Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2006-029 April 28, 2006

Coordinating Agile Systems through the Model-based Execution of Temporal Plans

Thomas LEAUTE

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

Coordinating Agile Systems through the Model-based Execution of Temporal Plans

by

Thomas Léauté

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2005

©Massachusetts Institute of Technology 2005. All rights reserved.
First Copy

Author .....
Department of Aeronautics and Astronautics
July 25, 2005

Certified by .....
Brian C. Williams
Associate Professor
Thesis Supervisor

Accepted by .....
Jaime Peraire
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Coordinating Agile Systems through the Model-based Execution of Temporal Plans

by

Thomas Léauté

Submitted to the Department of Aeronautics and Astronautics
on July 25, 2005, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Agile autonomous systems are emerging, such as unmanned aerial vehicles (UAVs), that must robustly perform tightly coordinated time-critical missions; for example, military surveillance or search-and-rescue scenarios. In the space domain, execution of temporally flexible plans has provided an enabler for achieving the desired coordination and robustness, in the context of space probes and planetary rovers, modeled as discrete systems. We address the challenge of extending plan execution to systems with continuous dynamics, such as air vehicles and robot manipulators, and that are controlled indirectly through the setting of continuous state variables.

Systems with continuous dynamics are more challenging than discrete systems, because they require continuous, low-level control, and cannot be controlled by issuing simple sequences of discrete commands. Hence, manually controlling these systems (or *plants*) at a low level can become very costly, in terms of the number of human operators necessary to operate the plant. For example, in the case of a fleet of UAVs performing a search-and-rescue scenario, the traditional approach to controlling the UAVs involves providing series of close waypoints for each aircraft, which incurs a high workload for the human operators, when the fleet consists of a large number of vehicles.

Our solution is a novel, model-based executive, called *Sulu*, that takes as input a *qualitative state plan*, specifying the desired evolution of the state of the system. This approach elevates the interaction between the human operator and the plant, to a more abstract level where the operator is able to "coach" the plant by *qualitatively* specifying the tasks, or *activities*, the plant must perform. These activities are described in a qualitative manner, because they specify regions in the plant's state space in which the plant must be at a certain point in time. Time constraints are also described qualitatively, in the form of flexible temporal constraints between activities in the state plan. The design of low-level control inputs in order to meet this abstract goal specification is then delegated to the autonomous controller, hence decreasing the workload per human operator. This approach also provides robustness to the executive, by giving it room to adapt to disturbances and unforeseen events, while

satisfying the qualitative constraints on the plant state, specified in the qualitative state plan.

Sulu reasons on a model of the plant in order to dynamically generate near-optimal control sequences to fulfill the qualitative state plan. To achieve optimality and safety, Sulu plans into the future, framing the problem as a disjunctive linear programming problem. To achieve robustness to disturbances and maintain tractability, planning is folded within a receding horizon, continuous planning and execution framework. The key to performance is a problem reduction method based on constraint pruning. We benchmark performance using multi-UAV firefighting scenarios on a real-time, hardware-in-the-loop testbed.

Thesis Supervisor: Brian C. Williams
Title: Associate Professor

Acknowledgments

I would first like to thank the people who actively helped me accomplish the work described in this thesis, and whose support was particularly helpful in the final writing phase: Hui, Tsoline, Andreas, Jake, and, above all, Brian, whose help and insight have been critical and inspiring, throug the months I spent working in the MERS group.

Other important contributors and lab mates are, in increasing lexicographic order: Bobby, Dimitri, Greg, John (a.k.a. Stedl), Lars, Marcia, Margaret (a.k.a. Peggy), Martin, Ollie, Paul E. and Paul R.; followed by the ones whose names start with an S: Seung, Shen, Steve and Stano. I also wanted to mention the people whose company I wish I had had more time to appreciate: Brad, I-hsiang, Jon, and Larry.

Apart from the research carried out in the MERS group, the work accomplished by the following people has also been very inspiring: Éric Féron, whose outstanding enthusiasm deserves recognition, Jon How, Yoshi Kuwata, Tom Schouwenaars, Arthur Richards, and John Bellingham.

I would like to thank my family: my parents and brothers, for helping me find my way, and helping me achieve what I have achieved so far; my grandparents and godparents for their support, with a special mention to Mamie Nanie. I would also like to mention some of the teachers who inspired and helped me discover my passion for sciences: M. Gilouppe, M. Douillet, Jean-Louis Clément, M. Deschamps, M. Capéran, Étienne Klein, as well as Christophe Barroy and Mr Waston.

Finally, I would like to thank the Rotary Foundation, and my extended Rotary Family (hoping that I am not forgetting anyone): Karen Swaim Babin, Jim and Susana Brown, Jean-Pierre Charlot, Nick and Rosemary Czifrik, Donna D'Agostino, Euiheon, Susan Frick, Thorsteinn Gislason, Hélène and Bernard Gourdeau, Klaus and Glenys Hachfeld, Keith Harris, Gene Hastings, Huria, Hans Ikier, Julia, Wilson Lee, Ranna Parekh, Phillippa, René, Jamie Santo, Sarah, and Stefano.

This research was supported in part by The Boeing Company under contract MIT-BA-GTA-1, and by the Air Force Research Lab award under contract F33615-01-C-1850.

Science sans conscience
n'est que ruine de l'âme.
François Rabelais (1494-1553)

Contents

1 Introduction **15**

1.1 Motivation 15

1.2 Challenges and Required Capabilities 16

1.3 Approach and Innovations 18

1.4 Example 20

1.5 Outline 22

2 Related Work **25**

2.1 Temporal Plan Execution 26

2.1.1 Previous Work in Dispatchable Plan Execution 26

2.1.2 Comparison with Our Approach 28

2.2 Qualitative, State-level Control 30

2.2.1 Model-based Execution 30

2.2.2 Hybrid Automata and Qualitative Control 34

2.3 Model Predictive Control 36

3 Problem Statement **39**

3.1 Multiple-UAV Fire-fighting Example 39

3.2 Definition of a *Plant Model* 40

3.2.1 Overall Definition of a *Plant Model* 41

3.2.2 Definition of a *Forbidden Region* 42

3.2.3 Definition of a *State Equation* 44

3.2.4 Application to Hybrid Automata 47

3.3 Definition of a *Qualitative State Plan* 50
3.3.1 Overall Definition of a *Qualitative State Plan* 51
3.3.2 Definition of a *Schedule* 52
3.3.3 Definition of an *Activity* 52
3.3.4 Definition of a *Temporal Constraint* 54
3.3.5 Definition of an *Objective Function* 55
3.3.6 Comparison with Metric Interval Temporal Logic 56
3.4 Definition of the *HMEx* Problem 58
3.4.1 Definition of *Hybrid Model-based Execution* 58
3.4.2 State Estimation 60
3.4.3 Control Sequence Generation 60
3.4.4 Comparison with Previous Work 63
3.5 Overall Approach to Solving HMEx 64
3.5.1 Infinite Horizon HMEx 65
3.5.2 Receding Horizon HMEx 66
3.5.3 Comparison with Related Work 71

4 Encoding the HMEx Problem as a Disjunctive Linear Program 75
4.1 Overall DLP Approach and Comparison with Previous Work 75
4.1.1 Motivation 76
4.1.2 Disjunctive Linear Programming Formalism 77
4.1.3 HMEx as a DLP 77
4.1.4 Relation with Previous Work 78
4.2 Encoding Infinite Horizon HMEx 78
4.2.1 Plant Model Encodings 79
4.2.2 Qualitative State Plan Encodings 87
4.3 Encoding Single-stage Limited Horizon HMEx 91
4.3.1 Revised Encoding for *End in* Activities 91
4.3.2 Guidance Heuristic for *End in* Activities 92

5 Constraint Pruning Policies **99**

5.1 Overall Constraint Pruning Framework 99

5.2 Plant Model Constraint Pruning 101

5.2.1 Forbidden Region Constraint Pruning 101

5.2.2 State Equation Constraint Pruning 104

5.2.3 State Initialization Constraint Pruning 104

5.3 Qualitative State Plan Constraint Pruning 105

5.3.1 Temporal Constraint Pruning 106

5.3.2 *Remain in* Constraint Pruning (Alg. 7 and 8) 113

5.3.3 *End in* Constraint Pruning (Alg. 9) 114

5.3.4 Guidance constraint pruning (Alg. 10) 116

6 Implementation and Performance Analysis **119**

6.1 Pseudocode for Sulu 119

6.1.1 Offline Algorithm (Alg. 11) 119

6.1.2 Online Receding Horizon HMEx Algorithm (Alg. 12) 120

6.2 Step-by-step Algorithm Demonstration 123

6.2.1 Initialization 124

6.2.2 First iteration ($t_0 = T_0 = 0$) 125

6.2.3 Second iteration ($t_0 = T_0 + n_t \cdot \Delta t = 10$) 127

6.3 Implementation on a UAV Testbed 128

6.4 Model-based Executive Performance Analysis on a More Complex Test Case 130

6.4.1 Description of the Test Case 131

6.4.2 Performance Analysis 133

7 Conclusion and Future Work **137**

7.1 Future Work 137

7.1.1 Improvements and Extensions of the HMEx Algorithm 137

7.1.2 Improvements on the Constraint Pruning Framework 141

7.1.3 Integration with an HTN Planner 144

7.2 Conclusion 144

A Proof of Equivalence between the Two State Equation Encodings in Eq. (4.8) and (4.9) 147

List of Figures

1-1 Qualitative state plan in the fire-fighting example. 20

1-2 Map of the environment for the multiple-UAV fire-fighting scenario, and initial partial trajectories computed by Sulu. 21

1-3 Modified trajectories computed by Sulu in order to adapt to a change in the environment. 22

2-1 Edge-splitting operation, applied to the edges in the STN graph (a), in order to construct the associated distance graph (b). 28

2-2 Traditional approach to designing embedded systems (left), versus the model-based approach (right), which elevates the level of interaction with the plant. 30

2-3 Block diagram of the model-based executive *Titan*. 31

3-1 Map of the terrain for the fire-fighting example. 40

3-2 Any general, non-convex region (A) can be approximated by a finite union of linearized, convex regions (B and C). *This figure was taken from [34].* 42

3-3 Example of a forbidden region $\mathcal{P}_S$ in the UAV fire-fighting scenario. . 43

3-4 Thermostat hybrid automaton. [23] 48

3-5 Qualitative state plan in the fire-fighting example. 51

3-6 Block diagram of a hybrid model-based executive. 59

3-7 Timeline illustrating the single-stage, limited horizon HMEx problem at time t_0 (Def. 15). 68

3-8 Information flow diagram for the single-stage, limited horizon HMEx problem at time t_0 (Def. 15). 69
3-9 Sulu's receding horizon hybrid controller. 70

4-1 Rectangular no-fly-zone in the UAV fire-fighting example. 83
4-2 **a)** Forbidden region corresponding to values of the velocity smaller than the minimum allowed value; **b)** Linearized version of the forbidden region. 84
4-3 Derivation of a *start in* activity from an *end in* activity. 88
4-4 Derivation of a *go through* activity from an *end in* activity. 88
4-5 Guidance heuristic for an *end in* activity involving a particular aircraft α_j. 94
4-6 Example of a guidance heuristic for a fire-fighting UAV. 95
4-7 Guidance heuristic used in [10]. 96
4-8 Simple example showing a limitation of the approach used in [10]. . . 97
4-9 Our guidance heuristic correctly guides the plant around the forbidden region. 98

5-1 Computation of the distance graph: each arc in the qualitative state plan (a) is split into two arcs in the distance graph (b). 106
5-2 Example of an implied temporal constraint. 107
5-3 Example of a state plan where the pruning policy in Alg. 5 entails infeasible schedules. 108
5-4 Illustration of the different cases in Alg. 5 (shaded areas are time periods outside of the current planning window $[t_0, t_{N_t}]$) : **a)** e_S has already been executed (line 1); **b)** e_S is out of reach within the current horizon (line 4); **c)** e_E has already been executed (line 7); **d)** e_E is out of reach within the current horizon (line 10). 109
5-5 Illustration of the different cases in Alg. 6: **a)** e has already been executed (line 1); **b)** e is out of reach within the current horizon (line 4).111

5-6 Illustration of the different cases in Alg. 7: **a)** The activity is completed (line 1); **b)** The activity is being executed (line 4); **c)** The activity will start beyond t_{N_t} (line 7); **d)** The activity will start within t_{N_t} (line 10). 112
5-7 Temporal propagation of a change in a unary temporal constraint. . 114
5-8 Illustration of the different cases in Alg. 9: **a)** e_E has already occurred (line 1); **b)** e_E will be scheduled within t_{N_t} (line 4); **c)** e_E will be scheduled beyond t_{N_t} (line 7). 115
5-9 Illustration of the different cases in Alg. 10: **a)** e_E will be scheduled within the horizon (line 1); **b)** e_S will be scheduled beyond the horizon (line 4). 116

6-1 Qualitative state plan in the fire-fighting example. 123
6-2 Map of the terrain for the fire-fighting example. 124
6-3 "Snapshots" of the schedule for the qualitative state plan in Fig. 3-5, at different steps in the algorithm execution: a) First iteration ($t_0 = T_0 = 0$), Step 3 (line 20); b) Second iteration ($t_0 = T_0 + n_t \cdot \Delta t = 10$), Step 3 (line 20); c) End of second iteration ($t_0 = T_0 + 2n_t \cdot \Delta t = 20$, line 33). The bold dots represent the values of $T(e)$ for each event e, and the segments represent the bounds $[T_e^{min}, T_e^{max}]$ on $T(e)$. 125
6-4 Trajectories computed at the first iteration (in bold: up to t_{n_t}; in light: between t_{n_t} and t_{N_t}) 126
6-5 Trajectories computed at the second iteration (in bold: up to t_{n_t}; in light: between t_{n_t} and t_{N_t}) 128
6-6 Architecture of the CloudCap testbed. *This picture was taken from [15]* 129
6-7 Map of the environment in the multi-UAV fire-fighting scenario. . . . 131
6-8 Qualitative state plan used for performance analysis. 132
6-9 Performance gain by constraint pruning. 134
6-10 Performance of Sulu. 135

Chapter 1

Introduction

1.1 Motivation

Robust, autonomous coordination of agile dynamic systems has application in a wide variety of fields. Much work has been done in the past few years on the coordinated control of autonomous unmanned air vehicles (UAVs) [3], which can be used, for instance, for surveillance and target identification on a widespread battlefield. Non-military applications can also be of high interest for these kinds of systems, such as situation assessment in case of an earthquake or a forest fire, in order to identify, localize and provide urgent assistance to populations waiting to be rescued. A fleet of small cooperative agile vehicles, like indoor/outdoor helicopters, or wheeled/legged ground vehicles, could be very useful to rescuers in the context of an accident where victims might be isolated in places of difficult or dangerous access, such as a building in fire.

Cooperative vehicles offer just one of a growing number of examples of agile systems for which autonomous coordinated control will enable unprecedented levels of capability or robustness. Agile systems with moving parts, such as robot arms and manipulators [46], represent another wide domain of application, as well as industrial robots performing robust coordinated tasks in a factory, or robots on orbit or on Mars, coordinating the assembly of a telescope or a Martian habitat. Moreover, a Martian life support system in itself can be considered a system with "agile", con-

tinuous dynamics, for which robust synchronized control is absolutely critical [25]. More generally, this also applies to industrial chemical plants, where the control of fast chemical processes requires robust coordinated control [67].

1.2 Challenges and Required Capabilities

Autonomous control over dynamic systems, such as the ones previously mentioned, raises a number of challenges, which need to be addressed by extending the current state of the art in autonomous control.

The first challenge comes from the fact that most of the aforementioned applications involve time-critical missions, in which the system under control, that is, the *plant*, must be able to operate under tight temporal constraints. For example, in the case of a team of UAVs cooperating to extinguish forrest fires, given estimates of the speed of progression of the fire fronts, it might be necessary to visit certain populated locations before a certain time, in order to look for victims who might get trapped between two fronts. It might also be necessary to monitor the evolution of the fire by flying over specific regions at regular time intervals.

This motivates the need for an autonomous controller that is able to provide some level of temporal coordination, by reasoning about deadlines and temporal constraints provided by the human operator, and by controlling the plant within these coordination constraints.

The second major challenge has to do with the continuous, dynamical nature of the plant. A fixed-wing aircraft, for instance, is a system with fast, agile, continuous dynamics, which requires continuous, low-level control. Current normal operations of such UAVs involves providing series of close waypoints that the aircraft must closely follow. It is possible for a human operator to provide such a low-level control over the plant; however, many applications such as forest fire suppression would require the coordination of several of these UAVs, in order to efficiently accomplish the mission within the given tight time constraints. As the number of aircraft in the team increases, low-level control of such fleets of cooperating UAVs quickly becomes

very costly in terms of the number of human operators necessary to operate the plant.

An autonomous controller for these systems should, therefore, elevate the level of interaction with the plant, so as to raise the ratio of the number of aircraft over the required number of human operators. The operator should be able to provide *task-level*, supervisory control of the plant, by specifying tasks that the system must perform, and that are described in terms of *abstract, qualitative states* that the plant must go through, hence delegating low-level control to the autonomous controller. Examples of such tasks in the fire-fighting scenario are high-level goals that each UAV must contribute to achieve, such as visiting a set of locations in minimum time. This elevates the interaction with the plant with regards to traditional control approaches, where the human operator must control the plant by providing low-level trajectories, rather than more abstract, qualitative tasks that the aircraft must perform.

Finally, a third important challenge raised by the autonomous control of these systems is that they often evolve in dangerous, dynamic and unpredictable environments. Forrest fire suppression is a very relevant example of a dynamic environment in which, although human operators might have access to models of forrest fires that enable them to predict the progression of the fire over a certain period of time, and to plan a strategy accordingly, it is often necessary to revise these strategies in real time, as the situation is evolving. A new fire front might appear that had not been anticipated, or the winds might change.

To deal with this third challenge, the autonomous controller must be able to exert robust, adaptive control over the dynamic system. It must be robust to low-level disturbances inherent to the dynamics of the plant, such as unpredictable wind conditions that affect the behavior and the responsiveness of the aircraft. This level of robustness can usually be provided by traditional, classical controllers, which are able to compensate for low-level disturbances in order to maintain the plant as close as possible to a given set point, or to make the plant follow series of set points. However, as mentioned before, an autonomous controller for the types of aforementioned autonomous systems should take in a more abstract, qualitative goal specification. It must therefore be able to adapt to higher-level unforeseen events. For instance,

the fire might be more widespread than initially estimated, requiring the UAVs to spend more time taking pictures of the fire in order to provide human operators with sufficient situation awareness. To this effect, controlling the plant by providing a qualitative description of the desired abstract plant state evolution delegates more control authority to the autonomous controller, and leaves it more room to adapt to high-level disturbances in real time.

1.3 Approach and Innovations

In order to provide those three main required capabilities, we propose a method to control the plant by specifying an abstract, task-level description of the desired plant state evolution, in the form of a *qualitative state plan*. We introduce a capability for robust, *model-based execution* of such qualitative state plans, for plants with continuous dynamics.

As we argued in the previous section, for time-critical applications such as multiple-UAV fire-fighting scenarios, it is particularly important for the autonomous controller to be able to provide temporal coordination to the plant. Furthermore, in order to elevate the level of interaction with the plant, the human operator should control the plant by providing task-level, qualitative descriptions of the abstract states the plant should go through, ignoring low-level control details. In particular, the operator must be able to reason in terms of the goals that must be achieved, rather than focus on the means to achieve these goals.

We provide this capability by writing the input to the autonomous controller in the form of a *qualitative state plan*. A qualitative state plan is a description of the mission the plant must accomplish, in terms of tasks, or *activities*, that the plant must perform in order to fulfill the mission. It is *qualitative*, in that the activities specify successive qualitative states the plant must go through. The qualitative state plan pieces these activities together in the form of a *temporal plan*. This temporal plan uses time constraints between activities in order to specify precedence constraints, and constraints on the duration of activities and the time at which they must be

performed. The temporal constraints in the plan are also described in a qualitative, temporally flexible manner; hence, the human operator is able to supervise the system by providing high-level temporal guidelines that the plant must follow.

In order to map the qualitative state plan to low-level control inputs that lead the plant through the specified qualitative state evolution, we introduce a *hybrid model-based executive*, called *Sulu*, which reasons from a *model* of the dynamics of the plant in order to design those command sequences. The model is *hybrid*, because rather than involving only discrete variables, it involves both continuous and discrete variables. The flexibility in the qualitative state plan provides margins of maneuver to the model-based executive, allowing it to robustly adapt to disturbances and unforeseen events. While a traditional, classical controller would only be able to adapt to low-level disturbances that move the plant state away from a provided set point or trajectory, our approach delegates more control authority to the model-based executive, which can adapt to higher levels of disturbances and unexpected events, by exploiting the flexibility in the qualitative state plan.

In order to adapt to disturbances, Sulu *interleaves planning and execution* over short planning windows, and continuously re-plans to take into account the latest knowledge of the state of the plant and the environment. This is done by framing the problem as an instance of *receding horizon control*; we call the resulting executive a *receding horizon, hybrid model-based executive*.

Reasoning iteratively over short planning windows also reduces the complexity of the problem, by allowing Sulu to look into the future only up to a limited *planning horizon*, rather than design control sequences for the whole qualitative state plan, which can soon become intractable when the plan involves a large number of activities. We also introduce novel *pruning policies* that Sulu uses to prune parts of the search space of all possible control sequences, in order to find an optimal control sequence for the plant in real time.

The resulting model-based executive we present in this thesis builds upon previous work in model-based programming [66], by applying the model-based paradigm to plants with continuous dynamics. By framing the problem that we are addressing

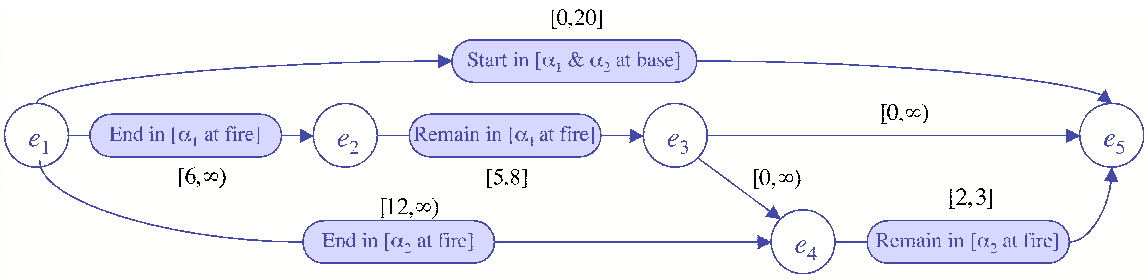


Figure 1-1: Qualitative state plan in the fire-fighting example.

as a temporal plan scheduling and execution problem, we are able to leverage off of previous work on execution of temporally flexible plans [7, 13, 49, 60, 61, 65]. This work addressed the problem of plan dispatching and execution, and provided robustness through the use of a receding horizon framework that interleaves planning and execution. This framework had previously been developed and applied to the control of plants with continuous dynamics, in the field of chemical process control [24, 51, 52, 54]. More recently, receding horizon control was also successfully applied to the control of multiple aircraft [10, 39, 57], through the use of Mixed Integer Linear Programming (MILP). Much of our work is inspired by this last field of research, although we use a *Disjunctive Linear Programming* (DLP) formalism [9, 34, 42, 43], rather than MILP. Another important difference is that we enable the human operator to control the plant by specifying desired qualitative regions in the state space that the plant should remain in or go through, as it is executing the qualitative state plan. This qualitative approach builds upon related work in *qualitative control* [37, 53], in which the plant dynamics are described in an abstract, qualitative manner, that enables the operator to reason in terms of abstract states that the plant should be in, rather than details of low-level control.

1.4 Example

In this section, we shortly present an example of application, in the context of a multiple-UAV fire-fighting scenario. This example will be presented in more detail in the next chapters, since we use it throughout this thesis to illustrate this work. In this

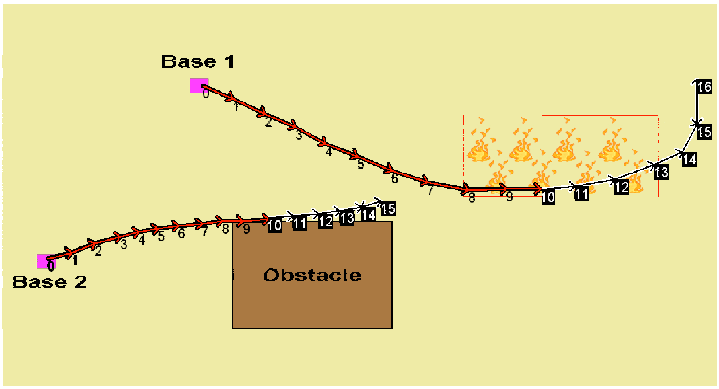


Figure 1-2: Map of the environment for the multiple-UAV fire-fighting scenario, and initial partial trajectories computed by Sulu.

example, the plant consists of two fixed-wing UAVs, which evolve in an environment (Fig. 1-2) that has a reported fire. The team of UAVs is assigned to collectively extinguish the fire, by navigating around forbidden regions (e.g. no-fly-zones) and by dropping water on the fire. The aircraft must also take pictures after the fire has been extinguished, in order to assess the damage. A natural language description for the mission's qualitative state plan is:

> *Aircraft* α_1 *and* α_2 *start at base stations* Base 1 *and* Base 2*, respectively.* α_1 *(a water tanker UAV) must reach the fire region and remain there for 5 to 8 time units, while it drops water over the fire.* α_2 *(a reconnaissance UAV) must reach the fire region after* α_1 *is done dropping water and must remain there for 2 to 3 time units, in order to take pictures of the damage. The overall plan execution must last no longer than 20 time units.*

This qualitative state plan can be represented by an acyclic, directed graph, illustrated in Fig. 1-1. The formalism and conventions used in this graphical representation will be introduced in Section 3.3.

As presented in Fig. 1-2, Sulu designs trajectories for the two UAVs up to a limited horizon of 15 time steps (the numbers next to each waypoint in the trajectories are

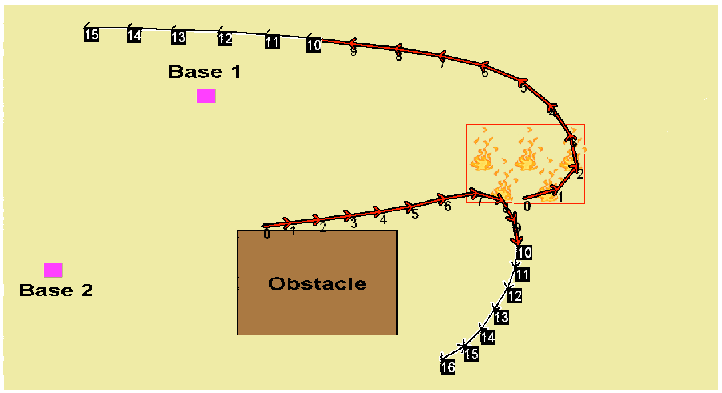


Figure 1-3: Modified trajectories computed by Sulu in order to adapt to a change in the environment.

the indexes of the corresponding time steps). Aircraft α_1 is scheduled to reach the fire at time step 8, and to remain in the fire region for 5 time steps, which is consistent with the temporal constraints in the qualitative state plan.

However, while the aircraft are following these initial trajectories, new information is gathered about the environment (for instance, by analyzing incoming satellite data), and the fire turns out to be less spread out than originally foreseen (Fig. 1-3). As a consequence, the initial trajectory for aircraft α_1 no longer satisfies the qualitative state plan, because it would make α_1 remain over the fire during less than 5 time steps. Sulu is able to adapt to this unforeseen event by modifying the trajectories for the UAVs, in order for aircraft α_1 to remain over the fire region for at least 5 time steps. The new trajectories are uploaded to the UAVs at time step 10, and lead the team to completion of the mission.

1.5 Outline

The rest of this thesis is organized as follows. In Chapter 2, we first describe relevant previous work in the fields of model-based programming, temporal planning, and

model-predictive control and qualitative control, and how this work tackled some of the challenges that we mentioned in this introduction. We then present the problem statement and a multiple-UAV fire-fighting example, used throughout the thesis to illustrate our work, and we introduce our general approach to execution of temporal plans for hybrid systems (Chapter 3). In Chapter 4, we present in more detail the mathematical formalism we use to encode the qualitative state plan and the plant model. Chapter 5 introduces novel constraint pruning policies that enable us to achieve tractability and real-time execution. In Chapter 6, we present the details of our algorithm, a walkthrough example to illustrate our approach, and empirical results and a performance analysis. Finally, Chapter 7 concludes this thesis, and presents possible areas of future work.

Chapter 2

Related Work

In Section 1.2, we introduced three challenges that arise from autonomous control of agile, dynamical systems, and we presented three capabilities that an autonomous controller should deliver, in order to tackle these three challenges. First, the autonomous controller should be able to provide temporal synchronization of the plant, by generating control sequences that satisfy deadlines and temporal constraints provided by the human operator. In Section 2.1, we describe related work in temporal plan execution, which provides this capability, by framing the problem as a temporal plan execution problem. Second, in order to enable the human operator to deal at a high level with the continuous dynamics of the plant, the autonomous controller should elevate the level of interaction with the plant, so as to enable the human controller to specify the desired qualitative behavior of the plant in terms of state, rather than in terms of low-level control inputs. Previous work in model-based programming tackled this challenge in the context of discrete systems; we describe this model-based paradigm in Section 2.2, as well as previous work on hybrid automata and qualitative control, which relates to the plant models we use in this thesis in order to describe plants with continuous dynamics. Finally, the autonomous controller should be robust to disturbances and unforeseen events; in Section 2.3, we refer to previous work in model predictive control, which provides robustness by interleaving planning and execution.

2.1 Temporal Plan Execution

As argued in Section 1.2, in order to control agile, dynamical systems, such as a fleet of fire-fighting UAVs performing a time-critical mission, the human operator should be able to specify deadlines and temporal constraints that the autonomous controller must satisfy, when designing control sequences for the plant. In Section 1.3, we presented our overall approach to providing this capability, which consists of formulating the goal specification for the controller in the form of a temporal plan. In this section, we show how our approach relates to previous work on temporally flexible plans.

2.1.1 Previous Work in Dispatchable Plan Execution

In order to represent temporally flexible plans, [17] introduced the concept of a *Simple Temporal Network* (*STN*), which built upon previous work on representing qualitative and metric temporal constraints [4]. An STN consists of instantaneous *events*, linked by *simple temporal constraints*, in the form of a lower bound and an upper bound on the allowed time between two events. An STN is represented as an acyclic, directed graph, in which nodes stand for events, and the temporal constraints are represented by arcs between events, labeled with the corresponding time bounds. Events usually stand for timepoints corresponding to the beginning or the end of an *activity*, specifying a command that must be sent to the plant, or an action that the plant must perform. In this thesis, in order to represent temporally flexible plans for systems with continuous dynamics, we use *qualitative state plans* (Section 3.3). A qualitative state plan can be seen as an STN, in which the events stand for start and end events of activities, where activities specify an abstract, qualitative region of the state space that the plant should be in, rather than a low-level command that the plant should execute.

In order to execute a temporally flexible plan, one must generate a *schedule* for the plan, that is, an assignment of execution times for every event in the plan. Although such a schedule could be computed beforehand, in real-life applications, the schedule is computed on the fly, as the events are being executed. This allows the system to

take advantage of the temporal flexibility in the plan, by dynamically generating the schedule in order to be robust to execution uncertainty, such as an unexpected delay between the time when the plant is commanded to start an action, and the time when the action eventually starts. Rather than computing a complete schedule beforehand, the problem then consists of, given a selected execution time for some early events in the plan, determining a range of allowed execution times for the later events, such that the overall schedule is temporally consistent.

The solution introduced in [62] and later applied in [17, 49, 60, 61] is, whenever an event in the plan gets assigned a fixed execution time, to propagate the consequences to the other events, in order to compute updated ranges of allowed execution times for these events. As described in Section 5.3.1, we use the same approach, in order to compute bounds on the times at which events in the plan can be scheduled. These bounds are then used both to enforce temporal consistency, as in the aforementioned previous work, and also to determine whether or not an event may be scheduled within the current planning window; this information is used by the pruning policies presented in Section 5.3.

Since this temporal propagation process happens in real time, as events are being executed, this must be done efficiently. In [17, 49, 60, 61], this is done by pre-compiling temporal constraints in the plan, in a way that local, one-step inference suffices. For a given event e that has just been executed, one can then perform temporal propagation only to the immediate neighboring events in the STN, by going through the list of incoming and outgoing arcs for e, and operating a one-step propagation along these arcs. This compilation process consists in making explicit all lower and upper time bounds, between all pairs of events in the STN; the resulting plan is then called a *dispatchable plan*, in which, by definition, single-step temporal propagation to immediate neighboring events is sufficient.

The dispatchable plan is generated using a *distance graph*, which is obtained from the graph corresponding to the STN by applying the edge-splitting operation in Fig. 2-1 to all the edges in the STN graph. The distances between any pair of events in the STN are then computed by running an all-pairs shortest path algorithm on the

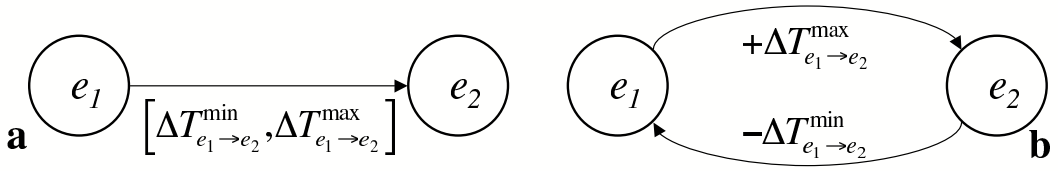


Figure 2-1: Edge-splitting operation, applied to the edges in the STN graph (a), in order to construct the associated distance graph (b).

distance graph; the resulting distances are stored in a new, fully connected distance graph. As previously mentioned, in this thesis, we use the exact same method in order to compute bounds on the times at which events may be scheduled (Section 5.3).

One issue that arises when using the method we just described, is that the dispatchable plan is fully connected, which can result in high computational loads in terms of memory usage. This can also lead to high computation times, since the propagation routine must effectively perform temporal propagation on all the events in the plan, whenever an event e is executed, since e is connected to every other event.

To prevent this computational blow up, the algorithms in [17, 49, 60, 61] prune temporal redundancy in the dispatchable plan, by only considering the arcs corresponding to tightest temporal bounds. This is preformed by running an edge-trimming routine on the dispatchable plan, in order to remove the edges that are *dominated*, that is, the edges that can be removed while keeping the plan dispatchable. The resulting plan is called a *minimal dispatchable plan*. This technique is not used in this thesis, but could be applied to our state plan pruning framework (Section 5.3); this is mentioned as future work in Section 7.1.2.

2.1.2 Comparison with Our Approach

As argued in Section 1.3 and in the previous paragraphs, our approach differs from traditional work on temporally flexible plans, in that we consider *qualitative state plans*, in which activities describe the desired qualitative behavior of the plant, in terms of abstract regions of the state space in which the plant state must remain, rather than low-level commands that the plant must execute. This approach is based

on a *model-based* paradigm, which we present in Section 2.2.1.

Another important difference between the work described in this thesis and previous work on dispatchable plan execution is the following. Approaches to temporal plan execution based on executing a dispatchable plan usually perform a partial evaluation of the schedule at compile time, by computing tight bounds on the time at which each event in the plan can be scheduled. The executive then commits to an execution time for each event at run time, just before executing the event. This is also the approach used in [26] in order to perform temporal plan execution for hybrid systems; we describe this work in more detail in Section 2.2.1.

In this thesis, we also compute a dispatchable graph in order to have access to bounds on the times at which events may be scheduled (Section 5.3.1); however, instead of delaying the choice of an execution time for each event until the event is about to be executed, our model-based executive commits to a complete schedule over a limited planning window, and then incrementally shifts the planning window, updating the schedule in order to respond to disturbances. This second approach is similar to the continuous planning approach used in [13], which generates plans over a limited planning window, and repairs the plans on the fly in response to unforeseen events.

Our choice of a continuous planning approach is motivated by the fact that our executive must generate control sequences for the plant, which are optimal with respect to some objective function, as defined in Section 3.3.5. A reactive approach would only work when the objective function is cumulative with time, in which case, minimizing the objective value at any point in time leads to minimizing the overall objective. In this thesis, however, we want to be able to specify more complex, non-time-cumulative objective functions, such as, in the fire-fighting UAV example, the minimization, over a given time window, of the number of episodes during which the UAV's absolute velocity remains greater than some threshold for more than 30 seconds. Such complex objective functions require the executive to be able to plan into the future, and could not be handled if the executive used a purely reactive approach.

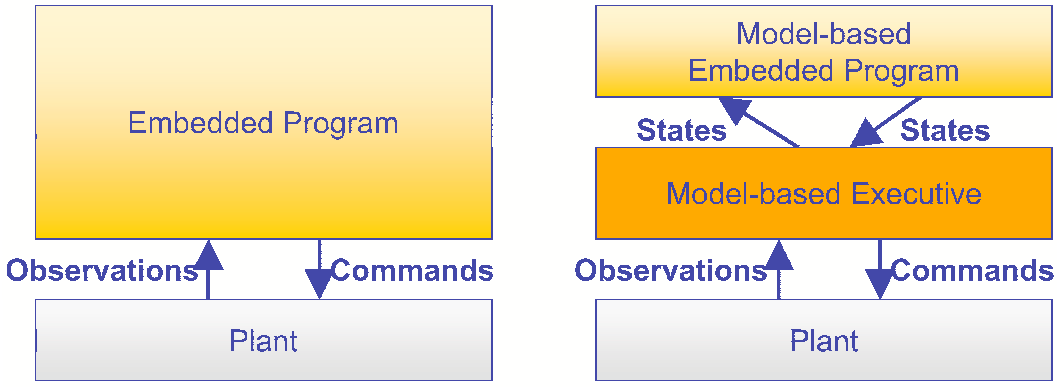


Figure 2-2: Traditional approach to designing embedded systems (left), versus the model-based approach (right), which elevates the level of interaction with the plant.

2.2 Qualitative, State-level Control

2.2.1 Model-based Execution

As mentioned in the introduction to this chapter, a second challenge raised by the autonomous control of agile, dynamical systems is the need for a high-level interaction between the human operator and the under-actuated plant with hidden state. This level of interaction should enable the operator to control the plant by specifying a desired abstract plant state evolution, rather than low-level commands that the plant must execute.

Model-based Execution of Discrete Systems

Previous work in model-based execution introduced a *model-based executive*, called *Titan* [66], which addresses this problem, by acting as an interface between the human operator and the plant. This is illustrated in Fig. 2-2. In the traditional approach to designing embedded systems (Fig. 2-2, left), the engineers or the human operators must design an embedded program that takes in observations directly from the plant, and generates low-level commands, in order to control the plant. This task can be difficult for complex systems, because the embedded program must be designed so as to infer the state of the plant from a very dense flow of sensor information, and

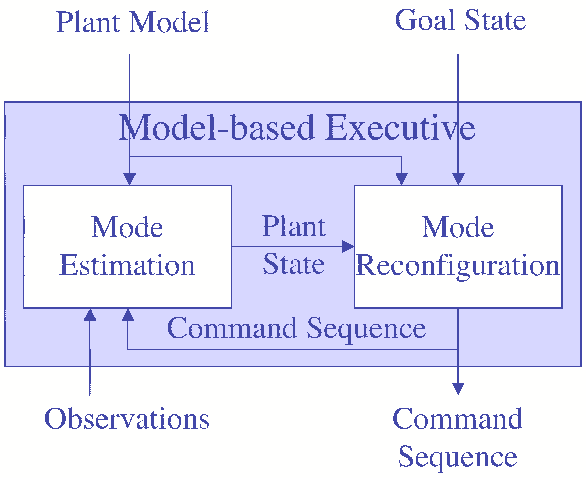


Figure 2-3: Block diagram of the model-based executive *Titan*.

design commands accordingly. On the contrary, in a model-based, embedded system (Fig. 2-2, right), a *model-based executive* provides low-level control and monitoring of the plant, by constantly estimating the plant state from the observations. The executive is also able to generate command sequences in order to achieve a given goal state. This elevates the interaction with the plant, by allowing the engineers to design embedded programs that effectively can read directly the values of state variables that, in reality, may not be directly observable. An embedded program can also directly "write" to state variables, that is, specify goal states for the plant, even if the state variables are, in reality, not directly controllable. It accomplishes all this by reasoning from a *model* of the plant.

In this thesis, we use the same model-based approach, by introducing *Sulu*, a model-based executive that takes in a description of the desired state evolution of the plant, in the form of a qualitative state plan, and generates low-level control sequences in order to execute that plan, given estimates of the plant state, inferred from the observations by reasoning over a model of the plant. In the following paragraphs, we describe Titan in more detail, and we compare it with Sulu.

Fig. 2-3 illustrates the two main functions of Titan. The *Mode Estimation* (*ME*) function takes in a sequence of observations and commands previously sent to the plant, and infers an estimate of the current plant state. This estimate is used by the

Mode Reconfiguration (*MR*) function in order to generate a sequence of commands that lead the plant to the desired goal state. These two functions can be mapped directly to the two functions in Sulu, *state estimation* and *control sequence generation*, illustrated in Fig. 3-6. As explained in more detail in Section 3.4.4, the two main differences between Titan and Sulu are that Titan was designed to reason on models that are purely discrete, and does not allow the human operator to specify temporal constraints on the plant. On the contrary, Sulu is able to control plants with continuous dynamics, and designs control sequences that satisfy the temporal constraints specified by the human operator, in the form of a qualitative state plan.

The plant models that the two model-based executives reason on are also consequently different. In order to map a goal state to a sequence of commands, and a sequence of observations to an estimate of the state of the plant, Titan reasons over a plant model that describes, at an intuitive engineering level, the different components in the plant, how each component works, and how all the components are connected to each other. The plant model is represented using *concurrent, constraint automata*, each automaton describing the laws that rule the transitions between discrete modes, for a given component. Sulu's plant model focuses more on describing the continuous dynamics of the plant, and the forbidden regions in the state space that the plant must avoid. This is presented formally in Sections 3.2 and 4.2.1. In Section 2.2.2, we also relate Sulu's plant model with previous work on hybrid automata and qualitative control.

Model-based Execution of Systems with Continuous Dynamics

Other related work in model-based programming introduced a model-based executive, called *Kirk*, which was designed for the coordination of cooperative mobile systems [32]. One of the major innovations with respect to Titan is that Kirk reasons on qualitative temporal constraints, in order to provide temporal synchronization over the plant. Kirk is comprised of a *temporal planner*, which generates a temporally flexible plan by selecting a hierarchy of possible contingencies, together with a *plan compiler* and *dispatcher*, which execute the temporal plan efficiently, while adapting

to disturbances.

To select among contingencies, Kirk uses a Hierarchical Temporal Network planner [33], which takes as an input a high-level description of the desired abstract activities that the plant must perform, written in *RMPL* (*Reactive Model-based Programming Language*). Kirk then expands these abstract activities into lower-level activities, using a database of *macros*, in order to generate a *Temporal Plan Network* (*TPN*). The TPN builds upon temporally flexible plans, similar to qualitative state plans; it involves events and flexible temporal constraints between events, as well as activities that the plant must perform. With respect to temporally flexible plan representation, a key difference with previous work described in Section 2.1.1 is that it encodes contingencies through an additional construct, called a `choose` *operator*, which specifies that one sequence of activities, among a set of redundant methods, must be selected for execution. Kirk must then select one sequence of activities for each `choose` operator in the TPN, such that the plan is temporally consistent [27], that is, such that there exists a schedule that satisfies all the temporal constraints in the TPN. [64] showed that this could be performed in a distributed fashion, and [63] introduced methods in order to produce optimal schedules.

In order to execute the resulting temporally flexible plan, Kirk uses a plan compiler and a dispatcher, in order to respectively compile the plan into a minimal dispatchable plan, and execute the dispatchable plan, using a method very similar to the one presented in Section 2.1.1. Plan compilation and dispatching are performed incrementally [58], in order to enable fast replanning, when Kirk needs to switch to a contingent plan to adapt to unforeseen events. [58] also showed that the dispatchable plans generated by the plan compiler could be executed on a distributed architecture, under communication limitations.

[63] modified Kirk's temporal planning algorithm in order to partially extend it to continuous plants, while the initial algorithm introduced in [33] could only handle discrete plants. Rather than reasoning over TPNs, [63] reasons over *RoadMap Temporal Plan Networks* (*RMTPNs*). RMTPNs extend TPNs by adding information about the desired physical location of the vehicles to the temporally flexible plan. This is

similar to the qualitative state plans used in this thesis, which can be seen as an extension of RMTPNs to more general, hybrid plants, since [63] only dealt with teams of vehicles, with purely continuous dynamics. The function of the temporal planning algorithm is then to compute control sequences that lead the vehicles through the desired locations, as well as a schedule for the plan. The algorithm uses *Rapidly exploring Random Trees* (*RRTs*) [41] that take into account the continuous dynamics of the vehicles. One limitation of this work is that, due to the randomized nature of the algorithm, there is no guarantee of optimality of the generated control sequences. In this thesis, we present a model-based executive that generates control sequences that are optimal over the considered planning window.

Related work in model-based execution [26] proposed an other approach to extending TPNs, in order to handle hybrid systems, rather than only discrete systems. [26] uses the same concept of *qualitative state plans* as in this thesis. The difference is that rather than encoding the plant dynamics using piecewise-linear functions, as described in Section 3.2.3, [26] uses a feedback-linearization technique that transforms the complex, hybrid plant into a set of much simpler linear systems, which can be controlled using classical PID controllers. The analogies with this work and our work are described in more detail in Section 3.5.3. As previously mentioned, [26] uses a partial evaluation approach similar to the one introduced in Section 2.1.2; this is possible because the objective function is time-cumulative, and consists of maximizing, at any point in time, the distance between the plant state and a reference trajectory, so as to maximize robustness to disturbances.

2.2.2 Hybrid Automata and Qualitative Control

In this section, we describe how related work on hybrid automata and qualitative control tackled the problem of encoding systems with continuous dynamics, through the use of models that involve continuous state variables, which evolve following state equations that depend on the current discrete *mode* of the plant, or on the current qualitative *operating region*. In this thesis, we use a plant model comparable with these types of *hybrid* models, which enables our model-based executive to control

systems with continuous dynamics.

Hybrid Automata

Previous work in hybrid systems [5, 23] extended the fully discrete models in [66] to models that are able to describe systems with continuous dynamics that depend on the modes of the plant's components. These models are described using *hybrid automata*, which extend the automata used in [66] by specifying that the different modes of a given component correspond to different differential equations, used to model the continuous dynamics of the plant. Such models are called *hybrid*, since they involve both discrete and continuous variables. We describe in more detail the concept of a hybrid automaton in Section 3.2.4, and we show that the plant model we use in this thesis is able to represent such hybrid automata.

Qualitative Reasoning and Control

The modeling of hybrid systems has also been tackled in the domain of *qualitative control* [35, 36]. In [35], the continuous dynamics of hybrid systems are described by dynamic equations that vary, depending on the current *operating region*. This is very similar to hybrid automata, whose continuous dynamics depend on discrete modes; however, while a hybrid automaton can only be in a single mode at a time, the operating regions in [35] describe regions of the state space over which the plant follows some qualitative behavior, and the operating regions are allowed to overlap. For a given point in the state space, a fuzzy set membership function is used in order to determine in which operating regions the point is located, and the dynamic equations at that point are computed by doing a weighted average of the dynamic equations associated with each operating region. In this thesis, we do not use this approach, since we assume that all operating regions are disjoint. Allowing operating regions to overlap could be an area of future work.

More recently, [37] introduced a framework in which, over each operating region, the dynamics of the plant are specified using a qualitative description, which only describes certain aspects of the system, leaving the remaining degrees of freedom

available for optimization at a lower level, according to criteria chosen by the designer. This means that, instead of using traditional differential equations to model the dynamics of the plant, the plant model uses *qualitative differential equations* (*QDEs*), which are a qualitative abstraction of a set of ODEs. This abstraction is useful in order to elevate the interaction between the human operator, to a level where the operator can specify desired qualitative regions of the state space that the plant should remain in or go through. The QDEs in the model are then used to describe the allowed transitions between such qualitative operating regions. As argued in [53], describing the desired plant behavior at an abstract, qualitative level delegates more control authority to the autonomous controller, enabling it to robustly react to larger disturbances.

The work described in this thesis is similar in that we also allow the user to specify the desired behavior of the plant at an abstract, qualitative level, in the form of a qualitative state plan. Our work, however, uses traditional ODEs to describe the dynamics of the plant, rather than QDEs. The use of QDEs would imply that there be a lower level of control, which would reason on a more accurate, quantitative model of the dynamics of the plant, in order to generate low-level control sequences. Our model-based executive was designed to provide this low-level control over the plant; hence, it requires quantitative models of the plant dynamics, rather than qualitative models, which would not be sufficient to generate low-level control sequences.

2.3 Model Predictive Control

As mentioned in the introduction to this chapter, the third capability that an autonomous controller for agile, dynamical systems must provide, is robustness to disturbances and unforeseen events. In Section 2.2, we argued that a qualitative description of the goal that the plant must achieve can be used in order to delegate more control authority to the executive, and give it more room to adapt to disturbances and unforeseen events. In this section, we show how the executive can take advantage of this authority in order to perform robust control of the plant. We relate our work

to previous work in *model predictive control* (*MPC*), which provides a framework for robust, low-level control of plants with continuous dynamics.

Model predictive control, also called *receding horizon control*, was first introduced in the field of Operations Research, in order to perform robust, optimal control of industrial chemical processes [21, 51, 54]. MPC formulates the problem of generating optimal control inputs for plants with continuous dynamics, by reasoning over receding, planning horizons. The controller designs precise, optimal control sequences over short planning windows, and uses a heuristic to compute an estimate of the cost of the control sequence beyond the horizon in order to reach a final goal state. This approach interleaves planning and execution, by reasoning over short planning windows, and replanning regularly, taking into account the latest information about the environment. This provides robustness to the autonomous controller, while enabling it to plan into the future in order to design optimal control sequences.

In this thesis, we use a very similar approach, by formulating the problem of generating optimal control sequences for the plant as a mathematical optimization problem, which we call *receding horizon, Hybrid Model-based Execution* (*receding horizon HMEx*, Section 3.5.2). Our approach consists of iteratively generating partial control sequences, by reasoning over short, receding planning windows.

Along with the use of MPC, other work in chemical process control [24, 52] used Linear Programming (LP) and Mixed Integer Linear Programming (MILP) approaches [11, 19] in order to encode the problem of generating optimal control sequences. In these approaches, the constraints in the problem, such as plant dynamics, or safe domains of operation, were encoded as linear constraints over a set of decision variables. LP or MILP optimizers would then be used in order to find optimal solutions to the problem. However, the domain of applications of MPC/MILP methods has long been restricted to plants with slow dynamics, due to the lack of computational power and efficient algorithms to solve MILPs. Typically, plants such as industrial chemical plants need model-predictive controllers that reason with time units on the order of minutes, or even hours. Computational power has dramatically increased since the eighties, and we now also have much more efficient MILP algo-

rithms; as a result, MPC/MILP methods recently became popular again, in order to control systems with much faster dynamics, for which model-predictive controllers need to run at frequencies on the order of 1Hz and above. In particular, these methods have been proved successful for the control of spacecraft [55, 56] and unmanned aerial vehicles (UAVs) [10, 39, 57].

The method presented in this thesis broadly builds upon this last thread of research, by using a linear programming technique to solve the problem of designing control sequences for plants with continuous dynamics, and reasoning over a receding horizon, following the MPC framework. However, our work differs in two main ways. As previously mentioned, we use a qualitative, model-based approach in which the plant is controlled through the use of qualitative state plans, that describe a desired abstract, qualitative plant state evolution with time. This approach allows for much richer goal specifications, providing flexible temporal synchronization of the plant. Furthermore, we use a Disjunctive Linear Programming formalism, rather than MILP, since [42] showed that significant improvements in solution time could be achieved by using conflict-directed algorithms on DLPs, over traditional MILP algorithms. We describe in more detail the differences between our approach and [39] throughout Chapter 4.

In this chapter, we showed how previous work had partially provided the three main capabilities that are necessary in order to perform autonomous control of agile, dynamical systems: temporal synchronization, high-level control of continuous dynamics, and robust adaptation to disturbances. We showed how our approach related to and built upon this previous work. In the next chapter, we more formally present the problem that we are trying to solve, and the overall receding horizon approach that we use to solve it.

Chapter 3

Problem Statement

Given a dynamic system (a *plant*), we define the *hybrid model-based execution* (*HMEx*) problem as the problem consisting of designing an optimal control sequence for the plant. This control sequence must satisfy the *plant model* and a given *qualitative state plan*, which specifies the desired evolution of the plant state over time, while minimizing some cost function.

In this chapter, we present a formal definition of the HMEx problem. We first introduce a simple example of a plant, consisting of multiple fire-fighting UAVs (Section 3.1), which we use to illustrate the problem. We then formally define a plant model (Section 3.2) and a qualitative state plan (Section 3.3). In Section 3.4, we define the HMEx problem, and we present the general definition of a *hybrid model-based executive*. Finally, we present our overall approach to solving the HMEx problem, and we introduce a corresponding hybrid model-based executive, called *Sulu* (Section 3.5).

3.1 Multiple-UAV Fire-fighting Example

The multiple-UAV fire-fighting example has a plant that consists of two fixed-wing UAVs, which evolve in an environment (Fig. 3-1) that has a reported fire. The team of UAVs is assigned to collectively extinguish the fire, by navigating around forbidden regions (e.g. no-fly-zones) and by dropping water on the fire. The vehicles must also take pictures after the fire has been extinguished, in order to assess the damage. A

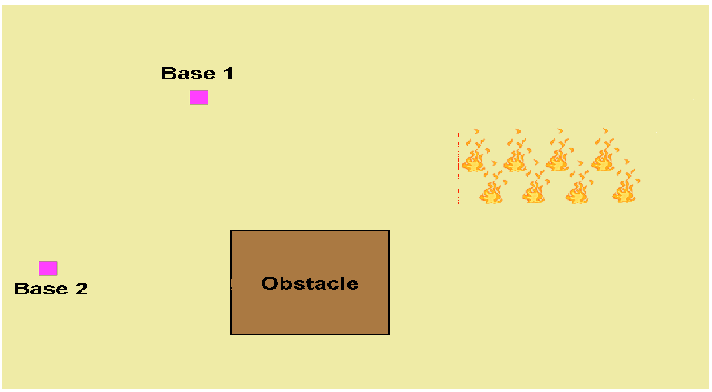


Figure 3-1: Map of the terrain for the fire-fighting example.

natural language description for the mission's qualitative state plan repeated from Chapter 1 is:

> *Aircraft* α_1 *and* α_2 *start at base stations* Base 1 *and* Base 2, *respectively.* α_1 *(a water tanker UAV) must reach the fire region and remain there for 5 to 8 time units, while it drops water over the fire.* α_2 *(a reconnaissance UAV) must reach the fire region after* α_1 *is done dropping water and must remain there for 2 to 3 time units, in order to take pictures of the damage. The overall plan execution must last no longer than 20 time units.*

Fig. 3-5 presents a graphical representation for this plan; the formalism and conventions used in this graphical representation will be introduced in Section 3.3.

3.2 Definition of a *Plant Model*

Recall that the overall motivation for a model-based executive, such as the one we introduce in this chapter, is to be able to autonomously design low-level control sequences for a plant, in order to satisfy a high-level qualitative description of the desired plant state evolution with time. In order to map a desired state evolution to a sequence of control inputs, the model-based executive reasons on a *model* of the plant.

3.2.1 Overall Definition of a *Plant Model*

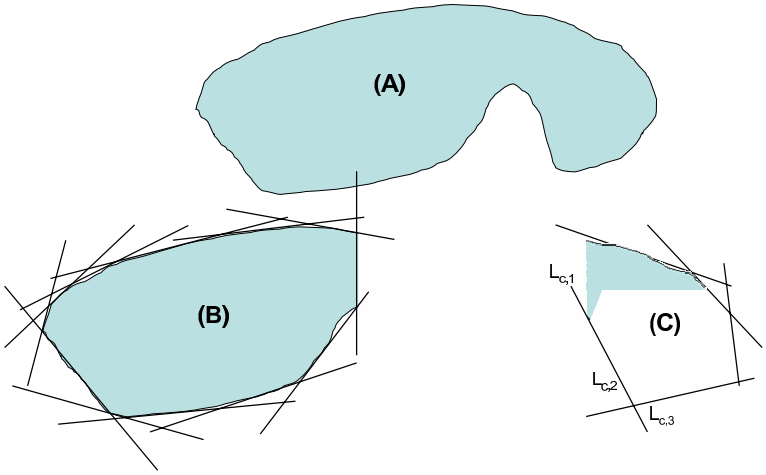
In the HMEx problem, a fundamental property of the type of plant we are considering is that it involves continuous dynamics, which must be controlled by sending sequences of continuous control inputs. Hence, the plant model that an executive reasons on should include a description of these dynamics, in terms of equations modeling the plant behavior with time, as a function of the control inputs. The model also includes constraints on the states that the plant can be in, and on the control inputs that are allowed; these constraints define the safe operating regions of the plant. This is defined formally in Def. 1.

Definition 1 *A* ***plant model*** $\mathcal{M} = \langle \boldsymbol{s}, \boldsymbol{u}, \mathcal{F}, \mathcal{SE} \rangle$ *consists of a vector* $\boldsymbol{s}(t)$ *of* ***state variables****, a vector* $\boldsymbol{u}(t)$ *of* ***input variables****, a set* $\mathcal{F}$ *of* ***forbidden regions*** *(Def. 2), defining unsafe operating conditions, and a set* $\mathcal{SE}$ *of* ***state equations*** *(Def. 3) describing the plant dynamics. The domain of the vector* $\langle \boldsymbol{s}, \boldsymbol{u} \rangle$ *is called the* ***state space*** $S = \mathbb{R}^n \times \mathbb{R}^m$.

The state vector $\mathbf{s}(t)$ is used to describe the state of the plant at time $t \in \mathbb{R}$, while the input vector $\mathbf{u}(t)$ stores the values of the control inputs exerted on the plant during the temporal interval $[t, t+1)$. As introduced in Chapter 1, *under-actuated plants* are plants for which $n > m$, which means that there are fewer control inputs than state variables. In our multiple-UAV example, $\mathbf{s}$ is the vector of 2-D Cartesian coordinates of the UAV positions and velocities, and $\mathbf{u}$ is the acceleration coordinates (Eq. (3.1)). Note that this plant is an example of an under-actuated plant, since $(n = 8) > (m = 4)$.

$$\begin{aligned} \mathbf{s} = &\ \langle x^{\alpha_1}, y^{\alpha_1}, v_x^{\alpha_1}, v_y^{\alpha_1}, x^{\alpha_2}, y^{\alpha_2}, v_x^{\alpha_2}, v_y^{\alpha_2} \rangle \\ \mathbf{u} = &\ \langle a_x^{\alpha_1}, a_y^{\alpha_1}, a_x^{\alpha_2}, a_y^{\alpha_2} \rangle \end{aligned} \tag{3.1}$$

A fundamental difference between this thesis and previous model-based executives, such as Titan [66], is that the state variables and input variables we use take on real values. For Titan [66], plants are modeled by concurrent constraint automata, each automaton corresponding to a state variable that only takes on a finite number


(A)
(B)
(C)
$L_{c,1}$
$L_{c,2}$
$L_{c,3}$