycle. Lateral push disturbances are a better indicator of robustness than forward ones because, as discussed previously, the system is more sensitive to lateral push disturbances than to forward ones due to the narrowness of the support base. This is true in both double-support, because the double-support stance is longer than it is wide, and in single-support, because the foot is longer than it is wide.

The lateral push disturbances were modeled using a continuously applied lateral force, of random value within the range $-F_{lat\_max} \leq F_{lat} \leq F_{lat\_max}$. The force was updated every 0.05 seconds, and the random value was chosen according to a uniform probability distribution in this range.

In this series of tests, the moment strategy, introduced in Section 1.4.3, was not used. The usefulness of the moment strategy is demonstrated in the tests described in Section 9.6.

Fig. 9.11 shows a phase-plane plot of velocity vs. position for the lateral CM, with no lateral disturbance. This serves as a base test, and corresponds to the motions shown in Figs. 9.2 – 9.5.

In Fig. 9.11, the lateral CM trajectory is shown in blue and represents data from three full walking cycles. The vertical axis represents velocity, and the horizontal, position. Recall from Fig. 6.2, that activity CM_Lat_1 corresponds to double support, with the left foot in front, as shown, also, in Fig. 9.11. CM_Lat_2 corresponds to left single support, CM_Lat_3, to double support with the right foot in front, and CM_Lat_4, to right single support. The black rectangles in Fig. 9.11 show the initial regions for each of these activities. Note that all CM trajectories pass through these regions, indicating that the plan has executed successfully with respect to these regions.

Fig. 9.12 shows phase-plane plots similar to the one in Fig. 9.11, but with non-zero random lateral disturbances, as described above. In Fig. 9.12a, the maximum lateral disturbance is 10 N, whereas in Fig. 9.12b, the maximum is 20 N. In all cases, the CM trajectories pass through the required initial regions. Note, however, that there is more variation in the trajectories than for the test shown in Fig. 9.11. In particular, the 20N test results in a CM position variation of as much as 0.018m, which is more than 10 percent of the overall range of the lateral motion of the CM.

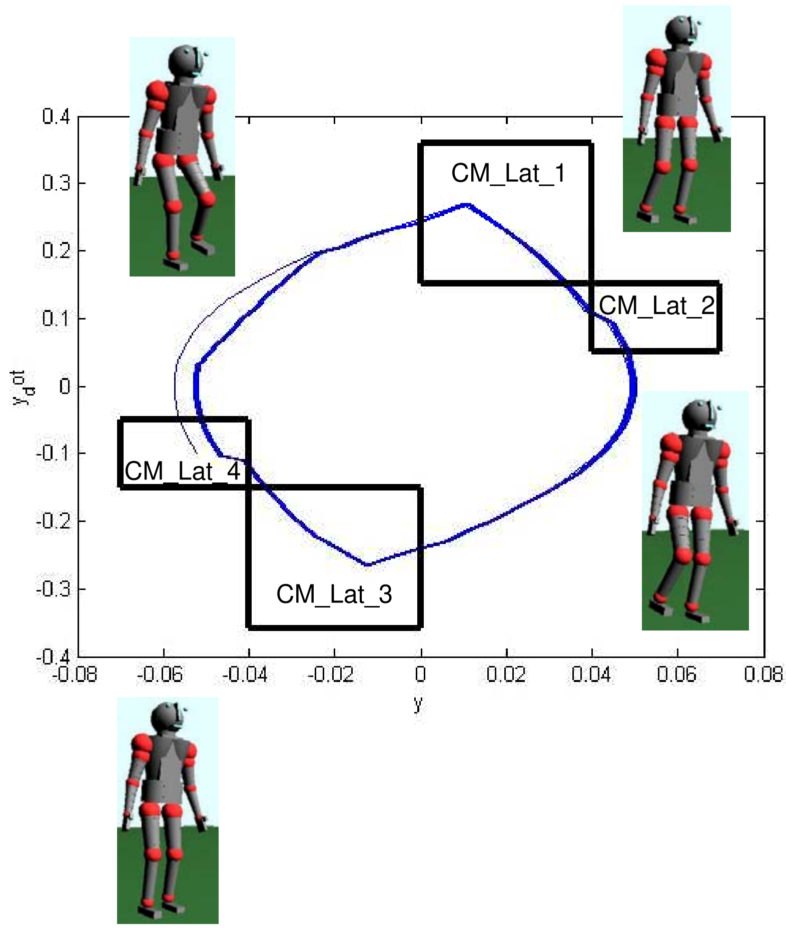
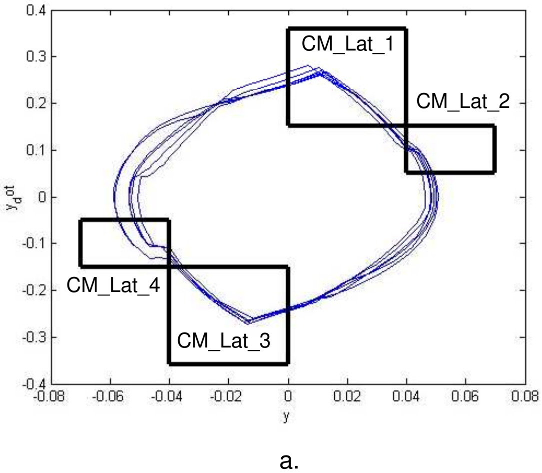


Fig. 9.11 – Phase-plane plot of lateral CM trajectory, shown in blue, for three walking cycles, with no lateral disturbance. The vertical axis represents velocity, the horizontal, position. Initial regions for each CM_Lat activity are shown in black.

In order to further investigate robustness to lateral push disturbances, we performed a more extensive series of walking tests, with a wider range of maximum lateral disturbance levels. We used maximum disturbance levels ranging from 10N to 35N. For each disturbance level, we performed 10 tests, each involving 3 full walking cycles. The larger maximum disturbance levels were large enough that plan execution would sometimes fail. In such cases, we noted the time of failure, and used this to compute the probability that plan execution would succeed at any dispatcher time increment. As described in Chapter 6, a dispatcher time increment is 0.05 seconds.



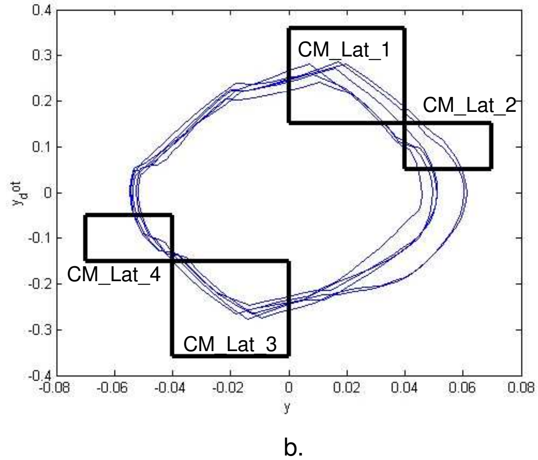


Fig. 9.12 - Phase-plane plots of lateral CM trajectory, shown in blue, for three walking cycles, with random lateral disturbances; a. – maximum lateral disturbance is 10 N; b. – maximum lateral disturbance is 20 N.

Fig. 9.13 shows the probability that the QCP shown in Table 9.4 will execute successfully, over the next dispatcher increment (over the next 0.05 seconds), for the range of maximum disturbances. As can be seen from this plot, the probability of success drops dramatically for disturbances of 30N or more.

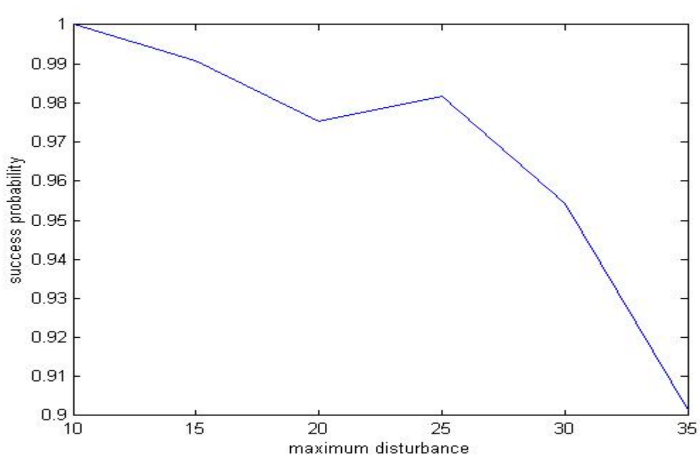


Fig. 9.13 – Probability of plan execution success, over the next dispatcher increment, for maximum lateral force disturbances ranging from 10 to 35 Newtons

The dispatcher increment of 0.05 seconds is a short time interval. A more practically useful interval to consider is 1 second of plan execution time. The success probability for a 1 second interval is obtained by raising the success probabilities in Fig. 9.13 to exponent 20. Fig. 9.14 shows this probability.

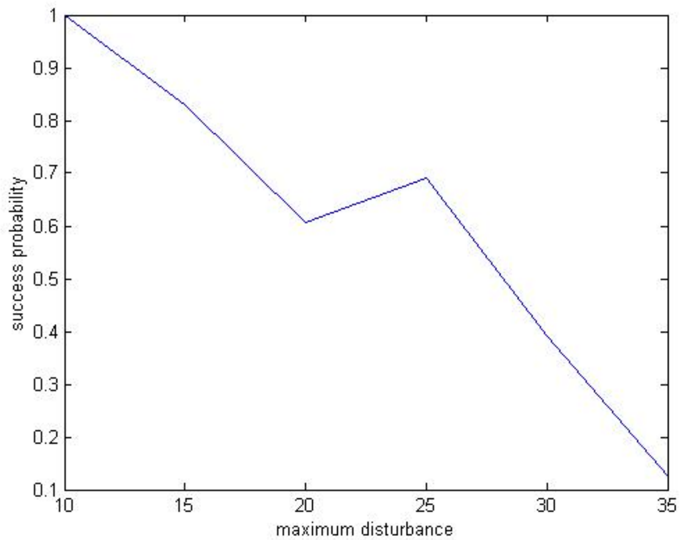


Fig. 9.14 – Probability of plan execution success, over 1 second
for maximum lateral force disturbances ranging from 10 to 35 Newtons

As can be seen from this plot, the probability of executing 1 second of the plan is less than 15% when the maximum lateral force disturbance is 35 Newtons.

Probability of plan execution success (or failure) is a useful way to think about high performance plan execution in unstructured environments. As the probability and magnitude of disturbances increases, so does the probability of plan execution failure. Probability analysis of the type depicted in Figs. 9.13 and 9.14 helps to quantify the performance capabilities of the system in a real environment, and to determine the need for contingency plans.

Note that for the plan executions shown in Figs. 9.13 and 9.14, execution failure does not, necessarily, imply that the biped will fall. It only means that the regions shown in Figs. 9.11 and 9.12 were not achieved; that the state trajectories went outside the bounds of the plan's flow tube approximations. If such plan failure is identified early enough, then the biped can switch to a contingency plan. This may involve walking more slowly or stopping after the disturbance, in order to regain balance, before continuing. It may also involve putting the stepping foot out further, in order to increase the base of support, as long as doing so does not violate foot placement constraints.

9.4 Irregular Foot Placement

Fig. 9.16 shows dynamic walking, but with an irregular stepping pattern. The irregular stepping pattern is necessary due to the irregular foot placements required by the blocks that the biped is walking on. These blocks move slowly, so the timing of foot placement, as well as the positioning is important. Timing requirements force the biped to move at a relatively fast speed, of about 0.8 m/s. At this speed, the biped can't just balance statically on each block. Instead, as with the fast walking described in Section 9.2, the fast speed requires dynamic balancing and coordination of the center of mass trajectory. Fig. 9.15 shows the CM trajectory and foot placements for this test. The dynamic nature of this task is indicated by the fact that the CM trajectory barely touches the foot placement polygons, and in one case, is 0.1 m away. This indicates that the system is not statically stable in this pose, and is relying on the subsequent foot placement sequence to maintain balance.

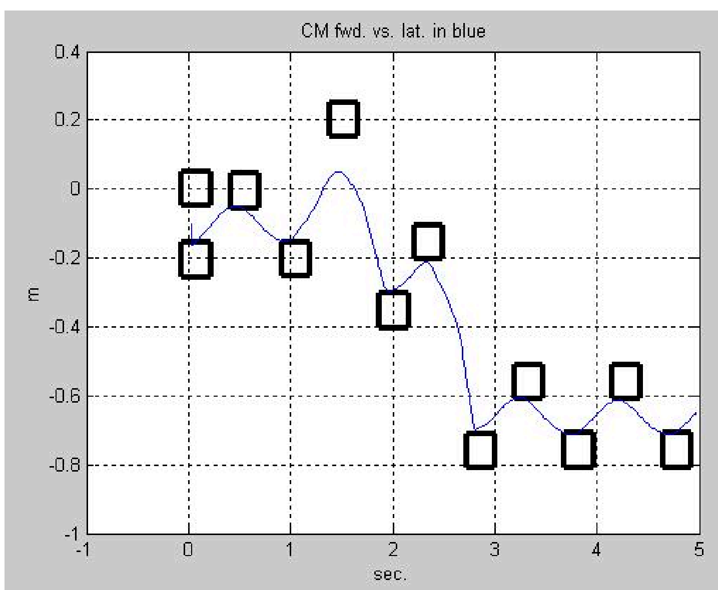


Fig. 9.15 – Foot placement and CM trajectory for irregular foot placement test.

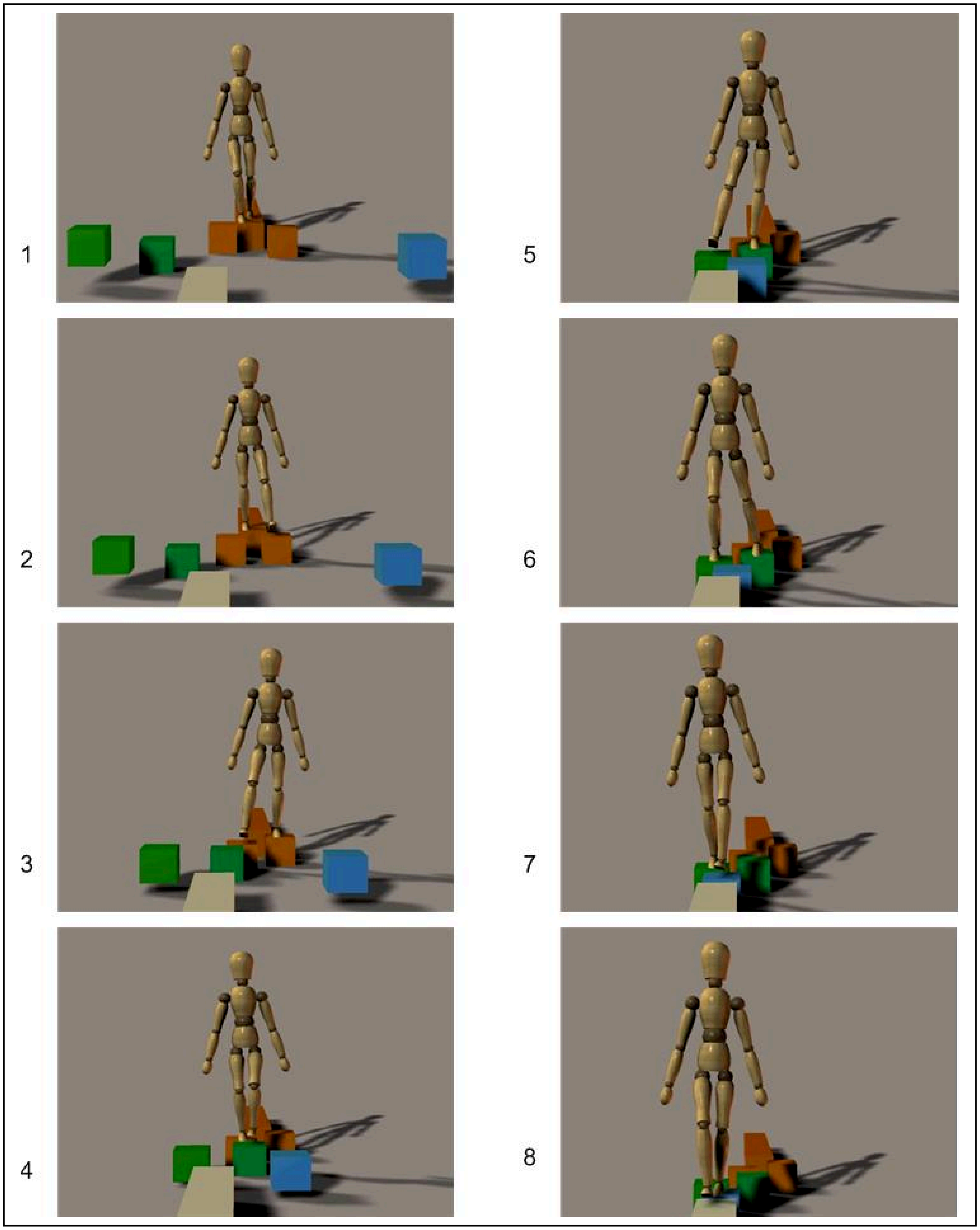


Fig. 9.16 – Walking by stepping on slowly moving blocks: 1) biped starts on long, narrow path; 2) steps with left foot onto the brown block; 3) steps with right foot onto the other brown block; 4) steps with left foot onto the green block; 5, 6) steps with right foot onto the other green block; 7) steps with left foot onto blue block; 8) finished

9.5 Kicking a Soccer Ball

In order to validate the system's ability to observe stringent temporal constraints, we performed a test involving kicking a moving soccer ball. The QSP for this test is similar to the one for walking, but with the temporal constraint set so that the biped is close to the ball when it has to kick it. Because the soccer ball is moving, the allowable temporal range is tighter than that for walking, as discussed in Section 7.4.2 (see Figs. 7.12 and 7.13).

Fig. 9.17 shows a motion sequence from this test. The kick is achieved by extending the goal region for the forward movement of the stepping foot, so that the foot moves further forward than for a regular step.

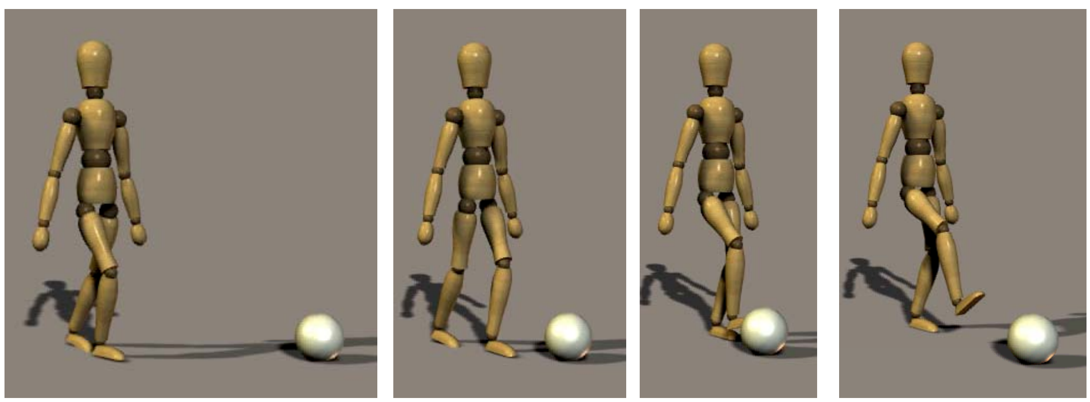


Fig. 9.17 – Walking to a moving soccer ball and kicking it.

9.6 Disturbance Recovery Using the Moment Strategy

Results for trip disturbance experiments were presented in Section 6.3.6. These experiments demonstrated how the system recovers from a trip disturbance by adjusting the spring constant control parameter in order to increase the speed with which the stepping foot moves forward.

Another type of disturbance is a push disturbance. As discussed previously, in Section 9.3, the biped is especially sensitive to lateral push disturbances when in single support, due to the limited support base provided by one foot. We performed the lateral push disturbance tests, described in Section 9.3, in order to investigate the system's robustness to such disturbances. In these tests, we did not utilize the moment strategy introduced in Section 1.4.3, and described further in Chapter 3.

In order to investigate the usefulness of this strategy, we performed an additional series of tests in which this strategy was used. The biped is most sensitive to lateral push disturbances when there are foot placement constraints, and when the push disturbance results in acceleration towards the outer edge of the stance foot. For example, if the biped is in left single support, a push towards the left is particularly problematic, especially if the subsequent placement of the right foot is restricted, as when walking on a balance beam. In order to investigate recovery from this extreme situation, we performed a series of tests with these conditions.

Fig. 9.18 shows recovery from a lateral push disturbance, while walking on a balance beam. The push occurs from the right side of the biped during left single support. Thus, the push results in an acceleration of the CM to the biped's left. Because foot placement is constrained by the narrowness of the balance beam, compensation by stepping is not an option. Furthermore, the disturbance, in this test, is too large to be handled by the ankle torque strategy alone (see Chapter 3). The system must use the spin angular momentum strategy in order to balance. This is accomplished through the angular movement of the torso and right leg, as shown in the third frame of the sequence (see also the discussion in Chapter 3, and further test results in Chapter 8). In particular, as shown in Fig. 9.18, the torso rotates clockwise, from the viewer's perspective, which induces a counter-clockwise rotation of the stance leg, which, in turn, engenders an acceleration of the biped's CM toward the biped's right. This corrects the CM position.

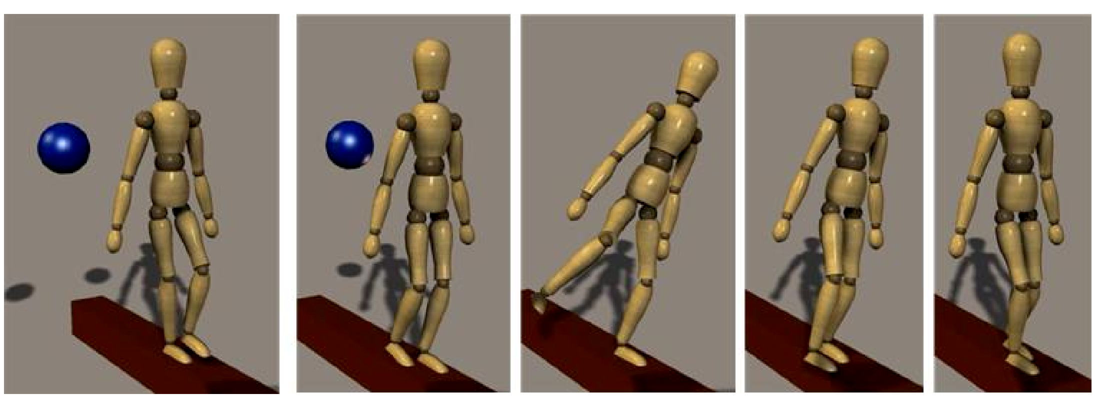


Fig. 9.18 – Recovery from lateral push while walking on a balance beam.

Due to joint acceleration limits, there is a limit to the angular acceleration that can be produced by the torso and the right leg. Therefore, recovery of lateral balance takes some time; the right leg is out for a significantly longer time (about two seconds) than it would be if it were just taking a normal step. This means that the forward center of mass velocity must be reduced while the lateral compensation movement is taking place. Otherwise, the biped would fall due to the forward CM being too far in front. This forward velocity reduction must be accomplished by the left (stance) foot alone. Due to support base limitations, there is a limit on the force that can be applied in this way, and therefore a limit to the negative forward acceleration that can be produced. Thus, the biped must be walking relatively slowly, in the first place, for this sort of maneuver to work at all. If this is the case, then forward movement of the CM can be slowed while the right leg is out, and then sped up again after the lateral compensation maneuver is completed. Thus, the forward CM position and the forward stepping position remain synchronized. This is one reason why people tend to walk slowly on tightropes or balance beams.

Fig. 9.19 shows a similar disturbance, occurring when the left foot is on the third block in a sequence of blocks that the biped must traverse.

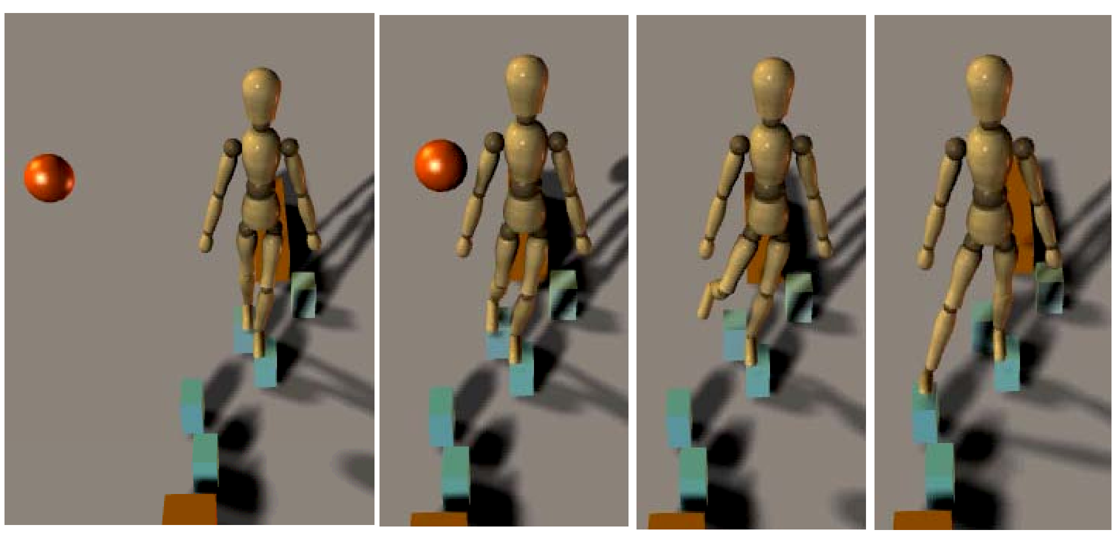


Fig. 9.19 – Recovery from lateral disturbance while walking on stones.

As with the balance beam, foot placement is restricted; the next step with the right foot must be such that the foot lands on the fourth block. The disturbance in this test is not as severe as the one for the balance test. However, the biped behaves in a similar manner.

The disturbance force vector points directly at the CM, which causes an undesired translational acceleration of the CM, but little rotational acceleration. However, as with the balance beam experiment, the system's reaction to the disturbance is rotational. Angular movement of the torso and right leg is used to help compensate for the disturbance. Fig. 9.20 shows a plot of roll angle of the torso about the forward axis. The negative spike indicates the compensating angular movement of the torso.

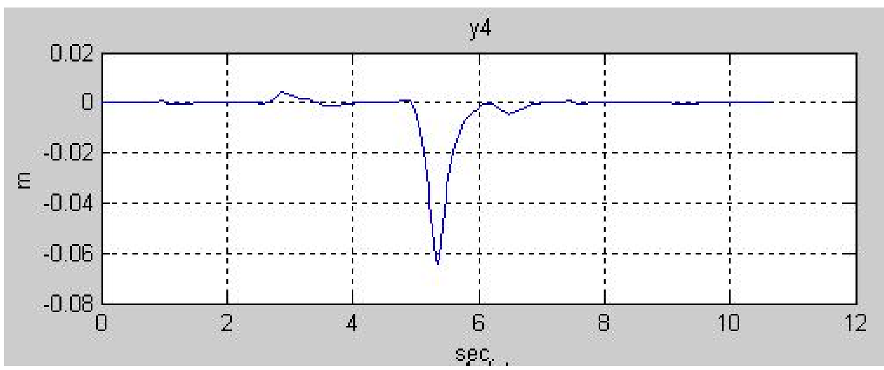
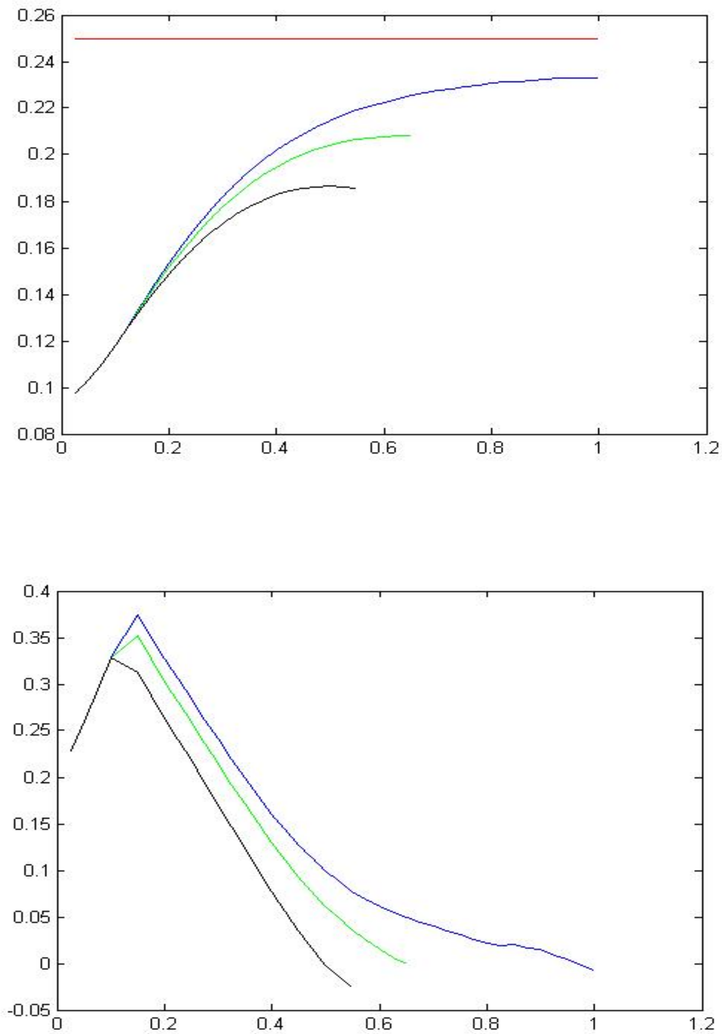


Fig. 9.20 – Roll angle about the forward axis of the torso, for the block walking experiment shown in Fig. 9.19. The negative spike represents compensating movement.

In order to further investigate this behavior, we performed this experiment with force disturbances of different magnitudes. Fig. 9.21 shows lateral CM trajectories resulting from lateral disturbance forces of 25, 35, and 40 N, applied over a period of 0.2 seconds, just before right toe-off. As in the previous experiments, angular movement, of the type shown in Fig. 9.20, is used to help compensate. This is achieved by allowing the CMP (see Chapter 3) to extend up to 4 cm beyond the outer edge of the foot. Table 9.5 shows the extreme (minimum) torso roll angles about the forward axis for each force disturbance.

a.



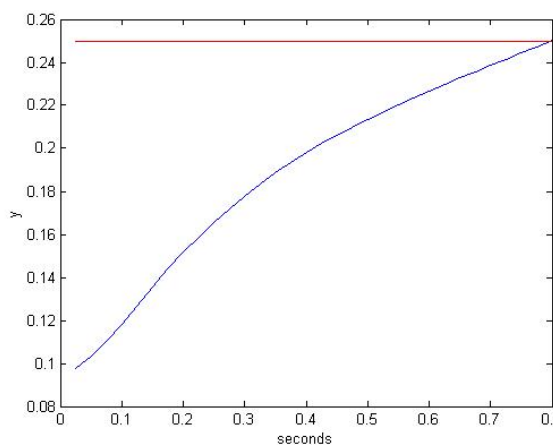
b.

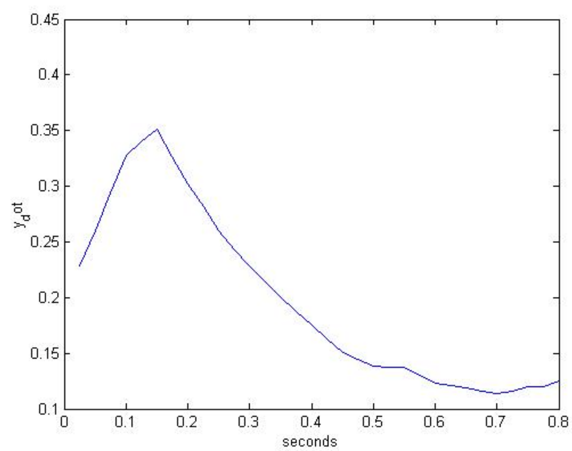
Fig. 9.21 – Lateral CM trajectories: a. position, b. velocity. These trajectories are in response to lateral force disturbances of 40 N (blue), 35 N (green), and 25 N (black). The force disturbance was applied for 0.2 sec. just before toe-off of the right foot. The angular momentum strategy was used. The red line in a. is the outer-left boundary of the left foot.

As shown in Fig. 9.21 a., the position trajectory for the 40 N disturbance, which is shown in blue, comes to within 2 cm of the outside edge of the left foot, before its velocity changes direction. If this trajectory were to reach the outside edge with positive velocity, balance could not be restored without appropriate stepping action to change the support base, or use of further angular acceleration. Note, however, that the latter strategy has its limits; the stability reservoir is not infinite, as discussed in Chapter 3. As shown in Table 9.5, the extreme torso roll angle for a 40 N disturbance is –0.64 rad. This is a significant deviation from the torso's upright orientation. Although this is not yet at the point where the torso is parallel to the ground, it is approaching a limit where further angular acceleration is not desirable.

Lateral disturbance force	Minimum roll angle
25 N	-0.025 rad
35 N	-0.09 rad
40 N	-0.64 rad

Table 9.5 – Extreme (minimum) torso roll angle used to compensate for force disturbances, for tests shown in Fig. 9.21.





a. b.

Fig. 9.22 - Lateral CM trajectories: a. position, b. velocity, in response to a 35 N lateral force disturbance, without using the angular momentum strategy.

In order to validate the importance of the angular momentum balance strategy, we repeated the experiment using the 35 N disturbance force, but without using this strategy. Fig. 9.22 shows the resulting lateral CM position and velocity trajectories. Note that the position trajectory reaches the outer edge boundary of the foot with positive velocity, indicating a loss of balance. In contrast, the position trajectory for the 35 N force (green trajectory) in Fig. 9.21, stops about 4 cm from this boundary. Therefore, use of the angular momentum strategy can significantly enhance balance stability.

9.7 Walking on Soft or Slippery Ground

The previously described tests, including the ones requiring irregular stepping, were all performed using firm terrain. However, many kinds of terrain that might be encountered by a biped in an unstructured environment may not be firm. For example, if the biped is walking on soft terrain, like sand or mud, the weight of the robot will cause the feet to sink into the ground. If the biped is walking on slippery terrain, like ice or a slick floor, lateral stability of the feet is reduced. Thus, soft and slippery terrain present control challenges not encountered with firm terrain.

To evaluate the system's behavior in such situations, we performed a series of tests where the ground was modeled to be soft or slippery. Recall, from Section 8.1, that we model ground contact using *contact points* at the four corners of the rectangular feet. When motion causes such a contact point to intersect the ground plane, the point of intersection is recorded as the *initial contact point*.

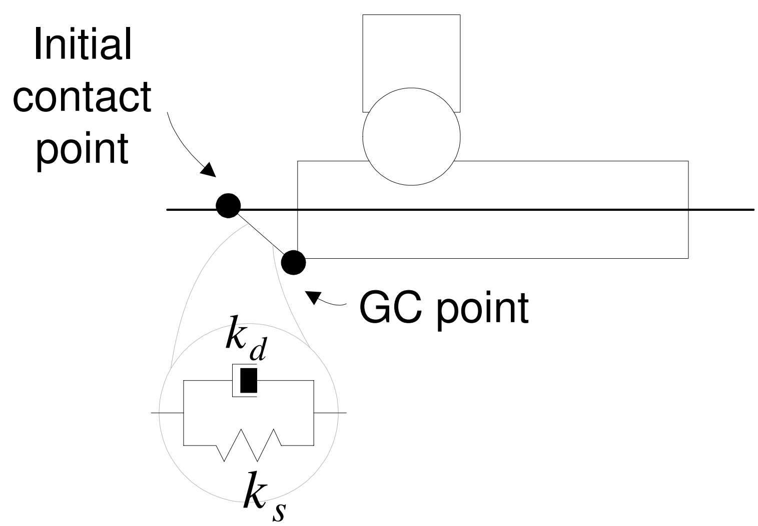


Fig. 9.23 – The ground contact model uses a nonlinear spring-damper system to exert a restoring force between the ground contact point and the initial point of contact.

While the ground contact point remains below the ground plane, a spring damper system is automatically instantiated between the ground contact point and the initial contact point, as shown in Fig. 9.23, in order to model the restoring forces exerted by the ground against the foot.

A nonlinear spring is used in the spring damper system. The restoring forces on the ground contact point are given by

$$F_x = k_x \, \Delta x^2 - b_x \dot{x} \qquad 9.1$$

$$F_y = k_y \, \Delta y^2 - b_y \dot{y}$$

$$F_z = k_z \, \Delta z^2 - b_z \dot{z}$$

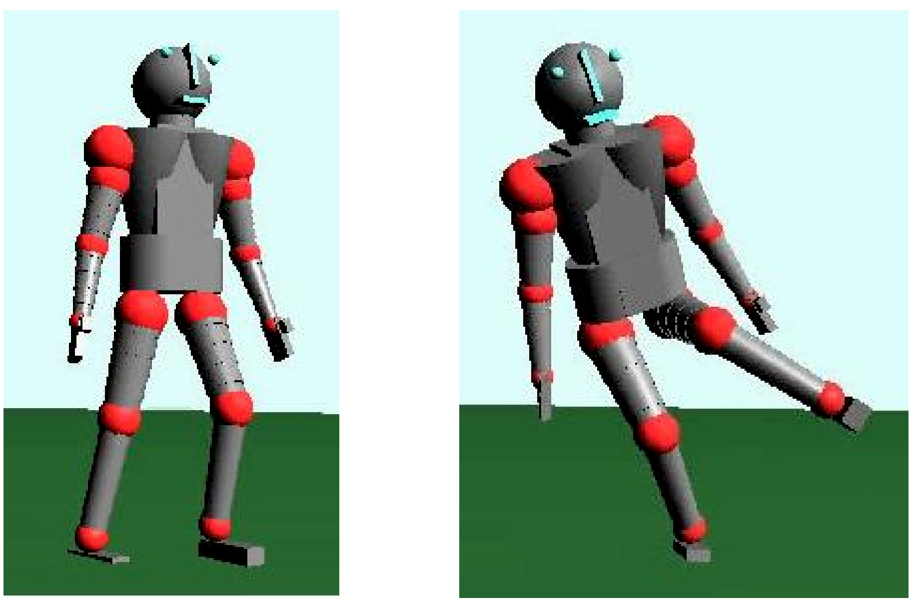
where F_x, F_y, and F_z are the forward, lateral, and vertical restoring forces, respectively, k_x, k_y, and k_z are the corresponding spring constants, b_x, b_y, and b_z are the damping constants, Δx, Δy, and Δz are the forward, lateral, and vertical components of the distance vector from the initial contact point to the ground contact point, and $\dot{x}$, $\dot{y}$, and $\dot{z}$ are the forward, lateral, and vertical components of the ground contact point's velocity. Values for the spring and damping constants, for firm terrain, are provided in Table 8.3. These constants were modified in order to model soft and slippery terrain. Table 9.6 shows the values used for different terrain types, and the associated maximum distance between the ground contact point and the initial contact point. The biped was able to walk successfully on each of these terrain types, at a slow walking speed of 0.3 m/s. The distances shown in Table 9.6 were obtained by recording maximum deflection of the ground contact points during slow walking, at 0.3 m/s.

For soft terrain, the vertical and horizontal spring constants were reduced from their firm terrain values. This caused the feet to sink below the ground plane by as much as 5 cm, as shown in Table 9.6, and Fig. 9.24a. Nevertheless, the execution system was able to maintain balance for slow walking speeds.

Terrain type	k_x, k_y (N/m^2)	k_z (N/m^2)	b_x, b_y $(N/m/s)$	b_z $(N/m/s)$	Max. vertical dist. (m)	Max. horizontal dist. (m)
Firm	2×10^6	2×10^6	400	400	0.01	0.01
Soft	1×10^5	1×10^5	100	100	0.05	0.05
Slippery	1×10^4	2×10^6	50	400	0.01	0.15

Table 9.6 – Spring and damping constants, and maximum ground contact point deflections, for firm, soft, and slippery terrain.

For slippery terrain, the vertical spring and damping constants were the same as those for firm terrain, but the horizontal spring and damping constants were significantly reduced. This allowed the foot to move, horizontally, by as much as 15 cm. The execution system was able to maintain balance, despite this instability, for slow walking speeds. When the horizontal spring constant was further reduced to 5000, the biped was no longer able to maintain balance, as shown in Fig. 9.24b.



a. b.

Fig. 9.24 – a. When walking on soft ground, the feet sink below the ground plane by as much as 5 cm. b. Very slippery ground causes the biped to fall.

9.8 Completeness of Flow Tube Approximation

In order to investigate the completeness of our flow tube approximation, we performed a preliminary analysis comparing our approximation with a more detailed one that is closer to the actual flow tube. Recall from Definition 5.7 that we approximate the true initial cross section of a flow tube using a rectangular approximation. Examples of such rectangular regions, for medium speed walking, are given in Table 9.4. Fig. 9.11 shows a phase-plane plot of lateral CM trajectories that passes through these regions, indicating successful plan execution.

In order to get a more detailed approximation of the actual flow tube, we extended the rectangular region representation to a more general polyhedral one. Thus, instead of a representation that was limited to 4 vertices and that required rectangular shape, this new representation allowed for n vertices, with no restrictions on the shape of the polyhedron. Computation for this polyhedral representation was similar to that for the rectangular representation, given in Theorem 7.3, in that GFT and GST trajectories were computed from each vertex in the initial region. However, instead of requiring that these trajectories meet a particular vertex in the goal region, such as point C or D in Theorem 7.3, we simply required that the final trajectory state be within the polyhedral goal region boundaries. Note that this is more computationally intensive than the algorithm given in Section 7.3.

In order to compare the representations, we used the QSP from Table 9.3, and computed the flow tube approximations for the CM_Lat_1 and CM_Lat_2 activities. The resulting initial regions for these activities, using an 8-vertex polyhedral representation, are shown in Fig. 9.25, in green. The original rectangular regions, from Fig. 9.11, are also shown, in black. As can be seen by comparing the regions, the polyhedral approximation covers a significantly larger area for CM_Lat_1. This suggests that a more complete representation, like the 8-vertex polyhedral one, would be more robust to disturbances. Thus, it is worth investigating whether use of such a more complete representation would increase the success probabilities shown in Figs. 9.13 and 9.14. We discuss approaches to more complete representations in more detail in the next Chapter.

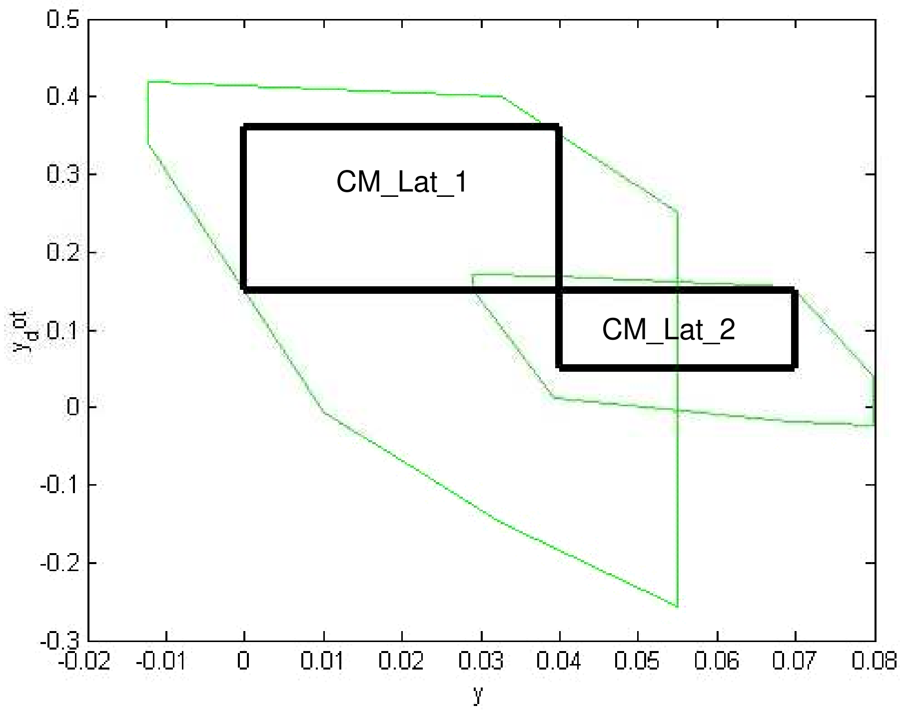


Fig. 9.25 – Initial region approximations for activities CM_Lat_1 and CM_Lat_2, from the QSP of Table 9.3. The 8-vertex polyhedral approximation is shown in green. The original rectangular region is shown in black, with thicker lines.

In this chapter, we have presented experimental results of tests of the system described in the previous chapters. These results demonstrate the system's ability to achieve stable walking at different speeds, walking on terrain requiring irregular stepping patterns, while observing temporal constraints, walking on soft or slippery terrain, and rejection of disturbances. In particular, the results of Section 9.6 demonstrate the usefulness of the moment strategy in rejecting disturbances when foot placement is constrained.

In the next chapter, we discuss limitations of our approach, and ways to address these limitations.

10 Discussion and Future Work

In this chapter, we discuss a number of limitations of our approach, and suggest additional work that may be done in the future to address these limitations.

In Section 10.1, we discuss the question of completeness of our flow tube approximation, and discuss ways that the approximation could be made more complete. As explained in this section, a fully complete flow tube representation is probably intractable due to the very large number of flow tubes that would have to be computed. Section 10.2 suggests an alternative, using a sparse flow tube network, but where flow tubes can be adjusted at runtime, by the dispatcher, in order to fit the current situation.

In Section 10.3, we discuss how recently developed learning algorithms might be incorporated into our architecture, in order to make it more robust to model error. Section 10.4 reviews the human trial data collected for this investigation, and discusses ways to extend the validation of our approach against this trial data. Section 10.5 discusses a number of interesting, recent developments in the area of biologically inspired control systems for balancing and walking, and how these might be used to augment our control capability. Section 10.6 discusses testing of our control architecture on a real robot. Finally, in Section 10.7, we summarize the contributions of this thesis

10.1 Completeness of Flow Tube Approximation.

An important issue is the completeness of the flow tube approximation described in Chapter 5. As stated in Section 5.1.1, a key requirement for this flow tube approximation is that it must include only feasible trajectories. Thus, the approximation may include a subset of all feasible trajectories, but it may not include a superset. This requirement provides the compile-time guarantee that any trajectory selected by the dispatcher from a flow tube will succeed, as long as there are no further disturbances to the system, as stated in Lemma 5.1, and in Theorems 5.1 and 5.2. Thus, we say that our flow tube approximation is *sound* in that it admits only trajectories that result in successful execution.

However, we have no requirement that our flow tube representation must include <u>all</u> feasible trajectories. Such a requirement would be difficult to satisfy, due to the complex geometry of the state space. Thus, we say that our flow tube approximation is not *complete*, because it does not include all trajectories that result in successful execution.

The fact that our flow tube approximation is not complete means that the dispatcher may abort plan execution prematurely, because it cannot find a feasible trajectory in the flow tube approximation, even if one exists in the actual flow tube. Therefore, it is useful to investigate, further, how good our flow tube approximation is, and whether it would be worthwhile to make it more complete.

In Section 9.3, we presented results from a series of tests involving random lateral disturbances. These tests were performed in order to investigate the level of robustness attainable using our incomplete flow tube approximation. We used these tests to quantify the variance in lateral CM trajectories, as shown in Figs. 9.11 and 9.12. Further, we used these tests to quantify the probability of plan execution success as a function of the maximum random disturbance level, as shown in Fig. 9.13. These results show that the system will reliably execute the test walking plan when the maximum random value for the lateral disturbance is less than 10 N. When this maximum disturbance value becomes greater than 20 N, performance becomes very unreliable.

In Section 9.8, we showed that the rectangular initial region of our flow tube approximation may omit significant sections of the actual flow tube's initial region. Therefore, our flow tube approximation may omit a significant set of trajectories that are feasible. This suggests that including more of the omitted sections of the actual flow tube's initial region in our flow tube approximation would improve robustness, perhaps significantly. For example, the success probabilities shown in Figs. 9.13 and 9.14 would increase because the initial regions shown in Figs. 9.11 and 9.12 would be larger. Therefore, it is worthwhile investigating ways to make our flow tube approximation more complete.

10.1.1 Multiple Initial Regions for Flexible-Duration Flow Tubes

As introduced in Section 5.1.5, flow tube sets can be used to represent feasible trajectories for an activity with flexible duration. All flow tubes in such a set have a common goal region, but they will have different initial regions, as shown in Figs. 5.7 and 5.8. Because duration of an activity is continuous, there is an infinite set of such initial regions. Therefore, a compact representation of the set of initial regions is needed, as described in Section 5.2.3.

In this thesis, our approach was to use a single initial region that is the intersection of initial regions, as described in Sections 5.1.5 and 5.2.3. The advantage of this approach is that it is simple; a single initial region is easier to represent than multiple ones, is easier to compute, using the relations described in Section 7.3, and is easier for the dispatcher to interpret. Also, because there is a single initial region, it is easy to satisfy the requirement, stated in Section 5.1.3, that the goal region of a flow tube for an activity must be a subset of the initial cross section of the flow tube of the activity's successor activity. This requirement guarantees soundness because it guarantees an unbroken path of feasible trajectories from the initial region of the first activity in a sequence to the goal region of the last activity in the sequence. It is a basis for Theorem 5.2, which provides a compile-time guarantee of execution success.

The disadvantage of this approach is that it sacrifices completeness, because the initial region is an intersection of multiple initial regions, all of which may be feasible, but which have different durations. The intersection results in an approximation with a single initial region that is smaller than any of the initial regions for the fixed-duration tubes, but that has a controllable duration that is larger than that of any of the fixed-duration tubes. This is the trade-off discussed in Section 5.1.5.

An alternative representation for feasible trajectories for an activity with flexible duration is to avoid the intersection of initial regions by preserving more of the initial regions in the fixed-duration flow tube set. This alternative representation, introduced in Sections 5.1.5 and 5.2.3, involves discretizing time, using an increment, Δt. We then include in the set only those initial regions of flow tubes that have a duration that is a multiple of this increment. With such a discretization, the set becomes finite. As discussed in Chapter 6, the dispatcher operates at a discrete time interval. If the dispatcher time increment is also Δt, then the dispatcher will only perform updates at multiples of Δt, and will only have to consider flow tubes with durations that are multiples of Δt. Therefore, a representation using a finite set of flow tubes, with durations that are multiples of Δt, satisfies the requirements stated in Section 5.1.5.

Consider the flow tube shown in Fig. 10.1. The flow tube has goal region G_1. The figure shows cross sections of the flow tube corresponding to durations d_1, d_2, and d_3, which are multiples of Δt, and where $d_1 > d_2 > d_3$.

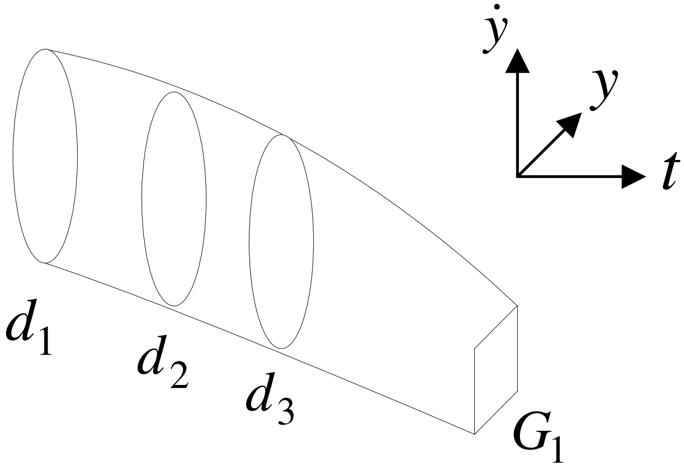


Fig. 10.1 – Flow tube with initial cross sections corresponding to durations.

The corresponding flow tube representation is shown in Fig. 10.2. Note that in contrast to the representation used for this thesis, which consisted of a single initial region, goal region, and duration, the representation in Fig. 10.2 is a tree, with multiple initial regions associated with different durations.

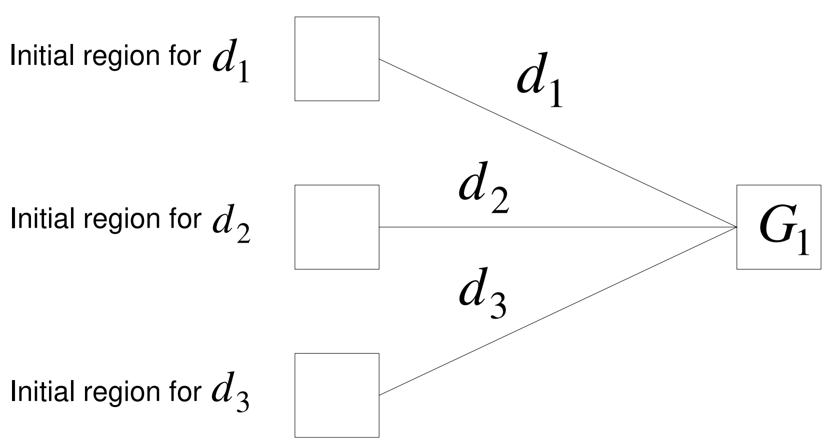


Fig. 10.2 – Flow tube representation for flow tube of Fig. 10.1.

The advantage of this approach is that it provides for a more complete representation of feasible trajectories. The set of initial regions in Fig. 10.2 covers a larger region of state space than their intersection. The disadvantage is that the approach is more complex, because it uses multiple initial regions, corresponding to different durations. This also complicates making compile-time guarantees about successful execution. In particular, connecting a flow tube of an activity with the flow tube of its successor is now a more complicated matter than just ensuring that the goal region of the activity flow tube is a subset of the initial region of the successor's flow tube. The successor flow tube will have multiple initial regions. To have a complete approximation requires expanding flow tubes back from each of these initial regions, resulting in a tree, as shown in Fig. 10.3. If each activity has a large number of initial regions, and if there are many activities in a sequence, then the fan-out of this tree could become very large. We will return to this problem, but first, we will discuss another aspect of completeness of the flow tube approximation: the fact that a single rectangle may not provide a very complete approximation of an initial region for a fixed-duration flow tube.

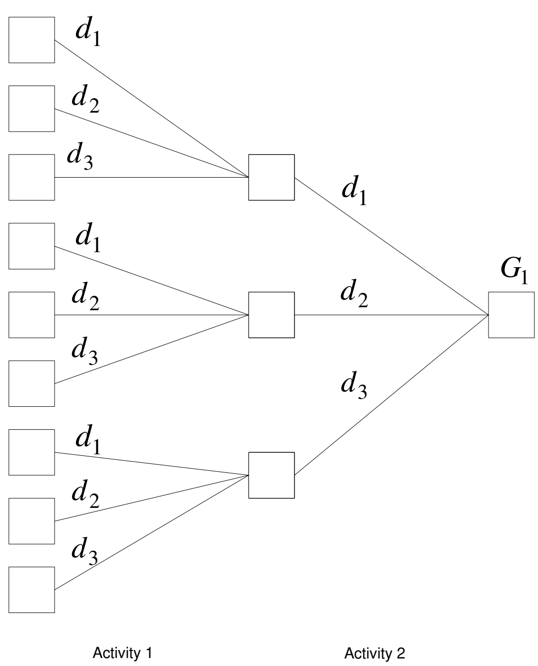


Fig. 10.3 – Fan-out from goal of Activity 2 to initial regions of Activity 1, where Activity 1 is the predecessor of Activity 2.

Note that, while the discussion in this subsection proposes use of multiple initial regions for the approximation of flexible duration flow tubes, no assumption is made about the representation used for each of these regions. A rectangular representation could be used, but other approaches are possible as well, as discussed in the next sub-section.

10.1.2 Initial Region Representation

As described in Section 5.2, we use rectangular initial and goal regions in our flow tube approximations. This has the advantage that rectangular regions are simple representations, making it easy to check whether a trajectory is within the region, and to ensure that a goal region fits within a successor's initial region. They are also easy to compute by the plan compiler, using the relations described in Section 7.2. The disadvantage is that a single rectangular region may not be a very good approximation of the true initial cross-section of a flow tube, as shown in Fig. 10.4.

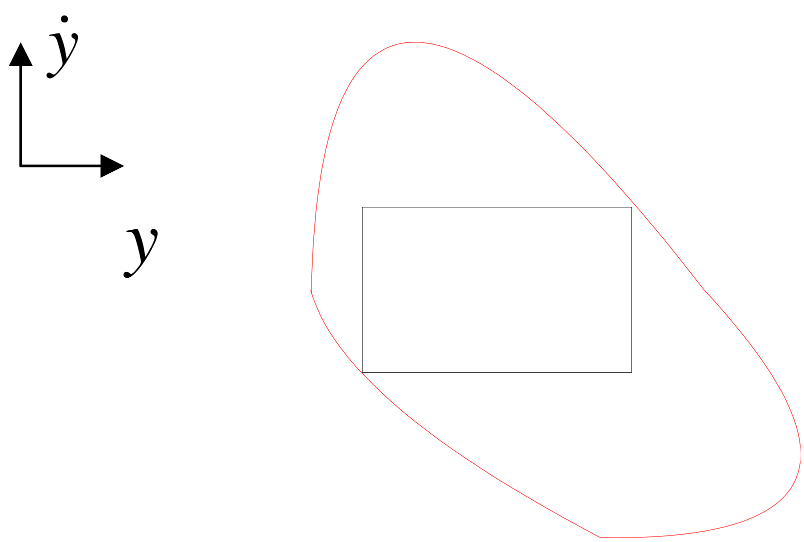


Fig. 10.4 – Example where a rectangular approximation covers less than half of the area of the flow tube initial cross-section, shown in red.

This problem was investigated in Section 9.8, where we showed that a rectangular initial region may omit significant sections of the actual flow tube's initial region, in a QCP for a typical locomotion plan.

A more complete approximation can be achieved using polygonal shapes [Vestal, 2001]. Polygonal representations can be viewed as a generalization of rectangular representations, and are worth investigating further. Based on the preliminary results presented in Section 9.8, a polygonal representation may provide significantly greater completeness than a rectangular one. Furthermore, many of the attractive properties for rectangular approximations, such as ease of checking whether a point or region is within another region, or the ability to incrementally adjust regions, as described in Section 10.2, may be extendable to polygonal representations. Further study will be required to investigate whether this is possible.

An alternative to polygonal representations that also addresses the problem of completeness is to simply use multiple rectangles, as shown in Fig. 10.5. This gives a more complete representation, while maintaining the attractive characteristics of a rectangular representation.

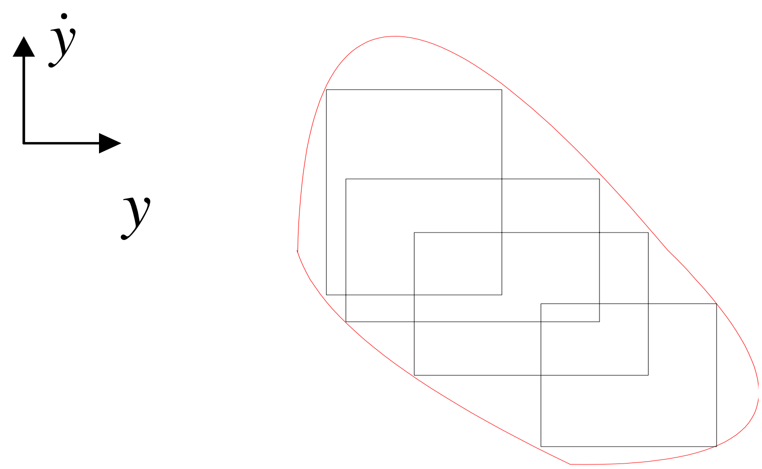


Fig. 10.5 – Approximation of initial region using multiple rectangles.

Unfortunately, use of multiple rectangles, as in Fig. 10.5, further complicates the fan-out problem. As shown in Fig. 10.3, use of fixed-duration flow tubes results in a fan-out from the goal region to multiple initial regions, one for each duration. If each such initial region is then approximated by multiple rectangles, as shown in Fig. 10.5, then there is a further fan-out, as shown in Fig. 10.6.

Due to this fan-out, an exhaustive computation of a large flow tube tree may not be tractable for tasks involving long activity sequences. A promising alternative is to compute just some of the flow tubes in such a tree, and then incrementally adjust them, as described in the next section.

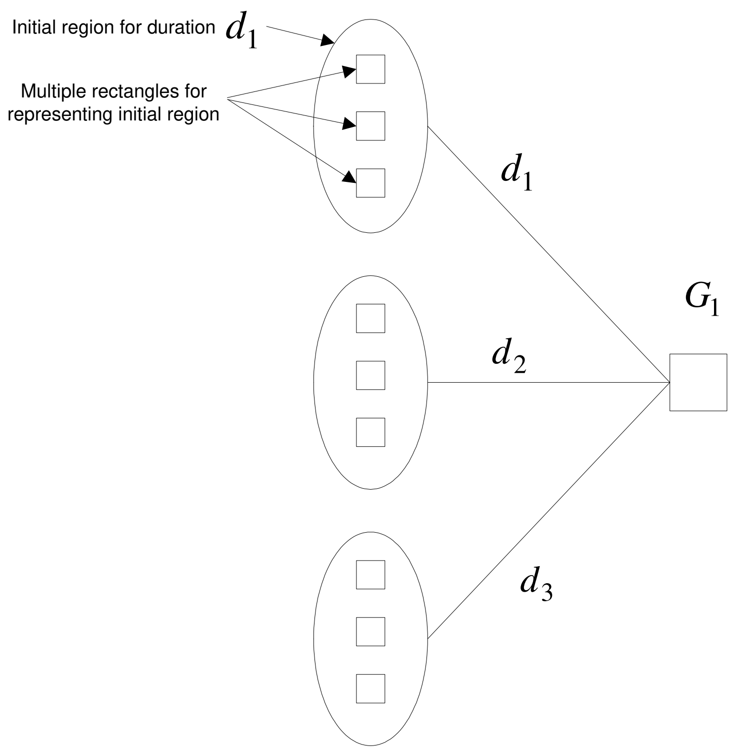


Fig. 10.6 – Fan-out due to multiple initial regions for each duration and multiple rectangles for each such initial region.

10.2 Incremental Adjustment of Flow Tubes

Computation of flow tubes is expensive in that it requires an SQP optimization, as described in Chapter 7. As discussed in Chapters 1 and 4, we use a plan compiler to pre-compute flow tubes so that this does not have to be done by the dispatcher at execution time.

If we were to use the flow tube representation described in the previous section, with trees of fixed-duration flow tubes, there will be significant fan-out, and exhaustive computation of a large flow tube tree may not be tractable. Because we would like to have as complete a flow tube representation as possible, it is important to investigate whether it would be feasible to pre-compute a tractable, but incomplete flow tube tree at compile time, and then adjust it, efficiently, at runtime, to fit new situations not anticipated at compile time.

Recall that Eq. 4.3 provides an analytic relation between start state, finish state, start time, finish time, and control parameter settings, for an SISO trajectory. This relation is of the form

$$\begin{aligned} y(t_2) &= f_1\left(y(t_1), \dot{y}(t_1), y_{set}, \dot{y}_{set}, kp, kd, t_D\right) \\ \dot{y}(t_2) &= f_2\left(y(t_1), \dot{y}(t_1), y_{set}, \dot{y}_{set}, kp, kd, t_D\right) \end{aligned} \tag{10.1}$$

(see also Eq. 7.1), where t_D is the duration. If the duration is fixed, as is the case for a fixed-duration flow tube, then this relation is linear. The relations for the GFT and GST trajectories, given by Eqs. 7.2 and 7.3, are then also linear. Thus, these equations are

$$\begin{aligned} y(D) &= f_1\left(y(B), \dot{y}(B), \left\langle y_{set}, \dot{y}_{set}, kp, kd \right\rangle(GFT), t_D\right) \\ \dot{y}(D) &= f_2\left(y(B), \dot{y}(B), \left\langle y_{set}, \dot{y}_{set}, kp, kd \right\rangle(GFT), t_D\right) \\ y(C) &= f_1\left(y(A), \dot{y}(A), \left\langle y_{set}, \dot{y}_{set}, kp, kd \right\rangle(GST), t_D\right) \\ \dot{y}(C) &= f_2\left(y(A), \dot{y}(A), \left\langle y_{set}, \dot{y}_{set}, kp, kd \right\rangle(GST), t_D\right) \end{aligned} \tag{10.2}$$

These equations linearly relate goal and initial regions. Thus, if a goal region is known and is shifted, these equations can be used to easily determine the corresponding shift in the initial region, or regions. Alternatively, they can be used to easily determine the shift in a goal region, given a shift in the initial region.

This has significant implications for the hybrid dispatcher. It suggests that, with relatively little effort, the dispatcher could adjust a flow tube that is almost right for the particular situation. It is almost right in that it has initial regions near the current state, but not so near that the current state is in the initial region. By incrementally shifting the flow tube, the tube can become useable for the current situation, if the shift is such that the current state is in the shifted flow tube's initial region and if the shifted flow tube's goal region fits inside an initial region of the successor activity. This allows the dispatcher to utilize the increased completeness of the representation, and thereby, to continue execution, rather than aborting the plan.

To get an idea of how useful this shifting capability is, consider a flow tube tree for a single activity, expanded back from a goal region, as shown in Fig. 10.7.

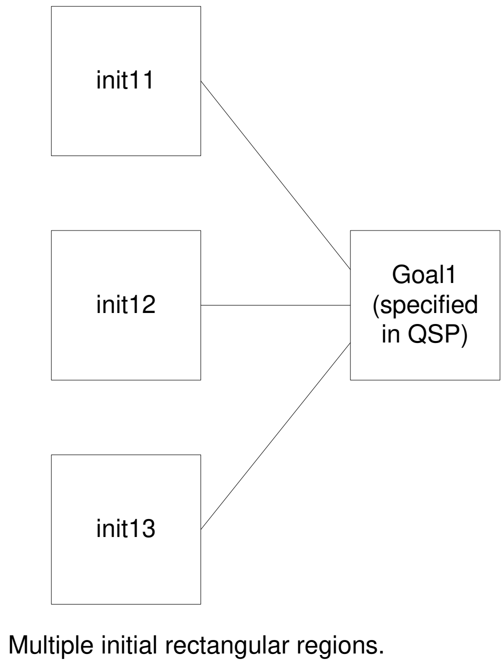


Fig. 10.7 – Flow tube tree for a single activity

As mentioned in the previous section, a significant problem with a fully compiled approach is the fan-out, depicted in Fig. 10.6. For example, if we wanted to extend the single activity flow tube tree shown in Fig. 10.7 to include flow tubes for a predecessor activity, we would have to expand flow tubes back from each of the initial regions init11, init12, and init13, resulting in a flow tube tree similar to the one shown in Fig. 10.3.

With the shifting capability, this problem is avoided, because the entire tree does not have to be expanded back. Consider Fig. 10.8, which shows flow tube trees for two, activities. Suppose that Activity1 in Fig. 10.8 is the successor to Activity2. Note that the flow tubes for these activities are not connected at compile time, that is, the region Goal2 is not a subset of regions init11, init12, and init13. This appears to violate the requirement stated in Section 5.1.3, that a flow tube goal region must be a subset of the successor's initial region. Note, however, that this violation is only at compile time. Because the Goal2, and its associated tree, can be shifted at runtime, Goal2 doesn't have to fit inside one of the initial regions init11, init12, or init13 at compile time, as long as it can be shifted appropriately at runtime. Thus, at runtime, the requirement of Section 5.1.3 is not violated. Thus, the fan-out problem is solved.

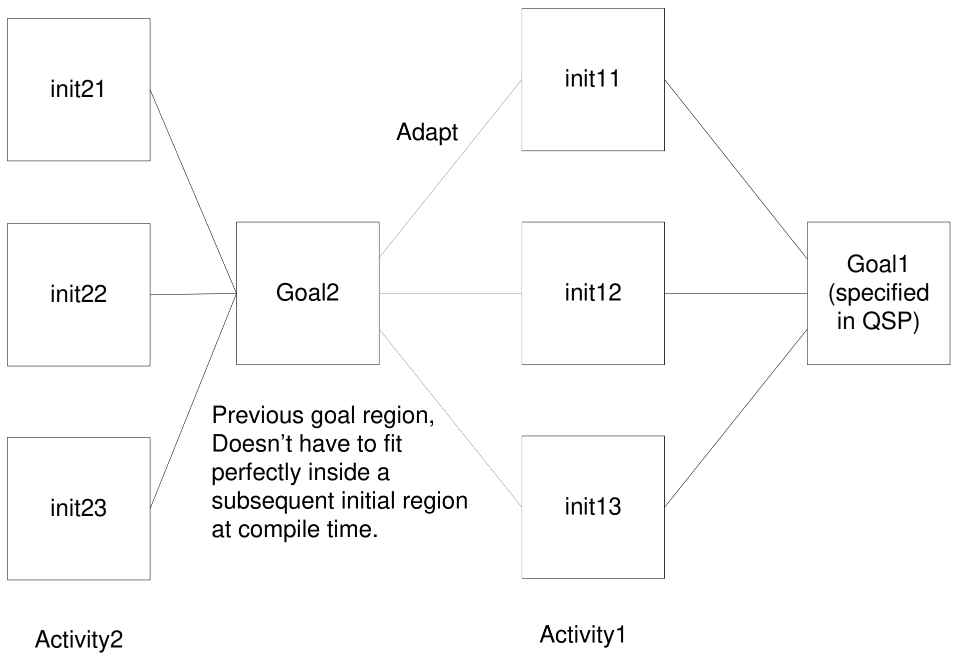


Fig. 10.8 – Shifting the goal of a predecessor activity to fit within the initial region of its successor

How should the dispatcher shift a flow tube tree, such as the one for Activity2 in Fig. 10.8? In particular, towards which initial region of Activity1 should Goal2 be shifted? Suppose the dispatcher is just beginning to execute Activity2. The current trajectory state may or may not be within init21, init22, or init23. The dispatcher must shift the flow tube tree for Activity2 such that the current trajectory state is in one of these initial regions, and such that Goal2 is a subset of init11, init12, or init13. If this is not possible, then plan execution fails. If it is possible, then there is a fully connected flow tube path from the current trajectory state to Goal1, and the plan is guaranteed to execute successfully if there are no further disturbances.

In shifting Goal2 so that it is a subset of init11, init12, or init13, the dispatcher is performing a runtime search. Note, however, that this search is efficient because the shift operation is fast. Each shift operation involves simply evaluating the analytical solution of a set of linear equations (Eq. 10.2). This is much faster than a runtime computation of a flow tube, which requires the solution of an optimization problem, as described in Chapter 7.

The runtime search performed by the dispatcher becomes more computationally intensive for longer activity sequences. Fig. 10.9 shows flow tube trees for a sequence of three activities. The trees are not connected at compile time; the system relies on runtime adjustments to connect them. Suppose that the dispatcher is just starting to execute the first activity in this sequence. The dispatcher must search to find a flow tube path from the current state to Goal1. Thus, the dispatcher must check whether the Goal2 and Goal3 trees can be shifted such that the current trajectory state is in one of the initial regions init31, init32, and init33, that Goal3 is a subset of init21, init22, or init23, and that Goal2 is a subset of init11, init12, or init13.

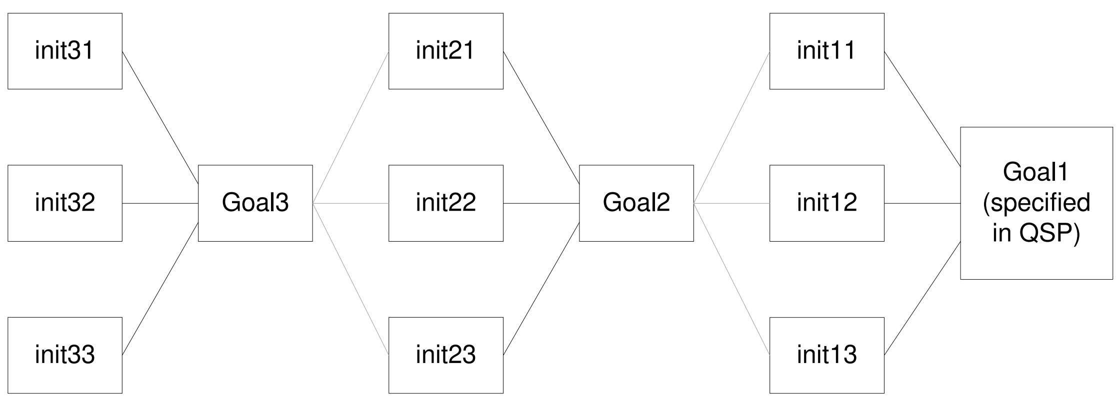


Fig. 10.9 – To achieve Goal1, the dispatcher must shift the trees for Goal2 and Goal3.

The shifting capability allows a sparse set of flow tube trees to cover the same region of state space as a fully expanded, fully connected flow tube tree. It allows for intermediate goals, where the goal regions don't have to, necessarily, be inside the initial regions of successors. With this approach, it is harder to make compile-time guarantees about success, because there isn't, necessarily, an unbroken path at compile time, from an initial region to a goal region. However, due to the speed of the runtime search, it is still possible to tell, very quickly, at runtime, whether a path from the current state to the goal exists. Thus, the key requirement of detecting imminent plan failure early, at runtime, is still satisfied. For many applications, this is more important than compile-time guarantees.

With this approach, the flow tube approximations are still sound in that they only admit feasible trajectories. However, the shifting capability makes the flow tube network adaptable to more situations, and therefore, more complete.

Another way to view flow tube trees, such as the one in Fig. 10.3, is as *motion primitives* [Schaal, 1999]. Motion primitives are basic, prototypical motions, which, when appropriately shifted, scaled and connected, form a complete motion that achieves a particular task goal. The shifting capability allows the flow tube trees representing the motion primitives to be adjusted and connected together in this way.

10.3 Learning

In recent years, there has been growing interest in the use of learning algorithms for motion control applications. Therefore, it makes sense to investigate whether learning algorithms could be applied to enhance the capabilities provided in this thesis.

One popular approach is to build a detailed, high-fidelity model of the robot to be controlled, and then to learn a control policy by running many thousands of simulations off-line, in combination with some type of reinforcement learning algorithm [Ng, 2003]. After the policy is computed off-line, it is loaded onto the actual robot. If the simulation used to learn the policy is accurate enough, this approach works well. For example, this approach was used to control an autonomous helicopter, capable of inverted flight [Ng, 2003].

Note that this approach is similar to the one used in this thesis in that an off-line optimization is used to compute a control policy. In fact, as long as an off-line model is used, and there are no stringent time constraints on speed of policy computation, then a wide variety of algorithms can be used to compute the control policy. With a reinforcement learning algorithm, the optimization is performed by running the model forward in time in order to predict and then evaluate a future state, given a candidate policy. Similarly, with an SQP algorithm, the model is used to evaluate multiple candidate policies and choose the best one. An important difference between these two approaches is that an SQP algorithm enforces all constraints at all times, providing it significant guidance in computing a solution. Therefore, an SQP algorithm may well be a more efficient method for computing a control policy than reinforcement learning, which requires running a simulation for a period of time, and then afterwards evaluating its outcome, to check whether constraints have been violated.

A completely different approach to learning is to perform the experiments that guide the learning process on a real robot, rather than a simulation. These experiments must be performed in real time, rather than faster than real time, as may be possible with a simulation. Given that the goal is to compute an adequate solution in a reasonable amount of time, the use of real time experiments on real robots imposes a limit on the number of such experiments that can be performed. Therefore, with this approach, the learning algorithm must either be more efficient than with a simulation-based approach, or there has to be less to learn. Thus, while simulation-based methods allow for brute force approaches, more guidance is required when learning directly on a real robot. Learning with a real robot works best when an adequate solution is almost known, a priori.

This approach was used recently to learn a control policy for a passive-dynamic walker that had been augmented with a minimal set of actuators [Tedrake, 2004]. This work is a perfect example of an approach where learning works because the system almost knows the solution before any learning. In this case, the system works well without learning because it is a passive-dynamic walker; it is able to walk down an incline without any learning, and without any actuation at all. The learning algorithm enhances this capability by computing a control policy for the actuators so that the robot is able to walk on level ground, not just on an incline. This enhancement is important, but the learning only works because the passive mechanism already "knows" the basic walking motions. After the system learns to walk on level terrain, the learning algorithm remains active in order to adapt to changes in terrain characteristics. This is an important capability; the learning process remains continuously active in order to allow for adaptation to changes in the environment.

For the problems addressed in this thesis, on-line learning algorithms could be applied in three areas. First, a learning process could be used to determine the model parameters used in the dynamic virtual model controller, which was described in Chapter

8. Second, a learning process could be used to compute flow tubes, and to improve the search of flow tube networks described in Section 10.2. Third, learning could be used to develop a policy for foot placement when traversing difficult terrain.

A capability for learning inertial parameters of the dynamic model used in the dynamic virtual model controller would improve the performance of the feedback linearizing component of the controller. This would result in more accurate feedforward control signals, resulting in less tracking error, and therefore, less reliance on corrective signals from the sliding control feedback component. Learning of the inertial parameters would be accomplished using hybrid mode estimation techniques [Hofbauer and Williams, 2002; Funiak and Williams, 2004; Funiak, 2004].

A learning process for computing or adapting flow tubes, would continuously extend and adapt the flow tube networks computed using off-line optimization. One approach for this kind of learning is to use *policy gradient reinforcement learning* methods [Tedrake, 2004]. These methods are well suited for learning continuous quantities, such as the dimensions of initial and goal regions of flow tubes. Furthermore, they allow for incorporation of prior knowledge through appropriate choice of parametric control policy forms.

A learning process for guiding the search described in Section 10.2 involves learning a policy that makes discrete choices in a possibly large search space. One approach for this kind of learning is to use *hierarchical policy gradient* methods [Ghavamzadeh and Mahadevan, 2003]. With this approach, choices for connecting flow tube networks are modeled as semi-Markov decision processes. A hierarchical task/subtask graph decomposition, that divides the learning problem into smaller sub-problems, is used to manage the complexity of the large search space.

A learning process to develop a policy for foot placement when traversing difficult terrain would supplement, and possibly replace, the foot placement information specified in the qualitative state plan. This would further simplify the process of creating such a plan, since the user would no longer have to specify foot placement explicitly. The policy that is learned would have to take into account the state of the terrain surrounding the robot, as well as the robot's current foot placement state. A combination of hierarchical policy gradient and policy gradient reinforcement learning methods could be used for this type of learning application, because it involves learning a policy that makes hybrid discrete/continuous choices in a large search space.

10.4 Detailed Comparison with Trial Data

As part of this thesis investigation, we have collected an extensive set of human walking trial data. Multiple human test subjects were used for these trials, and data was collected for three, self-selected walking speeds (slow, medium, and fast).

This data includes joint angle position and velocity measurements for all degrees of freedom relevant to walking, including ankle, knee, and hip. It also includes position and orientation of body segments, including torso, upper leg, lower leg, and foot. This data was used to compute values for center of mass position [Popovic, et al., 2003].

The trial data includes force information as well, including detailed measurements of the ground reaction force vector exerted during single and double-support phases of walking. This ground reaction force data, as well as the other measurements, were used to generate estimates of joint torques.

For this thesis, we used this data to perform a preliminary comparison between trial data center of mass trajectories, and corresponding trajectories produced by our model, as presented in Chapter 9. In the future, a more thorough comparison is required, using multiple models with morphologies corresponding to each test subject. This will allow us to perform a detailed comparison of joint angle and center of pressure trajectories. Comparison of joint torques will be more difficult, because trial data joint torques must be estimated based on inertial parameters of the human test subjects. These inertial parameters must, themselves, be estimated, because they cannot be measured directly.

We have not, at this point, collected human trial data for disturbance tests. This would be an interesting extension, although it is more difficult than collecting data for normal walking, because it requires applying unanticipated force disturbances such as trips and pushes to human test subjects. One interesting experiment that could be performed easily with human test subjects would be single support podium balancing experiments, corresponding to the ones performed on the model, as described in Chapter 8. This would allow for a detailed evaluation of how humans use augment the ankle strategy with the moment strategy in order to maintain balance, as discussed in Chapters 1 and 3.

10.5 Biological Models

In this thesis, we have focused on investigating performance limits due to the biomechanics of the biped and its environment. We have not restricted the control architecture in any way. In particular, we have not required that it correspond to or mimic anatomical features of the human central nervous system. As a next step, it would be interesting to investigate whether aspects of our architecture have biological analogs, and whether biologically inspired control approaches could be incorporated into our overall approach.

A number of previous and current investigations have achieved planar bipedal balancing and locomotion using biologically inspired approaches. In one such approach, [Taga, 1995], a planar neuro-musculo-skeletal model was controlled using neural oscillators located at each joint. These oscillators generated control signals for muscle models, which implemented a kind of impedance control. This combination of oscillators and impedance controllers produced stable limit cycle gaits in the sagittal plane.

More recently, a recurrent integrator proportional integral derivative (RIPID) model of cerebro-cerebellar control was developed [Massaquoi, 1999], which achieved arm posture and movement control in the horizontal plane. This model uses a particularly simple mechanism for stabilizing long-loop proprioceptive feedback loops. A number of features of human arm control, both for intact and compromised cerebellar function, appear to be well described by the model. This model was then augmented with gain scheduling in order to achieve human upright balance control in the sagittal plane [Jo and Massaquoi, 2004]. This model demonstrated that realistic balance control is possible in the without the use of detailed, internal dynamic models, and it suggests that the cerebellum and cerebral cortex may contribute to balance control by such a mechanism. The model is currently being extended to demonstrate bipedal walking in the sagittal plane.

It would be interesting to investigate whether such an approach could be used to de-emphasize, or eliminate the dependence on explicit dynamic models in the dynamic virtual model controller. A gain scheduling approach, where the gain parameters are learned automatically, could reduce reliance on accurate estimation of inertial parameters in the dynamic model.

The hybrid dispatcher, described in Chapter 6, performs functions that may be analogous to ones performed by the Cerebellum. It would be interesting to investigate these analogies further. Biological evidence suggests that the cerebellum plays a major role in synchronization of motion. This is based, partly, on studies of movements of patients with impaired cerebellar function [Pellionisz ref.]. In particular, it would be interesting to investigate whether the Cerebellum is involved in speeding up or slowing down aspects of motions so that overall motion is synchronized. This would be analogous to the synchronization between SISO systems performed by the dispatcher (see Chapter 6). Another interesting similarity is the concept of a discrete time interval. As described in Chapter 6, the hybrid dispatcher operates at a basic clock rate of 20 hz, corresponding to a 50 ms cycle time. There is evidence [Fahdi, 2002], that the Cerebellum operates at a discrete clock rate, although the cycle time is approximately 100 ms rather than 50 ms. It would be interesting to investigate this similarity further, and its implications for synchronization of complex movements.

Finally, it is reasonable to hypothesize that there is some type of representation of motion targets in the human brain, probably in the primary motor areas of the Cerebral Cortex. Evidence suggests that sequences of such targets or waypoints are used to form complex motions [Bizzi, 1992]. This would additionally require some representation or indication that a target has been achieved, implying some representation of a goal region. Thus, it is reasonable to expect that flow tube networks are represented, in some form, in the human brain. It would be interesting to investigate how the brain represents and learns these flow tubes.

10.6 Implementation on a Real Biped

In this thesis, we have validated our control approach using a hi-fidelity simulated biped. A logical next step would be to try our approach on an actual biped. Unfortunately, bipedal robots are not, currently, readily available to the research community. One possibility might be to use M2, a 12 degree of freedom bipedal walking machine, previously built at the MIT Leg Lab by Gill Pratt and his students [Wired, 2001]. Another possibility might be to use ASIMO, a humanoid robot developed by Honda (see Chapter 1).

However, it is not clear that these robots have the actuation capability to perform the dynamic balancing movements required for the agile motion tasks described in this thesis (see Chapter 1). Such movements may require a new generation of lightweight, high power actuators, such as the series-elastic energy-efficient actuators currently under development in the Biomechatronics Research Group. Until these actuators become available, it may be best to continue with simulation studies. When these actuators do become available, a new biped could be built that is electro-mechanically designed to take full advantage of the unique energy storage and release capabilities of these actuators. The control architecture developed in this thesis would have to be adapted as well, but we do not anticipate that this would require significant revision of our approach.

10.7 Conclusion

In this thesis, we have presented a plan execution system for robotic bipeds that observes externally specified state-space and temporal constraints, as well as dynamic limitations of the biped plant. The system compensates for disturbances, and detects when a disturbance is large enough to cause plan execution failure.

The system accepts a qualitative state plan as input. This plan specifies goals and restrictions using state-space and temporal constraints. State-space constraints are used, for example, to specify foot placement restrictions, and goals for center of mass location. Temporal constraints are used to specify an acceptable time range during which a sequence of activities must be performed. Qualitative state plans were described and formally defined in Chapter 4.

We achieve successful execution of such plans through three key innovations. First, we have developed a dynamic virtual model controller to decouple and linearize the biped, and thus, to provide an abstracted biped that is easier to control than the actual one. This controller was described in Chapter 8. Second, the plan compiler component of our system, described in Chapter 7, computes sets of allowed state trajectories, based on the qualitative state plan specification, and taking into account dynamic limitations of the biped plant. These state trajectory sets are represented using a flow tube approximation, which is included in the qualitative control plan generated by the plan compiler. Qualitative control plans are described and formally defined in Chapter 5. These plans are executed using a hybrid dispatcher, which keeps state trajectories in flow tubes by adjusting a small set of control parameters for the abstracted biped, as described in Chapter 6. Third, our system uses a novel strategy that employs angular momentum to enhance translational controllability of the system's center of mass, as described in Chapters 3 and 8. This strategy is particularly useful for tasks where foot placement is constrained.

The ability of the system to compile a qualitative state plan and execute the resulting qualitative control plan was demonstrated in Section 9.1. Robustness to lateral disturbances was demonstrated using the tests described in Section 9.3. Appropriate use of angular momentum was shown to extend robustness to such disturbances, using the tests described in Section 9.6. The system's ability to recover from trip disturbances was demonstrated as well, as explained in Section 6.3.6.

Additional tests validated the system's performance over difficult terrain. The tests of Section 9.4 demonstrated the system's ability to use irregular stepping patterns, while walking dynamically, in order to quickly cover terrain where foot placement is constrained. The tests of Section 9.7 demonstrated the system's ability to maintain its balance while walking on soft or slippery ground.

The soccer ball kicking test, described in Section 9.5, demonstrated the system's ability to observe stringent temporal constraints. In Section 9.8, we analyzed the completeness of our flow tube representation, and discussed, in Section 10, possible approaches for making it more complete.

The execution of challenging motion tasks in unstructured environments by articulated robots, including humanoid ones, is an exciting area of research. We believe that our approach to this type of problem, as described in this thesis, is promising. Further work, in extending the completeness of the flow tube representation, and testing with real bipeds, will be needed in order to completely solve this type of problem.

Bibliography

[Allum and Pfaltz, 1985] Allum, J., Pfaltz, C. (1985) "Visual and Vestibular Contributions to Pitch Sway Stabilization in the Ankle Muscles of Normals and Patients with Bilateral Peripheral Vestibular Deficits" Experimental Brain Research, 58:82-94

[Allum and Honegger, 1992] Allum, J., Honegger, F. (1992) "A Postural Model of Balance-correcting Movement Strategies", Journal of Vestibular Research, 2:323-347

[Allum, Honegger, and Schicks, 1993] Allum, J., Honegger, F., Schicks, H. (1993) "Vestibular and Proprioceptive Modulation of Postural Synergies in Normal Subjects", Journal of Vestibular Research, 3:59-85

[AMTI] AMTI OR6-5 Biomechanics Platforms, http://www.amtiweb.com.

[Anderson and Pandy, 2001] F. C. Anderson and M. G. Pandy. Dynamic optimization of human walking. *Journal of Biomechanical Engineering*, Vol. 123, Oct. 2001

[Arakawa and Fukuda, 1997] T. Arakawa and T. Fukuda. Natural motion generation of biped locomotion robot using hierarchical trajectory generation method consisting of GA, EP layers. *IEEE International Conference on Robotics and Automation* (ICRA)

[Benvenuti and Farina, 2002] L. Benvenuti and L. Farina. Linear programming approach to constrained feedback control. *International Journal of Systems Science*, 33(1), 45-53

[Bertsekas, 2005] D. P. Bertsekas. Dynamic Programming and Suboptimal Control: A Survey from ADP to MPC. CDC Proceedings, Seville, Spain

[Bhatia and Frazzoli, 2004] A. Bhatia and E. Frazzoli. Incremental Search Methods for Reachability Analysis of Continuous and Hybrid Systems,

[Bizzi et al, 1992] Bizzi, E., Hogan, N., Mussa-Ivaldi, F. A., Giszter, S. (1992) "Does the nervous system use Equilibrium-point Control to Guide Single and Multiple Joint Movements?", Behavioral and Brain Sciences, 15:603-613

[Bradley and Zhao, 1993] E. Bradley and F. Zhao. Phase-space control system design. *Control Systems*, 13(2),39-46 April, 1993

[Brown, 1987] G. A. Brown. Determination of Body Segment Parameters Using Computerized Tomography and Magnetic Resonance Imaging MIT Master of Science in Mechanical Engineering Thesis.

[Casagrande et al., 2004] A. Casagrande, A. Balluchi, L. Benvenuti, A. Policriti, T. Villa, and A. Sangiovanni-Vincentelli. Improving Reachability Analysis of Hybrid Automata for Engine Control.

[Clauser et al., 1969] C. E. Clauser, J. T. Mcconville, and J. W. Young. Weight, Volume, and Center of Mass Segments of the Human Body. Technical Report AMRL Tech. Report 69-70, Wright-Patterson Air Force Base, OH.

[Collins et al., 2001] S. Collins, M. Wisse, A. Ruina. A three-dimensional passive-dynamic walking robot with two legs and knees. *International Journal of Robotics Research*

[Collins et al., 2005] S. Collins, A. Ruina, R. Tedrake, M Wisse. Efficient bipedal robots based on passive-dynamic walkers. Science, 307:1082-1085, February

[Craig, 1989] Craig, J. J., (1989) "Introduction to Robotics: Mechanics and Control", Reading, Massachusetts, Addison-Wesley, pp. 152 – 180

[Cutler and Ramaker, 1979] C. R. Cutler and B. L. Ramaker. Dynamic Matrix Control – a computer control algorithm. AIChE National Meeting, Houston, TX.

[Dechter et al., 1991] R. Dechter, I. Meiri, and J Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61-95, May 1991

[Effinger et al., 2005] R. Effinger, A. Hofmann, B. Williams. Progress Towards Task-Level Collaboration between Astronauts and their Robotic Assistants. ISAIRAS

[Featherstone, 1987] Featherstone, R., 1987, "Robot Dynamic Algorithms", Boston, Massachusetts, Kluwer Academic Publishers, pp. 155 – 172

[Frazzoli, 2001] E. Frazzoli. Robust Hybrid Control for Autonomous Vehicle Motion Planning. Ph.D. Thesis, MIT

[Funiak et al., 2004] S. Funiak, L. Blackmore, B. Williams. Gaussian Particle Filtering for Concurrent Hybrid Models with Autonomous Transitions. Submitted to Journal of AI Research.

[Garcia and Prett, 1986] C. Garcia and D. Prett Advances in Industrial Model-Predictive Control. *Proceedings of the Third International Conference on Chemical Process Control* Elsevier

[Goswami et al., 1996] A. Goswami, B. Espiau, A. Keramane. Limit cycles and their stability in a passive bipedal gait. *IEEE International Conference on Robotics and Automation* (ICRA)

[Goswami, 1999] A. Goswami. Postural stability of biped robots and the foot rotation indicator (FRI) point. *International Journal of Robotics Research*, July/August 1999

[Hirai et al., 1997] Hirai K., 1997, "Current and Future Perspective of Honda Humanoid Robot" *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems* Grenoble, France:IEEE, New York, NY, USA. pp. 500-508.

[Hirai et al., 1998] K. Kirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of Honda humanoid robot. *IEEE International Conference on Robotics and Automation* (ICRA)

[Hofbaur and Williams, 2004] Hybrid Estimation of Complex Systems. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics

[Hofmann et al., 2002] A. Hofmann, M. Popovic, H. Herr. Humanoid Standing Control: Learning from Human Demonstration. *Journal of Automatic Control*, 12(1), 16–22

[Hofmann et al., 2004] A. Hofmann, S. Massaquoi, M. Popovic, and H. Herr. A sliding controller for bipedal balancing using integrated movement of contact and non-contact limbs. *Proc. International Conference on Intelligent Robots and Systems* (IROS). Sendai, Japan

[Hogan, 1985] N. Hogan. Impedance Control: An Approach to Manipulation – Part I: Theory, *Journal of Dynamic Systems, Measurement, and Control*, 107:1-7

[Hu, 2000] J. Hu. Stable Locomotion Control of Bipedal Walking Robots: Synchronization with Neural Oscillators and Switching Control

[Kagami, et al., 2001] Kagami, S., Kanehiro, F., Tamiya, Y., Inaba, M., Inoue, H., 2001, "AutoBalancer: An Online Dynamic Balance Compensation Scheme for Humanoid Robots", in "Robotics: The Algorithmic Perspective", Donald, B. R., Lynch, K. M., and Rus, D., editors, A. K. Peters Ltd. pp. 329 – 340

[Kailath, 1980] Linear Systems. Prentice-Hall.

[Kajita et al., 2001] S. Kajita, O. Matsumoto, and M. Saigo. Real-time 3D walking pattern generation for a biped robot with telescopic legs" *Proc. of the 2001 IEEE International Conference on Robotics and Automation*, 2001, pp. 2299-2306.

[Khatib et al., 2004] O. Khatib, L. Sentis, J. Park, J. Warren. *International Journal of Humanoid Robotics*, 1(1):1-15, March 2004

[Kim et al., 2001] P. Kim, B. C. Williams, M. Abramson. Executive Reactive Model-based Programs through Graph-based Temporal Planning. *International Joint Conference on Artificial Intelligence*, Seattle, WA

[Kuffner et al., 2001] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, H. Inoue. Motion Planning for Humanoid Robots Under Obstacle and Dynamic Balance Constraints. *Proceedings of the IEEE International Conference on Robotics and Automation* (ICRA)

[Kuffner et al., 2002] J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue. Dynamically-stable motion planning for humanoid robots. *Autonomous Robots*, vol. 12, No. 1, 105-118.

[Kuipers and Ramamoorthy, 2001] B. Kuipers, S. Ramamoorthy. Qualitative Modeling and Heterogeneous Control of Global System Behavior. Hybrid Systems Control Conference.

[Kurzhanski and Varaiya, 1999] A. Kurzhanski and P. Varaiya. Ellipsoidal Techniques for Reachability Analysis: Internal Approximation

[Kurzhanski and Varaiya, 2005] A. Kurzhanski and P. Varaiya. Ellipsoidal Techniques for Reachability Analysis of Discrete-Time Linear Systems

[Luenberger, 1989] D. Luenberger Linear and Nonlinear Programming. Addison-Wesley, Massachusetts.

[Lavalle et al., 2001] S. Lavalle, J. J. Kuffner. Randomized Kinodynamic Planning. *International Journal of Robotics Research*, May 2001

[Leaute, 2005] T. Leaute. Coordinating Agile Systems Through the Model-based Execution of Temporal Plans. Master's Thesis, MIT

[Leaute and Williams, 2005] T. Leaute, B. Williams. Coordinating Agile Systems Through the Model-based Execution of Temporal Plans. ICAPS, 2005

[Matlab a.] Matlab Function Reference, http://www.mathworks.com/access/helpdesk/help/techdoc/ref/ref.shtml Interpolation section, interp1.

[McGeer, 1990] T McGeer. Passive walking with knees. *IEEE International Conference on Robotics and Automation* (ICRA)

[Morris et al., 2001] P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. *Proceedings of the 17th International Joint Conference on A.I.* (IJCAI-01). Seattle (WA, USA).

[Muscettola et al., 1998] N. Muscettola, P. Morris, and I. Tsamardinos. Reformulating temporal plans for efficient execution. *Proc. Of Sixth Int. Conf. On Principles of Knowledge Representation and Reasoning*, 1998

[Muybridge, 1955] E. Muybridge. The Human Figure in Motion

[Nashner, 1981] L. M. Nashner. Analysis of Stance Posture in Humans. *Handbook of Behavioral Neurobiology* Vol. 5 Towe, A.L., Laschei, E.S., eds. Plenum Press pp. 527-565

[Nashner and McCollum, 1985] L. M. Nashner, G. McCollum. The organization of human postural movements: a formal basis and experimental synthesis, *The Behavioral and Brain Sciences.* 8, pp. 135-172.

[Ng et al., 2004] A. Ng. A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, E. Liang. Inverted autonomous helicopter flight via reinforcement learning. International Symposium on Experimental Robotics

[Nishiwaki et al., 1999] K. Nishiwaki, K. Nagasaka, M. Inaba, H. Inoue. Generation of reactive stepping motion for a humanoid by dynamically stable mixture of pre-designed motions. *IEEE International Conference on Robotics and Automation* (ICRA)

[Nishiwaki et al., 2001] K. Nishiwaki, T. Sugihara, S. Kagami, M. Inaba, H. Inoue. Online mixture and connection of basic motions for humanoid walking control by footprint specification. *IEEE International Conference on Robotics and Automation* (ICRA)

[Nishiwaki et al., 2002] K. Nishiwaki, S. Kogami, Y. Kuniyoshi, M. Inaba, and H. Inoue. Online Generation of Humanoid Walking Motion based on a Fast Generation Method of Motion Pattern that follows Desired ZMP. *Proc. of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, 2002, pp. 2684-2688.

[Paul, 1981] Paul, R. P., 1981, "Robot Manipulators", Cambridge, Massachusetts: The MIT Press, pp. 223–225

[Pellionisz, 1985] Tensorial brain theory in cerebellar modeling. In Cerebellar Functions, ed. J.R. Bloedel, J. Dichgans and W. Precht. Springer-Verlag

[Popovic and Witkin, 1999] Z. Popovic and A. Witkin. Physically based motion transformation. Siggraph 1999

[Popovic et al., 2004a] M. Popovic, A. Hofmann, H. Herr. Angular momentum regulation during human walking: biomechanics and control. *Proc. International Conference on Robotics and Automation*, (ICRA). New Orleans (LA, USA).

[Popovic et al., 2004b] M. Popovic, A. Hofmann, H. Herr. Zero spin angular momentum control: definition and applicability. (Humanoids). Los Angeles (CA, USA).

[Popovic et al., 2005] M. Popovic, A. Goswami, H. Herr. Ground Reference Points in Legged Locomotion: Definitions, Biological Trajectories, and Control Implications. International Journal of Robotics Research, 2005

[Pratt et al., 1996] J. Pratt, A. Torres, P. Dilworth, G. Pratt, 1996, Virtual Actuator Control, *Proc. International Conference on Intelligent Robots and Systems* (IROS)

[Pratt et al., 1997] J. Pratt, P. Dilworth, G. Pratt, Virtual Model Control of a Bipedal Walking Robot, *Proc. International Conference on Robotics and Automation* (ICRA)

[Pratt and Tedrake, 2005] J. Pratt, R. Tedrake. Velocity Based Stability Margins for Fast Bipedal Walking, International Seminal on Agile Robotic Motion, Heidelberg

[Raibert, 1986] Raibert, M. H., 1986, "Legged Robots that Balance", Cambridge, MA, MIT Press

[Richalet et al., 1978] J. Richalet, A. Rault, J. Testud, J. Papon. Model predictive heuristic control: applications to an industrial process. Automatica, 14.

[Rietdyk et al., 1999] S. Rietdyk, A. E. Patla, Winter, P.A., Ishac, M.E., and Little, C.E. (1999), "Kinetic strategies for balance recovery during medio-lateral perturbations of the upper body during standing" *Journal of Biomechanics* 32 pp. 1149-1158.

[Schaal, 1999] Schaal S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences* 3:233-242.

[Sentis and Khatib, 2004] L. Sentis, O. Khatib. Task-oriented Control of Humanoid Robots Through Prioritization. *IEEE-RAS/RSJ International Conference on Humanoid Robots,* Santa Monica, USA, November 2004.

[Slotine and Li, 1991] J. Slotine and W. Li. Applied Nonlinear Control. Ch. 6, Prentice Hall, NJ, USA

[Sugihara et al., 2002] T. Sugihara, Y. Nakamura, and H. Inoue. Realtime Humanoid Motion Generation through ZMP Manipulation based on Inverted Pendulum Control. *Proc. of the 2002 IEEE International Conference on Robotics and Automation*, 2002, pp. 1404-1409.

[Takanishi et al., 1985] A. Takanishi, M. Ishida, Y. Yamazaki, I. Kato. The realization of dynamic walking by the biped robot WL-10RD. In *International Conference on Advanced Robotics*, Tokyo, pages 459-466, 1985

[Tedrake, 2004] Applied Optimal Control for Dynamically Stable Legged Locomotion. Ph.D. Thesis, MIT

[Tilley and Dreyfuss, 1993] A. R. Tilley, H. Dreyfuss. The measure of man and woman. Whitney Library of Design, an imprint of Watson-Guptill Publications, New York.

[Vestal, 2001] S. Vestal. A New Linear Hybrid Automata Reachability Procedure. HSCC

[Vicon a.] Vicon Motion Systems, http://www.vicon.com.

[Vicon b.] Vicon Bodybuilder Software,
http://www.vicon.com/main/technology/bodybuilder.html

[Vukobratovic and Juricic, 1969] M. Vukobratovic and D. Juricic. Contribution to the Synthesis of biped Gait. *IEEE Transactions on Bio-Medical Engineering*, Vol. BME-16, No. 1, 1969, pp. 1 – 6.

[Walcott, 2004] A. Walcott. Unifying Model-Based Programming and Randomized Path Planning Through Optimal Search, Master's Thesis, MIT

[Westervelt et al., 2004] E. R. Westervelt, G. Buche, and J.W.Grizzle. Inducing Dynamically Stable Walking in an Underactuated Prototype Planar Biped. ICRA, New Orleans

[Williams, 1984] The Use of Continuity in Qualitative Physics. Proceedings of the National Conference on Artificial Intelligence. Austin. TX

[Williams and Nayak, 1997] B. Williams and P. Nayak. A Reactive Planner for a Model-based Executive. *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI, 1997)

[Pratt and Williamson, 1995] G. Pratt, M. Williamson. Series Elastic Actuators. Proceedings of IROS, Pittsburgh, PA

[Winters, 1990] D. A. Winters. Biomechanics and Motor Control of Human Movement. John Wiley & Sons, Inc., New York.

[Wolfram, 2005] http://mathworld.wolfram.com/DeltaFunction.html

[Yamaguchi et al., 1996] J. Yamaguchi, N. Kinoshita, A. Takanishi, and I. Kato. Development of a dynamic biped walking system for humanoid – development of a biped walking robot adapting to the human's living floor. *IEEE International Conference on Robotics and Automation* (ICRA)

[Yamaguchi et al., 1999] Yamaguchi, J., Soga, E., Inoue, S., Takanishi, A., 1999, "Development of a Bipedal Humanoid Robot -Control Method of Whole Body Cooperative Dynamic Biped Walking", ICRA '99, IEEE, pp. 368 - 374

[Yokoi et al., 2001] K. Yokoi, F. Kanehiro, K. Kaneko, K. Fujiwara, S. Kajita, and H. Hirukawa. A Honda humanoid robot controlled by AIST software, *Proc. of the 2001 IEEE-RAS International Conference on Humanoid Robots*, 2001, pp. 259-264.

Appendix A – Homogeneous Transforms

Homogeneous transforms are used extensively in robot kinematics [Paul, 1982]. Kinematics are used to transform between coordinate systems of the various articulated linkages in a robot. For example, in a manipulator, kinematics are used to determine end-effector position for a given joint angle position vector. In the controller described in Chapter 8, homogeneous transforms are used to transform from robot joint to workspace coordinates. For example, given the robot's joint angle position vector, these transforms are used to compute the biped's CM position or swing foot position and orientation.

In a homogeneous coordinate representation of objects in 3-space, a 4-element vector is used. The first 3 elements are the usual x, y, z coordinates, and the 4th element is a scale factor. The actual x, y, z coordinates of the object are computed by dividing each of the first 3 elements by the scale factor. The motivation for this is that it allows translation transformation matrices to be applied by multiplication (rather than addition) to achieve the transformation. This makes them consistent with rotation matrices, as will be seen shortly.

A homogeneous transformation is a 4x4 matrix that can represent translation, rotation, or some combination of these. It can represent scaling as well, although we do not use this feature; the scale factor in all of our homogeneous transformations is 1. Given a point u that has position u1 in local coordinate frame 1, its position, u0, in the global coordinate frame is given by

$$u_0 = {}_0X_1 u_1 \qquad \text{A1}$$

Here, the subscript 0 indicates the global coordinate frame, and the transformation ${}_0X_1$ converts from coordinate frame 1 to coordinate frame 0, the global coordinate frame. The inverse transformation,

$${}_1X_0 = {}_0X_1^{-1} \qquad \text{A2}$$

converts from the global coordinate frame to frame 1

$$u_1 = {}_1X_0 u_0 \qquad \text{A3}$$

Transformations can be combined into sequences of transformations. For example, a point u2 in local coordinate frame 2 is converted to a vector in global coordinates by

$$u_0 = {}_0X_2 u_2 \qquad \text{A4}$$

where

$$ {}_0X_2 = {}_0X_1\,{}_1X_2 \qquad \text{A5} $$

1 Translation Transformations

A translational transform is used to represent the translational shift between coordinate systems. It is a 4x4 matrix of the form

$$ {}_0X_1 = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{A6} $$

where a,b,c are the translations in the x, y, and z directions, respectively. For example, suppose that coordinate frame 1 is translated in the x direction by 5. Suppose that u1 is the origin of coordinate frame 1, in frame 1 coordinates:

$$ u_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \text{A7} $$

The equivalent vector, in frame 0 coordinates is then given by Eq. A1, where

$$ {}_0X_1 = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{A8} $$

so

$$ u_0 = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \text{A9} $$

Similarly, suppose that point p is the origin of frame 0 (the global coordinate frame). Then, its position, p0, in frame 0 coordinates is

$$p_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \text{A10}$$

and its position, p1, in frame 1 coordinates is

$$p_1 = {}_1X_0 \; p_0 = \begin{bmatrix} 1 & 0 & 0 & -5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -5 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \text{A11}$$

(as would be expected). Here,

$${}_1X_0 = {}_0X_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & -5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{A12}$$

Note that with homogeneous transformations, translation can be achieved by multiplying the translation transformation vector by the vector being transformed. If homogeneous transformations weren't used (if ordinary 3-element vectors were used) this wouldn't be possible; translation would have to be achieved by a vector add, which would make it different from rotation transformations, and would make it difficult to combine the two.

2 Rotation Transformations

A rotation transform is used to represent a change in orientation between coordinate systems. A rotation transformation can also be represented as a 4x4 matrix [Paul, 1982]. The transformations corresponding to rotations about the x, y, and z axes, by an angle θ_1, are

$${}_0X_1 = Rotx(\theta_1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_1 & -s_1 & 0 \\ 0 & s_1 & c_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{A13}$$

$$
{}_0X_1 = Roty(\theta_1) = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ 0 & 1 & 0 & 0 \\ -s_1 & 0 & c_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{A14}
$$

$$
{}_0X_1 = Rotz(\theta_1) = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{A15}
$$

These can be multiplied together, successively. The general form of a rotation transformation matrix, incorporating any set of rotations, is

$$
{}_0X_1 = \begin{bmatrix} n_x & o_x & a_x & 0 \\ n_y & o_y & a_y & 0 \\ n_z & o_z & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{A16}
$$

Note that there are many different ways to represent rotation of an object (see other memos on this). For example, quaternions, Euler angles, and other such representations require fewer parameters than the 9 in the above example. Thus, the 9 parameters are not independent, but have stringent constraints between them. In particular, the vectors n, o, and a are orthogonal.

3 General Translation and Rotation Transformations

Homogeneous translation and rotation matrices can be combined repeatedly by multiplication. The general form of the result is a translation matrix of the form

$$
{}_0X_1 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{A17}
$$

Appendix B – Jacobian Computation

1 Differential Relationships and Computation of Jacobian

As described in Appendix A, homogeneous transforms are used to compute workspace positions from joint angle positions. Similarly, Jacobians [Paul, 1982] are used to compute incremental changes in workspace position to incremental changes in joint angle position. Hence, they can be used to compute workspace velocities from joint space velocities.

In order to understand how Jacobians are computed, we first investigate differential transforms. These transforms relate incremental changes in one coordinate frame to incremental changes in another. As we will see, computation of differential transforms is a crucial step in computing columns of Jacobian matrices.

Let

T be a coordinate transformation
dT be a differential translation and rotation of T
Δ be a differential translation and rotation transformation

then

$$dT = \Delta T \qquad \text{B1}$$

In this equation, T transforms from local to global coordinates, and Δ implements an incremental translation and rotation transformation, in terms of global coordinates. Alternatively,

$$dT = T^T\Delta \qquad \text{B2}$$

Here, ${}^T\Delta$ implements the incremental translational and rotation transformation in terms of local coordinates. The result is transformed to global coordinates by T.

Now, Δ is of the form

$$\Delta = \begin{bmatrix} 0 & -\delta_z & \delta_y & d_x \\ \delta_z & 0 & -\delta_x & d_y \\ -\delta_y & \delta_x & 0 & d_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{B3}$$

where the upper left 3x3 matrix is a differential rotation [Paul, 1982, Section 4.3], and the vector $\begin{bmatrix} d_x & d_y & d_z \end{bmatrix}^T$ is a differential translation. The lower right (4,4) element is 0 because it represents a derivative of the constant scale factor 1 in the homogeneous transform. This can also be represented as a spatial vector

$$D = \begin{bmatrix} d_x \\ d_y \\ d_z \\ \delta_x \\ \delta_y \\ \delta_z \end{bmatrix} \qquad \text{B4}$$

where the first three elements represent differential translation, and the second three elements represent differential rotation.

The question now is, given Δ, what is ${}^T\Delta$? This is an important question in the derivation of the Jacobian because ${}^T\Delta$ transformations, expressed in the form of Eq. B4, are used as the columns of Jacobian matrices. Combining Eqs. B1 and B2 yields

$$\Delta T = T\,{}^T\Delta \qquad \text{B5}$$

so

$${}^T\Delta = T^{-1}\Delta T \qquad \text{B6}$$

On the right-hand side of this equation, T transforms from local to global coordinates, Δ implements the incremental translation and rotation transformation, in global coordinates, and T^{-1} transforms the result back to local coordinates. The result of this, the incremental translation and rotation transformation in local coordinates, could then be used in Eq. B2, for example.

As described previously, the general form for a transformation, T, is

$$T = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{B7}$$

Combining Eqs. B3 and B7 yields

$$\Delta T = \begin{bmatrix} 0 & -\delta_z & \delta_y & d_x \\ \delta_z & 0 & -\delta_x & d_y \\ -\delta_y & \delta_x & 0 & d_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{B8}$$

$$= \begin{bmatrix} -\delta_z n_y + \delta_y n_z & -\delta_z o_y + \delta_y o_z & -\delta_z a_y + \delta_y a_z & -\delta_z p_y + \delta_y p_z + d_x \\ \delta_z n_x - \delta_x n_z & \delta_z o_x - \delta_x o_z & \delta_z a_x - \delta_x a_z & \delta_z p_x - \delta_x p_z + d_y \\ -\delta_y n_x + \delta_x n_y & -\delta_y o_x + \delta_x o_y & -\delta_y a_x + \delta_x a_y & -\delta_y p_x + \delta_x p_y + d_z \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} (\delta \times n)_x & (\delta \times o)_x & (\delta \times a)_x & ((\delta \times p) + d)_x \\ (\delta \times n)_y & (\delta \times o)_y & (\delta \times a)_y & ((\delta \times p) + d)_y \\ (\delta \times n)_z & (\delta \times o)_z & (\delta \times a)_z & ((\delta \times p) + d)_z \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The inverse transform of T is

$$T^{-1} = \begin{bmatrix} n_x & n_y & n_z & -p \cdot n \\ o_x & o_y & o_z & -p \cdot o \\ a_x & a_y & a_z & -p \cdot a \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{B9}$$

This can be easily verified by multiplying Eq. B9 with T (Eq. B7); the result will be I, the identity matrix. We instantiate our mapping from Δ to ${}^T\Delta$ by substituting Eq. B9, and the result from Eq. B8, into Eq. B6, yielding

$${}^T\Delta = T^{-1}\Delta T \qquad \text{B10}$$

$$= \begin{bmatrix} n_x & n_y & n_z & -p \cdot n \\ o_x & o_y & o_z & -p \cdot o \\ a_x & a_y & a_z & -p \cdot a \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (\delta \times n)_x & (\delta \times o)_x & (\delta \times a)_x & ((\delta \times p) + d)_x \\ (\delta \times n)_y & (\delta \times o)_y & (\delta \times a)_y & ((\delta \times p) + d)_y \\ (\delta \times n)_z & (\delta \times o)_z & (\delta \times a)_z & ((\delta \times p) + d)_z \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} n \cdot (\delta \times n) & n \cdot (\delta \times o) & n \cdot (\delta \times a) & n \cdot ((\delta \times p) + d) \\ o \cdot (\delta \times n) & o \cdot (\delta \times o) & o \cdot (\delta \times a) & o \cdot ((\delta \times p) + d) \\ a \cdot (\delta \times n) & a \cdot (\delta \times o) & a \cdot (\delta \times a) & a \cdot ((\delta \times p) + d) \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The elements of Eq. B10 are of the form of vector triple products

$$a \cdot (b \times c) \qquad \text{B11}$$

We can exploit the properties of triple products to simplify Eq. B10 substantially. First, triple products have the property that

$$a \cdot (b \times c) = -b \cdot (a \times c) = b \cdot (c \times a) \qquad \text{B12}$$

In addition, if any two elements of a vector triple product are the same, the value of the triple product is 0. Thus, the diagonal terms of Eq. B10 are all 0. Additionally, rearranging the terms in Eq. B10 using the identities from Eq. B12 yields

$$^T\Delta = \begin{bmatrix} 0 & -\delta \cdot (n \times o) & \delta \cdot (a \times n) & \delta \cdot (p \times n) + d \cdot n \\ \delta \cdot (n \times o) & 0 & -\delta \cdot (o \times a) & \delta \cdot (p \times o) + d \cdot o \\ -\delta \cdot (a \times n) & \delta \cdot (o \times a) & 0 & \delta \cdot (p \times a) + d \cdot a \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{B13}$$

Next, since n, o, and a are orthogonal,

$$\begin{aligned} n \times o &= a \\ a \times n &= o \\ o \times a &= n \end{aligned} \qquad \text{B14}$$

Eq. B13 simplifies to

$$^T\Delta = \begin{bmatrix} 0 & -\delta \cdot a & \delta \cdot o & \delta \cdot (p \times n) + d \cdot n \\ \delta \cdot a & 0 & -\delta \cdot n & \delta \cdot (p \times o) + d \cdot o \\ -\delta \cdot o & \delta \cdot n & 0 & \delta \cdot (p \times a) + d \cdot a \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{B15}$$

Now, from Eq. B3, we also have

$$^T\Delta = \begin{bmatrix} 0 & -{}^T\delta_z & {}^T\delta_y & {}^Td_x \\ {}^T\delta_z & 0 & -{}^T\delta_x & {}^Td_y \\ -{}^T\delta_y & {}^T\delta_x & 0 & {}^Td_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{B16}$$

Equating elements between Eqs. B15 and B16 yields

$$
\begin{aligned}
{}^T d_x &= \delta(p \times n) + d \cdot n \\
{}^T d_y &= \delta(p \times o) + d \cdot o \\
{}^T d_z &= \delta(p \times a) + d \cdot a \\
{}^T \delta_x &= \delta \cdot n \\
{}^T \delta_y &= \delta \cdot o \\
{}^T \delta_z &= \delta \cdot a
\end{aligned} \qquad \text{B17}
$$

These equations can be expressed in matrix form as

$$
\begin{bmatrix} {}^T d_x \\ {}^T d_y \\ {}^T d_z \\ {}^T \delta_x \\ {}^T \delta_y \\ {}^T \delta_z \end{bmatrix} = \begin{bmatrix} n_x & n_y & n_z & (p \times n)_x & (p \times n)_y & (p \times n)_z \\ o_x & o_y & o_z & (p \times o)_x & (p \times o)_y & (p \times o)_z \\ a_x & a_y & a_z & (p \times a)_x & (p \times a)_y & (p \times a)_z \\ 0 & 0 & 0 & n_x & n_y & n_z \\ 0 & 0 & 0 & o_x & o_y & o_z \\ 0 & 0 & 0 & a_x & a_y & a_z \end{bmatrix} \begin{bmatrix} d_x \\ d_y \\ d_z \\ \delta_x \\ \delta_y \\ \delta_z \end{bmatrix} \qquad \text{B18}
$$

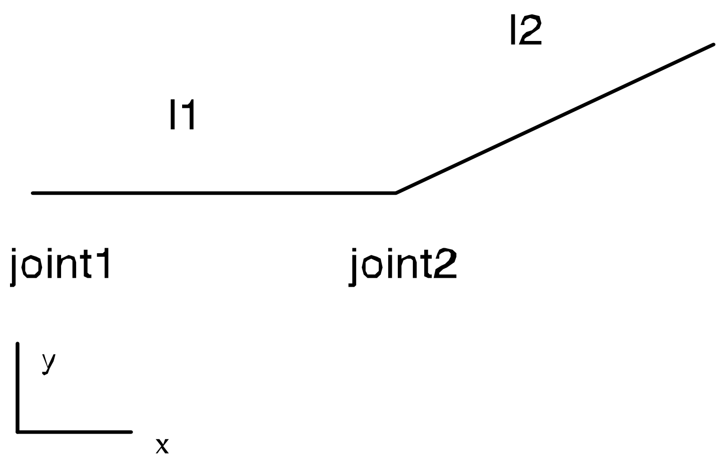
Using triple product properties, Eq. B17 can also be written as

$$
\begin{aligned}
{}^T d_x &= n \cdot ((\delta \times p) + d) \\
{}^T d_y &= o \cdot ((\delta \times p) + d) \\
{}^T d_z &= a \cdot ((\delta \times p) + d) \\
{}^T \delta_x &= n \cdot \delta \\
{}^T \delta_y &= o \cdot \delta \\
{}^T \delta_z &= a \cdot \delta
\end{aligned} \qquad \text{B19}
$$

Thus, Eq. B15 gives us ${}^T \Delta$, and Eq. B19 gives us this transform as a spatial vector. As we will see in the next section, we will use this form to compute columns of Jacobian matrices.

2 Simple Manipulator Jacobian

We now use a simple example to illustrate how the differential relationships derived in the previous section are used to compute Jacobians. Consider a simple two-link manipulator, as shown in the following diagram.



The transform for this manipulator, which gives position and orientation of the end of l_2 in global coordinates, is

$$T_2 = A_1 A_2 \qquad \text{B20}$$

where

$$A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & l_1 c_1 \\ s_1 & c_1 & 0 & l_1 s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} c_2 & -s_2 & 0 & l_2 c_2 \\ s_2 & c_2 & 0 & l_2 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The transform A_1 gives the position and orientation of the end of l_1 in global coordinates, and A_2 gives the position and orientation of the end of l_2 in the local coordinate frame of link l_1.

We would now like to compute the Jacobian in order to determine how the position and orientation of the end of l_2 changes as joint angles change. Consider an incremental change in the angle position of Joint 2. Eq. B3 can be used to determine the differential transform for Joint 2. Because, Joint 2 allows only rotation about the z axis, Eq. B3 simplifies to

$$\Delta = \begin{bmatrix} 0 & -\delta_z & 0 & 0 \\ \delta_z & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{B21}$$

$$= {}^{i-1}\Delta_i \, dq_i$$

where

$${}^{i-1}\Delta_i = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{B22}$$

and dq_i is the incremental joint angle, a scalar. From eqs. 1 and 2, the differential translation and rotation of T2 can be expressed as

$$dT_2 = A_1 \Delta A_2 \qquad \text{B23}$$
$$= A_1 \, {}^{i-1}\Delta_i \, dq_i \, A_2 = A_1 \, {}^{i-1}\Delta_i \, A_2 dq_i$$

Thus,

$$\frac{\partial T_2}{\partial q_i} = A_1 \, {}^{i-1}\Delta_i A_2 \qquad \text{B24}$$

This partial derivative indicates how the position and orientation of the end of l_2 changes as joint angle q_i changes. Thus, it represents the information in the column of a Jacobian matrix.

From Eq. B2, the differential translation of T2 can also be expressed as

$$dT_2 = T_2 \, {}^{T_2}\Delta_i dq_i \qquad \text{B25}$$

where, from Eq. B6,

$${}^{T_2}\Delta_i = A_2^{-1} \, {}^{i-1}\Delta_i A_2 \qquad \text{B26}$$

Thus, combining Eqs. B25 and B26 yields

$$\frac{\partial T_2}{\partial q_i} = T_2{}^{T_2}\Delta_i = T_2 A_2{}^{-1}\,{}^{i-1}\Delta_i A_2 \qquad \text{B27}$$

$$= A_1 A_2 A_2{}^{-1}\,{}^{i-1}\Delta_i A = A_1\,{}^{i-1}\Delta_i A_2$$

which matches the result from Eq. B24. More generally, Eqs. B26 and B27 can be written as

$$\frac{\partial T_n}{\partial q_i} = T_n{}^{T_n}\Delta_i \qquad \text{B28}$$

$${}^{T_n}\Delta_i = \left(A_i A_{i+1} \ldots A_n\right)^{-1}\,{}^{i-1}\Delta_i \left(A_i A_{i+1} \ldots A_n\right) \qquad \text{B29}$$

Note that this is in the form of Eq. B10 with

$$T = \left(A_i A_{i+1} \ldots A_n\right) \qquad \text{B30}$$

Thus, Eq. B17 can be used to compute the elements of ${}^{T_n}\Delta_i$. This can then be used, as in Eq. B25, to compute differential changes in end-effector position resulting from differential changes in joints:

$${}^{T_n}\Delta = {}^{T_n}\Delta_i\, dq_i \qquad \text{B31}$$

Consider, again, the simple two-link manipulator example. For joint i = 1,

$$T = A_1 A_2 = T_2 \qquad \text{B32}$$

and ${}^{i-1}\Delta_i$ is given by Eq. B22. Thus, d in Eq. B17 is 0, and

$$\delta_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad \text{B33}$$

Letting n_1, o_1, a_1, p_1 be the component vectors of T, Eq. B17 becomes

$$
\begin{aligned}
{}^{T_n}d_{1x} &= n_1 \cdot (\delta_1 \times p_1) = (-n_{1x}p_{1y} + n_{1y}p_{1x}) \\
{}^{T_n}d_{1y} &= o_1 \cdot (\delta_1 \times p_1) = (-o_{1x}p_{1y} + o_{1y}p_{1x}) \\
{}^{T_n}d_{1z} &= a_1 \cdot (\delta_1 \times p_1) = (-a_{1x}p_{1y} + a_{1y}p_{1x}) \\
{}^{T_n}\delta_{1x} &= n_z \\
{}^{T_n}\delta_{1y} &= o_z \\
{}^{T_n}\delta_{1z} &= a_z
\end{aligned}
\qquad \text{B34}
$$

By substituting these elements into Eq. B31, we are able to compute differential changes resulting from differential changes in joint 1:

$$
\begin{bmatrix} {}^{T_n}d_x \\ {}^{T_n}d_y \\ {}^{T_n}d_z \\ {}^{T_n}\delta_x \\ {}^{T_n}\delta_y \\ {}^{T_n}\delta_z \end{bmatrix} = \begin{bmatrix} {}^{T_n}d_{1x} \\ {}^{T_n}d_{1y} \\ {}^{T_n}d_{1z} \\ {}^{T_n}\delta_{1x} \\ {}^{T_n}\delta_{1y} \\ {}^{T_n}\delta_{1z} \end{bmatrix} dq_i \qquad \text{B35}
$$

Thus, the left side elements of Eq. B34 are the first column of the manipulator Jacobian. More generally, this equation is of the form

$$
\begin{bmatrix} {}^{T_n}d_x \\ {}^{T_n}d_y \\ {}^{T_n}d_z \\ {}^{T_n}\delta_x \\ {}^{T_n}\delta_y \\ {}^{T_n}\delta_z \end{bmatrix} = \begin{bmatrix} {}^{T_n}d_{1x} \ \ldots {}^{T_n}d_{nx} \\ {}^{T_n}d_{1y} \ \ldots {}^{T_n}d_{ny} \\ {}^{T_n}d_{1z} \ \ldots {}^{T_n}d_{nz} \\ {}^{T_n}\delta_{1x} \ \ldots {}^{T_n}\delta_{nx} \\ {}^{T_n}\delta_{1y} \ \ldots {}^{T_n}\delta_{nx} \\ {}^{T_n}\delta_{1z} \ \ldots {}^{T_n}\delta_{nx} \end{bmatrix} \begin{bmatrix} dq_1 \\ \ldots \\ dq_n \end{bmatrix} \qquad \text{B36}
$$

where the elements of the Jacobian are computed as in Eq. B34.

Thus, the Jacobian is a 6 x n matrix. Using Eq. B25, the elements computed by this can be put back into their 4x4 homogeneous transform format, and pre-multiplied by T_n to get dT_n. Often, however, the goal is just to compute the Jacobian, so this step isn't necessary.

Appendix C – Computation of Rotational Part of Jacobian and Hessian

1 Rotational Part of Jacobian

A Jacobian column is computed using the following equation

$$\frac{\partial T_n}{\partial q_i} = T_n {}^{T_n}\Delta_i \qquad (1)$$
$${}^{T_n}\Delta_i = \left(A_i A_{i+1} \ldots A_n\right)^{-1} {}^{i-1}\Delta_i \left(A_i A_{i+1} \ldots A_n\right)$$

(this is eq. 28 of Appendix 5.2.B). The translational part of the Jacobian column is simply rows 1 – 3 of column 4 of the matrix computed by eq. 1. The rotational part has idiosyncrasies that require special treatment.

From eq. 16 of Appendix 5.2.C, ${}^{T_n}\Delta_i$ is of the form

$${}^{T_n}\Delta_i = \begin{bmatrix} 0 & -\delta_z & \delta_y & d_x \\ \delta_z & 0 & -\delta_x & d_y \\ -\delta_y & \delta_x & 0 & d_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad (2)$$

If T_{n_rot} is the 3x3 rotation transform in T_n (upper-left 3x3 submatrix of T_n), then the incremental angular velocity due to an incremental change in q_i is

$$\omega_{n\_i} = T_{n\_rot} \begin{bmatrix} \delta_x \\ \delta_y \\ \delta_z \end{bmatrix} \qquad (3)$$

This is analogous to eq. 1. The problem is that this is an angular velocity vector, and because the plant control outputs are typically in terms of roll, pitch, and yaw (Euler angles), the derivatives of these outputs are in terms of derivatives of roll, pitch, and yaw, which is not the same thing as angular velocity.

To understand how the angular velocity vector can be converted to derivatives of roll, pitch, and yaw, it is useful to review orientation representation conventions.

1.1 Orientation Representation Conventions

Orientation of a rigid body can be represented in a variety of ways including rotation transformation matrices, fixed angles, Euler angles, and quaternions (ref. Craig, pg. 45). The convention used here is fixed angles (successive rotations about the axes of a fixed reference). The rotation sequence is pitch (rotation about fixed y axis) followed by roll (rotation about fixed x axis) followed by yaw (rotation about fixed z axis). The x axis points forward from the model, the y axis points left from the model, and the z axis points up (see previous description of humanoid model).

The rotation transformation matrix corresponding to this convention is

$$R = \begin{bmatrix} (c\alpha c\beta - s\alpha s\beta s\gamma) & -s\alpha c\gamma & (c\alpha s\beta + s\alpha c\beta s\gamma) \\ (s\alpha c\beta + c\alpha s\beta s\gamma) & c\alpha c\gamma & (s\alpha s\beta - c\alpha c\beta s\gamma) \\ -c\gamma s\beta & s\gamma & c\gamma c\beta \end{bmatrix} \tag{4}$$

where α is yaw (about z axis), β is pitch (about y axis), and γ is roll (about x axis), and $s\alpha$ and $c\alpha$ are shorthand for sine(α) and cosine(α). This is consistent with the result given in Craig (ref. Appendix B, pg. 444) for the fixed angle set $R_{YXZ}(\gamma, \beta, \alpha)$ (in Craig, γ represents pitch, and β represents roll).

Angles can be computed from this rotation matrix using the following formulas:

$$\gamma = a\tan 2\left(R_{32}, \sqrt{{R_{12}}^2 + {R_{22}}^2} \right) \tag{5}$$

$$\alpha = a\tan 2\left(\frac{-R_{12}}{c\gamma}, \frac{R_{22}}{c\gamma} \right) \tag{6}$$

$$\beta = a\tan 2\left(\frac{-R_{31}}{c\gamma}, \frac{R_{33}}{c\gamma} \right) \tag{7}$$

(ref. Craig, pg. 47). For eq. 5, the positive result of the square root term is chosen so that

$$-\frac{\pi}{2} \le \gamma \le \frac{\pi}{2}.$$

It is necessary to check for the case of $\gamma = \pm\frac{\pi}{2}$, since this means that $c\gamma = 0$. The solution degenerates due to 0 in the denominators of equations 6 and 7. In this case, α is arbitrarily set to 0. Then,

$$\beta = a\tan 2(R_{21}, R_{11}) \quad (\text{if } \gamma = \frac{\pi}{2}) \tag{8}$$

and

$$\beta = -a\tan 2(R_{21}, R_{11}) \quad (\text{if } \gamma = -\frac{\pi}{2}) \tag{9}$$

1.2 Conversion Between Angular Velocity Representations

The rotation transformation matrix (eq. 4) can be differentiated by taking partial derivatives with respect to α, β, and γ (ref. Craig pg. 163, AngularVelocityRep.mws). The resulting matrix is

$$E_{xyz} = \begin{bmatrix} c\alpha & -s\alpha c\gamma & 0 \\ s\alpha & c\alpha c\gamma & 0 \\ 0 & s\gamma & 1 \end{bmatrix} \tag{10}$$

This transforms angle derivatives to an angular velocity vector:

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = E_{xyz} \begin{bmatrix} \dot{\gamma} \\ \dot{\beta} \\ \dot{\alpha} \end{bmatrix} \tag{11}$$

The inverse of this transformation is

$$E_{xyz}^{-1} = \begin{bmatrix} c\alpha & s\alpha & 0 \\ -\frac{s\alpha}{c\gamma} & \frac{c\alpha}{c\gamma} & 0 \\ \frac{s\alpha s\gamma}{c\gamma} & -\frac{c\alpha s\gamma}{c\gamma} & 1 \end{bmatrix} \tag{12}$$

(see AngularVelocityRep.mws). This transforms an angular velocity vector to angle derivatives.

$$\begin{bmatrix} \dot{\gamma} \\ \dot{\beta} \\ \dot{\alpha} \end{bmatrix} = E_{xyz}{}^{-1} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{13}$$

1.3 Rotational Part of Jacobian In Terms of Angle Derivatives

Eq. 13 can now be combined with eq. 3 to computet the rotational part of the Jacobian in terms of angle derivatives (instead of the angular velocity vector computed by eq. 3).

$$\begin{bmatrix} \delta\gamma \\ \delta\beta \\ \delta\alpha \end{bmatrix} = E_{xyz}{}^{-1}\, T_{n\_rot} \begin{bmatrix} \delta_x \\ \delta_y \\ \delta_z \end{bmatrix} \tag{14}$$

This is stacked below the translational part of the Jacobian to form a 6-element Jacobian column.

$$\begin{bmatrix} \delta x & \delta y & \delta z & \delta\gamma & \delta\beta & \delta\alpha \end{bmatrix}^T \tag{15}$$

2 Rotational Part of Hessian

The translational part of the Hessian is computed as explained in section 5.2.1.4. From eq. 5.2.27,

$$\frac{\partial^2 \left({}_0\mathbf{T}_i \right)}{\partial q_j \partial q_k} = {}_0\mathbf{T}_i \, {}^{T_i}\Delta_j \, {}^{T_i}\Delta_k \tag{16}$$

The translational part of the Hessian column is simply rows 1 – 3 of column 4 of the matrix computed by eq. 1. As was the case with the Jacobian, the rotational part requires special treatment.

2.1 Spatial Acceleration Computations

The first step in computing rotational second derivatives is to compute the angular acceleration vector. For convenience, spatial notation is used here (ref. Featherstone). Thus, the spatial velocity at each link is

$$\hat{\mathbf{v}}_i = \hat{\mathbf{v}}_{i-1} + \hat{\mathbf{s}}_i \dot{q}_i \qquad \left(\hat{\mathbf{v}}_0 = \hat{\mathbf{0}} \right) \tag{17}$$

and the spatial acceleration is

$$\hat{\mathbf{a}}_i = \hat{\mathbf{a}}_{i-1} + \hat{\mathbf{v}}_i \hat{\times} \hat{\mathbf{s}}_i \dot{q}_i + \hat{\mathbf{s}}_i \ddot{q}_i \qquad \left(\hat{\mathbf{a}}_0 = \hat{\mathbf{0}}\right) \tag{18}$$

Here, the caret indicates a six-element spatial vector. All of these vectors are in global coordinates. The vector $\hat{\mathbf{s}}_i$ is the local axis vector, $\hat{\mathbf{s}}_i{}'$, for joint i transformed to global coordinates:

$$\hat{\mathbf{s}}_i = {}_0\hat{\mathbf{X}}_i \hat{\mathbf{s}}_i{}' \tag{19}$$

Note that this is a Jacobian column, exactly as computed in Appendix 5.2.B.

$$\hat{\mathbf{s}}_i = {}_0\hat{\mathbf{X}}_i \hat{\mathbf{s}}_i{}' \Leftrightarrow T^{T}\Delta_i \tag{20}$$

Eqs. 17 and 18 are in recursive form. They can be expanded out to separate terms associated with the first and second derivatives of the joint angles. Thus,

$$\hat{\mathbf{a}}_n = \sum_{i=1}^{n} \hat{\mathbf{s}}_i \ddot{q}_i + \sum_{i=2}^{n} \left[\sum_{j=1}^{i-1} \hat{\mathbf{s}}_j \dot{q}_j \right] \hat{\times} \hat{\mathbf{s}}_i \dot{q}_i \tag{21}$$

Note that this is of the form of eq. 5.2.13, where

$$\mathbf{\Psi} = \left[\hat{\mathbf{s}}_0 \quad \ldots \quad \hat{\mathbf{s}}_n\right] \tag{22}$$

$\mathbf{\Psi}$ is the Jacobian, and

$$\mathbf{\Psi}_{const} = \sum_{i=2}^{n} \left[\sum_{j=1}^{i-1} \hat{\mathbf{s}}_j \dot{q}_j \right] \hat{\times} \hat{\mathbf{s}}_i \dot{q}_i \tag{23}$$

To get acceleration in terms of angle derivatives, the appropriate form of the Jacobian must be used (see previous section). Also, the $\mathbf{\Psi}_{const}$, as expressed above, is a spatial acceleration vector. The rotational part of this is an angular acceleration vector. This must be converted to second derivatives of pitch, roll, and yaw. This is accomplished in the following way.

From eqs. 10 and 11,

$$\omega_x = \left[c\alpha \quad -s\alpha c\gamma \quad 0\right] \begin{bmatrix} \dot{\gamma} \\ \dot{\beta} \\ \dot{\alpha} \end{bmatrix} = c\alpha\,\dot{\gamma} - s\alpha c\gamma\,\dot{\beta} \tag{24}$$

The time derivative of this is the x component of the angular acceleration vector; it is given by

$$\frac{d\omega_x}{dt} = \frac{\partial \omega_x}{\partial \mathbf{r}} \frac{d\mathbf{r}}{dt} \tag{25}$$

where

$$\mathbf{r} = \begin{bmatrix} \alpha & \beta & \gamma & \dot{\alpha} & \dot{\beta} & \dot{\gamma} \end{bmatrix} \tag{26}$$

Now,

$$\frac{\partial \omega_x}{\partial \alpha} = -s\alpha\,\dot{\gamma} - c\alpha\, c\gamma\,\dot{\beta} \qquad \frac{\partial \omega_x}{\partial \dot{\alpha}} = 0 \tag{27}$$

$$\frac{\partial \omega_x}{\partial \beta} = 0 \qquad \frac{\partial \omega_x}{\partial \dot{\beta}} = -s\alpha\, c\gamma$$

$$\frac{\partial \omega_x}{\partial \gamma} = s\alpha\, s\gamma\,\dot{\beta} \qquad \frac{\partial \omega_x}{\partial \dot{\gamma}} = c\alpha$$

Combining eqs. 25, 26, and 27 yields

$$\frac{d\omega_x}{dt} = \left(-s\alpha\,\dot{\gamma} - c\alpha\, c\gamma\,\dot{\beta}\right)\dot{\alpha} + \left(s\alpha\, s\gamma\,\dot{\beta}\right)\dot{\gamma} + \left(-s\alpha\, c\gamma\right)\ddot{\beta} + c\alpha\,\ddot{\gamma} \tag{28}$$

Similarly,

$$\omega_y = \begin{bmatrix} s\alpha & c\alpha\, c\gamma & 0 \end{bmatrix} \begin{bmatrix} \dot{\gamma} \\ \dot{\beta} \\ \dot{\alpha} \end{bmatrix} \tag{29}$$

$$= s\alpha\,\dot{\gamma} + c\alpha\, c\gamma\,\dot{\beta}$$

Partial derivatives are:

$$\frac{\partial \omega_y}{\partial \alpha} = c\alpha\,\dot{\gamma} - s\alpha\, c\gamma\,\dot{\beta} \qquad \frac{\partial \omega_y}{\partial \dot{\alpha}} = 0 \tag{30}$$

$$\frac{\partial \omega_y}{\partial \beta} = 0 \qquad \frac{\partial \omega_y}{\partial \dot{\beta}} = c\alpha\, c\gamma$$

$$\frac{\partial \omega_y}{\partial \gamma} = -c\alpha\, s\gamma\,\dot{\beta} \qquad \frac{\partial \omega_y}{\partial \dot{\gamma}} = s\alpha$$

Therefore,

$$\frac{d\omega_y}{dt} = (c\alpha\,\dot{\gamma} - s\alpha\,c\gamma\,\dot{\beta})\dot{\alpha} + (-c\alpha\,s\gamma\,\dot{\beta})\dot{\gamma} + (c\alpha\,c\gamma)\ddot{\beta} + s\alpha\,\ddot{\gamma} \tag{31}$$

Similarly,

$$\omega_z = \begin{bmatrix} 0 & s_\gamma & 1 \end{bmatrix} \begin{bmatrix} \dot{\gamma} \\ \dot{\beta} \\ \dot{\alpha} \end{bmatrix} \tag{32}$$

$$= s\gamma\,\dot{\beta} + \dot{\alpha}$$

Partial derivatives are:

$$\frac{\partial\omega_z}{\partial\alpha} = 0 \qquad \frac{\partial\omega_z}{\partial\dot{\alpha}} = 1 \tag{33}$$

$$\frac{\partial\omega_z}{\partial\beta} = 0 \qquad \frac{\partial\omega_z}{\partial\dot{\beta}} = s\gamma$$

$$\frac{\partial\omega_z}{\partial\gamma} = c\gamma\,\dot{\beta} \qquad \frac{\partial\omega_z}{\partial\dot{\gamma}} = 0$$

Therefore,

$$\frac{d\omega_z}{dt} = (c\gamma\,\dot{\beta})\dot{\gamma} + \ddot{\alpha} + s\gamma\,\ddot{\beta} \tag{34}$$

Eqs. 28, 31, and 34 can be expressed as

$$\dot{\boldsymbol{\omega}} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \boldsymbol{\varphi} + \mathbf{E_{xyz}} \begin{bmatrix} \ddot{\gamma} \\ \ddot{\beta} \\ \ddot{\alpha} \end{bmatrix} \tag{35}$$

where

$$\boldsymbol{\varphi} = \begin{bmatrix} (-s\alpha\,\dot{\gamma} - c\alpha\,c\gamma\,\dot{\beta})\dot{\alpha} + (s\alpha\,s\gamma\,\dot{\beta})\dot{\gamma} \\ (c\alpha\,\dot{\gamma} - s\alpha\,c\gamma\,\dot{\beta})\dot{\alpha} + (-c\alpha\,s\gamma\,\dot{\beta})\dot{\gamma} \\ (c\gamma\,\dot{\beta})\dot{\gamma} \end{bmatrix} \tag{36}$$

Solving for rotation angle accelerations in eq. 35 yields

$$\begin{bmatrix} \ddot{\gamma} \\ \ddot{\beta} \\ \ddot{\alpha} \end{bmatrix} = \mathbf{E_{xyz}}^{-1}\left(\dot{\boldsymbol{\omega}} - \boldsymbol{\varphi}\right) \tag{37}$$

The Matlab function ComputeAnglesDotDot performs this computation.

Appendix D – Introduction to Sliding Control

1 Motivation and Background

Model-based nonlinear control algorithms, such as feedback linearization, are extremely powerful techniques for computing control actions for systems with nonlinear dynamics. However, for real systems, these algorithms are, by themselves, insufficient because they assume the models perfectly describe the actual plant. This is never the case in reality. Modeling inaccuracies can be classified into two major types:

- structured uncertainties, such as errors in model parameters, and
- unstructured uncertainties, such as unmodeled dynamics.

The first type corresponds to errors in terms that are included in the model, while the second type corresponds to terms that are missing from the model altogether, typically due to use of a reduced-order model, which underestimates the true system order.

A robust controller deals with this problem using a two-part structure. The first part is the nominal part; it assumes the model is perfect, and issues control commands based on this model. This part may be implemented using a feedback linearizing controller. The second part contains additional control terms that compensate for the model uncertainty.

Sliding control [Slotine, 1991] is a type of robust control algorithm that is based on feedback linearization techniques, but allows for model imperfections. In particular, sliding control guarantees bounds on tracking error given bounds on model imperfections.

2 Sliding Surfaces

2.2 Intuitive Basis of Sliding Control

Sliding control is based on the idea that it is much easier to control a 1st-order dynamic system than it is to control a general nth-order system. A sliding controller uses a notational simplification, based on the idea of a sliding surface, which, in effect, allows nth-order problems to be replaced by equivalent 1st-order problems. The 1st-order systems are then relatively easy to control.

An important advantage of the sliding controller technique is that it provides a systematic approach to the problem of maintaining stability and tracking a desired output in the face of modeling imprecisions. In particular, the technique allows the trade-offs between model accuracy and tracking performance to be quantified in a simple manner, so that it is possible to decide required model accuracy for a given desired level of performance.

Consider the single-input nth-order dynamic system

$$x^{(n)} = f(\mathbf{x}) + b(\mathbf{x})u \qquad \text{D.1}$$

Note that this is in controllability canonical form. The scalar x is the output to be controlled, the scalar u is the control input, and the vector $\mathbf{x} = \begin{bmatrix} x & \dot{x} & \ldots & x^{(n-1)} \end{bmatrix}^T$ is the state vector. The control problem is to get the state $\mathbf{x}$ to track a specific time-varying desired state $\mathbf{x}_d = \begin{bmatrix} x_d & \dot{x}_d & \ldots & x_d^{(n-1)} \end{bmatrix}^T$ in the presence of model errors on $f(\mathbf{x})$ and $b(\mathbf{x})$. Note that the overall system is thus nonlinear, and time-varying. It is assumed that upper bounds on $f(\mathbf{x})$ and $b(\mathbf{x})$ are known, either as constants, or as functions of time. It is also assumed that the initial state of the system matches the desired tracking state.

$$\mathbf{x}_d(0) = \mathbf{x}(0) \qquad \text{D.2}$$

The tracking error of the system is defined as

$$\tilde{x} = x - x_d \qquad \text{D.3}$$

and the tracking error vector is defined as

$$\tilde{\mathbf{x}} = \mathbf{x} - \mathbf{x}_d = \begin{bmatrix} \tilde{x} & \dot{\tilde{x}} & \ldots & \tilde{x}^{(n-1)} \end{bmatrix}^T \qquad \text{D.4}$$

A time-varying surface $S(t)$ in the state-space $\mathbf{R}^{(n)}$ is now defined by the scalar equation

$$s(\mathbf{x}, t) = 0 \qquad \text{D.5}$$

where

$$s(\mathbf{x}, t) = \left(\frac{d}{dt} + \lambda \right)^{n-1} \tilde{x} \qquad \text{D.6}$$

and λ is a strictly positive constant. For example, if n = 2, the surface is

$$s = \dot{\tilde{x}} + \lambda \tilde{x} = 0 \qquad \text{D.7}$$

This simply defines a straight line in the $\tilde{x}, \dot{\tilde{x}}$ plane, as shown in Fig. D.1.

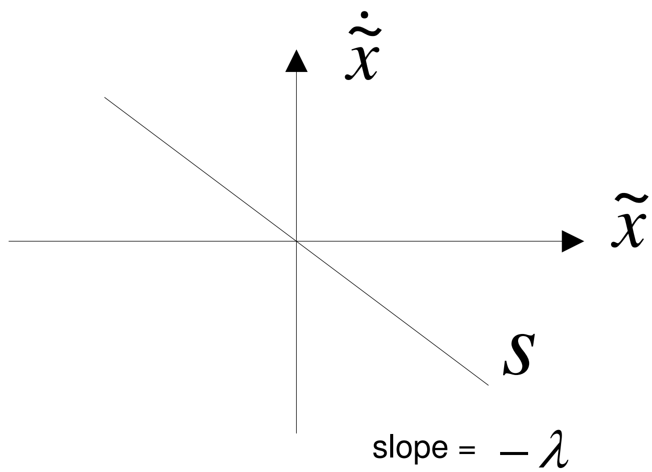


Fig. D.1 - Sliding surface for n = 2 is a straight line

If n = 3, the surface is

$$s = \ddot{\tilde{x}} + 2\lambda\dot{\tilde{x}} + \lambda^2\tilde{x} = 0 \qquad \text{D.8}$$

This is a 2-dimensional curved surface in the 3-dimensional space with axes $\ddot{\tilde{x}}, \dot{\tilde{x}}, \tilde{x}$.

Now, Eq. D.5 is a linear differential equation with the unique solution

$$\tilde{\mathbf{x}} = \mathbf{0} \qquad \text{D.9}$$

assuming Eq. D.2.

Thus, the problem of tracking the n-dimensional vector $\tilde{\mathbf{x}}_d$ can be reduced to that of keeping the scalar quantity s at zero.

How can the scalar quantity s be kept at zero? The problem of keeping the system on the sliding surface $S(t)$ can be addressed by a 1st order control law for $s(\mathbf{x},t)$ that drives this scalar to 0. Consider, for example, the following such control law.

$$\dot{s} = -ks \qquad \text{D.10}$$

For example, for n = 2, this becomes (using Eq. D.7)

$$\ddot{\tilde{x}} + \lambda\dot{\tilde{x}} = -k\left(\dot{\tilde{x}} + \lambda\tilde{x}\right) \qquad \text{D.11}$$

or

$$\ddot{\tilde{x}} = \ddot{x} - \ddot{x}_d = -(k+\lambda)\dot{\tilde{x}} + \lambda\tilde{x} \qquad \text{D.12}$$

Using Eq. D.1, this becomes

$$f(\mathbf{x}) + b(\mathbf{x})u - \ddot{x}_d = -(k+\lambda)\dot{\tilde{x}} + \lambda\tilde{x} \qquad \text{D.13}$$

Thus, setting the control input u to

$$u = \frac{1}{b(\mathbf{x})}\left(-(k+\lambda)\dot{\tilde{x}} + \lambda\tilde{x} + \ddot{x}_d - f(\mathbf{x})\right) \qquad \text{D.14}$$

results in the control law for s given by Eq. D.10. This has the effect of driving the system to the sliding surface if it ever leaves it. Note that the control law given by eq. 3.10 is not exactly the one used for a sliding controller, but it is similar, and it serves here to illustrate the point.

2.3 Controlling s

As explained in the previous section, the tracking control problem is solved by getting s to 0 and by keeping it there. This can be accomplished by the following simple 1^{st}-order control rule, similar to the one in Eq. D.10.

$$\frac{1}{2}\frac{d}{dt}\left(s^2\right) \le -\eta|s| \qquad \text{D.15}$$

where η is strictly positive. This just says that the norm distance to the surface, given by s^2, always decreases until it reaches 0, that is, until the system state reaches the surface. Once it is on the surface (once s = 0) it remains there, since Eq. D.15 then sets the derivative to 0. Thus, if Eq. D.15 can be enforced, then the tracking problem is solved. In this way, the nth-order tracking problem is converted to the 1^{st}-order problem of Eq. D.15.

Eq. D.15 is enforced by finding an appropriate rule for u in Eq. D.1 so that Eq. D.15 holds. Consider the case of n = 2 so that Eq. D.1 becomes

$$\ddot{x} = f(\mathbf{x}) + b(\mathbf{x})u \qquad \text{D.16}$$

Since n = 2, the sliding surface is defined by Eq. D.7. The first step in satisfying Eq. D.15 is to ensure that once the system is on the sliding surface, it stays there ($\dot{s} = 0$ if s = 0). Differentiating Eq. D.7 (as in Eq. D.11) yields

$$\dot{s} = \ddot{\tilde{x}} + \lambda\dot{\tilde{x}} = \ddot{x} - \ddot{x}_d + \lambda\dot{\tilde{x}} \qquad \text{D.17}$$

Substituting in Eq. D.16 and setting to 0 yields

$$\dot{s} = f(\mathbf{x}) + b(\mathbf{x})u - \ddot{x}_d + \lambda\dot{\tilde{x}} = 0 \qquad \text{D.18}$$

Solving for u yields the control law

$$u = \frac{1}{b(\mathbf{x})}\left(-f(\mathbf{x}) + \ddot{x}_d - \lambda\dot{\tilde{x}}\right) \qquad \text{D.19}$$

Note that this is similar to a feedback linearization control law, which would be

$$u = \frac{1}{b(\mathbf{x})}\left(-f(\mathbf{x}) + \ddot{x}_d\right) \qquad \text{D.20}$$

and which results in

$$\ddot{x} = \ddot{x}_d \qquad \text{D.21}$$

The only difference between this and Eq. D.19 is that Eq. D.19 adds the $-\lambda\dot{\tilde{x}}$ term. This allows for the case where the system does not begin at the desired tracking point.

Eq. D.19 assumes that the functions f and b are known perfectly (that there is no modeling error). Suppose, now, that f is not known perfectly, but rather, is approximated by $\hat{f}$. Suppose, also, that the estimation error is bounded by

$$\left|\hat{f} - f\right| \le F \qquad \text{D.22}$$

Similarly, suppose that b is approximated by $\hat{b}$, and that the estimation error is bounded by

$$\left|\hat{b} - b\right| \le B \qquad \text{D.23}$$

F and B may be constants, or functions of state. Using $\hat{f}$ and $\hat{b}$ in Eq. D.19 yields the control law

$$\hat{u} = \frac{1}{\hat{b}(\mathbf{x})}\left(-\hat{f}(\mathbf{x}) + \ddot{x}_d - \lambda\dot{\tilde{x}}\right) \qquad \text{D.24}$$

Because $\hat{f}$ and $\hat{b}$ are not perfect, this control law cannot guarantee that the system will remain on the sliding surface. To correct for this, and to get the system onto the sliding surface when it doesn't start there initially, an additional term has to be added to the control law. This term is based on the sign of s; on whether the system is above or below the surface. The term is

-k sgn(s)

where

sgn(s) = 1 if s > 0
sgn(s) = -1 if s < 0

The full control law is then

$$u = \hat{u} - k\,\mathrm{sgn}(s) \qquad \text{D.25}$$

The remaining question is what the value of k should be. First, note that the left side of Eq. D.15 can be written as

$$\frac{1}{2}\frac{d}{dt}\left(s^2\right) = \dot{s}s \qquad \text{D.26}$$

Substituting this into Eq. D.18 yields

$$\dot{s}s = \left(f(\mathbf{x}) + b(\mathbf{x})u - \ddot{x}_d + \lambda\dot{\tilde{x}}\right)s \qquad \text{D.27}$$

Substituting in Eq. D.25 for u yields

$$\dot{s}s = \left(f(\mathbf{x}) + b(\mathbf{x})\left(\hat{u} - k\,\mathrm{sgn}(s)\right) - \ddot{x}_d + \lambda\dot{\tilde{x}}\right)s \qquad \text{D.28}$$

For simplicity, let's assume b = 1 (see Slotine, pg. 287, section on gain margins, when this isn't the case). Substituting in Eq. D.24 yields

$$\dot{s}s = \left(f(\mathbf{x}) - \hat{f}(\mathbf{x}) - k\,\mathrm{sgn}(s)\right)s = \left(f(\mathbf{x}) - \hat{f}(\mathbf{x})\right)s - k|s| \qquad \text{D.29}$$

Combining Eqs. D.29 and D.15 yields

$$\dot{s}s = \left(f(\mathbf{x}) - \hat{f}(\mathbf{x})\right)s - k|s| \le -\eta|s| \qquad \text{D.30}$$

Re-arranging terms gives

$$\left(f(\mathbf{x}) - \hat{f}(\mathbf{x})\right)s + \eta|s| \le k|s| \qquad \text{D.31}$$

From Eq. D.22, F is always positive. Therefore,

$$\left(f(\mathbf{x}) - \hat{f}(\mathbf{x})\right)s \le F|s| \qquad \text{D.32}$$

so setting

$$k = F + \eta \qquad \text{D.33}$$

satisfies Eq. E.30. Thus, Eq. D.15 is satisfied, and the control problem is solved. Note the importance of the absolute value of the s term in Eq. D.32 (s alone is not sufficient). This is why the sgn term is necessary in the control law (Eq. D.25).

To summarize, the controller using the –k sgn(s) term uses the following intuitive feedback strategy: if there is an error (if s is not on the surface) push hard enough (as defined by k) in the direction of the surface. As can be seen from Eq. D.33, the value for k is a function of the estimation error F, and of the feedback gain η in Eq. D.15. This feedback gain controls how quickly the system state reaches the surface when it isn't on it (see pg. 281, Slotine).

Appendix E - Balance Recovery Through Stepping

Chapter 3 described a method of enhancing balance control by utilizing spin angular momentum to move the CMP outside the support polygon, but without changing the support polygon itself. This is important for situations where stepping is constrained such that the support polygon cannot be changed in a desirable way. In this Appendix, we lift this restriction; we describe balance recovery by stepping in order to change the support polygon. As in the previous section, the focus here is on disturbances that can be modeled as disturbances to the CM. A number of important simplifying assumptions are made here. These make the stability analysis problem tractable, while preserving key characteristics of the problem that make the analysis relevant to the actual system.

Suppose that a biped is moving with some horizontal velocity, possibly due to a disturbance. This can be represented by the system CM velocity, and associated kinetic energy. The problem addressed here is how to slow this velocity in order to bring the CM to a stop. Assuming that the only external force on the system is the ground reaction force, which is transmitted through the legs, the reduction in velocity can only be accomplished by an appropriate horizontal component of the ground reaction force. This is achieved, in this section, by appropriate foot placement of the swing leg in the direction of the CM velocity, subject to constraints due to the system's morphology. Thus, the principle questions investigated in this appendix are:

- What is the best foot placement, or sequence of foot placements, that bring the CM to a stop?
- How many steps are needed to stop?
- How long will it take to stop?

The analysis discussed here assumes a flat surface, and no restrictions on where the foot can be placed.

1.1 Virtual Leg Model

Consider a simplified model, comprised of a CM and a virtual leg. This leg can shoot out in any direction, in order to exert a horizontal force that is beneficial to the goal, by slowing or stopping the CM movement. A top view of this model is shown below.

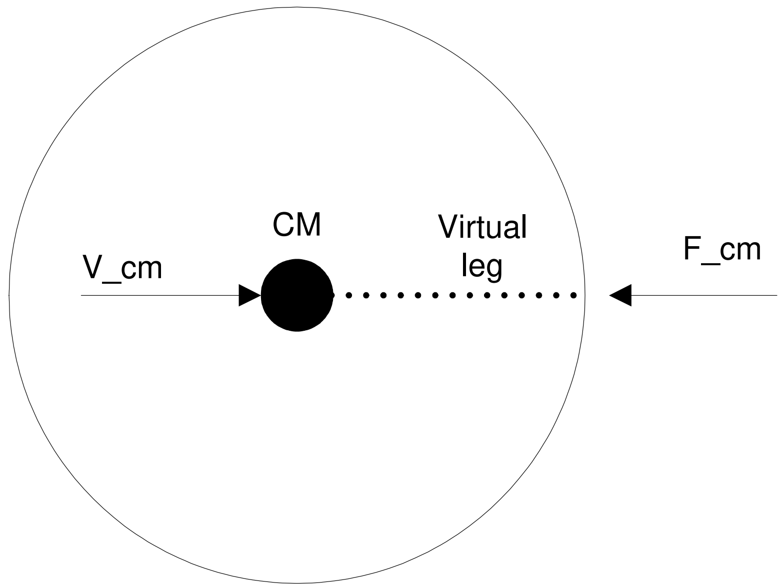


Fig. E.1 – Top view of the virtual leg model

The fact that the virtual leg always shoots out in the direction of the CM velocity means that this problem can be analyzed in terms of a single horizontal direction, rather than the full horizontal plane. The virtual leg will also exert a vertical force, as shown in the side view of Fig. E.2.

Important questions that we will examine include:

- What relations or constraints exist between the vertical and horizontal forces exerted by the swing leg?
- At any point in time, what is the maximum horizontal force that the virtual leg can exert?
- Over one step cycle, what is the integral of maximum horizontal force that the virtual leg can exert? This is equivalent to asking how much of the COM's kinetic energy the virtual leg can dissipate.

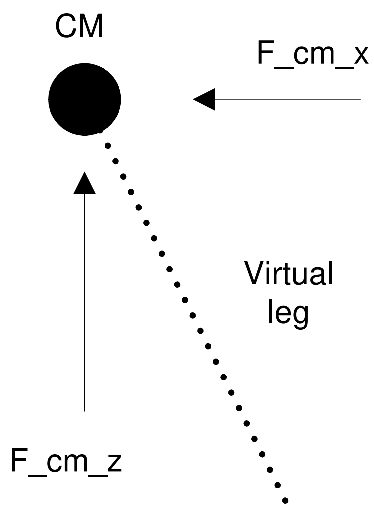


Fig. E.2 – Side view of virtual leg model

1.2 Stability Analysis for Fixed Leg Length Stepping

To answer the questions posed in the previous section, a more complete model, based on the virtual leg model, is necessary. The initial model considered here makes some additional simplifying assumptions, but it nevertheless is useful for providing important insights. Some of the simplifying assumptions will be relaxed in models introduced in subsequent sections.

Consider the simple 2D model shown in Fig E.3. It has 3 links: a body and two legs. The links have no mass. All the mass is concentrated in one point (the CM), which is at the hip joint of the model. The feet are points, so no ankle torque can be exerted. ZMP is, therefore, always at the point of contact, when in single support. The legs are assumed to be of fixed length l when they are in contact with the ground. However, the swing leg is allowed to shorten temporarily in order to clear the ground. We also assume that the line of force always goes from this ZMP through the CM. Therefore, the no spin torque assumption is valid, and, hence, the CMP is always coincident with the ZMP, as explained in Chapter 3.

A useful approach for analyzing stepping motions for such a model is to consider kinetic and potential energy. Let's begin with the case where the model begins in an upright position, gets a very gentle lateral nudge at the CM, and takes a step, as shown in Fig. E.4.

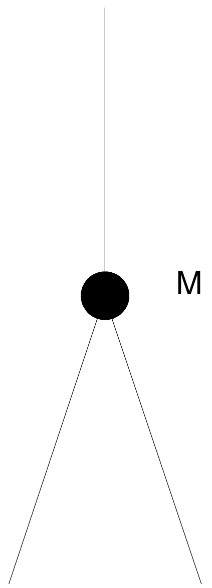


Fig. E.3 – Simple 2D stepping model

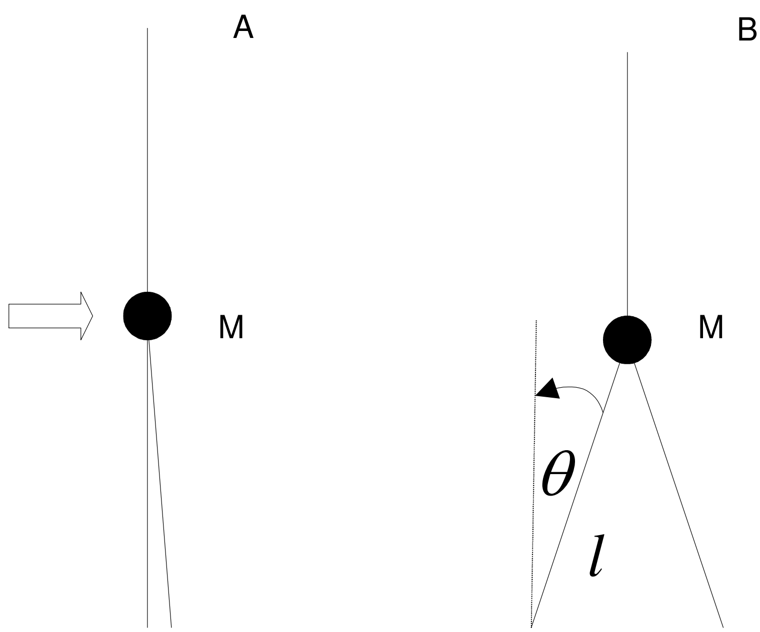


Fig. E.4 – Simple 2D model taking a step

Ignoring the energy added to the system by the initial gentle nudge in pose A, the system loses potential energy and converts it to kinetic energy as the CM falls. Just prior to foot strike, the potential energy lost is

$$\Delta PE = -Mgl\left(1-\cos(\theta)\right) \tag{E.1}$$

The kinetic energy just prior to foot strike is

$$KE = \frac{1}{2} M l^2 \dot{\theta}^2 \tag{E.2}$$

Since this is equal to the potential energy lost,

$$Mgl\left(1-\cos(\theta)\right) = \frac{1}{2} M l^2 \dot{\theta}^2 \tag{E.3}$$

or

$$\dot{\theta} = \sqrt{2\frac{g}{l}\left(1-\cos(\theta)\right)} \tag{E.4}$$

Upon impact, the CM position is

$$\begin{aligned} x_{cm} &= l\sin(\theta) \\ z_{cm} &= l\cos(\theta) \end{aligned} \tag{E.5}$$

The CM velocity is, therefore,

$$\begin{aligned} \dot{x}_{cm} &= l\cos(\theta)\dot{\theta} \\ \dot{z}_{cm} &= -l\sin(\theta)\dot{\theta} \end{aligned} \tag{E.6}$$

If the leg is acting as a damper, it does negative work in the direction of force. The ratio of lateral vs. vertical damping force is

$$\frac{F_{damp\_lateral}}{F_{damp\_vertical}} = \tan(\theta) \tag{E.7}$$

Assuming that this damping force acts instantaneously, with negligible change in length of the stance leg, the ratio expressed in Eq. E.7 also is the ratio of lateral to vertical negative work. If the vertical velocity goes quickly to 0 on impact, then the change in lateral velocity can be computed based on this ratio. Let's also assume, for now, that $\theta < \pi/4$ so that the ratio in Eq. E.7 is less than 1. The lateral change in velocity is then

$$\begin{aligned}\Delta v_{lateral} &= \tan(\theta)\Delta v_{vertical} \\ &= \tan(\theta) l \sin(\theta)\dot{\theta}\end{aligned} \tag{E.8}$$

The lateral velocity after foot strike is then the lateral velocity before foot strike, from Eq. E.6, plus the change in lateral velocity from Eq. E.8:

$$v_{lateral\_as} = l\cos(\theta)\dot{\theta} - l\tan(\theta)\sin(\theta)\dot{\theta} \tag{E.9}$$

The kinetic energy after foot strike is then

$$KE_{as} = \frac{1}{2} M v_{lateral\_as}{}^2 \tag{E.10}$$

The total energy reduction is

$$\Delta E = \Delta PE + KE_{as} \tag{E.11}$$

For example, if

$$\begin{aligned} M &= 100 \\ l &= 1 \\ \theta &= 30\deg \end{aligned} \tag{E.12}$$

then, from Eq. E.1,

$$\Delta PE = -100 \times 9.8 \times 1 \times (1 - 0.866) = -131J \tag{E.13}$$

From Eq. E.4,

$$\dot{\theta} = 1.62 \ \text{rad/sec} \tag{E.14}$$

From Eq. E.6, the COM velocity before foot strike is

$$\dot{x}_{com} = l\cos(\theta)\dot{\theta} = 0.866 \times 1.62 = 1.4 \ \text{m/s} \tag{E.15}$$
$$\dot{z}_{com} = -l\sin(\theta)\dot{\theta} = -0.5 \times 1.62 = 0.81 \ \text{m/s}$$

From Eq. E.8, the change in lateral velocity is

$$\Delta v_{lateral} = \tan(\theta)\Delta v_{vertical} = -0.5774 \times 0.81 = -0.47 \ \text{m/s} \tag{E.16}$$

From Eq. E.9, the lateral velocity after foot strike is then

$$v_{lateral\_as} = 1.4 - 0.47 = 0.93 \ \text{m/s} \tag{E.17}$$

From Eq. E.10, the kinetic energy after foot strike is

$$\begin{aligned} KE_{as} &= \frac{1}{2} M v_{lateral\_as}{}^2 \\ &= 50 \times (0.93)^2 \\ &= 43.25J \end{aligned} \tag{E.18}$$

From Eq. E.11, the total energy reduction is

$$\Delta E = -131 + 43.25 = -87.75J \tag{E.19}$$

The residual kinetic energy in Eq. E.18 must be dissipated in some way. This can be done in a number of ways, as will be explained in the following sections.

Let's consider the case where the initial nudge is significant, and the system has to take multiple steps to stabilize. How much energy is dissipated at each step? Let's assume, as in the previous discussion, that swing and stance leg have equal length l when in double support, so that pose B above is always symmetric. Suppose the initial nudge imparts a kinetic energy of 200 J to the system. This could be the result of a push of 1000N for 0.2 meters, or, perhaps, the result of a mostly elastic collision with another person with similar mass (100 kg) and moving at about sqrt(2) m/s. Thus, the initial energy of the system is

$$E = PE + KE = 980 + 200 = 1180J \qquad \text{(E.20)}$$

Let's suppose the model moves according to the sequence of poses shown in Fig. E.5. At pose B, the system is temporarily in double support, and the swing and stance legs switch. At pose C, the system is upright on the new stance leg. Just before foot strike, the change in kinetic energy is 131 J, as computed previously in Eq. E.13. Thus, the total kinetic energy just before foot strike is 331 J. From Eq. E.2,

$$\dot{\theta} = \sqrt{\frac{2}{Ml^2} KE} = \sqrt{\frac{331}{25}} = 3.64 \qquad \text{rad/sec} \qquad \text{(E.21)}$$

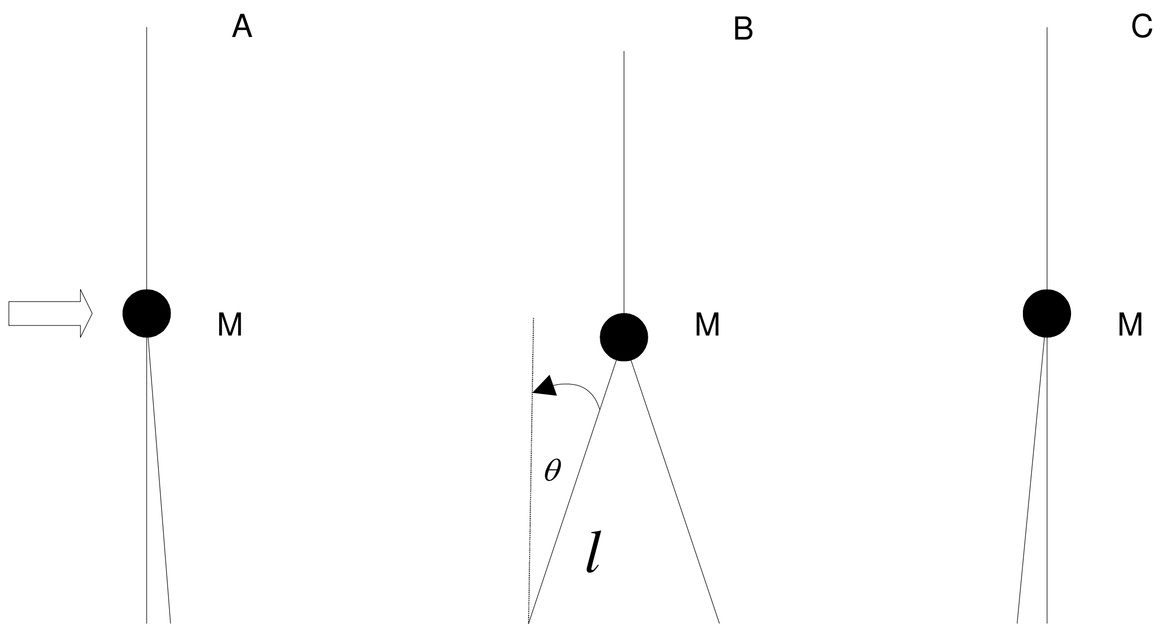


Fig. E.5 – Stepping pose sequence.

From Eq. E.6, the CM velocity before foot strike is

$$\dot{x}_{cm} = l\cos(\theta)\dot{\theta} = 0.866 \times 3.64 = 3.15 \text{ m/s} \tag{E.22}$$

$$\dot{z}_{cm} = -l\sin(\theta)\dot{\theta} = -0.5 \times 3.64 = 1.82 \text{ m/s}$$

From Eq. E.8, the change in lateral velocity is

$$\Delta v_{lateral} = \tan(\theta)\Delta v_{vertical} = -0.5774 \times 1.82 = 1.05 \text{ m/s} \tag{E.23}$$

From Eq. E.9, the lateral velocity after foot strike is then

$$v_{lateral\_as} = 3.15 - 1.05 = 2.1 \text{ m/s} \tag{E.24}$$

From Eq. E.10, the kinetic energy after foot strike is

$$KE_{as} = \frac{1}{2} M v_{lateral\_as}{}^{2} = 50 \times (2.1)^2 = 220.5J \tag{E.25}$$

The total energy reduction, just after impact of the swing leg, is

$$\Delta E = \Delta PE + \Delta KE = -131 + (220.05 - 200) = -111.95J \tag{E.26}$$

Thus, at pose B, the system is down to 1068 J. From pose B to pose C, no energy is lost, but potential energy increases (by 131 J). Potential energy in pose C is the same as in pose A (980 J). However, since overall energy has been reduced by 112 J to 1068 J, the kinetic energy at pose C must be 131 J less than it was at pose B. Thus, from Eq. E.25, kinetic energy has been reduced to

$$KE_c = 220.5 - 131 = 89.5J \tag{E.27}$$

Although one step isn't enough to dissipate the kinetic energy completely, it is clear that an additional step will. Further, the second step can be at a smaller angle than 30 degrees. After this second step, the system will end in pose C with no kinetic energy, and potential energy of 980 J, since all of the initial kinetic energy will have been dissipated.

This simple model clearly indicates some key points:

- Impact of the swing leg is critical for absorbing energy, since it is the only place where this can happen in this model
- Foot placement of the swing leg is important for determining how much energy will be absorbed.

1.3 Model with Extendable Legs

Let's consider some extensions to the previous simplified model. Suppose that the swing leg and stance leg can change lengths, so that they are not necessarily symmetric. Suppose also that when the swing leg lands and becomes the stance leg, it can dissipate energy by acting as a damper, by shortening. Consider, once again, the single step model, described above, with negligible initial nudge. From Eq. E.18, the kinetic energy after foot strike (in pose B) is 43.25 J. How should this energy be dissipated? If the legs can change in length, the CM can move laterally as shown in Fig. E.6.

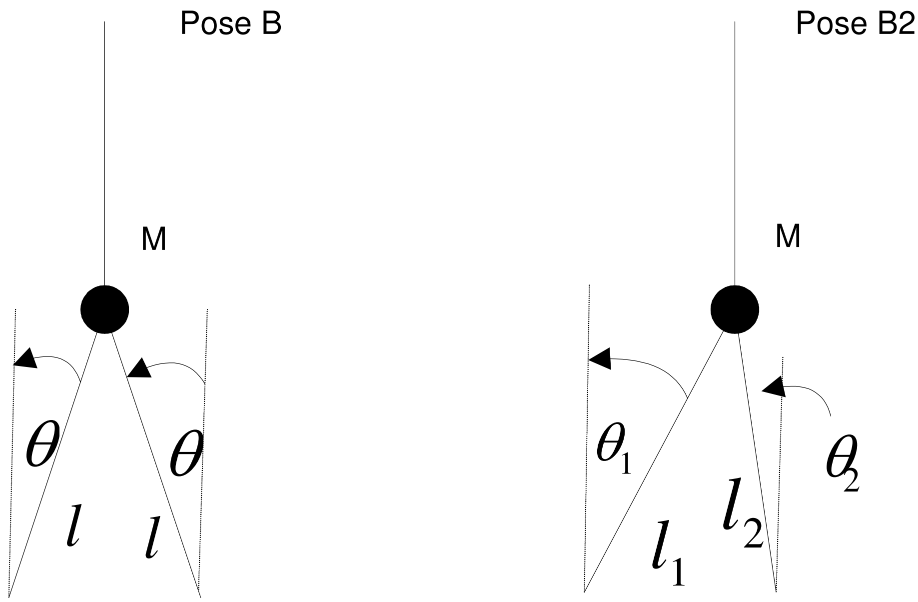


Fig. E.6 – Lateral movement of the CM while in double support.

The CM does not change vertical position from Pose B to Pose B2, so potential energy does not change. However, the forward leg, with length l_2 in pose B2, can exert a braking force that dissipates the lateral kinetic energy. Suppose that this leg is exerting all the force (the trailing leg exerts no force). Suppose also, as before, the force vector points from the foot contact point to the CM, so that there is no spin torque about the CM. The vertical component of the force vector must be Mg, because the CM vertical position does not change. The lateral component of the force vector is then a function of the lateral position of the CM.

The vertical position of the CM is fixed at

$$z_{cm} = l\cos(\theta) = 0.866 \qquad \text{(E.28)}$$

The ratio of lateral to vertical position is equal to the ratio of lateral to vertical force; therefore, the lateral force is

$$F_{lat} = Mg\frac{x_{cm}}{z_{cm}} \tag{E.29}$$

Lateral acceleration is simply

$$\ddot{x}_{cm} = \frac{F_{lat}}{M} \tag{E.30}$$

Combining Eqs. E.29 and E.30 yields the equation of motion:

$$\ddot{x}_{cm} - \frac{g}{z_{cm}}x_{cm} = 0 \tag{E.31}$$

In this case, lateral position is defined so that the origin is at the forward foot. Thus, initial lateral position is negative, and initial velocity is positive. The roots of the C.E. for Eq. E.31 are

$$s_1, s_2 = \frac{\pm\sqrt{4\dfrac{g}{z_{cm}}}}{2} = \frac{\pm\sqrt{4\dfrac{9.8}{0.866}}}{2} = \pm 3.36 \tag{E.32}$$

The solution for x_{cm} is

$$x_{cm} = k_1 e^{s_1 t} + k_2 e^{s_2 t} \tag{E.33}$$

and the velocity is

$$\dot{x}_{cm} = k_1 s_1 e^{s_1 t} + k_2 s_2 e^{s_2 t} \tag{E.34}$$

The constants are determined from initial conditions. The initial position is

$$x_{cm\_init} = -l\sin(\theta) = -0.5 \quad \text{(E.35)}$$

The initial velocity, from Eq. E.17, is 0.93 m/s. Setting t = 0 in Eqs. E.33 and E.34, and combining with Eqs. E.32 and E.35 yields

$$-0.5 = k_1 + k_2 \quad \text{(E.36)}$$
$$0.93 = 3.36(k_1 - k_2)$$

Solving these yields

$$k_1 = -0.11 \quad \text{(E.37)}$$
$$k_2 = -0.39$$

The last step is to find the time at which lateral velocity reaches 0. Setting Eq. E.34 to 0 yields

$$k_1 s_1 e^{s_1 t} = -k_2 s_2 e^{s_2 t} \quad \text{(E.38)}$$

Since $s_1 = -s_2$, this can be written as

$$k_1 e^{s_1 t} = k_2 e^{-s_1 t} \quad \text{(E.39)}$$

Re-arranging terms yields

$$e^{2 s_1 t} = \frac{k_2}{k_1} \quad \text{(E.40)}$$

or

$$2s_1 t = \ln\left(\frac{k_2}{k_1}\right) \tag{E.41}$$

or

$$t = \frac{1}{2s_1}\ln\left(\frac{k_2}{k_1}\right) \tag{E.42}$$

Substituting in values yields

$$t = \frac{1}{6.72}\ln\left(\frac{0.39}{0.11}\right) = 0.1883 \text{ s} \tag{E.43}$$

From Eq. E.33, the position at this point in time is

$$x_{com} = -0.11e^{3.36\times0.1883} - 0.39e^{-3.36\times0.1883} = -0.4142 \text{ m} \tag{E.44}$$

The distance traveled during this braking action is thus

$$\Delta x_{com} = -0.4142 + 0.5 = 0.09 \text{ m} \tag{E.45}$$

Thus, the lateral kinetic energy is quickly dissipated. At this point, the CM can be moved back to the symmetric position in pose B by gentle, concerted action of both legs.

Let's consider, now, another example of beneficial use of variable-length legs for handling a disturbance. Suppose that the CM begins in a vertical position that is not the maximum vertical position. Consider the motion sequence in Fig. E.7.

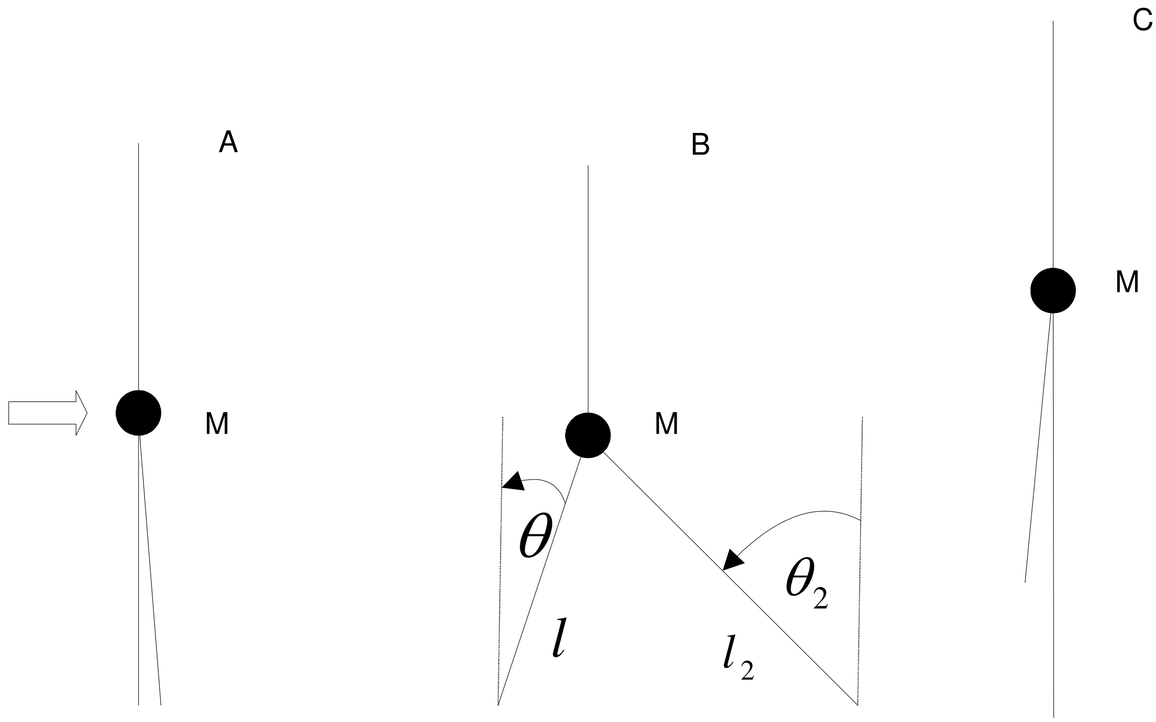


Fig. E.7 – Pose sequence beginning from crouch.

The model begins in pose A with stance leg having length l. The energy lost on impact of the swing leg is as described previously (see Eq. E.11 and related equations). The difference in this model is that the swing leg stretches to length l_2, which is greater than l. Thus, if the swing leg remains at this length, the CM of the model in pose C will be higher than in pose A. The potential energy gained, and the kinetic energy lost, from pose B to pose C is

$$\Delta PE = -Mgl_2(1-\cos(\theta_2))$$

Thus, stretching the swing leg like this increases the amount of lateral kinetic energy that can be absorbed. Of course, it requires that the system begin in a "crouched" position, where the legs are initially bent, so that the swing leg can stretch. This is what a baseball player does, while running, when approaching a base. Note also that stretching the leg in this way can also be beneficial for exerting additional lateral braking force in double support, even if the vertical position of the COM does not change from pose B to C (see Eqs. E.28 – E.31, and related discussion).

1.4 Model with Foot

The biped model is now extended so that the ground contact of the legs is not at a point, but rather, is via a foot that forms an extended, but finite, region of support. This allows for exertion of ankle torques. However, these torques are limited so that the FRI (see Chapter 3) remains within the support region, so that the foot does not roll. We consider, now, how the previous results change when such ankle torques are included.

Let's consider a single step, as before. From pose A to pose B, let's suppose that the FRI moves to the front of the foot, the most advantageous position for applying a lateral braking action. While the CM is to the left of the FRI (in the above diagrams), a braking force can be applied by the stance foot. This force is similar to that described in Eqs. E.28 – E.31. The magnitude of the lateral force, and the time during which it can be applied, is a function of how far the foot support extends.

Consider the detailed diagram in Fig. E.8 that shows the model between pose A and B.

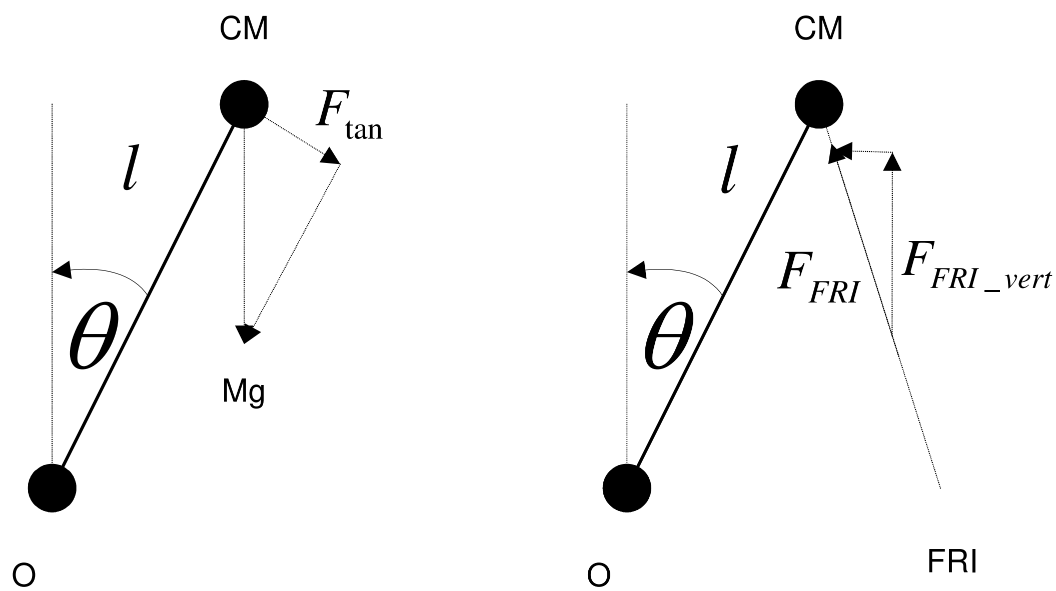


Fig. E.8 – Detailed diagram of force applied due to FRI.

The base of support of the foot is small compared with the length of the leg. Therefore, the angle θ, in Fig. E.8, will be small, and equations similar to Eqs. E.28 – E.31 can be employed. Thus, the vertical position of the CM is fixed at

$$z_{com} = l = 1 \qquad \text{(E.46)}$$

Once again, the ratio of lateral to vertical position is equal to the ratio of lateral to vertical force; therefore, the lateral force is

$$F_{lat} = Mg \frac{x_{cm}}{z_{cm}} \qquad \text{(E.47)}$$

Lateral acceleration is simply

$$\ddot{x}_{cm} = \frac{F_{lat}}{M} \qquad \text{(E.48)}$$

Combining Eqs. E.47 and E.48 yields the equation of motion:

$$\ddot{x}_{cm} - \frac{g}{z_{cm}} x_{cm} = 0 \qquad \text{(E.49)}$$

The roots of the characteristic equation are

$$s_1, s_2 = \frac{\pm\sqrt{4\frac{g}{z_{com}}}}{2} = \frac{\pm\sqrt{4\frac{9.8}{1}}}{2} = \pm 3.13 \qquad \text{(E.50)}$$

The solution is

$$x_{cm} = k_1 e^{s_1 t} + k_2 e^{s_2 t} \qquad \text{(E.51)}$$

$$\dot{x}_{cm} = k_1 s_1 e^{s_1 t} + k_2 s_2 e^{s_2 t}$$

Suppose the initial position is –0.2, and the initial velocity is 0.5. Setting t = 0 in Eq. E.51 yields

$$-0.2 = k_1 + k_2 \tag{E.52}$$
$$0.5 = (k_1 - k_2)3.13$$

Solving these equations yields

$$k_1 = -0.02 \tag{E.53}$$
$$k_2 = -0.18$$

Finally, substituting in values yields

$$t = \frac{1}{6.26}\ln\left(\frac{0.18}{0.02}\right) = 0.35 \text{ s} \tag{E.54}$$

From Eq. E.51, the position at this point in time is

$$x_{com} = -0.02e^{3.13\times 0.35} - 0.18e^{-3.13\times 0.35} = -0.06 - 0.0602 = 0 \tag{E.55}$$

This shows that, for slow walking speeds, ankle pitch torque can be enough to stop forward motion, without any additional stepping. This means that the foot base can play a significant role, and should not be omitted from models. Ankle torque can also be employed beneficially in pose B, after foot strike. Here, the FRI is moved forward, effectively increasing angle θ_2.

1.5 What is the best foot placement?

The above analysis provides useful guidelines for determining foot placement in response to disturbances. It is convenient to first consider disturbances to standing in an upright pose, and then to extend this to disturbances in crouched poses, or while walking.

The previous analysis shows that foot placement is largely determined by requirements of lateral kinetic energy that has to be absorbed. The larger the amount to be absorbed, the further the foot should be placed. This has to be balanced against the desire to limit impact forces, and kinematic joint limit constraints. Such limits vary from person to person. Thus, individual limits and intent plays a significant factor. An individual may prefer to take several smaller steps, or one large step, to dissipate the lateral kinetic energy.

As explained previously, a crouched position increases potential braking capability. Thus, if the individual anticipates a disturbance (a football player, for example), or the need to slow down quickly (a baseball player approaching a base, for example), the individual will assume a more crouched position.

Appendix F – Proofs of Lemmas and Theorems

Chapters 5 and 7, which describe the qualitative control plan and the plan compiler that computes it, contain a number of lemmas and theorems. The proofs for these lemmas and theorems are provided in this Appendix.

Lemma 5.1: Given a fixed-duration tube, $tube = TUBE(CA, D)$ (Def. 5.4), and an associated SISO system $s = S(A(CA))$ (Def. 5.2), if the state, $\langle y(t_i), \dot{y}(t_i)\rangle$, of s at time t_i is on a trajectory that is in $tube$, and t_i is between 0 and D, that is, $0 \le t_i \le D$, and if there are no disturbances after this time, that is, during $t_i \le t \le D$, then the state of s is guaranteed to reach the goal region of CA at time D. A state, $\langle y(t_i), \dot{y}(t_i)\rangle$, is in $tube$ at t_i if the trajectory position and velocity at t_i are in the tube's cross section at t_i; $\langle y(t_i), \dot{y}(t_i)\rangle \in SEC(tube, t_i)$ (Def. 5.5).

Proof of Lemma 5.1

From Def. 5.4, if a trajectory, $traj$, is an element of the tube ($traj \in TRAJ = TUBE(CA, D)$), then it will be in the activity's goal region at time D ($(y(D), \dot{y}(D))(traj) \in R_{goal}(CA)$). Now, if the state of s is on a plant trajectory (Def. 4.2), it will remain on that trajectory if there are no further disturbances; the state of the SISO system will evolve according to Definition 4.2. Thus, if the state $\langle y(t_i), \dot{y}(t_i)\rangle$, of s at time t_i is on trajectory $traj$ ($\langle y(t_i), \dot{y}(t_i)\rangle = \langle y(t_i), \dot{y}(t_i)\rangle(traj)$), then if there are no disturbances, it will remain on the trajectory for all time from t_i to D ($\langle y(t), \dot{y}(t)\rangle = \langle y(t), \dot{y}(t)\rangle(traj) \forall t : t_i \le t \le D$). In particular, the state will be on the trajectory at time D ($\langle y(D), \dot{y}(D)\rangle = \langle y(D), \dot{y}(D)\rangle(traj)$). Therefore, from Def. 5.4, the state will be in the goal region at time D.

Lemma 5.2: Given a controllable control activity, CA (Def. 5.7), if the state for CA is in $R_{init}(CA)$, then a control setting exists that causes the state to reach the activity's goal region, $R_{goal}(CA)$, at any desired time within the range $[l(CA), u(CA)]$, if there are no further disturbances during execution of CA.

Proof of Lemma 5.2

From Definition 5.7, the initial region of CA is a subset of the initial region of every fixed-duration tube in the controllable tube set of CA ($R_{init}(CA) \subseteq INITSEC_{controllable}$, where $INITSEC_{controllable} = \bigcap_{tube} SEC(tube, 0)$, and $tube \in TUBES_{controllable}$. Therefore, if the state $\langle y, \dot{y} \rangle$, of the associated SISO system, $s = S(A(CA))$, is in $R_{init}(CA)$, then it is also in the initial region of every fixed-duration tube in $TUBES_{controllable}$. From Definition 5.7, this set of tubes contains a fixed duration tube that reaches the goal region for every time in the interval $[l(CA), u(CA)]$ ($TUBES_{controllable} = \bigcup_{D} \{TUBE(CA, D) \mid l(CA) \leq D \leq u(CA)\}$). Therefore, from Lemma 5.1, because the state $\langle y, \dot{y} \rangle$ is in the initial region of each of these tubes, it can be made to reach the goal region after a duration corresponding to that of any of these tubes (it can be made to reach the goal region at any desired time within the range $[l(CA), u(CA)]$) if there are no further disturbances during execution of CA.

Theorem 5.1 (Successful execution of a controllable control activity): Let CA be a controllable control activity, and s, the SISO system associated with CA ($s = S(A(CA))$), (Def. 5.2). If the state of CA is in $R_{init}(CA)$, and if there are no further disturbances during execution of CA, then there exists a constant control parameter setting $\langle y_{set}, \dot{y}_{set} kp, kd \rangle$ which, when applied to s (Def. 4.1), results in a trajectory $y(t)$, and a duration, D, consistent with a schedule T, such that:

1) the activity in the QSP corresponding to CA, $A(CA)$, is satisfied by $y(t)$ and T, as defined by Definition 4.9
2) D, is within the temporal bounds of CA ($l(CA) \leq D \leq u(CA)$).

Proof of Theorem 5.1

The second point, that D, is within the temporal bounds of CA ($l(CA) \le D \le u(CA)$) follows directly from Lemma 5.2. For the first point, let $a = A(CA)$ be the QSP activity corresponding to CA. The start and finish times, t_s and t_f of a are specified, by T, as $t_s = T(ev_s(a))$, and $t_f = T(ev_f(a))$ (see Def. 4.4). If the state of CA is in $R_{init}(CA)$, at time t_s, then it is in $RS_{init}(a)$ ($\langle y(t_s), \dot{y}(t_s) \rangle \in RS_{init}(a)$), because, from Definition 5.2, $R_{init}(CA) \subseteq RS_{init}(a)$. This is one of the requirements of Definition 4.9. In order for D to be consistent with T, $t_f - t_s = D$. From Lemma 5.2, the state after duration D (at time t_f) is in $R_{goal}(CA)$. Therefore, it is in $RS_{goal}(a)$ ($\langle y(t_f), \dot{y}(t_f) \rangle \in RS_{goal}(a)$), because, from Definition 5.2, $R_{goal}(CA) \subseteq RS_{goal}(a)$. This is another requirement of Definition 4.9. Finally, from Lemma 5.2 and Definition 5.4, $y(t)$ satisfies the operating constraints of the activity. Therefore, all requirements of Definition 4.9 are satisfied, and hence, point 1 of Theorem 5.1 is satisfied.

Theorem 5.2 (Successful execution of a correct QCP for a QSP): Let qsp be a qualitative state plan, and qcp, a correct qualitative control plan for qsp. If for each initial activity, CA, in qcp, the state associated with CA is in $R_{init}(CA)$, and if there are no further disturbances, then there exists a schedule, T, and there exist constant control parameter settings for each activity, resulting in trajectory set Y, of SISO plant trajectories (Def. 4.2), such that Y and T satisfy qsp according to Definition 4.7. The set of initial activities is the set of activities with no predecessor.

Proof of Theorem 5.2

Definition 4.7 has two requirements. The second requirement is that $\langle Y, T \rangle$ satisfies all activities in qsp (Def. 4.9). If the state for each initial activity is in $R_{init}(CA)$, then, from Theorem 5.1, Definition 4.9 is satisfied for each of these initial activities.

Now, Definition 5.3 states that qcp must controllable, as defined in Definition 5.8, if it is to be a correct QCP for qsp. Point 3 of Definition 5.8 requires that the goal regions of all control activities in qcp are subsets of the initial regions of their successors. Therefore, the trajectory state of each SISO system after execution of the initial activities will be in the initial regions of all successors. Applying Theorem 5.1 recursively to the successors, Definition 4.9 is satisfied for all activities in qcp. Therefore, the second requirement of Definition 4.7 is satisfied.

The first requirement of Definition 4.7 is that T be consistent with qsp according to Definition 4.8. From point 2 of Definition 5.8, the temporal bounds, $[l,u]$, of all control activities in qcp are consistent with the plan's temporal constraints $TC(qsp)$. Because, as explained for the second requirement of Definition 4.7, Definition 4.9 is satisfied for all activities in qcp, all activity durations are within the temporal bounds $[l,u]$. Therefore, from Theorem 5.1, and from point 2 of Definition 5.8, a schedule T that is consistent with all activity durations will also be consistent with the plan's temporal constraints.

Theorem 7.1 (GFT and GST for a two spike control law): Let CA be a control activity (Def. 5.2), with controllable duration bound $[l,u]$, and regions R_{init} and R_{goal}, with points for these regions A, B, C, and D, as specified in Definition 7.5. For a two-spike control law, constraints on the GFT and GST are then specified as:

$$\dot{y}(D) = \dot{y}(B) + \Delta v_1 + \Delta v_2 \qquad \text{(GFT)}$$

$$y(D) = y(B) + (\dot{y}(B) + \Delta v_1)l$$

$$\dot{y}(C) = \dot{y}(A) + \Delta v_3 + \Delta v_4 \qquad \text{(GST)}$$

$$y(C) = y(A) + (\dot{y}(A) + \Delta v_3)u$$

where Δv_1 and Δv_2 are the areas of the first and second spikes for the GFT, and Δv_3 and Δv_4 are the areas of the first and second spikes for the GST. If the actuation bound on the two-spike control law is $\max A$ (Def. 7.2), then the spikes are limited by the following inequality constraints:

$$-\max A \le \Delta v_1 \le \max A$$

$$-\max A \le \Delta v_2 \le \max A$$

$$-\max A \le \Delta v_3 \le \max A$$

$$-\max A \le \Delta v_4 \le \max A$$

Additionally, to ensure that the initial region, the goal region, and the controllable duration, are not empty, we require that

$$\begin{aligned}
&l \le u \\
&y(A) \ge y(B) \\
&\dot{y}(A) \ge \dot{y}(B) \\
&y(C) \ge y(D) \\
&\dot{y}(C) \le \dot{y}(D)
\end{aligned}$$

Proof of Theorem 7.1

The form of the position and velocity trajectory equations for the GFT and GST is obtained by integrating the acceleration trajectory of the two-spike control action. All that remains is to show that the GFT begins at point B and ends at point D, and that the GST begins at point A and ends at point C.

As stated in Section 7.2.4, because the GFT is guaranteed to be the fastest trajectory from any point in the initial region (Def. 7.4), we must consider the worst-case starting point. This is point B, because it is the minimum velocity point in the initial region that is furthest from the goal. Hence, from any point in the initial region, we are guaranteed to get to the goal region at least as quickly as we can if we start from point B. In order to understand where the GFT should end, consider that Definition 7.4 requires that it end at some point in the goal region; any point in the goal region is acceptable. Therefore, we may consider the best-case ending point. This is point D, because it is the maximum velocity point in the goal region that is nearest to the goal. Hence, from any particular point in the initial region, we are guaranteed to get to point D at least as quickly as any other point in the goal region.

Similarly, because the GST is guaranteed to be the slowest trajectory from any point in the initial region, the worst-case starting point is point A, because it is the maximum velocity point in the initial region that is closest to the goal. Hence, from any point in the initial region, we are guaranteed to get to the goal region at least as slowly as we can if we start from point B. The best-case end point for the GST is point C, because it is the minimum velocity point in the goal region that is furthest from the goal. Hence, from any particular point in the initial region, we are guaranteed to get to point C at least as slowly as any other point in the goal region.

Theorem 7.2 (R_{init} for a two spike control law): The rectangular initial region, R_{init}, specified by any pair of initial GFT and GST points specified in Lemma 7.1 is a subset of $INITSEC_{controllable}$ $(R_{init} \subseteq INITSEC_{controllable})$. Furthermore, R_{init} is maximal in that the velocity $\dot{y}(A)$ for point A is the maximum possible velocity for position $y(A)$ of point A. Similarly, the velocity $\dot{y}(B)$ for point B is the minimum possible velocity for position $y(B)$ of point B.

Proof of Theorem 7.2

The first point follows directly from Lemma 7.2. The second point follows directly from Lemma 7.1.

Theorem 7.3 (GFT and GST for a PD control law): Let CA be a control activity (Def. 5.2), with controllable duration bound $[l,u]$, and regions R_{init} and R_{goal}, with points for these regions A, B, C, and D, as defined in Definition 7.5. The acceleration input to the SISO system of CA is computed according to a PD control law, using the SISO system's control parameters (Definitions 4.1 and 4.2). If there exists a control parameter setting that results in a trajectory from B to D that is a member of the controllable tube set (Def. 5.7), then there exists a control parameter setting that results in the GFT for CA, and this GFT begins at B and ends at D. Similarly, if there exists a control parameter setting that results in a trajectory from A to C that is a member of the controllable tube set (Def. 5.7), then there exists a control parameter setting that results in the GST for CA, and this GST begins at A and ends at C. The trajectory equations for the GFT and GST are then specified as

$$\begin{aligned} y(D) &= f_1\big(y(B), \dot{y}(B), \langle y_{set}, \dot{y}_{set}, kp, kd \rangle(GFT), 0, l\big) \\ \dot{y}(D) &= f_2\big(y(B), \dot{y}(B), \langle y_{set}, \dot{y}_{set}, kp, kd \rangle(GFT), 0, l\big) \end{aligned} \qquad \text{(GFT)}$$

$$
\begin{aligned}
y(C) &= f_1\left(y(A), \dot{y}(A), \langle y_{set}, \dot{y}_{set}, kp, kd \rangle (GST), 0, u\right) \\
\dot{y}(C) &= f_2\left(y(A), \dot{y}(A), \langle y_{set}, \dot{y}_{set}, kp, kd \rangle (GST), 0, u\right)
\end{aligned} \qquad \text{(GST)}
$$

To ensure that the initial region, the goal region, and the controllable duration, are not empty, we require that

$$
\begin{aligned}
&l \leq u \\
&y(A) \geq y(B) \\
&\dot{y}(A) \geq \dot{y}(B) \\
&y(C) \geq y(D) \\
&\dot{y}(C) \leq \dot{y}(D)
\end{aligned}
$$

Proof of Theorem 7.3

The form of the position and velocity trajectory equations for the GFT and GST is obtained from Eqs. 7.2 and 7.3. All that remains is to show that the GFT begins at point B and ends at point D, and that the GST begins at point A and ends at point C. The proof of this is the same as that for Theorem 7.1, because we assume that position changes monotonically.

Theorem 7.4 (R_{init} for a PD control law): The rectangular initial region, R_{init}, specified by any pair of initial GFT and GST points specified in Lemma 7.3 is a subset of $INITSEC_{controllable}$ $(R_{init} \subseteq INITSEC_{controllable})$. Furthermore, R_{init} is maximal in that the velocity $\dot{y}(A)$ for point A is the maximum possible velocity for position $y(A)$ of point A. Similarly, the velocity $\dot{y}(B)$ for point B is the minimum possible velocity for position $y(B)$ of point B.

Proof of Theorem 7.4

The first point follows directly from Lemma 7.4. The second point follows directly from Lemma 7.3.