DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2016

Mutli agent control with LTL specifications and abstraction with input memories

PAUL ROUSSE

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Mutli agent control with LTL specifications and abstraction with input memories

PAUL ROUSSE

Master of Science Thesis

KTH
SE-100 44 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av i .

Tryck:

Abstract

Formal controller synthesis methods with temporal logic specifications try to guarantee the correctness of given specifications with a system that might have complex behaviour. Most often, discrete abstractions of a continuous system are used in order to reduce the size of the problem. If a behavioural relationship between the system and its abstraction is met, then a valid controller for the abstraction will be valid as well for the system. The finiteness of the abstraction allows us to use computer science tools to synthesize a finite state controller (graphs search and fixed point algorithms, temporal logics). These methods are automatic (the controller synthesis can be created automatically from the specifications) and offline (the controller is created beforehand).

In this work, we investigate a system with a state extended by memories of the last inputs. These memories are used to compute the reachable sets, this counteract the loss of information from a partial observation of the state. The final abstraction is obtained by a discretization of the resulting state space, and expressed as an Augmented Finite Transition System (AFTS). This AFTS extends the classical Finite Transition System definition with a progress set that identifies terminating local control strategies. As the AFTS is alternatingly simulated by the extended system, a controller solution of the problem for the AFTS is a solution for the extended system. Specifications of the controller are given in Linear Temporal Logic formula (LTL). The product of the non deterministic AFTS and the Büchi automaton of the LTL specification is defined. An algorithm that can find non-blocking control strategy for the AFTS is then used to find a finite state controller.

This controller synthesis method with Linear Temporal Logic specifications has been successfully used and tested on single and multiple quadricopters scenarios in the Smart Mobility Lab of KTH.

Sammanfattning

Formella metoder för att syntetisera regulatorer med temporal logiks specifikationer försöker att garantera att den givna specifikationen uppfylls av ett system som kan ha komplexa beteenden. Oftast används diskreta abstraktioner av kontinuerliga system för att minska problemets storlek. Om en beteende-relation mellan systemet och dess abstraktion är uppfylld så är en regulator som är giltig för abstraktionen också giltig för systemet. Ändligheten av abstraktionen tillåter oss att använda datavetenskapliga verktyg för att syntetisera en ändlig tillståndsregulator (grafsökning och fastpunktsalgorithmer, temporal logik). Dessa metoder är automatiska (regulator syntetiseringen kan göras automatiskt från specifikationen) och offline (regulatorn skapas i förväg).

I det här arbetet undersöker vi ett system med utvidgat tillstånd som har minnen som sista indata. Dessa minnen används för att beräkna seten av nåbara tillstånd. Detta motverkar förlusten av information som kan uppstå på grund av en partiell observation av tillståndet. Den slutgiltiga abstraktionen fås genom en diskretisering av det resulterande tillståndsrummet och uttrycks som ett Utökat Ändligt Övergångs System (Augmented Finite Transition System - AFTS). AFTSen är en utökning av den klassiska Ändliga Övergångs Systems definitionen med ett utvecklings set som identifierar avslutande lokala regler strategier. Eftersom AFTSen är en alternerande simulering av det utökade systemet så är en regulator som löser AFTS problemet också en lösning för det utökade systemet. Specifikationerna för problemet är givna som Linjär Temporal Logiks formler (Linear Temporal Logic - LTL). Produkten av den icke-deterministiska AFTSen och Büchi automaten av LTL specifikationen är definierad. En algoritm som kan hitta icke-blockerande regler strategier för AFTSen används för att hitta en ändlig tillståndsregulator.

Den här metoden för syntetisering av regulatorer för Linjär Temporal Logiks specifikationer har använts framgångsrikt och har testats på scenarier med enstaka samt flera quadrocopters i Smart Mobility Lab på KTH.

Acknowledgements

This thesis was conducted in the Smart Mobility Lab (SML) at KTH. I owe my gratitude to all my colleagues that helped me throughout this work. Most particularly to my supervisor Pierre-Jean Meyer for his guidance and Dimos Dimarogonas for giving me the opportunity to work with him.

I am also grateful to Antonio and Pedro who put so much effort in the lab; without them, quads would barely walk! I would like to give special thanks to Rui who has always been very helpful in the lab (even as a poorly designed cardboard scarecrow). Lars, Spyros and Matthieu gave me wonderful feedback on my work that have helped me going further in my research.

I met (too many) master students during work, Manuel, Mario, Riccardo, Giorgio, Spyros, Ziwei, Antonio, Massimiliano, Mateo and Pedro: it has been a real pleasure to work with you and I wish you all the best for your future! I am thankful as well to: Sofie, David, Christos, Stefanos, Gonçalo, Lars, Konstantina, Gustav, Jérémy, Anaïs, Rémi, Nicolas, Elsa, Juliette and Félix.

Last but not least, I thank my mother Danielle, my father Vincent, my sister Cécile and my brother Mathieu for there stunning support despite the distance.

Contents

Introduction

Robot motion planning with temporal logic specifications has received a lot of attention in the control community. More recently, the expressiveness of temporal specifications coupled with automated control synthesis methods brought powerful tools. Such tools can be used for provably correct control and planning design of robotic systems under high-level specifications in the form of temporal logic formulas (Belta *et al.*, 2007).

The Linear Temporal Logic (LTL) specification language provides an expressive framework where safety, reachability and reactive properties are suitably formulated (Baier and Katoen, 2008). More importantly, they can be translated to finite automata (Clarke *et al.*, 1999; Babiak *et al.*, 2012) which makes relevant the use of computer science tools (e.g., graph search and fix point algorithms). In this work, we use LTL formulas that can be expressed as deterministic automata. Even if the expressiveness power is weaker than the whole LTL language, it still encapsulates a wide variety of specifications and is often used in control synthesis (Alur and La Torre, 2004; Fainekos *et al.*, 2006).

Common approaches for control synthesis under high-level specifications involve the construction of an abstraction of the system. This abstraction is supposed to be used in place of a dynamical system and aims at reducing the overall complexity of the initial synthesis problem. If the system and the abstraction verify some behavioural relationship (see e.g., Tabuada, 2009), then the controller synthesized on the abstraction under the LTL specification can be refined into a controller such that the original system satisfies the same specification. Therefore, the abstraction try to provide a model that, at the same time, is simpler than the original system, and conserves enough information such that the controller synthesis problem is solvable for some given specifications. These 2 constraints are antagonist and the overall challenge of abstraction design rely in choosing wisely the partitioning of the state space and the system representation. To cite few of them, refinement methods (Nilsson and Ozay, 2014) and order minimal systems (Tabuada, 2009) investigate the partitioning of the state space. If a finer partitioning results in a more accurate model, it increases the complexity of the abstraction as it increases the state space representation. In Reißig (2011), the state of the dynamical system is extended with memories of the past discrete states, thus the more memories the system have, the more accurate is the abstraction. As added memories increase

the state space dimension and at the same time increase the accuracy of the abstraction, a sufficient partitioning of the state space that solve a given problem might actually lead to a smaller abstraction than direct abstraction methods (as in Tabuada, 2009). Therefore, the benefit of such methods must be evaluated for each models. Still in this idea of reducing the state space size, Zamani *et al.* (2014) evaluates abstractions of stochastic hybrid systems where only the input space is partitioned. Tarraf (2014) considered only the input-output relationship to built the abstraction thus avoiding any state space partitioning process that can include unnecessary partitioning depending on the system representation.

Although some methods result in deterministic abstractions (Kloetzer and Belta, 2008b; Boskos and Dimarogonas, 2015), other abstraction approaches induce some non-determinism (i.e., a control input can induce several successors from the same initial state, see e.g., Moor and Raisch, 2002; Nilsson and Ozay, 2014). Non-deterministic transitions in the abstraction can arise due to the state space partitioning or because of initial disturbances of the continuous system. Methods to lower these effects exist, such as partitioning the state space according to the system flow (see e.g., Lafferriere *et al.*, 2000; Tabuada, 2009) or designing local controllers that confine disturbances in each cell of the partition (see e.g., Kloetzer and Belta, 2008b). However, none of them can guaranty to get ride of the non-determinism (see e.g., Habets *et al.*, 2006), and, in such cases, the non-deterministic nature of the system needs to be taken into account. The abstraction is meant to be a simpler model of the dynamical system. The abstraction encapsulate less information than the original system. The main objective of this step is to remove as much information as possible to reduce the complexity of the problem. However, if the abstraction is too simple (understand there is not enough information), the high level planner might not be able to find a solution. There is interest at selecting the information that is the most meaningful. In this thesis, we have been studying an abstraction that is using memories of the last sequence of inputs applied to the system. This strategy has been profitable compared to raw partitioning of the dynamical system's state space.

While control synthesis problems with deterministic models can easily be solved through a reformulation into a model checking problem (Clarke *et al.*, 1999), the same approach cannot be applied with non-deterministic models (Kloetzer and Belta, 2008a). Fixed points methods have been widely used in correct-by-design control synthesis for non-deterministic models (see e.g., Cimatti *et al.*, 2003; Kloetzer and Belta, 2008a). They determine the set of valid controllers by trying to maximize some winning region which corresponds to the subspace of states where there exists a control strategy solving the synthesis problem. As it has been highlighted in Cimatti *et al.* (2003), such planner prevents any cycle (path of state-transition that starts and ends at the same state) from being part of a solution as they might keep the state away from the goal set forever. Other approaches use a *fairness assumption*: for any infinite run, every transition will be taken an infinite number of times (Cimatti *et al.*, 2003; Fu *et al.*, 2011). A direct consequence of this assumption is that every trajectory cannot stay indefinitely in a cyclic path,

and thus, these cycles can be part of the solution plan. This fairness assumption is justified in probabilistic models where the probability of any transition is not zero. For this reason, it has been used in action planning where any action can be assigned to a success probability, and an infinite number of attempts will almost surely result in a success. Such probabilistic approaches cannot always hold in the case of control synthesis of a dynamical system: any transition from one state of the abstraction to itself might correspond to an equilibrium point in the original continuous system, and any cycle in the abstraction might correspond to a stable orbit of the system. Therefore the global fairness property is not granted.

Models with local fairness property have been investigated in De Giacomo *et al.* (2010) where the fairness assumption is modelled as an LTL formula $\square\diamond\varphi \Rightarrow \square\diamond\psi$ which stands for: "by trying φ infinitely often, ψ will happen infinitely often". This formulation can then be integrated in a general reactivity framework. In this approach, and contrary to fixed point techniques where cycles are all eliminated, or all accepted (when fairness property stands), only non-blocking cycles identified by the model can be part of the solution plan. Other common approaches in control synthesis have been using the quotient transition system in order to deal with cycles in motion planning: this abstraction suppresses any self-transition of the model and only considers a fragment of the LTL formulas (Tůmová *et al.*, 2010; Pappas, 2003).

In this work, Chapter 1 introduce a novel abstraction method: the input extended abstraction method. For a given a dynamical system, the state is extended with memories of the past inputs. These memories are used to compute the reachable sets. The final abstraction is obtained by a partitioning of the resulting state space. As the abstraction is alternatingly simulated by the extended system, a controller solution of the problem for the abstraction is a solution for the extended system. As the resultant abstraction lead to non deterministic transition systems, Chapter 2 presents a new method for control synthesis of non-deterministic transition systems under LTL specifications. A non-deterministic augmented Finite Transition System (FTS) (firstly introduced for switching systems by Nilsson and Ozay, 2014) is introduced. This model extends the standard FTS structure with the knowledge of a *progress set* that identifies local control strategies that are terminating (trajectories cannot stay indefinitely in the subset of states using the associated control inputs). Then the product automaton of the model and the Büchi Automaton are defined and translated to a terminating planning problem, solved with a backward reachability algorithm to find a path through a terminating control strategy. Correctness and termination of the solutions are investigated. Chapter 3 ensure the alternating simulation relationship of the AFTS with a given dynamical system. These approaches are then validated with simulations, in Chapter 4, and experiments, in Chapter 5, involving multiple Unmanned Aerial Vehicles (UAV).

Chapter 1

Abstraction with input-extended state

Notation

Let X be a set. $X^\star$ denotes the set of all finite sequence with elements in X. The cartesian product between the set X and Y is $X \times Y$. $X|_Y$ denotes the projection of the set X on the set Y.

Introduction

Hereby we present an abstraction of a dynamical model (discrete or continuous time) obtained by extending the state with the knowledge of the finite sequence of inputs lastly applied to the system. These memories are used in order to replace the partial observation of the state (the chosen model is not fully observable). A proof that the system and the abstraction verify a relation (alternating simulation relation) can be derived and therefore any property verified by the controlled abstraction will be verified as well by the controlled system with the same controller.

Firstly, preliminaries about abstraction method and reachable sets are presented (in Section 1.1). Secondly, the construction of the abstraction is detailed and the alternating simulation relation between the original system and the final abstraction is proved (in Section 1.2). Thirdly, computation of reachable sets out of a sequence of inputs are derived for a simple model: the linear time invariant system (in Sections 1.3, 1.4 and 1.5). Finally, the admissible noise of the abstraction is investigated (in Section 1.6).

1.1 Preliminaries

System

Let the following definition of the system (investiguated in Tabuada (2009)):

Definition 1 (System). $S = (X, X_0, \mathcal{U}, \xrightarrow{S}, Y, H)$ *where:*
- X *is a set of states;*
- $X_0 \subset X$ *a set of initial states;*
- $\mathcal{U}$ *a discrete set of inputs;*
- $\xrightarrow{S} \subseteq X \times \mathcal{U} \times X$ *a transition relation ;*
- Y *a set of outputs;*
- $H : X \to Y$ *an output map.* ▲

This definition covers both the discrete and continuous time case of dynamical systems. For the continuous case, the time variable is included as part of the set of inputs. Let φ be the trajectory function of the dynamical system. A transition of the dynamical system from $\mathbf{x}$ at t to $\mathbf{x}' = \varphi(t_0, \mathbf{x}, \mathbf{u})$ at $t + t_0$ will be written:

$$\mathbf{x} \xrightarrow{S \quad t_0, \mathbf{u}} \mathbf{x}'.$$

For the discrete case, a transition correspond to 1 timestep. The set of states and of inputs do not have to verify any specific property, they can be infinite or finite sets. In our case, we will limit this work with a finite and discrete set of inputs. However, by discretizing the input set, continuous and unbounded set can also be used.

In this work, an equivalent notation for the transition relation $\xrightarrow{S \quad \mathbf{u}}$ of a system S is used, $Post^S_{\mathbf{u}}$ defined for $\mathbf{x} \in X$ and $\mathbf{u} \in \mathcal{U}$ by:

$$Post^S_{\mathbf{u}}(\mathbf{x}) = \{\mathbf{x}' \in X \mid \mathbf{x} \xrightarrow{S \quad \mathbf{u}} \mathbf{x}'\}. \tag{1.1}$$

$Post^S_{\mathbf{u}}(\mathbf{x})$ corresponds to the set of all the successors of the state $\mathbf{x}$ after applying $\mathbf{u}$.

To solve the controller synthesis problem, an abstraction of the system is used. Such abstraction must verify a relationship so that the controller synthesised with the abstraction is a valid solution as well for the system. The alternative simulation relation (introduced in Tabuada, 2009) can ensure such a property:

Definition 2 (Alternating simulation). *Let $\mathcal{S}_A$ and $\mathcal{S}_B$ 2 systems with $Y_a = Y_b$, $\mathcal{S}_A$ is alternatingly simulated by $\mathcal{S}_B$ (noted $\mathcal{S}_A \preceq_{\mathcal{AS}} \mathcal{S}_B$) if there exists a relation $R \subseteq X_a \times X_b$ that verifies:*

1. $\forall x_{a0} \in X_{a0}, \exists x_{b0} \in X_{b0}, (x_{a0}, x_{b0}) \in R$
2. $\forall (x_a, x_b) \in R, H_a(x_a) = H_b(x_b)$
3. $\forall (x_a, x_b) \in R, \forall u_a \in \mathcal{U}_a, \exists u_b \in \mathcal{U}_b$
   $\forall x'_b \in Post^{\mathcal{S}_B}_{u_b}(x_b), \exists x'_a \in Post^{\mathcal{S}_A}_{u_a}(x_a), (x'_a, x'_b) \in R$ ▲

Remark 1. *The alternating simulation relation is transitive (see Tabuada, 2009): for 3 systems $\mathcal{S}_A$,$\mathcal{S}_B$ and $\mathcal{S}_C$, if $\mathcal{S}_A \preceq_{\mathcal{AS}} \mathcal{S}_B$ and $\mathcal{S}_B \preceq_{\mathcal{AS}} \mathcal{S}_C$, then $\mathcal{S}_A \preceq_{\mathcal{AS}} \mathcal{S}_C$.*

If $\mathcal{S}_A$ is alternatingly simulated by $\mathcal{S}_B$, then every property verified by the controlled system $\mathcal{S}_A$ is verified by the controlled system $\mathcal{S}_B$ (with the same controller). This key property allows us to use the abstraction for the controller synthesis problem instead of the initial complex dynamical system (please refer to Tabuada (2009) for further details).

Reachable sets

In the next sections, computation of reachable sets for systems with input memories is considered. A formal definition of these reachable sets is defined using Definition 1 of the system.

For a system S, let $Reach^S(U) \subseteq X$ defined for a finite sequence $U = \{\mathbf{u}_0, \ldots, \mathbf{u}_{N-1}\} \in \mathcal{U}^\star$ of $N \in \mathbb{N}$ control actions by:

$$\begin{aligned} \mathbf{x} \in Reach^S(U) \Leftrightarrow & \exists \mathbf{x}_0 \in X_0, \forall 0 < i < N, \exists \mathbf{x}_i \in X, \\ & \mathbf{x}_0 \xrightarrow{S \quad \mathbf{u}_0} \mathbf{x}_1 \xrightarrow{S \quad \mathbf{u}_1} \ldots \xrightarrow{S \quad \mathbf{u}_{N-1}} \mathbf{x} \end{aligned} \tag{1.2}$$

$Reach^S(U)$ corresponds to all the reachable states applying the sequence of inputs U.

Definition 3. *For a finite sequence $U_n = \{\mathbf{v}_{1-n}, \ldots, \mathbf{v}_0\}$ of n controls in $\mathcal{U}$, let $Reach^{*S}(U_n) \subseteq X$ the set of all the states reached by a control sequence terminating with U_n:*

$$Reach^{*S}(U_n) = \bigcup_{\{\mathbf{u}_i\}_{i<\infty} \in \mathcal{U}^\star} Reach^S(\{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{v}_{1-n}, \ldots, \mathbf{v}_0\}) \tag{1.3}$$

▲

1.2 Abstraction

The abstraction is created following these steps (summarized in Figure 1.1):

- $S \to S'$: definition of the system with a state extended by a sequence of past inputs,
- $S' \to S_a$: the output set is projected to only partially observe the state, the unobserved states are replaced by computation of reachable sets out of the input memories,
- $S_a \to S'_a$: finally, the state space is partitioned.

As it will be underlined, no alternating simulation relation between S and S' exists, mainly for technical reasons that do not affect the rest of the abstraction computation. The 2 last transformations require the alternating simulation relation so that the controller synthesis on S'_a is valid as well for the system S'.

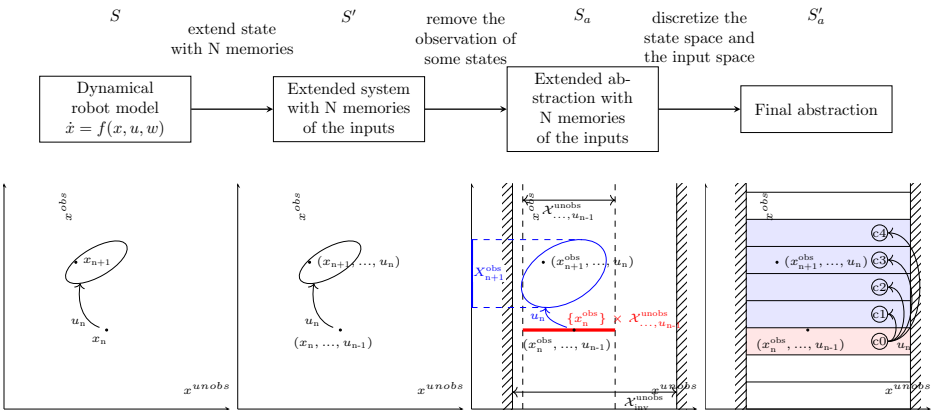

Figure 1.1: Steps to create the abstraction.

$S \to S'$

Let the system $\mathcal{S} = (X, X_0, \mathcal{U}, \xrightarrow{S}, Y, H)$. Let $\mathcal{S}'$, the same system than $\mathcal{S}$ with a state extended by memories of the last Δn_u control actions. $\mathcal{S}'$ is defined by $\mathcal{S}' = (X', X'_0, \mathcal{U}, \xrightarrow{S'}, Y', H')$ where:

- $X' = \{(\mathbf{x}, \mathbf{u}_{n-\Delta n_u}, ..., \mathbf{u}_{n-1}) \in X \times \mathcal{U}^{\Delta n_u} \mid \mathbf{x} \in Reach^{*S}(\mathbf{u}_{n-\Delta n_u}, ..., \mathbf{u}_{n-1})\}$ the set of states,
- $X'_0 = \{(\mathbf{x}_0, \mathbf{u}_{-\Delta n_u}, ..., \mathbf{u}_{-1}) \in X_0 \times \mathcal{U}^{\Delta n_u} \mid \mathbf{x}_0 \in Reach^{*S}(\mathbf{u}_{n-\Delta n_u}, ..., \mathbf{u}_{n-1})\}$ the set of initial states,
- $Y' = Y \times \mathcal{U}^{\Delta n_u}$ the set of outputs,
- H' the output map defined for all $(x, U_n) \in X'$ as $H'(x') = (H(x), U_n)$
- and the transition relation defined by:

$$\begin{aligned} &\forall(\mathbf{x}, \mathbf{u}_{n-\Delta n_u}, ..., \mathbf{u}_{n-1}) \in X', \forall \mathbf{x}' \in X, \forall \mathbf{u} \in \mathcal{U}, \\ &(\mathbf{x}, \mathbf{u}_{n-\Delta n_u}, ..., \mathbf{u}_{n-1}) \xrightarrow{S'\ \mathbf{u}} (\mathbf{x}', \mathbf{u}_{n+1-\Delta n_u}, ..., \mathbf{u}_{n-1}, \mathbf{u}) \\ &\qquad \Longleftrightarrow \mathbf{x} \xrightarrow{S\ \mathbf{u}} \mathbf{x}' \end{aligned} \tag{1.4}$$

As the system S and the system S' do not have the same output set, it is not possible to find an alternating simulation relation between the 2 systems. This has further implications: the controller synthesis will not be applicable for S but only for S' as the controller solution for S' cannot be composed with the system S. However, it is interesting to perform this first transformation from S to S' to underline the key difference between the 2 systems (apart from the state extension): every states not reachable with a finite sequence from $\mathcal{U}$ has been suppressed. And therefore, every initial state that is not reachable with the set of inputs $\mathcal{U}$ will not belong to the system S'.

$S' \rightarrow S_a$

In this part, the unobserved part of the system's state is removed. To counteract the loss of information, it is replaced with computation of reachable sets out of the sequence of inputs.

Lets assume that the state $\mathbf{x}$ of the system $\mathcal{S}$ can be decomposed in this way $\mathbf{x} = [\mathbf{x}^{o\top}, \mathbf{x}^{i\top}]^\top$ where $\mathbf{x}^o \in Y$ is observed and $\mathbf{x}^i$ is an unobserved (internal) state. Lets assume as well that $H(\mathbf{x}) = \mathbf{x}^o$. Let $\mathfrak{X}^i$ be the subspace of $\mathbf{x}^i$ and $\mathfrak{X}^o$ the one of $\mathbf{x}^o$. Let $X^o = X'|_{\mathfrak{X}^o} = Y$ the set of observed states, and $X_0^o = X_0'|_{\mathfrak{X}^o}$ the set of initial observed states. The state of the input extended state abstraction $\mathcal{S}_a$ is expressed by $\mathbf{x}_n^a = (\mathbf{x}_n^o, U_n)$ where $U_n = [\mathbf{u}_{n-\Delta n_u}, \ldots, \mathbf{u}_{n-1}] \in \mathcal{U}^{\Delta n_u}$.

Let the set

$$X^i(U_n) = Reach^{*S}(U_n)|_{\mathfrak{X}^i}.$$

$X^i(U_n)$ corresponds to all the unobserved states that can be reached by the system S' after applying a control sequence terminating with U_n. The knowledge of the state $\mathbf{x}^i$ can now be replaced by the set of all the possible states $X^i(U_n)$. Let the system $\mathcal{S}_a = (X_a, X_{a0}, \mathcal{U}, \xrightarrow{S_a}, Y_a, H_a)$ where:

- $X_a = X^o \times \mathcal{U}^{\Delta n_u}$ the set of states,
- $X_{a0} = X_0^o \times \mathcal{U}^{\Delta n_u}$ the set of initial states,
- $Y_a = Y' = X^o \times \mathcal{U}^{\Delta n_u}$ the set of outputs,
- H_a the output map defined for $\mathbf{x}^a = (\mathbf{x}^o, U_n) \in X_a$ by $H_a(\mathbf{x}^a) = (\mathbf{x}^o, U_n)$,
- and the transition relation is defined for $\mathbf{x}, \mathbf{x}' \in X'$, $\mathbf{u} \in \mathcal{U}$:

$$\mathbf{x} \xrightarrow{S' \quad \mathbf{u}} \mathbf{x}' \Longrightarrow \mathbf{x}|_{X_a} \xrightarrow{S_a \quad \mathbf{u}} \mathbf{x}'|_{X_a} \tag{1.5}$$

Property 1. *$\mathcal{S}_a$ is alternatingly simulated by $\mathcal{S}'$ ($S_a \preceq_{\mathcal{AS}} S'$).* ♦

Proof. Let R the relation defined by:

$$R = \{(\mathbf{x}', \mathbf{x}^a) \in X' \times X_a \mid H'(\mathbf{x}') = H_a(\mathbf{x}^a)\} \tag{1.6}$$

By definition of the systems S', $\mathcal{S}_a$ and of the relation R, conditions 1 and 2 of definition 2 are already verified.

Let $(\mathbf{x}', \mathbf{x}^a) \in R$, $\mathbf{u} \in U$ and $\mathbf{x}'_+ \in Post_{\mathbf{u}}^{S'}(\mathbf{x}')$. As $H'(\mathbf{x}') = H_a(\mathbf{x}^a)$, $\mathbf{x}'|_{\mathcal{U}^{\Delta n_u}} = \mathbf{x}^a|_{\mathcal{U}^{\Delta n_u}}$, to put this in words, the states have the same last Δn_u inputs, this sequence of inputs is denoted U_n. By definition of $X^i(U_n)$, $\mathbf{x}'|_{\mathfrak{X}^i} \in X^i(U_n)$, therefore $\mathbf{x}' \in \{\mathbf{x}'|_{\mathfrak{X}^o}\} \times X^i(U_n) \times U_n$ which implies that $\mathbf{x}'_+ \in Post_{\mathbf{u}}^{S'}(\{\mathbf{x}'|_{\mathfrak{X}^o}\} \times X^i(U_n) \times U_n)$. By taking $\mathbf{x}_+^a = H'(\mathbf{x}'_+) \in Post_{\mathbf{u}}^{S_a}(\mathbf{x}^a)$, thus $(\mathbf{x}'_+, \mathbf{x}_+^a) \in R$. □

$S_a \rightarrow S_a'$

In order to have a finite abstraction a partitioning of the state space is defined (as the input set is chosen discrete and finite, no partitioning is needed).

Let $P = \{P_1, \dots, P_{\mathcal{I}}\}$ define the observable state space partition, where P_i is a symbol associated with the subset $\mathcal{P}_i \subset X^o$ for $i \leq \mathcal{I} \in \mathbb{N}$ such that:

$$\begin{aligned} &\bigcup_{0<i\leq\mathcal{I}} \mathcal{P}_i = X^o \\ &\forall i, j \leq \mathcal{I}, 0 < i, j, \mathcal{P}_i \cap \mathcal{P}_j = \emptyset \end{aligned} \tag{1.7}$$

The final abstraction is then defined in this way. Let the system $S'_a = (X'_a, X'_{a0}, \mathcal{U}', \xrightarrow{S'_a}, Y'_a, H'_a)$ defined by:

- $X'_a = P \times \mathcal{U}^{\Delta n_u}$ the set of states,
- $X'_{a0} \subseteq X'_a$ the set of initial states defined for i,

$$(P_i, U_n) \in X'_{a0} \Leftrightarrow (\mathcal{P}_i, U_n) \cap X_{a0} \neq \emptyset, \tag{1.8}$$

- $\mathcal{U}' = \mathcal{U}$ the input set,
- $Y'_a = P \times \mathcal{U}^{\Delta n_u}$ the output set,
- H'_a the output map defined for $(P_i, U_n) \in X'_a$ by $H_a(P_i, U_n) = (P_i, U_n)$,
- and the transition relation is defined for $0 < i, j \leq \mathcal{I}$ and $u \in \mathcal{U}$:

$$\begin{aligned} &\exists \mathbf{x}^a \in \mathcal{P}_i \times \mathcal{U}^{\Delta n_u}, \mathbf{x}^a_+ \in \mathcal{P}_j \times \mathcal{U}^{\Delta n_u}, \\ &\mathbf{x}^a \xrightarrow{S_a \quad \mathbf{u}} \mathbf{x}^a_+ \Longrightarrow (P_i, \mathbf{x}^a|_{\mathcal{U}^{\Delta n_u}}) \xrightarrow{S'_a \quad \mathbf{u}} (P_j, \mathbf{x}^a_+|_{\mathcal{U}^{\Delta n_u}}). \end{aligned} \tag{1.9}$$

Remark 2. *The previous definition is using a small abuse of notation: the partition should have been done in the first place as the chosen output map now differs from the initial system. This is done mainly for simplification purposes, but does not change the results.*

Property 2. *S'_a is alternatingly simulated by S_a ($S'_a \preceq_{\mathcal{AS}} S_a$).* ♦

Proof. Let R the relation defined by:

$$\begin{aligned} &R = \{(\mathbf{x}'_a, \mathbf{x}_a) \in X'_a \times X_a \mid \\ &\exists 0 < i \leq \mathcal{I}, \mathbf{x}'_a|_P = P_i, \mathbf{x}_a|_P \in \mathcal{P}_i \text{ and } \mathbf{x}'_a|_{\mathcal{U}^{\Delta n_u}} = \mathbf{x}_a|_{\mathcal{U}^{\Delta n_u}}\}. \end{aligned} \tag{1.10}$$

$R|_{X^o \times P}$ associate to each state of X^o its corresponding symbol in P. By using the implication ($\Rightarrow$) of 1.8, we have:

$$\forall \mathbf{x}'_{a0} \in X_{a0'}, \exists \mathbf{x}_{a0} \in X_{a0}, (\mathbf{x}'_{a0}, \mathbf{x}_{a0}) \in R$$

(condition 1 in Definition 2). Condition 2 of Definition 2 is granted by 1.10 (please refer to Remark 2).

Let $(\mathbf{x}'_a, \mathbf{x}_a) \in R$, $\mathbf{u} \in U$ and $\mathbf{x}_{a+} \in Post^{S_a}_{\mathbf{u}}(\mathbf{x}_a)$. As the set X is fully partitioned (see 1.7), there exists $0 \leq i, j \leq \mathcal{I}$ such that $\mathbf{x}_a|_X \in \mathcal{P}_i$ and $\mathbf{x}_{a+}|_X \in \mathcal{P}_j$. Thanks to 1.9, we know that it exists $\mathbf{x}'_{a+} \in Post^{S'_a}_{\mathbf{u}}(\mathbf{x}'_a)$ where $\mathbf{x}'_{a+} = (P_j, \mathbf{x}_{a+}|_{\mathcal{U}^{\Delta n_u}})$. By definition of R, $(\mathbf{x}'_{a+}, \mathbf{x}_{a+}) \in R$, this proves the condition 3 of Definition 2. □

As the relation $\preceq_{\mathcal{AS}}$ is transitive (Remark 1), $S'_a \preceq_{\mathcal{AS}} S'$. This property is sufficient so that the controller synthesize problem's solution on S'_a is valid for S' as well.

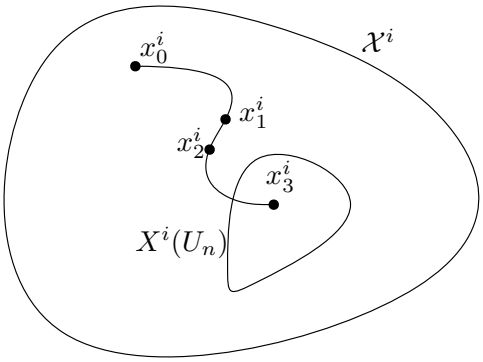

Figure 1.2: The image of the invariant set $\mathcal{X}^i$ after applying the control sequence U_n is $X^i(U_n)$

1.3 Dynamical systems

In the case of dynamical models (discrete or continuous time), the computation of the sets $X^i(U_n)$ is a reachability problem. It can be solved in 2 steps: find the smallest invariant $\mathcal{X}^i$ of $\mathcal{S}$ dynamics on $\mathfrak{X}^i$ and compute the image $X^i(U_n)$ of $\mathcal{X}^i$ after applying the control sequence U_n.

Interesting case scenarios happen when the knowledge about $X^i(U_n)$ is preferable to the actual observation of the state $\mathbf{x}^i$. If $X^i(U_n)$ is unbounded, the final abstraction might have an infinite number of successors. The impact on our ability to find a solution must be investigated for each systems, however this property is unlikely to be desirable. Therefore, this work put the focus on systems that have bounded invariant sets $\mathcal{X}^i$.

From now, discrete time linear time invariant systems will be used. They are chosen mainly for their ease of manipulation. However, as it has been shown until then, this can be applied to a broader class of systems.

1.4 Linear systems

Let the linear system S defined by:

Definition 4 (Linear Time Invariant system). *A Linear Time Invariant (LTI) system S is defined for m, $p \leq m$, $q \leq m$ in $\mathbb{N}$, the matrices $A \in \mathbb{R}^{m\times m}$, $B \in \mathbb{R}^{m\times p}$, $E \in \mathbb{R}^{m\times m}$, $C \in \mathbb{R}^{q\times m}$, the sets $\mathcal{U} \subseteq \mathbb{R}^p$, $\mathcal{W} \subseteq \mathbb{R}^m$, the initial state $\mathbf{x}_0 \in \mathbb{R}^m$. The dynamic of S is defined by:*

$$S : \begin{cases} \mathbf{x}_{n+1} = A\mathbf{x}_n + B\mathbf{u}_n + E\mathbf{w}_n \\ \mathbf{y}_n = C\mathbf{x}_n \end{cases} \tag{1.11}$$

where $\mathbf{x}_n$ is the state of the system at timestep $n \in \mathbb{N}$, $\mathbf{u}_n \in \mathcal{U}$ the input, $\mathbf{w}_n \in \mathcal{W}$ the noise, and $\mathbf{y}_n$ the output. ▲

The boundedness assumption of $X^i(U_n)$ of Section 1.3 can be implied by making further assumptions on the system S:

- the influence of the input on $\mathfrak{X}^i$ is bounded (i.e., $\{B\mathbf{u} \mid \mathbf{u} \in \mathcal{U}\}|_{\mathfrak{X}^i}$ is bounded),
- the influence of the noise on $\mathfrak{X}^i$ is bounded (i.e., $\{E\mathbf{w} \mid \mathbf{w} \in \mathcal{W}\}|_{\mathfrak{X}^i}$ is bounded),
- the system is asymptotically stable on the subspace $\mathfrak{X}^i$.

These assumptions are not necessary conditions in order to create the abstraction $\mathcal{S}_a$. However, they are sufficient and are chosen as matter of simplicity.

Finally, the following definition of the dynamical system S is adopted:

$$S : \begin{cases} \mathbf{x}_{n+1} = \begin{bmatrix} A_o & A_{ro} \\ 0 & A_r \end{bmatrix} \mathbf{x}_n + \begin{bmatrix} B_o & 0 \\ 0 & B_r \end{bmatrix} \mathbf{u}_n + \begin{bmatrix} E_o & 0 \\ 0 & E_r \end{bmatrix} \mathbf{w}_n \\ \mathbf{y}_n = \mathbf{x}_n^o \\ \mathbf{u}_n \in \mathcal{U} \\ \mathbf{w}_n \in \mathcal{W} \end{cases} \tag{1.12}$$

with,

$$\mathbf{x}_n = \begin{bmatrix} \mathbf{x}_n^o \\ \mathbf{x}_n^i \end{bmatrix}, \ \mathbf{u}_n = \begin{bmatrix} \mathbf{u}_n^o \\ \mathbf{u}_n^i \end{bmatrix}, \ \mathbf{w}_n = \begin{bmatrix} \mathbf{w}_n^o \\ \mathbf{w}_n^i \end{bmatrix}$$

where the unobserved system $\mathcal{S}^i = (A_r, B_r, E_r)$ is asymptotically stable (eigenvalues have strictly negative real part), input set $\mathcal{U}^i$ and noise sets $\mathcal{W}^i$ (respective projection of $\mathcal{U}$ and $\mathcal{W}$ on $\mathfrak{X}^i$) are bounded.

This system definition might seems restrictive as the unobserved part of the system is assumed to be independent of the observed part. However, this structure is met in many mechanical systems (mainly because of the Newton's second law of mechanic). This makes this model relevant especially in the case of robot motion planning.

1.5 Reachable set

The computation of dynamical system's reachable sets is a field in control theory on its own. So the reader might refer to the large literature about it for more precise methods in order to find invariants and reachable sets.

In this part, a method to compute the reachable sets for monotone systems is presented. For such systems, it is possible to obtain closed form (i.e., analytical) expression for the reachable set, this will be useful for further studies of the input extended state abstraction model.

As the computation of the reachable sets is not dependant on the entire system form according to the definition of the system 1.12, a simpler notation for this section is adopted. The notation of 4 is used with the assumptions of the unobserved system (boundedness of input and noise sets, system asymptotically stable). All these results will be then used for the unobserved system $\mathcal{S}^i$.

Monotone systems

In order to get a literal expression of the reachable sets, we have investigated a subset of the linear systems: the *linear monotone systems*.

A monotone system keeps a partial ordering relation $\preceq$ between 2 trajectories while time passes. The partial ordering relation is defined as followed: $\mathbf{x} \preceq \mathbf{y} \Leftrightarrow \mathbf{y} \in \mathcal{K} + \mathbf{x}$ where $\mathcal{K}$ is a cone. In our case we will work only with the orthant ordering, this means that $\mathbf{x} \preceq \mathbf{y}$ iff the relation $\leq$ is true element-wise between $\mathbf{x}$ and $\mathbf{y}$.

For a monotone system, 2 trajectories $\{\mathbf{x}_n\}_{n\in\mathbb{N}}$ and $\{\mathbf{y}_n\}_{n\in\mathbb{N}}$ with input sequence $\{\mathbf{u}_n\}_{n\in\mathbb{N}}$ and $\{\mathbf{v}_n\}_{n\in\mathbb{N}}$ respectively verify:

$$\mathbf{x}_n \preceq \mathbf{y}_n \wedge \mathbf{u}_n \preceq \mathbf{v}_n \Rightarrow \mathbf{x}_{n+1} \preceq \mathbf{y}_{n+1} \tag{1.13}$$

The partial ordering of the state space allows to over-approximate any bounded set X with 2 elements $\underline{\mathbf{x}}$ and $\overline{\mathbf{x}}$ so that $\forall \mathbf{x} \in X, \underline{\mathbf{x}} \preceq \mathbf{x} \wedge \mathbf{x} \preceq \overline{\mathbf{x}}$. A segment of $\mathbb{R}^n$ can be defined by $[\underline{\mathbf{x}}, \overline{\mathbf{x}}] = \{\mathbf{x} \in \mathbb{R}^n \mid \underline{\mathbf{x}} \preceq \mathbf{x} \wedge \mathbf{x} \preceq \overline{\mathbf{x}}\}$. More over, for any trajectory of a monotonic system starting from $\mathbf{x}_0 \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ with input sequence chosen in $[\underline{\mathbf{u}}, \overline{\mathbf{u}}]$:

$$\forall k \in \mathbb{N}, \mathbf{x}_k \in [\varphi(\underline{\mathbf{x}}, \underline{\mathbf{u}}, k), \varphi(\overline{\mathbf{x}}, \overline{\mathbf{u}}, k)] \tag{1.14}$$

where $\varphi(\mathbf{x}, \mathbf{u}, k)$ is the trajectory function of the system starting from $\mathbf{x}$ applying input $\mathbf{u}$ at timestep $k \in \mathbb{N}$. A linear monotone system is a linear system that verifies the monotonic property. The monotonic property simplifies the computation of reachable sets and will be used in computation for $\mathcal{S}^i$.

In order to obtain a literal form of the input extended state abstraction, the system S is assumed to be monotone, asymptotically stable, and with bounded input and noise set by a monotonic interval (if it is not the case, it is still possible to over approximate the bounds with a monotonic interval):

$$\mathcal{U} \subseteq [\underline{\mathbf{u}}, \overline{\mathbf{u}}], \mathcal{W} \subseteq [\underline{\mathbf{w}}, \overline{\mathbf{w}}]$$

As it has been mentioned in Section 1.2, the computation of the set $X(U_n)$ is done in 2 steps: the computation of the invariant and the computation of the image of this invariant after applying the sequence of inputs U_n. Taking advantage

of the monotonicity property of the system, every sets is be expressed as monotonic interval.

As the system S is asymptotically stable, the matrix $I - A$ have eigenvalues strictly greater than 0 and is invertible. Let:

$$\begin{aligned} \overline{\mathbf{x}} &= (I - A)^{-1}(B\overline{\mathbf{u}} + E\overline{\mathbf{w}}) \\ \underline{\mathbf{x}} &= (I - A)^{-1}(B\underline{\mathbf{u}} + E\underline{\mathbf{w}}) \end{aligned} \tag{1.15}$$

be the bounds of the smallest monotonic invariant set $\mathcal{X}$ of the system. So:

$$\mathcal{X} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \tag{1.16}$$

Let $\varphi(\mathbf{x}, U, W)$ the trajectory function of the system S starting from $\mathbf{x}_0$ applying the control sequence $U \in \mathcal{U}^k$ with the noise sequence $W \in \mathcal{W}^k$ and $k \in \mathbb{N}$. As the system S is monotone,

$$\forall \mathbf{x}_0 \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}], \forall U \in \mathcal{U}^k, \forall W \in \mathcal{W}^k, \varphi(\mathbf{x}_0, U, W) \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \tag{1.17}$$

If the initial state of the system is in $[\underline{\mathbf{x}}, \overline{\mathbf{x}}]$, then every trajectory stays in this set. For $U \in \mathcal{U}^k$, let $X(U)$ the subset of $\mathbb{R}^n$ defined by:

$$X(U) = [\varphi(\underline{\mathbf{x}}, U, \underline{\mathbf{w}}), \varphi(\overline{\mathbf{x}}, U, \overline{\mathbf{w}})] \tag{1.18}$$

$X(U)$ corresponds to the set of all the possible successors after applying the control sequence U on the system S. This results come from the monotonic property, and can be summarized in the following property:

$$\forall U \in \mathcal{U}^k, \forall \mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}], \forall W \in \mathcal{W}^k, \varphi(\mathbf{x}, U, W) \in X(U) \tag{1.19}$$

1.6 Error sensitivity

In the previous sections, the dynamical model S' was abstracted into S_a. In this part, the opposite problem is solved: we search for all the admissible models that can be alternatively simulated by S_a. More precisely, the search of admissible models is limited to systems with the same dynamic as the initial system S but with a different noise set.

As $\mathbf{x}^i$ is unobserved, a disturbance on the subspace $\mathfrak{X}^i$ does not have a direct effect on the output of the input extended state abstraction. However, as the dynamics on $\mathfrak{X}^i$ influence the one on $\mathfrak{X}^o$, it does have an impact in time. This part intends to show that the admissible noise set is actually bigger (in the sense of inclusion) than the noise set used in the model earlier.

In this part, the system is defined as in 1.12 enhanced with the monotonic property, the noise and input sets are expressed by 2 intervals:

$$\begin{aligned} \mathcal{U} &= [\underline{\mathbf{u}}, \overline{\mathbf{u}}] \\ \mathcal{W} &= [\underline{\mathbf{w}}, \overline{\mathbf{w}}] \end{aligned} \tag{1.20}$$

A sequence of noise is said to be *admissible* if it does not violate the transitions of the abstraction for any possible trajectory. In our case, this means that for any sequence of noise, the observation of the next state must belong to the interval of observations. This can be expressed in this way, along every trajectories, the observation must verify:

$$\mathbf{y}_{k+1} \in [\underline{\mathbf{y}}_{k+1}, \overline{\mathbf{y}}_{k+1}] \tag{1.21}$$

where

$$\begin{aligned}
\overline{\mathbf{y}}_{k+1} &= C\overline{\mathbf{x}}_{k+1} = CA \begin{bmatrix} \mathbf{x}_k^o \\ \overline{\mathbf{x}}^r \end{bmatrix} + CB\mathbf{u}_k + CE\overline{\mathbf{w}} \\
\underline{\mathbf{y}}_{k+1} &= C\underline{\mathbf{x}}_{k+1} = CA \begin{bmatrix} \mathbf{x}_k^o \\ \underline{\mathbf{x}}^r \end{bmatrix} + CB\mathbf{u}_k + CE\underline{\mathbf{w}} \\
\mathbf{y}_{k+1} &= C\mathbf{x}_{k+1} = CA \begin{bmatrix} \mathbf{x}_k^o \\ \mathbf{x}^r \end{bmatrix} + CB\mathbf{u}_k + CE\mathbf{w}_k
\end{aligned}$$

The inequalities 1.21 can be used to get conditions over the noise:

$$\begin{aligned}
\mathbf{y}_{k+1} \preceq \overline{\mathbf{y}}_{k+1} &\Leftrightarrow 0 \preceq \overline{\mathbf{y}}_{k+1} - \mathbf{y}_{k+1} \\
&\Leftrightarrow 0 \preceq CA \begin{bmatrix} 0 \\ \overline{\mathbf{x}}^r - \mathbf{x}_k^r \end{bmatrix} + CE(\overline{\mathbf{w}} - \mathbf{w}_k) \\
&\Leftrightarrow 0 \preceq A_{ro}(\overline{\mathbf{x}}^r - \mathbf{x}_k^r) + CE(\overline{\mathbf{w}} - \mathbf{w}_k)
\end{aligned} \tag{1.22}$$

Let $\mathbf{z}_k$ defined by:

$$\mathbf{z}_{k+1} = A_r\mathbf{z}_k + E_r\mathbf{w}_k + B_r\overline{\mathbf{u}} \tag{1.23}$$

Thanks to the monotone property and to $\mathbf{u}_k \preceq \overline{\mathbf{u}}$, for any trajectory if $\mathbf{x}_0^r \preceq \mathbf{z}_0$, then:

$$\mathbf{x}_k^r \preceq \mathbf{z}_k \tag{1.24}$$

Let $\mathbf{z}_0 = \mathbf{x}_0^r$ and $\mathbf{z}_k^* = \mathbf{z}_k - A_r^k\mathbf{x}_0^r + (A_r - I)^{-1}B_r\overline{\mathbf{u}}$, for all $k \in \mathbb{N}$:

$$\begin{aligned}
\mathbf{z}_{k+1}^* &= A_r\mathbf{z}_k^* + E_r\mathbf{w}_k \\
\mathbf{z}_0^* &= \mathbf{0},
\end{aligned} \tag{1.25}$$

by using the inequality 1.24 and equivalence 1.22, the following implication can be derived:

$$A_{ro}(\mathbf{z}_k^* + A_r^k\mathbf{x}_0^r - (A_r - I)^{-1}B_r\overline{\mathbf{u}}) + CE\mathbf{w}_k \preceq A_{ro}\overline{\mathbf{x}}^r + CE\overline{\mathbf{w}} \Rightarrow \mathbf{y}_{k+1} \preceq \overline{\mathbf{y}}_{k+1} \tag{1.26}$$

Finally:

$$A_{ro}\mathbf{z}_k^* + CE\mathbf{w}_k \preceq \overline{\sigma}_k \Rightarrow \mathbf{y}_{k+1} \preceq \overline{\mathbf{y}}_{k+1} \tag{1.27}$$

with $\overline{\sigma}_k = A_{ro}(\overline{\mathbf{x}}^r - A_r^k\mathbf{x}_0^r + (A_r - I)^{-1}B_r\overline{\mathbf{u}}) + CE\overline{\mathbf{w}}$.

The same implication can be derived with the lower bound of $\mathbf{y}_k$:

$$A_{ro}\mathbf{z}_k^* + CE\mathbf{w}_k \preceq \underline{\sigma}_k \Rightarrow \mathbf{y}_{k+1} \preceq \overline{\mathbf{y}}_{k+1} \tag{1.28}$$

with $\underline{\sigma}_k = A_{ro}(\underline{\mathbf{x}}^r - A_r^k \mathbf{x}_0^r + (A_r - I)^{-1} B_r \underline{\mathbf{u}}) + CE\underline{\mathbf{w}}$.

Please note that the term $A_r^k \mathbf{x}_0^r$ corresponds to a transient effect. We are only interested on the permanent behaviour of the admissible noise sequence, and as the system $\mathcal{S}^i$ is stable, the quantity $A_r^k \mathbf{x}_0^r$ goes to 0 as k goes to infinity. For the rest of the study, $\mathbf{x}_0^r$ is assumed to be 0 therefore: $\underline{\sigma}_k = \underline{\sigma}$ and $\overline{\sigma}_k = \overline{\sigma}$.

These 2 implications can be expressed with a filter inequality. Let the following filter $\mathcal{F}$:

$$\mathcal{F} : \begin{cases} \mathbf{z}_{k+1}^* = A_r \mathbf{z}_k^* + E_r \mathbf{w}_k \\ \mathbf{h}_k = A_{ro} \mathbf{z}_k + CE\mathbf{w}_k \end{cases} \tag{1.29}$$

Then implications 1.27 and 1.28 are satisfied if the output of the filter $\mathcal{F}$ is in $[\underline{\sigma}, \overline{\sigma}]$ for all the timesteps. The set $\mathcal{W}_a$ of noise sequences that verify:

$$\forall k \in \mathbb{N}, \mathbf{h}_k \in [\underline{\sigma}, \overline{\sigma}] \tag{1.30}$$

is a subset of the admissible noise sequences.

By linearity of the filter, if the sequences $\{\mathbf{w}_k^1\}_{k\in\mathbb{N}}$ and $\{\mathbf{w}_k^2\}_{k\in\mathbb{N}}$ are admissible, then the sequence defined by $\{a\mathbf{w}_k^1 + b\mathbf{w}_k^2\}_{k\in\mathbb{N}}$ with $a, b > 0$ and $a + b = 1$ is an admissible noise sequence. Lets now consider the bode function G of the filter $\mathcal{F}$ and the set $\mathcal{W}'$ defined by:

$$\mathcal{W}_G = \{\{G(w)cos(wk + \phi)\}_{k\in\mathbb{N}} \mid \phi \in [0, 2\pi], w \in [0, 2\pi]\}.$$

The convex polyhedra defined by the set $\mathcal{W}_G$ is a subset of the admissible noise set. This result gives us a practical criteria over the set of admissible noise sequences. It will also help us to understand some experiments.

Please note that the maximum constant noise remains unchanged. However, noises of higher frequencies can have greater magnitudes than the one modelled initially.

1.7 Conclusion

An abstraction with extended state is defined, input memories are used to replace the lost information of the unobserved part of the state. Experiments will show that this strategy might be interesting especially in case of systems with slow transient states (compared to the sampling time of the system).

The non observation of part of the state and the computation of the reachable sets in the linear monotone system case brought us to a bigger set of admissible noise sequences. This will be particularly helpful to explain why this abstraction was more robust to mismodellisation of the noise during experiments.

For most of the computation, the monotonic property has been used in order get analytical expressions. It resulted in models that are really restrictive (any linear systems with non real eigenvalue is not monotonic). However, it was suitable in the case of the quadricopter. Moreover, this properties has been chosen mainly to simplify the computations. Ones should be able to extend this work easily to more

complex models (by using numerical over approximation of the reachable set for example).

Chapter 2

Controller synthesis for non-deterministic augmented transition systems under LTL specifications

Notations

We denote by $\mathbb{N}$ the set of natural numbers and by $\mathbb{R}$ the set of real numbers. For the sets X, Y, we denote by $|X| \in \mathbb{N}$ the cardinality of X, 2^X its power set and X^ω the set of infinite words with elements chosen in X. For a word w of X^ω, $Inf(w) \subseteq X$ denotes the set of elements appearing infinitely often in w. For $Z \subseteq X \times Y$, $Z|_X \subseteq X$ denotes the projection of the set Z on the set X. Let $X \setminus Y$ be the set of elements of X not in Y. If $\mathbb{K}$ is $\mathbb{R}$ or $\mathbb{N}$, let $a, b \in \mathbb{K}$, $[a, b]_\mathbb{K}$ be the set $\{x \in \mathbb{K} \mid a \leq x \leq b\} \subset \mathbb{K}$.

Introduction

This chapter presents a new approach for control synthesis of non-deterministic transition systems under Linear Temporal Logic (LTL) specifications. The consideration of such systems is motivated by the non-determinism that can be introduced after the abstraction (using for example abstracting method presented in Chapter 1) of the dynamical systems into finite transition systems. More precisely, we consider transition systems enhanced with a *progress set* describing the fact that the system cannot stay indefinitely in some subset of states. The control synthesis problem is firstly translated into a terminating planning problem. Then, a backward reachability strategy searches for a path from the initial set to the goal set. At each iteration, subsets of states contained in the progress set are added to the path, thus ensuring the reachability to the goal set in finite time. If a solution to

the terminating problem is found, the obtained controller is translated back to the initial problem formulation. This approach will be used in the following chapters to find a control strategy for a given system and specifications.

We consider a non-deterministic augmented Finite Transition System (FTS) (firstly introduced for switching systems by Nilsson and Ozay, 2014). This model extends the standard FTS structure with the knowledge of a progress set that identifies local control strategies that are terminating (trajectories cannot stay indefinitely in the subset of states using the associated control inputs). We use this model as an abstraction for a dynamical system where the size of the progress set was large, and for this reason, the approach of De Giacomo *et al.* (2010), that efficiently deals with small progress sets, was not appropriate. We first introduce the model and the problem to be solved (Section 2.1). Then the product automaton of the model and the Büchi Automaton is defined and translated to a terminating planning problem (Section 2.2). A backward reachability algorithm is used (Section 2.2) to find a path through a terminating control strategy (Section 2.2) bringing the initial state deterministically in finite time to the goal set. Correctness and termination of the solutions are investigated.

2.1 Preliminaries

Definitions

In this work, we use a Finite Transition System (FTS) enhanced with a *progress set* adapted from Nilsson and Ozay (2014):

Definition 5 . *An Augmented Finite Transition System (AFTS) is a tuple defined by* $\mathcal{T} = \langle S, S_0, U_\mathcal{T}, \delta_\mathcal{T}, \mathcal{P}_\mathcal{T} \rangle$ *where:*

- S *is the set of states;*
- $S_0 \subseteq S$ *is the set of initial states;*
- $U_\mathcal{T}$ *is the input alphabet;*
- $\delta_\mathcal{T} : S \times U_\mathcal{T} \to 2^S$ *is the transition function;*
- $\mathcal{P}_\mathcal{T} \subseteq 2^{S \times U_\mathcal{T}}$ *is the progress set.*

An execution *of the AFTS* $\mathcal{T}$ *is an infinite sequence* $r \in (S \times U_\mathcal{T})^\omega$ *of state-control input pairs* $r = \{(s_k, u_k)\}_{k \in \mathbb{N}}$ *such that:*

- $s_0 \in S_0$*;*
- $\forall k \in \mathbb{N}, s_{k+1} \in \delta_\mathcal{T}(s_k, u_k)$*;*
- $\forall m \in \mathcal{P}_\mathcal{T}, Inf(r) \not\subseteq m.$ ▲

The progress set $\mathcal{P}_\mathcal{T}$ identifies local control strategies that are terminating, i.e., any execution reaching an element $m \in \mathcal{P}_\mathcal{T}$ is guaranteed to exit m in finite time. Note that the condition $\forall m \in \mathcal{P}_\mathcal{T}, Inf(r) \not\subseteq m$ does not forbid any execution r to visit infinitely often an element $m \in \mathcal{P}_\mathcal{T}$ as long as the execution also leaves m infinitely often (i.e., if $Inf(r) \cap m \neq \emptyset$ and $Inf(r) \setminus m \neq \emptyset$ then $Inf(r) \not\subseteq m$). In the case of switching systems, Nilsson and Ozay (2014) introduced the progress

set where each element is chosen in $2^S \times U_{\mathcal{T}}$. An element identifies a set of states $p_g \subseteq S$ (a progress group) and a control mode $u_g \in U_{\mathcal{T}}$, and any trajectory starting in the progress group p_g using the control mode u_g leaves p_g in finite time. In this paper, we do not consider control modes that are related to switching systems (control policy valid for a group of states) but a control strategy (state-control action pairs).

We say that $\rho \in S^\omega$ is a *run* of $\mathcal{T}$ if there exists an execution r of $\mathcal{T}$ such that $\rho = r|_{S^\omega}$. In the present work, we assume that the AFTS $\mathcal{T}$ is *well-formed* meaning that we have: $S_0 \neq \emptyset$; for every reachable state $s \in S$ there exists at least one control action leading to another state, i.e., $\exists u \in U_{\mathcal{T}}, |\delta_{\mathcal{T}}(s,u)| \geq 1$; and all elements of $\mathcal{P}_{\mathcal{T}}$ have an outgoing transition, i.e., $\forall m \in \mathcal{P}_{\mathcal{T}}, \exists (s,u) \in m, \delta_{\mathcal{T}}(s,u) \not\subseteq m|_S$.

The reader is referred to the dedicated literature about formal methods (such as Baier and Katoen, 2008) for a formal definition of the LTL language. In summary, an LTL formula over a set S is defined inductively by:

$$\varphi ::= \top \mid a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathsf{U} \varphi_2$$

where the logical operators $\top$ (*true*), $\neg$ (*negation*), $\wedge$ (*conjunction*), the temporal operators $\bigcirc$ (*next*) and U (*until*) are combined with elements $a \in S$ and LTL formulas φ_1 and φ_2. The useful operators $\vee$ (*disjunction*), $\Rightarrow$ (*implication*), $\Box$ (*always*) and $\Diamond$ (*eventually*) can be derived from the previous ones. These LTL formulas provide a user-friendly language to express specifications, for example "avoid area d, visit area c, and avoid area a until you reach area b" can be expressed by the following formula:

$$\varphi = (\Box\neg d) \wedge (\Diamond c) \wedge ((\neg a)\mathsf{U} b).$$

Every LTL formula over an input alphabet S can be translated into a Büchi Automaton (BA) (Clarke *et al.*, 1999):

Definition 6 . *A Büchi Automaton is a tuple* $\mathcal{A}_\varphi = \langle Q_\varphi, Q_{\varphi 0}, S, \delta_\varphi, \mathcal{F}_\varphi \rangle$ *where:*

- Q_φ *is the finite set of states;*
- $Q_{\varphi 0} \subseteq Q_\varphi$ *is the set of initial states;*
- S *is the input alphabet;*
- $\delta_\varphi : Q_\varphi \times S \to 2^{Q_\varphi}$ *is the transition function;*
- $\mathcal{F}_\varphi$ *is the set of accepting states.* ▲

If $Q_{\varphi 0}$ is a singleton and $|\delta_\varphi(q, a)| \in \{0, 1\}$ for all $a \in S$ and all $q \in Q_\varphi$, then $\mathcal{A}_\varphi$ is called Deterministic Büchi Automaton (DBA). It is called a Non-deterministic Büchi Automaton (NBA) otherwise. In this paper, we use LTL specifications that can be represented with DBA. Such DBA can be generated using a NBA of a given LTL formula and applying determinization processes (Alur and La Torre, 2004; Babiak *et al.*, 2012). However, this is not always achievable, and only a fragment of LTL formulas are translatable into DBA (see Theorem 4.50 in Baier and Katoen, 2008, pp. 190). Nonetheless, this fragment allows to express a wide range of specifications (Fainekos *et al.*, 2006). For an infinite input word $w = w_1 w_2 w_3 ... \in S^\omega$, there exists a set of infinite state trajectories in $\mathcal{A}_\varphi$ produced by w that we denote as $\mathcal{R}_\varphi(w) \subseteq {Q_\varphi}^\omega$.

Definition 7 . *The input word* $w \in S^\omega$ *is said to be* accepted *by* $\mathcal{A}_\varphi$ *if there exists* $r \in \mathcal{R}_\varphi(w)$ *such that* $Inf(r) \cap \mathcal{F}_\varphi \neq \emptyset$. ▲

Note that the input alphabet S of $\mathcal{A}_\varphi$ is chosen as the set of states of $\mathcal{T}$ so that a run of $\mathcal{T}$ produces an input word for the BA $\mathcal{A}_\varphi$. A run r of $\mathcal{T}$ can then be accepted or not by $\mathcal{A}_\varphi$.

Problem statement

The problem considered in this paper is formulated as follows:

Problem 1. *Given a non-deterministic and well-formed AFTS* $\mathcal{T} = \langle S, S_0, U_\mathcal{T}, \delta_\mathcal{T}, \mathcal{P}_\mathcal{T} \rangle$ *and a DBA* $\mathcal{A}_\varphi = \langle Q_\varphi, Q_{\varphi 0}, S, \delta_\varphi, \mathcal{F}_\varphi \rangle$*, find a control strategy such that all runs of the controlled system are accepted by* $\mathcal{A}_\varphi$*.*

To satisfy the acceptance condition of $\mathcal{A}_\varphi$ (see Definition 7), the control strategy of the system $\mathcal{T}$ should produce trajectories in $\mathcal{A}_\varphi$ that reach the acceptance set $\mathcal{F}_\varphi$ in finite time. Therefore, in the case of an FTS (AFTS with no progress set, i.e., $\mathcal{P}_\mathcal{T} = \emptyset$), every control strategy creating cyclic trajectories in $\mathcal{A}_\varphi$ outside of the acceptance set $\mathcal{F}_\varphi$ cannot be a solution of the problem. For the AFTS $\mathcal{T}$, if these corresponding cyclic executions are in one element of the progress set $\mathcal{P}_\mathcal{T}$, then these cycles are escaped in finite time, and the controller might be a solution of Problem 1.

2.2 Solution

Problem 1 is solved in several steps. First, we create the product automaton of $\mathcal{T}$ and $\mathcal{A}_\varphi$, and translate it into an equivalent formulation as a terminating planning problem (Section 2.2). Then, we identify control strategies on subsets of the state space which can reach, deterministically and in finite time, given sets of states (Section 2.2). Finally, we use these local terminating controllers to find a non-blocking control strategy starting the search from the goal set until the initial set is found (Section 2.2).

Terminating formulation

We first define the product automaton of a DBA and an AFTS:

Definition 8 . *The product automaton $\mathcal{A}_p$ of the DBA $\mathcal{A}_\varphi$ and of the AFTS $\mathcal{T}$ is defined by $\mathcal{A}_p = \mathcal{T} \otimes \mathcal{A}_\varphi = \langle Q_p, Q_{p0}, U_p, \delta_p, \mathcal{F}_p, \mathcal{P}_p \rangle$ where:*

- *$Q_p = S \times Q_\varphi$ is the set of states;*
- *$Q_{p0} = S_0 \times Q_{\varphi 0}$ is the set of initial states;*
- *$U_p = U_\mathcal{T}$ is the input set;*
- *$\delta_p : Q_p \times U_p \to 2^{Q_p}$ is the transition function defined by $(s', q') \in \delta_p((s, q), u)$ iff $s' \in \delta_\mathcal{T}(s, u)$, $q' \in \delta_\varphi(q, s)$;*
- *$\mathcal{F}_p = S \times \mathcal{F}_\varphi$ is the acceptance set;*
- *$\mathcal{P}_p \subseteq 2^{Q_p \times U_p}$ is the progress set defined for $m \in 2^{Q_p \times U_p}$ by $m \in \mathcal{P}_p \Leftrightarrow m|_{S \times U_\mathcal{T}} \in \mathcal{P}_\mathcal{T}$.* ▲

Note that the product automaton $\mathcal{A}_p$ is a Büchi Automaton augmented with the progress set of the AFTS $\mathcal{T}$. We say that an execution $r \in (Q_p \times U_p)^\omega$ of the product automaton is *valid* if $\forall m \in \mathcal{P}_p, Inf(r) \not\subseteq m$ (this condition is inherited from Definition 5 of the AFTS). A valid execution r of the product automaton is said to be *accepted* if the corresponding state trajectory $\rho = r|_{{Q_p}^\omega}$ satisfies $Inf(\rho) \cap \mathcal{F}_p \neq \emptyset$. An accepted execution of $\mathcal{A}_p$ thus corresponds to a valid execution of $\mathcal{T}$ that is accepted by $\mathcal{A}_\varphi$.

A controller of $\mathcal{A}_p$ is a map $\pi : Q_p \to U_p$ that associates to each state $p \in Q_p$ an *available control action* $u \in U_p(p)$ where $U_p(p) = \{u \in U_p \mid |\delta_p(p, u)| \geq 1\}$. The controller is said to be *closed* if every state reachable from the initial set Q_{p0} using the controller π is associated with a control action, and it is *terminating* if all the reachable states can reach the acceptance set $\mathcal{F}_p$ in finite time. Finding a controller for Problem 1 consists in finding a closed and terminating controller over $\mathcal{A}_p$. In what follows, this control problem is translated into a terminating formulation.

Property 3 (Theorem 4 in Patrizi *et al.*, 2013)**.** *Let a non-deterministic product automaton $\mathcal{A}_p = \langle Q_p, Q_{p0}, U_p, \delta_p, \mathcal{F}_p, \mathcal{P}_p \rangle$, where $\mathcal{F}_p = \{g_1, ..., g_n\}$, $|\mathcal{F}_p| = n$, and $Q_p = \{p_1, ..., p_m, g_1, ..., g_n\}$, $|Q_p| = n + m$. Let $G_t^0 = \{g_1^0, .., g_n^0\}$ be a duplicate definition of $\mathcal{F}_p$ (with different names for the corresponding states so that $G_t^0 \cap \mathcal{F}_p =$*

Figure 2.1: Construction of $\mathcal{A}_t$ from $\mathcal{A}_p$. Similar node shape represents equivalent state sets. Similar line style represents equivalent transition function between sets.

$\emptyset$). *The* terminating formulation *of* $\mathcal{A}_p$ *is denoted as* $\mathcal{A}_t = \langle Q_t, Q_{t0}, U_t, \delta_t, G_t, \mathcal{P}_t \rangle$ *such that:*

- $Q_t = Q_p \cup G_t^0$ *is the set of states;*
- $Q_{t0} = Q_{p0} \cup G_t^0$ *is the set of initial states;*
- $U_t = U_p$ *is an input alphabet;*
- $\delta_t : Q_t \times U_t \to 2^{Q_t}$ *is the transition function defined by* $p' \in \delta_p(p,u) \Leftrightarrow p' \in \delta_t(\sigma(p), u)$*;*
- $G_t = \mathcal{F}_p$ *is the goal set;*
- $\mathcal{P}_t = \mathcal{P}_p$ *is the progress set.*

where $\sigma : Q_p \to Q_t$ *is a function mapping* $Q_p \setminus \mathcal{F}_p$ *to* $Q_t \setminus (G_t \cup G_t^0)$ *and* $\mathcal{F}_p$ *to* G_t^0*, namely:*

$$\begin{cases} \sigma(p) = p & \text{if } p \in Q_p \setminus \mathcal{F}_p \\ \sigma(g_j) = g_j^0 & \text{for } j \in [\![1, n]\!] \end{cases}$$

$\mathcal{A}_t$ *is said to be* well-formed *if* $\mathcal{A}_p$ *is well-formed. A controller* $\pi_t : Q_t \to U_t$ *of* $\mathcal{A}_t$ *is said to be closed (resp. terminating) if the associated controller* $\pi_p = \pi_t \circ \sigma$ *of* $\mathcal{A}_p$ *is closed (resp. terminating).*

The terminating formulation $\mathcal{A}_t$ of $\mathcal{A}_p$ is built by mapping all transitions from $\mathcal{F}_p$ to transitions from a duplicate set G_t^0 of $\mathcal{F}_p$ and by adding G_t^0 to the initial set Q_{t0}. Figure 2.1 illustrates the construction of the transition function δ_t and of the set of states Q_t of $\mathcal{A}_t$. $\mathcal{A}_p$ is supposed to be well-formed (see Problem 1), thus $\mathcal{A}_t$ is also well-formed which means that for every state $q \in Q_t \setminus G_t$ there exists an available control action $u \in U_t(q)$ where $U_t(q) = \{u \in U_t \mid |\delta_t(q,u)| \geq 1\}$. Let an *execution* of $\mathcal{A}_t$ be a finite or infinite sequence $r = \{(q_k, u_k)\}_{k<I}$ of length $I \in \mathbb{N} \cup \{\infty\}$ with elements chosen in $Q_t \times U_t$ such that:

- $q_0 \in Q_{t0}$;
- $\forall k < I, q_{k+1} \in \delta_t(q_k, u_k)$;
- $\forall m \in \mathcal{P}_t, Inf(r) \not\subseteq m$.

For a closed controller π_t, a π_t-execution corresponds to a sequence of $\mathcal{A}_t$ such that $r = \{(q_k, \pi_t(q_k))\}_{k \leq I}$ with $I \in \mathbb{N} \cup \{\infty\}$. From Proposition 3, we can now equivalently solve Problem 1 by trying to find a closed and terminating controller π_t for the system $\mathcal{A}_t$ such that all the π_t-executions of $\mathcal{A}_t$ reach the goal set G_t in finite time.

Terminating modules search

Any control strategy in $\mathcal{A}_t$ creating cyclic execution might produce runs that loop ad infinitum, hence keeping the state away from G_t forever. If, however, each of these potential cycles is included into elements of the progress set $\mathcal{P}_t$, then none of them can block the system indefinitely. Thus, a control strategy π_t in $\mathcal{A}_t$ can ensure the termination property if the following statement is true: π_t bring the state from terminating modules (defined in the sequel) to other terminating modules strictly closer to the goal set G_t. This section details the search of a terminating module that reaches, deterministically and in finite time, a given set of states.

We first introduce some definitions. Let a *module* $m \subseteq Q_t \times U_t$ of $\mathcal{A}_t$ be a set of state-control input pairs such that $\forall (q,u) \in m, u \in U_t(q)$ and every state is associated to a unique control action ($\forall q \in m|_{Q_t}, |m \cap (\{q\} \times U_t)| = 1$). We call $\mathscr{M}$ the set of modules of $\mathcal{A}_t$. Then, let $Post : \mathscr{M} \to 2^{Q_t}$ be the function that associates to a module its successor states, i.e., for $m \in \mathscr{M}$, $Post(m) = \bigcup_{(q,u)\in m} \delta_t(q,u)$, let $\overline{Post} : \mathscr{M} \to 2^{Q_t}$ be defined as the set of the *outgoing* successor states of a given module, i.e., for $m \in \mathscr{M}$, $\overline{Post}(m) = Post(m) \setminus m|_{Q_t}$.

Definition 9 . *$m \in \mathscr{M}$ is a* terminating module *of $\mathcal{A}_t$ if all executions of $\mathcal{A}'_t = \langle Q_t, m|_{Q_t}, U_t, \delta_t, G_t, \mathcal{P}_t \rangle$ exit m in finite time, i.e., for every execution $r = \{(q_k, u_k)\}_{k<I}$, $I \in \mathbb{N} \cup \{\infty\}$ of $\mathcal{A}'_t$, there exists $i < I$ such that $q_i \notin m|_{Q_t}$, and $\forall k < i, (q_k, u_k) \in m$.* ▲

Note that each module $m \in \mathscr{M}$ in the progress set $\mathcal{P}_t$ or such that $Post(m) \cap m|_{Q_t} = \emptyset$ (i.e., without self-transitions) is terminating.

For $g \subseteq Q_t$, let PrecedingSingletonModules$(g) \subseteq \mathscr{M}$ be defined as the set of singleton modules with states chosen in $Q_t \setminus g$ that have one or more successors in g, namely:

$$\{(q,u)\} \in \textsc{PrecedingSingletonModules}(g) \Leftrightarrow \begin{cases} q \in Q_t \setminus g \\ \delta_t(q,u) \cap g \neq \emptyset \end{cases}$$

and, for a module $m \in \mathscr{M}$, let ExpandModule$(g,m) \subseteq \mathscr{M}$ be defined by:

$$m' \in \textsc{ExpandModule}(g,m) \Leftrightarrow \begin{cases} m'|_{Q_t} = m|_{Q_t} \cup (Post(m) \setminus g) \\ m \subset m' \end{cases}$$

ExpandModule(g,m) is the set of possible expansions of m with states chosen in $Q_t \setminus g$.

Algorithm 1 details the search of terminating modules that go, deterministically and in finite time, to a given subset g of Q_t. The search is implemented as a Breadth-First Search (BFS) over the set of possible terminating modules. For a set of winning regions $W \subseteq 2^{Q_t}$, $w \in W$ corresponds to a set of states where a winning

Algorithm 1: Breadth-First Search (BFS) of the terminating modules

Function PRECEDINGMODULEBFS($\mathcal{A}_t, g, W$)

Data: $\mathcal{A}_t$: the terminating problem formulation,
$g \subseteq Q_t$: a set of states to be reached,
$W \subseteq 2^{Q_t}$: a set of already visited sets of states.

Result: m if a valid module is found, $\emptyset$ otherwise

$\mathtt{M} \leftarrow \{\text{PRECEDINGSINGLETONMODULES}(g)\}$; `// modules to visit`

$\mathtt{M}_v \leftarrow \emptyset$; `// visited modules`

repeat

$m \leftarrow \arg\min_{x \in \mathtt{M}}(|x|)$; `// smallest cardinality module`

$\mathtt{M} \leftarrow \mathtt{M} \setminus \{m\}$; `// Remove` m `from M`

$\mathtt{M}_v \leftarrow \mathtt{M}_v \cup \{m\}$; `// mark` m `as visited`

if $\overline{Post}(m) \subseteq g$ *and* $(m \in \mathcal{P}_t$ *or* $Post(m) \subseteq g)$ *and* $g \cup m|_Q \notin W$ **then**

return m ; `// terminating module found`

else

$\mathtt{M} \leftarrow \mathtt{M} \cup (\text{EXPANDMODULE}(g, m) \setminus \mathtt{M}_v)$; `// Add the new modules to the set of modules to visit`

end

until $|\mathtt{M}| = 0$;

return $\emptyset$; `// no module found`

control strategy (i.e., terminating and reaching the goal set G_t) has already been found (the next section will more specifically state the definition of W). The set $\mathtt{M}$ of modules to visit is initialized (line 1) with the preceding singleton modules of g (PRECEDINGSINGLETONMODULES(g)). At each iteration, one of the smallest modules m of $\mathtt{M}$ (smallest in terms of cardinality; lines 1, 1 and 1) is removed from $\mathtt{M}$ and added to the set of visited modules $\mathtt{M}_v$. All possible successor modules of m are obtained with EXPANDMODULE(g, m), and those that have not been visited added to the set $\mathtt{M}$ of modules to visit (line 1). Finally, the module m is returned if it verifies the following properties (line 1):

- $\overline{Post}(m) \subseteq g$: all the successor states of m are either going to the set of states $m|_{Q_t}$ or to the set g,
- $m \in \mathcal{P}_t$ or $Post(m) \subseteq g$: m is a terminating module,
- $g \cup m|_Q \notin W$: no control strategy has been found yet.

The first property ensures that any trajectory beginning from module m either stays in m or reaches g in finite time. The second one ensures that the trajectory actually leaves $m|_{Q_t}$. The first two properties combined together show that modules returned by Algorithm 1 deterministically reach the set of states g and are terminating. The last property ensures that the new set of reachable states has not been considered yet. If there is no module found, $\emptyset$ is returned (line 1).

Backward Reachability Algorithm

Algorithm 2: Backward reachability algorithm (BRA)

Function BACKWARDSEARCH($\mathcal{A}_t$)
Data: $\mathcal{A}_t$: the terminating problem formulation
Result: K if a solution is found, $Fail$ otherwise
$L \leftarrow \{K_G\}$; // plans to visit
$L_v \leftarrow \emptyset$; // visited plans
repeat
$K \leftarrow \arg\max_{x \in L}(|x|)$; // biggest cardinality plan
if $Q_{t0} \subseteq \tilde{K}|_{Q_t}$ **then** // initial set found
return K ; // return valid plan
$L_v \leftarrow L_v \cup \{K\}$; // Add K to L_v
$W \leftarrow \{\tilde{l}|_{Q_t} \mid l \in L \cup L_v\}$; // set of visited sets of states
$m \leftarrow$ PRECEDINGMODULEBFS($\mathcal{A}_t, \tilde{K}|_{Q_t}, W$) ;
if $m \neq \emptyset$ **then**
$L \leftarrow L \cup \{K \cup \{m\}\}$; // add the new plan to L
else
$L \leftarrow L \setminus \{K\}$; // remove the plan from L
end
until $|L| = 0$;
return $Fail$; // no valid plan found

We are now able to find terminating modules that bring any execution of $\mathcal{A}_t$ deterministically and in finite time to a given set of states through Algorithm 1. The same algorithm can then be used to iteratively expand a winning region where a terminating and closed control strategy exists. Algorithm 2 implements this as a Depth-First Search (DFS) using a backward reachability strategy: preceding terminating modules are searched and added to a terminating controller starting from the goal set G_t until all initial states in Q_{t0} are found.

Let a *plan* $K \subseteq \mathscr{M} \cup \{m_0\}$ be a set of terminating modules of $\mathcal{A}_t$ (see Definition 9) and of:

$$m_0 = \{(g, \emptyset) \mid g \in G_t\} \tag{2.1}$$

a fake module comprised of each goal state associated to none of the control actions. Let $\tilde{K} = \bigcup_{k \in K} k$ be the corresponding set of state-control action pairs. L corresponds to the set of plans to visit and is initialized (line 2) with $K_G = \{m_0\}$. At each step, in the set L of plans to visit, the highest cardinality plan K is selected (line 2). If the initial set belongs to $\tilde{K}|_{Q_t}$ (the set of states of plan K), then plan K is returned (line 2). Otherwise, a terminating module m preceding $\tilde{K}|_{Q_t}$ (line 2) is returned by Algorithm 1, and the new plan $K \cup \{m\}$ is added to the list L of plans to visit (line 2). Note that the module m is chosen to produce a plan with

a set of reachable states that have not been created yet ($m|_{Q_t} \cup \tilde{K}|_{Q_t} \notin W$; line 1 of Algorithm 1). If there is no valid module found (lines 2 and 2), then plan K is removed from L (line 2). When the set L of plans to visit is empty (line 2), that means that among all the possible configurations of reachable states comprised of terminating modules, none of them could find a solution and a $Fail$ flag is returned (line 2).

Theorem 1. *Let K be a valid plan returned by Algorithm 2 ($K \neq Fail$) and π_K be the controller associated with K defined by $\pi_K(q) = u$ for all $(q, u) \in \tilde{K} = \bigcup_{k \in K} k$. The controller π_K is closed and terminating.*

Proof. Let K be a plan found from Algorithm 2, let $m_0, m_1, ...$ be the modules that compose the plan K indexed from the first one added (m_0, defined in (2.1)) to the last one ($m_{|K|-1}$), $K = \{m_0, m_1, ...\}$ with $m_0|_{Q_t} = G_t$. By construction of K (see Algorithm 1 line 1, and Algorithm 2 lines 2 and 2), for all $i \in [\![1, |K| - 1]\!]$:

$$\overline{Post}(m_i) \in \bigcup_{0 \leq j < i} m_j|_{Q_t} \tag{2.2}$$

$$m_i|_{Q_t} \cap \Big(\bigcup_{0 \leq j < i} m_j|_{Q_t} \Big) = \emptyset \tag{2.3}$$

Using (2.3) recursively, for all $i, i' \in [\![0, |K| - 1]\!]$ s.t. $i \neq i'$, we have:

$$m_i|_{Q_t} \cap m_{i'}|_{Q_t} = \emptyset. \tag{2.4}$$

As each module $m_i \in \mathscr{M}$, for $i \in [\![1, |K| - 1]\!]$, associates to each state of $m_i|_{Q_t}$ a unique control action (see the definition of $\mathscr{M}$ in Section 2.2), and thanks to (2.4), every state in $(\tilde{K} \setminus m_0)|_{Q_t}$ is associated to one and only one control input. Thus, the controller π_K is correctly defined in Theorem 1.

As for all $(q, u) \in \tilde{K} \setminus m_0$, $\delta_t(q, u) \subseteq \tilde{K}|_{Q_t}$ from (2.2) and as $Q_{t0} \subseteq \tilde{K}|_{Q_t}$ (Algorithm 2, line 2), the controller π_K is closed. As π_K is closed, there exists a π_K-execution $r = \{(q_k, u_k)\}_{k<I}$ with $I \in \mathbb{N} \cup \{\infty\}$ such that $(q_i, u_i) \in \tilde{K}$ for every $i < I$. From (2.4), for every $q \in \tilde{K}|_{Q_t}$ there exists a unique $j \in [\![0, |K| - 1]\!]$ such that $q \in m_j|_{Q_t}$. We define the unique sequence of indexes $J = j_0 j_1 ...$ such that $q_i \in m_{j_i}$ for every $i < I$. As $\mathcal{A}_t$ is well-formed (Problem 1 and Property 3), there is always an available transition as long as m_0 is not reached. Therefore, we choose r to be infinite ($I = \infty$) iff m_0 is not reached. By definition of r and as $(q_0, u_0) \in m_{j_0}$, we have $q_1 \in \delta_t(q_0, u_0) \subseteq Post(m_{j_0})$. Then from (2.2), all the successors of m_{j_0} are in the modules of smaller indexes, consequently $j_1 \leq j_0$. This argument can be used for all $i < I$, meaning that J is a decreasing sequence. Moreover J is lower bounded by 0, therefore it is converging to a value j^* such that $0 \leq j^* < |K|$. Assuming $j^* > 0$ implies that $Inf(r) \subseteq m_{j^*}$. However, this invalidates the fact that m_{j^*} is a terminating module (line 1 of Algorithm 1). The only solution is $j^* = 0$ and $m_{j^*}|_{Q_t} = m_0|_{Q_t} = G_t$, which means that there exists $i \in \mathbb{N}$ such that $q_i \in G_t$, i.e., the goal set is reached in finite time, and therefore, the controller π_K is terminating. □

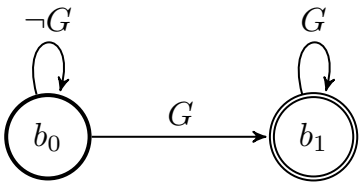

Figure 2.2: Deterministic Büchi Automaton that represents the formula $\varphi = (\neg G)\mathsf{U}(\Box G)$. The acceptance set is $\mathcal{F}_\varphi = \{b_1\}$ and the initial set is $Q_{\varphi_0} = \{b_0\}$.

As the controller π_K is closed and terminating, the controller (after mapping with σ - previously defined by $\forall p \in Q_p \setminus \mathcal{F}_p, \sigma(q) = q$ and $\forall j \in [\![1, n]\!], \sigma(g_j) = g_j^0$) is a valid solution of Problem 1.

Corollary 1. *Let K be a valid plan returned by Algorithm 2 (i.e., $K \neq Fail$) and π_K be the controller associated to K defined by $\pi_K(q) = u$ for all $(q, u) \in \tilde{K}$. The controller $\pi_p = \pi_K \circ \sigma$ is a solution of Problem 1.*

Remark 3. *The termination of Algorithm 1 and 2 is ensured by two facts: the browsed state space is finite and the use of a visited set ($\mathtt{M}_v$ and L_v in Algorithm 1 and 2) avoids any element to be visited twice.*

Illustration

Figure 2.3 illustrates how Algorithm 2 finds a solution plan. An agent starting in Q_0 needs to verify $\varphi = (\neg G)\mathsf{U}(\Box G)$, namely "reach and stay in G", using four given control actions (↖, ↗, ↙ and ↘). Note that φ can be represented by a DBA (e.g., see `ltl3ba`, Babiak *et al.*, 2012, Figure 2.2 shows a DBA representing φ). The progress set $\mathcal{P}_\mathcal{T}$ is partially described by $\{a, b, c, d\} \subset \mathcal{P}_\mathcal{T}$. The transition system $\mathcal{T}$ is not explicitly defined and the reader is instead referred to Figure 2.3.

Due to the special form of the LTL formula φ, the terminating formulation $\mathcal{A}_t$ can be defined equivalently than in Section 2.2 by: the goal set $G_t = G$, the set $G_t^0 = \{g_1^0, ..., g_4^0\}$ duplicate of $G = \{g_1, ..., g_4\}$, the set of states $Q_t = S \cup G_t^0$, the initial set $Q_{t0} = Q_0 \cup G_t^0$, the progress set $\mathcal{P}_t = \mathcal{P}_\mathcal{T}$, and the transition function defined by: $s' \in \delta_\mathcal{T}(s, u) \Leftrightarrow s' \in \delta_t(\sigma(s), u)$ (where $\sigma : S \to Q_t$ is the mapping function similarly defined as in Proposition 3, i.e., $\sigma(s) = s$ iff $s \in S \setminus G_t$ and $\sigma(g_i) = g_i^0$ for $i \in [\![1, 4]\!]$). Note that in Figure 2.3, G_t and G_t^0 are superimposed however, we remind that $G_t \cap G_t^0 = \emptyset$.

Algorithm 2 is initialized with a plan $K_0 = \{m_0\}$ where m_0 is the fake goal module introduced in Section 2.2 (i.e, $m_0|_{Q_t} = G_t$, green box in Figure 2.3). At the first iteration, the plan K_0 is selected from the set $L = \{K_0\}$ of plans to visit. Algorithm 1 searches for a terminating module preceding the set of states $G_t = \tilde{K}_0|_{Q_t}$: e.g., $m_1 = \mathtt{g}_1^0 = \{(g_1^0, \nearrow)\}$ is returned as all successors $Post(m_1)$ of m_1 are in $\tilde{K}_0|_{Q_t}$. The new plan $K_1 = K_0 \cup \{m_1\}$ is added to L. At the next iteration

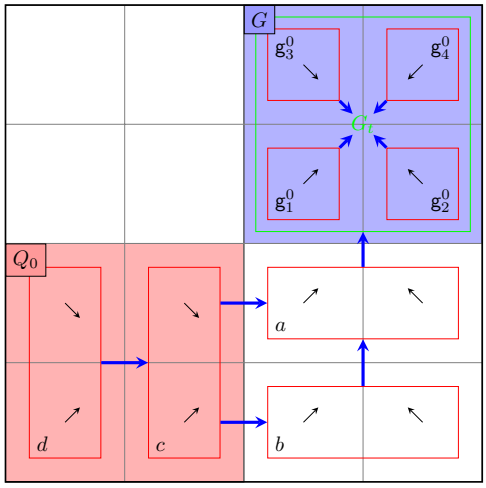

Figure 2.3: Illustration of Algorithm 1 and 2. The agent goes from set Q_0 (in red) to reach and stay in set G (in blue) following the control actions (black arrows). Four control actions are available (↖, ↗, ↙ and ↘) and each of them points in the direction of the 3 successors of the considered state (square cell). Each red box corresponds to a terminating module and the green box corresponds to the goal set G_t of $\mathcal{A}_t$.

of Algorithm 2 (and at each iteration in this example), the highest cardinality plan of L is the last one added to L. For the same reasons than for module m_1 with plan K_0, modules $m_2 = \mathtt{g}_2^0$, $m_3 = \mathtt{g}_3^0$ and $m_4 = \mathtt{g}_4^0$ are successively added to plans $K_2 = K_1 \cup \{m_2\}$, $K_3 = K_2 \cup \{m_3\}$ and $K_4 = K_3 \cup \{m_4\}$. At the 5^{th} iteration of Algorithm 2, K_4 is the biggest plan of $L = \{K_0, ..., K_4\}$ and Algorithm 1 searches for a terminating module going to $\tilde{K}_4|_{Q_t} = G_t \cup G_t^0$ deterministically and in finite time; there is no singleton terminating module, the smallest valid ones are composed of at least 2 elements, e.g., a is returned as it belongs to $\mathcal{P}_t$ and goes to $G_t \subset \tilde{K}_4|_{Q_t}$. $m_5 = a$ is added to $K_5 = K_4 \cup \{m_5\}$.

In the next iteration, the terminating module b is returned as it goes to $a|_{Q_t} \subset \tilde{K}_5|_{Q_t}$ and as $b \in \mathcal{P}_t$, then $m_6 = b$ is added to the plan $K_6 = K_5 \cup \{m_6\}$. $m_7 = c$ is added to $K_7 = K_6 \cup \{m_7\}$ as it goes deterministically to $a|_{Q_t} \cup b|_{Q_t} \subset \tilde{K}_5|_{Q_t}$ and as $c \in \mathcal{P}_{\mathcal{T}}$. Note that the transition between the modules is not deterministic ($\overline{Post}(c) \subseteq (a|_{Q_t} \cup b|_{Q_t})$). Finally, $m_8 = d$ is added to $K_8 = K_7 \cup \{m_8\}$, and as $d|_{Q_t} \cup c|_{Q_t} = Q_{t0} \subset \tilde{K}_8|_{Q_t}$, K_8 is a plan that brings all the trajectories starting in Q_{t0} to the goal set in finite time. Algorithm 2 returns the solution plan $K = K_8$. Every trajectory of the system $\mathcal{T}$ controlled by $\pi_p = \pi_K \circ \sigma$ (Theorem 1 and Corollary 1) verifies the specification φ.

2.3 Conclusion

We proposed a solution to the control synthesis problem of a non-deterministic system under Linear Temporal Logic specifications that can be represented by Deterministic Büchi Automata. The considered system is modelled as a finite transition system enhanced with a progress set. Elements of the progress set identify local control strategies that are terminating (the state will escape a given set of states in finite time). In the next chapters, experiments with multiple UAVs will show that this approach is relevant in case of real world applications where the non-determinism of the system cannot be narrowed after the abstraction of it.

Chapter 3

Escape time property

Notations

Let $[\![a, b]\!] = \{n \in \mathbb{N} \mid a \leq n \leq b\}$, $[a, b] = \{x \in \mathbb{R} \mid a \leq n \leq b\}$, $\mathbb{R}_+$ the positive subset of $\mathbb{R}$, $\mathbb{R}_+^\star = \mathbb{R}_+ \setminus \{0\}$. Let $\|X\|$ the maximum distance of 2 points in the set X ($\|X\| = \sup_{a,b \in X} \|a - b\|$). For a set X, $X^{\mathbb{N}}$ is an infinite sequence in X, for x in $X^{\mathbb{N}}$ and k in $\mathbb{N}$, x_k is the k^{th} element of the sequence x. The Minkowski sum of $A, B \in X$ noted $A + B$ is the set $\{a + b \mid a \in A, b \in B\}$. For $r > 0$, let $\mathcal{B}(r) = \{x \in \mathbb{R}^n \mid \|x\| \leq r\}$ the closed ball of radius r. For a, b 2 vectors of the euclidian space build on $\mathbb{R}^n$ for $n \in \mathbb{N}$, we denote by $a \cdot b$ the scalar product of a and b.

Introduction

This chapter details the construction of the AFTS's progress set introduced in Definition 5. In Chapter 1, the behavioural relationship is implied by the alternating simulation relationship between the system and the FTS; such method can be used in order to build the FTS structure of the AFTS. Then, the progress must be built such that the behavioural relationship between the dynamical system and the AFTS is ensured.

In practice, the progress set is not constructed entirely, instead we define an *escape time function* that map a control strategy on a subset of states to the maximum number of timesteps that any trajectory needs to leave the set of states. For a set of state-control actions, if the escape time is finite, then this element belongs to the progress set. This escape time function is deduced from the dynamical system. In this part, we present one possible escape time function for monotone systems.

First (Section 3.1), a practical criteria over control actions of a single integrator system is defined to check if the state can escape a subset of the state space in a finite time. Then (Section 3.2) this criteria is extended to the single integrator model with disturbances, then in the more general case of monotone dynamical

system (Section 3.3) . Finally (Section 3.4), the notion of escaping time is involved to extend the previous results for wider types of specifications: Metric Interval Temporal Logic (MITL).

3.1 Escaping proof

In this section, the first integrator system is used. A condition is levered over the control sequence to check if the system can escape a bounded subset of the state space in a finite time.

In the previous sections, the element of the progress set are set of discrete state-control action tuples. Hereby, we will not specifically associate a state with a control action, but rather demonstrate the escape time property for a set of states and a set of available controls actions. Then, each states can be associated with any control action available.

Let the system S:

$$S : \begin{cases} x_{k+1} & = x_k + u_k \\ x_k & \in \mathbb{R}^N \\ x_0 & \in X_0 \\ u_k & \in \mathcal{U} \end{cases} \tag{3.1}$$

with $k \in \mathbb{N}$ the timestep, $N \in \mathbb{N}$ the state space dimension, X_0 a bounded subset of $\mathbb{R}^N$, and $\mathcal{U} = \{v_1, ..., v_n\}$ a finite set of control inputs in $\mathbb{R}^N$, let n the cardinality of $\mathcal{U}$. Let $\mathcal{X}_S \subset (X \times \mathcal{U})^{\mathbb{N}}$ the set of all the possible trajectories of S.

Definition 10. *A trajectory $\{x_k, u_k\}_{k\in\mathbb{N}}$ of a system S escapes X_0 in finite time if there exists k in $\mathbb{N}$ so that $x_k \in X_0$ and $x_{k+1} \notin X_0$.* ▲

Property 4. *For a trajectory $(x, u) \in \mathcal{X}_S$*

$$\exists \alpha \in \mathbb{R}^N, \forall k \in \mathbb{N}, u_k \cdot \alpha > 0 \Rightarrow (x, u) \text{ escape } X_0 \text{ in a finite time}$$

Proof. Let (x, u) a trajectory of the system S. Lets assume that there exists α in $\mathbb{R}^N$ such that $\forall k \in \mathbb{N}, u_k \cdot \alpha > 0$. By computing the scalar product of the sequence $\{x_i\}_{i\in\mathbb{N}}$, with α and thanks to the strict inequality of the scalar product, the following paragraphs will show that the state diverges and thus escape the bounded set X_0.

Let $k \in \mathbb{N}$, by Definition 10, $x_k = x_0 + \sum_{i=0}^{k} u_i$. By taking the scalar product of the previous expression and α, $x_k \cdot \alpha = x_0 \cdot \alpha + \sum_{i=0}^{k} u_i \cdot \alpha$.

Let $E = \{u_k \cdot \alpha \mid k \in \mathbb{N}\}$, by assumption E is bounded by 0, so $\epsilon = \inf E$ exists. As $\mathcal{U}$ is closed and bounded, E is closed and bounded as well, so ϵ is a minimum and belong to E, so $\epsilon > 0$.

The scalar product of the state sequence is expressed by $x_k \cdot \alpha \geq x_0 \cdot \alpha + k\,\epsilon$, so $\lim_{k\to+\infty} x_k \cdot \alpha = +\infty$ this implies that $\lim_{k\to+\infty} \|x_k\| = +\infty$ (this implication can be proved by a *reductio ad absurdum*). As X_0 is bounded, there exists a timestep k_0 for which $x_{k_0} \notin X_0$. □

For $V \subset \mathbb{R}^N$:

$$F(V) = \left\{\alpha \in \mathbb{R}^N | \forall v \in V, v \cdot \alpha > 0\right\}$$

The property 4 can be expressed in this way: for a trajectory $(x, u) \in \mathcal{X}_S$ with a control sequence chosen in $\mathcal{U}_t \subseteq \mathcal{U}$, if $F(\mathcal{U}_t) \neq \emptyset$ then the state x_k escape X_0 in a finite time.

3.2 Extension to convex input sets

In this part, disturbances are added to the definition of S. Let the system S':

$$S' : \begin{cases} x_{k+1} & = x_k + u_k + w_k \\ x_k & \in \mathbb{R}^N \\ x_0 & \in X_0 \\ u_k & \in \mathcal{U} \\ w_k & \in \mathcal{W} \end{cases} \tag{3.2}$$

with $k \in \mathbb{N}$ the timestep, $N \in \mathbb{N}$ the state space dimension, X_0 a bounded subset of $\mathbb{R}^N$, and $\mathcal{U} = \{v_1, ..., v_n\}$ a finite set of controls inputs in $\mathbb{R}^N$, n the cardinality of $\mathcal{U}$ and $\mathcal{W} \subset \mathbb{R}^N$ the set of noise.

Contrary to the previous section, the difference between 2 consecutive states does not belong to a finite subset of controls inputs but in the set $\mathcal{U} + \mathcal{W}$. If $\mathcal{U} + \mathcal{W}$ is a closed and bounded set then the results of Property 4 can be applied to get equivalent results.

Property 5. *For a trajectory* $(x, u) \in \mathcal{X}_{S'}$

$$\begin{aligned} &\exists \alpha \in \mathbb{R}^N, \forall k \in \mathbb{N}, \forall v \in \{u_k\} + \mathcal{W}, v \cdot \alpha > 0 \\ &\qquad \Rightarrow (x, u) \textit{ escape } X_0 \textit{ in a finite time} \end{aligned} \tag{3.3}$$

Proof. Same proof as for Property 4. As by assumption the set $\mathcal{U} + \mathcal{W}$ is bounded and closed, it is possible to find an ϵ that minor all the scalar products of the elements of $\{u_k\} + \mathcal{W}$ for k in $\mathbb{N}$. □

In the case of the single integrator with a finite input set, the escaping property is implementable in practice because only a finite number of control inputs have to be checked. For the single integrator with disturbances, the property needs to be checked over an infinite amount of values: this is not algorithmically doable. In order to reduce the complexity of the problem, a convex bounded polyhedra set for $\mathcal{W}$ is used. The following paragraph discuss how Property 5 can be checked in an algorithm.

Any points in a convex bounded polyhedra set can be expressed as a linear combination of the vertices: if $\mathcal{P}$ is a convex bounded polyhedra with a set $\mathcal{V}$ of vertices, then for any x in $\mathcal{P}$, there exists $n_\mathcal{V} = |\mathcal{V}|$ real numbers $t_i \in [0, 1]$ for $i \in [\![1, n_\mathcal{V}]\!]$ that verifies $\sum_{i=1}^{n_\mathcal{V}} t_i = 1$, so that $x = \sum_{i=1}^{n_\mathcal{V}} t_i \, v_i$.

Moreover, the convexity property conserves the scalar product inequality: for u, v, α in $\mathbb{R}^N$, if $u \cdot \alpha > 0$ and $v \cdot \alpha > 0$, then for all t in $[0,1]$, $(t\,u + (1-t)\,v) \cdot \alpha > 0$. In the case of the convex bounded polyhedra $\mathcal{P}$ with a set $\mathcal{V}$ of vertices: if for all $v \in \mathcal{V}$ the scalar product with α is positive, then scalar products of all points inside $\mathcal{P}$ are positive. This reduces the size of the problem to a finite number of points.

The previous paragraph showed that $\forall \alpha \in \mathbb{R}^N, \alpha \in F(\mathcal{V}) \Rightarrow \alpha \in F(\mathcal{P})$, so $F(\mathcal{V}) \subseteq F(\mathcal{P})$. As $\mathcal{V} \subset \mathcal{P}$, $F(\mathcal{P}) \subseteq F(\mathcal{V})$. Therefore, $F(\mathcal{V}) = F(\mathcal{P})$.

Lets consider a sequence $\{u_k\}_{k\in\mathbb{N}}$ of control input chosen in $\mathcal{U}_t \subseteq \mathcal{U}$. As $\mathcal{U}_t + \mathcal{W} = \bigcup_{u\in\mathcal{U}_t} (\{u\} + \mathcal{W})$ where each $\{u\} + \mathcal{W}$ is a convex bounded polyhedra, we would like to compute the set F of an union of two sets. Let A and B two subsets of $\mathbb{R}^N$, and $\alpha \in \mathbb{R}^N$:

$$\begin{aligned} \alpha \in F(A) \cap F(B) &\Leftrightarrow \forall x \in A, x \cdot \alpha > 0 \ \textit{and}\ \forall x \in B, x \cdot \alpha > 0 \\ &\Leftrightarrow \forall x \in A \cup B, x \cdot \alpha > 0 \\ &\Leftrightarrow \alpha \in F(A \cup B) \end{aligned} \tag{3.4}$$

so $F(A \cup B) = F(A) \cap F(B)$.

Let $\mathcal{V}_i^t$ the set of all the vertices of the convex bounded polyhedra $\{v_i\} + \mathcal{W}$, thanks to the definition of convex sets and from the fact that a convex bounded polyhedra can be expressed as a finite sum of its vertices, we have:

$$\begin{aligned} F(\mathcal{U}_t + \mathcal{W}) &= F(\bigcup_{i\in[\![1,n^t]\!]} \{\{v_i\} + \mathcal{W}\}) \\ &= \bigcap_{i\in[\![1,n^t]\!]} F(\{v_i\} + \mathcal{W}) \qquad \text{(thanks to equivalence 3.4)} \\ &= \bigcap_{i\in[\![1,n^t]\!]} F(\mathcal{V}_i^t) \\ &= F(\bigcup_{i\in[\![1,n^t]\!]} \mathcal{V}_i^t) \end{aligned}$$

The set $\bigcup_{i\in[\![1,n^t]\!]} \mathcal{V}_i^t$ is a finite subset of $\mathbb{R}^N$ therefore, it can be used as a practical criteria.

It follows that for any trajectories (x, u) of the system S' with controls chosen in $\mathcal{U}_t \subseteq \mathcal{U}$, $F(\bigcup_{i\in[\![1,n^t]\!]} \mathcal{V}_i^t) \neq \emptyset$ implies that the state x_k will escape X_0 in a finite time.

3.3 Monotone system extention

The escape time function has been introduced for the single integrator model with disturbances. This is however very restrictive and this section extends the previous results for a wider class of systems by building an alternating simulation relation

between them and the single integrator system. Let the monotone system defined by:

$$S : \begin{cases} \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k) \\ \mathbf{x}_0 \in X_0 \\ \mathbf{u}_k \in \mathcal{U} \\ \mathbf{w}_k \in \mathcal{W} \end{cases} \tag{3.5}$$

with $\mathcal{U}$ finite and bounded, and $\mathcal{W}$ bounded and closed.

By creating the new system defined by:

$$S' : \begin{cases} \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{u}'_k + \mathbf{w}'_k \\ \mathbf{x}_0 \in X_0 \\ \mathbf{u}'_k \in \mathcal{U}' \\ \mathbf{w}'_k \in \mathcal{W}' \end{cases} \tag{3.6}$$

with

$$\mathcal{U}' = \{g(\mathbf{x}, \mathbf{u}) \mid \mathbf{u} \in \mathcal{U}, \mathbf{x} \in X_0\},$$
$$g(\mathbf{x}, \mathbf{u}) = \frac{1}{2}(f(\mathbf{x}, \mathbf{u}, \underline{\mathbf{w}}) + f(\mathbf{x}, \mathbf{u}, \overline{\mathbf{w}})) - \mathbf{x}$$

and

$$\mathcal{W}' = \bigcup_{\mathbf{x} \in X_0, \mathbf{u} \in \mathcal{U}} [\frac{1}{2}(f(\mathbf{x}, \mathbf{u}, \underline{\mathbf{w}}) - f(\mathbf{x}, \mathbf{u}, \overline{\mathbf{w}})), \frac{1}{2}(-f(\mathbf{x}, \mathbf{u}, \underline{\mathbf{w}}) + f(\mathbf{x}, \mathbf{u}, \overline{\mathbf{w}}))]. \tag{3.7}$$

X_0, $\mathcal{U}$ and $\mathcal{W}$ are bounded by assumption, if $f(X_0, \mathcal{U}, \mathcal{W})$ is assumed to be a bounded set as well, then $\mathcal{U}'$ and $\mathcal{W}'$ are bounded. $\mathcal{U}$ is finite, so $\mathcal{U}'$ is finite as well. $\mathcal{W}$ and X_0 are closed, the function $h(\mathbf{x}, \mathbf{u}) = f(\mathbf{x}, \mathbf{u}, \mathbf{w}) - g(\mathbf{x}, \mathbf{u})$ is continuously differentiable on $X_0 \times \mathcal{U}$, so the set $\mathcal{W}'$ is closed as well.

Property 6. *S' is alternatingly simulated by S ($S' \preceq_{\mathcal{AS}} S$).* ♦

Proof. Let define R by:

$$R = \{(\mathbf{x}', \mathbf{x}) \in X_0 \mid \mathbf{x}' = \mathbf{x}\}$$

By definition of R (using the identity output map) and by definition of S and S', condition 1 and 2 of Definition 2 are fulfilled. Moreover, for $(\mathbf{x}', \mathbf{x}) \in R$, $\mathbf{u} \in \mathcal{U}$, let $\mathbf{u}' = g(\mathbf{x}', \mathbf{u})$. For $\mathbf{x}_+ \in f(\mathbf{x}, \mathbf{u}, \mathcal{W})$, as f is monotone, we have:

$$\mathbf{x}_+ \in [f(\mathbf{x}, \mathbf{u}, \underline{\mathbf{w}}), f(\mathbf{x}, \mathbf{u}, \overline{\mathbf{w}})]. \tag{3.8}$$

If the noise for S' is chosen in:

$$[\underline{\mathbf{w}}', \overline{\mathbf{w}}'] = [\frac{1}{2}(f(\mathbf{x}, \mathbf{u}, \underline{\mathbf{w}}) - f(\mathbf{x}, \mathbf{u}, \overline{\mathbf{w}})), \frac{1}{2}(-f(\mathbf{x}, \mathbf{u}, \underline{\mathbf{w}}) + f(\mathbf{x}, \mathbf{u}, \overline{\mathbf{w}}))]$$

(one can verify that $[\underline{\mathbf{w}}', \overline{\mathbf{w}}'] \subset \mathcal{W}'$ thanks to 3.7), then, the set of successors of $\mathbf{x}'$ applying $\mathbf{u}'$ is $[\underline{\mathbf{x}}'_+, \overline{\mathbf{x}}'_+]$ where:

$$\underline{\mathbf{x}}'_+ = \mathbf{x}' + \mathbf{u}' + \underline{\mathbf{w}}' = f(\mathbf{x}, \mathbf{u}, \underline{\mathbf{w}})$$
$$\overline{\mathbf{x}}'_+ = \mathbf{x}' + \mathbf{u}' + \overline{\mathbf{w}}' = f(\mathbf{x}, \mathbf{u}, \overline{\mathbf{w}}).$$

We can choose $\mathbf{x}'_+ \in [\underline{\mathbf{x}}', \overline{\mathbf{x}}']$ such that $(\mathbf{x}'_+, \mathbf{x}_+) \in R$ (see 3.8) and thus prove the condition 3 of Definition 2. □

As the system S' is alternatingly simulated by the system S, all the behaviours of S are included in the behaviours of S'. This means that if the system S' escape X_0 in finite time, then the system S escape the set X_0 in finite time as well. Therefore, the previous results can be applied to the system S using the system S'.

3.4 Timing information

In this section, the previous results are used to show the existence of an escaping time upper bound. This information can be used to generate controllers with time specifications (such as in MITL frameworks).

The escaping property asserts that if each control action of a trajectory is oriented in a preferential direction, then the system always escape the initial subset X_0. The next paragraph shows the computation of the minimum norm of time average control sequences.

Property 7. *Let $\mathcal{U}_t \subseteq \mathcal{U}$ a subset of controls so that $F(\mathcal{U}_t) \neq \emptyset$. Let $\mathcal{U}_t = \{v_1^t, ..., v_{n_t}^t\}$ with n_t in $\mathbb{N}$. There exists $u_m = \arg\min_{u \in C_{\mathcal{U}_t}} \|u\|$ so that $u_m \neq 0$ with*

$$C_{\mathcal{U}_t} = \left\{ \sum_{i=1}^{n_t} k_i \, v_i^t \mid \forall (k_1, ..., k_{n_t}) \in [0,1]^{n_t}, \sum_{i=1}^{n_t} k_i = 1, \right\} \tag{3.9}$$

Every trajectory with control inputs chosen in $\mathcal{U}_t$ of system S starting in X_0 escapes X_0 in less than $T_{escape} = \frac{\|X_0\|}{\|u_m\|}$ timesteps.

Proof. Firstly, the existence of T_{escape} is showed. Secondly, the proof that every trajectory of the system S escapes X_0 in less than T_{escape} timesteps is given.

$C_{\mathcal{U}_t}$ corresponds to the set of time average of the admissible control sequences with value chosen in $\mathcal{U}_t$. The smallest element of $C_{\mathcal{U}_t}$ (in term of norm) represents the minimum average displacement of the state that is reachable with an infinite sequence of control. This control give information about the slowest escape from X_0 of our system for a given set of controls.

Let $\mathcal{K} = \{k \in [0,1]^n \mid \sum_{i=1}^n k_i = 1\}$, $\mathcal{K}$ is a closed and bounded set. Let $f_{\mathcal{U}_t}(\mathbf{k}) = \sum_{i=1}^n k_i \, v_i^t$ for $\mathbf{k} \in \mathcal{K}$, as $f_{\mathcal{U}_t}$ is continuous and bounded by the closed ball $\mathcal{B}\left(\sum_{u \in \mathcal{U}_t} \|u\|\right)$, $C_{\mathcal{U}_t} = f_{\mathcal{U}_t}(\mathcal{K})$ is bounded and closed.

Since $C_{\mathcal{U}_t}$ is a bounded and closed set, and $\|\cdot\|$ is continuous, the image of $C_{\mathcal{U}_t}$ by $\|\cdot\|$ is a bounded closed set as well. Therefore $\min_{u \in C_{\mathcal{U}_t}} \|u\|$ exists and is reached in $C_{\mathcal{U}_t}$. Let u_m an element of $C_{\mathcal{U}_t}$ so that $\|u_m\| = \min_{u \in C_{\mathcal{U}_t}} \|u\|$.

As $F(\mathcal{U}_t) \neq \emptyset$, $0 \notin \mathcal{U}_t$ and $0 \notin \mathcal{K}$ so $0 \notin C_{\mathcal{U}_t}$. Therefore $u_m \neq 0$ and $\|u_m\| \neq 0$. Let define the finite number:

$$T_{escape} = \frac{\|X_0\|}{\|u_m\|} \tag{3.10}$$

where $\|X_0\|$ is the maximum distance of 2 points taken in X_0.

Now that the existence of T_{escape} is ensured, it must be proved that for any control sequence in $\mathcal{U}_t^{\mathbb{N}}$ applied in X_0, the state escapes X_0 in $T \leq T_{escape}$.

As $F(\mathcal{U}_t) \neq \emptyset$, for all trajectories $\{x_k, u_k\}_{k \in \mathbb{N}}$ with controls chosen in $\mathcal{U}_t$ the state escapes the subset X_0, so there exists a T in $\mathbb{N}$ where $x_T \in X_0$ and $x_{T+1} \notin X_0$. Let define $\bar{u}$:

$$\bar{u} = \frac{1}{T} \sum_{i=0}^{T-1} u_i. \tag{3.11}$$

By Definition 3.1, $x_T = x_0 + T\,\bar{u}$. $\bar{u}$ is an element of $C_{\mathcal{U}_t}$ so $\|\bar{u}\| \geq \|u_m\|$, and by definition $\|x_0 - x_T\| \leq \|X_0\|$, therefore $T \leq T_{escape}$. □

The proof can be extended to the disturb case by using the same approach than in 3.2.

3.5 Conclusion

The escape time property has been introduced for a wide class of systems: the monotone linear systems. Thanks to this function, the behavioural relationship between the dynamical system and the AFTS is ensured for the progress set part of the AFTS definition. The progress set only covers the finiteness or infiniteness of trajectories in set of cells of the state space partition. An extension to the timing specification is provided.

Chapter 4

Simulations

Notation

LTI system: Linear Time Invariant system
DFS: Depth First Search
BFS: Breadth First Search
LTL: Linear Temporal Logic
AFTS: Augmented Finite Transition System

Introduction

This part intends to discuss the Augmented Finite Transition System (AFTS, introduced in Chapter 2) controller synthesis algorithm and the input extended state abstraction (introduced in Chapter 1) with practical scenarios involving quadricopter control. The quadricopter is modelled with the point particle model (particle of a given mass, without any momentum, and only subject to an input force) with a proportional velocity feedback.

The backward reachability algorithm introduced in Chapter 2 is used to find a path of terminating modules from the initial set to the goal set. In order to reduce the complexity of the algorithm's BFS (search of modules), we must limit as much as possible the number of cycles present in the FTS. This can be achieved by taking a finer discretization of the state space as unwanted cycles appear when we are not able to see the actual displacement of the state in a set of cells. However, this finer discretization creates bigger abstraction and increase as well the complexity of the algorithm's DFS. Therefore, we must investigate more specifically in which situation the cycles improve the controller synthesis in practice.

The abstraction is supposed to be a simpler model of a complex one. In the case of controller synthesis methods, abstractions are built to reduce the size of the model as much as possible by removing the unused information and keeping the essential one. Too much information is removed when the controller synthesis problem is not

solvable with the chosen abstraction. In this chapter, we study in which situation the input extended abstraction performs better than the non-extended abstraction.

As it has been introduced in Chapter 1, in the case of the input extended abstraction, the modelled noise set is actually bigger than the modelled one. This is the result of not observing part of the state: noise on the unobserved state is damped by the dynamic and does not result in a wrong transition in the FTS. Admissible noise sequences are defined by the dynamic of the unobserved subsystem, in practice (especially in mechanical system), high frequencies admissible noises can have bigger magnitudes and low frequencies will have a magnitude equal to the modelled noise magnitude.

We will first present the model of the quadricopter in the single-agent and multi-agent case (section 4.1). Then we will investigate the practical impact of the cycles on the complexity of the algorithm (section 4.2). And finally we will show the admissible noise set of the input extend abstraction.

4.1 Model of the quadricopter

Each quadricopter is modelled as a 1 or 2 axis double integrator with velocity feedback. The dynamics on each axis of the system are uncoupled. During the real experiments (see Chapter 5), unused axis of the quadricopter will be using a position controller. In the case of multi-agent experiments, N quadricopters are modelled as $d\,N$ axis models (where $d \in \{1, 2\}$ is the number of axis used for the quadricopter).

Model of one axis

Lets consider the following continuous time LTI system defined equivalently than in Definition 4 with the differential equation:

$$\begin{cases} \dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} + \mathbf{w} \\ \mathbf{y} = C\mathbf{x} \end{cases} \tag{4.1}$$

with

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -k \end{bmatrix}, B = \begin{bmatrix} 0 \\ k \end{bmatrix}, k \in \mathbb{R}_+^\star$$

and $\mathbf{x} \in \mathbb{R}^2$ the state, $\mathbf{u} \in \mathcal{U}$ the input and $\mathbf{w} \in \mathcal{W}$ the noise. We choose an input set $\mathcal{U} = \{-0.2, 0.0, 0.2\}$ and a noise set $\mathcal{W} = [\underline{\mathbf{w}}, \overline{\mathbf{w}}]$ where

$$\underline{\mathbf{w}} = \begin{bmatrix} -0.02 & -0.15 \end{bmatrix}^\top, \overline{\mathbf{w}} = \begin{bmatrix} 0.02 & 0.15 \end{bmatrix}^\top.$$

As it has been mentioned in the introduction, 2 models are used. The first one is the non-extended abstraction where we observe both the position and the velocity of the agent:

$$C_{dble} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.2}$$

and the second one is the input extended state abstraction where only the position will be observed:

$$C_r = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{4.3}$$

As different sampling rates are used in the following considerations, assumptions in order to use the input extended abstraction must be verified for the discrete system where the sampling rate is given as a parameter. The discrete LTI system (see Definition 4) sampled at $d_T \in \mathbb{R}_+^*$ is expressed by:

$$\mathbf{x}_{k+1} = A_d \mathbf{x}_k + B_d \mathbf{u}_k + E_d \mathbf{w}_k \tag{4.4}$$

with

$$A = \begin{bmatrix} 1 & (e^{-k\,d_T} - 1)\frac{1}{k} \\ 0 & e^{-k\,d_T} \end{bmatrix}, B_d = \begin{bmatrix} e^{-k\,d_T} - 1 \\ ke^{-k\,d_T} \end{bmatrix}, E_d = Ad.$$

We can identify the different matrices in 1.12 for the computation of the input extended state abstraction:

$A_r = \begin{bmatrix} e^{-k\,d_T} \end{bmatrix}$ $A_z = \begin{bmatrix} 1 \end{bmatrix}$ $A_{rz} = \begin{bmatrix} (e^{-k\,d_T} - 1)\frac{1}{k} \end{bmatrix}$
$B_z = \begin{bmatrix} e^{-k\,d_T} - 1 \end{bmatrix}$ $B_r = \begin{bmatrix} ke^{-k\,d_T} \end{bmatrix}$
$E_r = \begin{bmatrix} e^{-k\,d_T} \end{bmatrix}$ $E_z = \begin{bmatrix} e^{-k\,d_T} - 1 \end{bmatrix}$ $E_{rz} = \begin{bmatrix} (e^{-k\,d_T} - 1)\frac{1}{k} \end{bmatrix}$

For each sampling rate, the input and noise sets are bounded, the velocity is not coupled to the position, the dynamic on the velocity is asymptotically stable and monotonic. We can, according to Chapter 1, compute the input extended state abstraction.

Discretization of the environment

The discretization of the observed set is uniform for the 2 models (see figure 4.1). For experiments over 1 axes, we discretize the environment in position with a discretization step $\mathbf{x}_d \in \mathbb{R}_+^*$. In the case of the raw double integrator model, the velocity subspace is discretized every $\mathbf{v}_d \in \mathbb{R}_+^*$ steps. For 2D environment, we use the same discretization step $\mathbf{x}_d$ for the 2 position subspaces and $\mathbf{v}_d$ for the 2 velocity subspaces. We call $N^o \in \mathbb{N}$ dimension of the observed space $\mathfrak{X}^o$. There is no need of discretization for the input set as it is already a discrete set. We use the following notation for the input bounds: $\mathbf{u}_- = -0.2$, $\mathbf{u}_+ = 0.2$.

4.2 Controller synthesis for an AFTS

Chapter 2 presents an algorithm that can find control strategy despite the presence of cycles. These cycles can usually be avoided in the abstraction process by choosing a finer partition of the state space, thus any progress of the continuous state of the dynamical system implies a progress of the discrete state in the abstraction. However, it comes at a price of a bigger FTS in term of number of states. The gain of considering the cycles must be therefore investigated for each systems and specifications. Hereby, we present one situation where the abstraction with cycles

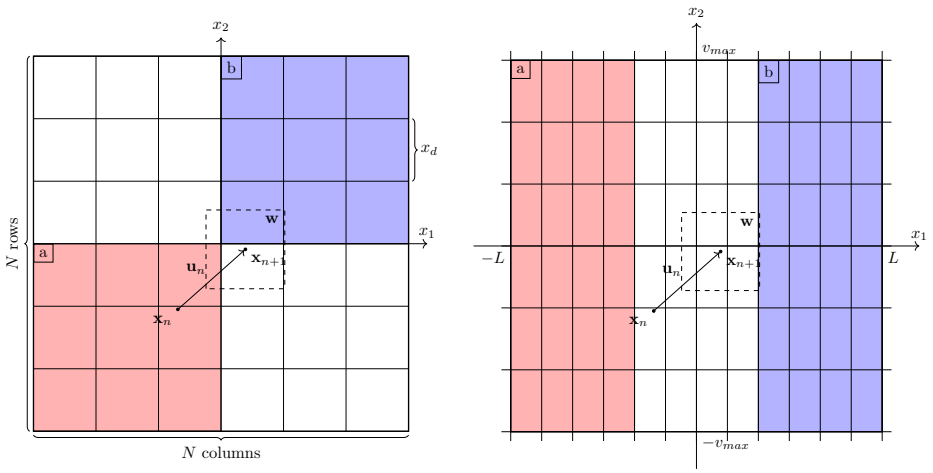

Figure 4.1: Discretization of the 2D environment (left) and of the second integrator model (right).

has a lower complexity compared to abstraction without cycles. Then, we discuss the complexity of the BFS search of the modules.

As these results only relate to the algorithm introduced in Chapter 2 (and not the abstraction methods presented in Chapter 1), the single integrator model (that can be identified as a 0-input extended abstraction) is used in this section. Let the dynamical system sampled with the sampling time d_T. Let S_{sl} the abstraction with self-loops and S_{nosl} the abstraction without any self-loops. The self-loops can be avoided in S_{nosl} by decreasing the discretization step of the state space (i.e., by choosing a finer partition) until that every discrete state goes to different states when a non null control input is applied. The same environment of length $L \in \mathbb{R}_+^*$ will be used as well as the same LTL specifications (see figure right in Figure 4.1) described by:

$$X = [-L, L] \qquad a = [-L, \tfrac{-L}{3}] \qquad \varphi = (\square\diamond a) \wedge (\square\diamond b)$$
$$X_0 = [-L, L] \qquad b = [\tfrac{L}{3}, L]$$

Self-loops

For the abstraction S_{sl}, one requirement such that there exists a controller solution of the problem is that for every noise $\mathbf{w}$, there exists a control $\mathbf{u}$ such that $|\mathbf{u}| > |\mathbf{w}|$. This inequality asserts that the system must at least be able to counteract the influence of the noise (otherwise, any control strategy cannot keep the state of the system in the environment). In the case of the abstraction S_{nosl}, the control input

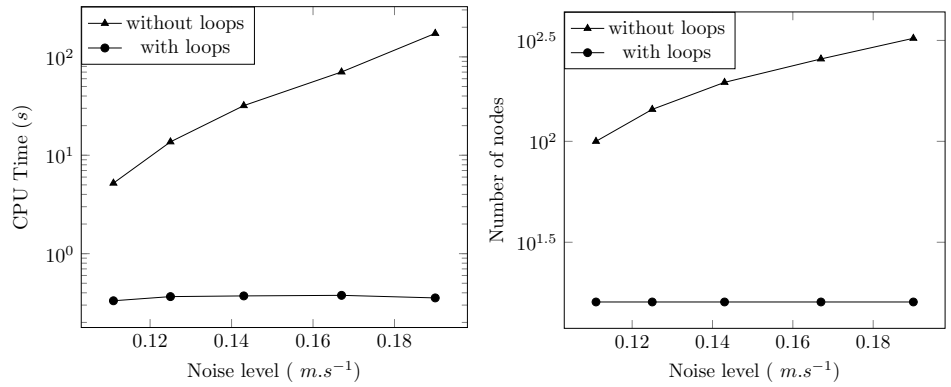

Figure 4.2: Comparaison of abstraction with self-loops and abstraction without self-loops. For the same model, when the noise increase, the model with self-loops does not need a finer discretization, this result in smaller abstraction in term of nodes (see right plot) and smaller computation time (see left plot) for the self-loop model.

must at least bring the state to the next cell (to avoid any self-loops). Let $\mathbf{x}_d^{nosl}$ the discretization step of S_{nosl}, then $\mathbf{x}_d^{nosl}$ must verify $\mathbf{x}_d^{nosl} < |\mathbf{u} + \mathbf{w}|\, d_T$ for any non null control actions $\mathbf{u}$ and any noise $\mathbf{w}$ (more over, the previous condition must hold as well, i.e., for all $\mathbf{w}$ there exists a $\mathbf{u}$ such that $|\mathbf{u}| > |\mathbf{w}|$ must hold as well).

Figure 4.2 shows that the model S_{sl} have the same complexity for any noise level. In the case of the abstraction S_{nosl}, as the number of cells needed to find a solution to the controller is inversely proportional to the noise magnitude (the number of nodes $N_{nosl} \approx {}^{L}/_{\mathbf{x}_d^{nosl}} \geq {}^{L}/_{\overline{\mathbf{u}}-\overline{\mathbf{w}}}$), when the maximal noise magnitude $\overline{\mathbf{w}}$ of the model is increasing, the discretization step $\mathbf{x}_d^{nosl}$ must decrease and the number of nodes increases. This results in a bigger model than by taking self-loops in account.

Size of the fixed points

The real drawback of AFTS method lie in the BFS search of a terminating control combination. In the previous part, we have been investigating a case where the maximal module size was equal to 2. However, modules might be bigger in other systems. For example, by taking the model of the previous section with 2 axis instead of 1, the maximal module size will be squared (so equals to 4). The complexity is increasing exponentially in the number of controls in $\mathcal{U}$ (the number of combinations of controls for a set of n nodes is equal to $|\mathcal{U}|^n$) and in the number of possible successors of the BA. The current implementation of our algorithm is not able to deal with computation of modules bigger that 9 nodes (which corresponds in our 2D model where $|\mathcal{U}| = 9$ to $|\mathcal{U}|^9 \approx 3.8e8$ possible module configurations).

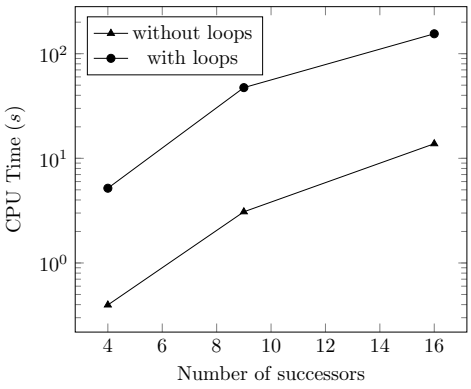

Figure 4.3: Influence of the cycles sizes.

We have been running a test (see figure 4.3) with increasing number of successors. The more successors we have, the biggest will be the modules of the plan, resulting in a longer computation time.

4.3 Number of memories of the abstraction

Hereby, we investigate how many memories it worth to consider in the input extended abstraction. This is done by considering the impact of a memory on the state space size and on the number of transitions of the abstraction.

State space size

The state space cardinality of the input extended state abstraction is mainly dependant on the cardinality of the set $\mathcal{U}$ (or the cardinality of the partition of $\mathcal{U}$). Let $N \in \mathbb{N}$ be the number of symbols of the continuous state space partitioning. The size of the abstraction state space is equal to $N\,|\mathcal{U}|^{\Delta n_u}$. As the size of the state space is exponentially dependant on the number of memories for the input set, we should reduce as much as possible Δn_u (with the constraint of finding a feasible solution for the control synthesis problem).

Number of transitions

In the case of the input extended state abstraction, for a given state, the number of successors is directly dependant on the volume of the reachable set $X^i(U_n)$ for a past input sequence $U_n = (\mathbf{u}_{n-\Delta n_u}, \ldots, \mathbf{u}_{n-1})$. The smaller this volume is the more accurate is the abstraction (as it is selecting fewer number of successors). Therefore, we will investigate how the volume of the set $X^i(U_n)$ is influenced by the number of input memory considered. Let T defined by $T = \Delta n_u dT$, where

dT is the sampling time of the input extended state abstraction. T corresponds to the duration of the past control input memory.

In order to see the influence of the sampling rate, we will work on the continuous model. We adopt a general formulation for the asymptotically stable monotone linear system for the unobserved system $\mathcal{S}^i$. Let the following unobserved system:

$$\dot{\mathbf{x}}^i = A\mathbf{x}^i + B\mathbf{u} + E\mathbf{w} \tag{4.5}$$

In the case of monotone linear system, the volume of successors is directly dependant on the bounds of the set $X^i(U_n)$. By computing:

$$\begin{aligned} \overline{\mathbf{x}}^i(T) &= \varphi(T, \overline{\mathbf{x}}^i, \mathbf{u}(t), \overline{\mathbf{w}}) \\ \underline{\mathbf{x}}^i(T) &= \varphi(T, \underline{\mathbf{x}}^i, \mathbf{u}(t), \underline{\mathbf{w}}) \end{aligned} \tag{4.6}$$

and thanks to the linearity of the differential equations, we have:

$$\overline{\mathbf{x}^i}(T) - \underline{\mathbf{x}}^i(T) = \varphi(T, \overline{\mathbf{x}}^i - \underline{\mathbf{x}}^i, \mathbf{0}, \overline{\mathbf{w}} - \underline{\mathbf{w}}). \tag{4.7}$$

We can then express the quantity $\Delta\mathbf{x}^i(T) = \overline{\mathbf{x}}^i(T) - \underline{\mathbf{x}}^i(T)$ with:

$$\begin{aligned} \Delta\mathbf{x}^i(T) &= \Delta\mathbf{x}^i(0)e^{AT} + \int_0^T e^{At}dtE\Delta\mathbf{w} \\ \Delta\mathbf{x}^i(0) &= A^{-1}(B\Delta\mathbf{u} + E\Delta\mathbf{w}) \end{aligned} \tag{4.8}$$

where $\Delta\mathbf{u} = \overline{\mathbf{u}} - \underline{\mathbf{u}}$ and $\Delta\mathbf{w} = \overline{\mathbf{w}} - \underline{\mathbf{w}}$. Therefore, the volume $V^i(T)$ of the set $[\underline{\mathbf{x}}^i(T), \overline{\mathbf{x}}^i(T)]$ does not depend on the input sequence. As the set $[\underline{\mathbf{x}}^i(T), \overline{\mathbf{x}}^i(T)]$ is a rectangle, $V^i(T)$ is equal to the product of all vector's components of $\overline{\mathbf{x}}^i(T) - \underline{\mathbf{x}}^i(T)$.

$$V^i(T) = \prod_{i=1}^{n} \Delta\mathbf{x}_i^i(T) \tag{4.9}$$

The expression of $V^i(T)$ shows that the volume of $X^i(U_n)$ is directly dependant on the dynamics of the unobserved system. In figure 4.4, the volume V^i over the number of memories is plotted as well as the volume of 1,2,3 and 4 cells. The sampling time used is $dT = 0.5s$ which corresponds to the decay time of the unobserved system. As the partitioning is uniform, each cell has the same volume V_{cell}. In the initial dynamical system, as the volume of successor is V^i and as the volume of a cell is V_{cell} in the partition, the number of successor in the abstraction must be strictly greater than:

$$\alpha\frac{V^i}{V_{cell}} \tag{4.10}$$

where α corresponds to the smallest eigenvalue of A_{ro}. For more than 2 memories, adding one memory does not decrease the volume V^i a lot, this means that after the partitioning process, the number of successors for each state will not decrease

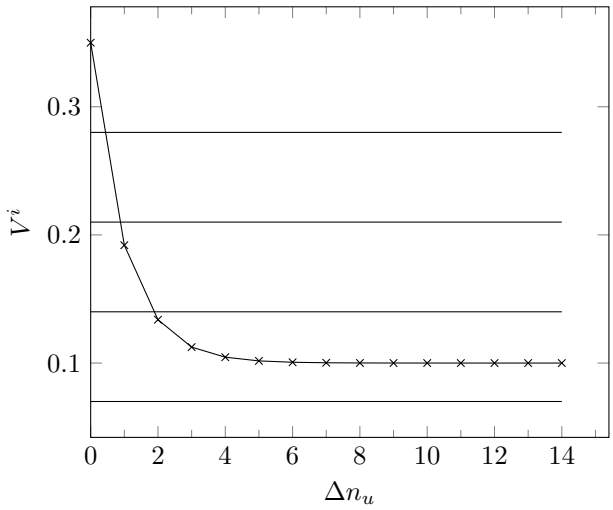

Figure 4.4: Impact of the number of memories Δn_u over the size of the possible state of the input extended state abstraction. Horizontal lines correspond to the volume of respectively 1,2,3 and 4 cells.

consequently. Therefore and according to the previous results (memories exponentially increase the state space size), in this case we will preferably choose models with 0, 1 or 2 memories. More memories will increase the state space size of the abstraction for very few improvements on the accuracy of the model. In practice, we have noticed that 0 or 1 memory were enough for most of our applications.

Self-loops

Let a trajectory $\{p_i, \mathbf{u}_i\}_{i\in\mathbb{N}}$ of the abstraction $\mathcal{S}_a$ as it is defined in chapter 1. If at timestep n the trajectory is taking a self-loop transition in the abstraction:

$$(p_n, \mathbf{u}_{n-\Delta n_u}, \ldots, \mathbf{u}_{n-1}) \xrightarrow{\mathcal{S}_a \quad \mathbf{u}_n} (p_{n+1}, \mathbf{u}_{n+1-\Delta n_u}, \ldots, \mathbf{u}_n) \tag{4.11}$$

then $p_n = p_{n+1}$ and the input sequence verifies $(\mathbf{u}_{n+1-\Delta n_u}, ..., \mathbf{u}_n) = (\mathbf{u}_{n-\Delta n_u}, ..., \mathbf{u}_{n-1})$. This means that self-loops only appear if the control input is constant over time during at least $n+1$ time steps. If the unobserved dynamic of the system is damped in less than Δn_u timesteps, then the input extended abstraction abstraction with Δn_u memories will not create any self-loops corresponding to the transient state.

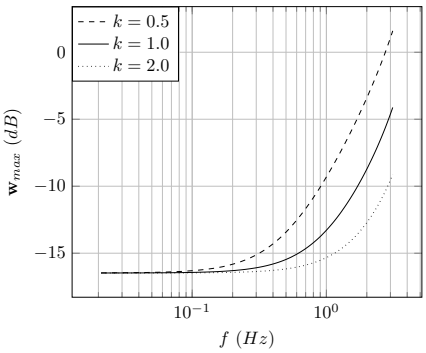

Figure 4.5: Bode plots of the admissible noise for different models.

Noise

In order to show how the admissible noise behave according to the system, we have been plotting the bode diagram for several models (see figure 4.5). When the gain k of the proportional velocity feedback loop decreases, the dynamic tends to be slower than the sampling time. As the noise on the unobserved system impact the observed system through the dynamic of the unobserved system, high frequency tends to be more damped when the dynamics of the unobserved system is slower. Please note that the maximum static noise admissible (reached for $f = 0Hz$) is the same for all the models and is the same than the second integrator model without input extended state.

4.4 Conclusion

Models with undesirable self-loops (self loops that appears on input different from $\mathbf{0}$) produce systems where a transition will be taken after an unknown finite time. This hide all the time information of our system. Thanks to the AFTS, the partitioning of the state space might be rougher (and therefore the abstraction is smaller) and still find a controller solution. This results in a possible state space that is generally too high to use any graph search algorithm on it.

The number of memories that worth to be considered depends on the partition of the state space and on the dynamic of the unobserved system. If more memories lead to a more accurate abstraction, they lead as well to a bigger size of the abstraction. In our case, 1 memory was enough in most of our applications.

Chapter 5

Experiments

Introduction

Experiments are performed with the Iris Plus 3DR™quadricopters in the testing environment of the Smart Mobility Lab of KTH. The environment is a 5×5 meter square area of 2 meters high. The position of each agent is measured with the motion capture system Qualisys™. The velocity feedback controller and the discrete controller run on an offboard computer. The transmission of the control inputs to the quadricopter is computed at 35Hz through an RC link. The overall program uses with the ROS framework. The computation of the abstractions and the plan solution of the problem are computed offline (see digram 5.1).

Tests are done on single agent (1 or 2 axis) and on multi-agent scenarios. Due to the state space explosion problem, we did not found any plans for some environments of too high dimensions (2D or multi-agent). The double integrator abstraction (without input memory) leads to models that were unusable in 2D environments. As well as the input extended state abstraction with 1 input memory could not be used with multi-agent planning.

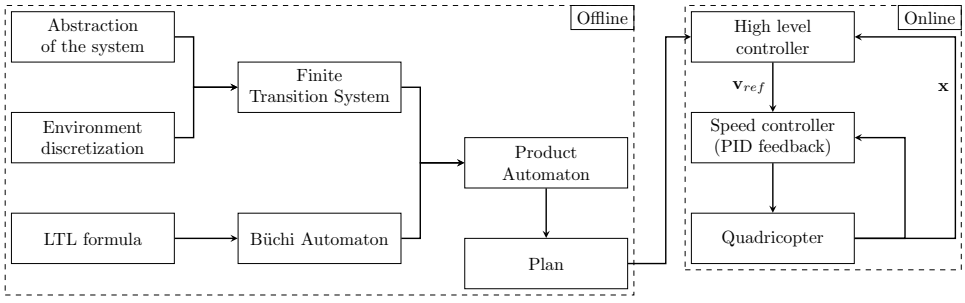

Figure 5.1: Diagram of the controller synthesis process.

5.1 Single axis

In this part, the extended input abstraction of the second integrator model is studied over a single axis and compared with the not-extended abstraction of the model (without input extended state). For each model, the size of the environment is chosen large enough such that a plan solution exists.

In our setup, the timing constant of the unobserved system is $\tau_r = 1/k = 0.5s$. This is relevant to use input memories when the time constants of the continuous system are close to the sampling time dT (which is $\approx 1.0s$ in our case). For higher sampling time ($dT \gg \tau_r s$) the single integrator model ($\Delta n_u = 0$) is more relevant as the transient state effect is vanished. For other cases, the benefit of using the input extended state abstraction must be investigated individually. By choosing a model with Δn_u memories, all self loops due to transient states that last less than $(\Delta n_u + 1)dT$ are be removed, thus the resulting abstraction is more deterministic.

Every scenario considered in this chapter corresponds to the one described in Section 4.2.

Second integrator model with velocity feedback

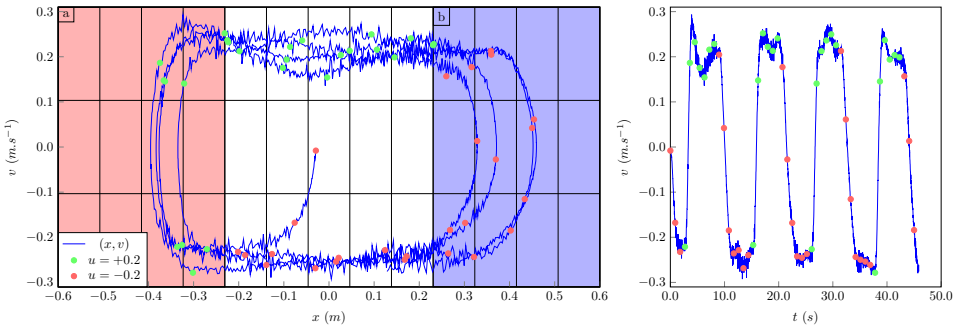

Figure 5.2: Trajectory (left) in the state space (position, velocity) and velocity profile (right) of the double integrator model. Each dot corresponds to a timestep of the discrete controller, the color indicates the control input applied. The agent is trying to achieve "go infinity often to a (in red) and go infinity often to b (in blue)'.

For the second integrator model, both the position and the velocity axis are discretized. Figure 5.1 shows a run for this model with a controller that enforces the system to verify $\varphi = (\square\diamond a) \wedge (\square\diamond b)$. The agent successfully verify φ for some timestep by going from a to b and from b to a. The non-deterministic behaviour can be seen when the state is in the cell $[0.4, 0.5] \times [-0.1, 0.1]$: in one case, the state is going to the cell $[0.4, 0.5] \times [-0.3, -0.1]$; in the other case, the state is going

to $[0.3, 0.4] \times [-0.3, -0.1]$. Despite the non-deterministic behaviour, every possible path of the controlled system will verify the formula.

The velocity plot in Figure 5.1 shows clearly an overshoot when a step is applied on the velocity command. This is not supposed to happen according to our model (the error is supposed to be a decreasing exponential function only). We could have chosen a model closer to the actual quadcopter behaviour, however this would come at a price of a bigger state space representation. We chose to increase the noise set instead. Thus the state space representation is still acceptable (2 dimensions only) and fits the capability of our algorithm implementation.

The position of the agent at $x \approx -0.3m$ is clearly going backward at the next timestep (the next position of the agent is $x^+ \approx -0.4m$) even if the control is equal to $v_{ref} = 0.2m.s^{-1}$. Therefore, the model with no memories ($\Delta n_u = 0$) cannot provide a solution for this environment: any state would have a transition going in the opposite direction of the reference velocity (which means that there is no plan solution for bringing the agent from one area to another). By adding 1 memory ($\Delta n_u = 1$), the transient state (that causes the backward transition) can be discriminated from the steady states.

The final controlled system (abstraction + controller) have self-transitions that usually comes from the steady states behaviour of the dynamical system. However, thanks to the escape time property and the AFTS definition, these self-transitions cannot be taken infinitely (they belong to the progress set).

Extended input abstraction with $\Delta n_u = 1$ memory

For the extended input abstraction, the velocity dimension is not discretized (only the position which corresponds to the observed state). Figure 5.1 show the trajectory in the state space of the initial dynamical system (position, velocity) and also a velocity profile with the reached set on the velocity axis after applying a control input. The size of the abstraction compared to the second integrator model (see previous subsection) is smaller: for the second integrator model there are $13 \times 3 = 39$ states; whereas for this model, there are $13 \times 2 = 26$ states. Therefore, for the same problem, the input extended state abstraction with 1 memory is less complex than the raw double integrator with velocity feedback model.

On the velocity profile in Figure 5.4, each time that a positive (respectively negative) control is applied ($u = 0.2m.s^{-1}$, green dot in the figure; resp. $u = -0.2m.s^{-1}$, red dot), the next state is supposed to belong to the corresponding reachable set of the velocity state (green stripe area; resp. red stripe area). It does not always happen, but at the same time no wrong transition was taken during the entire run. This can be understand by plotting the velocity profile and the maximum noise admissible such that the output of the filter (defined in Section 1.6) verifies the condition 1.30 (see Figure 5.4). Despite that the reachability property is not verified (which means that the quadricopter is not properly modelled), as the noise is always smaller than the maximum admissible noise, no wrong transition is taken. In the case of the second integrator model, any noise sequence with higher

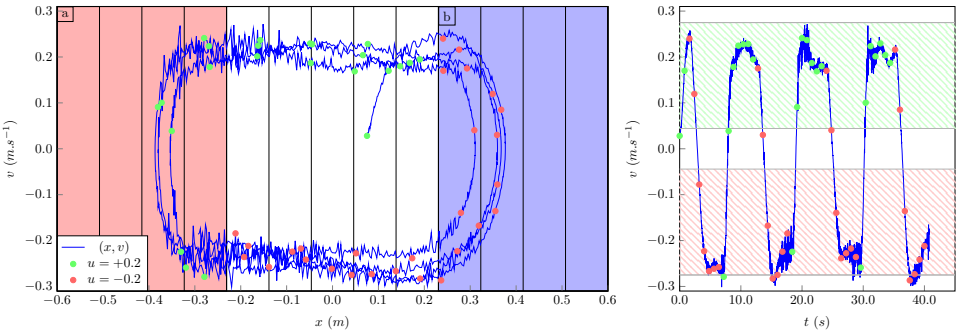

Figure 5.3: See description of Figure 5.1. The plot of the velocity (on the right) is extended with the red stripes area and the red stripes area that corresponds to admissible successors of the velocity for a constant input. After each red dot (resp. green dot), the next dot is supposed to be in the red area (resp. green area). However, thanks to Section 1.6, we know that the transient error might be bigger than the static admissible error and therefore this trajectory is still admissible.

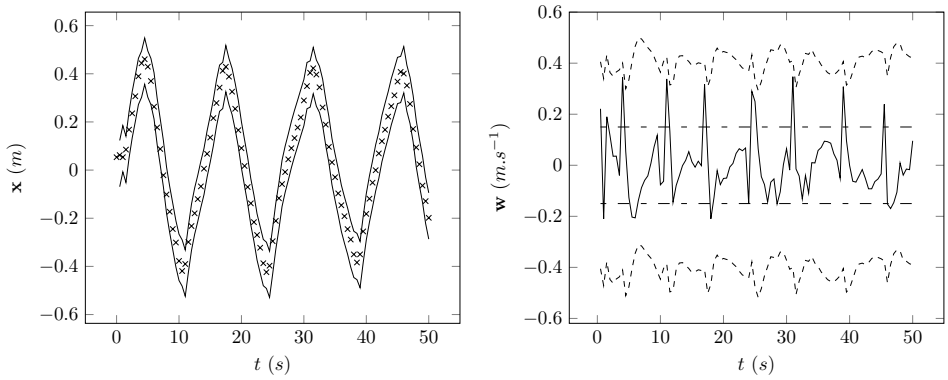

Figure 5.4: The admissible noise of the abstraction have higher magnitudes for the high frequencies. The first plot correspond to the trajectory of the quad, the second is a plot of the noise (plain line) with noise upper and lower bounds (in simple dashed lines) and the upper and lower noise bounds used in the model (double dashed line).

frequency might bring the model to take a non-existent transition. Therefore, this model is more robust to mismodeling of the dynamical system than the raw double integrator with velocity feedback model.

Abstraction without memory ($\Delta n_u = 0$)

The input extended abstraction with no memory only observe the position state while giving velocity commands, thus is equivalent to the single integrator model. As we have seen on the previous parts, the previous environment cannot accept solutions for the single integrator model. Thus, we are using a bigger environment for which a controller solution exists.

The sampling time ($dT \approx 2.0$ for the model with self-loops) is greater than the time constants of the system and therefore, no memories are necessary to abstract the system (i.e., $\Delta n_u = 0$). In such scenario, adding memories would unnecessarily increase the complexity of the problem (bigger state space size of the abstraction) for maybe a small increase in the accuracy of the abstraction.

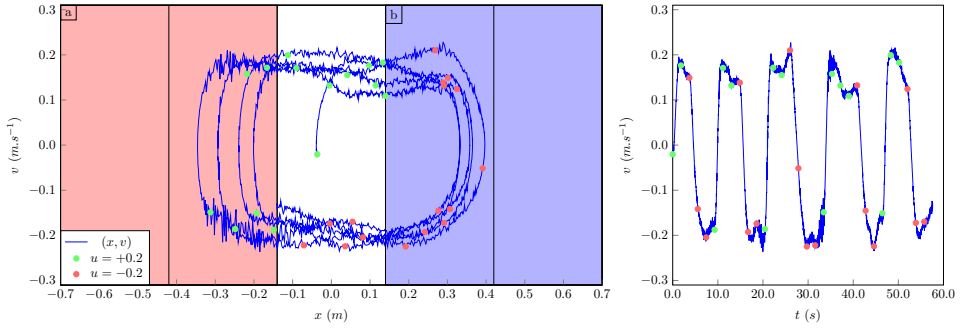

Figure 5.5: Trajectory and velocity profile of the simple integrator model with self loops. The agent is trying to achieve 'go infinity often to a and b'.

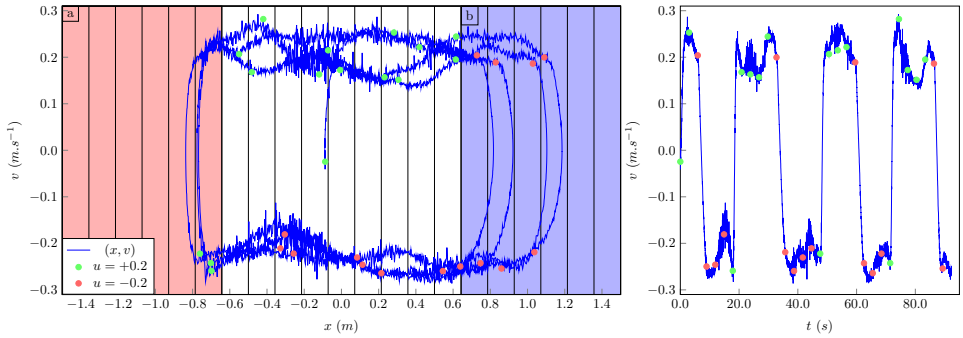

Figure 5.6: Trajectory and velocity profile of the simple integrator model without self loops. The agent is trying to achieve 'go infinity often to a and b'.

Figure 5.5 shows a trajectory for the model with self-transition allowed and Figure 5.6 is the model without self-transition. As it has been shown in Chapter

4, the model without self-transition require a bigger state space to actually be able to see a small displacement of the position. Whereas for the model with self-transitions, as any displacement can create a self-transition, there is no upper bound for the size of the cells.

Both of the model verify the formula during these few steps, bringing the state from a to b and from b to a.

5.2 Double axes

Models used for experiments in a 2D environment are built out of the 1D model concatenation (see Chapter 4, Section 4.2). And thus, previous conclusions still stand for each axis of the 2-axis models. As the size of the models is squared in comparison to the 1-axis model, plan solution has been found only for the 1 and 0 input memory extended state abstraction. The raw double integrator model is creating abstraction of too high complexity and thus it is not studied in this part.

$\Delta n_u = 1$ input memory model

Figure 5.7 shows the trajectory of the agent in a 2D environment (see Section 4.2) using the input extended abstraction with 1 memory. The 2D scenario described in Section 4.2 is considered. The controller successfully bring the agent from area a to area b and from area b to a.

One will notice that the agent is sometimes going in the opposite direction of the applied control when the control action is changed (see the 2 states on the extreme south-west of the trajectory in Figure 5.7). Therefore, the "no memory" input extended state abstraction was unusable in this case.

$\Delta n_u = 0$ input memory model, real case scenario

In the case of the wind turbine application, the quad have to go from the base station (area l in Figure 5.8) to the wind turbine blade and explore a given set of regions (area l ,m , n and o in Figure 5.8) while avoiding the blade (striped area in Figure 5.8). The wind can be modelled as a perturbation on the velocity of the quadricopters. In this scenario, the first integrator model is used (input extended abstraction with 0 memory).

As the backward reachability algorithm implementation is using a depth first search, the path is not optimal in time. In fact, the agent is visiting the area n then the area l, then m and o. The use of a breadth first search would have improve this. When all the area are visited, the controller is only trying to avoid the *blade* area.

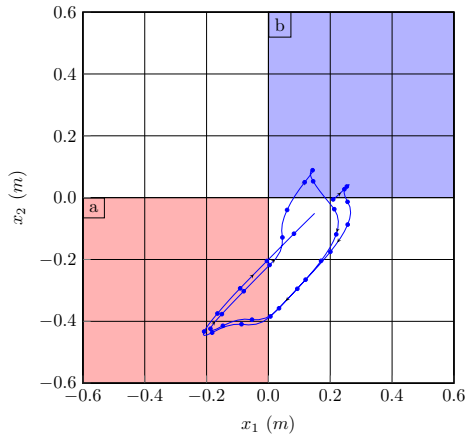

Figure 5.7: $\Delta n_u = 1$ input memory model in 2D environment. Black arrows corresponds to the direction of the agent. At each blue dot, the controller is updated. The chosen controller inputs are not represented in order to keep the graph clear.

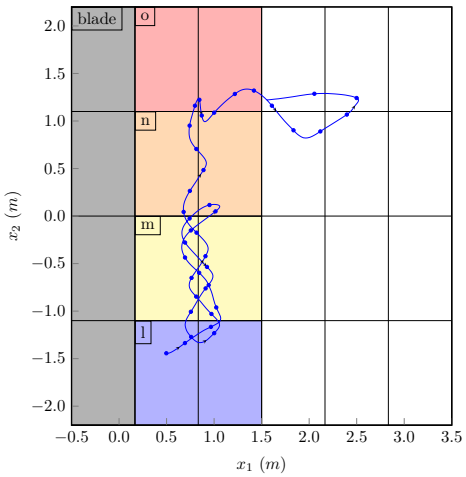

Figure 5.8: A trajectory of the quad starting from l verifying the formula $(\Diamond l) \wedge (\Diamond m) \wedge (\Diamond n) \wedge (\Diamond o) \wedge (\Box \neg blade)$. Arrows indicate the displacement of the agent. Each dot corresponds to a step of the discrete controller.

5.3 Multi agent

The complexity of the multi agent problem disallows us from using models producing high number of states. Therefore, the N quadricopters will be modelled as a $2N$ dimensions single integrators.

The quadricopters have to verify the following LTL formula:

$$\varphi = (\Box \neg out) \wedge (\Box \Diamond a) \wedge (\Box \Diamond b) \wedge (\Box \neg collide)$$

where *out* corresponds to one or more quadricopters outside of the environment, a agent 1 in north-east and agent 2 in south-west, b agent 1 in south-west and agent 2 in north-east and *collide* if there is a collision (quadricopters on the same cells). Figure 5.9 shows a run of the plan for few steps. In steps $T = 25$ to $T = 29$, the agents are taking cyclic transitions in the AFTS. However, as the chosen local control strategy is part of the progress set (i.e., the escape time property is granted), the agents are ensured to leave this set of states in finite time. At $T = 98$, agents have visited a then b and came back to a, therefore, they verify the formula for some steps. A video of the experiment is available online[1].

[1] `https://youtu.be/yj0_olU1tYI`

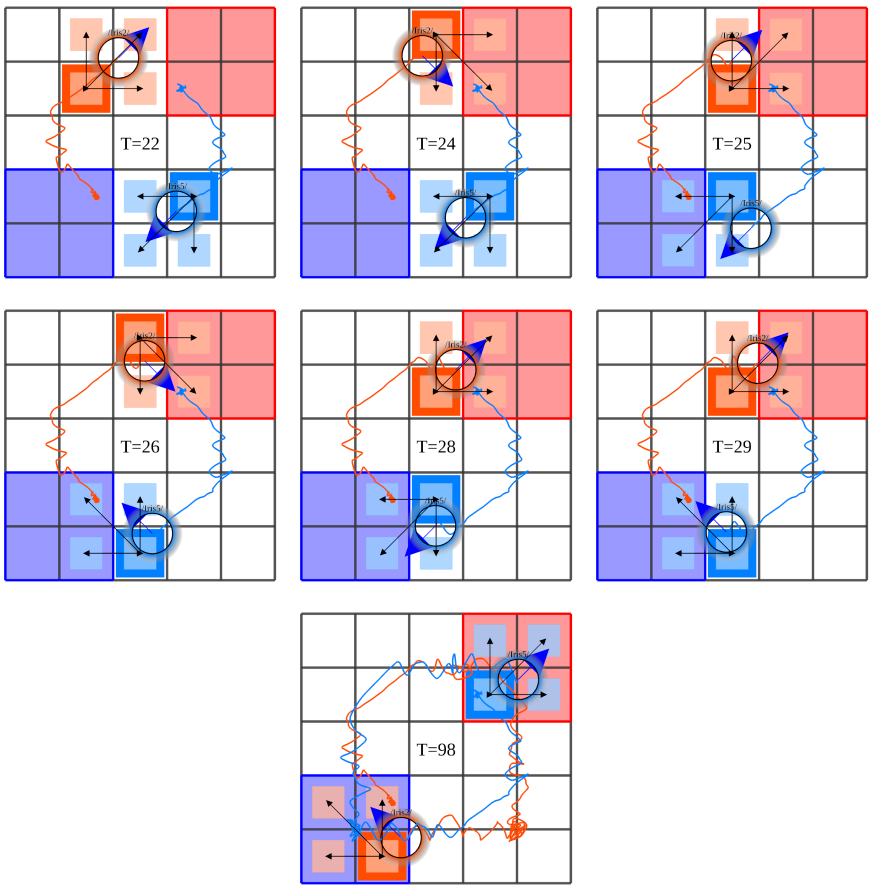

Figure 5.9: Run of the multiagent plan. Agent blue and red need to exchange position infinitely often. Cells in plain colors are the current state of the quadricopter. Cells in light color are the successors. Big blues arrowhead are the control action.

Conclusion

In this work, a new abstraction method using input memories has been introduced. These memories can efficiently discriminate between a transient state behaviour and a steady state one. The extension of the state with memories increase the state dimension and for a same partitioning of the state space, our method will produce discrete abstraction with a bigger state space representation. However, these memories give as well new information that lead to a more accurate abstraction and input memories might yield to a solvable problem that would have been unsolvable without them. In practice, the overall abstraction method lead to smaller state space representation.

In the same spirit of investigating abstraction methods to reduce the complexity of the problem, and also in order to fully benefit from the input extended abstraction method, the Augmented Finite Transition System (AFTS) has been introduced in Chapter 2. This model extends the FTS definition with a progress set that identifies terminating local control strategies (i.e., by applying this control strategy on an set of states, the trajectory will escape this set in finite time). The proof of simulation relation is developed in Chapter 3.

These methods have been successfully implemented and used in Chapter 4 and Chapter 5. Due to state space explosion problem, the input extended abstraction with more than 1 memory lead to models of too high complexity with poor benefit on the accuracy of the abstraction. This conclusion is obviously dependant the continuous model property and must be investigated individually for each application. However, dynamical system that have transient states as long as the sampling time of the system are good candidates for input extended abstraction.

As abstraction methods are dependant on the chosen system representation and partition of the state space, they directly affect the complexity of the controller synthesis problem. The information that is worth to consider needs to be investigated for each problem specification so that the complexity of the controller synthesis problem is not unnecessarily complex. Therefore, efforts on the state space representation and on finite models to use (with there corresponding path planning algorithm) should be of great interest as well as tools to validate the effectiveness of these abstraction method.

Bibliography

Rajeev Alur and Salvatore La Torre. 2004. Deterministic generators and games for LTL fragments. *ACM Transactions on Computational Logic (TOCL)*, 5(1):1–25.

Tomáš Babiak, Mojmír Křetínskỳ, Vojtěch Řehák, and Jan Strejček. 2012. LTL to büchi automata translation: Fast and more deterministic. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 95–109. Springer.

Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. The MIT Press. ISBN 026202649X, 9780262026499.

Calin Belta, Antonio Bicchi, Magnus Egerstedt, Emilio Frazzoli, Eric Klavins, and George J Pappas. 2007. Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):61–70.

Dimitris Boskos and Dimos V Dimarogonas. 2015. Decentralized abstractions for feedback interconnected multi-agent systems. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 282–287. IEEE.

A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1):35 – 84. ISSN 0004-3702.

Edmund M Clarke, Orna Grumberg, and Doron Peled. 1999. *Model checking*. MIT press.

Giuseppe De Giacomo, Fabio Patrizi, and Sebastian Sardina. 2010. Generalized planning with loops under strong fairness constraints. In *12th International Conference on the Principles of Knowledge Representation and Reasoning*.

Georgios E Fainekos, Savvas G Loizou, and George J Pappas. 2006. Translating temporal logic to controller specifications. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 899–904. IEEE.

Jicheng Fu, Vincent Ng, Farokh B Bastani, I-Ling Yen, *et al.* 2011. Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems.

In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1949.

LCGJM Habets, Pieter J Collins, and Jan H van Schuppen. 2006. Reachability and control synthesis for piecewise-affine hybrid systems on simplices. *IEEE Transactions on Automatic Control*, 51(6):938–948.

Marius Kloetzer and Calin Belta. 2008a. Dealing with nondeterminism in symbolic control. In *International Workshop on Hybrid Systems: Computation and Control*, pages 287–300. Springer.

Marius Kloetzer and Calin Belta. 2008b. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297.

Gerardo Lafferriere, George J. Pappas, and Shankar Sastry. 2000. O-minimal hybrid systems. *Mathematics of Control, Signals and Systems*, 13(1):1–21. ISSN 1435-568X.

Thomas Moor and Jörg Raisch. 2002. Abstraction based supervisory controller synthesis for high order monotone continuous systems. In *Modelling, Analysis, and Design of Hybrid Systems*, pages 247–265. Springer.

Petter Nilsson and Necmiye Ozay. 2014. Incremental synthesis of switching protocols via abstraction refinement. In *53rd IEEE Conference on Decision and Control*, pages 6246–6253. IEEE.

George J Pappas. 2003. Bisimilar linear systems. *Automatica*, 39(12):2035–2047.

Fabio Patrizi, Nir Lipovetzky, and Hector Geffner. 2013. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In *IJCAI*.

Gunther Reißig. 2011. Computing abstractions of nonlinear systems. *IEEE Transactions on Automatic Control*, 56(11):2583–2598.

Paulo Tabuada. 2009. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media.

Danielle C Tarraf. 2014. An input-output construction of finite state rho mu approximations for control design. *IEEE Transactions on Automatic Control*, 59 (12):3164–3177.

Jana Tůmová, Boyan Yordanov, Calin Belta, Ivana Černá, and Jiří Barnat. 2010. A symbolic approach to controlling piecewise affine systems. In *49th IEEE Conference on Decision and Control (CDC)*, pages 4230–4235. IEEE.

Majid Zamani, Ilya Tkachev, and Alessandro Abate. 2014. Bisimilar symbolic models for stochastic control systems without state-space discretization. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, HSCC '14, pages 41–50, New York, NY, USA. ACM. ISBN 978-1-4503-2732-9.