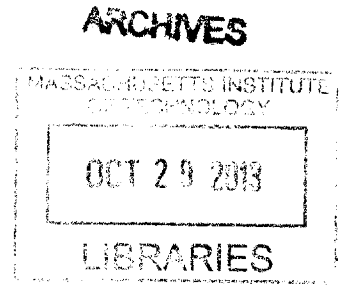
Reactive Integrated Motion Planning and Execution Using Chekhov



by

Ameya Shroff

S.B., Massachusetts Institute of Technology (2012)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

© Massachusetts Institute of Technology 2013. All rights reserved.

Author ....
Department of Electrical Engineering and Computer Science
May 24, 2013

Certified by...........................
Brian C. Williams
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by.............................
Prof. Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

Reactive Integrated Motion Planning and Execution Using Chekhov

by

Ameya Shroff

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2013, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

We envision a world in which robots and humans can collaborate to perform complex tasks in real-world environments. Current motion planners successfully generate trajectories for a robot with multiple degrees of freedom, in a cluttered environment, and ensure that the robot can achieve its goal while avoiding all the obstacles in the environment. However, these planners are not practical in real world scenarios that involve unstructured, dynamic environments for a three primary reasons. First, these motion planners assume that the environment the robot is functioning in, is well-known and static, both during plan generation and plan execution. Second, these planners do not support temporal constraints, which are crucial for planning in a rapidly-changing environment and for allowing task synchronisation between the robot and other agents, like a human or even another robot. Third, the current planners do not adequately represent the requirements of the task. They often over-constrain the task description and are hence unable to take advantage of task flexibility which may aid in optimising energy efficiency or robustness. In this thesis we present Chekhov, a reactive, integrated motion planning and execution executive that addresses these shortcomings using four key innovations. First, unlike traditional planners, the planning and execution components of Chekhov are very closely integrated. This close coupling blurs the traditional, sharp boundary between the two components and allows for optimal collaboration. Second, Chekhov represents temporal constraints, which allows it to perform operations that are temporally synchronised with external events. Third, Chekhov uses an incremental search algorithm which allows it to rapidly generate a new plan if a disturbance is encountered that threatens the execution of the existing plan. Finally, unlike standard planners which generate a single reference trajectory from the start pose to the goal pose, Chekhov generates a Qualitative Control Plan using Flow Tubes that represent families of feasible trajectories and associated control policies. These flow tubes provide Chekhov with a flexibility that is extremely valuable and serve as Chekhov's first line of defence.

Thesis Supervisor: Brian C. Williams

Title: Professor of Aeronautics and Astronautics

Acknowledgements

Over the course of this complex and eclectic montage that we so mundanely call Life, we meet wonderful new people and build strong friendships; we try to appreciate, understand and learn from unfamiliar experiences; we explore, question and challenge the world around us and, we try to grow as individuals while recognising and preserving the core values and beliefs that define us. We aspire to learn from the past and cherish the wonderful memories, rejoice in the present and look joyfully towards the future. These experiences, relationships, emotions and aspirations mould us in to the people we are, and strongly influence the individuals that we will grow into in the years to come. In the first few paragraphs of my thesis, I would like to thank those who have had a profound impact on my life, and have shaped me into the person that I am today. Without their companionship, support and advice, this thesis would not have been possible, and my life as I know it, would have been immeasurably diminished.

I would like to begin by expressing my gratitude to my thesis supervisor, Professor Brian Williams, for giving me the opportunity to work on this thesis and be part of the wonderful community at the Model-Based Embedded Robotics Systems (MERS) Group at CSAIL. His invaluable guidance, insightful suggestions, and constant encouragement were instrumental in ensuring that my research reached its full potential. I would also like to thank Dr. Andreas Hofmann for the inspiration, continuous support and patient guidance he has given me over this last year. His vision, technical expertise, experience and willingness to help, have greatly facilitated my progress and and have been absolutely indispensable in shaping this thesis.

This thesis has enjoyed the generosity of the Boeing Company. I would like to thank them for having faith in our ability to steadily deliver. I am extremely grateful to Mr. Scott Smith from the Boeing Company for his insightful suggestions over the last year.

I consider myself extremely lucky to have been part of the MERS community. Steve Levine, Pedro Santana, James Paterson, Enrique Gonzalez, Andrew Wang,

Peng Yu, David Wang, Simon Fang, Eric Timmons, Larry Bush, Dan Strawser and Derek Aylward, I am extremely grateful to each of you for your humility, humour and encouragement. Your versatility, hard-work and brilliance will never cease to amaze me; I have learnt a great deal from all of you and you have made my M.Eng year extremely enjoyable.

I would also like to thank my academic advisor, Professor Duane Boning, who has guided me throughout my tenure at MIT, Ms. Julie Norman, for her invaluable advice and her unflagging belief in me, Ms. Anne Hunter, for all her help, guidance and support over these last five years, and Professor Patrick Winston for his humour, brilliant life-lessons, advice, encouragement and overall genius.

To my closest friends, Keshav Puranmalka, Neel Patel, Nikola Otasevic, Reece Otsuka, Latif Alam, William Miraval Morejon, Burkay Gur, Thais Terceiro Jorge, Natasha Nath and Michalis Rossides, these last five years at MIT have been some of the most thrilling, wonderful and memorable years of my life. This incredible journey would have been indescribably altered if any of you had not been part of it. I hope that each of you will remember these marvellous times as I do: The walks along the Charles, Dorm-Row and Newbury Street; the lunches, brunches and dinners in Boston; the oft futile search for taxis along Boston Commons in the wee hours of the morning, the irreplaceable conversations, the sight of the early morning sun as we worked together on problem sets and the last-minute studying before Finals. I thank you all for these invaluable moments, these indelible memories. Though the sands of time will surely pass through the hourglass of Life and though the winds of change may carry us far away, I am certain that our friendship will not only endure, but flourish, for we have shared something spectacular, something special, something truly magical over these last few years. I will miss you dearly.

I would also like to thank my friends from the Cathedral and John Connon School in Bombay. Sarvesh Harivallabhdas, Zahan Malkani, Rahul Krishnan, Mrinal Kapoor and Arka Bhattacharyya, I have dearly missed seeing you everyday and the times that we shared in Bombay. Our conversations with the age-old humour and seeming frivolity have been something that I have cherished and thoroughly enjoyed during my

time at college; I am certain that these will continue in the years to come. You have all influenced the person that I am in more ways that I can say. I know that whatever we do and however much we change, our time together in Bombay will always be an integral part of who we are. For this and for everything else, I am extremely thankful.

Finally, I am tremendously grateful to my family. My four grandparents have always been my inspiration. Their kindness, humility, wisdom and compassion, and their ideals have been instrumental in shaping me into the person I am.

I am deeply and eternally indebted to my parents and my brother for the unconditional love and support they have shown me throughout my life. Their strength-of-character, foresight, dedication and intellect have had an indescribable impact on me. They have inculcated in me a strong appreciation for the importance of education, morality, humility, resilience and hard-work in one's life. I am truly privileged and lucky that theirs were the first hands that moulded my life. Mama, Papa and Rahul, you are the reason that I am here today; everything that I am, and everything that I have achieved is because of you, your love, your encouragement, your belief and your unwavering faith in me and in my abilities. You will always be the force that guides me. This thesis is dedicated to you.

Contents

1 Introduction **15**

2 Problem Statement **23**

2.1 Overview . . . 23

2.1.1 An Intuitive Introduction to the Problem Being Solved . . . 24

2.1.2 Disturbances and Adjustments . . . 27

2.2 Formal Chekhov Problem Statement . . . 32

3 The Chekhov Executive **35**

3.1 The System Architecture . . . 35

3.2 The Reactive Motion Planner . . . 36

3.2.1 Graphical Representation of the Search Space . . . 41

3.2.2 Flow Tubes . . . 43

3.2.3 Fast Incremental Plan Adjustment . . . 54

3.2.4 Summary . . . 60

3.3 The Motion Executive . . . 61

3.3.1 The Constraint Tightening Component . . . 63

3.3.2 Local Adjustment . . . 64

3.3.3 Local Adjustment in Velocity Limited Systems . . . 70

3.3.4 The Constraint Tightening Algorithm . . . 71

4 The Manufacturing Test Bed **77**

4.1 Vision for Robotic Manufacturing . . . 77

4.2 System Capabilities 78
4.3 System Architecture 79
4.3.1 The Chekhov Executive ROS Node 82
4.3.2 The Hardware Module 83
4.3.3 The Sensing and State Estimation Module 83
4.4 Results 84
4.4.1 Results for the Reactive Motion Planner 84
4.4.2 Results for the Motion Executive 88
4.4.3 Discussion 89

5 Contributions **95**

List of Figures

1-1 Auto-mobile manufacturing factory 16

1-2 An assembly task performed by two robotic manipulators. 20

2-1 A basic manipulation scenario 25

2-2 A basic representation of Chekhov's output 26

2-3 A disturbance caused by a change in the current state of the robot . . 28

2-4 A disturbance caused by a change in the goals to be achieved 29

2-5 A disturbance caused by a change to obstacles in the environment . . 30

3-1 The system architecture of the Chekhov Executive 37

3-2 Cars travelling along a road 39

3-3 Comparison of a plan generated by a traditional motion planner and one generated by Chekhov's Reactive Motion Planner. 40

3-4 The search-space graph generated by D* Lite. 42

3-5 Comparison of the state-space graph generated by a traditional motion planner and one generated by Chekhov's Reactive Motion Planner. . 44

3-6 The internal and external flow tube approximation. 46

3-7 The flow tube. 48

3-8 A Flow tube comprises a set of cross-sections. 51

3-9 Flow Tube computation. 53

3-10 Plan generated by the Reactive Motion Planner. 55

3-11 The salient features of the Reactive Motion Planner 62

3-12 A Local Adjustment made to a trajectory 65

3-13 Procedure for finding and making a Local Adjustment 68

3-14 The constraint tightening algorithm. 75

4-1 The system architecture of the Chekhov Robotic Test Bed 80

4-2 Representative test case for the Chekhov Executive 85

4-3 Representative examples of the cluttered environments used to test the Motion Executive. 90

4-4 Plan execution in a cluttered environment. 91

4-5 Flow Tubes generated by the Constraint Tightening algorithm. 92

List of Tables

4.1 Results obtained after testing the Reactive Motion Planner 86

Chapter 1

Introduction

The pinnacle of achievement for many researchers in the field of Artificial Intelligence is to create a robotic system that is truly sentient and aware. We envision a world in which humans and machines interact seamlessly in a rapidly-changing environment. Today, robots are used quite extensively in mass-production factories. However, these robots only perform repetitive tasks with little or no variability, and that require the robot to make no significant execution-time adjustments. These robots perform these tasks by simply executing a set of pre-defined commands or a pose trajectory that is specified by a human operator. Moreover, these robots can only function effectively in manufacturing scenarios that are highly structured and controlled. More specifically, the environment in which these robots operate is set up with a specific task in mind; they have limited sensing capabilities and function essentially in an open-loop system. They are not aware of their surroundings and are not robust to environment changes. These robots expect the world to be in a given state, and *only* that state, when they perform their function. For example, let us consider a typical manufacturing plant, such as the one depicted in Figure 1-1. In these factories, robots are often used to weld different parts of an auto-mobile together. The parts that are to be welded together must be placed in exactly the right position to obtain the expected result. Even a slight error in positioning these parts could result in a defective end-product.

Most manufacturing scenarios require a much greater level of flexibility than the one described above; for example, we imagine a manufacturing scenario in which

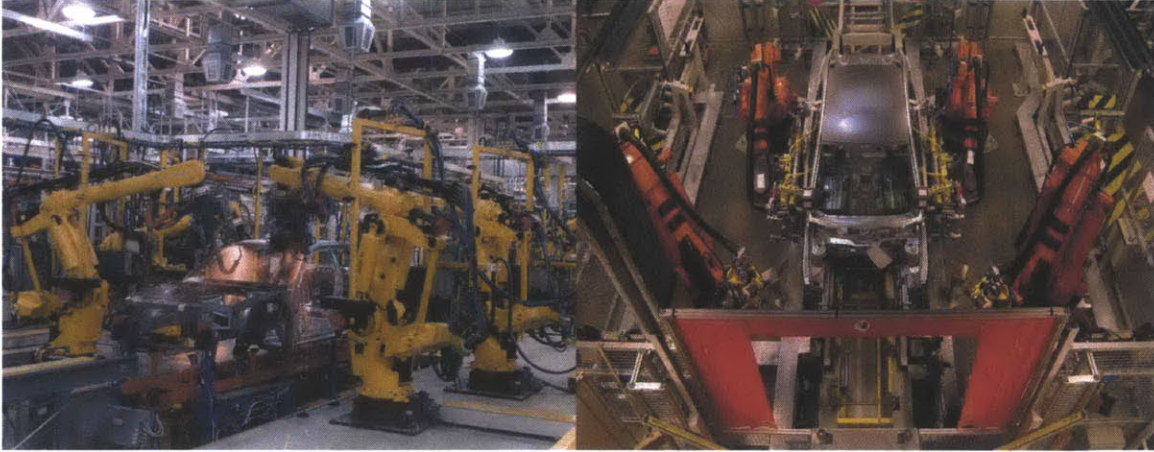


Figure 1-1: Auto-Mobile Manufacturing Factory.

humans and robots collaborate to perform tasks in an unstructured environment. One of the fundamental steps to realising this vision is to create a robotic system that can operate and perform complex manipulation tasks in a dynamic environment. In particular, a robot must be aware of the environment around it, and must stop immediately when it is performing a task if an obstacle, a robotic helper or a human, obstructs its path. Furthermore, if the obstacle remains in its path, the robot should not idly wait until another agent clears its path; it should try to circumvent this obstacle and continue performing its task. If the robot is unable to find a path around the obstacle, it should signal a failure to a human or to a robotic helper and actively ask these agents for help in fulfilling its task. In essence, the robot should react to a given situation much like a human would react if he was placed in a similar situation.

Creating a robotic system that can safely and effectively operate in uncertain environments is an intriguing problem that is fraught with a number of stimulating challenges. The first impediment arises from the geometric structure of the environment and the geometric structure of the robot itself. The geometric relations between the two can evolve and change in many unexpected ways. Moreover, most tasks in the real world have an associated duration (or time) constraint. In particular, these tasks specify a time interval within which the robot must complete the assigned task. In

addition, a robot has a set of actuation limits and associated dynamics that may make it impossible for the robot to successfully perform the assigned task in the allotted duration. A robotic system that aspires to operate in an unstructured environment must account for the actuation limits and associated dynamics of the robot and generate its plans accordingly. Furthermore, the task that the robot must perform, may itself be extremely intricate. This only adds to the overall complexity of the problem. In these situations, it becomes vital for the planner to ascertain whether there is any level of flexibility possible in the generation and the execution of the plan. Finally, the complexities of planning and control in higher-dimensional state spaces that arise when dealing with robotic manipulators in three-dimensional environments make the problem of reacting swiftly to disturbances particularly challenging due to the computational effort, combined with the real-time requirements and constraints on the planner.

Although current motion planners can solve complex planning problems, they do not address these requirements adequately. Current motion planners successfully generate trajectories for a robot with multiple degrees of freedom in a cluttered environment, and ensure that the robot can achieve its goal while avoiding all the obstacles in the environment. However, these planners are not practical in real world scenarios that involve unstructured, dynamic environments for a number of reasons. First, these motion planners assume that the environment that the robot is functioning in, is well-known and static, both during plan generation and plan execution. The environment model provided to these planners is a geometric representation of the environment determined by a vision system. Additionally, these planners do not support temporal (time-related) constraints which are a crucial requirement for planning in a rapidly-changing environment and for allowing task synchronisation between the robot and other agents, like a human or another robotic helper. Moreover, the current planners do not adequately represent the requirements of the task. They often over-constrain the task description and this prevents the planners from taking advantage of the flexibility in the task that may be available to them. The fact that these planners neglect temporal constraints only exacerbates the problem. The flexibilities

in the task may aid in optimising the energy efficiency and robustness of the plan generated by the motion planner.

For example, let us consider a situation in which a robot is tasked with picking up an object from a table. This task can be described as follows:

Move the end-effector in the vicinity of (x,y,z) in 10-20 seconds

However, current planners represent this in a much more constrained fashion:

Move the end-effector to position (x,y,z) in 15 seconds

These nominal goals, like the goal position (x,y,z) and the temporal constraint (15 seconds) become the only requirements as the planner cannot handle a more flexible representation.

In this thesis, we present Chekhov, a reactive, integrated motion planning and execution system that addresses these issues using four key innovations.

The first of Chekhov's innovations is the fact that the planning and execution components of Chekhov are very closely integrated. Much of Chekhov's novelty arises from this close coupling. This intimate interaction blurs the sharp boundary between planning and execution that exists in traditional planners and this allows for optimal collaboration. Chekhov's second innovation is that it represents and observes temporal constraints. This ability to incorporate temporal constraints allows Chekhov to perform operations that are temporally synchronised with external events. For example, Chekhov can be used to control robotic "assembly chains". Consider the scenario illustrated in Figure 1-2 in which Robot A and Robot B are tasked with completing an assembly. Robot A has a sub-assembly that it must complete by itself, as does Robot B. However, they both need to cooperate to complete the entire assembly. One approach would be to allow one of the robots to complete its sub-assembly before allowing the other robot to begin working. However, this would be extremely inefficient. Instead, Chekhov can use temporal constraints to make the robots work on their sub-assemblies in parallel and cooperate in performing a task only when it is required. In the example illustrated in Figure 1-2, the tasks that the robots perform are temporally constrained. Both robots must have a section of their sub-assemblies ready at $t = 10$. In the time interval, ($t = 10$, $t = 15$), Robot A must help Robot

B position its sub-assembly. Robot B must then complete its last sub-assembly by $t = 25$, as Robot A readies to move its completed sub-assembly into position. In the time interval, ($t = 25$, $t = 30$), Robot B helps Robot A position its sub-assembly. Finally, in the time interval, ($t = 32$, $t = 35$), Robot A helps Robot B complete the assembly.

The third of Chekhov's innovations is that it uses an incremental search algorithm which allows it to rapidly generate a new plan, that modifies the existing plan, if a disturbance is encountered that prevents the existing plan from being executed successfully. This ability to react quickly to disturbances allows Chekhov to perform tasks even if there are changes in the environment. For example, while picking up an object from a moving conveyor belt, if the speed of the conveyor belt varies, Chekhov will be able to adjust to these variations and still pick up the object. Similarly, if Chekhov is trying to stop a rolling ball and the ball suddenly hits an obstacle and changes direction, Chekhov will adjust to this disturbance. Most importantly, if an obstacle appears and obstructs the path that Chekhov has generated, Chekhov will modify its motion plan to avoid the obstacle and achieve its goal, if possible. Finally, the last of Chekhov's major innovations is that, unlike standard motion planners which generate a single reference trajectory from the start pose to the goal pose, Chekhov generates a Qualitative Control Plan (QCP) using Flow Tubes that represent families of feasible trajectories and associated control policies. These flow tubes provide Chekhov with a flexibility that is extremely valuable. This flexibility not only allows Chekhov to, initially, choose the best path in a certain circumstance, but also allows it to quickly change trajectories in response to a disturbance without the need to generate an entirely new plan. This serves as Chekhov's first line of defence and is tremendously useful.

We will analyse each of these innovations in detail in the subsequent chapters of this thesis.

This thesis comprises three major components. The first component comprises Chapter 2 (The Problem Statment). In this component we will introduce and motivate the problem that we try to solve in this thesis. We will then present the problem

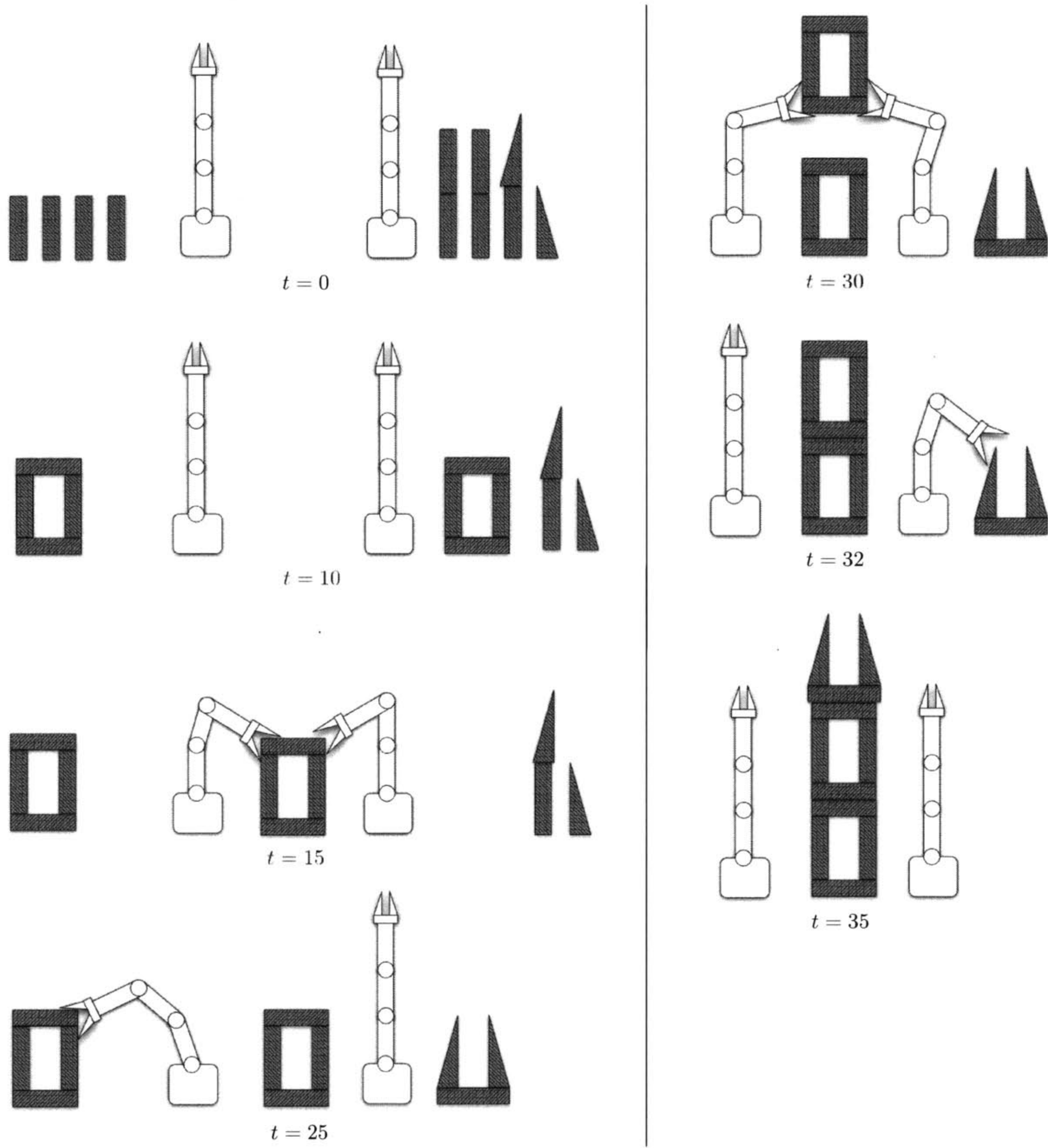


Figure 1-2: An assembly task performed by two robotic manipulators. Chekhov uses temporal constraints to make the robots work in parallel and cooperate, when necessary, to complete the assembly. Chekhov has not been demonstrated on this particular task, however, Chekhov's innovations make demonstrating such a collaborative task possible, in the future.

in an intuitive fashion before diving in and formally defining it. This component forms the foundation on which the rest of this thesis is based.

The second component of this thesis deals with the Approach and comprises Chapter 3 (The Chekhov Executive). It is the pith of this thesis and outlines our approach to solving the problem that we introduced in the Chapter 2. Specifically, this component introduces the Chekhov Executive and examines it in detail.

The final module of this thesis is the Evaluation component. Its comprises Chapter 4 (The Manufacturing Test Bed) and Chapter 5 (Contributions). In this component we validate the Chekhov Executive. We describe the system that we built in order to test the Chekhov Executive and present the results that were produced when the Chekhov Executive was tested in simulation and in reality, on a robotic test bed. Additionally, in this component we reiterate the contributions that this thesis has made to its field and discuss how we plan to continue this work in the future.

Chapter 2

Problem Statement

2.1 Overview

Chekhov is a reactive motion planning and execution system. Broadly speaking, when Chekhov is provided with a model of the environment,that is the robot and its workspace, an objective function, duration limits and a goal, it will generate an optimal, or near-optimal, plan that will allow a robot to move from its given start pose to a specified goal pose while avoiding the obstacles in the environment. If an obstacle moves and obstructs the path that Chekhov had originally generated, or if the goal state that was to be achieved is modified, Chekhov will adapt to these *disturbances* by modifying the existing plan, if possible. If Chekhov is unable to *successfully adjust* the existing plan so as to allow the robot to achieve the goal while avoiding the environment's disturbances, Chekhov will discard the original plan and generate an entirely new one. Chekhov focusses on providing a fast reactive capability that is essential for robotic manipulation in unstructured, real-world environments.

More specifically, Chekhov solves a problem that can be framed as follows:
Chekhov plans and executes the motions of a robot in order to accomplish a task goal that is specified by a set of temporal and spatial constraints. Furthermore, the resulting motion plan is optimal or near-optimal with respect to the specified objective function. This objective function can be used to optimise energy efficiency,

robustness, speed, smoothness and a variety of other objectives.

In order to provide robots with the ability to operate efficiently and effectively in uncertain, dynamic environments, Chekhov must be able to react swiftly to disturbances. In particular, once plan execution has begun, the time that Chekhov needs to adjust the plan to respond to a disturbance must be significantly less than both the time needed to execute the adjusted plan, and the expected time to the next disturbance that is severe enough to warrant plan adjustment. If Chekhov does not meet these requirements, it would spend a disproportionate amount of time on computation dealing with what needs to be done, instead of actually performing the action required to alleviate the problem it is facing. This indecisive behaviour will cause Chekhov to become overloaded with disturbances that have occurred after the one it is currently handling, and it will not be able to react to all these disturbances in time. Furthermore, this uncertain behaviour could result in Chekhov continuing on its original course of action even in the presence of a disturbance, or could cause Chekhov to suddenly stop during execution. Both these eventualities could have catastrophic consequences in a rapidly-changing environment as they could damage the robot, the environment around the robot or the product the robot is working on; most importantly, they could harm a human that happens to be near the robot.

2.1.1 An Intuitive Introduction to the Problem Being Solved

We can thus intuitively define the Chekhov problem statement as follows.

Chekhov takes the following as **inputs**:

- A model of the environment.
- A plant model representing the actuation limits of the robot.
- The current state of the robot.
- A set of spatial and temporal constraints that represent the goals to be achieved.
- A set of operating constraints that specify the regions of operation of the robot.

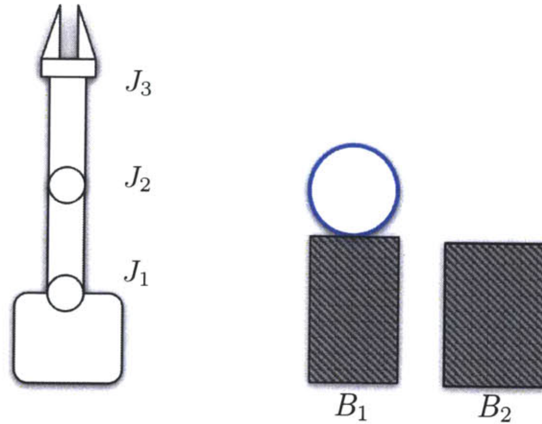


Figure 2-1: A basic manipulation scenario. The robotic manipulator is on the left. It is an arm with three degrees of freedom, $J1$, $J2$ and $J3$. The environment contains two obstacles, B_1 and B_2, which are depicted as shaded rectangles. The task goal is for the robotic arm to pick up the ball, which is located on B_1.

- An objective function.

Chekhov generates the following **output**:

A set of control commands to the robot that satisfies all the constraints defined in the input. Specifically, the control commands specify an optimal plan that achieves the goals set forth in the input and observes both the spatial constraints by avoiding all the obstacles in the environment, and the temporal constraints, by ensuring that the goal is achieved in the specified duration.

Definition 2.1 (Goal). *A **goal** is a particular robotic configuration, or robot pose, that Chekhov seeks to achieve.*

Consider the scenario in Figure 2-1. Figure 2-1 depicts a basic manipulation scenario. The figure is a snapshot of the initial state of the world. The robotic manipulator is shown on the left and it has three degrees of freedom, $J1$, $J2$ and $J3$. The environment has two obstacles, B_1 and B_2, which are illustrated as shaded rectangles. The task goal is for the robotic arm to pick up the ball, which is located on B_1.

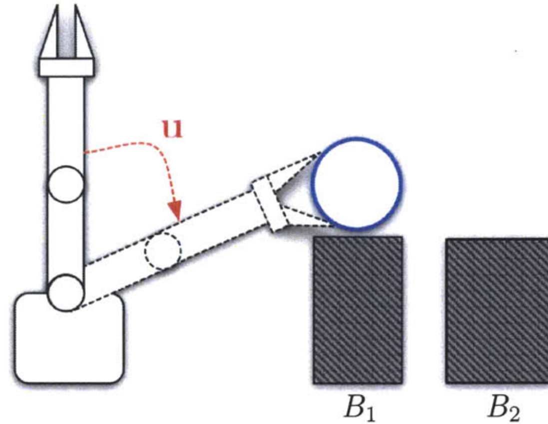


Figure 2-2: A basic representation of Chekhov's output. Chekhov issues a set of actuation commands **u**. The dashed arrow represents the movement caused by this actuation command set. The dashed illustration of the robotic arm, shows the final position of the arm once the set of actuation commands has been executed.

Chekhov will be provided with all this information as its inputs. In this example, the model of the environment will include a description of the world that the robot inhabits, including the positions of the obstacles and the free space that the robot can move through. Chekhov is also provided with information about the current state of the robot. This includes the current angle that each of the robot joints is in. In Figure 2-1, the robotic arm is shown to be standing upright, and hence, all its joint angles are 0 *rad*. Chekhov's goal, or goal state, is a pose in which the robotic arm is holding the ball on Block B_1. This goal must be achieved in in a specified time interval d.

In the current scenario depicted in Figure 2-1, Chekhov's output would be a set of actuation commands (a single command in this case) of this form:

Move Joint 1 of the robotic arm clockwise $\pi/4\,rad$*, with an angular velocity of* 3 *rad/sec.*

That is, a set of commands to the robotic arm such that the robotic arm can pick up the ball on Block B_1. This is depicted in Figure 2-2. Chekhov issues a set of actuation commands **u**.

$$\mathbf{u} = [u(t_i)\ u(t_{i+1})\ ...\ u(t_f)]$$

where $u(t_i)$ is the sequence of actuation commands that the robot executes at time t_i.

The dashed arrow represents the movement caused by this actuation command set. The dashed illustration of the robotic arm shows the final position of the arm once the set of actuation commands has been executed by the arm. The velocities and accelerations that result from the actuation commands must be within the actuation limits specified by the plant model.

2.1.2 Disturbances and Adjustments

As mentioned previously Chekhov's emphasis is on providing robots with the capability of operating in rapidly-changing environments. Consequently, Chekhov's inputs can change quickly and unexpectedly with time as the motion plan is being executed. In most practical applications, these *changes* can be classified into 3 types.

Changes to the Current State of the Robot

Changes to the current state of the robot may occur and can have numerous causes. For example, the sensor information that was used to estimate the state of the robot may have been incorrect. Initially, Chekhov will assume that the robot is in this incorrect state and generate a plan accordingly, however, there may be a correction made to this state estimate of the robot's current state based on newer, more accurate data. The current state of the robot will have to be updated in order to ensure that the plan is generated using the most accurate data. Another example situation, is one in which an unexpected force, like a falling object, is applied on the robot. This may result in a change in the current state of the robot. Finally, a third scenario is one in which the robot is grasping an object in its gripper. In such situations, it is common practice to incorporate the state of the grasped object into the overall robot state (while the object remains grasped by the robot). Consequently, if the object

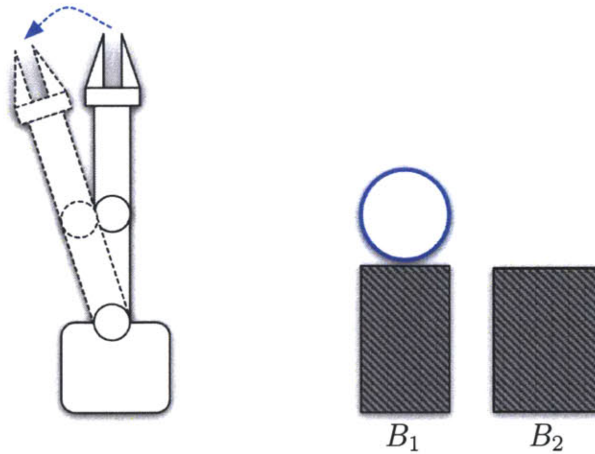


Figure 2-3: A disturbance caused by a change in the current state of the robot. The dashed illustration of the robotic arm represents the new state of the robotic arm, while the solid illustration of the robotic arm represents the initial state of the robotic arm. The dashed arrow represents the movement of the robotic arm from its initial state to its new state.

slips, it constitutes a change in the overall current state of the robot. Additionally, this slip may force the robot to further change its configuration in order to fix its grip, so that the object can still be placed correctly, as required by the task. Figure 2-3 is a representative example of these changes. It depicts a change in the current state of the robotic arm. The robotic arm moves from its initial state, illustrated by the solid robotic arm, to its new state, illustrated by the dashed robotic arm. The dashed arrow depicts this motion.

Changes to the Goals be to Achieved

Changes to the goals that Chekhov must achieve can occur frequently. For example, the human operator may suddenly decide that he does not want the robot to perform the task it was initially assigned and may decide to assign it a new task. As a result, the goals provided to Chekhov will change quite drastically. However, the goal can also be modified when the task remains the same; in fact, this change may be an essential component of the task itself. For example, consider a scenario in which

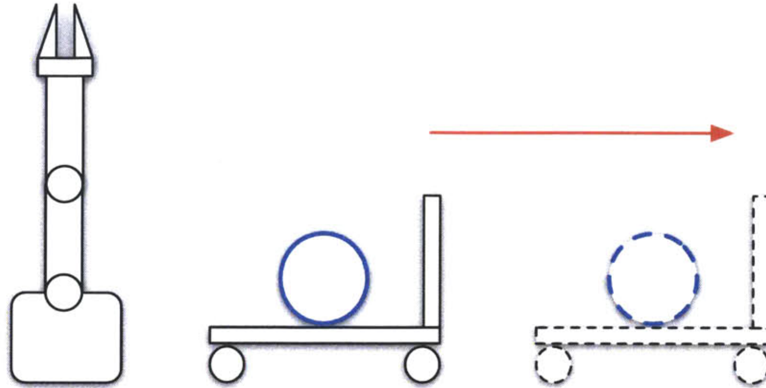


Figure 2-4: A disturbance caused by a change in the goals to be achieved by Chekhov. The solid illustration of the cart and the ball depict their current state. The solid arrow and the dashed illustration of the cart and the ball illustrate their motion. The motion of the cart and the ball represents the change change in Chekhov's goals. Chekhov's goal is to pick up the ball; consequently, when the ball is moved, it constitutes a change in the Chekhov's goals.

the robot is tasked with picking up an object from a moving conveyor belt. In this situation, the robot has to pick up an object that is constantly moving. Consequently, the goal that the robot needs to achieve changes although the task-goal remains the same. Figure 2-4 illustrates a variation of the scenario described above. The goal is for the robot to pick up the ball placed on the cart. However, this cart is moving; this motion is depicted by the solid arrow and the dashed illustration of the cart and the ball. The motion of the cart and the ball also represents the change in Chekhov's goals.

Changes to the Obstacles in the Environment

Obstacles constantly move around in real-world situations. At times these moving obstacles may not effect Chekhov's execution; at other times, however, obstacles may move into the path of robot. In Figure 2-5 the Block B_2 moves from its initial position, x_i, to a new position, x_f. As shown in this figure, in its new position the Block B_2 obstructs the path of the robotic arm. The solid arrow represents the movement caused by Chekhov's output, the actuation command set $\mathbf{u}$. This movement will

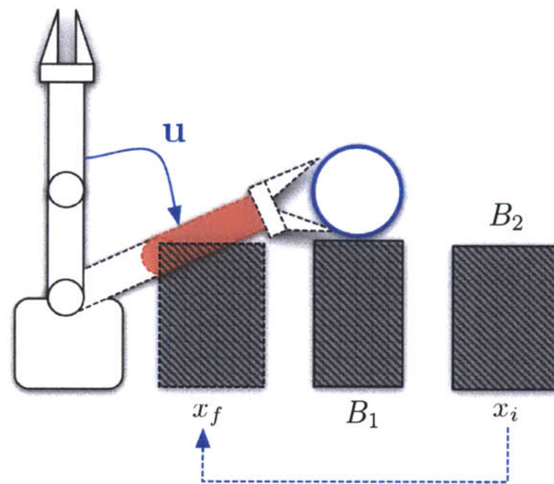


Figure 2-5: A disturbance caused by the movement of obstacles in the environment. The solid illustration of the arm represents the current state of the robot. The dashed illustration of the arm represents the final state of the robot after it has executed the set of actuation command issued to it by Chekhov. The solid arrow represents the movement caused by the actuation command set **u**. The dashed arrow represents the movement of the obstacle, B_2 from its initial position, x_i, to a new position, x_f. The new position of B_2 will obstruct the motion of the robotic arm from its initial state to its final state.

result in the collision with B_2 if it is executed completely.

We will now formally define some terms that we will use frequently in this thesis.

Definition 2.2 (Disturbance). *A* ***disturbance*** *is an unexpected change to the task goals, the environment or the state of the robot.*

Definition 2.3 (Successful Plan Execution). *A plan is said to have* ***executed successfully*** *if it achieves all its task goals.*

Definition 2.4 (Simple Plan Feasibility). *A plan is said to be* ***feasible*** *if it satisfies all constraints and if it is expected to execute successfully in the absence of any future disturbances.*

Definition 2.5 (Successful Adjustment). *When a plan becomes infeasible due to a disturbance, a* ***successful adjustment*** *is said to be have been made when a modification or adjustment to the plan achieves simple plan feasibility.*

If a plan becomes *infeasible* due to one or more disturbances, Chekhov tries to modify its plan to adapt to these disturbances and ensure that the task goals can still be achieved, if possible. If a successful adjustment cannot be made to the plan, Chekhov has to detect this early and inform the higher-level control authority that is issuing the motion goals. Thus, once again, the speed with which a successful adjustment is computed is a very important consideration.

Chekhov's Motion Executive provides updated actuation commands to the robot at each control-time increment. However, it is unreasonable to assume that all successful adjustments can be computed within the time window of a single control-time increment. In general, it is adequate if the successful adjustment is calculated in time to deal with the disturbance and if the time taken by this computation is less than the expected arrival time of the next disturbance. A detailed explanation regarding the speed of computing these adjustments is beyond the scope of this thesis.

This thesis focusses on fast performance in typical, practical situations. In particular the emphasis is on providing techniques that allow for swift plan-adjustment in response to disturbances. These techniques are essential in order to provide robots with the capability of operating in dynamic environments.

This thesis does not delve into a detailed analysis of precisely how fast plan-adjustments must be made in particular circumstances. Furthermore, we do not aspire to try and solve "piano mover" problems, that have overly complex collision environments. We assume that the tasks provided to Chekhov will be ordinary tasks in which the obstacle environment is relatively uncluttered. Additionally, we will focus on disturbances caused due to the movement of obstacles in the environment. We assume that disturbances caused by the changes in the state of the robot are minimal and that changes in the task goals to do not occur.

2.2 Formal Chekhov Problem Statement

We have now provided an intuitive description of the problem statement and discussed some of the formalisms that are necessary in order to describe the inputs and outputs of Chekhov rigorously. This section is devoted to formalising the intuitive problem statement using the insight and understanding that has been built in the previous sections.

Chekhov takes the following **inputs**:

- A model of the Environment $\mathcal{E}$ that can be expressed as a union of the collision space and free-space in the environment. $\mathcal{E} = \mathcal{C}(\mathcal{E}) \cup \mathcal{F}(\mathcal{E})$
- A Plant Model $\mathcal{P} = \langle \mathbf{q}_{min}, \mathbf{q}_{max}, \dot{\mathbf{q}}_{min}, \dot{\mathbf{q}}_{max} \rangle$
- The current state of the robot $\langle \mathbf{q}, \dot{\mathbf{q}} \rangle$
- The spatial constraints that
  - specify the goal constraints on the robot state $\mathbf{g}_{min}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}) < [\mathbf{q_{tf}}, \dot{\mathbf{q}}_{\mathbf{tf}}]^T < \mathbf{g}_{max}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t})$
  - ensure that the robot is not in collision $\neg(\mathbf{q} \cap \mathcal{C}(\mathcal{E}))$.
- The temporal constraints. $d_{min} < d < d_{max}$

- The operating constraints $\mathbf{o}_{min}, \mathbf{o}_{max}$ that specify feasible regions of operation for the robot. $\mathbf{o}_{min}, \mathbf{o}_{max} = \langle \mathbf{q}, \dot{\mathbf{q}}, \mathbf{t} \rangle$
- The acceleration limits implied by the Plant Model $\mathcal{P}$. $f_{min}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}), f_{max}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u})$
- The objective function c

where

- $\mathcal{C}(\mathcal{E})$ is the collision space of the environment $\mathcal{E}$
- $\mathcal{F}(\mathcal{E})$ is the free space in the environment $\mathcal{E}$
- $\mathbf{q}$ is the robot pose vector
- $\dot{\mathbf{q}}$ is the robot velocity vector
- $\mathbf{q}_{min}$ and $\mathbf{q}_{max}$ are the joint pose limits of the robot
- $\dot{\mathbf{q}}_{min}$ and $\dot{\mathbf{q}}_{max}$ are the velocity limits of the robot
- $\mathbf{u}$ is the set of actuation commands generated by Chekhov
- $\mathbf{q_{tf}}, \dot{\mathbf{q}}_{\mathbf{tf}}$ are the position and velocity at the robot's final state
- d is the task duration
- d_{min} and d_{max} are the bounds on the task duration
- $\mathbf{t}$ is the current time step

Chekhov generates the following **outputs**:

A set of command actuations, $\mathbf{u}$, which specifies a trajectory, $\mathbf{q}(t)$, that is optimal with respect to the objective function c and that observes all the constraints, both spatial and temporal.

$\mathbf{q}(t)$ comprises position and velocity vectors that are indexed by a discrete time index, k. Consequently, the $\mathbf{q}(t)$ is of the form:

$$\mathbf{q}(0)\ \mathbf{q}(1)\ ...\ \mathbf{q}(k)\ ...\ \mathbf{q}(n)$$

$$\dot{\mathbf{q}}(1)\ \dot{\mathbf{q}}(2)\ ...\ \dot{\mathbf{q}}(k)\ ...\ \dot{\mathbf{q}}(n)$$

The position and velocity in the trajectory are related by the following linear, equality constraint:

$$(\dot{\mathbf{q}}(k) = \mathbf{q}(k) - \mathbf{q}(k-1)) \quad \forall\, k = 1\ ....\ n$$

Hence, we can formulate the problem as follows:

$$\begin{gathered}
\mathit{minimize}\ c\left(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \mathbf{u}, \dot{\mathbf{u}}\right) \\
\mathit{such\ that}\ \neg\left(\mathbf{q} \cap \mathbf{C}(\mathbf{E})\right) \\
\mathbf{q}_{min} < \mathbf{q} < \mathbf{q}_{max}, \\
\dot{\mathbf{q}}_{min} < \dot{\mathbf{q}} < \dot{\mathbf{q}}_{max} \\
\mathbf{f}_{min}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}\right) < \ddot{\mathbf{q}} < \mathbf{f}_{max}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}\right) \\
\mathbf{o}_{min}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}\right) < \left[\mathbf{q}, \dot{\mathbf{q}}\right]^{T} < \mathbf{o}_{max}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}\right) \\
\mathbf{g}_{min}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}\right) < \left[\mathbf{q}_{\mathbf{tf}}, \dot{\mathbf{q}}_{\mathbf{tf}}\right]^{T} < \mathbf{g}_{max}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}\right) \\
(\dot{\mathbf{q}}(k) = \mathbf{q}(k) - \mathbf{q}(k-1)) \quad \forall\, k = 1\ ....\ n \\
d_{min} < d < d_{max}
\end{gathered} \tag{2.1}$$

Chapter 3

The Chekhov Executive

In Chapter 2, we specified the problem that the Chekhov Executive attempts to solve, both intuitively and formally. In Chapter 3, we describe our approach to solving this problem. We begin this chapter by providing an overview of the Chekhov Executive's system architecture. This introduces the major components of the Chekhov Executive and provides a high-level explanation of their function. It also clarifies how these components interact with one another and come together to form the Chekhov Executive. The subsequent sections will focus on each one of these major components and will delve into the details of their workings.

3.1 The System Architecture

This section provides an overview of the Chekhov Executive's system architecture. The overall system architecture is shown in Figure 3-1. Chekhov comprises two major components, the **Reactive Motion Planner** and the **Motion Executive**.

Reactive Motion Planner

The Reactive Motion Planner is the component responsible for generating a feasible plan from a given start pose to a specified goal pose. If no feasible plan can be generated Chekhov will return, signalling a failure.

The Motion Executive

The Motion Executive comprises two subcomponents, the **Execution Monitor** and the **Constraint Tightening** subcomponent.

The Motion Executive is responsible for executing the feasible plan that the Reactive Motion Planner provides to it. The Execution Monitoring subcomponent of the Motion Executive continuously monitors the execution of this feasible plan. If the Execution Monitor detects a disturbance that could prevent the plan from being executed successfully, the Constraint Tightening subcomponent is called. This subcomponent focusses on determining whether a successful adjustment can be made to the original motion plan. If this is possible, the constraint tightening component makes this adjustment and the execution continues in the Motion Executive. If however, no successful adjustment can be made, the Motion Executive returns with a failure and requests the Reactive Motion Planner for a new feasible plan.

Thus, there is substantial interaction between Chekhov's two components, however, each component has its own functionality that it performs independently of the other. In the subsequent sections we will examine each of these components in detail.

3.2 The Reactive Motion Planner

As mentioned in Section 3.1, the Reactive Motion Planner is responsible for generating a feasible plan from a given start pose to a specified goal pose. The crucial feature that distinguishes the Chekhov Executive's Reactive Motion Planner from other motion planners, is that Chekhov's Reactive Motion Planner generates a "flow tube", or a family of feasible trajectories and associated control policies, from an initial region to a goal region, rather than generating a single reference trajectory from the initial pose to the goal pose. These flow tubes provide the Chekhov Executive with a flexibility that is extremely valuable for plan execution in a dynamic setting. We emphasise the significance of this feature using an example. Consider the scenario illustrated in Figure 3-2. The figure depicts a road between two cities. In the upper portion of the figure, these two cities are connected by a single-lane road. Consequently, if there is

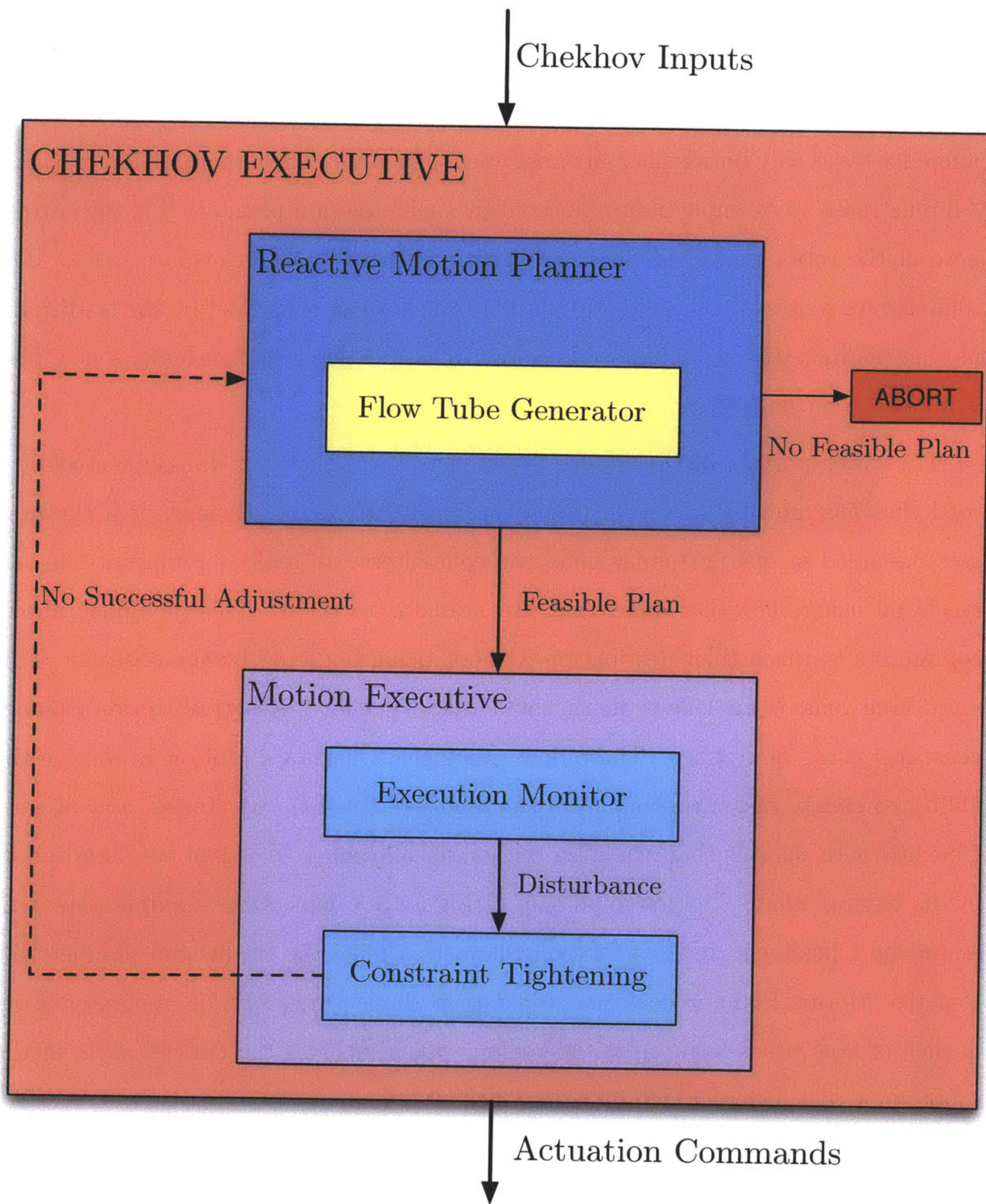


Figure 3-1: The System Architecture of the Chekhov Executive.

an accident on the road, or the road itself gets damaged, the commuters must wait until this obstacle is cleared away. This may become a long, tiresome affair, however, the commuters have no option or flexibility available to them. This single-lane road is much like the plans produced by current motion planners. They generate a single reference trajectory between a start pose and a goal pose. The upper portion of Figure 3-3 illustrates an example plan generated by such a motion planner. S is the current pose of the robot and G is Chekhov's goal. 1 and 2 are intermediate poses. The solid arrows represent the single trajectory, from S to G, generated by the traditional motion planner. If this trajectory becomes infeasible due to an obstacle, a new plan will have to be generated.

The lower portion of the Figure 3-2 depicts two cities that are connected by a road that has multiple lanes. In this scenario, if there is an accident, or if the road gets damaged in one particular lane, the commuters can easily circumvent this obstacle by using the other lanes that are available to them. This flexibility allows commuters to reach their destination without being bothered by the obstacle. This multi-lane road is analogous to the plan that Chekhov's Reactive Motion Planner generates using flow tubes. These flow tubes give Chekhov's Motion Executive the ability to choose one trajectory from the family of feasible trajectories, and provide Chekhov with options that are often critical for allowing it to adapt to disturbances in the environment. The lower portion of Figure 3-3 depicts an example plan generated by Chekhov's Reactive Motion Planner. Unlike the traditional planner, the Reactive Motion Planner does not generate a single trajectory; it generates a sequence of flow tubes from an initial region, which includes the current state of the robot, to a goal region, which includes Chekhov's goal.

Chekhov generates the feasible plan from the start pose to the goal pose using an incremental, search-based planning algorithm called D* Lite [6]. The key feature of the D* Lite algorithm is its ability to quickly re-plan when it is faced with environmental changes. This feature of the D* Lite algorithm makes it an ideal choice for Chekhov. The drawback of this algorithm is that it relies on a discretisation of the state space, which can lead to computational tractability issues when Chekhov has

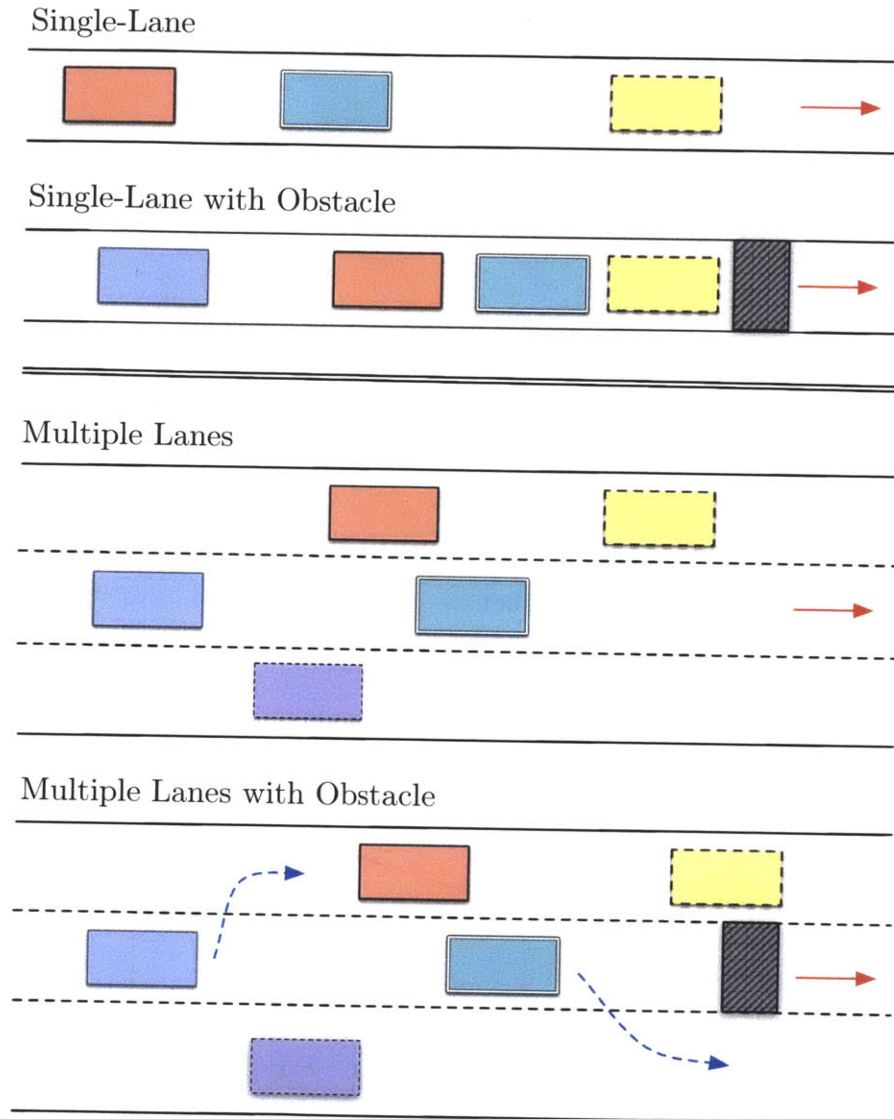
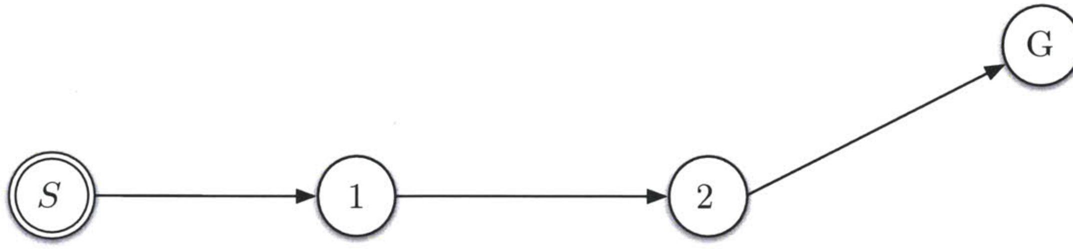


Figure 3-2: Cars travelling along a road from City A to City B. The filled rectangles in the figure illustrate the cars on the road. The shaded rectangles depict an obstacle on the road. The solid arrows depict the direction in which the cars are moving on the road. The dashed arrows depict the movement of the cars as they change lanes in order to circumvent the obstacle. The upper portion depicts two cities that are connected by a single-lane road. The lower portion illustrates two cities that are connected by a multi-lane road. The dashed lines demarcate different lanes in the multi-lane road.

Traditional Motion Planner



Chekhov's Reactive Motion Planner

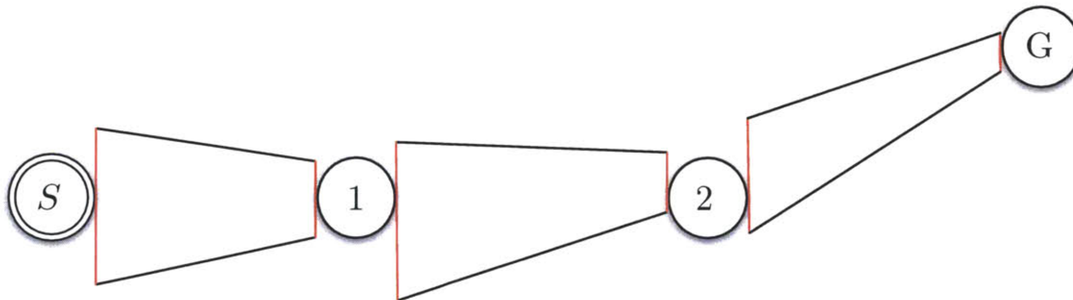


Figure 3-3: Comparison of a plan generated by a traditional motion planner and one generated by Chekhov's Reactive Motion Planner. S represents the current pose of the robot and G depicts Chekhov's goal. 1 and 2 are intermediate poses. In the upper portion, the solid arrows represent the single trajectory, from S to G, generated by the traditional motion planner. The lower portion illustrates the plan generated by Chekhov's Reactive Motion Planner. It generates a sequence of flow tubes from an initial region, which includes the current state of the robot, to a goal region, which includes Chekhov's goal.

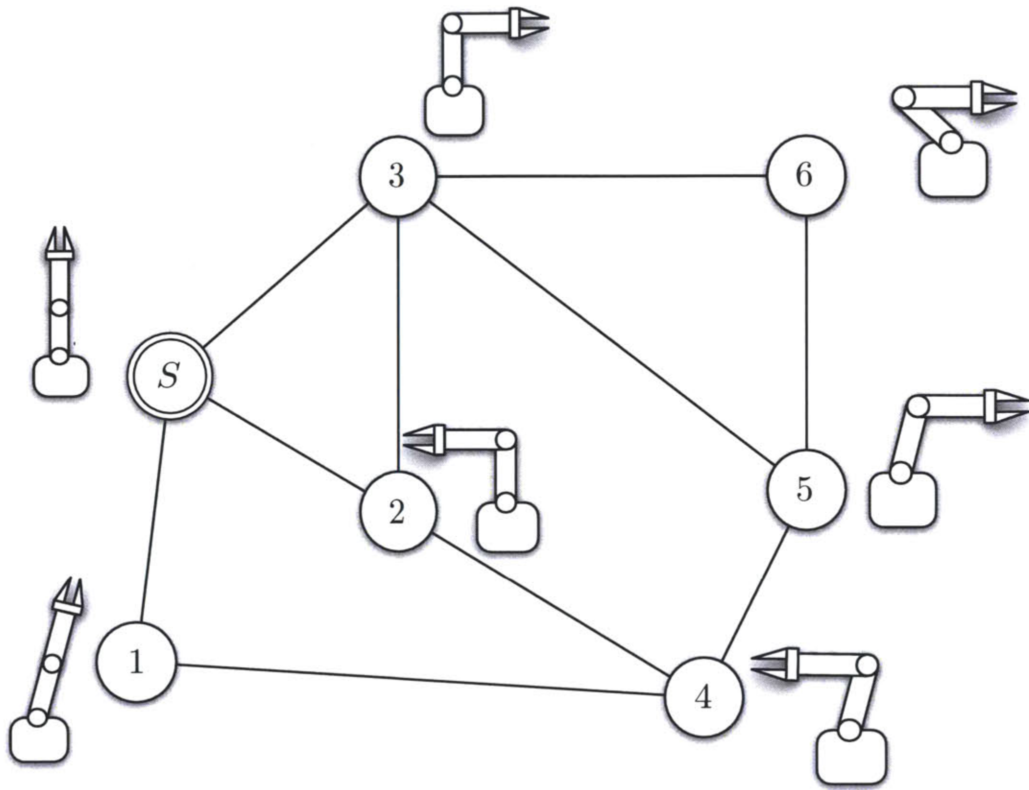
to deal with higher-dimensional state-spaces. However, the advantages of D* Lite far outweigh its shortcomings and we address this dimensionality problem by using a relatively coarse discretisation combined with a decoupling approach and a post-processing step that fine tunes the solution. The result is a comprehensive control policy for the entire manipulator workspace.

3.2.1 Graphical Representation of the Search Space

Chekhov's Reactive Motion Planner creates a graph of the search space with the discretised states as the vertices of the graph. Hence, each **state** on the graph represents a *specific, unique robot configuration.* The **edges** in the graph connect two states that can transition *directly* between one another. The edges of the graph are then augmented with **flow tube** information. These flow tubes represent the relationship between the dynamic limits of the robot and the feasible range of temporal duration for transitioning along the edge. We will describe flow tubes in detail in the next section.

Definition 3.1 (Neighbouring States). *Two states in the search-space graph are said to be **neighbouring** if there is an edge between them.*

The upper portion of Figure 3-4 shows a pictorial representation of the search-space graph that is generated by by the D* Lite Algorithm. In this figure, S denotes the starting state of the robot, while the numbers 1 through 6 represent the other states in the graph. In this example, we assume that the joint movements of the robot are decoupled. In particular, we assume that the robot can only move one joint at a given time. Consequently, states are only *neighbouring* if the robot can transition between the states by moving a single joint. For instance, $(S, 1)$ and $(S, 2)$ are examples of neighbouring states as we can transition between the states by moving a single joint. However, $(3, 4)$ are not neighbouring as the robot would need to move two joints simultaneously in order to transition between the two states directly. With the decoupling assumption the shortest path for the robot from state 3 to state 4 is by either going through state 2 or by going through state 5. The lower portion of



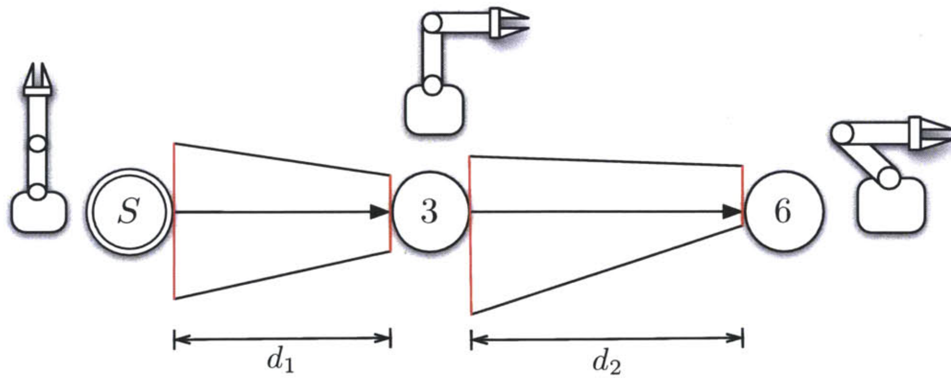


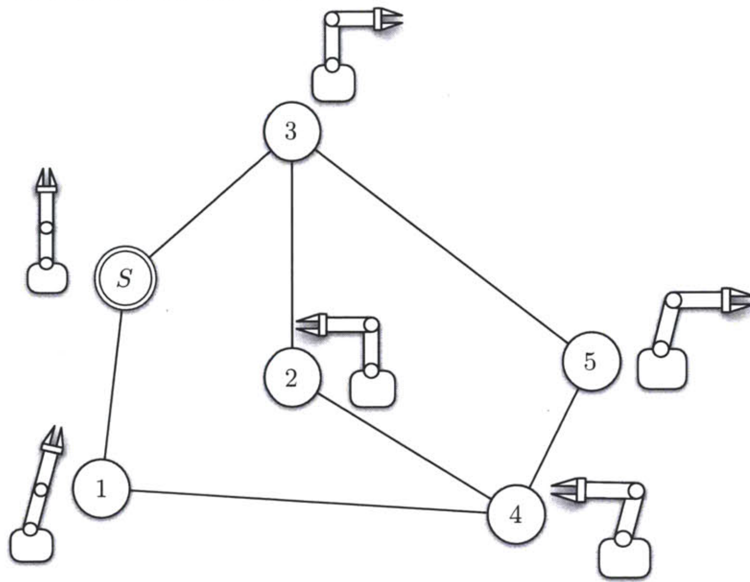
Figure 3-4: The upper portion illustrates the search-space graph generated by the D* Lite algorithm. S, is the starting state of the robot. The numbers, 1 through 6 are the other states or robot configurations depicted in this search-space graph. The robot configuration that is illustrated next to the each state, is the specific configuration associated with that state. The lower portion focusses on states S, 3 and 6, and the edges between them. It illustrates the flow tubes that are used to augment each edge in the search-space graph. The two flow tubes depicted have associated durations of d_1 and d_2. The shape of the flow tube is determined by the dynamics of the robot.

the figure focusses on states S, 3 and 6, and the edges between them. It illustrates the flow tubes that are used to augment each edge in the search-space graph. The two flow tubes depicted have associated durations of d_1 and d_2, which impose a temporal constraint on each transition. The shape of the flow tube is determined by the dynamics of the robot.

Figure 3-5 shows the state-space graph generated by a traditional motion planner and the state-space graph generated by Chekhov's Reactive Motion Planner. The state-space graph generated by the Reactive Motion Planner has its edges augmented with flow tubes, as mentioned previously.

3.2.2 Flow Tubes

As discussed previously in Chapter 2, this thesis deals with plan generation and plan execution in a dynamic environment. When dealing with an environment that is constantly changing, it is advantageous for the Motion Executive to be able to adapt to disturbances by choosing from a family of trajectories, rather than trying to follow a single, reference trajectory. The Reactive Motion Planner in Chekhov achieves this by generating flow tubes [3], that represent families of feasible trajectories and associated control policies from an initial region to a goal region. This explicit representation of multiple feasible paths provides a flexibility that is essential for motion execution in uncertain environments. A single-trajectory plan may become infeasible when the environment changes. Consequently, the execution will have to be stopped while the planner generates a new plan to account for the disturbance in the environment. This is analogous to the scenario depicted in the upper potion of Figure 3-2 (the single-lane road connecting the two cities). Flow tubes provide a family of feasible trajectories from the start state to the goal state that we can choose from during execution. If one, or even a large number, of the feasible trajectories in the flow tube become infeasible due to a disturbance, execution can continue by following one of those trajectories in the flow tube that still remains feasible (an analogous scenario is illustarted in the lower potion of Figure 3-2). The plan will be abandoned and the planner forced to re-plan, only if all the trajectories in the flow tube become infeasible.



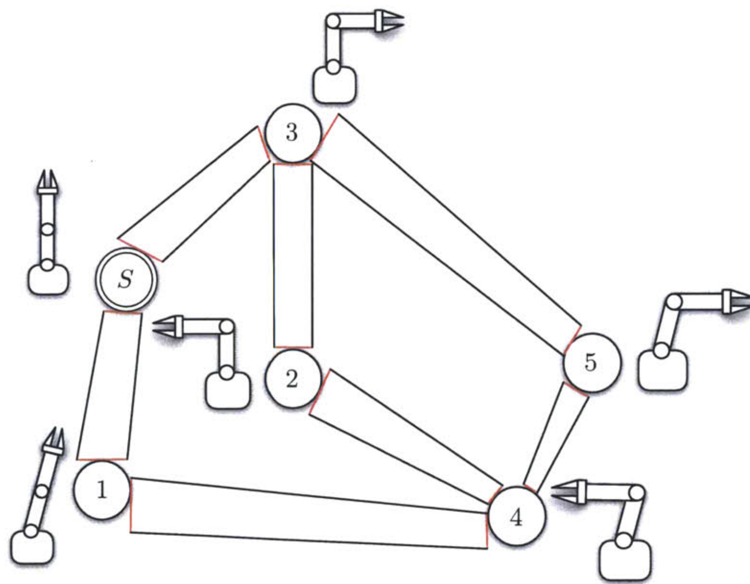


Figure 3-5: Comparison of the state-space graph generated by a traditional motion planner and one generated by Chekhov's Reactive Motion Planner. S, is the starting state of the robot. The numbers, 1 through 5 are the other states or robot configurations depicted in this search-space graph. The robot configuration that is illustrated next to the each state, is the specific configuration associated with that state. The upper portion depicts the state-space graph generated by a traditional motion planner. The poses are connected by a single trajectory. The lower portion illustrates the state-space graph generated by Chekhov's Reactive Motion Planner. The Reactive Motion Planner generates flow tubes, which are used to connect different poses.

Flow Tube Representation

In this section we will focus on providing an intuitive understanding of flow tubes and a description of our flow tube representation. In particular, we will lay emphasis on certain aspects of the flow tube representation that are essential in order to ensure that Chekhov can function effectively.

Flow Tubes Must Only Include Feasible Trajectories

In many scenarios the set of feasible trajectories may have a very complex geometry, which makes it impractical to define a flow tube that represents this complex geometry exactly. Hence, any tractable representation must be an *approximation* of the feasible trajectory set.

There are two approaches for such an approximation [1, 7, 8] :

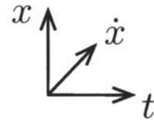
1. Internal (Under) Approximation.

2. External (Over) Approximation.

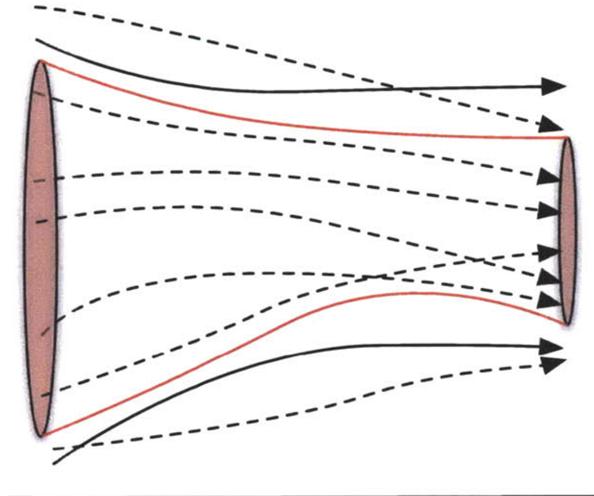
Internal Approximation: This is a more conservative approximation. It includes *only* feasible trajectories and excludes all infeasible trajectories. However, this representation may also exclude some feasible trajectories, and hence is *incomplete.*

External Approximation: This approximation includes *all* feasible trajectories, however, it may also include some infeasible trajectories.

In Figure 3-6 the dashed arrows depict the feasible trajectories and the solid arrows represent the infeasible trajectories. The solid lines depict the boundaries of the flow tube and the shaded ovals illustrate the flow tube cross-sections. The oval on the left represents the initial cross-section of the flow tube while the oval on the right illustrates the goal cross-section of the flow tube. The upper portion of the figure illustrates the internal flow tube approximation. This includes only feasible trajectories, as depicted in the figure; however, some feasible trajectories are excluded



Internal Approximation



External Approximation

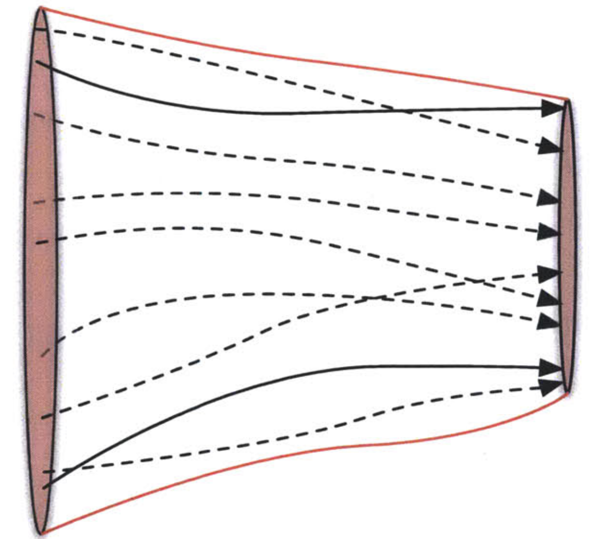


Figure 3-6: The internal and external flow tube approximation. The dashed arrows depict the feasible trajectories and the solid arrows represent the infeasible trajectories. The solid lines represent the boundaries of the flow tubes and the shaded ovals represent flow tube cross-sections. The oval on the left depicts the initial cross-section of the flow tube, while the oval on the right depicts the goal cross-section of the flow tube. The upper portion illustrates the internal flow tube approximation and the lower portion illustrates the external flow tube approximation.

by this representation. The lower portion of the figure depicts the external flow tube approximation. This approximation includes all the feasible trajectories. However, as shown in Figure 3-6, it also includes some of the infeasible trajectories.

It is important to note that the cross-sections of a flow tube can be defined over any n-dimensional state space. This thesis however, focusses on a two-dimensional flow tube representation, where one axis is a joint position and the other is time.

Approximation used in Chekhov: Chekhov requires a flow tube representation that will allow quick plan execution and fast re-planning, if necessary. The Motion Executive will only execute a feasible trajectory. Consequently, if Chekhov used the *external* flow tube approximation, each trajectory in the flow tube would need to be checked to ensure that it is feasible, before being approved for execution.This would dramatically slow down the execution time. Therefore, we will use the *internal* flow tube approximation that includes *only* feasible flow tubes. This guarantees that any trajectory within the flow tube can be executed successfully given that there is no change in the environment.

The major drawback of the internal representation is that it is incomplete as it may exclude some valid trajectories. Hence, we must choose an internal representation that maximises the number of feasible trajectories that it includes. In particular, the flow tube representation must not be overly conservative. It is important to reiterate that traditional planners have no notion of flow tubes and are restricted to following a single trajectory, while Chekhov can follow any trajectory within the flow tube.

Structure of a Flow Tube

In this section we will provide an overview of the structure of a flow tube. A flow tube is a function of the duration $\mathcal{D}$, the goal region R_g, and Plant Model $\mathcal{P}$, which represents the actuation limits of the robot under consideration.

Figure 3-7 shows a flow tube. The region R_i is the set of feasible trajectory states at time 0 and is called the *initial cross-section* of the flow tube, because it is the cross-section of the flow tube in the position plane at time 0. Any trajectory beginning in

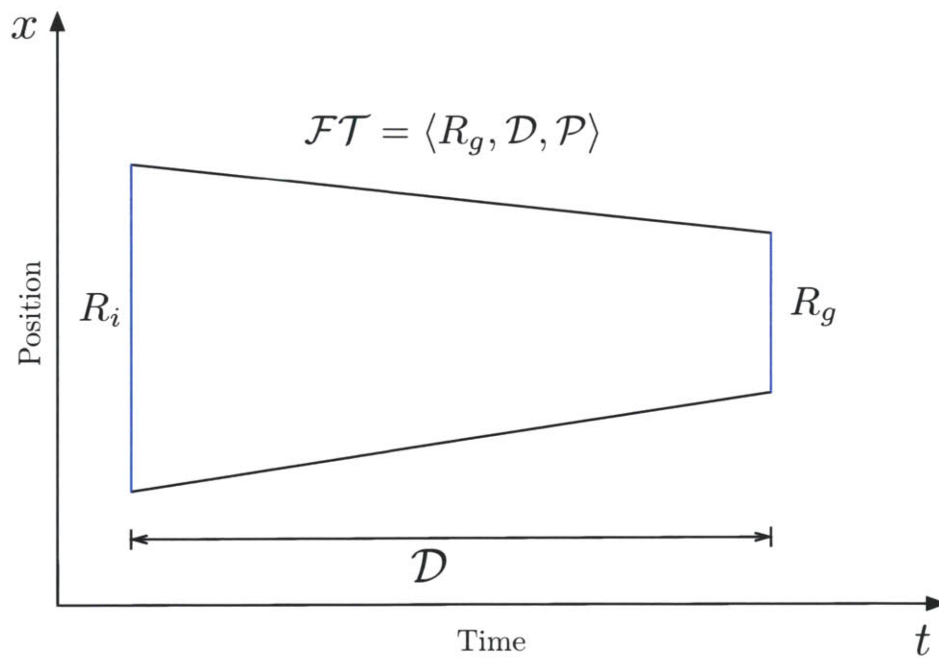


Figure 3-7: The region R_i represents the *initial cross-section* of the flow tube. The region R_g is called the *goal cross-section* of the flow tube and is a duration, $\mathcal{D}$, away from the initial cross-section.

this region is guaranteed to reach the region, R_g, after the duration $\mathcal{D}$. This region R_g is called the *goal cross-section* of the flow tube and it is the cross-section of the flow tube in the position plane at time $\mathcal{D}$.

The flow tube *must* represent each of the following explicitly

- The Goal Region R_g.
- The Duration $\mathcal{D}$.
- The Velocity Limits that are represented by the Plant Model $\mathcal{P}$.

The Flow Tube Must Represent the Goal Region Explicitly

The goal region, R_g, is a cross section in the position plane at time $\mathcal{D}$ after the initial region, R_i. It is essential that the flow tube representation include an explicit description of the goal region. The goal region, R_g, makes it possible to calculate any other cross-section of the flow tube.

The Flow Tube Must Represent the Duration Explicitly

A Flow Tube defines the family of feasible trajectories from the initial region to the goal region of the flow tube. A trajectory starting at the initial region in the flow tube is guaranteed to successfully reach the goal region after duration, $\mathcal{D}$. This duration serves as a means to determine whether the family of trajectories defined by the flow tube satisfies a goal's temporal constraint. Therefore, it is imperative that the flow tube represents the duration explicitly.

The Flow Tube Must Represent the Velocity Limits Explicitly

A Flow Tube guarantees that a trajectory starting at the initial cross-section of the flow tube will successfully reach the goal cross-section in a given time, $\mathcal{D}$. Therefore, while being generated, a flow tube must account for the velocity limits of the joints of the robot. If this is not considered during flow tube generation, the trajectories in the flow tube may not be feasible as they may require the joints to move at velocities

that exceed their maximum limit. For example, consider a scenario in which we wish to move a robot from a given pose A, to a goal pose B in a duration d. Going from Pose A to Pose B, requires the robot to move one of its joints $\pi/2$ rad clockwise. This joint has a maximum velocity, v_{max} and consequently, it is not possible for the robot to move from Pose A to Pose B in a duration d as this would require the robot to exceed this velocity limit. As a result, no flow tube with a duration d, can exist between the two poses. If the velocity limits were ignored, the flow tube would have been generated. If a trajectory in the flow tube was executed, it could cause the robot joint to exceed its nominal velocity limit, and this could damage the robot.

Formal Representation of a Flow Tube

Now that we have provided an intuitive understanding of flow tubes, we can formalise the flow tube representation using the insights that we have gained previously.

A Flow Tube can be defined as follows:

Definition 3.2. $\mathcal{FT} = \langle R_g, \mathcal{D}, \mathcal{P} \rangle$

- R_g: *Goal Cross-Section of the Flow Tube*
- $\mathcal{D}$: *Duration between the Initial Cross-Section and the Goal Cross-Section*
- $\mathcal{P}$: *Plant Model Representing the Actuation Limits of the Robot*

A flow tube comprises a set of cross-sections. Furthermore, the region of state-space between the goal region, R_g and each cross-section can be defined as a function of R_g, d_i and $\mathcal{P}$, which together imply that this region is a flow tube as well; where d_i is the time required to move from cross-section i to the goal region.

This implies that any trajectory starting at any cross-section in a flow tube is guaranteed to reach the goal cross-section in the specified duration. In Figure 3-8, the region of the state space between the goal region, R_g and the first cross section, $\mathcal{CS}_1$, defines a flow tube with goal cross-section, R_g, plant model $\mathcal{P}$ and duration d_1. All trajectories starting at $\mathcal{CS}_1$, are guaranteed to reach R_g in d_1 time units.

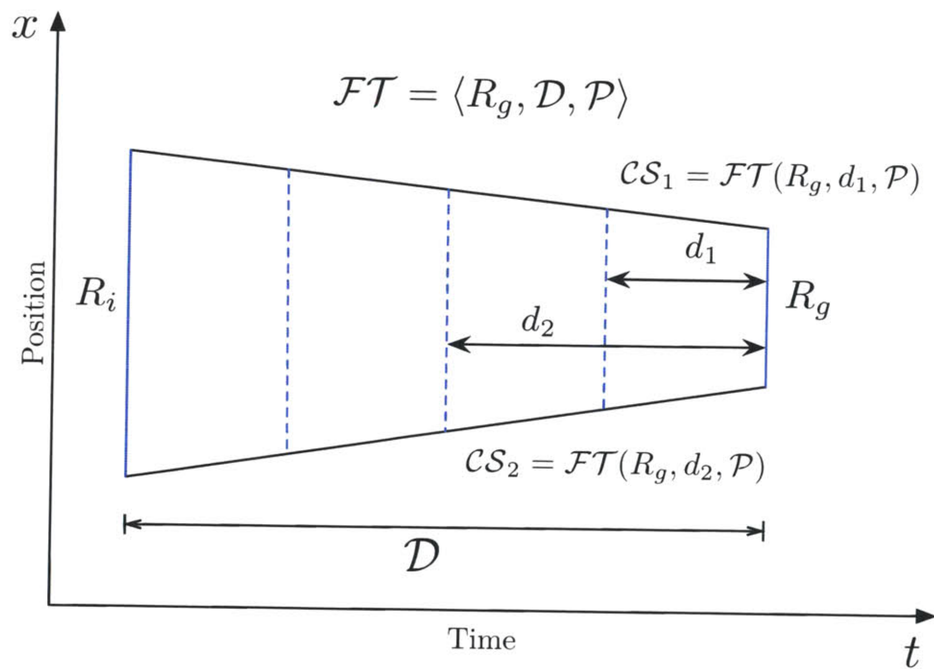


Figure 3-8: A Flow tube comprises a set of cross-sections. The region of state-space between the goal region, R_g and each cross-section can be defined in terms of R_g, d_i and $\mathcal{P}$, which together imply that this region is a flow tube a well. The region of the state space between the goal region, R_g and the first cross section, $\mathcal{CS}_1$, defines a flow tube with goal cross-section, R_g, plant model $\mathcal{P}$ and duration d_1.

Flowtube Computations

In the implementation of the concepts that we will discuss in this thesis we have focussed primarily on two-dimensional, velocity-limited flow tubes. These flow tubes are computed using the following equations:

$$x_{max}[\mathcal{CS}(d_i)] = x_{max}[R_g] - d_i\dot{\mathbf{q}}_{min}$$
$$x_{min}[\mathcal{CS}(d_i)] = x_{min}[R_g] - d_i\dot{\mathbf{q}}_{max}$$

- $x_{max}[\mathcal{CS}]$ is the maximum position in the cross-section $\mathcal{CS}$
- $x_{min}[\mathcal{CS}]$ is the minimum position in the cross-section $\mathcal{CS}$
- $\dot{\mathbf{q}}_{min}$ and $\dot{\mathbf{q}}_{max}$ are the velocity limits of the robot
- d_i is the duration of the ith cross section

The flow tubes are computed by *reaching back from the goal cross-section.* This implies that the first cross-section of the flow tube which is computed is the cross-section that is closest to the goal cross-section and hence, farthest from the initial cross-section. Each cross-section is a fixed time-step, Δ, from the cross-section that was computed before it. More specifically, the cross-sections in a flow tube are equidistant from one another in the temporal plane. Hence, the first cross-section, $\mathcal{CS}_1$, will have a duration of Δ, the second cross-section $\mathcal{CS}_2$, a duration of 2Δ and so on. Figure 3-9 demonstrates this procedure. The dashed arrow labelled 1, reaches back from the goal cross-section to the cross-section that was computed first. This cross-section has a duration Δ. Similarly the dashed arrow labelled 2, reaches back to the cross-section computed second. This cross-section has a duration 2Δ, and hence is a time step Δ from the first cross-section.

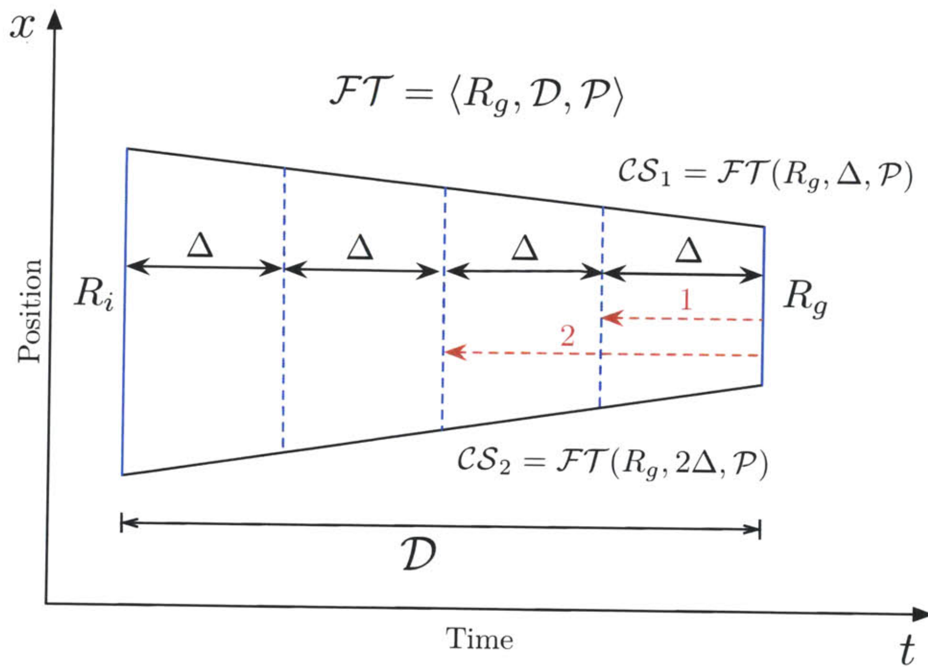


Figure 3-9: The flow tube is computed by reaching back from the goal cross-section. The red arrow labelled 1 demonstrates reaching back from the goal cross-section to the first cross-section. This cross-section will have a duration of Δ. as each cross-section in the flow tube is a fixed time-step, Δ, away from the previous cross-section in the flow tube.

3.2.3 Fast Incremental Plan Adjustment

In this section we provide a more detailed explanation of the algorithms that are used by Chekhov's Reactive Motion Planner. In the previous sections, we began to explore the workings of the Reactive Motion Planner and we provided a detailed explanation of how the Reactive Motion Planner uses the D* Lite algorithm to create a search-space, state graph. The edges of this graph are then augmented with flow tubes, which we have also examined in the previous section.

In this section we delve deeper and describe two algorithms, the **Basic Algorithm** and the **Enhanced Algorithm**, that are fundamental to the Reactive Motion Planner. Our aim in this section is to first illustrate the workings of each algorithm at a high, bird's-eye level and to then analyse each of these algorithms in detail. We begin with the Basic Algorithm and then move on to the Enhanced Algorithm.

Before we begin our scrutiny of these algorithms, however, we describe, more formally, the plan that is generated by the Chekhov's Reactive Motion Planner, using the intuition that we have built thus far.

Definition 3.3 (Plan). *The* ***plan*** *generated by Chekhov's Reactive Motion Planner comprises a sequence of flow tubes, such that, the initial region of the first flow tube in the sequence includes the current robot pose, the goal region of the last flow tube in the sequence includes Chekhov's goal , and for every flow tube in the sequence, each flow tube's goal region is a subset of (lies within) the initial region of the succeeding flow tube.*

The last condition ensures that when Chekhov's Motion Executive begins executing a trajectory in the first flow tube, the trajectory is not only guaranteed to reach the goal region of the first flow tube, but this trajectory will have a feasible continuation in the succeeding flow tube, and in every flow tube included in the plan sequence, until the goal is achieved. Figure 3-10 illustrates a plan generated by the Reactive Motion Planner. In this figure, S denotes the current state of the robot, and G denotes Chekhov's goal. 1 and 2 represent two other intermediate states. The plan that Chekhov generates comprises three flow tubes; d_1, d_2 and d_3 are the durations

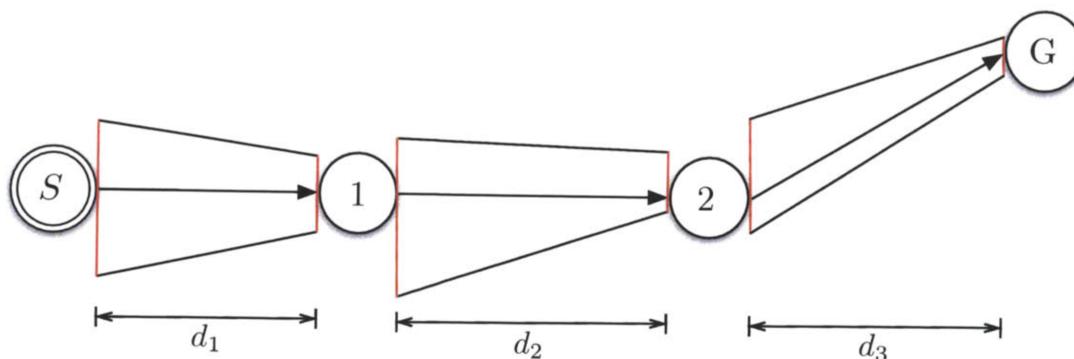


Figure 3-10: Plan generated by the Reactive Motion Planner. S denotes the current state of the robot, and G denotes the goal that Chekhov seeks to achieve. 1 and 2 represent two other intermediate states. The solid arrow depicts the trajectory chosen from the flow tubes; d_1, d_2 and d_3 are the durations associated with the flow tubes.

associated with these flow tubes. The current state, S, of the robot is within the initial region of the first flow tube and the goal, G, is within the goal region of the final flow tube. Furthermore, the goal regions of the first and second flow tube, are a subset of (lie within) the initial regions of the second and third flow tube respectively. This allows the trajectory, depicted by the solid arrow, to be executed smoothly.

The Basic Algorithm

We will begin by providing an overview of the Basic Algorithm. The algorithm takes three inputs, the task goals, the search-space state graph and the current state of the robot. It then searches through this state graph and tries to find a plan that will achieve the task goals it has received. If the algorithm cannot generate a feasible plan, it returns with a failure. If, however, the algorithm can generate a feasible plan, it begins to execute this plan. The algorithm continuously monitors the plan while it is being executed. If it detects a disturbance that is significant enough to prevent the plan from being successfully executed, it stops the execution and updates the state graph, until the state graph is consistent once again.

Definition 3.4 (Consistency in the state-space graph). *The* ***consistency*** *of the state-space graph is heavily influenced by the D* Lite algorithm. When a disturbance*

in the environment causes an edge in the state-space graph to go from being feasible to being infeasible or vice versa, the D Lite costs associated with the state-space graph no longer reflect the current state of the world. At this time, the state-space graph is said to be* ***inconsistent****.*

Definition 3.5 (Updating the state-space graph). *An update of the state-space graph involves running one step of the D* Lite Algorithm, so as to update the D* Lite costs associated with the state-space graph.*

This newly updated, consistent state-graph reflects the changes in the environment caused by the disturbance. It then searches through the updated state graph for a feasible plan. If it finds a plan that can achieve the task goals, it begins to execute this plan and the aforementioned process is repeated. This continues until the task is successfully executed. If, at any point, the algorithm is unable to find a feasible path through the state graph, it will return with a failure. The pseudo-code for this algorithm is shown in Algorithm 1.

Let us now scrutinise this algorithm further by providing a more detailed analysis. The Basic Algorithm begins with a call to the `DstarLite` function [Line 2] [6]. The `DstarLite` function updates the state graph data structure by propagating costs and generates a *single-source, shortest path tree.* When the D* Lite algorithm is run for the first time, it is functionally identical to an A* search. It has a worst-case computation time of $O(n)$, where n is the number of states in the graph. As we have mentioned previously the key feature of the D* Lite algorithm, and its major advantage over the A* algorithm lies in its quick replanning ability. Consider the situation in which an edge in the state graph that is included in the plan, becomes infeasible due to a disturbance, after plan execution has begun. This makes the current plan infeasible and a new plan must be generated, if one exists; a plan which does not rely on the infeasible edge. The D* Lite algorithm iteratively updates the state graph data structure until the graph is consistent, once again. After this, the single-source shortest path tree is updated. The `QueryPath` function [Line 3] can then be used to find a new path to the goal. This kind of plan adjustment, for a single

infeasible edge, that relies on the state graph being consistent, still has a worst-case computation of $O(n)$. Typically, however, this kind of adjustment can be computed at least an order of magnitude faster than the initial search.

The `QueryPath` function uses the shortest-path tree information from the consistent state graph generated by D* Lite to quickly compute the optimal path from the initial state to the goal state. The computation time for the `QueryPath` function is $O(log\, n)$ as it only involves traversing the shortest-path tree from the leaves to the root. If `QueryPath` can find a feasible path, it returns this path and `ExecutePath` [Line 6] is called in order to begin plan execution. If a feasible path cannot be found the function returns with a failure [Line 9]. Furthermore, while the plan is being executed the `WaitForDisturbance` function [Line 7] continuously monitors the execution, in parallel, and determines whether a disturbance is severe enough to prevent the successful execution of the plan, as this would require a new plan to be generated.

Algorithm 1: `Basic Planning Algorithm`

Input: Task Goals, Search-Space State Graph, Current Robot State
Output: If feasible path found, actuation commands

```
1 while task execution incomplete do
2     DstarLite()
3     QueryPath()
4     if path feasible then
5         In parallel:
6             ExecutePath()
7             WaitForDisturbance()
8     else
9         Error:"No feasible path found"
```

The Enhanced Algorithm

The Enhanced Algorithm is an extension of the Basic Algorithm. The key, driving insight behind the Enhanced Algorithm is the fact that the `QueryPath` function in the Basic Algorithm is much faster (much shorter execution time) than the `DstarLite` function. More specifically, it takes significantly less time to find a feasible path

using the shortest-path tree than it does to iteratively update the state graph until it is consistent. We will begin by providing an overview of the Enhanced Algorithm. The Enhanced Algorithm takes three inputs, the task goals, the search-space state graph and the current state of the robot. It then searches through this state graph and tries to find a path that will achieve the task goals that it has received. If the algorithm cannot generate a feasible plan, it returns with a failure. If a feasible plan is generated, the algorithm begins to execute the plan. The algorithm monitors the plan continuously while it is being executed. If it detects a disturbance that is severe enough to prevent the successful execution of the plan, it stops the execution and determines which edge in the state graph has become infeasible. The algorithm then updates the graph to reflect these in-feasibilities. It does *not* update the entire state graph until it is consistent. Instead, it searches through the updated, inconsistent state graph for a feasible path to the goal. If it does not find a feasible path, it runs a single step of the D* Lite algorithm and searches for a path in the newly-updated, possibly inconsistent graph, once again. This process continues either until a feasible path is found, or until the state graph becomes consistent.

If no feasible path is found once the state graph has been updated until it is consistent, the algorithm returns with a failure. If, however, a feasible path is found, the algorithm begins plan execution and continuously monitors the environment, in parallel, for any new disturbances. As the plan being executed may have been generated using an inconsistent state graph, the plan may be sub-optimal; consequently, the algorithm will use all available resources to perform execution monitoring and to improve the optimality of the sections of the path that have not yet been executed. The pseudo-code for this algorithm is shown in Algorithm 2.

Let us now explore this algorithm further by analysing it in detail. Like the Basic Algorithm, the Enhanced Algorithm begins with a call to the `DstarLite` function [Line 1]. This creates the search-space state graph data structure, which is used to generate the single-source shortest path tree. We have described both these data structures in detail in our analysis of the Basic Algorithm. The Enhanced Algorithm uses the `QueryPath` function [Line 2] to find a feasible path to the goal. This function

uses the shortest-path information from the state graph generated by the D* Lite algorithm to quickly compute an optimal path from the start state to the goal state. If `QueryPath` is unable to find a feasible path it returns with a failure [Line 4]. If a feasible path is generated, `QueryPath` returns this path and the `ExecutePath` function [Line 13] is called in order to begin plan execution. Furthermore, while the plan is being executed the `WaitForDisturbance` function [Line 15] continuously monitors the execution. If it detects a disturbance that could prevent the successful execution of the plan, it stops the execution and uses the `MarkInfeasibleEdges` function [Line 7] to determine which edge or edges of the state graph have become infeasible. This function then updates the state graph so that these edge or edges will not be used in the next path search. The `MarkInfeasibleEdges` function does *not* update the state graph until it is consistent. Its focus is to mark edges in the graph as infeasible by assigning these infeasible edges extremely high costs. The algorithm then calls the `TryQueryPath` function [Line 9] to search for a path to the goal in the updated, inconsistent graph. If `TryQueryPath` is unable to find a feasible path the `DstarLiteOneStep` function [Line 10] is called. As the name suggests, this functions runs only a single step of the iterative D* Lite algorithm. After calling this function, the `TryQueryPath` function is called on the newly-updated, possible inconsistent, graph. This continues either until a feasible path is found or until the D* Lite algorithm has run its final step and the state graph is consistent.

If no feasible path is found in the consistent state graph, the algorithm will return with an error [Line 17]. If a feasible path is found, the algorithm will begin execution once again by calling the `ExecutePath` function. As this path may have been generated using an inconsistent state graph, the path may not be optimal; consequently, the algorithm uses all available computing resources not only to constantly monitor plan execution but to also continuously call the `AnytimeImprovePath` function [Line 14] function. This function tries to improve the optimality of sections of the plan that have not yet been executed, by iteratively calling the `DstarLiteOneStep` function, followed by a call to the `TryQueryPath` function. Furthermore, the `AnytimeImprovePath` function checks that the new, improved path only makes changes to sections of the

original plan that have *not* yet been executed.

Algorithm 2: `Enhanced Planning Algorithm`

```
  Input: Task Goals, Search-Space State Graph, Current Robot State
  Output: If feasible path found, actuation commands
1  DstarLite()
2  QueryPath()
3  if path infeasible then
4      Error:"No feasible path found"
5  while task execution incomplete do
6      if disturbance detected then
7          MarkInfeasibleEdges()
8          while feasible path not found ∧ DstarLite not finished do
9              TryQueryPath()
10             DstarLiteOneStep()
11     if path feasible then
12         In parallel:
13           ExecutePath()
14           AnytimeImprovePath()
15           WaitForDisturbance()
16     else
17         Error:"No feasible path found"
```

3.2.4 Summary

In this section we have provided a detailed description of Chekhov's Reactive Motion Planner. We began by analysing the manner in which the Reactive Motion Planner uses the D* Lite algorithm to create a state-space graph. Next, we introduced the concept of flow tubes, a family of feasible trajectories and associated control policies. The edges of the state-space graph were augmented with these flow tubes in order to represent the dynamics limits of the robot and the feasible range of temporal durations for transitioning along the edge. Finally, we explored in great detail Chekhov's algorithms for fast incremental plan adjustment. We presented two algorithms, the Basic Algorithm and the Enhanced Algorithm, both of which find the optimal path

from the start pose to the goal pose, if such a path exists. Both these algorithms also allow Chekhov to quickly modify the existing plan to adjust to disturbances.

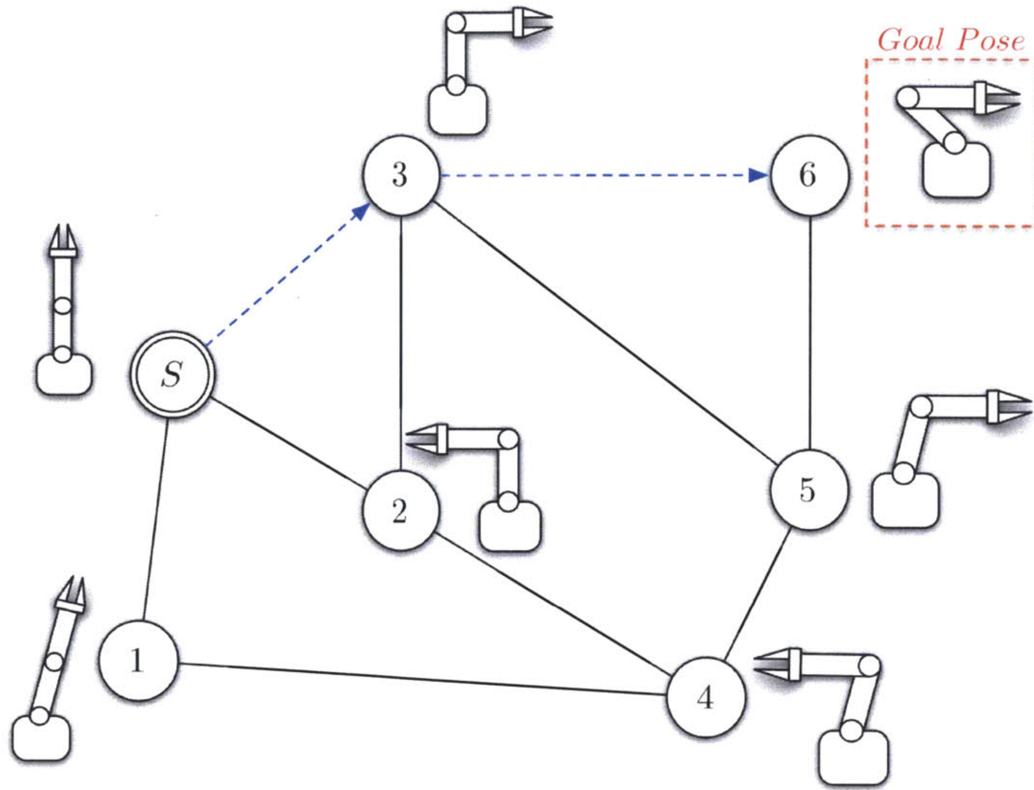
Figure 3-11 summarises the salient features of the Reactive Motion Planner pictorially. The upper portion of Figure 3-11 depicts the state-space graph generated by the D* Lite algorithm. State 6 is the goal state that Chekhov wishes to achieve. The robot configuration that is associated with state 6 is enclosed in a dashed rectangle. The dashed arrows in the state-space graph illustrate the optimal path, generated by Reactive Motion Planner's algoirthms, from the start pose to the goal pose. the bottom portion of Figure 3-11 focusses on this optimal path. This portion also depicts the flow tubes that are used to augment each edge in the state-space graph. The temporal constraint that is associated with the task, $\mathcal{D}$, is represented by the solid arrow at the very bottom of the figure, while d_1 and d_2 represent the durations associated with the first and second flow tube respectively.

3.3 The Motion Executive

As mentioned in Section 3.1, the Motion Executive is responsible for executing the feasible motion plan that is generated by the Reactive Motion Planner.
The Motion Executive accepts the following **inputs**:

- The feasible motion plan generated by the Reactive Motion Planner.
- The model of the robot.
- The state estimates of the robot state and of the environment state that are obtained from the sensing system. These estimates are continuously updated to reflect the most recent sensing information.

The Motion Executive generates the following **output**:
A set of control commands to the robot which specifies a trajectory that satisfies all the constraints defined in Equation 2.1. The output of the Motion Executive is also the output of the Chekhov Executive.



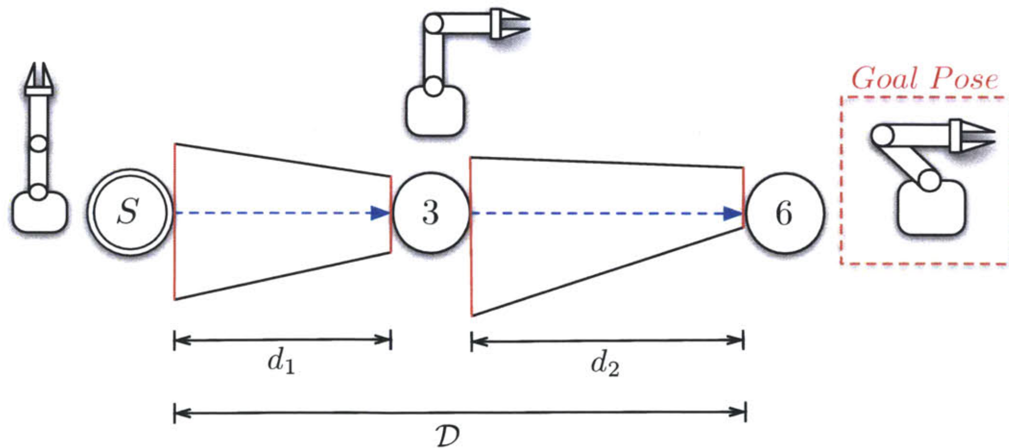


Figure 3-11: Salient Features of the Reactive Motion Planner. The upper portion depicts the state-space graph generated by D* Lite. State 6 is the Goal Pose. The robot configuration associated with this state is enclosed within a dashed rectangle. The dashed arrows in the state-space graph illustrate the optimal path, generated by the Reactive Motion Planner, from the start pose to the goal pose. The bottom portion focusses on this optimal path. It also shows the flow tubes that are used to augment each edge in the state-space graph, and the temporal constraints associated with the task.

The Motion Executive comprises two components, the **Execution Monitor** and the **Constraint Tightening** Component. The Execution Monitor executes the feasible motion plan that the Motion Executive receives as its input. Furthermore, it continuously monitors the execution of this plan by constantly obtaining updates of the state of the environment and the state of the robot. If the Execution Monitor detects a disturbance that will prevent the plan from being executed successfully, it calls the Constraint Tightening component of the Motion Executive. We will provide a detailed explanation of the Constraint Tightening component shortly; however, in order to complete this bird's-eye view of the Motion Executive it is sufficient to say that the Constraint Tightening component determines whether a successful adjustment can be made to the motion plan, when the feasibility of the motion plan is threatened by the presence of a disturbance. If this is possible, the Constraint Tightening component makes this adjustment and execution continues. If however, no successful adjustment can be made the Constraint Tightening component returns with a failure. The Motion Executive also returns with a failure and requests the Reactive Motion Planner for a new plan.

3.3.1 The Constraint Tightening Component

In this section we will provide a detailed analysis of the Constraint Tightening component of the Motion Executive. We will begin by introducing the concept of a **Local Adjustment** to a disturbance which is the inspiration behind, and basis of, the rest of the material that we will present in this section. Briefly stated, Local Adjustment is a novel approach to modifying the motion plan to deal with certain kinds of disturbances, without needing to request the motion planner for a new plan. Once this concept of Local Adjustment has been examined, we will analyse a special instance of this Local Adjustment that relates to velocity-limited systems. Finally, we will provide a detailed explanation of the algorithm that is used by Chekhov in order to perform Constraint Tightening.

3.3.2 Local Adjustment

In this section we introduce an innovative approach that modifies motion plans to deal with certain kinds of disturbances that threaten the feasibility of the motion plan. These modifications are made without needing to call the Reactive Motion Planner and request a new plan.

A disturbance may cause an originally feasible motion plan to become infeasible. A plan becomes infeasible when one or more of the constraints expressed in Equation 2.1 are violated. We can attempt to deal with such disturbances by requesting a new plan from the Reactive Motion Planner. If the Reactive Motion Planner returns a feasible plan, we can execute this new motion plan. This will continue until either the task is accomplished or the Reactive Motion Planner is unable to generate a feasible motion plan that will accomplish the task goal. However, this planning can be slow. Local Adjustment was developed with the vision of making this process much more efficient in many situations. The goal of Local Adjustment is to *perturb the control policy that is expressed by the motion plan in a manner so as to ensure that no constraints in Equation 2.1 are violated any longer.* This idea is similar to the concept of **plan diagnosis**.

In the case of Local Adjustment however, we *cannot* find a solution by relaxing one or more constraints; the problem can only be solved by *changing* the solution. The disturbance represents new, additional constraints. The new plan must satisfy the constraints in Equation 2.1 as well as the new constraints added by the presence of the disturbance.

Let us crystallise some of the major ideas behind Local Adjustment by grounding this concept using the notion of flow tubes that we developed previously in this chapter. While the feasible motion plan generated by the Reactive Motion Planner is being executed, a disturbance in the form of an obstacle, may move close to and intersect one or more of the flow tubes that represent the motion plan and hence also represent the motion trajectory. The intersection of a flow tube with an obstacle implies that some of the trajectories in the flow tube are no longer feasible.

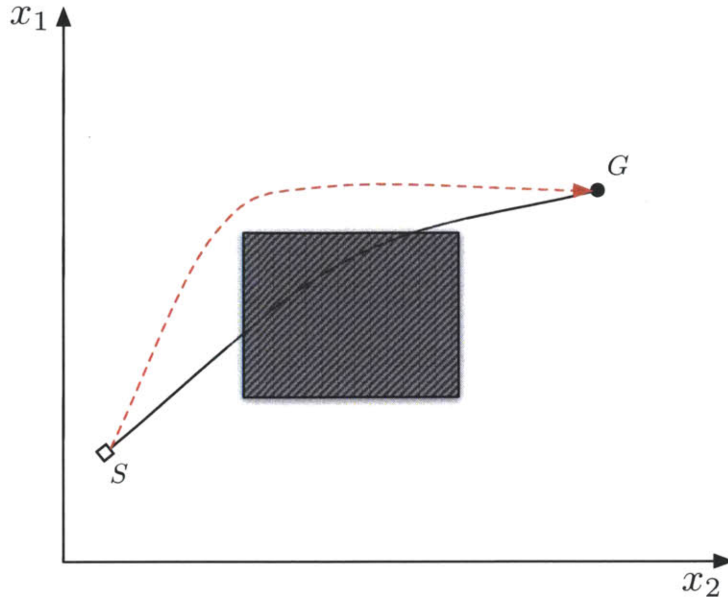


Figure 3-12: A Local Adjustment made to a trajectory. The vertical axis represents one of the joints of the robot while the horizontal axis represents another joint. The start pose of the robot is depicted by the unshaded rectangle, S. The goal pose is depicted by the black circle, G. The solid black line represents the original trajectory chosen by the Reactive Motion Planner. The obstacle in the environment is illustrated as a shaded rectangle. The dashed arrow illustrates the Local Adjustment that achieves the task goal while avoiding the obstacle.

Consequently, continuing the execution of the originally generated plan may result in a collision. In this scenario a Local Adjustment will involve shifting and possibly tightening (or constraining) some of the flow tubes so that the boundaries of the flow tubes no longer intersect with the obstacle. Once this Local Adjustment has been made we will then determine whether a feasible path to the goal still exists in these shifted and/or tightened flow tubes.

The techniques that are used to calculate a Local Adjustment are distinct from those that are used for planning and incremental replanning. Planning involves a *comprehensive, combinatorial* search. Local Adjustment also involves a search, however, the search is a simple, local search, one that is similar to those used in *non-linear, continuous, convex optimisation* problems. The planner is only called if this Local Adjustment is unsuccessful.

Let us consider the example in Figure 3-12. In this figure the vertical axis represents one of the joints of the robot while the horizontal axis represents another joint. The time axis is not shown in the diagram. The start pose of the robot is depicted by the unshaded rectangle, S, and the goal pose is depicted by the black circle, G. The obstacle is illustrated by a shaded rectangle. The solid black line represents the original trajectory that was chosen by the Reactive Motion Planner. In this example, the original trajectory is made infeasible by the sudden appearance of the obstacle. The continued execution of this trajectory would result in a collision. Consequently, a Local Adjustment needs to be made, failing which the Reactive Motion Planner will be requested for a new feasible motion plan. The Local Adjustment focusses on finding a new, *adjusted* trajectory by performing a simple, local search in the region around the original trajectory. In this example, the Local Adjustment is successful and the adjusted trajectory is illustrated by a dashed arrow. The Motion Executive can execute this trajectory to achieve the task goal without needing to request the Reactive Motion Planner for a new plan.

We wish to frame this kind of Local Adjustment as a non-linear, continuous, convex optimisation problem. For Local Adjustment we solve the same problem as that stated in Equation 2.1, however, we make a few additional, simplifying assumptions. It is important to note that the formulation in Equation 2.1 is not convex as $\neg(\mathbf{q} \cap \mathbf{C}(\mathbf{E}))$, the collision space constraints, are *not* convex. These collision space constraints generally make the problem *disjunctive* (non-convex). The planning algorithms can handle this non-convexity, however, these algorithms take time and are not fast enough to compensate for a disturbance in real-time. Consequently, our aim is to perform a Local Adjustment using convex optimisation techniques, which are much faster than their non-convex counterparts. We accomplish this by adding convex constraints to the problem formulation as collisions are detected and by ensuring that these collision are resolved when the problem is solved. This is analogous to how the Conflict-Directed A* [11] algorithm learns the structure of a problem and adds constraints that rule out infeasible solutions.

In order to understand this approach more clearly we delve deeper into the math-

ematics underlying this problem. Consider the following forward kinematics relation:

$$\mathbf{x} = f(\mathbf{q})$$

where $\mathbf{x}$ is the pose of a given point on the robot, which is close to a disturbance.

The flow tubes that represent the motion plan describe the valid evolution of $\mathbf{q}$ over time such that all the constraints in Equation 2.1 are satisfied. A trajectory is chosen from these flow tubes by the Reactive Motion Planner. The poses in this trajectory are checked using *collision detection algorithms.* These algorithms compute a set of collision points for each pose. If this set is non-empty for any pose, it implies that the pose is in collision and a Local Adjustment must be attempted. Furthermore, for each of these collision points, the collision detection algorithms compute not only the three-dimensional location of the collision, but also a *surface normal,* x_r, that indicates the direction in which the robot's colliding link should move in order to avoid or eliminate a collision. In addition, these algorithms also compute the *depth* or the *magnitude* of the collision. This depth information is crucial for resolving the issue of convexity, as this collision depth can be used to add *soft constraints* to Equation 2.1 when a collision is detected, instead of using the disjunctive collision space constraints, $\neg\left(\mathbf{q} \cap \mathbf{C}(\mathbf{E})\right)$. These soft constraints are convex and penalise collisions by associating these collisions with a cost that depends on the depth of the collision. The *deeper* the collision, that is, the more serious the collision, the higher will be the cost associated with that collision. When the collision is resolved, its cost will go to 0. Consequently, by adding these soft constraints to Equation 2.1, any algorithm that is used to solve this problem will eliminate the collisions in order to obtain the *optimal* (least-cost) solution.

We will now try to present this notion more concretely. As the pose $\mathbf{x}$ is close to a disturbance, we want to move $\mathbf{x}$ away from the disturbance in order to avoid a possible collision. This is achieved by calculating the surface normal, $\mathbf{x_r}$, which points away from the collision for the pose represented by $\mathbf{x}$. In particular, $\mathbf{x_r}$ specifies the direction in which $\mathbf{x}$ must move, in *cartesian* space, in order to prevent a possible

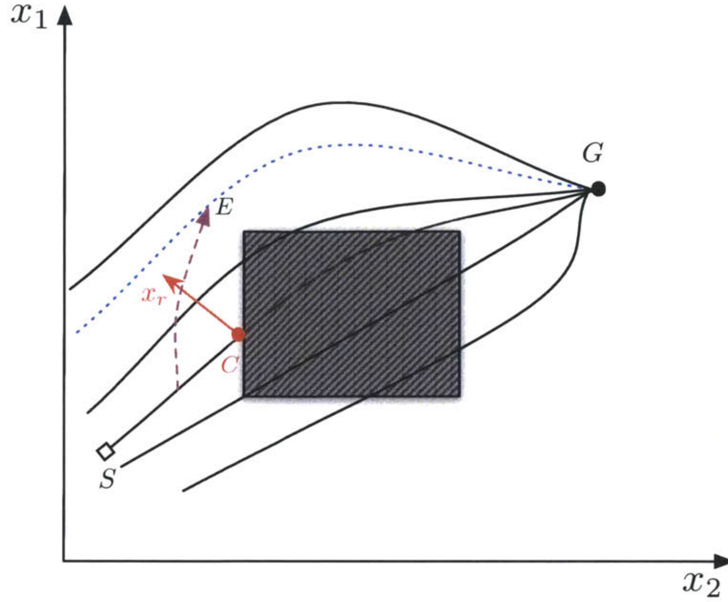


Figure 3-13: Procedure of finding and making a Local Adjustment. The start pose of the robot and the goal pose are depicted by the unshaded rectangle, S, and the black circle, G, respectively. The solid line connecting S and G is the original trajectory chosen by the Reactive Motion Planner. The other solid lines represent the other feasible trajectories within the flow tubes generated by the Reactive Motion Planner. The surface normal, $\mathbf{x_r}$, determined from the collision information, is illustrated as a solid arrow. The adjustment trajectory is illustrated as a dashed arrow. The new, feasible trajectory that is followed after the adjustment is made, is depicted using a dotted line.

collision with the disturbance.

In order to compute the direction corresponding to x_r in *joint* space, we use a **differential dynamic programming** approach which uses **Jacobians**.

$$\delta \mathbf{x_r} = \mathbf{J} \delta \mathbf{q} \tag{3.1}$$

where $\mathbf{J}$ is the Jacobian for the collision point. The pseudo-inverse of $\mathbf{J}$ is used to compute $\delta \mathbf{q}$ from $\delta \mathbf{x_r}$.

This employs an *iterative algorithm* that alternates between:

1. Obtaining all the $\delta \mathbf{x_r}$ vectors for all the collision points detected and computing the the corresponding Jacobians and $\delta \mathbf{q}$ vectors, and

2. Making an optimal adjustment for $\mathbf{q}$ by moving it in the direction $\delta\mathbf{q}$.

The first step in this algorithm represents a *linearisation* of the system, while the second step is an *optimisation* that uses the linearisation made in Step 1.

Consequently, the issue of determining an adjustment to $\mathbf{q}$ becomes a convex, non-linear, optimisation problem. Moreover, by using Jacobians this can be further simplified into a series of convex linear optimisation problems within the iterative framework. However, this iterative process must ensure that the constraints expressed in Equation 2.1 are satisfied for any attempted adjustment. If, at any point, an iteration fails because no improvement can be made in the desired direction, the Local Adjustment will return with a failure. This causes the Motion Executive to return with a failure as well and the Reactive Motion Planner is requested for a new plan.

If, however, the iteration succeeds in computing a $\mathbf{q}$ that resolves the collision, this adjustment is made using a control action. The trajectory generated by this control action is called the **adjustment trajectory**.

Definition 3.6 (Adjustment Trajectory). *The* ***adjustment trajectory*** *is a modification made to the original trajectory that successfully avoids the disturbances that threatened the original trajectory. This adjustment trajectory within the original flow tube.*

Once the adjustment is made, we search for a trajectory in the flow tube that includes $\mathbf{q}$ and achieves the task goal. If such a trajectory is found we begin execution of this new trajectory without ever needing to call the Reactive Motion Planner. If no such trajectory exists, the Local Adjustment will return with a failure, and this will ultimately result in a new plan being requested from the Reactive Motion Planner.

The procedure for finding an adjustment trajectory and for performing this Local Adjustment is summarised pictorially in Figure 3-13. The start pose of the robot and the goal pose are depicted by the unshaded rectangle, S, and the black circle, G, respectively. The solid line that connects S and G, is the original trajectory chosen by the Reactive Motion Planner. The other solid lines are the other feasible

trajectories within the flow tubes generated by the Reactive Motion Planner. The obstacle in the environment is illustrated by the shaded rectangle. Here, much like in the example depicted in Figure 3-12, the original trajectory is made infeasible by the sudden appearance of the obstacle. If execution continues, there will be a collision at point C. Information obtained from the collision detection algorithms is used to determine the surface normal, $\mathbf{x_r}$, which is shown in the figure as a solid arrow. The incremental algorithm is then used to generate a control action that modifies the original trajectory so as to avoid the obstacle. This adjustment trajectory is illustrated in the figure as a dashed arrow. The end of the adjustment trajectory, E, is the final $\mathbf{q}$ computed by the incremental algorithm. The dotted line in the figure represents the new, feasible trajectory which contains both the final $\mathbf{q}$ depicted by E and the goal pose, G.

3.3.3 Local Adjustment in Velocity Limited Systems

Now that we have analysed the concept of Local Adjustment in detail, we can use that knowledge and intuition to scrutinise a special case of Local Adjustment in velocity-limited systems; systems in which the acceleration limits are omitted. The omission of the acceleration constraints along with the elimination of the collision-space constraints, as discussed in the previous section, result in the following simplification of Equation 2.1:

$$\begin{aligned} &minimize\ c\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}, \dot{\mathbf{u}}\right) \\ &\mathbf{q}_{min} < \mathbf{q} < \mathbf{q}_{max} \\ &\dot{\mathbf{q}}_{min} < \dot{\mathbf{q}} < \dot{\mathbf{q}}_{max} \\ &\mathbf{g}_{fk}([\mathbf{q_{tf}}, \dot{\mathbf{q}}_{\mathbf{tf}}]^T) \in R_g \\ &d_{min} < d < d_{max} \end{aligned} \tag{3.2}$$

The collision-space constraints that have been omitted in Equation 3.2, are accounted for in the cost function c. Furthermore, the goal constraints in the original formulation in Equation 2.1, $\mathbf{g}_{min}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}\right) < [\mathbf{q_{tf}}, \dot{\mathbf{q}}_{\mathbf{tf}}]^T < \mathbf{g}_{max}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}\right)$, have been replaced by a set of convex, linear constraints in the cartesian space:

$$\mathbf{g}_{fk}([\mathbf{q_{tf}}, \dot{\mathbf{q}}_{\mathbf{tf}}]^T) \in R_g$$

where R_g specifies the goal region in cartesian space and $\mathbf{g}_{fk}$ represents the forward kinematics of the system.

In this formulation, we have also omitted the operating constraints for the sake of simplicity, however, these can be added to this formulation, if necessary. The trajectories that will result after solving this problem are represented by position and velocity vectors that are indexed by a discrete time index, k. Consequently, these trajectories are of the form:

$$\mathbf{q}(0)\ \mathbf{q}(1)\ ...\ \mathbf{q}(k)\ ...\ \mathbf{q}(n)$$
$$\dot{\mathbf{q}}(1)\ \dot{\mathbf{q}}(2)\ ...\ \dot{\mathbf{q}}(k)\ ...\ \dot{\mathbf{q}}(n)$$

The position and velocity in the trajectory are related by the following linear, equality constraint:

$$(\dot{\mathbf{q}}(k) = \mathbf{q}(k) - \mathbf{q}(k-1)) \quad \forall\ k = 1\ ....\ n$$

Moreover, the optimal adjustment made to $\mathbf{q}$, by moving it in the direction $\delta\mathbf{q}$ can be expressed as a quadratic function of the depth of the collision. Thus, this formulation is a linear, or easily linearisable, system that can be solved using linear programming.

3.3.4 The Constraint Tightening Algorithm

In the previous sections, we introduced and provided a detailed explanation of the concept of Local Adjustment and examined a special instance of Local Adjustment that relates to velocity-limited systems. In this section, we will analyse the algorithm that is used by the Constraint Tightening component of the Motion Executive. This algorithm is a simplified version of Local Adjustment, and it focusses on implementing

Algorithm 3: `ConstraintTightening`

```
   Input: A Flow Tube
   Output: The Constrained Flow Tube, if feasible
 1 foreach cross-section in the flow tube do
 2     MidPoint, ← FindMidPoint(cross-section);
 3     MaxPoint ← FindMaxPoint(cross-section);
 4     MinPoint ← FindMinPoint(cross-section);
 5     if !inCollision(MidPoint) then
 6         if DetectCollision(MidPoint, MaxPoint) then
 7             NewMaxPoint ← CollisionPoint;
 8         if DetectCollision(MidPoint, MinPoint) then
 9             NewMinPoint ← CollisionPoint;
10     else if !inCollision(MaxPoint) then
11         if DetectCollision(MaxPoint, MidPoint) then
12             NewMinPoint ← CollisionPoint;
13     else if !inCollision(MinPoint) then
14         if DetectCollision(MinPoint, MidPoint) then
15             NewMaxPoint ← CollisionPoint;
16     else
17         Error:"No feasible path found";
18     UpdateCrossSection ();
19 UpdateFlowTube ();
20 if CheckFeasible(tightened flow tube) then
21     return (tightened flow tube)
22 else
23     Error:"No feasible path found";
```

the notion of Local Adjustment using two-dimensional, velocity-limited flow tubes.

At a high-level, the algorithm iterates through each cross-section in the flow tube and determines whether there is a disturbance that obstructs the cross-section at any point. This information is obtained from the collision detection algorithms in the OpenRAVE environment [2]. If it has, this constitutes a collision and the $\neg(\mathbf{q} \cap \mathbf{C}(\mathbf{E}))$ constraint in Equation 2.1 is violated. Consequently, the cross-section is *cropped* at this collision-point. Once the algorithm has iterated through all the cross-sections in the flow tube, the flow tube is updated to reflect the changes made to its cross-sections. This updated, *constrained* flow tube is then tested to determine whether it is still feasible. In particular, the family of trajectories that the flow tube represents must satisfy all the constraints in Equation 2.1. If the flow tube is feasible, the algorithm returns this constrained flow tube. If not, the algorithm returns with a failure. The pseudo-code for this algorithm is shown in Algorithm 3.

Let us now scrutinise this algorithm further by providing a more detailed analysis. The algorithm accepts a flow tube as its argument and begins by iterating through each cross-section in the flow tube (Line 5). A cross-section is constrained, or tightened if it contains a pose for which the robot is in collision. In particular, if it violates the $\neg(\mathbf{q} \cap \mathbf{C}(\mathbf{E}))$ constraint in Equation 2.1. The collision check is performed by the `inCollision` and `DetectCollision` functions using the collision detection algorithms in the OpenRAVE environment. Lines [5:15] constitute the *collision-checking block* of the algorithm and they determine whether a given cross-section needs to be constrained. In addition, it stores the new limits of the cross-section if the cross-section has been constrained. The `UpdateCrossSection` function (Line 18) updates the cross-section to reflect the changes.

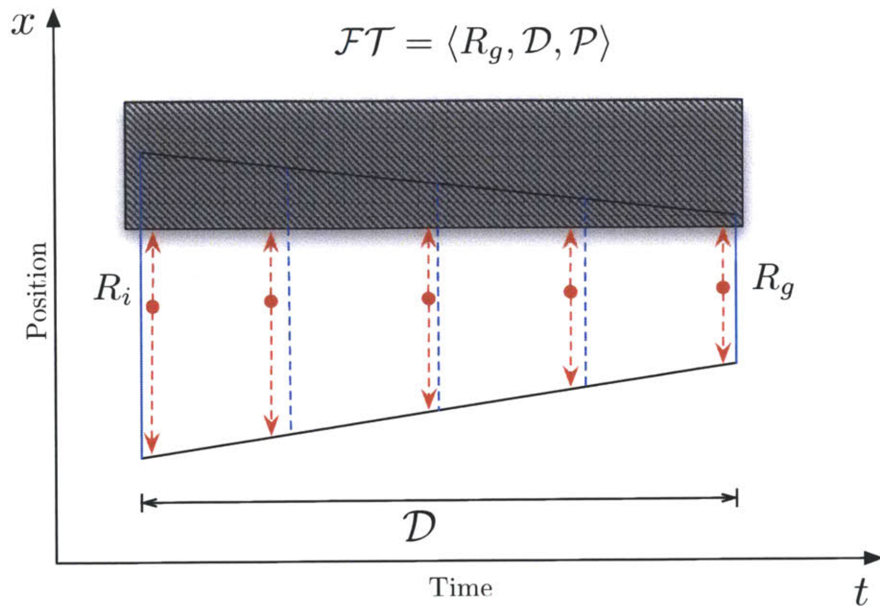
We begin checking for disturbances by starting at the mid-point of each cross-section and by moving outwards in both directions towards the maximum and minimum of the cross-section. If a collision is detected, the maximum and/or the minimum are updated as required. However, if the pose specified by the mid-point of the cross-section is in collision, we start at the maximum limit of the cross-section and move towards the mid-point. As the maximum point of the cross-section is not in collision

and the mid-point of the cross-section is in collision ,we can assume that the minimum of the cross-section will also be in collision.

Similarly, when the maximum and the mid-point of the cross-section are in collision, we start at the minimum of the cross-section and move only until the mid-point.

This procedure is illustrated in Figure 3-14. This figure depicts two flow tubes undergoing constraint tightening. The solid lines depict the boundaries of the flow tubes, while the dashed lines represent the cross-sections within the flow tubes. The filled circle next to each cross-section is the mid-point of that cross-section, and the dashed arrows illustrate the manner in which each cross-section is examined by the algorithm. The obstacle in the environment is illustrated by a shaded rectangle. The upper portion of the figure depicts the case in which the algorithm begins at the mid-point of each cross-section and moves outwards in either direction. When the obstacle is encountered, the cross-section is cropped as required. The lower portion of figure illustrates the case in which the mid-points of two of the cross-sections of the flow tube are in collision. Consequently, for these cross-sections the algorithm begins at the maximum point of the cross-section and then moves downwards, towards the mid-point of the cross-section. When the collision is encountered the cross-section is cropped as required.

Once all the cross-sections have been constrained, the `UpdateFlowTube` function (Line 19) is used to update the flow tube to reflect these changes. Furthermore, the `CheckFeasible` function (Line 20) is used to determine whether the constrained flow tube is feasible. If it is feasible, the algorithm returns with the constrained flow tube. If the flow tube is determined to be infeasible, the algorithm returns with an error, indicating that the flow tube is infeasible and implying the need to request the Reactive Motion Planner for a new plan.



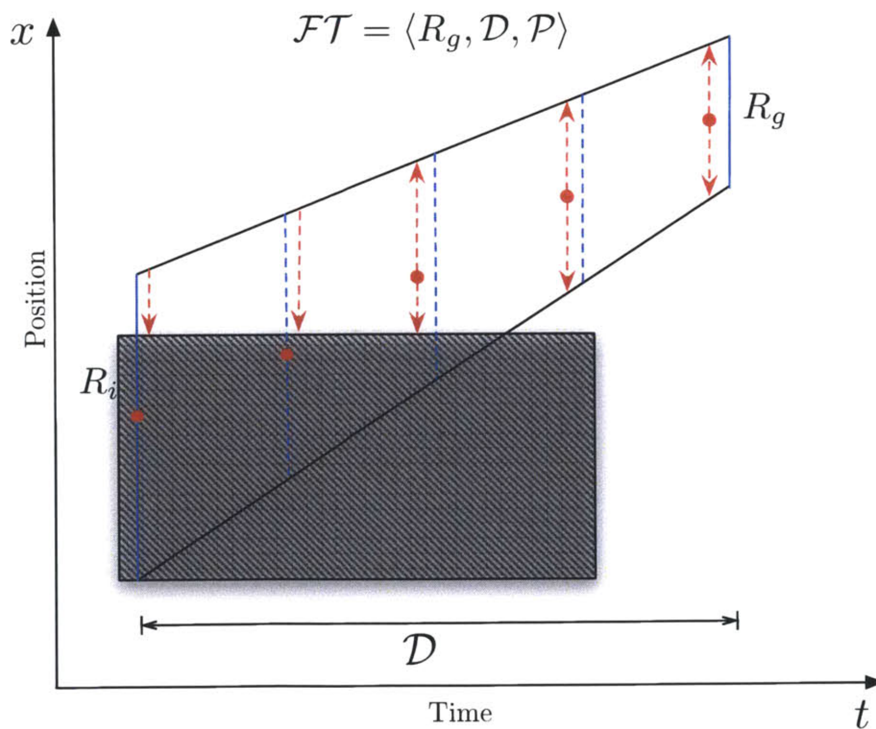


Figure 3-14: The constraint tightening algorithm. This figure depicts two flow tubes undergoing constraint tightening. The solid lines depict the boundaries of the flow tubes. The dashed lines represent the cross-sections within the flow tubes. The filled circle next to each cross-section is the mid-point of that cross-section. The dashed arrows illustrate the manner in which each cross-section is examined by the algorithm.

Chapter 4

The Manufacturing Test Bed

Previously in this thesis we presented the problem that we are trying to solve using the Chekhov Executive, and we described how the Chekhov Executive solves this problem. Now that every aspect of Chekhov's machinery has been presented and examined in detail, we move onto analysing the results that we have obtained while testing the Chekhov Executive.

We begin this chapter by first, re-iterating our vision for the future of robotic manufacturing. Next, we will provide an overview of the overall system architecture that is used in our robotic test bed, and then examine each of the components of this architecture in detail. Finally, we will conclude this chapter by presenting our results and by discussing the implications of these results.

4.1 Vision for Robotic Manufacturing

As we described in the introductory chapter of this thesis (Chapter 1), robots are used extensively in mass-production factories around the world today. However, these robots only perform monotonous and repetitive tasks with little or no variability. These robots are incapable of making significant execution-time adjustments. Additionally, these robots can only function effectively in manufacturing scenarios that are highly structured and controlled. These restrictions are primarily a result of the fact that these robots are unable to respond to unexpected failure and are unable

to adapt autonomously to new situations.

We envision a manufacturing scenario in the future, in which humans and robots collaborate to accomplish tasks in unstructured environments. These robot must satisfy a range of criteria in order to make this vision a reality. However, in this thesis, we focus on one essential capability that these robots must have if this vision of a collaborative, intelligent manufacturing environment is to be achieved. In this section we demonstrate this capability; a capability by which a robot can operate in a dynamic environment. Additionally, the robot in this demonstration is capable of autonomously adapting to new situations and disturbances. We have implemented a simplified version of such a robotic manufacturing scenario and will analyse this, in detail, in the next section.

4.2 System Capabilities

In this section we will provide a detailed explanation of our robotic manufacturing scenario. This demonstration showcases a simplified version of the capability by which a robot can operate in a dynamic environment and can autonomously adapt to new situations and disturbances. The robotic manufacturing scenario comprises :

- A Whole Arm Manipulator (WAM) produced by Barrett Technology Inc. that is equipped with a hand (also produced by Barrett).
- A number of boxes marked with black and white fiducial tags, that serve as obstacles.
- A number of inexpensive web-cameras (web-cams).

The goal of this scenario is for the robot to go from a given start pose to a specified goal pose. This will require the Chekhov Executive to sense the environment and generate a plan using the Reactive Motion Planner. Chekhov will also be responsible for executing this feasible plan using its Motion Executive. Furthermore, while the plan is being executed, the sensing system (the array of web-cams) will continuously

monitor the state of the environment and the state of the robot. If a disturbance is introduced that obstructs the execution of the robot, and prevents it from reaching its goal pose, the Motion Executive will try to salvage a feasible trajectory from the original plan using Local Adjustment. If this is not possible, the Motion Executive will request the Reactive Motion Planner for a new plan, and it will execute this new plan, if one exists.

Consequently, the robot is robust to environment changes and to failure. Explicitly stated, if the robotic agent detects a disturbance that will prevent the successful execution of its plan, the robot will automatically discern a new course of action to achieve its task goal and execute it. If there is no possible way for the robot to achieve the task goal, it will return with a failure.

Although the scenario that we have presented, by no means demonstrates the robot's mastery of every capability that would be required in a real manufacturing scenario, this demonstration does encapsulate many of the key aspirations that we have for a futuristic, intelligent, robotic manufacturing factory. This scenario strikes a balance between the practicality of building and implementing an academic demonstration, and showcasing the key contributions made by our research, which is the central focus of this thesis. We use this demonstration to emphasise the key techniques that allow for swift plan adjustment in response to disturbances.

4.3 System Architecture

In this section we provide a detailed explanation of the system architecture that is used in the robotic manufacturing test bed. We will begin by providing a high-level, bird's-eye view of the overall system architecture and the manner in which the major components in this architecture interact with one another. Next, we will analyse each component of this system architecture in detail.

The overall system architecture of the robotic manufacturing test bed is shown in Figure 4-1. This architecture comprises four major components:

- The Chekhov Executive ROS Node

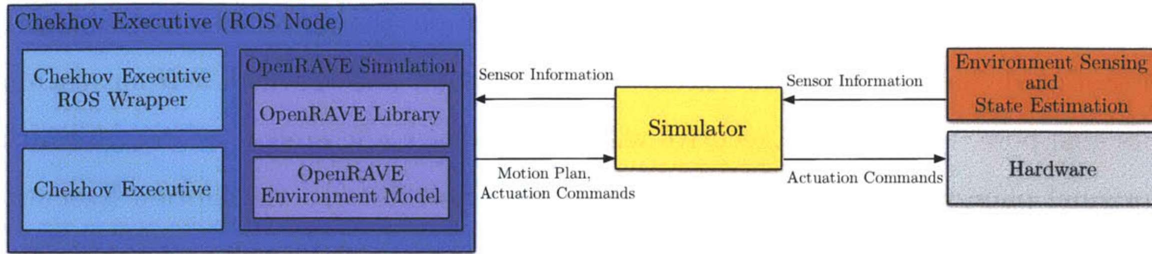


Figure 4-1: The system architecture of the Chekhov Robotic Test Bed

- The Simulator
- The Sensing and State Estimation Module
- The Hardware Module

The Chekhov ROS Node, the Sensing and State Estimation module and much of the Simulator are programmed in C++. However, the Simulator module does provide an API that is written in Python. The Robot Operating System (ROS) is used to combine these independent components into one overall system that functions seamlessly. Both the Chekhov Executive ROS Node and the Simulator module have their own OpenRAVE simulation environment [2] and rely heavily on it. The decision to have two separate OpenRAVE environments in the system architecture was a conscious decision. The primary motivation was to allow for parallel processing. This architecture design allows the OpenRAVE environment within the Chekhov ROS node to focus only on motion planning and execution, while the other OpenRAVE environment, in the Simulator, focusses on performing the simulation and state estimation.

The demonstration begins with the operator choosing a goal pose that the robot needs to achieve. This information is relayed to the Chekhov ROS node which generates a feasible plan from the current robot state to the specified goal pose, if one exists, using the Chekhov Executive that is within the Chekhov ROS Node. The plan generation is performed in the OpenRAVE simulation environment within the Chekhov ROS node.

The OpenRAVE environment within the Chekhov ROS node contains up-to-date

information regarding the current state of the robot and the state of the environment. This simulation environment is synced with the OpenRAVE simulation environment in the Simulator module. This was an important design decision and was made to ensure that all the sensor data from the Sensing and State Estimation module, are routed through the Simulator module and are not sent directly to the Chekhov ROS node. This has two primary benefits. First, as mentioned before, it allows for parallel processing. Second, it ensures that the two OpenRAVE environments are always synced and reflect the same state of the environment. Inconsistencies in the two environment would lead to a multitude of issues; however, this architecture prevents such inconsistencies from occurring.

Once the Chekhov Executive has generated a feasible plan, it sends the actuation commands to the Simulator module. The Simulator nodule sends these actuation commands to the Hardware module. In the Hardware module, this information is send to the WAM Hardware controller which communicates with the hardware and physically actuates the robot The Hardware component also sends proprioceptive sensory information, like the current joint angles of the robot, back to the Simulator module. When the OpenRAVE simulation in the Simulator module is updated, the OpenRAVE simulation environment in the Chekhov ROS node is automatically updated as well.

The Sensing and State Estimation module runs continuously in the background and tracks the black and white fiducial tags on the obstacles. This information is then routed back to the Simulator which updates the locations of the blocks in its OpenRAVE simulation. This causes the OpenRAVE simulation in the Chekhov ROS node to be updated as well. The Chekhov Executive's Motion Executive also runs constantly, until the task goal has been accomplished. If, at any point, the Motion Executive detects a disturbance, an obstacle in this demonstration, that will prevent the plan from being executed successfully, it attempts to make a Local Adjustment using its Constraint Tightening algorithm and continues execution. If a Local Adjustment cannot be made, the Reactive Motion Planner is requested for a new plan and execution continues if a new feasible plan is generated. If no feasible plan exists, the robot halts

and reports a failure.

Another critical design decision was to route all the hardware control and sensing information through the Simulator before it was sent to the Chekhov ROS node and before it was dispatched to the actual hardware in the Hardware module. This decision was made in order to allow this architecture to be used only in simulation, without being connected to the Hardware module and it provides us with a number of benefits. Most importantly, by doing away with the hardware we can perform many more extensive tests and we can simulate rare faults and random disturbances, which are difficult to recreate in hardware.

Now that we have presented the overall system architecture of the manufacturing test bed and discussed the manner in which these components interact, we will scrutinise each of these components. We will examine the Chekhov Executive ROS Node in detail, however we will provide only a brief description of the Simulator, Sensing and State Estimation, and Hardware modules, as a detailed analysis of these components is beyond the scope of this thesis and can be found in [9].

4.3.1 The Chekhov Executive ROS Node

As shown in Figure 4-1 the Chekhov Executive ROS Node comprises three main components:

- The Chekhov Executive ROS Wrapper
- The Chekhov Executive
- The OpenRAVE Simulation

We have examined the Chekhov Executive in great detail in Chapter 3 and we will rely on the intuition and understanding gained in Chapter 3 as we continue. The Chekhov Executive ROS Wrapper is an application programming interface (API) that allows an operator to use the Chekhov Executive effectively, without having an knowledge or understanding about the inner-workings of the system. The primary

purpose of this API is to allow the Chekhov Executive to communicate with other ROS nodes.

As we have mentioned previously in this chapter, the Chekhov Executive ROS Node has its own OpenRAVE simulation environment complete with its own OpenRAVE library and OpenRAVE environment model. The OpenRAVE environment in the Chekhov ROS node is used to perform the collision checking that is vital for motion planning and execution. This is also extremely valuable for testing purposes. The Chekhov Executive can be tested in a simulated environment with millions of obstacles and disturbances, something that would be difficult to recreate in the real world, on an academic test bed.

4.3.2 The Hardware Module

The physical hardware in the test bed comprises a seven degree-of-freedom Barrett WAM equipped with a three-fingered Barrett Hand. The WAM is extremely dexterous and can move from one pose to another very quickly. Moreover, with seven degrees of freedom the arm is kinematically redundant, which means that there are a number of different joint angle combinations that result in the same end-effector position. This is extremely useful for a system like the Chekhov Executive that may be required to find alternative paths to the same goal pose.

The Barrett Hand contains three independently actuatable fingers, two of which can spin symmetrically about its middle. A more detailed description of the hardware can be found in [9].

4.3.3 The Sensing and State Estimation Module

The Sensing and State Estimation module comprises a number of ROS packages that have been designed to track objects in the environment. At a high-level, this module accepts raw sensor data as its inputs, and produces two outputs:

1. Pose Estimates for objects with fiducials, that are being manipulated, and

2. Locations of obstacles that need to be avoided. This is done by using an octomap [4]. In this case, the pose estimates, along with the orientation of the obstacles, are unnecessary. The only pertinent information is the region of the workspace that is obstructed by the obstacle. This allows the planner to plan around these obstacles.

This module makes extensive use of the open source, ARToolkit libraries [5]. The data that is obtained from the fiducial tracking is extremely noise and consequently, we have used a series of filtering algorithms, involving Hidden Markov Models (HMMs) and Kalman filters, to extract more meaningful pose data from this noisy data. This filtered data is then routed back to the simulation which then uses this sensor data to update the position of the block. This Sensing and State Estimation module was implemented primarily by Pedro Santana. A more detailed explanation of this module is beyond the scope of this thesis and can be found in [9].

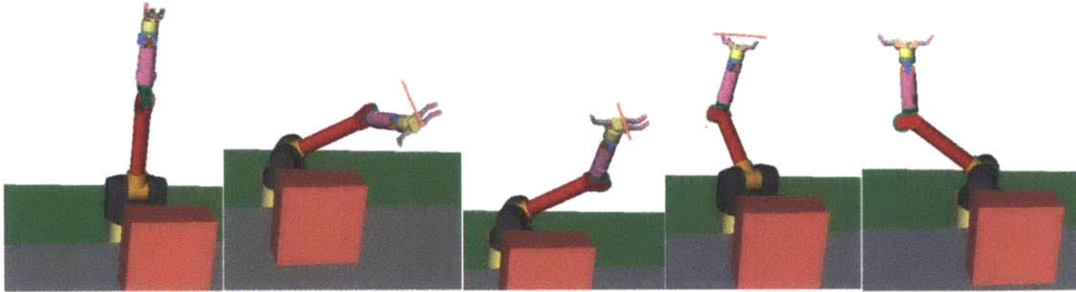
4.4 Results

Now that we have provided a detailed explanation of the system architecture that is used to test the Chekhov Executive, we will present the results of our tests. The Chekhov Executive was tested both in hardware (on the WAM) and in simulation (in the OpenRAVE environment).

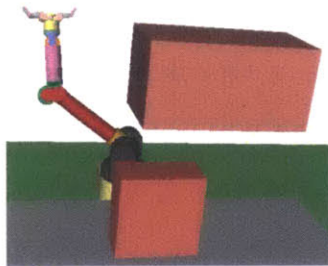
4.4.1 Results for the Reactive Motion Planner

A representative test case for the Reactive Motion Planner is shown in Figure 4-2. The uppermost portion of this figure illustrates snapshots of the execution of a motion plan generated by the Reactive Motion Planner in the OpenRAVE environment. The WAM successfully moves from its start pose to its goal pose without hitting any of the obstacles, like the table or the block that is on the table, present in the environment. The algorithm used to generate this initial trajectory required 367 update steps of the D* Lite algorithm. The middle portion of Figure 4-2 shows a new obstacle that

Original Plan



Obstacle Introduced



Modified Plan

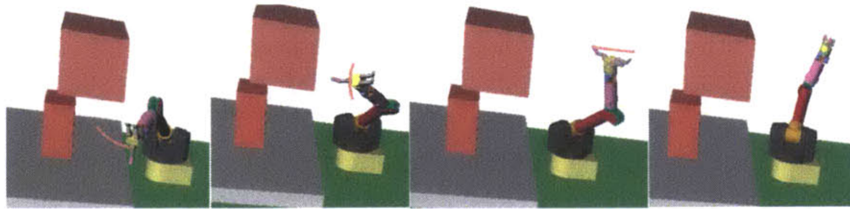


Figure 4-2: Representative test case for the Chekhov Executive. The uppermost portion shows snapshots of the WAM executing the motion plan in the OpenRAVE simulation environment. The middle portion shows the new obstacle that is introduced in the simulation, which prevents the WAM from executing the originally generated plan. The lower portion shows snapshots of the WAM executing a new plan that achieves the original task goal while avoiding the new obstacle.

Table 4.1: Results obtained after testing the Reactive Motion Planner

Initial Steps	Initial Cost	Quick Path Query Cost	Adjustment Steps	Adjustment Cost
692	3.525	3.9	8	3.53
806	4.566	4.566	0	4.566
783	6.11	6.139	28	6.139
1343	5.034	5.034	10	5.034
738	4.164	4.164	2	4.164
1077	3.989	3.989	0	3.989
705	3.208	3.375	73	3.375
791	4.763	5.165	13	5.084
736	4.25	4.274	36	4.274
648	3.028	3.294	63	3.294

has been introduced into the same OpenRAVE simulation environment. This obstacle makes the original plan generated by the Reactive Motion Planner infeasible and also intersects several edges in the state-space graph generated by the D* Lite algorithm. The lowermost portion of the figure depicts snapshots of the WAM executing a new plan that was generated by using the Reactive Motion Planner's incremental re-planning capability. This new, modified plan contains a trajectory that avoids the new obstacle and still successfully achieves the task goal. The incremental replanning algorithm required only 61 update steps of the D* Lite algorithm to generate the new plan. This reduces the computational effort of generating a plan by more than a factor of 5. The state-space graph used by the D* Lite algorithm had 2000 regularly-spaced joint vertices, in this instance. In addition, in this example the planning was performed for the four proximal joints of the WAM, while the 3 distal (wrist) joints were fixed at position 0.

The data that we have collected after testing the Reactive Motion Planner are shown in Table 4.1. The state-space graph used by the D* Lite algorithm for these tests consisted of 1000 randomly distributed state vertices. The first two columns in Table 4.1 relate to the intial solution. The first colum shows the number of D* Lite update steps required to generate the initial solution. The second column contains the

cost associated with this initial solution. As we have mentioned while analysing the Enhanced Algorithm in Chapter 3, a number of edges of the state-space graph may become infeasible when a new obstacle is introduced into the environment. There are two approaches to finding a plan that circumvents this new obstacle and still achieves the task goal. The first is to update the state-space graph until it is consistent and then search for the optimal path to the goal. The second is not to update the state-space graph until this is consistent, as this operation can prove to be computationally intensive. Instead, the edges of the state-space graph that have become infeasible due to the new obstacle are given extremely high traversal costs, which ensure that they will not be selected for use in any path. This updated, yet inconsistent, graph is then used to search for a plan. It is important to note that the two approaches are not mutually exclusive. The any-time algorithm combines the two and performs a quick, initial search over the inconsistent graph. If this search is successful, it returns this, possibly sub-optimal, solution. However, it continuously updates the solution so as to improve the optimality of the un-executed sections of the solution. If the inconsistent graph fails to provide a solution, it becomes necessary to update this graph, and search this newly updated graph for a solution. This will continue until either, a solution is found or the graph becomes consistent.

The third column in Table 4.1 is related to the second approach and shows the cost associated with the path that was returned when searching through the inconsistent state-space graph. The fourth column in the table shows the number of steps that the D* Lite algorithm requires in order to make the state-space graph consistent once again. It is extremely interesting to note that in all the cases shown in the table, the number of update steps required by the algorithm to re-converge on a new solution was substantially less, often by more than an order of magnitude, than the number of update steps required to compute the original solution. The fifth and final column in the table shows the cost of the path generated when the search was performed on the consistent state-space graph. In a majority of the cases shown in the table, the cost of the path returned by the any-time path query (the second approach) was as good as the cost of the path returned if the path was computed after the state-space graph

was consistent (the first approach). This is an intriguing finding, which demonstrates the importance of the any-time algorithm. Furthermore, it demonstrates that there may exist an important class of incremental replanning problems that can be solved with relatively little computation effort when compared with the computational effort required to compute the original solution.

In the future we intend to test this more extensively in a variety of different test scenarios, both in simulation and in hardware, in order to collect more detailed statistics on the effectiveness of this approach.

4.4.2 Results for the Motion Executive

Our test scenarios for the Motion Executive can be grouped into three broad cases. In the first set of test cases the test environments contained obstacles, however these obstacles did not constrain the flow tubes that represented the initial plan generated by the Reactive Motion Planner. In this first set of test scenarios, the best trajectory, based on the objective function, was extracted from the flow tubes and this trajectory was executed successfully.

For the second set of test cases, the test environments were extremely cluttered with obstacles. The obstacles severely constrained the flow tubes that represented the initial plan generated by the Reactive Motion Planner. In these scenarios, the constraint tightening algorithm successfully constrained the flow tubes and selected the best trajectory from the constrained flow tube. This feasible trajectory was then successfully executed by the Motion Executive within the Chekhov Executive. The robot avoided all the obstacles in the environment and moved from its start state to its goal state without needing to request a new plan from the Reactive Motion Planner.

The test environments for the third set of test cases were similar to those used in the second set and were extremely cluttered with obstacles. The constraint tightening algorithm successfully constrained the flow tubes in these scenarios. However, the constraint tightening algorithm was unable to salvage a feasible trajectory from these constrained flow tubes. Consequently, the Reactive Motion Planner was requested for a new plan.

Figure 4-3 depicts two representative examples of the cluttered environments that were used to test the Motion Executive. Figure 4-4 illustrates the execution a trajectory in the cluttered environment. This trajectory was chosen from the feasible trajectories in the constrained flow tubes.

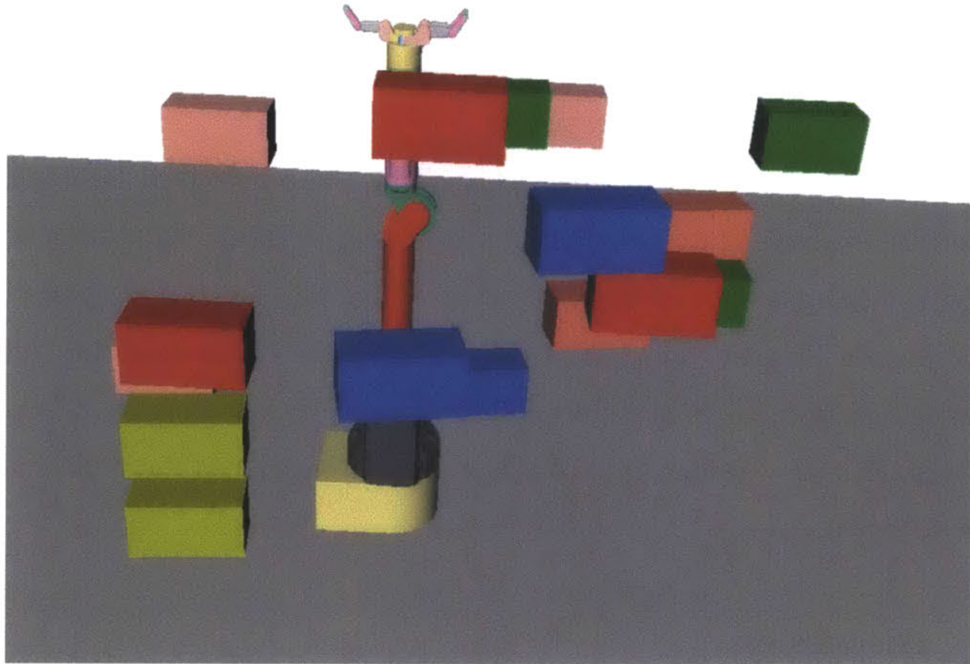
Figure 4-5 depicts the flow tubes returned by the constrained tightening algorithm for the example cluttered environment shown in the lower portion of Figure 4-3. The left column illustrates the flow tubes that were initially generated by the Reactive Motion Planner. The right column depicts the constrained flow tubes. All the constrained flow tubes in Figure 4-5 are feasible. Consequently, Chekhov can extract the optimal, feasible trajectory from these constrained flow tubes and then execute it.

4.4.3 Discussion

In the previous section we presented the results that we obtained after testing the Chekhov Executive. In this section we will analyse the implication of these results. The results, both from the Reactive Motion Planner and the constraint tightening algorithm, were extremely encouraging. The results obtained from testing the Reactive Motion Planner demonstrated the effectiveness of Chekhov's reactive capability and strongly supported the existence of a class of incremental planning problems that can be solved with relatively little computational effort when compared with the computational effort required to compute the original solution.

The constraint tightening algorithm proved to be extremely effective as well. The success of the algorithm in a variety of different test environments emphasised the potential of Local Adjustment. In particular, it underscored the importance of looking beyond an extensive search of the state space that the planner performs, and focussing on a smaller, more local search, which may be sufficient in many scenarios. Finally, it highlighted the importance of the idea of first searching for a new solution that is similar to the original solution, before straying farther away and performing a comprehensive search.

Together, at a general level, both these algorithms embody the higher level human



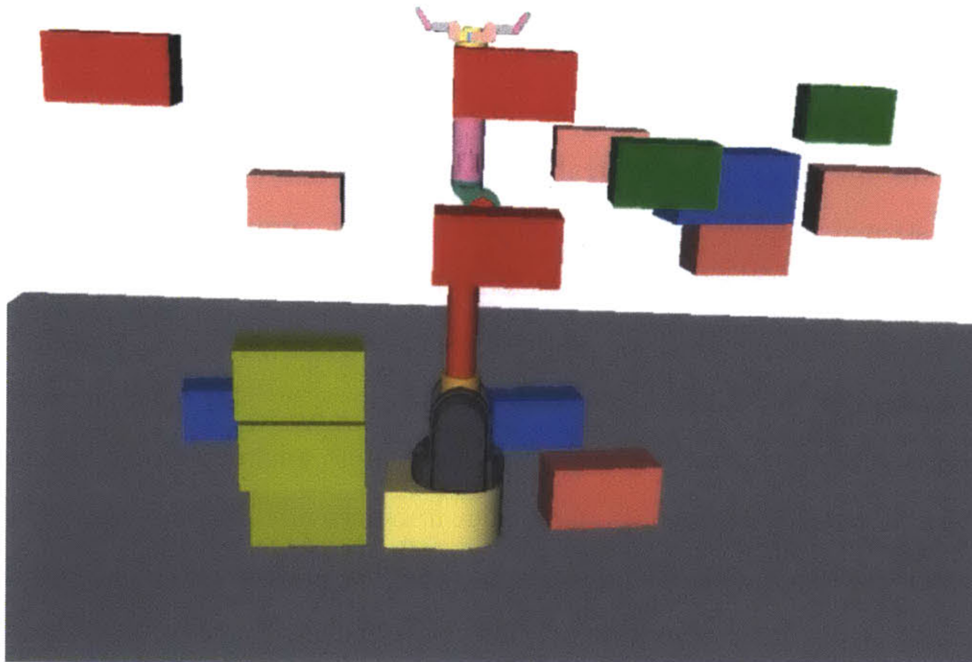


Figure 4-3: Representative examples of the cluttered environments used to test the Motion Executive.

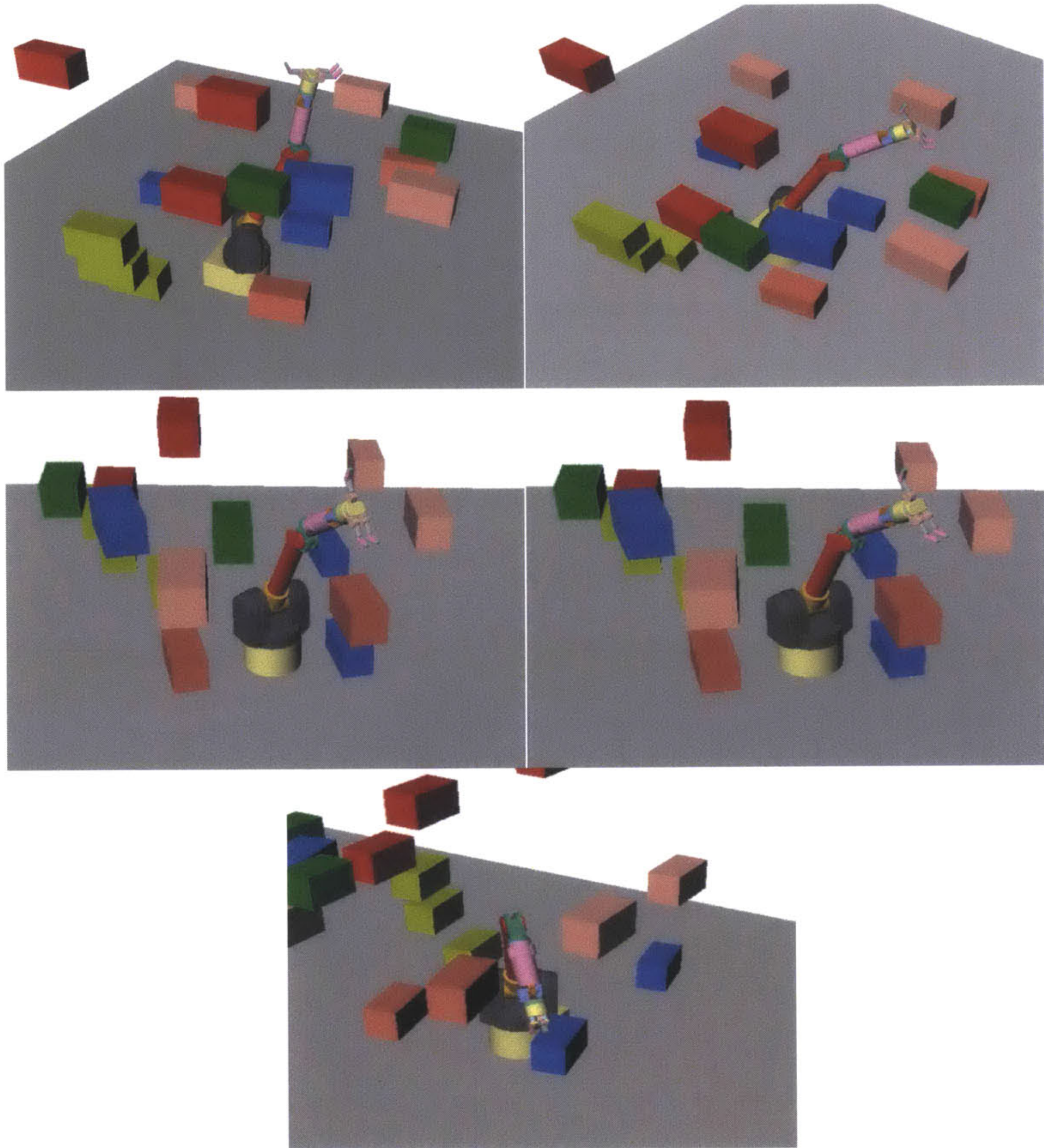


Figure 4-4: Plan execution in a cluttered environment. The trajectory that is being executed is chosen from the feasible trajectories in the constrained flow tubes.

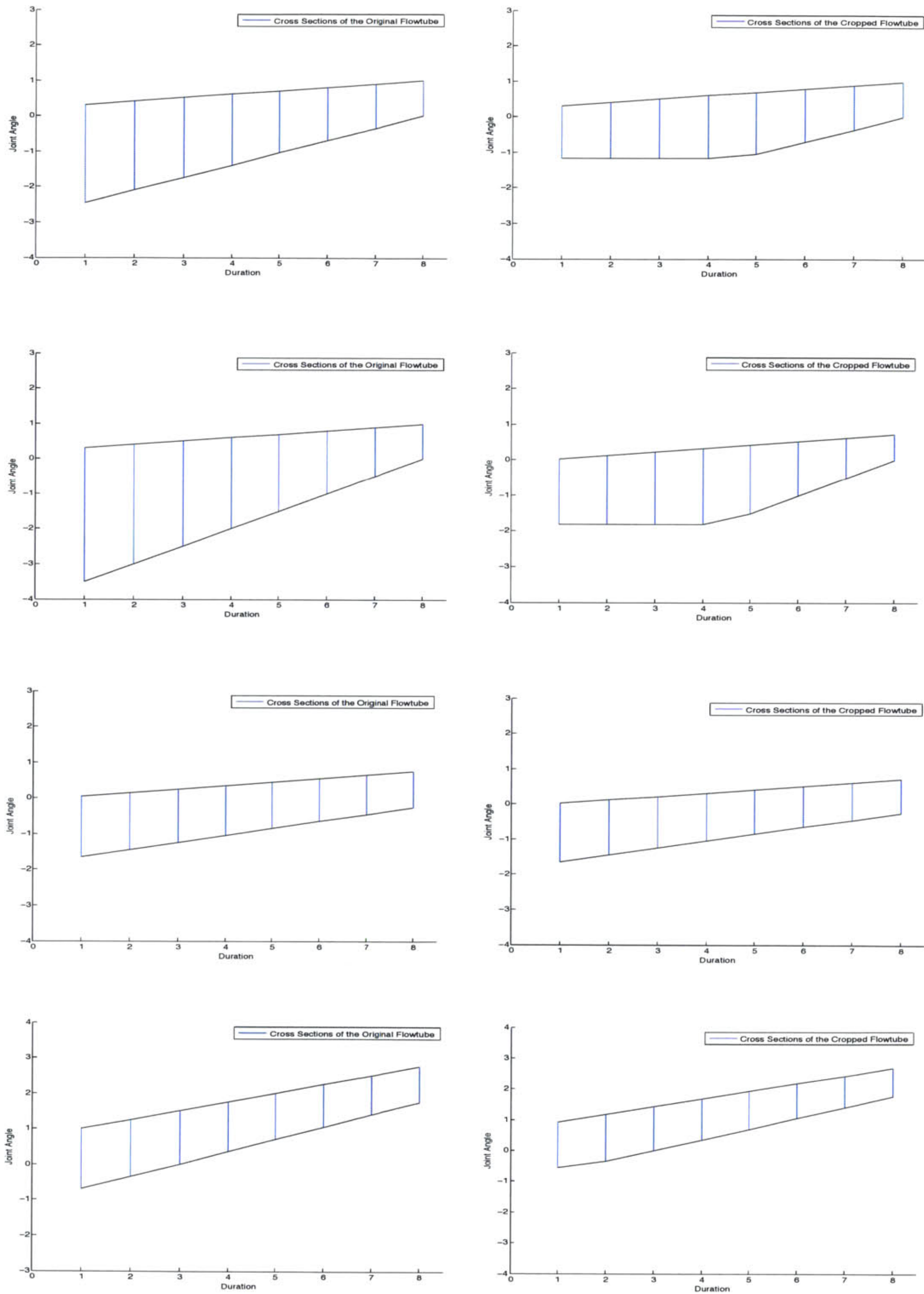


Figure 4-5: The flow tubes returned by the constrained tightening algorithm for the example cluttered environment shown in the lower portion of Figure 4-3. The left column depicts the initial flow tubes generated by the Reactive Motion Planner. The right column shows the constrained flow tubes.

technique of learning from experience. They use the information gained while finding the initial solution to bias their search for a new solution. These algorithms do require additional memory in order to cache aspects of the original solution that are useful for adjusting the original solution to form new solutions. However, memory is cheap and should be exploited if it can be used to help solve these kinds of complex problems.

Chapter 5

Contributions

This thesis has focussed on the development of a system that will enable a robot to function in an uncertain, real-world environment. In this thesis we introduce Chekhov, a reactive, integrated motion planning and execution executive that is capable of plan generation and plan execution in a relative uncluttered, dynamic and unstructured environment.

We have presented two algorithms, the Basic Algorithm and the Enhanced Algorithm, which form the very core of Chekhov's Reactive Motion Planner. These algorithms constitute one of the major contributions made by this thesis. Both the Basic Algorithm and the Enhanced Algorithm provide unique insights to replanning, when the original plan generated by the Reactive Motion Planner becomes infeasible due to a disturbance. The Enhanced Algorithm builds on the Basic Algorithm and makes a number of novel innovations. It employs an innovative any-time algorithm that rapidly finds a new plan, using a possibly inconsistent shortest-path tree, when the execution of the original plan is threatened by a disturbance. This new plan is not necessarily optimal, and consequently this algorithm constantly works to improve the optimality of the unexecuted sections of the plan. The decision to initially generate a plan using a possibly inconsistent tree, greatly speeds up the replanning process. This, coupled with the any-time improvement approach constitutes a fundamental innovation made by the Reactive Motion Planning component of the Chekhov Executive.

We have also introduced the novel concept of Local Adjustment, which plays a vital role in the functioning of the Motion Executive component of the Chekhov Executive. Local Adjustment serves as Chekhov's first line of defence when the execution of the original plan is threatened by a disturbance. Local Adjustment attempts to modify the original plan, such that all the constraints specified in the original planning problem are satisfied, without needing a new plan to be generated by the Reactive Motion Planner. This concept highlights the idea of utilising information procured from the original solution to influence, and bias, the search for a new, modified solution. Local Adjustment makes assumptions which simplify the original planning problem into a convex, non-linear optimisation problem that can be solved much faster than the original planning problem.

Finally, we offered a system architecture that showcases the Chekhov Executive, both in simulation and in the real world.

These insights culminate to help this thesis make a meaningful contribution to the field and provide a fundamental building block for research focussed on helping robots to operate in real-world environments.

Future Work

In Chekhov's current implementation, the positions of obstacles in the environment are estimated using the Sensing and State Estimation module that we have described previously. This only tracks obstacles that have a black and white coded fiducial on them. A first extension to the Chekhov Executive would be to track any obstacle in the environment, even those without a fiducial tag. We have already begun to work on integrating this capability into Chekhov. The untagged obstacles are tracked using an *octomap* [4].

Another interesting addition to Chekhov would be for us to create a system that predictively models the movement of obstacles in the environment. This probabilistic approach would allow Chekhov to be more intelligent in its decision making and would showcase the temporal aspects of Chekhov more clearly.

The predictive model of the environment can be further extended to include a disturbance model that considers the risk of a collision with a particular obstacle in the environment. Our group, the Model-Based Embedded Robotic Systems Group, has previously developed new algorithms for *risk-sensitive planning* [10]. These algorithms generate optimal plans within the risk bounds specified by chance constraints. The inclusion of these algorithms in the Chekhov Executive would greatly improve Chekhov's decision making in uncertain, dynamic environments.

Bibliography

[1] Alberto Casagrande, Andrea Balluchi, Luca Benvenuti, Alberto Policriti, Tiziano Villa, and Alberto Sangiovanni-Vincentelli. Improving reachability analysis of hybrid automata for engine control. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 3, pages 2322–2327. IEEE, 2004.

[2] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs.* PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[3] Andreas Hofmann, Steven Massaquoi, Marko Popovic, and Hugh Herr. A sliding controller for bipedal balancing using integrated movement of contact and non-contact limbs. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 2, pages 1952–1959. IEEE, 2004.

[4] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[5] H. Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99)*, San Francisco, USA, October 1999.

[6] Sven Koenig and Maxim Likhachev. D* lite. In *Proceedings of the national conference on artificial intelligence*, pages 476–483. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

[7] Alexander B Kurzhanski and Pravin Varaiya. Ellipsoidal techniques for reachability analysis: internal approximation. *Systems & control letters*, 41(3):201–211, 2000.

[8] Alex A Kurzhanskiy and Pravin Varaiya. Ellipsoidal techniques for reachability analysis of discrete-time linear systems. *Automatic Control, IEEE Transactions on*, 52(1):26–38, 2007.

[9] Steven James Levine. Monitoring the Execution of Temporal Plans for Robotic Systems. Master's thesis, Massachusetts Institute of Technology, 2012.

[10] Masahiro Ono and Brian C Williams. Iterative risk allocation: A new approach to robust model predictive control with a joint chance constraint. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 3427–3432. IEEE, 2008.

[11] Brian C Williams and Robert J Ragno. Conflict-directed A* and its role in model-based embedded systems. *Discrete Applied Mathematics*, 155(12):1562–1595, 2007.