

École polytechnique de Louvain

Reinforcement learning for the optimal control of hybrid systems

Author: **Jean BOUCHAT**
Supervisor: **Raphaël JUNGERS**
Advisor: **Benoît LEGAT**
Readers: **Anthony PAPAVALIOU, Zheming WANG**
Academic year 2018–2019
Master [120] in Mathematical Engineering

Abstract

Hybrid systems, because they comprise of both continuous dynamics and discrete events, allow the modeling of complex systems for which traditional dynamical systems are inadequate. However, despite considerable efforts from, notably, the control theory and computer science communities, their optimal control remains a challenging topic because of their inherent complexity. This work proposes to address this issue by designing an algorithm, called *hybrid dual dynamic programming* algorithm, capable of finding near-optimal trajectories for time-invariant discrete-time affine hybrid optimal control problems by iteratively learning their value function through the reciprocal succession of the exploration of the state-space by means of a model predictive controller, and the approximation of the value function in the explored regions via dual dynamic programming. Its application to discrete-time piecewise affine control systems and discrete-time linear control systems under linear temporal logic constraints is presented, which illustrates its capabilities and offers an opportunity to discuss the main challenges it faces, consisting in the presence of a duality gap due to the non-convexity of the problem and in its susceptibility to under-explore parts of the state-space, and potential solutions to overcome them.

Acknowledgments

First of all, I would like to thank Benoit Legat. Without his passionate participation, his endless patience and his precious help, this work could not have been successfully completed.

I would also like to express my gratitude to my master's thesis supervisor, Prof. Raphael Jungers, for his guidance, his insightful comments, and his constructive criticism which helped me improve the quality of my work as well as myself in the process.

Getting through my master's thesis and my studies required more than academic assistance, and I have my parents, my brother and my sister to thank for their unwavering support. You never stopped believing in me and for this I am forever grateful.

Finally, I would like to thank my partner, Clotilde, with whom I shared the good times and the hard times, and, most importantly, who made me believe in myself.

Contents

List of Notations and Acronyms	ix
List of Figures	xi
1 Introduction	1
2 Theoretical background	3
2.1 Hybrid systems	3
2.2 Hybrid optimal control	7
2.3 Reinforcement learning	10
2.4 Julia programming language	12
3 Hybrid dual dynamic programming algorithm	13
3.1 Dynamic programming	15
3.2 Forward pass : model predictive control	17
3.3 Backward pass : dual dynamic programming	17
3.4 Implementation	18
3.4.1 Forward pass	19
3.4.2 Backward pass	22
3.4.3 Termination criterion	24
3.4.4 Results and improvements	24
4 Examples and Applications	29
4.1 Simple switch	29
4.1.1 Hybrid optimal control problem	30
4.1.2 Results	32
4.2 Double integrator	34
4.2.1 Hybrid system formulation	35
4.2.2 Hybrid optimal control problem	37
4.2.3 Results	37
5 Conclusion	47
Bibliography	49
A Julia codes	55

List of Notations and Acronyms

$\dot{x}$	first derivative with respect to time of x
$\ddot{x}$	second derivative with respect to time of x
$\mathbb{R}_+$	set of non-negative real numbers, $\mathbb{R}_+ = \{x \in \mathbb{R} : x \geq 0\}$
H	(controlled) hybrid automaton
$\mathcal{Q}$	discrete state-space
$q, q_k, q(t)$	discrete state / mode
q_T	terminal discrete state
$\mathcal{X}$	continuous state-space (except in Section 4.2)
$x, x_k, x(t)$	continuous state
x_T	terminal continuous state
$\mathcal{V}$	(set of) discrete control input set(s)
$\mathcal{V}_q$	discrete control input set of mode q
$v, v_k, v(t)$	discrete control input
$\mathcal{U}$	(set of) continuous control input set(s)
$\mathcal{U}_q$	continuous control input set of mode q
$u, u_k, u(t)$	continuous control input
Init	set of initial hybrid states, $\text{Init} \subseteq \mathcal{Q} \times \mathcal{X}$
f	vector field
f_q	vector field of mode q
$2^{\mathcal{S}}$	power set of set $\mathcal{S}$
$\text{Dom}(q)$	domain of mode q
$\mathcal{E}$	set of (discrete) transitions
ϵ	(discrete) transition
$q \xrightarrow{v} \tilde{q}$	transition from mode q to mode $\tilde{q}$ by the means of a control v
$\mathcal{G}_\epsilon$	guard set of transition ϵ

$\mathcal{R}_\epsilon$	reset map of transition ϵ
G_H	graph of the hybrid automaton H
J^*	value/cost-to-go function
$\tilde{J}$	approximation of the value function
I_n	identity matrix of size n
HDDP	acronym for hybrid dual dynamic programming
HJB	acronym for Hamilton-Jacobi-Bellman
LTL	acronym for linear temporal logic
MIQP	acronym for mixed-integer quadratic programming/program
MPC	acronym for model predictive control/controller
PDE	acronym for partial differential equation
PMP	acronym for Pontryagin's maximum principle
QP	acronym for quadratic programming/program
RHS	acronym for right-hand side

List of Figures

2.1	Hybrid automaton representation of the bouncing ball in Example 2.1.	4
2.2	Hybrid automaton representation of the thermostat in Example 2.2.	5
3.1	Example of a convex polyhedral lower approximation of the value function in a one-dimensional setup.	18
3.2	Decision tree used to make a choice regarding the results returned by the MPC.	21
3.3	Example of relaxation in a one-dimensional setup.	25
3.4	Example of the refinement of the hybrid automaton through mode splitting around the reference trajectory.	26
3.5	Example of the expected consequence on the duality gap of splitting a mode in a one-dimensional setup.	26
4.1	Graphical representation of the state-space of the discrete-time piecewise affine system in Example 4.1.	30
4.2	Graph of the discrete-time affine hybrid automaton of the discrete-time piecewise affine system in Example 4.1.	31
4.3	Results for the discrete-time piecewise affine system in Example 4.1.	33
4.4	Graphical representation of the state-space of the discrete-time linear control system under LTL constraints in Application 4.1.	35
4.5	Graph of the discrete-time affine hybrid automaton of the discrete-time linear control system under LTL constraints in Application 4.1.	36
4.6	Influence of disabling the optional condition on the results for the discrete-time linear control system under LTL constraints in Application 4.1.	38
4.7	Influence of enabling the optional condition on the results for the discrete-time linear control system under LTL constraints in Application 4.1.	39

4.8	All the continuous state iterates generated during all the forward passes of the algorithm (with horizons $N = 2, 3$, and 4) for the discrete-time linear control system under LTL constraints in Application 4.2.	41
4.9	Influence of the horizon ($N = 2$) of the MPC on the results for the discrete-time linear control system under LTL constraints in Application 4.2.	43
4.10	Influence of the horizon ($N = 3$) of the MPC on the results for the discrete-time linear control system under LTL constraints in Application 4.2.	44
4.11	Influence of the horizon ($N = 4$) of the MPC on the results for the discrete-time linear control system under LTL constraints in Application 4.2.	45

Chapter 1

Introduction

Hybrid systems are dynamical systems that are characterized by interactions between continuous dynamics and discrete phenomena. Such systems arise in varied contexts such as air traffic management, robotics, and manufacturing. Notably, hybrid systems have a central role in embedded control systems, in which digital devices interact with continuous physical processes through sensors and actuators, and more broadly in cyber-physical systems [1], which are rapidly becoming ubiquitous in an ever more connected world.

Unfortunately, while hybrid systems offer a way to model complex phenomena for which traditional models are inadequate, the mix of continuous and discrete dynamics raises a number of difficulties. Their optimal control, in particular, remains a challenging topic today despite efforts from both the computer science and the control theory communities whose focus is on the two fields from which these complex mathematical tools most naturally arise. Indeed, although the optimal control of systems that are either continuous or discrete is a well-researched field of study, the methods stemming from it, despite their effectiveness, cannot readily be adapted to the resolution of hybrid optimal control problems due, in part, to their inherent complexity. Even though considerable efforts have been made in recent years to develop methods that are suitable for the control of real-life hybrid systems, no practically viable method that can solve considerably complex generic hybrid optimal control problems has thus far been successfully designed.

It is in this context that we have attempted, in this work, to develop an algorithm which provides approximate solutions to relatively general hybrid optimal control problems.

Contribution. The main contribution of this work is the design of an algorithm, called *hybrid dual dynamic programming* algorithm, capable of finding near-optimal trajectories for time-invariant discrete-time affine hybrid optimal control problems by iteratively learning their value function through the reciprocal succession of the exploration of the state-space by

means of a model predictive controller and the approximation of the value function in the explored regions via dual dynamic programming.

Structure of the report. Most of the work that occupied us during this academic year is divided between Chapters 2, 3 and 4.

The aim of Chapter 2 is to introduce the reader to the theory of hybrid systems and their optimal control. The most common ways of solving hybrid optimal control problems are presented, as well as some necessary definitions. The link between reinforcement learning and optimal control is then explored. Finally, the decision to implement it in the Julia programming language is briefly motivated.

Chapter 3 covers the main contribution of this work: the hybrid dual dynamic programming algorithm itself. It begins with a brief introduction to the classes of systems that the algorithm is meant to solve, then its working principles are explained as well as the details and choices made for its implementation.

Finally, in Chapter 4 are presented and discussed the behavior and capabilities of the algorithm through its application to the resolution of a discrete-time piecewise affine control system, and a discrete-time linear control system under linear temporal logic constraints.

Chapter 2

Theoretical background

This chapter aims at providing the reader with a general understanding of hybrid systems and their optimal control. Section 2.1 is intended as an introduction to the theory of hybrid systems. Then, a general definition of a hybrid optimal control problem and the methods most commonly used to solve it are presented in Section 2.2. Afterwards, Section 2.3 highlights how reinforcement learning relates to optimal control and how the algorithm we designed fits into its framework. Finally, the decision to use the Julia programming language to implement the aforementioned algorithm is briefly motivated in Section 2.4.

2.1 Hybrid systems

Generally speaking, hybrid systems are dynamical systems that exhibit both continuous and discrete behaviors. From a control theorist's perspective, hybrid systems can be seen as continuous-time dynamical systems interacting with discrete phenomena, the most common ones [2] being:

- autonomous jumps: the continuous state of the system reaches a new value discontinuously according to a map, called the *reset map*,
- controlled jumps: the continuous state of the system may be altered discontinuously in response to a command,
- autonomous switchings: the vector field describing the continuous dynamics of the system changes discontinuously,
- controlled switchings: the vector field of the system may be switched discontinuously in response to a command.

An instance of a hybrid system that arises from the presence of an autonomous jump is the bouncing ball in Example 2.1, and one that illustrates

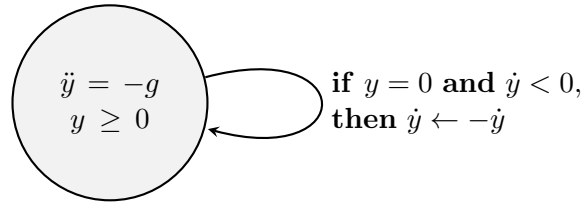


Figure 2.1: Hybrid automaton representation of the bouncing ball in Example 2.1. This hybrid systems comprises one discrete state with continuous dynamics describing the vertical motion of the ball of unit mass under constant gravity g , and a discrete transition describing the autonomous jump in the continuous state happening when the ball reaches the floor at $y = 0$.

the autonomous switching between two discrete states is the thermostat in Example 2.2.

Example 2.1 (Bouncing ball). The vertical motion of a ball of unit mass under constant gravity bouncing against a hard surface cannot only be described by Newton's equations of motion which give rise, in the simplest case, to a continuous-time dynamical system of two ordinary differential equations. At the moment the ball impacts the surface and is about to bounce, its speed suddenly changes direction. The mixing of this continuous-time dynamical system and the continuous state jump at the moment of impact constitutes a hybrid system.

A graphical representation of the system can be seen in Figure 2.1.

Example 2.2 (Thermostat). A thermostat controls a heater which has two modes: either it is functioning at maximum power, or not at all. When the temperature T measured by the thermostat falls bellow a certain threshold τ_1 , the thermostat turns the heater on so that the temperature rises at a rate described by $\dot{T} = -T + T_0$, with $T_0 \in \mathbb{R}_+$. Once the temperature reaches another threshold $\tau_2 > \tau_1$, the thermostat turns the heater off so the temperature decreases at a rate described by $\dot{T} = -T$. The autonomous switching between two separate continuous dynamics is what makes this system hybrid.

A graphical representation of the system can be seen in Figure 2.2.

While the previous examples are easy to grasp, a proper representation is beneficial, if not necessary, to allow for the analysis of more complex hybrid systems. Fortunately, there exists a great number of formalisms to describe hybrid systems to choose from. This diversity of notations appears to be primarily driven by the viewpoint of the two disciplines from which hybrid systems naturally arise: control theory [3], [4] and computer science [5]. One of the most commonly used representations is the hybrid automaton

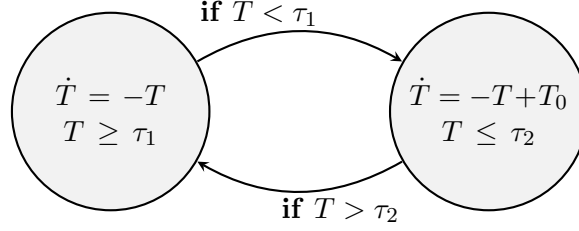


Figure 2.2: Hybrid automaton representation of the thermostat in Example 2.2. This hybrid system comprises two discrete states, each with its own continuous dynamics, which are switched autonomously depending on the value of the continuous state.

[2], [6], [7] which can be extended to account for the continuous and discrete controls one might have over a hybrid system [8].

Definition 2.1 (Controlled hybrid automaton). A controlled hybrid automaton H is a 10-tuple

$$H = (\mathcal{Q}, \mathcal{X}, \mathcal{V}, \mathcal{U}, \text{Init}, f, \text{Dom}, \mathcal{E}, \mathcal{G}, \mathcal{R}),$$

where

- $\mathcal{Q}$ is the finite set of discrete states or *modes*, it is also called the discrete state space;
- $\mathcal{X} \subseteq \mathbb{R}^n$ is the continuous state space;
- $\mathcal{V} = \{\mathcal{V}_q\}_{q \in \mathcal{Q}}$ is the finite set of discrete input sets;
- $\mathcal{U} = \{\mathcal{U}_q\}_{q \in \mathcal{Q}}$ is the set of continuous input sets;
- $\text{Init} \subseteq \mathcal{Q} \times \mathcal{X}$ is the set of initial states;
- $f : \mathcal{Q} \times \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}^n$ assigns to each mode q a vector field usually denoted by $f_q(x, u)$ such that in that mode, $\dot{x} = f_q(x, u)$;
- $\text{Dom} : \mathcal{Q} \rightarrow 2^{\mathcal{X}}$ assigns to each mode q a domain $\text{Dom}(q)$. It is assumed that $\text{Dom}(q) \subseteq \mathcal{X}$;
- $\mathcal{E} \subseteq \mathcal{Q} \times \mathcal{V} \times \mathcal{Q}$ is the set of discrete transitions between pairs of modes. It is assumed that $(q, \emptyset, q) \in \mathcal{E}$. Note that another, arguably more intuitive, notation for the transition from q to $\tilde{q}$ by the means of a control v is $q \xrightarrow{v} \tilde{q}$;
- $\mathcal{G} : \mathcal{E} \rightarrow 2^{\mathcal{X}}$ assigns a guard set to each discrete transition. It is assumed that for all $(q, v, \tilde{q}) \in \mathcal{E}$, $\mathcal{G}(q, v, \tilde{q}) \cap \text{Dom}(q) \neq \emptyset$, and that $\mathcal{G}(q, \emptyset, q) \subseteq \text{Dom}(q)$. Note that $\mathcal{G}_\epsilon$ will also be used to denote $\mathcal{G}(\epsilon)$;

- $\mathcal{R} : \mathcal{E} \times \mathcal{X} \times \mathcal{U} \rightarrow 2^{\mathcal{X}}$ assigns to each transition $\epsilon \in \mathcal{E}$ a reset map that designates the new continuous state resulting from a discrete transition. Note that, for ease of notation, $\mathcal{R}_\epsilon(x, u)$ will often be used to denote $\mathcal{R}(\epsilon, x, u)$.

The pair $(q, x) \in \mathcal{Q} \times \mathcal{X}$ is called the (hybrid) state of H .

The dynamics of the H can be described as follows. Let the system be in the initial hybrid state $(q_0, x_0) \in \text{Init}$ and follow an admissible trajectory. As long as the hybrid state is in the domain of the initial mode, $\text{Dom}(q_0)$, the continuous state evolves according to the dynamics described by the vector field $f_{q_0}(x, u)$. Once the continuous state enters the guard set of an existing transition $\epsilon \in \mathcal{E}$ from q_0 to a mode $q \in \mathcal{Q}$, the system is allowed to discontinuously jump from its current state (q_0, x) to a new hybrid state $(q, \mathcal{R}_\epsilon(x, u))$, according to the reset map assigned to ϵ , by using a discrete control input $v \in \mathcal{V}$. If the transition occurs, the evolution of the continuous state is then described by $f_q(x, u)$.

Definition 2.2. An *admissible trajectory* for a controlled hybrid automaton (Definition 2.1) is a 5-tuple $(\Theta, q(t), x(t), u(t), v(t))$ satisfying $(q(t_0), x(t_0)) \in \text{Init}$, $\forall \tau \in \Theta$,

- $(q(\tau) \xrightarrow{v(\tau)} q(\tau^+)) \in \mathcal{E}$,
- $x(\tau^+) = \mathcal{R}_{(q(\tau), v(\tau), q(\tau^+))}(x(\tau), u(\tau))$,

and, $\forall t \in [t_0, t_T)$,

- $\dot{x}(t) = f_{q(t)}(x(t), u(t))$,
- $q(t) \in \mathcal{Q}$,
- $x(t) \in \text{Dom}(q(t))$,
- $v(t) \in \mathcal{V}_{q(t)}$,
- $u(t) \in \mathcal{U}_{q(t)}$,

where t_0 is the initial time, t_T is the terminal time, and Θ is the strictly increasing finite sequence of times $\{\tau_0, \dots, \tau_{n-1}\}$, $n \in \mathbb{N}$, $t_0 < \tau_0$, $\tau_{n-1} < t_T$, at which the discrete transitions occur and of which τ_i^+ , $i = 0, \dots, n-1$, are the times right after.

Finally, for practical reasons, when discussing the discrete dynamics of hybrid systems, we will often use the notion of graph of the automaton, hence the need for the following definition:

Definition 2.3 (Graph of a hybrid automaton). We define the graph G_H of a (controlled) hybrid automaton H (Definition 2.1) as a pair

$$G_H = (\mathcal{Q}, \mathcal{E})$$

where the modes $q \in \mathcal{Q}$ and the discrete transitions $\epsilon \in \mathcal{E}$ of H are respectively the vertices and the edges of the graph.

2.2 Hybrid optimal control

Hybrid optimal control problems arise when the hybrid control inputs of a hybrid system are chosen so that they generate an admissible trajectory optimizing a performance criterion, usually called *cost* in the case of a minimization. In hybrid systems, the cost function often involves an integral cost accumulated along the continuous evolution of the state and a discrete cost incurred when transitions occur.

Definition 2.4 (Hybrid optimal control problem). For a controlled hybrid automaton (Definition 2.1), we define the following (finite time) hybrid optimal control problem. Let (q_0, x_0) be the initial state of the system, (q_T, x_T) its desired terminal state, t_0, t_T, Θ , and $\tau_i^+, i = 0, \dots, n-1$, be as in Definition 2.2, where n is the number of transitions occurring, and finally, for ease of notation, let $\theta(t) = (q(t), x(t), v(t), u(t))$. The problem can be written as an optimization problem:

$$\begin{aligned} J^*(q_0, x_0) = \underset{v(t), u(t)}{\text{minimize}} \quad & k(q(t_T), x(t_T)) \\ & + \sum_{i=0}^{n-2} \left(p(\tau_i, \theta(\tau_i), \theta(\tau_i^+)) + \int_{\tau_i}^{\tau_{i+1}} m(t, \theta(t)) \, dt \right) \\ & + \int_{t_0}^{t_0} m(t, \theta(t)) \, dt + \int_{\tau_{n-1}}^{t_T} m(t, \theta(t)) \, dt \\ \text{subject to} \quad & (\Theta, q(t), x(t), v(t), u(t)) \text{ is admissible} \end{aligned} \tag{2.1}$$

where $k : \mathcal{Q} \times \mathcal{X} \rightarrow \mathbb{R}_+$, $p : [t_0, t_T] \times (\mathcal{Q} \times \mathcal{X} \times \mathcal{V} \times \mathcal{U})^2 \rightarrow \mathbb{R}_+$ and $m : [t_0, t_T] \times \mathcal{Q} \times \mathcal{X} \times \mathcal{V} \times \mathcal{U} \rightarrow \mathbb{R}_+$ are assigning a cost to, respectively, the terminal state, the discrete transitions and the continuous evolution, and $J^* : \mathcal{Q} \times \mathcal{X} \rightarrow \mathbb{R}_+$ is called the *value function* of the system.

A straightforward approach to compute optimal solutions to (2.1) is to obtain the optimal trajectories of the continuous state for all possible discrete state sequences and select the cheapest one. However, while exact solutions to continuous-time optimal control problems can be computed by using Bellman's dynamic programming principle [9] or Pontryagin's maximum principle [10], this approach is susceptible to fail due to the significant

combinatorial complexity inherent to the discrete part of even moderately large systems. Indeed, in general, neither the number of discrete transitions, nor their order, nor the time at which they occur are known in advance, and, adding to this problem, the possible number of sequences can be infinite in problems where discrete trajectories are allowed to cycle. Moreover, the mixing of discrete events and continuous dynamics often results in nonlinear, non-convex optimization problems. Hence the need for specific mathematical tools appropriate for solving hybrid optimal control problems. Below are some of them.

Bellman’s dynamic programming principle. Coined in 1957 by Bellman, the term dynamic programming denotes a method of solving complex problems by dividing them into a sequence of more manageable ones. It is based on the *principle of optimality* which states that each optimal sub-trajectory of an optimal trajectory must be itself an optimal trajectory with regard to the initial and terminal states of the sub-trajectory [11]. This suggests that an optimal control problem can be solved backward and recursively by first finding an optimal solution for the last stage of an optimal trajectory, then using it to find a solution for the two last stages, and so forth for the entire state space.

For continuous-time problems, dynamic programming yields the Hamilton-Jacobi-Bellman (HJB) partial differential equation (PDE), a necessary and sufficient condition for optimality if solved over the whole state space, of which the value function is the solution. In turn, the value function, being the cost of completing an optimal trajectory, allows the determination of the optimal control inputs at any state of the system. In discrete-time cases, rather than a PDE, it yields a difference equation which is often simply called Bellman’s equation.

Numerous authors have worked on the generalization of the dynamic programming principle to hybrid systems [12]–[14], which is called *hybrid dynamic programming principle* and is based on the discretization of the continuous state-space, but while, in theory, dynamic programming can be used to find exact solutions to hybrid optimal control problems, the complexity of real world applications usually inhibits its use due to its two curses: the *curse of dimensionality*, which is the explosion of the computation required as the number of states increases, and the *curse of modelling*, which denotes the need for an exact model of the evolution of the system [15]. Note that, in order to narrow down the scope of our investigation, the latter will not be of concern in this work.

Pontryagin’s maximum principle. In 1962, Pontryagin devised a principle, based on the calculus of variations, that can be used to find solutions to continuous-time optimal control problems. This principle, known

as Pontryagin's maximum principle (PMP), yields Euler-Lagrange differential equations, the solution of which describes a characteristic of the HJB partial differential equation [16]. For continuous-time problems, the PMP gives necessary conditions for the optimality of the trajectories and the control inputs of hybrid systems. However, since the aforementioned conditions are not sufficient, each trajectory satisfying these conditions can only be considered as an optimal trajectory candidate, and another method has to be used to select the best among a possible infinite number of candidates.

As for the dynamic programming principle, efforts have been made, with some success, to extend the maximum principle to hybrid systems. They have resulted in what is called the *hybrid maximum principle* [17], [18]. Yet, unfortunately, this approach shares the same drawbacks as dynamic programming, in the sense that it is only practically applicable to relatively small hybrid systems [19].

Other methods. Various other methods exist that exploit particular structures of the problems they are meant to solve in order to compute their solutions efficiently. For example, switched and piecewise affine systems, which are two subclasses of hybrid systems, have received a lot of attention in the recent years. Switched systems consists in a collection of dynamical systems and a switching function specifying the active subsystem at each time, and piecewise-affine systems are switched systems the state-space of which is divided into polyhedral regions in each of which the state evolution is described by an affine expression. An important reason for this interest is that these subclasses are sufficiently specialized to be solvable reasonably efficiently in practice, but not so much that they have no useful application. For example, switched systems have been successfully applied in the design of hybrid power systems [20], the management of electrical vehicle energy [21], the control of autonomous puppets in robotics [22], and many others. A survey of recent results in optimal control of switched and piecewise affine systems can be found in [23].

Unfortunately, because of the complexity inherent to the interactions between continuous and discrete dynamics hybrid systems, it seems clear nowadays that, in all but their simplest instances, only approximate solutions can practically be determined for more general hybrid optimal control problems [24].

The two methods briefly presented below can be used to generate near-optimal solutions to hybrid optimal control problems, and are both exploited in the algorithm which is the central piece of this work, in Chapter 3.

Model predictive control. A powerful and widely used technique in applied control is Model Predictive Control (MPC) which designates an en-

semble of methods that make use of a model of the process to determine appropriate control inputs by minimizing the associated cost function over a finite receding horizon [25], [26]. The general working principle of model predictive controllers is the following. At each time step, based on the predicted future evolution of the system, optimal control inputs that minimize the cost while forming an admissible trajectory are computed up until a defined time horizon, then only the first instance of the command law is applied to the system. Regrettably, while it is obvious that using a finite horizon can reduce the combinatorial complexity inherent to the discrete dynamics of hybrid systems, the main drawbacks are that the resulting trajectories have no guarantee to be optimal, unless the value function of the system is known, to reach the desired terminal state, or even to stay feasible indefinitely. Note that a way to avoid this last shortcoming is to precompute controlled invariant sets as in [27]. Also, see [28] for the design of a model predictive controller which guarantees the satisfaction of linear temporal logic (LTL) specifications at all times and thus is not susceptible to the second drawback, although it does not scale well with the dimensionality of the state-space, as illustrated in the case study summary of [29].

Details about its use in the algorithm developed in this work will be provided in Section 3.2.

Dual dynamic programming. The dual dynamic programming algorithm is the deterministic counterpart to the renowned stochastic dual dynamic programming algorithm which is a method, pioneered by [30], used to solve multistage stochastic programming problems. It is a decomposition algorithm designed to solve convex, or convexified as in [31], optimization problems by iteratively computing linear functions, called *optimality cuts*, thereby exploiting the duality property of the problem, in order to generate a polyhedral lower approximation of the value function defined as the supremum of those cuts [32].

In the context of deterministic hybrid optimal control problems, the dual dynamic programming theory can be extended to generate an approximation of value function in each mode of the hybrid system, the quality of which depends on the convexity of the problem; for non-convex value functions, a duality gap can hinder the capabilities of the algorithm to estimate its exact value.

Details about its use in the algorithm developed in this work and ideas to alleviate the duality gap will be provided in Section 3.3.

2.3 Reinforcement learning

Reinforcement learning has its origin in the efforts from the artificial intelligence community to solve problems where an active decision-making agent

interacts with an uncertain environment, learning in the process, in order to achieve a certain goal. Accordingly, some optimal control methods can thus be considered to be reinforcement learning when they aim at determining optimal solutions to optimal control problems through successive and gradually better approximations of the value function or the optimal control law [33], which is the intended goal of the approximate dynamic programming methods that have been developed in order to overcome the limitations of dynamic programming due to the curse of dimensionality [34]. It is thus not surprising that some prominent authors use the terms reinforcement learning, approximate dynamic programming or even neuro-dynamic programming [35] almost interchangeably when considering optimal control problems [15]. Others simply define approximate programming as the result of bringing reinforcement learning and dynamic programming together [36].

Usually, reinforcement learning methods are separated in two categories: *model-free* and *model-based*. The latter use a model of the environment to decide their future best course of actions without the need for prior experience with them, while the former learn by trial-and-error only which make them less efficient in terms of the number of samples needed to learn an optimal trajectory. These categories are, however, not mutually exclusive. Some methods are designed so that they simultaneously learn a model through trial-and-error interactions with the environment and plan ahead thanks to it. In some optimal control problems, the dynamics are too complex to be accurately described by a model, hence the usefulness of model-free methods. But, as mentioned in the previous section, these problems are not a concern in the scope of this work.

The reinforcement learning methods more directly related to dynamic programming can be divided up further into *policy* (i.e. control law) *iteration* and *value iteration* methods which relate to the algorithmic strategy they implement. The value iteration algorithm starts with an approximation of the value function and iteratively improves it until the value function is obtained, in order to produce an optimal policy. Note that this strategy is closely related to *Q-learning* [37], [38], however Q-learning is mostly used in model-free contexts. The policy iteration strategy consists in defining a policy arbitrarily, and then alternating between *policy evaluation* and *policy improvement*. These two steps amount to evaluating the value function along the trajectory that the policy describes and improving the policy with regards to the increasingly better approximation of the value function devised in the first step. The goal of this algorithm is to converge towards an optimal policy, which it often does in a surprisingly little number of iterations [33]. See [39] for proofs of their convergence in deterministic optimal control problems.

The model-based algorithm we designed, and presented in Chapter 3, can be seen as a policy iteration method where instead of evaluating and improving a policy over all the states in the hybrid state-space, these two

steps are performed over a single trajectory from and to predefined states.

2.4 Julia programming language

Julia is modern high-level language that offers a very intuitive syntax, which gets close to the mathematical formulation, and the performance of statically-typed languages such as C++ [40]. Furthermore, thanks to its growing community, a large amount of packages for numerical computing are available, as well as the possibility to develop in a Jupyter notebook. For those reasons, Julia might very well be the future language of choice for the scientific community, and the decision was made to use it in this work.

Chapter 3

Hybrid dual dynamic programming algorithm

In this chapter we present the central piece of this work, which we call *hybrid dual dynamic programming* (HDDP) algorithm. Sections 3.1 to 3.3 are meant as a general introduction to the working principles of the algorithm, while section 3.4 is devoted to explanations about the details and choices made for its implementation. But first, the reader is introduced to the class of problems that the algorithm is intended to solve.

Remark 3.1. The term *hybrid* in hybrid dual dynamic programming is a reference to the nature of the systems that the algorithm is meant to solve rather than to its own nature.

—

The hybrid system denomination covers a vast ensemble of mathematical objects that are the subject of an extensive and diverse body of scientific literature. That is why, to work efficiently, we narrowed-down the scope of our investigation to time-invariant discrete-time affine hybrid automata that are controlled by the means of continuous and discrete inputs. Their more formal definition is the following:

Definition 3.1 (Discrete-time affine hybrid automaton). We define a (time-invariant) discrete-time affine hybrid automaton H as a hybrid automaton (Definition 2.1),

$$H = (\mathcal{Q}, \mathcal{X}, \mathcal{V}, \mathcal{U}, \text{Init}, f, \text{Dom}, \mathcal{E}, \mathcal{G}, \mathcal{R}),$$

where for all $q \in \mathcal{Q}$,

- $\mathcal{U}_q$ and $\text{Dom}(q)$ are polyhedral sets,
- the continuous dynamics of each mode is the identity: $f_q = I_n$, where n is the dimension of the continuous state space,

- and the reset maps from q to $\tilde{q}$ are time-invariant discrete-time affine dynamical systems: $\forall (q, v, \tilde{q}) \in \mathcal{E}, \mathcal{R}_{(q,v,\tilde{q})}(x, u) = A_q x + B_q u + c_q$.

Remark 3.2. In the case of digital applications, the continuous dynamics of hybrid systems must be discretized which means that it can be preserved in the framework described above by becoming the reset map of the self loop of each mode.

Definition 3.2. An *admissible trajectory* for a discrete-time affine hybrid automaton (Definition 3.1) is a sequence $\{q_k, x_k, v_k, u_k\}_{k \in \mathbb{N}}$ satisfying $(q_0, x_0) \in \text{Init}$ and, for all $k \in \mathbb{N}$,

- $(q_k \xrightarrow{v_k} q_{k+1}) \in \mathcal{E}$,
- $x_{k+1} = A_{q_k} x_k + B_{q_k} u_k + c_{q_k}$,
- $x_k \in \text{Dom}(q_k)$,
- $v_k \in \mathcal{V}_k$,
- $u_k \in \mathcal{U}_k$.

In order to find the best possible admissible trajectory starting at a predefined initial state and ending at a state called *terminal state*, we need to define a performance criterion. Hence, let $g : \mathcal{Q} \times \mathcal{X} \times \mathcal{V} \times \mathcal{U} \rightarrow \mathbb{R}_+$ and $g_n : \mathcal{Q} \times \mathcal{X} \rightarrow \mathbb{R}_+$ be positive quadratic convex cost functions that denote the cost incurred at each time step. Then, the cost of a sequence of hybrid control inputs $(v_0, u_0) \dots, (v_{n-1}, u_{n-1})$ starting at an initial state (q_0, x_0) and ending at the terminal state (q_n, x_n) is

$$J(q_0, x_0 ; v_0, u_0 \dots, v_{n-1}, u_{n-1}) = g_n(q_n, x_n) + \sum_{k=0}^{n-1} g(q_k, x_k, v_k, u_k), \quad (3.1)$$

where $g_n(q_n, x_n)$ is the *terminal cost* incurred upon reaching the terminal state.

Definition 3.3. For a discrete-time affine hybrid automaton (Definition 3.1), we define the following (finite time) *hybrid optimal control problem*. Let (q_0, x_0) be the initial state, (q_T, x_T) the desired terminal state, and $(v_0, u_0), \dots, (v_{n-1}, u_{n-1}), v_k \in \mathcal{V}_k, u_k \in \mathcal{U}_k, k = 0, \dots, n-1$, a finite sequence of hybrid control inputs minimizing the cost functional J as defined in (3.1). The problem can be written as an optimization problem:

$$\begin{aligned}
J^*(q_0, x_0) = & \begin{array}{ll} \text{minimize} & g_n(q_n, x_n) + \sum_{k=0}^{n-1} g(q_k, x_k, v_k, u_k) \\ & v_k \in \mathcal{V}_{q_k}, u_k \in \mathcal{U}_{q_k} \\ & k = 0, 1, \dots, n-1 \\ \text{subject to} & (q_k \xrightarrow{v_k} q_{k+1}) \in \mathcal{E}, \\ & x_{k+1} = A_{q_k} x_k + B_{q_k} u_k + c_{q_k}, \\ & x_{k+1} \in \text{Dom}(q_{k+1}), \\ & (q_n, x_n) = (q_T, x_T), \end{array}
\end{aligned}$$

where $J^* : \mathcal{Q} \times \mathcal{X} \rightarrow \mathbb{R}_+$ is called the *value function* of the system.

The value function is sometimes referred to as the Holy Grail of the control theorist since access to it, in a discrete-time setting, allows optimal trajectories to be computed in a sequence of one-stage optimization problems in which the optimal control inputs are determined by minimizing it [41]. However, this function is usually unknown and its exact value is hard to determine for most problems.

The algorithm developed in this work, and presented in the last three sections of this chapter, is meant to determine near-optimal solutions to the aforementioned hybrid optimal control problem (Definition 3.3) by iteratively generating a convex polyhedral lower approximation of the value function of the hybrid system that is accurate close to the optimal trajectory. The method to achieve this purpose alternates between

1. finding a reference trajectory between the initial and the terminal states by means of a *Model Predictive Controller* (Section 3.2), and
2. generating linear lower approximations of the value function via *Dual Dynamic Programming* (Section 3.3).

But before diving in the inner workings of the algorithm and the theory surrounding it, let us put the problem in the context of Bellman's dynamic programming [9].

3.1 Dynamic programming

Dynamic programming (with finite horizon n) consists in determining the value function J^* of the system iteratively for its whole state space. Extended to hybrid systems, it can be written as the following backward induction.

Let

$$J_n^*(q_n, x_n) = g_n(q_n, x_n), \quad \forall (q_n, x_n),$$

and then, backwards, for $k = n - 1, \dots, 0$,

$$J_k^*(q_k, x_k) = \min_{v_k \in \mathcal{V}_{q_k}, u_k \in \mathcal{U}_{q_k}} [g(q_k, x_k, v_k, u_k) + J_{k+1}^*(q_{k+1}, x_{k+1})], \quad \forall (q_k, x_k).$$

In this general case, $J_k^*(q, x)$ is the optimal cost incurred to the system starting in (q, x) and running for $n - k$ time steps.

For the hybrid optimal control problem in Definition 3.3, the recursion can be reformulated as the following optimization problem:

Let $k = n - 1, \dots, 0$, for all (q_k, x_k) ,

$$\begin{aligned} J_k^*(q_k, x_k) = & \begin{array}{ll} \text{minimize} & g(q_k, x_k, v_k, u_k) + J_{k+1}^*(q_{k+1}, x_{k+1}) \\ v_k \in \mathcal{V}_{q_k}, u_k \in \mathcal{U}_{q_k} & \\ \text{subject to} & \end{array} \quad (3.2) \\ & (q_k \xrightarrow{v_k} q_{k+1}) \in \mathcal{E}, \\ & x_{k+1} = A_{q_k} x_k + B_{q_k} u_k + c_{q_k}, \\ & x_{k+1} \in \text{Dom}(q_{k+1}), \\ & (q_n, x_n) = (q_T, x_T), \end{aligned}$$

where $J_k^*(q, x)$ now refers to the minimal cost incurred to the system for going from (q, x) to (q_T, x_T) in $n - k$ time steps. In this particular case, a more evocative term for the value function might be the *cost-to-go* function. The two denominations will be used interchangeably in the rest of this work.

Unfortunately, this scheme suffers from the curse of dimensionality [34], hence the need for techniques that approximate the value function rather than compute its exact value. Ideally, these techniques should focus on approximating the value function very well close to the optimal trajectory, and not care too much about it further away from it to save computing time. Furthermore, they should take advantage of the time-invariance of the problem and compute only one approximation $\tilde{J}$ of the cost-to-go function, which expresses the minimal cost of going from any state to the terminal state in any amount of time necessary to minimize it, rather than multiple functions $\tilde{J}_k$, $k = n, \dots, 0$, which represent the minimal cost of going from any state to the desired terminal state in $n - k$ time steps.

The hybrid dual dynamic programming is precisely one of these techniques that take advantage of the time-invariance of the problem and focuses on approximating the value function well near the optimal trajectory, and the remainder of this chapter is devoted to the explanation of its working principles (in Sections 3.2 and 3.3) and of its implementation (in Section 3.4).

3.2 Forward pass : model predictive control

At each iteration of the HDDP algorithm, a Model Predictive Controller (MPC) with finite horizon N [42] is used to determine a near-optimal trajectory from the initial state of the system to its desired terminal state. To help determine it, the MPC uses the latest approximation of the cost-to-go function it disposes of, allowing it to anticipate gradually better the future cost at the end of its horizon as the algorithm generates more and more precise approximations of the cost-to-go function while iterating.

Let (q_0, x_0) be the initial state of the system, and $\tilde{J}$ the convex piecewise linear lower approximation of the cost-to-go function J^* . At each time step $k \in \mathbb{N}$, the Model Predictive Controller first solves a Mixed-Integer Quadratic Programming (MIQP) problem [26]:

$$\begin{aligned}
 & \underset{\substack{\bar{v}_0, \dots, \bar{v}_{N-1} \\ \bar{u}_0, \dots, \bar{u}_{N-1}}}{\text{minimize}} && \tilde{J}(\bar{q}_N, \bar{x}_N) + \sum_{i=0}^{N-1} g(\bar{q}_i, \bar{x}_i, \bar{v}_i, \bar{u}_i) \\
 & \text{subject to} && (\bar{q}_0, \bar{x}_0) = (q_k, x_k), \\
 & && (\bar{q}_i \xrightarrow{\bar{v}_i} \bar{q}_{i+1}) \in \mathcal{E}, \quad i = 0, \dots, N-1, \\
 & && \bar{x}_{i+1} = A_{\bar{q}_i} \bar{x}_i + B_{\bar{q}_i} \bar{u}_i + c_{\bar{q}_i}, \quad i = 0, \dots, N-1, \\
 & && \bar{x}_{i+1} \in \text{Dom}(\bar{q}_{i+1}), \quad i = 0, \dots, N-1, \\
 & && \bar{v}_i \in \mathcal{V}_{\bar{q}_i}, \quad \bar{u}_i \in \mathcal{U}_{\bar{q}_i}, \quad i = 0, \dots, N-1,
 \end{aligned} \tag{3.3}$$

then selects the first control input iterates and uses them to control the system, while discarding subsequent ones:

$$v_k = \bar{v}_0, \quad u_k = \bar{u}_0, \quad q_{k+1} = \bar{q}_1, \quad x_{k+1} = \bar{x}_1,$$

and repeats the whole procedure until the system has reached the desired terminal state (q_T, x_T) .

3.3 Backward pass : dual dynamic programming

Let $(q_0, x_0), \dots, (q_n, x_n)$ be the trajectory defined by the MPC during the first phase of the current iteration of the algorithm. Going backwards, for each iterate x_k , $k = n, \dots, 0$, the algorithm solves the dual of the problem described in (3.2) with x_k considered as a right-hand side (RHS) parameter of the primal problem. Since the feasible set of the dual does not depend on the RHS parameters, any of its dual solutions is feasible for any value of the parameters, thus solving the dual provides the algorithm with a lower bound on the cost-to-go function in the mode q_k in the form of a linear function $\alpha_k^\top x + \beta_k$, called *cut*, where α_k is the vector of the optimal values of the dual

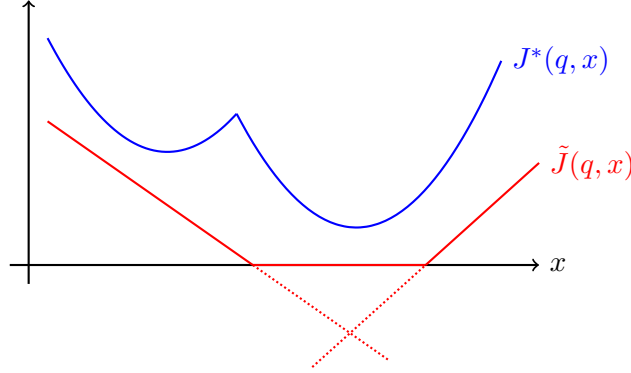


Figure 3.1: Example of a convex polyhedral lower approximation of the value function J^* (in blue) in a one-dimensional setup. The maximum of three cuts defines $\tilde{J}$ (in red), a piecewise linear lower approximation of the value function in one mode q of the hybrid system.

variables related to the primal constraints of which the corresponding x_k is a RHS parameter, and β_k is the optimal value of the objective of the dual minus the influence of the dual variables related to x_k [43].

Hence, for each iterate (q_k, x_k) of the forward pass, a new optimality cut is generated and the convex linear lower approximation of the cost-to-go function is updated as

$$\tilde{J}(q_k, x) = \max\{\alpha_k^\top x + \beta_k, \tilde{J}(q_k, x)\}.$$

A graphical representation of the approximation of the cost-to-go function in a mode q can be found in Figure 3.1. Notice the duality gap between the exact function, J^* , and its convex piecewise linear approximation, $\tilde{J}$. It will receive further attention in Subsections 3.4.2 and 3.4.4.

3.4 Implementation

The purpose of this section is to provide the reader with explanations about essential details and the choices made for the implementation of the algorithm that was loosely described in the previous ones.

Each iteration of the algorithm is composed of two parts : a forward and a backward pass. The former generates a near-optimal trajectory from the initial state, (q_0, x_0) , to the desired terminal state (q_T, x_T) , using the current best approximation of the cost-to-go function. The latter hopefully improves the aforementioned approximation by generating optimality cuts to allow the MPC to select a better trajectory during the next iteration.

To help follow the explanations provided in the section, the reader can find the gist of the algorithm in pseudocode on page 27.

Initialization. Since the cost incurred at each time step, $g(q_k, x_k, v_k, u_k)$, is set to be positive, the cost-to-go function J^* must be positive as well. Hence the initial approximation of the cost-to-go function is $\tilde{J}(q, x) = 0$, $\forall q \in \mathcal{Q}$.

3.4.1 Forward pass

At each time step k , a Model Predictive Controller with finite horizon N is used to select the best possible next iterate (q_{k+1}, x_{k+1}) , considering the exact value function is unknown, in order to define the reference trajectory for the current iteration of the algorithm.

From one MIQP to multiple QP. This requires the algorithm to solve the MIQP problem (3.3) at each time step which can be done reasonably well for small scale problems [44]. However, another approach was chosen to allow for the use of QP solvers which consists in solving the following problem for each admissible trajectory $\bar{q}_0 = q_k, \bar{q}_1, \dots, \bar{q}_N$ in the graph of the automaton,

$$\begin{aligned}
 & \underset{\bar{u}_0, \dots, \bar{u}_{N-1}}{\text{minimize}} && \tilde{J}(\bar{q}_N, \bar{x}_N) + \sum_{i=0}^{N-1} g(\bar{q}_i, \bar{x}_i, \bar{v}_i, \bar{u}_i) \\
 & \text{subject to} && \bar{x}_0 = x_k, \\
 & && \bar{x}_{i+1} = A_{\bar{q}_i} \bar{x}_i + B_{\bar{q}_i} \bar{u}_i + c_{\bar{q}_i}, \quad i = 0, \dots, N-1, \\
 & && \bar{x}_{i+1} \in \text{Dom}(\bar{q}_{i+1}), \quad i = 0, \dots, N-1, \\
 & && \bar{u}_i \in \mathcal{U}_{\bar{q}_i}, \quad i = 0, \dots, N-1,
 \end{aligned} \tag{3.4}$$

where $\bar{v}_i, i = 0, \dots, N-1$, are such that $\bar{q}_i \xrightarrow{\bar{v}_i} \bar{q}_{i+1}$, and choosing the least costly of them.

This problem can be solved much faster than the original one since it is a Quadratic Programming (QP) problem. Indeed, the cost function g is defined as quadratic convex, the approximation of the cost-to-go function $\tilde{J}$ is piecewise linear and convex by design, and for all $q \in \mathcal{Q}$, $\text{Dom}(q)$ and $\mathcal{U}_q$ are polyhedral sets so they can be expressed in terms of linear constraints. However, while this problem can be solved in a considerably shorter amount of time, the solver now has to solve multiple QP problems, one for each admissible trajectory, instead of one MIQP problem.

Not reaching the terminal state. In its current form, the problem offers no incentive for the system to actually reach the desired terminal state apart

from the zero cost that is incurred once the system has arrived there. Hence the MPC might, in some situations, find it preferable to use the least amount of control, even at the expense of never terminating, rather than do what is expected and go towards the desired terminal state, which renders the backward pass useless due to the time-invariance of the approximation of the cost-to-go function.

Usually, to remedy this issue, the length of the trajectory is set and a terminal cost is imposed at the end of this finite trajectory. But a finite horizon MPC might react to it too late and choose that the best course of action is to stay put anyway when the added terminal cost, an heuristic, is too small compared to the expected cost of reaching the desired terminal state. And in that case, where the trajectory does not end in the desired terminal state, the time-invariance of the approximation of the cost-to-go would be a problem. Indeed, while a time-variant $\tilde{J}$ would differentiate between being in a given state earlier or later in the trajectory, a time-invariant one would not, and would consider the cost incurred in the last iteration to be the cost to go to the desired terminal state from that given state at any point in time, which it is clearly not. Reaching a certain state can be advantageous early in the trajectory while being costly when the end is near. Hence the choice was made to force the controller to direct the system towards the desired terminal state and reach it in a predefined finite amount of time.

To achieve this, additional restrictions were imposed on the set $\mathcal{H}$ of admissible trajectories in the graph of the automaton for each of which (3.4) is solved.

Let $d : \mathcal{Q} \times \mathcal{Q} \rightarrow \mathbb{N}$ be the number of transitions in the shortest path separating two modes in the graph of the automaton, which is similar to the distance function in [28] with each transition cost equal to one. For each trajectory $\bar{q}_0, \dots, \bar{q}_N \in \mathcal{H}$ such that $\bar{q}_0 = q_k$,

1. (optional) the trajectory should be closer, at the horizon, to the desired terminal state in the graph of the automaton than it currently is: $d(q_k, q_T) < d(\bar{q}_N, q_T)$, and
2. the desired terminal state should be reachable by the last iteration of the forward pass.

The second condition means that a heuristic had to be chosen that approximated the number of iterations required for the system to reach its desired terminal state. Let T be the number of iterations left for the MPC to reach its target, under the assumption that any mode can be left for all of its neighbors with one iteration, imposing that any trajectory in $\mathcal{H}$ verifies $d(\bar{q}_N, q_T) < T - N$ ensures that the system can reach the desired terminal state within the imparted time. The distance $d(q, \tilde{q})$ is thus used as an upper bound for the number of time steps required to reach mode $\tilde{q}$ from q .

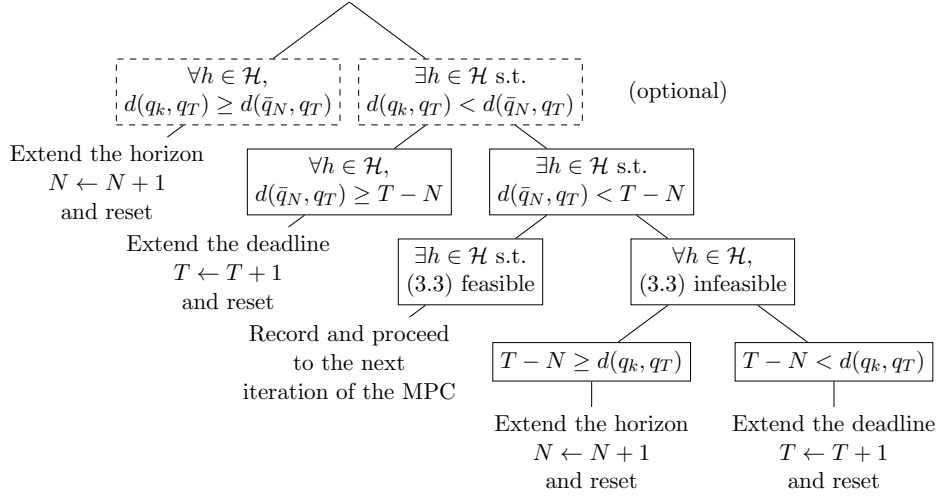


Figure 3.2: Decision tree used to make a choice regarding the results returned by the MPC. Let N the horizon of the MPC, T the number of time steps left for the system to reach its desired terminal state, $\mathcal{H}$ the set of admissible trajectories $\bar{q}_0, \bar{q}_1, \dots, \bar{q}_N$ in the graph of the automaton starting at the current discrete state $q_k = \bar{q}_0$ of the system for which (3.4) is solved, and $d : \mathcal{Q} \times \mathcal{Q} \rightarrow \mathbb{N}$ the number of transitions in the shortest path separating two modes in the graph of the automaton. “Reset” means generating a new set of trajectories $\mathcal{H}$ and solving (3.4) for each of them, then making a new decision thanks to this tree. Note that the extension of the horizon N and of the deadline T are only valid, respectively, until the “Record and proceed” leaf is reached and for the current forward pass.

The first condition is not necessary for the system to reach its desired terminal state, but it accelerates the forward pass noticeably in some situations where the algorithm has not generated many cuts yet and it happens that the system has no incentive to move forward early. Furthermore, since the second condition restricts much more than the first the set of candidate trajectories $\mathcal{H}$, moving early sometimes allows more trajectories to be considered by the MPC which might help finding one that is optimal. In addition, while the first condition may seem conservative, this conservativeness can be decreased by increasing the size of the horizon of the MPC.

Evidently, those restrictions can lead to a set of trajectories $\mathcal{H}$ that is empty, or even to a set of trajectories in the graph of the automaton which all lead to the infeasibility of (3.4). In these cases, a decision had to be made to either extend the horizon N of the MPC until a feasible trajectory was found, or to extend, for the remainder of that particular pass, the number of iterations T left for the forward pass to reach its target. The decision tree that was used can be found in Figure 3.2.

Remark 3.3. If the time-invariance assumption discussed in Section 3.1 did not hold, the algorithm developed in this work would still be usable provided that the approximation of the value function becomes time-dependant. That way, cuts would define the under-approximation of the cost-to-go in a certain mode at a certain time, rather than at any time, which would render the restrictions on the set of admissible trajectories, discussed above, irrelevant because trajectories would not be needed to finish for the backward pass to generate useful cuts. Moreover, cycles in the graph of the automaton would not be a concern anymore since the directed graph would become acyclic. However, the major downside of this variant would certainly be the sharp increase in the number of iterations needed for the algorithm to generate a proper approximation of the value function due to the increase in the number of functions that need to be approximated. Note that the comparison of these two approaches could be the subject of future work.

3.4.2 Backward pass

For each iterate of the reference trajectory of the current iteration of the algorithm, starting from the end, the dual of (3.2) is solved to improve the convex piecewise linear lower approximation of the cost-to-go function.

Relaxation. Even though there exist a multitude of methods to solve the dual of a MIQP problem [45], we chose, in order to be able to use QP solvers, to formulate the primal (3.2) in such a way that the dual of its relaxation and the one of the original problem are the same. A weaker formulation could of course have resulted in a reduced computing time, but also in a gap between the true dual of the one obtained, which was deemed undesirable.

The method used to obtain a strong formulation of (3.2) is explained in the following paragraphs.

Since, for each $q \in \mathcal{Q}$, $\tilde{J}_{k+1}(q, x)$ is a piecewise linear function, its graph can be described by a finite set of ordered vertices. Let $\mathcal{I} \subseteq \mathcal{Q}$ be the set of modes such that there exists a transition in $\mathcal{E}$ from the current mode q_k to them, and m_i the number of vertices $V_{i,j} \in \mathcal{X} \times \mathbb{R}_+$, $j = 1, \dots, m_i$, describing the graph of $\tilde{J}_{k+1}(i, x)$, $i \in \mathcal{I}$. The problem (3.2) can exactly be rewritten as

$$\begin{aligned}
& \text{minimize} && g(q_k, x_k, v, u) + \theta \\
& \theta, u \in \mathcal{U}_{q_k}, \lambda_{i,j} \geq 0, \\
& \forall i \in \mathcal{I}, j = 1, \dots, m_i \\
& \text{subject to} && A_{q_k} x_k + B_{q_k} u + c_{q_k} = \sum_{i \in \mathcal{I}} \sum_{j=1}^{m_i} \lambda_{i,j} V_{i,j}^p, \\
& && \theta \geq \sum_{i \in \mathcal{I}} \sum_{j=1}^{m_i} \lambda_{i,j} \tilde{J}_{k+1}(i, V_{i,j}^p), \\
& && \sum_{i \in \mathcal{I}} \sum_{j=1}^{m_i} \lambda_{i,j} = 1, \\
& && \sum_{j=1}^{m_i} \lambda_{i,j} \in \{0, 1\}, \quad \forall i \in \mathcal{I},
\end{aligned}$$

where $q_{k+1} \in \mathcal{I}$ and $v \in \mathcal{V}$ are such that, respectively, $\sum_{j=1}^{m_{q_{k+1}}} \lambda_{q_{k+1},j} = 1$ and $q_k \xrightarrow{v} q_{k+1}$, and $V_{i,j}^p$ is the orthogonal projection of $V_{i,j}$ onto $\mathcal{X}$.

Now, when relaxing the integrality constraints on the variables of the problem above,

$$\lambda_{i,j} \in [0, 1], \quad \forall i \in \mathcal{I}, j = 1, \dots, m_i,$$

we obtain its tightest possible relaxation since $(A_{q_k} x_k + B_{q_k} u + c_{q_k}, \theta)$ then belongs to the lower convex envelope of the graph of the approximation of the cost-to-go function over the modes in $\mathcal{I}$ [46].

A one-dimensional example illustrating this, with complementary explanations, is available in Figure 3.3.

The result of the relaxation described above is the following QP problem:

$$\begin{aligned}
& \text{minimize} && g(q_k, x_k, v, u) + \theta && (3.5) \\
& \theta, u \in \mathcal{U}_{q_k}, \lambda_{i,j} \geq 0, \\
& \forall i \in \mathcal{I}, j = 1, \dots, m_i \\
& \text{subject to} && A_{q_k} x_k + B_{q_k} u + c_{q_k} = \sum_{i \in \mathcal{I}} \sum_{j=1}^{m_i} \lambda_{i,j} V_{i,j}^p,
\end{aligned}$$

$$\theta \geq \sum_{i \in \mathcal{I}} \sum_{j=1}^{m_i} \lambda_{i,j} \tilde{J}(i, V_{i,j}^p),$$

$$\sum_{i \in \mathcal{I}} \sum_{j=1}^{m_i} \lambda_{i,j} = 1,$$

where $q_{k+1} \in \mathcal{I}$ and $v \in \mathcal{V}$ are such that, respectively, $A_{q_k} x_k + B_{q_k} u + c_{q_k} \in \text{Dom}(q_{k+1})$ and $q_k \xrightarrow{v} q_{k+1}$.

3.4.3 Termination criterion

Forward and backward passes are repeated until the algorithm converges towards a (locally) optimal trajectory.

One way to determine if the algorithm should stop iterating would be to monitor the change in the total cost of the trajectories planned by the MPC in the forward passes, but in some cases this cost seems to converge for a few iterations before decreasing at once because the algorithm is still actually learning. Another way consists in monitoring this learning by observing whether the new cuts modify the approximation of the cost-to-go function $\tilde{J}$, while checking on the convergence of the total cost, to see if the reference trajectory is still susceptible to change.

3.4.4 Results and improvements

Duality gap. Iterating allows the algorithm to gradually learn through the generation of new cuts refining the approximation of the cost-to-go function of the system. Unfortunately, the trajectories found are not always optimal certainly due to the duality gap between the exact cost-to-go function, which is non-convex, and its convex piecewise linear approximation. A solution to diminish this phenomenon might be to refine the hybrid automaton near the reference trajectory in such a way that the non-convexity of the cost-to-go function is reduced around it. A simple example demonstrating the splitting of a mode can be found in Figure 3.4, and its expected consequence on the duality gap in Figure 3.5. However, note that this refining must be done in a smart way as to not exacerbate the problem discussed just below.

Exploration vs exploitation. Another problem arises when some modes are not visited enough or at all during the forward passes, even though their promise of a low cost becomes more and more attractive with each new update of the approximation of the cost-to-go in the modes in their vicinity. This situation leads to the underestimation of the cost-to-go in these modes and in the ones from which the former are connected since the generation of the cuts in the latter is influenced by the quality of the approximation of the cost-to-go in their neighbors. This problem can be related to the *exploration versus exploitation* dilemma in reinforcement learning. Choosing the control

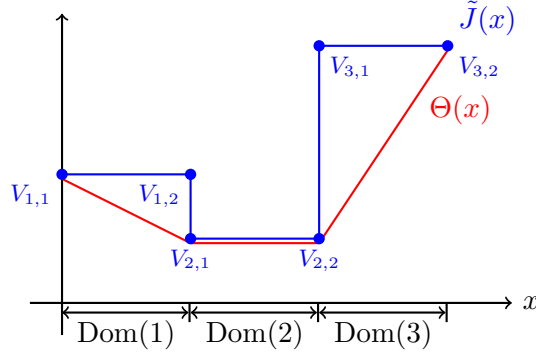


Figure 3.3: Example of relaxation in a one-dimensional setup. Let $\tilde{J}(x)$ (in blue) denote the non-convex approximation of the value function over the modes in $\mathcal{I} = \{1, 2, 3\}$, and $V_{i,j} \in \mathcal{X} \times \mathbb{R}_+$, $i \in \mathcal{I}$, $j \in \{1, 2\}$, the vertices describing it. The tightest convex underestimator of $\tilde{J}(x)$ is the lower convex envelope of its graph [47]: $\Theta(x) = \min_{\theta} (x, \theta)$ (in red), where $x = \sum_{i=1}^3 \sum_{j=1}^2 \lambda_{i,j} V_{i,j}^p$, $\theta \geq \sum_{i=1}^3 \sum_{j=1}^2 \lambda_{i,j} \tilde{J}(V_{i,j}^p)$, and $V_{i,j}^p$ the orthogonal projection of $V_{i,j}$ onto $\mathcal{X}$, subject to $\lambda_{i,j} \geq 0$, $\sum_{i=1}^3 \sum_{j=1}^2 \lambda_{i,j} = 1$.

inputs that are the best given a limited knowledge of the value function is exploitation. But, while this can obviously increase the speed of the search for an optimal trajectory, it can also lead to a poor exploration of the state-space, and a low quality approximation of the value function in those less explored parts. To remedy this, the algorithm might benefit from having an additional incentive to explore these rarely visited modes, or from the reduction of the horizon of the MPC, up to a point where they become attractive enough. Unfortunately, there is no evident answer that would allow to strike a balance between exploration and exploitation. It remains one of the key challenges in reinforcement learning [34], and is a classic pitfall of policy iteration algorithms [48].

Short-sighted MPC. If there are modes such that the only feasible transitions in the horizon of the MPC are self-loops, the controller can, in some situations, make the system go towards a region of the space which it cannot leave, and hence deny the forward pass any way of terminating in the desired terminal state. Fortunately, these situations can be resolved by simply removing such regions from the domain of the modes involved when they are encountered, and redirecting the forward pass towards a new course. A way to determine these regions is to determine the backward reachable tube of size M , where $M \in \mathbb{N}$ is a reasonably high number taking into account the parameters of the algorithm, of each guard set of the outgoing transitions in the mode and remove any regions outside their union.

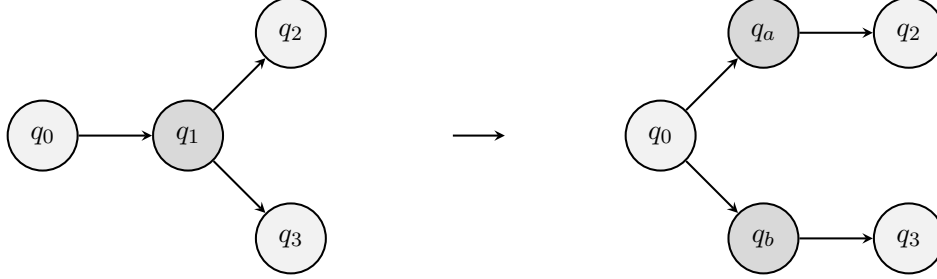


Figure 3.4: Example of the refinement of the hybrid automaton through mode splitting around the reference trajectory. The quality of the approximation of the value function in the mode q_1 is improved by splitting its domain into two non-overlapping parts in such a way that two new modes q_a and q_b are created which are only connected to either q_2 or q_3 and not both anymore. Since q_a and q_b are only connected to one other mode, (3.2) has a convex quadratic objective and the duality gap is annulled.

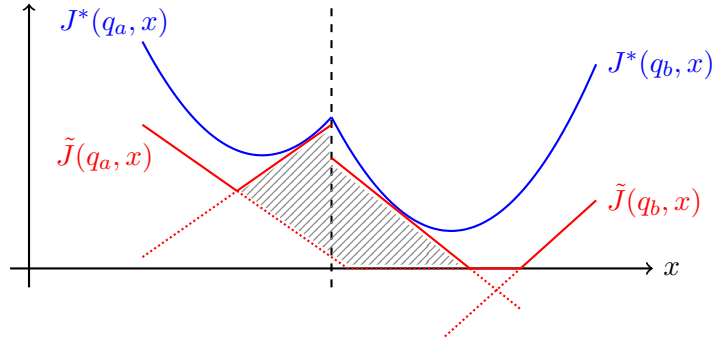


Figure 3.5: Example of the expected consequence on the duality gap of splitting a mode in a one-dimensional setup. The mode q_1 is split in two forming q_a and q_b where $\text{Dom}(q_1) = \text{Dom}(q_a) \cup \text{Dom}(q_b)$ and $\text{Dom}(q_a) \cap \text{Dom}(q_b) = \emptyset$ in the same way as in Figure 3.4, and two cuts are added to those already present in Figure 3.1. The duality gap between the previous cuts and J^* is due to the non-convexity of the situation anterior to the splitting of q_1 . The striped area represents the improvement over the previous situation.

Algorithm 3.1: Hybrid dual dynamic programming algorithm.

Input: Discrete-time affine hybrid automaton H , and its graph G_H .
 Desired terminal state (q_T, x_T) , and (soft) maximum number of iterations T to reach it. Initial hybrid state (q_0, x_0) .
 Positive quadratic convex cost function
 $g : \mathcal{Q} \times \mathcal{X} \times \mathcal{V} \times \mathcal{U} \rightarrow \mathbb{R}_+$. Minimum horizon N for the MPC.
 Function returning the number of transitions between two vertices in G_H : $d : \mathcal{Q} \times \mathcal{Q} \rightarrow \mathbb{N}$.

Output: Approximation of the cost-to-go function $\tilde{J}$.

```

1 // Initialization:
2  $\tilde{J}(q, x) = 0, \forall q \in \mathcal{Q}, x \in \mathcal{X}$ 
3 while cost has not converged do
4   // Forward pass (in Subsection 3.4.1):
5    $k \leftarrow 0$ , and cost  $\leftarrow 0$ 
6   while  $(q_k, x_k) \neq (q_T, x_T)$  and  $T > 0$  do
7     let  $\mathcal{H}$  the set of trajectories of length  $N$  in  $G_H$  starting in  $q_k$ ,
8     for  $h \in \mathcal{H}$  do
9        $(\text{obj}, \bar{v}_0, \bar{u}_0, \bar{q}_1, \bar{x}_1) \leftarrow \text{solve (3.4) // MPC}$ 
10      if obj is the best so far then
11         $v \leftarrow \bar{v}_0, \quad u \leftarrow \bar{u}_0, \quad q \leftarrow \bar{q}_1, \quad x \leftarrow \bar{x}_1$ 
12      // Decision tree (in Figure 3.2):
13      if  $\exists h \in \mathcal{H}$  s.t.  $d(q_k, q_T) < d(\bar{q}_N, q_T)$  then
14        if  $\exists h \in \mathcal{H}$  s.t.  $d(\bar{q}_N, q_T) < T - N$  then
15          if  $\exists h \in \mathcal{H}$  s.t. (3.4) is feasible then
16             $v_k \leftarrow v, \quad u_k \leftarrow u, \quad q_{k+1} \leftarrow q, \quad x_{k+1} \leftarrow x$ 
17            cost  $\leftarrow \text{cost} + g(q_k, x_k, v_k, u_k)$ 
18            reset  $N$  to its initial value
19             $T \leftarrow T - 1$ , and  $k \leftarrow k + 1$ 
20          else
21            if  $d(q_k, q_T) \geq T - N$  then
22               $N \leftarrow N + 1$ 
23            else
24               $T \leftarrow T + 1$ 
25        else
26           $T \leftarrow T + 1$ 
27      else
28         $N \leftarrow N + 1$ 
29   // Backward pass (in Subsection 3.4.2):
30   let  $n$  the number of iterates of the reference trajectory just
    computed in the forward pass,
31   for  $k = n, \dots, 0$  do
32      $(\alpha_k, \beta_k) \leftarrow \text{solve the dual of (3.4) with } x_k \text{ as right-hand side}$ 
    parameters
33      $\tilde{J}(q_k, x) \leftarrow \max\{\alpha_k^\top x + \beta_k, \tilde{J}(q_k, x)\}$ 

```


Chapter 4

Examples and Applications

The purpose of this chapter is to introduce the reader to the behavior and capabilities of the hybrid dual dynamic programming algorithm, presented in Chapter 3, through its application to the optimal control of a discrete-time piecewise affine control system, in Section 4.1, and a discrete-time linear control system under linear temporal logic (LTL) constraints, in Section 4.2.

—

The three main parameters of the algorithm are the horizon of the MPC, the desired maximum number of time steps for the forward pass to reach the terminal set, and whether the optional condition restricting the set of candidate trajectories for the MPC (see Subsection 3.4.1) is enabled. The influence of each of those parameters will be demonstrated in Section 4.2. But, before that, the first example will serve to describe how the algorithm operates on a relatively simple problem, and illustrate how the algorithm is able to learn an increasingly accurate approximation of the value function iteratively which allows the determination of counter-intuitive solutions to hybrid optimal control problems.

4.1 Simple switch

Example 4.1. We consider a discrete-time linear dynamical system with an autonomous switching. Let the time index $k \in \mathbb{N}$, the (continuous) state of the system $x_k \in \mathbb{R}$, the (continuous) control input $u_k \in \mathbb{R}$, and the switching region $\mathcal{S} = \{x \in \mathbb{R} : 0 \leq x \leq 1\}$,

$$x_{k+1} = \begin{cases} x_k + u_k + \frac{1}{2}, & \text{if } \exists i \leq k \text{ such that } x_i \in \mathcal{S}, \\ x_k + u_k + \frac{1}{100}, & \text{else.} \end{cases}$$

The specification is to start at the initial state $x_0 = \frac{5}{2}$, to stay in the safe region $\mathcal{X} = \{x \in \mathbb{R} : 0 \leq x \leq 7\}$, and to reach the target region

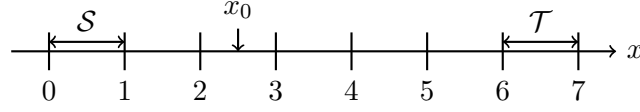


Figure 4.1: Graphical representation of the state-space of the discrete-time piecewise affine system in Example 4.1. The continuous state of the system is forbidden to leave the set $\mathcal{X} = \{x \in \mathbb{R} : 0 \leq x \leq 7\}$ and must reach the terminal set $\mathcal{T}$. The state space $\mathcal{X}$ is subdivided into seven polyhedral regions.

$\mathcal{T} = \{x \in \mathbb{R} : 6 \leq x \leq 7\}$. The continuous control constraint set is $\mathcal{U} = \{u \in \mathbb{R} : |u| \leq 1\}$.

A graphical representation of the problem can be seen in Figure 4.1.

This problem is hybrid by essence since the continuous dynamics change discontinuously when the continuous state of the system reaches a prescribed region of the space, $\mathcal{S}$. It can be described as a discrete-time affine hybrid automaton (Definition 3.1) where the continuous state space, $\mathcal{X}$, is arbitrarily divided into seven contiguous polyhedral regions of the same length, $\{x \in \mathbb{R} : i - 1 \leq x \leq i, i = 1, \dots, 7\}$, which define the domains of the modes of the system such that $\text{Dom}(1) = \mathcal{S}$, $\text{Dom}(7) = \mathcal{T}$, and $\text{Dom}(i) = \text{Dom}(14 - i)$, $i = 2, \dots, 6$. Hence, the discrete state space is chosen to be $\mathcal{Q} = \{1, \dots, 12\}$. Furthermore, a transition $\epsilon \in \mathcal{E}$ between two modes q_i and q_j is defined if q_j is reachable in one iteration from a subset of $\text{Dom}(q_i)$. Two exceptions to that rule is that there is no transition between the modes in $\{2, \dots, 6\}$ and $\{8, \dots, 12\}$, and no transition from 7 to any other mode. The guard of ϵ is the aforementioned subset, $\mathcal{G}_\epsilon = \{x \in \text{Dom}(q_i) : \exists u \in \mathcal{U}, Ax + Bu + c \in \text{Dom}(q_j)\}$, and its reset map is defined as $\mathcal{R}_\epsilon(x, u) = Ax + Bu + c$, where $c = \frac{1}{100}$ if $q_i \in \{2, \dots, 6\}$ and $c = \frac{1}{2}$ if $q_i \in \{1, 8, \dots, 12\}$.

A graphical representation of the graph of this discrete-time hybrid automaton can be seen in Figure 4.2.

4.1.1 Hybrid optimal control problem

To penalize the use of the continuous control input and the time to reach the desired state, the cost incurred at each time step is defined as

$$g(u) = u^2 + 1.$$

The total cost of a trajectory starting at (q_0, x_0) and finishing in n steps,

$$J(q_0, x_0 ; u_0, \dots, u_{n-1}) = \sum_{k=0}^{n-1} g(u_k) = \sum_{k=0}^{n-1} u_k^2 + n,$$

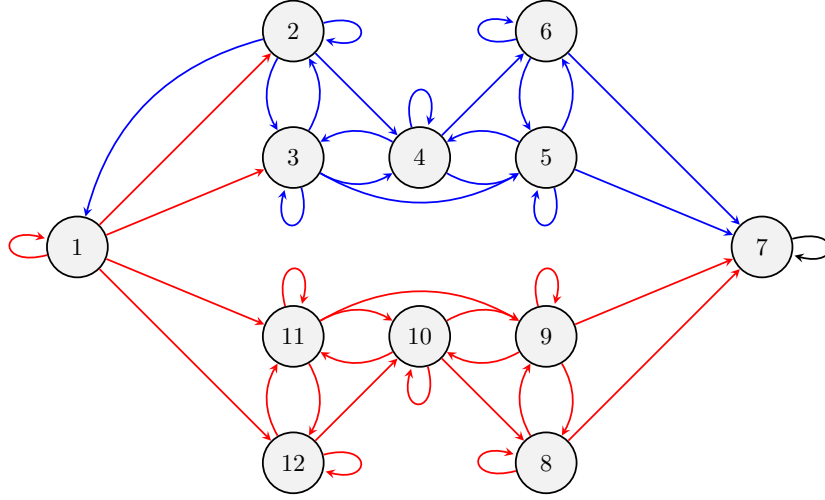


Figure 4.2: Graph of the discrete-time affine hybrid automaton of the discrete-time piecewise affine system in Example 4.1. The modes 8 to 12 are duplicates of the modes 2 to 6 concerning their domains ($\text{Dom}(i) = \text{Dom}(14 - i)$, $i = 2, \dots, 6$), but the reset maps of their exiting transitions are different. The ones in blue have reset maps with $\frac{1}{100}$ for constant term, while the ones in red have $\frac{1}{2}$. As for the self-loop of 7, its constant term might as well be zero. The domains of the outermost modes are $\text{Dom}(1) = \mathcal{S}$ and $\text{Dom}(7) = \mathcal{T}$. Note that if the initial state is in the upper part, the lower part cannot be reached unless the continuous state enters 1 and the switching happens.

is thus a convex function. However, the convexity of the unknown value function of the system is unlikely since the minimum of convex functions generally is not convex.

4.1.2 Results

The following results can be reproduced by running the Jupyter notebook `Bouchat_51191200_2019_Appendix1.ipynb` referenced in Appendix A.

Let the aforementioned hybrid optimal control problem be solved by the hybrid dual dynamic algorithm with, for the MPC, a horizon $N = 3$, a desired maximum of 10 time steps to reach the terminal set, and no optional restriction on the set of candidate trajectories in the graph of the automaton (see Subsection 3.4.1). Note that the 10 steps deadline is susceptible to increase as mentioned in the decision tree in Figure 3.2. The results obtained can be seen in Figure 4.3.

The behavior of the algorithm for the first five iterations is the following:

1. At first, the approximation of the value function is initialized at zero for the whole state-space, hence the controller uses no control at all for the first five time steps since it has no incentive to do otherwise. Then, the deadline approaching, the effect of the second condition on the set of candidate trajectories (see Subsection 3.4.1) compels the controller to direct the state towards the terminal set $\mathcal{T}$, which it manages to reach in six iterations. Afterwards, the backward pass takes place and optimality cuts are generated in the modes that have been explored, i.e. 3, 5, and 7.
2. In the second forward pass, the controller directs the state backwards (towards zero) immediately, against the dynamics of the system, since the new approximation of the value function is more favorable in the modes which have not been explored yet. However, the control remains weak, again, until the aforementioned second condition compels the controller to move. The current state is then close enough to the switching set $\mathcal{S}$ that the MPC realizes that visiting it is its best option in order to benefit from much more favorable dynamics ($\frac{1}{2}$ instead of $\frac{1}{100}$). The approximation of the value function is then refined in modes 1, 2, 3, 7, 9, 10, and 12.
3. The generation of these new optimality cuts renders the direct path (i.e. any $\{q_k, x_k\}_{k \in \mathbb{N}}$ such that $\forall i, x_i \geq x_0$) more appealing again and the controller directs the system towards it without waiting to be compelled to do so by the second condition this time.
4. Again, the increasingly better approximation of the value function just computed for some of the modes in the direct path now pushes

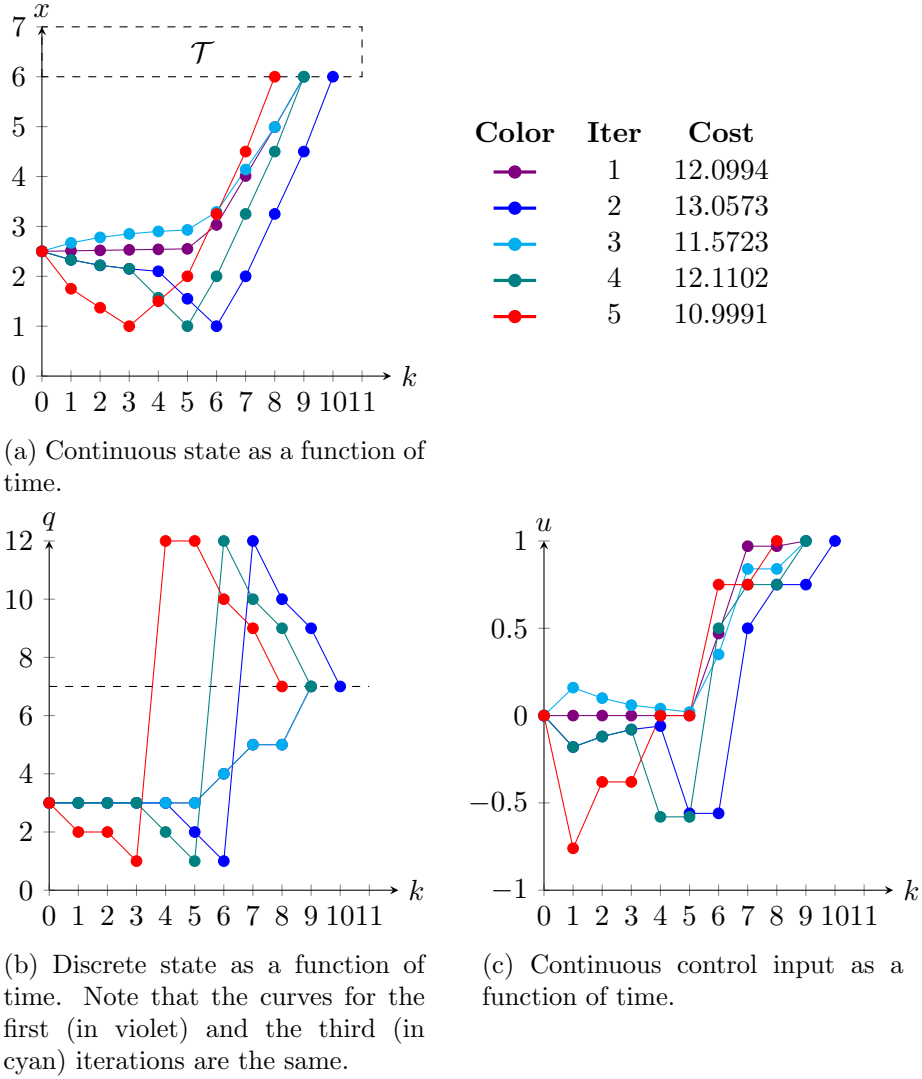


Figure 4.3: Results for the discrete-time piecewise affine system in Example 4.1 with $(q_0, x_0) = (3, \frac{5}{2})$ and a cost function $g(u) = u^2 + 1$. The MPC has an horizon $N = 3$, a desired maximum of 10 time steps to reach the terminal state $\mathcal{T}$, and the optional condition restricting the set of candidate trajectories (see Subsection 3.4.1) is disabled. Note that all iterations subsequent to the fifth result in trajectories which are the same as the latter.

the controller to direct the system towards the indirect path (i.e. any $\{q_k, x_k\}_{k \in \mathbb{N}}$ where $\exists i$ such that $x_i \leq x_0$).

5. Finally, this indirect path reveals itself to be less costly than the direct one and the algorithm converges at the fifth iteration, i.e. all subsequent iterations of the algorithm result in trajectories which are the same as this one.

This example demonstrates how the algorithm learns to consider trajectories that regular model predictive controllers would ignore (provided that their horizon does not extend way beyond the switching set initially) because they require the use of locally less than optimal control inputs for the sake of the (near-)optimality of the whole trajectory.

4.2 Double integrator

The application below was introduced in [28].

In this work, Applications 4.1 and 4.2 serve to illustrate how problems under LTL constraints can be expressed in the controlled hybrid automaton framework and hence be solved by the hybrid dual dynamic algorithm. Furthermore, results will be generated in order to demonstrate the effect of the optional condition restricting the set of candidate trajectories for the forward pass (see Subsection 3.4.1), and the impact of the horizon of the MPC.

Application 4.1 (Double integrator). We consider a discrete-time linear dynamical system. Let the time index $k \in \mathbb{N}$, the (continuous) state of the system $x_k \in \mathbb{R}^2$, and the (continuous) control $u_k \in \mathbb{R}$,

$$x_{k+1} = Ax_k + Bu_k,$$

where $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $x_0 = \begin{bmatrix} 1 \\ -6 \end{bmatrix}$, and the control constraint set is $\mathcal{U} = \{u \in \mathbb{R} : |u| \leq 2\}$. The specification is first for the state to reach either region $\mathcal{A} = \{x \in \mathbb{R}^2 : -6 \leq x_1 \leq -5, 1 \leq x_2 \leq 2\}$ or $\mathcal{B} = \{x \in \mathbb{R}^2 : -5 \leq x_1 \leq -3, -3 \leq x_2 \leq -2\}$, and then the target region $\mathcal{T} = \{x \in \mathbb{R}^2 : |x_i| \leq 0.5, i = 1, 2\}$, while always staying in the safe region $\mathcal{X} = \{x \in \mathbb{R}^2 : -10 \leq x_1 \leq 1.85, -10 \leq x_2 \leq 2\}$ and avoiding the obstacle regions $\mathcal{O}_1 = \{x \in \mathbb{R}^2 : -10 < x_i < -5, i = 1, 2\}$ and $\mathcal{O}_2 = \{x \in \mathbb{R}^2 : -5 < x_1 < 1.85, -4 < x_2 < -3\}$.

A graphical representation of the problem can be seen in Figure 4.4.

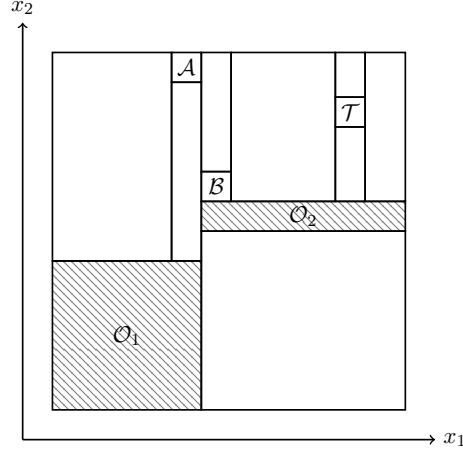


Figure 4.4: Graphical representation of the state-space of the discrete-time linear control system under LTL constraints in Application 4.1. The continuous state of the system is forbidden to enter the sets $\mathcal{O}_1$ and $\mathcal{O}_2$ and must reach either set $\mathcal{A}$ or $\mathcal{B}$ before the terminal set $\mathcal{T}$. The state space $\mathcal{X} \setminus (\mathcal{O}_1 \cup \mathcal{O}_2)$ is subdivided into eleven polyhedral regions.

4.2.1 Hybrid system formulation

Let H be a discrete-time affine hybrid automaton (Definition 3.1) where the discrete state space is $\mathcal{Q} = \{1, \dots, 20\}$ and the continuous state space is $\mathcal{X} \setminus (\mathcal{O}_1 \cup \mathcal{O}_2)$. The latter is arbitrarily divided into eleven non-overlapping polyhedral sets which define the domains of the modes such that $\text{Dom}(10) = \mathcal{A}$, $\text{Dom}(11) = \mathcal{B}$, $\text{Dom}(20) = \mathcal{T}$, and $\text{Dom}(i) = \text{Dom}(i + 11)$, $i = 1, \dots, 9$. Furthermore, a transition $\epsilon \in \mathcal{E}$ between two modes q_i and q_j is defined if q_j is reachable in one iteration from a subset of $\text{Dom}(q_i)$ without violating the obstacle regions. Two exceptions to that rule are that there are no transitions between the modes in $\{1, \dots, 9\}$ and $\{12, \dots, 20\}$, and that there is no transition from 10 and 11 to the modes in $\{1, \dots, 9\}$ and from $\{12, \dots, 20\}$ to 10 and 11. The guard of ϵ is the aforementioned subset and its reset map is defined as $\mathcal{R}_\epsilon(x, u) = Ax + Bu$.

A graphical representation of G_H , the graph of H , can be seen in Figure 4.5. Note that it is made of two (almost) identical parts only connected through modes 10 and 11, which have respectively $\mathcal{A}$ and $\mathcal{B}$ for domain, forcing the algorithm to select a trajectory passing through one of them provided the initial discrete state $q_0 \in \{1, \dots, 11\}$.

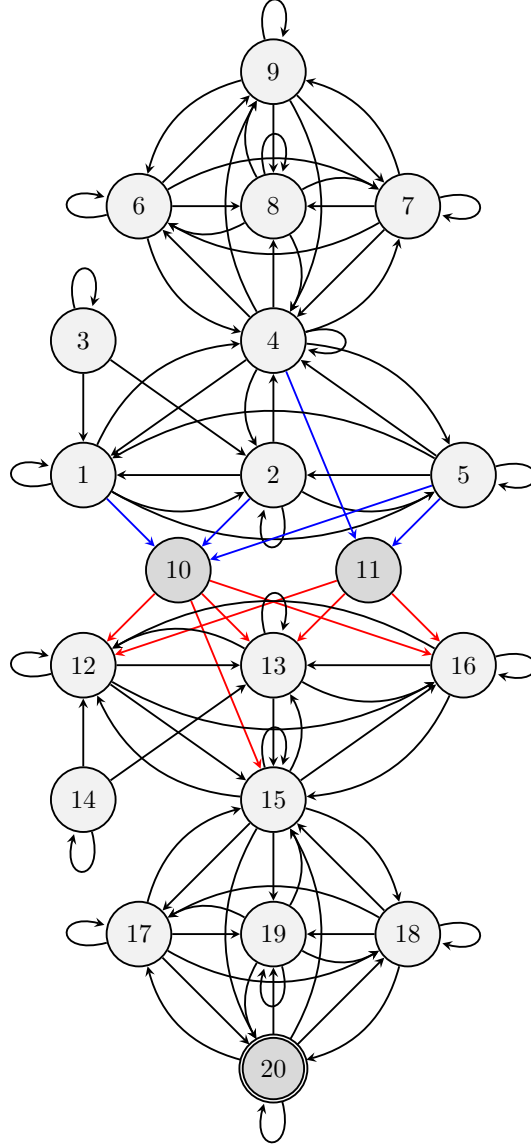


Figure 4.5: Graph of the discrete-time affine hybrid automaton of the discrete-time linear control system under LTL constraints in Application 4.1. Note that $\text{Dom}(10) = \mathcal{A}$, $\text{Dom}(11) = \mathcal{B}$, and $\text{Dom}(20) = \mathcal{T}$. By duplicating each mode of the automaton, except modes 10 and 11, and only allowing "in" transitions (in blue) from the upper part towards 10 and 11 and "out" transitions (in red) from 10 and 11 towards the lower part, the algorithm, when starting in a mode $i \leq 9$, has to go through 10 or 11 before finishing, so the requirements are fulfilled. The duplication was made according to the following rule: $\text{Dom}(i) = \text{Dom}(i + 11)$, $i = 1, 2, \dots, 9$.

4.2.2 Hybrid optimal control problem

The same cost function as defined in Subsection 4.1.1 is used. $g(u) = u^2 + 1$ is the cost incurred at each time step.

4.2.3 Results

The following results can be reproduced by running the Jupyter notebook `Bouchat_51191200_2019_Appendix2.ipynb` referenced in Appendix A.

Influence of the optional condition

The term *optional condition* refers to the first of the two conditions restricting the set of candidate trajectories in the graph of the automaton considered by the MPC during the forward pass, which were defined in order to force the controller to make the system reach the desired terminal state in a finite amount of time (see Subsection 3.4.1).

Figures 4.6 and 4.7 clearly show the effect of the optional condition on the behavior of the algorithm. Both figures show the trajectories obtained by the algorithm for the hybrid optimal control problem described above. In the case of Figure 4.7 the optional condition is enabled, and in Figure 4.6, it is disabled.

While it appears that, in this example, the convergence is slower when the optional condition is enabled, this result must be qualified. Indeed, it should be noted that the trajectories determined by the algorithm during the learning process vary much less when the option is enabled than when it is not. Hence the case could be made that, depending on the tolerance used to define the convergence of the cost, or even on whether the convergence is defined with relation to the shape of the trajectory rather than the cost, the algorithm actually converges faster to its final trajectory when the condition is enabled. That being said, this observation is not as relevant to describe the influence of the optional condition on the results as the fact that this restriction leads to a lack of diversity in the trajectories of the forward pass. This, in turn, leads to a lack of exploration of the state-space which, combined with the *a priori* elimination of trajectories that might nonetheless be less costly by the restriction on the set of candidate trajectories, appears to be responsible for the poorer results obtained when the optional condition is enabled.

What we can learn from these observations is that the use of the optional condition reduces the exploration of the state space, which can, in general, result in a lower performance of the algorithm. That being said, the link between a wider exploration and better results is not always directly verified

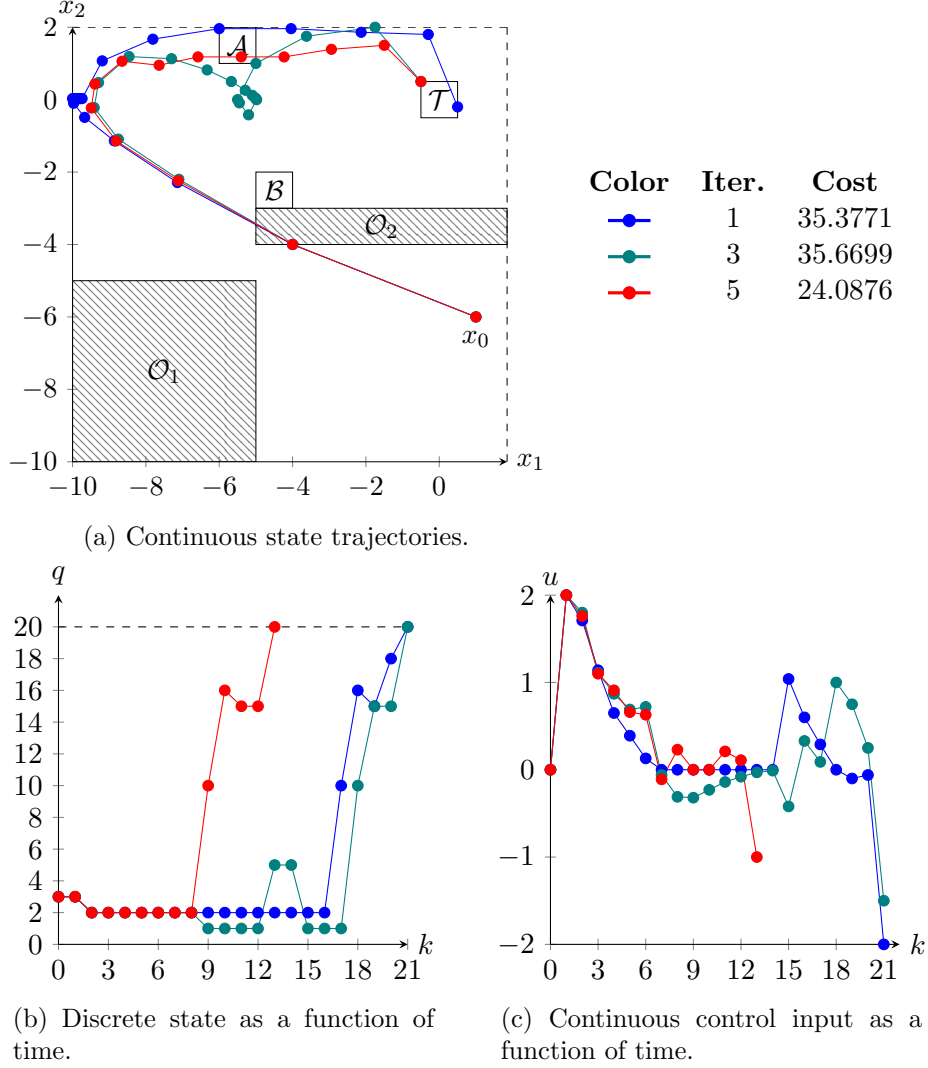
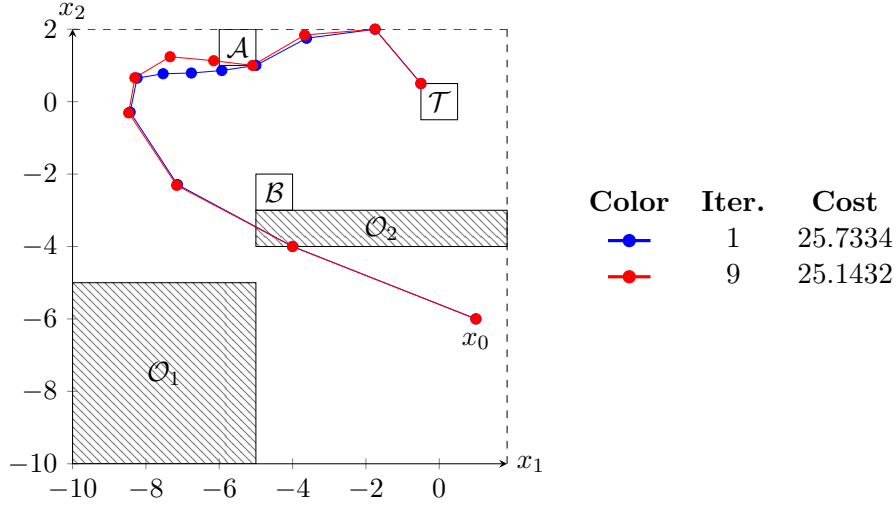
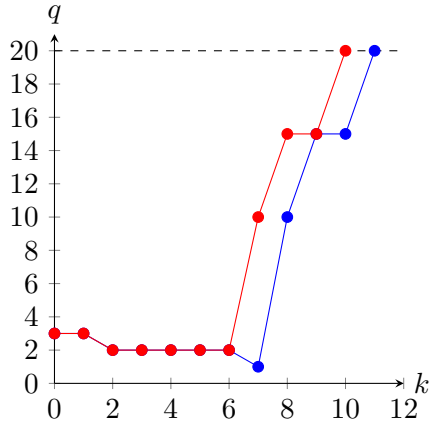


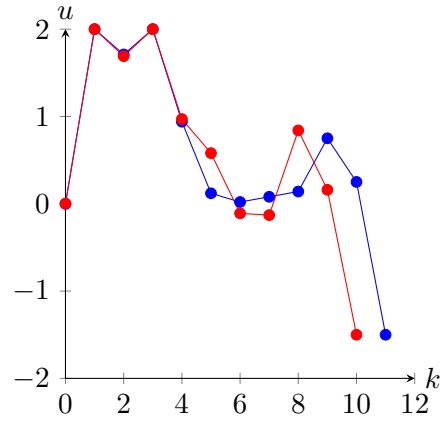
Figure 4.6: Results for the discrete-time linear control system under LTL constraints in Application 4.1 with $(q_0, x_0) = (3, [\frac{1}{-6}])$ and a cost function $g(u) = u^2 + 1$. The MPC has an horizon $N = 3$, a desired maximum of 20 time steps to reach the terminal state $\mathcal{T}$, and the optional condition restricting the set of candidate trajectories (see Subsection 3.4.1) is *disabled*. Note that all iterations subsequent to the fifth result in trajectories which are the same as the latter.



(a) Continuous state trajectories.



(b) Discrete state as a function of time.



(c) Continuous control input as a function of time.

Figure 4.7: Results for the discrete-time linear control system under LTL constraints in Application 4.1 with $(q_0, x_0) = (3, [\frac{1}{-6}])$ and a cost function $g(u) = u^2 + 1$. The MPC has an horizon $N = 3$, a desired maximum of 20 time steps to reach the terminal state $\mathcal{T}$, and the optional condition restricting the set of candidate trajectories (see Subsection 3.4.1) is *enabled*. Note that all iterations subsequent to the ninth result in trajectories which are the same as the latter.

since the duality gap is also of major significance in the learning process on which it can have a negative impact. What can be argued, however, is that a better exploration leads to an improvement of the accuracy of the approximation of the value function, and consequently, if there were no duality gap, one could observe a direct link between the quality of the exploration and that of the result.

Influence of the desired maximum number of time steps

This parameter denotes the maximum number of iterations allowed for the forward pass to reach the desired terminal state. However, note that the use of the term *maximum* here is somewhat abusive since this parameter is susceptible to be incrementally increased (see the decision tree in Figure 3.2).

The effect of the desired maximum number of time steps for the forward pass to reach the terminal set is especially visible in Figure 4.6. Because of the nature of the system, if the controller is not compelled to go towards the terminal state, it has a tendency to try and reach the region $\{x \in \mathcal{X} \subseteq \mathbb{R}^2 : x_2 = 0\}$ where the cost incurred at each time step is minimized by not using any continuous control when the approximation of the value function is still zero in the mode that the system is in. This behavior is clear in the first and third iterations, where it happens in two different modes which were not explored before they reached them.

The purpose of the second condition on the set of candidate trajectories for the forward pass is precisely to avoid a situation where the forward pass never finishes, which renders the backward pass useless, as mentioned in Subsection 3.4.1, and the time at which it takes effect depends on the aforementioned desired maximum number of iterations. The smaller it is, the less time the trajectories will stay still, or at least not move more or less directly towards the desired terminal state, and the larger the longer. Note that this parameter has thus an influence on the exploration. Indeed, the smaller its value, the less time it has to explore the state-space before the second condition forces it to go nearly directly towards the desired terminal state, which can have a nefarious effect on the approximation of the value function, as mentioned in Subsection 3.4.4.

In Figure 4.6c, one can clearly see that the second condition begins forcing the controller to move on towards the terminal state on the fourteenth time step for the first and third iterations because the algorithm assumes that reaching the end should take him at least six more time steps and the desired maximum number of iterations is set at twenty.

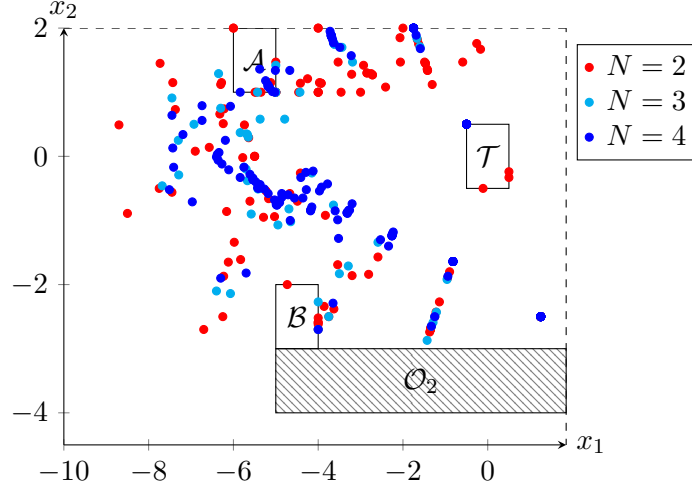


Figure 4.8: All the continuous state iterates generated during all the forward passes of the algorithm (until convergence) with horizons $N = 2$ (in red), 3 (in cyan), and 4 (in blue) for the discrete-time linear control system under LTL constraints in Application 4.2 with $(q_0, x_0) = (8, [\frac{1.25}{-2.5}])$ and a cost function $g(u) = u^2 + 1$. The MPCs have a desired maximum of 20 time steps to reach the terminal state $\mathcal{T}$, and the optional condition restricting the set of candidate trajectories (see Subsection 3.4.1) is enabled. Note that some iterates are superposed: the ones in blue are on top of the ones in cyan which are themselves on top of the ones in red, and that only a part of the state-space is visible for the sake of clarity.

Influence of the horizon

The depth N of the horizon of the MPC in the forward pass has a crucial influence on the behavior of the algorithm. To illustrate it, let us first repurpose the double integrator defined above:

Application 4.2. We consider the same discrete-time linear dynamical system and specification as in Application 4.1, but a different initial state: $x_0 = [\frac{1.4}{-2.8}]$.

The resolution of the hybrid optimal control problem defined by Application 4.2 and the same cost function, $g(u) = u^2 + 1$, as earlier results in Figures 4.9, 4.10, and 4.11, which are the trajectories obtained for $N = 2$, $N = 3$, and $N = 4$ respectively.

One can clearly see in Figure 4.8 the link between the depth of the horizon and the exploration of the state-space. The deeper the horizon, the shallower the exploration, which, as mentioned in Subsection 3.4.4, results

in an underestimation of the value function in the modes that are visited less often and in those in their vicinity. Because of its detrimental effect on the precision of the approximation of the value function, this lack of exploration can thus indirectly induce a decrease in the performance of the algorithm, as attested by the results in Figures 4.9, 4.10, and 4.11.

As to why deepening the horizon results in a decrease in the exploration of the state-space, an intuitive explanation is that a shallow horizon renders the controller more susceptible to go towards parts of the state-space that are such that going within them seems less costly in the short term, even though it results in an increase in the total cost of the trajectory in the long run. A priori unappealing modes are thus more likely to be visited, and more often, if the controller is kept in the dark thanks to a shallow horizon. It should be noted, however, that decreasing the depth of the horizon also makes the controller more likely to direct the system towards regions of the state-space where it can become stuck, a problem which was discussed in the last paragraph of Subsection 3.4.4.

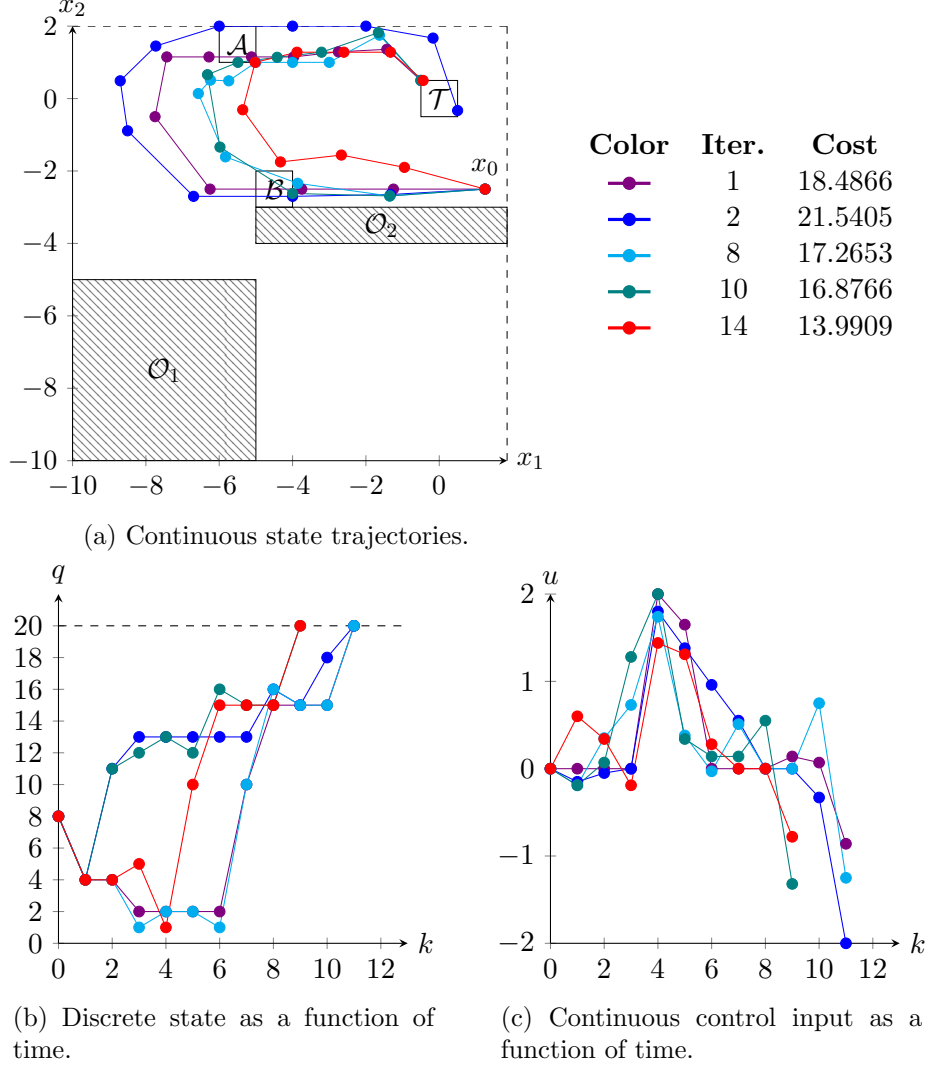


Figure 4.9: Results for the discrete-time linear control system under LTL constraints in Application 4.2 with $(q_0, x_0) = (8, [1.25, -2.5])$ and a cost function $g(u) = u^2 + 1$. The MPC has an horizon $N = 2$, a desired maximum of 20 time steps to reach the terminal state $\mathcal{T}$, and the optional condition restricting the set of candidate trajectories (see Subsection 3.4.1) is enabled. Note that all iterations subsequent to the fourteenth result in trajectories which are the same as the latter.

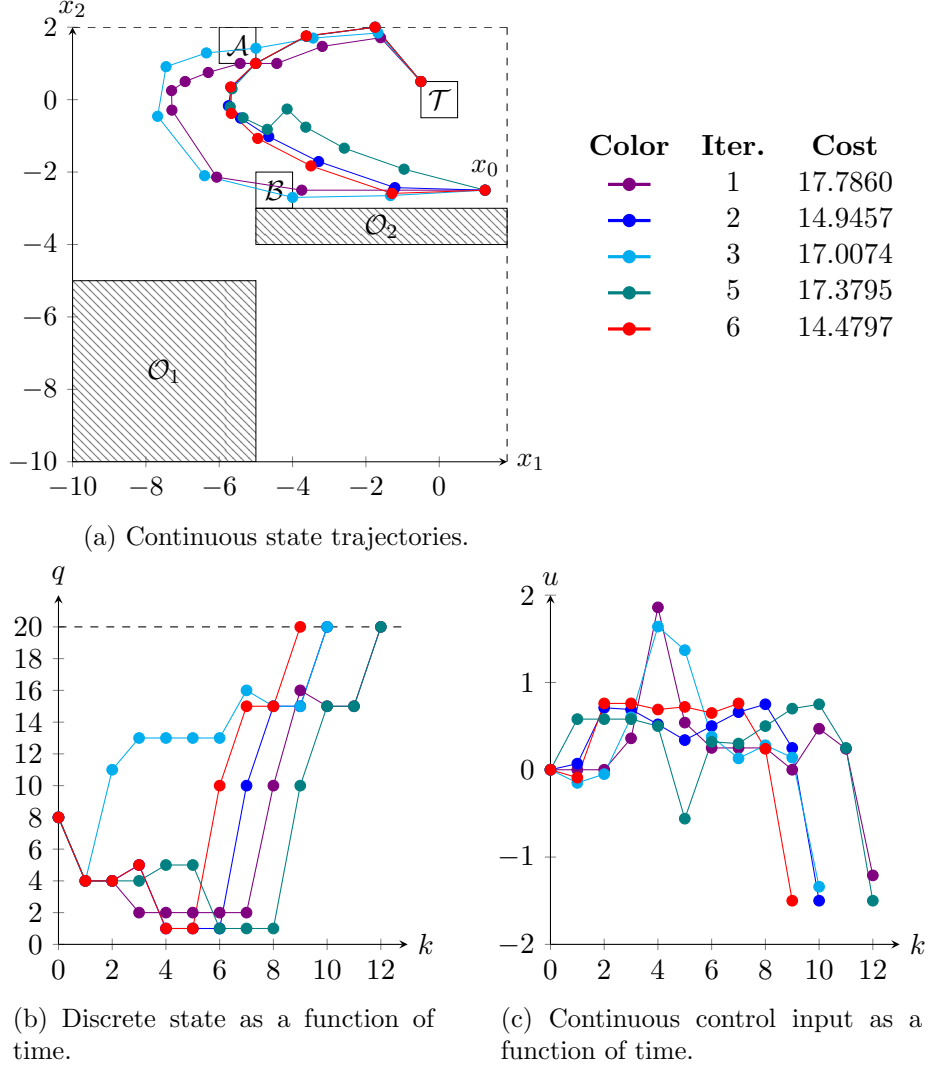


Figure 4.10: Results for the discrete-time linear control system under LTL constraints in Application 4.2 with $(q_0, x_0) = (8, \begin{bmatrix} 1.25 \\ -2.5 \end{bmatrix})$ and a cost function $g(u) = u^2 + 1$. The MPC has an horizon $N = 3$, a desired maximum of 20 time steps to reach the terminal state $\mathcal{T}$, and the optional condition restricting the set of candidate trajectories (see Subsection 3.4.1) is enabled. Note that all iterations subsequent to the sixth result in trajectories which are the same as the latter.

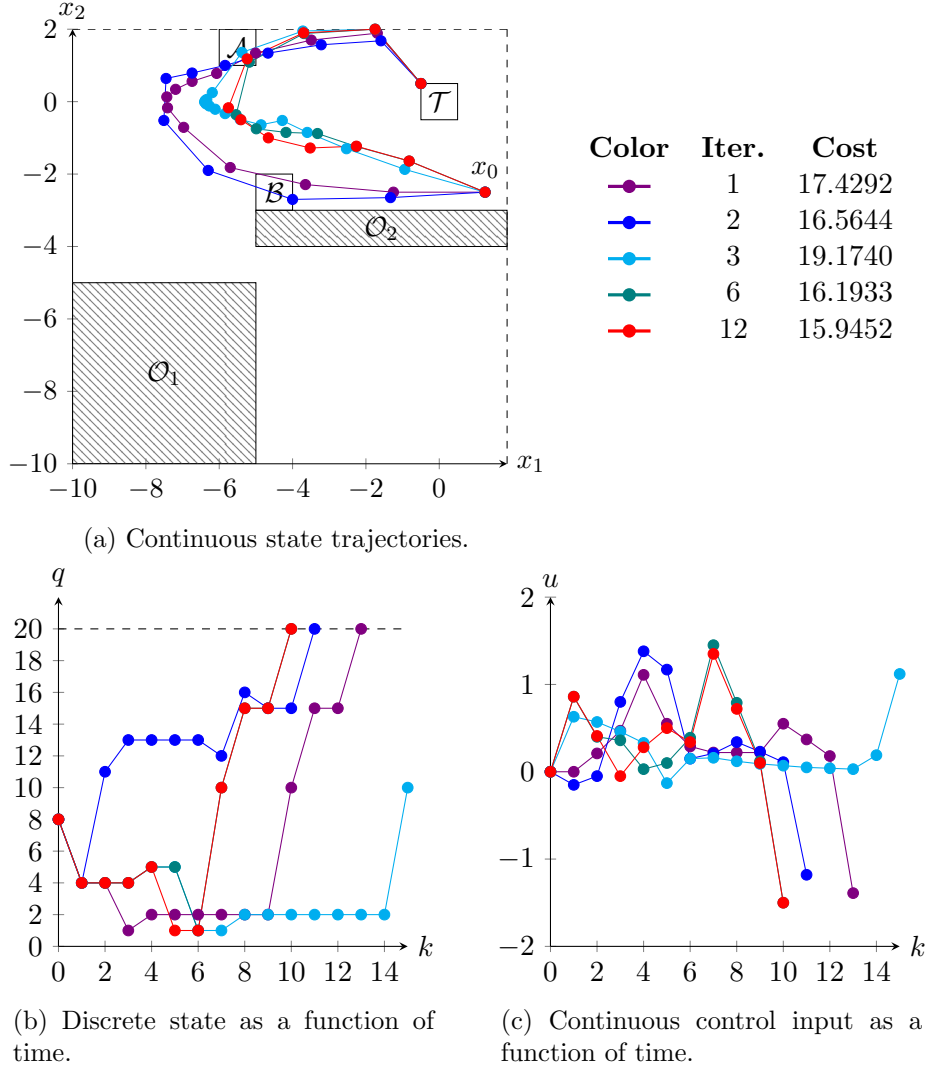


Figure 4.11: Results for the discrete-time linear control system under LTL constraints in Application 4.2 with $(q_0, x_0) = (8, [\frac{1.25}{-2.5}])$ and a cost function $g(u) = u^2 + 1$. The MPC has an horizon $N = 4$, a desired maximum of 20 time steps to reach the terminal state $\mathcal{T}$, and the optional condition restricting the set of candidate trajectories (see Subsection 3.4.1) is enabled. Note that all iterations subsequent to the fifth result in trajectories which are the same as the latter.

Chapter 5

Conclusion

We have presented in this work a model-based reinforcement learning algorithm, which combines, in a policy iteration scheme, two techniques commonly used for the control of dynamical systems: dual dynamic programming and model predictive control. We called it the *hybrid dual dynamic programming* algorithm.

The main contribution of this work is the design and the showing of the applicability of this algorithm, which alternates between determining a near-optimal trajectory with a model predictive controller, which exploits an approximation of the value function, and refining this approximation via dual dynamic programming, to the resolution of time-invariant discrete-time affine hybrid optimal control problems.

Concerning the performance of the algorithm, which was discussed in Subsection 3.4.4, it should be noted that, while the algorithm converges in a small number of iterations for all the problems it has been tested on, the trajectories obtained have no guarantee to be optimal. This expected, and yet somewhat disappointing, result can always to some extent be attributed to the duality gap resulting from the non-convexity of the value function. The other challenging problem decreasing, sometimes drastically, the efficiency of the algorithm in some cases is the under-exploration of parts of the discrete state-space during the forward passes which tends to reduce the accuracy of the approximation of the value function in their neighborhood and accentuate the size of the aforementioned duality gap.

Trying to reduce the impact of these problems which are inherent to the algorithm could be the subject of future work. A first approach to improve the accuracy of the optimality cuts might be to refine the hybrid automaton around the reference trajectory by dividing its modes, and, potentially, to modify the strategy used in the forward pass to obtain reference trajectories in such a way that more of the state-space is explored and, hopefully, strike a balance between exploration and exploitation.

Finally, now that a proof of concept has been provided for the hybrid dual

dynamic programming algorithm, another good suggestion for future work would be to try and optimize it, and then offer a quantitative comparison of its relative performance of against state-of-the-art methods.

Bibliography

- [1] A. Benveniste, T. Bourke, B. Caillaud, and M. Pouzet, “Hybrid systems modeling challenges caused by cyber-physical systems,” *Cyber-Physical Systems (CPS) Foundations and Challenges*. Available online: <http://people.rennes.inria.fr/Albert.Benveniste/pub/NIST2012.pdf>, 2013.
- [2] M. S. Branicky, “Introduction to hybrid systems,” in *Handbook of networked and embedded control systems*, Springer, 2005, pp. 91–116.
- [3] R. Goebel, R. G. Sanfelice, and A. R. Teel, “Hybrid dynamical systems,” *IEEE Control Systems Magazine*, vol. 29, no. 2, pp. 28–93, 2009.
- [4] A. J. Van Der Schaft and J. M. Schumacher, *An introduction to hybrid dynamical systems*. Springer London, 2000, vol. 251.
- [5] T. Stauner, “Properties of hybrid systems—a computer science perspective,” *Formal Methods in System Design*, vol. 24, no. 3, pp. 223–259, 2004.
- [6] W. Heemels, D. Lehmann, J. Lunze, and B. De Schutter, “Introduction to hybrid systems,” *Handbook of Hybrid Systems Control–Theory, Tools, Applications*, pp. 3–30, 2009.
- [7] T. A. Henzinger, “The theory of hybrid automata,” in *Verification of Digital and Hybrid Systems*, Springer, 2000, pp. 265–292.
- [8] S. Engell, H. Gueguen, and J. Zaytoon, *Analysis and Design of Hybrid Systems 2003 (ADHS 03): A Proceedings Volume from the IFAC Conference, St. Malo, Brittany, France, 16-18 June 2003*. Elsevier, 2003.
- [9] R. Bellman, *Dynamic programming*. Courier Corporation, 2013.
- [10] L. S. Pontryagin, *Mathematical theory of optimal processes*. Routledge, 2018.
- [11] M.-t. Shing *et al.*, *Combinatorial Algorithms: TC Hu and MT Shing*. Courier Corporation, 2002.

- [12] A. Pakniyat and P. E. Caines, “On the minimum principle and dynamic programming for hybrid systems,” *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 9629–9634, 2014.
- [13] M. Sheikh, “On the hybrid dynamic programming principle,” in *International Multi Topic Conference, 2002. Abstracts. INMIC 2002.*, IEEE, 2002, pp. 59–59.
- [14] A. Bensoussan and J. Menaldi, “Hybrid control and dynamic programming,” *Dynamics of Continuous, Discrete and Impulsive Systems Series B: Application and Algorithm*, vol. 3, no. 4, pp. 395–442, 1997.
- [15] D. P. Bertsekas, “Dynamic programming and optimal control 3rd edition, volume ii,” *Belmont, MA: Athena Scientific*, 2011.
- [16] M. Buss, O. von Stryk, R. Bulirsch, and G. Schmidt, “Towards hybrid optimal control,” *at-Automatisierungstechnik Methoden und Anwendungen der Steuerungs-, Regelungs-und Informationstechnik*, vol. 48, no. 9/2000, p. 448, 2000.
- [17] A. V. Dmitruk and A. M. Kaganovich, “The hybrid maximum principle is a consequence of pontryagin maximum principle,” *Systems & Control Letters*, vol. 57, no. 11, pp. 964–970, 2008.
- [18] H. J. Sussmann, “A maximum principle for hybrid optimal control problems,” in *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No. 99CH36304)*, Ieee, vol. 1, 1999, pp. 425–430.
- [19] J. Lunze and F. Lamnabhi-Lagarigue, *Handbook of hybrid systems control: theory, tools, applications*. Cambridge University Press, 2009.
- [20] T. Ruby, “Optimal control of hybrid power system,” *Jurnal Sains MIPA Universitas Lampung*, vol. 5, no. 1, 2012.
- [21] X. Wu, K. Zhang, and M. Cheng, “Optimal control of constrained switched systems and application to electrical vehicle energy management,” *Nonlinear Analysis: Hybrid Systems*, vol. 30, pp. 171–188, 2018.
- [22] P. Martin and M. Egerstedt, “Optimal timing control of interconnected, switched systems with applications to robotic marionettes,” in *2008 9th International Workshop on Discrete Event Systems*, IEEE, 2008, pp. 156–161.
- [23] F. Zhu and P. J. Antsaklis, “Optimal control of hybrid switched systems: A brief survey,” *Discrete Event Dynamic Systems*, vol. 25, no. 3, pp. 345–364, 2015.
- [24] C. Iung and P. Riedinger, “Optimal control in hybrid systems,” *IFAC Proceedings Volumes*, vol. 39, no. 5, pp. 4–5, 2006.

- [25] E. F. Camacho, D. R. Ramírez, D. Limón, D. M. De La Peña, and T. Alamo, “Model predictive control techniques for hybrid systems,” *Annual reviews in control*, vol. 34, no. 1, pp. 21–31, 2010.
- [26] A. Bemporad and M. Morari, “Control of systems integrating logic, dynamics, and constraints,” *Automatica*, vol. 35, no. 3, pp. 407–427, 1999.
- [27] B. Legat, P. Tabuada, and R. M. Jungers, “Computing controlled invariant sets for hybrid systems with applications to model-predictive control,” *IFAC-PapersOnLine*, vol. 51, no. 16, pp. 193–198, 2018.
- [28] E. Aydin Gol and M. Lazar, “Temporal logic model predictive control for discrete-time systems,” in *Proceedings of the 16th international conference on Hybrid systems: computation and control*, ACM, 2013, pp. 343–352.
- [29] E. A. Gol, M. Lazar, and C. Belta, “Language-guided controller synthesis for linear systems,” *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1163–1176, 2013.
- [30] M. V. Pereira and L. M. Pinto, “Multi-stage stochastic optimization applied to energy planning,” *Mathematical programming*, vol. 52, no. 1-3, pp. 359–375, 1991.
- [31] S. Cerisola, J. M. Latorre, and A. Ramos, “Stochastic dual dynamic programming applied to nonconvex hydrothermal models,” *European Journal of Operational Research*, vol. 218, no. 3, pp. 687–697, 2012.
- [32] V. Guigues, “Dual dynamic programming with cut selection: Convergence proof and numerical experiments,” *European Journal of Operational Research*, vol. 258, no. 1, pp. 47–57, 2017.
- [33] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [34] W. B. Powell, “Perspectives of approximate dynamic programming,” *Annals of Operations Research*, vol. 241, no. 1-2, pp. 319–356, 2016.
- [35] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*. Athena Scientific Belmont, MA, 1996, vol. 5.
- [36] D. Görges, “Relations between model predictive control and reinforcement learning,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4920–4928, 2017.
- [37] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
- [38] D. Ernst, M. Glavic, P. Geurts, and L. Wehenkel, “Approximate value iteration in the reinforcement learning context. application to electrical power system control,” *International Journal of Emerging Electric Power Systems*, vol. 3, no. 1, 2005.

- [39] D. P. Bertsekas, “Value and policy iterations in optimal control and adaptive dynamic programming,” *IEEE transactions on neural networks and learning systems*, vol. 28, no. 3, pp. 500–509, 2015.
- [40] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, “Julia: A fresh approach to numerical computing,” *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.
- [41] J. Warrington, P. N. Beuchat, and J. Lygeros, “Generalized dual dynamic programming for infinite horizon problems in continuous state and action spaces,” *arXiv preprint arXiv:1711.07222*, 2017.
- [42] B. Mehta and Y. Reddy, “Chapter 19 - advanced process control systems,” in *Industrial Process Automation Systems*, Oxford: Butterworth-Heinemann, 2015, pp. 547–557, ISBN: 978-0-12-800939-0.
- [43] J. R. Birge and F. Louveaux, *Introduction to stochastic programming*. Springer Science & Business Media, 2011.
- [44] A. Bemporad, “Solving mixed-integer quadratic programs via nonnegative least squares,” *IFAC-PapersOnLine*, vol. 48, no. 23, pp. 73–79, 2015.
- [45] R. Lazimy, “Mixed-integer quadratic programming,” *Mathematical Programming*, vol. 22, no. 1, pp. 332–349, 1982.
- [46] J. P. Vielma, S. Ahmed, and G. Nemhauser, “Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions,” *Operations research*, vol. 58, no. 2, pp. 303–315, 2010.
- [47] J. P. Vielma, “Mixed integer linear programming formulation techniques,” *Siam Review*, vol. 57, no. 1, pp. 3–57, 2015.
- [48] D. P. Bertsekas, “Approximate policy iteration: A survey and some new methods,” *Journal of Control Theory and Applications*, vol. 9, no. 3, pp. 310–335, 2011.
- [49] B. Legat, M. Forets, and C. Schilling, *Hybridsystems.jl: V0.1.5*, Apr. 2019. DOI: 10.5281/zenodo.2640699. [Online]. Available: <https://doi.org/10.5281/zenodo.2640699>.
- [50] L. Gurobi Optimization, *Gurobi optimizer reference manual*, 2018. [Online]. Available: <http://www.gurobi.com>.
- [51] I. Dunning, J. Huchette, and M. Lubin, “Jump: A modeling language for mathematical optimization,” *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017. DOI: 10.1137/15M1020575.
- [52] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, “OSQP: An operator splitting solver for quadratic programs,” *ArXiv e-prints*, Nov. 2017. arXiv: 1711.08013 [math.OC].

- [53] B. Legat, R. Deits, O. Evans, G. Goretkin, T. Koolen, J. Huchette, D. Oyama, M. Forets, G. Berger, R. Schwarz, E. Saba, and C. Coleman, *Polyhedra.jl: V0.5.1*, May 2019. DOI: 10.5281/zenodo.3066026. [Online]. Available: <https://doi.org/10.5281/zenodo.3066026>.

Appendix A

Julia codes

Main packages

Hybrid systems

`HybridSystems.jl` (**v0.1.5**) - Interface for defining and working with Hybrid Systems [49].
<https://github.com/blegat/HybridSystems.jl>

Optimization

`Gurobi.jl` (**v0.6.0**) - Julia wrapper of the Gurobi solver [50] (through its C interface).
<https://github.com/JuliaOpt/Gurobi.jl>

`JuMP.jl` (**v0.19.0**) - JuMP is a domain-specific modeling language for mathematical optimization embedded in Julia [51].
<https://github.com/JuliaOpt/JuMP.jl>

`OSQP.jl` (**v0.5.1**) - Julia wrapper for the Operator Splitting QP Solver (OSQP) [52].
<https://github.com/oxfordcontrol/OSQP.jl>

`ParameterJuMP.jl` (**v0.0.1**) - - JuMP extension to use parameters in constraints constants.
<https://github.com/JuliaStochOpt/ParameterJuMP.jl>

Others

`CDDLib.jl` (**v0.4.1**) - Wrapper for `cdd`.
<https://github.com/JuliaPolyhedra/CDDLib.jl>

IJulia.jl (v1.18.0) - Julia-language backend combined with the Jupyter interactive environment.

<https://github.com/JuliaLang/IJulia.jl>

Polyhedra.jl (v0.5.1) - Unified interface for Polyhedral Computation Libraries such as CDDLib.jl [53].

<https://github.com/JuliaPolyhedra/Polyhedra.jl>

Notebooks

Simple switch in Section 4.1

[Bouchat_51191200_2019_Appendix1.ipynb](#)

Double integrator in Section 4.2

[Bouchat_51191200_2019_Appendix2.ipynb](#)

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl