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

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

List of Figures

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", continuous 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

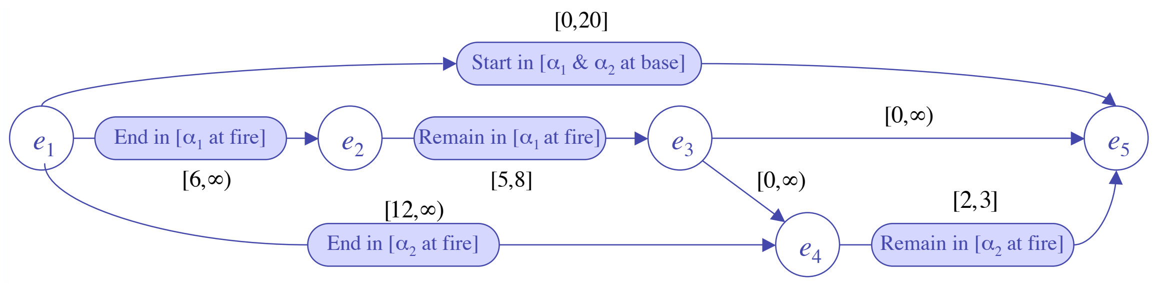


Figure 1-1: Qualitative state plan in the fire-fighting example.

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

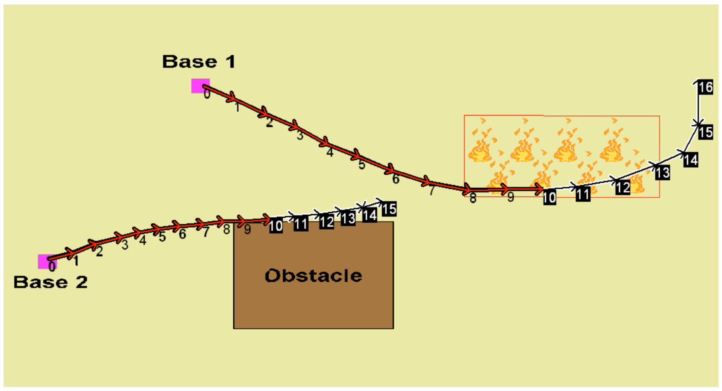


Figure 1-2: Map of the environment for the multiple-UAV fire-fighting scenario, and initial partial trajectories computed by Sulu.

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

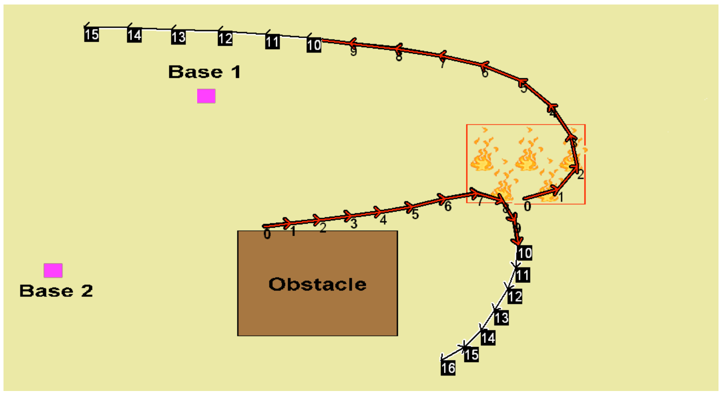


Figure 1-3: Modified trajectories computed by Sulu in order to adapt to a change in the environment.

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

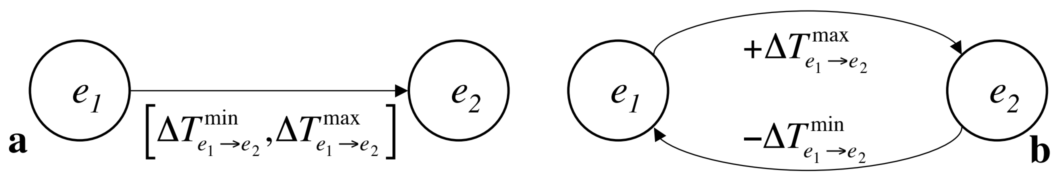
In order to represent temporally flexible plans, [17] introduced the concept of a *Simple Temporal Network* (*STN*), which built upon previous work on representing qualitative and metric temporal constraints [4]. An STN consists of instantaneous *events*, linked by *simple temporal constraints*, in the 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



a
e_1
$\left[\Delta T^{\min}_{e_1\to e_2}, \Delta T^{\max}_{e_1\to e_2}\right]$
e_2
e_1
$+\Delta T^{\max}_{e_1\to e_2}$
$-\Delta T^{\min}_{e_1\to e_2}$
e_2
b

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.

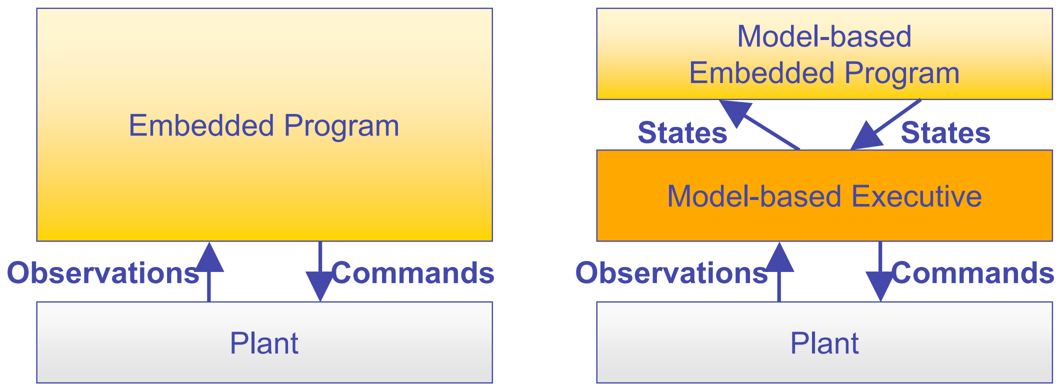


Figure 2-2: Traditional approach to designing embedded systems (left), versus the model-based approach (right), which elevates the level of interaction with the plant.

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

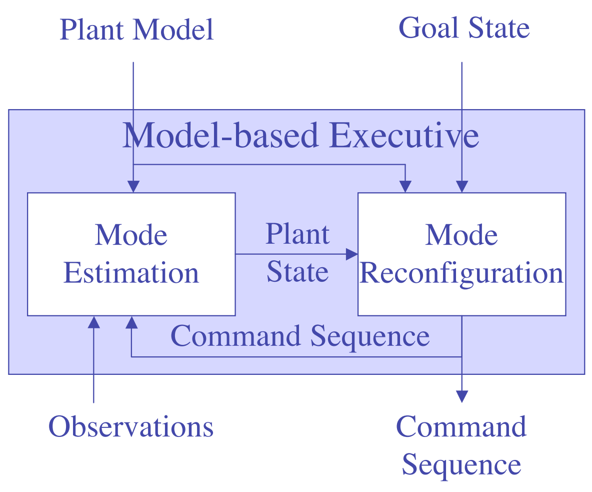


Figure 2-3: Block diagram of the model-based executive *Titan*.

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 *RoadMapTemporal 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 algorithms; 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

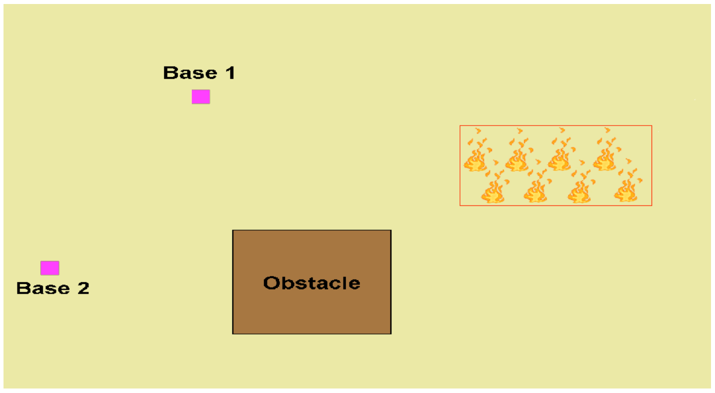


Figure 3-1: Map of the terrain for the fire-fighting example.

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

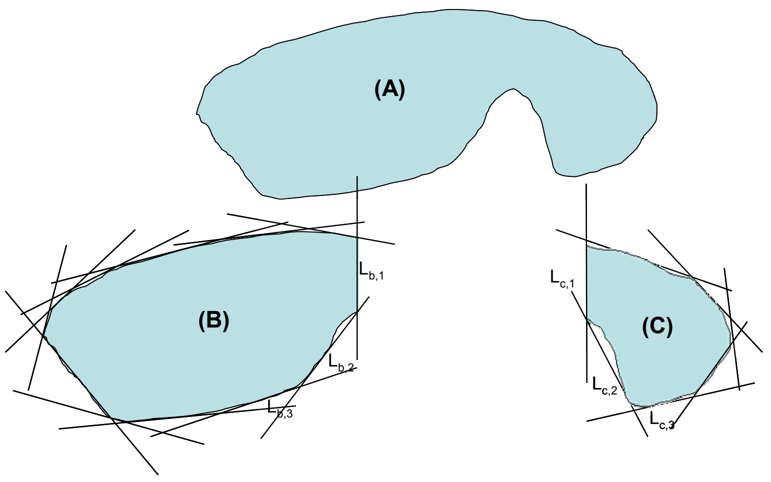


Figure 3-2: Any general, non-convex region (A) can be approximated by a finite union of linearized, convex regions (B and C). *This figure was taken from [34].*

of discrete values, corresponding to the discrete states (or *modes*) that the automaton can be in. In such a model, transitions between modes are instantaneous, and governed by *guards* involving state variables and control variables, which also only take on a finite number of discrete values. This type of fully-discrete model is not suited to describe systems with continuous dynamics, in which there are no such instantaneous transitions between discrete states, but rather a continuum of transitions between continuous states, where the transitions are governed by continuous input variables.

Fully-discrete models and fully-continuous models are two ends of a spectrum. Between these two ends are *hybrid models*, used to represent plants whose state is described by a combination of discrete and continuous variables. Although we define our model using only continuous state variables, the formalism we use is not restricted to fully-continuous plants; in Section 3.2.4 we describe how it can be used to model hybrid systems expressed in terms of *hybrid automata*.

3.2.2 Definition of a *Forbidden Region*

Recall (Section 3.2.1) that the models we use to describe the plant involve *forbidden regions*, describing disallowed operating regions in the plant state space S. In this section, we formally define the concept of a forbidden region (Def. 2).

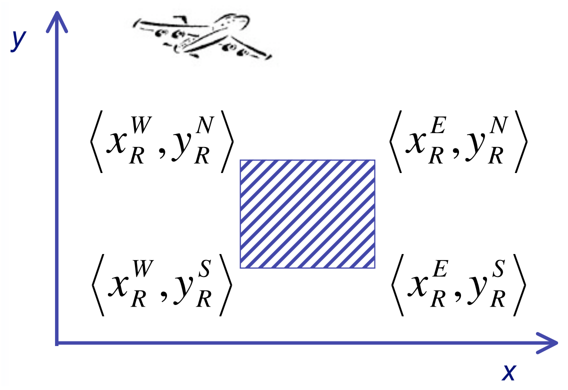

y
$\langle x_R^W, y_R^N \rangle$
$\langle x_R^E, y_R^N \rangle$
$\langle x_R^W, y_R^S \rangle$
$\langle x_R^E, y_R^S \rangle$
x

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

Definition 2 *A* ***forbidden region*** *from the set $\mathcal{F}$ is defined as a polyhedron $\mathcal{P}_S$ in the state space S [11], specifying a disallowed operating region in the plant state space (Eq. (3.2), where I is a finite set of integer indexes).*

$$\mathcal{P}_S = \left\{ \boldsymbol{x} \in S \;\middle|\; \bigwedge_{i \in I} \boldsymbol{a}_i^T \boldsymbol{x} \leq b_i \right\} \tag{3.2}$$

This definition of a forbidden region as a polyhedron is based on two assumptions: that the forbidden regions are convex, and that they are linear. The convexity assumption is motivated by the fact that any non-convex region can be approximated by a finite union of convex regions. The linearity assumption can be achieved by linearizing the borders of the regions (Fig. 3-2).

A simple example of a forbidden region in the fire-fighting UAV scenario is a no-fly-zone that the UAVs must avoid (Fig. 3-3 and Eq. (3.3), for UAV α_i). Other examples of forbidden regions in the multiple-UAV scenario are the bounds on nominal velocities and accelerations; these are presented in detail in Section 4.2.1. Forbidden regions may also be defined over both state variables and input variables; this enables us to define unsafe regions of the form "*The acceleration of a UAV should not exceed some safety value when the UAV is close to a mountain by a certain distance.*"

$$\mathcal{P}_S = \left\{ \langle \mathbf{s}, \mathbf{u} \rangle \in S \;\middle|\; \begin{array}{rcccrcl} & y^{\alpha_i} & \leq & y_R^N & \wedge & x^{\alpha_i} & \leq & x_R^E \\ \wedge & -y^{\alpha_i} & \leq & -y_R^S & \wedge & -x^{\alpha_i} & \leq & -x_R^W \end{array} \right\} \tag{3.3}$$

3.2.3 Definition of a ***State Equation***

As introduced in Def. 1, a plant model describes the dynamics of the plant through a set $\mathcal{SE}$ of *state equations*, which model the evolution of the state vector $\mathbf{s}(t)$ as a function of time and of the input vector $\mathbf{u}(t)$. In this section, we formally define a state equation (Def. 3), and we illustrate the definition using our multiple-UAV fire-fighting example.

General Case

Def. 3 introduces the definition of a *state equation*, in the general case.

Definition 3 *Given a time discretization $\langle t_0, t_1, \ldots \rangle \in \mathbb{R}^{\mathbb{N}}$, a* ***state equation*** *in $\mathcal{SE}$ is a piecewise-linear relation, expressing the value of a state variable s_k, at all time steps t_i, as a function of $\boldsymbol{s}$ and $\boldsymbol{u}$, at time step t_{i-1}. This is presented in Eq. (3.4), where $f_k : S \mapsto \mathbb{R}$ is a piecewise-linear function.*

$$\forall t_i, \quad s_k(t_i) = f_k(\boldsymbol{s}(t_{i-1}), \boldsymbol{u}(t_{i-1})) \tag{3.4}$$

The set of state equations $\mathcal{SE}$ can also be written in a compact form, presented in Eq. (3.5), where $\mathbf{F} = \langle f_1, \ldots, f_n \rangle$ is the vector function whose coordinates are the scalar piecewise-linear functions f_k.

$$\forall t_i \;\; \mathbf{s}(t_i) = \mathbf{F}(\mathbf{s}(t_{i-1}), \mathbf{u}(t_{i-1})) \tag{3.5}$$

We define a general piecewise-linear function over the state space S as a function for which there exists a finite partition $\mathcal{S}$ of S into subsets S_j, such that the function is linear over each of the subsets (Def. 4). We motivate the choice of a piecewise-linear representation of the plant dynamics by the fact that it enables us to model approximately any type of regular continuous plant dynamics, by locally linearizing the physical laws of the plant. More generally, any regular function over S can be approximated by a piecewise-linear function. Following the same argument as for the forbidden regions in $\mathcal{F}$ (Section 3.2.2), we can assume that the subsets S_j are polyhedra $\mathcal{P}_S$ of S.

Definition 4 *A function $f : S \mapsto \mathbb{R}$ is* ***piecewise-linear*** *if there exists a partition $\mathcal{S}_f$ of S such that, for all $S_j \in \mathcal{S}_f$, the restriction $f_{|S_j} : S_j \mapsto \mathbb{R}$ of f on S_j is linear. $\mathcal{S}_f$ is called an* ***underlying partition*** *of f.*

The following is an intuitive explanation for Eq. (3.5). Consider a partition $\mathcal{S}_{\mathbf{F}}$ that is an underlying partition for the piecewise-linear function $\mathbf{F}$. Then Eq. (3.5) can be rewritten as in Eq. (3.6), where the functions $\mathbf{F}_{|S_j}$ are linear. We provide an example in the fire-fighting UAV domain at the end of this section.

$$\forall t_i \; \forall j \;\; \langle \mathbf{s}, \mathbf{u} \rangle (t_{i-1}) \in S_j \Rightarrow \mathbf{s}(t_i) = \mathbf{F}_{|S_j}(\mathbf{s}(t_{i-1}), \mathbf{u}(t_{i-1})) \tag{3.6}$$

Note that in Eq. (3.4) and (3.5), we explicitly assume that the dynamic equations are time-invariant, namely, that $\mathbf{F}$ does not depend on t_i. We make this assumption in order to simplify the equations presented in this thesis; however, it can easily be relaxed by replacing Eq. (3.5) with Eq. (3.7), without fundamentally altering the methods and algorithms presented in the subsequent chapters.

$$\forall t_i, \quad \mathbf{s}(t_i) = \mathbf{F}_{t_i}(\mathbf{s}(t_{i-1}), \mathbf{u}(t_{i-1})) \tag{3.7}$$

Finally, in order for Eq. (3.5) to be a reasonable approximation of the plant dynamics, the time discretization must be sufficiently fine-grained. In particular, in the case of a regular time discretization $t_i = T_0 + i \cdot \Delta t$, the granularity of the time discretization Δt must be sufficiently small with respect to the plant dynamics.

Examples of State Equations

Eq. (3.8) presents $\mathcal{SE}$ for a single fire-fighting UAV α_1, where $\mathbf{F}$ is linear, rather than piecewise-linear, and is represented in matrix format. Δt defines the granularity of the time discretization [39]. This can be extended to the multi-UAV case by expressing $\mathbf{F}$ with block matrices whose primitive matrices correspond to each of the UAVs (Section 4.2.1).

$$\begin{bmatrix} x^{\alpha_1} \\ y^{\alpha_1} \\ v_x^{\alpha_1} \\ v_y^{\alpha_1} \end{bmatrix}(t_i) = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x^{\alpha_1} \\ y^{\alpha_1} \\ v_x^{\alpha_1} \\ v_y^{\alpha_1} \end{bmatrix}(t_{i-1}) + \begin{bmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \cdot \begin{bmatrix} a_x^{\alpha_1} \\ a_y^{\alpha_1} \end{bmatrix}(t_{i-1}) \quad (3.8)$$

In the previous example, the state equations we use to model the dynamics of the UAVs are purely linear, rather than piecewise-linear (Eq. (3.8)). However, to give a simple example to illustrate Eq. (4.9) in the more general piecewise-linear case, consider a UAV model in which the x-component of the velocity v_x saturates once it reaches the value $v_x^{\max}$. This can be expressed by the piecewise-linear relation in Eq. (3.9), where f is a linear function.

$$\bigwedge_{i=1\ldots N_t} \left\{ \begin{array}{rcl} v_x(t_{i-1}) \leq v_x^{\max} & \Rightarrow & v_x(t_i) = f(v_x(t_{i-1}), a_x(t_{i-1})) \\ \wedge \;\; v_x(t_{i-1}) > v_x^{\max} & \Rightarrow & v_x(t_i) = v_x^{\max} \end{array} \right\} \quad (3.9)$$

An underlying partition for this piecewise-linear relation is the set of polyhedra $\{\mathcal{P}_1, \mathcal{P}_2\}$, where $\mathcal{P}_1 = \{\mathbf{x} \in S \mid v_x \leq v_x^{\max}\}$ and $\mathcal{P}_2 = \{\mathbf{x} \in S \mid v_x > v_x^{\max}\}$. Note that $v_x > v_x^{\max}$ must be replaced by the approximate relation $v_x \geq v_x^{\max}$ for $\mathcal{P}_2$ to be a proper polyhedron.

One can draw a parallel between this piecewise-linear representation of the plant dynamics, and previous work on *hybrid systems* [5, 23]. Hybrid systems are systems whose continuous dynamics depend on discrete modes they can be in. For instance, in Eq. (3.9), the plant is described by two modes, identified by their respective constraints on v_x ($v_x(t_{k-1}) \leq v_x^{\max}$ and $v_x(t_{k-1}) \geq v_x^{\max}$). We describe in more detail in Section 3.2.4 how our plant modeling formalism can be used to encode *hybrid automata*, as defined in [23].

There are also many similarities with previous work in qualitative control [35, 36]. In [35], the plant's state space is decomposed into a set of qualitative *operating regions*; however, these operating regions do not necessarily form a partition of the state space. On the contrary, two operating regions are allowed to overlap. Instead of using a strict set membership function for the operating regions, a fuzzy set membership function is used, and the state equation at a given pair $\langle \mathbf{s}, \mathbf{u} \rangle \in S$ is defined as the weighted average of the purely linear state equations over each operating region, where the weights are given by the values of the corresponding fuzzy membership functions at point $\langle \mathbf{s}, \mathbf{u} \rangle$.

3.2.4 Application to Hybrid Automata

In this section, we show how our definition of a plant model can be used to encode hybrid systems, described as *hybrid automata* [5, 23]. We illustrate our encoding using a thermostat example from [23]. Def. 5 is a definition of a hybrid automaton, adapted from [23].

Definition 5 *A* ***hybrid automaton*** *consists of the following components:*

- *A set $X = \{\sigma_1, \ldots, \sigma_n, \upsilon_1, \ldots, \upsilon_{n'}\}$ of real-valued variables, where the vector $\boldsymbol{\sigma} = \langle \sigma_1, \ldots, \sigma_n \rangle$ is the vector of* ***state variables****, and $\boldsymbol{\upsilon} = \langle \upsilon_1, \ldots, \upsilon_{n'} \rangle$ is the vector of* ***input variables****.*

- *A finite directed multigraph (V, E), where the vertices in V correspond to the different* ***modes*** *that the plant can be in, and the edges in E correspond to* ***mode switches****, describing possible transitions between modes.*

- ***Invariant conditions****, represented by a node labeling function that associates, to each mode in V, a predicate on X, that is, a set of linear equalities or inequalities on the state variables, which must be satisfied when the plant is in that mode.*

- ***Flow conditions****, similarly represented by a node labeling function that associates, to each mode in V, a first-order ODE on the state vector $\boldsymbol{\sigma}$, as a function of the input vector $\boldsymbol{\upsilon}$.*

- ***Jump conditions****, represented by an edge labeling function that associates, to each mode switch in E, a predicate on X (or* ***guard****) that must be satisfied for the mode transition to be allowed.*

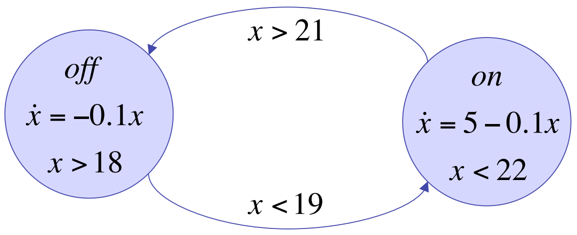


Figure 3-4: Thermostat hybrid automaton. [23]

An example of a hybrid automaton is the thermostat automaton in Fig. 3-4. The continuous state variable x represents the temperature, whose evolution with time depends on the discrete mode of the thermostat. If the thermostat is *off*, x evolves following the flow condition $\dot{x} = -0.1x$, and if it is *on*, the flow condition is $\dot{x} = 5 - 0.1x$. The invariant conditions in this hybrid plant specify that the thermostat may only be *off* if the temperature is above 18 ($x > 18$), and *on* if it is below 22 ($x < 22$). Finally, the jump conditions specify that the plant may transition from *off* to *on* only if $x < 19$, and from *on* to *off* if $x > 21$.

To represent a hybrid automaton as a plant (according to Def. 1), we introduce a *mode variable* m, whose domain corresponds to the different modes of the automaton. For instance, for the thermostat automaton, $m = 0$ corresponds to the plant being in the *off* mode, while $m = 1$ corresponds to the thermostat being *on*. Note that this temporarily contradicts our initial assumption (Section 3.2.1), stating that all variables in the plant model are continuous. We will show in Section 4.2.1 how we can remove this contradiction, by relaxing the domain of m to $\mathbb{R}$, while using a underlying partition for the state equations that effectively constrains m to take on values from a discrete domain.

We then set the input vector $\mathbf{u}(t_k)$ of the plant to the input vector $\boldsymbol{\upsilon}(t_k)$ of the hybrid automaton, augmented with the mode variable $m(t_k)$. The state vector $\mathbf{s}(t_k)$ is presented in Eq. (3.10), where $\tilde{\boldsymbol{\sigma}}(t_k)$, $\tilde{\boldsymbol{\upsilon}}(t_k)$ and $\tilde{m}(t_k)$ store respectively the values of $\boldsymbol{\sigma}$, $\boldsymbol{\upsilon}$ and m at the previous time step t_{k-1}. This is enforced by introducing the state equation $\forall k = 1 \dots N_t \ \langle \tilde{\boldsymbol{\sigma}}, \tilde{\boldsymbol{\upsilon}}, \tilde{m} \rangle (t_k) = \langle \boldsymbol{\sigma}, \boldsymbol{\upsilon}, m \rangle (t_{k-1})$. The reason we augment the state vector with a history of the values of the state and input variables at the previous time step, is to be able to encode the jump conditions as forbidden regions in the state space, as we describe in a later paragraph.

$$\mathbf{s}(t_k) = \langle \boldsymbol{\sigma}(t_k), \tilde{\boldsymbol{\sigma}}(t_k), \tilde{\boldsymbol{\upsilon}}(t_k), \tilde{m}(t_k) \rangle \tag{3.10}$$

We then encode the flow conditions as a piecewise-linear state equation, using a partition $\mathcal{S}$ of the state space S that splits S into subsets S_i corresponding to the different allowed values $m^1 \dots m^{|V|}$ of the mode variable m, as presented in Eq. (3.11).

$$S_i = \left\{ \langle \mathbf{s}, \mathbf{u} \rangle \in S \,\middle|\, m = m^i \right\} \tag{3.11}$$

The intuition behind this partition is presented in Eq. (3.12), for the thermostat example, where $\frac{x(t_k)-x(t_{k-1})}{\Delta t}$ is the time discretization for $\dot{x}(t_{k-1})$. Note the similarity with Eq. (3.9). Note also that in the general case, the flow conditions may not be purely linear; in that case, we use a sub-partition of $\mathcal{S}$, so that the flow conditions are linear on each subset S_i. This corresponds to piecewise-linearizing the flow conditions.

$$\bigwedge_{k=1\dots N_t} \left\{ \begin{array}{rcl} m_{k-1} = 0 & \Rightarrow & \frac{x(t_k)-x(t_{k-1})}{\Delta t} = -0.1x(t_{k-1}) \\ \wedge \quad m_{k-1} = 1 & \Rightarrow & \frac{x(t_k)-x(t_{k-1})}{\Delta t} = 5 - 0.1x(t_{k-1}) \end{array} \right\} \tag{3.12}$$

We encode invariant conditions as forbidden regions in the state space (Section 3.2.2). In the thermostat example, the invariant condition on the *off* mode specifies that $x > 18$ when the plant is in that mode. The corresponding forbidden region is the polyhedron $\mathcal{P}_S = \{\langle \mathbf{s}, \mathbf{u} \rangle \in S \,|\, m = 0 \ \wedge \ x \leq 18\}$, imposing that the thermostat is not allowed to be in the *off* mode ($m = 0$) when the temperature is below 18 ($x \leq 18$).

Finally, transitions between two modes are also encoded as forbidden regions in the state space. In the thermostat example, consider the transition from *off* ($m = 0$) to *on* ($m = 1$), guarded by the condition $x < 19$. This transition is represented by the polyhedron $\mathcal{P}_S = \{\langle \mathbf{s}, \mathbf{u} \rangle \in S \,|\, \tilde{m} = 0 \ \wedge \ m = 1 \ \wedge \ x \geq 19\}$, which specifies that a transition from *off* to *on* may **not** happen when the temperature is above 19. Note that, according to the definition of a hybrid automaton in Def. 5, a guard on a transition specifies whether or not the transition **may** happen. One could consider a slightly different model, in which the guard would specify whether or not the transition **must** happen; for instance, the guard $x < 19$ on the transition from *off* to *on* would then specify that this transition must happen every time the thermostat is *off* and the temperature is below 19 degrees. This would be encoded using the forbidden region $\mathcal{P}_S = \{\langle \mathbf{s}, \mathbf{u} \rangle \in S \mid \tilde{m} = 0 \ \wedge \ m = 0 \ \wedge \ x \leq 19\}$, which imposes that the plant is not allowed to remain in the state $m = 0$ when $x \leq 19$. This means that the plant must necessarily transition to the only other mode possible, which is $m = 1$. If the automaton had more than two modes, we would add one forbidden region $\mathcal{P}_S = \{\langle \mathbf{s}, \mathbf{u} \rangle \in S \mid \tilde{m} = 0 \ \wedge \ m = m^i \ \wedge \ x \leq 19\}$ for each mode $m^i \neq 1$, such that the plant is not allowed to transition to any other mode than $m = 1$.

3.3 Definition of a *Qualitative State Plan*

As introduced in Chapters 1 and 2, our approach to the robust, coordinated control of agile systems is a model-based approach, which elevates the interaction between the human operator and the plant, to the level where the operator is able to control the plant by specifying successive desired states the plant should be in, rather than low-level command sequences that it should execute. Furthermore, these state trajectories are specified at a qualitative level, as a series of feasible regions of the state space, rather than specific states. This way, the operator can focus on the goals to achieve, rather than on the means. This is especially important for agile systems with continuous dynamics, for which manually designing command sequences would be a very involved process. In addition, this level of abstraction offers the executive much greater latitude to adapt to disturbances, than specifying a concrete state trajectory.

We allow the human operator, or a high-level planner, to specify the desired sequences of goal states in a rich, temporally flexible manner, by formulating the intended plant state evolution in the form of a *qualitative state plan*. A qualitative state plan is a temporally flexible plan [17], with activities that specify qualitative constraints on the state of the plant. The flexibility in the plan, both in terms of time

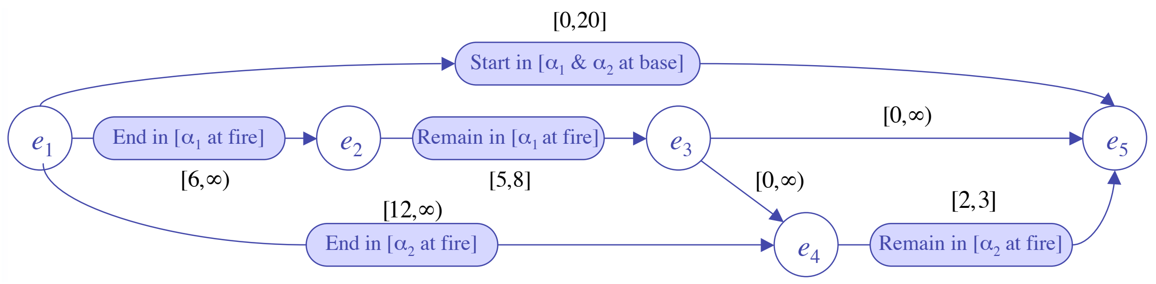


Figure 3-5: Qualitative state plan in the fire-fighting example.

constraints and in terms of state constraints, provides sufficient control authority to the autonomous controller to be able to robustly adapt to high-level disturbances and unforeseen events, as introduced in Section 1.2.

In this section, we formally define a qualitative state plan, and we illustrate our definitions with examples from the multiple-UAV fire-fighting scenario in Section 3.1.

3.3.1 Overall Definition of a *Qualitative State Plan*

Definition 6 *A* ***qualitative state plan*** $P = \langle \mathcal{E}, \mathcal{C}, \mathcal{A}, F \rangle$ *specifies a desired evolution of the plant state over time, and is defined by a set* $\mathcal{E}$ *of discrete events, a set* $\mathcal{A}$ *of* ***activities****, imposing constraints on the plant state evolution, a set* $\mathcal{C}$ *of* ***temporal constraints*** *between events, and an* ***objective function*** F*, which must be minimized.*

We illustrate a qualitative state plan diagrammatically by an acyclic directed graph in which the discrete events in $\mathcal{E}$ are represented by nodes, drawn as circles, and the activities as arcs with ovals. The qualitative state plan for the multiple-UAV fire-fighting mission example (Section 3.1) is shown in Fig. 3-5. The mission description involves five events:

1. The first event corresponds to the start event, at which both UAVs are at their respective base stations (event e_1 in Fig. 3-5).

2. The second event mentioned in the mission corresponds to aircraft α_1 reaching the fire, and starting to extinguish it (event e_2).

3. Event e_3 in Fig. 3-5 is associated with the time instant when α_1 has just finished extinguishing the fire.

4. Similar to event e_2, event e_4 happens when aircraft α_2 reaches the fire region and starts taking pictures of the damage.

5. Finally, the last event mentioned in the mission is the end event, at which α_2 has just finished taking pictures, and the mission is complete (event e_5).

3.3.2 Definition of a *Schedule*

As mentioned before, a fundamental feature of qualitative state plans is that they are *temporally flexible plans*, that is, the times at which each event in the plan must be executed are specified in a flexible manner, rather than strictly scheduled beforehand. As will be presented in Section 3.3.4, this means that temporal constraints between two events are specified by lower and upper bounds in the duration between the two events, rather than by a fixed imposed duration. This motivates the definition of a *schedule* for a qualitative state plan P.

Definition 7 *A* ***schedule*** *T for a qualitative state plan P is an assignment $T : \mathcal{E} \mapsto \mathbb{R}$ of execution times to all the events in P.*

Note that this is the same definition as that of a schedule for simple temporal networks [17]. Not all schedules are valid for a given qualitative state plan P; we will later introduce a criterion for schedule validity (Def. 13). An important point here is that, although in Section 3.2 we used a time discretization $\langle t_0, t_1, \ldots \rangle$ to express the dynamics of the plant, here a schedule is not restricted to take on values from this time discretization; as mentioned in Def. 7, it can take on any real values.

3.3.3 Definition of an *Activity*

Def. 6 defined a qualitative state plan P as a tuple $P = \langle \mathcal{E}, \mathcal{C}, \mathcal{A} \rangle$, where $\mathcal{A}$ is a set of *activities* that describe the desired evolution of the plant state throughout the execution of the state plan. In this sectoin, we formally define an activity, in terms of a time interval that imposes a given *state constraint* on the plant.

Definition 8 *An **activity** $a = \langle e_S, e_E, c_S \rangle$ has an associated start event e_S and an end event e_E. c_S is called a **state constraint** on the variable $\langle \boldsymbol{s}, \boldsymbol{u} \rangle$, and can take on one of the following four forms, where R_S, R_E, $R_\forall$ and $R_\exists$ are regions of the state space S, and T is a schedule for P:*

1. Start in state region R_S: $\langle \boldsymbol{s}, \boldsymbol{u} \rangle (T(e_S)) \in R_S$;
2. End in state region R_E: $\langle \boldsymbol{s}, \boldsymbol{u} \rangle (T(e_E)) \in R_E$;
3. Remain in state region $R_\forall$: $\forall t \in [T(e_S), T(e_E)], \langle \boldsymbol{s}, \boldsymbol{u} \rangle (t) \in R_\forall$;
4. Go through state region $R_\exists$: $\exists t \in [T(e_S), T(e_E)], \langle \boldsymbol{s}, \boldsymbol{u} \rangle (t) \in R_\exists$.

As can be seen in Def. 8, there are two main types of state constraints in a qualitative state plan: *durative* state constraints (*remain in*) and *instantaneous* state constraints (*start in*, *end in* and *go through*). We will show in Section 4.2.2 that all instantaneous state constraints can be expressed using an *end in* activity, such that we can only consider *remain in* and *end in* activities, without loss of expressivity.

In Fig. 3-5, activities are represented by ovals between two events. An example of a *Remain in state region* $R_\forall$ activity is the one between events e_2 and e_3, which specifies that the plant should remain in the state where α_1 is in the fire region. An example of an *End in state region* R_E is the activity between events e_1 and e_2, specifying that the plant should follow a state evolution such that α_1 is in the fire region at event e_2. An additional example of an activity is one specifying that vehicle α_1 should remain in a state where its velocity is limited by a small maximum value between events e_2 and e_3, while it is dropping water on the fire.

The concept of a state activity, which specify a state constraint on the plant, differs from previous work in hierarchical temporal planning [32] and classical STRIPS planning [18], in that the basic element in the plan is not a command or a control sequence that the plant must execute, but rather a qualitative description of the desired state of the plant, regardless of the control sequences that are used to achieve that state. Planners like Europa and HSTS incorporate state activities, but these are not qualitative, and the planner does not deduce the mapping from states to control actions from a model of the plant. As mentioned before, we use qualitative state plans to encode a goal specification for the plant, rather than the means to reach that goal. This allows the human operator or a mission-level planner to interact with the plant at a higher level, and gives more flexibility to the model-based executive to design low-level control sequences to achieve the qualitative high-level plan.

This flexibility in the goal specification is desirable for three reasons; first, it gives the executive more opportunities to succeed, by defining a wider set of feasible options, described in an abstract, qualitative manner. Second, it delegates more control authority to the executive, providing it with more options to recover from failure. And third, it also gives the executive more options to achieve greater optimality, since it has a wider set of feasible options from which to choose an optimal solution.

These three reasons are also the reasons presented in [53], in order to motivate the use of *qualitative control* (Section 2.2.2). In [53], the plant is described using a model that involves *qualitative differential equations* (QDEs), specifying the dynamics of the plant at an abstract, qualitative level. The QDEs describe allowed transitions between qualitative operating regions of the plant's state space, allowing the human operator to control the plant by specifying desired abstract, qualitative states the plant should be in, leaving the task of designing detailed control inputs to the low-level, autonomous controllers.

3.3.4 Definition of a *Temporal Constraint*

In a qualitative state plan, activities are composed together in order to describe valid plant state trajectories over time. This composition is achieved through the use of *temporal constraints* (Def. 9). As with the state constraints, these temporal constraints are also qualitative, in that they are temporally flexible: rather than specifying hard constraints on the times at which each event must be scheduled, they specify lower and upper bounds on the duration between pairs of events in the qualitative state plan.

Definition 9 *A* ***temporal constraint*** $c = \langle e_S, e_E, \Delta T^{min}_{e_S \to e_E}, \Delta T^{max}_{e_S \to e_E} \rangle$ *is a constraint, specifying that the duration from a start event* e_S *to an end event* e_E *be in the real-valued interval* $[\Delta T^{min}_{e_S \to e_E}, \Delta T^{max}_{e_S \to e_E}] \subseteq [0, +\infty]$.

Temporal constraints are represented diagrammatically by arcs between nodes, labeled with the time bounds $[\Delta T^{min}_{e_S \to e_E}, \Delta T^{max}_{e_S \to e_E}]$. In the fire-fighting, qualitative state plan, in Fig. 3-5, an example of a temporal constraint is the one represented by the arc $e_1 \to e_5$ labeled with the bounds $[0, 20]$; this specifies that the time interval between events e_1 and e_5 should last longer than 0 time unit and no longer than 20 time units. Note that we allow the upper bound to be infinite; for instance, the temporal constraint between events e_1 and e_4 specifies that e_4 should be scheduled at least 12 time units after e_1, but effectively specifies no upper bound. In the case when the lower bound is 0 and the upper bound is infinite, the temporal constraint is equivalent to a simple precedence constraint, such as the one between events e_3 and e_4, specifying that e_4 should be scheduled after e_3.

Note that this concept of a flexible temporal constraint is the same as the temporal constraints used in simple temporal networks [17].

3.3.5 Definition of an ***Objective Function***

As introduced in Def. 6, a qualitative state plan describes the desired behavior of the plant in time, both in terms of state and temporal constraints that must be satisfied, and in terms of an *objective function* F that must be minimized. In this section, we formally define the concept of objective function (Def. 10).

Definition 10 *Given sequences* $\boldsymbol{S} = \langle \boldsymbol{s}(t_0), \boldsymbol{s}(t_1), \ldots \rangle$ *and* $\boldsymbol{U} = \langle \boldsymbol{u}(t_0), \boldsymbol{u}(t_1), \ldots \rangle$ *of state variables and inputs variables, respectively, and given a schedule* T *for a set of events* $\mathcal{E}$*, an* ***objective function*** F *is a piecewise-linear, real-valued function over* $\boldsymbol{S}$*,* $\boldsymbol{U}$ *and* T.

Following the same argument as in Section 3.2.3, we can justify the piecewise-linearity assumption by the fact that any regular function over $\mathbf{S}$, $\mathbf{U}$ and T can be approximated by a piecewise-linear function. An example of an objective function in the fire-fighting UAV scenario (Eq. (3.1)) is a function that accounts for the amount of fuel required by the control sequence $\mathbf{U} = \langle \mathbf{u}(t_0), \mathbf{u}(t_1), \ldots \rangle$, where $\mathbf{u} = \langle a_x^{\alpha_1}, a_y^{\alpha_1}, a_x^{\alpha_2}, a_y^{\alpha_2} \rangle$. This cost function is given in Eq. (3.13).

$$F(\mathbf{S}, \mathbf{U}, T) = \sum_{k=0,1,\ldots} |a_x^{\alpha_1}(t_k)| \ + \ |a_y^{\alpha_1}(t_k)| \ + \ |a_x^{\alpha_2}(t_k)| \ + \ |a_y^{\alpha_2}(t_k)| \qquad (3.13)$$

Another example of a simpler objective function is $F(\mathbf{S}, \mathbf{U}, T) = T(e_{end})$, which can be used to minimize total plan execution time, by enforcing that the end event e_{end} of the qualitative state plan be scheduled as soon as possible. This is the objective function that we use in the multiple-UAV fire-fighting scenario.

Previous work in multi-vehicle model predictive control [57] tackled the problem of designing fuel-optimal control sequences for multiple vehicles. Their approach, briefly introduced in Section 2.3, is similar to ours, but has the key limitation that the goal specification is limited to a single goal position that each vehicle must reach, using a minimum amount of fuel. We extend this approach to richer goal specifications in terms of qualitative state plans, within a model-based framework.

3.3.6 Comparison with Metric Interval Temporal Logic

One can draw a parallel between our definitions of state and temporal constraints, and *metric interval temporal logic* (*MITL*), introduced in [6]. MITL is a logic-based language to model real-time systems, which uses a dense representation of time. An MITL-formula is formally defined in Eq. (3.14), where p is a proposition that is required to be true. $\mathcal{U}_I$ is the *until* operator, where I is a time interval of left bound $l(I)$ and right bound $r(I)$. By definition, $\phi_1 \, \mathcal{U}_I \, \phi_2$ is true at time t if and only if ϕ_1 is true at time t, and remains true for at least $l(I)$ and at most $r(I)$ time units, *until* ϕ_2 becomes true.

$$\phi := p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \, \mathcal{U}_I \, \phi_2 \tag{3.14}$$

The following additional operators can be constructed from Eq. (3.14): the *eventually* operator $\Diamond_I$, the *always* operator $\Box_I$, and the *unless* operator ${}_I\mathcal{W}$. $\Diamond_I \phi$ means that ϕ must *eventually* be true at some time instant within the time interval I, and is equivalent to $\texttt{true} \, \mathcal{U}_I \, \phi$. $\Box_I \phi$ means that ϕ must *always* be true during the time interval I, and is equivalent to $\neg\Diamond_I \neg\phi$. Finally, $\phi_1 \, {}_I\mathcal{W} \, \phi_2$ means that ϕ_1 must be true at all times during the time interval I, *unless* ϕ_2 becomes true before the end of interval I, in which case ϕ_1 may only be true until ϕ_2 becomes true. $\phi_1 \, {}_I\mathcal{W} \, \phi_2$ is equivalent to $\neg((\neg\phi_2) \, \mathcal{U}_I \, (\neg\phi_1))$.

We can then encode activities from the qualitative state plan as MITL-formulae, using the previous operators. We illustrate this on two examples: *end in* and *remain in* activities.

- Consider an *End in state region* R_E activity, starting at event e_S and ending at event e_E, associated with a temporal constraint on events e_S and e_E specifying that the duration between the two events should be within a time interval I. Then, the state constraint on this activity imposes that the MITL-formula $\Diamond_I(\langle \mathbf{s}, \mathbf{u} \rangle \in R_E)$ be true at time $T(e_S)$. This formula means that the state constraint should *eventually* be verified at some point within interval I.

- Consider a *Remain in state region* $R_\forall$ activity, starting at event e_S and ending at event e_E, associated with a temporal constraint with time interval I. Then, the state constraint imposes $(\langle \mathbf{s}, \mathbf{u} \rangle \in R_\forall) \, \mathcal{U}_I \, \texttt{true}$ at time $T(e_S)$, which means that the state constraint must be enforced *until* `true` is enforced, which may happen during the time interval I.

This completes our definitions for *qualitative state plan* and *plant model*; in the next section (Section 3.4), we use these to define the model-based execution problem.

3.4 Definition of the ***HMEx*** Problem

In the introduction to this chapter, we introduced informally the *Hybrid Model-based Execution* (*HMEx*) problem as the problem of designing an *optimal control sequence* for a plant with continuous dynamics, that generates a plant state evolution satisfying a qualitative state plan (Section 3.3). In this section, we formalize this problem, and define its solution, through a *hybrid model-based executive* that maps the qualitative state plan to an optimal control sequence, by reasoning on a model of the plant (Section 3.2).

3.4.1 Definition of ***Hybrid Model-based Execution***

Def. 11 presents the formal definition of a hybrid model-based executive. As discussed later in this section, the main innovation of this executive with respect to previous model-based executives is that it is able to control plants with continuous dynamics, by reasoning from a model of the plant dynamics, and by designing an optimal, complete, continuous control sequence, in order to execute the input qualitative state plan.

Definition 11 *Given an initial state $\boldsymbol{s}(t_0)$, a plant model $\mathcal{M}$, a qualitative state plan P, and its corresponding objective function F, the* ***Hybrid Model-based Execution (HMEx) problem*** *consists of incrementally generating, for every time step t_i, an optimal control sequence $\langle \boldsymbol{u}(t_0), \ldots, \boldsymbol{u}(t_i) \rangle$, given a sequence of observations $\langle \boldsymbol{o}(t_0), \ldots, \boldsymbol{o}(t_i) \rangle$. The final resulting control sequence $\boldsymbol{U} = \langle \boldsymbol{u}(t_0), \boldsymbol{u}(t_1), \ldots \rangle$ must verify that there exist a state trajectory $\boldsymbol{S} = \langle \boldsymbol{s}(t_0), \boldsymbol{s}(t_1), \ldots \rangle$ and a schedule T for the qualitative state plan P, such that the following propositions hold:*

1. *The objective function $F(\boldsymbol{S}, \boldsymbol{U}, T)$ (Def. 10) is minimized;*

2. *The state sequence $\boldsymbol{S}$ is consistent with the sequence of observations $\boldsymbol{O} = \langle \boldsymbol{o}(t_0), \boldsymbol{o}(t_1), \ldots \rangle$ and the input sequence $\boldsymbol{U}$ (Section 3.4.2);*

3. *$\langle \boldsymbol{S}, \boldsymbol{U} \rangle$ satisfies the plant model $\mathcal{M}$ (Def. 12);*

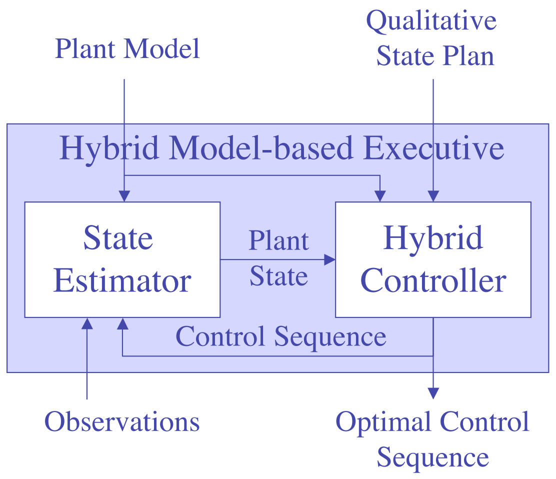


Figure 3-6: Block diagram of a hybrid model-based executive.

4. $\langle \boldsymbol{S}, \boldsymbol{U}, T\rangle$ *satisfies the qualitative state plan* P *(Def. 13);*

5. $\langle \boldsymbol{S}, \boldsymbol{U}, T\rangle$ *is complete (Def. 14).*

As illustrated in Fig. 3-6, the HMEx problem can be decomposed into two different sub-problems:

1. The problem of estimating the current state of the plant, at each point in time, by reasoning from the plant model, the prior observations from the plant, and the control inputs previously sent to the plant. This task is accomplished by the *state estimator*, and corresponds to Item 3.4.2 in Def. 11: the state estimator is responsible for enforcing that the state sequence $\mathbf{S}$ be consistent with the observations.

2. The problem of incrementally generating a complete, optimal control sequence in order to execute the input qualitative state plan, based on the plant model and on estimates of the state of the plant. This task is performed by the *hybrid controller*, and corresponds to Items 1, 3, 4 and 5 in Def. 11.

In the following subsections, we present in more detail each subproblem: state estimation (Section 3.4.2), and control sequence generation (Section 3.4.3). We also present the definitions for the concepts of *consistency*, *satisfaction* and *completeness* introduced in Def. 11.

3.4.2 State Estimation

As shown in Fig. 3-6, the function of the state estimator is to provide estimates of the plant state to the hybrid controller, by reasoning on the plant model, observations from the plant, and previous control sequences sent to the plant. Hence, the state estimator performs a mapping from an observation sequence $\mathbf{O} = \langle \mathbf{o}(t_1), \mathbf{o}(t_2), \ldots \rangle$ and a control sequence $\mathbf{U} = \langle \mathbf{u}(t_0), \mathbf{u}(t_1), \ldots \rangle$, to a maximum likelihood state sequence $\mathbf{S} = \langle \mathbf{s}(t_1), \mathbf{s}(t_2), \ldots \rangle$, given the plant model (Def. 11, Item 2).

In this thesis, we do *not* provide a solution for the state estimation problem. In order to enable Sulu to estimate the state of the plant, the definition of a plant model, presented in Section 3.2, would need to be extended in order to account for observations. However, related work on hybrid estimation [12, 25] provides a framework for the state estimator. [12] reasons on a model of the plant, described using *probabilistic, hybrid, concurrent automata* (*PHCAs*), in order to continuously track the most probable plant state trajectories.

In our work, we abstract away the state estimation problem, by assuming that the hybrid controller has a unique, non-probabilistic knowledge of the state of the plant at all times. This is equivalent to making a maximum likelihood assumption, that is, assuming that the most likely trajectory provided by the state estimator is the true trajectory. For instance, the hybrid estimator described in [12] continuously tracks the state of the plant, by maintaining a finite set of trajectory estimates. In that case, Sulu would only consider the estimate with the highest cumulative density, and use its mean as the estimate of the plant state.

3.4.3 Control Sequence Generation

As introduced in Section 3.4.1, given estimates of the plant state provided by the state estimator, the task of the hybrid controller is to iteratively generate a control sequence $\mathbf{U}$, for which there exist a state sequence $\mathbf{S}$ and a schedule T, such that:

1. *(Item 1 in Def. 11)* The control sequence $\mathbf{U}$ is *optimal*, that is, it minimizes the objective function $F(\mathbf{S}, \mathbf{U}, T)$ (Def. 10);
2. *(Item 3 in Def. 11)* The state sequence $\mathbf{S}$ is the sequence that the plant follows when it executes the control sequence $\mathbf{U}$, that is, $\langle \mathbf{S}, \mathbf{U} \rangle$ satisfies the plant model $\mathcal{M}$ (Def. 12);
3. *(Item 4 in Def. 11)* The state and input sequences and the schedule satisfy the qualitative state plan P (Def. 13);
4. *(Item 5 in Def. 11)* The state and input sequences and the schedule are complete. (Def. 14);

We formally define the three concepts of *plant model satisfaction*, *qualitative state plan satisfaction* and *completeness* in the following definitions (Def. 12, 13 and 14, respectively).

Definition 12 (Plant model satisfaction) *Given a plant model $\mathcal{M} = \langle \boldsymbol{s}, \boldsymbol{u}, \mathcal{SE}, \mathcal{F} \rangle$, we say that a sequence $\langle \boldsymbol{S}, \boldsymbol{U} \rangle = \langle \langle \boldsymbol{s}, \boldsymbol{u} \rangle(t_0), \langle \boldsymbol{s}, \boldsymbol{u} \rangle(t_1), \ldots \rangle$* ***satisfies plant model*** *$\mathcal{M}$ if the following two propositions hold:*

1. *$\langle \boldsymbol{S}, \boldsymbol{U} \rangle$ satisfies all the state equations in $\mathcal{SE}$ (Def. 3 and Eq. (3.5)):*

$$\forall k = 1, 2, \ldots \quad \boldsymbol{s}(t_k) = \boldsymbol{F}(\boldsymbol{s}(t_{k-1}), \boldsymbol{s}(t_{k-1}))$$

2. *$\langle \boldsymbol{s}(t_k), \boldsymbol{u}(t_k) \rangle$ always remains outside of all forbidden regions $\mathcal{P}_S \in \mathcal{F}$:*

$$\forall k = 0, 1, \ldots \ \forall \mathcal{P}_S \in \mathcal{F} \quad \langle \boldsymbol{s}(t_k), \boldsymbol{u}(t_k) \rangle \notin \mathcal{P}_S$$

Recall (Def. 1) that a plant model $\mathcal{M}$ imposes two types of constraints on the state and input variables: constraints related to the state equations describing the plant dynamics, and constraints corresponding to forbidden regions of the state space. As

Table 3.1: Example of a temporally consistent schedule T for the qualitative state plan in Fig. 3-5.

e	e_1	e_2	e_3	e_4	e_5
$T(e)$	0	6	12	12	14

defined in Def. 12, a sequence $\langle \mathbf{S}, \mathbf{U} \rangle$ of state and input variables *satisfies the plant model* if it satisfies these two types of constraints, at all time steps in the sequence.

Definition 13 (Qualitative state plan satisfaction) *Given a qualitative state plan $P = \langle \mathcal{E}, \mathcal{C}, \mathcal{A}, F \rangle$, a state sequence $\boldsymbol{S}$, an input sequence $\boldsymbol{U}$, and a schedule T, we say that $\langle \boldsymbol{S}, \boldsymbol{U}, T \rangle$* ***satisfies qualitative state plan*** *P if the following two propositions hold:*

1. *The schedule T is* ***temporally consistent****, that is, T satisfies all the temporal constraints in $\mathcal{C}$, as defined in Def. 9;*

2. *$\langle \boldsymbol{S}, \boldsymbol{U}, T \rangle$ does not violate any of the activities in $\mathcal{A}$, where the concept of* activity violation *is defined as the opposite of* activity satisfaction *(Def. 8).*

Intuitively, a tuple $\langle \mathbf{S}, \mathbf{U}, T \rangle$ *satisfies qualitative state plan P* if it is consistent both with the state constraints imposed by the activities, and with the temporal constraints between these activities.

Consider the example qualitative state plan in Fig. 3-5. An example of a temporally consistent schedule for this plan is presented in Table 3.1. This schedule is one that minimizes total plan execution time; however, it does not correspond to any tuple $\langle \mathbf{S}, \mathbf{U}, T \rangle$ satisfying all the activities in the qualitative state plan, as will be illustrated in Section 6.2.

Note that the concept of temporal consistency for a schedule relates directly to that of temporal consistency for simple temporal networks [17]: a simple temporal network is said to be *temporally consistent* if it possesses a schedule that satisfies all the simple temporal constraints in the network.

Definition 14 (Completeness) *A tuple* $\langle \boldsymbol{S}, \boldsymbol{U}, T\rangle$ *is* ***complete*** *if and only if the* ***spans*** *of the state sequence* $\boldsymbol{S}$ *and input sequence* $\boldsymbol{U}$ *cover the whole schedule* T *(Eq. (3.15)), where the span of a sequence* $\boldsymbol{X} = \langle \boldsymbol{x}(t_0^{\boldsymbol{X}}), \ldots, \boldsymbol{x}(t_{N_{\boldsymbol{X}}}^{\boldsymbol{X}})\rangle$ *is defined as the time interval* $[t_0^{\boldsymbol{X}}, t_{N_{\boldsymbol{X}}}^{\boldsymbol{X}}]$. *An incomplete tuple* $\langle \boldsymbol{S}, \boldsymbol{U}, T\rangle$ *is said to be* ***partial****.*

$$\left\{ \begin{array}{rll} & t_0^{\boldsymbol{S}} & \leq \min_{e\in\mathcal{E}}(T(e)) \leq \max_{e\in\mathcal{E}}(T(e)) \leq \; t_{N_{\boldsymbol{S}}}^{\boldsymbol{S}} \\ \wedge & t_0^{\boldsymbol{U}} & \leq \min_{e\in\mathcal{E}}(T(e)) \leq \max_{e\in\mathcal{E}}(T(e)) \leq \; t_{N_{\boldsymbol{U}}}^{\boldsymbol{U}} \end{array} \right\} \tag{3.15}$$

Intuitively, a tuple $\langle \mathbf{S}, \mathbf{U}, T\rangle$ is *complete* if and only if both the state sequence $\mathbf{S}$ and input sequence $\mathbf{U}$ start at or before the time $T(e_{start})$ at which the start event e_{start} of the qualitative state plan is scheduled, and end at or after the time $T(e_{end})$ at which the end event e_{end} is scheduled. This way, the two sequences cover the complete qualitative state plan execution.

For instance, consider the example schedule T presented in Table 3.1, for which $T(e_{start}) = T(e_1) = 0$ and $T(e_{end}) = T(e_5) = 14$. Consider also a state sequence $\mathbf{S} = \langle \mathbf{s}(t_0^{\mathbf{S}}), \ldots, \mathbf{s}(t_{N_{\mathbf{S}}}^{\mathbf{S}})\rangle$, and an input sequence $\mathbf{U} = \langle \mathbf{u}(t_0^{\mathbf{U}}), \ldots, \mathbf{u}(t_{N_{\mathbf{U}}}^{\mathbf{U}})\rangle$. Then the tuple $\langle \mathbf{S}, \mathbf{U}, T\rangle$ is complete if and only if Eq. (3.16) holds.

$$\left\{ \begin{array}{rll} & t_0^{\mathbf{S}} & \leq 0 \leq 14 \leq \; t_{N_{\mathbf{S}}}^{\mathbf{S}} \\ \wedge & t_0^{\mathbf{U}} & \leq 0 \leq 14 \leq \; t_{N_{\mathbf{S}}}^{\mathbf{U}} \end{array} \right\} \tag{3.16}$$

3.4.4 Comparison with Previous Work

The concept of a model-based executive, as illustrated in Fig. 3-6, is similar to previous model-based executives, such as Titan [66]. As presented in Section 2.2.1, Titan consists of a *Mode Estimation* component (*ME*), and a *Mode Reconfiguration* component (*MR*). Similar to our state estimator, the function of ME is to provide estimates of the state of the plant, from observations and from the history of commands previously sent to the plant. The function of MR is analogous to that of our hybrid controller, in that it designs command sequences to be sent to the plant, in order to reach a provided goal state. Both components use a plant model to map observations to plant states, and abstract goal states to low-level commands.

However, as suggested by the names of ME and MR, Titan reasons on *modes* that the plant can be in, where a mode is one of a finite number of discrete states, described in the plant model. The commands sent to the plant are discrete commands that incur instantaneous transitions between modes. In contrast, we control plants with continuous dynamics, whose states are described by continuous variables, and which require sequences of continuous control inputs.

The second fundamental difference with Titan is in the input goal specification. While Titan takes as an input a single goal state and must design a command sequence such that the plant reaches that goal state, our hybrid model-based executive accepts a much richer goal specification, in terms of a qualitative state plan. The role of the executive is then to design a sequence of continuous control inputs, in order to follow the desired plant state evolution, within the temporal constraints of the state plan.

3.5 Overall Approach to Solving HMEx

Previous model-based executives, such as Titan [66], focus on reactively controlling discrete-event systems. This approach is not applicable to temporal plan execution of systems with continuous dynamics; our model-based executive, called *Sulu*, uses a different approach (presented in Section 3.5.2) that consists of continuously planning into the future, in order to perform optimal, safe execution of temporal plans.

One simple approach to planning into the future consists of starting from a known initial position (provided by the state estimator), and generating a complete, optimal control sequence that will lead the plant through a complete sequence of states that satisfies the qualitative state plan. We call this approach *infinite horizon HMEx*, and we present it in Section 3.5.1.

However, solving the whole HMEx problem over an infinite horizon presents two major challenges. First, the problem is intractable in the case of long-duration missions. Second, it requires perfect knowledge of the qualitative state plan and the environment beforehand; this assumption does not always hold in real-life applications, such as our fire-fighting scenario, in which the size and shape of the fire area might precisely be known only at a late time during the execution of the mission (Section 1.4). Furthermore, the executive must be able to compensate on-the-fly for possible approximations or errors in the plant model. Section 3.5.2 presents how we use a Receding Horizon framework (introduced in Section 2.3) in order to provide this real-time adaptation functionality, and also to make the problem tractable.

3.5.1 Infinite Horizon HMEx

In Section 3.4, we defined the HMEx problem as the problem of designing an optimal control sequence that drives the plant through a sequence of states satisfying the desired plant state evolution specified in the qualitative state plan. In the *infinite horizon* version of this problem, the model-based executive generates a complete control sequence over a quasi-infinite horizon, in an offline phase, and then sends the control sequence to the plant for execution. In this context, the planning horizon corresponds to the *span* of the generated state and input sequences, and is assumed to be sufficiently long to produce a complete, optimal control sequence that satisfies the qualitative state plan.

As suggested in the introduction of this chapter, and as argued in [57] in the similar context of multi-vehicle path planning, solving infinite horizon HMEx raises a few important issues. First, in the case of long-duration missions, the model-based executive needs to design control sequences over a planning horizon that is sufficiently long, for the generated plant state sequences to be complete. Since the qualitative state plan is temporally flexible, it is not possible to know beforehand how long it will take to complete it. In this case, we need to adopt a conservative approach, by choosing a very large planning horizon. This results in significant computational costs that tend to make the infinite horizon HMEx intractable. A less conservative approach would consist of choosing a reasonably small planning horizon, and augmenting it iteratively if the model-based executive discovers that it is too short to complete the qualitative state plan, or that the best solution found with this small planning horizon is sub-optimal. This approach, however, can also be very computationally intensive, if it takes several iterations to find a planning horizon that is sufficiently long to generate complete, optimal sequences.

Furthermore, the infinite horizon approach lacks robustness, since it involves computing a potentially very long control sequence, and then blindly executing it, assuming that the plant follows the computed state trajectory. In practice, there are numerous causes that could prevent the plant from following that trajectory. This includes unforeseen forbidden regions in the state space that could make the planned trajectory infeasible, or disturbances that could make the plant deviate from its trajectory. There could also be errors or approximations in the plant model used to compute the control sequences, in which case the plant would not follow exactly the planned state trajectory.

The above reasons motivate the use of an approach to solving HMEx that interleaves planning and execution over short horizons. Planning over short horizons makes HMEx more tractable, and continuous replanning provides robustness to disturbances. We refer to this approach as *receding horizon HMEx*, which is presented in the next section.

This approach is very similar to related work described in Section 2.3, which applies receding horizon control to plants with continuous dynamics, such as industrial chemical plants [21, 24, 51, 52, 54], or teams of UAVs [10, 39, 57]. The most important distinguishing feature with respect this work is that we use receding horizon control in the context of the execution of a qualitative state plan, which is a much richer description of the goal that the plant must accomplish than the ones used in previous work. By describing the desired plant state evolution in an abstract, qualitative manner, our approach also delegates more control authority to the autonomous controller to adapt to disturbances and unforeseen events, as argued in Section 3.3.

3.5.2 Receding Horizon HMEx

As introduced in Section 3.5.1, full horizon HMEx has limited applicability, since it can quickly become intractable, and is not able to adapt to disturbances. In this section, we present a similar approach, based on *model predictive control*, that makes the problem more tractable, and provides robustness to disturbances.

Model predictive control (*MPC*, Section 2.3), also called *receding horizon control*, is a method introduced in the field of Operations Research, in order to control industrial chemical plants [21, 24, 51, 52, 54]. It was recently successfully applied to the low-level control of spacecraft [55, 56] and teams of UAVs [10, 39, 57]. MPC solves the control problem up to a limited *planning horizon*, and re-solves it when the plant reaches a shorter *execution horizon*. This method makes the problem tractable by restricting control sequence generation to a small planning window, and generates control sequences that are optimal over the planning window, and globally near-optimal.

In this section, we extend MPC to model-based execution of temporal plans for hybrid systems, by describing our *receding horizon, hybrid, model-based executive*, called *Sulu*. We formally define receding horizon HMEx as follows.

Definition 15 *Let $N_t \in [0, \infty)$ be the* ***planning horizon****, and $n_t \in (0, N_t]$ be the* ***execution horizon****. We use* ***single-stage, limited horizon HMEx at time*** *t_0 to refer to the problem of designing a partial control sequence $\langle \boldsymbol{u}(t_0), \ldots, \boldsymbol{u}(t_{N_t-1}) \rangle$, given:*

1. *A plant model $\mathcal{M}$;*
2. *A known plant state $\boldsymbol{s}(t_{-2n_t})$;*
3. *A history of observations for the plant $\langle \boldsymbol{o}(t_{-2n_t}), \ldots, \boldsymbol{o}(t_{-n_t}) \rangle$;*
4. *A history of control inputs $\langle \boldsymbol{u}(t_{-2n_t}), \ldots, \boldsymbol{u}(t_{-n_t-1}) \rangle$ that have already been executed by the plant, and the control sequence $\langle \boldsymbol{u}(t_{-n_t}), \ldots, \boldsymbol{u}(t_{-1}) \rangle$ that the plant is currently executing;*
5. *A qualitative state plan P, and a history $\tau = \{\langle e_i, T_{e_i} \rangle \mid T_{e_i} < t_0\}$ of events in the qualitative state plan that are scheduled before t_0;*

such that there exist a partial state trajectory $\langle \boldsymbol{s}(t_{-2n_t}), \ldots, \boldsymbol{s}(t_{N_t}) \rangle$ and a schedule T satisfying:

1. *The objective function $F(\langle \boldsymbol{s}(t_0), \ldots, \boldsymbol{s}(t_{N_t}) \rangle, \langle \boldsymbol{u}(t_0), \ldots, \boldsymbol{u}(t_{N_t-1}) \rangle, T)$ is minimized;*

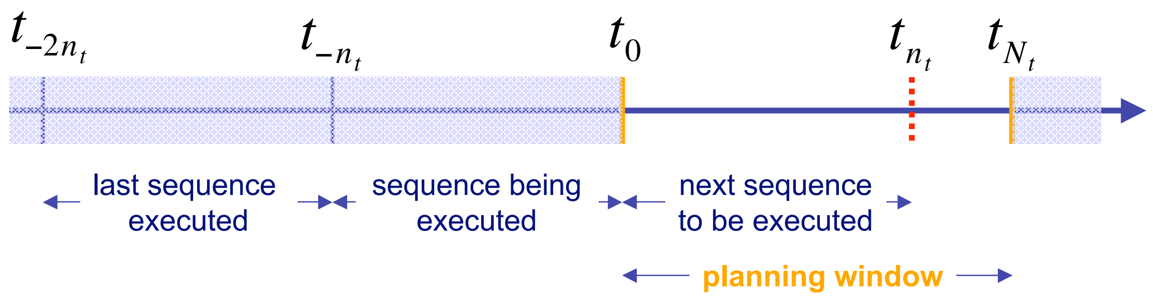


Figure 3-7: Timeline illustrating the single-stage, limited horizon HMEx problem at time t_0 (Def. 15).

2. *The state sequence* $\langle \boldsymbol{s}(t_{-2n_t}), \ldots, \boldsymbol{s}(t_{-n_t})\rangle$ *is a maximum likelihood trajectory estimate for the sequence of observations* $\langle \boldsymbol{o}(t_{-2n_t}), \ldots, \boldsymbol{o}(t_{-n_t})\rangle$*, the input sequence* $\langle \boldsymbol{u}(t_{-2n_t}), \ldots, \boldsymbol{u}(t_{-n_t-1})\rangle$*, and the plant model;*

3. $\langle\langle \boldsymbol{s}(t_{-n_t}), \ldots, \boldsymbol{s}(t_{N_t})\rangle, \langle \boldsymbol{u}(t_{-n_t}), \ldots, \boldsymbol{u}(t_{N_t-1})\rangle\rangle$ *satisfies the plant model* $\mathcal{M}$*;*

4. $\langle\langle \boldsymbol{s}(t_0), \ldots, \boldsymbol{s}(t_{N_t})\rangle, \langle \boldsymbol{u}(t_0), \ldots, \boldsymbol{u}(t_{N_t-1})\rangle, T\rangle$ *satisfies the qualitative state plan* P*;*

5. *The times at which past events are scheduled remain unchanged, that is:*

$$\forall \langle e_i, T_{e_i}\rangle \in \tau \quad T(e_i) = T_{e_i}$$

Definition 16 *Given a plant model* $\mathcal{M}$*, a qualitative state plan* P*, a sequence of observations* $\langle \boldsymbol{o}(T_0), \ldots\rangle$*, and an initial state* $\boldsymbol{s}(T_0)$*,* ***receding horizon HMEx*** *is the problem of iteratively solving single-stage, limited horizon HMEx at successive times* $t_0 = T_0, T_0 + n_t\Delta t, T_0 + 2n_t\Delta t \ldots$ *such that the tuple* $\langle\langle \boldsymbol{s}(T_0), \ldots\rangle, \langle \boldsymbol{u}(T_0), \ldots\rangle, T\rangle$ *is complete (Def. 14), where* Δt *is the granularity of the time discretization.*

Intuitively, the single-stage, limited horizon HMEx problem is the restriction of the general HMEx problem (Def. 11) over a limited planning window (Fig. 3-7). One important difference with general HMEx is that, since the spans of the state and input sequences are now limited, the sequences are no longer required to be complete (Item 5 in Def. 11), but rather are allowed to be only partial. We also require that

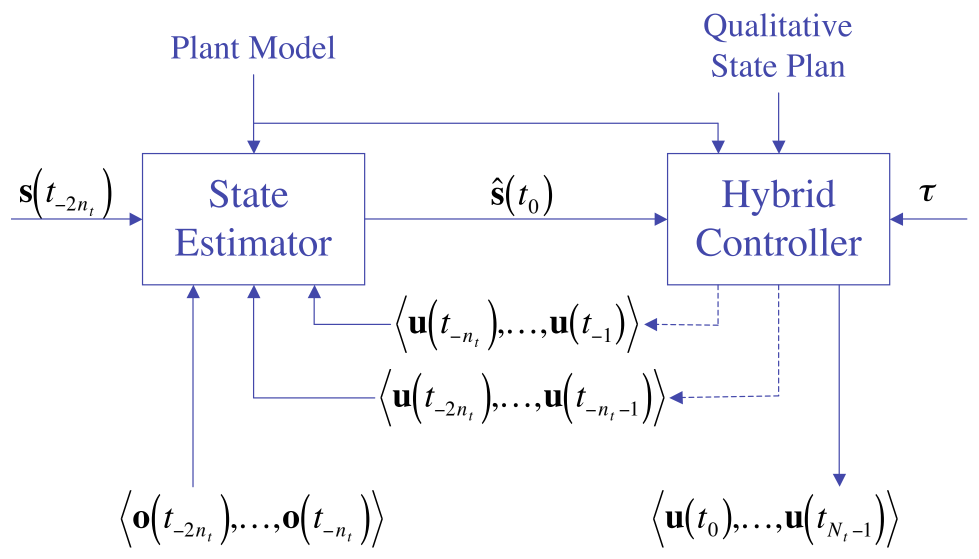

Plant Model
Qualitative State Plan
$\mathbf{s}(t_{-2n_t})$
State Estimator
$\hat{\mathbf{s}}(t_0)$
Hybrid Controller
τ
$\langle \mathbf{u}(t_{-n_t}),\dots,\mathbf{u}(t_{-1})\rangle$
$\langle \mathbf{u}(t_{-2n_t}),\dots,\mathbf{u}(t_{-n_t-1})\rangle$
$\langle \mathbf{o}(t_{-2n_t}),\dots,\mathbf{o}(t_{-n_t})\rangle$
$\langle \mathbf{u}(t_0),\dots,\mathbf{u}(t_{N_t-1})\rangle$

Figure 3-8: Information flow diagram for the single-stage, limited horizon HMEx problem at time t_0 (Def. 15).

events that have previously been scheduled before t_0 remain scheduled at the same time (Item 5 in Def. 15).

We illustrate Def. 15 in Fig. 3-8. Similar to the general HMEx problem, single-stage, limited horizon HMEx can be decomposed into two subproblems: state estimation, performed by the state estimator, and control sequence generation, accomplished by the hybrid controller. The overall goal of receding horizon HMEx is for the hybrid controller to design a control sequence $\langle \mathbf{u}(t_0), \dots, \mathbf{u}(t_{N_t-1})\rangle$ for the plant, over the current planning window. To do so, the controller reasons from the plant model $\mathcal{M}$, the qualitative state plan P, and an initial plant state $\mathbf{s}(t_0)$. The purpose of the state estimator is to compute an estimate $\hat{\mathbf{s}}(t_0)$ of this initial state.

Recall that, following the receding horizon control framework (Section 2.3), while the model-based executive, Sulu, is solving single-stage, limited horizon HMEx at time t_0, the plant is executing the partial control sequence $\langle \mathbf{u}(t_{-n_t}), \dots, \mathbf{u}(t_{-1})\rangle$ computed at the previous iteration (Fig. 3-7). Hence, the state estimator does not have direct access to $\mathbf{s}(t_0)$, since the time t_0 has not been reached yet. In order to compute the expected value $\hat{\mathbf{s}}(t_0)$ of $\mathbf{s}(t_0)$, the state estimator first computes an estimate $\hat{\mathbf{s}}(t_{-n_t})$ of the plant state at time t_{-n_t}, using the plant model, the previous known state $\mathbf{s}(t_{-2n_t})$,

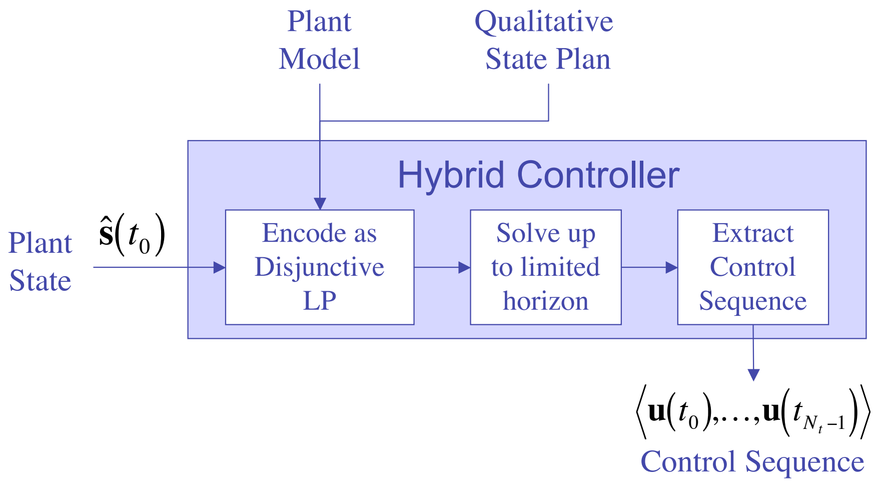


Figure 3-9: Sulu's receding horizon hybrid controller.

and the history of observations and control inputs received from (respectively, sent to) the plant during the time interval $[t_{-2n_t}, t_{-n_t}]$. It then uses the plant model in order to forecast an estimate $\hat{\mathbf{s}}(t_0)$ of the state that the plant will reach, once it has finished executing the current control sequence $\langle \mathbf{u}(t_{-n_t}), \ldots, \mathbf{u}(t_{-1}) \rangle$.

The architecture for Sulu's model-based, receding horizon, hybrid controller is presented in Fig. 3-9. Given an initial plant state $\hat{\mathbf{s}}(t_0)$ provided by the state estimator, the hybrid controller encodes both the plant model and the qualitative state plan as a mathematical, optimization program, called a *Disjunctive Linear Program* (*DLP*, Section 4.1). This DLP is then solved up to a limited horizon, corresponding to the planning window $[t_0, t_{N_t}]$ in Fig. 3-7. The new control sequence $\langle \mathbf{u}(t_0), \ldots, \mathbf{u}(t_{N_t-1}) \rangle$ is then extracted from the solution to the DLP. As defined in Def. 16, this process is repeated in order to compute an overall complete control sequence, by shifting the planning window by $n_t \cdot \Delta t$ at every iteration.

At every iteration, the hybrid controller gets a new estimate of the plant state from the state estimator; this allows Sulu to adapt to disturbances and model approximations that may have caused the plant to follow a trajectory different from the one that was planned when the corresponding control sequence was generated. By re-planning every $n_t \cdot \Delta t$ time units, Sulu is also able to revise the schedule T computed at the previous iteration, in order to adapt to possible changes in the qualitative state plan. This was already introduced in Section 1.4, and will be illustrated in more detail in Section 6.2.

3.5.3 Comparison with Related Work

Other ongoing work in the field of model-based programming addresses HMEx using a slightly different approach [26]. In order to simplify the problem to make it tractable, rather than using a receding horizon approach, [26] uses a feedback-linearization approach that effectively transforms a non-linear, multiple-input, multiple-output (*MIMO*) plant, into a set of partially decoupled, linear, single-input, single-output (*SISO*) plant abstractions, which are much easier to control than the original plant. To control each linear SISO abstraction, they use a classical PID controller, whose gains and set points are determined by the model-based executive, in order to maintain synchronization between the different abstractions, and complete the qualitative state plan, while adapting to disturbances.

The feedback-linearization technique in [26] potentially enables the model-based executive to control more complex plants than the simple piecewise-linearization method we use in this thesis. However, their approach relies on the fact that the SISO abstractions can be almost completely decoupled, and that the only coupling constraints are temporal constraints from the qualitative state plan, which specify synchronization requirements between the SISO abstractions. In particular, they assume that any given state constraint in the qualitative state plan can only involve two state variables, which must correspond to the position and velocity of the plant, for a particular dimension of motion. For instance, in the case of a bipedal walking plant [26], a goal region may only involve the position and velocity of one component of the center of gravity or center of pressure of the biped. This results in a loss of expressivity, compared to our more general approach, as defined in Def. 8.

On the other hand, the use of automatically generated PID controllers in order to control the SISO abstractions provides much more reactivity to the model-based executive. In the approach presented in this thesis, the reaction time of the executive is equal to the execution horizon, which corresponds to the rate at which the executive replans, taking into account the latest knowledge about the state of the world. In some applications, such as the simple example of an inverted pendulum, the executive must be able to react very fast, since the plant is highly unstable and has very fast dynamics. This would require choosing a very short execution horizon, which would entail that the performance of the executive would heavily rely on the quality of the guiding heuristic. Future work could look into first applying the offline, feedback-linearization technique in [26] to the plant, and using automatically generated PID controllers for the resulting SISO abstractions; Sulu would then generate gains and set points for the controllers, rather than directly generate the control inputs for the plant. Furthermore, Sulu would not require quasi-decoupling between the SISO abstractions, which is the main assumption that induces a loss of expressivity in the qualitative state plans in [26].

This approach would be very similar to the approach in [26]; however, in [26], the model-based executive relies on "tubes", that is, on pre-computed trajectory envelopes that the SISO abstractions must remain in. The fact that the tubes are pre-computed offline makes the executive less robust to high disturbances, and to higher-level unforeseen events. For instance, if a disturbance occurs that displaces one of the SISO abstractions out of its tube, then the executive aborts, returning that the problem is infeasible. Being able to recover from such high disturbances would require the capability to compute new tubes on the fly, or switch to other pre-computed tubes, which is mentioned in [26] as future work. This would also enable changes to the qualitative state plan at execution time, which is currently not allowed. Sulu's continuous planning approach would solve this problem, because it would not rely on pre-computation of tubes in order to control the SISO abstractions. The PID controllers would provide the resulting model-based executive with high-frequency reactivity, and robustness to limited, low-level disturbances, while Sulu would enable it to react in real-time, at a lower frequency, to higher disturbances and high-level unforeseen events, such as changes in the qualitative state plan. This is a promising area of future work.

In this chapter, we defined the problem we are addressing in this thesis, which we call *Hybrid, Model-based Execution* (*HMEx*, Section 3.4). HMEx is the problem of generating a control sequence for a plant, given a plant model, and a goal specification, described in the form of a *qualitative state plan*. The plant model we use (Section 3.2) is able to describe plants with continuous dynamics, and can also be used to describe hybrid systems, whose continuous dynamics depend on discrete modes. We defined a qualitative state plan (Section 3.3) as a qualitative, flexible description of the goal that the plant must achieve; this abstract, qualitative description allows us both to elevate the interaction between the plant and the human operator, and also to give more control authority to the autonomous controller, in order to adapt to disturbances and unforeseen events. Finally, we introduced *Sulu*, a hybrid, model-based executive that solves HMEx by using a receding horizon approach, which consists of iteratively reasoning over shifting planning windows (Section 3.5).

Chapter 4

Encoding the HMEx Problem as a Disjunctive Linear Program

In this chapter, we present in detail how the plant model and the qualitative state plan are encoded as a Disjunctive Linear Program (DLP). We first present the high-level concepts involved, and we compare our approach with previous approaches (Section 4.1). We then present the DLP encodings in detail, in the simplified case of Infinite Horizon HMEx (Section 4.2). We finally introduce the changes that have to be made to the encodings, in order to solve Receding Horizon HMEx (Section 4.3).

4.1 Overall DLP Approach and Comparison with Previous Work

Recall (Section 3.5.2) that part of our approach to solving the HMEx problem is to encode it as a Disjunctive Linear Program (DLP). This involves encoding all the constraints mentioned in both the plant model and the qualitative state plan, using the DLP formalism introduced in this section.

4.1.1 Motivation

As described in the previous chapter, the HMEx problem is a hybrid decision/control problem (*HDCP*) [34]. The decision-making component of the problem comes from the plant model, which specifies that the plant state is constrained to evolve in a non-convex state space. For instance, in the UAV no-fly-zone example presented in Fig. 3-3, the hybrid model-based executive must design trajectories for the UAV that avoid the no-fly-zone so that, at all times, the UAV is either on top, below, to the right, or to the left of the no-fly-zone. The optimization part of the problem comes from the fact that the qualitative state plan defines an *objective function* (Section 3.3.5) that the model-based executive must minimize. In the fire-fighting UAV example, the objective can be to minimize fuel consumption.

In order to encode both the plant model and the qualitative state plan as a single mathematical program, as described in Section 3.5.2, the mathematical programming formalism must be able to express the two aspects of the problem: the decision-making aspect, and the optimization aspect. In this section, we show that the *Disjunctive Linear Programming* formalism [9, 34, 42, 43] is able to express these two aspects, and allows us to encode HMEx as an optimization problem, subject to constraints described as logical formulae over linear constraints on the variables of the problem.

DLPs can be solved by reformulating them as Mixed-Integer Linear Programs (MILP), and by using off-the-shelf MILP solvers, such as Ilog CPLEX [2]. Other work addresses solving DLPs directly [34, 42, 43]. In particular, [42] showed that one could design a DLP solver that takes advantage of the structure of the DLP, and use conflict-directed branch-and-bound search techniques that lead to a significant improvement in solving time, over traditional branch-and-bound MILP approaches, for which the search tree is much larger. This is an important motivation for using the DLP formalism in this thesis, rather than MILP. As argued in Section 4.2.1, another reason for choosing DLP is that it involves encodings that are more natural for a human to comprehend than MILP.

4.1.2 Disjunctive Linear Programming Formalism

As introduced in Fig. 3-9, we solve each single-stage limited horizon HMEx problem by encoding it as a *disjunctive linear program* [9, 34, 42, 43], defined in conjunctive normal form in Definition 17.

Definition 17 *A* ***disjunctive linear program (DLP)*** *is an optimization problem with respect to a linear cost function f over a vector $\boldsymbol{x}$ of decision variables, subject to linear constraints on $\boldsymbol{x}$. In* ***conjunctive normal form (CNF)****, each constraint is a disjunction of linear inequalities (Eq. (4.1)).*

$$\begin{array}{ll} Minimize: & f(\boldsymbol{x}) \\ Subject\ to: & \bigwedge_i (\bigvee_j g_{i,j}(\boldsymbol{x}) \leq c_{i,j}) \end{array} \tag{4.1}$$

Any arbitrary propositional logic formula whose propositions are linear inequalities is reducible to a DLP in conjunctive normal form. Hence, in this thesis, rather than the formulation in Definition 17, we use a more convenient DLP formulation, presented in Definition 18.

Definition 18 *A disjunctive linear program in* ***propositional form*** *is a DLP in which the constraints are expressed by a formula in propositional form (Eq. (4.2)), where $\Phi(\boldsymbol{x})$ is defined in Eq. (4.3).*

$$\begin{array}{ll} Minimize: & f(\boldsymbol{x}) \\ Subject\ to: & \Phi(\boldsymbol{x}) \end{array} \tag{4.2}$$

$$\begin{array}{rl} \Phi(\boldsymbol{x}) := & \Phi(\boldsymbol{x}) \wedge \Phi(\boldsymbol{x}) \mid \Phi(\boldsymbol{x}) \vee \Phi(\boldsymbol{x}) \mid \neg\Phi(\boldsymbol{x}) \mid \\ & \Phi(\boldsymbol{x}) \Rightarrow \Phi(\boldsymbol{x}) \mid \Phi(\boldsymbol{x}) \Leftrightarrow \Phi(\boldsymbol{x}) \mid g(\boldsymbol{x}) \leq c \end{array} \tag{4.3}$$

4.1.3 HMEx as a DLP

As previously mentioned in Section 3.5.2, we use the DLP formalism to encode both the plant model and the qualitative state plan. To do so, we encode the vector $\mathbf{x}$ of decision variables of the DLP as follows (Eq. (4.4)).

$$\mathbf{x} = \left\langle \mathbf{s}(t_0), \ldots \mathbf{s}(t_{N_t}), \mathbf{u}(t_0), \ldots \mathbf{u}(t_{N_t-1}), T(e_1), \ldots T(e_{|\mathcal{E}|}) \right\rangle \tag{4.4}$$

As presented in Eq. (4.4), the vector $\mathbf{x}$ consists of both the plant state and input variables $\mathbf{s}(t)$ and $\mathbf{u}(t)$ at each time step t in the planning window, and the time $T(e)$ at which each event $e \in \mathcal{E}$ is scheduled. In the case where the objective is to minimize total plan execution time, the DLP cost function $f(\mathbf{x})$ would simply be equal to $T(e_{end})$, where e_{end} is the last event in the qualitative state plan. In the following sections, we describe in more detail how we encode the objective function in the general case, and what formula $\Phi(\mathbf{x})$ we use to encode the constraints in the plant model and in the qualitative state plan.

4.1.4 Relation with Previous Work

One can draw some similarities between the approach to encoding and solving HMEx, as introduced in this section, and previous work in planning. For instance, [44] encodes temporal planning using the PDDL2.1 framework for *durative actions*, and solves the STRIPS problem using a modified version of the Graphplan algorithm. This new algorithm, called *LPGP*, uses the layers in the graph in order to capture points in time at which events occur, rather than the uniform flow of time, as in traditional Graphplan-based systems. Other work in *Satplan* [28, 29, 30, 31] encodes classical STRIPS planning using propositional logic, and uses a SAT solver to solve the resulting SAT formulae. Similarly, in this thesis, we encode the HMEx temporal planning and control sequence generation problem as a disjunctive linear program, which is a combination of propositional logic and linear programming, and we use a DLP solver to solve the problem.

4.2 Encoding Infinite Horizon HMEx

This section introduces the DLP encodings for the plant model (Section 4.2.1), and for the qualitative state plan (Section 4.2.2), in the context of Infinite Horizon HMEx.

Recall (Section 3.5.1) that the fundamental assumption of Infinite Horizon HMEx is that the number of time steps N_t in the planning horizon is sufficiently large to cover the whole plan execution; hence, all activities are scheduled between the initial time step t_0 and the final time step t_{N_t}.

4.2.1 Plant Model Encodings

We first present the encodings in the general, domain-independent case; we then illustrate how these general encodings apply to the multiple-UAV model, before turning to qualitative state plan encoding.

Plant Model Encodings in the General Case

Recall (Section 3.2) that a plant model $\mathcal{M}$ consists of a set of forbidden regions $\mathcal{F}$ defining disallowed operating regions in the plant's state space, and a set of state equations $\mathcal{SE}$, describing the dynamics of the plant. There is also a third type of constraints, consisting of the constraint specifying an initial value $\mathbf{s}_0$ for the plant state vector $\mathbf{s}$ at the first time step t_0. In the next paragraphs, we present the DLP encodings for these three types of model-related constraints.

Forbidden Region Encodings: Forbidden regions are represented by polyhedra $\mathcal{P}_S$ in the plant's state space S, as defined in Def. 2 and Eq. (3.2), reported below (Eq. (4.5), where I is a finite set of integer indexes).

$$\mathcal{P}_S = \left\{ \mathbf{x} \in S \;\middle|\; \bigwedge_{i \in I} \mathbf{a}_i^T \mathbf{x} \leq b_i \right\} \tag{4.5}$$

Eq. (4.6) presents the DLP encoding for the constraint that the plant state $\mathbf{s}$ and the input vector $\mathbf{u}$ remain outside of $\mathcal{P}_S$ at all times. Intuitively, this encoding corresponds to the constraint $\bigwedge_{k=0\ldots N_t} \langle \mathbf{s}(t_k), \mathbf{u}(t_k) \rangle \notin \mathcal{P}_S$. This requires, for all time steps t_k, that $\langle \mathbf{s}(t_k), \mathbf{u}(t_k) \rangle$ be outside of the forbidden region.

$$\bigwedge_{k=0\ldots N_t} \left\{ \bigvee_{i=1\ldots n_{\mathcal{P}_S}} \mathbf{a}_i^T \langle \mathbf{s}(t_k), \mathbf{u}(t_k) \rangle \geq b_i \right\} \tag{4.6}$$

Examples of such forbidden regions and their corresponding encodings for the UAV case will be presented later in this section.

State Equation Encodings: In the following paragraphs, we present the DLP encodings for state equations in the plant model. We first present them in the general case, and we then give an example.

State Equation Encodings in the General Case: Recall that a state equation in $\mathcal{SE}$ is defined by a piecewise-linear relation expressing, at all time steps t_k, a state variable s_i as a function of $\mathbf{s}$ and $\mathbf{u}$ at time step t_{k-1}. This is expressed in compact form in Eq. (4.7), where $\mathbf{F}$ is a piecewise-linear function (Def. 4) over the plant's state space S.

$$\bigwedge_{k=1\ldots N_t} \mathbf{s}(t_k) = \mathbf{F}(\mathbf{s}(t_{k-1}), \mathbf{u}(t_{k-1})) \tag{4.7}$$

Let $\mathcal{S} = \{S_j \subset S\}$ be an underlying partition for $\mathbf{F}$ (Def. 4). Then, as introduced in Section 3.2.3, Eq. (4.7) becomes equivalent to Eq. (4.8). This translates to the fact that, for all time steps t_k, and for all subsets $S_j \in \mathcal{S}$, if $\langle \mathbf{s}, \mathbf{u} \rangle(t_{k-1}) \in S_j$, then the piecewise-linear expression $\mathbf{F}(\mathbf{s}(t_{k-1}), \mathbf{u}(t_{k-1}))$ simplifies to the purely linear expression $\mathbf{F}_{|S_j}(\mathbf{s}(t_{k-1}), \mathbf{u}(t_{k-1}))$, where $\mathbf{F}_{|S_j}$ is the restriction of $\mathbf{F}$ to the subset S_j.

$$\bigwedge_{k=1\ldots N_t} \left\{ \bigwedge_{S_j \in \mathcal{S}} \left\{ \begin{array}{rl} & \langle \mathbf{s}, \mathbf{u} \rangle(t_{k-1}) \in S_j \\ \Rightarrow & \mathbf{s}(t_k) = \mathbf{F}_{|S_j}(\mathbf{s}(t_{k-1}), \mathbf{u}(t_{k-1})) \end{array} \right\} \right\} \tag{4.8}$$

Empirical tests using CPLEX (Section 6.3) showed an important gain in solution time by using the equivalent encoding in Eq. (4.9). The rationale behind this encoding is the following. For all time steps t_k, since $\mathcal{S}$ is a partition of S, $\mathcal{S}$ covers the whole state space S, hence, whatever the value of $\langle \mathbf{s}, \mathbf{u} \rangle(t_{k-1})$, there must exist a subset $S_j \in \mathcal{S}$ that contains $\langle \mathbf{s}, \mathbf{u} \rangle(t_{k-1})$, and over which $\mathbf{F} \equiv \mathbf{F}_{|S_j}$. The formal proof for the equivalence between Eq. (4.9) and Eq. (4.8) is presented in Appendix A.

$$\bigwedge_{k=1\ldots N_t} \left\{ \bigvee_{S_j \in \mathcal{S}} \left\{ \begin{array}{ll} & \langle \mathbf{s}, \mathbf{u} \rangle (t_{k-1}) \in S_j \\ \wedge & \mathbf{s}(t_k) = \mathbf{F}_{|S_j}(\mathbf{s}(t_{k-1}), \mathbf{u}(t_{k-1})) \end{array} \right\} \right\} \tag{4.9}$$

A key property of Eq. (4.9) is that it is in DLP form, since $\mathbf{F}_{|S_j}(\mathbf{s}(t_{k-1}), \mathbf{u}(t_{k-1}))$ is a linear expression ($\mathbf{F}$ is piecewise-linear of underlying partition $\mathcal{S}$), and, following the argument in Section 3.2.2, we can assume that the subets S_j are polyhedra $\mathcal{P}_S$, such that $\langle \mathbf{s}(t_{k-1}), \mathbf{u}(t_{k-1}) \rangle \in S_j$ can be re-written in the linear form given in Eq. (4.10).

$$\bigwedge_{l=1\ldots n_{\mathcal{P}_S}} \mathbf{a}_l^T \langle \mathbf{s}(t_k), \mathbf{u}(t_k) \rangle \le b_l \tag{4.10}$$

Examples of State Equation Encodings: In Section 3.2.3, we introduced a very simple example of piecewise-linear state equation, in the case of a UAV model in which the x-component of the velocity v_x saturates once it reaches the value $v_x^{\max}$ (Eq. (3.9)). Similar to Eq. (4.9), we encode Eq. (3.9) using the DLP formula presented in Eq. (4.11).

$$\bigwedge_{k=1\ldots N_t} \left\{ \begin{array}{ll} & \left\{ \begin{array}{ll} & v_x(t_{k-1}) \le v_x^{\max} \\ \wedge & v_x(t_k) = f(v_x(t_{k-1}), a_x(t_{k-1})) \end{array} \right\} \\ \vee & \left\{ \begin{array}{ll} & v_x(t_{k-1}) \ge v_x^{\max} + \epsilon \\ \wedge & v_x(t_k) = v_x^{\max} \end{array} \right\} \end{array} \right\} \tag{4.11}$$

Another example introduced in Section 3.2.4 involves a thermostat, modeled as a hybrid automaton. We showed that the state equations for this plant were described by Eq. (3.12). The corresponding DLP encoding is presented in Eq. (4.12).

$$\bigwedge_{k=1\ldots N_t} \left\{ \begin{array}{ll} & \left\{ \begin{array}{ll} & m_{k-1} = 0 \\ \wedge & \frac{x(t_k) - x(t_{k-1})}{\Delta t} = -0.1 x(t_{k-1}) \end{array} \right\} \\ \vee & \left\{ \begin{array}{ll} & m_{k-1} = 1 \\ \wedge & \frac{x(t_k) - x(t_{k-1})}{\Delta t} = 5 - 0.1 x(t_{k-1}) \end{array} \right\} \end{array} \right\} \tag{4.12}$$

Note that, in Section 3.2.4, we argued that, in order to be able to describe the thermostat automaton using the plant model formalism introduced in Section 3.2, we had to constrain the mode variable m to take on a finite set of discrete values, which contradicted the definition of a plant model, in which all variables involved were real-valued variables. Eq. (4.12) allows us to remove this contradiction, by extending the domain of m to $\mathbb{R}$, since Eq. (4.12) effectively constrains m to take on values only from the set $\{0, 1\}$. Therefore, it is not necessary to explicitly require m to have a finite, discrete domain.

Initialization Constraint Encoding: This constraint specifies an initial value $\mathbf{s}_0$ for the plant state vector $\mathbf{s}$ at the first time step t_0. The DLP encoding is straightforward, and presented in Eq. (4.13).

$$\mathbf{s}(t_0) = \mathbf{s}_0 \tag{4.13}$$

Plant Model Encodings in the Multiple-UAV Example

In this section, we demonstrate the plant model encodings on the multiple-UAV example, before turning to the encoding of the qualitative state plan. These example encodings are adapted from the model introduced in [39].

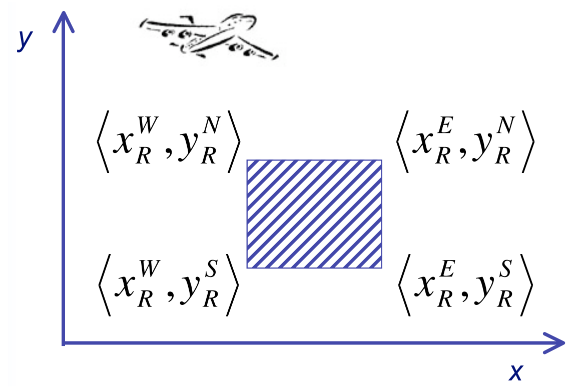
Recall that the constraints in the plant model are of three types: the constraints imposed by the forbidden regions in $\mathcal{F}$, those corresponding to the state equations in $\mathcal{SE}$, and the initialization constraints. In our multiple-UAV example, $\mathcal{F}$ includes the following forbidden regions:

1. No-fly-zones in the x/y subspace of each aircraft;

2. Regions of S in which the velocity of a given aircraft is lower than its minimum allowed value;

3. Regions of S in which the velocity or the acceleration of a given aircraft is greater than its maximum allowed value;

4. Unsafe regions of S in which the two aircraft are too close to each other.



y
$\langle x_R^W, y_R^N \rangle$
$\langle x_R^E, y_R^N \rangle$
$\langle x_R^W, y_R^S \rangle$
$\langle x_R^E, y_R^S \rangle$
x

Figure 4-1: Rectangular no-fly-zone in the UAV fire-fighting example.

In the following paragraphs, we go through each of these types of forbidden regions, and we give the corresponding DLP encodings. Note that most of these encodings were adapted from [39]; the main difference is that [39] uses a MILP formalism, rather than a DLP formalism to encode the constraints. The MILP formalism results in encodings that are arguably less natural for a human to comprehend than the ones we present in this thesis. More importantly, as mentioned in Section 4.1.2, efficient conflict-directed branch-and-bound algorithms can be designed in order to solve DLPs, which take advantage of the structure of the DLP to construct a search tree that is significantly smaller than MILP search trees, leading to significant improvement in solution time and space [34, 42, 43].

No-fly-zone Avoidance [39]: Eq. (4.14) presents the DLP encoding for a forbidden region for aircraft α_i, when the forbidden region R is a no-fly-zone in the vehicle's x/y subspace. To keep this example simple, we assume that R is rectangular (Fig. 4-1), and is represented by its North-East corner $\langle x_R^E, y_R^N \rangle$ and South-West corner $\langle x_R^W, y_R^S \rangle$. In a more general case, when forbidden region R is represented by a polyhedron $\mathcal{P}_S$ (Eq. (4.5)), the corresponding encoding is the same as in Eq. (4.6). It can also be extended to the case when forbidden region R has a more general, non-convex shape, by approximating R by a finite union of convex regions, and by linearizing each region (Section 3.2.2).

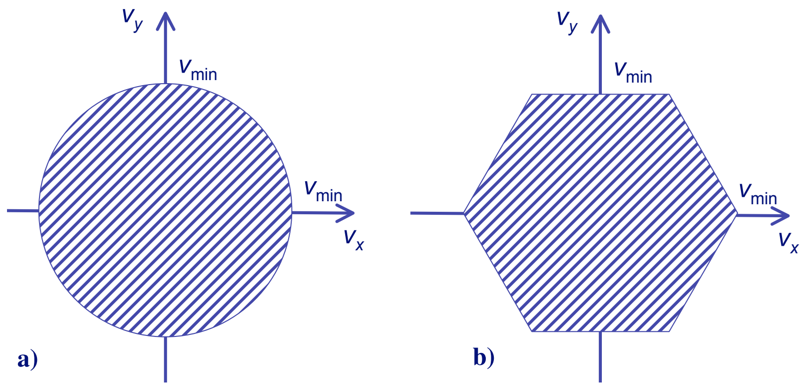


Figure 4-2: **a)** Forbidden region corresponding to values of the velocity smaller than the minimum allowed value; **b)** Linearized version of the forbidden region.

$$\bigwedge_{k=0\ldots N_t} \left\{ \begin{array}{rlcl} & y^{\alpha_i}(t_k) & \geq & y_o^N \\ \vee & x^{\alpha_i}(t_k) & \geq & x_o^E \\ \vee & y^{\alpha_i}(t_k) & \leq & y_o^S \\ \vee & x^{\alpha_i}(t_k) & \leq & x_o^W \end{array} \right\} \tag{4.14}$$

Intuitively, Eq. (4.14) specifies that, at every time step t_k, aircraft α_i should be either on top, to the right, below, or to the left of the forbidden region. Note that this encoding assumes that vehicles are modeled by points that have no volume; as mentioned in [39], safety margins must be added around the no-fly-zones to account for the vehicles' actual volume, and also to prevent a trajectory from crossing the corner of a forbidden region between two time steps. More specifically, we assume that navigation is taking place in the configuration space [45].

Minimum Velocity [39]: Consider a given aircraft α_i; the forbidden region corresponding to values of the velocity smaller than the minimum allowed value $v_{\text{min}}^{\alpha_i}$ for α_i is simply a sphere in the v_x/v_y subspace of α_i, centered at $\langle 0, 0\rangle$, and of radius equal to the minimum velocity (Fig. 4-2a). In order to encode the forbidden region using the DLP formalism, we linearize this forbidden region by approximating the sphere with a regular polyhedron (Fig. 4-2b). In our implementation, we use a dodecahedron.

The DLP constraint can then be encoded using the general encoding in Eq. (4.6).

For this special type of regular polyhedron, we use the encoding in Eq. (4.15), for each aircraft α_i (with $J = 12$ in the case of a dodecahedron). Similar to Eq. (4.14), this encodes that, at all time steps t_k, the velocity vector $\langle v_x^{\alpha_i}, v_y^{\alpha_i} \rangle$ should be outside of the polyhedron, with respect to one of its sides (identified by its index j).

$$\bigwedge_{k=0 \ldots N_t} \left\{ \bigvee_{j=1 \ldots J} \left\{ \begin{array}{rl} & v_x^{\alpha_i}(t_k) \cdot \cos(\frac{2j\pi}{J}) \\ + & v_y^{\alpha_i}(t_k) \cdot \sin(\frac{2j\pi}{J}) \end{array} \geq v_{\min}^{\alpha_i} \right\} \right\} \tag{4.15}$$

Maximum Velocity and Acceleration [39]: To impose a maximum value on the velocity and the acceleration of every aircraft α_i, we use the same method as for the minimum velocity in the previous paragraph, but we require that the plant state remain *inside* of the polyhedron. The corresponding encodings are presented in Eq. (4.16) and (4.17).

$$\bigwedge_{k=0 \ldots N_t} \left\{ \bigwedge_{j=1 \ldots J} \left\{ \begin{array}{rl} & v_x^{\alpha_i}(t_k) \cdot \cos(\frac{2j\pi}{J}) \\ + & v_y^{\alpha_i}(t_k) \cdot \sin(\frac{2j\pi}{J}) \end{array} \leq v_{\max}^{\alpha_i} \right\} \right\} \tag{4.16}$$

$$\bigwedge_{k=0 \ldots N_t} \left\{ \bigwedge_{j=1 \ldots J} \left\{ \begin{array}{rl} & a_x^{\alpha_i}(t_k) \cdot \cos(\frac{2j\pi}{J}) \\ + & a_y^{\alpha_i}(t_k) \cdot \sin(\frac{2j\pi}{J}) \end{array} \leq a_{\max}^{\alpha_i} \right\} \right\} \tag{4.17}$$

Aircraft Inter-collision Avoidance: Consider two aircraft α_i and α_j. For a given position $\langle x_0^{\alpha_i}, y_0^{\alpha_i} \rangle$ of aircraft α_i, aircraft α_j must remain outside of the forbidden region corresponding to the sphere in the $x^{\alpha_j}/y^{\alpha_j}$ subspace, centered at $\langle x^{\alpha_j}, y^{\alpha_j} \rangle = \langle x_0^{\alpha_i}, y_0^{\alpha_i} \rangle$, and of radius $(\epsilon^{\alpha_i} + \epsilon^{\alpha_j})$ (where ϵ^{α_i} is the safety margin to be maintained around aircraft α_i [39]). In much the same way as for the bounds on the velocity and accelerations, presented in the preceding paragraphs, we linearize this sphere using a regular polyhedron. In our implementation, we simply use a square.

$$\bigwedge_{k=0 \ldots N_t} \left\{ \begin{array}{rl} & x^{\alpha_j}(t_k) - x^{\alpha_i}(t_k) \geq \epsilon^{\alpha_i} + \epsilon^{\alpha_j} \\ \vee & x^{\alpha_i}(t_k) - x^{\alpha_j}(t_k) \geq \epsilon^{\alpha_i} + \epsilon^{\alpha_j} \\ \vee & y^{\alpha_j}(t_k) - y^{\alpha_i}(t_k) \geq \epsilon^{\alpha_i} + \epsilon^{\alpha_j} \\ \vee & y^{\alpha_i}(t_k) - y^{\alpha_j}(t_k) \geq \epsilon^{\alpha_i} + \epsilon^{\alpha_j} \end{array} \right\} \tag{4.18}$$

The DLP encoding for the corresponding constraint is presented in Eq. (4.18). Intuitively, it encodes the requirement that for all time steps t_k, aircraft α_i and α_j must remain distant from each other by at least $\epsilon^{\alpha_i} + \epsilon^{\alpha_j}$, either in the x or in the y direction.

State Equations: As introduced in Section 3.2.3, all the state equations for a given aircraft α_i can be written in compact form, presented in Eq. (4.19) and (4.20).

$$\bigwedge_{k=1\ldots N_t} \left\{ \begin{bmatrix} x^{\alpha_i} \\ y^{\alpha_i} \\ v_x^{\alpha_i} \\ v_y^{\alpha_i} \end{bmatrix} (t_k) = \mathbf{A} \cdot \begin{bmatrix} x^{\alpha_i} \\ y^{\alpha_i} \\ v_x^{\alpha_i} \\ v_y^{\alpha_i} \end{bmatrix} (t_{k-1}) + \mathbf{B} \cdot \begin{bmatrix} a_x^{\alpha_i} \\ a_y^{\alpha_i} \end{bmatrix} (t_{k-1}) \right\} \tag{4.19}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ; \quad \mathbf{B} = \begin{bmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \tag{4.20}$$

Matrices **A** and **B** can then be used to write the DLP encoding for the overall plant state equation, presented in compact form in Eq. (4.21).

$$\bigwedge_{k=1\ldots N_t} \left\{ \mathbf{s}(t_k) = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{A} \end{array} \right] \cdot \mathbf{s}(t_{k-1}) + \left[\begin{array}{c} \mathbf{B} \\ \hline \mathbf{B} \end{array} \right] \cdot \mathbf{u}(t_{k-1}) \right\} \tag{4.21}$$

Initialization Constraint: Recall that the initialization constraint is the constraint that initializes the value of the state vector $\mathbf{s}$ at the initial time step t_0. As presented in Eq. (3.1), in the multi-UAV example, the state vector $\mathbf{s}$ consists of the Cartesian coordinates of the position and the velocity of each UAV. As a result, in the case of two aircrafts α_1 and α_2, the general DLP encoding in Eq. (4.13) translates to Eq. (4.22).

$$\left\{\begin{array}{rlrl} & x^{\alpha_1}(t_0) = x_0^{\alpha_1} & \wedge & y^{\alpha_1}(t_0) = y_0^{\alpha_1} \\ \wedge & v_x^{\alpha_1}(t_0) = (v_x^{\alpha_1})_0 & \wedge & v_y^{\alpha_1}(t_0) = (v_y^{\alpha_1})_0 \\ \wedge & x^{\alpha_2}(t_0) = x_0^{\alpha_2} & \wedge & y^{\alpha_2}(t_0) = y_0^{\alpha_2} \\ \wedge & v_x^{\alpha_2}(t_0) = (v_x^{\alpha_2})_0 & \wedge & v_y^{\alpha_2}(t_0) = (v_y^{\alpha_2})_0 \end{array}\right\} \tag{4.22}$$

4.2.2 Qualitative State Plan Encodings

The second component of the problem statement to be encoded is the qualitative state plan. Recall that the qualitative state plan specifies the set of feasible trajectories that the plant is allowed to traverse. This section presents the qualitative state plan encodings as a DLP. Recall that the qualitative state plan incorporates two types of constraints: temporal constraints between events, and state constraints, associated with activities in the plan. It also incorporates an objective function, which Sulu must minimize in order to achieve optimality.

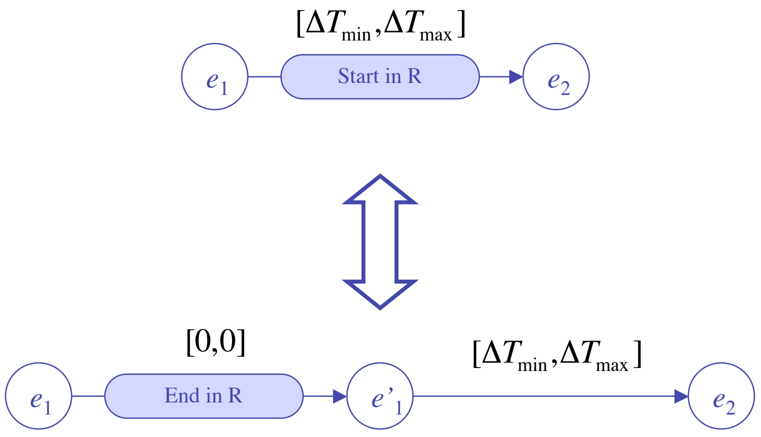
Temporal Constraints Between Events

Eq. (4.23) encodes a temporal constraint between two events, e_S and e_E. For example, in Fig. 3-5, events e_1 and e_5 must be distant from each other by at least 0 and at most 20 time units; in that case, Eq. (4.23) becomes $T(e_5) - T(e_1) \leq 20 \wedge T(e_5) - T(e_1) \geq 0$.

$$\left\{\begin{array}{rl} & T(e_E) - T(e_S) \leq \Delta T^{\max}_{e_S \rightarrow e_E} \\ \wedge & T(e_E) - T(e_S) \geq \Delta T^{\min}_{e_S \rightarrow e_E} \end{array}\right\} \tag{4.23}$$

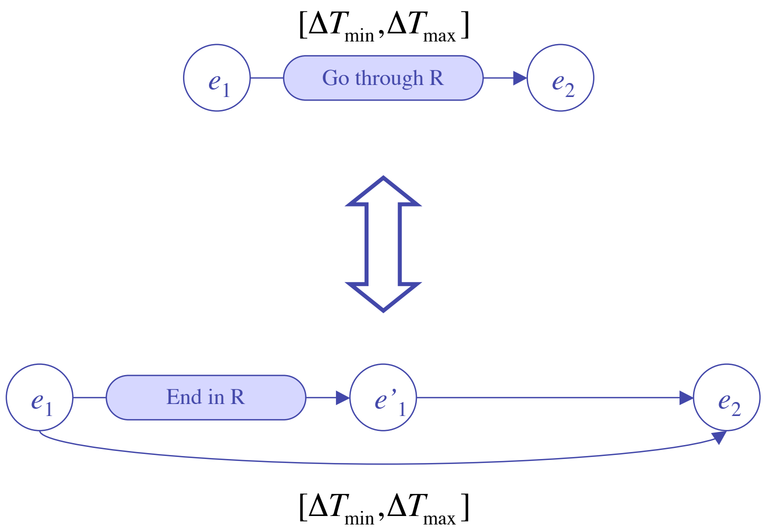
State Constraints

Recall (Section 3.3) that activities are of the following types: “Start in state region R_S”, “End in state region R_E”, “Remain in state region $R_\forall$” and “Go through state region $R_\exists$”. *Start in* and *go through* activities are derivable from *end in* activities (Fig. 4-3 and 4-4). Hence, we only present the encodings for the two primitive types *remain in* and *end in.* In each case, we assume that the state regions R_E and $R_\forall$ are polyhedra (Eq. (4.5)), such that $\langle \mathbf{s}(t), \mathbf{u}(t) \rangle \in R_E$ and $\langle \mathbf{s}(t), \mathbf{u}(t) \rangle \in R_\forall$ can be expressed as DLP constraints, similar to (Eq. (4.6)).



$[\Delta T_{\min},\Delta T_{\max}]$
e_1
Start in R
e_2
[0,0]
$[\Delta T_{\min},\Delta T_{\max}]$
e_1
End in R
e'_1
e_2

Figure 4-3: Derivation of a *start in* activity from an *end in* activity.



$[\Delta T_{\min},\Delta T_{\max}]$
e_1
Go through R
e_2
e_1
End in R
e'_1
e_2
$[\Delta T_{\min},\Delta T_{\max}]$

Figure 4-4: Derivation of a *go through* activity from an *end in* activity.

***Remain in* activity:** Eq. (4.24) presents the encoding for a *remain in state* $R_\forall$ activity between events e_S and e_E. This imposes $\langle \mathbf{s}(t), \mathbf{u}(t) \rangle \in R_\forall$ for all time steps $t \in [T(e_S), T(e_E)]$.

$$\bigwedge_{k=0\ldots N_t} \left\{ \left\{ \begin{array}{rl} & T(e_S) \leq t_k \\ \wedge & T(e_E) \geq t_k \end{array} \right\} \Rightarrow \langle \mathbf{s}(t_k), \mathbf{u}(t_k) \rangle \in R_\forall \right\} \tag{4.24}$$

Intuitively, this encodes that, for all time steps t_k, if the start event e_S is scheduled before t_k, and the end event e_E is scheduled after t_k, then we must enforce that $\langle \mathbf{s}, \mathbf{u} \rangle$ be in the goal region $R_\forall$ at time step t_k. In the multi-UAV fire-fighting example, the activity "Remain in state [α_1 at fire]" imposes the constraint that α_1 must be in the fire region between e_2 and e_3, while it is dropping water.

***End in* activity:** Consider an *end in* activity, imposing $\langle \mathbf{s}, \mathbf{u} \rangle \in R_E$ at time $T(e_E)$, that is, when event e_E is scheduled. The encoding (for infinite horizon HMEx) is presented in Eq. (4.25), which translates to the fact that there must exist a time step t_k that is ϵ-close to $T(e_E)$ and for which $\langle \mathbf{s}(t), \mathbf{u}(t) \rangle \in R_E$. We assume here that the time discretization is regular, of granularity Δt ($t_{k+1} = t_k + \Delta t$ for all k), such that the time step t_k is required to be $\frac{\Delta t}{2}$-close to $T(e_E)$.

$$\bigvee_{k=0\ldots N_t} \left\{ \begin{array}{rl} & T(e_E) \geq t_k - \frac{\Delta t}{2} \\ \wedge & T(e_E) \leq t_k + \frac{\Delta t}{2} \\ \wedge & \langle \mathbf{s}(t_k), \mathbf{u}(t_k) \rangle \in R_E \end{array} \right\} \tag{4.25}$$

Intuitively, one would want to encode the constraint $\langle \mathbf{s}(T(e_E)), \mathbf{u}(T(e_E)) \rangle \in R_E$. Recall, however, that $T(e_E)$ is allowed to take any real value, while $\mathbf{s}(t)$ and $\mathbf{u}(t)$ are only defined for values of t that correspond to one of the time steps t_k. For this reason, we encode that $\langle \mathbf{s}(t_k), \mathbf{u}(t_k) \rangle \in R_E$ must be enforced for the time step t_k that is the closest to $T(e_E)$.

An example of such an activity in our fire-fighting scenario is the "End in [α_2 at fire]" activity imposing α_2 to be in the fire region at event e_4.

Objective Function Encodings

Recall (Section 3.3.5) that the goal specification for Sulu, described in the form of a qualitative state plan, involves minimizing a given *objective function* $F(\mathbf{S}, \mathbf{U}, T)$, which is a piecewise-linear function over a state sequence $\mathbf{S}$, an input sequence $\mathbf{U}$ and a schedule T. The intuitive, direct DLP encoding for the objective function would be to add $F(\mathbf{S}, \mathbf{U}, T)$ to the DLP cost function. However, the definition of a disjunctive linear program (Def. 17) requires that the cost function be a purely linear function of the DLP variables, which is not the case of $F(\mathbf{S}, \mathbf{U}, T)$ in general, since F is piecewise-linear. To solve this issue, we introduce a new DLP variable c, which we add to the DLP cost function, while constraining c to be equal to the value of the objective function. The resulting encoding is presented in Eq. (4.26), where $\mathcal{S}_F$ is an underlying partition for F, in which each of the subsets $S_j \in \mathcal{S}_F$ can be assumed to be polyhedral. Note the similarity with the encoding for the piecewise-linear state equations (Eq. 4.9).

$$\begin{array}{ll} \mathit{Minimize:} & \qquad\qquad c \\ \mathit{Subject\ to:} & \bigvee_{S_j \in \mathcal{S}_F} \left\{ \begin{array}{cc} & \langle \mathbf{S}, \mathbf{U}, T \rangle \in S_j \\ \wedge & c = F_{|S_j}(\mathbf{S}, \mathbf{U}, T) \end{array} \right\} \end{array} \tag{4.26}$$

In this section, we presented how we encode both the plant model and the qualitative state plan as a DLP, in the case of infinite horizon HMEx. However, as argued in Section 3.5.2, the infinite horizon approach to HMEx is not robust, and can quickly become intractable. To tackle this issue, we introduced another approach, *receding horizon HMEx*, that consists of iteratively solving HMEx over small, shifting planning windows. In the following section, we describe what changes need to be done to the DLP encodings in order to encode receding horizon HMEx.

4.3 Encoding Single-stage Limited Horizon HMEx

In the previous section (Section 4.2), we presented the DLP encodings for the plant model and the qualitative state plan, in the case of infinite horizon HMEx. Recall that the fundamental assumption of infinite horizon HMEx is that the number of time steps N_t considered, is sufficiently large to cover the whole state plan execution; hence all events in the plan are guaranteed to be scheduled between the first time step t_0 and the last time step t_{N_t}.

In receding horizon HMEx, we iteratively solve single-stage limited horizon HMEx problems, where each problem considers only a small number of time steps; hence, all events are no longer guaranteed to be scheduled between t_0 and t_{N_t}. As a consequence, the encoding of *end in* activities has to be revised, as presented in Sections 4.3.1 and 4.3.2.

4.3.1 Revised Encoding for *End in* Activities

Consider an *end in* activity that imposes $\langle \mathbf{s}, \mathbf{u} \rangle \in R_E$ at the time $T(e_E)$, when event e_E is scheduled. Eq. (4.25) can no longer be used, since the planning window is no longer guaranteed to cover the whole plan execution. As a result, there might not exist any time step t_k in the planning window that is $\frac{\Delta t}{2}$-close to $T(e_E)$. To handle this, we use the revised encoding presented in Eq. (4.27).

$$\begin{array}{cl} \bigvee\limits_{k=0\ldots N_t} & \left\{ \begin{array}{rl} & T(e_E) \geq t_k - \frac{\Delta t}{2} \\ \wedge & T(e_E) \leq t_k + \frac{\Delta t}{2} \\ \wedge & \langle \mathbf{s}(t_k), \mathbf{u}(t_k) \rangle \in R_E \end{array} \right\} \\ \vee & T(e_E) \leq t_0 - \frac{\Delta t}{2} \\ \vee & T(e_E) \geq t_{N_t} + \frac{\Delta t}{2} \end{array} \tag{4.27}$$

Eq. (4.27) translates to the fact that, either there exists a time step t_k in the planning window that is $\frac{\Delta t}{2}$-close to $T(e_E)$ and for which $\langle \mathbf{s}(t_k), \mathbf{u}(t_k) \rangle \in R_E$, or event e_E must be scheduled outside of the current planning window (either before the beginning of the current planning window t_0, or beyond the planning horizon t_{N_t}).

4.3.2 Guidance Heuristic for *End in* Activities

Consider an "End in state region R_E" activity, starting at event e_S and ending at event e_E, and assume that the start event has already been executed ($T(e_S) \leq t_0$). Assume also that the end event has not been executed yet, and that the goal region R_E is unreachable within the current execution horizon t_{n_t}. This means that the *end in* activity is currently being executed, and that the plant's state should be currently evolving towards R_E. However, since R_E is unreachable within t_{n_t} and e_E has not yet been executed, the only linear constraint that is active in the DLP encoding (Eq. (4.27)) is the last constraint $T(e_E) \geq t_{N_t} + \frac{\Delta t}{2}$, and none of the constraints $\langle \mathbf{s}(t_k), \mathbf{u}(t_k) \rangle \in R_E$ is active. As a result, the plant's state trajectory is effectively unconstrained (except by the forbidden regions in the state space): the plant is free to "go in any direction", as long as it stays outside of the forbidden regions. In particular, there is absolutely no guarantee that the plant's state is currently evolving towards the goal region R_E; it might be "going in the wrong direction", away from R_E.

Proposed Guidance Framework in the General Case

To solve the aforementioned issue, Sulu uses a *guidance heuristic*, in order to guide the trajectory towards R_E. The heuristic is a function $h_{R_E} : S \mapsto \mathbb{R}$ that associates, with any vector $\langle \mathbf{s}, \mathbf{u} \rangle \in S$, an estimate $h_{R_E}(\mathbf{s}, \mathbf{u})$ of the "distance" in S, that is "cost to go," from $\langle \mathbf{s}, \mathbf{u} \rangle$ to the goal region R_E. The value of this heuristic at $\langle \mathbf{s}(t_{n_t}), \mathbf{u}(t_{n_t}) \rangle$ is then added to the DLP cost function (Eq. (4.28)); hence, Sulu favors partial trajectories that end as "close" as possible to R_E.

$$Minimize: \quad h_{R_E}(\mathbf{s}(t_{n_t}), \mathbf{u}(t_{n_t})) \tag{4.28}$$

However, for Eq. (4.28) to be a valid DLP cost function, h_{R_E} must be linear over the whole state space S. This is an unacceptable limitation: in many applications such as the multiple-UAV example, to be a helpful heuristic, h_{R_E} needs to take into account the forbidden regions in S, in order to avoid designing trajectories that lead the plant into local minima, for example, in the presence of non-convex forbidden regions. As a result, h_{R_E} can have a complex shape; in this case, it cannot be appropriately linearized over the complete state space.

To relax this limitation, rather than only considering linear heuristics, we allow the heuristic to be piecewise-linear (Def. 4), and we use the same method as in Section 4.2.2 to add h_{R_E} to the DLP cost function. Let $\mathcal{S} = \{S_i \subset S\}$ be a finite underlying partition of h_{R_E}, where we can assume that each S_i is a polyhedron (see Section 4.2.1 for a justification of this assumption). Eq. (4.28) then becomes Eq. (4.29), where $(h_{R_E})_{|S_i}$ is the restriction of h_{R_E} to S_i, and h is a new DLP variable. Note the similarity with Eq (4.26) in the way we encode piecewise-linearity.

$$\begin{array}{ll} \mathit{Minimize:} & h \\ \mathit{Subject\ to:} & \bigvee_{S_i \in \mathcal{S}} \left\{ \begin{array}{cc} & \langle \mathbf{s}(t_{n_t}), \mathbf{u}(t_{n_t}) \rangle \in S_i \\ \wedge & h = (h_{R_E})_{|S_i}(\mathbf{s}(t_{n_t}), \mathbf{u}(t_{n_t})) \end{array} \right\} \end{array} \tag{4.29}$$

As mentioned before, this guidance heuristic only needs to be applied when either the *end in* activity is currently being executed, or is scheduled to start in the current execution window ($T(e_S) < t_{n_t}$), and to end beyond the planning horizon ($T(e_E) \geq t_{N_t}$). Hence, we can modify Eq. (4.29) in order for the constraint to be only enforced when these two conditions are verified (Eq. (4.30)). Note that in Eq. (4.30) we added the constraint $h \geq 0$ in order to prevent the cost function from being unbounded below, when $T(e_S) < t_{n_t}$ and $T(e_E) \geq t_{N_t}$. This implies that the guidance heuristic may only take on non-negative values.

$$\begin{array}{ll} \mathit{Min:} & h \\ \mathit{S.\ t.:} & h \geq 0 \end{array}$$
$$\left\{ \begin{array}{cc} & T(e_S) < t_{n_t} \\ \wedge & T(e_E) \geq t_{n_t} \end{array} \right\} \Rightarrow \bigvee_{S_i \in \mathcal{S}} \left\{ \begin{array}{cc} & \langle \mathbf{s}(t_{n_t}), \mathbf{u}(t_{n_t}) \rangle \in S_i \\ \wedge & h = (h_{R_E})_{|S_i}(\mathbf{s}(t_{n_t}), \mathbf{u}(t_{n_t})) \end{array} \right\} \tag{4.30}$$

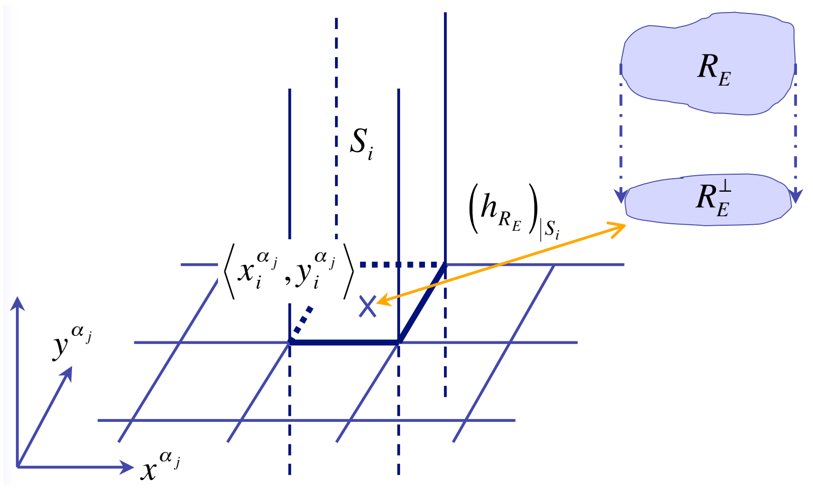


Figure 4-5: Guidance heuristic for an *end in* activity involving a particular aircraft α_j.

A possible intuitive interpretation of Eq. (4.30) is the following. One can think of $\mathcal{S}$ as a discrete "cost map" that contains subgoals S_i, for which the cost to go to the final goal region R_E is known, and given by the linear guiding heuristic $(h_{R_E})_{|S_i}$. If the *end in* activity is scheduled to start within the current planning horizon and to end beyond $(T(e_S) < t_{n_t} \wedge T(e_E) \geq t_{n_t})$, then Sulu must choose a subgoal S_i that minimizes the remaining cost to go from $\langle \mathbf{s}(t_{n_t}), \mathbf{u}(t_{n_t}) \rangle \in S_i$ to the destination R_E.

Guidance Heuristic in the Multiple-UAV Fire-fighting Example

In the fire-fighting example, for an *end in* activity involving a specific aircraft α_j, $\mathcal{S}$ is a partition of S in which each subset S_i is a polyhedral cylinder orthogonal to the $x^{\alpha_j}/y^{\alpha_j}$ subspace, and whose base in the $x^{\alpha_j}/y^{\alpha_j}$ subspace is a square centered at a given position $\langle x_i^{\alpha_j}, y_i^{\alpha_j} \rangle$ (Fig. 4-5). For each S_i, $(h_{R_E})_{|S_i}$ is then chosen to be constant, and equal to an estimate of the time necessary to go from $\langle x_i^{\alpha_j}, y_i^{\alpha_j} \rangle$ to the projection $R_E^{\perp}$ of the goal region R_E onto the $x^{\alpha_j}/y^{\alpha_j}$ subspace. Similar to [10], we compute $(h_{R_E})_{|S_i}$ for each i by constructing a visibility graph based on the forbidden regions in the $x^{\alpha_j}/y^{\alpha_j}$ subspace. This is illustrated in Fig. 4-6.

As suggested in Fig. 4-6, Eq. (4.30) can be simplified by reducing $\mathcal{S}$ to a subset $\tilde{\mathcal{S}} \subset \mathcal{S}$ that excludes all the regions S_i for which the proposition $\langle \mathbf{s}(t_{n_t}), \mathbf{u}(t_{n_t}) \rangle \in S_i$ is

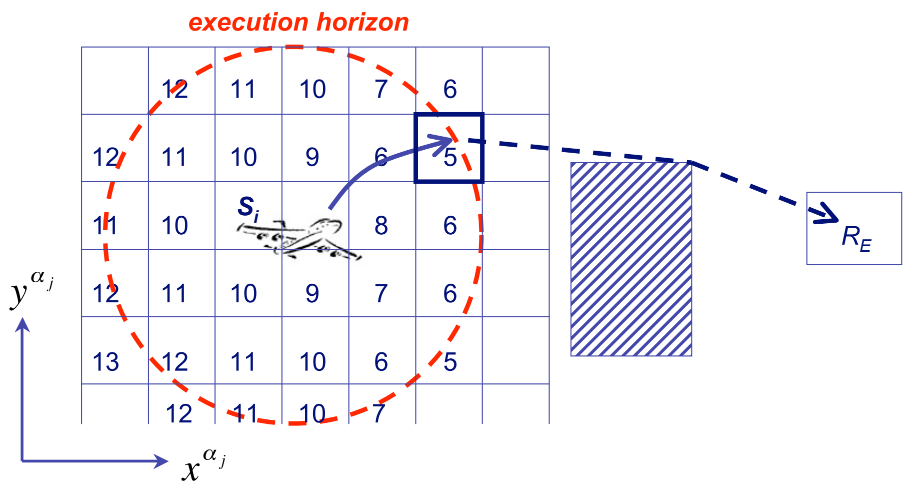


Figure 4-6: Example of a guidance heuristic for a fire-fighting UAV.

guaranteed to be unsatisfiable. This is the case when S_i is inside a forbidden region, or when S_i is unreachable within the current execution horizon. For instance, in the multiple-UAV example, the maximum velocity constraints allow us to ignore the regions that are not reachable by the UAVs within the execution horizon, since they are too far away from the initial position $\mathbf{s}(t_0)$. This allows us to only consider the regions S_i that are within a limited distance of $\mathbf{s}(t_0)$, hence, in our implementation, $\mathcal{S}$ only contains about twenty subgoals.

We present in Chapter 5 how the plant model $\mathcal{M}$ can allow us to determine, in the general case, when a region of the state space is unreachable.

Comparison with Previous Work [10]

Our guidance heuristic approach builds upon the guidance framework introduced in [10]. However, our approach is generic, while the heuristic in [10] is domain-specific, and does not extend to the general type of plants considered in this thesis. There are two main differences between the two approaches. First, [10] computes the cost to go starting from the *planning horizon* t_{N_t} rather than the execution horizon t_{n_t} (Fig. 4-7). The reason we use the execution horizon is to lower the complexity of Eq. (4.30): for a given size of the subgoals S_i, reasoning with the execution horizon, rather than the planning horizon, yields a lower number of subgoals in the reduced

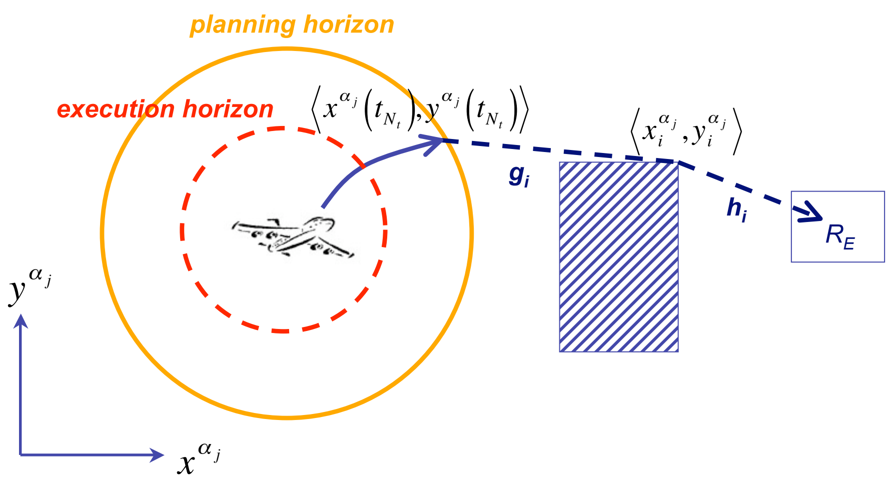

planning horizon
execution horizon
$\langle x^{\alpha_j}(t_{N_t}), y^{\alpha_j}(t_{N_t})\rangle$
$\langle x_i^{\alpha_j}, y_i^{\alpha_j}\rangle$
g_i
h_i
R_E
y^{α_j}
x^{α_j}

Figure 4-7: Guidance heuristic used in [10].

cost map $\tilde{\mathcal{S}}$, since the plant can reach a larger part of the state space within t_{N_t} than within $t_{n_t} < t_{N_t}$.

Second, [10] uses a cost map that consists of subgoal points $\langle x_i^{\alpha_j}, y_i^{\alpha_j}\rangle$ for which an estimate h_i of the cost to go to the final goal R_E is known; these subgoals are simply the corners of the forbidden regions in the visibility graph. The value of the guidance heuristic for a given subgoal $\langle x_i^{\alpha_j}, y_i^{\alpha_j}\rangle$ is then chosen equal to the sum of h_i and an estimate g_i of the cost to go from $\langle x^{\alpha_j}(t_{N_t}), y^{\alpha_j}(t_{N_t})\rangle$ to $\langle x_i^{\alpha_j}, y_i^{\alpha_j}\rangle$ (Fig. 4-7). g_i is computed simply by taking the straight-line Euclidian distance between the two points. To make sure that this estimate is valid, they only allow their controller to choose subgoals $\langle x_i^{\alpha_j}, y_i^{\alpha_j}\rangle$ that are "visible" from $\langle x^{\alpha_j}(t_{N_t}), y^{\alpha_j}(t_{N_t})\rangle$; the concept of visibility corresponds to the existence of a straight line from $\langle x^{\alpha_j}(t_{N_t}), y^{\alpha_j}(t_{N_t})\rangle$ to $\langle x_i^{\alpha_j}, y_i^{\alpha_j}\rangle$ that does not cross any forbidden region.

The validity of g_i relies on the concept of visibility, and on the fact that, in the case of a fixed-wing UAV, a straight line is effectively the shortest trajectory between two points that is consistent with the dynamics of the aircraft. This assumption does not hold in the general case: if the forbidden regions are no longer simply in the x/y subspace, but include state variables such as velocities and/or accelerations, there is no concept of visibility, and their method to compute g_i falls short. More generally, a straight-line trajectory might not be consistent with the dynamics of the plant, in

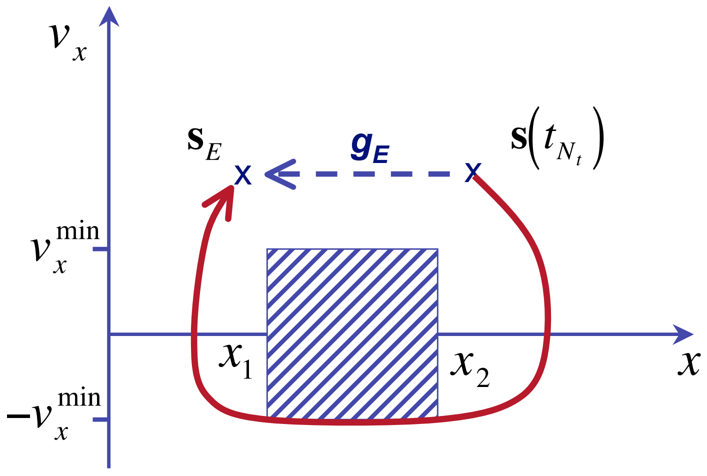


Figure 4-8: Simple example showing a limitation of the approach used in [10].

which case the straight-line Euclidian distance might be a very poor heuristic. Our approach relaxes this assumption, and is applicable in the general case, given that we have access to a guidance heuristic for the plant.

Consider the example of a 1-D plant whose state vector is $\mathbf{s} = \langle x, v_x \rangle$, where x is the position, and v_x is the velocity. The input is the acceleration, which has finite lower and upper bounds. Consider the forbidden region presented in Fig. 4-8, corresponding to a segment $[x_1, x_2]$ in which the plant is required to maintain a minimum absolute velocity $v_x^{\min}$. The guidance heuristic approach in [10] would consider the goal state $\mathbf{s}_E$ "visible" from $\mathbf{s}(t_{N_t})$, and the heuristic value g_E assigned to $\mathbf{s}_E$ would be equal to the Euclidian distance from $\mathbf{s}(t_{N_t})$ to $\mathbf{s}_E$ (in this case, $h_E = 0$ since $\mathbf{s}_E$ is the goal state). However, the straight-line trajectory from $\mathbf{s}(t_{N_t})$ to $\mathbf{s}_E$ is inconsistent with the plant dynamics, since x can only increase when v_x is positive, and decrease when v_x is negative. The shortest feasible trajectory in that case would be one that goes around the forbidden region. Our approach would build a cost map of subgoal regions whose cost to go would be the value of the guidance heuristic, computed, for instance, by running a shortest path algorithm on a discretization of the state space (Fig. 4-9). This leads to a better guidance heuristic, which is able to guide the plant around the forbidden region, rather than suggest to go straight towards the goal state, which is infeasible.

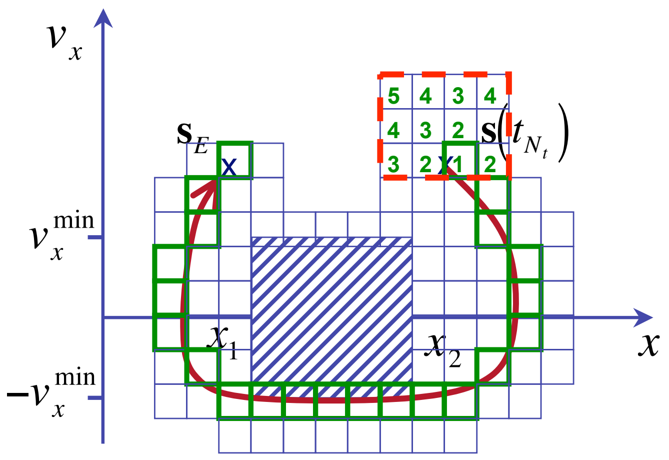


Figure 4-9: Our guidance heuristic correctly guides the plant around the forbidden region.

In this chapter, we first presented our overall approach to solving Receding Horizon HMEx, which consists of encoding the plant model and the qualitative state plan as a mathematical, optimization problem, using a Disjunctive Linear Programming formalism (Section 4.1). We motivated the use of DLP by the fact that it was suited to encode the hybrid logic/optimization nature of the HMEx problem. We then presented in more detail how we use this formalism in order to encode the constraints in the plant model and in the qualitative state plan, in the case of infinite horizon HMEx (Section 4.2). We illustrated the encodings using the multiple-UAV fire-fighting example introduced in Section 3.1, and the hybrid thermostat automaton presented in Section 3.2.4. Finally, we showed how the encodings could be modified in order to solve receding horizon HMEx (Section 4.3). In particular, we introduced a heuristic that Sulu uses in order to guide the plant towards regions of the state space specified by *end in* activities in the qualitative state plan, when these goal regions are not reachable within the current planning window.

Chapter 5

Constraint Pruning Policies

In the preceding chapter, we saw that our receding horizon model-based executive, Sulu, solves the HMEx problem by encoding it as a DLP, and iteratively solving it over small planning windows. The ability of Sulu to solve the problem in real-time is limited by the complexity of the DLP, in terms of the number of variables and the number of constraints. In the second half of the previous chapter, we presented how, by limiting the number of time steps N_t in the planning window, we could lower the number of variables, since every time step t_i introduces $n + m$ variables in the DLP, corresponding to the n components of the state vector $\mathbf{s}(t_i)$ and the m components of the input vector $\mathbf{u}(t_i)$.

In this chapter, we describe a method to further lower the complexity of the DLP, by lowering the number of constraints. This is done through the use of a set of novel *pruning policies* that enable Sulu to *prune* constraints, without loss of correctness.

5.1 Overall Constraint Pruning Framework

In this section, we formally define a *pruning policy* as a function that returns whether or not a constraint in the HMEx problem can be pruned (Def. 19).

Definition 19 *A* ***pruning policy*** *is a function* $p : \mathcal{C}_{HMEx} \mapsto \{\texttt{true}, \texttt{false}\}$ *that, for each constraint* $c \in \mathcal{C}_{HMEx}$*, returns* `true` *if* c *can be pruned, and* `false` *if it cannot* *be pruned.* $\mathcal{C}_{HMEx}$ *is the set of* ***HMEx constraints****, where a HMEx constraint is a constraint introduced either by the plant model or the qualitative state plan (Chap. 3); the list of all types of HMEx constraints is recalled below:*

1. *A constraint imposed by the plant model* $\mathcal{M}$ *can be of the following three types:*

   (a) *A constraint imposed by an forbidden region* $R \in \mathcal{F}$*, which constrains the plant vector* $\langle \boldsymbol{s}(t_i), \boldsymbol{u}(t_i) \rangle$ *to remain outside of* R *for all time steps* t_i *in the planning window;*

   (b) *A constraint corresponding to a state equation* $s_i(t_k) = f_i(\boldsymbol{s}(t_{k-1}), \boldsymbol{u}(t_{k-1}))$*;*

   (c) *The constraint imposing an initial value to* $\boldsymbol{s}(t_0)$*.*

2. *A constraint imposed by the qualitative state plan* P *is of two types (one more type of state plan constraints will be introduced in Section 5.3.1):*

   (a) *A temporal constraint* $c \in \mathcal{C}$*, specifying lower and upper bounds on the time between two events in the qualitative state plan;*

   (b) *A state constraint* c_S*, associated with a given activity* $a \in \mathcal{A}$*. This last category includes the guidance constraints introduced in Section 4.3.2.*

As will be presented in Chapter 6, the pruning policy is called on all HMEx constraints at the beginning of each iteration of the receding horizon HMEx algorithm, in order to incrementally update the DLP. Consider a HMEx constraint $c \in \mathcal{C}_{HMEx}$; if the pruning policy returns $p(c) = \texttt{true}$, then c can be pruned, which means that its corresponding DLP encoding can be removed from the DLP (if it was in the DLP at the previous iteration), or ignored (if it was not in the DLP at the previous iteration). Similarly, if $p(c) = \texttt{false}$, then c may influence the solution; therefore, the DLP encoding for c must be added to the DLP (if it was not in the DLP at the previous iteration), or updated (if it was previously in the DLP, but it involves parameters such as t_0 or t_{N_t}, whose values have changed since the previous iteration).

Alg. 1 Constraint pruning policy for the DLP constraint associated with a given forbidden region $\mathcal{P}_S$

```
1: if $\mathcal{R} \cap \mathcal{P}_S = \emptyset$ then
2:     return true
3: else
4:     return false
5: end if
```

5.2 Plant Model Constraint Pruning

As mentioned in Def. 19, some of the HMEx constraints are specified by the plant model $\mathcal{M}$, while others are introduced by the qualitative state plan. In this section, we present pruning policies for the constraints imposed by the plant model: forbidden region constraints (Section 5.2.1), state equation constraints (Section 5.2.2), and the state initialization constraint (Section 5.2.3).

5.2.1 Forbidden Region Constraint Pruning

General Case

Recall that the plant model $\mathcal{M}$ defines forbidden regions in the state space S as polyhedra of S (Eq. (4.5)). The corresponding DLP encoding for a given polyhedron $\mathcal{P}_S \in S$ was presented in Eq. (4.6), and is repeated in Eq. (5.1).

$$\bigwedge_{k=0\ldots N_t} \bigvee_{i=1\ldots n_{\mathcal{P}_S}} \mathbf{a}_i^T \langle \mathbf{s}(t_k), \mathbf{u}(t_k) \rangle \geq b_i \tag{5.1}$$

Eq. (5.1) can be pruned if $\mathcal{P}_S$ can be guaranteed to be unreachable from the initial plant state $\mathbf{s}(t_0)$, within the planning horizon t_{N_t}. That is, if the region $\mathcal{R}$ of all states $\mathbf{s}$ reachable from $\mathbf{s}(t_0)$ within t_{N_t} is disjoint from $\mathcal{P}_S$ (Alg. 1). $\mathcal{R}$ is formally defined in Eq. (5.2) to (5.4).

$$\mathcal{R}_0 = \{\mathbf{s}(t_0)\} \tag{5.2}$$

$$\forall k = 0 \ldots N_t - 1, \quad \mathcal{R}_{k+1} = \left\{ \mathbf{s}_{k+1} \in \mathbb{R}^n \;\middle|\; \begin{array}{ll} & \mathbf{s}_{k+1} = \mathbf{A}\mathbf{s}_k + \mathbf{B}\mathbf{u}_k \\ \wedge & \mathbf{s}_k \in \mathcal{R}_k \\ \wedge & \forall R \in \mathcal{F} \;\; \langle \mathbf{s}_k, \mathbf{u}_k \rangle \notin R \end{array} \right\} \tag{5.3}$$

$$\mathcal{R} = \bigcup_{k=0 \ldots N_t} \mathcal{R}_k \tag{5.4}$$

As shown in Eq. (5.3), $\mathcal{R}_{k+1}$ can be constructed as the set of states $\mathbf{s}_{k+1}$ that are reachable in one time step from any state $\mathbf{s}_k \in \mathcal{R}_k$, using any input $\mathbf{u}_k$ that does not make $\langle \mathbf{s}_k, \mathbf{u}_k \rangle$ violate any forbidden region $R \in \mathcal{F}$.

Consider the simple example of a plant whose state vector is $\mathbf{s} = \langle x \rangle$, and whose input vector is $\mathbf{u} = \langle v_x \rangle$, where x and v_x are linked by the state equation $x(t_k) = x(t_{k-1}) + \Delta t \cdot v_x(t_{k-1})$. $\mathcal{F}$ consists of two forbidden regions, imposing a lowerbound $v_x^{\min}$ and an upper bound $v_x^{\max}$ on the velocity v_x. If the initial state is $x(t_0) = 0$, then the set of possible values for $x(t_1 = t_0 + \Delta t)$ is $\mathcal{R}_1 = [\Delta t \cdot v_x^{\min}, \Delta t \cdot v_x^{\max}]$. Iteratively, the set of reachable states at time step $t_2 = t_1 + \Delta t$ from $\mathcal{R}_1$ is $\mathcal{R}_2 = [2 \cdot \Delta t \cdot v_x^{\min}, 2 \cdot \Delta t \cdot v_x^{\max}]$.

Efficient techniques have been developed in order to compute $\mathcal{R}$, such as in [8, 16, 22, 38, 40, 59]. In the following paragraphs, we describe how, for our fire-fighting UAV example, we use an approximation of $\mathcal{R}$, which can be computed easily.

UAV Example

Recall that, in the UAV example, the forbidden regions considered are the following (Section 4.2.1):

1. No-fly-zones in the x/y subspace of each aircraft;
2. Regions of S in which the velocity of a given aircraft is lower than its minimum allowed value;
3. Regions of S in which the velocity or the acceleration of a given aircraft is greater than its maximum allowed value, and

Alg. 2 Constraint pruning policy for collision avoidance between aircraft α_i and no-fly-zone $R \in \mathcal{F}$

```
1: if $dist(\langle x^{\alpha_i}, y^{\alpha_i}\rangle(t_0), R) > N_t \cdot \Delta t \cdot v_{\alpha_i}^{\max}$ then
2:     {forbidden region $R$ is out of reach within $N_t$;}
3:     return true
4: else
5:     return false
6: end if
```

4. Unsafe regions of S in which two aircraft are too close to each other.

In this section, we illustrate the pruning policy in Alg. 1 on the forbidden regions of Type 1 and Type 4. For these two types of forbidden regions, we use a fast, approximate method to compute R: for a given aircraft α_i, we approximate its reachability set $\mathcal{R}$ by a 2-sphere $\tilde{\mathcal{R}}$ centered at $\langle x^{\alpha_i}, y^{\alpha_i}\rangle(t_0)$ and of radius $N_t \cdot \Delta t \cdot v_{\alpha_i}^{\max}$, which is simply the length of the longest straight-line trajectory that α_i can travel within the planning horizon. Note that, since this is an optimistic approximation ($\mathcal{R} \subset \tilde{\mathcal{R}}$), it is a sound approximation: it will not lead us to prune forbidden regions that are reachable within the current planning horizon.

No-fly-zone Avoidance Constraint Pruning: Recall that the DLP constraint encoding no-fly-zone avoidance is presented in Eq. (4.14), in the simplified example of a rectangular no-fly-zone. The corresponding pruning policy is presented in Alg. 2, for a given aircraft α_i and a given no-fly-zone R in the $x^{\alpha_i}/y^{\alpha_i}$ subspace. Similar to Alg. 1, the constraint can be pruned if $\tilde{\mathcal{R}} \cap R = \emptyset$; however, here $\tilde{\mathcal{R}}$ is the 2-sphere centered at $\langle x^{\alpha_i}, y^{\alpha_i}\rangle(t_0)$ and of radius $N_t \cdot \Delta t \cdot v_{\alpha_i}^{\max}$, hence $\tilde{\mathcal{R}} \cap R = \emptyset$ is equivalent to $dist(\langle x^{\alpha_i}, y^{\alpha_i}\rangle(t_0), R) > N_t \cdot \Delta t \cdot v_{\alpha_i}^{\max}$, where $dist$ is the Euclidian distance in the $x^{\alpha_i}/y^{\alpha_i}$ subspace.

Vehicle Collision Avoidance Constraints Pruning: Consider the DLP constraint encoding collision avoidance between two aircraft α_i and α_j (Eq. (4.18)). Effectively, the pruning policy for this constraint (Alg. 3) is a special case of the policy for no-fly-zones presented in the previous paragraph; here, the no-fly-zone R that air-

Alg. 3 Constraint pruning policy for collision avoidance between aircraft α_i and α_j

```
1: if $\|\langle x^{\alpha_i}, y^{\alpha_i}\rangle(t_0) - \langle x^{\alpha_j}, y^{\alpha_j}\rangle(t_0)\|_{\mathcal{L}_2} > N_t \cdot \Delta t \cdot (v^{\alpha_i}_{\max} + v^{\alpha_j}_{\max}) + \epsilon^{\alpha_i} + \epsilon^{\alpha_j}$ then
2:     return true
3: else
4:     return false
5: end if
```

craft α_i must avoid is the 2-sphere in the $x^{\alpha_i}/y^{\alpha_i}$ subspace, centered on aircraft α_j, and of radius $\epsilon^{\alpha_i} + \epsilon^{\alpha_j}$ (where ϵ^{α_i} is the safety margin to be maintained around aircraft α_i). Hence, $dist(\langle x^{\alpha_i}, y^{\alpha_i}\rangle(t_0), R) = \|\langle x^{\alpha_i}, y^{\alpha_i}\rangle(t_0) - \langle x^{\alpha_j}, y^{\alpha_j}\rangle(t_0)\|_{\mathcal{L}_2} - (\epsilon^{\alpha_i} + \epsilon^{\alpha_j})$.

5.2.2 State Equation Constraint Pruning

As introduced in Section 3.2.3, the dynamics of the plant are modeled by state equations, which predict the values of the plant variables at every time step t_i, from their values at time step t_{i-1} and the values of the control inputs at time step t_{i-1}. These state equations are fundamentally necessary for Sulu to be able to design control sequences for the plant, by planning into the future; for this reason, state equation constraints are never pruned (the pruning policy always returns `false`).

As presented in Section 5.1, this means that the state equation constraints always remain encoded in the DLP, and that they may have to be updated every time Sulu shifts the planning window, if they depend on parameters whose values change when the planning window changes (such as t_0 or t_{N_t}). This is the case, for instance, when we remove the assumption that the state equations are time-invariant (Section 3.2.3).

5.2.3 State Initialization Constraint Pruning

Recall (Section 4.2.1) that, for the plant model to be able to properly predict the behavior of the plant over time, the plant state vector $\mathbf{s}(t)$ must be given an initial value $\mathbf{s}_0$ at the first time step t_0 in the planning window. As described in Section 3.5.2, this initial value is computed by the state estimator, using the last control sequence $\mathbf{U} = \langle \mathbf{u}(t_{-n_t}), \ldots, \mathbf{u}(t_{-1})\rangle$ previously sent to the plant, and using an estimate of the state of the plant at the time t_{-n_t} when it starts executing that control

Alg. 4 Pruning policy for the state initialization constraint.

```
1: query the state estimator for the expected value $\mathbf{s}_0$ of the plant state at time
   step $t_0$
2: return false
```

sequence $\mathbf{U}$. The value of the initial plant state $\mathbf{s}_0$ hence changes every time the planning window changes; therefore, the DLP encoding for the state initialization constraint (Eq. 4.13) must be updated at every iteration, when the corresponding pruning policy is called. This policy is presented in Alg. 4. Note that, similar to the state equation pruning policy (Section 5.2.2), the policy always returns `false`, since the state initialization constraint is a fundamental constraint necessary for Sulu to be able to plan into the future from a known initial position.

As described in Alg. 4, the policy queries the state estimator for a prediction $\mathbf{s}_0$ of the plant state at time step t_0, when it has finished executing the control sequence $\mathbf{U}$. $\mathbf{s}_0$ is the value used to initialize the state variable in Eq. 4.13.

Note that the model used by state estimator in order to compute $\mathbf{s}_0$ does not have to be the same as the plant model used by the hybrid controller, introduced in Section 3.2.3. In particular, it does not need to verify the piecewise-linearity assumption, and it may use a more fine-grained time discretization, if any. The piecewise-linearity assumption and the time discretization are only necessary to encode the model using the DLP formalism. Here, the model used by the state estimator may be a better, non-linear, continuous model of the plant dynamics. Sulu then uses this better model to compensate for the approximations in the piecewise-linear model used to design the control sequences.

5.3 Qualitative State Plan Constraint Pruning

Recall that a constraint mentioned in a qualitative state plan P can be either a temporal constraint between two events, a *remain in* constraint, an *end in* constraint, or a heuristic guidance constraint for a given *end in* activity. In this section, for each type, we show that the problem of finding a pruning policy is equivalent to that of

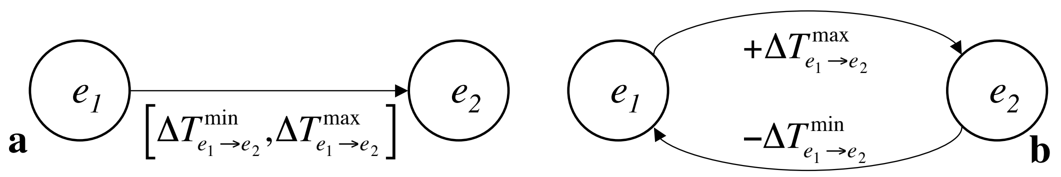

a
e_1
e_2
$\left[\Delta T^{\min}_{e_1 \to e_2}, \Delta T^{\max}_{e_1 \to e_2}\right]$
e_1
e_2
$+\Delta T^{\max}_{e_1 \to e_2}$
$-\Delta T^{\min}_{e_1 \to e_2}$
b

Figure 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).

foreseeing if an event in the qualitative state plan could possibly be scheduled within the current planning horizon. This is solved by computing bounds $\langle T^{\min}_e, T^{\max}_e \rangle$ on $T(e)$, for every event e. Given the execution times of past events, these bounds are computed from the bounds $\langle \Delta T^{\min}_{\langle e,e' \rangle}, \Delta T^{\max}_{\langle e,e' \rangle} \rangle$ on the distance between any pair of events $\langle e, e' \rangle$, obtained using the method in [17]. This involves running an all-pairs shortest path algorithm on the distance graph corresponding to the qualitative state plan (Fig. 5-1), which can be done offline (Chapter 6, Alg. 11).

5.3.1 Temporal Constraint Pruning

Overall Approach

In this section, we describe a temporal constraint pruning policy that effectively prunes any event in the qualitative state plan that is guaranteed to be scheduled outside of the current planning window. Any temporal constraint involving such an event is also pruned. This choice of policy can be motivated by the two following arguments:

1. If an event is guaranteed to be scheduled before the beginning of the current planning window t_0, this means that the event has already been executed. There is no need to schedule that event anymore; hence, it can be pruned from the DLP.

2. If an event is guaranteed to be scheduled beyond the current planning horizon t_{N_t}, then we can defer the scheduling of that event to a later time, when it is no longer guaranteed to be scheduled outside of the current planning window.

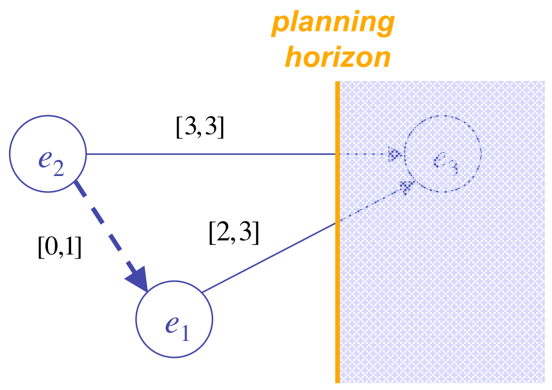


Figure 5-2: Example of an implied temporal constraint.

Hence, such an event can also be pruned from the DLP.

In some domains of applications, however, this pruning policy might not be desirable. For instance, if the human operator needs to know in advance the intentions of the autonomous system in order to schedule his or her own actions accordingly, then it might be necessary for Sulu to design complete schedules for the whole qualitative state plan, rather than only for parts of the plan that may be scheduled within the current planning window. In that case, the events and the temporal constraints should not be pruned. Note that the complete schedules that would then be generated would not be guaranteed to be unchanging, since Sulu might need to reschedule future events in order to adapt to disturbances, and to account for the fact that, following the receding horizon framework, state constraints beyond the planning horizon are ignored.

Pruning all the temporal constraints that involve an event that is guaranteed to be scheduled outside of the current planning window, however, can have three bad consequences.

First, implied temporal constraints between two events that can be scheduled within the current planning window might no longer be enforced. Implied temporal constraints are constraints that do not appear explicitly in the qualitative state plan, but are a logical consequence of several explicit temporal constraints. This is illustrated in Fig. 5-2. Consider the two temporal constraints $e_2 \rightarrow e_3$ and $e_1 \rightarrow e_3$,

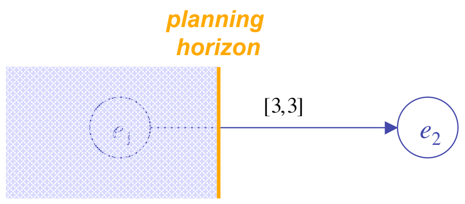


Figure 5-3: Example of a state plan where the pruning policy in Alg. 5 entails infeasible schedules.

which are explicitly mentioned in the state plan. The implied constraint $e_1 \rightarrow e_2$ follows from the two explicit constraints. Since event e_3 is guaranteed to be scheduled beyond the planning horizon, the two explicit constraints are pruned. As a result, events e_1 and e_2 are no longer linked by any temporal constraint; therefore, the schedule designed by Sulu might violate the implied constraint.

Second, the schedule might violate temporal constraints between events that remain to be scheduled, and events that have already been executed. This is illustrated in Fig. 5-3, where pruning the temporal constraint between the two events results in the domain of $T(e_2)$ being unbounded below; since e_2 is no longer linked to any past event, it can be scheduled at any point in time, regardless of when e_1 was executed.

Third, when the objective is to minimize total plan execution time, we argued previously (Section 4.1.3) that this objective could be encoded in the DLP by just adding $T(e_{end})$ to the DLP cost function, where e_{end} is the end event of the qualitative state plan. This is no longer valid; if e_{end} is guaranteed to be scheduled beyond the planning window, then all temporal constraints on e_{end} have been relaxed, and minimizing $T(e_{end})$ does not have any effect on the time at which the other events are scheduled.

To address the first two issues, instead of just pruning temporal constraints involving events that are guaranteed to be scheduled outside of the current planning window, we *compile them out* of the DLP. This involves explicitly encoding all temporal constraints between any pair of events, instead of encoding only those that are specified in the qualitative state plan. It also involves introducing new *unary* temporal constraints (different from the *binary* temporal constraints previously introduced),

Alg. 5 Pruning policy for the binary temporal constraint between events e_S and e_E

```
1:  if $T^{\max}_{e_S} < t_0$ then
2:      {$e_S$ has already been executed;}
3:      return true
4:  else if $T^{\min}_{e_S} > t_{N_t}$ then
5:      {$e_S$ is out of reach within the current horizon;}
6:      return true
7:  else if $T^{\max}_{e_E} < t_0$ then
8:      {$e_E$ has already been executed;}
9:      return true
10: else if $T^{\min}_{e_E} > t_{N_t}$ then
11:     {$e_E$ is out of reach within the current horizon;}
12:     return true
13: else
14:     return false
15: end if
```

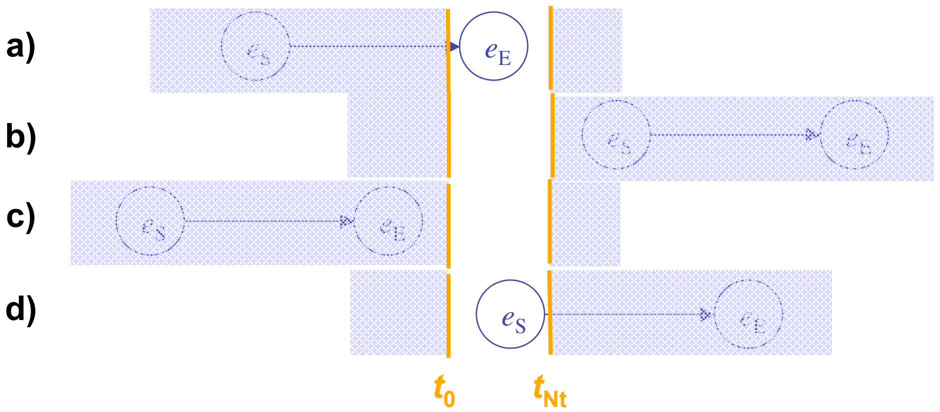


Figure 5-4: Illustration of the different cases in Alg. 5 (shaded areas are time periods outside of the current planning window $[t_0, t_{N_t}]$) : **a)** e_S has already been executed (line 1); **b)** e_S is out of reach within the current horizon (line 4); **c)** e_E has already been executed (line 7); **d)** e_E is out of reach within the current horizon (line 10).

which specify absolute lower and upper bounds on the times at which events can be scheduled. In the following subsections, we describe how this is done, as well as how we address the third issue, when the objective function consists of minimizing total plan execution time.

Note that the method we use in order to compile the irrelevant temporal constraints out of the DLP is very similar to previous work on dispatchable plans [17, 49, 60, 61], introduced in Section 2.1. The method described in the following paragraphs is analog to computing a dispatchable plan for the qualitative state plan.

Binary Temporal Constraint Pruning

As perviously introduced, a temporal constraint between a pair of events $\langle e_S, e_E \rangle$ can be pruned if the time bounds on either event guarantee that the event will be scheduled outside of the current planning window (Alg. 5, Fig. 5-4).

However, in order to avoid cases such as the one in Fig. 5-2, rather than applying this policy only to the explicit temporal constraints that are mentioned in the qualitative state plan, we encode and apply the policy to all temporal constraints between any pair of events $\langle e, e' \rangle$, derived from the temporal bounds $\langle \Delta T^{\min}_{\langle e,e' \rangle}, \Delta T^{\max}_{\langle e,e' \rangle} \rangle$ computed by the method in [17]. This way, no implied temporal constraint is unintentionally ignored, since all temporal constraints between all pairs of events are explicitly encoded in the DLP.

Unary Temporal Constraint Pruning (Alg. 6, Fig. 5-5)

As previously introduced, in order to properly compile out binary temporal constraints without allowing situations such as the one in Fig. 5-3, we introduce a new HMEx constraint that enforces unary temporal constraints on every event e; the DLP encoding is presented in Eq. (5.5), where the bounds $T_e^{\min}$ and $T_e^{\max}$ are the ones introduced at the beginning of Section 5.3.

$$T_e^{\min} \leq T(e) \leq T_e^{\max} \tag{5.5}$$

Alg. 6 Pruning policy for the unary temporal constraint on an event e.

```
1: if $T_e^{\max} < t_0$ then
2:     {$e$ has already been executed;}
3:     return true
4: else if $T_e^{\min} > t_{N_t}$ then
5:     {$e$ is out of reach within the current horizon;}
6:     return true
7: else
8:     return false
9: end if
```

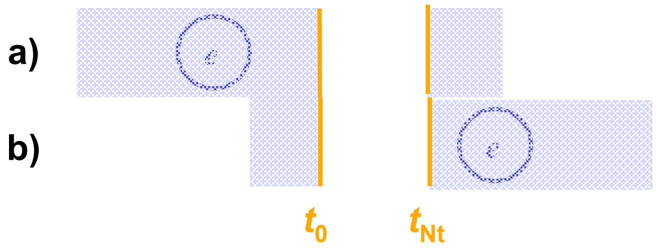


Figure 5-5: Illustration of the different cases in Alg. 6: **a)** e has already been executed (line 1); **b)** e is out of reach within the current horizon (line 4).

A unary temporal constraint on an event e can be pruned whenever e is guaranteed to be scheduled before the beginning of the current planning window (line 1, Fig. 5-5 a), or beyond the current planning horizon (line 4, Fig. 5-5 b).

Minimizing Total Plan Execution Time

This subsection only applies when the objective function consists of minimizing the overall plan execution time. In Section 4.1.3, we argued that this objective could be encoded in the DLP by just adding $T(e_{end})$ to the DLP cost function, where e_{end} is the end event of the qualitative state plan. As previously mentioned, this approach does not work if e_{end} is pruned. To address this issue, rather than minimizing the time $T(e_{end})$ at which the end event is scheduled, we minimize the time $T(e)$ at which every event e is scheduled. This is encoded in the DLP by modifying the encoding for unary temporal constraints (Eq. (5.5)); the new encoding is presented in Eq. (5.6). The pruning policy remains the same (Alg. 6, Fig. 5-5).

Alg. 7 Pruning policy for a "Remain in state region $R_\forall$" activity starting at event e_S and ending at event e_E

```
1:  if $T^{\max}_{e_E} < t_0$ then
2:      {activity is completed;}
3:      return true
4:  else if $T^{\max}_{e_S} < t_0$ then
5:      {activity is being executed;}
6:      return false
7:  else if $T^{\min}_{e_S} > t_{N_t}$ then
8:      {activity will start beyond $t_{N_t}$;}
9:      return true
10: else if $T^{\max}_{e_S} \leq t_{N_t}$ then
11:     {activity will start within $t_{N_t}$;}
12:     return false
13: else if $\mathcal{R} \cap R_\forall = \emptyset$ then
14:     {$R_\forall$ is unreachable within $t_{N_t}$; postpone start event $e_S$:}
15:     $TIGHTEN\_BOUNDS(e_S, t_{N_t}, T^{\max}_{e_S})$
16:     return true
17: else
18:     return false
19: end if
```

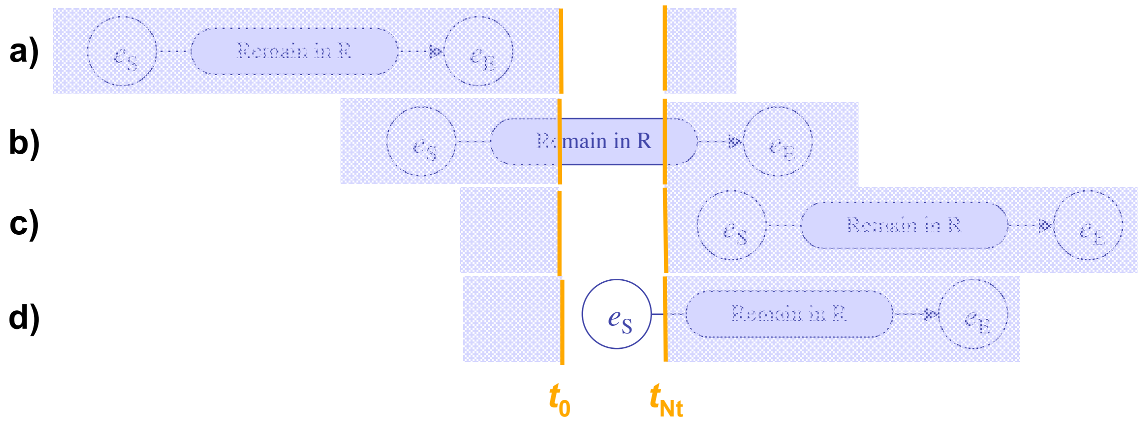


Figure 5-6: Illustration of the different cases in Alg. 7: **a)** The activity is completed (line 1); **b)** The activity is being executed (line 4); **c)** The activity will start beyond t_{N_t} (line 7); **d)** The activity will start within t_{N_t} (line 10).

Alg. 8 *TIGHTEN_BOUNDS*$(e, T_{\min}, T_{\max})$ routine to tighten the time bounds on an event e

```
1: if $T_e^{\min} < T_{\min}$ then
2:     {the new lower bound is tighter than the old one;}
3:     $T_e^{\min} \leftarrow T_{\min}$
4:     {propagate to other events:}
5:     for all events $e' \in \mathcal{E}$ do
6:         $T_{e'}^{\min} \leftarrow \max(T_{e'}^{\min}, T_e^{\min} + \Delta T_{\langle e,e' \rangle}^{\min})$
7:     end for
8: end if
9: if $T_e^{\max} > T_{\max}$ then
10:    {the new upper bound is tighter than the old one;}
11:    $T_e^{\max} \leftarrow T_{\max}$
12:    {propagate to other events:}
13:    for all events $e' \in \mathcal{E}$ do
14:        $T_{e'}^{\max} \leftarrow \min(T_{e'}^{\max}, T_e^{\max} + \Delta T_{\langle e,e' \rangle}^{\max})$
15:    end for
16: end if
```

$$\begin{aligned} &\mathit{Minimize:} && T(e) \\ &\mathit{Subject\ to:} && T_e^{\min} \le T(e) \le T_e^{\max} \end{aligned} \tag{5.6}$$

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

Consider the state constraint c_S on a "Remain in state region $R_\forall$" activity a, between events e_S and e_E (Eq. (4.24)). If e_E is guaranteed to be scheduled in the past (Fig. 5-6**a**), that is, it has already occurred (line 1), then a has been completed and c_S can be pruned. Otherwise, if e_S has already occurred (line 4, Fig. 5-6**b**), then a is being executed and c_S must not be pruned. Else, if a is guaranteed to start beyond the planning horizon (line 7, Fig. 5-6**c**), then c_S can be pruned. Conversely, if a is guaranteed to start within the planning horizon, (line 10, Fig. 5-6**c**), then c_S must not be pruned.

Otherwise, the time bounds on $T(e_S)$ and $T(e_E)$ provide no guarantee, but we can still use the plant model $\mathcal{M}$ to try to prune the constraint; if $\mathcal{M}$ guarantees that $R_\forall$ is unreachable within the planning horizon, then c_S can be pruned (line 13; refer to Eq. (5.2) to (5.4) for the definition of $\mathcal{R}$). In that case, e_S must be explicitly postponed

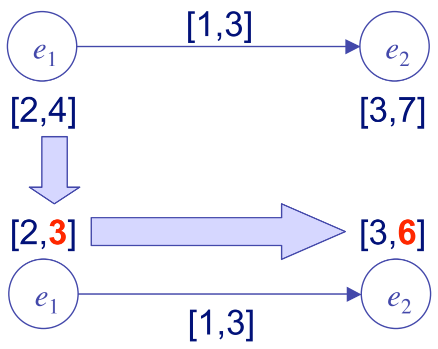


Figure 5-7: Temporal propagation of a change in a unary temporal constraint.

beyond the current planning horizon (line 15); otherwise, it could be scheduled within the planning window, without the *remain in* constraint being enforced. Event e_S is postponed after t_{N_t} by calling the $TIGHTEN\_BOUNDS$ routine in Alg. 8.

This routine changes the temporal bounds on an event (if the new bounds are tighter than the old ones), and propagates the changes to the bounds on the other events. Fig. 5-7 illustrates this on an example. In this example, the upper time bound on event e_1 is tightened from 4 to 3. By propagating this change to other events, we are able to tighten the upper time bound on event e_2 from 7 to 6.

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

Consider a constraint c_S on an "End in state region R_E" activity ending at event e_E (Eq. (4.27)). If e_E is guaranteed to be scheduled in the past, that is, it has already occurred (Fig. 5-8**a**, line 1), then c_S can be pruned. Otherwise, if the value of $T_{e_E}^{\max}$ guarantees that e_E will be scheduled within the planning horizon (line 4, Fig. 5-8**b**), then c_S must not be pruned. Conversely, it can be pruned if $T_{e_E}^{\min}$ guarantees that e_E will be scheduled beyond the planning horizon (line 7, Fig. 5-8**c**). Finally, c_S can also be pruned if the plant model guarantees that R_E is unreachable within the planning horizon from the current plant state (line 10). Similar to Alg. 7, e_E must then be explicitly postponed.

Alg. 9 Pruning policy for an "End in state region R_E" activity ending at event e_E

```
1:  if $T_{e_E}^{\max} < t_0$ then
2:      {$e_E$ has already occurred;}
3:      return true
4:  else if $T_{e_E}^{\max} \leq t_{N_t}$ then
5:      {$e_E$ will be scheduled within $t_{N_t}$;}
6:      return false
7:  else if $T_{e_E}^{\min} > t_{N_t}$ then
8:      {$e_E$ will be scheduled beyond $t_{N_t}$;}
9:      return true
10: else if $\mathcal{R} \cap R_E = \emptyset$ then
11:     {$R_E$ is unreachable within $t_{N_t}$; postpone end event $e_E$:}
12:     $TIGHTEN\_BOUNDS(e_E, t_{N_t}, T_{e_E}^{\max})$
13:     return true
14: else
15:     return false
16: end if
```

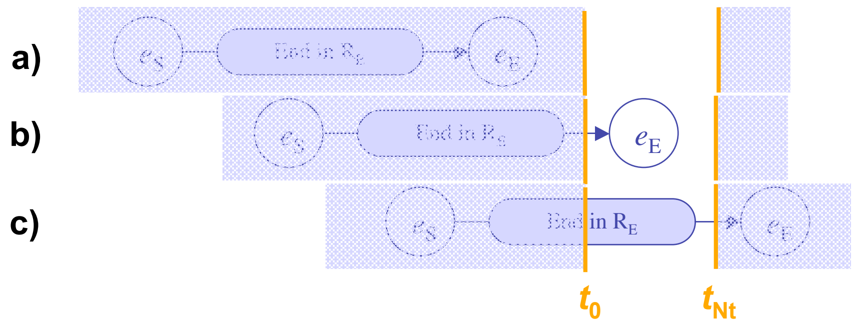


Figure 5-8: Illustration of the different cases in Alg. 9: **a)** e_E has already occurred (line 1); **b)** e_E will be scheduled within t_{N_t} (line 4); **c)** e_E will be scheduled beyond t_{N_t} (line 7).

Alg. 10 Pruning policy for the guidance constraint for an "End in state region R_E" activity ending at event e_E

```
1: if $T_{e_E}^{\max} < t_{n_t}$ then
2:     {$e_E$ will be scheduled within the horizon;}
3:     return true
4: else if $T_{e_S}^{\min} \geq t_{n_t}$ then
5:     {$e_S$ will be scheduled beyond the horizon;}
6:     return true
7: else
8:     return false
9: end if
```

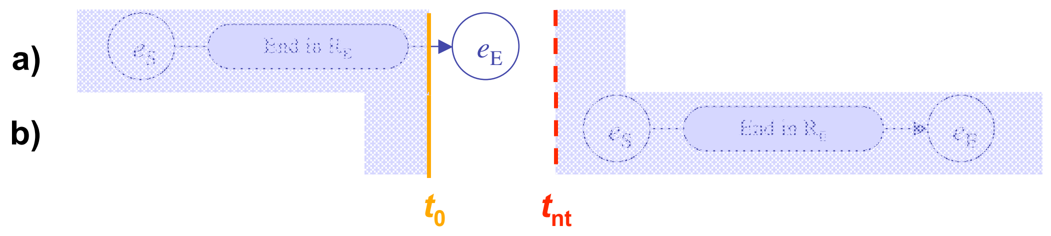


Figure 5-9: Illustration of the different cases in Alg. 10: **a)** e_E will be scheduled within the horizon (line 1); **b)** e_S will be scheduled beyond the horizon (line 4).

5.3.4 Guidance constraint pruning (Alg. 10)

In Section 4.3.2, we introduced a new HMEx constraint that uses a guidance heuristic to guide the plant towards the goal state of *end in* activities. As mentioned in Section 4.3.2, this guidance constraint is only necessary when the end event e_E is scheduled beyond the execution horizon t_{n_t}, and the start event e_S is scheduled before t_{n_t}. Therefore, this HMEx constraint can be pruned when e_E is guaranteed to be scheduled before t_{n_t} (line 1), or when e_S is guaranteed to be scheduled after t_{n_t} (line 4).

In this chapter, we presented the set of pruning polices used by Sulu in order to simplify the DLP, by pruning constraints that are irrelevant since they refer to parts of the state space that are unreachable within the current planning horizon, or parts of the qualitative state plan that can be guaranteed not to be scheduled within the current planning window. We first presented the pruning policies for the constraints imposed by the plant model (Section 5.2). Pruning such constraints involves computing a reachability set for the plant, in order to decide whether or not a given forbidden region in the plant state space is reachable within the current planning horizon. We then presented our pruning policies for the temporal constraints and the state constraints imposed by the qualitative state plan (Section 5.3). These policies involve computing bounds on the times at which events in the qualitative state plan can be scheduled, in order to predict whether or not these events will be scheduled within the current planning window. In the following chapter, we show how the use of these policies enables Sulu to run in real time.

Chapter 6

Implementation and Performance Analysis

In this chapter, we present in more detail the Hybrid Model-based Execution (HMEx) algorithm (Section 6.1), and we demonstrate it (Section 6.2) by going through the simple multi-UAV fire-fighting example from Section 3.1. We then provide a technical description of our implementation, and of the real-time, hardware-in-the-loop testbed that we use to demonstrate our executive (Section 6.3). We finally present experimental results obtained with this testbed and an analysis of the performance of our model-based executive, *Sulu*, on a more complex test case (Section 6.4).

6.1 Pseudocode for Sulu

This section presents the pseudocode for the hybrid model-based executive. The executive includes offline and online components, described, respectively, in the next two subsections.

6.1.1 Offline Algorithm (Alg. 11)

The first part of the hybrid model-based executive algorithm (Alg. 11) is executed offline. First, the lower and upper bounds ($\Delta T^{\min}_{\langle e,e' \rangle}$ and $\Delta T^{\max}_{\langle e,e' \rangle}$) on the time between

Alg. 11 Offline Algorithm

```
1: {compute the explicit time bounds on all pairs of events:}
2: for all pairs of events $\langle e, e' \rangle$ do
3:    {compute shortest paths within distance graph:}
4:    $\Delta T^{\min}_{\langle e,e' \rangle} \leftarrow -dist(e', e)$
5:    $\Delta T^{\max}_{\langle e,e' \rangle} \leftarrow dist(e, e')$
6: end for
7: {infer absolute time bounds on all events:}
8: for all events $e$ do
9:    $T^{\min}_e \leftarrow T_0 + \Delta T^{\min}_{\langle e_{start}, e \rangle}$
10:   $T^{\max}_e \leftarrow T_0 + \Delta T^{\max}_{\langle e_{start}, e \rangle}$
11: end for
12: $t_0 \leftarrow T_0$
13: {"freeze" start event to time $T_0$:}
14: $TIGHTEN\_BOUNDS(e_{start}, T_0, T_0)$
```

events e and e' are computed, for every pair of events $\langle e, e' \rangle$, by running a shortest path algorithm on the distance graph introduced in Section 5.3 (lines 1 to 6). These bounds are then used to compute the absolute time bounds $T_e^{\min}$ and $T_e^{\max}$ on every event e, given a start time T_0 for the execution of the qualitative state plan (lines 7 to 11). Recall (Section 5.3) that these absolute time bounds are used by the constraint pruning policies, to identify the portion of the plan that is relevant to each planning horizon.

Finally, we initialize receding horizon HMEx, by fixing the time t_0 of the beginning of the planning window to T_0 (line 12). T_0 corresponds to the time at which the start event e_{start} of the plan is required to be scheduled; this is enforced by calling the $TIGHTEN\_BOUNDS$ routine (Alg. 8) to set the time bounds on e_{start} to T_0 (line 14). Finally, at this point, the main loop of the online algorithm (Alg. 12) is ready to be started.

6.1.2 Online Receding Horizon HMEx Algorithm (Alg. 12)

The online component of the receding horizon HMEx algorithm is presented in Alg. 12. Following the receding horizon planning and execution framework, at each iteration of the loop, the algorithm reasons over a short planning window $[t_0, t_{N_t}]$, which is shifted by $n_t \cdot \Delta t$ at the end of each iteration (line 32). The algorithm terminates as soon as the end event e_{end} of the qualitative state plan is scheduled for execution during the next iteration (line 33).

The consecutive steps, performed during each receding horizon control cycle, are similar to the three main steps introduced in Fig. 3-9. They differ in that the first step, involving encoding the HMEx problem as a DLP, is split into two steps (Steps 1 & 4).

(Step 1) Encoding HMEx as a DLP (lines 3 to 11): As introduced in Section 5.1, the algorithm calls the pruning policy on every HMEx constraint in $\mathcal{C}_{HMEx}$, and updates the DLP accordingly. When a constraint can be pruned (line 6), then it is removed from the DLP (if it was in the DLP at the previous iteration), or simply ignored (if it was not in the DLP before). When it cannot be pruned (line 9), it is added to the DLP (if it was not in the DLP before), or updated (if it was already in the DLP, but it involves parameters whose values changed, such as t_0 or t_{N_t}).

(Step 2) Solving the DLP (lines 13 to 18): The algorithm calls the DLP solver to solve the DLP. Since the algorithm must run in real time (each iteration must last no longer than $n_t \cdot \Delta t$), the solution process is interrupted after the allocated time for solving the DLP has elapsed, and the solution retained is the best solution found thus far. Note that this solution might be sub-optimal, in the event that the solver did not have enough time to search through the complete feasible set. In the case that no solution has been found (line 17), the algorithm aborts. This happens when the qualitative state plan is simply infeasible with respect to the plant model, or when N_t was chosen too high, and as a result, the algorithm is unable to run in real time, due to the complexity of the DLP. This is discussed in more detail in Section 6.4.

(Step 3) Extracting the control sequence (line 20): This step simply consists of extracting the partial control sequence $\mathbf{U} = \langle \mathbf{u}(t_0), \ldots, \mathbf{u}(t_{n_t-1}) \rangle$ from the solution found for the DLP. Recall that the $\mathbf{u}(t_k)$ are part of the decision variables of the DLP (Eq. (4.4)). This control sequence $\mathbf{U}$ corresponds to the control inputs $\mathbf{u}$ for the first n_t steps in the current planning window, that is, up to the execution horizon.

Alg. 12 Online Receding Horizon HMEx Algorithm

```
1: repeat
2:    {(Step 1) Encode HMEx as a DLP:}
3:    for all HMEx constraints $c \in \mathcal{C}_{HMEx}$ do
4:       if $p(c) = true$ then
5:          {$c$ can be pruned:}
6:          remove $c$ from DLP / ignore $c$
7:       else
8:          {$c$ cannot be pruned:}
9:          add $c$ to DLP / update $c$ in DLP
10:      end if
11:   end for
12:   {(Step 2) Solve the DLP:}
13:   {solve under limited computation time to make sure it runs in real-time:}
14:   solve DLP for $\langle \mathbf{u}(t_0), \ldots, \mathbf{u}(t_{n_t}) \rangle$ and $T$
15:   if no solution found then
16:      {the state plan is infeasible;}
17:      abort
18:   end if
19:   {(Step 3) Extract the control sequence:}
20:   $\mathbf{U} \leftarrow \langle \mathbf{u}(t_0), \ldots, \mathbf{u}(t_{n_t - 1}) \rangle$
21:   {(Step 4) Prepare for next iteration:}
22:   for all events $e \in \mathcal{E}$ do
23:      if $t_0 \leq T(e) < t_{n_t}$ then
24:         {$e$ has been scheduled within the execution window; "freeze" event:}
25:         $TIGHTEN\_BOUNDS(e, T(e), T(e))$
26:      else if $T(e) \geq t_{n_t}$ then
27:         {$e$ has been scheduled beyond the execution horizon; postpone event:}
28:         $TIGHTEN\_BOUNDS(e, t_{n_t}, T_e^{\max})$
29:      end if
30:   end for
31:   {shift the planning window by $n_t \cdot \Delta t$:}
32:   $t_0 \leftarrow t_0 + n_t \cdot \Delta t$
33: until $T(e_{end}) < t_0$ {loop until $e_{end}$ is scheduled in the past}
```

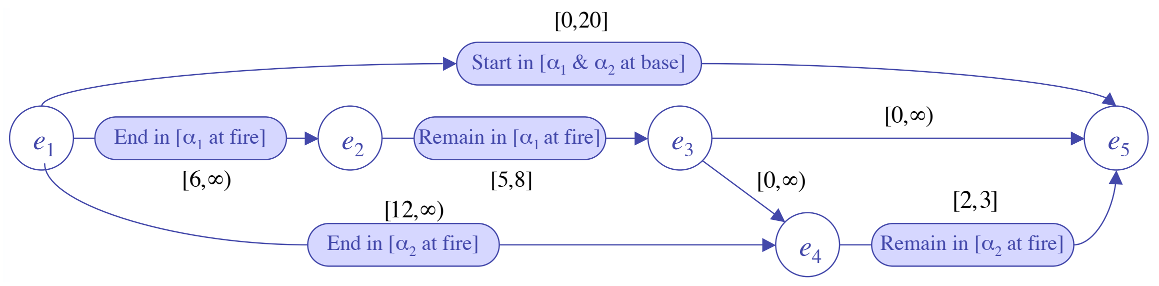


Figure 6-1: Qualitative state plan in the fire-fighting example.

(Step 4) Preparing for next iteration (lines 22 to 30): This step involves updating the time bounds on all the events in the qualitative state plan, to make sure that:

1. Events e that have just been scheduled within the current execution window are "frozen" ($T(e)$ is frozen to its current value), since they are about to be executed during the following iteration, and their execution time may no longer be modified (line 25); and

2. Events that have been scheduled beyond the execution horizon are postponed (line 28), in order to enforce that they do not get scheduled before t_0, at the next iteration.

This is accomplished by calling the routine $TIGHTEN\_BOUNDS$ (Alg. 8).

6.2 Step-by-step Algorithm Demonstration

In this section, we demonstrate the receding horizon HMEx algorithm, step by step, using the fire-fighting example introduced in Section 3.1. This simplified scenario involves two UAVs, cooperating to extinguish a fire; the map of the terrain and the qualitative state plan for this mission have been reported in Fig. 6-2 & 6-1. Alg. 11 is first run once to initialize the planner, and Alg. 12 is then run several times iteratively, shifting the horizon each time until the plan is completed, as presented in the following paragraphs.

Table 6.1: Lower and upper bounds $[\Delta T^{\min}_{\langle e,e'\rangle}, \Delta T^{\max}_{\langle e,e'\rangle}]$ on the time between any pair of events $\langle e, e'\rangle$ in the qualitative state plan in Fig. 6-1.

$e \backslash e'$	e_1	e_2	e_3	e_4	e_5
e_1	$[0, 0]$	$[6, 13]$	$[11, 18]$	$[12, 18]$	$[14, 20]$
e_2	$[-13, -6]$	$[0, 0]$	$[5, 8]$	$[5, 12]$	$[7, 14]$
e_3	$[-18, -11]$	$[-8, -5]$	$[0, 0]$	$[0, 7]$	$[2, 9]$
e_4	$[-18, -12]$	$[-12, -5]$	$[-7, 0]$	$[0, 0]$	$[2, 3]$
e_5	$[-20, -14]$	$[-14, -7]$	$[-9, -2]$	$[-3, -2]$	$[0, 0]$

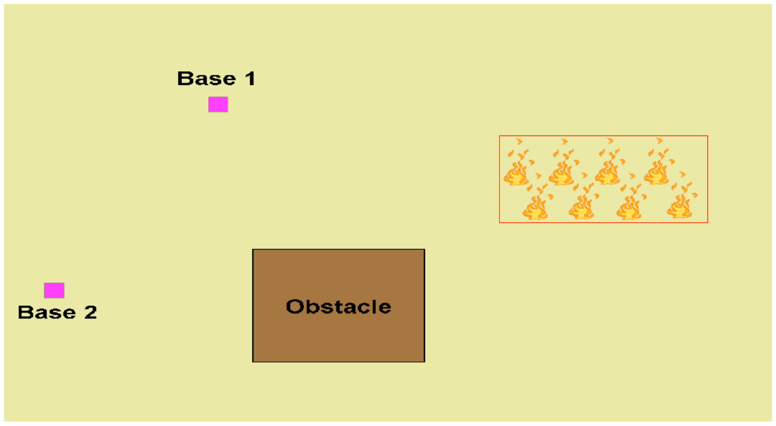


Figure 6-2: Map of the terrain for the fire-fighting example.

6.2.1 Initialization

Initialization is performed by running Alg. 11. First, the table of values for $\Delta T^{\min}_{(e_1,e_2)}$ and $\Delta T^{\max}_{(e_1,e_2)}$ (Table 6.1) is obtained, by computing the shortest paths through the distance graph, corresponding to the qualitative state plan (lines 1 to 6). The absolute time bounds, $T^{\min}_e$ and $T^{\max}_e$, for each event e are then inferred from the provided execution start time, $T(e_1) = T_0 = 0$ (lines 7 to 11). Next, the beginning of the first planning window is set to $t_0 = T_0$ (line 12). Finally, the start event e_1 is "frozen" to $T(e_1) = T_0$ (line 14) in order to enforce that it always be scheduled at time $T(e_1) = T_0$ in the following iterations. The $TIGHTEN\_BOUNDS$ routine (Alg. 8) then propagates the change to the bounds on the other events. The resulting new bounds are presented in Fig. 6-3 a).

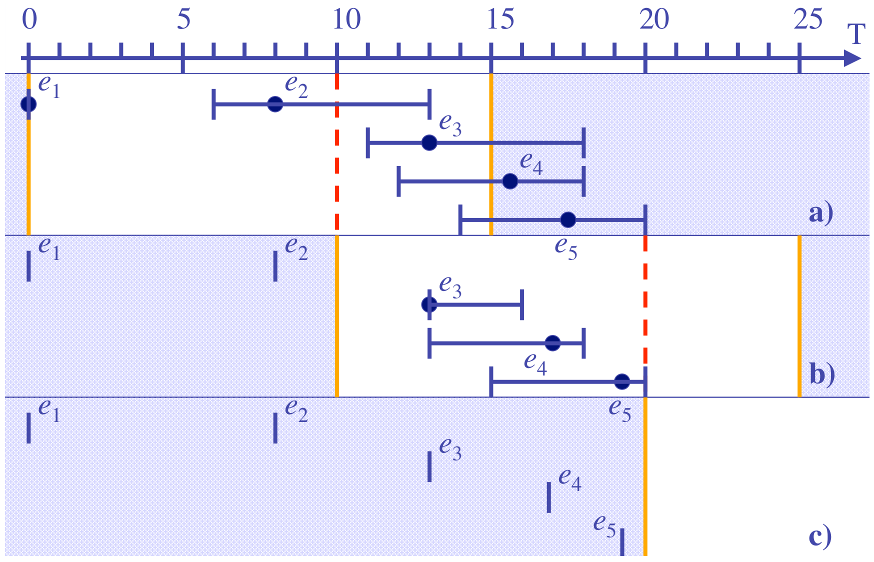


Figure 6-3: "Snapshots" of the schedule for the qualitative state plan in Fig. 3-5, at different steps in the algorithm execution: a) First iteration ($t_0 = T_0 = 0$), Step 3 (line 20); b) Second iteration ($t_0 = T_0 + n_t \cdot \Delta t = 10$), Step 3 (line 20); c) End of second iteration ($t_0 = T_0 + 2n_t \cdot \Delta t = 20$, 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 $T(e)$.

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

First, the pruning policy is run on every HMEx constraint (lines 3 to 11), in order to determine which constraints should be encoded in the DLP, and which constraints can be pruned. For instance, the time bounds in Fig. 6-3 a) guarantee that e_2 will be scheduled within the planning horizon; this fact is used by the pruning policy in Alg. 10, to prune the heuristic constraint for activity "End in [α_1 at fire]" (Alg. 10, line 1). The same fact is used by the pruning policy in Alg. 9, to infer that the *end in* state constraint, imposing aircraft α_1 to be at the fire at event e_2, must not be pruned (Alg. 9, line 1).

Next, the resulting DLP is solved (Step 2, lines 13 to 18), and the new control sequence **U** is extracted from the solution found to the DLP (Step 3, line 20). The corresponding schedule is presented in Fig. 6-3 a). Note that, although the bounds on $T(e_2)$ inferred from the temporal constraints in the qualitative state plan allowed event e_2 to be scheduled as early as $T(e_2) = 4$, it was scheduled at time $T(e_2) = 8$,

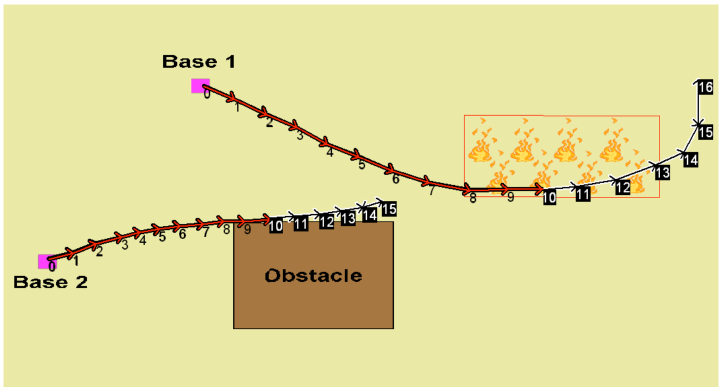


Figure 6-4: Trajectories computed at the first iteration (in bold: up to t_{n_t}; in light: between t_{n_t} and t_{N_t})

which is the earliest time that aircraft α_1 could get to the fire (Fig. 6-4), given the constraint on its maximum allowed velocity. As a consequence, event e_3 was scheduled at time $T(e_3) = 13$, in order to satisfy the temporal constraint specifying that e_3 must be scheduled no earlier than 5 time units after e_2. Fig. 6-4 also shows that aircraft α_2 could not get to the fire within the current planning horizon; following the encoding in Eq. (4.27), event e_4 was, hence, postponed to $t_{N_t} + \frac{\Delta t}{2} = 15.5$, with $\Delta t = 1$ time unit. The earliest time that the end event e_5 could be scheduled was then $T(e_5) = 17.5$, according to the temporal constraint, specifying that e_5 may not be scheduled earlier than 1 time unit after e_4.

Finally (Step 4, lines 22 to 30), the algorithm prepares for the next iteration, by tightening the bounds on any event that can be tightened. In our example, event e_2 has been scheduled within the current execution window; it is, hence, frozen to its current scheduled time of $T(e_2) = 8$ (line 25). The routine $TIGHTEN\_BOUNDS$ then propagates this to the other events; the resulting new bounds are illustrated in Fig. 6-3 b).

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

As argued in Section 3.5.1, the receding horizon framework that we use to solve the HMEx problem allows Sulu to adapt to disturbances and unforeseen events. In this section, we illustrate this capability by assuming that, from the first to the second iteration, new updated information about the fire has been gathered, for instance, from analysis of satellite imagery, and that the fire region turns out to be narrower than first expected (Fig. 6-5). This information is used to update the qualitative state plan, by updating the state constraints on the activities mentioning the fire. Note that the trajectories, initially designed at the previous iteration (Fig. 6-4), would no longer satisfy the qualitative state plan, since they would lead aircraft α_1 to remain in the fire region for less than 5 time units. Hence, while the vehicles are following the trajectories uploaded at the end of the previous iteration, up to the execution horizon t_{n_t} only, Sulu needs to design new trajectories that start from t_{n_t}, and take into account this change in the plan.

Following Alg. 12, the DLP is first updated (Step 1, lines 3 to 11), by pruning the HMEx constraints that are no longer necessary, and by adding the constraints that can no longer be pruned. For instance, the constraint that imposes aircraft α_1 to be at the fire at event e_2 is pruned and removed from the DLP, since e_2 has already been executed (Alg. 9, line 1).

The algorithm then solves the updated DLP (Step 2, lines 13 to 18), and the new control sequence $\mathbf{U}$ is extracted from the solution found to the DLP (Step 3, line 20). The corresponding trajectories are illustrated in Fig. 6-5. Note that they are different from the ones initially computed at the previous iteration, due to the change in the shape of the fire region. The new trajectory for aircraft α_1 now satisfies the qualitative state plan, since α_1 remains in the fire region for 5 time units, as specified by the temporal constraint on the "Remain in [α_1 at fire]" activity.

The schedule corresponding to these new trajectories is presented in Fig. 6-3 b). Event e_3 was scheduled at time $T(e_3) = 13$, similar to the previous iteration, contrary to event e_4, which was scheduled at time $T(e_4) = 17$. This is later than the earliest

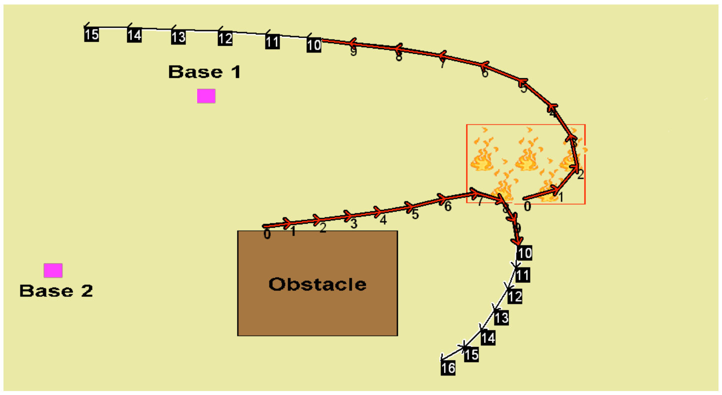


Figure 6-5: Trajectories computed at the second iteration (in bold: up to t_{n_t}; in light: between t_{n_t} and t_{N_t})

time $T_{e_2}^{\min} = 13$ allowed by the temporal constraints in the qualitative state plan, since the upper bound on α_2's velocity prevented it from reaching the fire earlier. Event e_5 was delayed accordingly. Note that events e_1 and e_2 were not scheduled at all: since they have already been executed, their time variables, $T(e_1)$ and $T(e_2)$, have been pruned from the DLP, using the pruning policy in Alg. 6.

The time bounds on events are then tightened, when possible, following the method in Step 4 (lines 22 to 30). In this case, events e_3, e_4 and e_5 have all been scheduled within the current execution horizon, so they are "frozen" to their current respective times (line 25). The resulting, new bounds are presented in Fig. 6-3 c). The planning window is then shifted by $n_t \cdot \Delta t$ (line 32), and the algorithm terminates, since the end event e_5 is now guaranteed to be scheduled before $t_0 = T_0 + 2 \cdot n_t \cdot \Delta t = 20$ (line 33).

6.3 Implementation on a UAV Testbed

Sulu has been implemented in C++, using the commercial software *Ilog CPLEX* [2] to solve the DLPs. Note that, in order to solve a DLP, CPLEX first reformulates it

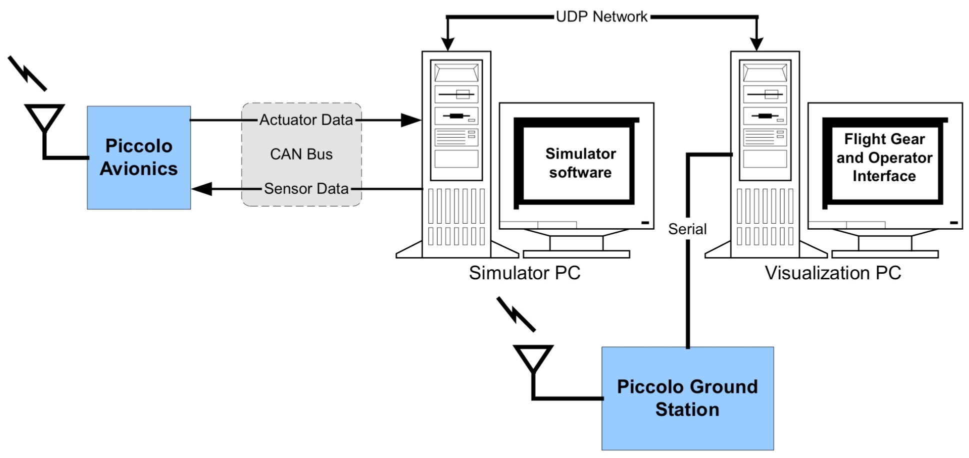


Figure 6-6: Architecture of the CloudCap testbed. *This picture was taken from [15]*

into a Binary Integer Program (BIP), and then uses traditional branch-and-bound techniques to solve the BIP. Current work [34, 42, 43] has showed that one can use more efficient, conflict-directed techniques to solve the DLP, by solving it directly in order to exploit its structure, rather than first convert it into a BIP. However, the DLP solver algorithm introduced in [42] can currently only handle DLPs in Conjunctive Normal Form (Def. 17), and, even though algorithms exist in order to reduce any DLP in propositional form (Def. 18) into CNF, this reduction is worst-case exponential in time and space. In practive, tests have showed that our computer would run out of memory when it tried to perform this reformulation on DLPs generated by the test case presented in Section 6.4.1. For this reason, we have been using CPLEX, until the DLP solver in [42] can handle DLPs in general propositional form.

Our receding horizon HMEx algorithm has been tested on a 1.7GHz computer with 512MB RAM, using a hardware-in-the-loop testbed that consists of two *Cloud-Cap Piccolo Plus autopilots* (Fig. 6-6, left) [1]. Each autopilot is a complete integrated avionics system, including GPS and a flight sensor interface. Typically, an autopilot would be mounted onboard a model aircraft, and provide low-level control by taking in sensor measurements (static pressure, dynamic pressure, inertial data...) and sending control inputs to the onboard actuators. The operator communicates with the autopilot via radio wavelength, through a *Piccolo ground station* (Fig. 6-6, bottom). The ground station is connected via a serial port to a PC running the *operator interface* (Fig. 6-6, right), which enables the operator to monitor the state of the aircraft, and to send lists of waypoints to the autopilots.

In order to test Sulu, rather than mount the autopilots on real model aircrafts, the autopilots interact with a *simulator* through a CAN bus (Fig. 6-6, center). The simulator takes in control inputs from the autopilots, applies them to an aircraft dynamics model, and simulates in real time the sensor measurements needed by the autopilots. The model used is a very realistic model of the dynamics of the aircraft; hence, this real-time, hardware-in-the-loop setup provides a simulation testbed that is only one step away from real flight testing.

Furthermore, as previously mentioned, the autopilots take in series of waypoints, rather than low-level control inputs. This enables us to abstract away the problem of low-level control of the aircraft, and to use the simple plant model introduced in Section 3.2.1. Sulu designs trajectories in the plant's state space, which are then converted into sequences of close x/y waypoints. The sequences are sent through a TCP/IP interface to the operator interface, which relays them to the autopilots. The operator interface also relays information about the state of the aircraft to the executive through the same interface.

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

In this section, we first present a multiple-UAV fire-fighting scenario (Section 6.4.1) that is more complex than the one previously presented (Section 3.1). This sce-

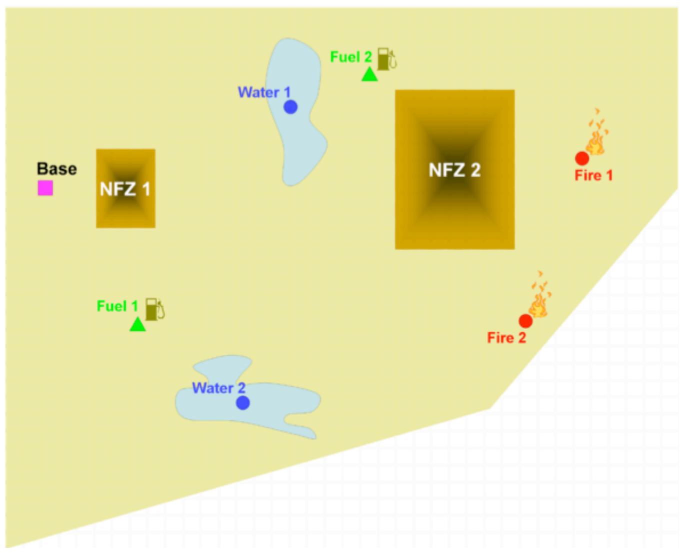


Figure 6-7: Map of the environment in the multi-UAV fire-fighting scenario.

nario is the test case that we then use to discuss the real-time performance of Sulu (Section 6.4.2).

6.4.1 Description of the Test Case

In Section 3.1, we introduced a simple multiple-UAV fire-fighting example that we used to illustrate the concepts and algorithms presented in this thesis. In this section, we introduce a similar, more complex example, used in Section 6.4 to assess the performance of Sulu. This scenario involves two aircraft, executing a qualitative state plan consisting of 26 activities, for a total mission duration of about 1,300 sec. As shown in Fig. 6-7, the environment consists of two no-fly-zones, and a set of goal waypoints that the aircrafts must visit in a specific order, according to the qualitative state plan. The following is a short, natural language description of the plan, represented in Fig. 6-8.

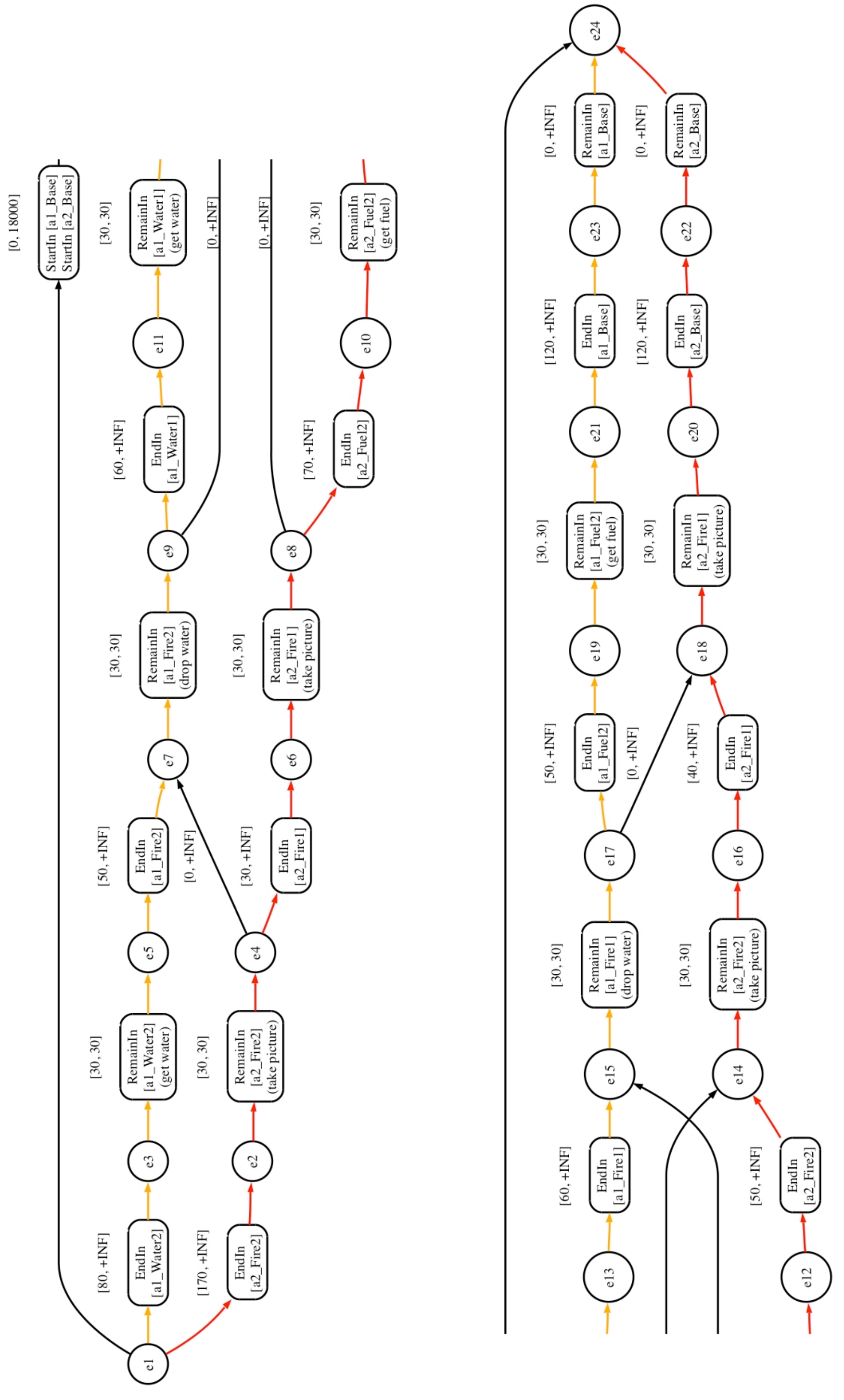


Figure 6-8: Qualitative state plan used for performance analysis.

The two aircraft α_1 and α_2 start at a common Base*. Aircraft α_1 is a water tanker; its mission is to first fill up its water tank at* Water 2 *and drop it on* Fire 2*. It must then refill its water tank at* Water 1*, and drop water on* Fire 1*. Finally, it must go back to the* Base*, refilling its fuel tank along the way at* Fuel 2*. Aircraft α_2 is a reconnaissance UAV; its mission is to take a picture of each fire, before and after aircraft α_1 drops water on it. It must start with* Fire 2 *and then* Fire 1*, refilling its fuel tank between the two rounds at* Fuel 2*. Each time they perform an action at a specific waypoint (e.g. refilling a tank, or taking a picture), the aircraft must remain in the vicinity of the corresponding waypoint for 30 sec. The overall mission duration must be lower than 18,000 sec.*

For this test case, the objective function corresponds to minimizing total plan execution time.

6.4.2 Performance Analysis

Fig. 6-9 and 6-10 present an analysis of the performance of Sulu on the test case introduced in Section 6.4. These results show runtimes for a single iteration of the receding horizon HMEx algorithm, which were obtained by averaging runtimes over the whole plan execution, and over 5 different runs with random initial conditions. At each iteration, the computation was cut short if and when it passed 200 sec, and we retained the best solution to the DLP found thus far.

In both figures, the x axis corresponds to the length of the execution horizon, $n_t \cdot \Delta t$, in seconds. For these results, we maintained a planning buffer of $t_{N_t} - t_{n_t} =$ 10 sec. The y axis corresponds to the average time in seconds required by CPLEX to solve the DLP at each iteration. As shown in Fig. 6-9, the use of pruning policies entails a significant gain in performance. Note that these results were obtained by disabling the *Presolve* function in CPLEX, which also internally prunes some of the constraints in order to simplify the DLP.

Fig. 6-10 presents an analysis of Sulu's capability to run in real time. The dotted

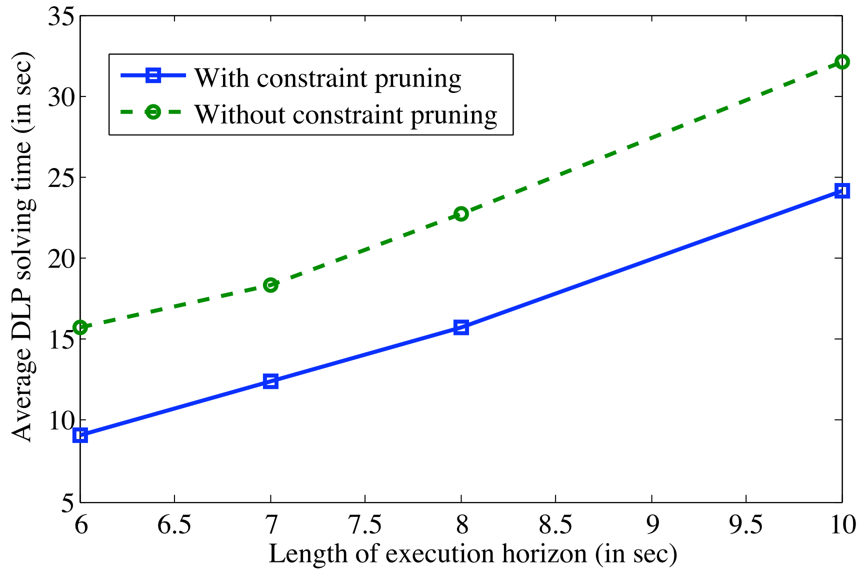


Figure 6-9: Performance gain by constraint pruning.

line is the line $y = x$, corresponding to the real-time threshold. It shows that below the value $x \simeq 7.3s$, Sulu is able to compute optimal control sequences in real time, since the average DLP solving time is below the length of the execution horizon. For longer horizons, corresponding to values of x above $7.3s$, CPLEX is unable to find optimal solutions to the DLPs before Sulu has to replan, that is, the solving time is greater than the execution horizon. Note that in this case, since CPLEX performs branch and bound, which runs as an anytime algorithm, we can still interrupt it and use the best solution found thus far, in order to generate sub-optimal control sequences.

Also note that the number of the disjunctions in the DLP grows linearly with the length of the planning horizon; therefore, the complexity of the DLP is worst-case exponential in the length of the horizon. In Fig. 6-10, however, the relationship appears to be linear. This can be explained by the fact that the DLP is very sparse, since no disjunct in the DLP involves more than three or four variables.

In this chapter, we first presented in detail the pseudocode for our receding horizon, hybrid, model-based executive (Section 6.1). We showed that the algorithm could be

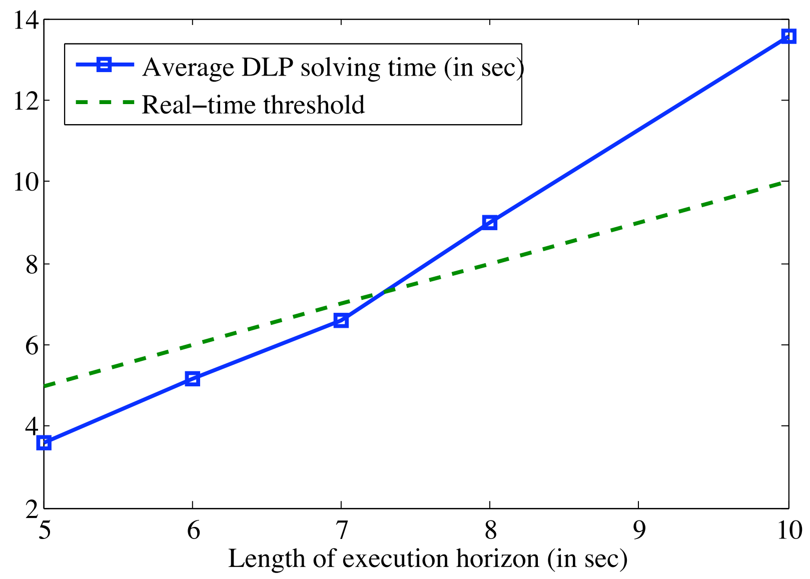


Figure 6-10: Performance of Sulu.

decomposed into an initial offline phase, during which we compute the absolute time bounds on events that are needed by the pruning policies, and an online phase, that generates control sequences for the plant in real-time, over shifting planning windows. In Section 6.2, we then demonstrated our algorithm step by step, on the simple multiple-UAV fire-fighting introduced in Section 3.1. We showed that our receding horizon approach enabled Sulu to adapt to disturbances, by replanning regularly, taking into account the latest knowledge about the state of the plant, and possible unforeseen events. We then briefly described how we implemented Sulu and tested it using a real-time, hardware-in-the-loop UAV testbed (Section 6.3). Finally, in Section 6.4, we presented some real-time performance analysis, which showed that our executive was able to run in real-time, on a more complex UAV scenario.

Chapter 7

Conclusion and Future Work

In this chapter, we propose areas of future work, in order to improve and extend the capabilities of the HMEx algorithm in general (Section 7.1.1), and, more specifically, the constraint pruning algorithms, presented in Chapter 5 (Section 7.1.2). We also discuss how Sulu can be integrated with related work in model-based programming and contingent, temporal plan execution (Section 7.1.3), in order to enable higher-level coordination and control of cooperative systems. Finally, we conclude this thesis by summarizing the capabilities and achievements that we have presented in previous chapters (Section 7.2).

7.1 Future Work

7.1.1 Improvements and Extensions of the HMEx Algorithm

In this section, we describe improvements that could be made to our HMEx algorithm. First, we describe how it could be extended to handle probabilistic state estimates from the state estimator; we then present possible extensions with respect to handling uncertainty and uncontrollable events in the qualitative state plan. Finally, we show how the algorithm could be distributed, in order to be deployed on plants with multiple subplants, such as a team of UAVs.

Probabilistic Control Sequence Generation

As mentioned in Section 3.4.2, in this thesis, we abstracted away the state estimation problem, by assuming that the hybrid controller had a unique, non-probabilistic knowledge of the state of the plant at all times. In practice, this means that the state estimator outputs the most likely state that is consistent with the plant model, the observations from the plant, and the control sequences previously sent to the plant. The hybrid controller then assumes that this most likely state is the true state, and designs control sequences that start from this initial state.

This approach is not always applicable, since the maximum likelihood assumption does not always hold. For instance, consider the case where the state estimator infers from the observations that the plant may be in only two possible states, with probabilities 51% and 49%, respectively. In that case, Sulu would assume that the plant is in the first state. However, there is a high probability that it is in the second state, and that the control sequence generated by Sulu, based on the wrong assumption, will drive the plant into a forbidden region of the state space. Hence, Sulu is not robust to state estimation errors.

In order to make it more robust, future work could remove the maximum likelihood assumption, by allowing the hybrid controller to take in a *belief state*, consisting of a probability distribution over the plant's state space. Given such a belief state, computed by the state estimator, the function of the hybrid controller would then be to generate control sequences that are guaranteed to satisfy the plant model and to complete the qualitative state plan, with some given probability α. Alternatively, it could also design control sequences that minimize the probability of failure, either due to "collision" with a forbidden region of the state space, or due to the inability to complete the qualitative state plan.

Qualitative State Plans with Uncertainty

Another assumption we have made in this thesis is that Sulu is free to choose a schedule for the qualitative state plan, as long as it satisfies the plan's temporal constraints and is consistent with the control sequence. In other words, we have assumed that all events in the qualitative state plan are controllable, since Sulu is free to choose the time at which they are executed. In some applications, however, the model-based executive might not have full control over one or more events, because the time at which they are executed is given by nature, or controlled by another system in the environment. For instance, in the multiple-UAV fire-fighting example, one event in the qualitative state plan might be required to be synchronized perfectly with an external event over which the UAVs have no control, such as the arrival of a ground support unit. Alternatively, the event could correspond to the end event of an activity whose duration is not exactly known in advance. For instance, the exact time needed by a UAV to refuel might only be known within certain time bounds.

Previous work on *simple temporal networks with uncertainty* (*STNUs*) have tackled the problem of providing guarantees on the existence of a temporally consistent schedule for the plan, regardless of the outcome of the uncontrollable events [48, 50, 58, 60, 61]. For instance, they define the concepts of *strong controllability* and *dynamic controllability*. An STNU is *strongly controllable* if one can compute, beforehand, a schedule for all the controllable events, such that, for all possible execution times for the uncontrollable events, the complete schedule is always temporally consistent. Similarly, an STNU is *dynamically controllable* if one can compute, *on the fly*, a schedule for the controllable events, knowing the execution times of past uncontrollable events, such as the overall schedule is temporally consistent.

Future work could look into applying these concepts to Sulu, in order to allow it to reason on *qualitative state plans with temporal uncertainty*.

Distributed HMEx Algorithm

The algorithm described in this thesis is fully centralized; this means that it must be executed on a single processor, which has control over the whole plant. In the case of a plant that consists of multiple vehicles, for instance, multiple UAVs coordinating to extinguish fires, this fully centralized architecture might not be applicable, due to communication limitations: all aircraft might not be able to communicate with a central processor at all times, and communication latency might prevent this centralized architecture from meeting real-time requirements. Furthermore, this approach does not scale well with the number of UAVs, since the computational power needed to run the algorithm might become too high for a single processor.

In this case, it becomes necessary to distribute the algorithm, so that it can run on different processors, for instance at different ground stations, or even onboard each of the vehicles. The main challenge in order to distribute the algorithm is to be able to split the receding horizon HMEx problem into several, loosely coupled subproblems. The state equations in the plant model could easily be decoupled across different vehicles, assuming that the state equation for a state variable related to a given aircraft does not depend on state variables related to different aircraft. Forbidden regions pertaining to a subspace of the state space corresponding to a given vehicle can also be easily decoupled. Coupling is introduced by state equations or forbidden regions involving state variables corresponding to different vehicles. The qualitative state plan can also introduce coupling between HMEx subproblems, either through state constraints that involve state variables corresponding to different vehicles, or through the temporal constraints in the plan, which specify synchronization between the aircraft.

Previous work has been done on distributed branch-and-bound algorithms, in order to solve MILPs in a distributed fashion, on parallel processors [47]. Such distributed algorithms, however, usually heavily rely on the fact that all processors can communicate with each other, and that communication is almost instantaneous, so that the capability of the algorithm to run in real time is not threatened by communication delays.

Previous work has tackled the problem of decoupling STNs for parallel scheduling and execution, under communication limitations [58]. Note, however, that this work only dealt with STNs, rather than with qualitative state plans. For this reason, this work only applies when the coupling constraints in the qualitative state plan are temporal constraints. Under this assumption, [58] decouples STNs using a hierarchical reformulation and decoupling algorithm, which identifies a set of *group plans* that describe the desired behavior of a set of sub-plants, and an overall *mission plan*, which introduces temporal coupling between the different group plans. Based on a strong controllability property of the mission plan, the algorithm compiles out the coupling temporal constraints in the mission plan, in order for each sub-plant to be able to execute its own group plan, without the need to communicate with other sub-plants. This approach could be used to distribute the HMEx algorithm, when the only coupling constraints in the qualitative state plans are temporal constraints.

7.1.2 Improvements on the Constraint Pruning Framework

In this section, we describe improvements on the HMEx algorithm that are specific to the constraint pruning framework presented in Chapter 5. We first suggest a small, incremental improvement on the pruning policies for temporal constraints, presented in Section 5.3.1. We then introduce possible threads of future work that would require more fundamental changes to the overall pruning framework, in order to make it more efficient.

Incremental Improvement on the Temporal Constraint Pruning Policies

As mentioned in Section 5.3.1, the pruning policy for temporal constraints in the qualitative state plan uses a method that consists of *compiling out* the constraints that are irrelevant, given the current planning window. To do so, we explicitly encode all binary temporal constraints between all events in the plan, and all unary temporal constraints on any given event. Then, we can prune a constraint if it involves an event that is guaranteed to be scheduled outside the current planning window, without loss of correctness.

However, some of the remaining temporal constraints that have not been pruned might be redundant. One could use an edge trimming algorithm similar to [17], on the graph corresponding to the qualitative state plan, in order to further prune some of the remaining temporal constraints, until no constraint can be pruned, without loss of correctness. This method would effectively be equivalent to computing a *minimal* *dispatchable plan* for the qualitative state plan, as described in Section 2.1.

This incremental improvement to the pruning policies for constraints imposed by the qualitative state plan is likely to only lead to a small gain in efficiency. In the following subsection, we describe how one could modify more radically the pruning framework, in order to lead to greater improvements in efficiency.

Proposed Changes to the Pruning Framework

One possible improvement to the current pruning framework is motivated by the following simple observation. Consider a forbidden region R in the plant's state space, and consider that the planning horizon is $N_t = 10$ time steps. The plant is required to remain outside of R at all time steps within the planning window; hence, our current pruning policy for the corresponding HMEx constraint returns `true`, that is, the constraint is prunable, if and only if R is out of reach of the plant within the current planning horizon, in which case the plant is guaranteed to remain outside of R at all time steps in $[t_0, t_{N_t}]$. However, consider the case when R is unreachable within 5 time steps, but becomes reachable at the 6th time step. In that case, the HMEx constraint (Eq. (4.6)) would not be pruned by our current pruning policy, although the part of Eq. (4.6) corresponding to the first 5 time steps could be pruned, since the plant is guaranteed to remain outside of R during these first 5 time steps. This suggests that, instead of applying the pruning policies to the HMEx constraint that specifies that the plant should remain outside of R at all time steps, the policy should be applied to the HMEx "sub-constraints" that specify that the plant should be outside of R at a specific time step t_k in the planning window.

More generally, this suggests a fundamental change to the current pruning framework: rather than using pruning policies, in order to decide whether or not an HMEx constraint can be pruned, one could instead apply *simplification policies*. In that context, pruning or not pruning a constraint corresponds to two ends of a spectrum; simplification policies would allow cases in the middle of that spectrum, where constraints might be pruned only partially. This idea applies to all HMEx constraints, not only the constraints corresponding to forbidden regions in the state space, as described in the previous paragraph. For instance, consider the HMEx constraint that encodes an *end in* activity, ending at event e_E (Eq. (4.25)). If the upper bound on $T(e_E)$ verifies $T_{e_E}^{\max} < t_k - \frac{\Delta t}{2}$ for some time step t_k in the planning window, then it follows that $T(e_E)$ is guaranteed to be strictly smaller than $t_k - \frac{\Delta t}{2}$, and Eq. (4.25) can be simplified, by replacing the disjunct $T(e_E) \geq t_k - \frac{\Delta t}{2}$ by `false`, since it is guaranteed to be violated. This would correspond to simply removing $T(e_E) \geq t_k - \frac{\Delta t}{2}$ from the disjunction. To do this in a systematic manner, one could design a routine which would parse all the DLP constraints, and replace by `true` (or `false`) the linear equalities or inequalities that are guaranteed to be satisfied (or violated, respectively).

The concept of a constraint simplification policy relates to similar ongoing work in model-based programming for continuous, under-actuated plants [26]. As introduced in Section 3.5.3, the model-based executive in [26] also takes in a qualitative state plan. The executive then uses constraint tightening techniques based on an analysis of the plant model, in order to tighten/simplify the constraints mentioned in the state plan. For instance, consider an activity in the plan that specifies a goal region R for the plant. Some parts of R might not be reachable by the plant, because they violate one or more of the forbidden regions in the plant's state space. These parts can then be removed from R without loss of correctness; this can be done by replacing R with the subtraction from R of all the regions in the set $\mathcal{F}$ of forbidden regions. Temporal constraints can also be tightened; for instance, consider a "Remain in state region R_1" activity, followed by an "End in state region R_2" activity, with lower time bound 0. The lower time bound on the second activity can be tightened, by computing an optimistic approximation of the time required by the plant to go from R_1 to R_2. This constraint tightening technique could be used to improve the pruning framework presented in this thesis, by pruning state sequences that are allowed by the qualitative state plan, but disallowed by the plant model.

Other work on knowledge compilation [20] could also be applied to make the pruning process more efficient. Instead of systematically going through the list of all the constraints in the DLP, and applying the pruning policies to each constraint in order to determine whether or not it can be pruned, one could compile the constraints into a hierarchical, spacial data structure, which would describe what part of the state space is disallowed by each constraint. Using this pre-computed, structural decomposition of the problem, one could more efficiently look up which constraints are applicable and which constraints can be pruned, given the current planning window.

7.1.3 Integration with an HTN Planner

Finally, in this section, we present ongoing work aiming at integrating Sulu with a model-based executive for contingent temporal plans, called *Kirk* [32], presented in Section 2.2.1. Kirk has been designed to generate the input qualitative state plan for Sulu, in very much the same way as a *control sequencer* is used to generate the input for Titan's *Mode Reconfiguration* component [66], described in Section 2.2.1. This further elevates the level of interaction between the human operator and the plant, by allowing the operator to control the plant through an RMPL program, which describes the desired behavior of the plant, at a very high, abstract level. It also provides more robustness to the system, by allowing it to recover from failures, when the HMEx problem is found infeasible. When this happens, Sulu passes the reason for infeasibility back to Kirk, which chooses a contingent sequence of activities to execute, in order to recover from the failure. However, as mentioned in Section 2.2.1, one important difference with [66] is that Titan is a purely reactive model-based executive, while Kirk and Sulu are model-predictive, since they use a model of the plant in order to plan control sequences into the future.

Section 3.5.3 also suggested the integration of Sulu with another model-based executive introduced in [26], which would enable Sulu to be more robust to frequent, low-level disturbances, by using classical PID controllers to control the plant, and designing gains and setpoints for these controllers.

7.2 Conclusion

In this thesis, we have presented a receding horizon, hybrid, model-based executive, called *Sulu*, which is capable of performing robust, model-based execution of temporally flexible plans, on continuous dynamical systems (or *plants*), such as cooperative vehicles or chemical plants. Sulu enables the human operator to control the plant at an abstract, task level, by specifying the desired state evolution of the plant, in the form of a *qualitative state plan*. A qualitative state plan describes families of allowed state trajectories for the plant. Sulu reasons over a model of the plant, in order to continuously generate optimal trajectories that are consistent with this plan.

The main innovation in this thesis is that we perform temporal plan execution, on under-actuated plants with continuous dynamics and hidden state. While previous work has tackled the problem of temporal plan execution [7, 13, 49, 60, 61, 65], it has only been applied to plants that are described using discrete models. Previous work in model-based programming [66] also presented a framework to control under-actuated systems with hidden state, but their model-based executive was designed only for discrete systems. In this thesis, we have described how we extend these two threads of research in order to handle plants with continuous dynamics. This includes hybrid systems, whose continuous dynamics are dependent on discrete modes the plant can be in. We use a *receding horizon* approach [10, 24, 39, 51, 52, 54, 57] in order to perform robust execution of the qualitative state plans, and we achieve real-time performance through the use of novel *constraint pruning policies*.

A qualitative state plan involves *activities* that specify abstract, qualitative regions in the state space that the plant must go through. The plan pieces these activities together in the form of a temporal plan, using flexible temporal 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 use of such an abstract, qualitative goal specification for the plant elevates the interaction with the plant, so as to enable the human operator to perform high-level, supervisory control of the system. It also delegates more control authority to the model-based executive, giving it more opportunities to achieve the goal in an optimal fashion, and giving it also more room to adapt to disturbances and unforeseen events.

Sulu formulates the problem of designing optimal control sequences for the plant (*HMEx problem*) as a *Disjunctive Linear Program* (*DLP*). The DLP formalism is a mixed logic/optimization mathematical formalism that enables us to encode both the logical, decision-making component of HMEx, due to non-convex constraints imposed by the plant model and the qualitative state plan, and its continuous, optimization component, due to the continuous plant dynamics.

We achieve robustness and tractability by interleaving planning and execution, following a receding horizon framework; this enables Sulu to plan partial control sequences, and reactively revise these sequences in order to adapt to disturbances and unforeseen events, by replanning regularly, taking into account the latest knowledge about the state of the world. We use a set of novel constraint pruning policies to enable real-time execution, by pruning some of the constraints in HMEx, and hence, simplifying the problem. We demonstrated these capabilities on a real-time, hardware-in-the-loop testbed, in the context of a multiple-UAV fire-fighting scenario.

Appendix A

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

In this appendix, we formally prove the equivalence between the intuitive encoding for the plant state equation (Eq. (4.8)), and the encoding we use in the DLP (Eq. (4.9)). It is important that the two encodings be perfectly equivalent, in order to make sure that choosing one over the other does not introduce new solutions to the problem, nor remove solutions from the original problem. For this purpose, we formally show that the DLP formula in Eq. (4.8) is equal to the formula in Eq. (4.9).

Let p_j be the proposition $\langle \mathbf{s}, \mathbf{u} \rangle(t_{k-1}) \in S_j$, and q_j be $\mathbf{s}(t_k) = \mathbf{F}_{|S_j}(\mathbf{s}(t_{k-1}), \mathbf{u}(t_{k-1}))$. Then Eq. (4.8) is equivalent to $\bigwedge_k \left\{ \bigwedge_j \{p_j \Rightarrow q_j\} \right\}$. Since $\mathcal{S}$ is a partition of S, it must completely cover S, hence $\bigvee_i p_i = \texttt{true}$. Therefore:

$$\begin{aligned}
\textstyle\bigwedge_j \{p_j \Rightarrow q_j\} &= \{\textstyle\bigvee_i p_i\} \wedge \left\{\textstyle\bigwedge_j \{p_j \Rightarrow q_j\}\right\} \\
&= \textstyle\bigvee_i \left\{p_i \wedge \left\{\textstyle\bigwedge_j \{p_j \Rightarrow q_j\}\right\}\right\} \\
&= \textstyle\bigvee_i \left\{p_i \wedge \left\{\textstyle\bigwedge_j \{\neg p_j \vee q_j\}\right\}\right\} \\
&= \textstyle\bigvee_i \left\{\textstyle\bigwedge_j \{p_i \wedge \{\neg p_j \vee q_j\}\}\right\} \\
&= \textstyle\bigvee_i \left\{\textstyle\bigwedge_j \{\{p_i \wedge \neg p_j\} \vee \{p_i \wedge q_j\}\}\right\} \\
&= \textstyle\bigvee_i \left\{\{\{p_i \wedge \neg p_i\} \vee \{p_i \wedge q_i\}\} \wedge \left\{\textstyle\bigwedge_{j\neq i} \{\{p_i \wedge \neg p_j\} \vee \{p_i \wedge q_j\}\}\right\}\right\} \\
&= \textstyle\bigvee_i \left\{\{p_i \wedge q_i\} \wedge \left\{\textstyle\bigwedge_{j\neq i} \{\{p_i \wedge \neg p_j\} \vee \{p_i \wedge q_j\}\}\right\}\right\} \\
&= \textstyle\bigvee_i \{\{p_i \wedge q_i\} \wedge \phi_i\}
\end{aligned}$$

Furthermore, since $\mathcal{S}$ is a partition of S, $\neg p_j = \bigvee_{l\neq j} p_l$ for all j; therefore:

$$\begin{aligned}
\phi_i &= \textstyle\bigwedge_{j\neq i} \left\{\left\{p_i \wedge \left\{\textstyle\bigvee_{l\neq j} p_l\right\}\right\} \vee \{p_i \wedge q_j\}\right\} \\
&= \textstyle\bigwedge_{j\neq i} \left\{\left\{\textstyle\bigvee_{l\neq j} \{p_i \wedge p_l\}\right\} \vee \{p_i \wedge q_j\}\right\}
\end{aligned}$$

Since $\mathcal{S}$ is a partition of S, $p_i \wedge p_l$ is always false, except when $i = l$; hence, $\bigvee_{l\neq j} \{p_i \wedge p_l\} = p_i \wedge p_i = p_i$, for all $j \neq i$. Therefore:

$$\phi_i = \bigwedge_{j\neq i} \{p_i \vee \{p_i \wedge q_j\}\} = \bigwedge_{j\neq i} p_i = p_i$$

One can then infer the following equality:

$$\bigwedge_j \{p_j \Rightarrow q_j\} = \bigvee_i \{\{p_i \wedge q_i\} \wedge p_i\} = \bigvee_i \{p_i \wedge q_i\}$$

This concludes the proof, since Eq. (4.9) is equivalent to $\bigwedge_k \{\bigvee_i \{p_i \wedge q_i\}\}$.

Bibliography

[1] Cloudcap technology home page: www.cloudcaptech.com/piccolo_plus.htm.

[2] Ilog CPLEX home page: www.ilog.com/products/cplex.

[3] Aviation week & space technology 2005 source book, January 2005.

[4] James F. Allen. Maintaining knowledge about temporal intervals. In *Proceedings of the ACM*, pages 832–843, November 1983.

[5] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. *Hybrid Systems*, NLCS(736):209–229, 1993.

[6] Rajeev Alur, Tomas Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43:116–146, 1996.

[7] J. A. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence.* AAAI Press, 1988.

[8] Eugene Asarin, Olivier Bournez, Thao Dang, and Oded Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control (HSCC-00)*, pages 20–31, 2000.

[9] Egon Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51, 1979.

[10] John Bellingham, Arthur Richards, and Jonathan How. Receding horizon control of autonomous aerial vehicles. In *Proceedings of the American Control Conference*, 2002.

[11] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[12] Lars Blackmore, Stanislav Funiak, and Brian C. Williams. Combining stochastic and gredy search in hybrid estimation. In *Proceedings of the 20th National Conference on Artificial Intelligence*, 2005.

[13] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve responsiveness of planning and scheduling. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO, April 2000.

[14] F. J. Christophersen, M. Baotić, and M. Morari. Optimal Control of Piecewise Affine Systems: A Dynamic Programming Approach. Technical Report AUT05-04, Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH), May 2005.

[15] CloudCap Technologies. *CloudCap Piccolo Quick Setup Guide*, 10 2002.

[16] T. Dang and O. Maler. Reachability analysis via face lifting. In *Hybrid Systems: Computation and Control*, 1998.

[17] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence Journal*, 1991.

[18] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[19] C.A. Floudas. *Nonlinear and Mixed-Integer Programming - Fundamentals and Applications*. Oxford University Press, 1995.

[20] H. Fuchs, Z. Kedem, and B. Naylor. On visible surface generation by a-priori tree structures. In *Proceedings of SIGGRAPH'80*, pages 124–133, 1980.

[21] Carlos E. Garcia. Advances in industrial model-predictive control. In *Chemical Process Control (CPC-III)*, 1986.

[22] M.R. Greenstreet and I. Mitchell. Reachability analysis using polygonal projections. In *Hybrid Systems: Computation and Control*, 1999.

[23] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, 1996.

[24] David S. Hirshfeld. Mathematical programming and the planning, scheduling, and control of process operations. *Computer Aided Process Operations*, 1987.

[25] M. W. Hofbaur and B. C. Williams. Hybrid estimation of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 2004.

[26] Andreas Hofmann. Safe execution of bipedal walking tasks from biomechanical principles. Master's thesis, MIT, 2005.

[27] I hsiang Shu. Enabling fast flexible planning through incremental temporal reasoning. Master's thesis, MIT, 2002.

[28] Henry Kautz, David McAllester, and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, 1996.

[29] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of ECAI-92*, 1992.

[30] Henry Kautz and Bart Selman. Encoding plans in propositional logic. In *Proceedings of KR-96*, 1996.

[31] Henry Kautz and Bart Selman. Unifying sat-based and graph-based planning. In *Proceedings of IJCAI-99*, 1999.

[32] P. Kim, B.C. Williams, and M. Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, 2001.

[33] Philip K. Kim. Model-based planning for coordinated air vehicle missions. Master's thesis, MIT, 2000.

[34] Raj Krishnan. Solving hybrid decision-control problems through conflict-directed branch and bound. Master's thesis, MIT, 2004.

[35] Benjamin Kuipers and Karl Åström. The composition of heterogeneous control laws. In *Proceedings of the 1991 American Control Conference (AAC-91)*, June 1991.

[36] Benjamin Kuipers and Karl Åström. The composition and validation of heterogeneous control laws. *Automatica*, 30(2):233–249, 1994.

[37] Benjamin Kuipers and Subramanian Ramamoorthy. Qualitative modeling and heterogeneous control of global system behavior. In *Proceedings of the 2002 International Workshop on Hybrid Systems: Computation and Control (HSCC-02)*, 2002.

[38] A.B. Kurzhanski and P. Varaiya. On ellipsoidal techniques for reachability analysis. *Optimization: methods and software*, 9 2000.

[39] Yoshiaki Kuwata. Real-time trajectory design for unmanned aerial vehicles using receding horizon control. Master's thesis, MIT, 2003.

[40] Gerardo Lafferriere and ChrisMiller. Uniform reachability algorithms. In *Proceedings of the 2000 International Workshop on Hybrid Systems: Computation and Control (HSCC-00)*, pages 215–228, 2000.

[41] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1999.

[42] Hui Li. Generalized conflict learning for hybrid discrete linear optimization. Master's thesis, MIT, 2005.

[43] Hui Li and Brian C. Williams. Efficiently solving hybrid logic/optimization problems through generalized conflict learning. ICAPS Workshop 'Plan Execution: A Reality Check'. http://mers.csail.mit.edu/mers-publications.htm, 2005.

[44] D. Long and M. Fox. Exploiting a graphplan framework in temporal planning. In *Proceedings of ICAPS'03*, pages 51–62, 2003.

[45] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, 1979.

[46] Matthew T. Mason. *Mechanics of Robotic Manipulation*. MIT Press, 2001.

[47] G. Mitra, I. Hai, and M.T. Hajian. A distributed processing algorithm for solving integer programs using a cluster of workstations. *Parallel Computing*, 3(6), 6 1997.

[48] P. Morris and N. Muscettola. Execution of temporal plans with uncertainty. In *Proceedings of AAAI-2000*, pages 491–496, 2000.

[49] P. Morris, N. Muscettola, and I. Tsamardinos. Reformulating temporal plans for efficient execution. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 444–452, 1998.

[50] P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *Proceedings of IJCAI-01*, pages 494–502, 2001.

[51] Larry Popiel, Ted Matsko, and Coleman Brosilow. Coordinated control. In *Chemical Process Control (CPC-III)*, 1986.

[52] A. I. Propoi. Use of linear programming methods for synthesizing sampled-data automatic systems. *Automation and Remote Control*, 24(7):837–844, 1963.

[53] Subramanian Ramamoorthy and Benjamin Kuipers. Qualitative heterogeneous control of higher oder systems. In *Proceedings of the 2003 International Workshop on Hybrid Systems: Computation and Control (HSCC-03)*, 2003.

[54] J. Richalet, A. Rault, J.L. Testud, and J. Papon. Algorithmic control of industrial processes. In *Proceedings of the 4th IFAC Symposium on Identification and System Parameter Estimation*, pages 1119–1167, 1976.

[55] A. Richards, J. How, T. Schouwenaars, and E. Féron. Plume avoidance maneuver planning using mixed integer linear programming. In *Proceedings of AIAA-2001*, 2001.

[56] Arthur Richards and Jonathan How. Model predictive control of vehicle maneuvers with guaranteed completion time and robust feasibility. In *Proceedings of the 2003 American Control Conference*, 2003.

[57] Tom Schouwenaars, Bart De Moor, Eric Féron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *Proceedings of the ECC Conference*, 2001.

[58] John Stedl. Managing temporal uncertainty under limited communication: A formal model of tight and loose team communication. Master's thesis, MIT, 2004.

[59] Ashish Tiwari. Approximate reachability for linear systems. In *Proceedings of the Sixth International Workshop on Hybrid Systems: Computation and Control (HSCC)*, pages 514–525, 2003.

[60] Ioannis Tsamardinos, Martha E. Pollack, and Sailesh Ramakrishnan. Assessing the probability of legal execution of plans with temporal uncertainty. In *Proceedings of the 13th International Conference on Automatic Planning and Scheduling*, 2003.

[61] T. Vidal and M. Ghallab. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 48–52, 1996.

[62] M. Villain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the AAAI-86*, pages 377–382, 1986.

[63] Aisha Walcott. Unifying model-based programming and randomized path planning though optimal search. Master's thesis, MIT, 2004.

[64] Andreas Frederik Wehowsky. Safe distributed coordination of heterogeneous robots through dynamic simple temporal networks. Master's thesis, MIT, 2003.

[65] D. E. Wilkins and K. L. Myers. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation*, 5(6):731–761, December 1995.

[66] B. C. Williams, Michel Ingham, Seung H. Chung, and Paul H. Elliott. Model-based programming of intelligent embedded systems and robotic space explorers. In *Proceedings of the IEEE: Special Issue on Modeling and Design of Embedded Software*, 2003.

[67] Brian C. Williams and P. Pandurang Nayak. Immobile robots: Artificial intelligence in the new millennium. *Cover article of AI Magazine*, 17(3):16–35, 1996.