Sampling-based algorithms for motion planning with temporal logic specifications

Felipe J. Montana

Department of Automatic Control and Systems Engineering
University of Sheffield

This thesis is submitted for the degree of
Doctor of Philosophy

January 2019

Acknowledgements

I would like to express my gratitude to my supervisors, Prof. Tony J. Dodd and Dr. Jun Liu, for all the guidance and support throughout my PhD studies.

Thank you to all my friends in Sheffield Robotics for all the discussions and good moments that we shared during this journey.

Also, I would like to thank my sponsor, CONACyT, for giving me the great opportunity to study abroad and contribute to the development of science and technology.

Finally, a special thank you to my family for being a constant source of support throughout all these years.

Abstract

Autonomous mobile robots are machines capable of performing tasks, operating without human intervention. Their presence has increased in applications such as personal assistance, manufacturing, etc. One of the main challenges of controlling a robot autonomously lies in the area of motion planning. This planning needs to consider elements such as system dynamics, uncertainties, dynamic environments, safety and reliability. These requirements have motivated the development of methods that combine control theory and model checking techniques to automatically compute plans that provably guarantee the execution of a given specification.

Although model checking has been successfully used to verify discrete systems, its application in autonomous mobile systems presents certain challenges such as: (i) the problem of computing finite models from high-dimensional systems with infinite number of states; (ii) the computation of controllers for systems with kinematic and dynamic constraints; (iii) the computation of robust and reactive controllers to deal with uncertainties and dynamic environments; and (iv) the state explosion problem due to the total number of possible behaviour or states when multiple robots are considered.

The challenges presented above are addressed in this thesis. Specifically, the proposed methods in this work are focused on the motion planning of mobile systems based on linear temporal logic and metric interval temporal logic specifications. They are based on sampling methods which are widely used in the point to point motion planning for high-dimensional systems with dynamic constraints. By using these methods and automata-based theory, the solutions in this thesis mitigate the state explosion problem presented in available methods.

The main contributions of the thesis are summarised as follows. Chapter 4 develops a new algorithm to find optimal trajectories for deterministic systems with kinematic and differential constraints subject to co-safe temporal logic specifications. Systems with uncertainty in motion and sensing are considered in Chapters 5 and 6. In these chapters, two novel approaches to maximise the probability of completing temporal logic specifications are proposed. Finally, Chapter 7 presents a solution for multi-robot systems subject to co-safe linear temporal logic specifications. All the proposed algorithms are demonstrated with several numerical examples.

Table of contents

List of figures **xi**

List of tables **xiii**

List of abbreviations **xv**

1 Introduction **1**
- 1.1 Aim and objectives . . . 4
- 1.2 Contributions . . . 5
- 1.3 Publications . . . 6
- 1.4 Thesis overview . . . 7

2 Background and Related Work **9**
- 2.1 Motion planning . . . 9
  - 2.1.1 Roadmaps . . . 12
  - 2.1.2 Cell decomposition . . . 12
  - 2.1.3 Method of potential fields . . . 13
  - 2.1.4 Optimisation-based methods . . . 14
  - 2.1.5 Sampling-based methods . . . 15
- 2.2 Model checking and control synthesis . . . 21
  - 2.2.1 Model checking . . . 22
  - 2.2.2 Control synthesis algorithms . . . 22
- 2.3 Related work . . . 26
  - 2.3.1 Optimal control . . . 26
  - 2.3.2 Time constrained specifications . . . 29
  - 2.3.3 Uncertainty in motion and sensing . . . 31
  - 2.3.4 Multi-robot systems . . . 34
- 2.4 Concluding remarks . . . 37

3 Preliminaries **39**
3.1 System models . . . 39
3.2 Linear temporal logic . . . 42
3.3 ω-automata . . . 44
3.4 Co-safe linear temporal logic . . . 46
3.5 Metric interval temporal logic . . . 47
3.6 Timed automata . . . 48

4 Optimal Kinodynamic Motion Planning with Co-safe LTL Specifications **51**
4.1 Problem formulation . . . 52
4.2 Solution . . . 55
4.2.1 Overview . . . 55
4.2.2 Stable sparse RRT . . . 56
4.2.3 Stable sparse RRT with temporal logic constraints . . . 60
4.3 Analysis . . . 65
4.3.1 Probabilistic Completeness and Asymptotic Optimality . . . 65
4.3.2 Complexity . . . 67
4.4 Examples . . . 67
4.5 Concluding remarks . . . 69

5 Stochastic Optimal Control with MITL Specifications **71**
5.1 Problem formulation . . . 72
5.2 Solution . . . 73
5.2.1 Overview . . . 73
5.2.2 Workspace discretisation and local policies . . . 74
5.2.3 BMDP model . . . 78
5.2.4 Product BMDP . . . 79
5.2.5 Optimal global policy computation . . . 80
5.2.6 Policy implementation . . . 81
5.3 Analysis . . . 82
5.4 Example . . . 82
5.4.1 Discussion . . . 83
5.5 Concluding remarks . . . 85

6 Reactive Motion Planning with LTL Constraints and Imperfect State Information **87**
6.1 Problem formulation . . . 88

6.2 Solution . . . 90
6.2.1 Overview . . . 90
6.2.2 Feedback-based information roadmap . . . 91
6.2.3 Incremental transition system . . . 93
6.2.4 Product MDP . . . 95
6.2.5 Optimal policy computation . . . 97
6.2.6 Local targets . . . 98
6.2.7 Obstacle avoidance . . . 100
6.3 Examples . . . 102
6.4 Concluding remarks . . . 105

7 Path Planning for Multi-robot Systems with Co-safe LTL Specifications 107
7.1 Problem formulation . . . 108
7.2 Solution . . . 109
7.2.1 Overview . . . 109
7.2.2 Probabilistic roadmap . . . 109
7.2.3 Composite roadmap exploration . . . 111
7.2.4 Product automaton update . . . 113
7.2.5 Guided exploration . . . 114
7.2.6 Implementation . . . 116
7.3 Examples . . . 117
7.4 Concluding remarks . . . 118

8 Conclusions and Future Work 121
8.1 Summary and conclusions . . . 121
8.2 Future work . . . 124

References 127

List of figures

1.1 Illustration of synthesis of controllers using model checking. 3

2.1 Workspace and configuration space of a circular mobile robot. 10
2.2 Example of visibility graph. 12
2.3 Triangular decomposition. 13
2.4 Obstacle encoded as a combination of linear equality constraints. 15
2.5 Example of probabilistic roadmap. 17
2.6 Illustration of rapidly-exploring random tree. 18
2.7 Expansion of RRT. 19
2.8 Example of automata-based method 23
2.9 Example of an abstraction 24

3.1 Example of transition system. 40
3.2 Trace defined by a run in transition system 41
3.3 Illustration of a Markov decision process and bounded-parameter Markov decision process 43
3.4 Semantics of linear temporal logic. 44
3.5 Büchi automaton. 45
3.6 Rabin automaton. 46
3.7 Timed automaton. 49

4.1 Illustration of two δ-similar trajectories. 53
4.2 Illustration of the small-time locally accessible property. 54
4.3 Illustration of SST. 57
4.4 Selection of state to expand tree. 57
4.5 Illustration of expansion and pruning of tree. 59
4.6 Neighbourhoods defined by witnesses over the state space. 62
4.7 Selection of states to expand transition system. 64
4.8 Growing trees with different atomic propositions. 66

4.9 Path followed by a quadrotor satisfying a LTL specification. 68
4.10 3D view of path followed by a quadrotor satisfying a LTL specification. . . 69
4.11 Convergency of SST_LTL approach. 69

5.1 Partitioning of the workspace. 74
5.2 Illustration of incremental Markov decision process. 75
5.3 Illustration of the continuous-time interpolation of Markov chain. 76
5.4 Computation of policies for transitioning between segments. 77
5.5 Trajectories followed by a Dubin's car constrained by a MITL specification. 83
5.6 3D view of trajectory followed by a system following a MITL specification. 84

6.1 Example of a multivariate Gaussian distribution. 89
6.2 Illustration of a feedback-based information roadmap. 93
6.3 Illustration of expansion of transitions system. 95
6.4 Example of Rabin automaton. 96
6.5 Target detected during online execution. 100
6.6 Obstacle detected during online execution. 101
6.7 Example 1 of trajectory followed by a system with uncertainty in motion and sensing. 103
6.8 Example 2 of trajectory followed by a system with uncertainty in motion and sensing. 104

7.1 Illustration of tensor product. 110
7.2 Selection of vertex and edge using dRRT. 111
7.3 Incremental construction of a transition system. 112
7.4 Illustration of the path followed by two robots satisfying a LTL specification. 117
7.5 Illustration of the path followed by four robots satisfying a LTL specification. 118

List of tables

6.1 Average required time to solve two problems with uncertainty in motion and sensing. 104

7.1 Average required time to solve two multi-robot problems. 118

List of abbreviations

AEC	Accepting end component
BMC	Bounded model checking
BMDP	Bounded-parameter Markov decision process
CE	Cross-entropy
CEGIS	Counterexample-guided inductive synthesis
CTL	Computational tree logic
DRA	Deterministic Rabin automaton
DTA	Deterministic timed automaton
FIRM	Feedback-based information roadmap
GDTL	Gaussian distribution temporal logic
iMDP	incremental Markov decision process
IVI	Interval value iteration
LQG	Linear quadratic Gaussian
LTL	Linear temporal logic
MDP	Markov decision process
MILP	Mixed-integer linear programming
MITL	Metric interval temporal logic
MPC	Model predictive control

MTL	Metric temporal logic
PCTL	Probabilistic computational tree logic
POMDP	Partially observable Markov decision process
PRM	Probabilistic roadmap
RRT	Rapidly-exploring random tree
sc-LTL	Syntactically co-safe linear temporal logic
SST	Stable sparse-RRT
STL	Signal temporal logic

Chapter 1

Introduction

Autonomous mobile robots are machines capable of performing tasks operating without human intervention. Their presence has increased in applications such as personal assistance, security, manufacturing, warehouse distribution, etc. One of the main challenges of controlling a robot autonomously lies in the area of motion planning [31]. In its most basic form, the goal of motion planning consists of finding a plan to guide a system from an initial to a final configuration while avoiding collisions. This planning needs to consider elements such as system dynamics, uncertainties and dynamic environments that are present in most applications. Moreover, as the number of applications involving interaction with humans increases, it is essential to consider factors such as safety and reliability.

The problem of motion planning has been studied for decades [97]. Although numerous algorithms have been proposed, these mainly focus on driving a system from an initial state to a single final state. As the tasks capable to be performed by a robot become more complex, these methods present some limitations. For instance, consider a mobile system used for surveillance. The mission of the system could include the reachability of several areas of a building in a particular order. While methods that allow tasks such as sequencing and timing of targets have been proposed [14, 129], they are limited to certain type of tasks, e.g. only tasks with finite horizon properties. Hence, methods capable of automatically computing plans based on given complex missions or specified behaviours are desirable.

On the other hand, research in an area of computer science, namely formal methods, solves a similar problem. Formal methods have been applied in computer science for task specification and verification of software and hardware. Their aim is to provide a rigorous mathematical approach to establish the correctness of a system. Examples such as the crash of the rocket Ariane 5 in 1996 due to a software error [109] or the faulty Intel's Pentium II processors [35] have demonstrated the necessity of such methods for the verification of software and hardware. These methods have allowed the development of approaches, called

correct-by-construction, that return an implementation that guarantees a desired behaviour based on a specification.

In particular, a formal method, called model checking [34], allows to verify whether a given specification is violated by a system. This verification is performed by exhaustively exploring a model which describes all the possible behaviours of the system. After the verification is completed, model checking algorithms inform whether there exists a behaviour violating or satisfying the specification.

Model checking algorithms have been used for years for software verification. However, it was not until the last decade that they started to be applied in the field of motion planning or, more general, in control. By combining these two areas, methods capable of automatically synthesising plans that provably guarantee the execution of a given specification have been developed. Here, synthesis is referred to as the action of constructing controllers to produce a system's trajectory that meets certain constraints. Because of the flexibility to define a wide range of specifications and the guarantees offered by these algorithms, they have become useful in robotics, especially for safety-critical systems.

In general, these synthesis methods receive as an input a formal specification of the desired system's behaviour and a model of the system. If the model contains a behaviour capable of satisfying the specification, a sequence of states or controllers is returned. Otherwise, it is reported that the specification cannot be satisfied by the model, Fig. 1.1. Specifications are commonly expressed using computational tree logic (CTL) and linear temporal logic (LTL) . These logics allow to unambiguously express useful specifications or behaviours for robotic systems, such as the rules required for the DARPA challenges [191]. Examples of possible specifications include: (i) sequencing, e.g. go to A, then to C and finally B; (ii) recurrence, e.g. go to regions A and B infinitely often; and (iii) safety, e.g. always avoid region C.

Although model checking has been successfully used to verify discrete systems, its application in robotics presents certain challenges:

- **Computation of abstractions.** Since model checking was designed for finite discrete systems, the problem of computing finite models from high-dimensional systems with infinite number of states has been an important area of research. The computation of such models, also called abstractions, must preserve the properties of interest from the original system in order to guarantee the correct behaviour. This problem becomes more challenging when nonlinear or high-dimensional systems, i.e. systems with a large number of variables defining their state, are considered [194].

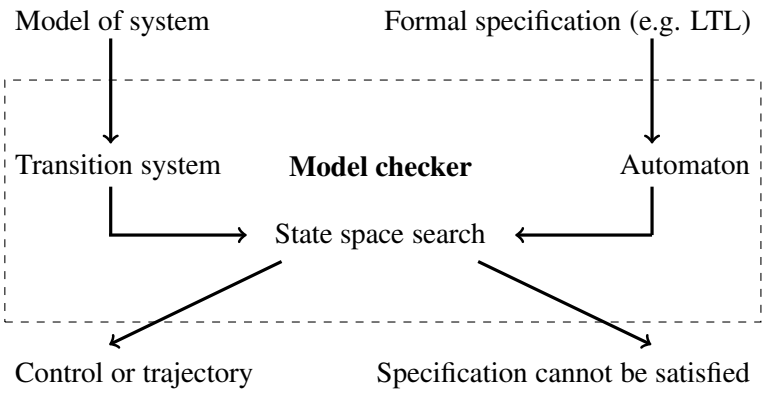


Fig. 1.1 Illustration of a typical approach used to synthesise controllers using model checking. A finite state model of the original system is first computed and modelled as a transition system. The specification is transformed into an automaton. Using approaches such as automaton-based techniques, an attempt at finding a trajectory satisfying the specification is performed. If a solution is found, the sequence of controllers required to follow the trajectory is returned.

- **System dynamics.** Another challenge is the implementation of the computed plan. As mentioned above, model checking techniques perform the verification at a discrete level. Hence, the computed plan consists of a series of transitions between discrete states. In order to perform such plans at the continuous level, controllers must be computed to achieve the transitions of the discrete plan. While this computation does not represent a problem for systems with simple dynamics, for nonlinear systems with more complex dynamics, such as those with kinematic and dynamic constraints, i.e. kinodynamic systems, this computation is not trivial [37].

- **Uncertainties and dynamic environments.** As aforementioned, the uncertainty to which autonomous systems are subjected must also be considered during the motion planning. These elements represent an additional complication to the previous challenges given the complexity of solving stochastic problems or problems with partially observable systems that arise when uncertainties are considered. Moreover, when a system operates in a dynamic environment, the computed plan or control must be reactive to act against changes. In order to guarantee a level of safety or satisfiability of a specification under all these circumstances, robust controllers and reactive controllers that maximise the probability of completing a task are required.

- **Specifications.** One of the main benefits of using model checking for motion planning is the flexibility to define complex tasks in a human-like language using modal

logics. In general, there exists a trade-off between the expressiveness of the logic and the complexity of the algorithm required to use such a logic. For instance, when systems with complex dynamics are considered, finding a trajectory to satisfy tasks with infinite horizon, e.g. visit regions A and B infinitely often, represents a more challenging problem compared to tasks with finite horizon, e.g. visit region A and then B. Another example is the inclusion of time in the specifications. Logics such as metric temporal logic (MTL) [88] permit to express tasks with time constraints. Nevertheless, considering time constraints during the motion planning increases the computational complexity of the problem.

- **Multi-robot systems.** Considering multiple robots brings a series of complications. How robots should collaborate to accomplish a task or what to do when the specifications of two different robots conflict with each other are some of the problems that must be solved for multi-robot systems [189]. Another challenge is the state explosion problem due to the total number of possible behaviours or states, which increases with the number of robots [84].

1.1 Aim and objectives

Motivated by the challenges mentioned above, the aim of this thesis is the development of methods to solve the problem of motion planning for linear and nonlinear systems such that the correct behaviour, based on high-level temporal specifications, is guaranteed. The main focus is on the scalability and applicability to a variety of system dynamics of the methods. Moreover, in order to be applicable to several real situations, the developed frameworks must be able to consider factors such as uncertainties, time constraints, number of robots and dynamic environments. In order to achieve this aim, the following objectives are defined:

- Develop a method to compute optimal plans while temporal logic specifications are satisfied. In order to be applicable for a variety of systems, the method should be able to consider systems with kinematic and dynamic constraints.

- Develop a scalable method to consider real-time constraints. This approach must be able to mitigate the increasing number of states due to the consideration of time in the specification.

- Develop a method to find plans satisfying high-level specifications for systems affected by uncertainty in motion and sensing. For scenarios where the correct be-

haviour cannot be guaranteed, the method should maximise the probability of satisfying the specifications.

- Develop a method to compute plans for systems operating on dynamic environments. The plan must allow the system to react to changes while the temporal logic specification is satisfied.

- Develop a scalable method to find paths for multiple robots satisfying a global or individual specifications. Due to the increasingly number of states with the number of robots, the method needs to efficiently search for paths satisfying the specifications.

1.2 Contributions

The aim and objectives led to the following contributions:

- The problem of optimal motion for kinodynamic systems subject to temporal logic specifications is a challenging problem due to the need to compute high quality trajectories while kinematic and differential constraints are satisfied. A framework capable of finding optimal trajectories for high-dimensional kinodynamic system subject to co-safe linear temporal logic specifications is presented in Chapter 4. In contrast to other methods that can only be applied to limited classes of system dynamics, the proposed approach can be applied to a wider range of dynamics. This is achieved by a method that only requires the forward propagation of the system dynamics while other methods require the computation of abstractions or to solve two-point boundary value problems. Therefore, the proposed method opens the possibility to solve new problems.

- While qualitative properties such as those that can be specified by co-safe LTL [91] (a fragment of LTL) could be sufficient for some tasks, others require specifications with time limits. Considering temporal logic specifications with real-time constraints, in general, requires the discretisation of the time and state space. This discretisation becomes a problem when the dimension of the considered system increases. A method that reduces the complexity of motion planning problems considering real-time constraints, given as a metric interval temporal logic, and uncertainty in motion is presented in Chapter 5. Although this problem has been addressed in the literature, solutions are limited by their scalability due to a fine discretisation required. The presented approach achieves computational tractability by combining a coarse abstraction of the workspace with a sampling-based method that approximates the

properties of the original stochastic system. As a result, the approach can be applied to high-dimensional systems.

- The solution presented in Chapter 5 considers stochastic motion of the system while perfect information about the system state and a static environment are assumed. Nevertheless, in many applications, the state of the system becomes uncertain due to factors such as noisy sensors. Moreover, dynamic environments are also commonly presented in many applications. Planning under motion and sensing uncertainties represents a difficult problem because of complexity provoked by the curse of history associated with these conditions. Because of this complexity, solutions are usually computed offline. Nevertheless, these solutions become invalid when the environment is dynamic. The solution proposed in Chapter 6 allows systems with uncertainty in motion and sensing to react to elements, such as obstacles, in a dynamic environment while a linear temporal logic specification is satisfied. This problem has mainly been solved in the literature at the discrete level. In contrast, the proposed approach finds a solution in the continuous state space. Hence, it imposes less assumptions, e.g. precomputed abstractions.

- With the possibility of expressing complex tasks by using temporal logics, the range of possible tasks can be limited by the capabilities of a single robot. In Chapter 7, an approach for multiple robots subject to co-safe linear temporal logic specifications is presented. A main challenge with multiple robots is the scalability. This problem arises due to the increasing number of possible states with the number of robots. Available solutions do not offer good scalability due to the use of discretisation or navigations functions. The proposed approach extends sampling-based methods for the multi-robot problem and uses a novel algorithm that guides the exploration of the state spaces allowing the reduction of the considered states. As a result, less time is required to find solutions.

1.3 Publications

The proposed methods in this thesis are based on the following author's works:

i. Montana, F. J., Liu J., and Dodd, T. J. (2016). Sampling-based stochastic optimal control with metric interval temporal logic specifications. In Proceedings of Conference on Control Applications, pages 767-773. IEEE.

ii. Montana, F. J., Liu, J. and Dodd, T. J. (2017). Sampling-based reactive motion planning with temporal logic constraints and imperfect state information. In Proceedings of Critical Systems: Formal Methods and Automated Verification, pages 134-149. Springer.

iii. Montana, F. J., Liu, J. and Dodd, T. J. (2017). Sampling-based path planning for multi-robot systems with co-safe linear temporal logic specifications. In Proceedings of Critical Systems: Formal Methods and Automated Verification, pages 150-164. Springer.

1.4 Thesis overview

Chapter 2 presents a background of methods developed to solve the traditional point to point motion planning problem and presents the necessity of including model checking techniques to address the current limitations. Then, the main ideas of control synthesis using model checking techniques are introduced. This section also discusses different developed approaches and the main gaps and possibles areas for improvement. Finally, an analysis of the work more related to the methods in this thesis is presented. This related work is divided into four sections, one for each of the main chapters, and discusses the problems of optimal control, time constraints, uncertainty in motion and sensing and multi-robot systems.

Chapter 3 introduces linear temporal logic and metric interval temporal logic which are used in the rest of the chapters to specify the desired behaviour of the systems. This chapter also presents the concepts of ω-automata and timed automata used to verify whether the behaviour of a system satisfies a specification. Finally, the graph structures used to model deterministic and stochastic systems are also described in this chapter.

Chapter 4 presents a solution to the optimal motion planning problem subject to temporal logic specifications. The solution is geared towards high-dimensional kinodynamic systems. This is based on new approaches that only require the forward propagation of the system dynamics to find optimal trajectories. The presented method generalises these methods, allowing to find asymptotically optimal trajectories satisfying co-safe linear temporal logic specifications. The method iteratively expands a transition system, where the states are formed by a state of the system and a state of a Büchi automaton while transitions represent trajectories of the system. At each iteration, a new state and transition are added after verifying if the trajectory is valid in the Büchi automaton. The process continues until a specified number of iterations is completed.

Chapter 5 addresses the problem of computing controllers for stochastic systems under metric interval temporal logic specifications. The presented approach reduces the computational complexity of discretising the state space and time by dividing the solution into two phases. During the first phase, the system is coarsely discretised into distinct regions

and local policies are computed using a sampling method. In the second phase, a Cartesian product between the abstraction and a timed automaton is used to compute an optimal global policy that maximises the probability of satisfying the specification. The number of states in this product is reduced thanks to the coarse discretisation of the workspace instead of the state space as in other methods. Hence, the approach is less sensitive to the dimension of the system. The work in this chapter is based on the author's work [124].

Chapter 6 considers problems where systems with uncertainty in motion and sensing operate in dynamic environments. To reduce the complexity of the curse of history caused by the aforementioned uncertainties, the approach presented in this chapter uses feedback-based information roadmaps to create a transition system. Based on results in probabilistic model checking, an optimal solution to the problem is found by constructing a product Markov decision process with the transition system and a Rabin automaton. In order to permit a fast reaction to the environment, another feedback-based information roadmap is computed offline and used during the online execution. The work in this chapter is based on the author's work [126].

Chapter 7 focuses on motion planning for multiple robots. Specifically, a global specification, given as a co-safe linear temporal logic, that must be satisfied by a team of robots is considered. The proposed method explores the state space of the robots. During the exploration, a transition system is incrementally expanded by adding new states from individual roadmaps. With each expansion, the product automaton of the transition system and a Büchi automaton is updated. This process continues until a solution is found. To reduce the number of states in the product automaton, and therefore reduce the required time to find a solution, an algorithm that uses the Büchi automaton of the specification to guide the expansion of the transition system is presented. The work in this chapter is based on the author's work [125].

Chapter 8 presents a summary of the contributions and its conclusions. This chapter also presents possible future directions for the developed work.

Chapter 2

Background and Related Work

This chapter presents the background to the motion planning problem and a discussion of methods that combine motion planning with model checking techniques. In the first section, an introduction to the problem of motion planning is described. Then, some of the available techniques to solve the problem of motion planning are presented. Particular emphasis is made on the section of sampling-based methods which serves as a background for the methods proposed in this thesis. In the second section, the necessity of methods that combine model checking and motion planning to solve problems such as reliability and the limited applicability of traditional motion planning methods for complex tasks is presented. Moreover, this section presents a background of methods using model checking techniques and motion planning by describing some of the most commonly used approaches. Finally, an analysis of the state-of-the-art methods more related to those presented in this thesis and their limitations is presented in the last section.

2.1 Motion planning

A fundamental and challenging problem in the area of robotics is the task of motion planning [101]. The problem of motion planning has been studied extensively for decades. As a result, developed algorithms have spread to diverse areas such as robotics, manufacturing, computer animation, medical applications, etc. [98]. In a broad sense, the goal of motion planning algorithms is to find a path for a robot from an initial to a final configuration without colliding with obstacles.

Initial algorithms only considered the problem from the geometrical point of view [31]. In other words, the algorithm only requires to find a series of configurations between the initial and final ones assuming that the motion between them is realisable. Since then, motion planning has evolved to consider differential constraints such as velocity and acceleration

[43]. When these constraints are considered, motion planning algorithms need to validate the transition from one configuration to another during the planning.

Before formally defining the problem of motion planning, the concept of configuration space [31] must be introduced. In order to avoid collisions, the physical space occupied by a robot in the environment in which is operating, called workspace, must be known. A representation of the robot that permits to obtain this information is called the configuration of the robot and is represented as q. An example of a configuration is the vector of six parameters, three defining position and three orientation, for a vehicle operating in a three-dimensional physical space. This vector contains all the information required to know the space occupied by the robot in its workspace and hence to detect if the vehicle is colliding with an obstacle. The set of all possible configurations is called the configuration space ($\mathcal{C}$-space). The minimum number of parameters required to define a configuration is called the degrees of freedom. Therefore, the dimension of the configuration space is determined by the number of degrees of freedom of the robot.

The problem of motion planning requires to find a path from an initial configuration q_{init} to a final configuration q_{final}. Formally, a path is a function $\boldsymbol{q} : [0, \tau] \to \mathcal{C}$, where τ is the final time. In order to find a path without collisions, the free configuration space and the obstacle configuration space are computed. Hence, the position of the obstacles must be known in advance. The free configuration space $\mathcal{C}_{\text{free}}$ is the set of points from which the robot does not intersect any obstacle in the workspace. On the other hand, the obstacle configuration space $\mathcal{C}_{\text{obs}}$ is defined by the set of points where a collision occurs, i.e. $\mathcal{C}_{\text{obs}} = \mathcal{C} \setminus \mathcal{C}_{\text{free}}$, Fig. 2.1.

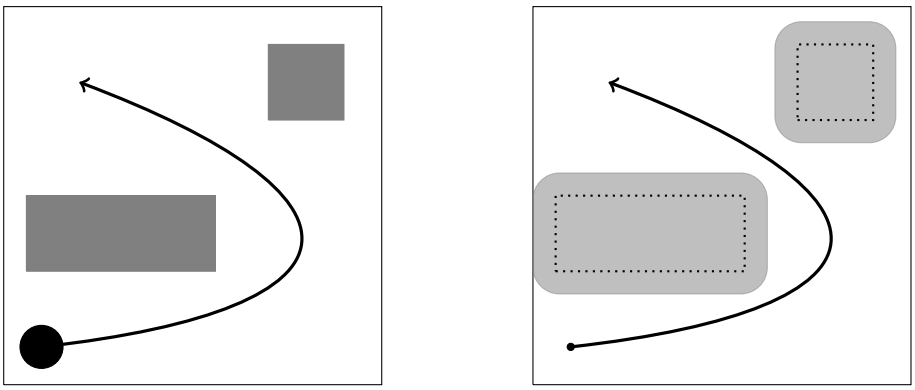


Fig. 2.1 Workspace and configuration space of a circular mobile robot. The figure on the left shows a workspace with two grey obstacles, the initial position of the robot and a collision-free trajectory followed by it. The figure on the right shows the free configuration space (white region), the obstacle configuration space (light grey regions) and the robot represented as a point.

A formal definition of the motion planning problem, from the geometrical point of view, is now presented.

Problem definition 2.1.1. *Given an initial and a final configuration, q_{init} and q_{final}, find a collision-free path $\boldsymbol{q}$ such that $\boldsymbol{q}(0) = q_{init}$, $\boldsymbol{q}(\tau) = q_{final}$ and for every $t \in [0, \tau]$, $\boldsymbol{q}(t) \in \mathcal{C}_{free}$.*

When the motion of mobile robots is constrained by elements such as friction, velocities, etc, motion planners must be able to solve the problem of motion planning subject to kinematic and differential constraints. This problem is referred to as kinodynamic motion planning [43]. In this type of problems, the configuration q is not enough to represent the system at a certain moment. Instead, the system is represented by a state x which not only defines the position and orientation but also defines other variables such as velocities at a given instant of time. Similar to the configuration space, a state space X is the set of all possible configurations and other system variables that represent the state of the system. To express the constraints and evolution of the state, a system can be modelled by a function:

$$\dot{x} = f(x, u), \tag{2.1}$$

in which $x \in X$ is the state, $u \in U$ is the control input and U is the set of all possible inputs. The set U is also called the control space.

The kinodynamic motion planning problem requires to find a trajectory $\boldsymbol{x} : [0, \tau] \to X$, where τ is the duration. Formally, the kinodynamic motion planning problem can be defined as follows.

Problem definition 2.1.2. *Given an initial and a final state, x_{init} and x_{final}, find a collision-free trajectory $\boldsymbol{x}$ such that $\boldsymbol{x}(0) = x_{init}$, $\boldsymbol{x}(\tau) = x_{final}$ and for every $t \in [0, \tau]$, $\boldsymbol{x}(t) \in X_{free}$ and Eq. 2.1 is satisfied, where X_{free} is the set of states where the system does not violate any constraint.*

For kinodynamic motion planning problems, the search for a trajectory is performed over the state space X, which is usually bigger than the configuration space $\mathcal{C}$ by a factor of two [102].

Different techniques have been developed to compute collision-free paths for a variety of scenarios. In the following subsections, several of these approaches are presented. More specifically, approaches based on roadmaps, cell decomposition, methods of potential fields, optimal control and sampling-based methods are discussed.

2.1.1 Roadmaps

In several situations, it is required to solve several motion planning problems on the same workspace. For these cases, creating a reusable structure, called roadmap, that can be used to find a path between two configuration is convenient. Methods such as visibility graphs [115] and generalised Voronoi diagrams [128] compute such structures.

Visibility graphs, Fig. 2.2, are defined over polygonal configuration spaces. The nodes of the graph include the vertices of the obstacles and the initial and final configurations of the system. Two nodes are connected by an edge if the points in the straight line between them lie on the free configuration spaces. Although visibility graphs return the optimal path for system without dynamic constraints, since the path passes arbitrary close to obstacles, it cannot be safely used when uncertainty in position exists.

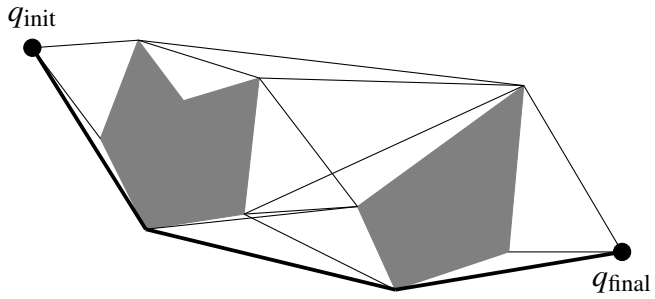


Fig. 2.2 Example of visibility graph defined by the thin lines. The edges of the graph connect vertices of the obstacles, the initial configuration q_{init} and final configuration q_{final}. The thick line shows the shortest path between configurations.

Generalised Voronoi diagrams (GVD) take a different approach since it is defined by the set of points where the distance between the two closest obstacles are the same. In other words, the diagram follows the maximum distance between two obstacles. Since the distance to obstacles is required to construct the diagram, a robot equipped with a range sensor can be employed to incrementally construct a diagram of an unknown workspace. Once a GVD is constructed, path planning is achieved by moving away from obstacles until the diagram is reached.

2.1.2 Cell decomposition

When coverage of the configuration free space is required instead of moving from one initial configuration to a final one, methods based on cell decomposition are used to achieve this task [30]. In this approach, the free configuration space is presented by a union of regions

called cells. Once the decomposition is computed, a graph can be constructed where the cells and their adjacency are represented by nodes and edges, respectively. Examples of cell decomposition include vertical decomposition, triangulation, Morse decomposition and cylindrical algebraic decomposition.

To find a path from one configuration to another, a planer searches for a path in the graph connecting the nodes containing the initial and final configurations. Finally, a path in the configuration space is found by connecting the centroids of the cells to the midpoint of their boundaries, Fig. 2.3. When coverage is required, the system first explores the whole cell before moving to the adjacent one. Similar to the GVDs, the approach is not optimal.

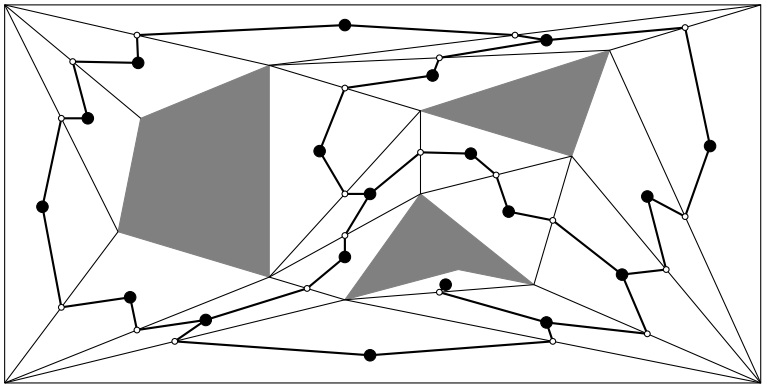


Fig. 2.3 Example of triangulation and adjacency graph. Each segment of the triangulation is represented by a node (black dots). The edges between nodes pass through the midpoints of the segment's boundary and define a collision-free path from one segment to another.

2.1.3 Method of potential fields

To avoid the explicit computation of configuration free spaces, planners such as potential fields has been proposed for single [69, 172], and multi-robot systems [10, 39]. This approach permits to incrementally explore the free configuration space. A potential function is a differentiable real-valued function. Its gradient defines a vector field over the configuration space of the system. If one considers the potential function as energy, the gradient can be seen as negative and positive forces acting on the robot. Hence, obstacles and targets can be made repulsive and attractive, respectively. On this vector field, the robot moves until a point with a gradient equal to zero is reached.

A drawback of this method is that it presents the problem of local minima [31]. In other words, the robot can be led to a point that does not correspond to the desired final destination. Solutions such as virtual obstacles [105], navigation functions with only one local minima

[149] and simulated annealing [199], can be used to avoid this problem. Another limitation of the potential function approach is the scalability. The vector defined at each point of the configuration space by the vector filed is a function of the distance to obstacles. To compute this distance, methods discretise the configuration space into cells. Such a discretisation presents poor scalability with the dimensionality of the configuration space.

2.1.4 Optimisation-based methods

A different type of approaches used to solve the problem of motion planning are optimisation-based methods. These approaches are rooted in calculus of variations and its application in optimal control [141]. One of the first works to consider the problem of motion planning as a variational problem is [195]. In this work, a discretisation of the continuous problem and techniques for nonlinear programming are used to solve the variational problem. In [51], the negative formulation of the Pontryagin maximum principle is used to find optimal trajectories of non-holonomic systems. Calculus of variations has also been applied to find trajectories for cooperating non-holonomic systems [38].

A more recent work considers a covariant gradient descent approach [200]. This work proposes trajectory costs that are invariant to time parameterisation of the trajectory. To account for obstacles, the method uses a signed distance field representation of the environment and minimises a combination of obstacles cost and smoothness of the trajectory. The functional gradient optimisation, as other optimisation methods for non-convex objectives, finds local minimum. To improve the returned solution the approach uses a Hamiltonian Monte Carlo method. To avoid the local minimum problem, other approaches use stochastic trajectory optimisation methods that rely on an initial trajectory to explore the state space [72].

The set of configurations where a system does not collide is highly non-convex for most robotic systems [154]. To handle this non-convexity, several approaches have used sequential convex optimisation to repeatedly construct convex subproblems [96, 154]. In [96], the system is parameterised in time by using a B-spline representation. Obstacle avoidance and constraints are included in a parametric nonlinear optimisation problem. In [154], signed distances using convex-convex collision detection are directly integrated into the convex optimisation problem to avoid obstacles collision.

Another common approach to find optimal solutions and to avoid obstacles is to reformulate the motion planning problem as a linear program subject to mixed integer constraint, known as mixed-integer linear program (MILP) [147]. In this approach, obstacles are represented by polygonal convex region and encoded as a combination of linear constraints, Fig. 2.4. To avoid the obstacles, an and-constraint is created by introducing binary vari-

ables. Then, a MILP problem is solved by combining the mixed integer constraints with the system constraints. For long trajectories, this approach can be implemented in a receding horizon fashion to reduce the complexity of the problem [13]. This approach has also been extended to consider more complex problems such as multiple systems [153] and systems with uncertainty in motion [21].

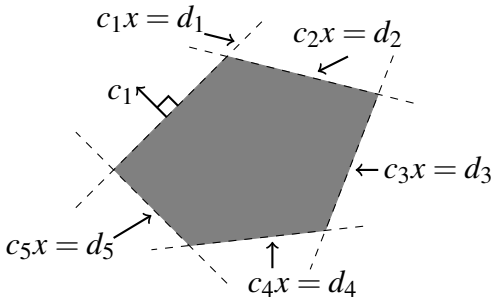


Fig. 2.4 Obstacle encoded as a combination of linear equality constraints. The variables x, c_i and d_i represent the system state, a vector of appropriate dimension and a real number, respectively. (Adapted from [21])

In general, the problem of optimal motion planning for kinodynamic systems is a challenging problem. Since for several problems, an analytical solution is intractable or not have a closed-form solution, methods rely on optimisation-based methods as presented in this section. On the other hand, when the minimised (maximised) objective function or the set of constraints induces a non-convex region, these approaches relax the optimality to locally optimal solutions.

On the other hand, several of the methods presented in the previous subsections use an explicit computation of the obstacle space $\mathcal{C}_{\text{obs}}$. Therefore, they have poor scalability for high-dimensional spaces [77]. This limitation motivated the development of sampling-based methods. In the next subsection, the principles used by these methods and some common algorithms are analysed.

2.1.5 Sampling-based methods

The main idea of sampling-based methods is to avoid the explicit construction of $\mathcal{C}_{\text{obs}}$, which is a computationally difficult task for systems with a large number of degrees of freedom. Instead, these methods create a graph by randomly sampling configurations from $\mathcal{C}_{\text{free}}$ and connecting them if no collision exists. Once enough samples are added to the graph, a path connecting the initial and the final configuration is sought on it, Fig. 2.5.

By avoiding the explicit computation of $\mathcal{C}_{\text{obs}}$, sampling-based methods have been successfully used to solve problems with high-dimensional configuration spaces [70]. However, the scalability of these methods is gained at the cost of completeness. An algorithm is said to be complete if the returning of an answer, if one exists, is guaranteed. Otherwise, the algorithm must report failure. Sampling-based methods provide a weaker notion of completeness: probabilistic completeness, i.e. the probability of finding a solution, if one exists, converges to one when the number of samples tends to infinite.

As aforementioned, the methods presented in this thesis are based on sampling-based methods. Specifically, probabilistic roadmaps (PRMs) [81] and rapidly-exploring random trees (RRTs) [100] are used. These two methods are now analysed.

Probabilistic roadmaps. PRMs are an example of multi-query planners. That is, once a PRM is constructed, it can be used to solve several problems in the same workspace. The main benefit is the possibility of a fast path computation between multiple initial and final configurations. Nevertheless, the initial construction of the roadmap can be a slow process.

The method consists of two phases. In the first phase, the learning phase, an undirected graph $G = (V, E)$ is constructed, where V is a set of configurations in $\mathcal{C}_{\text{free}}$ and E is a set of edges connecting elements of V. Initially, G is empty. To add a configuration to this graph, the configuration space $\mathcal{C}$ is randomly sampled. If a sampled configuration q is collision-free, i.e. $q \in \mathcal{C}_{\text{free}}$, q is added to the set V. Otherwise, the configuration is discarded. This process is repeated until a predefined number, N, of configurations are added, i.e. $|V| = N$, where $|V|$ indicates the cardinality of the set V. Once all the configurations are added to V, each of the configurations $q \in V$ is connected to other configurations as follows. First, a set Q_q with the k-closest configurations in V to a given configuration q is computed. The closeness of the configurations is determined by some metric such as Euclidean distance. Then, a local planner verifies whether the motion between configuration q to each element of the set Q_q satisfies all the constraints, e.g. no collisions. When the motion between configurations is valid, an edge between these configurations is added. Algorithm 2.1 shows the procedure described above.

In the second phase, called query phase, the initial and final configurations, q_{init} and q_{final}, are connected to the graph G. Then, a graph search is performed to return a sequence of configurations connecting q_{init} to q_{final}, Fig. 2.5.

As presented above, a local planner is used to connect two different configurations during the construction of a PRM. Since the process of connecting two configurations is constantly repeated, the selection of the local planner plays an important role in this method. In most cases, speed is preferred over the completeness of the planner. A common approach is to use a quick planner that connects the configurations by a straight-line motion in $\mathcal{C}$ [176].

Algorithm 2.1. PRMCONSTRUCTION($\mathcal{C}, N, k$)

$V \leftarrow \emptyset, E \leftarrow \emptyset$;
while $|V| < N$ **do**
    $q \leftarrow$ SAMPLE($\mathcal{C}$);
    if COLLISIONFREE(q) **then**
        $V \leftarrow V \cup \{q\}$;
for each $q \in V$ **do**
    $Q_q \leftarrow$ NEAR(q, V, k);
    for each $q' \in Q_q$ **do**
        if COLLISIONFREE((q, q')) **then**
            $E \leftarrow E \cup \{(q, q')\}$;
return $G = (V, E)$;

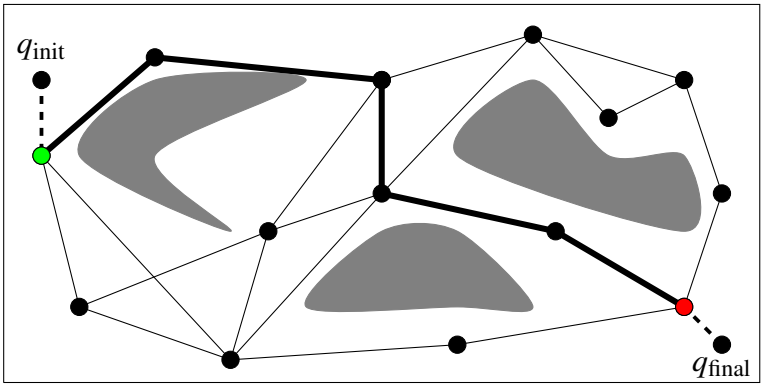


Fig. 2.5 Example of path planning using a probabilistic roadmap. The initial and final configurations q_{init} and q_{final} are connected to the PRM through the green and red configurations, respectively. The shortest path between these configurations is found using graph search algorithms.

Nevertheless, this approach only works for problems not affected by differential constraints as presented later in this section.

Another important aspect of PRMs, and in general for sampling-based methods, is the detection of whether a configuration produces a collision. There exist available libraries to perform this task, e.g. SOLID [177], RAPID [60] and V-Clip [122]. Most of these libraries are implemented using algorithms such as the Gilbert-Johnson-Keerthi algorithm [54] which measures the distance between two convex sets. Although the techniques to detect collisions have been improved, this is the task where most time is spent on by sampling-based methods. Approaches such as lazy-PRM* [63] and safety certificates [19] have been proposed to reduce the number of collision checks required.

Recall that sampling-based methods can only offer probabilistic completeness. This completeness has been proven for PRMs [80]. Moreover, a variant of this approach, called PRM*, can achieve asymptotic optimality [79], i.e. the returned solution approaches the optimal one as the number of samples tends to infinite. PRMs have also been adapted to included factors such as systems with uncertainty [2, 143] and dynamic environments [73, 174].

Rapidly-exploring random trees. As aforementioned, PRMs are used for multiple queries. When only a single problem needs to be solved, the RRT approach provides a better performance. RRT approaches are geared towards quickly finding a path from an initial to a final configuration. However, in contrast to PRMs, the algorithm needs to run again for any change in the initial configuration.

RRT methods construct a tree $T = (V, E)$, where V and E are defined as in the PRMs. As opposed to a graph, in a tree, two vertices are connected by exactly one edge. The tree, rooted at the initial configuration q_{init}, is iteratively expanded until the final configuration q_{final} is reached, Fig 2.6.

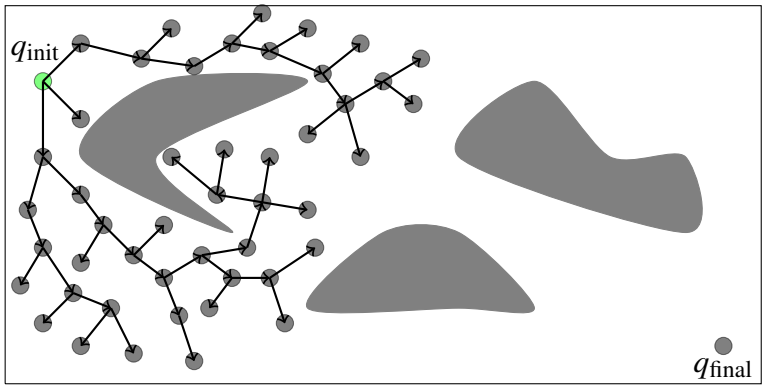


Fig. 2.6 Illustration of path planning using a rapidly-exploring random tree. The tree T is rooted at the initial configuration q_{init} and expanded by adding configurations randomly sampled from the configuration space. The expansion continues until the tree is connected to the final configuration q_{final}.

Initially, the tree only contains the initial configuration q_{init}. Then, to expand T, the following procedure is repeated. A configuration q_{rand} is randomly sampled from the configuration space. Then, a search for the closest configuration q_{near} in the tree to q_{rand} is performed. This search is based on a chosen metric. A planner attempts to drive the system towards the sampled configuration q_{rand} from q_{near}. In contrast to PRMs, an exact connection between configurations is not required. Instead, the final configuration of the motion computed towards q_{rand}, called q_{new}, is considered as a candidate to be added to the tree. If

the path from q_{near} to q_{new} is collision free, q_{new} is added to the set V and the path is added as an edge to E. Otherwise, the configuration is discarded, Fig. 2.7.

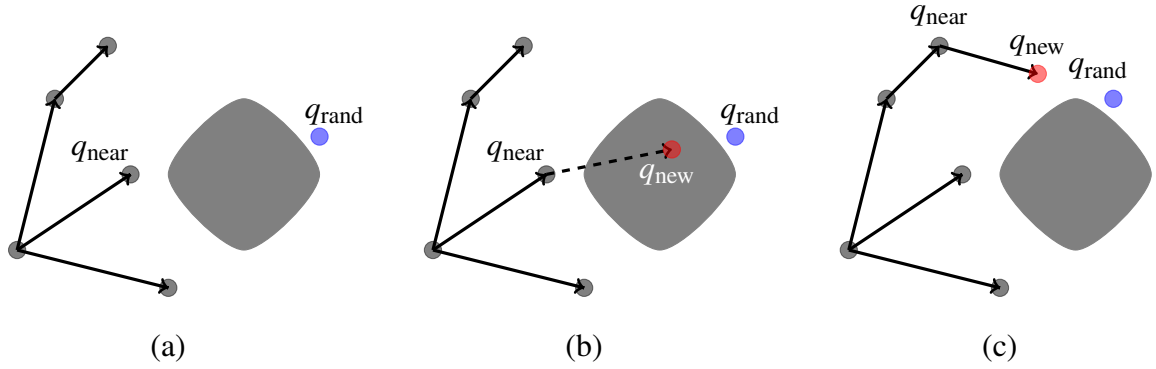


Fig. 2.7 Expansion of RRT T. (a) A configuration q_{rand} is randomly sampled from the configuration space and the closest configuration q_{near} in T is found. (b) A path from q_{near} in the direction towards q_{rand} is computed and checked for collision. Since the path from q_{near} to the last state of the path q_{new} collides with an obstacle, q_{new} is discarded. (c) A new configuration q_{rand} is sampled and the closest configuration q_{near} in T is found. A new path from q_{near} towards q_{rand} is computed. As a result of this new collision-free path, q_{new} is added to T with the edge $(q_{\text{near}}, q_{\text{new}})$.

Configurations are added to the tree until the final configuration is reached or another condition is satisfied, e.g. a minimum number of samples is added to the tree. Then, a path from the initial to the final configuration can be extracted from the tree by backwardly following the edges, starting from the final configuration. Algorithm 2.2 presents the procedure to construct an RRT, described above.

Algorithm 2.2. RRT$(\mathcal{C}, N, q_0)$

$V \leftarrow \{q_0\}, E \leftarrow \emptyset$;
for $i = 1$ to N **do**
  $q_{\text{rand}} \leftarrow$ SAMPLE$(\mathcal{C})$;
  $q_{\text{near}} \leftarrow$ NEAR(q_{rand}, V);
  $q_{\text{new}} \leftarrow$ EXTEND$(q_{\text{near}}, q_{\text{rand}})$;
  if COLLISIONFREE$((q_{\text{near}}, q_{\text{new}}))$ **then**
    $V \leftarrow \{q_{\text{new}}\}$;
    $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$;
return $T = (V, E)$;

Similar to PRM, the probabilistic completeness of RRTs and the asymptotic optimality of a modified version have been proved [77, 102]. Due to the success of RRTs in solving

difficult problems, they have been extended to include complex dynamics [108, 183], stability requirements [90, 167], uncertainty in motion [2, 116] and uncertainty in the state of the system [23, 173], among other aspects.

From the above procedure, it can be seen that the metric chosen to find the nearest configuration affects the expansion of the trees. The selection of the metric is not trivial. This selection is especially difficult for the motion planning of systems affected by differential constraints. Indeed, computing the ideal metric capable of reflecting the cost to go from one configuration or state to another is as hard as the original problem [102]. Examples of metrics include weighted Euclidean distance [6], pseudo-metrics based on the cost to go using linearisation [56, 134] and offline learning using regression models [107, 131].

The sampling strategy is another variable in sampling-based methods. Sampling strategies such as uniform sampling, quasi-random sampling [22] or sampling close to obstacles [160], have been proposed. Nevertheless, the relationship between the planner performance and the sampling strategy has not been proven [53]. A different approach consists of guiding the sampling by using information from the workspace [138] or information obtained during the sampling process [159].

Although PRMs and RRTs can be used for kinodynamic systems, PRMs could present some limitations for systems with significant constraints on the dynamics. This limitation is caused by the exact connection between states required in PRMs. In contrast, the steering function or planner in RRTs does not require to generate an exact trajectory from one state to another. Instead, only a trajectory towards a sampled state suffices to add a new state.

In general, steering methods can be classified into analytical, state-based and control-based steering functions [24]. In analytical steering, as the name indicates, the analytical trajectory with respect to the differential constraints is computed. Although a perfect steering is achieved, it cannot be applied to many kinodynamic systems due to the differential constraints. For the control-based steering, a control is selected from the control space or a set of primitives and a trajectory is computed by forward propagating the system. While this type of steering is easy to compute, it does not provide control over the destination of the system. The state-based steering interpolates a trajectory and computes a control that tracks it.

Due to the benefits presented above, sampling-based techniques have become one of the main approaches to solve motion planning problems [148]. Nevertheless, these methods are focused on finding trajectories from an initial state to a goal region. With the increase of capabilities and automation of robots, this focus could present some limitations. In the next section, the idea of motion planning based on complex missions is presented.

2.2 Model checking and control synthesis

The increasing capabilities of autonomous systems have opened new possibilities and applications for their use. Therefore, a natural question is how can the available motion planning techniques be used to develop methods capable of solving the problem of motion planning such that a complex task is satisfied? For instance, consider a robot surveying a building with several areas of interest. Traditional methods, such as those presented in the previous section, are not capable of computing a trajectory where the robot needs to visit these areas infinitely often in a particular order. In other words, traditional methods cannot be used for tasks with temporal requirements.

An important aspect to extend the applicability of motion planning methods is the ability to formally define the desired tasks or specifications. As presented in Chapter 1, formal methods have been used to verify the behaviour of systems given a formal specification. To express the desired specification, logics such as computational tree and linear temporal logic are commonly used. For example, consider the specification $\varphi = \square(\pi_1 \rightarrow \Diamond\pi_2)$, given as a linear temporal logic. The symbols π_1 and π_2 represent two different regions in a two dimensional space, e.g. π_1 is the set formed by the inequalities $2 \leq x_1 \leq 3$ and $4 \leq x_2 \leq 5$. The symbols $\square$ and $\Diamond$ represent the operators *always* and *eventually*, respectively. Intuitively, the specification φ indicates that every time the system reaches the region defined by π_1, the system needs to eventually reach the region π_2.

Although these logics can be used to unambiguously express tasks for mobile systems, the idea of computing controllers or planning the continuous behaviour of the system based on a discrete or logical specification represents a challenge in the sense that the relationship between these two elements must be understood [165]. To illustrate this problem, consider a system modelled as in Eq. 2.1 and the specification presented above. In order to find a plan to satisfy the specification, a framework capable of solving the logic of the specification and the control problem to drive the system is required.

Motivated by the challenge described above, recently there has been an increasing interest in combining control theory and model checking techniques to automatically synthesise controllers for dynamic systems such that high-level specifications are satisfied [15, 89, 166]. By combining these two fields, researchers have developed algorithms for the synthesis of controllers that ensure the correct behaviour of the system while complex tasks are performed. In the next subsection, the idea of model checking is introduced.

2.2.1 Model checking

Model checking has been used successfully in computer science to verify whether a system satisfies a specification. To perform such a verification, a model of the original system is first created. This model is usually represented by a structure similar to a graph where vertices define states of the system and edges represent the ability of the system to change from one state to another. Once the model is computed, this is exhaustively searched to find a behaviour satisfying or violating a specification. This verification has been typically performed to guarantee safety requirements in software and hardware.

To unambiguously define the desired behaviour of the system, logics such as μ-calculus [142], Linear Temporal Logic (LTL) [139] and Computation Tree Logic (CTL) [33] are commonly used. These logics are built over a set of atomic propositions, which are variables that evaluate true whenever a property of the system is satisfied. For example, in the problem specification presented at the beginning of the section, the symbols π_1 and π_2 are atomic propositions that evaluate true if the system is within the regions defined by the inequalities. Otherwise, these are false. By using such logics, a variety of specifications can be rigorously defined. A formal presentation of these elements is presented in Chapter 3.

Once the model of the system is searched, model checking algorithms can return a series of states that represent a behaviour of the system violating or satisfying a specification. This ability of interpreting the logic of a specification, the ability of verifying the behaviour of a system and the flexibility of defining complex specifications in a human-like language motivated the development of methods that combine model checking techniques and control theory. These methods synthesise or compute controllers given a model of the system and a formal specification. A general background of these methods is presented in the next subsection.

2.2.2 Control synthesis algorithms

Control synthesis algorithms use ideas of model checking to synthesise controllers such that the behaviour of a system is guaranteed to follow a given specification. Because of the guarantees offered by these methods, they have been applied in different control problems including the motion planning of robots. In particular, these are especially important for safety-critical systems.

The first element required for methods that synthesise controllers is a model of the system. Several approaches, some of which are presented later in this section, have been used to compute such models. In general, the form of the computed model depends on the technique used to find a plan to satisfy the specification. As in model checking techniques, a

common model of the system is a graph structure, called a transition system, which models the possible states and transitions of the system.

Recall that atomic propositions are used to identify properties of interest of the system. To identify which property is satisfied at each state, the states of the transition system are labelled with the atomic propositions satisfied by each state, Fig. 2.8. The second element is the specification which can be defined using one of the logics mentioned above. These logics use the atomic propositions, logical and temporal operators to formally state the desired behaviour of a system.

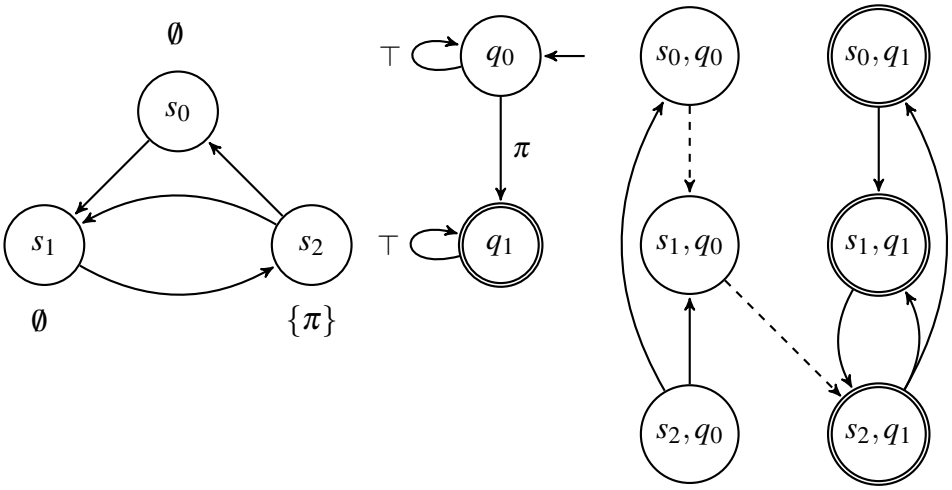


Fig. 2.8 Example of automata-based method. (a) The transition system $\mathcal{T}$ models a system with three states. Each of these states is labelled with the set of atomic propositions satisfied by the state. (b) The automaton $\mathcal{A}$ is obtained from a specification that indicates that the atomic proposition π must be satisfied. (c) The product automaton $\mathcal{P}$ is constructed by computing the Cartesian product of $\mathcal{T}$ and $\mathcal{A}$. The dashed edges show the sequence of states required to satisfy the specification.

Given the model of the system and the specification, approaches such as those based on automata can be used to verify whether the model of the system contains a behaviour satisfying the specification. These approaches rely on the computation of finite automata from a specification. This automaton captures the conditions required to satisfy the specification. Given the transition system $\mathcal{T}$ modelling the system and the automaton $\mathcal{A}$ from the specification, automata-based techniques find a series of states satisfying the specification by computing the Cartesian product of $\mathcal{T}$ and $\mathcal{A}$, Fig. 2.8. On this Cartesian product or product automaton, the algorithm searches for a path satisfying certain conditions, presented in the next chapter. If such a path exists, this is used as a discrete plan, which contains the

sequence of states that the system must follow to satisfy the specification. Finally, in order to perform the transitions in the discrete plan, the algorithm must compute a sequence of control inputs to drive the system from one state to another as indicated by the plan.

Two important aspects to consider in these methods is the computation of the model and the feasibility of the discrete plan, i.e. the system must be able to perform the transitions in the plan. A system can be modelled directly using a transition system. Nevertheless, this model could have an infinite number of states. In order to obtain a finite model, an abstraction or symbolic model is commonly created. This abstraction preserves all the desired properties of the original system and ignores the properties that do not affect the result. Intuitively, an abstraction groups a large or infinite number of states of the original system in such a manner that they are equivalent or share a property of the system, Fig. 2.9. The equivalence between the abstraction and the original system is based on the notion of (bi)simulation [5].

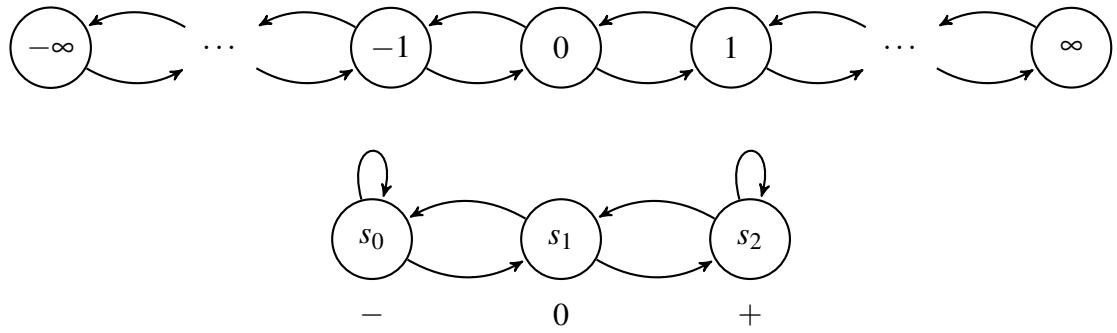


Fig. 2.9 Example of an abstraction (below) of the transition system (above) with a set of states formed by the integer numbers. The abstraction has three states, each of these states groups states from the original system by the sign of the number. (Adapted from [16])

Mainly, there are two different methods to compute such abstractions. The first method requires the discretisation of the state and control space [111, 112, 140, 146]. The second method creates a partition of the state space in such a way that the states on a particular region share the same property of interest [58, 130, 192]. While the latter approach yields abstractions with a smaller number of states, the former allows the use of metrics to measure the fidelity of the abstraction [112].

An advantage of methods using such an abstraction over other methods presented in the rest of this section is that solutions are robust due to the feedback controllers obtained in the abstraction. Nevertheless, they have several limitations. Because abstraction-based methods rely on the discretisation of the control and continuous state space, they tend to scale poorly with the dimension of the system. Moreover, the construction of the abstraction relies on solving reachability problems which can be computationally expensive [186]. To alleviate

this problem, some methods consider an initial coarse abstraction that is then refined based on a discrete plan [59, 186].

Another limitation of the abstraction is that the optimality of these methods is not guaranteed with respect to the original system [48]. Finally, these methods are, in general, not complete since the behaviours depend on the method used for the abstraction [76].

To avoid the aforementioned limitations, other approaches use optimisation methods such as linear programming. These methods encode the specifications as mixed-integer linear constraints. Then, a MILP problem is solved to find a sequence of control inputs such that the system satisfies the specification. Although these approaches have been applied successfully for systems such as mixed logical dynamical [17] and flat systems [46] considering different logics, e.g. signal temporal logic [45, 144, 145], LTL [79] or fragments of LTL [184], some limitations exist. The main drawback is the sensitivity of the method to the size of the specification. That is, the number of integer constraints increases with the size of the specification causing the complexity to grow exponentially due to the complexity of solving the MILP problem, i.e. MILP problems are NP-hard. Although this problem can be alleviated by generating constraints as required [151], the MILP approach has only been applied to systems belonging to the class of systems described above.

A different type of approach, geared towards high-dimensional systems, uses sampling-based methods, such as PRM and RRT, to incrementally create a transition system modelling dynamically feasible trajectories of the system [18, 76, 137, 180]. As presented in Section 2.1.5, a transition system can be created by sampling the state space of the system and computing controllers to drive the system from one state to another. The construction of the transition system is combined with incremental model checking methods to verify whether the current model contains a trajectory satisfying the specification. As with the previous approaches, these methods also present some limitations. In contrast to abstraction-based methods, the solution returned by sampling-based methods is typically an open-loop trajectory. Hence, they are not robust. Another limitation is the probabilistic completeness described in Section 2.1.5. However, in practice, probabilistic completeness suffices in many situations. Moreover, feedback controllers can be used to track open loop trajectories.

At this point, an overview of some of the most commonly used approaches in the literature have been presented. Nevertheless, the application of such approaches changes depending on the conditions considered. For example, elements such as stochasticity, number of robots or the type of specifications affect the method required to solve the problem. In the next section, the work more related to the cases addressed in this thesis is analysed.

2.3 Related work

This section provides a presentation of the works more related to the proposed methods in this thesis. This presentation includes an overview of the methods and an analysis of their limitations. Hence, this section motivates the development of the methods presented in the rest of the thesis. The section is divided into four subsections, one for each of the remaining main chapters. First, methods that compute solutions that are optimal in terms of a cost function while satisfying LTL specifications are presented. Then, approaches that permit specifications with time constraints are analysed. Next, methods that find solutions for systems with imperfect state information are discussed. Finally, approaches considering multi-robot systems are presented.

2.3.1 Optimal control

Using the approaches presented in the previous section, several paths can be obtained such that a temporal logic specification is satisfied. Nevertheless, in many situations, an optimal path with respect to a cost function is desirable. Approaches considering deterministic, non-deterministic and stochastic system have been developed for this purpose.

Due to the developed techniques to abstract dynamic systems, several works assume the existence of such abstractions and just focus on the discrete problem. In other words, a graph modelling the mobility of the system is assumed. One of the first approaches focused on obtaining optimal paths under such an assumption is presented in [157]. An optimal path, in terms of the time required to reach regions of interest, is found by solving a bottleneck problem on the product automaton of the specification and a transition system. A similar problem is solved in [185] where the average cost of a task that is repeated infinitely often is minimised using dynamic programming techniques. Using this approach, optimal paths can be found in polynomial time. When non-deterministic systems are considered, approaches based on dynamic programming can be used for fragments of LTL to avoid the complexity, in terms of the number of states, of using automata-based techniques for non-deterministic systems [187].

In the previous cases, the cost at each state is assumed to be invariant. When this assumption is relaxed, an offline strategy can be combined with an online strategy, based on receding horizon approaches, to improve the performance of the first one [163].

Methods focusing on the discrete problem have also been developed for stochastic systems. These works consider a Markov decision process (MDP) as a model of the system. Similar to the deterministic problem, a common practice is to compute the product MDP of an automaton that represents the specification and an MDP modelling the system. Using

this product MDP, policies that maximise the probability of satisfying the specification [41] or minimise a cost function while satisfying the specification [40, 162] can be computed. Other approaches use tools from available model checkers, such as PRISM [93], to find policies that satisfy probabilistic CTL (PCTL) specifications [95].

As mentioned before, the previous approaches do not consider the dynamics of the system but only focus on the discrete problem. When mixed logical dynamic systems are considered, optimal control can be found by solving mixed integer linear programming problems. In [79], the authors present an algorithm to encode finite horizon LTL formulae as constraints of a MILP. As in other works using a MILP approach, atomic propositions indicate whether the system is in a polyhedral subset of the state space. Once the constraints of the specification and the system dynamics have been computed, the MILP problem can be solved using off-the-shelf numerical optimisation solvers, e.g. CPLEX [36]. The previous work is extended in [188] to include full LTL using ideas of bounded model checking (BMC) [20]. To achieve this, trajectories are restricted to those creating loops or lassos.

MILP approaches have also been combined with automata-based model checking. In [186], the state space is coarsely partitioned and modelled as a transition system. Then, a product automaton is computed in the usual way to find a discrete plan. The feasibility of this plan is checked by solving MILP problems to find a control for each transition according to the plan. If a transition is not feasible, the partition is refined until the discrete plan can be achieved. A similar approach is presented in [57] where a dual automaton defines the partition of the state space.

The main limitation of approaches based on linear programming is that the number of constraints increases with the size of the specification causing the complexity to grow exponentially. Although this problem can be alleviated by generating constraints as required for a subset of specifications [151], the MILP approach can only be applied to linearisable systems as opposed to the solution proposed in Chapter 4.

To avoid the discretisation or a possible non-convex optimisation in the previous approaches, hybrid state spaces have been considered. The hybrid state space is formed by the continuous-time continuous-state of the system and the discrete states of an automaton. Then, the optimal control is found by solving a mixed continuous-discrete Hamilton-Jacobi-Bellman equation. For linear systems, an approximate solution to this problem can be computed by semidefinite programming [133] and for nonlinear systems the solution can be found using importance sampling methods [48]. However, the solution may converge to a local optimum.

Direct computation in the state space has also been considered for optimal control of stochastic systems. Similar to [186], an automaton is used to guide the computation of

stochastic constrained reachability problems in [65]. Each reachability problem is solved via a Hamilton-Jacobi-Bellman partial differential equation. The solutions to the individual problems are chained together using dynamic programming.

Other approaches use sampling-based methods that iteratively create a discrete representation of the state space until a solution is found or other conditions are satisfied. In [78], a rapidly-exploring random graph is iteratively created to model a subset of the system motion and a model checker algorithm is used to verify whether a μ-calculus specification can be satisfied using the current graph. The graph construction is based on the idea of RRT* [77] that requires an exact steering function to constantly rewire the graph. This steering function solves a two-point boundary value problem [101] to optimally drive the system from one state to another. However, finding a solution to this problem is not easy for certain kinodynamic systems [183]. In Chapter 4, a sampling-based solution that does not require such exact steering functions is presented.

The previous approach is asymptotically optimal. Hence, an important aspect is the rate of convergence. To improve the convergence, methods based on cross-entropy (CE) [150] have been combined with automata-based methods [29, 114]. In these approaches, the trajectory of the robot is parametrised and the CE method is used to incrementally find promising trajectories in terms of a cost function. Although this approach can be used for high-dimensional nonlinear systems, the parametrisation is in general non-trivial [87].

Sampling-based methods have also been used to find optimal paths for stochastic systems. In [118], an MDP is created by partitioning the workspace. To compute the probability of transitioning from one segment to another, a Markov chain is computed by sampling the state space of the system and the control space. Then, the MDP is combined with an automaton representing the specification to compute the optimal policy using dynamic programming.

From the discussion above, it can be concluded that although several approaches have been proposed to find optimal paths considering different factors, some problems remain open. As in most problems considering model checking, the complexity of finding optimal paths is a limitation for most methods. This complexity arises from the large number of states that must be analysed to find a path that satisfies a specification. Although some of the methods aforementioned focus on the reduction of the number of states, their applicability to any kind of system dynamics is limited. The applicability to nonlinear systems with differential constraints is another main limitation of most methods. Indeed, the field of motion planning has not yet achieved a general purpose method to find optimal paths [64]. Hence, computing optimal paths for any constrained system based on high-level specifications remains an open problem. In Chapter 4, a method that provides a first step to solve

this problem is proposed. The method requires only the forward propagation of the system dynamics to find asymptotically optimal paths satisfying temporal logic specifications. Therefore, it can be applied to a wider range of system dynamics. To achieve this, however, the method is limited to a subclass of LTL specifications.

2.3.2 Time constrained specifications

In Section 2.3.1, methods designed to find optimal controls for deterministic and stochastic systems under LTL and μ-calculus constraints were presented. Although useful missions can be expressed using these logics, they are limited to qualitative specifications. In other words, only the order of events can be expressed. However, in many situations, real-time constraints are required. To solve this limitation, logics such as Metric Temporal Logic (MTL) and Signal Temporal Logic (STL) [120] have been proposed. In contrast to LTL and μ-calculus, MTL and STL permit to define specifications where tasks must be executed within a period of time instead of an arbitrary time. Using these logics several approaches have been developed to include time constraints.

In [113], the authors consider the problem of designing switching controllers for nonlinear systems subject to MTL specifications. They propose a method to compute discrete abstractions with robustness margins for nonlinear systems in such manner that MTL properties are preserved. Moreover, in order to ensure the correctness of the strategy obtained from the abstraction in the original system, a transformation of MTL formulae is presented. Once the abstraction and the modified formulae are computed, methods found in the literature can be applied by transforming the MTL specification into an LTL specification [113] or by considering the discrete abstraction as a timed automaton such as in [198].

In [198], a timed transducer is computed based on a Metric Interval Temporal Logic (MITL) specification. The abstraction of the system is performed by partitioning the environment into cells and estimating the time required to drive the system from one cell to another. The solution to the motion planning problem is found in the product of the timed automaton and the transducer using the model checker UPPAAL [12]. As presented in Section 2.2, due to scalability limitations of computing an abstraction, other works have been presented using the MILP approach which has better scalability with respect to the system dimensionality [145].

One of the first works to use MTL specifications with MILP-based algorithms is [75]. In this work, the authors find the optimal path for a variant of the vehicle routing problem where the task is defined by a MTL specification. In order to reduce the number of constraints due to the size of the specification in the MILP approaches, in [151], the authors propose a method where constraints are added as required. The main idea is that solving

a series of small MILP problems is faster than a single complex problem. To identify the required constraints, a robustness measurement is introduced. This measurement indicates how much the system can be perturbed before violating the specification at different points of a trajectory. Then, constraints are added repeatedly where the trajectory is violated by the largest margin and the MILP is solved. Results show a significant complexity reduction compared to adding constraints at each time step. Nevertheless, the approach may not return the optimal trajectory except for a fragment of MTL.

The previous works considered systems with deterministic motion. However, most systems are affected by uncertainties. These arise due to modelling errors, external disturbances, etc. To address this problem, methods considering bounded uncertainty and stochastic uncertainty have been proposed.

In [144], the authors consider a Model Predictive Control (MPC) based approach with STL specifications. By using STL, quantitative properties can be considered instead of only Boolean. The authors propose a method to automatically encode bounded specifications, based on BMC, into mixed-integer linear constraints in such a way that the trajectory obtained maximises the robustness of the satisfaction of the specification. At every time step, a MILP problem is solved to compute an optimal control within a horizon subject to the constraints imposed by the system dynamics and STL specifications. A main limitation of using a receding horizon approach is that the global optimality is not ensured.

The previous work is extended in [145] to include unbounded specifications and reactiveness to a possible adversarial nondeterministic environment. This extension uses a counterexample-guided inductive synthesis (CEGIS) approach to react to an environment which tries to minimise the robustness of the satisfaction. The method finds a control satisfying a specification only for a finite set of environments satisfying predefined assumptions. These environments represent input disturbances. When a solution is found for a particular set of disturbances, new disturbances are added to the set until a solution cannot be found. Hence, a possible limitation of this work is that using the CEGIS approach the convergence of the method is not guaranteed if the set of disturbances is not finite [45]. Although techniques such as multi-parametric MILP and Monte Carlo approaches can be used to ensure the termination of the algorithm [45], in general, they require long computational time. In contrast to the work presented in Chapter 5, the system in [145] is nondeterministic rather than stochastic and considers a set of disturbances. Instead, the method in this thesis considers a probability distribution, which is more appropriate to model elements such as an estimate of position computed via a Kalman filter or external disturbances such as wind [21].

For stochastic dynamics, the authors in [49] compute an optimal policy with respect to the probability of satisfying a MITL specification. The original system is first approximated using the Markov chain approximation method [92]. This approximation relies on the discretisation of the continuous state space and time in such a way that the properties of the original system are conserved. Then, the problem of finding a policy that maximises the probability of satisfying the specification is reduced to a reachability problem in a product MDP. This product is obtained from the MDP that approximates the original system and a deterministic timed automaton obtained from the specification. The main limitation of this approach is the scalability. Since the method discretises the state space and time, the number of states in the product becomes intractable when high-dimensional systems are considered.

Similar to the previous section, one of the main problems of including time in the specifications is the complexity of the problem. While some approaches need to solve a NP-hard problem, e.g. MILP, other approaches required the discretisation of time and state space. The number of states of this discretisation becomes impractical for systems with high dimensional spaces. In Chapter 5, a method that achieves computational tractability by using a coarse discretisation and sampling-based methods is presented. More specifically, the method is divided into two phases. In the first phase, policies to drive the system between partitions are computed. In the second phase, a global policy is computed to find a trajectory to satisfy the specification. This division of the problem allows to compute policies faster than other methods and provides a trade-off between computation speed and the smoothness of the trajectory.

2.3.3 Uncertainty in motion and sensing

In the previous section, uncertainty is considered in the motion of the system. However, uncertainty is also often present in sensing, resulting in partially observable states of the system. In these cases, the system cannot decide the best action based on a single state but only on a probability distribution over all possible states. In its more general form, a system with various sources of uncertainty such as action, sensing or environment can be modelled as a partially observable Markov decision process (POMDP) [71]. Planning in POMDPs consists of finding policies that map a history of observations and actions to an action such that the expected reward is maximised. Most of the solutions proposed to solve these problems consider a compact representation of the history of observations and actions, called belief. In general, solving these problems is computationally intractable [132], therefore, methods that approximately solve the problem by considering points over the belief state instead of the entire belief state have been developed [135]. These methods can be used to solve the traditional point to point motion planning problem. However, they extension to

problems with high-level specifications, especially to those with infinite horizon properties, is not trivial. To consider such specifications, other methods have been proposed.

POMDPs with tasks that are satisfied in an infinite horizon are considered in [26]. In general, the analysis of whether infinite horizon objectives given as a parity objective can be ensured with probability one is undecidable [8]. To make the problem tractable, in [26], an algorithm that considers finite-memory policies [27] and a series of heuristics are employed. The approach demonstrates applicability for robotics applications with dynamic environments [164].

In a similar problem, the authors in [155] consider the computation of controllers that maximise the probability of satisfying LTL specifications for POMDPs. In contrast to the previous work, parametrised controllers with pre-defined number of states are considered. While this assumption makes the problem more tractable, it also limits the class of controllers than can be considered to solve the problem. This method solves a constrained optimisation problem to find the parameters that maximise the probability of being absorbed by sets of accepting states in the product of the POMDP and a deterministic Rabin automaton. Besides the limitation in the class of controllers, because this solution considers only the current observation, instead of a history of them, feasible problems can become infeasible. Moreover, the method suffers from local maxima.

To avoid the problem of the previous work, a learning-based approach is presented in [196]. A finite-state automaton is iteratively created and used as a supervisor of the POMDP. The learning process, used to create the automaton, is based on counterexamples returned by the Markov chain of the POMDP and the automaton. Although sound, the method is not complete and limited to finite horizon specifications expressed as PCTL.

In the previous works only dynamic but not adversarial environments, i.e. the environment tries to falsify the specification, are considered. For the latter case, policies that maximise the probability of the system satisfying the specification are computed. One such approach is presented in [190], where policies that maximise the worst-case probability of satisfying a LTL specification for partially known environments are computed. They assume that the environment can be in one of several modes, which are modelled as Markov chains. Although the system does not know exactly which is the current mode of the environment at each time, all the possible environment models are known by the system. This is a limitation since in many applications these models are not available. The policies are computed using a parallel composition between a MDP modelling the system and the set of Markov chains.

In [47], a reactive controller is computed assuming partial information and adversarial environments. The method focuses on safety specifications where the system needs to avoid a set of states. The control or strategy is obtained by solving a two-player game with partial

observation. To cope with a large number of states, the abstract game is constructed from the original structure that represents the interaction between the system and the environment. When a strategy fails to satisfy the specification, counterexamples are used to refine the abstraction and to update the sensor model until the specification is realisable. The previous work is improved in [50] to allow a subset of LTL specifications and the use of sensing actions to reduce the complexity of the problem. The approach obtains a belief-based strategy from a game with complete information. This strategy maps belief states to actions. When a robot finds a belief where the strategy is not defined, the robot applies sensing actions until a belief with a defined strategy is found.

Most of the previous works solve the problem at the discrete level. In other words, a partition of the environment and controllers to drive the system between them are assumed. As presented before, this could limit the type of system from which a solution can be found. Therefore, other approaches that consider the continuous state space have been proposed.

An approach to maximise the probability of satisfying bounded linear temporal logic specifications for stochastic systems, where only the initial state is known, is considered in [32]. At each time step, the only information available is the measurement of two noisy encoders. An MDP is created by mapping the incremental encoder measurements. Due to the size of the MDP, the approach uses a statistical model checking approach, where policies are used to generate traces on the MDP to find a solution. Then, the probability of the trace satisfying the specification is computed and the policy is improved and reapplied until the probability converges. Due to the cumulative uncertainty, the approach is limited to finite specifications.

In [121], an approach that computes control laws for piecewise-affine systems, operating in possible adversarial environments, to satisfy LTL specifications is proposed. In order to use available methods for solving two-player games with perfect state information, e.g. [193], an observer is used to estimate the state of the system. The satisfaction of the LTL specification is guaranteed by bounding the error from the observer. Then, it is proved that a control computed for the observer system satisfying a robust version of the LTL specification, can be applied in the original system to satisfy the original specification.

In [182], the authors propose a specification language, called Gaussian distribution temporal logic (GDTL), that permits including noise mitigation in the specification. The work creates a graph by sampling the state space of the system and computing feedback controllers that stabilise the system at each node. For each transition, a particle-based method is used to compute the probability of transitioning from one state to another. Finally, a policy that maximises the probability of satisfying the specification is computed on a MDP with the probability previously computed. In contrast to the method proposed in Chapter 6, this

approach assumes a static environment. Hence, the system cannot react to changes in the environment.

In general, solving problems for partially observable MDPs is a hard task [132]. The problem of path planning with temporal logic specifications for systems with uncertainty in state and dynamic environments has been addressed mainly at the discrete level. In other words, a POMDP or a series of MDPs is assumed to model the possible transitions of the system. Although some solutions consider the continuous state space, they do not consider a possible changing environment. In Chapter 6, a method that finds policies that maximise the probability of satisfying a LTL specification on a dynamic environment is presented. In order to react to the changing environment, a graph representing the motion of the system is computed offline and used to drive the system when a previously unknown object is detected. This graph considers the temporal logic specification to react to the environment without violating the specification during the operation of the system. Since the system must be able to reach specified states with zero velocity in the graph, the method cannot be applied to certain systems, e.g. fixed-wing aircrafts.

2.3.4 Multi-robot systems

The methods presented in the previous three sections are designed for tasks that require a single robot. However, in many cases, the cooperation of several robots is required to accomplish tasks that would be impossible for a single robot. For most of the aforementioned methods, their adaptation for the multi-robot case is not trivial. Moreover, as the number of robots increases, the scalability of the methods is compromised due to the total number of possible states. Therefore, different techniques are required.

The motion planning of multiple robots subject to temporal logic constraints can be divided into two categories depending on the task: (i) a task can be assigned to each robot, where the cooperation among them might be required; or (ii) all the robots work as a team to perform a global task. In this section, works considering both situations are presented.

Tasks with individual specifications. A possible solution to the multi-robot problem is to compute a synchronous product of the transition systems of the robots and the automata of the specifications. This approach is computationally expensive. To reduce this computational complexity, a receding horizon approach is proposed in [169]. The method creates a synchronised automaton based on a predefined horizon of the automaton of all the specifications. Moreover, a progressive function is defined to indicate the progress towards the satisfaction of the specifications. This function is used to define the temporal goals with the maximal progression in each synchronised automaton. To compute a path in the transition system such that the local targets are reached, a product automaton of the synchronised

automaton and the transition system modelling all the robots is constructed considering a certain horizon. Once the local target is reached, a new plan is computed and the process is repeated until the specifications are satisfied.

A different approach to reduce the complexity of computing a synchronised automaton is presented in [170]. In this work, robots have to satisfy motion and action specifications, which might require cooperation with other robots. Although the final motion and action plan are computed in a synchronised automaton, the number of states is reduced by eliminating states where the motion of each robot is not planned.

In the previous works, the plan is computed in a synchronised automaton. Therefore, the transitions of the robots must be synchronised during the execution of the plan. This represents a limitation in terms of robustness and flexibility. This synchronisation problem is solved in [61], where a method for loosely coupled systems is proposed. An offline plan is obtained in the product automaton of a transition system modelling a discretised environment and the automaton of the specification for each robot. During the execution of the plan, each robot observes whether cooperation is required within a horizon. In case of cooperation, a request is sent to robots to indicate the position and time of collaboration. When a robot accepts a request, it modifies its original plan to include the location of the requested action.

A method for partially known environments is proposed in [62]. The method finds paths that satisfy hard constraints such as safety properties while minimising the violation of soft constraints due to the partial knowledge of the environment. This is achieved by constructing a product automaton of a transition system modelling different points of the environment and a modified synchronised automaton. The approach allows robots to discover the environment via sensing or communication. To reduce communication, a protocol where robots send only relevant information to each neighbour in terms of regions of interest is proposed. With every update about the environment, the discrete plan is updated. As in the previous case, a navigation function is used to drive the system between points.

Specifications with real-time constraints have also been considered for multi-robot systems [127]. In this case, robots are assigned with individual specifications given as MITL specifications. Moreover, a global specification is required to be satisfied at the same time. The time of the transition, in a partitioned environment, is computed by considering the worst time required to move from any point of one segment to its boundary. The individual plans for each robot are obtained by computing individual products of transition systems and timed automaton of individual specifications. Then, a product is computed with the previous product automata and finally, a global one is constructed by adding the timed automaton of the global specification.

Tasks with global specifications. As aforementioned, other solutions consider a global specification that needs to be satisfied by a team of robots. Similar to the previous case, the problem can be solved by creating a parallel composition of the individual transition systems to create a model of the motion of all the robots as a group. Then, this composition is used to compute a Cartesian product with the automaton. Although available model checking solvers can be used to obtain a plan using this approach, it is computationally expensive [84]. Moreover, synchronisation between robots is required for each transition. Although the latter problem can be partially alleviated by computing the moment where the synchronisation is required [85], in general, this approach does not scale well with the number of robots. Therefore, solutions have been developed aiming to improve this scalability.

An approach that uses an abstraction that does not depend on the number of robots is presented in [86]. A Petri net is used to model the environment using tokens to represent the positions of the robots. To find a path for the robots, a path is sought in the automaton of the specification. Then, a search is performed in the Petri net looking for a sequence of firings or transitions that generate the atomic propositions required for the chosen path. The sequence of transitions is obtained by solving an integer linear programming. As a path is selected before searching for a sequence on the petri net, the solution is suboptimal.

Other approaches decompose the global specification into local ones [28]. This can be achieved by checking whether the specification is distributable. If this is possible, product automata of individual transition systems and mixed automata are computed. The individual transition systems and mixed automata model the motion of the robots and the specification of each of them, respectively. To find the global strategy a synchronous product automaton is formed by the individual product automata. Using a similar approach, the problem of deploying a team of robots with the purpose of gathering information from an uncertain environment while the motion of the robots is constrained by a temporal logic specification has been solved [103]. In this case, the global specification is distributed among the teams which execute a receding horizon planner where a dynamic programming problem is solved to minimise the uncertainty. The idea of distributability has also been used to verify whether a specification can be violated when uncertainty in travelling times exists [171]. In this work, asynchronous individual plans are computed for each robot such that the global behaviour minimises a cost function in terms of time. Since uncertainty in travelling times can produce a global behaviour such that the specification is violated, a method is presented to verify whether the specifications are sensitive for this uncertainty. In such cases, communication is used to coordinate the motion of the robots just when this is required.

The approaches that use distributability are conservative in the sense that if the property is not distributable, the approach cannot find a solution even if one exists. Hence, these meth-

ods are limited to subclasses of LTL. Moreover, the previous cases assume the availability of a discrete abstraction for the system and only treat the discrete problem. Nevertheless, the generation of these abstractions is non-trivial for several systems. In contrast, the approach presented in Chapter 7 finds a solution directly on the configuration space of the robots. More complex dynamics in the form of nonholonomic systems are considered in [197], where a method is proposed to incorporate reachability properties of the robots in a graph modelling the workspace.

Using a sampling-based method, the authors in [74] create a tree that approximates the product automaton. This approximation permits to solve large problems, in terms of the number of states in the product automaton, that are not solvable considering the product automaton itself. However, in contrast to the solution proposed in Chapter 7, they sample states from a transition system representing regions of the environment and not from the configuration space of the robots.

Similar to the previous sections, the number of states or state explosion is one of the main problems when multiple robots are considered. This is due to the large number of states that must be considered to find a solution, especially for multiple high-dimensional systems. Different approaches have been proposed to mitigate this problem. In most of these approaches, the motion of the robot between regions of the workspace is assumed or the use of navigation functions is required. However, for high-dimensional systems, these approaches could present limited scalability. As presented in Section 2.1.5, sampling based methods were developed to handle problems with systems with multiple degrees of freedom. Nevertheless, these methods have not been used for multiple robots and temporal logic specifications. In Chapter 7, a novel method that extends sampling-based methods for multiple robots and temporal logic specifications is proposed. In contrast to other available methods, the proposed approach considers the continuous configuration space to find a solution rather than a partition of an environment like must of the available methods. Moreover, an exploration of the configuration is guided by the proposed method to find a solution in a short period of time. Since the method is geared towards fast solutions, it does not find optimal trajectories.

2.4 Concluding remarks

This chapter has presented an introduction to the problem of motion planning. Moreover, an explanation of sampling-based methods as a way to deal with problems where high-dimensional systems are considered has also been provided. The main limitations of these methods to solve problems with complex tasks and the methods used to solve these limi-

tations have been presented. Finally, a review of methods that synthesise controllers using model checking techniques for deterministic, stochastic and multi-robot systems has been presented

A common problem with the available methods is the scalability and the limited applicability in terms of system dynamics. In the traditional point to point motion planning, sampling-based methods have been used to alleviate the problem of state explosion and the computation of controllers for systems with kinodynamic constraints. Motivated by the results demonstrated using these methods, in the remaining chapters, methods based on sampling-based approaches are proposed to find trajectories based on temporal logic specifications in different situations. More specifically, in Chapter 4 optimal trajectories for kinodynamic systems are found by sampling the state space of the system. In Chapter 5, a sampling-based method is used to approximate the dynamics of a stochastic system. Finally, in Chapter 6 and 7 large state spaces are explored by using sampling-based methods. Before presenting these methods, preliminaries of model checking concepts are presented in Chapter 3.

Chapter 3

Preliminaries

This chapter introduces definitions and concepts that will be used throughout the thesis. Specifically, it presents a background of model checking and automata theory, which is used in the methods presented in the rest of the chapters. First, the graph structures used as models for deterministic and stochastic dynamic systems are defined. Then, the logics used to formally specify the desired behaviour of the systems are presented together with the automata that can be computed to represent such specifications.

3.1 System models

In order to model the motion capabilities of deterministic systems, deterministic transition systems are used. Before formally defining transition systems, some definitions are required.

Definition 3.1.1. (Alphabet, word and language) *A set of symbols $\Sigma = \{\sigma_1, \sigma_2, \cdots\}$ is called an alphabet. A word w is a sequence of symbols over Σ. The set of finite and infinite words are denoted by Σ^* and Σ^ω, respectively. A set of words over an alphabet Σ is called a language [168].*

Definition 3.1.2. (Atomic propositions) *An atomic proposition π is a true or false statement about a property of a system.*

To illustrate the idea of atomic propositions more intuitively, consider a robot operating in a workspace with two regions of interest. The atomic propositions π_1 and π_2 can be used to identify whether the robot is in one of these areas. In other words, π_1 would evaluate true if the system is in the first region, otherwise would be false, Fig 3.2. In this thesis, atomic propositions are used to identify the position of a system with respect to areas of the workspace. However, any element of the state of a system can be labelled by an atomic proposition.

Definition 3.1.3. (Transition system) *A deterministic transition system $\mathcal{T}$ is a tuple $(S, s_0, \delta_{\mathcal{T}}, \Pi, L)$, where:*

- *S is a finite set of states,*
- *$s_0 \in S$ is an initial state,*
- *$\delta_{\mathcal{T}} \subseteq S \times S$ is a transition relation,*
- *Π is a set of atomic propositions,*
- *$L : S \to 2^{\Pi}$ is a labelling function.*

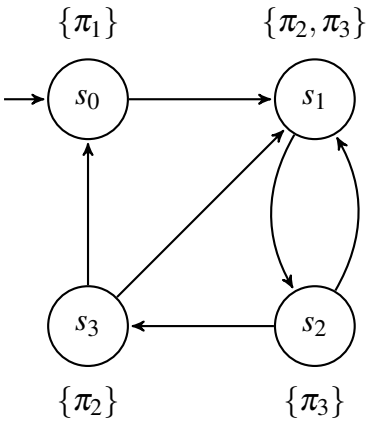


Fig. 3.1 Example of transition system. The set next to each state is the result of applying the labelling function, e.g, $L(s_0) = \pi_1$.

An example of a transition system is shown in Fig. 3.1. An execution or run on $\mathcal{T}$ is a sequence of states $\boldsymbol{s} = s_0 s_1 s_2 \ldots$, where s_i is the state at time i and $(s_i, s_{i+1}) \in \delta_{\mathcal{T}}$ for all $i \geq 0$. Note that an execution in $\mathcal{T}$ must start with the initial state. A trace $L(s_0)L(s_1)L(s_2)\ldots$ describes a run $\boldsymbol{s}$ in terms of the atomic propositions that evaluate true. This trace generates a word w over the power set of the alphabet, i.e. 2^{Π}, Fig. 3.2. The set of all possible words, i.e. language, generated by $\mathcal{T}$ is denoted by $\mathcal{L}(\mathcal{T})$.

As aforementioned, the transition system defined above is used to model deterministic systems. Nevertheless, in Chapters 5 and 6, systems with stochastic motion are considered. Hence, a mathematical framework capable of expressing the stochastic aspect of the system is required. For this purpose, Markov decision processes (MDP) and Bounded-parameter Markov decision processes (BMDP) are used.

Definition 3.1.4. (Markov decision process) *A Markov decision process [9] $\mathcal{M}$ is a tuple (S, s_0, A, P, Π, L), where:*

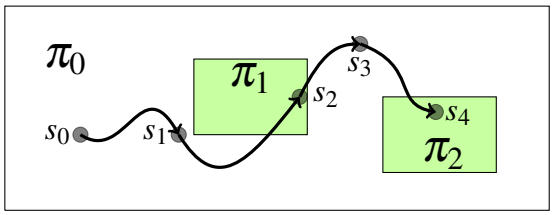


Fig. 3.2 Illustration of a sequence of states in a workspace labelled with atomic propositions. The behaviour of the system can be modelled as a transition system $\mathcal{T}$. The figure shows a specific run $\boldsymbol{s} = s_0 s_1 s_2 s_3 s_4$ on $\mathcal{T}$ reaching the regions associated with atomic propositions π_1 and π_2. The word $w = \{\pi_0, \neg\pi_1, \neg\pi_2\}\{\pi_0, \neg\pi_1, \neg\pi_2\}\{\neg\pi_0, \pi_1, \neg\pi_2\}\{\pi_0, \neg\pi_1, \neg\pi_2\}\{\neg\pi_0, \neg\pi_1, \pi_2\}$ is generated due to the atomic propositions satisfied by $\boldsymbol{s}$.

- *S is a finite set of states,*
- *s_0 is an initial state,*
- *A is a finite set of actions,*
- *$P(\cdot|\cdot,\cdot) : S \times S \times A \to [0,1]$ is the probability of transitioning to state s' from state s under action $a \in A$,*
- *Π is a set of atomic propositions,*
- *$L : S \to 2^{\Pi}$ is a labelling function.*

The set $A(s)$ defines all the available actions at state s such that $P(s'|s,a) > 0$ for all $a \in A(s)$. The probability function P satisfies the following condition:

$$\forall a \in A(s), \sum_{s' \in S} P(s'|s,a) = 1.$$

A run on $\mathcal{M}$ is a sequence of states $\boldsymbol{s} = s_0 s_1 s_2 \ldots$, where where s_i is the state at time i and for all $i \geq 0$, $P(s_{i+1}|s_i, a_i) > 0$. Let S^n_{fin} be the set of all finite runs $\boldsymbol{s}_{\text{fin}} = s_0 s_1 s_2 \ldots s_n$ for any $n \in \mathbb{N}$. A control strategy or policy is a function $\mu : S^n_{\text{fin}} \to A$ such that $\mu(\boldsymbol{s}_{\text{fin}}) \in A(s_n)$ for all $\boldsymbol{s}_{\text{fin}} \in S^n_{\text{fin}}$. A policy is memoryless if for two runs $\boldsymbol{s}_{\text{fin}} = s_0 s_1 s_2 \ldots s_n$ and $\boldsymbol{s}'_{\text{fin}} = s'_0 s'_1 s'_2 \ldots s'_{n'}$ with $s_n = s'_{n'}$, $\mu(\boldsymbol{s}_{\text{fin}}) = \mu(\boldsymbol{s}'_{\text{fin}})$. In other words, a policy is memoryless if it always selects the same action in a given state s. Hence, a memoryless policy can be viewed as a function $\mu : S \to A$. When systems with stochastic motion are considered in this thesis, the objective is to maximise the probability of reaching subsets of states in MDPs. For such problems, memoryless policies are sufficient to return the optimal result [9].

Definition 3.1.5. (Bounded-parameter Markov decision process) *A bounded-parameter Markov decision process [55] is a generalisation of the exact MDP and is defined by a tuple* $\mathcal{B} = (S, s_0, A, \hat{P}, \check{P}, \Pi, L)$*, where:*

- S *is a finite set of states,*
- s_0 *is an initial state*
- A *is a finite set of actions,*
- $\hat{P}(\cdot|\cdot,\cdot) : S \times S \times A \to [0,1]$ *is the upper bound of the probability of transitioning to state* s' *from state* s *under action* $a \in A$*,*
- $\check{P}(\cdot|\cdot,\cdot) : S \times S \times A \to [0,1]$ *is the lower bound of the probability of transitioning to state* s' *from state* s *under action* $a \in A$*,*
- Π *is a set of atomic propositions,*
- $L : S \to 2^{\Pi}$ *is a labelling function. function.*

For all states $s \in S$ and any action $a \in A$, the probability functions $\hat{P}$ and $\check{P}$ satisfy the following conditions:

$$0 \leq \check{P}(\cdot|s,a) \leq \hat{P}(\cdot|s,a) \leq 1,$$

$$0 \leq \sum_{s' \in S} \check{P}(s'|s,a) \leq 1 \leq \sum_{s' \in S} \hat{P}(s'|s,a).$$

A run on $\mathcal{B}$ is a sequence of states $\boldsymbol{s} = s_0 s_1 s_2 \ldots$ such that for every $i \geq 0$, $\hat{P}(s_{i+1}|s_i, a_i) > 0$. The control strategy or policy of a BMDP is defined as in the MDP. An example of a MDP and a BMDP is shown in Fig. 3.3.

3.2 Linear temporal logic

In most of the chapters in this thesis, linear temporal logic (LTL) [139] is employed to define the desired behaviour of a system. Formally, LTL is a propositional logic extended by temporal operators. The syntax of LTL, given in the Backus-Naur form, is defined over the set of atomic propositions Π as follows:

$$\varphi := \pi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \mathcal{U} \varphi_2,$$

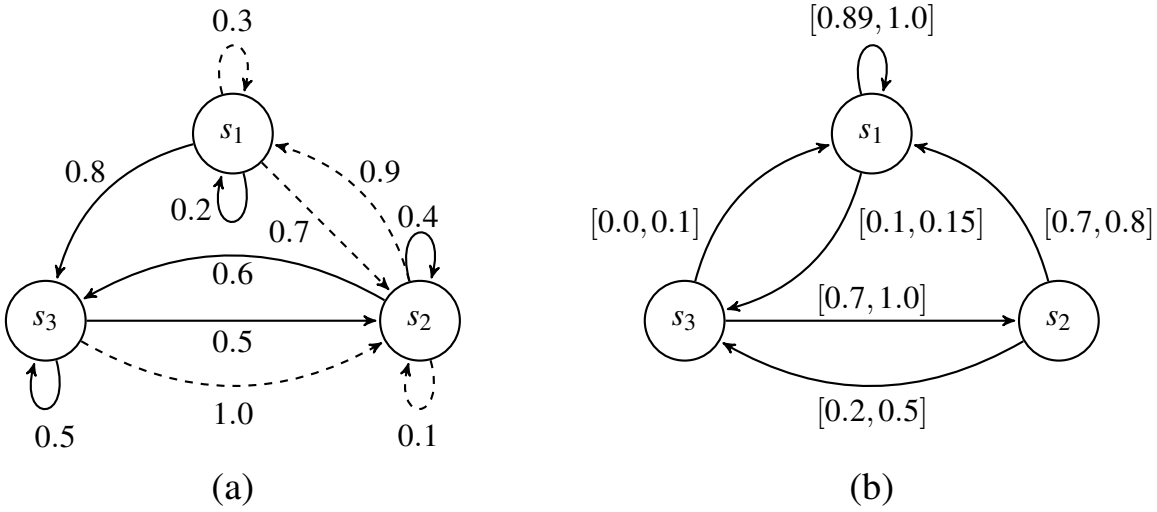


Fig. 3.3 (a) MDP with 3 states and two actions. Solid and dashed transitions correspond to different actions. Each edge label shows the probability of the transition. (b) Example of a BMDP with 3 states and a single action. The transitions are labelled with the lower and upper probability of transitioning.

where $\pi \in \Pi$ is an atomic proposition; and $\neg$, $\vee$, $\bigcirc$ and $\mathcal{U}$ represent the operators *negation*, *disjunction*, *next* and *until*, respectively. Using these operators, other logical and temporal operators such as $True = \pi \vee \neg\pi$, *conjunction*, $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$, *eventually*, $\Diamond\varphi = True\mathcal{U}\varphi$, and *always*, $\Box\varphi = \neg\Diamond\neg\varphi$, can be defined. The discrete semantics of LTL are defined over words $w \in \Sigma^\omega$. Given a run $\boldsymbol{s} = s_0 s_1 s_2 \ldots$ of a transition system $\mathcal{T}$, a LTL specification φ and the satisfaction relation $\models$, the semantics are inductively defined as follows:

- $s_i \models \pi$ iff $\pi \in L(s_i)$,
- $s_i \models \varphi_1 \wedge \varphi_2$ iff $s_i \models \varphi_1$ and $s_i \models \varphi_2$,
- $s_i \models \varphi_1 \vee \varphi_2$ iff $s_i \models \varphi_1$ or $s_i \models \varphi_2$,
- $s_i \models \bigcirc\varphi$ iff $s_{i+1} \models \varphi$,
- $s_i \models \varphi_1 \mathcal{U} \varphi_2$ iff $\exists j \geq i$ such that $s_j \models \varphi_2$ and $s_k \models \varphi_1, \forall k \in [i, j)$.

An illustration of the semantics is shown in Figure 3.4.

LTL has been widely used to express relevant behaviours in the area of autonomous systems. Examples of such behaviours include [26]:

- Liveness: Given a set of states where the atomic proposition π evaluates true, the objective is to eventually reach one of such states. Example: $\Diamond\pi$.

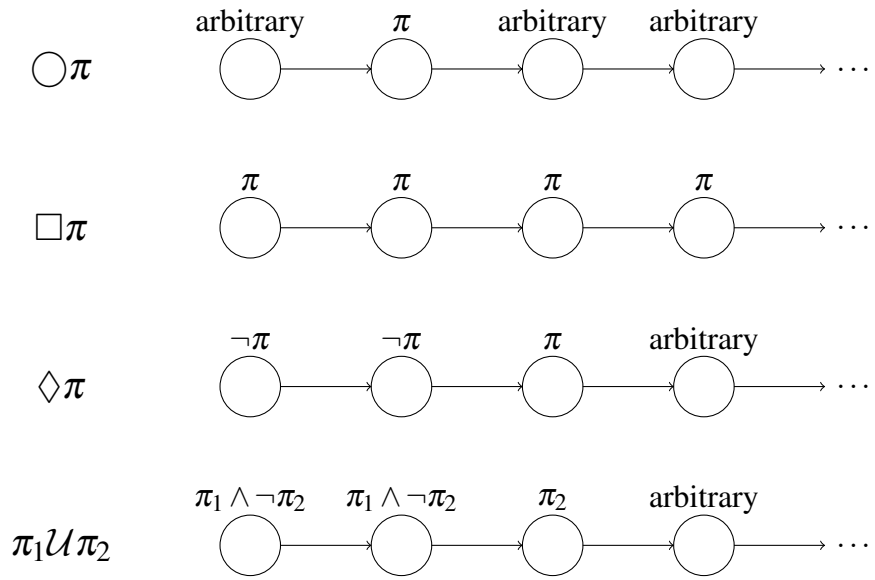


Fig. 3.4 Semantics of linear temporal logic. Intuitively, the first three formulae indicate that the atomic proposition π must be satisfied, in the next step, always and eventually, respectively. The last specification states that π_1 must be satisfied until π_2 is satisfied. (Adapted from [9])

- Safety: Given a set $S' \subseteq S$ of safe states, where $s \models \pi \ \ \forall s \in S'$, the objective is to remain on these states. Example: $\square\pi$.
- Sequencing: Given two sets of states $S', S'' \subseteq S$ where the atomic propositions π_1 and π_2 evaluate true, respectively, the objective of sequencing specifications is to indicate that the set S'' must be visited after reaching S'. Example: $\Diamond(\pi_1 \wedge \bigcirc\Diamond(\pi_2))$.
- Recurrence: The objective of recurrence specification is to visit a set infinitely often. Example: $\square\Diamond\pi$.

3.3 ω-automata

Automata that accept infinite words, i.e. $w \in \Sigma^\omega$, are called ω-automata and are defined as follows.

Definition 3.3.1. (ω-automaton) *A non-deterministic ω-automaton [168] is a tuple $\mathcal{A} = (Q, Q_0, \Sigma, \delta_\mathcal{A}, F)$, where:*

- *Q is a set of finite states,*

- $Q_0 \subseteq Q$ *is a set of initial states,*
- Σ *is a finite alphabet,*
- $\delta_{\mathcal{A}} : Q \times \Sigma \to 2^Q$ *is a transition function,*
- F *is the acceptance component.*

An automaton $\mathcal{A}$ is deterministic if $|Q_0| = 1$ and for all $q \in Q$ and $\sigma \in \Sigma$, $|\delta_{\mathcal{A}}(q,\sigma)| \leq 1$. A run $\boldsymbol{q}$ on $\mathcal{A}$, induced by a word $w = \sigma_0\sigma_1 \cdots \in \Sigma^\omega$, is a sequence of states $\boldsymbol{q} = q_0 q_1 \ldots$, such that $q_0 \in Q_0$ and for every $i \geq 0$, $q_{i+1} \in \delta_{\mathcal{A}}(q_i, \sigma_i)$. A run $\boldsymbol{q}$ is accepting if the accepting condition holds. The set of all words w that generate accepting runs is called the language accepted by $\mathcal{A}$ and is denoted by $\mathcal{L}(\mathcal{A})$.

From a LTL specification, it is possible to compute ω-automata that accept all and only words that satisfy such a specification [178]. Available tools exist to perform such computations, e.g. LTL2BA [52], LTL2DSTAR [83]. The results presented in the remaining chapters use the Büchi and Rabin acceptance conditions [168].

Definition 3.3.2. (Büchi acceptance condition) *Given an ω-automaton $\mathcal{A} = (Q, Q_0, \Sigma, \delta_{\mathcal{A}}, F)$ with $F \subseteq Q$ and a run $\boldsymbol{q}$, let* INF*($\boldsymbol{q}$) be the set of states that appear infinitely often in $\boldsymbol{q}$. Then, a word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$ with a Büchi acceptance condition iff there exists a run $\boldsymbol{q}$ satisfying the condition:*

$$\text{INF}(\boldsymbol{q}) \cap F \neq \emptyset.$$

An ω-automaton with Büchi acceptance condition is called a Büchi automaton. An example of a LTL specification and a Büchi automaton are presented in Fig. 3.5. The size of the Büchi automaton $\mathcal{B}_\varphi$, corresponding to the formula φ, has size $2^{\mathcal{O}(|\varphi|)}$ in the worst case, where $|\varphi|$ corresponds to the length of φ, i.e. the number of symbols. Nevertheless, this worst case is rarely encountered in practice.

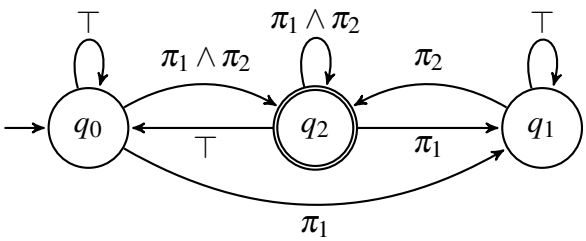


Fig. 3.5 Büchi automaton of formula $\varphi = (\Box\Diamond\pi_1 \wedge \Box\Diamond\pi_2)$, where π_1, π_2 are atomic propositions, $\top$ is unconditionally true and $F = \{q_2\}$. The formula indicates that the atomic propositions π_1 and π_2 have to be satisfied infinitely often.

Definition 3.3.3. (Rabin acceptance condition) *Let $\mathcal{A} = (Q, Q_0, \Sigma, \delta_{\mathcal{A}}, F)$ be an ω-automaton with $F = \{(L_1, K_1), \ldots, (L_n, K_n)\}$, where $L_i, K_i \subseteq Q \; \forall i \in \{1, \ldots, n\}$. Moreover, let $\boldsymbol{q}$ be a run on $\mathcal{A}$ and* INF*($\boldsymbol{q}$) be the set of states that appear infinitely often in $\boldsymbol{q}$. A word $w \in \Sigma^{\omega}$ is accepted by $\mathcal{A}$ with a Rabin acceptance condition iff there exists a run $\boldsymbol{q}$ satisfying the condition:*

$$\exists (L, K) \in F : \text{INF}(\boldsymbol{q}) \cap L = \emptyset \text{ and } \text{INF}(\boldsymbol{q}) \cap K \neq \emptyset.$$

Intuitively, a run $\boldsymbol{q}$ satisfies a Rabin acceptance condition if $\boldsymbol{q}$ visits the set L a finite number of times, or not at all, and the set K is visited infinitely often. Similar to the previous case, an ω-automaton with Rabin acceptance condition is called a Rabin automaton. An example is presented in Fig. 3.6.

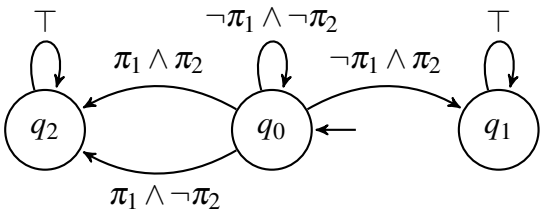


Fig. 3.6 Rabin automaton of LTL formula $\varphi = \neg\pi_2 \mathcal{U} \pi_1$, where π_1, π_2 are atomic propositions, $\mathcal{U}$ is the operator *until* and $\top$ is unconditionally true. The formula indicates that the atomic proposition π_2 has to be avoided until π_1 is satisfied. The component F is formed by the pair $L = \{q_1\}$ and $K = \{q_2\}$.

3.4 Co-safe linear temporal logic

In Chapters 4 and 7 a class of LTL, called syntactically co-safe linear temporal logic (sc-LTL) [91], is used. LTL formulae where negations only occur in front of atomic propositions and which only use the operators $\bigcirc$, $\mathcal{U}$ and $\Diamond$ are sc-LTL formulae. This class focuses on properties that can be satisfied in a finite horizon. The infinite words satisfying such specifications always have a good prefix.

Let φ be a sc-LTL specification. A finite word $w \in \Sigma^*$ is called a good prefix iff for all infinite words $w' \in \Sigma^{\omega}$, the concatenation $w \oplus w'$ satisfies the specification φ. In other words, a word satisfying a co-safe LTL specification is formed by a finite prefix followed by an infinite continuation which does not affect the satisfiability of the formula.

Let $\mathcal{B} = (Q, Q_0, \Sigma, \delta_{\mathcal{A}}, F)$ be a Büchi automaton. In contrast to the full LTL, a word w satisfying a sc-LTL specification is accepted by an automaton $\mathcal{B}$ if the produced run $\boldsymbol{q}$ reaches the set of accepting states F. Alternatively, given a sc-LTL formula φ, a determinis-

tic finite automaton, which accepts the prefixes of all words satisfying φ, can be constructed. Available tools exist to perform such computations [99].

3.5 Metric interval temporal logic

In Chapter 5, specifications with time constraints are considered. For the purpose of defining such specifications, the metric interval temporal logic (MITL) [4] is used. Before formally defining this logic, time sequences and timed runs are defined.

Definition 3.5.1. (Time sequence) *A time sequence $\eta = \eta_0\eta_1\ldots$ is an infinite sequence, where $\eta_i \in \mathbb{R}_+$, satisfying the following constraints:*

- *Initialisation: $\eta_0 = 0$.*
- *Monotonicity: $\eta_i < \eta_{i+1}$ for all $i \geq 0$.*
- *Progress: For every $t \in \mathbb{R}_+$, there is some $i \geq 1$ such that $\eta_i > t$.*

Let $\mathcal{T} = (S, S_0, \delta_\mathcal{T}, \Pi, L)$ be a transition system. A timed run $\boldsymbol{s_\eta}$ is a sequence of pairs $\boldsymbol{s_\eta} = (s_0,\eta_0)(s_1,\eta_1)\ldots$, where $(s_i, s_{i+1}) \in \delta_\mathcal{T}$ for all $i \geq 0$. Given an alphabet Σ and the labelling function $L : S \to 2^\Pi$, the trace of a run $\boldsymbol{s_\eta}$ defines a timed word (w, η).

MITL was introduced as an extension of LTL for real-time systems. In contrast to LTL, MITL allows to express quantitative temporal properties by extending the operator until $\mathcal{U}$ with a time interval I. The syntax of MITL, given in the Backus-Naur form, is defined over the set of atomic propositions Π as follows:

$$\varphi := \pi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2,$$

where $\pi \in \Pi$ and $\neg$, $\vee$, $\wedge$, $\mathcal{U}$, I represent the operators *negation*, *disjunction*, *conjunction*, *until* and a nonsingular interval with integer end points, respectively.

The discrete semantics of MITL are defined over words $w \in \Sigma^\omega$. Given a timed run $\boldsymbol{s_\eta} = (s_0,\eta_0)(s_1,\eta_1)\ldots$ of a transition system $\mathcal{T}$, a MITL specification φ and the satisfaction relation $\models$, the semantics are inductively defined as follows:

- $(s_i,\eta_i) \models \pi$ iff $\pi \in L(s_i)$,
- $(s_i,\eta_i) \models \varphi_1 \wedge \varphi_2$ iff $(s_i,\eta_i) \models \varphi_1$ and $(s_i,\eta_i) \models \varphi_2$,
- $(s_i,\eta_i) \models \varphi_1 \vee \varphi_2$ iff $(s_i,\eta_i) \models \varphi_1$ or $(s_i,\eta_i) \models \varphi_2$,
- $(s_i,\eta_i) \models \varphi_1 \mathcal{U}_I \varphi_2$ iff $\exists \eta_j > \eta_i$ such that $\eta_j - \eta_i \in I$, $(s_j,\eta_j) \models \varphi_2$ and $(s_k,\eta_k) \models \varphi_1, \forall \eta_k \in [\eta_i,\eta_j)$.

3.6 Timed automata

Let $C = \{c_1, c_2, \ldots, c_n\}$ be a set of real-valued variables, called clocks, and let Ω be a valuation for all $c_i \in C$. The valuation of the i-th clock is denoted by $\Omega(i)$. Given $t \in \mathbb{R}_+$, let $\Omega' = \Omega + t$ be the valuation with an increment t with respect to Ω in all the clocks, i.e. $\Omega'(i) = \Omega(i) + t$, for all $c_i \in C$. For the set C, $\Lambda(C)$ is a set of constraints defined as:

$$\lambda := c_i \leq k \mid c_i < k \mid c_i \geq k \mid c_i > k \mid \lambda_1 \wedge \lambda_2,$$

where $\lambda \in \Lambda(C)$ is a clock constraint, $c_i \in C$ and $k \in \mathbb{N}$.

Definition 3.6.1. (Timed automaton) *A deterministic timed automaton [3] is a tuple* $T = (Q, q_0, \Sigma, C, \Lambda(C), \delta_T, F)$*, where:*

- Q *is a set of finite states,*
- $q_0 \in Q$ *is an initial state,*
- Σ *is a finite alphabet,*
- C *is a finite set of clocks,*
- $\delta_T : Q \times \Sigma \times \Lambda(C) \to Q \times 2^C$ *is a transition function,*
- $F \subseteq Q$ *is a set of accepting states.*

The transition $(q', \varsigma) \in \delta_T(q, \sigma, \lambda)$ indicates a transition from q to q' with the input $\sigma \in \Sigma$ such that the clock constraint λ is met and the clocks in the set $\varsigma \subseteq C$ are reset to zero. Let Ω_0 be the valuation with all clocks equal to zero, i.e. $\Omega_0(j) = 0$ for all $j \in \{1, \ldots, |C|\}$. A run $\boldsymbol{q_\eta}$ on T, induced by a timed word (w, η), is a sequence of the form $\boldsymbol{q_\Omega} = (q_0, \Omega_0)(q_1, \Omega_1)\ldots$, where for all $i \geq 1$, $(q_i, \varsigma_i) \in \delta_T(q_{i-1}, \sigma_i, \lambda_i)$ such that $\Omega_{i-1} + \eta_i - \eta_{i-1}$ meets the constraint λ_i, $\Omega_i(k) = 0$ for all clocks $c_k \in \varsigma_i \subseteq C$ and for all clocks $c_j \in C \setminus \varsigma_i$, $\Omega_i(j) = \Omega_{i-1}(j) + \eta_i - \eta_{i-1}$.

Similar to the Büchi automaton, see Definition 3.3.2, a word (w, η) is accepted by a timed Büchi automaton, if $\textsc{Inf}(\boldsymbol{q_\eta}) \cap F \neq \emptyset$, where $\boldsymbol{q_\eta}$ is the run induced by (w, η) and $\textsc{Inf}(\boldsymbol{q_\eta})$ is the set of states that appear infinitely often in $\boldsymbol{q_\eta}$. An example of a timed automaton is shown in Fig. 3.7.

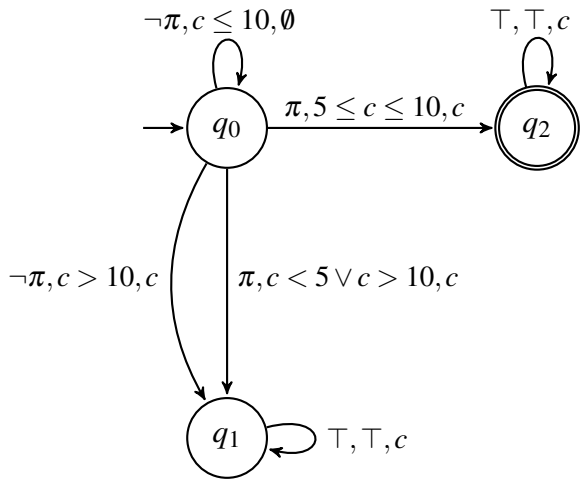


Fig. 3.7 Timed automaton of formula $\varphi = \Diamond_{[5,10]}\pi$, where π is an atomic proposition, $\Diamond$ is the operator eventually, $\top$ is unconditionally true and $F = \{q_2\}$. The formula indicates that the atomic propositions π has to be satisfied within 5 to 10 units of time. The edges are labelled by an atomic proposition, clock constraint and the set of resetting clocks.

Chapter 4

Optimal Kinodynamic Motion Planning with Co-safe Linear Temporal Logic Specifications

Research on motion planning based on temporal logic constraints has resulted in several approaches as presented in Chapter 2. These methods allow to find paths that satisfy high-level specifications expressed using temporal logics. Nevertheless, frequently there exist multiple paths satisfying these specifications. Hence, it most cases it is desirable to select the optimal path according to a cost function. This chapter focuses on the problem of finding optimal trajectories for deterministic high-dimensional kinodynamic systems subject to syntactically co-safe linear temporal logic (sc-LTL) specifications. This is a challenging problem due to the kinematic and differential constraints imposed by the dynamics of the system.

Most of the works in the literature are limited to systems with relatively simple dynamics or small state spaces [114, 180]. This limitation is caused mainly by the discretisation of state spaces or use of optimisation methods with poor scalability with respect to the dimension of the system or constraints. Other approaches use sampling-based methods, which have been used in the classical motion planning problem to reduce the complexity of high-dimensional systems [70]. However, as discussed in Chapter 2, most sampling-based methods require a steering function to expand a graph structure until a path is found [76, 137]. Current methods require the use of exact steering functions to achieve optimality [78, 179]. This function solves a two-point boundary value problem [101] to optimally drive a system from one state to another. However, finding a solution to this problem is not easy for certain kinodynamic systems [183].

The method proposed in this chapter is based on sampling methods. Specifically, it is based on a sampling-based method, called SST [108] that does not require such exact

steering functions to find optimal trajectories. This property allows the applicability of the approach to a broader range of system dynamics. However, the SST method is limited to the task of driving the system from an initial state to a final one. The solution in this chapter generalises the SST approach to accept temporal logic specification while the properties of the original approach, such as the optimality of the trajectory, are maintained. Therefore, the main contribution of this chapter is a method that finds optimal trajectories, subject to temporal logic specifcations, for kinodynamic systems.

The proposed method creates a graph that contains valid transitions between states. This graph is iteratively expanded by adding more states and transitions. During each iteration, it is verified if the graph contains a trajectory satisfying the sc-LTL specification. Moreover, the quality of the trajectory with respect to a cost function is improved as more states are added to the graph. On the other hand, recall from Chapter 2 that most solutions to the problem of optimal motion planning present some type of relaxation, e.g local optimality, or approximation due to the complexity of the problem. The method is this chapter offers asymptotic optimality. In other words, the solution found by the method approximates the optimal one as the number of iteration tends to infinity.

The rest of this chapter is divided as follows. First, the addressed problem is formally formulated in Section 4.1. Then, in Section 4.2, the proposed solution is presented in detail. An analysis in terms of optimality and complexity is presented in Section 4.3. To illustrate the proposed method examples considering a 10-dimensional quadrotor are presented in Section 4.4. Finally, conclusions are presented in Section 4.5.

4.1 Problem formulation

This chapter focuses on deterministic dynamic systems, Γ, that evolve according to the differential equation:

$$\dot{x}(t) = f(x(t), u(t)), \tag{4.1}$$

where $x(t) \in X \subseteq \mathbb{R}^{d_x}$ and $u(t) \in U \subseteq \mathbb{R}^{d_u}$ are the system state and control input at time t. The sets X and U are assumed to be compact and the system to be Lipschitz continuous for both of its arguments. $\mathbb{R}^{d_x}$, $\mathbb{R}^{d_u}$ are the d_x-dimensional and d_u-dimensional Euclidean spaces, respectively. The subset of X where the system collides with an obstacle is denoted by X_{obs}. On the other hand, the collision-free subset of X, i.e. $X \setminus X_{\text{obs}}$, is denoted by X_{free}. The closed ball of radius r centred at $x \in X$ is denoted by $B_r(x)$. A trajectory of Γ is a function $\boldsymbol{x} : [0, \tau] \to X_{\text{free}}$, where τ is the duration. The system operates in a static workspace $\mathcal{W}$. With slight abuse of notation, $\mathcal{W}(x)$ is used to define the projection of a state $x \in X$ onto the workspace.

The results presented in this chapter depend on the following definitions from [108].

Definition 4.1.1. (δ-similar trajectories) *Two trajectories, $\boldsymbol{x}$ and $\boldsymbol{x}'$, are δ-similar if for a continuous function $\alpha : [0,\tau] \to [0,\tau']$, the condition $\boldsymbol{x}'(\alpha(t)) \in B_\delta(x(t))$ holds for all $t \in [0,\tau]$.*

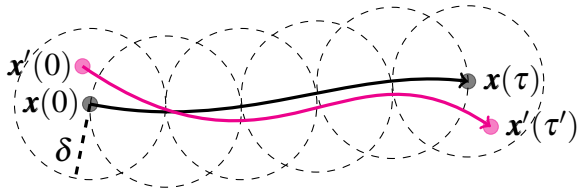


Fig. 4.1 Illustration of two δ-similar trajectories, $\boldsymbol{x}$ and $\boldsymbol{x}'$.

Definition 4.1.2. (Obstacle clearance) *The obstacle clearance ε of a trajectory $\boldsymbol{x}$ is the minimum distance from obstacles over all the states. Formally, the obstacle clearance is defined as:*

$$\varepsilon = \inf_{t\in[0,\tau], x_{obs}\in X_{obs}} \|\boldsymbol{x}(t) - x_{obs}\|. \tag{4.2}$$

Definition 4.1.3. (Chow's theorem [152]) *Let V be a neighbourhood of a state $x \in X \subseteq \mathbb{R}^{d_x}$ and let $R^V(x,T)$ be the set of reachable states at time T by trajectories remaining inside V. A system is small-time locally accessible from x if $R^V(x,\leq T)$ contains a full d_x-dimensional subset of X for all $T > 0$ and all neighbourhoods V.*

In the definition above, *small-time* and *locally* indicate that the property hold for any $T > 0$ and any arbitrarily small room around the state x, respectively. An illustration of the small-time locally accessible property is shown in Fig. 4.2.

Definition 4.1.4. (Dynamic clearance) *Given the system Γ satisfying Chow's condition, there exists a value δ_c, called dynamic clearance, such that for all $\delta \in (0,\delta_c]$, $x \in B_\delta(\boldsymbol{x}(0))$ and $x' \in B_\delta(\boldsymbol{x}(\tau))$, there exists a trajectory $\tilde{\boldsymbol{x}}$ such that (i) $\tilde{\boldsymbol{x}}(0) = x$ and $\tilde{\boldsymbol{x}}(\tau') = x'$; and (ii) $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$ are δ-similar.*

Definition 4.1.5. (δ-robust trajectory) *A trajectory $\boldsymbol{x}$ is called δ-robust if its dynamic clearance δ_c and obstacle clearance ε are greater than δ.*

The solution presented in this chapter is a sampling-based method. These methods incrementally explore the state space of the system to find a trajectory from an initial state to a goal region. As more trajectories are computed with each iteration, sampling-based algorithms improve the quality of the returned solution. This property of the algorithms is called asymptotic optimality.

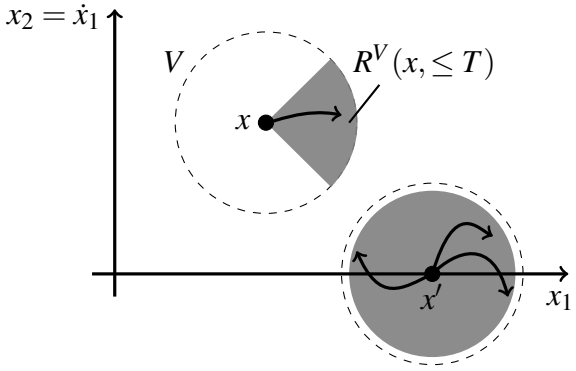


Fig. 4.2 Two initial states, x and x', of a double integrator. The system cannot reach a state on the left of the initial state x without leaving $R^V(x, \leq T)$. Therefore, the system is small-time locally accessible from the state x with $x_2 \neq 0$. On the other hand, from the state x' with $x_2 = 0$, the system is small-time locally controllable, i.e. $R^V(x', \leq T)$ contains a neighbourhood of x' for all neighbourhoods V and all $T > 0$. (Adapted from [31])

Definition 4.1.6. (Asymptotic optimality) *Let A^n be an algorithm that iteratively creates a graph of feasible trajectories of Γ, where n is the number of iterations. The set of all trajectories found by A^n is denoted by χ^n. An algorithm is said to be asymptotically optimal if the following condition holds:*

$$\lim_{n\to\infty} \min_{\boldsymbol{x}\in\chi^n} c(\boldsymbol{x}) \to c^*, \tag{4.3}$$

where c is a cost function applied to a trajectory $\boldsymbol{x} \in \chi^n$ and c^* denotes the minimum cost over all possible trajectories.

Let $\boldsymbol{x} \oplus \boldsymbol{x}'$ be the concatenation of trajectories $\boldsymbol{x}$ and $\boldsymbol{x}'$ such that $\boldsymbol{x}(\tau) = \boldsymbol{x}'(0)$. The cost function is assumed to be Lipschitz continuous and satisfies the following conditions:

- Additivity: $c(\boldsymbol{x} \oplus \boldsymbol{x}') = c(\boldsymbol{x}) + c(\boldsymbol{x}')$.
- Monotonicity: $c(\boldsymbol{x}) \leq c(\boldsymbol{x} \oplus \boldsymbol{x}')$.
- Non-degeneracy: For all $t' > t \geq 0$, there exist $M > 0$ such that $t' - t \leq M \cdot |c(\boldsymbol{x}(t')) - c(\boldsymbol{x}(t))|$.

Note that the cost function c is only a function of the state x. This a requirement to proof asymptotic optimality based on the parameter δ which depends on the distance between trajectories in the state space and distance to obstacles. Moreover, this requirement is also necessary for the expansion of a tree that represents the motion of the system, see Section

4.2. Although the type of admitted cost functions is limited by this condition, this is a common requirement in most asymptotical optimal sampling-based methods [77].

As mentioned in the introduction of this chapter, the proposed method finds optimal trajectories subject to sc-LTL specifications, see Chapter 3. Let Π be a set of atomic propositions associated with regions of the workspace $\mathcal{W}$ and $L : X \to 2^{\Pi}$ be a function mapping a state x to the atomic propositions satisfied by the projection of x onto the workspace, i.e. $\mathcal{W}(x)$. Given a trajectory $\boldsymbol{x}$, a discrete finite word over 2^{Π} can be obtained as follows. Let $\Delta(\boldsymbol{x}) = \{t_i \mid L(x(t_i)) \neq lim_{k \to t_i^-} L(x(k))\}$ be the set of discontinuities in the labelled trajectory $L(\boldsymbol{x})$. This set $\Delta\boldsymbol{x} = \{t_1, \ldots, t_n\}$ produces a timed word $w_\Delta = (L(x(t_0)), d_0)(L(x(t_1)), d_1) \ldots (L(x(t_n)), d_n)$, where $t_0 = 0$, $d_i = t_{i+1} - t_i$ for all $0 \leq i < n$ and $d_n = \tau - t_n$. From w_Δ, a finite word $w = L(x(t_0))L(x(t_1) \ldots L(x(t_n))$ that represents the atomic propositions satisfied during the trajectory $\boldsymbol{x}$ can be obtained. This type of discretisation has been used in the literature [25, 136]. A trajectory $\boldsymbol{x}$ satisfies an sc-LTL specification φ if the word w, produced by $\boldsymbol{x}$, is accepted by a Büchi automaton $\mathcal{B}_\varphi$. This condition is denoted by $L(\boldsymbol{x}) \models \varphi$. The problem addressed in this chapter is now formally defined.

Problem definition 4.1.1. *Given a system Γ and a sc-LTL specification φ, the objective is to find an asymptotically optimal algorithm A^n such that $L(\boldsymbol{x}^*) \models \varphi$, where $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in \chi^n} c(\boldsymbol{x})$.*

In contrast to the traditional point to point motion planning for kinodynamic systems, a sc-LTL specification can indicate several regions of the workspace $\mathcal{W}$ that must be reached by the system. Given this specification, the method must find an optimal trajectory satisfying such a specification.

4.2 Solution

4.2.1 Overview

The proposed solution is based on a new approach, called stable sparse-RRT (SST) [108], that only requires the forward propagation of the system dynamics to find optimal trajectories. This property allows to apply this method to a system with complex dynamics or where the system is simulated using a physics engine. The proposed method iteratively expands a transition system by sampling states from X. The states of the transition system are formed by pairs (x, q), where $x \in X_{\text{free}}$ and $q \in Q$ is a state of the Büchi automaton $\mathcal{B}_\varphi$. At each iteration, a new state and transition is added after verifying if the trajectory is valid in $\mathcal{B}_\varphi$. This process continues until a specified number of iterations is completed. Finally, the optimal trajectory is extracted from the transition system. The described procedure is performed

offline. Once the algorithm finds a solution, the obtained trajectory can be tracked online by the system.

In the following section, the idea of SST is presented before presenting the proposed solution which generalises the SST to consider sc-LTL specifications.

4.2.2 Stable sparse RRT

The SST algorithm is a sampling-based tree motion planner. It iteratively expands a tree $T = (V, E)$, where V is a set of states of Γ and E are edges, or transitions, between states. These transitions define collision-free trajectories of Γ. After an N number of iterations, the returned tree contains an optimal trajectory from an initial state x_0 to a goal region $X_G \subset X_{\text{free}}$.

Before explaining the expansion of the tree T, the following notation is required. The trajectory from state x to state x' is denoted by $\boldsymbol{x}(x, x')$ and its cost by $c(\boldsymbol{x}(x, x'))$. The cost of transitioning from the initial state x_0 to state x, called cost-to-go, is represented by $c_{x_0}(x)$. Given two states x and x' connected by an edge $(x, x') \in E$, x is called the parent of x' and x' is the child of x.

In order to expand T, the SST algorithm divides the states in V into two sets, V_{active} and V_{inactive}. Intuitively, the states in V_{active} are the states with the lowest cost-to-go in a neighbourhood defined by some states w called witnesses. On the other hand, the states in V_{inactive} do not have the lowest cost-to-go but have a child in V_{active}, Fig. 4.3. Since the states in V_{active} have the lowest cost-to-go, these are used to expand the tree. Once a state in V_{active} is selected, the tree is expanded by adding a new trajectory starting from the chosen state. After the tree is expanded, it is verified if the last state of the new trajectory has the lowest cost to go in its neighbourhood. If this condition is not true, the trajectory is removed from the tree. Otherwise, the trajectory remains on the tree and any other trajectory in the neighbourhood is removed, Fig. 4.5. By removing unnecessary trajectories, the SST algorithm provides an upper bound on the number of states in T. This is in contrast to other methods where the number of states do not have a bound and only depends on the number of iterations of the algorithm [77].

The functions used by the SST algorithm are now presented.

a) BESTFIRSTSELECTION: Given the set of vertices $V_{\text{active}} \subseteq V$ and a constant $\delta_{BN} > 0$, the function randomly samples a state $x_{\text{rand}} \in X_{\text{free}}$ and returns the state $x_{\text{selected}} \in V_{\text{active}}$ within $B_{\delta_{BN}}(x_{\text{rand}})$ with the minimum cost-to-go, i.e. $\arg\min_{x \in V \cap B_{\delta_{BN}}(x_{\text{rand}})} c_{x_0}(x)$. If $V_{\text{active}} \cap B_{\delta_{BN}}(x_{\text{rand}}) = \emptyset$ then the nearest state in V_{active} to x_{rand} is returned, Fig. 4.4.

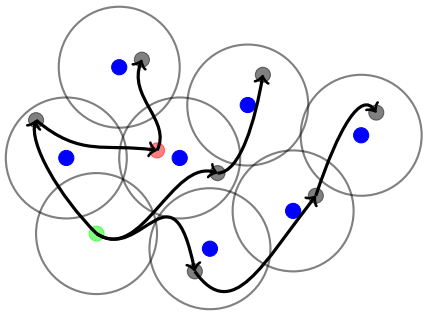


Fig. 4.3 Illustration of SST $T = (V, E)$. The green, red and black disks represent states of the set V while edges in E are represented by the curves between states. States in black form the set $V_{\text{active}} \subset V$ and the red state forms the set $V_{\text{inactive}} \subset V$. States in blue define balls in the state space X. The initial state is shown in green.

(a) (b)

Fig. 4.4 Selection of the best state to expand the tree: (a) based on the cost-to-go and (b) based on the closeness to sampled state x_{rand}.

b) MONTECARLOPROP: Given a state x and a time duration $T_p > 0$; a single control input $u \in U$ and a time $t \in (0, T_p]$ are randomly sampled. Then, the dynamics of the system are propagated forward for t units of time. The generated trajectory $\boldsymbol{x}(x, x_{\text{new}})$ is returned.

c) COLLISIONFREE: Given a trajectory $\boldsymbol{x}(x, x')$ and a set of obstacles Obs, the function returns true if the trajectory lies on X_{free}. Otherwise, the function returns false.

d) LOCALLYBEST: Let W be a set of states in X_{free} called witnesses. For every $w \in W$, there exists a state $x^w_{\text{rep}} \in V$ such that x^w_{rep} has the lowest cost-to-go within the ball $B_{\delta_s}(w)$. The state x^w_{rep} is called the representative of w. Then, given a state x', a set W and a constant $\delta_s > 0$, the function LOCALLYBEST searches for the closest witness w_{close} to x'. If the distance between x' and w_{close} is bigger than δ_s, the function returns true. Moreover, if the distance is less than δ_s and the cost $c_{x_0}(x')$ is less than the cost of the representative of w_{close}, i.e. $c_{x_0}(x') < c_{x_0}(x^{w_{\text{close}}}_{\text{rep}})$, the function also returns true. Otherwise, it returns false.

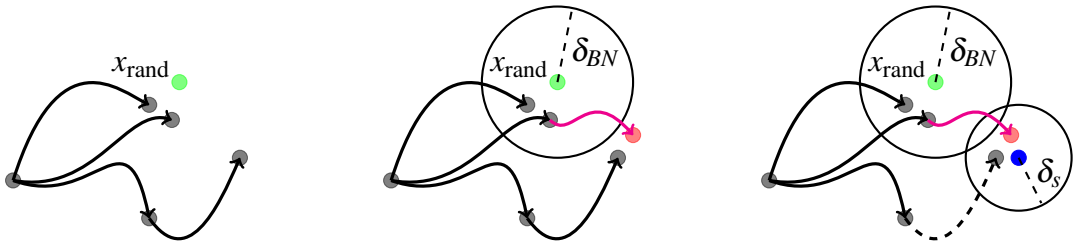
Intuitively, the function returns true only if x' has the lowest cost-to-go in the neighbourhood of size δ_s defined by the closest witness.

e) PRUNENODES: When a new state x is added to T, the function PRUNENODES moves the old representative of the ball $B_{\delta_s}(w)$, containing x, from V_{active} to V_{inactive}. Then, the state x becomes the representative of the neighbourhood defined by the ball. Finally, the function checks if any of the states x' in V_{inactive} has a child in V_{active}. If that condition is not satisfied, the state x' and edges to it are removed from T, Fig. 4.5.

Algorithm 4.1. SST$(\Gamma, X, x_0, U, T_p, N, \delta_{BN}, \delta_s, Obs)$

1: $V_{\text{active}} \leftarrow x_0$, $V_{\text{inactive}} \leftarrow \emptyset$, $E \leftarrow \emptyset$;
2: $w_0 \leftarrow x_0, W \leftarrow w_0$;
3: **for** $i = 1$ to N **do**
4: $\quad x_{\text{selected}} \leftarrow \text{BESTFIRSTSELECTION}(X_{\text{free}}, \Gamma, V_{\text{active}}, \delta_{BN})$;
5: $\quad x_{\text{new}} \leftarrow \text{MONTECARLOPROP}(x_{\text{selected}}, U, T_p)$;
6: $\quad$ **if** $\text{COLLISIONFREE}(\boldsymbol{x}(x_{\text{selected}}, x_{\text{new}}), Obs)$ **then**
7: $\quad\quad$ **if** $\text{LOCALLYBEST}(x_{\text{new}}, W, \delta_s)$ **then**
8: $\quad\quad\quad V_{\text{active}} \leftarrow V_{\text{active}} \cup \{x_{\text{new}}\}$;
9: $\quad\quad\quad E \leftarrow E \cup \{\boldsymbol{x}(x_{\text{selected}}, x_{\text{new}})\}$;
10: $\quad\quad\quad V_{\text{active}}, V_{\text{inactive}}, E \leftarrow \text{PRUNENODES}(x_{\text{new}}, V_{\text{active}}, V_{\text{inactive}}, E)$;
11: **return** $T = (V, E)$;

The SST algorithm, Alg. 4.1, is now described. The tree T is initialised with the initial state of the system x_0 (line 1). This vertex becomes the representative of the witness located at the same state (line 2). Then, the following procedure is repeated $N > 0$ times. The function BESTFIRSTSELECTION is called to select a state $x_{\text{selected}} \in V_{\text{active}}$ (line 4). From this state, a single control input u is forward propagated t units of time using the function MONTECARLOPROP. This propagation generates a trajectory $\boldsymbol{x}(x_{\text{selected}}, x_{\text{new}})$ (line 5). Then, this trajectory is checked for collision (line 6). If no collision is detected, the function LOCALLYBEST is called to verify if x_{new} has the lowest cost-to-go within the neighbourhood defined by the closest witness (line 7). If x_{new} is locally the best, it is added to the set V_{active} with the transition $\boldsymbol{x}(x_{\text{selected}}, x_{\text{new}})$ (lines 8-9). Otherwise, the trajectory $\boldsymbol{x}(x_{\text{selected}}, x_{\text{new}})$ is discarded. Finally, the function PRUNENODES is called to promote the new state x_{new} as a representative of its neighbourhood and to remove the states in V_{inactive} that are not necessary (line 10). By removing states and transitions from T, the SST maintains a sparse data structure. In other words, since only one state, the one with the lowest cost, is maintained in each of the neighbourhoods defined by the witnesses; and the minimum distance between witnesses is defined by δ_s, the number of states in the tree is upper bounded. An illustration of the expansion and pruning operations is presented in Fig. 4.5.



(a) (b) (c)

Fig. 4.5 Illustration of expansion and pruning of tree. (a) In order to add a new state to the tree, a new state x_{rand} (green disk) is sampled from the state space. (b) The state with the minimum cost-to-go in the ball of radius δ_{BN} centred at x_{rand} is selected for expansion by adding a new trajectory (red curve). (c) The final state of the trajectory is then compared with the states of the tree within the ball of radius δ_s centred at the closest witness states (blue disk). If the cost-to-go of the new trajectory is better than any other trajectory, the new state and trajectory are added and any previous trajectory without children are pruned from the tree, e.g. the dotted trajectory.

The SST requires the selection of the parameters δ_{BN} and δ_s. These parameters influence the exploration of the state space and the sparsity of the structure, respectively. To guarantee a good performance of the algorithm, these parameters must satisfy the condition [108]:

$$\delta_{BN} + 2\delta_s < \delta,$$

where δ defines the robust clearance. Although the SST algorithm presented above is asymptotically near-optimal, see Section 4.3, asymptotic optimality can be achieved by reducing the radii δ_{BN} and δ_s over time. Intuitively, this reduction allows to select near-optimal states closer to the optimal ones and propagate them.

The SST algorithm provides solutions to the classical point to point motion planning for systems where solving a two-point value boundary problem is not possible or where only a physics engine is available for simulation. This allows its application to a wider range of situations than current sampling-based methods. Nevertheless, the algorithm cannot handle cases where more than one target has to be reached or an order of events is required. In the following section, a solution to these limitations is presented.

4.2.3 Stable sparse RRT with temporal logic constraints

This section presents an algorithm, called SST_LTL, that conserves the asymptotic optimality property of the SST while high-level tasks, expressed as sc-LTL specifications, are satisfied.

Similar to the SST algorithm, the SST_LTL iteratively expands a transition system $\mathcal{T} = (S, s_0, \delta_{\mathcal{T}}, \Pi, L)$ by sampling the state space X to model a subset of the system motion. To verify whether a run $\boldsymbol{s}$ on $\mathcal{T}$ satisfies a sc-LTL specification φ, a product automaton is used.

Definition 4.2.1. (Product automaton) *Let $\mathcal{T} = (S, s_0, \delta_{\mathcal{T}}, \Pi, L)$ be a deterministic transition and $\mathcal{B} = (Q, q_0, \Sigma, \delta_{\mathcal{B}}, F)$ be a deterministic Büchi automaton, where $\Sigma = 2^{\Pi}$. A product automaton $\mathcal{P}$ is a tuple $(S_{\mathcal{P}}, s_{\mathcal{P},0}, \Sigma, \delta_{\mathcal{P}}, F_{\mathcal{P}})$, where:*

- *$S_{\mathcal{P}} = S \times Q$ is a set of finite states,*
- *$s_{\mathcal{P},0} = s_0 \times q_0$ is an initial state,*
- *Σ is a set of atomic propositions,*
- *$\delta_{\mathcal{P}} \subseteq S_{\mathcal{P}} \times S_{\mathcal{P}}$ is a transition relation, where $((s,q),(s',q')) \in \delta_{\mathcal{P}}$ iff $(s,s') \in \delta_{\mathcal{T}}$ and $\delta_{\mathcal{B}}(q, L(s')) = q'$,*
- *$F_{\mathcal{P}} = S \times F$ is a set of accepting states.*

A run $\boldsymbol{s_{\mathcal{P}}} = (s_0, q_0)(s_1, q_1)\ldots$ is a sequence of pairs where $((s_i, q_i), (s_{i+1}, q_{i+1})) \in \delta_{\mathcal{P}}$ for all $i \geq 0$. Since sc-LTL specifications are considered, a run $\boldsymbol{s_{\mathcal{P}}}$ on $\mathcal{P}$ is accepted if it reaches the set of accepting states.

By construction, for an accepting run $\boldsymbol{s_{\mathcal{P}}}$ on $\mathcal{P} = \mathcal{T} \times \mathcal{B}_{\varphi}$, its projection $s_0 s_1 \ldots$ onto $\mathcal{T}$ satisfies the specification φ. Conversely, for any run $\boldsymbol{s} = s_0 s_1 \ldots$ on $\mathcal{T}$ satisfying φ, there exists an accepted run $\boldsymbol{s_{\mathcal{P}}} = (s_0, q_0)(s_1, q_2)\ldots$ on $\mathcal{P}$.

Based on the above statement, the problem of satisfying a sc-LTL specification φ can be reformulated as a reachability problem in a product automaton. To construct this product automaton simultaneously with the transition system $\mathcal{T}$, the states of the transition system are augmented with a state of the Büchi automaton to create pairs of the form (x, q), where x is a state of Γ and q is a state in Q. Therefore, a transition $((x,q),(x',q')) \in \delta_{\mathcal{T}}$ is valid if $\boldsymbol{x}(x,x')$ is a collision-free trajectory and $\delta_{\mathcal{B}}(q, L(x')) = q'$.

The transition system $\mathcal{T}$ grows similar to the tree in the SST algorithm with the following variation. At each iteration, a state (x, q) for all $q \in Q$ is expanded with a new trajectory $\boldsymbol{x}(x,x')$. This trajectory is validated using the word w generated by it as an input to the Büchi automaton. After the last iteration of the algorithm, the state formed by a state $q \in Q_F$ with the lowest cost-to-go is found and used to get the optimal trajectory.

In the rest of the section, the functions used in the SST_LTL algorithm are firstly presented, followed by the explanation of the algorithm. During the description of the functions, with abuse of notation, $c_{s_0}(\cdot)$ and $\boldsymbol{x}(\cdot,\cdot)$ are used for the states of $\mathcal{T}$, i.e. $c_{s_0}(s) = c_{x_0}(x)$ and $\boldsymbol{x}(s,s') = \boldsymbol{x}(x,x')$, where $s = (x,q)$ and $s' = (x',q')$.

a) BESTSELECTIONLTL: The function divides the states in S_{active} into sets $\{S_{q_i}\}_{i=1}^{|Q|-1}$ depending on the Büchi state forming the state s. Then, the function samples a system state $x_{\text{rand}} \in X_{\text{free}}$ and for each set, a state is selected following the procedure described in paragraph 4.2.2.a. The function is presented in Alg. 4.2.

Algorithm 4.2. BESTSELECTIONLTL($X_{\text{free}}, S_{\text{active}}, \delta_{BN}, \mathcal{B}$)

```
1: x_random ← SAMPLE(X_free);
2: for i = 1 to |Q| − 1 do
3:     S_{q_i} ← {s ∈ S_active : s = (x, q_i)}
4:     if S_{q_i} ≠ ∅ then
5:         S_near ← NEAR(x_random, S_{q_i}, δ_BN);
6:         if S_near ≠ ∅ then
7:             s_{q_i} ← arg min_{s∈S_near} c_{s_0}(s);
8:         else
9:             s_{q_i} ← NEAR(x_random, S_{q_i}, X);
10: return {s_{q_i}}_{i=1}^{|Q|−1}
```

b) RUNBUCHI: Given a Büchi automaton $\mathcal{B}$, state q and a word w_Δ, the function RUNBUCHI (Alg. 4.3) returns the last state q_{final} of the run $\boldsymbol{q}$ on $\mathcal{B}$, starting from state q, generated by w_Δ.

Algorithm 4.3. RUNBUCHI($\mathcal{B}, w_\Delta, q_{\text{init}}$)

```
1: for i = 1 to LENGTH(w_Δ) do
2:     q_final ← δ_B(q_init, w_{Δ,i});
3:     if q_final ≠ ∅ then
4:         q_init ← q_final;
5:     else
6:         return ∅
7: return q_final
```

c) LOCALLYBESTLTL: Recall that the states in W define neighbourhoods in X, where the state with the lowest cost-to-go is called the representative of $w \in W$. These neighbourhoods are used to prune states that do not contribute to the lowest-cost path to any point in X. When considering LTL specifications, multiple representatives may exist for each state $w \in W$. That is, each state w can have as many representatives as states in the Büchi

automaton, Fig. 4.6. Formally, a state $s = (x,q)$ is a representative of w, denoted as $s_{\text{rep}}^{w,i}$, if $s = \arg\min_{(x \in B_{\delta_s}(w),\, q=q_i)} c_{s_0}(s)$. The function LOCALLYBESTLTL returns true or false following the conditions presented in paragraph 4.2.2.d with the difference that only the representatives formed by the Büchi state in turn are considered. The function is presented in Alg. 4.4.

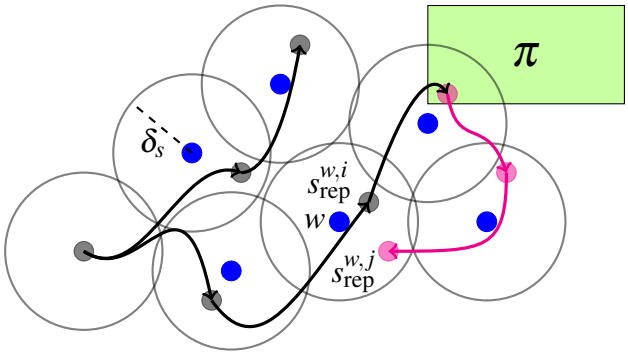


Fig. 4.6 Neighbourhoods of radius δ_s defined by witnesses (blue disks) and area associated with the atomic proposition π. The colour change in the path shows a change in the Büchi state. The witness w has two representatives, $s_{\text{rep}}^{w,i}$ and $s_{\text{rep}}^{w,j}$, corresponding to the Büchi states q_i and q_j, respectively.

Algorithm 4.4. LOCALLYBESTLTL$(s_{\text{new}}, W, X, \delta_s, q)$

```
1: w_close ← NEAR(s_new, W, X);
2: if DIST(s_new, w_close) ≤ δ_s then
3:     if c_{s_0}(s_new) < c_{s_0}(s_rep^{w_close,q}) then
4:         return True;
5:     else
6:         return False;
7: else
8:     return True;
```

d) PRUNENODESLTL: Given a state s_{new}, the sets S_{active}, S_{inactive} and a Büchi state q; the function PRUNENODESLTL, Alg. 4.5, moves the current representative $s_{\text{rep}}^{w,q}$ of the ball $B_{\delta_s}(w)$ from S_{active} to S_{inactive}. Then, the state s_{new} becomes the representative of the neighbourhood. Finally, the function checks if any of the states s' in S_{inactive} has a child in S_{active}. If that condition is not satisfied, the state s' and the transitions to it are removed from $\mathcal{T}$.

A detailed presentation of the SST_LTL algorithm, Alg. 4.6, is now presented. The transition system $\mathcal{T}$ is initialised with the state $s_0 = (x_0, q_0)$, where x_0 is the initial state of

Algorithm 4.5. PRUNENODESLTL($s_{\text{new}}, S_{\text{active}}, S_{\text{inactive}}, W, X, \delta_{\mathcal{T}}, q$)

1: $w_{\text{close}} \leftarrow \text{NEAR}(s_{\text{new}}, W, X)$;
2: $S_{\text{active}} \leftarrow S_{\text{active}} \setminus s_{\text{rep}}^{w_{\text{close}},q}$;
3: $S_{\text{inactive}} \leftarrow S_{\text{inactive}} \cup s_{\text{rep}}^{w_{\text{close}},q}$;
4: $s_{\text{child}} \leftarrow s_{\text{rep}}^{w_{\text{close}},q}$;
5: **while** $\text{CHILD}(s_{\text{child}}) = \emptyset$ and $s_{\text{child}} \in S_{\text{inactive}}$ **do**
6: $\quad s_{\text{parent}} \leftarrow \text{PARENT}(s_{\text{child}})$;
7: $\quad S_{\text{inactive}} \leftarrow S_{\text{inactive}} \setminus s_{\text{child}}$;
8: $\quad \delta_{\mathcal{T}} \leftarrow \delta_{\mathcal{T}} \setminus \boldsymbol{x}(s_{\text{parent}}, s_{\text{child}})$;
9: $\quad s_{\text{child}} \leftarrow s_{\text{parent}}$;
10: $s_{\text{rep}}^{w_{\text{close}},q} \leftarrow s_{\text{new}}$;

the system and q_0 is the initial state of the Büchi automaton of the specification φ (line 1). The set of witnesses W is also initialised with the initial state x_0 (line 2). Then, the following procedure is repeated N times. The function BESTSELECTIONLTL returns one state s_{selected}^j for each state in the Büchi automaton (line 4), Fig. 4.7. This set of states is used to expand the transition system $\mathcal{T}$ as follows. First, the function MONTECARLOPROP, presented in paragraph 4.2.2.b, is called to generate a new trajectory starting from s_{selected}^j (line 6). This trajectory is denoted as $\boldsymbol{x}(s_{\text{selected}}^j, x_{\text{new}})$. An important difference with respect to the original SST algorithm is that trajectories are restricted to those that intersect the boundary of a region in the workspace at most once, i.e. $|\Delta(\boldsymbol{x})| \leq 1$. By using this restriction, only the last state of the trajectory must be checked to verify the satisfaction of a specification. If $\boldsymbol{x}(s_{\text{selected}}^j, x_{\text{new}})$ is collision-free (line 7), the function RUNBUCHI is called to verify whether the word w_Δ, generated by $\boldsymbol{x}(s_{\text{selected}}^j, x_{\text{new}})$, is valid in the Büchi automaton $\mathcal{B}_\varphi$ (line 9). The function RUNBUCHI returns the state q_{final} reached in $\mathcal{B}_\varphi$. A new state $s_{\text{new}} = (x_{\text{new}}, q_{\text{final}})$ is created with the last state of the trajectory and the reached Büchi state (line 11). This state is then compared with the states in $\mathcal{T}$ to verify whether it is useful to generate an optimal path. This operation is performed by the function LOCALLYBESTLTL (line 12). If the new state s_{new} is accepted, it is added to $\mathcal{T}$ (lines 13-14). Finally, the function PRUNENODESLTL is called to remove the states and transitions that are not required for any optimal path (line 15).

Once the SST_LTL algorithm returns the transition system $\mathcal{T}$, after N iterations, the set $S_{Q_F} = \{(x,q) \in S_{\text{active}} : q \in Q_F\}$ is computed. If the set is empty, the specification cannot be satisfied by $\mathcal{T}$. Otherwise, the state $s_{\text{best}} = \arg\min_{s \in S_{Q_F}} c_{s_0}(s)$ is found. The optimal path, satisfying the sc-LTL specification φ, can then be found by recursively computing the parent of each state starting from s_{best}. Note that the method is limited to sc-LTL specifications due to the tree structure created by the algorithm. While other methods create graphs to satisfy

Algorithm 4.6. SST_LTL($\Gamma, X, x_0, U, \mathcal{T}, T_p, N, \delta_{BN}, \delta_s, \mathcal{B}_\varphi$)

$S_{\text{active}} \leftarrow (x_0, q_0)$, $S_{\text{inactive}} \leftarrow \emptyset$, $\delta_{\mathcal{T}} \leftarrow \emptyset$;
$w_0 \leftarrow x_0, W \leftarrow w_0, i \leftarrow 0$;
while $i < N$ **do**
  $\{s^j_{\text{selected}}\}_{j=1}^{|Q|-1} \leftarrow$ BESTSELECTIONLTL$(X_{\text{free}}, S_{\text{active}}, \delta_{BN}, \mathcal{B}_\varphi)$;
  for $j = 1$ to $|Q| - 1$ **do**
    $\boldsymbol{x}(s^j_{\text{selected}}, x_{\text{new}}) \leftarrow$ MONTECARLOPROP$(s^j_{\text{selected}}, \Gamma, U, T_p)$;
    if COLLISIONFREE$(\boldsymbol{x}(s^j_{\text{selected}}, x_{\text{new}}), Obs)$ **then**
      $q_{\text{init}} \leftarrow q_j$;
      $q_{\text{final}} \leftarrow$ RUNBUCHI$(\mathcal{B}_\varphi, L(\boldsymbol{x}(s^j_{\text{selected}}, x_{\text{new}})), q_{\text{init}})$;
      if q_{final} is valid **then**
        $s_{\text{new}} = (x_{\text{new}}, q_{\text{final}})$;
        if LOCALLYBESTLTL$(s_{\text{new}}, W, \delta_s, q_{\text{final}})$ **then**
          $S_{\text{active}} \leftarrow S_{\text{active}} \cup \{s_{\text{new}}\}$;
          $\delta_{\mathcal{T}} \leftarrow \delta_{\mathcal{T}} \cup \{\boldsymbol{x}(s^j_{\text{selected}}, s_{\text{new}})\}$;
          PRUNENODESLTL$(s_{\text{new}}, S_{\text{active}}, S_{\text{inactive}}, \delta_{\mathcal{T}}, q_{\text{final}})$;
          $i \leftarrow i + 1$;
return $\mathcal{T} = (S, s_0, \delta_{\mathcal{T}})$;

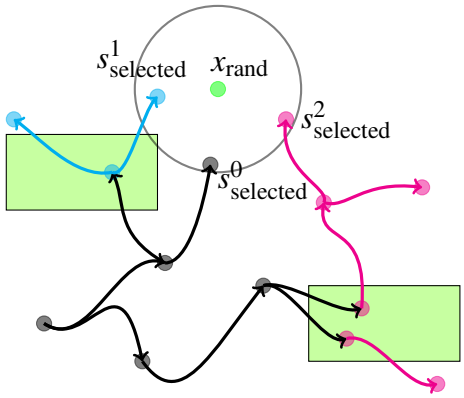


Fig. 4.7 Expansion of transition system using SST_LTL. States $\{s^i_{\text{selected}}\}_{i=0}^{2}$ of the transition system with different Büchi state components (colours) are selected to generate new trajectories from these states.

specifications with infinite horizon properties given as full LTL, they can be applied to a limited type of systems.

4.3 Analysis

This section analyses the SST_LTL algorithm in terms of optimality, completeness and complexity. Before presenting the analysis, some additional definitions from [108] are introduced.

Definition 4.3.1. (δ-robust feasible motion planning problem) *Given a system Γ, an initial state $x_0 \in X$, a goal region $X_G \subset X$ and that a δ-robust trajectory connecting x_0 with a state $x_f \in X_G$ exists, find a trajectory $\boldsymbol{x}$ such that $\boldsymbol{x}(0) = x_0$ and $\boldsymbol{x}(\tau) = x_f$.*

Definition 4.3.2. (Probabilistic δ-robust completeness) *An algorithm A^n is probabilistic δ-robust complete if for any δ-robust feasible motion planning problem Ξ, the following condition holds:*

$$\liminf_{n\to\infty} P(\exists\, \boldsymbol{x} \in \chi^n : \boldsymbol{x} \text{ is a solution to } \Xi) = 1. \tag{4.4}$$

Definition 4.3.3. (Asymptotic δ-robust near-optimality) *Given a function $h : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ of the optimum cost and the δ clearance, where $h(c^*, \delta) \geq c^*$, an algorithm A^n is asymptotic δ-robust near-optimal if for all runs:*

$$P(\limsup_{n\to\infty}(\arg\min_{\boldsymbol{x}\in\chi^n} c(\boldsymbol{x})) \leq h(c^*, \delta)) = 1. \tag{4.5}$$

4.3.1 Probabilistic Completeness and Asymptotic Optimality

The probabilistic completeness and asymptotic optimality of the SST_LTL algorithm are now discussed.

Theorem 4.3.1. (see Theorem 29 in [108]) *Stable sparse-RRT is probabilistically δ-robustly complete.*

The proof of Theorem 4.3.1 is based on the proofs that the MONTECARLOPROP and BESTFIRSTSELECTION will eventually generate a δ-similar trajectory to the optimal path with a probability $\rho_\delta > 0$ and select a near-optimal state for propagation with probability $\gamma > 0$, respectively.

Theorem 4.3.2. (see Theorem 30 in [108]) *Stable sparse-RRT is asymptotically δ-robust near-optimal.*

Again, the proof of Theorem 4.3.2 follows from the positive probability ρ_δ of SST generating a δ-similar trajectory to the optimal one and the conditions assumed for the cost function, see Section 4.1.

Corollary 4.3.1. *The SST_LTL algorithm is probabilistically δ-robustly complete and asymptotically δ-robust near-optimal.*

Proof (Sketch). From Theorems 4.3.1 and 4.3.2, it is guaranteed that, starting from the initial state (root of the tree), a δ-similar trajectory to the optimal one will be generated with a positive probability, if one exists, to any region of the state space. On the other hand, every change in the Büchi state generates a root of a new tree that is only affected by trees having the same Büchi state in the root, see paragraph 4.2.2.d. Hence, from each new root, δ-similar trajectories to the optimal ones are generated. Therefore, the trajectory satisfying the specification, returned by the SST_LTL algorithm, can be seen as a concatenation of δ-robust near optimal trajectories, with different Büchi states, resulting in a longer trajectory with the same property. □

To illustrate Corollary 4.3.1, consider the case where the system, starting from the origin, has to visit the regions associated with the atomic propositions π_1 and π_2 to satisfy a specification. For simplicity, assume that the tree with root (0,0) generates only 3 optimal trajectories to the region π_1. The end of each trajectory is a root for a tree with δ-similar trajectories to the optimal ones to any region of the state space. Hence, to satisfy the specification, the system has to follow the optimal path from the origin to the region π_1 and then follow the trajectory with lowest cost to the region π_2, Fig. 4.8.

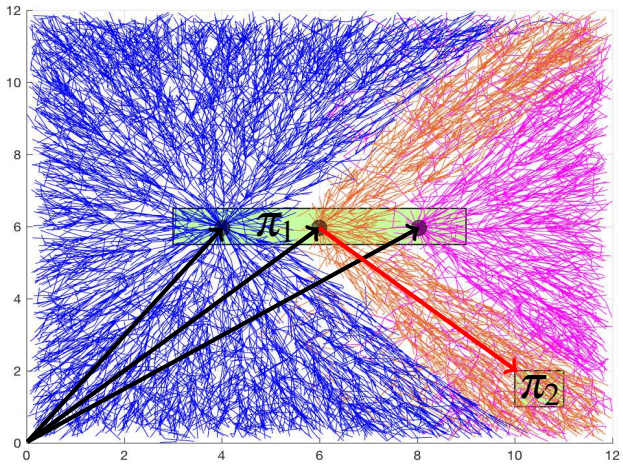


Fig. 4.8 Tree optimal paths (black lines) to region π_1 from the tree rooted at the origin. From each end of a path, a tree grows (each one with a different colour) containing δ-similar trajectories to the optimal ones. Because each tree has states with the same Büchi state, they affect each other, i.e. they do not overlap. The optimal solution is a concatenation of two paths from different trees.

As mentioned in Section 4.2.2, asymptotical optimality can be achieved by reducing the parameters δ_{BN} and δ_s [108].

4.3.2 Complexity

Recall that states are pruned from $\mathcal{T}$ if they are not part of a trajectory leading to a state with the lowest cost in the neighbourhoods defined by δ_s. Moreover, the maximum number of states in each neighbourhood is limited to the number of states in the Büchi automaton. Therefore, if the state space X is bounded, the maximum number of states in $\mathcal{T}$ is finite and bounded by $\mathcal{O}(|Q| \cdot \delta^{-d_x})$. Consequently, operations such as finding nearest neighbours have bounded time complexity in contrast to other methods such as the Rapidly-exploring Random Graph and Rapidly-exploring Random Trees with space complexity $\mathcal{O}(n \log n)$ and $\mathcal{O}(n)$ [77], where n is the number of samples.

4.4 Examples

The proposed method is demonstrated with a 10-dimensional quadrotor with the following dynamics [183]:

$$\dot{x} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/m & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \ell/j & 0 \\ 0 & 0 & \ell/j \end{bmatrix} u, \tag{4.6}$$

where g, m, ℓ and j are constants representing the gravity, the mass of the quadrotor, the distance between the rotors and the centre of the vehicle and the moment of inertia, respectively. The state $x = (p, v, r, w)$ consists of the three-dimensional position p and velocity v; and the two-dimensional orientation r and angular velocity w (the yaw and its derivative are constrained to zero). The control input $u = (u_f, u_x, u_y)$ is formed by the total thrust u_f needed for hovering and the thrust u_x, u_y required for producing roll and pitch, respectively.

The system operates in two different environments, each one with three regions of interest associated with the atomic propositions π_1, π_2 and π_3. For these examples, once the system reaches π_1, it has to reach the regions π_2 and π_3 in that particular order. Formally the specification can be defined as $\varphi = \Diamond(\pi_1 \wedge \bigcirc\Diamond(\pi_2 \wedge \bigcirc\Diamond(\pi_3)))$. The examples are implemented in MATLAB on a computer with a 3.1 GHz i7 processor and 8 GB of RAM.

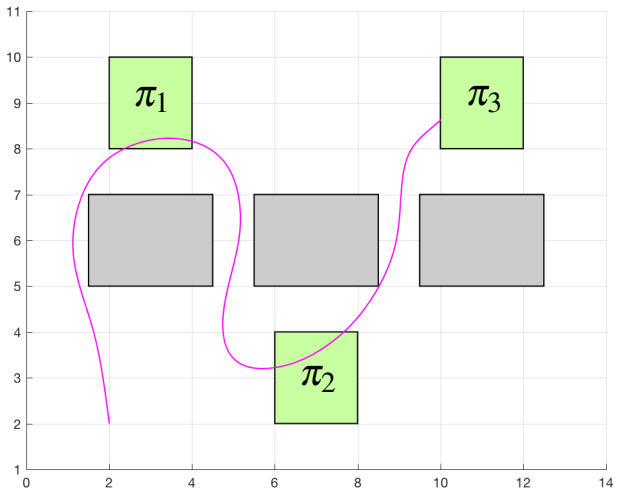


Fig. 4.9 Path followed by quadrotor satisfying the specification φ. The green areas are associated with atomic propositions while the grey areas are considered as obstacles.

To find an optimal trajectory satisfying the specification presented above, the SST_LTL runs for different values of N, i.e. number of iterations, and the Euclidean distance is considered as a cost function. An example of the computed path for each environment is presented in Fig. 4.9 and 4.10, respectively. Figure 4.11 shows that the algorithm constantly improves the quality of the path with the number of iterations. These iterations correspond to the loop in line 5 in Alg. 4.6. In other words, the path converges to the optimal one as the number of states in the transition system increases. While other available sampling-based approaches, e.g. [78, 179], also offer this asymptotical optimality, they require the computation of optimal paths between states using a steering function. This function solves a two-point boundary value problem which is not easy to solve for systems with kinodynamic constraints [183]. In contrast, the proposed method only requires to sample the control space to achieve asymptotical optimality.

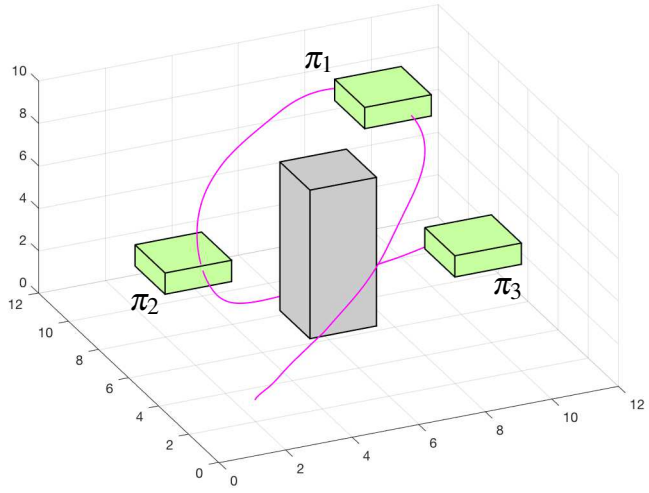


Fig. 4.10 Path followed by quadrotor satisfying the specification φ. The green areas represent regions of interest while the grey cuboid is considered an obstacle.

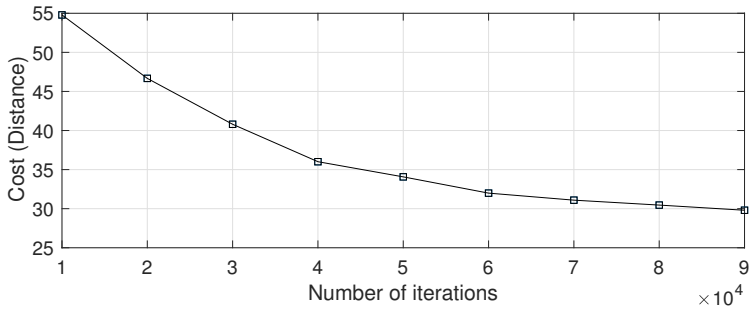


Fig. 4.11 Average (mean) distance travelled by quadrotor v.s. number of iterations.

4.5 Concluding remarks

This chapter has presented a novel sampling-based method to compute asymptotically optimal trajectories for high-dimensional kinodynamic systems constrained by temporal logic specifications. This is a relevant problem due to the frequent existence of multiple paths satisfying high-level specifications. The main advantage of the method compared to other methods is that optimality can be achieved by only forward propagating the system dynamics. Therefore, it does not require a solution to two-point boundary value problems or solving a NP-hard problem such as MILP required in other methods, which cannot be solved for certain systems. On the other hand, due to the complexity of the problem, finding a closed-form solution is not possible. Instead, the proposed method only finds an approx-

imate solution which is asymptotic optimal. Hence, it cannot be compared to methods that solve constrained optimal control problems for which solution to many problems does not exist.

The proposed solution is limited to a fragment of LTL. This limitation is the result of the method used to expand the transition system. However, the method can find solutions to a broader range of dynamic systems compared to other methods. Another advantage is that the sparse structure modelling the motion of the system guarantees a bounded complexity as the number of iterations increases in contrast to other available methods.

On the other hand, a deterministic system is considered in this chapter. Nevertheless, uncertainty in motion is presented in most robotic systems. This uncertainty can be caused by faulty actuators, uneven terrains, wind, etc. Another important element in the method discussed in this chapter is the logic used for the specification. This logic presents the limitation that only qualitative properties can be expressed by using it. This could be a problem for tasks that require the specification of hard time constraints. For example, consider a robot operating with a limited battery. In this situation, a task must be performed in a limited time before the battery discharges. In the next chapter, these aspects are addressed.

Chapter 5

Stochastic Optimal Control with Metric Interval Temporal Logic Specifications

The previous chapter addressed the problem of optimal motion planning subject to qualitative specifications, given as sc-LTL specifications, for deterministic systems. In this chapter, the assumption of deterministic motion is relaxed. Instead, stochastic systems are considered. Moreover, specifications that impose time constraints are also introduced. These two improvements permit to have more complex missions for systems with more realistic motion.

The problem of optimal control for stochastic systems subject to temporal logic constraints has recently attracted attention [49, 65]. This is a relevant topic for mobile systems since errors in modelling and external disturbances are most of the time present. Although solutions considering stochastic systems and temporal logic constraints have been proposed, most of the available solutions focus on this problem at the discrete level [40, 94]. That is, the computation of a MDP modelling the system is assumed. However, the computation of such MDPs is not trivial. Other solutions approach this problem at the continuous level [49, 65]. Nevertheless, solutions that consider a continuous state space and time in the specifications discretise the time and state space to compute optimal policies. This discretisation limits the applicability of the methods to low dimensional systems.

In this chapter, a solution to the problem of computing optimal controllers for stochastic systems under metric interval temporal logic (MITL) specifications is presented. In contrast to LTL specifications, using MITL, specifications with time constraints can be expressed. On the other hand, the optimality of the solution is in terms of maximising the probability of satisfying the specification rather than in terms of a cost function as in the previous chapter. As presented in this chapter, the addition of time constraints increases the computational complexity of the problem. To reduce this complexity, the proposed solution again uses a

sampling-based method to compute optimal policies and a coarse partition of the environment. As a result, the dimensionality of the state space of the system has a less negative impact compared with other available methods. Therefore, the main contribution of this chapter is a method that reduces the complexity of problems considering MITL. This reduction is achieved thanks to a coarse discretisation of the environment and the introduction of a new timed automaton to allow this discretisation.

The rest of the chapter is divided as follows. The addressed problem is formulated in Section 5.1. Then, in Section 5.2, the discretisation of the workspace and the computation of the optimal policy is presented. An analysis of the complexity of the method is presented in Section 5.3. In Section 5.4, the proposed solution is compared with the closest related work in a numerical example. Finally, conclusions are discussed in Section 5.5.

5.1 Problem formulation

This chapter focuses on stochastic dynamic systems, Υ, called controlled diffusion processes, that evolve according to the stochastic differential equation:

$$dx(t) = f(x(t),u(t))dt + D(x(t))dv(t), \tag{5.1}$$

where $x(t) \in X \subseteq \mathbb{R}^{d_x}$, $u(t) \in U \subseteq \mathbb{R}^{d_u}$ and $v(t)$ are the system state, the control input and the d_v-dimensional Wiener process on a probability space $(\Omega, \mathcal{F}, \mathcal{P})$ at time t, respectively. The sets X and U are assumed to be compact. $\mathbb{R}^{d_x}$, $\mathbb{R}^{d_u}$ are the d_x-dimensional and d_u-dimensional Euclidean spaces, respectively. The vector and matrix valued functions $f : X \times U \to \mathbb{R}^{d_x}$ and $D : X \to \mathbb{R}^{d_x \times d_v}$ are measurable, bounded and continuous. Similar to the previous chapter, controllability is not a requirement since a similar approach is used, i.e., sampling of control space and propagation of system dynamics. The matrix $D(\cdot)$ is assumed to have full rank and the control $u(\cdot)$ is admissible with respect to $v(\cdot)$ [92]. Given a continuous sample path $\boldsymbol{x}(\cdot, v)$ of system Υ and an interval $\Delta t_i > 0$, the timed behaviour or run of Υ can be denoted as $\boldsymbol{x}_{\Delta t} = (x_0, \Delta t_0)(x_1, \Delta t_1)\ldots$, where x_i occurs at time $\Sigma_0^{i-1}\Delta t_j$. The system operates in a static workspace $\mathcal{W}$. With slight abuse of notation, $\mathcal{W}(x)$ is used to define the projection of a state $x \in X$ onto the workspace.

This chapter considers MITL specifications, see Section 3.5, built on a set Π of atomic propositions associated with non-overlapping regions of the workspace. A function $L : X \to 2^{\Pi}$ is used to map a state x to the atomic propositions satisfied by the projection of x onto $\mathcal{W}$. A timed word $(w, \eta) = (L(x_0), \Delta t_0)(L(x_1), \Delta t_1)\ldots$, induced by a run $\boldsymbol{x}_{\Delta t}$, gives the behaviour of the system Υ in terms of the atomic propositions satisfied. Let $\mathcal{T}_\varphi = (Q, q_0, \Sigma, C, \Lambda(C), \delta_T, F)$ be a deterministic timed automaton computed from φ. A

sample path of the system satisfies a specification φ, denoted by $L(\boldsymbol{x}_{\Delta t}) \models \varphi$, if the timed word $L(\boldsymbol{x}_{\Delta t})$ is accepted by the timed automaton $\mathcal{T}_\varphi$, i.e. the set of accepting states is reached. The problem addressed in this chapter is now formally defined.

Problem definition 5.1.1. *Given a stochastic dynamic system Υ and a MITL formula φ, compute a control policy $\mu : X \to U$ such that the probability of satisfying φ is maximised.*

Intuitively, the MITL specifications indicate the regions on the workspace $\mathcal{W}$ that must be visited and the time constraints. For instance, a specification $\varphi = \Diamond_{[0,5]}\pi$ indicates that the region associated with the atomic proposition π must be visited within five units of time. Since stochastic systems are considered, the method must return a policy that indicates what control input must be used, based on the current state and time, to maximise the probability of satisfying a specification.

5.2 Solution

5.2.1 Overview

Similar to the previous chapter, the solution of this chapter is based on automata theory. More specifically, a product bounded-parameter Markov decision process (BMDP) is computed to find a policy. However, the addition of time constraints increases the complexity of the problem. Specifically, the size of the product BMDP is affected by the consideration of time. Hence, using an approach similar to the one presented in Chapter 4 will result in an intractable problem due to the large number of states in the structure that models the transitions of the system. To reduce the computational complexity, the proposed solution is this chapter is divided into two phases. In the first phase, the workspace is coarsely discretised into non-overlapping regions. In each of these regions, local policies that drive the system from one region to the adjacent ones are computed, Fig. 5.1. This computation is performed by a sampling-based approach that approximates the properties of the original stochastic system [67]. Once the local policies are computed, a BMDP is constructed to model the probabilities of transitioning between different regions when the policies are applied [117].

In the second phase, a deterministic timed automaton (DTA) is constructed based on the MITL specification. Then, the original problem is reformulated as a reachability problem in the Cartesian product of the BMDP and the DTA. The solution to the reachability problem yields an optimal global policy that selects a local policy for each region of the workspace such that the probability of satisfying the specification is maximised. An advantage of dividing the method into two parts is that only the second phase is recomputed when a new specification is considered in the same workspace.

In the rest of this section, the discretisation and computation of local policies are first presented. Then, the computation of the BMDP and product BMDP is described. Finally, the solution to the reachability problem and the implementation of the global policy are presented.

5.2.2 Workspace discretisation and local policies

Similar to the previous chapter, atomic propositions are associated to regions of interest of the workspace $\mathcal{W}$. This workspace is decomposed into a partition $\mathcal{P} = \{p_i\}_{i=1}^{M}$ by a Delaunay triangulation [104]. Nevertheless, this process can be performed by any other partitioning method. This decomposition is proposition preserving. In other words, for all the states in p_i, the same atomic propositions are true, i.e. for $\mathcal{W}(x) \in p_i$ and $\mathcal{W}(x') \in p_i$, $\pi \in L(x)$ iff $\pi \in L(x')$. The lower dimensionality of the decomposed workspace $\mathcal{W}$ avoids the exponential computational cost of discretising the state space X. In each region p_i for $i \in \{1,\ldots,M\}$, local policies are computed, one for each adjacent region, Fig. 5.1.

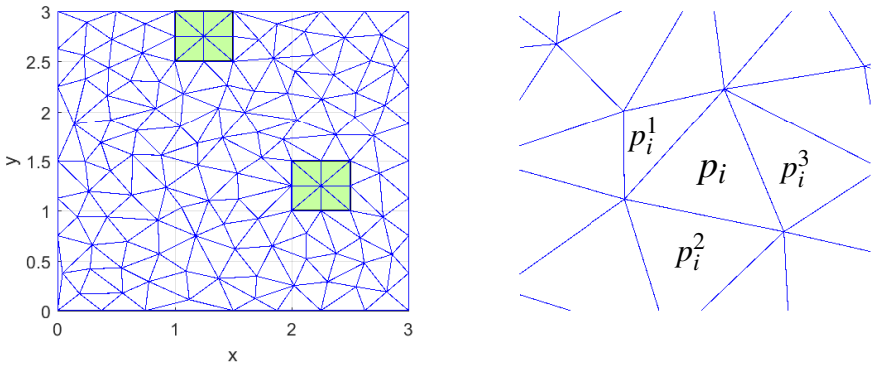


Fig. 5.1 Partitioning of the workspace $\mathcal{W}$. The picture on the left shows the workspace of the system with two areas of interest and the regions generated by the decomposition. In each region p_i, local policies are computed to drive the system from the region p_i to the adjacent regions p_i^1, p_i^2 and p_i^3.

The computation of the local policies is now presented. While any available method could be used to compute a policy that maximises the probability of transitioning from one region to another, a sampling-based method, called incremental Markov decision process (iMDP) [67], is used in this chapter. The selection of this method is due to the benefits that sampling-based methods offer, presented in Chapter 2.

The iMDP algorithm approximates the continuous dynamics of the system by using the idea of the Markov chain method [92]. This approximation is performed by a sequence of MDPs $\mathcal{M}_n = (Z_n, U, P_n, G_n, H_n)$ for $n \geq 0$, where Z_n is a discrete subset of X, U is the

original control space, $P_n(\cdot|\cdot,\cdot) : Z_n \times Z_n \times U \to [0,1]$ gives the probability of transitioning to the state $x_j \in Z_n$ from the state $x_i \in Z_n$ under action $u \in U$, $G_n : Z_n \times U \to \mathbb{R}$ is an immediate cost function and $H_n : Z_n \to \mathbb{R}$ is a terminal cost function, i.e. a cost associated with the final reached state.

In each iteration of the iMDP algorithm, a new $\mathcal{M}_n$ is created by adding randomly sampled states to the set Z_{n-1} from the interior and boundary of a region $p \in \mathcal{P}$, Fig. 5.2. To each state $x \in Z_n$ added to $\mathcal{M}_n$, a non-negative interpolation interval $\Delta t_n(x)$, a cost value $J_n(x)$ and a control $u \in U$ are assigned. Intuitively, the cost value $J_n(x)$ is used to compute a control input for each state x such that the system is driven to a particular area. Before introducing these elements formally, the required conditions to approximate the continuous system are presented.

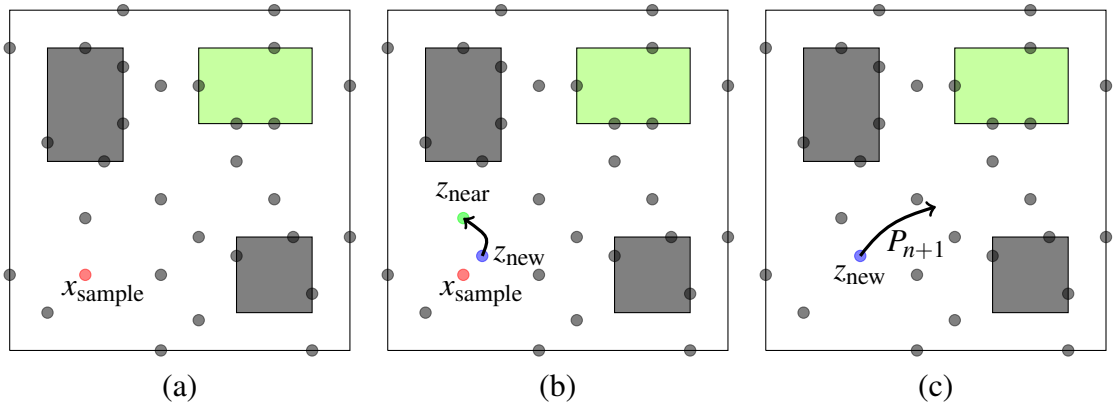


Fig. 5.2 Illustration of an incremental Markov decision process in a workspace with two obstacles (grey areas) and one goal (green area). (a) The states (black disks) forming the MDP $\mathcal{M}_n$ have an assigned cost. This cost is lower in states closer to the target. To add a new state, a state x_{sample} is sampled from the interior of the workspace. (b) The closest state z_{near} in $\mathcal{M}_n$ to the sampled state is found and a new state z_{new} is computed by propagating the system dynamics. (c) After the new state is added to the MDP $\mathcal{M}_{n+1}$, the assigned control is updated based on the cost of the other states in the MDP.

Let $\{\chi_i^n, i \in \mathbb{Z}_+\}$ be a controlled Markov chain on $\mathcal{M}_n$ with transition probability P_n and let $\Delta\chi_i^n = \chi_{i+1}^n - \chi_i^n$ denote the distance between two consecutive states in the Markov chain. In order to maintain the properties of the original system, $\Delta t_n(x)$ and P_n need to satisfy the following local consistency properties [92]:

- For all $x_i \in Z_n$:

$$\lim_{n\to\infty} \Delta t_n(x_i) = 0. \tag{5.2}$$

- For all $x_i \in Z_n$ and $u_i \in U$:

$$\mathbb{E}_{x_i}(\Delta\chi_i^n) = f(x_i,u_i)\Delta t_n(x_i) + \mathcal{O}(\Delta t_n(x_i)), \tag{5.3}$$

$$\mathbb{E}_{x_i}([\Delta\chi_i^n - \mathbb{E}(\Delta\chi_i^n)][\Delta\chi_i^n - \mathbb{E}(\Delta\chi_i^n)]) = D(x_i)D(x_i)^T\Delta t_n(x_i) + \mathcal{O}(\Delta t_n(x_i)), \tag{5.4}$$

$$\lim_{n\to\infty}\sup_{i\in\mathbb{N}}\|\Delta\chi_i^n\| = 0, \tag{5.5}$$

where $\mathbb{E}_{x_i}$ is the conditional expectation given $\chi_i^n = x_i$ and $\mathcal{O}(\cdot)$ indicates an upper bound on the error due to the discrete time approximation.

To approximate the discrete Markov chain $\{\chi_i^n, i \in \mathbb{Z}_+\}$ to the continuous process $x(\cdot)$, the continuous-time interpolations $\xi_x(\cdot)$ and $\xi_u(\cdot)$ are used. These interpolations are defined by:

$$\xi_x(t) = \chi_i^n \quad \xi_u(t) = u_i, \quad t \in [t_i^n, t_i^{n+1}),$$

where $t_i^n = \sum_0^{i-1}\Delta t_n(\chi_i^n)$, Fig. 5.3.

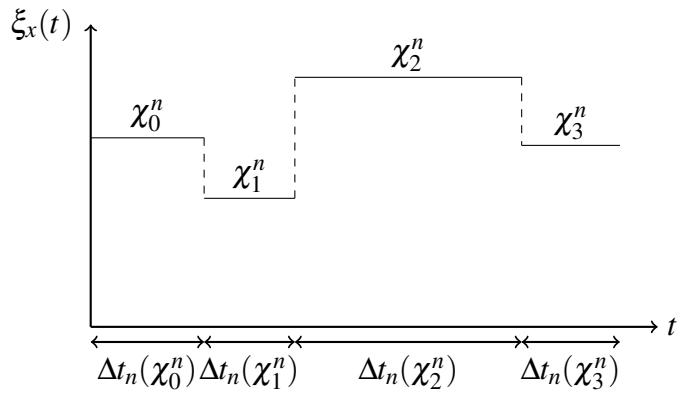


Fig. 5.3 Illustration of the continuous-time interpolation of Markov chain $\{\chi_i^n, i \in \mathbb{Z}_+\}$.

The computation of the control u and cost $J(x)$ assigned to the states of an MDP $\mathcal{M}_n$ is now presented. Recall that each region $p_i \in \mathcal{P}$ requires one policy for each adjacent region in order to drive the system between regions. To compute a policy to drive the system from the interior of p_i to a particular contiguous region, for instance p_i^1, a negative terminal cost is assigned to states sampled from the boundary shared with p_i^1. To avoid the other adjacent regions, p_i^2 and p_i^3, a positive terminal cost is assigned to states sampled from the boundary shared with these regions. As presented below, these terminal costs are used to find control inputs that minimise the cost of a function associated to the states within a region. In other words, the inputs will drive the system to states with the lowest cost, Fig. 5.4.

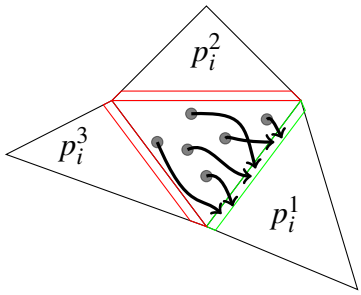


Fig. 5.4 Illustration of transitions from states within p_i to the adjacent region p_i^1. During the computation of the MDP $\mathcal{M}_n$, samples obtained from the border with the desired adjacent region (green region) are assigned with a negative cost. Samples from the red regions are assigned with a positive cost.

Let $\mathbf{U}$ be the set of all possible policies $\mu_n : X \to U$ that map a state $x \in Z_n$ to a control input $u \in U$. To compute a policy such that the probability of reaching an adjacent region is maximised, the following cost-to-go function is minimised [67]:

$$J_{n,\mu_n}(x) = \mathbb{E}_{P_n}\left[\sum_{i=0}^{T_n-1} \alpha^{t_i^n} G_n(\chi_i^n, \mu_n(\chi_i^n)) + \alpha^{t_{T_n}^n} H_n(\chi_{T_n}^n)\right], \tag{5.6}$$

where $\alpha \in [0,1)$ is a discount rate, $\mathbb{E}_{P_n}$ is the conditional expectation given $\chi_0^n = x$ under P_n and T_n is the expected first exit of the controlled Markov chain $\{\chi_i^n, i \in \mathbb{Z}_+\}$ under the policy $\mu_n \in \mathbf{U}$ from region p_i. The cost to go $J_{n,\mu_n}(x)$ is the accumulated cost of each transition, starting from x, and the cost of the reached state under the control $\mu_n(\cdot)$. Since negative and positive costs are assigned to the desired area and obstacles, respectively, by minimising the cost $J_{n,\mu_n}(x)$, the probability of reaching the target is maximised. The optimal policy μ_n^* satisfies $J_{n,\mu_n^*}(x) = \inf_{\mu_n \in \mathbf{U}} J_{n,\mu_n}(x)$.

The policy μ_n^* assigns a control value $\mu_n^*(x)$ to each non-boundary state $x \in Z_n$. This process is repeated to obtain a local optimal policy for each adjacent region in each region of $\mathcal{P}$. That is, a MDP is created in each segment for each adjacent region. To each state of these MDPs, a control is assigned by using the procedure described above. Because of the Delaunay triangulation, the obtained regions are triangles. Therefore, for this decomposition, each region $p_i \in \mathcal{P}$ has three local policies, denoted as $\mu_{p_i}^1$, $\mu_{p_i}^2$ and $\mu_{p_i}^3$, to drive the system from the interior of p_i to its adjacent regions p_i^1, p_i^2 and p_i^3, respectively. Different partitioning would lead to a different number of local policies.

To guarantee the converge of the cost J_n to the optimal one as the number of iterations increases, the interval $\Delta t_n(x)$, also called holding, is computed by:

$$\Delta t_n(x) = \gamma \left(\frac{\log |Z_n|}{|Z_n|} \right)^{\theta \varsigma \rho / d_x}, \tag{5.7}$$

where $\gamma > 0$, $\theta \in (0,1]$, $\varsigma \in (0,1)$ and $\rho \in (0,1]$ are constants. This rate allows to find the optimal cost J_n by solving a Bellman equation via sampling the control space [67].

5.2.3 BMDP model

The probability of transitioning to region p_j from p_i under a policy $\{\mu^l_{p_i}, l \in \{1,2,3\}\}$ varies among the sampled states $x \in Z_n$ within the region p_i. Hence, the probability of transitioning from the region p_i to the region p_j is given by a range. To model all the possible transitions between regions and their range of probabilities, a BMDP $\mathcal{B} = (S, A, \hat{P}, \check{P}, L)$ is used. The set of states S is formed by the segments of the partition $\mathcal{P}$. For ease of explanation, in the rest of the chapter, states $s \in S$ are also referred as regions. The set of actions A contains all the local policies, $\mu^l_{p_i}$ for all $p_i \in \mathcal{P}$ and $l \in \{1,2,3\}$, computed by the iMDP algorithm. The set of available actions at state s is denoted by $A(s)$. The minimum and maximum probability, $\check{P}$ and $\hat{P}$, of transition from one state or region to another are calculated as follows:

$$\check{P}(s_j | s_i, \mu^l_{s_i}) = \min_{x \in Z_n} P(s_j | x, \mu^l_{p_i}), \tag{5.8}$$

$$\hat{P}(s_j | s_i, \mu^l_{s_i}) = \max_{x \in Z_n} P(s_j | x, \mu^l_{p_i}), \tag{5.9}$$

where $P(s_j | x, \mu^l_{p_i})$ is the probability of state x inside region s_i to finish in the region s_j when the local policy $\mu^l_{p_i}$ is applied. Since the Markov chain χ^n_i, induced by the policy $\mu^l_{p_i}$, is absorbing [66], these probabilities can be computed using the fundamental matrix [82], which gives the probability of transitioning from one state to another in a specific number of steps. The label function $L : S \to \Pi$ maps each state x within a region p to the set of atomic propositions Π.

The algorithm to compute a BMDP for a given system Υ and workspace $\mathcal{W}$ is presented in Algorithm 5.1. First, the set of states, actions, minimum and maximum probabilities are initialised as empty sets (line 1). Then, the workspace $\mathcal{W}$ is decomposed into M non-overlapping regions (line 2). To perform such decomposition available tools exist [156]. Since each region corresponds to a state in the BMDP, M states are added to the BMDP (line 3). Once the decomposition is completed, for each region, its adjacent regions are selected (line 5). Then, each of the adjacent regions is selected, one at a time, as a target while the others are considered obstacles for the computation of the local policies (line 7). The iMDP algorithm returns the policy according to the target and obstacle regions and the

probability of transitioning to each adjacent region when the computed policy is applied (line 8). Finally, the computed policies are then added to the set of actions while a matrix with the probabilities of transitioning is added to a set containing probabilities matrices for all the computed policies (line 9).

Algorithm 5.1. BMDPCOMPUTATION$(\Upsilon, X, U, \mathcal{W})$

1: $S \leftarrow \emptyset$, $A \leftarrow \emptyset$, $\check{P} \leftarrow \emptyset$, $\hat{P} \leftarrow \emptyset$;
2: $\mathcal{P} = \{p_i\}_{i=1}^{M} \leftarrow$ DECOMPOSITION$(\mathcal{W})$;
3: $S \leftarrow \{s_i\}_{i=1}^{M}$;
4: **for** each $s_i \in S$ **do**
5: $\quad \{p_i^l\}_{l=1}^{K} \leftarrow$ ADJACENTREGIONS(p_i);
6: $\quad$ **for** $l \in \{1,\ldots,K\}$ **do**
7: $\quad\quad target \leftarrow p_i^l$, $obstacles \leftarrow p_i^j$ for $j \in \{1,\ldots,K\}/l$;
8: $\quad\quad \mu_{s_i}^l, \check{P}_{\mu_{s_i}^l}, \hat{P}_{\mu_{s_i}^l} \leftarrow$ IMDP$(\Upsilon, X, U, target, obstacles)$;
9: $\quad\quad A \leftarrow A \cup \mu_{s_i}^l$, $\check{P} \leftarrow \check{P} \cup \check{P}_{\mu_{s_i}^l}$, $\hat{P} \leftarrow \hat{P} \cup \hat{P}_{\mu_{s_i}^l}$;
10: **return** $\mathcal{B} = (S, A, \check{P}, \hat{P})$;

As explained in Section 5.2.1, one of the benefits of the proposed method is that, if the workspace $\mathcal{W}$ does not change, Alg. 5.1 only needs to be executed one time even if the MITL specification changes. In the next section, the computation of optimal policies that map states of the BMDP to actions from the set A is presented. In order to select an action in each state of the BMDP such that the probability of satisfying a specification φ is maximised, the Cartesian product of the BMDP and the timed automaton obtained from φ is created. This procedure is explained in the next section.

5.2.4 Product BMDP

Using the iMDP method, a discrete model of the original system is obtained. In order to discretise the time for the BMDP abstraction $\mathcal{B}$, a timed automaton $T = (Q, q_0, \Sigma, C, \Lambda(C), \delta_T, F)$ is introduced, see Section 3.6. For this timed automaton, the range of the clocks in C are discretised as follows. For each clock $c_i \in C$, let $\hat{c}_i$ be the maximum value in the range of clock c_i. This value can be obtained from the sum over all nested upper bounds of the temporal operators in the specification. Moreover, let $\Delta\tau_i \in \mathbb{R}$ be a constant such that $\hat{c}_i \equiv 0 (\text{mod } \Delta\tau_i)$. Then, the range of each clock c_i is divided into intervals of the form $[(k_i - 1)\Delta\tau_i, k_i\Delta\tau_i]$, where $k \in \{1, 2, \ldots, \frac{\hat{c}_i}{\Delta\tau_i}\}$. Similar to the clock valuation Ω a range valuation Ω_T is introduced. This element has entries equal to the interval containing the value of each clock, i.e. $\Omega_T(i) = [\tau_i^a, \tau_i^b]$ such that $\tau_i^a \leq \Omega(i) \leq \tau_i^b$ for $i \in \{1, \ldots, c_n\}$. The set of all possible values in

Ω_T is denoted by $\boldsymbol{\Omega_T}$. A clock constraint of the form $c_i \leq k$ is satisfied by $\Omega_T(i)$ if $\tau_i^b \leq k$. On the other hand, a clock constraint $c_i \geq k$ is satisfied by $\Omega_T(i)$ if $\tau_i^a \geq k$.

Given the partition of the clocks range into intervals, a configuration of the discretised timed automaton $\mathcal{T}$ is defined by pairs (q, Ω_T). Similar to the timed automaton T, a run on $\mathcal{T}$ induced by a timed word (w, η) is an infinite sequence $\boldsymbol{q_{\Omega_T}} = (q_0, \Omega_{T,0})(q_1, \Omega_{T,1})\ldots$, where $\Omega_{T,0}(i) = [0, \Delta\tau_i]$ for $i \in \{1, \ldots, c_n\}$ and for all $j \geq 1$, $\Omega_j(i) = \Omega_{j-1}(i) + \Delta\eta_j$ if $c_i \notin \varsigma$, $\Omega_j(i) = 0$ if $c_i \in \varsigma$ and $\Omega_j(i) \in \Omega_{T,j}(i)$.

Recall from Chapter 3 that a run (w, η) is accepted by a timed automaton if the set of accepting states in T is reached. In order to find a policy to drive the system in such a manner that the produced timed word is accepted by $\mathcal{T}$, the Cartesian product of the BMDP and $\mathcal{T}$ is computed as follows. Given the timed automaton $\mathcal{T}_\varphi$ representing the formula φ and the BMDP $\mathcal{B}$, the product BMDP $\mathcal{P} = \mathcal{B} \times \mathcal{T}_\varphi$ is defined by the tuple $\mathcal{P} = (S_\mathcal{P}, S_0, A_\mathcal{P}, \hat{P}_\mathcal{P}, \check{P}_\mathcal{P}, \Pi, L)$, where:

- $S_\mathcal{P} = S \times Q \times \boldsymbol{\Omega_T}$ is a finite set of states,
- $S_0 = S \times q_0 \times \boldsymbol{\Omega_T}$ is a set of initial states,
- $A_\mathcal{P} = A$,
- $\hat{P}_\mathcal{P}((s', q', \Omega_T')|(s, q, \Omega_T), a_s^l)) = \hat{P}(s'|s, \mu_s^l) T_\Delta^{\mu_s^l}$ iff $(q, \Omega_T) \xrightarrow{L(s'), \lambda'} (q', \Omega_T')$ and 0 otherwise,
- $\check{P}_\mathcal{P}((s', q', \Omega_T')|(s, q, \Omega_T), a_s^l)) = \check{P}(s'|s, \mu_s^l) T_\Delta^{\mu_s^l}$ iff $(q, \Omega_T) \xrightarrow{L(s'), \lambda'} (q', \Omega_T')$ and 0 otherwise,
- Π is a set of atomic propositions,
- $L : S_\mathcal{P} \to 2^\Pi$ is a labelling function,

$(q, \Omega_T) \xrightarrow{L(s'), \lambda'} (q', \Omega_T')$ is a transition in $\mathcal{T}$ such that the clock constraints λ' are met, $T_\Delta^{\mu_s^l}$ denotes the probability of the system leaving region s in Δ units of time under the local policy μ_s^l and $\Delta = \Omega_T' - \Omega_T$. The probability $T_\Delta^{\mu_s^l}$ can be computed similar to the probabilities in Eqs. 5.8 and 5.9. Once the product BMDP $\mathcal{P}$ is created, a global policy that maps states of the product BMDP to an action in $A_\mathcal{P}$ can be computed as shown in the next subsection. The acceptance component is defined by the set $F_\mathcal{P} = S \times F \times \boldsymbol{\Omega_T}$.

5.2.5 Optimal global policy computation

In this subsection, the computation of the policy $\mu_\mathcal{P} : S_\mathcal{P} \to A_\mathcal{P}$ is presented. This policy maximises the probability of satisfying the specification φ. This probability is equal to

the probability of the controlled Markov chain $\{\chi_i, i \in \mathbb{Z}_+\}$ on $\mathcal{P}$, induced by a policy $\mu_{\mathcal{P}}$, reaching the set of final states $F_{\mathcal{P}}$ [49].

To find the policy that maximises the probability of reaching $F_{\mathcal{P}}$ in the product BMDP, the interval value iteration algorithm (IVI) [55] is utilised. This algorithm can optimise a value function using the lower bound $\check{P}_{\mathcal{P}}$ or the upper bound $\hat{P}_{\mathcal{P}}$. In [55], these are referred to as pessimistic and optimistic value functions, respectively. In this chapter, the pessimistic value function is utilised. However, the optimistic value function can also be used. The idea is to maximise the value function:

$$V(s_{\mathcal{P}}) = \max_{\mu^l_{s_{\mathcal{P}}} \in A_{\mathcal{P}}(s_{\mathcal{P}})} \min_{\bar{P} \in [\hat{P}_{\mathcal{P}}, \check{P}_{\mathcal{P}}]} \sum_{s'_{\mathcal{P}} \in S_{\mathcal{P}}} \bar{P}(s'_{\mathcal{P}} | s_{\mathcal{P}}, \mu^l_{s_{\mathcal{P}}}) V(s'_{\mathcal{P}}), \tag{5.10}$$

for all $s_{\mathcal{P}} \in \{S_{\mathcal{P}} \setminus F_{\mathcal{P}}\}$ and $V(s_{\mathcal{P}}) = 1$ for all $s_{\mathcal{P}} \in F_{\mathcal{P}}$. Intuitively, the value $V(s_{\mathcal{P}})$ is the probability of reaching the set of final states $F_{\mathcal{P}}$ starting from $s_{\mathcal{P}} \in S_{\mathcal{P}}$. Projected to the discrete approximation of the system, $V(s_{\mathcal{P}})$ represents the worst-case probability of satisfying the specification φ from states x within the region s given that $s_{\mathcal{P}} = (s, \cdot, \cdot)$. Hence, the policy that maximises $V(s_{\mathcal{P}})$ for each state, denoted by $\mu^*_{\mathcal{P}}$, is selected as an optimal global policy for the product BMDP $\mathcal{P}$.

5.2.6 Policy implementation

The computed optimal global and local policies are implemented in the following manner. Given the initial system state $x(0)$ and the valuation of the clocks Ω with all entries equal to zero, the product BMDP state $s_{\mathcal{P}} = (s, q_0, \Omega_T) \in S_{\mathcal{P}}$ that satisfies: (i) $\mathcal{W}(x(t)) \in s$, i.e. $x(t)$ is in the interior of the region s; and (ii) $\Omega_T(j) = [0, \Delta\tau]$, for all $j \in \{1, \ldots, c_n\}$, is identified. The local policy μ^l_s that corresponds to the optimal action $\mu^*_{\mathcal{P}}(s_{\mathcal{P}})$ is selected to control the system. To apply a local policy, the nearest sampled state $x_{nearest}$, in the interior of region s, to the current system state is sought. Then, the control $\mu^l_s(x_{nearest}) \in Z_n$ is applied for $\Delta t_n(x_{nearest})$ units of time. At the next state $x(t')$, where $t' = t + \Delta t_n(x_{nearest})$, the set of clocks $C \setminus \varsigma$ are incremented by $\Delta t_n(x_{nearest})$ units of time and clocks in ς are set to zero. The new product BMDP state $(s', q', \Omega'_T) \in S_{\mathcal{P}}$ satisfying: (i) $\mathcal{W}(x(t')) \in s'$; (ii) $q \xrightarrow{L(s'), \lambda'} q'$; and (iii) $\Omega'_T(i) = [\tau^a_i, \tau^b_i]$ such that $\tau^a_i \leq \Omega(i) \leq \tau^b_i$ for all $i \in \{1, \ldots, c_n\}$ is selected and the process is repeated.

5.3 Analysis

In this section, the complexity of the proposed method is presented. The complexity of the proposed algorithm can be divided into two parts: the computation of local policies in each region (phase 1) and the computation of the global policy in the product BMDP (phase 2). The iMDP algorithm used to find local policies has a time complexity $\mathcal{O}(|Z_n|^{\theta}(\log|Z_n|)^2)$, where $|Z_n|$ is the number of sampled states within a region and $\theta \in (0,1]$ is a constant [67]. On the other hand, the number of iterations of the IVI algorithm, required to converge to an optimal interval value, is polynomial in the number of states in the product BMDP $\mathcal{P}$ [55], which has at most $|S| \times |Q| \times |\boldsymbol{\Omega_T}|$ states, where S, Q and $\boldsymbol{\Omega_T}$ are the set of regions, the number of states in the timed automaton $\mathcal{T}$ and the set of all possible valuations Ω_T, respectively. Note that by partitioning the environment instead of considering all the states of a tree like in the previous chapter, the number of states in $\mathcal{P}$ is significantly reduced. Moreover, in contrast to other methods such as [49], the dimension of the system does not affect the computation of the product BMDP.

5.4 Example

In this section, the proposed approach is illustrated with a numerical example and compared to the most related work in the literature. A Dubin's car [44] is considered. This system can be represented by the function:

$$f(x(t),u(t)) = \begin{pmatrix} u_1(t)\cos\theta(t) \\ u_1(t)\sin\theta(t) \\ u_2(t) \end{pmatrix}, \quad D(x(t)) = 0.05I_3, \tag{5.11}$$

where $x = (x_1, x_2, \theta)$ is the position and heading angle, $u_1 \in [-1,1]$ is the linear velocity input, $u_2 \in [-\pi,\pi]$ is the angular velocity input and I_3 is the identity matrix of size 3×3. In this example, the workspace is constrained by $0 \le x_1 \le 3$ and $0 \le x_2 \le 3$ and has two areas of interest marked by the atomic propositions π_1 and π_2, Figure 5.5. The objective is to maximise the probability of satisfying the MITL specification $\varphi = \Diamond_{[0,20]}(\pi_1 \wedge \Diamond_{[10,20]}(\pi_2))$, which indicates that the system has to reach areas π_1 and π_2 within 20 units of time with the restriction of visiting π_2 after the tenth unit of time.

For this example, the workspace is partitioned into 132 discrete regions by using the library Triangle [156] and the size of the intervals, i.e. $\Delta\tau$, is 1. This discretisation produces a product BMDP $\mathcal{P}$ with 10560 states. To compute local policies, 300 discrete states are randomly sampled in each region. This process requires 3361.72 seconds. On the other

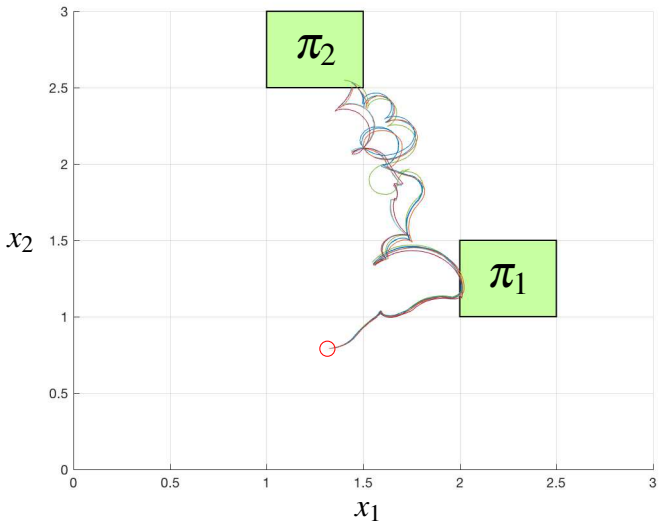


Fig. 5.5 Illustration of 10 sample paths of the system in Eq. 5.11. The system has to visit region π_1 and π_2 within 20 units of time, nevertheless, region π_2 has to be visited after the tenth unit of time. Formally, the specification can be written as $\varphi = \Diamond_{[0,20]}(\pi_1 \wedge \Diamond_{[10,20]}(\pi_2))$. The initial position is marked by a red disk.

hand, the construction of $\mathcal{P}$ and the computation of the optimal global policy requires, on average, 1871.36 seconds with a standard deviation of 104.76. On average, the system reaches π_2 in 19.44 seconds, Figure 5.6. The results presented above represent the average over 10 runs. The example is implemented in MATLAB on a computer with a 3.1 GHz i7 processor and 8 GB of RAM.

5.4.1 Discussion

In this section, the proposed method is compared with the most related work [49]. In [49], the system is modelled as a MDP, which is computed by discretising the state space. Then, a product MDP is constructed with a timed automaton. Because of the discretisation, the solution in [49] is limited by its scalability. Moreover, it requires more time to find a solution than the proposed solution in this chapter. The example presented in this section requires, on average, 5233.08 seconds to be solved. In contrast, the method in [49] requires 19080 seconds for the same system model. Although the time of the method proposed in this chapter would increase if more samples are used to compute the local policies, see Section 5.3, for a reasonable number of samples, the proposed method is faster than [49] as demonstrated in the example above. Moreover, for new MITL formulae, the method proposed in this chapter would be always faster than [49]. This is achieved because only the second phase has to

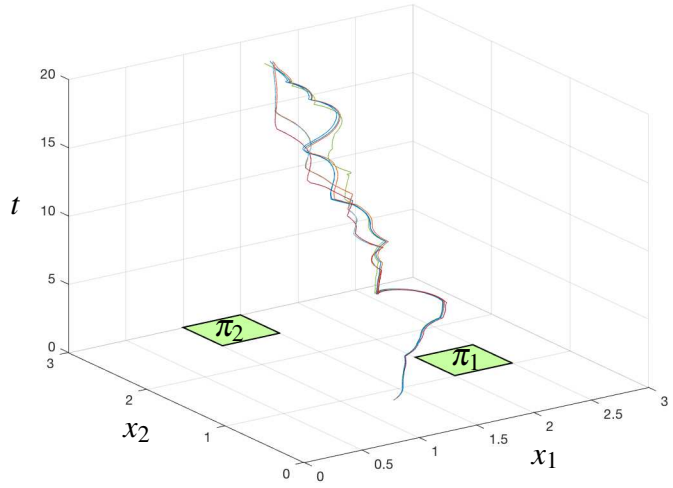


Fig. 5.6 3D view of 10 trajectories of the system in Eq. 5.11 following the MITL specification $\varphi = \Diamond_{[0,20]}(\pi_1 \wedge \Diamond_{[10,20]}(\pi_2))$. The x, y and t axis show the position of the system and the time, respectively. The average time required to reach π_2 is 19.44 seconds.

be solved. Formally, an optimal policy is obtained using a value iteration algorithm in the Cartesian product in both methods. Recall that the number of iterations of the algorithm, required to converge to an optimal value, is polynomial in the number of states. Since, in the proposed method, the dynamics of the system are reasoned in the first phase, the number of states in the Cartesian product depends only on the number of discrete regions of the coarse segmentation. In contrast, in [49], the number of states depends on a finer discretisation of the state space.

The improvement presented above is obtained at the cost of the smoothness of the trajectory. The 'zigzag' pattern shown in Fig. 5.5 is caused by the local policies computed in each region. Since the local optimal policies are obtained by solving an optimisation problem, all the sampled states have assigned the control that produces the shortest internal path to the adjacent regions. Therefore, a quick change in the direction can be observed when the system reaches a new region. A possible solution is to reduce the size of the regions to obtain a finer segmentation. Nevertheless, this would have an impact on the time required to solve the problem. In other words, the method offers a trade-off between the smoothness of the trajectory and the time needed to find a solution.

5.5 Concluding remarks

This chapter has presented a novel sampling-based method to compute policies for stochastic systems such that the probability of satisfying a MITL specification is maximised. Similar to the previous chapter, there is no closed-form or exact algorithm solutions for general continuous-time continuous-space stochastic optimal control problems [67]. Hence, the method is this chapter finds approximate solutions to this problem by using a Markov chain approximation. Another difficulty of the problem in this chapter is the consideration of time constraints and a continuous state space. Available methods that consider these elements are limited in scalability because they require a fine discretisation of the state space. The solution in this chapter achieves computational tractability by dividing the problem into two phases, one where the system dynamics are considered and another where a solution to a BMDP is found. As a result of this division, optimal policies can be found faster compared to current methods. Nevertheless, this speed is gained at the cost of the smoothness of the trajectory.

In contrast to the previous chapter, systems with stochastic motion were considered. While this addresses certain type of uncertainties that could affect a system, other factors must be considered in order for a method to be applicable in real situations. In the next chapter, uncertainty in motion and sensing is assumed. This is a more realistic and at the same time more challenging problem due to the lack of certainty in the current state of the system. Moreover, in the next chapter, dynamic environments are also considered.

Chapter 6

Reactive Motion Planning with Temporal Logic Constraints and Imperfect State Information

In the previous chapter, the assumption of deterministic motion was removed to account for uncertainties that affect a mobile system in real applications. Nevertheless, it was still assumed that the state of the system was known all the time. In this chapter, this assumption is relaxed. Instead, systems with uncertainty in motion and sensing are considered. This is an important problem since most systems operate in uncontrolled real-world environments where the assumption of perfect motion and sensing does not hold. Another important element to consider is dynamic environments. Due to the complexity of these types of problems, most solutions are computed offline. However, if the assumed environment changes, the computed solution could be not valid anymore.

When a system has imperfect information about its state, it cannot decide the best action based on a single state. Instead, a probability distribution over all possible states must be considered. These type of problems are usually modelled as partially observable Markov decision processes (POMDPs). In general, solving problems for POMDPs is a hard task due to the number of states that depends on the horizon [132]. This complexity increases when temporal logic specifications are considered due to the possibility of having tasks with infinite horizon. Similar to the previous chapter, most of the current solutions approach these problems at the discrete level [26, 155, 196]. In other words, finite set of states, actions and observations are considered. Based on this assumption, several solutions have been proposed to find policies for POMDPs to satisfy temporal logic specifications as presented in Chapter 2 [26, 155]. Nevertheless, these approaches do not handle continuous spaces and are limited to relatively small problems in terms of the number of states.

Other solutions consider more realistic scenarios by considering continuous state, control and observation spaces [182]. However, they assume static environments. In contrast to these solutions, in this chapter, changes in the environment are considered. By using a sampling-based approach, the proposed method computes policies that maximise the probability of satisfying LTL specifications. Moreover, the approach permits systems to react to previously unknown targets and obstacles in a short period. In these situations, the method uses a policy that minimises the probability of violating the specification until the obstacles are avoided or targets are attended. As a result of these policies, the proposed method can be applied to more realistic tasks where the system is affected by uncertainties and dynamic environments. The main contribution of this chapter is a method that permits the reaction to previously unknown targets and obstacles for systems with uncertainty in motion and sensing. This reaction is achieved by computing a dynamic graph structure that considers the temporal logic specification to find online policies.

The rest of this chapter is divided as follows. First, the addressed problem is formulated in Section 6.1 by presenting background and notation for systems with uncertainty in sensing and motion. Then, in Section 6.2, the proposed solution is presented in detail. Two examples are shown in Section 6.3 to illustrate the solution. Finally, conclusions are discussed in Section 6.4.

6.1 Problem formulation

This chapter focuses on dynamic systems with motion and sensing uncertainty that evolve according to the following controllable and smoothly differentiable system:

$$x_{k+1} = f(x_k, u_k, w_k), \tag{6.1}$$

where $x_k \in X \subseteq \mathbb{R}^{d_x}$ is the system state, $u_k \in U \subseteq \mathbb{R}^{d_u}$ is the control input and w_k is the process noise at time k. The compact sets X and U define the state space of the system and control input space, respectively. Note that in this chapter the system is not modelled as a controlled diffusion processes like in the Chapter 5 because a Markov chain approximation is not used in this chapter. On the other hand, the noise w_k is considered as a zero-mean Gaussian noise with covariance Q_k. In partially observable systems, the system state is observed according to an observation model:

$$z_k = h(x_k, v_k), \tag{6.2}$$

where $z_k \in Z \subseteq \mathbb{R}^{d_z}$ denotes the observation and v_k is a zero-mean Gaussian noise with covariance R_k at time k. In contrast to the previous chapter, the method in this chapter requires the system to be controllable in order to reach a set of predefined states, see Section 6.2.2.

Since the state of the system is only partially known due to sensing uncertainty, the information available at each time k is a distribution over the set of possible states [135]:

$$b_k = Pr(x_k | z_k, u_{k-1}, z_{k-1}, \ldots, u_1, z_1, u_0, b_0). \tag{6.3}$$

This distribution, called belief, compresses the history of observations $z_{1:k}$ and control actions $u_{0:k-1}$ taken from time 0 to time k and $k-1$, respectively. The updated belief for an applied control u_k and received observation z_{k+1} is given by:

$$b_{k+1} = \frac{Pr(z_{k+1}|x_{k+1})}{Pr(z_{k+1}|b_k, u_k)} \int_X b_k Pr(x_{k+1}|x_k, u_k) dx_k. \tag{6.4}$$

In a Gaussian belief space $\mathbb{B}$, the belief is characterised by the mean $\hat{x}$ and covariance P, i.e. $b = (\hat{x}_k, P_k) \in X \times \mathbb{S}_+^{d_x \times d_x}$, where $\mathbb{S}_+^{d_x \times d_x}$ represents the set of all possible positive semi-definite matrices with $d_x \times d_x$ entries, Fig. 6.1.

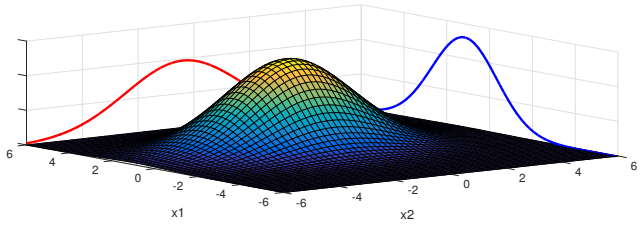


Fig. 6.1 Example of belief with mean $\hat{x} = \begin{bmatrix} -0.5 \\ 1 \end{bmatrix}$ and covariance $P = \begin{bmatrix} 0.5 & 0 \\ 0 & 1.5 \end{bmatrix}$ of a two dimensional system. The surface shows the probability density for any value of (x_1, x_2).

In this chapter, a changeable workspace $\mathcal{W}$ with static obstacles is considered. To illustrate this idea, consider as an example a robot moving objects in a dynamically changing warehouse with two areas of interested, denoted by the atomic propositions π_1 and π_2, respectively. A possible task for a robot in this scenario could be the movement of objects from π_1 to π_2 continuously. Since a changeable environment is considered, new local targets, e.g. objects in the warehouse, or obstacles can appear during the operation of the robot. Therefore, in addition to following the behaviour defined by a specification, the proposed method in this chapter allows the system to react to sensed local targets and obstacles in the environment.

The method finds optimal policies $\mu : \mathbb{B} \to U$ that map a belief state to a control input. The policies are computed in such a manner that the probability of satisfying a specification is maximised. In order to specify the desired behaviour, LTL specifications are used. These specifications are built on a set of atomic propositions Π that are associated with different regions of the workspace $\mathcal{W}$. A labelling function L is used to identify the satisfaction of atomic propositions at each time k. That is, $L(x_k) = \pi_i$ if the system is in the region defined by π_i at time k. By labelling the system state at each time k in a trajectory $\boldsymbol{x} = x_0 x_1 \ldots$, a word $w = L(x_0)L(x_1)$ expressing the behaviour of the system in terms of the atomic propositions Π is obtained. A trajectory $\boldsymbol{x}$ satisfies a LTL specification if the word w is accepted by a Rabin automaton computed from the specification. Since the state is unknown in partially observable systems, the word $w = L(x_0)L(x_1)\ldots$ cannot be used to verify the satisfaction of the specification. Instead, all the possible words generated during the transitions between beliefs are considered, as presented in the next section. The problem addressed in this chapter is now formally defined.

Problem definition 6.1.1. *Given a dynamic system with motion and sensing uncertainty of the form Eqs. 6.1 and 6.2; and a LTL formula φ, compute a policy $\mu : \mathbb{B} \to U$ such that the probability of satisfying φ is maximised.*

As in the previous chapters, the LTL specifications indicate the regions of the workspace that must be visited. Because the system is affected by uncertainty in motion and sensing, the proposed method finds the control inputs required to maximise the probability of satisfying the specifications. Moreover, the system must be able to react to previously unknown obstacles or targets. While the system is reacting to these changes, the method finds controllers that minimise the probability of violating the specification.

6.2 Solution

6.2.1 Overview

As presented in the previous section, this chapter considers two types of uncertainties, motion and sensing; and dynamic environments. An important aspect of the method proposed in this chapter is the online reaction to changes in the environment. In other words, since the system will find new elements during its operation, the computation of a policy must be performed in a short time. Due to the approaches used in the previous two chapters, the consideration of dynamic environments will require some recomputation. In Chapter 4 several iteration of the algorithm are required to achieve asymptotical optimality. Since the

approach uses a RRT approach to create a transition system, any change in the environment would require the recomputation of several parts of the tree, which can be time-consuming for an online process. A similar problem is presented in Chapter 5. In that chapter, in order to reduce the size of a Cartisian product of two graphs, the environment is partitioned and policies are computed in each segment. Hence, any modification of the environment will require the computation of a new partition to maintain the proposition preserving property. In contrast, the approach presented in this chapter uses a PRM approach that allows fast computation of policies for changes in the workspace.

The main idea of the proposed solution is to create a graph that models the motion of the system. In this graph, vertices represent belief nodes and edges represent controllers that drive the system from one belief node to another, Fig. 6.2. The graph is initialised with a single vertex, the initial belief of the system. Then, the graph is incrementally expanded by adding new vertices that represent new beliefs created by randomly sampling the state space of the system. After each expansion, it is verified whether there is a path such that the LTL specification is satisfied. If such a path does not exist, a new belief is added to the graph and the process is repeated until a valid path is found. Nevertheless, because a changing environment is considered, the computed path could be invalidated by obstacles. To solve this problem, two novel algorithms that rely on a precomputed graph are presented. This graph, formed by beliefs such as the one described above, is used to guide the system to the local targets or to avoid obstacles. In the rest of this section, the computation of the graph used to satisfy the specification and the verification of it are first presented. Then, the computation of the second graph and its usage to avoid obstacles or to reach local targets is presented.

6.2.2 Feedback-based information roadmap

The main difficulties of solving POMDPs are the so-called curse of dimensionality and curse of history [135]. Roughly, the curse of dimensionality states that for a problem with n states, the belief space in which a solution is found is a $(n-1)$-dimensional space. On the other hand, the curse of history is due to the exponential growth in the number of different actions and observations with the planning horizon. As a consequence, solutions using approaches such as discretisation of belief spaces have poor scalability [23].

To alleviate this problem, in this chapter, feedback-based information roadmaps (FIRMs) [1] are used. The main idea of FIRMs is to stabilise the system around known beliefs. By driving the system to these predefined beliefs, the history of action and observations is not required. Hence, FIRMs generalise probabilistic roadmaps to account for motion and sensing uncertainty. While several sampling-based methods [68, 173] have been proposed to

reduce the complexity described above, in most of these works, each edge of the graph depends on the path travelled by the system, i.e. actions and observations taken from the initial belief. Therefore recalculation is necessary when the initial belief changes. In contrast, in a FIRM, each edge is independent of the others as a consequence of the stabilisation around the predefined belief nodes. This property is exploited to perform most of the computation offline in the proposed method.

Without loss of generality, SLQG-FIRMs, where stationary linear quadratic Gaussian (SLQG) controllers are used as belief stabilisers, are used in this chapter. Nevertheless, any other type of controller can be used provided that the reachability of a belief is guaranteed. To construct a FIRM, a PRM is first constructed by sampling the state space of the system. Let $G = (V, E)$ represent a PRM, where V is the set of vertices $v \in X$ and E is the set of edges connecting the elements of V. Each node v of the PRM is used to create a FIRM node as follows. First the system and observation models, Eqs. 6.1 and 6.2, are linearised with respect to a node v resulting in the linear models:

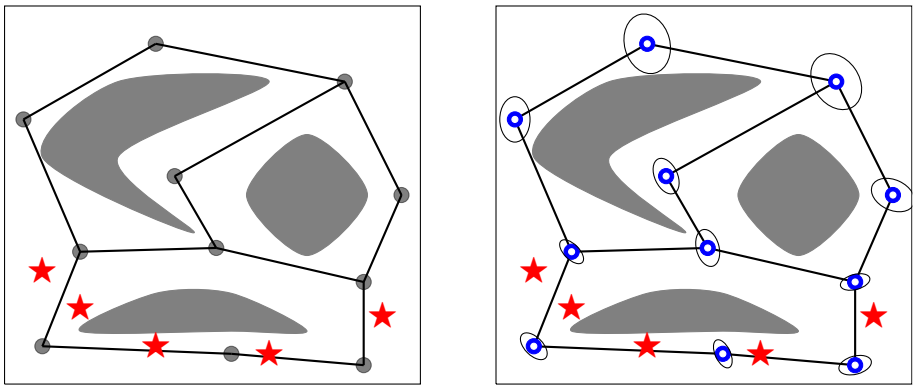
$$x_{k+1} = A_v x_k + B_v u_k + w_k, \tag{6.5}$$

$$z_{k+1} = H_v x_k + v_k, \tag{6.6}$$

where $A_v \in \mathbb{R}^{d_x \times d_x}$, $B_v \in \mathbb{R}^{d_x \times d_u}$ and $H_v \in \mathbb{R}^{d_z \times d_x}$ are obtained through Jacobian linearisation:

$$A_v = \left.\frac{\partial f}{\partial x}\right|_{\substack{x=v \\ u=0 \\ w=0}} \quad B_v = \left.\frac{\partial f}{\partial u}\right|_{\substack{x=v \\ u=0 \\ w=0}} \quad H_v = \left.\frac{\partial h}{\partial x}\right|_{\substack{x=v \\ v=0}} \tag{6.7}$$

For each vertex in V, a SLQG controller is designed to maintain the system state x as close as possible to v while a Kalman filter is used to estimate the belief. Under the assumption that the pairs (A_v, B_v) and (A_v, H_v) are controllable and observable, respectively, the SLQG controller stabilises the system to an expected belief $b_v = (v, P_v)$, where the covariance P_v can be determined offline for each node v [1]. Hence, a belief node is defined as $\mathfrak{b} = \{b : \|b - b_v\|_b < \varepsilon\}$, where $\|\cdot\|_b$ is a suitable norm in $\mathbb{B}$ and ε determines the size of the belief node. Each node $\mathfrak{b}$ is associated with its SLQG controller, denoted by $\mu_{\mathfrak{b}}$, as belief stabiliser. The edges in E of the PRM are used to design time-varying LQG controllers that drive the system to the proximity of the FIRM nodes where the stabilisers can maintain the system within the nodes. Therefore, an edge between two FIRM nodes $\mathfrak{b}$ and $\mathfrak{b}'$ is formed by the combination of the time-varying LQG and the stabiliser controller and is denoted by $\mu_{\mathfrak{b},\mathfrak{b}'}$. The set of all controllers starting from $\mathfrak{b}$ is denoted by $\mathcal{E}(\mathfrak{b})$. A FIRM can be presented as a graph $\mathcal{G} = (\mathcal{B}, \mathcal{E})$, where $\mathcal{B}$ is the set of FIRM nodes and $\mathcal{E}$ is the set of controllers used as edges, Fig. 6.2.



(a) (b)

Fig. 6.2 Illustration of a feedback-based information roadmap. (a) A PRM in an environment with three obstacles (grey regions) and five landmarks (stars). (b) The FIRM created using the PRM. The landmarks are used by the system to localise itself. Hence, the uncertainty on the system state increases with the distance to the landmarks. The centre $b_v = (v, P_v)$ of the FIRM nodes is represented with a white disk and the covariance is illustrated by an ellipse. The blue area around v denotes the neighbourhood that defines the belief node.

Recall that using feedback controllers that guarantee the convergence of the belief to predefined belief nodes, the curse of dimensionality can be broken. Hence, the idea of FIRMs is used to incrementally create a transition system where a path that satisfies the LTL specification is sought. In the next subsection, the procedure to create such a transition system is presented.

6.2.3 Incremental transition system

Although FIRMs [1] permit to find policies for systems with uncertainty in motion and sensing, they are limited to tasks such as travelling from one region to another in a workspace. In this section, an approach to create a transition system that maintains the properties of FIRMs but allows the computation of policies to satisfy LTL specifications is presented.

In order to maintain a low number of states to analyse, a transition system is incrementally created until a path satisfying the specification is created. This transition system $\mathcal{T} = (\mathcal{B}_\mathcal{T}, \mathfrak{b}_0, \delta_\mathcal{T}, \Pi, L)$ has a finite set $\mathcal{B}_\mathcal{T}$ of belief nodes, an initial belief $\mathfrak{b}_0 \in \mathcal{B}_\mathcal{T}$ and a set $\delta_\mathcal{T}$ of controllers to drive the system between belief nodes. As these controllers produce transitions, with abuse of notation, $\delta_\mathcal{T}(\mathfrak{b}, \mathfrak{b}')$ is used to represent the transition between $\mathfrak{b}$ and $\mathfrak{b}'$. This transition system is created similar to a rapidly-exploring random graph [78] to allow satisfying words of infinite length. The procedure to construct $\mathcal{T}$ is now presented, Alg. 6.1.

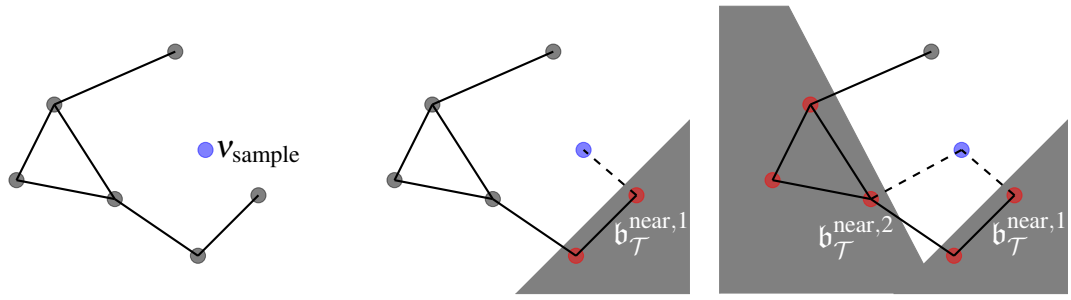
Algorithm 6.1. TRANSITIONSYSTEMEXPANSION($f, h, X, U, \mathfrak{b}_0$)

1: $\mathcal{B}_\mathcal{T} \leftarrow \mathfrak{b}_0$;
2: **while** φ not satisfied **do**
3: $\quad \mathcal{X} \leftarrow X$, $i \leftarrow 1$
4: $\quad v_{\text{sample}} \leftarrow \text{SAMPLE}(X)$;
5: $\quad \mathfrak{b}_\mathcal{T}^{\text{new}}, \mu_{\mathfrak{b}_\mathcal{T}^{\text{new}}} \leftarrow \text{CREATENODE}(f, h, v_{\text{sample}})$;
6: $\quad$ **while** $\mathcal{B}_\mathcal{T} \cap \mathcal{X} \neq \emptyset$ **do**
7: $\quad\quad \mathfrak{b}_\mathcal{T}^{\text{near},i} \leftarrow \text{NEAR}(\mathfrak{b}_\mathcal{T}^{\text{new}}, \mathcal{B}_\mathcal{T} \cap \mathcal{X})$;
8: $\quad\quad i \leftarrow i + 1$;
9: $\quad\quad \mathcal{X} \leftarrow \mathcal{X} \setminus H$;
10: $\quad \mathcal{B}_\mathcal{T} \leftarrow \mathcal{B}_\mathcal{T} \cup \mathfrak{b}_\mathcal{T}^{new}$;
11: $\quad \delta_\mathcal{T} \leftarrow \delta_\mathcal{T} \cup \mu_{\mathfrak{b}_\mathcal{T}^{\text{new}}, \mathfrak{b}_\mathcal{T}^{\text{near},j}} \cup \mu_{\mathfrak{b}_\mathcal{T}^{\text{near},j}, \mathfrak{b}_\mathcal{T}^{\text{new}}} \quad \forall j \in \{1, \ldots, i\}$;
12: **return** $\mathcal{T} = (\mathcal{B}_\mathcal{T}, \mathfrak{b}_0, \delta_\mathcal{T}, \Pi, L)$;

Initially, the transition system $\mathcal{T}$ includes only the initial node $\mathfrak{b}_0$ which contains the initial belief of the system (line 1). To add a new node, a state $v_{sample} \in X$ is sampled from the state space (line 4). This state is used to compute the FIRM node $\mathfrak{b}_\mathcal{T}^{\text{new}}$ including a belief stabiliser as presented in Section 6.2.2 (line 5). After computing the belief node, the closest node $\mathfrak{b}_\mathcal{T}^{\text{near}}$ in $\mathcal{B}_\mathcal{T}$ is found and connected to $\mathfrak{b}_\mathcal{T}^{\text{new}}$ (line 7). Then, all the nodes in $\mathcal{T}$ on the half-space H containing $\mathfrak{b}_\mathcal{T}^{\text{near},i}$ but not $\mathfrak{b}_\mathcal{T}^{new}$ are marked as unavailable for connection (line 9), Fig 6.3. The process of searching and connecting belief nodes continues until no more nodes are available for connection. Once no more nodes can be connected, the new node $\mathfrak{b}_\mathcal{T}^{\text{new}}$ is added to $\mathcal{T}$ with the transitions $(\mathfrak{b}_\mathcal{T}^{\text{new}}, \mathfrak{b}_\mathcal{T}^{\text{near},i})$ and $(\mathfrak{b}_\mathcal{T}^{\text{near},i}, \mathfrak{b}_\mathcal{T}^{\text{new}})$, where i is the index of the nearest nodes found in the process described above (lines 10-11). For each transition $(\mathfrak{b}_\mathcal{T}, \mathfrak{b}'_\mathcal{T}) \in \delta_\mathcal{T}$, the edge controller $\mu_{\mathfrak{b}_\mathcal{T}, \mathfrak{b}'_\mathcal{T}}$ is computed. This process continues until it is determined that the transition system contains a path that satisfies the LTL specification, see Section 6.2.5.

In order to reduce the number of nodes in $\mathcal{T}$ and at the same time cover most of the workspace $\mathcal{W}$, a coarse partition is computed over the workspace. To add a new belief node to $\mathcal{T}$, a segment of the partition is randomly selected based on the number of samples associated with this segment. Then, a state is sampled uniformly such that $\mathcal{W}(v_{\text{sample}})$ is in the selected segment, where $\mathcal{W}(\cdot)$ is the projection of a state onto the workspace.

Based on results from probabilistic verification [9], the product MDP of the transition system $\mathcal{T}$ and the Rabin automaton representing the LTL specification is created and used to find a path such that the LTL specification is satisfied. The computation of this product MDP is presented in the next subsection.



(a) (b) (c)

Fig. 6.3 Illustration of expansion of transition system $\mathcal{T}$ as a rapidly-exploring random graph. (a) In order to expand $\mathcal{T}$, a state v_{sample} is randomly sampled from the state space. (b) The closest node already in the $\mathcal{T}$, $\mathfrak{b}_{\mathcal{T}}^{\text{near},1}$, is found and connected to v_{sample}. Then, all the nodes in $\mathcal{T}$ on the half-space (grey region) containing $\mathfrak{b}_{\mathcal{T}}^{\text{near},1}$ (red disks) are marked as unavailable for connection. (c) A new search for the closest nodes in $\mathcal{T}$ is performed by considering all the nodes in $\mathcal{T}$ not marked as unavailable. This process continues until no more nodes are available for connection.

6.2.4 Product MDP

The product MDP $\mathcal{P} = \mathcal{T} \times \mathcal{R}$ of the transition system $\mathcal{T} = (\mathcal{B}_{\mathcal{T}}, \mathfrak{b}_0, \delta_{\mathcal{T}}, \Pi, L)$ and the deterministic Rabin automaton (DRA) $\mathcal{R} = (Q, q_0, \Sigma, \delta_{\mathcal{R}}, F)$ is a tuple $\mathcal{P} = (S, s_0, A, P, \Pi, L)$, where:

- $S = \mathcal{B}_{\mathcal{T}} \times Q$ is a finite set of states,
- $s_0 = \mathfrak{b}_0 \times q_0$ is an initial state,
- A is a finite set of actions,
- $P(\cdot|\cdot,\cdot) : S \times S \times A \to [0,1]$ is the probability of transitioning to the state s' from the state s under action $a \in A$,
- Π is a set of atomic propositions,
- $L : S \to 2^{\Pi}$ is a labelling function.

The acceptance component is defined by the set $F_{\mathcal{P}} = \{(L_1^{\mathcal{P}}, K_1^{\mathcal{P}}), \ldots, (L_r^{\mathcal{P}}, K_r^{\mathcal{P}})\}$, where $L_i^{\mathcal{P}} = \mathcal{B}_{\mathcal{T}} \times L_i$ and $K_i^{\mathcal{P}} = \mathcal{B}_{\mathcal{T}} \times K_i$ for all $i \in \{1, \ldots, r\}$. A run on $\mathcal{P}$ is defined as a sequence $\boldsymbol{s} = s_0 s_1 \ldots$, where $P(s_{i+1}|s_i, a) > 0$ for all $i \geq 0$. The set of actions A corresponds to set $\delta_{\mathcal{T}}$ of computed controllers associated with each transition in $\mathcal{T}$. Therefore, the set of actions available at state $s = (\mathfrak{b}_{\mathcal{T}}, \cdot)$, denoted as $A(s)$, are the controllers computed for the transitions

$(\mathfrak{b}_\mathcal{T}, \cdot) \in \delta_\mathcal{T}$. The probability $P(s'|s, \mu_{\mathfrak{b}_\mathcal{T}, \mathfrak{b}'_\mathcal{T}})$, where $s = (\mathfrak{b}_\mathcal{T}, q)$ and $s' = (\mathfrak{b}'_\mathcal{T}, q')$, is the probability of ending on the DRA state q' starting from q when the transition $(\mathfrak{b}_\mathcal{T}, \mathfrak{b}'_\mathcal{T}) \in \delta_\mathcal{T}$ is performed using the controller $\mu_{\mathfrak{b},\mathfrak{b}'} \in A(s)$.

Let $\boldsymbol{b} = b_0 b_1 \dots b_n$ be the sequence of beliefs followed after applying $\mu_{\mathfrak{b}_\mathcal{T}, \mathfrak{b}'_\mathcal{T}}$, such that $b_0 \in \mathfrak{b}_\mathcal{T}$ and $b_n \in \mathfrak{b}'_\mathcal{T}$. Due to the uncertainties affecting the system, more that one sequence $\boldsymbol{b}$ can be obtained after applying $\mu_{\mathfrak{b}_\mathcal{T}, \mathfrak{b}'_\mathcal{T}}$. Now, recall that a word $w = L(b_0)L(b_1)$, where $L(\mathfrak{b})$ is the set of atomic propositions satisfied by the mean $\hat{x}$ of the belief b, expresses the behaviour of the system in terms of the atomic propositions in the set Π. To find the DRA state q' reached in $\mathcal{R}$ after the transition $(\mathfrak{b}_\mathcal{T}, \mathfrak{b}'_\mathcal{T}) \in \delta_\mathcal{T}$, the word w produced by $\boldsymbol{b}$ is used as an input into the DRA $\mathcal{R}$, starting from the state $q \in Q$. The last state of the run $\boldsymbol{q}$ on $\mathcal{R}$, produced by w, is used as a state q' for the transition $s = (\mathfrak{b}_\mathcal{T}, q)$ to $s' = (\mathfrak{b}'_\mathcal{T}, q')$.

To illustrate this, consider the initial state q_0 of the Rabin automaton in Fig. 6.4 and assume that during the transition $(\mathfrak{b}_\mathcal{T}, \mathfrak{b}'_\mathcal{T})$ in $\mathcal{T}$ the words $w_1 = \{\neg\pi_1\neg\pi_2\}\{\neg\pi_1\neg\pi_2\}\{\pi_1\neg\pi_2\}$ and $w_2 = \{\neg\pi_1\neg\pi_2\}\{\neg\pi_1\neg\pi_2\}\{\neg\pi_1\pi_2\}$ are generated with probability 0.90 and 0.10, respectively. It can be seen that starting from state q_0 in the automaton $\mathcal{R}$ and using the word w_1 (w_2) as an input, the run $\boldsymbol{q} = q_0q_0q_0q_2$ ($\boldsymbol{q} = q_0q_0q_0q_1$) is produced. Therefore, the probability of transitioning from state $(\mathfrak{b}_\mathcal{T}, q_0)$ to $(\mathfrak{b}'_\mathcal{T}, q_2)$ is 0.90 and to $(\mathfrak{b}'_\mathcal{T}, q_1)$ is 0.10.

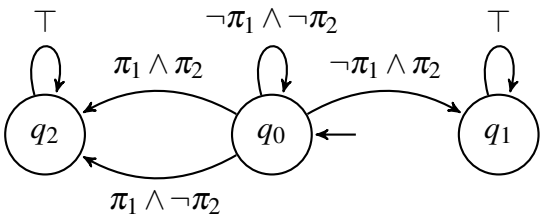


Fig. 6.4 Rabin automaton of LTL formula $\varphi = \neg\pi_2 \mathcal{U} \pi_1$, where the atomic propositions π_1 and π_2 represent two regions in a workspace, respectively. The formula indicates that the region marked as π_2 must be avoided until the region π_1 is reached.

Recall that a specification is satisfied by the system if the word w produces a run on $\mathcal{R}$ such that it visits finitely often times the set L_i and infinitely many times the set K_i, for $i \in \{1, \dots, r\}$. Because during the transition s to s' in $\mathcal{P}$, more than one DRA state can be reached, in order to find a run on $\mathcal{P}$ satisfying a specification, each transition in $\mathcal{P}$ is associated with a probability of visiting a state in a pair $(L_i, K_i) \in F$. These probabilities are denoted as $P^{L_i}_{s,s'}$ and $P^{K_i}_{s,s'}$, respectively.

Computing probabilities of transitioning from s to $s' \in S$ is computationally expensive [1]. In this chapter, these probabilities are approximated using particle-based methods. The probability $P((\mathfrak{b}'_\mathcal{T}, q')|(\mathfrak{b}_\mathcal{T}, q), \mu_{\mathfrak{b}_\mathcal{T}, \mathfrak{b}'_\mathcal{T}})$ is computed based on the number of particles that produced a word w, during the transition $\mathfrak{b}_\mathcal{T}$ to $\mathfrak{b}'_\mathcal{T}$ under $\mu_{\mathfrak{b}_\mathcal{T}, \mathfrak{b}'_\mathcal{T}}$, and generated a run $\boldsymbol{q}$

starting from q and finishing in q'. A similar procedure is used to calculate the probability of intersecting the pairs $(L_i, K_i) \in F$ during the transition from s to s'.

The product MDP $\mathcal{P}$ is updated with each new node $\mathfrak{b}_{\mathcal{T}}^{new}$ added to $\mathcal{T}$. After each update, it is checked whether the LTL specification can be satisfied. In the next subsection, such a verification is explained together with the computation of a policy $\mu_{\mathcal{P}} : S \to A$ in $\mathcal{P}$ that satisfies the LTL specification. Using $\mu_{\mathcal{P}}$, a policy $\mu : \mathbb{B} \to U$ that solves the formulated problem is finally obtained.

6.2.5 Optimal policy computation

This subsection presents the calculation of the policy that maximises the probability of satisfying a LTL specification φ. A run $\boldsymbol{s} = s_0 s_1 \dots$ on $\mathcal{P}$ is accepted if there exists a pair $(L_i^{\mathcal{P}}, K_i^{\mathcal{P}}) \in F_{\mathcal{P}}$ such that $L_i^{\mathcal{P}}$ and $K_i^{\mathcal{P}}$ are visited finitely and infinitely many times, respectively. To find if such a run exists, the method uses accepting end components (AECs), which are defined as follows. An AEC of $\mathcal{P}$ for a pair $(L_i^{\mathcal{P}}, K_i^{\mathcal{P}}) \in F_{\mathcal{P}}$ is a subgraph of $\mathcal{P}$ where each state is reachable from every other state, $P_{s,s'}^{L_i} = 0$ for all transitions and there exists a transition with $P_{s,s'}^{K_i} > 0$. After each increment of the transition system $\mathcal{T}$, the existence of an AEC is checked. Once an AEC is found, an optimal policy is computed.

It has been shown in probabilistic model checking that maximising the probability of reaching an AEC is equivalent to maximising the probability of satisfying φ [9]. On the other hand, a policy $\mu_{\mathcal{P}}(s)$ on $\mathcal{P}$, where $s = (\mathfrak{b}_{\mathcal{T}}, q)$, induces a policy $\mu(\mathfrak{b}_{\mathcal{T}})$ on $\mathcal{T}$ by defining $\mu(\mathfrak{b}_{\mathcal{T}}) = \mu_{\mathcal{P}}(s)$. Hence, computing a policy on $\mathcal{P}$ that maximises the probability of reaching an AEC is equivalent to finding a policy on $\mathcal{T}$ that maximises the probability of satisfying the LTL specification. The value iteration method is used to compute the optimal policy by maximising the value function:

$$V(s) = \max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a) V(s'), \tag{6.8}$$

$$\mu_{\mathcal{P}}(s) = \arg\max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a) V(s'), \tag{6.9}$$

for all $s \notin$ AEC and $V(s) = 1$ for all $s \in$ AEC.

Since the product MDP is updated with each addition of nodes to $\mathcal{T}$, the end components of $\mathcal{P}$ must be maintained after each update. The complexity of maintaining the end components on $\mathcal{P}$ is $O(|F||S|^{\frac{3}{2}})$ [182], where the number of states in S is $|\mathcal{B}_{\mathcal{T}}| \times |Q|$. On the other hand, the running time of each iteration to find the optimal policy is $O(|S||A|^2)$ [110].

At this point, it has been shown how to compute a policy to maximise the probability of satisfying a LTL specification. Nevertheless, since a dynamic environment is considered in this chapter, the computed policy becomes invalid if a new obstacle or local target appears in the workspace. In the following two subsections, the algorithms used to reach detected local targets and to avoid previously unknown obstacles are presented.

6.2.6 Local targets

Approximating the probability of each transition on $\mathcal{P}$ using particle-based methods is, in general, a slow process [1, 182]. The construction and computation of a policy for $\mathcal{T}$ is computed offline and hence this slow task can be tolerated. Nevertheless, for fast reactions to targets or obstacles sensed in real-time, this long time is restrictive. In this and the following section, two novel algorithms that permit the online reaction to previously unknown elements are presented. These algorithms are based on the offline computation of a FIRM $\mathcal{G}$, which is used to drive the system when obstacles or targets are detected. In addition to permitting reactions in a short period of time, PRM-like structures such as FIRM can present better performance than methods using RRG techniques on difficult scenarios [73].

To maximise the coverage of the workspace and to obtain a dynamic FIRM (see Section 6.2.7), an offline partition of the environment is first created. In this chapter, a grid-based partition is used. However, any other type of partition can be utilised. Then, the process of selecting and sampling in cells is performed similarly to the process presented in Section 6.2.3. After a minimum number of samples on each cell are obtained, the FIRM $\mathcal{G} = (\mathcal{B}, \mathcal{E})$ is created as presented in Section 6.2.2.

When a local target is sensed by the system at time k, the FIRM is used to drive the system from its current belief b_k to a predefined service region of the local target while the specification is satisfied. To use the transition system and the FIRM, three aspects have to be considered: (i) the connection of the current belief to a node in $\mathcal{G}$; (ii) the optimal path in $\mathcal{G}$; and (iii) the reconnection to $\mathcal{T}$ after the local target has been attended. This procedure is presented in Alg. 6.2.

In the first step, when a local target is sensed by the system, a subgraph of $\mathcal{G}$ is created within the sensing area with radius r (line 1, Alg. 6.2), Fig. 6.5. In this subgraph, the nearest FIRM node $\mathfrak{b}_{\text{near}}$ to the current belief b_k is sought (line 2, Alg. 6.2). Then, the local stabiliser of $\mathfrak{b}_{\text{near}}$ is applied to drive the system to the FIRM node (lines 3-4). Once the system is in the subgraph of $\mathcal{G}$, an optimal policy is computed to drive the system to the local target (line 5, Alg. 6.2). This path is optimal in terms of minimising the probability of violating the specification. To achieve this, it is necessary to verify which transitions of the FIRM do not violate the LTL specification. A similar problem has been solved in the

Algorithm 6.2. PATHLOCALTARGET($f, h, \mathcal{G}, \mathcal{T}, target$)

1: $\mathcal{G}' = (\mathcal{B}', \mathcal{E}') \leftarrow$ COMPUTESUBGRAPH$(\mathcal{G}, r)$;
2: $\mathfrak{b}_{\text{near}} \leftarrow$ NEAREST$(b_k, \mathcal{B}')$, $\mathfrak{b}_{\text{target}} \leftarrow$ NEAREST$(target, \mathcal{B}')$;
3: **while** $b_k \notin \mathfrak{b}_{\text{near}}$ **do**
4: $\quad x_{k+1} = f(x_k, \mu_{\mathfrak{b}_{\text{near}}}(x_k), w_k)$;
5: $\mu_{\mathcal{G}} \leftarrow$ OPTIMALPOLICY$(\mathfrak{b}_{\text{target}})$;
6: Apply policy $\mu_{\mathcal{G}}$;
7: $\mathfrak{b}_{\text{close}} \leftarrow$ NEAREST$(\mathfrak{b}_{\mathcal{T}}, \mathcal{B}')$;
8: $\mu_{\mathcal{G}} \leftarrow$ OPTIMALPOLICY$(\mathfrak{b}_{\text{close}})$;
9: Apply policy $\mu_{\mathcal{G}}$;
10: **while** $b_k \notin \mathfrak{b}_{\mathcal{T}}$ **do**
11: $\quad x_{k+1} = f(x_k, \mu_{\mathfrak{b}_{\mathcal{T}}}(x_k), w_k)$;

literature for deterministic systems with perfect state information [7, 181] using a monitor [11] which identifies if a specification has been satisfied or falsified as early as possible. In the current work, since the state of the system is uncertain, using a monitor is not an option. Instead, the following procedure is used. Recall that in order to satisfy a specification, for a pair $(L_i, K_i) \in F$, the set L_i must be visited only finitely many times. Therefore, the method calculates the probability of reaching states in L_i with a self transition, Fig. 6.4. Similar to the computation of $P^{L_i}_{s,s'}$ and $P^{K_i}_{s,s'}$ in $\mathcal{T}$, the probability of reaching such states starting in every Rabin state q during the transition from one node to another in $\mathcal{G}$ is computed during the FIRM construction. These probabilities are used to compute a policy on $\mathcal{G}$. Since the probability of reaching a state L_i on a transition $(\mathfrak{b}, \mathfrak{b}')$ depends on the DRA state q, the current DRA state is tracked all the time during the online operation. Because all the transitions are precomputed offline, only the computation of the optimal policy, using the probabilities according to the current DRA state, is performed online. This policy can be computed by finding the controller that minimises the cost-to-go for each belief node. The cost-to-go for a belief node $\mathfrak{b}$ is given by:

$$V(\mathfrak{b}) = \min_{\mu \in \mathcal{E}(\mathfrak{b})} P(L|\mathfrak{b}, \mu) V(L) + \sum_{\mathfrak{b}' \in \mathcal{B}} P(\mathfrak{b}'|\mathfrak{b}, \mu) V(\mathfrak{b}'), \tag{6.10}$$

where $V(L)$ is a user-defined cost assigned for violating the specification, $P(L|\mathfrak{b}, \mu)$ is the probability of intersecting the set L when travelling from $\mathfrak{b}$ using $\mu \in \mathcal{E}(\mathfrak{b})$ and $P(\mathfrak{b}'|\mathfrak{b}, \mu)$ is the probability of reaching $\mathfrak{b}'$ from $\mathfrak{b}$. The cost-to-go of the belief node in the local target is defined by $V(\mathfrak{b}_{\text{target}}) = -1$.

After applying the policy (line 6, Alg. 6.2), the last FIRM node in the path has to be connected to the transition system $\mathcal{T}$ in order to continue with the specification. This is achieved by searching the closest node $\mathfrak{b}_{\mathcal{T}}$ in $\mathcal{T}$ such that $V(s) > 0$ and $s = (\mathfrak{b}_{\mathcal{T}}, q)$, where

q is the current $\mathcal{R}$ state after following the path in $\mathcal{G}$. Once this state has been found, the closest node $\mathfrak{b}_{\text{close}}$ of $\mathcal{G}$ to $\mathfrak{b}_{\mathcal{T}}$ is sought (line 7, Alg. 6.2). Then, a policy is computed in $\mathcal{G}$ to drive the system to $\mathfrak{b}_{\text{close}}$ (line 8, Alg. 6.2). After applying the policy (line 9, Alg. 6.2), the stabiliser of the node $\mathfrak{b}_{\mathcal{T}}$ is applied to make the connection (lines 10-11, Alg. 6.2).

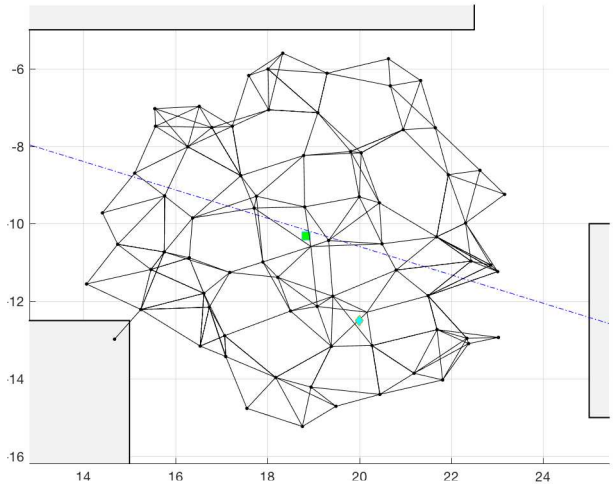


Fig. 6.5 Subgraph of the FIRM within the sensing area of the system. The offline path obtained by solving the product MDP $\mathcal{P}$ is shown as a blue dotted line. The current belief and local target are represented by a green rectangle and blue diamond, respectively.

6.2.7 Obstacle avoidance

Similar to the local target case, $\mathcal{G}$ is used to avoid detected obstacles during the online operation. The main difference is that the presence of obstacles invalidates parts of $\mathcal{G}$. Because edges of the FIRM are independent of each other, ideas from dynamic roadmaps [73, 106] can be applied as follows.

Recall that the environment is partitioned into cells. Each of these cells is associated to FIRM nodes and transitions as follows. During the computation of the probabilities from node $\mathfrak{b}$ to $\mathfrak{b}'$, the probability of visiting a cell c_i during a transition can be computed as follows. Let $p^k_{0:T^k}$ be the sample path of the k-th particle p from $\mathfrak{b}$ at time zero to $\mathfrak{b}'$ at time T^k. The probability of the system reaching a state such that $\mathcal{W}(x)$ is on the cell c_i during the transition from $\mathfrak{b}$ to $\mathfrak{b}'$ is approximated by:

$$Pr_{\mathfrak{b},\mathfrak{b}'}(c_i) \approx \sum_{k=1}^{K} \gamma^k \mathbb{1}_{c_i}(p^k_{0:T^k}), \tag{6.11}$$

where γ^k is a weight assigned to the particle p^k and $\mathbb{1}_{c_i}(\cdot)$ is an indicator that returns one, if a particle enters the cell c_i, and zero otherwise. Based on these probabilities, a cell is associated with the FIRM nodes $\mathfrak{b}, \mathfrak{b}'$ and its transition if $Pr_{\mathfrak{b},\mathfrak{b}'}(c_i) > 0$, Fig. 6.6.

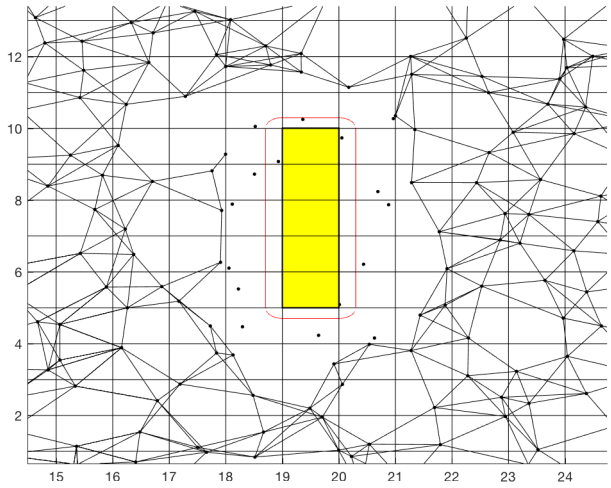


Fig. 6.6 Subgraph of the FIRM without transitions affected by the obstacle. The obstacle and estimated position of it are shown with a yellow and red rectangle, respectively. The cells (shown in blue) occupied by the obstacle determine the invalid nodes and transitions of the FIRM.

When an obstacle is detected, the cells occupied by the obstacle are computed. Then, the nodes and transitions associated with these cells are invalidated from the FIRM. Since the current state of the system is uncertain, i.e. it is given by a mean and covariance over the belief space, the exact location of the obstacle cannot be determined by the system. To include the uncertainty on the obstacle's location, the Minkowski sum of the detected obstacle and the contour of the 3σ ellipse of the current Gaussian is computed. To illustrate the obstacle avoidance procedure (Alg. 6.3), assume that the system is transitioning between the nodes $\mathfrak{b}_{\mathcal{T}}$ and $\mathfrak{b}'_{\mathcal{T}}$ in $\mathcal{T}$ when an obstacle is detected. A subgraph of $\mathcal{G}$ is created within the sensing area (lines 2-4, Alg. 6.3) as presented in Section 6.2.6. Note that this subgraph does not include any of the nodes affected by the estimation of the obstacle's location. In this subgraph, the closest node $\mathfrak{b}_{\text{near}}$ to the current belief b_k is sought. The stabiliser of $\mathfrak{b}_{\text{near}}$ is applied to drive the system to this node. Then, a policy to drive the system to $\mathfrak{b}_{\text{target}}$, the closest node to $\mathfrak{b}'_{\mathcal{T}}$, is computed as presented in Section 6.2.6 (lines 5-6, Alg. 6.3). If, after applying the policy, the obstacle is still detected, a new subgraph is computed by removing

the invalid nodes. This process is repeated until the obstacle is not sensed. Then, $\mathcal{G}$ is connected to $\mathcal{T}$ (lines 8-12, Alg. 6.3) as presented in Section 6.2.6.

Algorithm 6.3. OBSTACLEAVOIDANCE($f, h, \mathcal{G}, \mathcal{T}, obstacle$)

1: **while** obstacle detected **do**
2: $obstacleposition \leftarrow$ EstimatedPosition$(b_k, P_k, obstacle)$;
3: $C \leftarrow$ AffectedCells$(obstacleposition)$;
4: $\mathcal{G}' = (\mathcal{B}', \mathcal{E}') \leftarrow$ COMPUTESUBGRAPH$(\mathcal{G}, r, obstacleposition, C)$;
5: $\mathfrak{b}_{\text{target}} \leftarrow$ Neareast$(\mathfrak{b}'_{\mathcal{T}}, \mathcal{B}')$;
6: $\mu_{\mathcal{G}} \leftarrow$ OPTIMALPOLICY$(\mathfrak{b}_{\text{target}})$;
7: Apply policy $\mu_{\mathcal{G}}$;
8: $\mathfrak{b}_{\text{close}} \leftarrow$ NEAREAST$(\mathfrak{b}'_{\mathcal{T}}, \mathcal{B}')$;
9: $\mu_{\mathcal{G}} \leftarrow$ OPTIMALPOLICY$(\mathfrak{b}_{\text{close}})$;
10: Apply policy $\mu_{\mathcal{G}}$;
11: **while** $b_k \notin \mathfrak{b}'_{\mathcal{T}}$ **do**
12: $x_{k+1} = f(x_k, \mu_{\mathfrak{b}_{\mathcal{T}}}(x_k), w_k)$;

6.3 Examples

In this section, the proposed method is illustrated with the three-wheel omnidirectional mobile robot presented in [1]. For this robot, Eq. 6.1 becomes:

$$f(x_k, u_k, w_k) = \begin{pmatrix} -\frac{2}{3}\sin(\theta) & -\frac{2}{3}\sin(\frac{\pi}{3}-\theta) & \frac{2}{3}\sin(\frac{\pi}{3}+\theta) \\ \frac{2}{3}\cos(\theta) & -\frac{2}{3}\cos(\frac{\pi}{3}-\theta) & -\frac{2}{3}\cos(\frac{\pi}{3}+\theta) \\ \frac{1}{3l} & \frac{1}{3l} & \frac{1}{3l} \end{pmatrix} u + w. \tag{6.12}$$

The state $x = [x_1, x_2, \theta]^T$ is composed of the robot position (x_1, x_2) and the orientation θ. The control input $u = [u_1, u_2, u_3]^T$ is formed of the linear velocities of each wheel. The distance of the wheels from the centre of the robot are equidistant and denoted by l. The process noise w is a zero-mean Gaussian with covariance Q.

The robot uses landmarks, with known location on the workspace, to localise itself, Fig. 6.7. Let (LM_1^i, LM_2^i) denote the location of the i-th landmark; and η_r, σ_b^r, η_θ and σ_b^θ be constants. The observation model in Eq. 6.2 with respect to the i-th landmark is expressed as:

$$z^i = [\|d^i\|, \text{atan2}(d_2^i, d_1^i) - \theta]^T + v^i, \tag{6.13}$$

where $d = [x_1, x_2] - [LM_1^i, LM_2^i]$ and v^i is zero-mean Gaussian noise with covariance R:

$$R^i = \text{diag}((\eta_r \|d^i\| + \sigma_b^r)^2, (\eta_\theta \|d^i\| + \sigma_b^\theta)^2). \tag{6.14}$$

The system operates in a workspace with 7 areas associated with the atomic propositions π_1, π_2, π_3 and π_4. Two different LTL specifications are considered:

1. Regions π_1, π_2 and π_3 have to be visited in any order. During this process, the areas marked with π_4 must be avoided. LTL specification: $\varphi_1 = \neg\pi_4 \mathcal{U} \pi_1 \wedge \neg\pi_4 \mathcal{U} \pi_2 \wedge \neg\pi_4 \mathcal{U} \pi_3$, Fig. 6.7.

2. After reaching the region marked as π_2, the system has to go to region π_3 and then go and stay in the region π_1. The regions marked as π_4 must be avoided all the time. LTL specification: $\varphi_2 = \Diamond\Box\pi_1 \wedge \Diamond(\pi_2 \wedge \Diamond(\pi_3)) \wedge \Box\neg\pi_4$, Fig. 6.8.

In both missions, the robot encounters a local target and previously unknown obstacles. The examples are implemented in MATLAB on a computer with a 3.1 GHz i7 processor and 8 GB of RAM.

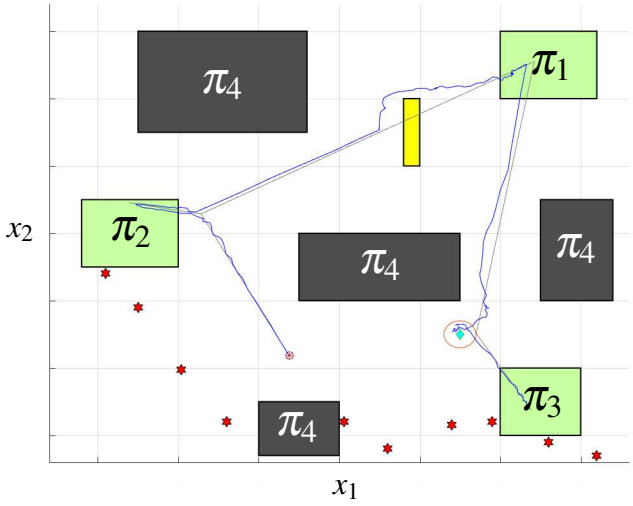


Fig. 6.7 Environment containing seven areas identified by the atomic proposition π_1, π_2, π_3 and π_4; a local target (blue diamond) with its service region (red ellipse), an unknown obstacle (yellow rectangle) and ten landmarks (red stars). The objective of the robot is to visit the areas marked as π_1, π_2 and π_3 while areas π_4 have to be avoided. The grey line shows the path computed offline. The blue line shows a sample path of the system followed after detecting the local target and previously unknown obstacle. The initial position is marked by a red disk.

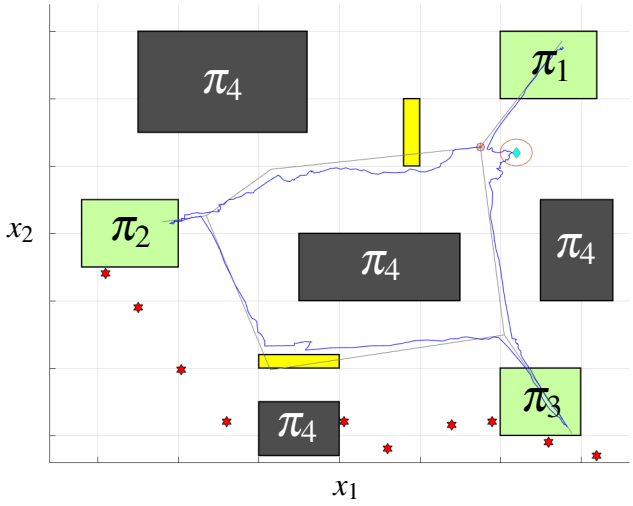


Fig. 6.8 Environment containing seven areas identified by atomic propositions; a local target (blue diamond), two unknown obstacles (yellow rectangles) and ten landmarks (red stars). The objective of the robot is to visit the areas marked as π_2 and π_3, in order, and then reach the region π_1 while areas π_4 have to be avoided. The grey line shows the path computed offline. The blue line shows a sample path of the system followed after detecting the local target and previously unknown obstacles. The initial position is marked by a red disk.

The results presented below were obtained from 20 simulations, but for the purpose of clarity, only one run is presented in Fig. 6.7 and Fig. 6.8. On average (mean), the offline path for the two missions is found in 90.18 seconds and 112.32 seconds, respectively. More details about the number of states in the transition system and product MDP are shown in table 6.1.

Table 6.1 Average required time (seconds) to solve the problems, number of states in transition system $\mathcal{T}$ and product MDP $\mathcal{P}$ over 20 different runs.

Specification	Time (mean)	Std. Dev.	States in $\mathcal{T}$	States in $\mathcal{P}$
φ_1	90.18	31.94	33.44	300.9
φ_2	112.32	39.35	41.38	248.3

Computing the probability in each transition requires 0.528 seconds. The PRM used in both mission to create $\mathcal{G}$ has 1224 vertices, each vertex is connected to its seven nearest vertices. The FIRM requires on average 5108.61 seconds to be constructed. Since computing the probabilities for each edge in $\mathcal{G}$ is the most computationally demanding operation, the

time to construct $\mathcal{G}$ could be reduced by limiting the number of edges on each vertex. Note that all the previous computations are performed offline. Finding a path in $\mathcal{G}$, online, to reach the local targets and to avoid the obstacles requires 0.083 and 0.655 seconds, respectively. Based on these results, it can be observed that computing a path in $\mathcal{G}$ to reach targets or avoid obstacles requires less time than the time that would take to expand the transition system $\mathcal{T}$ with the purpose of finding an alternative path.

6.4 Concluding remarks

In this chapter, a novel method to compute control policies for mobile robots that can react to unknown environments under uncertainty in motion and sensing has been introduced. The method computes an offline policy that maximises the probability of satisfying the specification by using an incrementally constructed transition system and a Rabin automaton. To achieve short reaction times to changes in the environment during the online operation, a feedback-based information roadmap that considers the probability of violating the specification in each transition is precomputed. Once the system finds an unknown element in the environment, the FIRM is used to reach or avoid this element. Results show that a system can react in seconds to changes in the environment while minimising the probability of violating the task. In order to compute such a graph, the ability of the system to reach states with zero velocities is assumed. Although previous works have considered the synthesis of controllers under uncertainty constraints and temporal logic specifications in continuous spaces, reaction to unknown elements of the environment had not been considered under this scenario. Hence, it is not possible to compare the proposed method to those available in the literature.

The last three chapters have presented solutions for systems under different scenarios. However, all these solutions are designed for a single system. While in many situations a single robot is enough to perform a task, other situations could require more than one robot. In the next chapter, a method that extends sampling-based methods for multi-robot systems subject to sc-LTL specifications is presented.

Chapter 7

Path Planning for Multi-robot Systems with Co-safe Linear Temporal Logic Specifications

The methods proposed in previous chapters consider a single robot. Although these methods compute trajectories that satisfy high-level specifications considering factors such as optimality and uncertainty, the applicability of the methods could be limited by the capacity of the robot. In other words, some tasks or applications require more than one robot in order to be performed. Therefore, methods to find paths for multiple robots are required.

One of the main difficulties of problems with multiple systems is the scalability of the problem. Although some of the methods developed for single robots, such as constructing a product of the transition systems of all the robots, could be adapted for multiple robots, the large number of possible states make them computationally expensive [84]. Other problems such as synchronisation and collision between robots must also be contemplated in the development of a method.

A possible solution to the multiple robot problem with a global temporal logic specification is the distribution of individual specifications in such a manner that the completion of these results in the satisfaction of the global specification [28, 103]. The individual plans are then completed using navigation functions. A drawback to these approaches is that the range of specifications is limited. Other approaches assume the motion of the robots in a discretised workspace and solve the discrete problem using Petri nets [86] or other methods. Both techniques used to drive the system, i.e. discretisation and navigation functions, suffer from scalability with the dimension of the configuration space of the system [148]. To reduce this problem, the method proposed in this chapter extends sampling-based techniques for multiple robots to satisfy sc-LTL specifications. Moreover, an algorithm that guides the

construction of the graph is presented. Results show that this algorithm reduces the amount of time to find a solution. Therefore, the contribution of this chapter is a method that guides the sampling of a configuration space to find paths for multiple robots subject to temporal logic specifications.

The rest of this chapter is divided as follows. First, the addressed problem is formally formulated in Section 7.1. Then, in Sections 7.2.3 and 7.2.4, the construction of a graph modelling the behaviour of all the robots is presented. To reduce the number of states in such a graph, an algorithm is presented in Section 7.2.5. Two examples are shown in Section 7.3 to illustrate the efficiency of the approach. Finally, conclusions are discussed in Section 7.4.

7.1 Problem formulation

This chapter considers a group of R robots operating in a static two dimensional workspace $\mathcal{W}$. In contrast to previous chapters, the method in this chapter does not focus on any particular type of dynamics. Instead, holonomic robots are considered for simplicity. The type of systems that could be considered is presented later. Let $X^i \subseteq \mathbb{R}^{d_x}$ be a compact set defining the configuration space of robot i, where i is an element of the set $\mathcal{R} = \{1,\dots,R\}$ that indexes the robots and $\mathbb{R}^{d_x}$ is the d_x-dimensional Euclidean space. Each robot has a collision-free configuration space X^i_{free}. The configuration space of the full system, i.e. all the robots, is denoted as $X = \prod_{i\in\mathcal{R}} X^i$. The collision-free configuration space $X_{\text{free}} = \prod_{i\in\mathcal{R}} X^i_{\text{free}}$ does not include configurations where collisions between robots occurs. In other words, configurations where multiple robots overlap are not considered part of X_{free}. Now, let $\boldsymbol{x} = x_0 x_1 \dots$, where $x_i = (x_i^1,\dots,x_i^R)$ for all $i \geq 0$, be a sequence of configurations describing a path followed by the full system. A path is said to be collision-free if $x_i \in X_{\text{free}}$ for all $i \geq 0$.

In order to specify the desired behaviour of the group of robots, sc-LTL specifications are used to specify a global behaviour. These specifications are built on a set of atomic propositions Π that are associated with different regions of the workspace $\mathcal{W}$. To interpret atomic propositions over the configuration space X, let $L : X \to 2^{\Pi}$ be a function that maps a configuration $x = (x^1,\dots,x^R)$ to the atomic propositions satisfied by the configurations $x^i\ \forall i \in \mathcal{R}$. Hence, a word $w = L(x_0)L(x_1)\dots$ expresses a path $\boldsymbol{x}$ in terms of the atomic propositions satisfied by $\boldsymbol{x}$. A path $\boldsymbol{x}$ satisfies a sc-LTL specification φ if the word w, produced by $\boldsymbol{x}$, is accepted by the Büchi automaton $\mathcal{B}_\varphi$ that accepts words satisfying φ. The problem addressed in this chapter is now formally defined.

Problem definition 7.1.1. *Given a group of R robots with initial configurations x_0^i for $i \in \mathcal{R}$ and a sc-LTL specification φ, find a collision-free path $\boldsymbol{x}$ such that φ is satisfied.*

Similar to the previous chapters, the sc-LTL specifications indicate the regions of the workspace that must be visited. In contrast to previous cases, where only single robots were considered, a specification could now include the task of reaching two or more regions at the same time. In this chapter, the assumption of exact transition times is ignored. This assumption has also been considered in the literature [197].

7.2 Solution

7.2.1 Overview

The main idea of the method is to create a transition system modelling the motion of all the robots as a single system. In other words, multiple robots are concatenated into a single large system which is used to find a solution. Although this concatenation implies a centralised control, this a required condition for problems with multiple robots and a global specification [28, 74]. Each state of the transition system represents a combination of single configurations of all the robots. On the other hand, transitions represent collision-free paths between these configurations. This transition system is iteratively expanded by adding new configurations until a path in the graph is able to satisfy the specification. To obtain the new configurations for the expansion, individual roadmaps that model the motion of each robot are used. During each expansion, a product automaton is updated to verify whether the transition system contains a solution. Although the approach is similar to the one used the previous chapters, this chapter requires the coordination of multiple robots. As shown in this chapter, because of this coordination, applying naive sampling-based methods would require transition systems with large number of states and as a result long computation times. To improve the required time to find a solution, a new guidance algorithm based on the temporal logic specification is proposed. In the rest of this section, the construction of the individual roadmaps is first presented. Then, the incremental construction of the transition system and the search for a path satisfying the specification are explained. Finally, the algorithm that guides the expansion of the graph is explained in detail.

7.2.2 Probabilistic roadmap

The first step of the proposed method consists of creating probabilistic roadmaps for each robot $i \in \mathcal{R}$. A roadmap of a robot i models a subset of the possible trajectories of the robot and is formed by a set of sampled configurations $x \in X^i_{\text{free}}$ connected by collision-free paths. A graph $G^i = (V^i, E^i)$ is used to represent the roadmap of the robot i. Each vertex $v \in V^i$ is associated with a unique robot configuration $x \in X^i_{\text{free}}$. This association is given by the

function $\chi : V^i \to X^i$. Connectivity between two configurations is represented by an edge $(v, v') \in E^i$. All the vertices v' that share an edge with v are called neighbours of v. To verify the satisfaction of a specification using only the atomic propositions that are true in each state of the transition system, see Section 7.2.4, the edges of each G^i are limited to those edges that intersect the boundary of a region in the workspace at most once. Moreover, to reduce the size of the roadmaps, sparse roadmaps [42] are used. Since each vertex $v \in V^i$ is associated with a configuration $x \in X^i_{\text{free}}$, with abuse of notation, $L(v)$ is used to denote the atomic propositions satisfied by $\chi(v)$. The set of vertices on a roadmap G^i that satisfy an atomic proposition $\pi \in \Pi$ is denoted by $[\![\pi]\!]_i$.

To consider the configuration of all the robots, a composite roadmap [161] $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ is constructed as the tensor product of the individual roadmaps $\{G^i\}_{i=1}^R$, Fig. 7.1. Formally, $\mathbf{v} = (v^1, \ldots, v^R)$ is a vertex of $\mathbb{G}$ if $v^i \in V^i$ for all $i \in \mathcal{R}$ and $\chi(\mathbf{v}) \in X_{\text{free}}$. Let $\mathbf{v} = (v^1, \ldots, v^R)$ and $\mathbf{v}' = (v'^1, \ldots, v'^R)$ be two vertices in $\mathbb{G}$. In a tensor product, an edge $(\mathbf{v}, \mathbf{v}') \in \mathbb{E}$ is defined if for every $i \in \mathcal{R}$, $(v^i, v'^i) \in E^i$. The projection of a composite vertex $\mathbf{v} \in \mathbb{V}$ onto the vertex $v^i \in V^i$ of robot i is denoted by $\mathbf{v} \downarrow_i$, i.e. $\mathbf{v} \downarrow_i = v^i$. The atomic propositions satisfied by a vertex $\mathbf{v} = (v^1, \ldots, v^R)$ are the union of the atomic propositions satisfied by the individual vertices forming $\mathbf{v}$, i.e. $L(\mathbf{v}) = \cup_{i=1}^R L(v^i)$, where $v^i = \mathbf{v} \downarrow_i$.

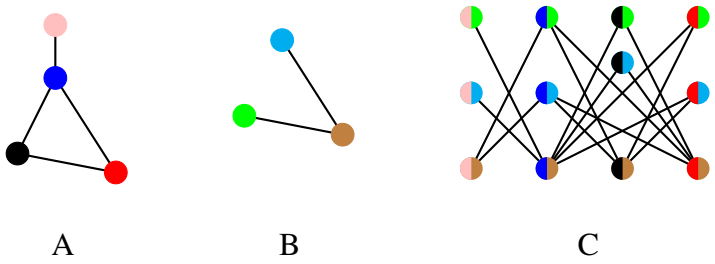


Fig. 7.1 Illustration of tensor product, graph C, of graphs A and B. Vertices of the tensor product are formed by the combination of vertices of graphs A and B. Two vertices are connected in C if a transition exists in A and B connecting the individual vertices forming the vertices in C.

As explained in the introduction of the chapter, it is possible to find a path for each robot satisfying a specification by creating a product automaton between the composite roadmap $\mathbb{G}$ and the Büchi automaton $\mathcal{B}_\varphi$ of the specification φ. However, this procedure is only applicable for small problems due to its poor scalability; the number of vertices in $\mathbb{G}$ is $|V|^R$. To avoid this problem, the method implicitly represents $\mathbb{G}$ by iteratively creating a transition system that models a subset of $\mathbb{G}$. The construction of the transition system is presented in the next subsection.

7.2.3 Composite roadmap exploration

A transition system $\mathcal{T} = (S, s_0, \delta_{\mathcal{T}}, \Pi, L)$ is used to model the explored part of the composite roadmap $\mathbb{G}$. To differentiate the vertices $v \in \mathbb{V}$ added to $\mathcal{T}$ from those not added to $\mathcal{T}$, s is used instead of v in the rest of this section. Initially, $\mathcal{T}$ contains only the state that represents the vertex corresponding to the initial configuration of all robots, i.e. $s_0 = (v_0^1, v_0^2, \ldots, v_0^R)$, where $\chi(v_0^i) = x_0^i \; \forall i \in \mathcal{R}$. Then, more vertices from $\mathbb{G}$ are added to $\mathcal{T}$ using the idea of discrete RRTs [158] as follows.

First, a state $s = (v^1, \ldots, v^R) \in S$ is randomly selected from the transition system $\mathcal{T}$. Then, each of the elements v^i of s is expanded using the following procedure. A configuration $x^i_{\text{sample}} \in X^i$ is randomly sampled. Next, the rays $\rho_{v^i, v^{i,j}}$, for all $j \in \{1, \ldots, l\}$, that start from v^i and pass through the l neighbours of v^i are computed. Similarly, the ray $\rho_{v^i, x_{\text{sample}}}$ passing through x^i_{sample} is calculated. To choose a neighbour of v^i in G^i to be added to $\mathcal{T}$, the angles between the ray $\rho_{v^i, x_{\text{sample}}}$ and each of the rays $\rho_{v^i, v^{i,j}}$ are computed. The neighbour that generates the ray with the smallest angle is selected and denoted as v^i_{new}, Fig. 7.2. This process is repeated for all the vertices forming s, resulting in a candidate state $s_{\text{new}} = (v^1_{\text{new}}, \ldots, v^R_{\text{new}})$.

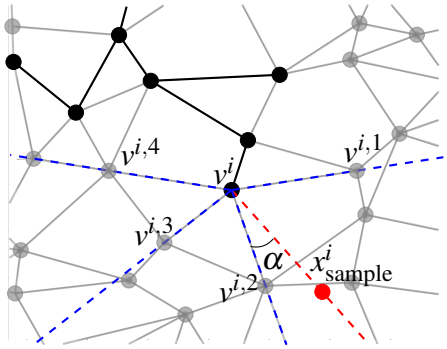


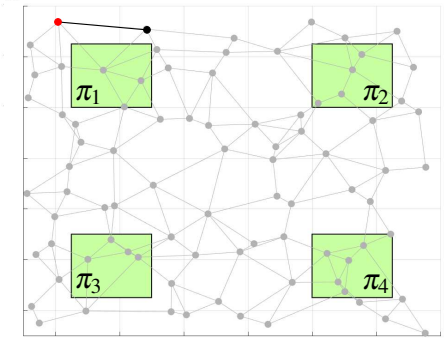
Fig. 7.2 Selection of vertex and edge in roadmap G^i. The states and transitions in $\mathcal{T}$ are illustrated with black vertices and edges. The roadmap G^i is shown in grey. To choose which neighbour $\{v^{i,j}\}_{j=1}^4$ of v^i is added to $\mathcal{T}$, the rays starting from v^i and passing through x^i_{sample} and the neighbours of v^i are computed. The smallest angle, α in the figure, determines which neighbour and edge are added to $\mathcal{T}$, neighbour $v^{i,2}$ in this example.

Before adding s_{new} to $\mathcal{T}$, it is verified whether a collision between robots exists. To avoid collisions during the transitions (v^i, v^i_{new}) for all $i \in \mathcal{R}$, priorities are assigned to each robot according to the following rules [175]:

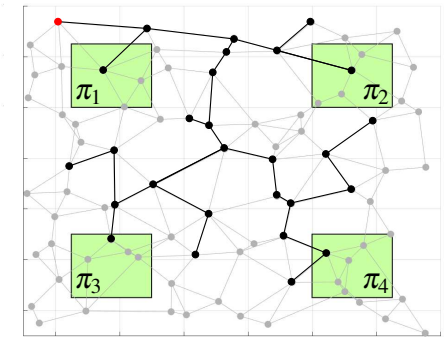
i. If robot i, transitioning from v^i to v^i_{new}, causes a collision with robot j, located in v^j_{new}, the robot i receives higher priority than j.

ii. If robot i collides with robot j placed in v^j during the transition (v^i, v^i_{new}), then, robot i receives lower priority than j.

The state s_{new} is discarded if there is no ordering such that collisions are avoided. Otherwise, the state is added to $\mathcal{T}$ with the transitions (s, s_{new}) and (s_{new}, s). Note that by choosing only neighbours of each individual vertex v^i, $(v^1_{\text{new}}, \ldots, v^R_{\text{new}})$ is an element of the composite roadmap $\mathbb{G}$. Intuitively, the transition system $\mathcal{T}$ represents the explored part of $\mathbb{G}$. An example of such exploration, for the case $R = 1$, is shown in Fig. 7.3.



(a)



(b)

Fig. 7.3 Incremental construction of a transition system. The roadmap of the robot is shown in grey. The roadmap is used to create a transition system. (a) The transition system (shown in black), initially containing only the initial position of the robot (red vertex), is expanded by choosing a vertex and edge from the roadmap. (b) The expansion continues until the specification, visiting the four green regions in this example, is satisfied.

Since each vertex v of $\mathbb{G}$ is associated with a configuration $x \in X_{\text{free}}$, a run $\boldsymbol{s} = s_0 s_1 \ldots$ on $\mathcal{T}$ represents a path of the full system in the configuration space X_{free}. Hence, this exploration continues until a path that satisfies the specification φ is found. The procedure to determine whether the current transition system contains such a path is presented in the next subsection.

7.2.4 Product automaton update

Based on model checking techniques, the verification of a run $\boldsymbol{s}$ satisfying the sc-LTL specification φ is made on the Cartesian product $\mathcal{P} = (S_{\mathcal{P}}, s_{\mathcal{P},0}, \delta_{\mathcal{P}}, F_{\mathcal{P}})$, where:

- $S_{\mathcal{P}} = S \times Q$ is a finite set of states,
- $s_{\mathcal{P},0} = s_0 \times q_0$ is an initial state,
- $\delta_{\mathcal{P}} \in S_{\mathcal{P}} \times S_{\mathcal{P}}$ is a transition relation, where $((s,q),(s',q')) \in \delta_{\mathcal{P}}$ iff $(s,s') \in \delta_{\mathcal{T}}$ and $\delta_{\mathcal{B}}(q, L(s')) = q'$,
- $F_{\mathcal{P}} = S \times F$.

The product automaton $\mathcal{P}$ is first created when the transition system contains only the initial state s_0. Since the transition system $\mathcal{T}$ changes with each new state s_{new}, the product $\mathcal{P}$ requires to be updated. The procedure to incrementally update $\mathcal{P}$ and to search for a path satisfying φ is now presented.

When a new state s_{new} and transition (s, s_{new}) are added to $\mathcal{T}$, the set $S'_{\mathcal{P}}$ of states $s'_{\mathcal{P}} = (s_{\text{new}}, q')$, such that $\delta_{\mathcal{B}}(q, L(s_{\text{new}})) = q'$ and $(s,q) \in S_{\mathcal{P}}$, is computed. Then, for each state $s'_{\mathcal{P}} \in S'_{\mathcal{P}}$, it is verified if $s'_{\mathcal{P}}$ is already in $S_{\mathcal{P}}$. If that is not the case, the state is added to $\mathcal{P}$ and is removed from $S'_{\mathcal{P}}$. Moreover, the set of states $s''_{\mathcal{P}} = (s', q'')$, such that $(s_{\text{new}}, s') \in \delta_{\mathcal{T}}$ and $\delta_{\mathcal{B}}(q', L(s')) = q''$, is computed. If a state $s''_{\mathcal{P}}$ is not already in $S_{\mathcal{P}}$, $s''_{\mathcal{P}}$ is added to $S_{\mathcal{P}}$ and to $S'_{\mathcal{P}}$. This recursive procedure continues until the set $S'_{\mathcal{P}}$ is empty.

By construction, if a run on $\mathcal{P}$, starting from $s_{\mathcal{P},0}$, reaches a state in the set $F_{\mathcal{P}}$ of accepting states, the word produced by the run $\boldsymbol{s}$ on $\mathcal{T}$ is accepted by the Büchi automaton $\mathcal{B}$ computed from the sc-LTL formula. In other words, a run $\boldsymbol{s}$ on $\mathcal{T}$ satisfies the specification if a run on $\mathcal{P}$ reaches the set of accepting states. Hence, the process of exploring $\mathbb{G}$ and updating $\mathcal{P}$ continues until a state $s_{\mathcal{P}} = (\cdot, q)$ is added to $\mathcal{P}$ such that $q \in Q_F$. The procedure for exploring $\mathbb{G}$ and updating $\mathcal{P}$ is shown in Alg. 7.1.

Using the procedure described above, a solution to the problem addressed in this chapter would be eventually found. Nevertheless, depending on the number of robots and the specification, this process could take an impractical amount of time. To reduce the time, in the next subsection, an novel algorithm that guides the exploration of $\mathbb{G}$ is presented.

Algorithm 7.1. IncrementalExploration($\{G^i\}_{i=1}^R, \mathcal{B}, X$)

1: $S \leftarrow s_0 = (v_0^1, \ldots, v_0^R)$;
2: $\mathcal{P} \leftarrow \mathcal{T} \times \mathcal{B}$;
3: **while** $s_\mathcal{P} = (s,q) \notin S_\mathcal{P} : q \in Q_F$ **do**
4: $\quad s_{\text{new}}, (s, s_{\text{new}}), (s_{\text{new}}, s) \leftarrow$ EXPLORECOMPROADMAP$(\{G^i\}_{i=1}^R), X)$;
5: $\quad S \leftarrow S \cup s_{\text{new}}$;
6: $\quad \delta_\mathcal{T} \leftarrow \delta_\mathcal{T} \cup \{(s, s_{\text{new}}), (s_{\text{new}}, s)\}$;
7: $\quad \mathcal{P} \leftarrow$ UPDATEAUTOMATON$(\mathcal{P}, s_{\text{new}}, (s, s_{\text{new}}), (s_{\text{new}}, s))$;
8: **return** $\mathcal{T} = (S, s_0, \delta_\mathcal{T}, \Pi, L)$;

7.2.5 Guided exploration

The guided exploration of the composite roadmap $\mathbb{G}$ is based on the selection of a state in $\mathcal{T}$ that must be expanded in order to satisfy the sc-LTL specification. The main idea is to find the shortest path, in terms of transitions, in the Büchi automaton to an accepting state and use the atomic propositions required in such a path to search in the individual roadmaps $\{G^i\}_{i=1}^R$.

Before explaining the algorithm, the concept of a waiting robot is presented. Depending on the transition in the Büchi automaton, one or more atomic propositions have to be satisfied at the same time. Since each robot can only satisfy one atomic proposition at a time, when more than one proposition is required, collaboration between robots is needed. When a robot is able to reach a state satisfying one of the required atomic propositions, the robot remains in its current state. This robot is called a waiting robot and it remains in the same state until other robots are capable of reaching states that satisfy the rest of the atomic propositions. The algorithm is now explained in detail, Alg. 7.2.

The algorithm receives as input the set $S'_\mathcal{P}$ of states added to the product automaton $\mathcal{P}$ after the last update. These states have the form (s,q), where $s = (v^1, \ldots, v^R) \in S$ and $q \in Q$. The states are divided into different sets depending on their Büchi state component (line 1, Alg. 7.2). In other words, for each state q_i in the Büchi automaton, a set g_i containing states $s = (v^1, \ldots, v^R)$ such that $s_\mathcal{P} = (s, q_i) \in S'_\mathcal{P}$ is created. These sets are used to search states in the PRMs such that a transition in $\mathcal{B}$, starting from q_i, is produced by the atomic propositions satisfied in the PRM states. Then, the algorithm sorts, from shortest to longest, the different paths from the initial state $q_0 \in Q$ to the closest accepting state $q \in Q_F$. Using these sorted paths, the algorithm tries to reach atomic propositions required in the paths, starting from the shortest path (line 2, Alg. 7.2).

For each state s in g_i, the individual vertices forming s of all non-waiting robots are considered to be connected to vertices in G^i satisfying the required atomic propositions

Algorithm 7.2. LocalConnector ($\{G_i\}_{i=1}^{R}$,$\mathcal{B}$,$\mathcal{T}$,$S'_{\mathcal{P}}$)

1: $g_i \leftarrow \{s : (s,q_i) \in S'_{\mathcal{P}},\ \forall i \in \{1,\ldots,|Q|\}\}$
2: **for** $q_i \in \textsc{Sort}(Q)$ **do**
3: **for** $j \in \{\mathcal{R} : j \notin W_{\text{Robot}}\}$ **do**
4: **for** $s_n \in g_i$ **do**
5: $v_j = s_n \downarrow_j$;
6: **for** $q_k \in \textsc{Sort}(\delta_{\mathcal{B}}(q_i,\cdot))$ **do**
7: $\Pi_{\text{req}} \leftarrow \textsc{ReqAP}(q_i,q_k)$;
8: $v_c \leftarrow \textsc{Connect}(v_j, [\![\pi_m]\!]_{m=1,j}^{|\Pi_{\text{req}}|})$;
9: **if** $v_c \neq \emptyset$ **then**
10: **if** TRANSCOMPLETE **then**
11: $S \leftarrow S \cup s_{\text{new}}$;
12: $\delta_{\mathcal{T}} \leftarrow \delta_{\mathcal{T}} \cup \{(s_n, s_{\text{new}}), (s_{\text{new}}, s_n)\}$;
13: $W_{\textsc{Robot}} \leftarrow \emptyset$;
14: **else**
15: $W_{\textsc{Robot}} \leftarrow W_{\textsc{Robot}} \cup j$;

in the Büchi automaton transition. An individual vertex, denoted as v_j, is considered for connection in each iteration (lines 3-5, Alg. 7.2). The transition that is attempted to be satisfied in the Büchi automaton is selected based on the sorted paths (line 6, Alg. 7.2). The required atomic propositions in the selected transition are assigned to the set Π_{req} (line 7, Alg. 7.2). Then, all the vertices in the roadmap G^j that satisfy an atomic proposition that cannot be satisfied by a waiting robot are assigned as a target of the connection (line 8, Alg. 7.2). By connecting the transition system to vertices satisfying atomic propositions required for the specification, the time needed to solve the proposed problem is reduced.

In order to find a path between the vertices v_j and vertices in G^j satisfying a required atomic proposition, any method can be used. However, because this process is constantly repeated, a method that sacrifices completeness for speed is preferred. This is a common practice as presented in Section 2.1.5. Note that the type of method used for the connections defines the type of dynamics accepted by the proposed method. If the path between v_j and v_c, the connected vertex, is collision-free, the connection is considered successful (line 9, Alg. 7.2). Depending on the number of atomic propositions in the selected transition in $\mathcal{B}$, three different situations can occur:

Case 1: Only one atomic proposition is required in the transition, i.e. $|\Pi_{req}| = 1$. In this case, if robot j can satisfy the required atomic proposition through the connection, a new state $s_{\text{new}} = (v^1,\ldots,v^R)$ is created. Intuitively, the new state has the same components as the composite state s, except the element of robot j that is replaced by v_c. If the new state is

not in the transition system $\mathcal{T}$, the state is added with the transitions (s_j, s_{new}) and (s_{new}, s_j) (lines 11-12, Alg. 7.2).

Case 2: More than one atomic proposition is required and at least one more is still required after the connection. When a robot j can satisfy one of the required atomic propositions but at least one more is needed for the transition in the Büchi automaton, the robot stays in the vertex v_j waiting for the remaining robots to satisfy the other atomic propositions. To indicate that the robot is waiting, the index j is added to the set W_{Robot} (line 15, Alg. 7.2). This set restricts the states that can be selected in the exploration of $\mathbb{G}$. The selected state in the exploration must be formed by the vertices v^i, where $i \in W_{\text{Robot}}$. After adding the index j of the robot to the set W_{Robot}, the vertex that can be reached, i.e. v_c, is saved to be used once all the atomic propositions of the selected automaton transition can be satisfied. Note that the restriction explained above guides the sampling process of $\mathbb{G}$.

Case 3: The last required atomic proposition is satisfied with the connection. Similar to case 1, when a robot j can satisfy the last required atomic proposition, a new state s_{new} is created with v_c and the saved states. This state is added to $\mathcal{T}$ with the transitions (s_j, s_{new}) and (s_{new}, s_j) and the set W_{Robot} becomes empty indicating that all the robots can move again (lines 11-13, Alg. 7.2).

Every time a new state s_{new} is added to $\mathcal{T}$, the product automaton $\mathcal{P}$ is updated and the process of guiding the expansion is repeated. As mentioned in the previous section, the expansion of $\mathcal{T}$ stops once a product state with a final state $q \in Q_F$ is added to $\mathcal{P}$.

7.2.6 Implementation

This subsection presents how a solution is obtained from $\mathcal{P}$ together with the implementation of it in the robots. Once the exploration of $\mathbb{G}$ stops, the shortest path $\boldsymbol{s}_{\mathcal{P}} = s_{\mathcal{P},0} \dots s_{\mathcal{P},n}$ on $\mathcal{P}$, where $s_{\mathcal{P},n} \in F_{\mathcal{P}}$, is sought. Since a state $s_{\mathcal{P}}$ is formed by the pair of the form (s, q), only the first element of each state is considered to create the path $\boldsymbol{x}$ that satisfies the sc-LTL specification. Each configuration in $\boldsymbol{x}$ is projected to the individual vertices in $\{G^i\}_{i=1}^{R}$. Finally, the function χ is used to find the configurations in X^i to define a path for each robot.

To execute the path, each robot stores a list of the vertices to visit in its roadmap together with the configurations where the robot has to wait for other robots before performing a transition. When a robot finishes a transition, it broadcasts a unique identifier number and a signal indicating that the transition has been completed. If a robot needs to wait for other robots, the transition is not performed until the robot receives the signal of all the robots with higher priority.

7.3 Examples

The proposed method is illustrated with different sc-LTL specifications and number of robots. A differential wheeled robot, called e-puck [123], operating in a workspace with 4 areas associated with the atomic propositions π_1, π_2, π_3 and π_4 is considered. The computation of the path is implemented in MATLAB on a computer with a 3.1 GHz i7 processor and 8GB of RAM. The dynamics of the e-pucks are simulated using Enki [119].

Two different examples, considering two and four robots, respectively, are considered:

1. Regions π_1, π_2 have to be visited at the same time as well as π_3, π_4 with the same restriction. LTL specification: $\varphi_1 = \Diamond(\pi_1 \wedge \pi_2) \wedge \Diamond(\pi_3 \wedge \pi_4)$, Fig. 7.4.

2. Regions π_1, π_2, π_3 and π_4 cannot be visited until all of them are visited at the same time. LTL specification: $\varphi_2 = \neg(\pi_1 \vee \pi_2 \vee \pi_3 \vee \pi_4)\mathcal{U}(\pi_1 \wedge \pi_2 \wedge \pi_3 \wedge \pi_4)$, Fig. 7.5.

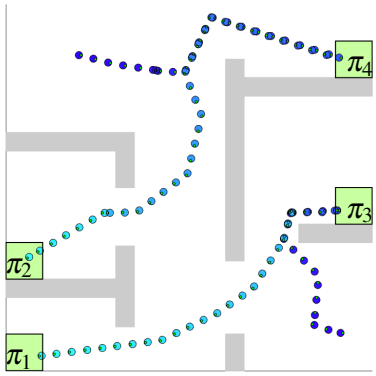


Fig. 7.4 Illustration of the path followed by two robots satisfying the specification $\varphi_1 = \Diamond(\pi_1 \wedge \pi_2) \wedge \Diamond(\pi_3 \wedge \pi_4)$. This sc-LTL specification requires the robots to visit the areas marked as π_1 and π_2 at the same time and the areas π_3 and π_4 with the same restriction. The colour of the robots changes, from darker to lighter blue, over time to show that the atomic propositions are satisfied at the same time step.

The required mean time, over 20 runs, to solve these examples is presented in Table 7.1. From this table, it can be seen that a solution can be found in a few seconds for problems with a big number of states. For instance, in the example with four robots, the parallel composition $\mathbb{G}$ has more than 96 million vertices. This short time can be attributed to the guided exploration. To show the effect of using the algorithm to guide the exploration, the first example is solved using only the exploration as presented in Section 7.2.3. On

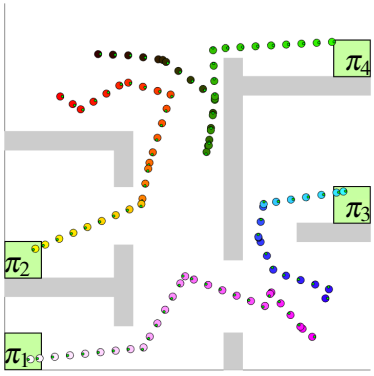


Fig. 7.5 Illustration of the path followed by two robots satisfying the specification $\varphi_2 = \neg(\pi_1 \vee \pi_2 \vee \pi_3 \vee \pi_4)\mathcal{U}(\pi_1 \wedge \pi_2 \wedge \pi_3 \wedge \pi_4)$. This sc-LTL specification requires the robots to visit the four marked areas at the same time. The colour of the robots changes, from darker to lighter colour, over time to show that the atomic propositions are satisfied at the same time step.

Table 7.1 Average required time (seconds) to solve the problems, number of robots in the workspace, and number of states in transition system $\mathcal{T}$ over 20 different runs.

Specification	Time (mean)	Std. Dev.	Number of robots	States in $\mathcal{T}$
φ_1	6.30	1.53	2	278.55
φ_2	7.48	1.56	4	270.4

average, the solution is found in 1057.91 seconds and required the exploration of 7242.43 vertices. In contrast, using the guided exploration, the solution is found in 6.30 seconds. A direct comparison with other sampling-based methods for multiple robots, e.g. [74], is not possible because the proposed method samples the continuous configuration space instead of a discrete representation of the robot's mobility.

7.4 Concluding remarks

In this chapter, a new method to find collision-free paths that satisfy syntactically co-safe linear temporal logic formulae for multi-robot systems has been proposed. Most of the available methods assume precomputed controls to drive the system between partitions of an environment or have poor scalability with respect to the dimension of the robot's configuration space due to a required discretisation. The proposed method extends sampling-based

methods, previously proposed to alleviate the scalability problem, to multi-robot systems. Hence, fewer assumptions are imposed. The method explores a composite roadmap modelling the possible behaviour of all the robots. This exploration stops when a path satisfying the specification is found. Additionally, a new algorithm that guides the exploration to reduce the time required to find a solution has been presented. Numerical results show that only a small portion of the composite roadmap is explored as a result of using this algorithm. A comparison to other methods is not possible because of the considered assumptions, e.g. precomputed environment partitions. Since the method is geared towards fast solutions, a disadvantage of the proposed solution is that it does not return optimal paths in contrast to other methods that consider a partitioned environment [171]. Moreover, it is limited to sc-LTL specifications.

Chapter 8

Conclusions and Future Work

8.1 Summary and conclusions

The problem of motion planning has been studied for decades. Typically, this planning considers elements such as system dynamics, uncertainties and dynamic environments. However, as the number of applications where interaction with humans increases, it is essential to consider factors such as safety and reliability. The search for methods to consider these requirements has resulted in a new area of research. This area combines traditional motion planning algorithms, and more general, control theory with model checking techniques to create methods capable of synthesising controllers that guarantee the execution of high-level specifications. One of the main challenges in the development of these methods is the scalability due to factors such as high-dimensional systems, uncertainties, type of specifications and multi-robot systems. In this thesis, methods focused on reducing the complexity have been proposed to better manage the scalability.

The proposed methods are based on sampling techniques that have been used in traditional point to point motion planning to handle systems with high-dimensional state spaces. In contrast to other motion planning methods, sampling-based algorithms do not compute explicitly the state space of the system. This allows to obtain a better scalability compared to other methods. These sampling-based techniques are combined with automata theory to solve motion planning problems subject to high-level specifications expressed with temporal logics. A variety of scenarios considering optimality, different types of uncertainty and multi-robot systems were considered.

Optimal trajectories based on sc-LTL specifications for deterministic high-dimensional kinodynamic systems were computed by using the SST_LTL algorithm in Chapter 4. This algorithm iteratively constructs a transition system with states augmented by states of a Büchi automaton until a condition is satisfied. The probabilistic completeness and optimal-

ity were proved and demonstrated with illustrative examples considering a 10-dimensional quadrotor. In contrast to current solutions, this algorithm requires only the forward propagation of the system dynamics to find optimal trajectories. As a result, the algorithm can be applied to a wider range of dynamics. Another difference is the bounded size of the transition system compared with other sampling-based methods that constantly increase in size with the number of iterations. This is an important characteristic due to the large number of iterations required to converge to an optimal solution. The benefits obtained by the proposed approach comes at the cost of a restricted type of temporal specification, i.e. sc-LTL specifications. Because the approach does not connect states directly, loops in the transition system cannot be created. As a result, temporal logic specifications with infinite-horizon properties cannot be satisfied.

The previous method computes a sequence of control inputs to achieve an optimal path. This is done in an open-loop fashion since deterministic systems are considered. Nevertheless, when a system is affected by uncertainty in motion, a closed-loop controller is required. In Chapter 5, a method to compute optimal policies in terms of maximising the probability of satisfying a specification with time constraints expressed as MITL was presented. Approaches in the literature construct MDPs to represent the stochasticity of the system. However, in order to approximate the continuous-time continuous-space of the underlying system, a fine discretisation is required resulting in a large number of states. The proposed method reduces this problem by using a discretisation of the workspace instead of the state space. By considering a space with lower dimensionality, the scalability of the method increases. The mobility of the robot in this discretised workspace is modelled by a BMDP, where the transitions are computed using a sampling-based method. The results showed that solutions can be computed faster compared to current approaches. On the other hand, the results also showed that as a result of the coarse partition used to reduce the complexity of the problem, the trajectory followed by the system presents some abrupt changes in direction. In other words, the trajectory is not as smooth as in other approaches. Hence, the approach has a trade-off the between computational speed and the smoothness of the trajectory.

On the other hand, systems can be also affected by uncertainties in sensing due to noisy sensors. This could result in uncertainty in the knowledge of the system state. In other words, the precise state of the system is not known and only a probability distribution over all the states is available. Most of the available solutions to such problems do not consider the continuous state space of the system due to the complexity generated by the curse of dimensionality and history. Moreover, static workspaces are mostly considered in this type of problem. In Chapter 6, a method that computes optimal policies based on LTL specifica-

tions for systems with uncertainty in motion and sensing operating in dynamic environments was presented. Similar to the previous case, the policy maximises the probability of satisfying the specification during the normal operation of the system. When the system detects a previously unknown obstacle or target, the system switches to a policy that minimises the probability of violating the specification until the detected element disappears. To break the aforementioned curses, the method constructs a transition system in the belief space of the system and computes SLQG controllers as stabilisers. To handle changes in the workspace, a graph in the belief space is precomputed and used to avoid obstacles or reach targets. Results demonstrated that the system is able to react to changes in the workspace in a short period of time making the method applicable to more realistic scenarios. A limitation of the proposed approach is the type of dynamic system that the method can handle. Since the system must be stabilised at certain belief nodes, the system must be able to reach states with zero velocities [1]. However, this is not always possible such as in fixed-wing aircrafts.

In the last chapter, Chapter 7, sampling-based methods were extended to solve the problem of motion planning for multi-robot systems subject to sc-LTL specifications. One of the main problems with multiple-robot systems is the large number of possible states. This problem is aggravated for systems with high-dimensional state spaces. Current solutions use discretisation or potential functions to drive the system, both of which have poor scalability. The proposed method constructs a transition system using precomputed PRMs. The transition system is combined with an automaton to find a path satisfying the specification. To accelerate the process of finding a solution, an algorithm that guides the construction of the transition by taking advantage of the PRMs was proposed. A comparison of the sampling-based method with and without the algorithm was presented. The results showed that the guided construction reduces the required number of state in the transition to find a solution. Because the approach is focused on reducing the number of states required to find a solution, the optimality of the path is not considered. Therefore, the returned solution can contain unnecessary long paths.

Overall, the use of sampling techniques with automata theory have resulted in methods that allow the computation of trajectories for systems with complex dynamics and at the same time to increase the scalability of the methods. The efficiency of these methods has been shown in this thesis considering diverse dynamics, uncertainties and multi-robot systems. From these results, it can be concluded that the methods proposed in this thesis can increase the applicability of methods combining control theory and model checking for motion planning for different type of systems and scenarios. The contributions of this thesis are now summarised:

- A method capable of finding optimal trajectories for high-dimensional kinodynamic system subject to co-safe linear temporal logic specifications. In contrast to other methods that can only be applied to limited classes of system dynamics, the proposed approach can be applied to a wider range of dynamics.
- A method to solve motion planning problems considering real-time constraints, given as a metric interval temporal logic, and uncertainty in motion. The presented approach presents better computational tractability compared to current methods.
- A method to compute optimal trajectories for systems with uncertainty in motion and sensing subject to linear temporal logic specifications. In contrast to other methods, the method allows systems to react to previously unknown elements such as obstacles.
- A method for multi-robot systems with co-safe linear temporal logic specifications and an algorithm to reduce the number of states required to find a solution. The method is based on sampling approaches which present better scalability compared to current methods that used discretisation or navigations functions.

8.2 Future work

While the proposed methods are capable of computing trajectories satisfying temporal logic specifications for a variety of systems, these methods can be extended in several directions.

The method in Chapter 4 is limited to sc-LTL specifications. While this logic can be used to express many useful specifications for robotic systems, it does not allow specifications with infinite horizon. Extending the proposed method to allow full LTL specifications would bring new challenges. For instance, most of the methods considering full LTL rely on the computation of trajectories forming a loop or lasso. However, computing such trajectories is not trivial without solving a two-point boundary value problem, which limits the type of dynamics for which a trajectory can be computed.

The method proposed in Chapter 5 can be improved in two directions. The first one is the smoothness of the computed trajectory. Because of the discretised workspace and the optimal policies computed to drive the system from one segment to another, the computed path creates a zigzag pattern. A possible solution to this problem is to reduce the size of the segments. Nevertheless, this would increase the number of states in the product automaton. A possible different solution is to create more than one policy to transition to a segment. By having more than one policy, a policy could be chosen depending on the target's direction. However, an analysis of the optimality would be required. The second direction is the

extension of the allowed specifications to include infinite horizon specifications. Although a loop could be created on the partitioned workspace, the method would need to reason about time in an infinite horizon.

While the method proposed in Chapter 6 considers uncertain workspaces that could change, static obstacles are considered. Hence, a possible direction for future work is the inclusion of dynamic obstacles that could move and possibly try to falsify the specification. This problem can be addressed by solving a two-player game with partial observations. This problem is currently computational impractical. Moreover, methods that use this approach do not consider continuous state spaces in contrast to the proposed method.

Similar to the cases mentioned above, the method in Chapter 7 could be extended to include full LTL specifications. Moreover, the proposed method does not consider a cost function to find a trajectory satisfying the specification. Therefore, another direction to improve the work is the computation of optimal paths. These two elements, full LTL and optimality, would require the consideration of all the possible states to find an optimal trajectory. This can be achieved using the composite roadmap. However, the large number of states in it make this approach impractical. Therefore, a new sampling-based method and algorithm to guide the expansion of the transition system would be required.

References

[1] Agha-Mohammadi, A.-A., Chakravorty, S., and Amato, N. M. (2014). Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research*, 33(2):268–304.

[2] Alterovitz, R., Siméon, T., and Goldberg, K. Y. (2007). The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty. In *Proceedings of Robotics: Science and Systems*, volume 3, pages 233–241.

[3] Alur, R. and Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235.

[4] Alur, R., Feder, T., and Henzinger, T. A. (1996). The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146.

[5] Alur, R., Henzinger, T. A., Lafferriere, G., and Pappas, G. J. (2000). Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984.

[6] Amato, N. M., Bayazit, O. B., Dale, L. K., Jones, C., and Vallejo, D. (1998). Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proceedings of International Conference on Robotics and Automation*, volume 1, pages 630–637. IEEE.

[7] Ayala, A. M., Andersson, S. B., and Belta, C. (2013). Temporal logic motion planning in unknown environments. In *Proceedings of Conference on Intelligent Robots and Systems*, pages 5279–5284. IEEE.

[8] Baier, C., Größer, M., and Bertrand, N. (2012). Probabilistic ω-automata. *Journal of the ACM*, 59(1):1.

[9] Baier, C. and Katoen, J. P. (2008). *Principles of Model Checking*. MIT Press Cambridge.

[10] Balch, T. and Hybinette, M. (2000). Social potentials for scalable multi-robot formations. In *Proceedings of International Conference on Robotics and Automation*, volume 1, pages 73–80. IEEE.

[11] Bauer, A., Leucker, M., and Schallhart, C. (2011). Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology*, 20(4):14.

[12] Behrmann, G., David, A., Larsen, K. G., Hakansson, J., Petterson, P., Yi, W., and Hendriks, M. (2006). Uppaal 4.0. In *Proceedings of International Conference on Quantitative Evaluation of Systems*, pages 125–126. IEEE.

[13] Bellingham, J., Richards, A., and How, J. P. (2002). Receding horizon control of autonomous aerial vehicles. In *Proceedings of American Control Conference*, volume 5, pages 3741–3746. IEEE.

[14] Bellingham, J., Tillerson, M., Richards, A., and How, J. P. (2003). *Multi-task allocation and path planning for cooperating UAVs*. Springer.

[15] Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., and Pappas, G. J. (2007). Symbolic planning and control of robot motion. *IEEE Robotics & Automation Magazine*, 14(1):61–70.

[16] Belta, C., Yordanov, B., and Gol, E. A. (2017). *Formal Methods for Discrete-Time Dynamical Systems*, volume 89. Springer.

[17] Bemporad, A. and Morari, M. (1999). Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427.

[18] Bhatia, A., Kavraki, L. E., and Vardi, M. Y. (2010). Sampling-based motion planning with temporal goals. In *Proceedings of International Conference on Robotics and Automation*, pages 2689–2696. IEEE.

[19] Bialkowski, J., Otte, M., Karaman, S., and Frazzoli, E. (2016). Efficient collision checking in sampling-based motion planning via safety certificates. *The International Journal of Robotics Research*, 35(7):767–796.

[20] Biere, A., Cimatti, A., Clarke, E., and Zhu, Y. (1999). Symbolic model checking without BDDs. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207.

[21] Blackmore, L., Ono, M., and Williams, B. C. (2011). Chance-constrained optimal path planning with obstacles. *IEEE Transactions on Robotics*, 27(6):1080–1094.

[22] Branicky, M. S., LaValle, S. M., Olson, K., and Yang, L. (2001). Quasi-randomized path planning. In *Proceedings of International Conference on Robotics and Automation*, volume 2, pages 1481–1487. IEEE.

[23] Bry, A. and Roy, N. (2011). Rapidly-exploring random belief trees for motion planning under uncertainty. In *Proceedings of International Conference on Robotics and Automation*, pages 723–730. IEEE.

[24] Caron, S., Pham, Q.-C., and Nakamura, Y. (2017). Completeness of randomized kinodynamic planners with state-based steering. *Robotics and Autonomous Systems*, 89:85–94.

[25] Castro, L. I. R., Chaudhari, P., Tumova, J., Karaman, S., Frazzoli, E., and Rus, D. (2013). Incremental sampling-based algorithm for minimum-violation motion planning. In *Proceedings of Conference on Decision and Control*, pages 3217–3224. IEEE.

[26] Chatterjee, K., Chmelík, M., Gupta, R., and Kanodia, A. (2015). Qualitative analysis of POMDPs with temporal logic specifications for robotics applications. In *Proceedings of International Conference on Robotics and Automation*, pages 325–330. IEEE.

[27] Chatterjee, K., Chmelík, M., and Tracol, M. (2016). What is decidable about partially observable markov decision processes with ω-regular objectives. *Journal of Computer and System Sciences*, 82(5):878–911.

[28] Chen, Y., Ding, X. C., and Belta, C. (2011). Synthesis of distributed control and communication schemes from global LTL specifications. In *Proceedings of Conference on Decision and Control and European Control Conference*, pages 2718–2723. IEEE.

[29] Cho, K., Suh, J., Tomlin, C. J., and Oh, S. (2017). Cost-aware path planning under co-safe temporal logic specifications. *IEEE Robotics and Automation Letters*, 2(4):2308–2315.

[30] Choset, H. (2001). Coverage for robotics–a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1):113–126.

[31] Choset, H. M. (2005). *Principles of robot motion: theory, algorithms, and implementation*. MIT press.

[32] Cizelj, I. and Belta, C. (2014). Control of noisy differential-drive vehicles from time-bounded temporal logic specifications. *The International Journal of Robotics Research*, 33(8):1112–1129.

[33] Clarke, E. M. and Emerson, E. A. (1981). Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of Workshop on Logic of Programs*, pages 52–71. Springer.

[34] Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model checking*. MIT press.

[35] Coe, T. (1995). Inside the pentium-FDIV bug. *Dr Dobb's Journal*, 20(4):129.

[36] CPLEX, I. I. (2009). V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53):157.

[37] DeCastro, J. A. and Kress-Gazit, H. (2015). Synthesis of nonlinear continuous controllers for verifiably correct high-level, reactive behaviors. *The International Journal of Robotics Research*, 34(3):378–394.

[38] Desai, J. P. and Kumar, V. (1999). Motion planning for cooperating mobile manipulators. *Journal of Robotic Systems*, 16(10):557–579.

[39] Dimarogonas, D. V. and Kyriakopoulos, K. J. (2007). Decentralized navigation functions for multiple robotic agents with limited sensing capabilities. *Journal of Intelligent & Robotic Systems*, 48(3):411–433.

[40] Ding, X. C., Smith, S. L., Belta, C., and Rus, D. (2011a). MDP optimal control under temporal logic constraints. In *Proceedings of Conference on Decision and Control and European Control Conference*, pages 532–538. IEEE.

[41] Ding, X. C. D., Smith, S. L., Belta, C., and Rus, D. (2011b). LTL control in uncertain environments with probabilistic satisfaction guarantees. In *International Federation of Automatic Control Proceedings Volumes*, volume 44, pages 3515–3520. Elsevier.

[42] Dobson, A. and Bekris, K. E. (2014). Sparse roadmap spanners for asymptotically near-optimal motion planning. *The International Journal of Robotics Research*, 33(1):18–47.

[43] Donald, B., Xavier, P., Canny, J., and Reif, J. (1993). Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066.

[44] Dubins, L. E. (1957). On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516.

[45] Farahani, S. S., Raman, V., and Murray, R. M. (2015). Robust model predictive control for signal temporal logic synthesis. *IFAC-PapersOnLine*, 48(27):323–328.

[46] Fliess, M., Lévine, J., Martin, P., and Rouchon, P. (1995). Flatness and defect of non-linear systems: introductory theory and examples. *International Journal of Control*, 61(6):1327–1361.

[47] Fu, J., Dimitrova, R., and Topcu, U. (2014). Abstractions and sensor design in partial-information, reactive controller synthesis. In *Proceedings of American Control Conference*, pages 2297–2304. IEEE.

[48] Fu, J., Papusha, I., and Topcu, U. (2017). Sampling-based approximate optimal control under temporal logic constraints. In *Proceedings of International Conference on Hybrid Systems: Computation and Control*, pages 227–235. ACM.

[49] Fu, J. and Topcu, U. (2015). Computational methods for stochastic control with metric interval temporal logic specifications. In *Proceedings of Conference on Decision and Control*, pages 7440–7447. IEEE.

[50] Fu, J. and Topcu, U. (2016). Synthesis of joint control and active sensing strategies under temporal logic constraints. *IEEE Transactions on Automatic Control*, 61(11):3464–3476.

[51] Galicki, M. (2017). The planning of optimal motions of non-holonomic systems. *Nonlinear Dynamics*, 90(3):2163–2184.

[52] Gastin, P. and Oddoux, D. (2001). Fast LTL to Büchi automata translation. In *Proceedings of International Conference on Computer Aided Verification*, pages 53–65. Springer.

[53] Geraerts, R. and Overmars, M. H. (2006). Sampling and node adding in probabilistic roadmap planners. *Robotics and Autonomous Systems*, 54(2):165–173.

[54] Gilbert, E. G., Johnson, D. W., and Keerthi, S. S. (1988). A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203.

[55] Givan, R., Leach, S., and Dean, T. (2000). Bounded-parameter Markov decision processes. *Artificial Intelligence*, 122(1-2):71–109.

[56] Glassman, E. and Tedrake, R. (2010). A quadratic regulator-based heuristic for rapidly exploring state space. In *Proceedings of International Conference on Robotics and Automation*, pages 5021–5028. IEEE.

[57] Gol, E. A. and Belta, C. (2014). An additive cost approach to optimal temporal logic control. In *Proceedings of American Control Conference*, pages 1769–1774. IEEE.

[58] Gol, E. A., Ding, X., Lazar, M., and Belta, C. (2014). Finite bisimulations for switched linear systems. *IEEE Transactions on Automatic Control*, 59(12):3122–3134.

[59] Gol, E. A., Lazar, M., and Belta, C. (2012). Language-guided controller synthesis for discrete-time linear systems. In *Proceedings of International Conference on Hybrid Systems: Computation and Control*, pages 95–104. ACM.

[60] Gottschalk, S., Lin, M. C., and Manocha, D. (1996). Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of Conference on Computer Graphics and Interactive Techniques*, pages 171–180. ACM.

[61] Guo, M. and Dimarogonas, D. V. (2015a). Bottom-up motion and task coordination for loosely-coupled multi-agent systems with dependent local tasks. In *Proceedings of International Conference on Automation Science and Engineering*, pages 348–355. IEEE.

[62] Guo, M. and Dimarogonas, D. V. (2015b). Multi-agent plan reconfiguration under local LTL specifications. *The International Journal of Robotics Research*, 34(2):218–235.

[63] Hauser, K. (2015). Lazy collision checking in asymptotically-optimal motion planning. In *Proceedings of International Conference on Robotics and Automation*, pages 2951–2957. IEEE.

[64] Hauser, K. and Zhou, Y. (2016). Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space. *IEEE Transactions on Robotics*, 32(6):1431–1443.

[65] Horowitz, M. B., Wolff, E. M., and Murray, R. M. (2014). A compositional approach to stochastic optimal control with co-safe temporal logic specifications. In *Proceedings of International Conference on Intelligent Robots and Systems*, pages 1466–1473. IEEE.

[66] Huynh, V. A. (2014). *Sampling-based algorithms for stochastic optimal control*. PhD thesis, Massachusetts Institute of Technology.

[67] Huynh, V. A., Karaman, S., and Frazzoli, E. (2016). An incremental sampling-based algorithm for stochastic optimal control. *The International Journal of Robotics Research*, 35(4):305–333.

[68] Huynh, V. A. and Roy, N. (2009). icLQG: combining local and global optimization for control in information space. In *Proceedings of International Conference on Robotics and Automation*, pages 2851–2858. IEEE.

[69] Hwang, Y. K. and Ahuja, N. (1992). A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32.

[70] Janson, L., Schmerling, E., Clark, A., and Pavone, M. (2015). Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research*, 34(7):883–921.

[71] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134.

[72] Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., and Schaal, S. (2011). Stomp: Stochastic trajectory optimization for motion planning. In *Proceeding of International Conference on Robotics and Automation*, pages 4569–4574. IEEE.

[73] Kallman, M. and Mataric, M. (2004). Motion planning using dynamic roadmaps. In *Proceedings of International Conference on Robotics and Automation*, volume 5, pages 4399–4404. IEEE.

[74] Kantaros, Y. and Zavlanos, M. M. (2017). Sampling-based control synthesis for multi-robot systems under global temporal specifications. In *Proceedings of International Conference on Cyber-Physical Systems*, pages 3–13. ACM.

[75] Karaman, S. and Frazzoli, E. (2008). Vehicle routing problem with metric temporal logic specifications. In *Proceedings of Conference on Decision and Control*, pages 3953–3958. IEEE.

[76] Karaman, S. and Frazzoli, E. (2009). Sampling-based motion planning with deterministic μ-calculus specifications. In *Proceedings of Conference on Decision and Control*, pages 2222–2229. IEEE.

[77] Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The international Journal of Robotics Research*, 30(7):846–894.

[78] Karaman, S. and Frazzoli, E. (2012). Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In *Proceedings of American Control Conference*, pages 735–742. IEEE.

[79] Karaman, S., Sanfelice, R. G., and Frazzoli, E. (2008). Optimal control of mixed logical dynamical systems with linear temporal logic specifications. In *Proceedings of Conference on Decision and Control*, pages 2117–2122. IEEE.

[80] Kavraki, L. E., Kolountzakis, M. N., and Latombe, J.-C. (1998). Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14(1):166–171.

[81] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580.

[82] Kemeny, J. G. and Snell, J. L. (1960). *Finite markov chains*. Springer.

[83] Klein, J. and Baier, C. (2006). Experiments with deterministic ω-automata for formulas of linear temporal logic. *Theoretical Computer Science*, 363(2):182–195.

[84] Kloetzer, M. and Belta, C. (2010). Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics*, 26(1):48–61.

[85] Kloetzer, M., Ding, X. C., and Belta, C. (2011). Multi-robot deployment from LTL specifications with reduced communication. In *Proceedings of Conference on Decision and Control and European Control Conference*, pages 4867–4872. IEEE.

[86] Kloetzer, M. and Mahulea, C. (2016). Multi-robot path planning for syntactically co-safe LTL specifications. In *Proceedings of International Workshop on Discrete Event Systems*, pages 452–458. IEEE.

[87] Kobilarov, M. (2012). Cross-entropy motion planning. *The International Journal of Robotics Research*, 31(7):855–871.

[88] Koymans, R. (1990). Specifying real-time properties with metric temporal logic. *Real-time Systems*, 2(4):255–299.

[89] Kress-Gazit, H., Fainekos, G. E., and Pappas, G. J. (2007). Where's waldo? sensor-based temporal logic motion planning. In *Proceedings of International Conference on Robotics and Automation*, pages 3116–3121. IEEE.

[90] Kuffner, J. J., Kagami, S., Nishiwaki, K., Inaba, M., and Inoue, H. (2002). Dynamically-stable motion planning for humanoid robots. *Autonomous Robots*, 12(1):105–118.

[91] Kupferman, O. and Vardi, M. Y. (2001). Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314.

[92] Kushner, H. and Dupuis, P. G. (2013). *Numerical methods for stochastic control problems in continuous time*, volume 24. Springer Science & Business Media.

[93] Kwiatkowska, M., Norman, G., and Parker, D. (2011). Prism 4.0: Verification of probabilistic real-time systems. In *Proceedings of Conference on Computer Aided Verification*, pages 585–591. Springer.

[94] Lahijanian, M., Andersson, S., and Belta, C. (2011). Control of Markov decision processes from PCTL specifications. In *Proceedings of American Control Conference*, pages 311–316. IEEE.

[95] Lahijanian, M., Andersson, S. B., and Belta, C. (2012). Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Transactions on Robotics*, 28(2):396–409.

[96] Lampariello, R., Nguyen-Tuong, D., Castellini, C., Hirzinger, G., and Peters, J. (2011). Trajectory planning for optimal robot catching in real-time. In *Proceedings of International Conference on Robotics and Automation*, pages 3719–3726. IEEE.

[97] Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers.

[98] Latombe, J.-C. (1999). Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *The International Journal of Robotics Research*, 18(11):1119–1128.

[99] Latvala, T. (2003). Efficient model checking of safety properties. In *Proceedings of International SPIN Workshop on Model Checking of Software*, pages 74–88. Springer.

[100] LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Department, Iowa State University.

[101] LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press.

[102] LaValle, S. M. and Kuffner Jr, J. J. (2001). Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400.

[103] Leahy, K., Jones, A., Schwager, M., and Belta, C. (2015). Distributed information gathering policies under temporal logic constraints. In *Proceedings of Conference on Decision and Control*, pages 6803–6808. IEEE.

[104] Lee, D.-T. and Schachter, B. J. (1980). Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242.

[105] Lee, M. C. and Park, M. G. (2003). Artificial potential field based path planning for mobile robots using a virtual obstacle concept. In *Proceedings of Conference on Advanced Intelligent Mechatronics*, volume 2, pages 735–740. IEEE.

[106] Leven, P. and Hutchinson, S. (2000). Toward real-time path planning in dynamic environments. In *Proceedings of Workshop on the Algorithmic Foundations of Robotics*.

[107] Li, Y. and Bekris, K. E. (2011). Learning approximate cost-to-go metrics to improve sampling-based motion planning. In *Proceedings of International Conference on Robotics and Automation*, pages 4196–4201. IEEE.

[108] Li, Y., Littlefield, Z., and Bekris, K. E. (2016). Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564.

[109] Lions, J.-L., Lübeck, L., Fauquembergue, J.-L., Kahn, G., Kubbat, W., Levedag, S., Mazzini, L., Merle, D., and O'Halloran, C. (1996). Ariane 5 flight 501 failure report by the inquiry board.

[110] Littman, M. L., Dean, T. L., and Kaelbling, L. P. (1995). On the complexity of solving Markov decision problems. In *Proceedings of Conference on Uncertainty in Artificial Intelligence*, pages 394–402. Morgan Kaufmann Publishers Inc.

[111] Liu, J. (2017). Robust abstractions for control synthesis: Completeness via robustness for linear-time properties. In *Proceedings of International Conference on Hybrid Systems: Computation and Control*, pages 101–110. ACM.

[112] Liu, J. and Ozay, N. (2014). Abstraction, discretization, and robustness in temporal logic control of dynamical systems. In *Proceedings of International Conference on Hybrid Systems: Computation and Control*, pages 293–302. ACM.

[113] Liu, J. and Prabhakar, P. (2014). Switching control of dynamical systems from metric temporal logic specifications. In *Proceedings of International Conference on Robotics and Automation*, pages 5333–5338. IEEE.

[114] Livingston, S. C., Wolff, E. M., and Murray, R. M. (2015). Cross-entropy temporal logic motion planning. In *Proceedings of International Conference on Hybrid Systems: Computation and Control*, pages 269–278. ACM.

[115] Lozano-Pérez, T. and Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570.

[116] Luders, B., Kothari, M., and How, J. P. (2010). Chance constrained RRT for probabilistic robustness to environmental uncertainty. In *Proceedings of AIAA guidance, Navigation, and Control Conference*, volume 36, pages 856–863.

[117] Luna, R., Lahijanian, M., Moll, M., and Kavraki, L. E. (2014). Fast stochastic motion planning with optimality guarantees using local policy reconfiguration. In *Proceedings of International Conference on Robotics and Automation*, pages 3013–3019. IEEE.

[118] Luna, R., Lahijanian, M., Moll, M., and Kavraki, L. E. (2015). Asymptotically optimal stochastic motion planning with temporal goals. In *Proceedings of Workshop on the Algorithmic Foundations of Robotics*, pages 335–352. Springer.

[119] Magnenat, S., Waibel, M., and Beyeler, A. (2011). Enki: The fast 2d robot simulator. *URL http://home. gna. org/enki*.

[120] Maler, O. and Nickovic, D. (2004). Monitoring temporal properties of continuous signals. In *Proceedings of International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems Formal Modeling and Analysis of Timed Systems*, volume 3253, pages 152–166. Springer.

[121] Mickelin, O., Ozay, N., and Murray, R. M. (2014). Synthesis of correct-by-construction control protocols for hybrid systems using partial state information. In *Proceedings of American Control Conference*, pages 2305–2311. IEEE.

[122] Mirtich, B. (1998). V-clip: Fast and robust polyhedral collision detection. *ACM Transactions On Graphics*, 17(3):177–208.

[123] Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., and Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In *Proceedings of Conference on Autonomous Robot Systems and Competitions*, volume 1, pages 59–65. IPCB.

[124] Montana, F. J., Liu, J., and Dodd, T. J. (2016). Sampling-based stochastic optimal control with metric interval temporal logic specifications. In *Proceedings of Conference on Control Applications*, pages 767–773. IEEE.

[125] Montana, F. J., Liu, J., and Dodd, T. J. (2017a). Sampling-based path planning for multi-robot systems with co-safe linear temporal logic specifications. In *Proceedings of Critical Systems: Formal Methods and Automated Verification*, pages 150–164. Springer.

[126] Montana, F. J., Liu, J., and Dodd, T. J. (2017b). Sampling-based reactive motion planning with temporal logic constraints and imperfect state information. In *Proceedings of Critical Systems: Formal Methods and Automated Verification*, pages 134–149. Springer.

[127] Nikou, A., Tumova, J., and Dimarogonas, D. V. (2016). Cooperative task planning of multi-agent systems under timed temporal specifications. In *Proceedings of American Control Conference*, pages 7104–7109. IEEE.

[128] Ó'Dúnlaing, C. and Yap, C. K. (1985). A "retraction" method for planning the motion of a disc. *Journal of Algorithms*, 6(1):104–111.

[129] Ono, M., Williams, B. C., and Blackmore, L. (2013). Probabilistic planning for continuous dynamic systems under bounded risk. *Journal of Artificial Intelligence Research*, 46:511–577.

[130] Ozay, N., Liu, J., Prabhakar, P., and Murray, R. M. (2013). Computing augmented finite transition systems to synthesize switching protocols for polynomial switched systems. In *Proceedings of American Control Conference*, pages 6237–6244. IEEE.

[131] Palmieri, L. and Arras, K. O. (2015). Distance metric learning for RRT-based motion planning with constant-time inference. In *Proceedings of Conference on Robotics and Automation*, pages 637–643. IEEE.

[132] Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450.

[133] Papusha, I., Fu, J., Topcu, U., and Murray, R. M. (2016). Automata theory meets approximate dynamic programming: Optimal control with temporal logic constraints. In *Proceedings of Conference on Decision and Control*, pages 434–440. IEEE.

[134] Perez, A., Platt, R., Konidaris, G., Kaelbling, L., and Lozano-Perez, T. (2012). LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *Proceedings of Conference on Robotics and Automation*, pages 2537–2542. IEEE.

[135] Pineau, J., Gordon, G., Thrun, S., et al. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of International Joint Conference on Artificial Intelligence*, volume 3, pages 1025–1032.

[136] Plaku, E. (2012a). Path planning with probabilistic roadmaps and co-safe linear temporal logic. In *Proceedings of Conference on Intelligent Robots and Systems*, pages 2269–2275. IEEE.

[137] Plaku, E. (2012b). Planning in discrete and continuous spaces: From LTL tasks to robot motions. In *Proceedings of Conference Towards Autonomous Robotic Systems*, pages 331–342. Springer.

[138] Plaku, E., Kavraki, L. E., and Vardi, M. Y. (2010). Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Transactions on Robotics*, 26(3):469–482.

[139] Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of Symposium on Foundations of Computer Science*, pages 46–57. IEEE.

[140] Pola, G., Girard, A., and Tabuada, P. (2008). Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 44(10):2508–2516.

[141] Pontryagin, L. S. (1962). *Mathematical theory of optimal processes*. Interscience.

[142] Pratt, V. R. (1981). A decidable mu-calculus: Preliminary report. In *Proceedings of Symposium on Foundations of Computer Science*, pages 421–427. IEEE.

[143] Prentice, S. and Roy, N. (2009). The belief roadmap: Efficient planning in belief space by factoring the covariance. *The International Journal of Robotics Research*, 28(11-12):1448–1465.

[144] Raman, V., Donzé, A., Maasoumy, M., Murray, R. M., Sangiovanni-Vincentelli, A., and Seshia, S. A. (2014). Model predictive control with signal temporal logic specifications. In *Proceedings of Conference on Decision and Control*, pages 81–87. IEEE.

[145] Raman, V., Donzé, A., Sadigh, D., Murray, R. M., and Seshia, S. A. (2015). Reactive synthesis from signal temporal logic specifications. In *Proceedings of International Conference on Hybrid Systems: Computation and Control*, pages 239–248. ACM.

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

[147] Richards, A. and How, J. P. (2002). Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of American Control Conference*, volume 3, pages 1936–1941. IEEE.

[148] Rickert, M., Sieverling, A., and Brock, O. (2014). Balancing exploration and exploitation in sampling-based motion planning. *IEEE Transactions on Robotics*, 30(6):1305–1317.

[149] Rimon, E. and Koditschek, D. E. (1992). Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518.

[150] Rubinstein, R. Y. and Kroese, D. P. (2004). *The Cross-Entropy Method*. Springer.

[151] Saha, S. and Julius, A. A. (2016). An MILP approach for real-time optimal controller synthesis with metric temporal logic specifications. In *Proceedings of American Control Conference*, pages 1105–1110. IEEE.

[152] Sastry, S. (2013). *Nonlinear systems: analysis, stability, and control*, volume 10. Springer Science & Business Media.

[153] Schouwenaars, T., De Moor, B., Feron, E., and How, J. (2001). Mixed integer programming for multi-vehicle path planning. In *Proceedings of European Control Conference*, pages 2603–2608. IEEE.

[154] Schulman, J., Ho, J., Lee, A. X., Awwal, I., Bradlow, H., and Abbeel, P. (2013). Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, pages 1–10. Citeseer.

[155] Sharan, R. and Burdick, J. (2014). Finite state control of POMDPs with LTL specifications. In *Proceedings of American Control Conference*, pages 501–508. IEEE.

[156] Shewchuk, J. R. (1996). Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Applied Computational Geometry Towards Geometric Engineering*, pages 203–222. Springer.

[157] Smith, S. L., Tůmová, J., Belta, C., and Rus, D. (2011). Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research*, 30(14):1695–1708.

[158] Solovey, K., Salzman, O., and Halperin, D. (2016). Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning. *The International Journal of Robotics Research*, 35(5):501–513.

[159] Şucan, I. A. and Kavraki, L. E. (2012). A sampling-based tree planner for systems with complex dynamics. *IEEE Transactions on Robotics*, 28(1):116–131.

[160] Sun, Z., Hsu, D., Jiang, T., Kurniawati, H., and Reif, J. H. (2005). Narrow passage sampling for probabilistic roadmap planning. *IEEE Transactions on Robotics*, 21(6):1105–1115.

[161] Švestka, P. and Overmars, M. H. (1998). Coordinated path planning for multiple robots. *Robotics and Autonomous Systems*, 23(3):125–152.

[162] Svoreňová, M., Černá, I., and Belta, C. (2013a). Optimal control of MDPs with temporal logic constraints. In *Proceedings of Conference on Decision and Control*, pages 3938–3943. IEEE.

[163] Svoreňová, M., Černá, I., and Belta, C. (2013b). Optimal receding horizon control for finite deterministic systems with temporal logic constraints. In *Proceedings of American Control Conference*, pages 4399–4404. IEEE.

[164] Svoreňová, M., Chmelík, M., Leahy, K., Eniser, H. F., Chatterjee, K., Černá, I., and Belta, C. (2015). Temporal logic motion planning using POMDPs with parity objectives: case study paper. In *Proceedings of International Conference on Hybrid Systems: Computation and Control*, pages 233–238. ACM.

[165] Tabuada, P. and Pappas, G. J. (2003). From discrete specifications to hybrid control. In *Proceedings of Conference on Decision and Control*, volume 4, pages 3366–3371. IEEE.

[166] Tabuada, P. and Pappas, G. J. (2006). Linear time logic control of discrete-time linear systems. *IEEE Transactions on Automatic Control*.

[167] Tedrake, R. (2010). LQR-trees: Feedback motion planning on sparse randomized trees. *International Journal of Robotics Research*.

[168] Thomas, W. et al. (2002). *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media.

[169] Tumova, J. and Dimarogonas, D. V. (2014). A receding horizon approach to multi-agent planning from local LTL specifications. In *Proceedings of American Control Conference*, pages 1775–1780. IEEE.

[170] Tumova, J. and Dimarogonas, D. V. (2015). Decomposition of multi-agent planning under distributed motion and task LTL specifications. In *Proceedings of Conference on Decision and Control*, pages 7448–7453. IEEE.

[171] Ulusoy, A., Smith, S. L., Ding, X. C., Belta, C., and Rus, D. (2013). Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research*, 32(8):889–911.

[172] Vadakkepat, P., Tan, K. C., and Ming-Liang, W. (2000). Evolutionary artificial potential fields and their application in real time robot path planning. In *Proceedings of Congress on Evolutionary Computation*, volume 1, pages 256–263. IEEE.

[173] Van Den Berg, J., Abbeel, P., and Goldberg, K. (2011). LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7):895–913.

[174] Van Den Berg, J. and Overmars, M. (2007). Kinodynamic motion planning on roadmaps in dynamic environments. In *Proceedings of International Conference on Intelligent Robots and Systems*, pages 4253–4258. IEEE.

[175] Van Den Berg, J., Snoeyink, J., Lin, M. C., and Manocha, D. (2009). Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In *Robotics: Science and systems*, volume 2, pages 2–3.

[176] Van Den Berg, J. P. and Overmars, M. H. (2005). Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. *The International Journal of Robotics Research*, 24(12):1055–1071.

[177] Van Den Bergen, G. (1999). A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7–25.

[178] Vardi, M. Y. and Wolper, P. (1986). An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331. IEEE Computer Society.

[179] Varricchio, V., Chaudhari, P., and Frazzoli, E. (2014). Sampling-based algorithms for optimal motion planning using process algebra specifications. In *Proceedings of Conference on Robotics and Automation*, pages 5326–5332. IEEE.

[180] Vasile, C. I. and Belta, C. (2013). Sampling-based temporal logic path planning. In *Proceedings of International Conference on Intelligent Robots and Systems*, pages 4817–4822. IEEE.

[181] Vasile, C. I. and Belta, C. (2014). Reactive sampling-based temporal logic path planning. In *Proceedings of International Conference on Robotics and Automation*, pages 4310–4315. IEEE.

[182] Vasile, C.-I., Leahy, K., Cristofalo, E., Jones, A., Schwager, M., and Belta, C. (2016). Control in belief space with temporal logic specifications. In *Proceedings of Conference on Decision and Control*, pages 7419–7424. IEEE.

[183] Webb, D. J. and van den Berg, J. (2013). Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *Proceedings of Conference on Robotics and Automation*, pages 5054–5061. IEEE.

[184] Wolff, E. M. and Murray, R. M. (2016). Optimal control of nonlinear systems with temporal logic specifications. In *Robotics Research*, pages 21–37. Springer.

[185] Wolff, E. M., Topcu, U., and Murray, R. M. (2012). Optimal control with weighted average costs and temporal logic specifications. *Proceedings of Robotics: Science and Systems*.

[186] Wolff, E. M., Topcu, U., and Murray, R. M. (2013a). Automaton-guided controller synthesis for nonlinear systems with temporal logic. In *Proceedings of International Conference on Intelligent Robots and Systems*, pages 4332–4339. IEEE.

[187] Wolff, E. M., Topcu, U., and Murray, R. M. (2013b). Optimal control of non-deterministic systems for a computationally efficient fragment of temporal logic. In *Proceedings of Conference on Decision and Control*, pages 3197–3204. IEEE.

[188] Wolff, E. M., Topcu, U., and Murray, R. M. (2014). Optimization-based trajectory generation with linear temporal logic specifications. In *Proceedings of International Conference on Robotics and Automation*, pages 5319–5325. IEEE.

[189] Wong, K. W. and Kress-Gazit, H. (2015). Let's talk: Autonomous conflict resolution for robots carrying out individual high-level tasks in a shared workspace. In *Proceedings of International Conference on Robotics and Automation*, pages 339–345. IEEE.

[190] Wongpiromsarn, T. and Frazzoli, E. (2012). Control of probabilistic systems under dynamic, partially known environments with temporal logic specifications. In *Proceedings of Conference on Decision and Control*, pages 7644–7651. IEEE.

[191] Wongpiromsarn, T., Karaman, S., and Frazzoli, E. (2011). Synthesis of provably correct controllers for autonomous vehicles in urban environments. In *Proceedings of Conference on Intelligent Transportation Systems*, pages 1168–1173. IEEE.

[192] Wongpiromsarn, T., Topcu, U., and Murray, R. M. (2012). Receding horizon temporal logic planning. *IEEE Transactions on Automatic Control*, 57(11):2817–2830.

[193] Wongpiromsarn, T., Topcu, U., and Murray, R. M. (2013). Synthesis of control protocols for autonomous systems. *Unmanned Systems*, 1(01):21–39.

[194] Zamani, M., Tkachev, I., and Abate, A. (2017). Towards scalable synthesis of stochastic control systems. *Discrete Event Dynamic Systems*, 27(2):341–369.

[195] Zefran, M. and Kumar, V. (1997). A variational calculus framework for motion planning. In *Proc. IEEE International Conference on Robotics and Automation*, pages 415–420.

[196] Zhang, X., Wu, B., and Lin, H. (2015). Learning based supervisor synthesis of POMDP for PCTL specifications. In *Proceedings of Conference on Decision and Control*, pages 7470–7475. IEEE.

[197] Zhang, Z. and Cowlagi, R. V. (2016). Motion-planning with global temporal logic specifications for multiple nonholonomic robotic vehicles. In *Proceedings of American Control Conference*, pages 7098–7103. IEEE.

[198] Zhou, Y., Maity, D., and Baras, J. S. (2016). Timed automata approach for motion planning using metric interval temporal logic. In *Proceedings of European Control Conference*, pages 690–695. IEEE.

[199] Zhu, Q., Yan, Y., and Xing, Z. (2006). Robot path planning based on artificial potential field approach with simulated annealing. In *Proceedings of Conference on Intelligent Systems Design and Applications*, volume 2, pages 622–627. IEEE.

[200] Zucker, M., Ratliff, N., Dragan, A. D., Pivtoraiko, M., Klingensmith, M., Dellin, C. M., Bagnell, J. A., and Srinivasa, S. S. (2013). Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193.