Title
Robot Planning with Constrained Markov Decision Processes

Author
Feyzabadi, Seyedshams

Publication Date
2017

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, MERCED

Robot Planning with Constrained Markov Decision Processes

by

Seyedshams Feyzabadi

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Electrical Engineering and Computer Science

Committee in charge:
Professor Stefano Carpin, Chair
Professor Marcelo Kallmann
Professor YangQuan Chen

Summer 2017

The dissertation of Seyedshams Feyzabadi is approved and it is acceptable in quality and form for publication on microfilm and electronically.

Stefano Carpin, Chair	Date
Marcelo Kallmann	Date
YangQuan Chen	Date

University of California, Merced

For a while in childhood we attended a master
for a while we were happy with our own mastery
at the end of the story see what happened to us
like water we entered, and like the wind we departed
"Omar Khayyam"

Dedicated to my father

Acknowledgments

Once I started going to school at the age of seven, I had never thought that the journey would take so long. Now, it is the time to write the last page of it that makes me think of the entire path with all of its ups and downs. It all started in my little home town, Sabzevar. After twelve years, it took me to the crazy capital city of Tehran. I have lots of great memories of living and studying in Tehran with wonderful friends. Then the roller coaster made a turn toward Germany. There I started my master's at Jacobs University Bremen where nobody speaks German. It was an awesome time with many international friends who are still my close friends. Finally, we landed in a tiny village in the united states which is called Merced. It is where the end the story begins.

During my PhD at UC Merced, I have received tons of supports from my advisor Stefano. He helped me in multiple ways and deserves better students than me. I never forget one of his sentences, "Numbers don't lie". He spent hours to teach me different concepts in robotics. He financially supported my research by different means. I would like to highly appreciate his helps during these years. One of the greatest resources for my financial support was provided by National Institute of Standards and Technology (NIST) under cooperative agreement 70NANB12H143. Therefore, I would like to formally thank NIST for their generous support.

Next, I need to thank my committee members, Dr Chen and Dr Kallmann for their valuable comments and suggestions. It is also my pleasure to thank Dr Chen again for letting my wife work in his lab. He fully supported her for the last few years to have a better life in Merced.

In addition, I very much appreciate the supports from school of engineering by offering multiple fellowships in forms of Bobcat, travel or EECS awards. They gave us the possibility to focus on the research rather than looking out for funding resources.

It should also be noted that I thank my colleagues in the lab. Starting with Shuo Liu who I spent the most number of hours with him to solve different problems. We shared a room together in Seattle. Then I need to thank Jose Luis Susa, Andres Torres Garcia and Thomas Thayer for being wonderful people and helping me in various ways.

The support of my family is appreciated. My mother taught me alphabet when I was very little, and spent numerous number of hours to teach me millions of things. She patiently answered all of my stupid questions such as "Which one is stronger, a lion or a tiger?". Furthermore, I thank my father who just left us for everything I have. Similarly, my brother and sisters assisting me since I was a little kid until now.

Last but not least, I would like to extremely thank my wife, Niloufar, for being very supportive, helpful and patient all these years. It is not easy to have a PhD life, and it is harder to support a PhD student. She has gone through several hard moments during these years that I admire it, and thank her for all of that. Furthermore, she has been a great mother since the birth of our little angel Dorina. I will express my gratitude to Niloufar on behalf of Dornia too. I hope to make a better life for her after graduation as she deserves it.

Curriculum Vitae

Education

- **Ph.D.** Electrical Engineering and Computer Science, University of California, Merced, CA, USA, Expected Aug. 2017
  Ph.D Thesis: Robot Planning with Constrained Markov Decision Processes
- **M.Sc.** Computer Science (Smart Systems), Jacobs University Bremen, Bremen, Germany, Sep. 2010
  Master Thesis: GPU-accelerated SLAM 6D
- **B.Sc.** Computer Engineering (Software), Iran University of Science and Technology (IUST), Tehran, Iran, Dec. 2007
  Bachelor Project: Implementation of Grid Databases Using Globus

Experiences

- **Robotics Software Engineer**, Intuitive Surgical, Sunnyvale, CA, USA May 2016 to date
- **Research Contractor**, Intuitive Surgical, Sunnyvale, CA, USA Sep. 2015 to May 2016
- **Research Intern**, Intuitive Surgical, Sunnyvale, CA, USA May 2015 to Aug. 2015
- **Research Assistant**, University of California, Merced, CA, USA Feb. 2013 to Aug. 2016
- **Junior Researcher**, German Research Center of Artificial Intelligence (DFKI), Bremen, Germany Dec. 2010 to Feb. 2013
- **Research Assistant**, DFKI, Bremen, Germany Jun. 2009 to Oct. 2010
- **Research Assistant**, Jacobs University Bremen, Bremen, Germany Dec. 2008 to Jun. 2009
- **Robocup Manager**, Islamshahr Azad University, Tehran, Iran Nov. 2007 to Jun. 2008
- **Software Engineer**, Douran Software Technologies, Tehran, Iran Feb. 2006 to Feb. 2008
- **Site Administrator**, Hakimieh dormitory, Iran University of Science and Technology, Tehran, Iran Nov. 2002 to Jun. 2003

Academic Honors

- EECS BobCat Award fellowship from Department of Electrical Engineering and Computer Science at UC Merced, 2016
- EECS BobCat Award Award fellowship from Department of Electrical Engineering and Computer Science at UC Merced, 2016
- Dean's travel fellowship from School of Engineering at UC Merced, 2015

- BobCat Fellowship from Graduate School of University of Califonia, 2014
- EECS Fellowship from Department of Electrical Engineering and Computer Science at UC Merced, 2013
- Full graduate scholarship from University of California (Feb. 2013 to Dec. 2016)
- Full graduate scholarship from Jacobs University Bremen (Sep. 2008 to Aug. 2010)
- 1st place of RoboCupRescue Simulation league, JapanOpen 2008, Numazu, Japan
- 3rd place of RoboCupRescue Simulation league, IranOpen 2007, Tehran, Iran
- 3rd place of RoboCupRescue Simulation league, world cup 2006, Bremen, Germany
- 2nd place of RoboCupRescue Simulation league, world cup 2005, Osaka, Japan
- 1st place of RoboCupRescue Simulation league, German Open 2005, Paderborn
- 2nd place of RoboCupSoccer Simulation league, US Open 2005, Georgia, Atlanta
- 3rd place of RoboCupRescue Simulation league, world cup 2004, Lisbon, Portugal
- Top ranked undergraduate researcher at IUST in academic years 2003-04, 2004-05 and 2005-2006

Publications

Journal Publications

- Feyzabadi, S, and Carpin, S.: Planning Using Hierarchical Constrained Markov Decision Processes, Autonomous Robots, pp. 1,19, 2017.

- Feyzabadi, S., Straube, S., Folgheraiter, M., Kirchner, E., Kim, S., and Albiez, J.: Human Force Discrimination during Active Arm Motion for Force Feedback Design, Haptics, IEEE Transactions on , vol.6, no.3, pp.309,319, July-Sept. 2013

Peer-Reviewed Conference Proceedings

- Feyzabadi, S, and Carpin, S.: A Toolbox for Multi Objective Planning in Non-Deterministic Environments with Simulation Validation, In Proceedings of IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR) San Francisco, CA, 2016

- Feyzabadi, S., and Carpin, S.: Multi Objective Planning with Multiple High-level Task Specifications, In Proceedings of IEEE International Conference on Robotics and Automation (ICRA) Stockholm, Sweden, 2016

- Feyzabadi, S., and Carpin, S.: HCMDP: a Hierarchical Solution to Constrained Markov Decision Processes, In Proceedings of IEEE International Conference on Robotics and Automation (ICRA) Seattle, WA, 2015

- Feyzabadi, S., and Carpin, S.: Motion Planning with Safety Constraints and High-Level Task Specifications, In Proceedings of IEEE International Conference on Robotics and Automation (ICRA) Workshops Seattle, WA, 2015

- Feyzabadi, S., and Carpin, S.: Risk-aware Path Planning Using Hierarchical Constrained Markov Decision Processes, In Proceedings of IEEE International Conference on Automation Science and Engineering (CASE) Taipei, Taiwan, 2014

- Feyzabadi, S., and Carpin, S.: Knowledge and Data Representation for Motion Planning in Dynamic Environments, In Proceedings of the International Conference on Robot Intelligence Technology and Applications (RITA) Denver, USA, 2013

- Nüchter, A., Feyzabadi, S., Qiu, D. and May, S.: SLAM a la carte GPGPU for Globally Consistent Scan Matching. In Proceedings of the 4th European Conference on Mobile Robots (ECMR 11), Örebro, Sweden, September 2011

- Baumann, P., Feyzabadi, S. and Jucovschi, C.: Putting Pixels in Place: A Storage Layout Language for Scientific Data, ICDM Workshops 2010, Pp. 194-203

- Feyzabadi, S. and Schönwälder, J.: Identifying TCP Congestion Control Algorithms Using Active Probing, Poster, PAM Conference, Zurich, Switzerland, 2010

- Bülow, H., Birk, A. and Feyzabadi, S.: Creating Photo Maps with an Aerial Vehicle in USARsim, RoboCup 2009, Robot WorldCup XIII, Lecture Notes in Artificial Intelligence (LNAI), Springer

- Hamraz, S., and Feyzabadi, S.: General-Purpose Learning Machine Using K-Nearest Neighbors Algorithm. RoboCup 2005 (LNAI) 2006

- Hamraz, S., and Feyzabadi, S.: IUST Robocup Rescue Simulation Agent Competition team description, In Proceedings of RoboCup 2006

- Hamraz, S., and Feyzabadi, S.: Caspian Robocup Rescue Simulation Agent Competition team description, In Proceedings of RoboCup 2005

- Sedaghati, M., Gholami, N., Rafiee, E., Zadeh, O., Nezhad, L., Hamraz, H., Feyzabadi, S., Khamooshi, S. and Kangavari, M.: Caspian 2004 Rescue Simulation Team Description, In Proceedings of RoboCup 2004

Contents

Abstract

Robotic technologies have advanced significantly that improved capabilities of robots. Such robots operate in complicated environments and are exposed to multiple resources of uncertainties. The uncertainties causes robots actions to be non-deterministic. Robot planning in non-deterministic environments is a challenging problem that has been extensively discussed in the literature. In this dissertation, we tackle this class of problems and are more particularly interested in finding an optimal solution while the robot faces several constraints. To do so, we leverage Constrained Markov Decision Processes (CMDPs) which are extensions to Markov Decision Processes (MDPs) by supporting multiple costs and constraints. Despite all the capabilities of CMDPs, they are not very popular in robot planning. One of our goals in this work is to show that CMDPs can also be used in robot planning.

In the first part of this dissertation, we focus on optimizing CMDPs to solve large problems in a timely manner. Therefore, we propose a hierarchical approach that significantly reduces the computational time of solving a CMDP instance while preserving the existence of a valid solution. In other words, the Hierarchical CMDP (HCMDP) guarantees to find a valid solution for a specific problem if the non-hierarchical CMDP is able to find one. Although, the experimental evaluation shows that the HCMDP and the non-hierarchical CMDP generate comparable results, we do not provide any guarantees in terms of optimality.

In the second part, we aim for more complicated constraints represented as tasks. Tasks are usually specified by Linear Temporal Logic (LTL) properties and determine a desired temporal sequence of states to be visited by the robot. For instance, an autonomous forklift may be tasked to go to a pick-up station, load an object, drive toward a delivery point and drop it off. As seen the order of states is critical. Thus, we propose a planner that finds a plan to satisfy multiple tasks with given probabilities while having various constraints on its cost functions. The proposed solver utilizes the theory of LTL properties to define tasks, and the theory of CMDPs to find an optimal solution. We also present a special form of product operation between LTL properties and CMDPs that is repeatable. This repeatability lets us apply the product operation several times to take all of the tasks into account. The proposed approach is extensively tested in Matlab, robot simulation and on a real robot.

This solver runs the product operation many times which results in increasing number of states. Therefore, it is crucial to reduce the number of states in order to have a faster solver. In part three of this thesis, we aim for optimizing the solver in part two. We propose two improvements. The first improvement considers the order of product operations. Although the product operation is commutative and the order of operations does not influence the final result, it affects the computational time. Thus, we present an algorithm to find the best order of operations. The second improvement runs a pruning algorithm to reduce the number of states by removing the states that play little or no role in the final product. As opposing to the first improvement, it may change the final solution. However, we analyze different cases that may appear and show the effects.

Chapter 1

Introduction

Starting from the first general purpose mobile robot[1], *Shakey*, in 1966, robots have significantly changed. While early robots were primarily designed to showcase some future functionalities, today's robots are truly capable and functional at the time being. Some advanced robots such as the Mars rover *Curiosity* are equipped with numerous sensors and actuators together with extraordinary computational power to perform very complex missions. Such robots are able to accomplish difficult tasks involving accurate perception, localization, planning, etc. while operating in environments with significant uncertainties.

One of the most fundamental requirements to operate a robot in an environment is to convert a human specified task into a sequence of low-level control inputs, representing how to move. The terms of *motion planning*, *trajectory planning* or *mission planning* usually address these type of problems. However, the term *planning* refers to a wider range of algorithms by considering any form of automation of any mechanical system equipped with sensors, actuators and computation capabilities [111].

One major question that arises is *what is a plan?* The answer to this question heavily depends on the application. In the context of robotics, we consider a plan to be a set of rules computed by a planner or decision maker which imposes a certain behavior for the system [111]. A plan is defined as a sequence of actions which is either computed by a machine or calculated by a human. If the plan is computed by a machine, the algorithm is referred to as the planner.

Planners can be categorized into two classes; online and offline planners. An online planner checks the current state of the system and creates a plan to reach the current goal at the present time. Therefore, online planners are required to compute a new plan (replan) every time that the system evolves and changes its state. On the contrary, offline planners calculate a plan before the system starts to evolve, and then generate a list of actions for every possible state of the system. At execution time, a robot determines the state and performs the pre-computed action at any state. In general, online planners are preferred when the environment is dynamic and the robot requires immediate change of the plan, while offline planners are chosen when the environment is static. However, in a real scenario, a mixture of both planners is more popular. Such algorithms work on two layers. On the top layer, they run an offline planner which finds optimal solutions without

[1] `https://www.sri.com` Accessed on 06/12/2017

considering any dynamical behavior in the environment. The online decision makers are applied on the lower layer where the dynamical behavior is visible to the robot. Thus, the real time planner takes care of small dynamic changes around the robot while receiving intermediate goals from the offline planner. Throughout this dissertation, we also leverage hybrid planners, and utilize them in most of our experiments.

Imagine a Mars rover on a mission. A robot is sent to a planet which is either unknown or partially known. The robot is assumed to perform long term missions, and survive on its own without any external help. The amount of noise on the sensors is non-negligible, and the surface of the planet is not convenient for any mobile robot, then how can the Mars rover work. Moreover, the robot is sent there to explore the unknowns and accomplish numerous complicated missions.

Here is an example mission. The robot is asked to drive and read some sensory data from multiple points on the planet. The robot runs on a battery which has a limited life and if the robot drives too fast, the battery life shrinks. On the other hand, if it slows down to save some energy while the solar cells recharge the battery, it increases the total time of the mission which might have influences on validity of data. Similarly, the robot might be required to perform few sub tasks while driving to its primary goal such as picking up an object on the way and dropping that off at the destination.

As seen in the aforementioned example, a planner needs to take multiple factors into account in order to complete a mission. It is almost impossible to optimize all given factors for any type of planning problems at once. Therefore, most of the planners select one or few of the factors that are more critical for the operations and optimize them only. Meanwhile, good planners keep an eye on non-critical parameters as well to avoid complications [44]. Tailoring this concept to our former example, such planners may optimize the time to reach the goal while applying an upper bound on the battery consumption which prevents unexpected failures.

Uncertainties also play a key role in this scenario. If the sensors are so noisy that the robot is not able to localize itself on a given map, planning will be extremely challenging. Similarly, uneven or slippery surfaces reduce maneuverability of the robot which is another form of uncertainty for the planner. In such situation, it is evident that the planner has to account for the non-determinism from these uncertainties to propose an optimal plan. Otherwise, numerous replanning steps are required to complete a simple task.

Throughout this dissertation, we only focus on planning under uncertainties where the outcome of an action is not fully predictable, but a probabilistic distribution of that is known.

We start with risk-aware planners in chapter 4. The necessity of having such planners lies on the increased interactions of human and robots. Nowadays, robots perform a wide variety of tasks from mopping floors to doing precise and complicated surgeries. Therefore, interactions between robots and humans are unavoidable. The safety of such interactions is the most important factor for the planners which cannot be compromised in any sense. According to the reports from department of labor in the united states, 20 fatalities were caused by robots since the year 2000[2]. Therefore, we chose the risk to be the primary factor for our planner. However, we did not aim at defining risk metrics where several books and standard documents have already discussed that topic from different aspects.

[2] `https://www.osha.gov/` Accessed on 06/12/2017

Although it is not a novel idea to consider risk as the primary objective, as in [122], we present a new approach and consider multiple factors. We focus on minimizing the total risk along the path while having a bound on total path length. Moreover, our approach is flexible and can be extended to consider multiple additional factors such as limiting energy consumption, time or any similar metric. We present a planner which can find such plans in a reasonable time with limited computational resources. The proposed planner is not tied or limited to risk-aware planning only. It can be applied to any similar scenario where there is a primary objective to be minimized or maximized and a set of additional factors to be bounded.

Next in Chapter 5, we introduce robot tasks. For a human being, tasks have different meanings and multiple approaches exist to accomplish a task. Imagine a simple task of closing a door. It can be performed in multiple ways by different people. It could be done with right or left hand. Some people may prefer to lock the door, too. The route to approach the door may also be different. Robots are not that flexible and smart, and they require precise definition of every aspect. The definition of a task should then contain all the details. It also needs to consider the capabilities of the robot which is assigned to that task. Sometimes, a large task has to be broken down into smaller sub-tasks to be performed.

On the other hand, tasks have to be universally defined which means a unified language has to be used to sketch any form of tasks, regardless of the application or type of robots. Therefore, we propose a planner in Chapter 5 which utilizes the standard task definition methodologies and includes that in the planning phase to extract plans that satisfy a set of predefined tasks. The planner is capable of solving a problem with the following criteria:

- The robot operates in non-deterministic environments.
- The planner optimizes over one primary parameter (also known as *primary cost function*, explained in 3.3.2).
- The planner can include several additional factors to be bounded (also known as *additional cost functions*, explained in 3.3.2).
- It considers multiple tasks to be satisfied (expressed as LTL properties, described in 3.2.2).
- Every task is satisfied with a target probability (explained in 3.4).

The planner can solve general purpose problems and easily be used in completely different scenarios as long as the new problem fits in the above criteria. As shown in the experiments section of the same chapter, we have already applied this planner into different applications such as autonomous navigation, rapid deployment, factory forklift scenarios and there are still many more unexplored areas. Another area of focus in this dissertation, and more specifically in Chapters 6 and 5 was to provide the optimal plans in a timely manner. To achieve this goal, we applied different types of optimization such as pruning the graph and reordering the commutative operations to reduce the processing times.

1.1 Dissertation Contributions

In this dissertation, we focus on planning mobile robots in environments with existing uncertainties. We leveraged *Constrained Markov Decision Processes (CMDPs)* as the primary methodology to solve such problems. Our contributions can be listed as following:

1. In the first step, we use CMDPs to find an optimal path when multiple constraints exist. Besides that, we drive robots along the path to verify the constraints.

2. One of the major drawbacks of CMDPs is their computational time. Accordingly, our next contribution is to propose a hierarchical solver for CMDPs which reduces the computational requirements while showing comparable results.

3. Even though, hierarchical solvers reduce the computational time, they might eliminate some valid solutions. Therefore, we propose a new type of hierarchical CMDP (HCMDP) which provides guarantees about existence of solutions. In other words, HCMDP guarantees to find a solution, if the non-hierarchical CMDP is able to find one. However, we do not provide any assurance about the optimality of such solutions. However, extensive experimental results on Matlab, robot simulation and real robot showed that the results are comparable to the optimal outcomes by the non-hierarchical CMDP.

4. In the next step, we consider more complex missions or tasks. As commonly used in robotics, we use *Linear Temporal Logic (LTL)* properties to define tasks. Then we included LTL properties in CMDP planners to find optimal solutions for CMDPs while satisfying some tasks as well as some costs and constraints. To do so, we have the following sub-contributions:

   (a) We define *Labeled CMDPs (LCMDPs)* which are the model sustaining our solver. LCMDPs bring labels into the context of CMDPs.

   (b) We introduce *Extended Total Deterministic Finite Automata (DFAs)*. By converting LTL properties into extended-total DFAs, we are able to connect the theory of LCMDPs with LTL properties.

   (c) We define a new product operation between an LCMDP and an extended-total DFA which is iterable. This repeatability plays a critical role in applying the product between multiple LTL properties and an LCMDP. In other words, this addition lets us propose a planner for a mobile robot which can simultaneously take multiple tasks into account.

   (d) We propose a principled way to solve an LCMDP with a linear program which considers all costs, constraints and tasks.

   (e) Finally, we validate our findings on Matlab, robot simulation and on a real robot. All of the experiments confirm our theory by satisfying all the constraints in expectation.

5. Including multiple tasks into a planner requires applying the product operation multiple times. This increases the size of the LCMDP, and may create unsolvable problems. Therefore, we propose a set of optimization steps to reduce the size of the problem and then decrease the computational time.

6. After developing our theory, we implement it in an open source toolbox. It consists of an offline planner which generates the table of rules, and also an online planner to drive the real or simulated robot along the desired path.

Chapter 2

Literature Review

In this chapter we introduce some related research work. In Section 2.1, we discuss discrete planning. Then, in Section 2.2 we briefly describe the state of the art in motion planning and present different approaches to solve motion planning problems including their strengths and weaknesses. Subsequently, in Section 2.3 we provide some insights of task specifications, and how to universally define them. Finally, we present some research works that integrate tasks into motion planning algorithms.

2.1 Discrete Planning

In a discrete planning problem, every distinct situation of the world is referred to as a *state*, and denoted by s. Then *state space*, denoted by S, is defined as the set of all possible states. Discrete planning is the simplest type of planning where the state space is finite or it is countably infinite [111]. Since the real world is not discrete, it is required to convert a continuous world into a discrete environment. This step is referred to as the *abstraction* step [119] which is one of the very important steps in any discrete planning approach.

Another key concept in planning is *actions*, which are denoted by u. Actions can change the state of a system from s to s'. In many cases, the set of viable actions in different states are different. Usually the set of all the actions is represented by U, and the set of feasible actions in a specific state s is referred to as $U(s)$. The set of all actions can be calculated by taking the union of individual action sets as follows:

$$U = \cup_{s \in S} U(s)$$

The *transition function* defines the outcome of actions at every state. If a transition function is indicated by f we have:

$$s' = f(s, u)$$

In discrete planners, we use the terms of *initial* and *goal* states to indicate which state the system starts from and where it must end. A simple example of discrete planning can be finding the shortest path in a 2D environment. A small example is shown in Figure 2.1. Standard graph search algorithms such as A* [78], Dijkstra [131], Breadth-First-Search [131], and others can be used to find the shortest path or exploring all the reachable states.

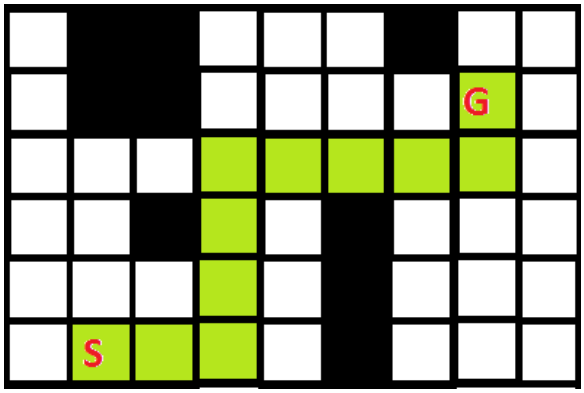


Figure 2.1: An example of discrete planning where S is the initial and G is the goal state. The green path shows a feasible trajectory from the initial state to the goal state. White cells are traversable while black area is blocked.

2.2 Motion Planning

We next consider continuous spaces where the problem is more complicated and classic graph search algorithms cannot directly solve the problem. The key concept in motion planning is a *configuration*. A configuration specifies positions of all the points in a system at a given time. Consequently, *configuration space* refers to the set of all possible configurations. The configuration space is split into two subsets; *free space*, C_{free}, and *obstacle space*, C_{obs}. C_{obs} contains all the configurations where the robot intersects any obstacle. On the contrary, C_{free} represents the set of configurations where the robot does not collide with any obstacle.

The *piano mover's problem* is a classical problem for robot motion planning where the models for a house and a piano are given. The idea is to look for an algorithm which takes these inputs and determines how to move the piano between two configurations without hitting any object [111]. The problem in motion planning is to find a sequence of robot configurations from the starting configuration to the goal configuration where all of these configurations are contained in C_{free} only [111]. An example is shown in Figure 2.2.

Even though nowadays robots have changed significantly in shape, sensing, and other features, the basic problem of driving a robot safely in complex situation has not changed. The problem of motion planning has been widely discussed in the literature, and there are numerous studies to tackle the same problem from different perspectives. There are several classic text books describing motion planning with many details and under different conditions and constraints (See [37, 109, 111]). On the other hand, motion planning algorithms have also been popular in other fields such as, computer animations and computational biology [6, 90, 110].

Among all these works, the classical survey of *gross motion planning* by Hwang and Ahuja, [89], is one of the overview publications in early 1990s. They classified motion planning algorithms in the following ways: the environment can be either *stationary* or *time-varying* which means that the environment does not change in stationary domains but it some objects in the platform might move, appear or disappear in time-varying environments. Motion planning is either *constrained*

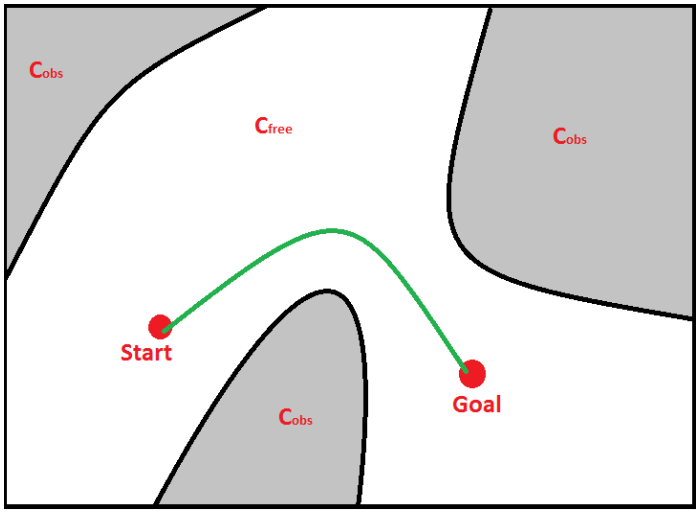


Figure 2.2: An example of motion planning from start to goal by selecting configurations from C_{free} only

or *non-constrained* where the robot motion model has some limitations such as max/min speed in the constrained form, but there is no limits in non-constrained case. Similarly, the motion planning can be *static* or *dynamic* where static refers to finding a plan in offline manner. On the contrary, the motion plan has to be continuously updated in dynamic model. The solvers are either *exact* or *heuristic* which means they either find the exact solution or to find a solution which is good enough using some heuristics. Similarly, solvers can be classified as *global* vs *local* in order to find a plan for the global or a local problem. Throughout this document we focus on *stationary, constrained, static, exact* and *global* solvers.

When uncertainty is added to motion planning, the problem changes, and many extra considerations need to be taken into account. Uncertainty generally exists in the real world. However, to simplify the problem, some planners do not take that into account. As a result, a replanning step is activated as soon as uncertainty is detected. There are two known sources of uncertainties [111]:

- State evolution: If the outcome of an action is not fully predictable. Uncertainties can cause failure of actions or deviation from the trajectories.
- Perception: If the current state is not precisely known, the current state is usually estimated by starting from a known initial state using sensors and the history of previous states/actions.

In the following subsections 2.2.1 and 2.2.2, we present some of the existing approaches to solve motion planning problems in deterministic and non-deterministic situations.

2.2.1 Deterministic Planning

Given an initial and a goal configuration, the simplest planning approach finds a sequence of intermediate configurations that leads the robot to the goal state if it assumes that every configuration

is reachable from its neighboring configurations in the sequence and the initial state is unique. These two assumptions imply that the state of the system is exactly predictable after following any sequence.

Deterministic planning is divided into two main categories of *sampling-based* and *combinatorial* motion planning. The focus of *sampling-based* approaches is to avoid building a complete configuration space, and rely on collision checking tools. *Sampling based* planners are very beneficial when the number of parameters involved in planning is high. On the contrary, *combinatorial* planners emphasize precise solutions by complete geometric modeling of the environments. Therefore, they build a complete representation of the configuration space, and seek an exact solution. As a consequence, the complexity of the problem increases significantly which require great computational resources to solve a moderate problem.

2.2.1.1 Sampling-based Motion Planning

Since the introduction of Probabilistic Road Maps (PRMs) by Kavraki et al. in [95, 96], there has been increased use of *sampling-based* methods in motion planning, and they became the dominant paradigm in the field. In short, PRM based approaches construct a graph, or *roadmap*, of the state space by connecting configurations that are sampled randomly. PRM works in two phases: *construction* followed by *query*. In the first phase, PRM picks a set of random configurations. Then a collision checker algorithm is used to verify the selected random point is in C_{free}. If verified, the node is added to the graph. After having a set of nodes in the graph, it is needed to build the edges. Therefore, PRM iteratively checks all the nodes in its graph, and finds the closest n nodes to every node. Then a local planner is used to verify the existence of a collision free path between the pairs. If such path exists, the edge is added to the graph.

In the phase of querying, the shortest path from every start and goal state pairs are extracted from the graph. They are, therefore referred to as *multiple query* methods [94]. PRM has proven to be probabilistically complete, meaning that the probability of failure approaches zero as the number of samples increases. An example of PRM is illustrated in Figure 2.3. There have been numerous PRM variants proposed by many researchers in order to improve the approach for specific reasons, such as [7,23,86,121]. Geraerts and Overmars presented a comprehensive comparison study among most popular PRM variants in [72].

The popularity of *sampling-based* methods increased after the introduction of Rapidly exploring Random Trees (RRTs) by Lavalle and Kuffner [112]. PRM and RRT both rely on the same concept of graphs (trees in case of RRT) and random sample selection. But there are some fundamental differences between them. 1) RRT builds the tree incrementally while PRM generates the graph all at once. Therefore, RRT is more suitable for online planning applications. 2) RRT can include robot motion models into the planner, but PRM lacks this feature. It makes RRT a better candidate planner for robots with differential drive constraints [125].

RRT works in only one phase. The algorithm starts by building a tree at the start configuration, then randomly selects some points in the configuration space. A similar collision checker toolbox is called to ensure that the new random configuration is not in C_{obs}. If this condition is satisfied, the closest point in the tree is chosen to be expanded toward the new point. In the event that the motion is possible, and all constraints are satisfied, we can add the node to the tree until the

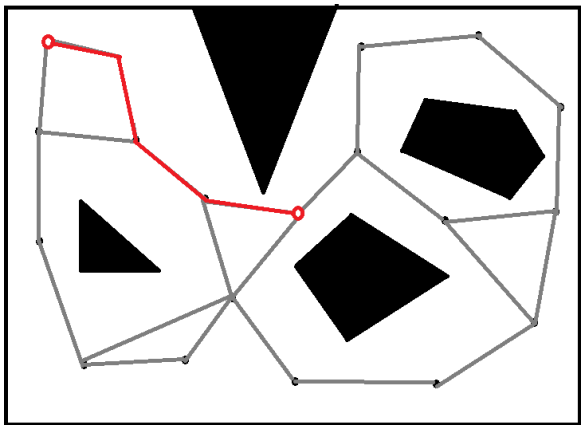


Figure 2.3: An example of PRM approach with a highlighted path

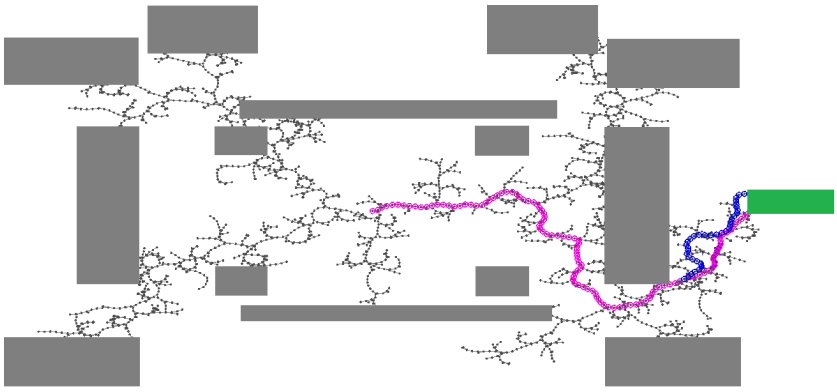


Figure 2.4: An example for RRT where two different paths in the same tree are highlighted to the goal area which is marked in green.

goal configuration is reached. An example of RRT is shown in Figure 2.4.

As in PRM, RRT is probabilistically complete. However, it does not provide any benefit in terms of optimality of the solution. Lavalle and Kuffner improved RRT by taking advantage of growing two trees from start and goal states, and proposed RRT-connect in [100]. The idea of connecting two trees reduces the calculation time significantly, but it is hard to connect the trees when the motion model has some constraints such as differential drive. RRT* is another popular extension for RRT [94] which focuses on optimality of paths. The idea is to find multiple paths to the goal area and *rewire* the tree several times until the optimal solution is reached. Similar to PRM, there are many variants for RRT, such as [20,30,69]. Likewise, Karaman presented a comprehensive survey on different types of incremental sampling-based approaches in [92].

2.2.1.2 Combinatorial Motion Planning

Combinatorial motion planning algorithms are referred to as *exact* algorithms which look for paths through the continuous configuration space without approximations. Almost all such algorithms are complete in the sense that they either find the exact solution, or report that there is no feasible solution for a problem. In contrast to that, sampling-based approaches are *resolution* and *probabilistically* complete. The most critical step in combinatorial planning is to represent the environment precisely. Almost all of the algorithms rely on the geometric representation of the world, and calculating a *roadmap* to the goal. They use methods from computational geometry to decompose the configuration space into geometric entities. Then the planner selects a set of optimal cells, and builds the roadmap on top of it [111].

The algorithms can provide an elegant solution if the problem is simple (i.e. low dimensional and convex models). Consequently, it may be unreasonable to use combinatorial motion planning to solve a high dimensional problem. Despite the variety of existing methods, only a few of them are efficient and easy to implement [111]. A dated and comprehensive survey of such approaches was presented by Schwartz [133]. Lindemann and Lavalle published a work to compare sampling-based approaches with combinatorial algorithms in [113]. For examples and popular studies in combinatorial planning, see [31, 116, 137]. Figure 2.5 shows an example of a feasible path with a combinatorial planning algorithm.

2.2.2 Planning under Uncertainty

As explained earlier in this section, there are various sources of non-determinism. Therefore, it is very beneficial to have planners that include uncertainty while solving problems.

Markov Decision Processes (MDPs) are one of the most popular planning tools in non-deterministic environments where the state of the system is observable. Even though we provide some background information about MDPs in Section 3.3.1, we refer the readers to [19, 128] for comprehensive references. We also refer the readers to [142] for applications of MDPs in robotics.

MDPs provide a mathematical foundation for sequential stochastic decision problems where the output of actions is random with a known distribution. MDPs are offline algorithms which generate optimal *policies*. A policy informs the action that must be taken at each state of the state space. In other words, policies are computed before the system starts to evolve. Later, the plan is

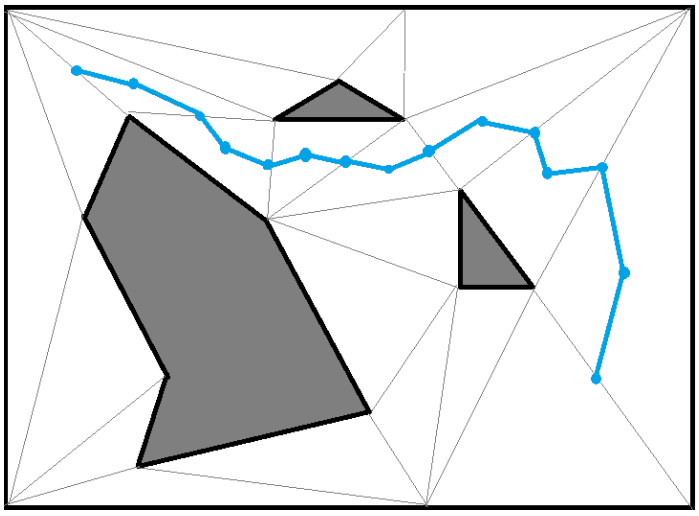


Figure 2.5: An example of combinatorial planning with triangulation. The picture is inspired by Figure 6.17 in [111].

used to read the pre-computed policy and operate the robot accordingly. We will elaborate more about MDPs in section 3.3.1. An example of a policy is shown in Figure 2.6.

Literature on MDPs is vast, and it has been tackled from several view points. There are multiple solvers for an MDP such as value iteration, policy iteration, linear programming and etc. [111]. There are also many variants of MDPs such as *factored MDPs* [25, 26]. The motivations behind proposing factored MDPs comes from high-dimensional MDPs which are composed of multiple parts that are weakly interconnected. Therefore, factored MDPs are introduced as structured MDPs which define them in a compact form. In factored MDPs, the state space is decomposed into multiple parts, and every part has its own state space, cost function, action set and transition probabilities. Thus, solving one part does not require considering the entire problem in one phase. It eases proposing efficient solvers. For any variant of MDP there are many proposed solvers to find the optimal solution, e.g. [49, 74] for factored MDPs.

Solving MDPs is computationally expensive. Therefore, many researchers put a lot of effort to improve the timing by using different approximation tools. One of the popular approaches is to approximate the solution for MDPs by using hierarchical models. In a hierarchical approach, the original MDP is decomposed into smaller MDPs and every small MDP is resolved independently. The decomposition can be applied on states, actions or transition functions. Here we describe some of these approaches.

MAXQ is one of the classical approaches which was proposed by Dietterich [50] as a general purpose methodology which computes hierarchies in multiple layers and solves the problem recursively. MAXQ builds a hierarchical model of the actions where it builds a graph of dependencies of different actions. The nodes in such graph are either *Max node* or *Q node*. In this model, Max nodes which have no child are primary actions, and the ones with children are imaginary aggregate actions. The children of Max nodes are *Q* nodes, and they represent subgoals. Imagine a taxi driving scenario where the actions of the taxi are *East*, *West*, *North*, *South*, *Pickup* and *DropOff*. The

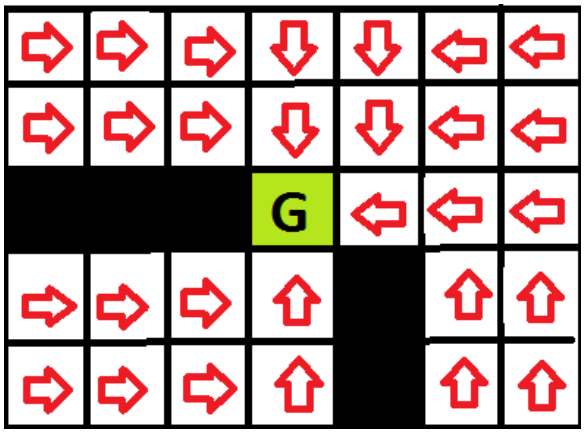


Figure 2.6: An example stationary policy for an MDP. The policy drives the planner to the goal area which is painted in green.

MAXQ hierarchical model for such planner is shown in Figure 2.7.

Another paper by Hauskrecht et al. [80] defined the term of *macro-actions* which relies on a technique called *region-based decomposition* to partition states and actions. It is very efficient if the environment is composed by some specifically separated areas such as rooms. The algorithm looks for entrances to the rooms and builds a set of new actions "macro-actions" which can be extracted from primary actions (e.g. *enter* and *exit* are sample macro actions.).

As explained earlier factored MDPs represent MDPs in compact form. More precisely, they change the representation of value functions by using *binary decision diagrams* (BDD) and *algebraic decision diagrams* (ADD) [8, 32]. This structured or symbolic representation helps the MDP replace its lookup table such as transition, cost or value functions with decision diagrams. In addition to reducing the size of the problem, it naturally clusters the states with similar behavior. Therefore, it does not require to keep numerous redundant copies of all of such states. Instead it processes the whole subset once, and applies the result to all of the states.

Algebraic decision diagrams (ADDs) are binary trees. Each internal node has a label which corresponds to a state in the MDP. The labels for leaves are real numbers. Figure 2.9 shows the example of an ADD and how it helps to reduce the size of state space. It is very efficient when the underlying state space is structured by itself e.g. some variables influence only a specific part of the state space, and can be ignored in others. The difference between ADD and BDD is on leaf level where the leaves are binary values in BDDs, and real numbers in ADDs.

Factored MDPs take advantage of ADD and BDDs in order to represent the state space. Two major solvers for factored MDPs are Symbolic LAO* [62] and Stochastic Planning using Decision Diagrams (SPUDD) [83]. SPUDD provides methods to convert value, transition and cost functions into ADDs, then they reduce the size of ADDs without any loss. At the end SPUDD presents a value iteration function which solves the problem. Symbolic LAO* is an extension to LAO* [77] which is tailored for factored MDPs. It uses ADDs with some heuristics in order to exclude a set of unreachable states. Similarly there are many other approaches to solve factored MDPs; e.g. with policy iteration as in [26, 27].

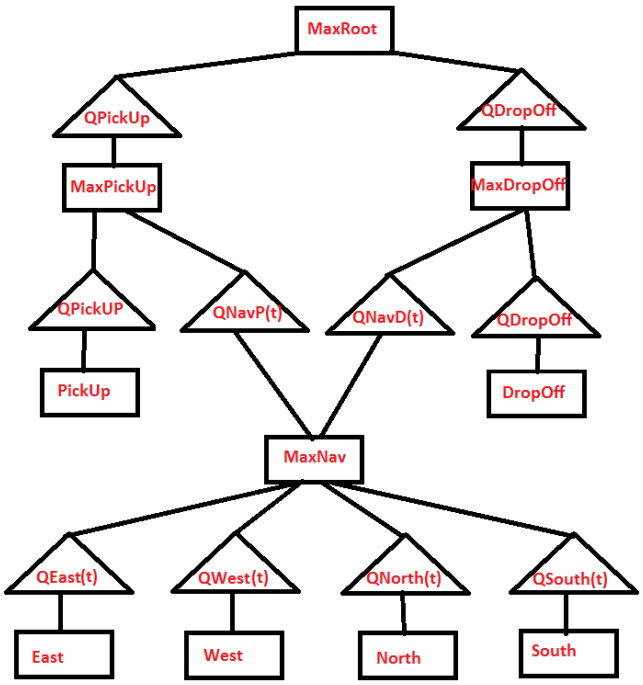


Figure 2.7: MAXQ hierarchical model for taxi driving problem. Rectangular nodes are Max nodes and Triangular nodes represent Q nodes. QNavP is a subgoal to drive the taxi to a pickup location, and QNavD similarly drives the taxi to a drop off station. This is inspired by Figure 2 in [50].

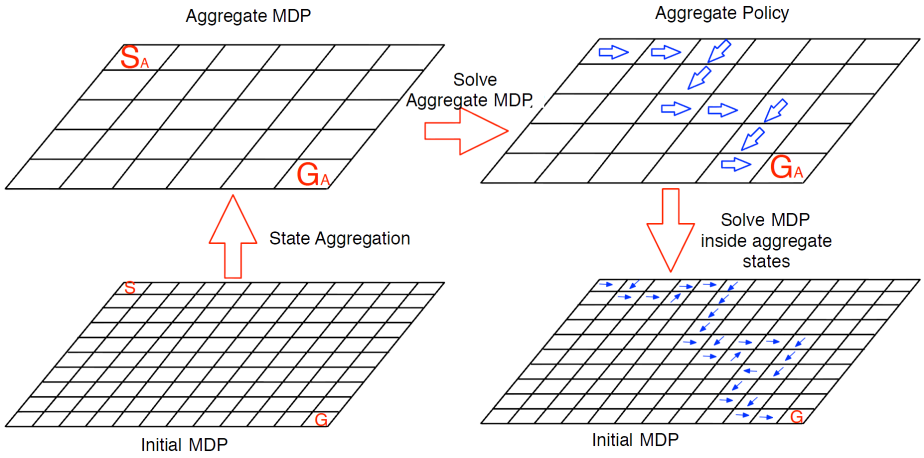


Figure 2.8: An example of hierarchical models.

S1	S2	S3	F(S)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	3
1	1	1	3

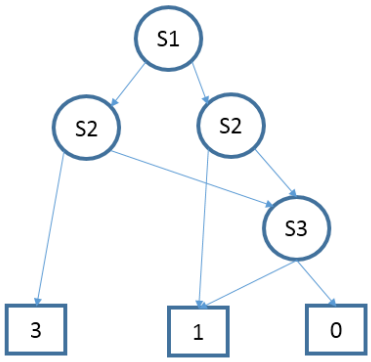


Figure 2.9: The left side picture shows a standard lookup table. The picture on the right side illustrates the ADD representation for the same lookup table. The picture is inspired by Figure 5.1 in [98].

Most of the mentioned hierarchical solvers for factored MDPs, focus on improving ADDs and reducing the size. More recently, Barry et al. looked at the problem from another perspective. They tried to reduce the size of state space by building aggregate states from primary states as seen in Figure 2.8. The idea is to provide a guarantee on the existence of a solution, i.e. if there is a solution on the non-hierarchical MDP, there has to be a solution on the hierarchical MDP as well. Their first attempt in [11] was to use the theory of strongly connected components [45] to build the partitions in a way that any state in a partition is connected to every other state in the same partition. However, the partitioning itself was computationally expensive. Therefore, they simplified their partitioning methodology in [12] by defining a set of rules, and excluded the usage of strongly connected components. The new algorithm, DetH*, provided the same type of connectivity guarantee, but was superior in performance.

Even though all the mentioned approaches provided an approximation for the optimal solution, Dai et al. [46] pioneered a new method by proposing Topological Value Iteration (TVI) which is a hierarchical model that guarantees the optimality of the solution. In other words, it saves time by solving a hierarchical model without sacrificing optimality of the solution. The intuition is to start building the hierarchical model from the goal state. Then it incrementally builds the partitions layer by layer. Anytime a new partition is generated, TVI runs a value iteration on that partition until it completely converges. At that time, the values in the value function will not change anymore. Then, these values will be propagated and used in running the value iteration on the next layer/partition. This process is repeatedly applied until the optimal values for the entire state space are calculated.

The same authors improved TVI, and proposed Focused TVI (FTVI) which takes advantage of the initial state, and focuses only on calculating values for the required subset of states [47].

All the above approaches are only a sample of hierarchical solvers for MDPs while there are many other methods such as [9, 28, 145] to solve MDPs hierarchically. There is an extensive survey by Kolobov which takes a look at different MDP optimization algorithms [98].

All the aforementioned MDP variants and solvers have one limitation i.e. they all have a single objective function which has to be minimized or maximized in expectation. Constrained

MDPs (CMDPs) emphasize multiple cost functions where one of them is the *primary* cost function which has to be minimized and the rest of them are *additional* cost functions which have to be bounded. Altman provides an extensive introduction to CMDPs in [5]. There were some efforts to include multiple cost functions into a single objective function of an MDP [70, 120]. However, their resulting objective function is an artificial measure and can not be easily interpreted or extended. On the contrary CMDPs provide an easy-to-define and easy-to-extend method to solve planning problems under uncertainty with multiple cost functions.

Despite the strength of CMDPs, they have not been very popular in robotics. CMDPs are usually solved with linear programming and even a small problem may result in a large number of variables in the linear program which makes CMDPs computationally expensive, esp. in robotics where robots have limited resources for calculations.

Among the limited number of researches which used CMDPs in robotics applications, we briefly present some of them. Ding and coauthors have been particularly active in this domain.

Ding et al. have used CMDPs in some instances to solve motion and mission planning problems. More specifically, they proposed a hierarchical mission planning solver where a very small CMDP is the high level mission planner, and a PRM-based method runs on the low level to drive the robot. In this study they used the theory of model verification to satisfy some subgoals [51]. This specific work will be further discussed later in Section 2.3.2, because it bridges the two topics of CMDPs with linear temporal logic. They also used similar approaches to solve mission planing problems as in [52, 53].

El Chamie and Acikmese proposed a new solver for finite-horizon CMDPs which is based on linear programming as well. They managed to control swarm robots by using this approach [35]. Boussard and Miura used CMDPs in search problems where the objective is to find as many objects and possible in a limited time frame [24].

Carpin and his coauthors solved rapid deployment problems with CMDPs where robots have to maximize the probability of reaching the goals with some limits on time [34, 39]. Dolgov leveraged CMDPs to describe the constraints on the architecture of autonomous agents [56]. CMDPs have also been used in other fields of science such as in communications [139, 140].

2.3 Motion Planning with High-Level Task Specifications

As robots become more sophisticated, they are gettin capable of accomplishing complex missions with constraints (for example see Figure 2.10). Therefore, the problem in motion planning is not only to find a feasible path from one point to another, but also to complete some other tasks along the path. This raises two important questions:

- How to formally and universally define tasks in robotics applications?
- How to compute a plan to solve such complicated problems without being application-specific?

The rest of this section reviews literature related to these questions.

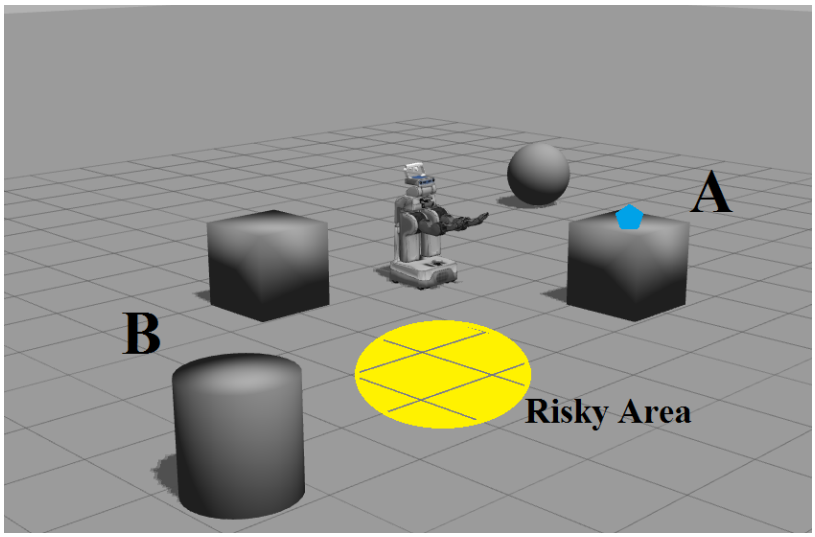


Figure 2.10: A robot is tasked to drive to point *A*, pick up the blue object then drop it off at point *B*. The planner has to avoid the risky area painted in yellow as much as possible. At the same time, the entire mission should be completed in a prescribed amount of time.

2.3.1 Task Specification

Robot tasks vary significantly based on many factors such as the type of the robot, its application and capabilities, the environment, etc. Consider a simple manipulator robot in a factory. The robot has very limited capabilities and its sample task can be defined as picking up an object, turning 90 degrees and dropping that off. On the contrary, a Mars rover equipped with many sensors and actuators is a robot with high capabilities. Its mission could be to go to a specific location, read some sensory data, perform a drilling action, collect some samples, go back to the station, analyze the samples, and send the results back to the base. It is obvious that this second task includes many subtasks. The challenge is to offer a methodology that specifies tasks without being tied to any specific application.

There are many ways to define tasks in robotics. As an example, Rodriguez et al. tried to define tasks in a positioning problem where the position of an object in 3D world should be calculated from some relative positions [129]. They proposed a specific task definition framework to satisfy their needs based on *symbolic geometric constraints*. Their method is applicable to positioning applications only.

A very similar problem was tackled by De Schutter et al. for calculating the absolute motion, based on a set of relative motions [48]. They also interpreted the task in a customized form which was tailored toward their own applications. Ulem et al. stepped further and developed their own customized C++ code to handle tasks for their specific application [143]. They suggested their code to be used as a framework for task specifications.

MissionLab is also a toolbox providing the user with a graphical interface to describe tasks in multi agent systems and generating the required C++ code to specify the tasks [118]. There are many other instances of task specifications for a limited set of applications as in [3, 97, 132]. The

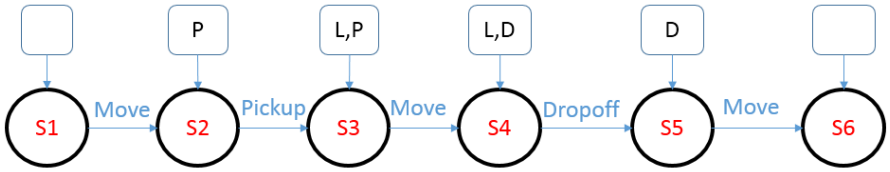


Figure 2.11: An example for sequence of states in a run. The circles show the state of the system and the boxes above them represent which properties are true. If the box is empty, it means that none of the properties are true. Last but not least, the blue labels on edges are actions. The robot starts at state $S1$ where none of the properties is true. Then it performs the *move* action to go to $S2$. In this state the property P is true which indicates that it is a pickup station. It operates *pickup* action which results in the robot being loaded while still at pickup station. The next *move* action drives the robot to the delivery station $S4$. After performing the *Dropoff* action, it unloads at the delivery station. Finally, it *move*s to the goal state $S6$.

main drawback of these approaches is that they are not easily extendable to neither new tasks nor new applications.

In recent years, formal languages have been used more and more often to define general-purpose tasks in robotics applications. Belta et al. proposed one of the first utilizations of Linear Temporal Logic (LTL) in robot task specifications [17]. Even though the primary purpose of formal languages was to analyze strings and sequences of symbols [130], the temporal nature of LTL formulas is its main strength to define tasks that evolve over time [17]. The literature about formal languages and model checking is very rich and the reader is referred to Baier's book [10] for a complete reference.

Here we will briefly elaborate the connection between robot motion planning and LTL properties. While the robot is running, the system traverses a set of states where every state has a set of properties. Thus, the status of the system is recognized by a sequence of states. Let us consider an example of a forklift in a factory that performs pickup and delivery tasks. Suppose there are three possible actions of *move*, *pickup* and *dropoff*. Figure 2.11 shows an example sequence of states. Every state of the system can have three properties as following:

- Loaded, denoted by L: This property shows whether the robot has already loaded an object and carries that or no.
- At pickup location, denoted by P: This property indicates if the robot is at the pickup location. Only in this location the *pickup* action can be applied.
- At delivery location, denoted by D: Similarly, it indicates that the robot is at the delivery location. The action of *dropoff* is only valid if both L and D properties are true.

Having such a system, we can define desirable behaviors. LTL properties are used to differentiate between desirable and undesirable behaviors. In other words, LTL properties accept the sequences of states that are desirable and reject otherwise. Figure 2.12 illustrates a task which ensures the system loads only in a loading zone, and unload in a delivery location. It also checks to avoid loading or unloading multiple times.

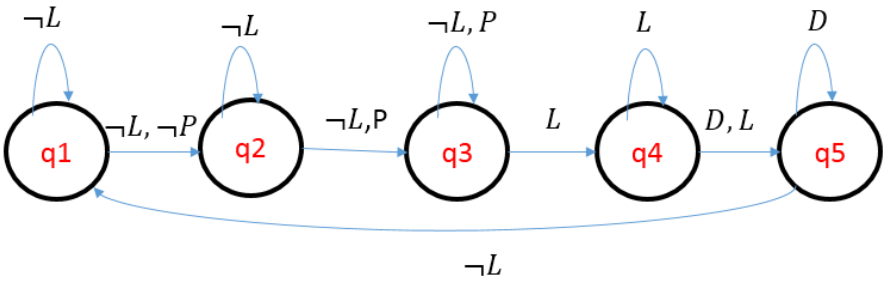


Figure 2.12: An example of acceptable behavior of a robot. It ensures that the loading only happens in the loading zone, and the delivery is performed in the delivery station.

2.3.2 Planning with Formal Methods

Planners exist for deterministic or non-deterministic environments. Consequently, task specification should be applied to both types of planners as well. Linear Temporal Logic (LTL) formulas, referred to as LTL properties as well, are constructed by a set of atomic boolean propositions and a set of operators which are either logical or temporal [73]. The temporal operators play a critical role in task specifications and represent the desirable sequence of atomic propositions where every atomic proposition may be a distinct subgoal to achieve.

The first step is to convert the task into an LTL property or one of its variants such as syntactically co-safe LTL (sc-LTL) properties. The next step is to use the LTL formulas in one of the following two ways:

- LTL formulas as runtime verification tools: A planner calculates a plan and the LTL properties are used to verify whether the plan satisfies the task or not [60, 108, 127].
- LTL formulas in planners or controllers: The LTL property is included in the planning phase which makes the output policy *correct-by-design* as in [36, 115, 124].

In robot applications, the second approach is more popular, and there have been multiple studies in that area. Plaku and Karaman provide a comprehensive survey about robot motion planning with LTL specifications where they consider numerous different scenarios [126]. They also analyzed the use of LTL properties in different layers of controllers. Among the vast literature in this context, we only present a handful of related work. We focus on *correct-by-design* policies in both deterministic and non-deterministic environments, and also considers a selected runtime verification tools.

2.3.2.1 Runtime Verification

Model checking algorithms analyze every possible state of the system to predict any possible failure. Such algorithms usually require a huge amount of memory. Therefore, state space explosion is a common problem in those methods [42]. Thus, runtime verification tools were proposed as a limited alternative solution. Runtime verification is a lightweight technology to verify whether

the current state of the system satisfies a predefined property or violates that [16]. Among these approaches, we mention few of them.

Cini and Francalanza proposed a proof system that checks the current state together with the trace of the system to verify or reject certain LTL properties [41]. The proof system is also capable of determining why and when the finite undesirable trace happens. They also claimed that their approach can be used as a unifying framework to check other systems against runtime verification tools.

Fainekos et al. go one step further [61]. They not only verify the boolean values of LTL properties, but also check some metric values and the topological information regarding distances from unsatisfiability by taking advantage of Metric Temporal Logic (MTL) properties [99]. MTL properties are extensions to LTL properties where one can define a time limit in which a property becomes true. This addition makes MTL properties suitable for real time monitoring [141].

Since Robot Operating System (ROS) is becoming the dominant framework to develop robotics softwares, Huang designed a ROS package, ROSRV, for runtime verification of robot behavior [87]. This toolbox is planned to be general purpose and easy to plug into any planner. Moreover, there are many general purpose runtime verification tools which can be applied to robotics applications as well [13–15, 71, 134].

2.3.2.2 Correct-by-design Planners

The primary use of LTL properties in motion planning algorithms is to drive the robot away from some specific regions without eliminating those points from the configuration space.

Starting from deterministic planners, there are two ways of using LTL properties in sampling-based algorithms:

- **Type I**: Apply LTL formulas while building the graph or roadmap.
- **Type II**: Run sampling-based planner and LTL in separate layers.

The number of approaches from type I is very limited while there are several studies emphasizing type II planners. It is also very popular to use type II planners in non-deterministic environments. We briefly present some works in both categories, starting from Type I.

One of the early works by Plaku [124] included LTL properties in RRT algorithm. He proposed a discretized planner focusing on high dimensional environments. He applied LTL properties during the expansion step of building the tree. Therefore, the planner grows the tree toward the area which satisfy the predefined LTL properties.

The same author published another work to include LTL formulas into PRM algorithms [123]. The fundamental difference between using LTL properties in PRM and RRT is that PRM graphs are not built incrementally from a starting state as opposed to RRT trees. Moreover, it is vital for LTL properties to have the exact sequence of states starting from the initial state. Because of the tree structure of RRTs, the starting state is always the root of the tree. Therefore, such sequences are easily calculated for every node in the tree by traversing back to the roots. On the contrary, PRM approaches rely on graphs where the starting node is not specified. Besides that, PRM algorithms are multi-query approaches which means a trajectory segment can be accepted by an LTL property

but be rejected by the same LTL property if the starting state is different. In short, it is impossible to validate or invalidate an LTL formula when building the PRM graph. Thus, they proposed a different query method from the standard PRM algorithms. Their proposed algorithm takes LTL properties into account while finding a path on the graph. More precisely, it only adds vertexes to the solution that satisfy the LTL properties.

Similarly, Karaman and Frazzoli proposed a general framework which utilizes sampling-based approaches and focuses on LTL properties that require recursions [93]. They proposed to use Rapidly-exploring Random Graphs (RRGs) instead of Rapidly-exploring Random Trees to replace trees with graphs. The graph nature of RRG enables the approach to consider recursive LTL properties.

We next consider planners of type II. As mentioned before, type II planners are more popular than type I planners. Such planners are usually hierarchical where the high level planner normally works on a discrete model of the environment, and takes care of satisfying LTL properties. Sampling-based approaches are used on the low level to plan in continuous domain where the motion model of the robots has to be considered. Most of the approaches that we will explain in the rest of this section follow this type of planning, and the difference appears in what type of high level planners they use. Some approaches use deterministic planning on high level such as combinatorial planning, but more frequently non-deterministic planners such as MDPs are used as high level solvers.

As an example, Bhatia et al. worked on a hybrid approach where the high level planner is a geometric approach which emphasizes satisfying the tasks and the low level planner uses a sampling-based approach to drive the robot [21]. Their focus is on creating an abstraction of the system by utilizing geometric properties of obstacles and the propositions. The high level planner performs a graph search in this specific instance. Later on, they extended their work and provided that as a software package [22].

Task specification during planning under uncertainty has also been widely discussed in the literature. The theory of Markov Decision Processes (MDPs) is the dominant toolbox in such planners, similar to standard planners in non-deterministic situations. The general approach for planning with MDPs and LTL properties which is commonly used, consists of the following steps:

1. Step I: Set up an MDP from a discretized model of the environment.
2. Step II: Build a Deterministic Finite Automaton (DFA) for each LTL property.
3. Step III: Compute a product between MDP and DFAs.
4. Step IV: Run an optimization algorithm to find a policy.
5. Step V: Pass the policy to a sampling-based approach to drive the robot.

The product calculation between an MDP and a DFA is a standard approach which can be found in [10]. We will also explain it with more details in Section 5.3. We present some of the work in this area and explain their pros and cons.

Among all the studies, Lacerda et al. utilized MDP and its combination with LTL properties to satisfy a single task for a mobile service robot [103]. Their emphasis was on finding a path

with lowest possible cost which satisfies the task. They also tried to automatically generate the required DFAs by using Probabilistic Real-time Systems (PRISM) [101] in order to translate the LTL properties into deterministic finite automaton (DFA) [82]. Since the planning happens in uncertain environments, it is very probable that the robot deviates from the original path, and falls into an unexpected state which does not satisfy the required task. When the path is optimal, there is not a defined boundary of satisfying the task which can be considered as the most important weakness of this approach.

In another study, Wongpiromsarn et al. used MDPs and LTL properties to show an application in multi-agent systems problems where they maximized the probability of satisfying the task [147]. This approach also suffers from a similar problem. When they focus on maximizing the probability of satisfying a task, they do not consider the total cost of the process.

In a similar integration of MDP and LTL properties, Ulusoy and coauthors worked on the interaction between the robot and the environment where there is no control on the environment. An example application is driving an autonomous car in a pedestrian crossing area where the vehicle should be aware of the movements of people but cannot control them [144]. Their MDP is also tuned to maximize the probability of satisfying tasks.

A study by Wolff et al. used the same methodology but focused on worst case scenarios where not only the task has to be satisfied with highest possible probability, but also the worst-case probability of failure has to be minimized [146].

Luna et al. replaced MDPs with Bounded parameter Markov Decision Processes (BMDPs) to solve control problems. In a BMDP, the transition probabilities are not precisely known. Instead they are bounded to some specific ranges [117]. They focused on maximizing the probability of reaching a single task.

While most of the approaches highlight the boolean satisfaction of LTL properties, Lahijanian et al. calculate the distance from satisfying an LTL property. They define an extra cost function which depends on how far the solution is from being accepted by an LTL property, and use weighted automaton in their proposed algorithmic planner [104]. Then they extended their work, and stressed on soft rather than hard constraints [107]. They proposed an algorithm for "specification revision". Anytime the distance from satisfying an LTL property increases, the algorithm revises the DFA and proposes an alternative DFA which is more likely to be satisfied.

Lahijanian also published some more studies in this area where he used Probabilistic Computation Tree Logic (PCTL) instead of LTL properties in conjunction with MDPs in order to satisfy tasks in probabilistic form, and showed the flexibility of this method [105, 106]. PCTL is an extension to Computation Tree Logic (CTL) [57] properties and can express statements such as "*a property holds for a certain amount of time with a specific probability*" [40].

There are other ways to solve planning under uncertainty with existence of an LTL property. As an example, Gol an Belta tackled the same problem by focusing on convex optimization [75]. However, their main focus was on control side of the problem.

As seen, almost all the approaches suffer from two limitations:

- They take only one single objective into account which is either maximizing the probability of satisfying tasks or minimizing the cost of accomplishing a mission. It is possible that maximizing the probability of satisfying a task causes a very high cost. On the contrary,

minimizing the cost may reduce the probability of satisfying tasks. Therefore, it is not ideal to only consider one of them without controlling other factors.

- Their proposed planners take only a single task into account. However, there are numerous applications that require several tasks to be considered.

To address the second issue, Etessami et al. proposed a planner which is capable of considering multiple LTL properties in a solver [58, 59]. Their proposed algorithm satisfies every LTL property with a predefined satisfaction rate. They also followed the same steps as formerly described with some modifications. They run a graph reachability algorithm, after applying the product step, to find paths that satisfy the LTL properties. After assuring the possibility of satisfying all LTL properties with desired rates, they run a linear programming solver to calculate the final policy. However, this approach has some limitations as well:

- The reachability graph algorithm is dependent on a single start state. The algorithm faces problems, if the initial state is a generic mass distribution, not concentrated in a single state.

- The product operation does not consider dependencies of different LTL properties. In other words, if two LTL properties contradict with each other, the output of the product algorithm is an empty set.

- They do not consider any cost function, and only focus on satisfying LTL properties with desired rates.

Following Etessami's research, there have been some improved versions of that algorithm by adding a cost function [67], or proposing an iterative solver with some limitations [67]. Using the same methodology Kwiatkowska et al. suggest how to compose and decompose different LTL properties to speed up calculations [102]. Similarly, Struck et al. show a framework to solve such planners with Pareto curves [138].

Alternatively, Ding et al. replaced MDPs with Constrained MDPs (CMDPs) to consider multiple cost functions [51, 52, 54]. They took advantage of the theory of occupation measures [5], and estimated the probability of reaching a certain state on a state space. This estimated measure is used as a constraint in the CMDP to assure the generated policy by the CMDP satisfies the task with the predefined probability. However, their approach also suffers from the weaknesses of simplicity where only one LTL property can be used.

Chapter 3

Theoretical Background

3.1 Introduction

This chapter presents the theoretical background required for the rest of this document. It starts by explaining *model verification* in Section 3.2. The section provides some necessary definitions including *finite automata* and *linear temporal logic properties*. Most of the presented materials in the section are standard, but the definition of *extended total deterministic finite automata* is novel. Later, in Section 3.3, we elaborate on *sequential decision models* which include standard *Markov Decision Processes* and *Constrained Markov Decision Processes*. Finally, in section 3.4, we present the concept of *Labeled Constrained Markov Decision Processes* which is another contribution of this thesis.

3.2 System Verification

System verification approaches verify the state of a system against some specified properties [10]. The output of a verification algorithm is whether the system satisfies the gievn properties or not. It also indicates the failure point, in case of one. There are numerous methods to verify softwares such as review processes, implementation of test cases or etc.

Model checking is a formal verification technique which provides a principled approach to systematically inspect all the states of a system to verify a set of desired behavioral properties [10]. Model checking approaches are attractive, because they can be automatically applied. Since formal methods offer a rich foundation to describe the progress of a system, they have been extensively used in model checking approaches. We briefly present the minimal set of theoretical background for formal languages and formal methods in the rest this section.

System verification and model checking have been discussed in multiple textbooks [10,43,88] which can be used as references.

3.2.1 Finite State Automata

We recall basic concepts in automata theory. The reader is referred to classic textbooks like [84, 114, 135] for a thorough introduction to the topic. Most of the material we present in

this section is standard, and can be found in the aforementioned references. Our only addition is the definition of *extended total deterministic finite automata* which is also a form of finite state automata with some extensions on its transition function and special input words. We start with some basic definitions:

Definition 1 (Alphabet) *A finite and nonempty set of symbols Σ is called alphabet.*

Definition 2 (Word) *A word w over an alphabet Σ is a sequence of 0 or more elements of Σ. A word of length $n > 0$ will be indicated as $w = \sigma_1\sigma_2 \dots \sigma_n$, where σ_i is the ith symbol in the sequence. The set of all possible words over the alphabet Σ is denoted by Σ^*.*

Definition 3 (Language) *A language $\mathcal{L}$ over Σ is a subset of all possible words over Σ, i.e., $\mathcal{L} \subset \Sigma^*$.*

For sake of brevity, we do not present all the required background knowledge. We refer the reader to [114] for a comprehensive description of formal languages, operators and etc.

As per definition 3, a language is a set with a finite or infinite number of elements. The theory of automata provides a finite representation of languages. In other words, an automaton is capable of recognizing languages. At this point, we present some variants of automata.

Definition 4 (Nondeterministic finite automaton) *A nondeterministic finite automaton (NFA) is a 5-tuple $\mathcal{N} = (Q_\mathcal{N}, q_0, \delta_\mathcal{N}, F, \Sigma)$ where:*

- *$Q_\mathcal{N}$ is a finite set of states.*
- *$q_0 \in Q_\mathcal{N}$ is the initial state.*
- *$\delta_\mathcal{N} : Q_\mathcal{N} \times \Sigma \to 2^{Q_\mathcal{N}}$ is the transition function.*
- *$F \subseteq Q_\mathcal{N}$ is the set of accepting states.*
- *Σ is a finite set of symbols, or the alphabet.*

Note that according to the definition 4, it can be that $\delta_\mathcal{N}(q, a) = \varnothing$ for some couple (q, a). In such case the transition is not defined. In an NFA it is also possible to have $|\delta_\mathcal{N}(q, a)| > 1$ which means the successor state can be multiple states.

Definition 5 ($\delta^*_\mathcal{N}(q_0, w)$) *It represents the state reached by recursively applying the transition function $\delta_\mathcal{N}$ to all symbols in w.*

NFA $\mathcal{N}$ rejects the word w in one of the following two conditions, otherwise it accepts the word:

- While computing $\delta^*_\mathcal{N}(q_0, w)$ the transition function is not defined for one of the symbols in w, then automoton stops and the word is rejected.
- The automoton processes the word completely, and ends in a state which is not final. Therefore, the w is rejected. In other words, $\delta^*_\mathcal{N}(q_0, w) \notin F$.

The set of words over Σ is partitioned into the set of words accepted by $\mathcal{N}$ (also called the language accepted by $\mathcal{N}$) and the set of words rejected by $\mathcal{N}$. A deterministic finite automaton is a special case of an NFA where the successor state is uniquely determined, as implied by the following definition.

Definition 6 (Deterministic finite automaton (DFA)) *A DFA $\mathcal{D} = (Q_{\mathcal{D}}, q_0, \delta_{\mathcal{D}}, F, \Sigma)$ is an NFA in which $|\delta_{\mathcal{D}}(q, a)| \leq 1$ for each $q \in Q_{\mathcal{D}}$ and $a \in \Sigma$.*

It is well known that every NFA $\mathcal{N}$ can be converted into an equivalent DFA $\mathcal{D}$, i.e., a DFA accepting all and only the words accepted by the NFA. Hence in this document, we will simply consider DFAs.

A total DFA is a special type of DFA where there is one and only one transition at any state for every input symbol. The addition to the transition function helps eliminate the first condition for rejecting a word. Therefore, the only condition that can cause failure in processing a word is the second one.

Definition 7 *A total deterministic finite automaton (total-DFA) is a DFA in which $|\delta(q, a)| = 1$ for each $q \in Q$ and $a \in \Sigma$.*

We will later define a new type of automata, the extended-total-DFA, aiming at accepting a certain class of infinite length words. Its utility will become evident in the later part of the document. We start by defining infinite length words.

Definition 8 (Infinite length word) *An infinite length word over an alphabet Σ is a function $w : \mathbb{N} \to \Sigma$. We indicate with $w(i) = \sigma_i$ the i-th character in the infinite length word.*

Definition 9 ($\mathscr{G}$-ending infinite length word) *Let w be a finite length word of length $k > 0$ over alphabet Σ (with $\mathscr{G} \notin \Sigma$). The corresponding $\mathscr{G}$-*ending infinite length word *is an infinite length word w' over the alphabet $\Sigma' = \Sigma \cup \{\mathscr{G}\}$ such that $w'(i) = \sigma_i$ for $1 \leq i \leq k$ and and $w'(i) = \mathscr{G}$ for $i > k$.*

Informally speaking, definition 9 establishes that the $\mathscr{G}$-ending infinite length word associated with w is obtained by repeatedly appending $\mathscr{G}$ characters at the end of w. This requirement, which may seem odd at first, will be essential to establish the probability that a given specification is satisfied. Starting from these definitions, we introduce the extended-total-DFA associated with a total DFA, i.e., an automata aimed at processing $\mathscr{G}$-ending infinite length words. Before explaining its rationale, we give its formal definition.

Definition 10 (Extended Total DFA) *Let $\mathcal{D} = (Q_{\mathcal{D}}, q_0, \delta_{\mathcal{D}}, F, \Sigma)$ be a DFA. Its associated extended-total-DFA $\mathcal{E} = (Q_{\mathcal{E}}, q_0, \delta_{\mathcal{E}}, F, \Sigma_{\mathcal{E}})$ is defined as:*

- *$Q_{\mathcal{E}} = Q_{\mathcal{D}} \cup \{q_a, q_s\}$ where q_a, q_s are new states not in $Q_{\mathcal{D}}$;*
- *$\Sigma_{\mathcal{E}} = \Sigma_{\mathcal{D}} \cup \{\mathscr{G}\}$ where $\mathscr{G}$ is a character not in $\Sigma_{\mathcal{D}}$;*

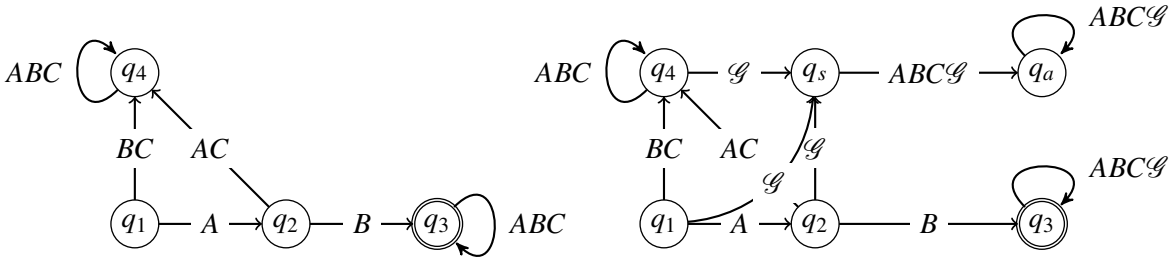


Figure 3.1: The figure shows on the left a DFA accepting the regular expression $AB(A + B + C)^*$, and on the right the associated extended-total-DFA.

- $\delta_{\mathcal{E}} : Q_{\mathcal{E}} \times \Sigma_{\mathcal{E}} \to Q_{\mathcal{E}}$ *is the transition function defined as follows:*

$$\delta_{\mathcal{E}}(q, \sigma) = \begin{cases} \delta_{\mathcal{D}}(q, \sigma) & \text{if } q \notin F \cup \{q_s, q_a\} \wedge \sigma \neq \mathscr{G} \\ q & \text{if } q \in F \\ q_s & \text{if } q \in Q_{\mathcal{D}} \setminus F \wedge \sigma = \mathscr{G} \\ q_a & q = q_s \vee q = q_a. \end{cases} \tag{3.1}$$

Besides the different alphabet $\Sigma_{\mathcal{E}}$ and state set $Q_{\mathcal{E}}$, the extended-total-DFA $\mathcal{E}$ differs from $\mathcal{D}$ in the transition function. The first case in the definition of the transition function stipulates that as long as $\mathcal{E}$ processes characters different from $\mathscr{G}$ and it has not yet entered a final state or the two new states q_s, q_a it proceeds exactly as $\mathcal{D}$. The second case establishes that once the state enters a final state, it remains there. Note that this is different from what happens in a DFA, where the state may first enter, but then leave the set of final states as more symbols in the word w are processed. Instead, the extended-total-DFA recognizes prefixes, i.e., as soon as a prefix leads to F, it stays there forever. The third case establishes that when the first instance of $\mathscr{G}$ is encountered, if the state is not final, it moves to q_s. The final case ensures that all remaining occurrences of $\mathscr{G}$ lead to q_a and the state remains there forever. Note that $\delta_{\mathcal{E}}$ stipulates that q_s is visited at most once while w' is processed, that is if the state enters q_s upon processing the first instance of $\mathscr{G}$, it will leave it as soon as the second instance of $\mathscr{G}$ is encountered, and never return there (see Figure 3.1 for an example of extended-total-DFA derived from another DFA accepting regular expressions). A few simple examples will clarify this mechanism and the differences between $\mathcal{D}$ and $\mathcal{E}$.

1. Let w be a finite length word on Σ, part of the language accepted by $\mathcal{D}$, and let w' be its associated $\mathscr{G}$-ending infinite length word. When w' is processed by $\mathcal{E}$, the state eventually enters F, and remains there forever.

2. Let w be a finite length word on Σ, not part of the language accepted by $\mathcal{D}$, but such that while $\mathcal{D}$ processes w, the state first enters the set F, but then leaves it. Let w' be its associated $\mathscr{G}$-ending infinite length word. When w' is processed by $\mathcal{E}$, the state eventually enters F and remains there forever.

3. Let w be a finite length word on Σ, not part of the language accepted by $\mathcal{D}$. When w' is processed by $\mathcal{D}$, the state will enter q_s when the first instance of $\mathscr{G}$ is encountered, and then moves to q_a, and remains there forever.

Therefore, for each $\mathscr{G}$-ending infinite length word the state will eventually be either a state in F or q_a. Moreover, in each case the state q_s is visited at most once, and it is never visited if any state in F is entered. The new state q_s is called *sink state*, whereas the new state q_a is called *absorbing state*. In the next section we introduce a specific subset of Linear Temporal Logic that is characterized by "good prefixes." This will clarify why the extended-total-DFA has been defined this way.

3.2.2 Linear Temporal Logic

LTL is a formalism used to specify desired behaviors of reactive systems. There is a rich body of work in this domain and even a superficial literature review in this area is beyond the scope of this document. The reader is referred to [10] for a general introduction to the topic, and to [126] for a recent overview with an emphasis on motion planning applications. It is worth recalling that in this context the term *temporal* refers to the sequencing (or relative order) of certain events, and is not strictly related to when they occur. In the following, the terms *formula* and *property* will be used as synonyms. LTL formulas are built on top of a finite set of atomic propositions Π. Each element in Π can either hold or not. An evaluation of Π determines which atomic propositions hold, and which ones do not, i.e., it assigns a value of either 0 (false) or 1 (true) to each element in Π. An evaluation is then completely specified by just stating the set of atomic propositions that hold, i.e., by specifying a subset of Π. Starting from Π, the power set 2^{Π} can be thought as an alphabet over which words of finite or infinite length can be built. In particular, $(2^{\Pi})^{\omega}$ is the set of infinite length words built over Π. The sequence of symbols in a word $w \in (2^{\Pi})^{\omega}$ can be used to model a temporal evolution. That is to say that the i-th symbol in a word w is an element of 2^{Π}, and can therefore be thought as the set of atomic propositions holding at the i-th time instant. Starting from Π, an LTL formula φ can be produced using this grammar [10]:

$$\varphi ::= \text{true} \mid p \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \, \mathtt{U} \, \varphi_2$$

where $p \in \Pi$ is an atomic proposition. Besides the classic logical operators $\wedge$ and $\neg$, LTL introduces the two temporal operators $\bigcirc$ and $\mathtt{U}$. The unary operator $\bigcirc$ is called *next*, and the formula $\bigcirc\varphi$ holds at the current time step if φ is verified at the next time step (hence the name). The binary operator $\mathtt{U}$ is called *until*, and the formula $\varphi_1 \, \mathtt{U} \, \varphi_2$ holds at the current time step if in the future φ_2 holds and φ_1 holds until φ_2 holds. Combining these basic operators, it is customary to introduce two additional unary temporal operators, namely $\Diamond$ (eventually) and $\Box$ (always). The formula $\Diamond\varphi$ holds if φ is true some time in the future, whereas $\Box\varphi$ holds if φ is verified at every time step. Starting from this definition of LTL, interesting specifications for robot behaviors can be formulated, and this leads also to automatic synthesis of controllers (see references in Chapter 2). Every formula φ partitions $(2^{\Pi})^{\omega}$ in two sets, i.e., the words satisfying the formula and those not satisfying the formula.

In [52] the so called syntactically co-safe LTL (sc-LTL) properties were considered to specify desired robot behaviors. Starting from Π, a sc-LTL formula is built using the operators and ($\wedge$), or ($\vee$), not ($\neg$), and the temporal operators *eventually* ($\Diamond$), *next* ($\bigcirc$), and *until* ($\mathtt{U}$). Furthermore, the operator $\neg$ can only be used in front of atomic propositions. Note that sc-LTL formulas miss the *always operator*. This refinement is introduced for practical and computational reasons. First, as

pointed out in [52], robots operate over missions with finite duration, and the *always* operator is therefore not particularly useful, since it is often meant to describe behaviors of indefinite length. Second, sc-LTL formulas are verified in a finite amount of time, and for each sc-LTL formula φ there exists a DFA accepting all an only the strings satisfying φ. In particular, every string satisfying an sc-LTL formula must start with a prefix from a set of *good* prefixes, and the DFA indeed accepts all and only these good prefixes. Throughout this document, we will focus on sc-LTL properties.

The connection between sc-LTL properties and extended-total DFAs should be more clear now. Every word with infinite length satisfies an sc-LTL property based on its good prefix with finite length. Similarly, its associated extended total DFA accepts the same word based on its good prefix as well.

3.3 Sequential Decision Models

The main entities in a sequential decision model are the following [128]:

- A set of epochs (or time).
- A set of states.
- A set of actions.
- A set of immediate rewards or costs which is dependent on the pair of state and action.
- A set of transition probabilities which is dependent on the pair of state and action.

At every point in time, or epoch, and each system state, the decision maker has to choose an appropriate action. Applying an action incurs one or multiple costs or results in rewards. However, the outcome of an action is non-deterministic. In this context, a *policy* is considered as a lookup table which tells the decision maker what action to select in each state at any time in the future. A *decision rule* provides the decision maker with a methodology to select an action.

Applying a policy produces a sequence of rewards or costs. The final goal of probabilistic sequential decision problem is to find the optimal policy which provides the best outcome for the selected cost model before the first epoch starts. It means that the optimal policy has to be calculated before the system starts to evolve. We talk about different cost models in Section 3.3.1.

A special set of sequential decision models is known as *Markov Decision Processes (MDPs)*. In this particular subset, the state at time t only depends on the state at time $t-1$ and the action at time $t-1$. Even though minimizing a cost function or maximizing a reward function does not change the nature of the optimization problem we are facing, our emphasis is on minimizing the cost functions, and we will only discuss this case throughout this document.

3.3.1 Markov Decision Processes (MDPs)

A finite MDP is defined by a quadruple $\mathcal{M} = (S, U, P, c)$ where:

- S is a finite state space with n elements.

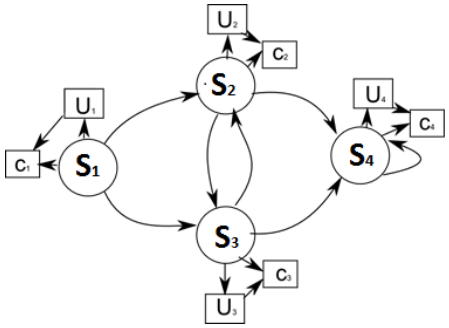


Figure 3.2: An Example of MDP: The cost function c assigns a cost value to each pair of state and action. U function provides the list of available actions at each state. The edges of the graph represent the transition function.

- for each $s \in S$ the finite set $U(s)$ is the set of actions that can be executed in state s. From these n sets, we define $U = \cup_{s \in S} U(s)$ as the set of all actions. Furthermore, from S and U we define the state/action set $K = \{(s, u) \mid s \in S, u \in U(s)\}$.

- $P : K \times S \to [0, 1]$ is a probability mass function called *transition probability*. $P(s, u, s')$ is the probability of transitioning from state s to state s' when executing action $u \in U(s)$. In the following, we write this as $P^u_{ss'}$ for brevity. Note that since P is a probability mass function, it satisfies the probability axioms. In particular $\sum_{s'} P(s, u, s') = 1$ for each $(s, u) \in K$.

- $c : K \to \mathbb{R}_{\geq 0}$ is a non-negative cost function. $c(s, u)$ is the cost incurred when executing action u while being at state s. It is not a requirement of MDPs to have $c(s, u) \geq 0$. However, we make this assumption to have simpler discussions.

A graphical illustration of an MDP is shown in Figure 3.2. In general, there are four popular cost models in solving MDPs as following:

1. *Finite horizon*: The simplest way to use an MDP is finite horizon where the system evolves for a limitted time, T, only. In this case, the goal of solvers is to optimize the following expected value: $\min_{u_t} \mathbb{E}[\sum_{t=0}^{T} c(s_t, u_t)]$, where s_t and u_t represent the state and action at time t, respectively.

2. *Infinite horizon discounted cost/reward*: Running a system for an infinite-time can easily cause the expectation to reach infinity. A discount factor $0 \leq \gamma < 1$ is proposed in order to reduce the effect of events which happen later. Thus, the optimization formula is changed to $\min_{u_t} \mathbb{E}[\lim_{T \to \infty} (\sum_{t=0}^{T} \gamma^t c(s_t, u_t))]$

3. *Infinite horizon average cost*: If the discount factor is set to 1, the time has no effect on the expected value. Therefore, the expectation with respect to average cost of actions is calculated with the following formula: $\min_{u_t} \mathbb{E}[\lim_{T \to \infty} \frac{1}{T} (\sum_{t=0}^{T} c(s_t, u_t))]$.

4. *Infinite horizon total cost*: One of the most popular uses of MDPs considers the case where the discount factor is 1, and the total cost of reaching a target is needed. To ensure that the expectation does not reach infinite, the underlying MDP has to be absorbing. In absorbing MDPs, the accrued cost turns to zero after reaching the absorbing state. We will further elaborate it later. In this case, the optimization function is $\min_{u_t} \mathbb{E}[\lim_{T\to\infty}(\sum_{t=0}^{T} c(s_t, u_t))]$.

Although, almost all of robotic applications happen in a finite period of time, finite-horizon MDPs are not popular approaches in planning a robot. The main reason lies in finding the optimal policy. Achieving an optimal solution for finite-time horizon problems is often more difficult than finding that for infinite-time horizon problems. Finite-time horizon problems add more complexities to the solvers by introducing dependencies between optimal actions and the time. Therefore, the actions at the far end of the horizon have higher influences. Moreover, if the time length changes, the optimal actions change significantly [142]. Similarly, the accrued cost of applying an action usually does not depend on when the action is taken. This also shows a preference of undiscounted approached to discounted MDPs. Last but not least, robotic applications are mostly interested in the total cost of a plan such as total path length or total energy consumption along a path. Therefore, infinite horizon total cost MDPs are widely used in robotic application, and it is also the focus of this thesis.

A deterministic policy is a function $\pi : S \to U$, associating to each state s an action in $U(s)$. A finite MDP $\mathcal{M}$ and a policy π induces a stochastic process over the set of states S. In the following, we use the symbol S_i for the random variable representing the state at time i obtained starting from an initial state $s_0 \in S$ and repeatedly applying π. It is well known that for the most commonly used cost criteria (e.g., finite horizon, discounted infinite horizon) deterministic policies are optimal [19]. In this document, we focus on the *infinite horizon total cost* models defined as

$$c(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} c(S_t, \pi(S_t))\right] \tag{3.2}$$

where the expectation is taken with respect to the probability distribution over the set of realizations of the stochastic process S_i induced by π. It is evident from Eq. (3.2) that without additional hypotheses the total cost may in general be infinite. To assure that the total cost is always a finite value, the MDP has to be absorbing or satisfy the following requirements. The state space S can be partitioned into two subsets S' and M such that:

1. $\forall s \in M \ : \ c(s, u) = 0$;
2. $P^{u}_{ss'} = 0$ for each $s \in M$, $u \in U(s)$, and $s' \in S'$;
3. for each $s \in S$ there exists one state $s' \in M$ and a policy π such that in the Markov chain associated with policy π state s' is accessible from s.

The first two conditions establish that no more cost is accrued in M and that once the state enters M it remains there. The last condition implies the existence of a deterministic Markovian policy leading to M. Collectively, these conditions establish that there exists at least one Markovian policy

π' such that $c(\pi')$ as defined in Eq. (3.2) is finite. This guarantees that the infinite horizon total cost problem admits a solution. Note that while in general M may consist of more than one state, since we assumed $c(s, u) = 0$ for each state in M, one can without loss of generality assume that M consists of a single state. This simplifies the formulation of the last condition, where accessibility is then requested from each state in S' to the only state in M. In the remainder of this document, we will assume that these three conditions hold. When these conditions hold, solving an infinite horizon total cost MDP requires to determine the policy minimizing the cost, i.e., to determine

$$\pi^* = \arg\min_{\pi} c(\pi).$$

As for the other cost criteria, the optimal policy for this problem is a deterministic policy.

3.3.2 Constrained Markov Decision Processes (CMDPs)

In an MDP, a single cost $c(s, u)$ is incurred every time an action is executed. When multiple costs are defined, a CMDP approach can instead be used. In CMDPs one determines a policy minimizing one cost function while putting constraints on the others. Formally, a finite CMDP is defined by the tuple $C = (S, U, P, c_i, \beta)$ where the definitions of S, U and P are as in MDP and the extensions are as following:

- c_i: $K \to \mathbb{R}_{\geq 0}$, $i = 0, \ldots, n$ are $n + 1$ cost functions. When action u is executed in state s, each of the costs $c_i(s, u)$ is incurred. We refer to c_0 as the *primary* cost function which has to be minimized and the others as *additional* cost functions that must be bounded. Therefore, we require n constants, $B_1, \ldots, B_n$ to define the upper bound on each of the additional cost functions. Sometimes the upper bounds are also included in the definition of CMDPs.

- β is a probability mass distribution over S, i.e., $\beta(s_j)$ is the probability that the initial state of the CMDP is s_j.

Executing action u at state s in a CMDP incurs each of the costs $c_0(s, u), \ldots, c_n(s, u)$. For each of them, different cost criteria could be defined. Extending the framework we just introduced, we will consider infinite horizon total costs for all these cost functions. As for the MDP case, it will be necessary to introduce an absorbing property to ensure that these costs are well defined. The definition of an absorbing CMDP mirrors the definition of an absorbing MDP with the additional requirement that $c_i(s, a) = 0$ for each $s \in M$. For an absorbing CMDP and policy π the following $n + 1$ total costs are then defined:

$$c_i(\pi, \beta) = \mathbb{E}\left[\sum_{t=0}^{\infty} c_i(S_t, \pi(S_t))\right] \tag{3.3}$$

where the expectation is now taken with respect to both the policy π and the initial distribution β. This additional parameter is needed because the solution of a CMDP in general depends on the initial distribution of states. The CMDP problem asks to determine a policy π^* solving the following

constrained optimization problem:

$$\pi^* = \arg\min_{\pi} c_0(\pi, \beta) \tag{3.4}$$
$$\text{s.t. } c_i(\pi, \beta) \leq B_i \quad 1 \leq i \leq n.$$

Although MDPs and CMDPs share many traits in their definitions, some important differences emerge when computing the optimal policies. First, the optimal policy for a CMDP is in general randomized, whereas optimal policies for MDPs are deterministic. Moreover, because the costs in Eq. (3.3) depend on the initial distribution β, the optimal policy too depends on it, whereas the optimal policy for an MDP is independent from the initial state distribution.

As explained in [5], CMDPs can be solved by using a Lagrangian approach. This helps to convert a constrained control problem into an equivalent minmax non-constrained control problem. The approach solves the problem in Eq. (3.4) by adding a Lagrangian multiplier per additional cost while every Lagrangian multiplier results in a separate policy. Then the optimal randomized policy of a CMDP is computed as a mix policy of multiple optimal pure policies for all the Lagrangian multipliers. Therefore, if there exists m constraints, there will be $m + 1$ policies represented by π^i where $0 \leq i \leq m$. The optimal policy π^* is the combination of all of these policies that randomly selects among them [55]. Assuming there is only one constraint, π^* chooses π^0 with the probability of q where $0 \leq q \leq 1$ and chooses π^1 with probability of $1 - q$. It should be noted that there is at most one state where $\pi^0(s) \neq \pi^1(s)$ [55]. A Similar rule is applied to more than one constraint as well. We refer the reader to [55, 79, 91] for comprehensive discussions about this topic.

The optimal policy for a CMDP is determined by solving a constrained linear program based on so-called occupancy measures. Let $K' = \{(s, u) \mid s \in S', u \in U(s)\}$ be the state-action space restricted to the non absorbing states, and let $\rho(s, u)$ be a set of optimization variables associated to each element of K'. The optimization problem in Eq. (3.4) has a solution if and only if the following constrained linear program is feasible:

$$\begin{aligned}
&\min_{\rho} \sum_{(s,u)\in K'} \rho(s,u) c_0(s,u) \\
&\text{s.t.} \sum_{(s,u)\in K'} \rho(s,u) c_i(s,u) \leq B_i \;\; 1 \leq i \leq n \\
&\qquad \sum_{(s',u)\in K'} \rho(s',u)(\delta_s(s') - P^u_{s's}) = \beta(s) \;\; \forall s \in S' \\
&\qquad \rho(s,u) \geq 0 \;\; \forall (s,u) \in K'
\end{aligned} \tag{3.5}$$

where $\delta_s(s') = 1$ when $s = s'$ and 0 otherwise. If the linear program is feasible, then the optimal solution $\rho(s, u)$ induces an optimal randomized policy π^* defined as

$$\pi^*(s,u) = \frac{\rho(s,u)}{\sum_{u\in U(s)} \rho(s,u)} \quad s \in S', u \in U(s) \tag{3.6}$$

where $\pi^*(s, u)$ is the probability of executing action u while in state x. The reader will note that this definition allows for randomized policies, as we formerly stated. The policy is not defined for states

M, but this is irrelevant because no more costs are accrued when the state is in M, and the state cannot leave it once it enters.

While the CMDP formalism allows to tackle many different practical problems, one of the main limitations is given by the number of variables that may emerge while solving the associated linear program. In fact, one may easily end up with problem instances with tens of thousands of variables. In a fully static scenario this could sometimes be acceptable as the policy could be computed off-line upfront. However, when the environment changes one may be required to recompute the policy on the fly, and the computational requirements could become difficult to accommodate within a tightly timed control loop.

3.4 Labeled Constrained Markov Decision Processes (LCMDPs)

To connect the classic theory of CMDPs with properties expressed in sc-LTL, we extend the definition of CMDPs introducing atomic propositions tracking which properties are verified during an execution.

An LCMDP is defined by the tuple $\mathcal{L} = (S, \beta, U, c_i, P, \Pi, L)$ where S, β, U, c_i, P are as in the CMDP definition and:

- Π is a finite set of atomic propositions;
- $L\colon S \to 2^{\Pi}$ is a labeling function assigning to each state the subset of atomic propositions holding in that state.

Informally speaking, LCMDPs are defined to assign a label to each state in order to verify which atomic propositions are true at each state. In terms of costs, actions, transition probabilities and initial states there is no difference between CMDPs and LCMDPs. The most important addition of LCMDPs is the ability to generate words from trajectories. Given a start state $s_0 \in S$ and a policy π, by repeatedly applying the actions defined by π a stochastic process $\Omega = s_0, u_0, s_1, u_1, \dots$ is induced, with $u_i \in U(s_i)$. The trajectory Ω generates an infinite length word in $L(\Omega) = L(s_0)L(s_1)\dots$ where $L(\Omega) \in (2^{\Pi})^{\omega}$ that is obtained by concatenating the sequence of symbols which are the outputs of labeling function.

In this document, we only focus on absorbing LCMDPs where the conditions for an LCMDP to be absorbing is the same as in CMDPs. Policies are randomized as in CMDPs, and calculated the same way. Similarly, the optimal policy is generated by following the same linear program approach. For sake of brevity, we do not repeat all the conditions and approaches again here. The additional labeling function does not influence any calculation, and only helps to keep traces of trajectories.

Note that we label the states in the absorbing set M with $\mathscr{G}$, or $L(s) = \mathscr{G},\ \forall s \in M$. Since every trajectory ends at the absorbing state set and remains there forever, all of the generated words have an infinite repetition of $\mathscr{G}$ at the end. At this point, it is clear why we defined $\mathscr{G}$-ending infinite length words in Definition 9. Informally speaking, robots operate in non-deterministic environments which is modeled by CMDPs. Every trajectory starts from an initial state, enters the set of absorbing states and stays there forever. Such trajectories generate words that are obtained from the labeling function where every word is composed of two parts:

- Finite prefix: This part contains the path from initial state until reaching the absorbing set.
- Infinite $\mathscr{G}$-ending suffix: After entering the absorbing set, $\mathscr{G}$ is concatenated infinite times to the word of the trajectory.

In order for a trajectory to satisfy an sc-LTL property, its associated extended total DFA must accept the $\mathscr{G}$-ending word which is generated by the labeling function of such trajectory. This part will make a bridge between this Section and Section 3.2.

Chapter 4

Planning with Hierarchical Constrained Markov Decision Processes

4.1 Introduction

Planning in non-deterministic environments is a challenging problem. The robot takes an action to move to a target point, but may end up at a different location. To overcome this problem, several planners have been introduced such as Markov Decision Processes (MDPs) [19], Partially Observable MDPs (POMDPs) [136] or Constrained MDPs (CMDPs) [5]. These planners include the existing uncertainties during the planning phase, and are able to choose the optimal solution under such conditions. MDPs have been widely used in different applications to tackle planning under uncertainty in fully observable environments. Here, we point out to few advantages of MDPs. First, different problems can be easily modeled with MDPs. Second, there are numerous solvers for MDPs in discrete [85] or continuous [29] domains. Third, various cost functions can be easily applied to different MDP models.

Despite the advantages of using MDPs, they also have some limitations. In practical applications one is often interested in performance measures combining multiple criteria; e.g. finding the shortest path subject to a bound on the risk, or, alternatively, finding the path with the lowest risk subject to a bound on the length. These practically relevant objectives can hardly be accommodated using MDPs only, but can be naturally formulated as CMDP problems. In fact, the CMDP framework allows to consider an arbitrary number of additional constraints. One can then for example seek the shortest path subject to different bounds to various types of risk.

Throughout this chapter, we use *risk* as primary cost function which should be minimized. However, our proposed planner is not tied to any special application and risk is chosen as an example metric only. Therefore, risk can be easily replaced with any other measure such as fuel consumption. We also do not aim for defining a risk factor, because there are multiple standard documents proposing such measurements as in [1, 2, 81].

This chapter consolidates three of our previously published works [63, 65, 66]. In Section 4.2, we propose a planner by using CMDPs. This planner solves CMDPs hierarchically by introducing a simple rectangular fixed-size partitioning method. Then we show an improved version of the hierarchical CMDP model in Section 4.3 which uses adjustable partition sizes. This enhance-

ment increases the flexibility of the simple fixed-size partitioning by guaranteeing the feasibility of the solutions. Subsequently, we present our experimental results in Section 4.4 to compare non-hierarchical vs hierarchical (fixed-size and variable-size) CMDPs in solving motion planning problems. Various experiments were done in Matlab, robot simulation scenarios and also on real robots. Finally, we conclude the chapter in Section 4.5.

4.2 Planning with Hierarchical CMDPs using Fixed-Size Partitioning

One of the well known limitations of MDPs and CMDPs, as discussed in Chapter 3, relies on their computational complexity. CMDPs, in particular, require the solution of linear program whose size may quickly become unmanageable (see section 4.4 for some numbers). For this reason, hierarchical approaches can be a viable alternative, as long as there is a clear understanding that this will lead to suboptimal solutions. Hierarchical methods in the area of MDPs have been practiced in the past mainly with the objective of determining policies that can be reused, see e.g., [80]. We instead pursue hierarchical solutions with the objective of reducing the computational time. In particular, we are interested in reducing the computational effort with the eventual objective of enabling reactive replanning when changes in the environment are detected by the robot. Changes may for example concern a rearrangement of the obstacles on the floor plan, or a person moving from one area to another, with the consequent necessity to update the associated risk.

Our approach is based on the idea of aggregate states (see [18], Vol 1, pg. 321), an idea that has been proposed for MDPs, but needs to be adapted for CMDPs. In the following, we make the assumption that the action set $U(s)$ is identical for all states. This is consistent with the application scenario we present in the following and can also be imposed in the general case.[1] In the interest of simplicity, we furthermore assume that $\beta(s) = 1$ for just one state and is 0 for all other states, and we assume there is just one goal state. These two states will be indicated as I (initial) and G (goal).

The general idea is as follows. We start from a given CMDP with a large state space and build a hierarchical version through state aggregation. The aggregate CMDP is then solved and a policy over the aggregate states is determined. In order to map the high level policy back into actions that can be executed in the original CMDP, a smaller CMDP is solved for each aggregate state traversed by the high level policy. Since action execution is stochastic, one cannot anticipate the precise sequence of aggregate states that will be traversed when the policy is followed. Therefore the solution of the smaller CMDPs is interleaved with execution, because the precise sequence of traversed aggregate states cannot be predicted upfront (see Figure 4.1).

In the following, quantities with the subscript A refer to the aggregate CMDP. First, to define the state space S_A of the aggregate CMDP we partition the state space S into a set of m aggregate states $A_1, A_2, \ldots, A_m$. Each aggregate state represents a set of states in the original CMDP. The problem of how to split S into S_A is a long standing issue. A rule of thumb is to group together states with similar costs, when this is possible. In our application, targeting path planning in planar environments this calls to grouping together states that are nearby in a metric sense. Indeed, nearby

[1] One can define a new set $U = \bigcup_s U(s)$ and assign it to each state. If through this assignment a state s receives a new action a' not in its original action set, one adds a loop transition probability $P_{ss}^{a'} = 1$ and large costs $c_i(s, a')$ so that a' is never included in the optimal policy.

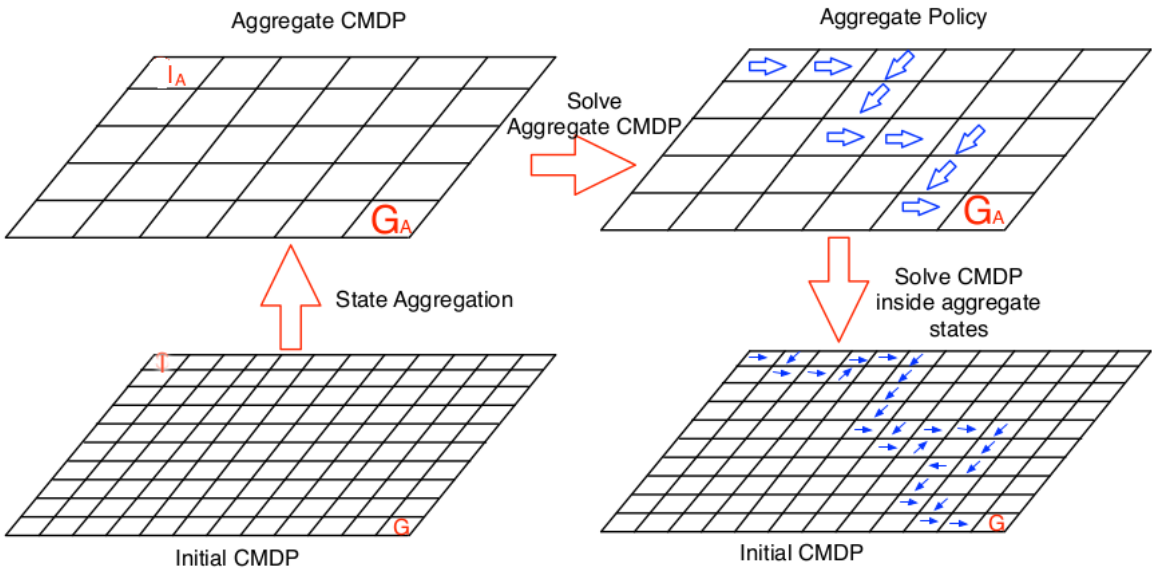


Figure 4.1: Large CMDPs can be solved through aggregation. Note that for CMDPs both the start and the goal state (I and G in the figure) need to be specified, as opposed to the MDP case where an optimal policy is specific by the goal state only. It is evident that the hierarchical model can be applied in more than too layers too.

states have usually comparable distances to the goal location. The problem of how to partition S using different criteria is described in Section 4.3 where variable size partitions are proposed. Two quantities called *aggregation* and *disaggregation* probabilities are defined to represent the aggregation. The aggregation probability $w_{ij} = 1$ if $s_j \in A_t$ and 0 otherwise. The disaggregation probability is $q_{si} = \frac{1}{|A_s|}$ if $s_i \in A_s$ and 0 otherwise. These definitions correspond to the so-called *hard assignment*, i.e., every state in the original CMDP is assigned to one and only one aggregate state. *Soft* assignments are possible too, with states possibly belonging to multiple aggregate states, but this extension is left to future investigation. The initial and goal states I and G in the original CMDP obviously induce initial (or start) and goal states I_A and G_A in the aggregate CMDP because I and G belong to one and only aggregate state.

Next, it is necessary to define the action set, transition probabilities, and the costs $c_{i,A}$ for the aggregate CMDP. Since we assumed all states in the original CMDP had the same action set U, it follows that we can assume all states in the aggregate CMDP also have the same action set, and this is also U. Every action executed in a certain aggregate state A_i has an intended effect in terms of state transition (e.g., this can be the state with towards which there is the highest transition probability). The function NextState(A_t, u), used in the implementation of algorithm 4 returns such state. Moreover, we define the frontier between aggregate-state A_t and A_n as the set of states in A_n that can be reached in one transition from one state in A_t. Transition probabilities between state A_s and A_t in the aggregate CMDP are then defined as follows:

$$P_{A_s A_t}(u) = \frac{1}{|A_s|} \sum_{i \in A_s} \sum_{j \in A_t} P_{ij}(u).$$

Similar definitions can be given for the costs:

$$c_{i,A}(A_s, u) = \frac{1}{|A_s|} \sum_{i \in A_s} c_i(x_i, u).$$

Algorithm 4 presents an algorithmic sketch of the idea we described.

Algorithm 1: Algorithmic Sketch

```
   Data: CMDP = (S, U, P, c_i, β)
 1 Build CMDP (S_A, U_A, P_a, c_{i,A}, β);
 2 Solve aggregate CMDP ;
 3 Extract optimal aggregate policy π*_A (Eq. 3.4);
 4 s ← β;
 5 while s ≠ G do
 6     Determine state A_t for which w_xt = 1;
 7     if A_t = A_G then
 8         GoalSet ← {G}
 9     else
10         u ← π*_A(A_t);
11         A_n ← NextState(A_t, u);
12         GoalSet ← Frontier(A_t, A_n)
13     π_L ←SolveLocalCMDP(s, GoalSet);
14     repeat
15         Follow policy π_L and update s
16     until s reaches GoalSet or exits A_t;
```

The algorithm starts creating and solving the aggregate CMDP (line 2 and 3). Then, it extracts the optimal policy for the aggregate CMDP and starts to follow it (while loop). Inside the loop, the CMDPs for the aggregate states are solved *on demand*, i.e., the CMDP for an aggregate state A_i is created and solved only when the state enters the corresponding aggregate state. Note also that, because of the unpredictability in the motion of the robot, it is possible that when the robot tries moving from macro state A_i to A_j it instead reaches a different macro state A_k (think for example to the case of a robot moving along the boundary of two macro states). This event does not create a problem. A policy over the macro states is preliminarily computed, so even if the state deviates from the intended trajectory the corresponding high-level policy is always available. Next, CMDPs for the macro states are solved on the fly, so an unpredicted transition into a different macro state can be dealt with by the algorithm. Note also that the algorithm does not solve CMDPs for the macro states not visited during the main while loop. The interleaved planning and execution strategy is essential to limit the computational cost.

4.3 Planning with Hierarchical CMDPs using Variable-Size Partitioning

As explained in Section 4.2, hierarchical solvers for CMDPs provide an estimations for the optimal solution while saving time. However, the proposed method has the following drawbacks:

- The major problem with fixed-size partitioning is the possibility of losing a feasible solution (See Figure 4.10). In other words, there are some cases that the non-hierarchical solver is able to find a viable solution, but the hierarchical solver fails to calculate that because of how the partitions are calculated. However, this drawback can be resolved by adjusting the size of partitions. By looking at Figure 4.10, we notice that fixed partitioning may succeed to find a solution if partitions have bigger sizes. But larger partition sizes may increase the computational times as analyzed in Figure 4.13. According to our experimental analysis in this chart, too large or too small partitions increase the time to solve the problem. If we try to intelligently choose the size of partitions in order to achieve a desired resolution, we may need to create a single partition from the entire state space in the worst case which does not save us any time.

- If the partitions are selected only based on their geometrical proximity of states while cost functions are independent from this factor, the solution of the hierarchical solvers could be quite far from the optimal solution. Therefore, it is preferable to take cost functions into account when considering partitioning.

- If the state space has a non-standard shape (e.g. non-rectangular in this case), the fixed-size partitioning method will suffer extensively to adapt to the abnormal structure. This could cause the final solution to be farther from optimal.

In this section we present a new type of partitioning that is capable of addressing all the aforementioned problems. The new hierarchical CMDP, which will be referred to as *HCMDP* from now on, is based on defining partitions with variable sizes and some similarities in terms of cost functions. The major differences between HCMDP and fixed-size partitioning consists on how to calculate partitions, cost functions, transition probabilities and action sets.

By clustering the state space of the original CMDP one creates a new CMDP with fewer states, computes a policy for the smaller instance, and then utilizes the policy for the smaller CMDP to derive a policy for the original CMDP (see Figure 4.2).

These ideas can be formalized as follows. Let $C = (\mathcal{S}, U, P, c_i, \beta)$ be a CMDP. From C we extract an HCMDP $C_H = (\mathcal{S}_H, U_H, P_H, c_{i,H}, \beta_H)$ with $|\mathcal{S}_H| << |\mathcal{S}|$. Next, we compute the optimal policy π^*_H for C_H and we use it to extract a policy π' for C. While π^*_H is optimal for C_H, in general π' will not be optimal C and in fact one can anticipate that the gap between π' and π^* (the optimal policy for C) widens as the size of the clusters shrinks, and viceversa narrows as the size of the clusters increases. However, our tenet is that the loss in optimality is limited and compensated by the significant gain in computational efficiency. Moreover, the hierarchical solution is also less memory intensive, i.e., by solving multiple smaller problems it requires less memory than the original non-hierarchical problem. Various subproblems must be tackled to build C_H from C. In the following subsections we identify all of them and provide pertinent solutions.

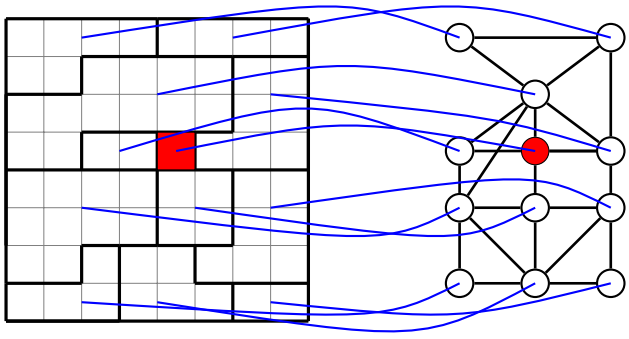


Figure 4.2: The figure illustrates the idea behind the HCMDP approach. The left side shows a CMDP whose numerous states are represented by the small grid cells. On top of these states, irregularly shaped clusters are defined, as indicated by the thick line partitions. Clusters define a new hierarchical CMDP shown in the right panel. Neighboring relationships between clusters define connectivity in the hierarchical CMDP. Absorbing states in the original CMDP (red grid cell) are mapped into absorbing states in the hierarchical CMDP (red circle). Edges in the clustered graph are added between macro states sharing a border in the original state space.

4.3.1 Clustering

The first problem is how to compute $\mathcal{S}_H$ from $\mathcal{S}$. Elements of $\mathcal{S}_H$ will be in the following called *macro states* or *clusters*. There are evidently multiple ways to determine these clusters. The method we propose is based upon both an intuition and a formal requirement. The intuition is that clusters should be composed of *similar* states, where similarity is measured in terms of the cost function c. The formal requirement is that clusters must preserve the connectivity property we define in the following.

Definition 11 *Let $C = (\mathcal{S}, U, P, c_i, \beta)$ be a CMDP, and x, y be two states in $\mathcal{S}$. We say that x is connected to y and we write $x \rightsquigarrow y$ if there exists a sequence of states $s_1, \ldots, s_m \in \mathcal{S}$ and actions $u_1, \ldots u_{m-1} \in U$ such that $s_1 = x$, $s_m = y$, and $P^{u_i}_{s_i, s_{i+1}} > 0$ for $1 \leq i \leq m-1$.*

Based on this definition we introduce the concept of connectivity preserving HCMDP.

Definition 12 *Let $C = (\mathcal{S}, U, P, c_i, \beta)$ be a CMDP and let $C_H = (\mathcal{S}_H, U_H, P_H, c_{i,H}, \beta_H)$ be an HCMDP for C. We say that C_H preserves the connectivity of C if the two following conditions hold:*

1. *$\mathcal{S}_H$ is a partition of $\mathcal{S}$;*
2. *For $z \in \mathcal{S}'$ and $y \in M$, let Z_H and Y_H be the states in $\mathcal{S}_H$ such that $z \in Z_H$ and $y \in Y_H$. If $z \rightsquigarrow y$, then $Z_H \rightsquigarrow Y_H$.*

Where M and $\mathcal{S}'$ were formerly introduced in Section 3.3.1 as requirements for absorbing MDPs. Note that because of the requirement that $\mathcal{S}_H$ is a partition of $\mathcal{S}$, the states Z_H and Y_H

are unique and the second condition is then well posed. Moreover, observe that the connectivity preserving property is a function of C_H as a whole and not just $\mathcal{S}_H$, because connectivity depends not only on states, but also on transition probabilities. To define the clustering algorithm, for $S \subset \mathcal{S}$, $x \in \mathcal{S}$, and $u \in U(x)$ it is convenient to define the following operators:

$$\text{Pre}(S) = \{x \in \mathcal{S} \mid \exists u \in U(x) \wedge \exists y \in S \wedge P^u_{xy} > 0\}$$
$$\text{Post}(x, u) = \{Y \in \mathcal{S}_H \mid \exists y \in Y \wedge P^u_{xy} > 0\}$$
$$\text{Post}(S) = \{Y \in \mathcal{S}_H \mid \exists y \in Y \wedge \exists x \in S \wedge P^u_{xy} > 0\}.$$

Note that $\text{Pre}(S)$ is a subset of states of $\mathcal{S}$, whereas $\text{Post}(x, u)$ and $\text{Post}(S)$ are sets of macrostates of $\mathcal{S}_H$. Starting from these operators, Algorithm 2 shows our proposed approach to clustering. The method depends on two parameters, namely the maximum cluster size MS and a constant δ used to define whether two states have similar cost. A single cluster is initially created with the set M (line 1). Then the algorithm loops until all states in $\mathcal{S}$ have been assigned to a state in $\mathcal{S}_H$ (loop starting at line 2). At each iteration, each of the existing clusters is examined (line 3) and the states that can reach one of the clusters in one step with non-zero probability are identified (set S at line 4). For each element in S (loop starting at line 5) every possible action is evaluated (line 6) to determine to which macrostate the action would lead to (line 7). The state is assigned to a cluster H (line 10) only if two conditions are simultaneously verified, i.e., cluster H includes less than MS states, and the average cost of cluster $C(H)$ is similar to the cost $c(s, u_s)$ incurred to move from s into H (test at line 9). As soon as a state is assigned to a cluster, it is not considered anymore (line 11). If state s cannot be assigned to any cluster (line 12), then a new cluster including just s is created (line 13) and added to the set of clusters (line 14). At the end, an optional merging step described in the following is performed (line 15) and the set of clusters is returned (line 16.)

Note that because of the third hypothesis we formerly made (accessibility of M from each state), Algorithm 2 terminates. This is because the hypothesis ensures that at every iteration at least one state is assigned to a cluster (either an existing one or a new one). Therefore, after a finite number of iterations the clustering algorithm terminates. As previously stated, the clusters determined by Algorithm 2 depend on MS and δ and their impact on the clusters should now be clear. A too large value for δ leads to cluster with heterogeneous values for c, while small values for δ lead to the creation of many small clusters. Similar conclusions can be made for MS. In the experimental section we will analyze the sensitivity to MS. The negative effect of a poorly chosen δ can be mitigated by MS (when δ is too large) or by the merging algorithm described next (when δ is too small).

Merging. The cluster size influences the performance of the algorithm, because too large clusters diminish the effectiveness of the algorithm, whereas too many small clusters induce large approximations. The MS parameter is used to counter the first problem, i.e., the creation of too large clusters. The merging algorithm presented here counters the other problem, i.e., the presence of too many small clusters. To this end, small clusters are merged together, where the definition of *small cluster* is defined by a new parameter mS (minimum size). Algorithm 3 shows how merging is performed. All clusters are examined (line 2), and if they are smaller than mS (line 3) they are combined with one of their neighboring clusters. To this end, all neighbors are determined (line 4) and the one with the most similar cost is picked (line 4). If the combined size does not exceed the

Algorithm 2: Clustering Algorithm

Data: $\mathcal{S}, U, P$
Result: $\mathcal{S}_H$: Set of macro-states

```
1  S_H ← {M} ;
2  while There exist states not assigned to any cluster do
3      foreach C ∈ S_H do
4          S ← Pre(C) ∩(S \ ∪S_H) ;
5          foreach s ∈ S do
6              for u_s ∈ U(s) do
7                  Y_s ← Post(s, u_s);
8                  for H ∈ Y_s do
9                      if c_0(H) ≈ c_0(s, u_s) ∧ |H| < MS then
10                         assign s to cluster H;
11                         break out of the two inner for loops;
12             if s not assigned yet then
13                 create new cluster M_s = {s} and assign s to it;
14                 add M_s to S_H;
15 S_H ← merge(S_H) ;
16 return S_H;
```

upper limit MS (line 6) then the clusters are merged (line 7). The process is repeated until no more clusters can be merged (loop starting at line 1). The algorithm terminates returning the new set of merged macrostates (line 9.) Note that the merging algorithm does not merge the macro state M.

4.3.2 Hierarchical Action Set

Once the set of macro states $\mathcal{S}_H$ has been created, the set of actions U_H for the HCMDP easily follows. For $Y \in \mathcal{S}_H$, the action set $U_H(Y)$ is identified considering all macro states that can be reached in one step from the states in C, i.e.,

$$U_H(Y) = \{Z \in \mathcal{S}_H \mid \exists y \in Y \wedge z \in Z \wedge u \in U(y) \wedge P^u_{yz} > 0\}. \tag{4.1}$$

It is important to note that according to this definition $U_H(Y)$ is generated starting from the original action set U, but its elements are actions that are not in U. In particular, each action in $U_H(Y)$ is a macrostate in $\mathcal{S}_H$.

4.3.3 Transition probabilities, costs, and initial probability distribution

The definition of an HCMDP is completed by the definition of the transition probabilities (P_H), costs ($c_{i,H}$), bounds ($B_{i,H}$), and initial probability distribution β_H. With the objective of creating a method applicable independently from the structure of the underlying state space, we opt for a

Algorithm 3: Merge Algorithm

Data: $\mathcal{S}_H$: Set of all macro-states excluding the macro state $\{M\}$
Result: $\mathcal{S}_H$: New set of macro-states

```
repeat
    for M ∈ S_H do
        if |M| < mS then
            M_adj = Post(M) ;
            M_c ← arg min_{M_c ∈ M_adj} |c_0(M_c) − c_0(M)|;
            if |M_c| + |M_adj| < MS then
                merge M and M_c and update S_H;
until no more states are merged;
return S_H;
```

Monte Carlo approach whereby transition probabilities and costs are estimated through sampling.[2]

Given $M_1, M_2 \in \mathcal{S}_H$ and $M_3 \in U(M_1)$, we aim at estimating $P^{M_3}_{M_1,M_2}$. It is worthwhile recalling that although M_3 is a macro state too, in this context it is an action applied from macro state M_1 and it shall be interpreted as the action aiming at moving from macro state M_1 to macro state M_3. As per the definition of $U_H(M_1)$ given in Eq. (4.1), $P^{M_3}_{M_1,M_2}$ is non zero only if M_1 and M_2 are adjacent, where adjacency is defined by Eq.(4.1). The probability estimation method is as follows. Let $B_{M_1,M_3} = \{y \in M_3 \mid \exists x \in M_1 \wedge \exists u \in U(x) \wedge P^u_{xy} > 0\}$, i.e., the boundary between M_1 and M_3. We then build a graph $G = (V, E)$ where $V = M_1 \cup B$ and an edge is added between vertices $v_1, v_2 \in V$ whenever $P^u_{v_1,v_2} > 0$ for some $u \in U(v_1)$. Then, for each state $x \in M_1$ we compute the shortest path from x to B, where shortest is defined in terms of number of edges. These paths define a policy[3] π over M_1 to move from M_1 to M_3. Note that this policy is not optimal. Next, using a uniform probability mass function over the states in M_1 we randomly select one state $x \in M_1$ and simulate policy π. Following π the state eventually leaves M_1 to enter either M_3 or some other macro state. Through repeated executions, these policy simulations allow to estimate $P^{M_3}_{M_1,M_j}$ for each M_j.

A similar graph is built to estimate $c_{i,H}(M_1, M_3)$. We simulate a path from a random state $x \in M_1$ toward a state $y \in B_{M_1,M_3}$ by following the aforementioned policy. Then, $c_{i,H}(M_1, M_3)$ is estimated by taking the average of costs accrued during the simulation. For the hierarchical bounds $B_{i,H}$ we use the same costs in the original CMDP. Finally, β_H is built from β by adding up the probabilities of the states within each macro state, i.e., for each macro state Y_H we define $\beta_H(Y_H) = \sum_{x \in Y_H} \beta(x)$.

4.3.4 Hierarchical planning

The hierarchical planner operates in two steps and Algorithm 4 illustrates them. As mentioned in section 3.3.2, upper bound values can be included in the definition of CMDPs. Therefore, in this algorithm, we used the upper bounds in the definitions in order to provide estimations for

[2]To the best of our knowledge no method has been proposed to analytically estimate costs and probabilities.

[3]The set of paths define a policy because for each vertex they identify an edge to traverse along the shortest path, and by construction this edge is associated with an action.

them. The algorithm takes as input a CMDP C, a starting state s and a set of goal states M. Since we assumed that a starting state $s \in \mathcal{S}$ is provided as input, the initial distribution β is zero everywhere except in s. An HCMDP C_H is built as per our previous description (line 1), and an optimal policy is determined (loop starting at line 3). At each iteration the linear program is solved (line 4), and an optimal policy π^*_H for C_H is computed. If a policy cannot be found (line 5), the bounds $B_{i,H}$ are increased until the associated linear program is solved (line 6). In particular, each of the $B_{i,H}$ is increased by a fixed percentage. In our examples presented later on, the increment is 10% and there is an obvious tradeoff between selecting a larger or a smaller increase, since a smaller increase will give a sharper bound, but multiple small increases may be necessary before a satisfactory bound is determined and the problem solved. The rationale behind increasing the bounds, is that the hierarchical problem may be unsolvable due to the approximations induced in computing the costs through Monte Carlo sampling (in particular, overestimation of the costs $c_{i,H}$). It shall however be noted that these increased bounds may end up being higher than the original ones. While we believe that for simple problem settings it may be possible to derive quantitative bounds, this appear to be not trivial for the general case, and is left for future work. After a solution is found (line 8), the optimal policy for the hierarchical CMDP is determined (line 9). This concludes the first stage of the algorithm. In the second stage, starting at line 10, a policy for the original CMDP is extracted solving a sequence of smaller CMDPs to move from one macro state to the next. To be specific, the smaller CMDPs are built as follows. Assume the state in the original CMDP C is $s \notin M$ (otherwise the problem is already solved and the second loop at line 11 terminates.) This state belongs to exactly one macro state $Y_H \in \mathcal{S}_H$ because $\mathcal{S}_H$ is a partition of $\mathcal{S}$ (line 12.) The optimal policy π^*_H defines the action $Z_H \in U(Y_H)$ to execute, i.e., it identifies the next macro state to move into (line 14.) By construction, this macro state shares a boundary with Y_H, i.e., there exist a set of states $GoalSet \subset \mathcal{S}$ in the original CMDP reachable from some state in Y_H with a single transition (line 15). Therefore, from the original CMDP we extract a smaller CMDP whose state set is $Y_X \cup GoalSet$ and we compute a policy π_L to reach $GoalSet$ (line 16). Once the policy is computed, it is executed (line 18) until the state exists Y_H (either entering $GoalSet$ or another macro state). From there, the optimal policy π^*_H provides a new action to execute and the cycle terminates when the goal set M is reached.

We conclude this section stating two theorems and their proofs.

Theorem 1 *Let $C = (\mathcal{S}, U, P, c_i, \beta)$ be a CMDP and let $C_H = (\mathcal{S}_H, U_H, P_H, c_{i,H}, \beta_H)$ be an HCMDP built from C with the method described in this section. If the number of samples tends to infinity, then C_H preserves the connectivity of C.*

Proof. Definition 12 establishes two conditions for saying that an HCMDP preserves connectivity. The first requires that $\mathcal{S}_H$ is a partition of $\mathcal{S}$. Algorithm 2 never considers a state twice, i.e., once a state has been assigned to a cluster it will not be considered again for assignment (line 4). Moreover, the main loop ensures that all states in $\mathcal{S}$ are assigned to a cluster. Therefore, $\mathcal{S}_H$ is a partition of $\mathcal{S}$.

We next turn to the second condition. Let $z \in \mathcal{S}'$ and $y \in M$ be two states such that $z \rightsquigarrow y$. By definition this means that there exists a sequence of states $\mathcal{S} = s_1, s_2, \ldots, s_n$ such that for each each $1 \leq i \leq n-1$ $P^{u_i}_{s_i,s_{i+1}}$ for some $u_i \in U(s_i)$ and $s_1 = y$ and $s_n = z$. Since $\mathcal{S}_H$ is a partition of $\mathcal{S}$, this sequence of states is associated with a sequence of macrostates $Z_H = S_1 \ldots S_n = Y_H$ such

Algorithm 4: HCMDP Planning

Data: CMDP $C = (\mathcal{S}, U, P, c_i, \beta, B_i)$, s, M

Build HCMDP $C_H = (\mathcal{S}_H, U_H, P_H, c_{i,H}, \beta_H, B_{i,H})$;
$Solved \leftarrow false$;
while *Not Solved* **do**
    Solve LP associated with HCMDP;
    if *LP unfeasible* **then**
        Increase each bound $B_{i,H}$ of $\Delta B_{i,H} > 0$;
    else
        $Solved \leftarrow true$;
Extract optimal aggregate policy π^*_H (Eq. 3.6);
$x \leftarrow s$;
while $x \notin M$ **do**
    Determine state Y_H containing s ;
    if $Y_H \neq M$ **then**
        $Z_H \leftarrow \pi^*_H(Y_t)$;
        $GoalSet \leftarrow$ Frontier(Y_H, Z_H);
        $\pi_L \leftarrow$SolveLocalCMDP$(x, GoalSet)$;
        repeat
            Follow policy π_L and update x ;
        until *x exits* Y_H;

that $s_i \in S_i$ for each i. Note that in general there could be some repeated elements in the sequence of macrostates. Let $S_1, \dots S_k$ ($k \leq n$) be the sequence obtained removing subsequences of repeated macrostates.[4] First note that this sequence includes at least two elements. This is true because we started assuming $y \notin M$ while $z \in M$. According to Algorithm 2 all and only the states in M are mapped to an individual macrostate (line 1), so y cannot be in the same macrostate as z. Next, consider two successive elements in the sequence of macrostates, say S_i and S_{i+1}. By construction, there exist two successive states in $\mathcal{S}$, say s_j and s_{j+1}, such that $s_j \in S_i$ and $s_{j+1} \in S_{i+1}$. Since these two states are part of $\mathcal{S}$, there exists one input $u_j \in U(s_j)$ such that $P^{u_j}_{s_j,s_{j+1}} > 0$. As per Eq.(4.1), this implies that an action S_{j+1} is added to the set of actions $U(S_j)$. Next, consider the method described in subsection 4.3.3, and in particular the definition of the boundary B between two macro states. It follows that $s_{j+1} \in B_{S_i,S_{i+1}}$. The algorithm further continues computing the *shortest path* between each state in S_i and B, where the shortest path is computed over the induced graph G. For s_i the path trivially consists of a single edge to s_{i+1} (or some other vertex in B that is also one hop away from s_i.) Next, the algorithm randomly selects one vertex from S_j using a uniform distribution and executes the policy to reach B. Let m be the total number of Monte Carlo samples generated. Then the probability that the estimate of $P^{S_{i+1}}_{S_i,S_i+1} = 0$ is bounded from above by

$$(1-\gamma)^{k_1}(1-P^{u_j}_{s_j,s_{j+1}})^{k_2}$$

where $\gamma = \frac{1}{S_i}$, k_1 is the number of times s_j was not sampled and k_2 is the number of times s_j was sampled ($k_1 + k_2 = m, k_{1,2} \geq 0$). This proves that as the total number of samples m grows, the estimate for $P^{S_{i+1}}_{S_i,S_i+1}$ will be eventually be positive. This reasoning can be repeated for each couple of successive macro states, thus showing that $Z_H \rightsquigarrow Y_H$, and this concludes the proof.

Theorem 2 *Let $C = (\mathcal{S}, U, P, c_i, \beta), s, M$ be the input to Algorithm 4. Then, as the number of samples tends to infinity, Algorithm 4 builds a solvable HCMDP.*

Proof. We start observing that Algorithm 4 builds and solves a sequence of HCMDPs. Each is a CMDP with a suitable set of parameters and at every iteration the constrained linear program given in Eq. (3.5) is solved. Theorem 1 guarantees that state M_H is accessible from every macrostate, and therefore there exists at least one policy π' for which $c(\pi')$ is finite. Let us next consider the inequality constraints in Eq. (3.5). If the linear program is not feasible, then each bound $B_{i,H}$ is increased by $\Delta B_{i,H}$ (line 6.) By construction, all additional costs $c_{i,H}(x,u) \geq 0$ for each state/action pair (x,u). Let $n_s = |\mathcal{K}'_H|$ be the number of state/action pairs in the HCMDP, $c_{max} = \max_{(x,u)\in K'_H}\{c_{i,H}(x,u)\}$, $1 \leq \mathrm{i} \leq \mathrm{n}$, the largest among the additional costs, and $B_{min} = \min\{\Delta B_{i,H}\}$ the smallest among the increments in line 6. Therefore after at most $\lceil \frac{n_s d_{max}}{B_{min}} \rceil$ iterations all inequality constraints become feasible.

Note that the second theorem states the existence of a policy, but does not state anything about its optimality. Theorems 1 and 2 only guarantee asymptotic convergence in the number of samples, but do not provide indications on the rate of convergence as a function of the number of

[4]This means that if $S_i = S_{i+1}$ we remove the latter from the sequence and we reiterate this step until $S_i \neq S_{i+1}$ for all symbols left in the sequence.

samples. This is consistent with results in existing literature for motion planning, like the probabilistic roadmap method [95], rapidly exploring random trees [112], and the more recent optimal method RRT* [94]. In some instances, knowledge of certain parameters characterizing the environment, or making simplifying assumptions about its structure, it may be possible to derive relationships between the number of samples and the rate of convergence. However, for more general cases this is an open question and subject to future work.

4.4 Experimental Evaluation

In this section we compare the two hierarchical models and the non-hierarchical CMDPs. We provided simulation and real robot experiments to show the advantages and disadvantages of each method.

4.4.1 Planning with Hierarchical CMDPs using Fixed-Size Partitioning

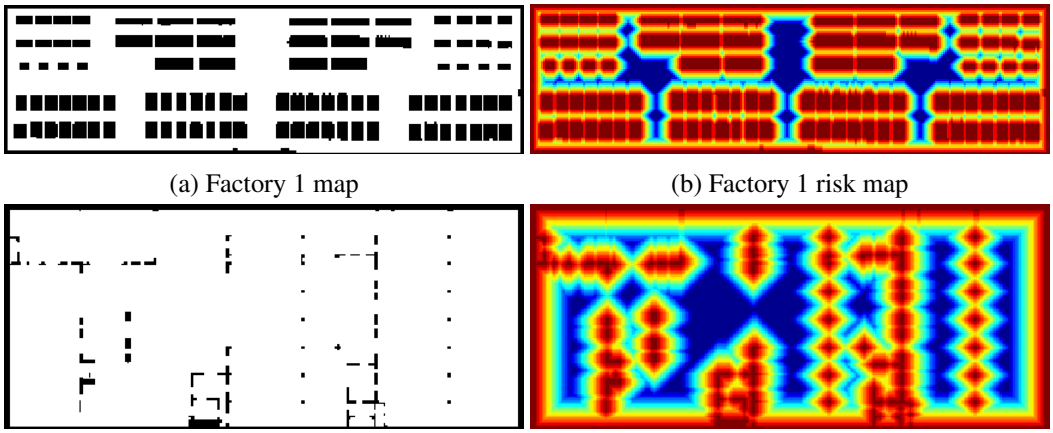
4.4.1.1 Setup

In this section, we present some experiments outlining the advantages and limitations of the hierarchical decomposition. Figure 4.3 shows the floorplans of two factories retrieved from the web, with white pixels encoding free space and black pixels indicating obstacles. The same picture shows two corresponding risk maps. For sake of simplicity and ease of visualization, we assumed that risk is a function of the state only, and risk is defined as distance from the closest obstacle. Both maps are divided into equally sized cells. The first maps consist of 78×272 cells, whereas the second one includes 111×270 cells. We assume that the robot fits inside one of the cells, and furthermore assume 4-connectivity, i.e., from every cell the robot can move up, down, left, right, assuming the neighboring cell is not occupied by an obstacle. These numbers define the number of variables in the linear program given in Eq. 3.5 and shows why a hierarchical approach is necessary. For the first case (Factory 1), the linear program has 54542 variables, whereas for the second case (Factory 2) it has 114862 variables. Evidently, the time needed to setup and solve these large optimization problems prevents rapid replanning when needed, thus showing the necessity to go for a hierarchical approach. In the following, we assume the objective is to compute the policy giving the expected shortest path between a couple of start/goal points, while keeping the expected risk below a given threshold. Risk is here defined as the sum of the risks accrued throughout the path, i.e., it is the sum of the risks of the traversed cells.

The stochastic motion model is defined as follows. When the robot executes action u from state x trying to reach state y (say moving up on the grid), it succeeds with probability 0.8 and fails with probability 0.2. When it fails, it may remain in x or move to any of the free cells adjacent to x and different from y (all with equal probability).

4.4.1.2 Results

We contrast the performance of the hierarchical approach with the non hierarchical one. In particular, we compare the time spent to compute a solution, the average path length and the average risk. In order to compare the time spent to compute a solution, for the non hierarchical approach



(a) Factory 1 map (b) Factory 1 risk map

(c) Factory 2 map (d) Factory 2 risk map

Figure 4.3: The two test environment used for experimental evaluation. Both environments are factory floorplans retrieved from the web. The left subfigures show the structure of the environment, whereas the right subfigures show the associated risk maps (warmer colors indicate riskier states).

we consider the time spent solving the linear program given in Eq. 3.5. For the hierarchical method we give the sum of the time spent solving all instances of linear programs Eq. 3.5 (see Algorithm 4). In both cases the linear program is solved using Matlab's builtin command to solve linear programs (`linprog` – in particular we use the interior-point method). We decided to compare the time spent solving instances of the linear program because this is where most of the time is spent, and the remaining time (e.g., setting up the matrices for the linear program) strongly depends on the characteristics of the used programming language (Matlab, in this case). Moreover, with respect to timing, emphasis should be given to the ratios between hierarchical and non-hierarchical, and not to the absolute values. For each environment we consider five different couples of start/goal points and we test both the hierarchical and non-hierarchical methods on the same problem instances. For the case Factory 1, the aggregate CMDP is obtained splitting the original grid into a coarser 6×8 grid with , whereas Factory 2 was divided into a 9×10 grid.

Table 4.1 and 4.2 show the performance of the non-hierarchical method for the five test cases in the two environments (time is given in seconds). Table 4.3 and 4.4 show instead the performance of the hierarchical method for the five test cases in the two environments. Statistics for path length and risk are computed on the basis of 100 repeated trials.

Test Case	Time	Avg. Length	Std Length	Avg. Risk	Std. Risk
Case 1	32.7	415.7	18.8	2108.8	187.4
Case 2	28.2	103.3	10.7	266.6	104.5
Case 3	368.5	106.7	8.6	534.7	63.5
Case 4	47.8	381.7	17.0	1020.4	80.4
Case 5	359.2	143.4	10.6	505.3	126.1

Table 4.1: Performance of the non-hierarchical method on Factory 1.

Test Case	Time	Avg. Length	Std Length	Avg. Risk	Std. Risk
Case 1	198.7	484.7	22.2	2019.7	198.7
Case 2	180.1	356.9	17.5	2158.4	435.1
Case 3	223.0	156.1	11.9	525.1	63.2
Case 4	218.0	176.2	13.4	429.2	203.6
Case 5	228.5	265.4	13.5	1117.0	139.6

Table 4.2: Performance of the non-hierarchical method on Factory 2.

Test Case	Avg. Time	Avg. Length	Std Length	Avg. Risk	Std Risk
Case 1	5.5	450.9	20.5	1963.7	284.8
Case 2	1.46	107.5	11.6	283.2	85.4
Case 3	2.5	142.7	10.4	646.1	98.9
Case 4	4.6	395.2	17.9	1396.4	163.8
Case 5	2.2	148.1	18.5	428.0	121.7

Table 4.3: Performance of the hierarchical method on Factory 1.

Test Case	Avg. Time	Avg. Length	Std Length	Avg. Risk	Std Risk
Case 1	6.8	444.5	34.1	5392.0	834.0
Case 2	7.1	319.5	29.5	2690.2	838.6
Case 3	3.03	142.1	17.9	1165.1	156.4
Case 4	4.3	153.0	14.4	476.6	52.5
Case 5	6.0	240.8	25.2	1611.3	181.6

Table 4.4: Performance of the hierarchical method on Factory 2.

This preliminary round of simulations, although limited, allows to draw some interesting insights. First, the speedup in terms of computational time is large and varies from a factor of 5 to a factor of 150. For the case of Factory 1 (Table 4.1 and Table 4.3), the average length obtained with the hierarchical method is comparable with the length obtained with the non hierarchical solution. Similarly, average risks are almost the same (in Case 5 the hierarchical method performs even better, but this is due to averaging a limited number of cases). For the Factory 2 case (Table 4.2 and Table 4.4), things instead are different. The average length of the path is again almost the same. But in a couple of cases (Case 1 and Case 3) the hierarchical method incurs in significantly higher risk, whereas the other cases are comparable. It is of course expected that the hierarchical method will in general generate paths with higher length and higher risk, although the significant variations deserve additional investigation. In particular, it will be interesting to observe these trends while varying the number of grid cells in the hierarchical models, for example increasing their number. In general, however, in 8 out of 10 cases the hierarchical method produces solutions of comparable quality in terms of path length and risk, but spending just a fraction of the computational time.

4.4.2 Planning with Hierarchical CMDPs using Variable-Size Partitioning

In this section we investigate the viability of the method we presented as HCMDP. In particular, we aim at assessing the trade off between the computational gains we obtain using a hierarchical solution and the gap between the optimal non-hierarchical solution and the sub-optimal hierarchical solution. Moreover, we also experimentally determine the sensitivity to the parameters influencing the clustering method we presented. The algorithm is evaluated in three different scenarios, namely a matlab based simulation, a Gazebo based simulation, and an implementation on a Pioneer P3AT robot operating in an indoor environment. Each of the three different setups offer complementary features.

4.4.2.1 Matlab Simulations

We start our study with numerous matlab simulations allowing to quickly evaluate how the performance changes as the parameters are modified. Three different methods are compared. The first is the baseline non-hierarchical CMDP solver producing optimal policies solving the linear program given in Eq. (3.5). This method will be indicated as NH (non-hierarchical) in the tables and charts. The second is the method we presented in section 4.2 that relies on a fixed structure for partitioning of the state space. In particular, for mobile robot tasks it relies on a state space clustering leading to a rectangular decomposition of the environment (see Figure 4.10). In tables and charts this method will be indicated as "Fixed". The third is the method we propose in section 4.3. All methods are subject to the same costs and transition probabilities.

All examples we will consider in the following feature two costs and then cannot be considered with a plain MDP solver. The first cost c_0 is the cumulative risk accrued along a trajectory based on a preassigned risk map. The second cost c_1 is the Euclidean path length. Note that although we consider two costs only, all solving approaches can handle an arbitrary number of costs. The first test environment is shown in Figure 5.3 and it models an outdoor environment in which the robot has to navigate between selected couples of start/goal locations. The heat map shows the risk map, with higher risk locations indicated by warmer colors. The sub figure on the right shows two different solutions obtained imposing different constraints on the path length. Note that the white trajectory traverses areas with higher risk because it is subject to a more stringent bound on path length, so it cannot afford to take the same detour generated by the red trajectory. This map is referred to as "terrain map" in the following. Figure 4.5 shows the clusters created by Algorithm 2 when processing this map.

The second test environment is depicted in Figure 4.6 and it is based on an indoor factory-like environment. The associated risk map is shown as well, where the darker areas with highest risk are associated with the obstacles. This map is referred to as "maze" in the following. In both maps the cost c_1 (Euclidean distance) is set to 1 for every state/action pair.

Both environments are defined over a grid in which we assume 4-connectivity and we correspondingly define four actions for each state when possible.[5] Each action succeeds with probability 0.8. When the action fails, any of the nearby neighbors can be reached with equal probability. Three

[5]For states close the boundary or to an obstacle, the action set is adjusted by removing actions that would violate these constraints.

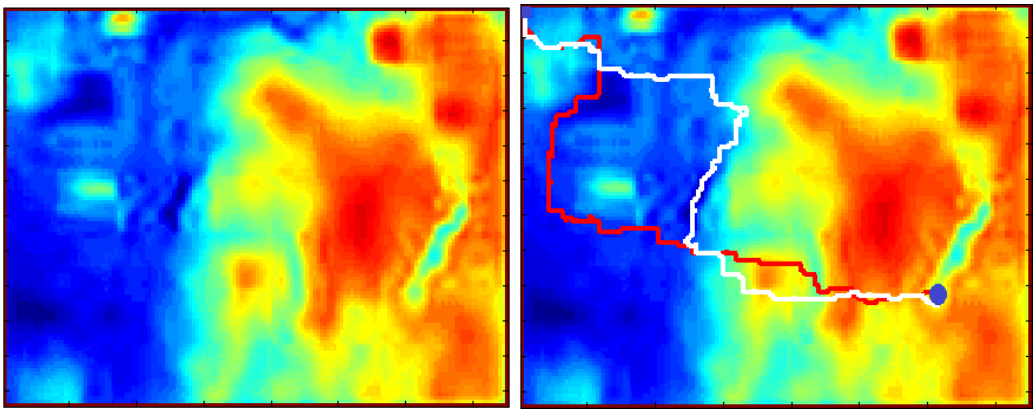


Figure 4.4: Sample terrain map where warmer colors represent high risk areas to be avoided. The right picture shows two different solutions obtained with different constraints.

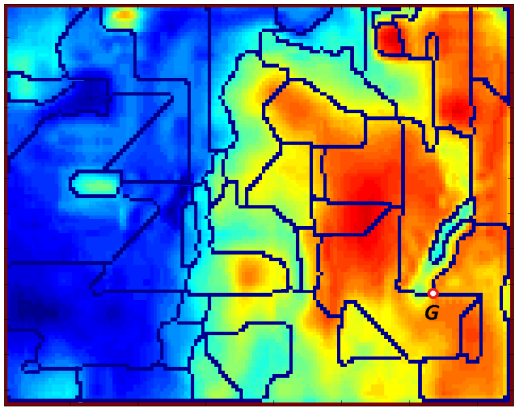


Figure 4.5: Macro states created by Algorithm 2 on the terrain map environment. The goal state is evidenced by the letter G and a red circle as it constitutes a standalone cluster.

different performance measures are used in the following to compare the different algorithms. The first is the time spent to determine the policy. The second is the expected value of the risk objective function c_0, and the last is the value for the additional constrained cost c_1 (path length). All algorithms are coded in matlab and rely on the built in `linprog` function to solve the linear programs. To ensure a fair comparison, since the hierarchical methods solve multiple instances of smaller linear programs, the displayed time is the cumulative sum of the time spent to solve all linear programs needed to compute a solution. The non-hierarchical method instead solves just one large linear program and the time spent to solve it is what we use for comparison.

Table 4.5 displays the time spent to solve 12 different problem instances on the terrain map environment. Each instance is defined by a different couple of start/goal locations. The top two rows display the performance of the non hierarchical and fixed resolution planner whereas the bottom twelve rows display the time spent by the HCMDP method for different combinations of its parameters. In particular, X/Y means that the maximum cluster size MS is X, and the number of

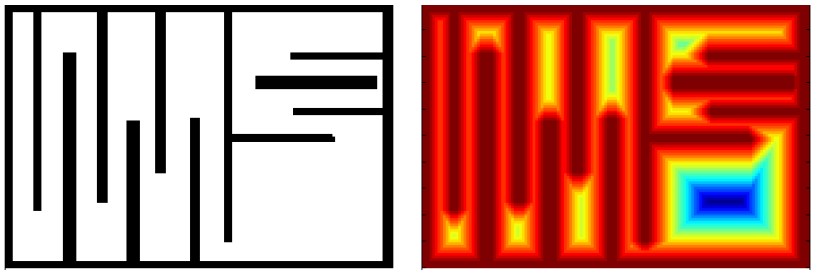


Figure 4.6: Maze map where fixed partitioning fails. The left panel shows the map whereas the right panel displays the associated risk map.

samples used to estimate probabilities and costs is $Y\%$ of the number of states in the clusters. The prefix *nm* stands for *non merged*, and indicates that the merging step at the end of Algorithm 2 was not performed. Throughout this section, the δ parameter is set as the average difference for the c_0 function between each node and all of its neighbors. The table shows that the performance difference between the non hierarchical method and the hierarchical strategies considered (Fixed and HCMDP) varies between a factor of 17 (case 3) and 1000 (case 4). Similarly, Table 4.6 shows the standard deviations for the same data. It should be noted that every experiment was repeated 100 times. In Figure 4.7, we show how 100 trials is enough for such an experiment.

Alg	1	2	3	4	5	6	7	8	9	10	11	12
NH	107	94	54	1303	613	43	860	86	743	151	59	106
Fixed	3.19	2.91	2.2	1.3	0.8	1.79	2.3	1.5	2.5	0.89	1.89	1.7
100/10	4.59	5.49	3.89	1.72	2.72	2.20	2.43	1.82	3.21	1.43	3.39	2.55
100/30	7.04	5.68	4.02	1.82	1.79	7.33	2.48	3.02	3.59	1.79	3.06	3.59
100/50	6.17	8.45	3.79	1.70	1.65	7.48	2.37	5.48	5.14	2.58	4.34	4.21
nm/100/50	5.3	3.48	3.15	1.76	1.56	2.54	11.8	2.87	2.51	1.92	2.93	2.76
200/10	4.9	3.53	4.06	2.57	1.08	3.47	2.58	3.53	2.9	1.02	3.44	4.57
200/30	3.89	3.21	3.74	2.84	1.49	2.72	2.98	4.01	3.72	1.28	2.84	3.81
200/50	4.75	4.50	4.45	2.38	1.64	2.36	2.48	3.78	5.21	1.21	3.51	3.77
nm/200/50	4.03	3.44	3.02	1.31	1.76	2.63	1.84	4.83	4.07	1.49	2.45	2.20
400/10	7.17	6.02	5.25	5.64	1.71	8.12	3.17	3.76	3.31	2.26	3.45	3.85
400/30	6.41	6.83	5.05	2.52	2.31	8.06	5.43	2.90	4.08	2.42	7.73	4.86
400/50	6.80	5.65	5.41	5.14	2.48	8.09	3.06	2.56	4.39	2.3	3.47	3.89
nm/400/50	5.32	3.98	9.72	6.28	7.16	11.0	4.9	5.92	4.5	2.32	3.83	4.62

Table 4.5: Time spent (seconds) by the various algorithms for the terrain map on 12 different instances.

Table 4.5 also shows that, as expected, HCMDP is slower than the fixed method since it performs a more sophisticated partitioning, but the performance gap is in general modest, though it widens as expected as the number of samples in the Monte Carlo estimation grows.

Another interesting point to assess the performance gain is by looking at the number and

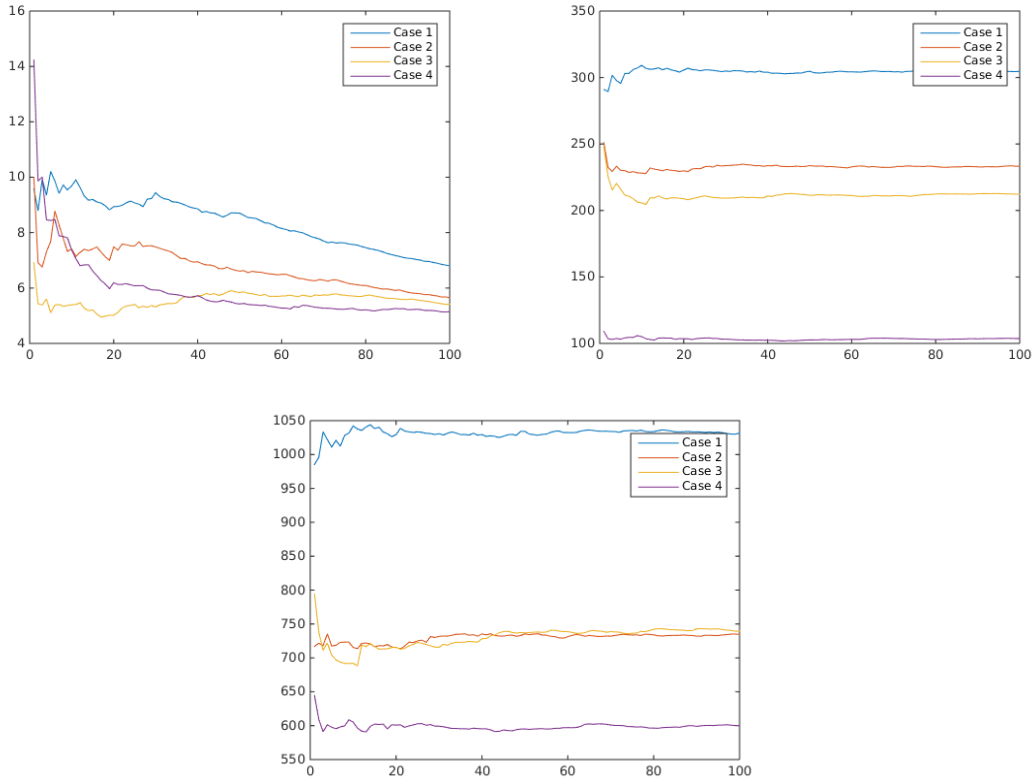


Figure 4.7: The effect of number of trials on average calculations. We computed cumulative average of time, risk and path length over trail number for the first four cases in Table 4.5 were the maximum macro-state size is set to 400 and sample rate is 50%. The top left picture shows the time, top right is the path length and the bottom picture represents the total risk. As seen at the beginning of trails some fluctuations are visible. However, after passing 50th trial the average becomes quite stable. These plots confirm that 100 trials can reflect the true average value of all of these metrics.

Alg	1	2	3	4	5	6	7	8	9	10	11	12
Fixed	0.51	0.42	0.2	0.15	0.6	0.15	0.45	0.3	0.21	0.29	0.18	0.05
100/10	0.81	0.84	0.75	0.35	0.1	0.41	0.41	0.13	0.87	0.13	0.75	0.08
100/30	1.2	0.05	0.8	0.29	0.08	1.05	0.38	0.91	0.21	0.1	0.21	1.1
100/50	0.5	0.36	0.7	0.27	0.2	0.81	0.81	0.75	0.99	0.31	0.7	0.74
nm/100/50	1.85	1.23	0.54	0.95	0.54	0.61	2.51	0.93	0.75	0.71	1.01	0.51
200/10	0.35	0.75	1.01	0.51	0.1	1.05	0.25	0.31	0.38	0.41	0.31	0.38
200/30	0.75	0.65	0.78	0.18	0.03	0.71	0.31	0.61	0.17	0.02	0.08	0.74
200/50	0.95	0.81	0.81	0.25	0.01	0.28	0.51	0.58	0.54	0.5	0.17	0.85
nm/200/50	1.32	2.02	1.92	0.51	0.48	0.35	0.42	1.8	1.03	0.85	0.62	0.96
400/10	0.9	0.51	1.51	0.71	0.8	2.01	0.64	0.3	0.81	0.25	0.6	0.4
400/30	1.12	0.32	1.28	0.43	0.57	1.81	0.2	0.11	0.25	0.11	1.31	1.3
400/50	1.35	0.71	1.02	0.5	0.71	0.28	0.17	0.05	0.36	0.37	0.14	0.81
nm/400/50	1.1	0.98	1.96	1.87	0.99	2.2	1.3	0.85	1.68	0.68	0.85	1.01

Table 4.6: Standard deviation in timing by the various algorithms for the terrain map on 12 different instances. There is no standard deviation for non-hierarchical CMDP as its linear program runs only once.

size of the linear programs solved by the non-hierarchical and hierarchical methods. The terrain map environment generates a constrained linear program with more than 65,000 variables, and the maze environment creates instances with more than 41,000 variables. The number of variables is (roughly) given by the number of states multiplied by four, since each variable corresponds to a state/action pair (x, u) and there are four or less actions per state (states close to the boundary or to the obstacles have less than four actions.) Table 4.7 instead displays the size and number of subproblems generated by the hierarchical methods for the twelve problems considered in the terrain environment. Each row shows two numbers, with the top one being the number of variables of the largest linear program solved, and the bottom number showing the average number of problems solved. Both numbers are averaged over one hundered runs, and rounded to the closest integer. Note that the percentage of samples does not influence these numbers and is therefore not displayed. These numbers justify the large difference in time between non-hierarchical and hierarchical methods and motivate this line of research.

While the time to solve a problem instance is an important aspect, hierarchical methods would not be very useful if the performance increase comes at the cost of a inferior performance in terms of costs of the functions being optimized. Figure 4.8 and 4.9 analyze how these costs vary across the 12 different cases we considered. In particular, Figure 4.8 shows the average cost for the primary c_0 cost (risk), whereas Figure 4.9 plots the average c_1 cost (path length). In both instances averages are obtained over 100 independent runs (variances are small, and so averages are representative.)

Figure 4.8 shows that as expected the non-hierarchical method achieves in general the best performance in terms of minimizing the expected c_0 cost. In some cases (e.g., 2 and 8) some instances of the HCMDP solver obtains a better cost. This is due to the fact that if the HCMDP planner cannot solve a certain problem instance for given constraints on the c_1 cost, it will increase

Alg	1	2	3	4	5	6	7	8	9	10	11	12
100	392 10	380 8	354 8	380 4	382 3	384 5	376 6	348 4	376 5	356 3	380 5	384 6
nm/100	356 14	348 13	316 10	284 6	358 6	348 8	304 8	320 9	320 7	340 8	312 9	276 6
200	756 6	780 5	800 3	800 3	800 2	760 5	780/4 4	756 3	800 4	740 3	704 2	760 4
nm/200	740 8	680 8	656 5	632 3	740 4	556 8	724 4	712 6	680 8	644 4	720 3	716 6
400	1560 4	1524 4	1460 3	1388 3	1512 2	1560 3	1540 3	1452 2	1524 3	1512 3	1420 2	1420 3
nm/400	1320 5	1272 6	1156 4	1420 3	1260 3	1340 2	1410 5	1316 3	1220 4	1340 8	1360 6	1360 4

Table 4.7: Size of the largest linear program (top number) and number of supbroblems (bottom number) for each algorithm and problem considered in the terrain map problem.

the B bounds in an attempt to make the linear program feasible. Hence, in some instances, taking advantage of the increased bound for the c_1 cost, it is possible to lower the c_0 cost as well because the additional budget for the traversed length may give the possibility to avoid risky areas. The reader could also observe that in many instances the Fixed partitioning method works competitively with both the non hierarchical method and HCMDP. However, as it will be evidenced discussing the maze environment, this strategy is in general prone to failure, although in the terrain map this is not evident since there are no obstacles.

Figure 4.9 shows instead the average for the constrained path length cost c_1. Here we see that the variations are more contained, demonstrating that only in few instances the bound needs to be lifted for making the problem feasible.

We next consider the maze map. This environment is interesting because it shows that approaches relying on clustering methods not considering the underlying map are in general doomed to fail. Figure 4.10 shows the policy produced by the algorithm we formerly presented in section 4.2. The dotted lines show the clusters boundaries, whereas the red arrows illustrate the computed policy. The example shows that in some macro states the policy suggests transitions that cannot be executed because of an obstacle cutting through. This problem could be overcome reducing the size of the clusters, but for any fixed clustering strategy one can devise an environment leading to unfeasible policies. For this reason, adaptive clustering algorithms considering the underlying structure of the state space are needed. These include the one we presented, as well as [12].

Given that the fixed-clustering approach is not a viable solution for this environment, and having shown in the previous section that the non-hierarchical solution is largely outperformed by our method, the analysis of the maze environment is then restricted to the HCMDP algorithm. Similarly to Table 4.5, Table 4.8 show the time spent (in seconds) to compute the policy as the size of the cluster and the number of samples is varied. The table confirms that for this metric the size of the cluster is the most relevant parameter, whereas there is less sensitivity to the number of samples.

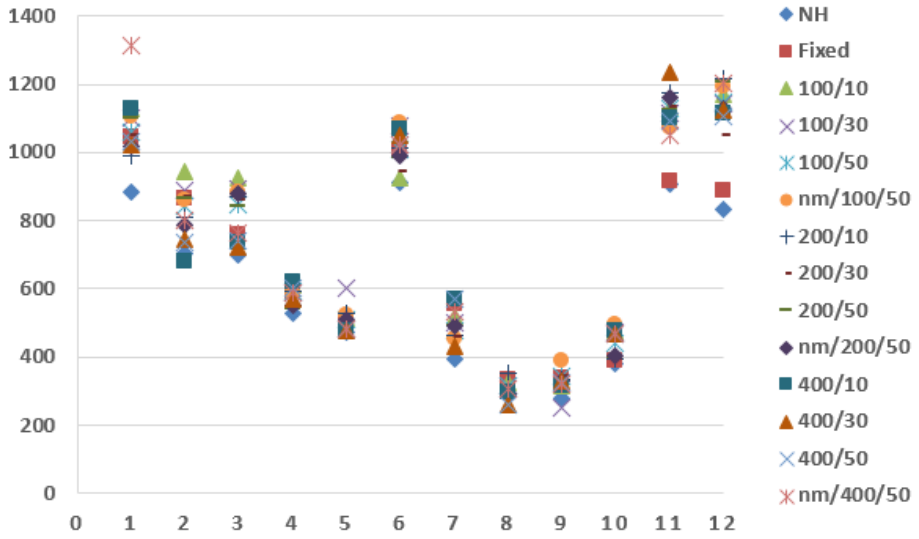


Figure 4.8: Average c_0 cost (risk) over 100 independent runs for the terrain map environment.

Similarly, Table 4.9 shows the standard deviations for the same data set.

Finally, Figures 4.11 and 4.12 show the performance for the functions c and d. The findings confirm what previously observed, i.e., that the performance loss is contained.

In the introduction we outlined that gain in speed is only one of the advantages of the proposed method. Another important aspect to consider is memory requirements and the associated feasibility. To put this aspect into context, if we double the resolution along the x and y directions for either the terrain map example or the maze example, the number of state/action variables (roughly) grows by a factor of four. Under these conditions, on the computer system we used to perform our experiments, the non-hierarchical method systematically fails because it runs out of memory, whereas the hierarchical method succeeds, since, as shown in Table 4.7 it solves significantly smaller problems.

Alg	1	2	3	4
100/10	5.63	4.64	3.41	2.22
100/30	3.08	3.89	3.20	2.10
100/50	5.69	4.54	3.76	2.54
100/70	5.69	5.22	3.92	2.44
100/90	5.73	4.80	3.78	2.45
200/10	8.52	7.95	5.05	8.48
200/30	9.89	5.44	6.97	7.76
200/50	9.08	8.88	5.66	6.74
200/70	9.62	7.51	6.37	6.28
200/90	9.07	9.11	5.84	7.31

Table 4.8: Time spent (seconds) by the various algorithms for the maze map on 4 different instances.

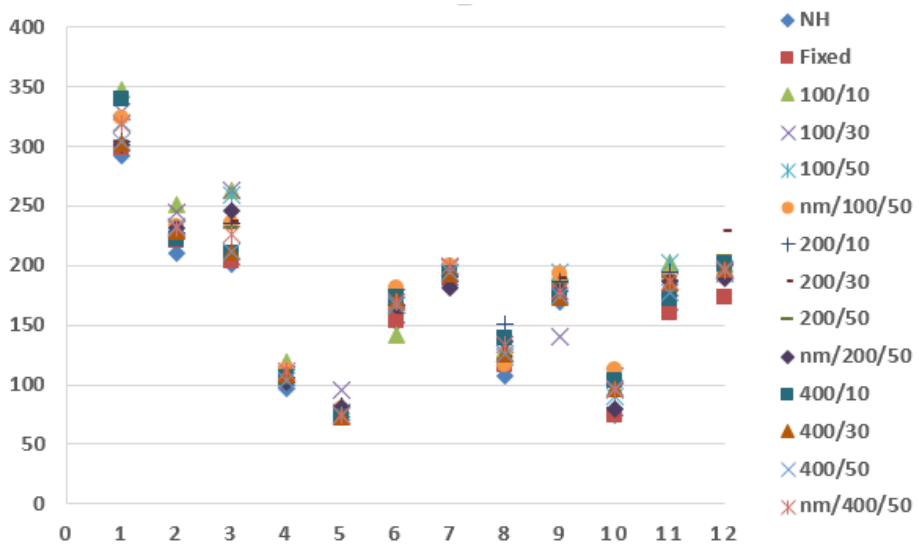


Figure 4.9: Average c_1 cost (path length) over 100 independent runs for the terrain map environment. Horizontal green bars show the value of the constraint in the original, non-hierarchical problem formulation.

One last question to be addressed is how to pick the parameters for the HCMDP planner. Looking at Table 4.5, for example, we see that the performance varies with the maximum size of the cluster (MS) and the number of samples. The second parameter has rather straightforward interpretation, since the number of samples is a fixed percentage of the number of states in the macrostate, and there is an obvious tradeoff, in the sense that more samples imply higher accuracy but require more time. The choice for the cluster size, instead, is less intuitive. With the objective of getting some experimental insights on how to select this value, we performed a series of experiments where we varied the size of the cluster and observed its impact on the computational time and the quality of the solutions. Figure 4.13 shows the results for three different problem instances in both the terrain and the maze problem. Note that since the problems have different state spaces, results are normalized expressing in both cases MS as the percentage of the number of states rather than as an absolute number. It is interesting to notice that for both problems and all cases considered, picking MS equal to 1% of the number of states in the non-hierarchical problem seems to give the best result.

Moreover, in Figures 4.14 and 4.15 we evaluate how the cluster size influences the objective function (risk) and the additional constrained costs (path length). Experimental data show that the impact of MS on these quantities is limited. Therefore, it is reasonable to conclude that setting MS equal to 1% of the size of the state space seems to be a good starting point. Of course, when domain specific knowledge is available, better choices could be made. An analytic investigation on how to select the cluster size is subject to future work.

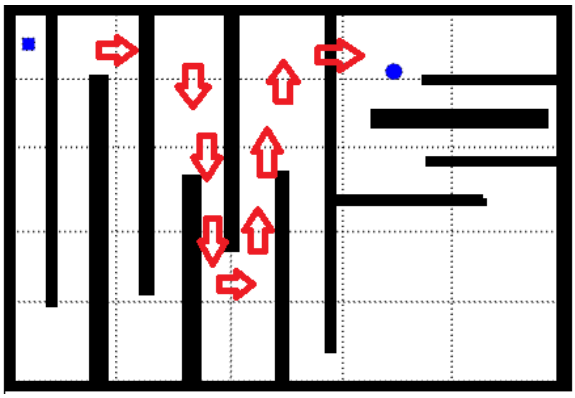


Figure 4.10: Example showing how fixed partitioning fails. Partitions are shown using dotted lines. The hierarchical CMDP induces the policy shown by the red arrows that cannot be executed due to the obstacles cutting through the macro states.

Alg	1	2	3	4
100/10	1.05	0.82	0.27	0.4
100/30	0.95	0.1	0.18	0.21
100/50	0.84	1.01	0.8	0.09
100/70	1.01	0.5	0.54	0.35
100/90	1.3	0.92	0.28	0.18
200/10	1.8	0.74	0.37	0.27
200/30	2.05	1.01	0.44	0.07
200/50	1.8	0.21	0.66	0.71
200/70	2.01	1.08	0.91	0.34
200/90	1.3	0.91	0.08	0.72

Table 4.9: Standard deviations in timing by the various algorithms for the maze map on 4 different instances.

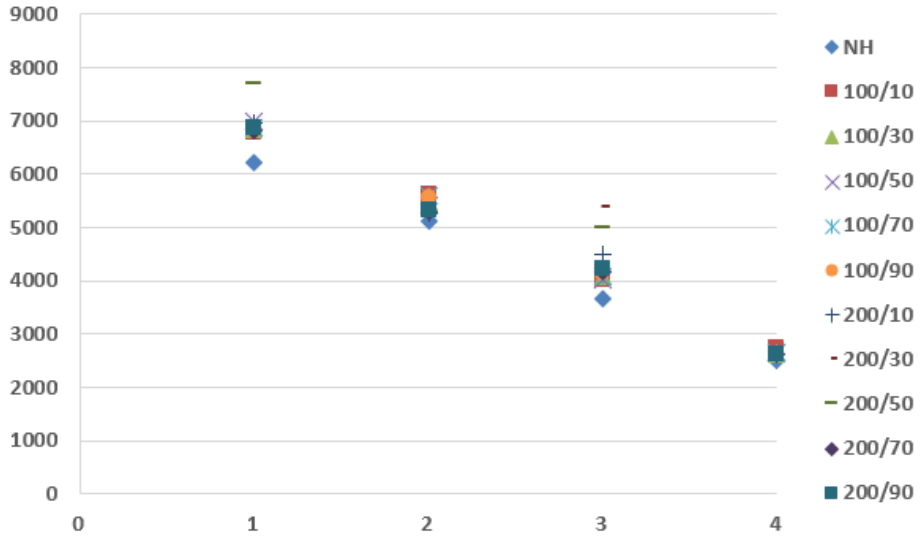


Figure 4.11: Average c cost (risk) over 100 independent runs for the maze environment.

4.4.2.2 Gazebo Simulations

With the objective of eventually validating the HCMDP planner on a real robotic platform, as an intermediate step we developed and tested the planner using ROS and the Gazebo simulator. The simulated robotic platform exactly matches the robot we use in the real world experiments, i.e., a P3AT equipped with odometry, sonar, and the SICK LMS proximity range finder (see Figure 4.21). The software architecture used both in Gazebo and on the real robot relies on standard ROS nodes. In particular, we use a particle filter for localization [142] and we therefore assume that a map of the environment is available. In both cases the map was preliminarily built driving the robot around manually and using the GMapping SLAM algorithm [76] available in ROS. Figure 4.16 shows a rendering of the simulated environment (top), the map built by the SLAM algorithm (bottom left) and the associated risk map for the primary cost c_0 (bottom right). The simulated map is $21.7m \times 20m$ and is discretized into square cells of size $0.5m \times 0.5m$. In each cell we assume the availability of four actions (up, down, right, left). This assumption is consistent with our software design, where navigation to a given target point relies on the ROS navigation stack. In particular, the ability to execute an action (say "up") is not influenced by the current orientation of the robot, and therefore the state is represented just by the cell where the robot is located (without considering its orientation). According to the framework we presented, for each state x (i.e., grid cell) the policy computed by the planners returns an action $\pi(x)$ that can be translated to a point in one of the nearby grid cells and then fed to the navigation stack to move the robot there.

An essential step to setup a planner based on an MDP approach is to estimate the transition probabilities, i.e., P^u_{xy}. To estimate these quantities, each action is executed 100 times and transition probabilities are derived by counting. To explore the sensitivity of the policies to transition probabilities, two surfaces with different friction coefficients were used. In the first case the probability of executing a motion with success is 0.66 whereas in the second case it is 0.51. When a motion

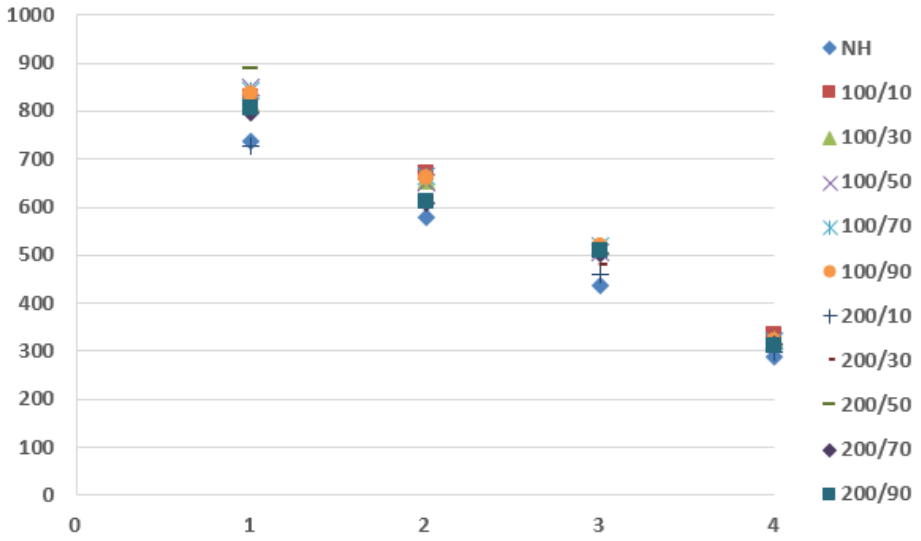


Figure 4.12: Average c_1 cost (path length) over 100 independent runs for the maze environment. Horizontal green bars show the value of the constraint in the original, non-hierarchical problem formulation.

does not succeed, a uniform probability over adjacent states (excluding the target state) is used.

Figure 4.16 (bottom left) shows four different points in the map used as start/goal locations to compute the policy. Four different cases were considered. The first starts at P_1 and ends at P_2; the second starts at P_2 and ends at P_3; the third starts at P_3 and ends at P_4; and the fourth starts at P_4 and ends at P_1. Policies were computed using three different planning strategies, namely MDP, CMDP, and HCMDP. For the MDP planner the objective is to minimize the cumulative c_0 cost according to the risk map shown in figure 4.16. MDP is included as an additional term of comparison, although it solves a simpler problem because it considers just the primary cost c_0 (risk) and ignores the additional cost c_1 (Euclidean distance). Figure 4.17 shows some examples of the resulting paths. Every test was repeated 20 times to estimate average behaviors. The leftmost panel in figure 4.17 shows that the MDP solution takes the safest path since it almost entirely lies inside the low risk areas (refer to figure 4.16). This path is however longer. This is consistent with the objective function we setup, and to the lack of constraints on path length. The middle panel shows that, due to the constraints on path length, the planner produces a path that cuts through some higher risk areas to meet the bound. Finally, the rightmost panel shows that the HCDMP planner displays a behavior that is somehow intermediate, in the sense that sometimes it favors a safer subpath, and sometimes it instead selects a shorter path. This is very evident when considering point P_2. In such case the HCMDP planner produces a path that approaches P_2 like the CMDP planner (shorter but riskier), but then leaves the point along the same path determined by the MDP planner. This is due to the fact that when leaving P_2 the hierarchical planner has to relax the constraints to achieve feasibility, and therefore it can produce a path with higher length but lower risk.

Figures 4.18 and 4.19 show the first set of results, where the HCMDP planner was tested using different values for the maximum cluster size and sampling rate. Consistently with the expectations,

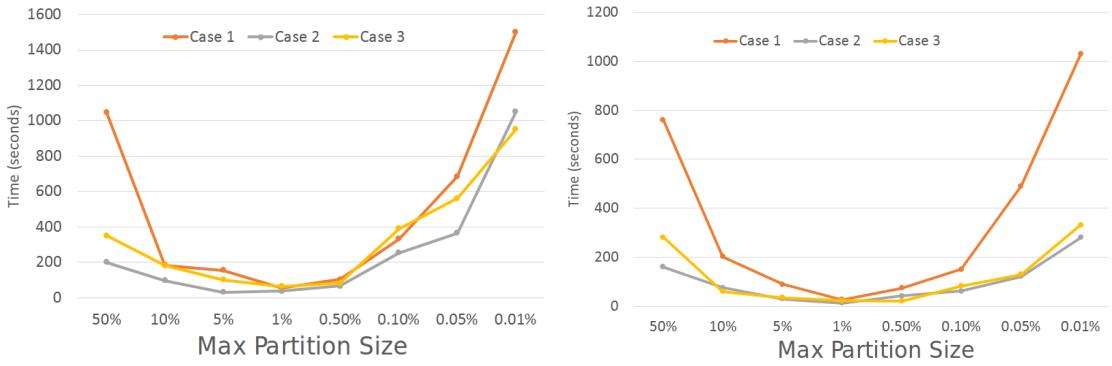


Figure 4.13: Time to compute the solution with HCMDP as a function of the cluster size MS. The left figure shows the trends for the terrain problem, and the right figure shows the trends for the maze problem.

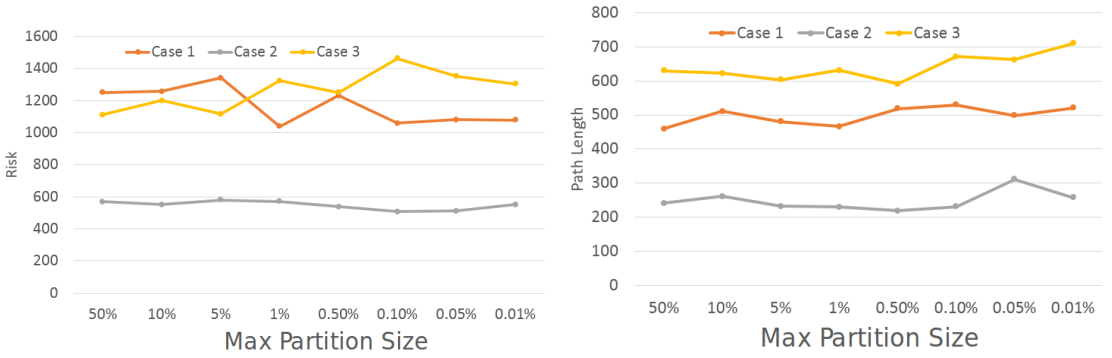


Figure 4.14: Variations of the primary objective function (risk) as the size of the clusters vary.

Figure 4.18 shows that the MDP achieves the lowest risk, whereas it is again shown that tuning the parameters it is possible to obtain similar performance between CMDP and HCMDP. Figure 4.19 also confirms our previous observations with regard to variations in the secondary cost. Based on these two sets of results, it appears that the most convenient setup for the HCMDP planner is obtained when the maximum cluster size is set to 200 and the sampling rate is 90%. Therefore in the following we will just consider these parameters.

Figure 4.20 shows an additional set of results where the friction coefficient is set to be much lower and then the robot is subject to more slippage when moving (and then more uncertainty). The figure confirms the conclusions we formerly derived, thus showing insensitivity to the transition probability rates.

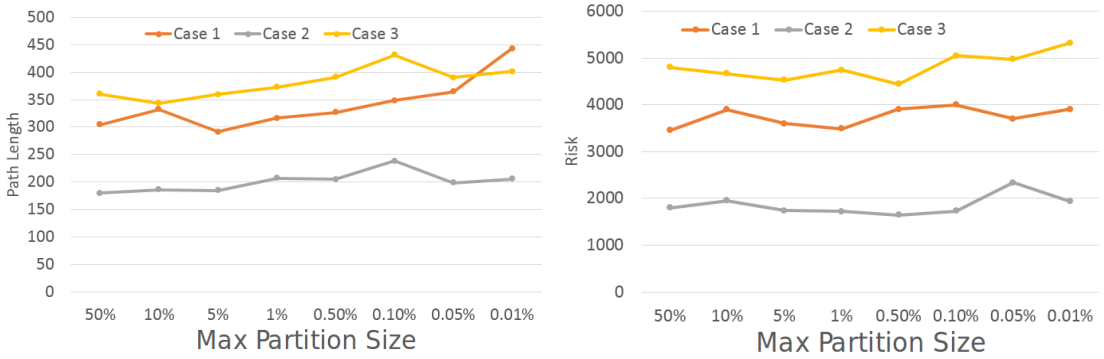


Figure 4.15: Variations of the secondary constrained objective function (path length) as the size of the clusters vary.

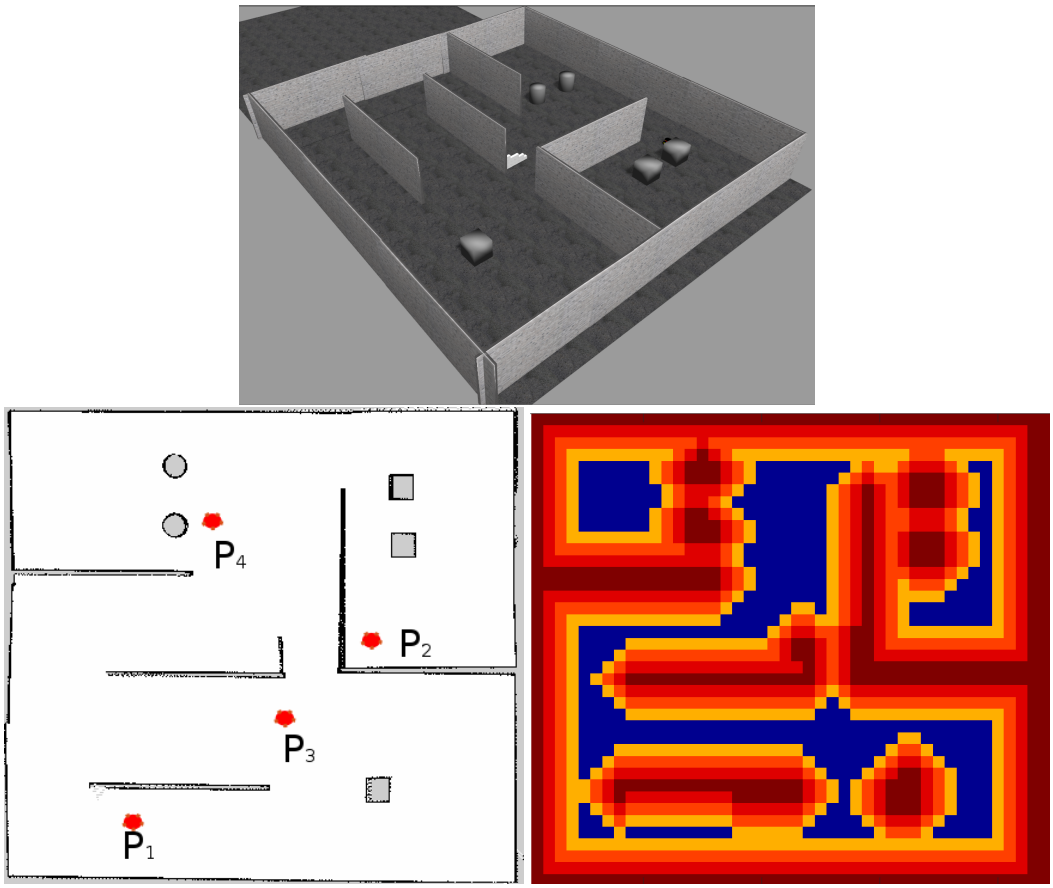


Figure 4.16: Simulated environment used for the Gazebo simulations and associated risk map.

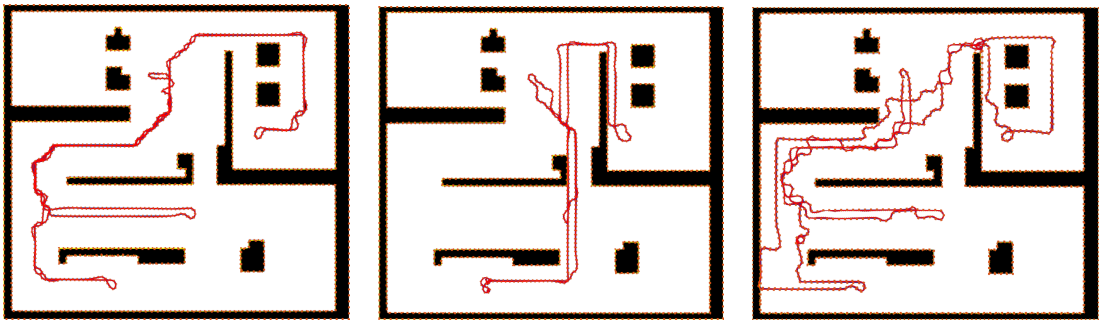


Figure 4.17: Sample paths obtained by the three planners solving the same problem instance: Left: MDP, Center: CMDP, Right: HCMDP

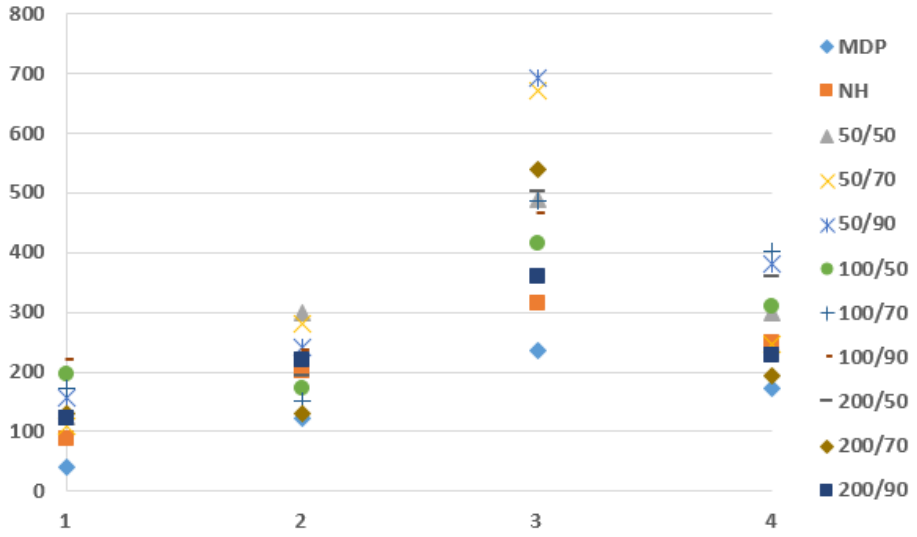


Figure 4.18: Average c_0 cost (risk) over 20 independent runs for intermediate friction.

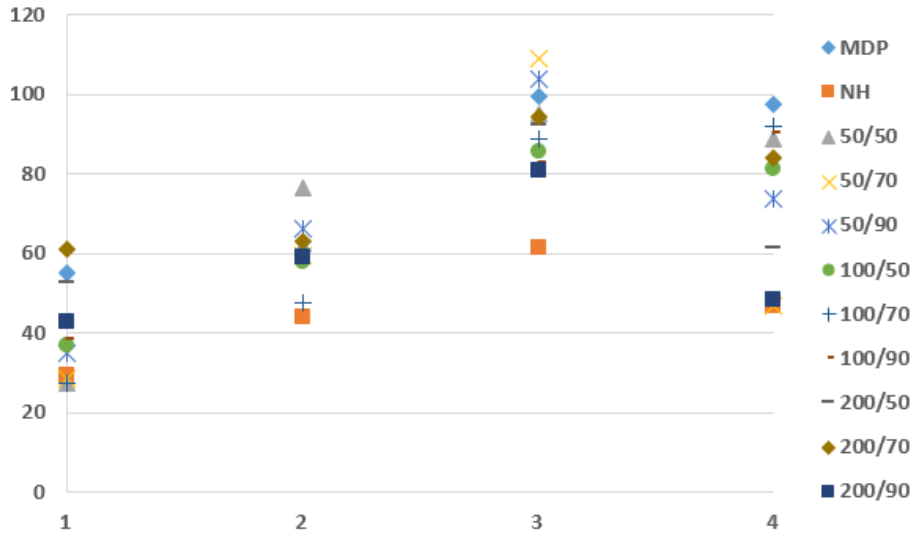


Figure 4.19: Average c_1 cost (path length) over 20 independent runs for intermediate friction. Horizontal green bars show the value of the constraint in the original, non-hierarchical problem formulation.

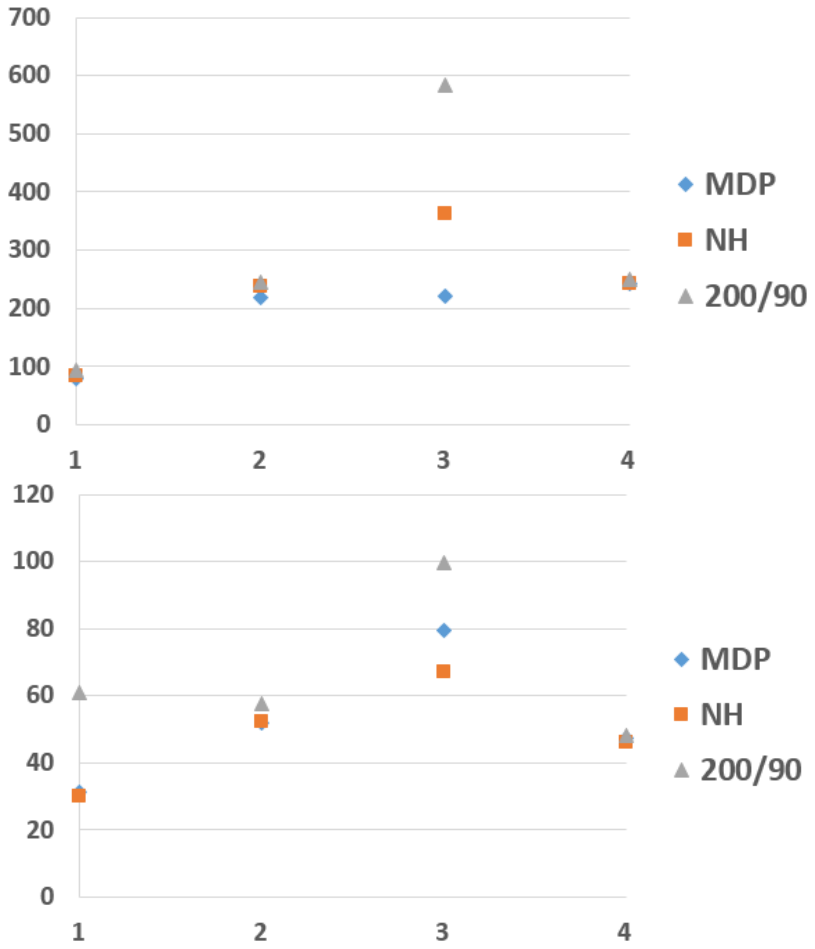


Figure 4.20: Top: average c_0 cost (risk). Bottom: average c_1 cost (path length). Charts display the average of 20 independent runs in an environment with reduced friction between the wheels and the soil. Horizontal green bars show the value of the constraint in the original, non-hierarchical problem formulation.

Figure 4.21: The P3AT robot used to test the HCMDP planner in the real world.

4.4.2.3 Real Robot Experiments

To conclude, we executed the same three algorithms to control a P3AT robot (see figure 4.21) navigating inside one of the university buildings. Figure 4.22 shows the map of the environment preliminarily obtained using the GMapping algorithm. The maximum dimensions of the environment are $42m \times 71m$, and the map was split in square cells of size $0.5m \times 0.5m$. Consistently with the previous experiments, Figure 4.23 displays the risk map associated with the map. As for the Gazebo simulations, transition probabilities were extracted through repeated simulation of the various actions. Figure 4.22 also shows four points used to define the start and goal locations for the various missions. To determine averages, every mission was executed 10 times, and over the various runs, the robot drove more than 5.5km inside the building totaling over more than 25 hours of autonomous operation. Figure 4.24 shows four paths produced by the HCMDP planner while computing a policy from point P_1 to P_2, then to P_3, afterwards to P_4, and ending at P_1. Columns in Figure 4.25 represent these four paths respectively.

As we did in the previous cases, Figure 4.25 displays the average primary cost c_0 (risk) and the average additional cost c_1 (path length). Three different versions of the HCMDP planner were compared, where differences were in the size of the clusters (100, 200, 400), whereas the percentage of samples was fixed at 90. Experiments with the real robot confirm the conclusions we anticipated in Matlab and Gazebo simulations. In particular, referring to the bottom figure we outline that the path length constraint was 45 in the first case, 55 in the second, 40 in the third, and 85 in the fourth. It can be observed that the planner with cluster size 400 is rather competitive with the non hierarchical version for both costs.

4.5 Conclusions and Future Work

This chapter proposed two hierarchical models to solve CMDPs, fixed-size partitioning and variable-size partitioning or HCMDP. Fixed-size partitioning was the primary step to implement HCMDP, and is significantly outperformed by HCMDP. Theses techniques presented in this paper are valuable because they offer an efficient way to solve sequential stochastic decision making prob-

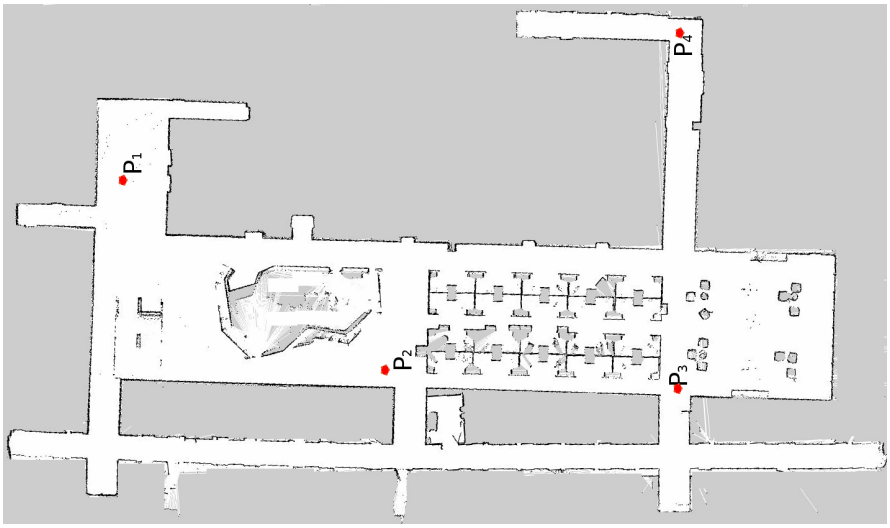


Figure 4.22: Map of the university building used to the the HCMDP planner.

Figure 4.23: Risk map associated with Figure 4.22.

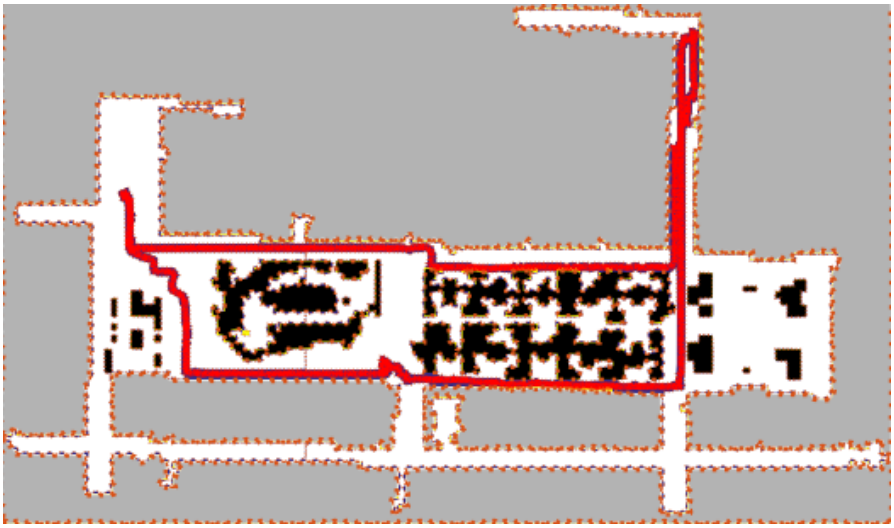


Figure 4.24: Four paths produced by the HCMDP planner while computing a policy from point P_1 to P_2, then to P_3, afterwards to P_4, and ending at P_1 (see also Figure 4.22).

lems considering more than one cost. While CMDPs offer an optimal solution to these problems, their computational burden limits their applicability in real world applications. HCMDP overcomes this problem while at the same time ensuring that valuable properties (e.g., feasibility) are maintained. The algorithm has been extensively validated in simulation and on a real robot, and the results show significant reductions in the computational time paired with modest gaps in the expectations of the cost functions.

Various interesting open questions remain. The first is whether it is possible to derive formal bounds for the performance gap between the policy produced by the non-hierarchical CMDP and HCMDP. This gap is likely to be a function of parameters like the maximum cluster size and the number of samples used in the Monte Carlo estimation, but at the moment it appears that tight bounds may not be easy to determine. The second question pertains to the choice of the parameters. It would be interesting to derive intuitions on how they should be selected for a given problem instance, perhaps in terms of an heuristic, or through an iterative guessing process. Another question is related to the estimation of the parameters for the HCMDP. Within the Monte Carlo framework, estimation can be performed in many different ways, and the one we explored in this paper is just one of the possible choices. A more in depth study would most likely determine better strategies, either in terms of performance or estimation accuracy.

On the experimental side, at the moment we have contrasted our proposed hierarchical methods only with the optimal non-hierarchical alternative. Of course, there is a range of sub-optimal methods one could use to expedite the solution of sequential stochastic decision problems, in particular for the unconstrained MDP scenario. As part of the future work, more numerical comparisons between sub-optimal solvers can be done.

Finally, with the emergence of massive multi-core architectures, it would be interesting to experiment how performance scales if one uses multiple cores to perform the Monte Carlo estimation

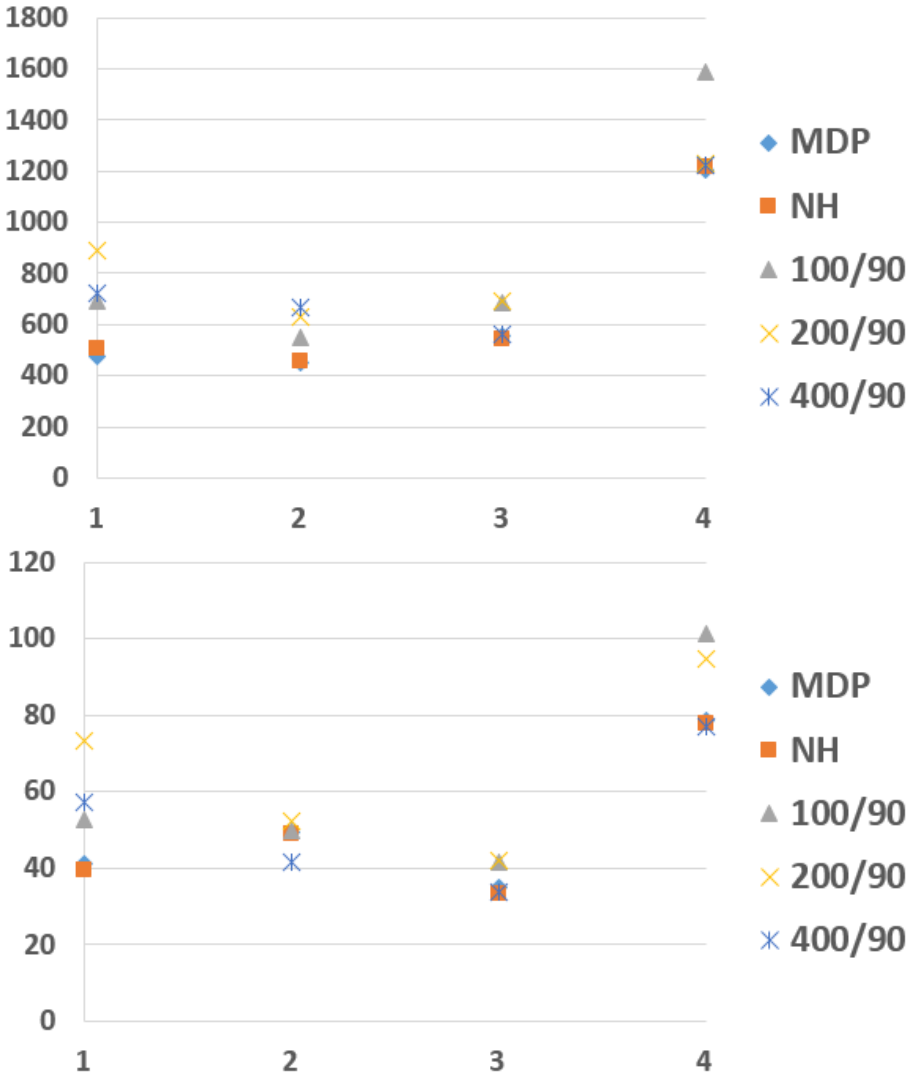


Figure 4.25: Results obtained averaging 10 runs with the real robot. Top: average c cost (risk). Bottom: average d cost (path length).

and to solve the intermediate CMDP instances.

Chapter 5

Multi Objective Planning with Multiple Costs and Multiple High Level Task Specifications

5.1 Introduction

As presented in Chapter 4, CMDPs can be used to program robots in complex missions where a robot faces multiple challenges modeled as cost functions. In such circumstances, a planner takes all the cost constrains into account and offers a viable solution optimizing the plan with respect to all the cost functions.

Obviously, we cannot model every possible objective with cost functions only. Imagine a complex mission which requires a sequence of actions to be taken with a specific order. It is difficult to interpret such sequences as cost functions. Therefore, the concept of *task* has been used to specify such sequences. With robots duties getting more complex, their tasks have to be accordingly more advanced. To this end, formalisms like Linear Temporal Logic (LTL) have been considered to express tasks that can be automatically translated into control policies. The use of LTL properties to specify tasks in robotics applications has increased significantly which makes LTL properties the primary language to interpret robotic tasks.

Moreover, robots usually do not face a single task, and they are required to accomplish several tasks or sub-tasks in a mission. Therefore, it is beneficial to have a planner that can consider multiple tasks at the same time and generate the optimal policy that satisfies all of them. It is clear that dealing with multiple tasks raises many issues such as the importance of each task, ordering and priorities of them. One way to overcome such discussions is to define soft constraints. For example having three tasks of ϕ_1, ϕ_2 and ϕ_3, we are interested in plans that satisfy them with probabilities of 0.3, 0.5 and 0.6 respectively. In other words, ϕ_1 should be satisfied in 30 % of robot runs while ϕ_2 and ϕ_3 should be satisfied in 50 % and 60 % of the runs, respectively. Then we avoid all of the mentioned complications and let the planner choose the optimal solution.

In this chapter we present a planning algorithm that simultaneously tackles all these challenges at once, i.e., it allows to consider multiple costs incurred while pursuing multiple objectives

expressed in a subset of LTL, each associated with a different target probability. Our work builds upon the theory of constrained Markov Decision Processes (CMDPs). We extend the classical definition of CMDPs introducing so-called labeled CMDPs (LCMDPs), where atomic propositions are associated to the CMDP states as labels (See Section 3.4). Labels provide a way to define a product between CMDPs and deterministic finite automata (DFAs) used to express a suitable subset of LTL formulae (sc-LTL, defined in Section 3.2.2). Based on our definitions, a product between an LCDMP and a DFA is defined. The product can be iterated because the product between an LCMDP and a DFA gives a new LCDMP. Therefore, given one LCMDP and multiple DFAs, the product between all of them can be considered, and the result is a comprehensive LCMDP considering all DFAs. Starting from these definitions, we provide conditions for the existence of an optimal solution and formulate an algorithm to determine it. Since the product operation significantly increases the size of the state space, a pruning step is introduced to remove all states that cannot be reached under any realization. The solution policy will minimize one of the cost functions, satisfy bounds on the remaining costs, and satisfy each of the formulas with the desired probabilities. The key step in our construction is the definition of the product operation so that the probability of satisfying a formula is equivalent to the probability of traversing a given state in a suitably augmented state space. Finally, we demonstrate the power of the proposed technique both in simulation and on field experiments with a Husky robot operating outdoor.

The remainder of this chapter is organized as following. In Section 5.2, we clarify how our proposed solution improves the existing literature. Later, in Section 5.3, the general problem is formally defined. Then we provide the proposed solution. Next, we present a toolbox which makes our software usable for other researchers in Section 5.4. Section 5.5 shows how the proposed approach works in Matlab and robot simulation scenarios. It also provides the results on real robot experiment. Finally, Section 5.6 concludes the chapter.

5.2 Major Contributions

The area of motion planning with LTL specifications is very dense and the differences between distinct works is often narrow. Therefore, we compare our proposed approach with the state of the art in details throughout this section.

Etessami et al. in [58, 59] presented a multi-objective planner capable of applying multiple LTL properties with predefined satisfaction rates. They proposed a three step approach to first estimate the existence of a viable solution by using Pareto Curves, then solve the problem for LTL properties which only consists of *eventually* statements and finally extend the solution to general LTL properties. This work was later extended in [68] by improving the solver to value iteration. However, our approach has four differences with theirs as following:

1. We consider multiple cost functions where one cost function has to be minimized and additional cost functions have to be bounded. On the contrary, their approach only focuses on satisfying multiple LTL properties disregarding any cost function.

2. The base of our approach is on a initial distribution of states which might consist of many initial states. Etessami's approach to estimate Pareto curves is graph based and focuses on

one initial state only. Therefore, having a distribution of initial states may require to solve the problem multiple times.

3. One of our major contributions is to define sink states on DFAs to solve contradicting LTL properties with higher probabilities. Etessami et al. used the standard product function which reduces the probability of satisfying some LTL properties. For example imagine two LTL properties $\phi_1 = \Diamond A$ and $\phi_2 = \Box(\neg A)$. The product $\phi_1 \otimes \phi_2$ will result in an empty graph according to Etessami's approach, but it might have many valid solutions in our solver if the sum of the target probabilities is less than one. We further elaborate this topic in Chapter 6 by showing the effect of such dependencies on the final solution.

4. Our work proposes a planner that finds a policy toward an absorbing state (goal state). On the contrary, Etessami's method does not consider any goal state. Their aim is to find a policy that satisfies LTL properties with predefined probabilities.

Another similar work was proposed by Forejt et al., [67], targeting non-deterministic environments that presents a framework which is capable of solving verification problems by taking advantage of *Probabilistic Automaton* (PA). They extend Etessami's work by adding the reward/cost function into the solver. However, they are bounded to total cost functions only. This research was extended in [102] by decomposing the properties into smaller ones to speed up the process. However, similar differences apply. The advantages of our work over Forejt's work is summarized as:

- Our solver considers multiple cost functions while they only take one cost function into account.
- Their focus is to satisfy a bound on their cost function, not minimizing that. However, we minimize the primary cost function in the CMDPs.
- Their approach is bound to total cost problems while CMDPs can be solved in different forms such as average cost as well as total costs. They can be approximated as well [4].
- Similar to Etessami's work, they do not accommodate contradicting LTL properties.

In another work, Ding et al. used CMDPs in conjunction with LTL properties [52]. But their approach is very limited by considering a single LTL property under special conditions only. The method cannot accommodate multiple LTL properties at once. Moreover, their approach is computationally expensive by lacking any form of optimization or pruning methodology for the generated graph.

In general, the novel idea behind our approach is to address multiple LTL properties into a CMDP without losing any functionalities. By introducing extended-total-DFAs, we offer a product operation that is repeatable. This addition creates a product operation which is fully commutative, i.e. the order in which the product operation is applied does not have any effect on the final output. We provide further evidences in Chapter 6 regarding the advantages of using this type of product operation. Note that none of the aforementioned multi-objective solvers provides a commutative product operation. One of the main drawbacks of such methods lies in the importance of ordering the products. Therefore, the LTL property which is applied first receives a higher priority than the ones that are applied later.

5.3 Multi Objective Planning with High Level Task Specifications

5.3.1 Problem formulation

Building upon the definitions we introduced in Chapter 3, we can now formulate the problem we tackle in this chapter. In particular, we want to determine policies for situations where a robot is tasked with multiple objectives at once, and is subject to multiple costs. Each task is formalized as an sc-LTL formula. Rather than requesting that each task is completed for sure, we assign to each task (and then formula) a desired probability of success. It is important to notice that every formula can be associated with a different target probability. Therefore, a form of *soft* task assignment is defined. These considerations lead to the following formulation.

Multi-Objective, Multi-Property (MOMP) MDP – Given:

- an LCMDP $\mathcal{M} = (S, \beta, U, c_i, \Pr, \Pi, L)$ with $n+1$ costs functions $c_0, c_1, \ldots, c_n$;
- m sc-LTL formulas $\varphi_1, \ldots, \varphi_m$ over Π;
- n cost bounds $B_1, \ldots, B_n$;
- m probability bounds $P_{\varphi_1}, \ldots, P_{\varphi_m}$;

determine a policy π for $\mathcal{M}$ that:

- minimizes in expectation the cost $c_0(\pi, \beta)$;
- for each cost c_i $(1 \leq i \leq n)$ it satisfies in expectation the bound $c_i(\pi, \beta) \leq B_i$;
- for every trajectory Ω, each of the m formulas φ_i is satisfied with at least probability P_{φ_i}.

Before discussing the proposed solution, it is worth recalling that each formula φ_i is satisfied with a certain probability and that bounds on the cost functions are satisfied in expectation, coherently with the CMDP theory. This means that a specific trajectory Ω could miss all properties and violate all constraints. However, over repeated iterations, all of these goals will be achieved at once. This aspect will become evident when discussing simulation and experimental results.

5.3.2 Proposed Solution

Our two-step solution strategy can be sketched as follows. We start from the fact that for every sc-LTL formula φ_i there exists a DFA $\mathcal{D}_i$ accepting all and only the words satisfying φ_i, and then, for the equivalences we discussed in section 3.2.1, there is an extended-total-DFA $\mathcal{E}_i$ accepting the language satisfying φ_i. To be more precise, there exist a DFA $\mathcal{D}_i$ accepting all and only the finite length words starting with prefixes satisfying the sc-LTL formula φ_i, and then there is an extended-total-DFA $\mathcal{E}_i$ accepting the $\mathscr{G}$-ending infinite length words obtained extending the language accepted by $\mathcal{D}_i$. We define a product operation between an LCMDP and an extended-total-DFA that yields a new LCMDP whose trajectories are related to the probability of satisfying the formula associated with the extended-total-DFA. Since the product between an LCMDP and an extended-total-DFA

gives a new LCMDP, given multiple extended-total-DFAs the product can be iterated multiple times to include all extended-total-DFAs until eventually a comprehensive LCMDP is obtained. Trajectories generated by this LCMDP allow to compute the probability that each formula φ_i is satisfied. However, at each step the new LCMDP produced through the product operation has a larger state set, and after multiple iterations the state space may become rather large and create a computational bottleneck. To counter this problem, we introduce a pruning algorithm that reduces the size of the state space by removing unnecessary states, i.e., states that cannot be reached by any realization. Finally, the pruned LCMDP is solved using a linear program built upon the classic theory of CMDPs. In the remaining of this section we detail each of these steps.

5.3.2.1 Product definition

Products between transition systems or CMDPs and DFAs were already considered in literature [10, 52], but since our objective is to precisely assess the probability that each different formula φ_i is satisfied, our approach is different. In particular, our definition of product between an LCDMP and an extended-total-DFA gives another LCMDP. Therefore, given multiple DFAs it is possible to iterate the product between the initial LCMDP and all the DFAs. We first formally define the product and then discuss the rationale behind its structure.

Definition 13 *Let $\mathcal{M} = (S, \beta, U, c_i, \Pr, \Pi, L)$ be an LCMDP with absorbing state s_a, and let $\mathcal{E} = (Q, q_0, \delta, F, \Sigma)$ be an extended-total-DFA with alphabet $\Sigma = 2^{\Pi}$, sink state q_s and absorbing state q_a. The product between $\mathcal{M}$ and $\mathcal{E}$ is an absorbing LCDMP $\mathcal{M} \otimes \mathcal{E} = (S_p, \beta_p, U_p, c_{p_i}, \Pr_p, \Pi, L_p)$ where:*

- $S_p = S \times Q$.
- $\beta_p(s, q) = \begin{cases} \beta(s) & \text{if } q = q_0 \\ 0 & \text{otherwise} \end{cases}$
- $U_p = U$.
- $c_{p_i}((s, q), u) = c_i(s, u)$ for $0 \leq i \leq n$.
- $$\Pr_p((s, q), u, (s', q')) = \begin{cases} \Pr(s, u, s') & \text{if } q' = \delta(q, L(s)) \\ 0 & \text{otherwise} \end{cases}$$
- $L_p(s, q) = L(s)$.

States of the type (s, q_s) where q_s is the sink state in the extended-total-DFA are called *sink states* for the product LCMDP. Note that in the original LCMDP definition we did not introduce any sink state, therefore sink states will appear only in LCMDPs obtained through a product with a total-extended-DFA. The key element in the product definition is the transition probability $\Pr_p$ indicating that a transition between two states (s, q) and (s', q') may occur under input u only if s' can be reached from s under input u, and q' is obtained from q when the input for the extended-total-DFA

is the set of atomic propositions $L(s)$ verified in s. It is easy to verify that since $\mathcal{M}$ is absorbing then $\mathcal{M} \otimes \mathcal{E}$ is absorbing too, and its absorbing states are (s_a, q) where $q \in F \cup \{q_a\}$, i.e., it is either a final state or the absorbing state of the extended-total-DFA. Let $Q_{a_i} = F_i \cup \{q_{a_i}\}$, i.e., the set of final states and the absorbing state of the i-th extended-total-DFA. Q_{a_i} is the set of *terminal* states for $\mathcal{E}_i$, that is while processing a $\mathscr{G}$-ending infinite length word w' the extended-total-DFA will always eventually reach either a state in F (it w' is accepted) or q_a (if w' is rejected). Let us consider product between an LCMDP $\mathcal{M}$ and m extended-total-DFAs $\mathcal{E}_1, \ldots, \mathcal{E}_m$. This product $\mathcal{M} \otimes \mathcal{E}_1 \otimes \ldots \mathcal{E}_m$ is also an absorbing LCMDP whose states have the form $(s, q_1, \ldots, q_m) \in S \times Q_{\varepsilon_1} \times Q_{\varepsilon_2} \cdots \times Q_{\varepsilon_m}$. Its set of absorbing states is

$$S_a = \{(s, q_1, \ldots, q_m) \mid s = s_a \wedge \forall i \; q_i \in Q_{a_i}\}.$$

Accordingly, the set $S'_p = S_p \setminus S_a$ is the set of transient states in the product. Figure 5.1 shows an example of product between an LCMDP and two total-extended-DFAs.

In both extended-total-DFAs, if the string does not belong to the accepted regular language ($AB(A + B + C)^*$ for the first automaton, and $AC(A + B + C)^*$ for the second automaton), when the first instance of $\mathscr{G}$ occurs the state moves to the sink state q_s and then at the next step moves to the absorbing state q_a and remains there. The product LCMDP shown at the bottom features some interesting properties that will be useful to compute the probability that an sc-LTL formula is verified or not. The top branch in the diagram traverses the state $S_4 q_3 q'_s$. This path is associated with any word matching the regular expression AB^* that is accepted by the first total-extended-DFA but not by the second. In the path this is modeled by the passage through a state including q'_s, i.e., the sink state for the extended-total-DFA not accepting the string. Similarly, the bottom branch goes through $S_4 q_s q'_3$ since it is associated with strings matching AC^*, i.e., accepted by the second total-extended-DFA, but not by the first. Finally, the middle branch is associated with the regular expression $A\mathscr{G}^*$ that is not accepted by either automaton. Accordingly, the path traverses a composite state including both sink states q_s and q'_s, i.e. $S_4 q_s q'_s$. Note that in the product LCMDP there are three absorbing states at the very right and that without loss of generality one could combine them into a single absorbing state. However in the figure we do merge them to better illustrate the process leading to the product.

5.3.2.2 Reducing the Size of State Space

Given an LCMDP $\mathcal{M}$ and an extended-total-DFA $\mathcal{E}$, the product we just defined creates an LCMDP $\mathcal{M} \otimes \mathcal{E}$ where $S_p = S \times Q$. In practice numerous states in S_p will be *unreachable*, i.e., they will not be visited under any policy because of the way the new transition probability function $\Pr_p$ has been introduced. As it will be evidenced later, the number of states has a linear relationship with the number of variables in the linear program to be solved to determine the solving policy. Therefore, to reduce this number, instead of creating S_p as per the definition, we replace it with a set of reachable states $\mathcal{R} \subset S \times Q$, i.e., states that can be reached. The key observation is that if $\Pr_p((s, q), u, (s', q')) = 0$ for each $(s, q) \in S \times Q$ and $u \in U$, then (s', q') can be omitted from S_p. These states can be easily identified from the definition of $\Pr_p$ as illustrated in Algorithm 5. The algorithm takes as input an LCMDP $\mathcal{M}$ and an extended total-DFA $\mathcal{E}$, and returns the set of reachable states $\mathcal{R} \subset S \times Q$. $\mathcal{R}$ is iteratively grown from the state of initial states with strictly positive probability (line 1). The set $\mathcal{R}_{an}$ (reachable analyzed) is created as well, initially empty. $\mathcal{R}_{an}$ is the

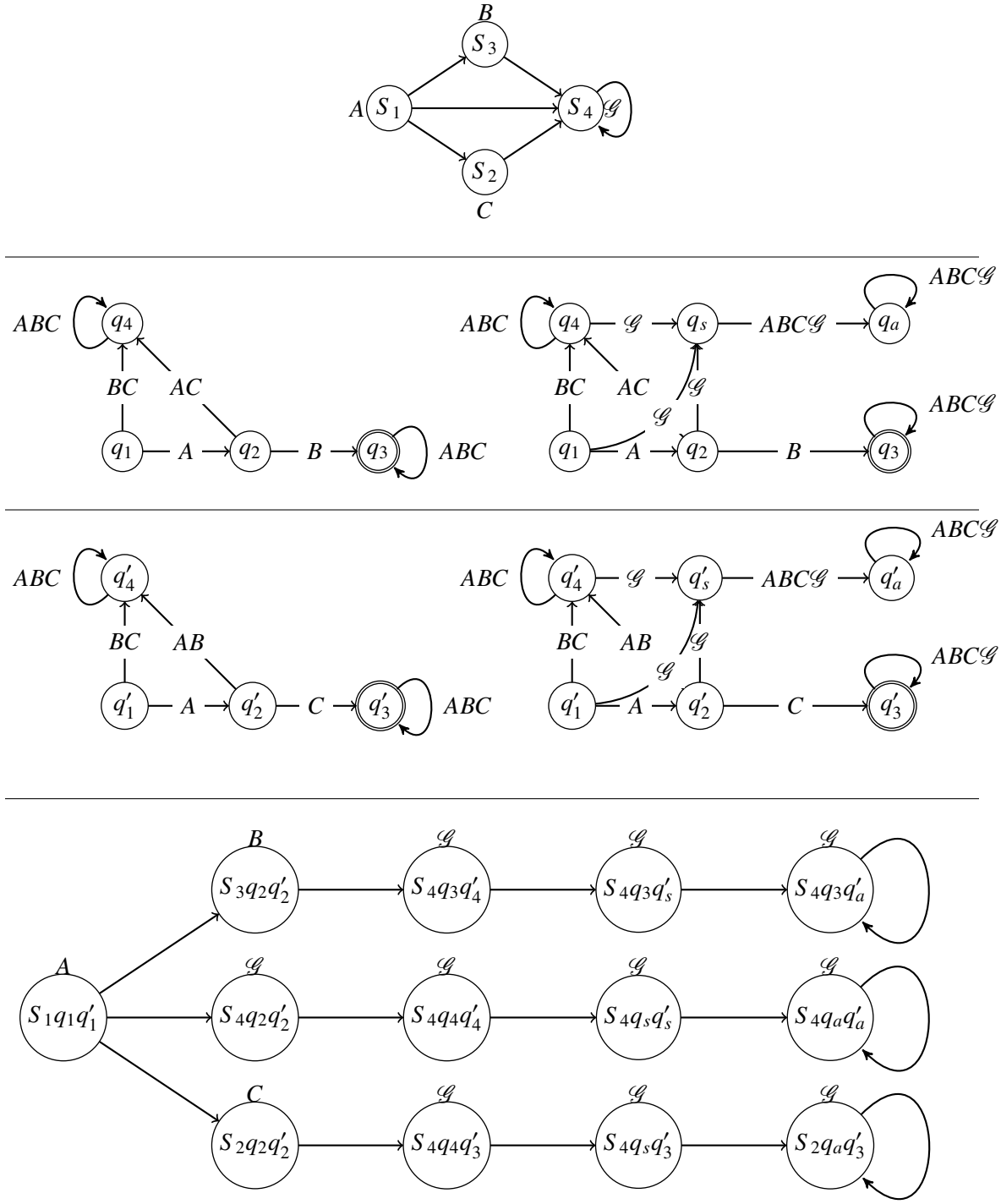


Figure 5.1: Top picture shows the original LCMDP with 4 states, absorbing state S_4 and atomic propositions set $\Pi = \{A, B, C, \mathcal{G}\}$. Atomic propositions are placed near the states in which they are verified, as per the labeling function L. Note that this trivial LCMDP is absorbing. The second figure shows on the left a DFA accepting the regular expression $AB(A + B + C)^*$, and on the right the associated extended-total-DFA. Note that the initial states of the left and right DFAs are q_1 and q'_1 respectively. Similarly, the third figure shows on the left a DFA accepting the regular expression $AC(A + B + C)^*$, and on the right the associated extended-total-DFA. Note the added sink states (q_s and q'_s) and absorbing states (q_a and q'_a). The bottom picture shows the pruned product model $\mathcal{M} \otimes \mathcal{E}_1 \otimes \mathcal{E}_2$ with three absorbing states.

set of reachable states whose successors have already been determined. The algorithm continues as long as the successors of all reachable states have not been determined (line 3). At each iteration the algorithm considers a reachable state not yet analyzed (line 4), and using the function Reachable-States-From it determines the set of states that are reachable in one transition from this state (line 5). These new states become themselves reachable and are added to $\mathcal{R}$ (line 6) and the loop continues until all reachable states have been analyzed.

Algorithm 5: Selection of reachable states in product LCMDP

Data: LCMDP, $\mathcal{M} = (S, \beta, U, c_i, \Pr, \Pi, L)$, extended-total-DFA $\mathcal{E} = (Q, q_0, \delta, F, \Sigma)$
Result: Set of reachable states in the product ($\mathcal{R} \subset S \times Q$)

1 $\mathcal{R} = \{(s, q) \mid s \in S \wedge q = q_0 \wedge \beta(s) > 0\}$;
2 $\mathcal{R}_{an} = \varnothing$;
3 **while** $\mathcal{R} \neq \mathcal{R}_{an}$ **do**
4 | select $(s, q) \in \mathcal{R} \setminus \mathcal{R}_{an}$;
5 | $Q' \leftarrow$ Reachable-States-From$((s, q), \mathcal{M}, \mathcal{E})$;
6 | $\mathcal{R} \leftarrow \mathcal{R} \cup Q'$;
7 | $\mathcal{R}_{an} \leftarrow \mathcal{R}_{an} \cup \{(s, q)\}$;

Algorithm 6 shows how states reachable in a single step can be determined. It uses a function Post(s) that takes a state in $\mathcal{M}$ and returns the set of states that can be reached from state s in one step with non zero probability. The algorithm simply returns the states (s', q') having non zero probability of being reached from (s, q) as per the definition of $\Pr_p$.

Algorithm 6: Reachable-States-From

Data: $(s, q) \in S \times Q$, $\mathcal{M} = (S, \beta, U, c_i, \Pr, \Pi, L)$, $\mathcal{E} = (Q, q_0, \delta, F, \Sigma)$
Result: The set of reachable states from (s, q)

1 $\Delta \leftarrow L(s)$;
2 $C \leftarrow \varnothing$;
3 $S' \leftarrow$Post(s);
4 **for** $a \in \Delta$ **do**
5 | $q' \leftarrow \delta(q, a)$;
6 | **for** $s' \in S'$ **do**
7 | | $C = C \cup (s', q')$;
8 **return** C

After computing $\mathcal{R}$, we can then consider $\mathcal{M} \otimes \mathcal{E} = (\mathcal{R}, \beta_p, A_p, c_{p_i}, \Pr_p, \Pi, L_p)$. Since by construction states excluded from $\mathcal{R}$ cannot be reached under any policy, substituting S_p with $\mathcal{R}$ does not change the underlying properties of the LCMDP.

5.3.2.3 Solving the Optimization Problem

The reason for introducing the product LCMDP is to convert the problem of computing the probability of satisfying multiple sc-LTL formulas into a reachability problem. In the example

discussed in Figure 5.1 we saw that trajectories of the product LCMDP are associated with the sc-LTL formulas being satisfied by $L(\Omega)$ or not. Therefore, to evaluate the probability that one or more formulas are satisfied, it is sufficient to compute the probability that certain states are reached by a trajectory Ω. In particular, we are interested in the composite states including the sink states defined by the extended-total-DFAs. Note that in the definition of the product LCMDP, states including one or more sink states are visited at most once, i.e., if Ω reaches on of them at a certain time t, it will move to an absorbing state at time $t + 1$. The relevance of this property hinges on the definition of occupancy measure induced by a policy π. Note that this definition is given for the general case of CMDP without labels.

Definition 14 (Occupancy measure) *Given an absorbing CMDP, let S' be its set of transient states and let $K' = \{(s, u) \mid s \in S' \wedge u \in U(s)\}$. For a policy π we define the occupation measures $\rho(s, u)$ for each state/action element in K' as*

$$\rho(s, u) = \sum_{t=0}^{+\infty} \Pr_{\beta}^{\pi}[S_t = s, U_t = u]$$

where $\Pr_{\beta}^{\pi}[S_t = s, U_t = u]$ is the probability induced by the policy π and the initial distribution β.

According to the definition, the occupation measure is a sum of probabilities and then in general not a probability itself. However, if the structure of the underlying CMDP is such that a state s' is entered at most once, then $\sum_{u \in U(s')} \rho(s', u)$ is instead a probability because at most one term is different from zero in the sum and that term by definition is a probability. It is indeed this property that led us to the definition of the total-extended-DFA and the introduction of the hypothesis concerning the extra symbols $\mathscr{G}$ at the end. When this is the case the sink state is always visited at most once. Therefore, if there is an event occurring only when the sink state is visited, then the occupancy measure associated with the sink state is exactly the probability that such event happens. As we discussed in the beginning of this subsection, a visit to a state including one or more sink states is associated with the event that $L(\Omega)$ does not satisfy the corresponding sc-LTL formulas. This approach is further expanded in the following. We start with this simple theorem.

Theorem 3 *Let $\mathcal{M}$ be an LCMDP and $\mathcal{E}$ be an extended-total-DFA associated with the sc-LTL formula φ. Let $\Omega = (s_1, q_1), \ldots, (s_n, q_n), \ldots$ be a trajectory of $\mathcal{M} \otimes \mathcal{E}$ and $\mathcal{L}(\Omega)$ be the corresponding string. $\mathcal{L}(\Omega)$ satisfies φ if and only if Ω does not include any state $(s, q) \in S_p$ where $q = q_s$.*

Proof The proof follows from the definition of product $\mathcal{M} \otimes \mathcal{E}$ and the assumption that $\mathcal{E}$ is the extended-total-DFA associated with φ. This latter property states that $\mathcal{L}(\mathcal{E})$ is the language or $\mathscr{G}$-ending infinite length words with a prefix satisfying φ. Since we assumed that the LCMDP is absorbing and that $L(s) = \mathscr{G}$ for all an only states in the absorbing set, it follows that $\mathcal{L}(\Omega)$ is a $\mathscr{G}$-ending infinite length word. Therefore, if $\mathcal{L}(\Omega)$ is not accepted by $\mathcal{E}$ the state enters q_s once and then moves into q_a, whereas if $\mathcal{L}(\Omega)$ is accepted by $\mathcal{E}$ the state never enter q_s. Given that $S_p = S \times Q$ and the transition rule stated in Definition 13, the claim follows.

According to the theorem, if a trajectory Ω of $\mathcal{M} \otimes \mathcal{E}$ reaches a sink state, then the corresponding sc-LTL property φ is not satisfied by $\mathcal{L}(\Omega)$. Otherwise, it does. Therefore, we are looking for

traces starting from an initial state, without passing the sink state and ending in the absorbing state. These trajectories satisfy φ by construction and as we will show in the following we can compute the probability of satisfying φ.

Theorem 4 *Let $\mathcal{M}_p = \mathcal{M} \otimes \mathcal{E}$ be a product LCMDP where $\mathcal{E}$ is the extended-total-DFA associated with an sc-LTL property φ. If a policy π generates trajectories satisfying φ with probability P_φ, then with probability $1 - P_\varphi$ the policy will generate trajectories visiting the sink states $(s, q_s) \in S_p$.*

Proof Because of the definition of transition function for an extended-total-DFA and the definition of product $\mathcal{M} \otimes \mathcal{E}$, every state of $\mathcal{M}_p$ of the type (s, q_s) is entered at most once. Moreover a state of the type (s, q_s) is entered if and only if the property φ is not satisfied by $\mathcal{L}(\Omega)$, so if the policy π satisfies φ with probability P_φ, the trajectory enters states (s, q_s) with probability $1 - P_\varphi$.

Given an LCMDP $\mathcal{M}$ and m extended-total-DFAs $\mathcal{E}_1, \ldots, \mathcal{E}_m$ associated with sc-LTL properties $\varphi_1, \ldots, \varphi_m$, let

$$\mathcal{M}_p = \mathcal{M} \otimes \mathcal{E}_1 \otimes \mathcal{E}_2 \cdots \otimes \mathcal{E}_m.$$

For each property φ_i, we define the set

$$\mathscr{S}_i = \{(s, q_1, \ldots, q_i, \ldots, q_m) \in S_p \mid q_i = q_{s_i}\}$$

i.e., the set of states in $\mathcal{M}_p$ including the sink state for the extended-total-DFA associated with φ_i. Note that according to how we defined the product operation and the transition function for the total-extended-DFA (third case in Eq. (3.1)) s must be the absorbing state s_a in the LCMDP. This is true because q_{s_i} is entered only after the symbol $\mathscr{G}$ is processed, and $L(s) = \{\mathscr{G}\}$ only in the absorbing state s_a. Consequently, for any state in $\mathscr{S}_i$, there is only one defined action, i.e., the *loop* action u_a that in the original absorbing LCMDP goes from state s_a to state s_a with probability 1 (see Section 3.4). To ease the notation, in the linear program that follows we write the tuple $((s, q_1, q_2, \ldots, q_n), u)$ as (x, u) with the understanding that x is a state in $\mathcal{M}_p$ and $u \in U(x)$ is an action for x as recursively defined by the product. At this point we have all the elements to formulate the main theorem providing the solution for the problem we defined in Section 5.3.1.

Theorem 5 *Let $\mathcal{M} = (S, \beta, U, c_i, \Pr, \Pi, L)$, $\varphi_1, \ldots, \varphi_m$, $B_1, \ldots, B_n$ and $P_{\varphi_1} \ldots P_{\varphi_m}$ be as in Section 5.3.1. Let $\mathcal{E}_1, \ldots, \mathcal{E}_m$ be m extended-total-DFAs associated with $\varphi_1, \ldots, \varphi_m$, $\mathcal{M}_p = \mathcal{M} \otimes \mathcal{E}_1 \otimes \mathcal{E}_2 \cdots \otimes \mathcal{E}_m$, and K be the state action space associated with the set S'_p of transient states in $\mathcal{M}_p$. The multi-objective, multi-property MDP problem admits a solution if and only if the following linear program is feasible:*

$$\min_{\rho(x,u) \in K} \sum_{x \in S'_p} \sum_{u \in U(x)} c_0(x, u) \rho(x, u) \tag{5.1}$$

subject to

$$\sum_{x \in S'_p} \sum_{u \in U(x)} c_i(x, u) \rho(x, u) \leq B_i, i = 1, ..., n \tag{5.2}$$

$$\sum_{x \in \mathscr{S}_i} \sum_{u \in U(x)} \rho(x, u) \leq 1 - P_{\varphi_i}, i = 1, ..., m \tag{5.3}$$

$$\sum_{x' \in S'_p} \sum_{u \in U(x')} \rho(x,u)[\delta_x(x') - \Pr_p(x',u,x)] = \beta(x), \forall x \in S'_p \tag{5.4}$$

$$\rho(x,u) \geq 0, \forall x \in S'_p \tag{5.5}$$

where $\delta_x(x')$ equals to one, if $x = x'$ and zero otherwise.

Proof The proof follows from the theory of CMDPs and our definition of product. First consider the linear program excluding the constraints in Eq. (5.3). This is precisely the linear program to solve a CMDP with an occupancy measure approach (see, e.g., [5]). Next, consider the additional m constraints in Eq. (5.3). By definition, states in $\mathscr{S}_i$ have the form $(s_a, q_1, \ldots, q_i, \ldots, q_m)$ with $q_i = q_{s_i}$, i.e., the ith component is the sink state of the total-extended-DFA associated with formula φ_i. Moreover, they are associated with just one action, i.e., u_a. Therefore, we can write

$$\rho(x,u_a) = \sum_{t=0}^{+\infty} \Pr^{\pi}_{\beta}[S_t = x, U_t = u_a] = \sum_{t=0}^{+\infty} \Pr^{\pi}_{\beta}[S_t = x]$$

and the ith constraint in Eq. (5.3) can therefore be rewritten as

$$\sum_{x \in \mathscr{S}_i} \sum_{t=0}^{+\infty} \Pr^{\pi}_{\beta}[S_t = x] \leq 1 - P_{\varphi_i}.$$

Let ζ be the sample space for Ω, i.e., the stochastic process for the state evolution, and consider the event $A^k_t = \{S_t = s_k\}$, where $s_k \in \mathscr{S}_i$. From the rules defining how the state evolves, if the state is in $\mathscr{S}_i$ at time t, it will not be in $\mathscr{S}_i$ at time $t' > t$, because the ith component will move from q_{s_i} to q_{a_i} and remain there. Therefore the events A^j_t and $A^k_{t'}$ are mutually incompatible for any choice of indices j, k and t, t' (unless of course $j = k$ and $t = t'$, but this will never happen in the sum we consider in the following). The above sum is therefore the same of probabilities of mutually incompatible events, and it is therefore equal to the probability of the union of the events, that is

$$\sum_{x \in \mathscr{S}_i} \sum_{t=0}^{+\infty} \Pr^{\pi}_{\beta}[S_t = x] = \Pr\left[\bigcup_{t=0}^{+\infty} \bigcup_{k} A^k_t\right] = \Pr\left[\bigcup_{t=0}^{+\infty} \{S_t \in \mathscr{S}_i\}\right].$$

This last probability is therefore the probability that Ω reaches $\mathscr{S}_i$. Combining this fact with Theorem 4, it follows that each of the m constraints in (5.3) bounds the probability that formula φ_i is not satisfied, and this concludes the proof.

At this point it should be clear why the sink states q_s and q_a were introduced when defining the extended-total-DFAs. q_s is separated from q_a in order to make sure q_s will be visited at most one time for every policy realization, and our definition of product allows to track multiple formulas in parallel. If the linear program is solvable, its solution provides a randomized, stationary, Markovian policy as follows (see [5]):

$$\pi(x,a) = \frac{\rho(x,a)}{\sum_{a \in A(x)} \rho(x,a)}, x \in S'_p, a \in A(x)$$

where $\pi(x,a)$ is the probability of taking action a when in state x.

5.4 MOMP-MDP Toolbox

Since the presented theory is a general-purpose solver which is not tied to any specific problem or application, we present a toolbox in this section to be released to the scientific community. The proposed toolbox is implemented in MATLAB, receives a problem specification, its constraints and objectives, and returns a policy. Here, we illustrate more details of the toolbox.

The input parameters of this toolbox are as following:

- An LCMDP $\mathcal{M}$ where $n+1$ cost functions are defined for each state. The toolbox provides a class for LCMDPs.
- m DFAs. The DFA class implemented in the toolbox is used for this purpose.
- n upper bounds for the non-primary cost functions, encoded as a vector $\mathcal{A}$.
- m satisfaction probabilities for each DFA. These values are appended to $\mathcal{A}$.

Upon proper setup, the toolbox calculates the optimal policy according to the linear program formerly presented. If a solution satisfying the constraints is found, the toolbox generates a feedback policy π specifying which action should be taken at any given state. Additionally, the planner simulates multiple executions of the generated policy to experimentally ensure that the policy satisfies the given constraints.

This toolbox comes with some auxiliary functions to convert an image into an LCMDP (see section 5.5.2 for examples), and also to verify that the user defined DFAs are correct and total. The class diagram of the toolbox is given in Figure 5.2.

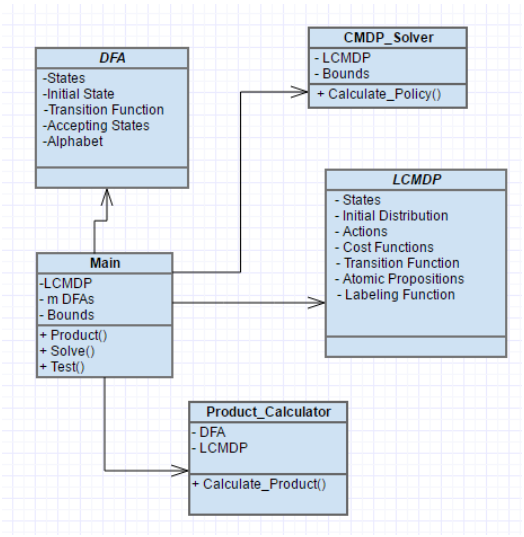


Figure 5.2: Class Diagram for the MOMP-MDP Toolbox

5.5 Experimental Evaluations

In this section, we illustrate how our planner operates under different scenarios. We start with a simulation scenario to explore a wide variety of parameters and to perform a large number of test cases. In Section 5.5.3 we show how the same framework can be applied to solve planning problems for a robot operating in real environments. In all scenarios, a grid based abstraction for the environment is adopted, whereby the robot moves on a grid and at each location it can move up/down/left/right, unless the motion is forbidden by the environment (e.g., when the robot is at the boundary or when there are obstacles.)

5.5.1 Matlab Simulations

5.5.1.1 Outdoor navigation

In the first case we consider a robot moving in an outdoor environment without obstacles. (see Figure 5.3). The leftmost panel shows an elevation map retrieved from the web. The map is discretized into a grid as shown in the middle panel. To each grid cell a *risk* level is associated based on the elevation. Warmer colors are associated with riskier areas, and this will be one of the costs considered in the LCMDP formulation. The set of atomic propositions is $\Pi = \{R_1, R_2, R_3, R_4, X\}$ and the figure shows in which grid cells these propositions hold. The figure shows also the start location S and the goal location G. The rightmost panel shows an example of the path followed by the robot while satisfying all the tasks we specify in the following.

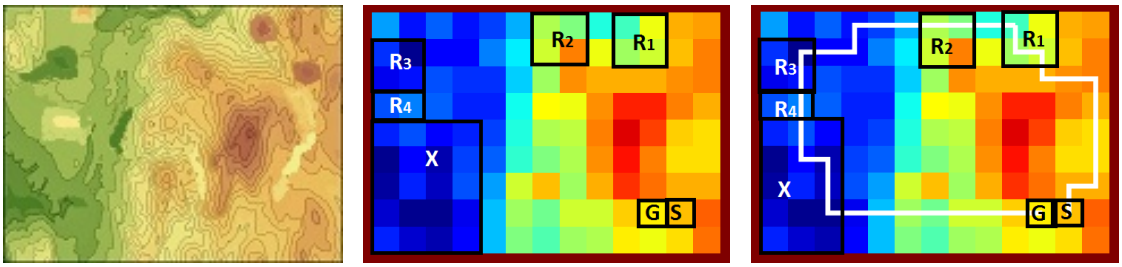


Figure 5.3: Navigation experiment: Left: Terrain map retrieved from Internet. Middle: Risk map and labels. Right: A sample path satisfying all tasks.

Coherently with the MDP framework, the outcome of actions is non-deterministic and the transition probabilities are determined by the elevation map. To be specific, an action attempting to move the robot towards a location not higher than the current location succeeds with probability 0.9 and fails with probability 0.1. If the action fails the robot remains at its own location or moves to one of its neighbors with lower height (excluding the intended target location). Each of these locations is selected with equal probability. When the robot selects an action trying to move to a higher location, the action succeeds with probability of 0.3, and fails in similar way.

In all experiments we consider two costs, c_0 and c_1. The first one is the cumulative risk along a path, i.e., the sum of the risk values associated with all cells traversed along a path. This objective function must be minimized. The second cost c_1 is the path length and must be bounded by a given

constant $B_1 = 30$. Every time an action is executed a cost of one is incurred, i.e., $c_1(x, a) = 1$ for each $x \in S$. The planner is tasked with the following three objectives formulated as sc-LTL formulas:

1. Visit region R_1 then R_2 to read some sensory data: $\varphi_1 = \Diamond(R_1 \wedge \bigcirc\Diamond R_2)$.

2. Visit region R_3 and leave region R_3 from its border to region R_4: $\varphi_2 = \Diamond(R_3 \wedge \bigcirc(R_3 \,\mathtt{U}\, R_4))$.

3. Visit at least two consecutive states in region X to explore that region: $\varphi_3 = \Diamond(X \bigcirc X)$.

Consistently with our problem definition, we assigned different probability bounds to each of the properties specifying the three tasks, namely φ_1 should be satisfied with probability 0.7, φ_2 with probability 0.8 and φ_3 with probability 0.5. The right panel in Figure 5.3 shows a stochastic realization of the path that satisfies all properties.

5.5.1.2 Factory environment

We next consider a pickup-deliver task in a factory-like environment. This is similar to our former work considered in [64, 65], and Figure 5.4 shows the floor plan.

As in the previous case, a risk map is associated to the map (middle panel in Figure 5.4), where risk in this case is a function of the distance from walls and obstacles. The figure also shows that $\Pi = \{P_1, P_2, R, D\}$ and in which grid cells the atomic propositions hold. As in the previous example, the robot starts in the cell marked S and ends in the goal cell marked G. Actions are non-deterministic and succeed with probability 0.8. When failure occurs, the robot ends up in one of the free adjacent locations (excluding the target location). As in the previous example, the robot has to minimize the overall cumulative risk along the path while obeying to a bound on the traveled length ($B_1 = 150$ in this case). As in the previous scenario, $c_1(x, a) = 1$ for each $x \in S$ and every action. The following tasks are defined as sc-LTL formulas:

1. Go to P_1 to pick up an object, then deliver it at location D: $\varphi_1 = \Diamond(P_1 \wedge \bigcirc\Diamond D)$;

2. Go to P_2 to pick up an object, then deliver it at D location: $\varphi_2 = \Diamond(P_2 \wedge \bigcirc\Diamond D)$;

3. Stay away from region R: $\varphi_3 = (\neg R)\,\mathtt{U}\,G$. Since there is no *always* ($\Box$) operator for sc-LTL properties, it needs to be defined requiring that the robot ends in G.

In this case the desired probability for φ_1 and φ_2 is 0.7, whereas for φ_3 it is 0.8. The bottom panel in Figure 5.4 shows a path satisfying all formulas.

5.5.1.3 Rapid Deployment

As third scenario we consider the rapid deployment problem we studied in [33, 38]. In the rapid deployment problem a robot is tasked with visiting various location within a given temporal deadline while maximizing the chance of succeeding. Figure 5.5 shows a map where four regions are labeled as A, B, C and D and the goal area is marked with G. The rapid deployment problem can be formulated assigning a target probability to each of the following tasks:

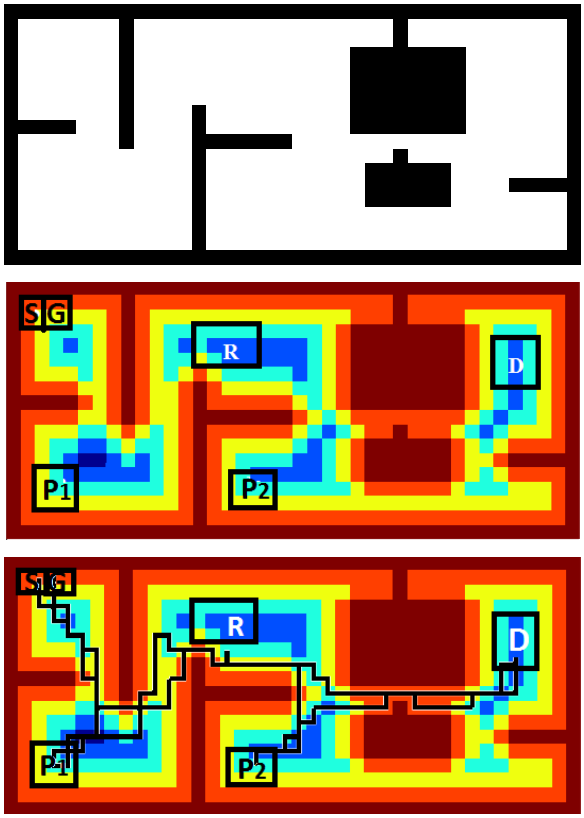


Figure 5.4: Factory experiment: Left: factory floor map. Middle : Risk map of the factory and its labels. Right: A sample path satisfying all properties.

1. Visit region A: $\varphi_1 = \Diamond A$
2. Visit region B: $\varphi_2 = \Diamond B$
3. Visit region C: $\varphi_3 = \Diamond C$
4. Visit region D: $\varphi_4 = \Diamond D$

Since the tasks are defined independent from each other, assuming the probability of visiting A is $P(A)$ and so on, the probability of successfully completing the mission is then $P(A)P(B)P(C)P(D)$. The desired probability for the four formulas is 0.6, 0.8, 0.5, and 0.9, respectively.

5.5.1.4 Results

To compute solving the policy, the linear program formerly outlined is solved, and we next simulated 1000 executions of the optimal policy to experimentally derive costs and success probabilities for each of the three problems. In this case all code is run in Matlab, and no efforts were

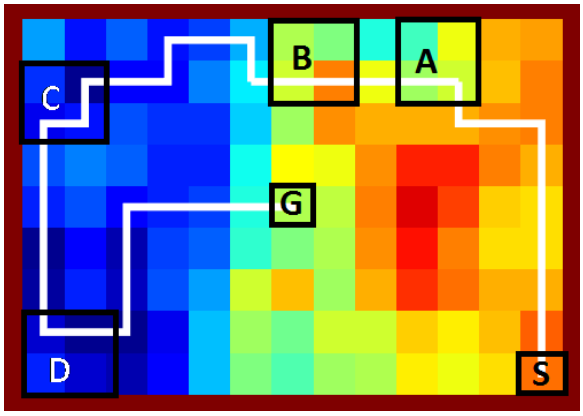


Figure 5.5: Rapid deployment experiment: A sample path satisfying all tasks.

made to optimize the code. Table 5.1 shows the time spent to perform the various steps in the solution. In particular, we show the time spent to compute the product between the initial LCMDP and the various formulas considered. The last column shows the time spent to solve the linear program using Matlab's builtin `linprog` function. The code was executed on a 2.5 GHz quadcore i7 CPU with 12 GB of RAM running Linux.

Experiment	φ_1	φ_2	φ_3	φ_4	Linear Program
Navigation	36.6	244.6	2427	N/A	91.6
Factory	157.6	1012.4	8928.4	N/A	158.5
Deployment	11.4	20.4	72.9	274.5	47.1

Table 5.1: Time spent for every LTL product and the time for solving linear programming problems (in seconds.)

As anticipated, the time grows with the number of formulas considered since the state space grows, and these numbers should also be compared with Table 5.2 for the number of states. In absolute terms, it should be considered that these policies are computed offline, as it is typical with MDP based planners, and therefore these numbers would not impact real world performance.

In Table 5.2 we show the number of states, i.e., the number of states in the original LCMDP (first column) and the number of states in the LCMDPs obtained after considering the various formulas. The last column (NP – no pruning) shows how many states would have been generated if the product was calculated without pruning unreachable states and thus shows the effectiveness and importance of the proposed pruning method. As it can be observed comparing the last two columns for each scenario, the reduction in the number of states is significant, and varies between a factor of 5 and 15. The precise number of course depends on the structure of the formulas considered and of the underlying LCMDP.

Finally in Table 5.3, we summarize the results of 1000 simulated executions of the optimal policy. For every experiment each column shows the number of times that the sink state of every extended-total-DFAs associated with each sc-LTL properties has been reached. To understand these

Experiment	Original	φ_1	φ_2	φ_3	φ_4	NP
Navigation	117	600	1526	5024	N/A	25272
Factory	608	1550	3136	9400	N/A	87552
Deployment	117	356	474	950	1894	29952

Table 5.2: Original number of states, and the number of states after each product operation. NP stands for not pruned.

results, it is important to recall that the sink state is reached when the associated sc-LTL formula is not satisfied. That is to say that for a given target probability P_φ, in N trials we would expect that the sink state is traversed $(1 - P_\varphi)N$ times. The table shows that this is indeed the case and small deviations from the desired value are due to the unavoidable approximation introduced when estimating the expectation from a limited set of test cases. For example, in the first row we see the number 311 under φ_1. This means that the sink state associated with formula φ_1 was traversed 311 times in 1000 realizations. This means that experimentally φ_1 was satisfied with probability 1-0.311=0.689. This is close to our desired target $P_{\varphi_1} = 0.7$. Similar conclusions can be drawn in all other cases. Total risk along the path for experiments I, II and III are 106.5, 1049.9 and 128.1 respectively. Average path length for experiments I, II and III is 24.55, 142.7 and 34.5 in order.

Experiment	φ_1	φ_2	φ_3	φ_4
Navigation	311	195	455	N/A
Factory	259	305	194	N/A
Deployment	393	188	495	102

Table 5.3: Number of times every sink state of every extended-total-DFA is reached.

5.5.2 Gazebo Simulations

Following the extensive Matlab experiments which showed the behavior of the planner when all the simulation parameters are known, we describe multiple simulations aiming at assessing strengths and limits of the planner in this section. In particular, we aim at showing its ability to express suitable missions in scenarios relevant to manufacturing and automation, and its robustness to modeling errors. We also provide a detailed description on how the planner can be implemented using the mainstream ROS software platform. For all our simulations we rely Gazebo, i.e., the standard simulation environment associated with ROS.

5.5.2.1 Environment Setup

We consider an autonomous forklift, i.e., an unmanned ground vehicle equipped with a forward facing gripper used to grab and release items. The Gazebo simulation environment does not provide a model for this vehicle, and therefore we developed one building upon the existing model for the Pioneer P3AT robot. Starting from the P3AT, we added four extra links and joints to enable the robot to restrain an object, and lift it. Figure 5.6 shows the model we developed. Note that on

Figure 5.6: On leftmost figure, the autonomous vehicle with its gripper fully opened at its lowest position. In the center figure, the vehicle carrying an object. Note that the fork as been closed and lifted. On the right, the environment used for testing.

top of the gripper we placed a laser range finder to localize the robot inside the known map of the environment.

From a mobility perspective the P3AT is a differential drive platform, and one could argue that forklifts instead use Ackermann steering, where the rear wheels are used for steering. While this is true, this distinction is immaterial to the nature of the experiments we present, because, as explained in the next subsection, navigation is fully handled by the ROS navigation stack that can manage different mobility configurations. The robot operates in known environment displayed in the rightmost panel of Figure 5.6. In our implementation a map of the environment is acquired upfront using one of ROS' built in SLAM algorithms. The map is then discretized into cells of 0.5×0.5 meters. With reference to the LCMDP formulation, the state of the robot is (x, y, c), where x, y identify the grid cell where the robot is located, and c is a binary variable indicating whether the robot is carrying something or not. The set of actions is $A = \{$*left*, *right*, *up*, *down*, *load*, *unload*$\}$. The first four actions describe motions in the grid, whereas the last two indicate the action on the gripper. Note that the orientation of the robot is missing from the state because thanks to the underlying motion controller the robot is capable of executing any of the four motion actions irrespectively of the direction it is facing. For what concerns the transition probabilities, when one of the *left*, *right*, *up*, *down* actions is executed, it succeeds with probability 0.63. This value was derived experimentally by simulating the actions multiple times and observing how often it succeeds. The action *load* succeeds with probability 0.6, whereas the action *unload* succeeds with probability 1.

5.5.2.2 Software architecture

From a software perspective we organized the system into the two-layer architecture shown in Figure 5.7. The top layer (Layer 2) is in charge of generating the policy by solving the linear program in Eq. (5.1), and to determine the desired action $\pi(s)$ as a function of the current state. Hence, it receives from the lower layer (Layer 1) the robot pose and the grasp status. These values determine the current state $s = (x, y, c)$, and the next action $\pi(s) \in A(s)$ can therefore be identified. If the action is either *left*, *right*, *up* or *down*, it is sent to the *Move Base* node in Layer 1, whereas if

the action is *load* or *unload* it is sent to the node *Load Controller*.

Layer 1 consists of a set of standard ROS nodes (*AMCL*, *Move Base*, *Map Server*) and of a custom developed node (*Load Controller*) in charge of the gripper device we introduced. AMCL handles the localization task using a particle filter. Given a map of the environment (provided by Map Server), and the data from the range finder and odometry, it provides an estimate of the robot pose. In our experiments, since the map is static, the estimate provided by AMCL is reliable and consistent with our hypothesis of state observability. Move Base implements the ROS mobility stack, i.e., drives the robot to a desired target location. Finally, Load Controller implements a PID algorithm to open/close the gripper and move it up or down.

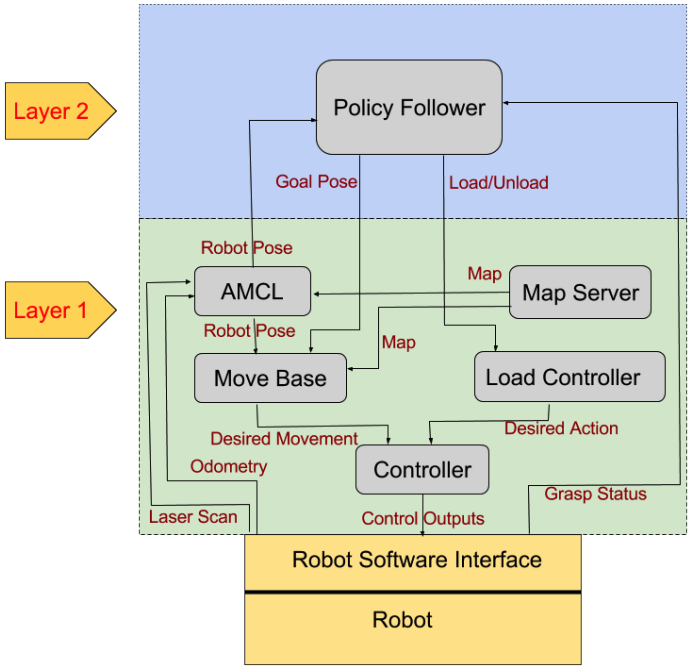


Figure 5.7: Software architecture

5.5.2.3 Costs, Atomic Propositions, and Tasks

According to the formulation presented in Section 3.3.2, we introduce two costs, i.e., risk (c_0) and traveled distance (c_1). The risk cost is defined by a risk map extracted from the environment, where locations near to obstacles are deemed more risky (see Figure 5.8). The overall risk along a path is the sum of the risk values of the cells traversed along the path. Next, we define two missions characterized by different atomic propositions and tasks.

Mission 1. For the first mission the set of atomic propositions is $\Pi = \{P_1, P_2, D, G, \mathcal{L}\}$, and the associated labeling function L is as follows (refer to the right panel in Figure 5.6).

- $L(P_1)$ is true when the robot is in the state corresponding to location *Pickup 1* and false otherwise.

Figure 5.8: Risk Map. Warmer colors indicate higher risk areas.

- $L(P_2)$ is true when the robot is in the state corresponding to location *Pickup 2* and false otherwise.
- $L(D)$ is true when the robot is in the state corresponding to location *Delivery* and false otherwise.
- $L(G)$ true when the robot is in the state corresponding to location *Goal* and false otherwise.
- $L(\mathcal{L})$ is true if the robot is carrying something (loaded) and false otherwise.

The following two tasks are defined as sc-LTL formulas over Π. In both cases, we assume that the robot starts without carrying anything.

1. Task 1: $\phi_1 = \neg\mathcal{L}\,\mathtt{U}\,P_1 \wedge (\bigcirc\mathcal{L}\,\mathtt{U}\,D \wedge (\bigcirc\neg\mathcal{L}\,\mathtt{U}\,G))$. In plain words this task requires to go to location *Pickup 1*, retrieve the element, bring it to location *Delivery*, release it, and then terminate in location *Goal*.
2. Task 2: $\phi_2 = \neg\mathcal{L}\,\mathtt{U}\,P_2 \wedge (\bigcirc\mathcal{L}\,\mathtt{U}\,D \wedge (\bigcirc\neg\mathcal{L}\,\mathtt{U}\,G))$. This task requires to go to location *Pickup 2*, retrieve the element, bring it to location *Delivery*, release it, and then terminate in location *Goal*.

We associate to ϕ_1 a probability $P_{\phi_1} = 0.4$ and to ϕ_2 a value $P_{\phi_2} = 0.3$. When solving the linear program, we minimize the risk cost, while setting a bound $B_1 = 70m$ on the cost c_1 (path length).

Mission 2. For the second mission the set of atomic propositions is $\Pi = \{S_1, S_2, P_1, P_2, D, G, \mathcal{L}\}$. The labeling function for $P_1, P_2, D, G, \mathcal{L}$ is as in Mission 1, whereas for S_1 and S_2 it is as follows (refer to Figure 5.9).

- $L(S_1)$ is true when the robot is in the state corresponding to location *Station 1* and false otherwise.
- $L(S_2)$ is true when the robot is in the state corresponding to location *Station 2* and false otherwise.

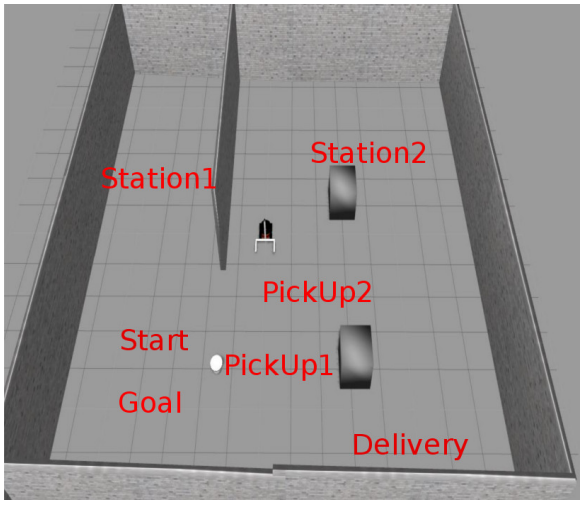


Figure 5.9: Relevant locations for the tasks in the second mission.

We define two tasks also for the second mission, and we again assume that the robot starts without carrying anything.

1. Task 1: $\phi_1 = \neg\mathcal{L}\,\mathrm{U}\,S_1 \wedge (\bigcirc\neg\mathcal{L}\,\mathrm{U}\,P_1 \wedge (\bigcirc\mathcal{L}\,\mathrm{U}\,D \wedge (\bigcirc\neg\mathcal{L}\,\mathrm{U}\,G)))$. Informally speaking, the robot is required to go to *Station 1* while unloaded, receive some orders, then drive to *Pick Up 1* to pick up an object. Then it unloads the object at the *Delivery* location and terminates the task by going to the *Goal* location.

2. Task 2: $\phi_2 = \neg\mathcal{L}\,\mathrm{U}\,P_2 \wedge (\bigcirc\mathcal{L}\,\mathrm{U}\,S_2 \wedge (\bigcirc\mathcal{L}\,\mathrm{U}\,D \wedge (\bigcirc\neg\mathcal{L}\,\mathrm{U}\,G)))$. In this task the robot goes to *Pick Up 2* location, loads the object. Then it drives towards *Station 2* which can be a check point in order to verify the status of the object being carried. After the check point, it moves towards the *Delivery* location to drop it off and then drives to the *Goal* location.

We associate to ϕ_1 a probability $P_{\phi_1} = 0.4$ and to ϕ_1 a probability $P_{\phi_2} = 0.3$. When solving the linear program, we minimize the risk cost (c_0), while setting a bound $B_1 = 100m$ on the path length (c_1).

5.5.2.4 Results

Starting from the experimental setup we described in the previous section, we solve the linear program corresponding to the cases described in the first and second mission, and we then repeatedly executed the policy in Gazebo, logging the results in terms of traveled path, accrued risk, and percentage of accomplishments of the specified tasks. In every experiment the policy is executed 500 times. For both missions we first solve the linear program with values for the transition probability equal to the probabilities we experimentally determined, as described earlier. Next, to assess the robustness of the algorithm to modeling errors, we repeat the same experiments after having solved the linear program using transition probabilities that differ from the value we observed, and evaluate how the performance degrades.

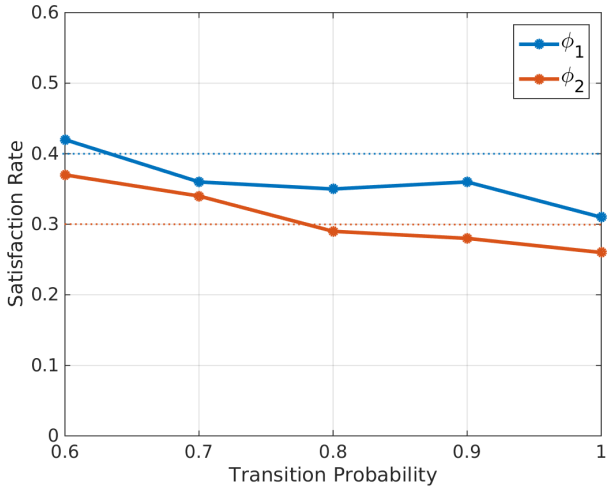


Figure 5.10: Satisfaction rates of two tasks in mission 1.

5.5.2.4.1 Mission 1 In the first batch of experiments, we solve the linear program using our best estimates for the transition probabilities. Table 5.4 shows the results we obtained. It can be seen that over 500 runs the bound $B_1 = 70$ on the path is on expectation met with a rather small variance.

Quantity	Mean	Variance
Risk	362.9	64.2
Path Length	61.2	9.3

Table 5.4: Mission 1 results in terms of total accrued cost and path length

Moreover, ϕ_1 was satisfied with probability 0.43 and ϕ_2 and with probability 0.25. The reason for the mismatch between $P_{\phi_2} = 0.3$ and the observed value of 0.25 is twofold. First, the number of samples is relatively small and then does not necessarily converge to the expected value. Second, as we will show in the next experiment, it seems like that the transition probability value we experimentally estimated is not extremely accurate.

Next, to experimentally evaluate the robustness of the algorithm to errors in the transition probabilities, we solved again the linear program using transition probability values different from the one we experimentally determined. Figure 5.10 shows the satisfaction rates for ϕ_1, ϕ_2 as the transition probability varies, whereas figure 5.11 shows the Kullback-Leibler divergence between the desired values for P_{ϕ_1} and P_{ϕ_2} and the observed values. From this second figure we can observe that 0.7 appears to be a better estimate for the transition probability.

5.5.2.4.2 Mission 2 Similar experiments were performed for mission 2. Table 5.5 confirms that the policy satisfies in expectation the bound $B_1 = 100$ for the expected path length.

In this case ϕ_1 was satisfied with probability of 0.49, and ϕ_2 with probability of 0.25. As compared to the values in the setup step, the desired value for P_{ϕ_2} is slightly off from the desired

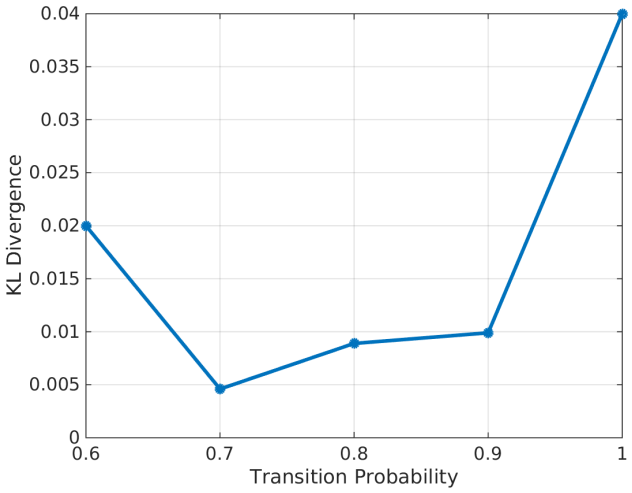


Figure 5.11: KL Divergence for mission 1.

Quantity	Mean	Variance
Risk	556.4	89.4
Path Length	102.6	18.2

Table 5.5: Mission 2 results in terms of total accrued cost and path length

value. In our interpretation this is due to the fact that the transition probability value we used to solve the linear program is not necessarily the one best fitting the underlying model, as confirmed by Figure 5.12 and Figure 5.13 in which we observe the performance of the planner as the transition probability vary.

We conclude this section outlining that the presented results, although not completely aligned with the desired results, do not undermine the correctness of the planner. In fact, in the previous section we showed that if the linear program in Eq. 5.1 is solved using transition probabilities accurately matching the underlying model, then all target probabilities are achieved.

5.5.3 Real World Experiments

To further validate our proposed method, we performed additional experiments using a mobile platform operating outdoor. Figure 5.14 shows the system we used, i.e., a Husky robot marketed by Cleaprath Robotics. All software controlling the robot is based on ROS Indigo and runs on the laptop onboard the robot. The robot is equipped with a SICK for obstacle avoidance. For localization it uses two Hiper SR GPS receivers by Topcon (seen on top of the yellow and red posts), and a myAHRS+ inertial navigation unit. Data from these sensors, as well as odometry readings and motion commands, is processed by a ROS node providing localization using an Extended Kalman Filter. The system was tested outdoor on the campus of the University of California, Merced (see Figure 5.15), and drove more than 7.8 km during the missions described in the following.

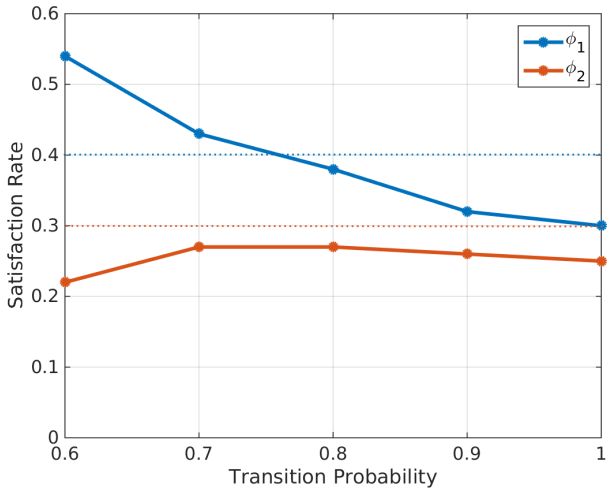


Figure 5.12: Satisfaction rates of two tasks in mission 2.

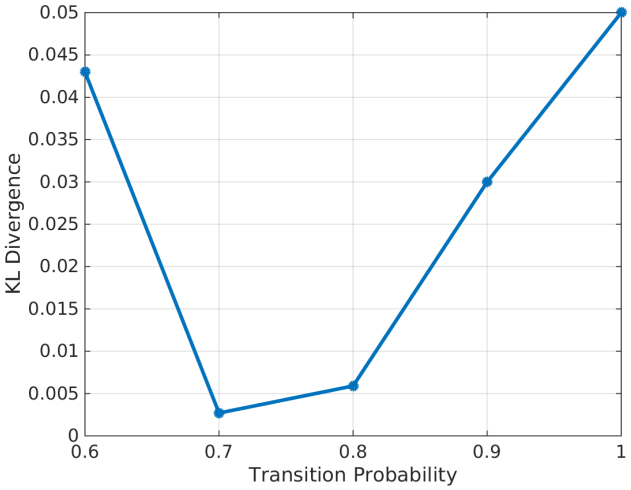


Figure 5.13: KL Divergence for mission 2

Figure 5.14: The Husky robot used to validate the planner proposed in the paper.

A map for the campus was retrieved from Google maps and discretized into rectangular cells. For the first two experiments the environment consisted of a rectangle of size $102m \times 115m$ discretized into cells of $1.86m \times 1.88m$. In the third scenario the environment consisted of a rectangle of size $203m \times 232m$ discretized into cells of $2.36m \times 2.35m$. To align the real world experiments as much as possible with the simulations presented in the previous section, a risk value was associated to each grid cell based on the distance from the closest obstacle. Figure 5.16 shows the grid cells and risk maps for the first two testcases, whereas Figure 5.17 displays the grid cells and risk maps for the last case. As for the simulations, warmer colors indicate higher risk.

Solving the linear program in Eq. (5.1), a policy consisting of *up/down/left/right* actions is determined. To execute the commands, the ROS navigation stack was used, where the center location of the desired grid cell to reach was given as goal to the navigation stack. Note that knowing the current location on the map and the desired action, this point can be univocally determined. Moreover, as in the simulated case, the orientation of the robot is not considered during the planning stage, but is of course taken into account by the navigation stack. In all experiments, two cost functions c_0 and c_1 were used, as in the simulations. The first one, to be minimized, accounts for the cumulative risk along the path, whereas the second one considers that path length to be bounded. Each mission was repeated 15 times.

5.5.3.1 Pickup and Delivery

During this mission, the robot starts from a point, drives to a pickup station, then to a delivery station, and finally goes to the goal point. Two pairs of pickup/delivery stations were defined and associated with two sc-LTL formulas. Figure 5.18 (left) shows the labels relevant for the formulas

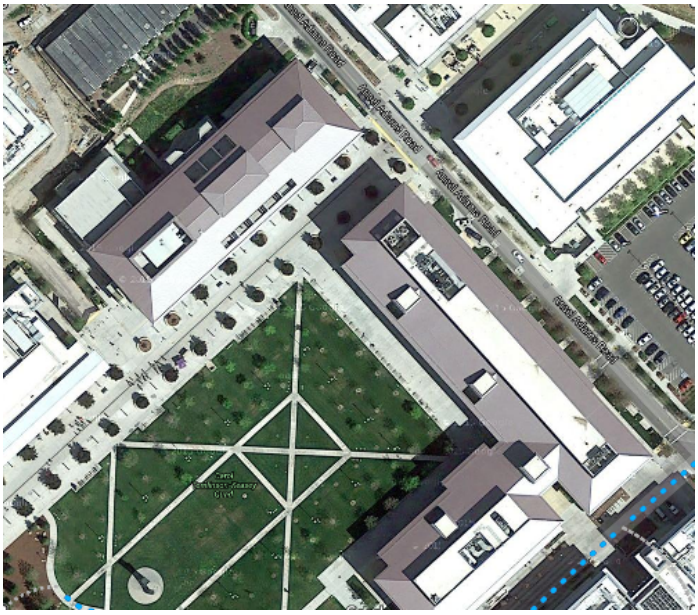


Figure 5.15: Aerial view of the campus area where the navigation experiments took place.

describing the tasks:

- Task 1: pickup from location A, deliver to location B and go to the goal location: $\varphi_1 = \diamond(A \wedge \bigcirc\diamond B(\wedge \bigcirc \diamond G))$.
- Task 2: pickup from location C, deliver to location D and go to the goal location: $\varphi_2 = \diamond(C \wedge \bigcirc\diamond D(\wedge \bigcirc \diamond G))$.

Both tasks were given a target probability of 0.4. The bound on path length was set to 100.

5.5.3.1.1 Results Sample paths for both tasks in mission I are shown in Figure 5.19. Among the 15 experiments, 6 paths satisfied task 1 and 7 paths satisfied task 2 while 3 paths did not satisfy any of the tasks. The average path length among all the cases is 90.92 where the unit of measurement is the number of cells which is approximately 167m. Meanwhile the total accrued risk is 621.

5.5.3.2 Navigation

Three tasks are defined for this mission according to the label map in the right panel of Figure 5.18. The area labeled as A is a forbidden area where the robot should not enter and is represented in the first task. A goal area G is evidenced and should be entered either by the left area B or the right area C. These tasks are formalized in sc-LTL as follows.

- Task 1: stay out of area A: $\varphi_1 = (\neg A)\ \mathtt{U}\ G$
- Task 2: enter the goal location G from the left side $\varphi_2 = \diamond(B \wedge \bigcirc(B\ \mathtt{U}\ G))$

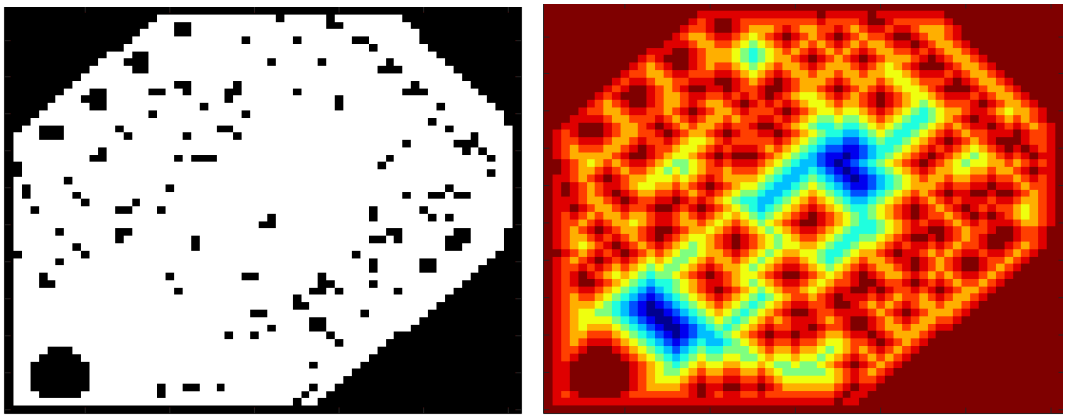


Figure 5.16: Grid map (left) and risk map (right) for the first two scenarios. On the map, black grid cells are associated with untraversable areas. These are obstacles like trees, benches, sculptures or areas that the robot shall not enter.

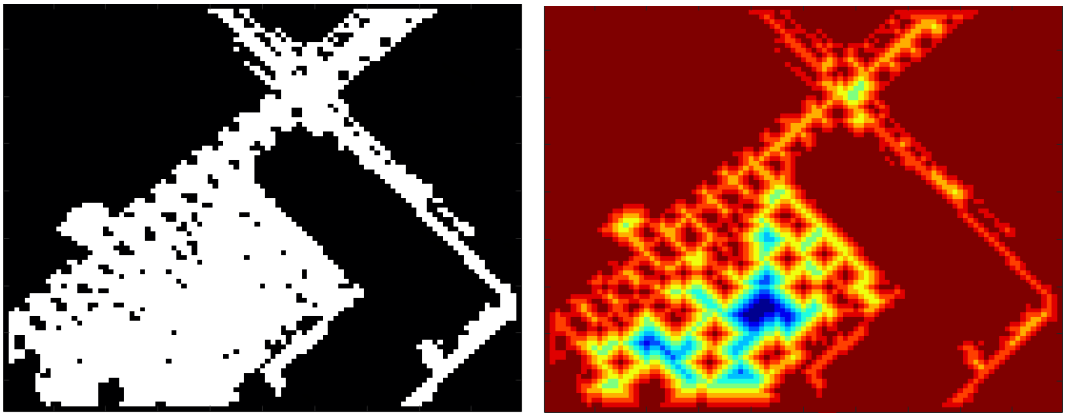


Figure 5.17: Grid map (left) and risk map (right) for the third scenario.

- Task 3: enter the goal location G from the right side $\varphi_3 = \Diamond(C \wedge \bigcirc(C \,\mathtt{U}\, G))$

Formula φ_1 should be satisfied with probability 0.9, φ_2 with probability 0.5, and φ_3 with probability 0.4. The bound on path length was set to 90. Note that φ_1 resembles the construct *always* that is not available in sc-LTL. This is possible because we impose to negate atomic proposition A until the goal location G is reached.

5.5.3.2.1 Results As in the previous case, the Navigation mission was repeated 15 times where task 1 was satisfied in 14 paths, task 2 in 7 paths and task 3 in 7 paths. Two sample paths for this mission are shown in Figure 5.20. The average total path length is 71.2 or approximately 130m. The total accrued risk is 340.8.

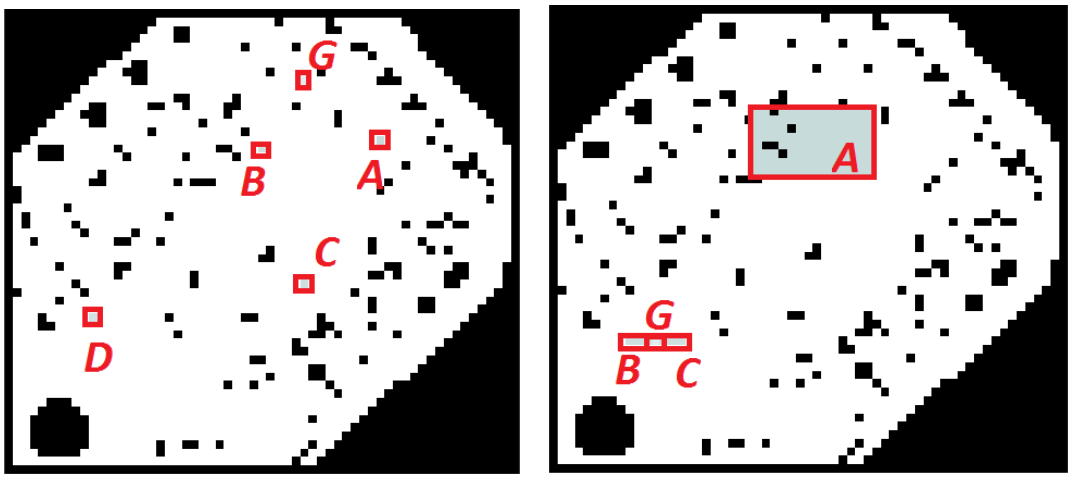


Figure 5.18: Left: labels for the *Pickup and Delivery* task. Right: labels for the *Navigation* task.

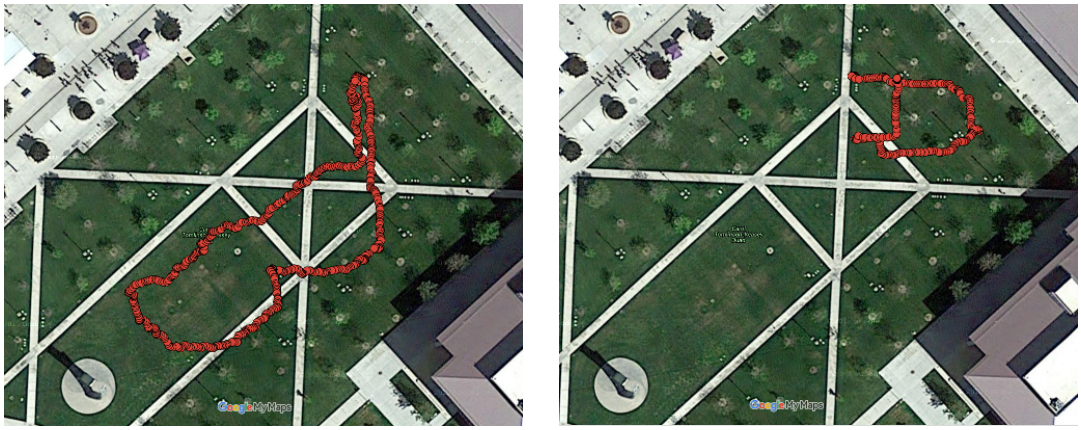


Figure 5.19: Sample paths obtained during the Pickup and Delivery tests.

5.5.3.3 Autonomous Driving

This mission features three tasks resembling situations arising in autonomous driving. We define a label map for this mission as in Figure 5.21.

Informally speaking, the street is labeled as A while a pedestrian crossing is labeled as B. A bike rack is marked with C while a charging station for the robot is labeled with D. According to these labels, the following three tasks are defined:

- Task 1: Avoid the bike rack: $\varphi_1 = (\neg C)\ \mathtt{U}\ G$;
- Task 2: Do not cross the street anywhere other than the pedestrian area. $\varphi_2 = (\neg A)\ \mathtt{U}\ G$;
- Task 3: Visit the charging station: $\varphi_3 = \Diamond D$;

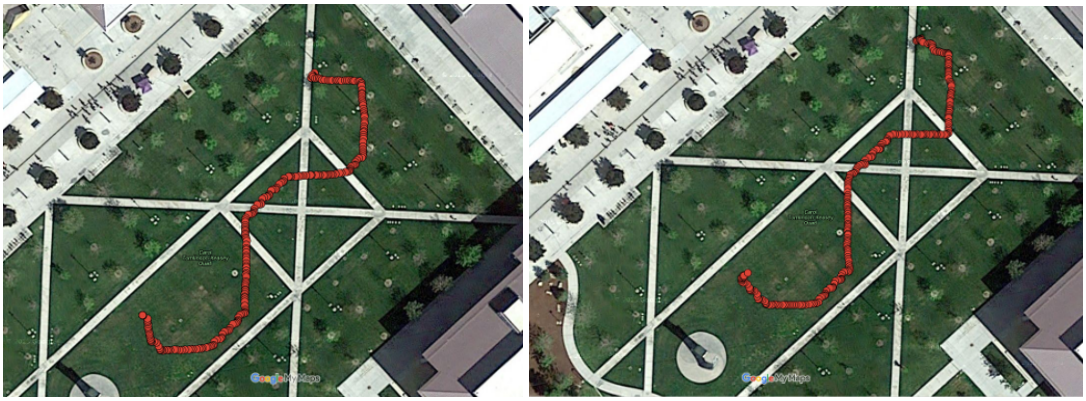


Figure 5.20: Sample paths obtained during the Navigation test.

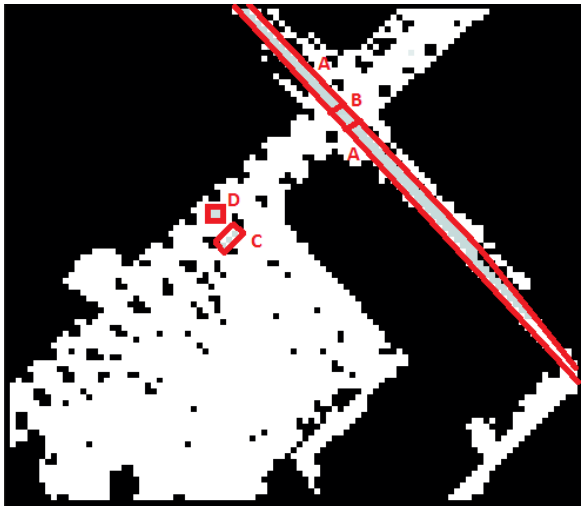


Figure 5.21: Label Map for mission III

The first formula has a target probability of 0.9, the second has a probability of 0.9, and the third has a probability of 0.5. The bound on path length was set to 110.

5.5.3.3.1 Results In 15 runs, all paths all satisfied the requirements for task 1 while 14 of them satisfied task 2. Task 3 was realized in 7 paths. Two sample paths are shown in 5.22. The average path length for this mission was 95 and the average total risk is 677.

5.6 Conclusions

In this chapter, we have presented a novel planning algorithm capable of simultaneously considering multiple costs and multiple tasks expressed as sc-LTL formulas. In particular, each task is associated with a different target probability and the planner produces a policy that in expectation

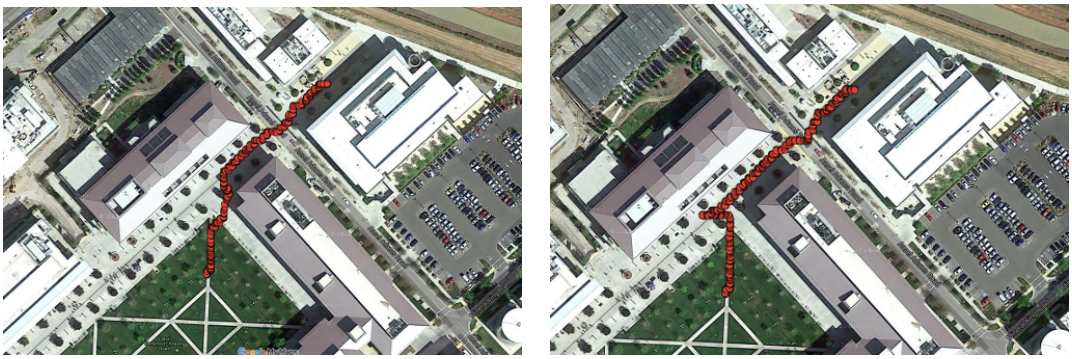


Figure 5.22: Mission III paths. The left picture satisfies task 1 and 2 while the right picture satisfies all three tasks.

will satisfy each formula with the target probability. The algorithm builds upon the theory of constrained MDPs, and is therefore suitable for situations where dynamics are non-deterministic but the state is observable. The key insight of our approach is the definition of a product between the CMDP and the DFAs defining the good prefixes satisfying the given sc-LTL formulas. In particular, this definition allows to precisely compute the probability that a certain state in the product definition will be reached during a stochastic evolution, and this can be associated with the probability that a given sc-LTL formula is not satisfied. A vanilla implementation of the product definition would lead to an unmanageable growth in the size of the state space and therefore a suitable pruning algorithm was developed. Our approach can be extended in different ways. The order in which the DFAs are multiplied with LCMDP has an impact on the size of the product and could be optimized. Another extension we will consider is risk aversion, i.e., producing policies that not only minimize expectations, but bound variance from the expectation.

Chapter 6

Improvements to Multi Objective Planning with Multiple Costs and Multiple High Level Task Specifications

6.1 Introduction

In chapter 5 we presented a planner that solves multi-cost and multi-task planning problems in non-deterministic environments by combining the theory of constrained Markov decision processes (CMDPs) with linear temporal logic (LTL). In such scenario a robot faces the challenge of finding a path to a goal location while being subject to multiple cost functions. Besides that, the robot is required to accomplish some tasks with specified target probabilities. The major innovation of our proposed planner is the ability to generate optimal plans which satisfy both types of constraints at once. For example, the planner may aim at minimizing the time to complete a mission while at the same time: 1) obey to constraints on multiple other cost functions, like consumed energy, traveled length, etc.; 2) probabilistically satisfy n tasks expressed in LTL, where every task is associated with a different target probability. For example consider a factory forklift robot. The robot drives to a pick-up station, loads an object, stays loaded while driving toward a drop-off location and unloads the object. We emphasize defining tasks by LTL properties because it has emerged as a very practical tool to easily specify new tasks that is easy to understand even for non-specialists. Our approach builds upon the definition of a product operation between the CMDP state space and the states of the discrete finite state automata associated with each of the LTL formulas.

While this approach has been successfully validated both in simulation and in extensive outdoor navigation experiments, it also has some drawbacks related to its scalability. In particular, even though the method eventually requires the solution of a linear program that can be efficiently solved, determining the state space defining the parameters of the linear program can be time consuming, in particular when many different tasks expressed in LTL are considered. This preliminary step is usually the most time consuming stage of the algorithm. In Chapter 5, we made no attempt to optimize the process aiming at considering all LTL formulas and all objective functions at once. In this chapter, we address this issue by proposing two techniques to improve the previous work as

following:

- Order of Operations: The order in which the various LTL formulas are considered during the planning stage does not influence the final result. However, it has a significant impact on the processing time. To overcome this limit, in this chapter we propose a step to optimize the order in which the various formulas are considered, and show that this can lead to a 50% improvement in the performance.

- Pruning: During the product calculation process, composite states satisfying only a subset of the LTL formulas are produced. In some instances there can be a large number of such states that are relevant only to exactly solve the given planning problem, i.e., to satisfy the LTL formulas with the given probability. In this chapter we present a method to eliminate these states, thus obtaining an approximate but faster solution.

The remainder of this chapter is organized as follows. Section 6.2 gives the problem formulation and the discusses the proposed improvements. In Section 6.3, we present some experiments substantiating our results and finally in Section 6.4, we draw conclusions and discuss possible future work.

6.2 Improvements

In this section we present the improvements to the method we presented in Chapter 5. The proposed improvements only target the length of the calculations. The rest of this chapter explains each of the improvements with their advantages and disadvantages. The readers are referred to Chapter 3 and 5 for the required background knowledge.

6.2.1 Order of Operations

Given an LCMDP $\mathcal{M} = (S, \beta, U, c_i, \Pr, AP, L)$ and m sc-LTL formulas $\phi_1, \ldots, \phi_m$ over AP, our method requires to compute the product between $\mathcal{M}$ and all the DFAs associated with the m formulas, i.e., $\mathcal{M} \otimes \phi_{i_1} \otimes \phi_{i_2} ... \otimes \phi_{i_m}$. Note that since there is a one-to-one matching between DFAs and formulas, we use the same symbol for both. The product is commutative, but the order influences the computational time and there can be significant variations. To see why the order is relevant, consider the small example shown in Figure 6.1. The original LCMDP $\mathcal{M}$ is shown on the left, with two DFAs are presented alongside. Since the product is commutative, $\mathcal{M} \otimes \mathcal{D} \otimes \mathcal{D}' = \mathcal{M} \otimes \mathcal{D}' \otimes \mathcal{D}$ as illustrated in Figure 6.2. But in the two cases the intermediate steps are different as shown in Figure 6.3. $\mathcal{M} \otimes \mathcal{D}'$ produces one more state which requires additional computation. On large scale examples, there can be many such instances, and the compounded effect is significant. Let $Nr(\mathcal{M})$ be the number of states in the LCMDP $\mathcal{M}$.The total number of states which require processing is:

$$N = Nr(\mathcal{M} \otimes \phi_1) + \cdots + Nr(\mathcal{M} \otimes \phi_1 \otimes \cdots \otimes \phi_m)$$

Having that $Nr(\mathcal{M} \otimes \phi_1) > Nr(\mathcal{M})$, we know $Nr(\mathcal{M} \otimes \phi_1) = Nr(\mathcal{M}) + n_1$ where n_1 is a positive integer. Then,

$$N = (Nr(\mathcal{M}) + n_1) + \ldots (Nr(\mathcal{M}) + n_1 \ldots n_m)).$$

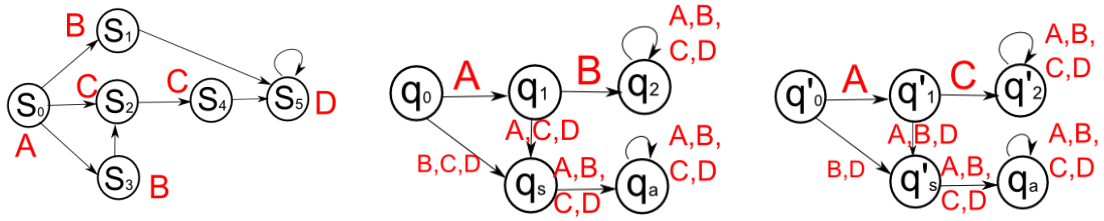


Figure 6.1: The left picture is the original LCMDP. The DFA $\mathcal{D}$ is shown in the middle and $\mathcal{D}'$ is illustrated in the right picture.

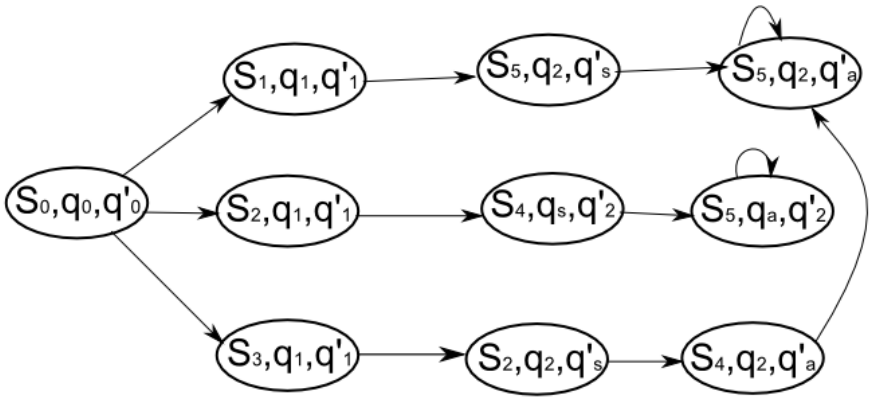


Figure 6.2: The final product $\mathcal{M} \otimes \mathcal{D} \otimes \mathcal{D}' = \mathcal{M} \otimes \mathcal{D}' \otimes \mathcal{D}$.

n_1 is repeated m times, similarly n_2 is repeated $m-1$ times and so on. To minimize N, the following inequality has to hold

$$n_1 \leq n_2 \cdots \leq n_m$$

This means the DFA which produces the smallest LCMDP has to be applied first in the product operation, and so on, recursively. The problem is then how to estimate $Nr(\mathcal{M} \otimes \phi)$ before calculating the product completely. In the following subsection we show how to efficiently estimate this value.

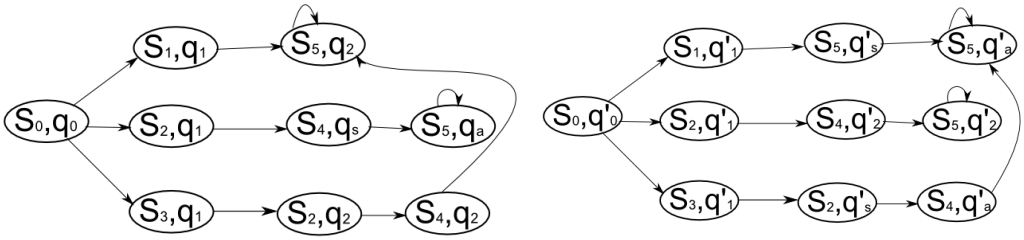


Figure 6.3: Intermediate product results. Top: LCMDP as the result of $\mathcal{M} \otimes \mathcal{D}$. Bottom: $\mathcal{M} \otimes \mathcal{D}'$

6.2.1.1 Proposed Solution

Our solution consists of three steps.

1. In the first step, we build an aggregate LCMDP from the original LCMDP by merging all states with similar labels into a single aggregated state under the condition of connectivity. This means that if there is a valid path between two states with the same label, and all of the intermediate states involved in the path have the same label, the two states can be merged into the same aggregate state.

2. The second step is to calculate the product between all the DFAs and the aggregate LCMDP.

3. Step three is to evaluate the resulting product LCMDPs to estimate how many original states will be generated.

To further illustrate step 1, we plotted a simple example in Figure 6.4. If the original LCMDP is represented by $\mathcal{M} = (S, \beta, U, c_i, \Pr, AP, L)$, and the aggregate LCMDP that is computed from $\mathcal{M}$ is shown by $\mathcal{A} = (S_{\mathcal{A}}, \beta_{\mathcal{A}}, U_{\mathcal{A}}, c_{\mathcal{A}_i}, \Pr_{\mathcal{A}}, AP, L_{\mathcal{A}})$, normally $|S_{\mathcal{A}}| << |S|$ which makes the product calculation faster. Algorithm 7 generates the aggregate LCMDP $\mathcal{A}$ from LCMDP $\mathcal{M}$. In the algorithm, we assume to have the following functions:

- *primaries*: Takes in an aggregate state and returns the set of primary states in that.

- *aggregate*: Takes in a primary state and returns the aggregate state that it belongs to. The function returns empty set if it has not been assigned yet.

- *post*: Takes in a primary state, and returns a set of primary states with non-zero probability of reaching in one step.

Step two has been discussed in Chapter 5. The number of states in an aggregate LCMDP is equal to sum of its primary states (Step three).

6.2.2 Pruning the LCMDP

The result of a product between an LCMDP $\mathcal{M} = (S, \beta, U, c_i, \Pr, AP, L)$ and a DFA $\mathcal{D} = (Q, q_0, \delta, F, \Sigma)$ is a product LCMDP. In the product LCMDP, the set of states can be divided into two subsets:

- Rejected Set: Any state that contains a q_a or q_s has already been rejected.

- Acceptable Set: Any other state which has a chance of being accepted.

Since the most important subset is the Acceptable Set, we can prune away some of the states in the Rejected Set, to save calculation time. The parameter R is defined as the percentage of states in the Rejected Set, or non-accepting states, to be removed. In case $R = 0$ all the non-accepting states will remain in the product LCMDP, and in case of $R = 100$ the product LCMDP will only

Algorithm 7: Aggregation of an LCMDP

Data: LCMDP, $\mathcal{M} = (S, \beta, U, c_i, \Pr, AP, L)$
Result: LCMDP $\mathcal{A} = (S_{\mathcal{A}}, \beta_{\mathcal{A}}, U_{\mathcal{A}}, c_{\mathcal{A}_i}, \Pr_{\mathcal{A}}, AP, L_{\mathcal{A}})$

Init LCMDP $\mathcal{A}$ empty;
Build $S_{\mathcal{A}}$ from a random state s where $\beta(s) > 0$;
$A_{\mathcal{A}} = \emptyset$; $\beta_{\mathcal{A}}(S_{\mathcal{A}}) = 1$; $S_{seen} = \emptyset$;
while *There is a change* **do**
    select $s_{\mathcal{A}} \in S_{\mathcal{A}}$;
    for $s \in primaries(s_{\mathcal{A}})$ **do**
        $S' = post(s)$;
        for $s' \in S'$ **do**
            if $L(s) = L(s')$ **then**
                if $aggregate(s') \neq \emptyset$ **then**
                    merge $aggregate(s')$ into $s_{\mathcal{A}}$;
                if $s' \notin S_{seen}$ **then**
                    Add s' to $primaries(s_{\mathcal{A}})$;
            else
                if $s' \notin S_{seen}$ **then**
                    Build $s'_{\mathcal{A}}$ from s' ;
                    $S_{\mathcal{A}} = S_{\mathcal{A}} \cup s'_{\mathcal{A}}$;
                    $U_{\mathcal{A}} = U_{\mathcal{A}} \cup \text{dummy}_{s'_{\mathcal{A}}}$;
                    $Pr_{\mathcal{A}}(s_{\mathcal{A}}, s'_{\mathcal{A}}, \text{dummy}_{s'_{\mathcal{A}}}) = 1$;
                    $L_{\mathcal{A}}(s'_{\mathcal{A}}) = L(primaries(s_{\mathcal{A}}))$;
        $S_{seen} = S_{seen} \cup \{s\}$

contain the set of accepting states. It is important to note that a rejected state for a DFA $\mathcal{D}_1$ might lead to an accepting state in $\mathcal{D}_2$.

Given an LCMDP $\mathcal{M} = (S, \beta, U, c_i, \Pr, AP, L)$ with a single absorbing state s_a and a DFA $\mathcal{D} = (Q, q_0, \delta, F, \Sigma)$ with an absorbing state q_a, product $\mathcal{M} \otimes \mathcal{D} = (S_p, \beta_p, U_p, c_{i_p}, \Pr_p, AP, L_p)$ will contain a set of states $S_a = \{(s, q) \in S_p | q = q_a\}$. S_a is the set of states in S_p where the words are rejected. This set contains one specific state $s_f = \{(s, q) \in S_a | s = s_a\}$ which is the non-accepting absorbing state of the LCMDP. Every state in S_a cannot be accepted by the DFA $\mathcal{D}$ and all the trajectories which contain at least one state in S_a will be absorbed at s_f. In this pruning step we aim at eliminating a subset of states in S_a that are less effective in the final result. This is done as follows.

1. Calculate the product LCMDP, then extract S_a and s_f from that.
2. Revert the graph of LCMDP.
3. Calculate the shortest path using Dijkstra algorithm from s_f to all the states in S_a.
4. Sort the states by the number of times they appear in the shortest path to s_f.

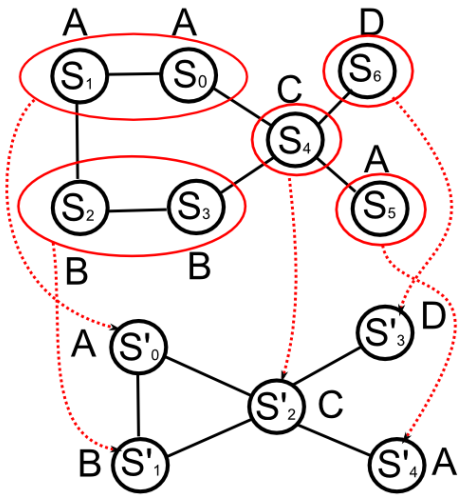


Figure 6.4: An example of an original LCMDP and its aggregate LCMDP. The top picture shows the original LCMDP and the lower figure is its equivalent aggregate LCMDP.

5. Remove the given percentage, R, of states from the ordered list where least repeated states are removed first.

It is important to note that states $\{(s,q) \in S_p | q = q_s\}$ are also non-accepting, but cannot be removed from the set of states since they are required to calculate the probability of satisfying the DFAs.

6.3 Experimental Evaluation

We use a planning problem in the terrain map shown in Figure 6.5, where the actions are up, down, left and right at every state. Actions succeed with probability 0.9 if the target state is lower than the current state, and 0.5 otherwise. In case of failure the robot stays at its current location or moves to one of its neighbors with uniform probability distribution. We assign a risk value to every state between zero and ten that is proportional to the altitude of the state. In other words, the highest point in the map is assigned the risk value of ten, and the lowest point has zero risk. The objective is to calculate a policy that finds a path to the goal area G in the picture. The primary cost function to be minimized is the total accrued risk along the path. Moreover, there is a constraint on the path length, and are tasks to be satisfied. We defined two sets of tasks, one for evaluating the effects of order of operations, and the other for pruning.

6.3.0.1 Order of Operations

We defined five sc-LTL properties and calculated the product between the original LCMDP extracted from the terrain map and all DFAs. The tasks are:

- $\phi_1 = \diamond A$
- $\phi_2 = \diamond B$
- $\phi_3 = \diamond C$
- $\phi_4 = \diamond((A \,\mathtt{U}\, B) \wedge \bigcirc\diamond(C \wedge \bigcirc\diamond D))$
- $\phi_5 = \diamond(A \wedge \bigcirc\diamond(C \wedge \bigcirc\diamond(D \wedge \bigcirc\diamond C)))$

In plain words, ϕ_1, ϕ_2 and ϕ_3 require the robot to eventually reach the areas labeled by A, B and C respectively. ϕ_4 tasks the robot to go to area A, stay in A until leaving it from its border to area B. Then it eventually visits C, next eventually D. ϕ_5 forces the robot to eventually visit areas A, C, D and C in order.

6.3.0.2 Pruning

For the pruning experiment, we selected three different scenarios:

1. We choose three sc-LTL properties where they are fully related to each other as following:

   - $\phi_1 = \diamond A$
   - $\phi_2 = \diamond(A \wedge \bigcirc\diamond B)$
   - $\phi_3 = \diamond(A \wedge \bigcirc\diamond(B \wedge \bigcirc\diamond C))$

   Again in plain words, ϕ_1 requires the robot to eventually reach area A. ϕ_2 eventually drives the robot to A, then it eventually visits B. Similarly, ϕ_3 asks the robot to visit areas A, B and C in order. If a word w is accepted by ϕ_3 it is also accepted by ϕ_2 and ϕ_1. That means even if we remove all the rejected states by ϕ_1, there will be no effect on the final result.

2. Three independent sc-LTL properties are selected at this step as following: $\phi_1 = \diamond A, \phi_2 = \diamond B, \phi_3 = \diamond C$. In this case, removing some states from the first rejected influences the next product operation, and causes estimations of the final policy.

3. Two incompatible sc-LTL properties are selected: $\phi_1 = \diamond A, \phi_2 = \neg A \,\mathtt{U}\, G$.

6.3.0.3 Order of Operations

The summary of results between the best and worst performance are shown in Table 6.1. The results show that instead of calculating the product for the total of 894037 nodes, we only need to calculate it for 540551 nodes and obtain the same results. It also reduces the computation time from 2388.9 seconds to 1425.4.

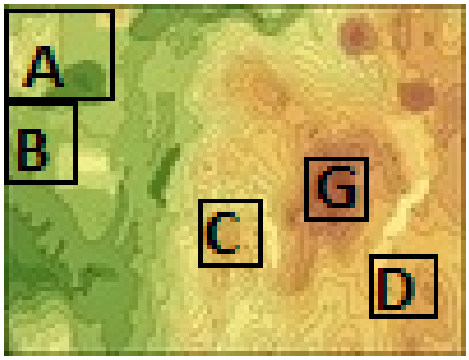


Figure 6.5: Terrain map with labels.

Case	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	Total
Best	29013	36183	58209	160406	256740	540551
Worst	50814	87589	242161	256733	256740	894037

Table 6.1: Number of computed states after each product operation in order of operations experiment

6.3.0.4 Pruning

Three different cases where considered.

- **Case I** We set the satisfaction rate on ϕ_1 to 0.7, ϕ_2 to 0.6 and ϕ_3 to 0.5. Table 6.2 shows the effect of pruning on the number of states. It can be seen that it can reduce the number of states by almost half without any change in the final result. However, if the sc-LTLs were different, where the amount of overlap between them was less, the reduction would show better improvements but without any approximation. The calculation time drops from 431 to 177 seconds without any change on the final policy (See Table 6.3). Moreover, the linear program solver succeeded in all three cases, i.e., there is no loss in terms of results.

- **Case II** With the same probability rates, Table 6.4 presents the results in terms of the number of states. The gain on the number of states is better than case I, but it sacrifices the convergence. So the linear program only converges for pruning of 0, 25% and 50%. Similarly, Table 6.5 shows the effect of pruning on calculation time.

- **Case III** This case is different because the two sc-LTL formulas contradict each other. Therefore, we set the satisfaction rate of 0.4 for both of them. Even though Table 6.6 illustrates a good improvement, convergence rate is significantly impacted, and it only converges if the pruning rate is 0 which means no pruning. In any other case, it fails to find an optimal solution. The calculation time is also shown in Table 6.7 which shows around 80% improvement.

Pruning Percentage	ϕ_1	ϕ_2	ϕ_3	Total
0%	29027	36350	53604	118981
25 %	27214	32624	23518	83356
50 %	25401	29000	22028	76429
75 %	23589	23512	21962	69063
100%	21776	22117	21910	65803

Table 6.2: Number of computed states (Pruning Case I)

Pruning Percentage	ϕ_1	ϕ_2	ϕ_3	Total
0%	63.2	105.2	262.8	431.2
25 %	64.2	92.6	70.6	227.4
50 %	65.7	87.5	66.7	219.2
75 %	66.4	72.7	65.6	204.7
100%	64.4	51.6	61.4	177.4

Table 6.3: Time in seconds(Pruning Case I)

Pruning Percentage	ϕ_1	ϕ_2	ϕ_3	Total
0%	29027	65445	160009	254481
25 %	27214	57000	88502	172716
50 %	25401	41252	74139	140792
75 %	23589	34659	45599	103847
100%	21776	30453	37534	89763

Table 6.4: Number of computed states (Pruning Case II)

Pruning Percentage	ϕ_1	ϕ_2	ϕ_3	Total
0%	65.9	186.2	556.5	808.6
25 %	70.15	177.2	278.1	525.45
50 %	71.8	140.9	255.4	468.1
75 %	74.8	134.5	200.1	409.4
100%	83.5	131	198.2	412.7

Table 6.5: Time in seconds(Pruning Case II)

Pruning Percentage	ϕ_1	ϕ_2	Total
0%	29027	36375	65402
25 %	27214	30550	57576
50 %	25401	21762	47163
75 %	23589	13320	36909
100%	21776	3843	25619

Table 6.6: Number of computed states (Pruning Case III)

Pruning Percentage	ϕ_1	ϕ_2	Total
0%	81.2	107.2	188.4
25 %	74.2	95.3	169.5
50 %	72.5	84.1	156.6
75 %	77.5	53.5	131
100%	78.8	21.5	100.3

Table 6.7: Time in seconds(Pruning Case III)

6.4 Conclusions

The approach presented in Chapter 5 is effective and has been validated in various scenarios such as Matlab simulations and on real robot, but it falls short when the size of its state space grows. Since the product operations are iterative, and every iteration increases the size of the state space, it is foreseeable that the size of the state space tremendously grows and results in unsolvable problems. To counter this problem, we run a shallow pruning algorithm to eliminate a set of unreachable states which is somehow effective, we still require a more advanced mechanism to improve scalability of our method. In this chapter, we then proposed two enhancements.

The first improvement considers how to order different product operations. Even though the product operation is commutative, and the order does not influence the final result, it has a great effect on the number computations and consequently the total time. We proposed an algorithmic approach to create aggregate-states from the states that have similar labels and are connected to each other to significantly reduce the size of the state space. Then we apply the product operation between the aggregate state space and the original DFAs to estimate the size of the final LCMDP. This helps us to predict the best order in which the product operation has to be performed. This improvement is loss-less and has showed speed-ups of more than 50 % in our experiments.

The second improvement proposes a pruning approach to eliminate some of the states that have lower possible influence on final result. To do so, we proposed a rating methodology to rank all the states in the product LCMDP by their reachabilities from the initial states. Then we eliminate a pre-determined ratio of them according to the parameters that are given by the user from least reachable state to the highest. This is a parametric approach and can be tuned, and its effects vary significantly based on the problem specifications. Therefore, depending on similarities of different LTL properties, the influences of the pruning algorithm change. In short, if two LTL properties comply with each other e.g. $\Diamond A$ and $\Diamond(A \wedge \bigcirc\Diamond B)$, the pruning algorithm is very effective and loss-less. On the contrary, if the two LTL formulas contradict e.g. $\Diamond A$ and $\neg A \,\mathtt{U}\, \mathcal{G}$, the algorithm sacrifices some viable solutions to speed-up the process.

This work can be extended in multiple ways. One may look at the type of a problem and recognize what pruning category it falls in. Then it can predict a possible loss in the solution, and apply a certain level of pruning to the planner only. Similarly, a planner can look some steps ahead and considers the effect of the pruning on all the LTL properties and their propagated influences on the future product operations in order to set the proper parameters. Another proposed enhancement is to combine this solver with the HCMDP approach which was presented in Chapter 4 in order to propose a hierarchical model for the resulting LCMDP. HCMDP has already proven that it is a loss-less approach, and results in great speed-ups.

Chapter 7

Conclusions

In this dissertation we developed novel planners to drive robots autonomously in environments with uncertainties while accomplishing complex missions. Technological advances have improved the capabilities of robots by offering a wide range of sensors and actuators. As a result, they should be programmed to perform multiple complex tasks at once. One of the most common challenges for robots is to work in environments with uncertainties. The non-determinism which is caused by the uncertainty, makes the robot motions unpredictable.

One of the most popular algorithms to find optimal solutions in non-deterministic environments is Markov Decision Processes (MDPs). MDPs are widely used in many different robotics applications. However, all of MDP-based approaches are limited to optimize over a single objective function. But a real robot has to consider several factors to drive optimally in its work space. Considering a real scenario of driving a mobile robot to a target point, the robot is required to take several factors into account such as energy consumption, time of execution, etc. A planner is then required to balance all the parameters, and select the most appropriate ones.

Therefore, Constrained Markov Decision Processes (CMDPs) are proposed to solve multi-objective optimization problems in non-deterministic environments. They are capable of taking multiple factors or cost functions into account while finding the optimal plan. Despite all the capabilities of CMDPs, they are not very popular in robotic planning, and researchers prefer MDPs over CMDPs. The main reason lies in CMDP's computation complexity. CMDPs are mostly solved with linear programming which is not easily scalable. Popular solvers require huge computational resources. But MDP-based solvers usually use dynamic programming which is based on breaking down a large problem into a set of sub-problems and analyzing each of them separately. It reduces the amount of required resources by running multiple iterations. An alternative to use CMDPs is to linearly combine multiple cost functions in a single one and plug that into an MDP. However, the interpretation of such metrics are very hard to interpret and difficult to generalize.

Throughout this dissertation, we used CMDPs as our primary planning algorithm, and showed some of their capabilities. We started in Chapter 3 by defining the theoretical concept of CMDPs, and very briefly showed how to solve its optimization problem.

Next, we showed an application of CMDPs in solving risk-aware planning problems in Chapter 4. Risk-aware planning is only an example of optimal planning with multiple constraints, and the proposed approach can be applied to any similar application. In the selected application, the

robot runs in a non-deterministic environment, and is tasked to go to a goal location by minimizing the total risk along the path while having an upper bound on the path length. Such problems can be easily expressed as CMDP instances. Nevertheless, CMDP solvers very commonly use linear programming approaches that may be memory demanding, due to the extremely large number of optimization variables.

To overcome such limitation, we proposed a hierarchical model for CMDP that partitions the state space into fixed size clusters. Our proposed solver calculates the optimal policy for each cluster separately in order to break down the large problem into smaller sub-problems. Then it stitches the sub-solutions together and creates the complete plan. Comparing the non-hierarchical with the hierarchical approach, the hierarchical method showed great improvements in timing, and in some cases the speed-up reached a factor of 150, while the average risk and path length are closely comparable. Moreover, we showed that the hierarchical solver brings scalability to the main CMDP problem. In this case, we did not require high amount of memory or processing powers, and the problem would be solved on a standard laptop.

Although the hierarchical solver adds numerous advantages to the original CMDP, it does not necessarily preserve existence of a solution. As mentioned in the chapter, there may be some cases where a non-hierarchical CMDP solver is able to find a feasible plan, but the hierarchical solver fails to do so, because of the fixed rectangular shape of the clusters. This limitation led us to introduce a new type of partitioning mechanism (HCMDP) which outperformed the hierarchical solver by addressing this disadvantage.

The theory of HCMDP proposes to generate clusters by adaptively selecting partitions with two conditions: connectivity and similarity. In this approach, we do not require all the partitions to have the same size and shape. Instead, we focus on clusters to be properly connected. Satisfying such conditions guarantees that if a non-hierarchical CMDP finds a viable plan, its HCMDP will find one, too. However, it does not provide any guarantees on optimality of such solutions. We suggested a Monte Carlo approach to estimate the cost and transition function on high level HCMDP. Finally, we ran extensive tests on all the proposed solvers to show the advantages and disadvantages of each method. We compared the non-hierarchical, hierarchical CMDP (with fixed size partitioning) and HCMDP approaches in Matlab experiments in which all the simulation parameters are fully known. We also compared them on Gazebo simulator where the uncertainties are controllable but not fully known. Lastly, we showed the comparison when a real mobile robot (Pioneer 3AT) operates autonomously for more than 5.5 km by executing our plans.

We tested HCMDP in multiple experimental ways to compare it with different approaches. As a future work one might analyze the difference between HCMDP results and the optimal solution from the non-hierarchical approach by some mathematical means and provide an upper bound or limit on the amount of loss. Although we guarantee existence of solutions, we cannot bound the difference from the optimal solution. Another useful addition is to identify a principled way to select a proper sampling rate in Monte Carlo approach to achieve the best results.

In Chapter 5, we presented more complex scenarios by introducing tasks. A task is described by temporal sequence of conditions to happen during the execution of a plan. An example task for an autonomous forklift is to drive toward a pick up station, load an object, move to a delivery location, then drop off the item. Such tasks can be used in several robotics applications where the order of executing actions is important.

Therefore, we used Linear Temporal Logic (LTL) properties to formalize the definition of tasks. Then we proposed a solver which is capable of finding a plan with the following conditions:

- There are n tasks to be performed during robot runs.
- Every task has to be satisfied with a target probability (in expectations).
- There is a primary cost function which has to be minimized.
- Multiple additional cost functions, m, should be bounded by m predefined values (in expectations).

To solve such a problem, we took multiple steps. In the first step, we converted every LTL property into an extended total Deterministic Finite Automaton (DFA). We also converted the original state space into a Labeled CMDP (LCMDP). In step two, we presented a product operation which can be repeatedly applied between the original LCMDP and every extended total DFA where the output of every product is a new LCMDP which can be used in the next product operation. Step three consists of solving the final LCMDP using a linear programming approach by taking advantage of occupation measures. In this approach, we can replace CMDPs with MDPs as alternative solvers. However, we will not be able to provide support multiple cost function unless we combine multiple cost functions into a single one as explained above. Another weakness of using MDPs is that it will not be easy to satisfy all the sc-LTL properties with their target probabilities. However, CMDP makes both of them easier.

To analyze the performance of our algorithm, we ran extensive experiments under different circumstances. We started by running numerous Matlab experiments. We applied the solver thousands of times to illustrate the satisfaction rates for tasks and upper bounds for additional costs. Then we set up a factory environment on the Gazebo simulator, and showed the performance of a forklift which runs our solver in multiple scenarios. We also proposed an open source implementation and showed its architecture. In the final step, we deployed our planner on a robot (Husky) and ran it outdoor for about 8 km.

Our planner was able to include several tasks in a linear program, but it suffered from scalability problems. Since the product operation is repeated multiple times, and each iteration increases the size of state space significantly, it is possible that the state space expands and the problem becomes unsolvable. Thus, we improved the proposed algorithm in Chapter 6 by changing the order of product operations, and pruning a set of unreachable states. In the experiments section, we showed that the first improvement is loss-less and can reduce the calculation time to about half. But the second improvement may cause some loss in the final result while dropping the calculation time to less than one third.

This work can be extended in multiple ways. One addition can be to apply HCMDP on the product LCMDP and solve the final LCMDP using a hierarchical approach. Another improvements in this approach can be to choose the pruning algorithm intelligently by measuring the amount of possible loss before applying the pruning. This can be done by comparing multiple sc-LTL properties with each other before selecting the pruning percentages. Additionally, one can break down complicated sc-LTL properties into multiple shorter and easier ones when possible. This

can be done by exploring the labels and the way they are distributed. Therefore, some improbable combinations can be excluded from the beginning which saves time and computational resources.

Bibliography

[1] ISO 14121-1. Safety of machinery - risk assessment - part 1: Principles, 2007.

[2] ISO/TS 15066. Robots and robotic devices - collaborative robots, 2011.

[3] N. Abe, S. Sako, and S. Tsuji. High-level robot task specification. In *Artificial Intelligence for Industrial Applications, 1988. IEEE AI '88., Proceedings of the International Workshop on*, pages 341–346, 1988.

[4] E. Altman. Denumerable constrained markov decision processes and finite approximations. *Mathematics of operations research*, 19(1):169–191, 1994.

[5] E. Altman. *Constrained Markov Decision Processes*. Stochastic modeling. Chapman & Hall/CRC, 1999.

[6] N. M. Amato, K. A. Dill, and G. Song. Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. *Journal of Computational Biology*, 10(3-4):239–255, 2003.

[7] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 1, pages 113–120. IEEE, 1996.

[8] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebric decision diagrams and their applications. *Formal methods in system design*, 10(2-3):171–206, 1997.

[9] A. Bai, F. Wu, and X. Chen. Online planning for large mdps with maxq decomposition. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1215–1216, 2012.

[10] C. Baier and J.P Katoen. *Principles of model checking*. MIT Press, 2008.

[11] J. Barry, L. P. Kaelbling, and T. Lozano-Pérez. Hierarchical solution of large markov decision processes. Technical report, MIT, 2010.

[12] J. L. Barry, L. P. Kaelbling, and T. T. Lozano-Pérez. DetH*: Approximate hierarchical solution of large markov decision processes. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[13] A. Bauer and Y. Falcone. Decentralised ltl monitoring. In *International Symposium on Formal Methods*, pages 85–100. Springer, 2012.

[14] A. Bauer, M. Leucker, and C. Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In *International Workshop on Runtime Verification*, pages 126–138. Springer, 2007.

[15] A. Bauer, M. Leucker, and C. Schallhart. Comparing ltl semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010.

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

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

[18] D. P. Bertsekas. *Dynamic Programming & Optimal Control*, volume 1. Athena Scientific, 2005.

[19] D. P. Bertsekas. *Dynamic programming and optimal control*, volume 2. Athena Scientific Belmont, MA, 2005.

[20] A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 142–156. Springer, 2004.

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

[22] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi. Motion planning with complex goals. *IEEE Robotics & Automation Magazine*, 18(3):55–64, 2011.

[23] R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 521–528. IEEE, 2000.

[24] M. Boussard and J. Miura. Objects search: a constrained mdp approach. In *Workshop Active Percept. Object Search Real World, San Francisco, CA, USA*, 2011.

[25] C. Boutilier, T. Dean, and S. Hanks. Planning under uncertainty: Structural assumptions and computational leverage. In *Proceedings of the Second European Workshop on Planning*, pages 157–171. Citeseer, 1995.

[26] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1):49–107, 2000.

[27] C. Boutilier, R. Dearden, M. Goldszmidt, et al. Exploiting structure in policy construction. In *IJCAI*, volume 14, pages 1104–1113, 1995.

[28] J. Bouvrie and M. Maggioni. Efficient solution of markov decision problems with multiscale representations. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 474–481. IEEE, 2012.

[29] S. J. Bradtke and M. O. Duff. Reinforcement learning methods for continuous-time markov decision problems. In *Advances in neural information processing systems*, pages 393–400, 1995.

[30] M. S. Branicky, M. M. Curtiss, J. Levine, and S. Morgan. Sampling-based planning, control and verification of hybrid systems. *IEE Proceedings Control Theory and Applications*, 153(5):575, 2006.

[31] R. A. Brooks. Solving the find-path problem by good representation of free space. *IEEE Transactions on Systems, Man, and Cybernetics*, (2):190–197, 1983.

[32] R. E Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on computers*, 100(8):677–691, 1986.

[33] S. Carpin, M. Pavone, and B. M. Sadler. Rapid multirobot deployment with time constraints. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1147–1154. IEEE, 2014.

[34] S. Carpin, M. Pavone, and B.M. Sadler. Rapid multirobot deployment with time constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1147–1154, 2014.

[35] M. Chamie and B. Acikmese. Finite-horizon markov decision processes with state constraints. *arXiv preprint arXiv:1507.01585*, 2015.

[36] Y. Chen, J. Tumova, and C. Belta. Ltl robot motion control based on automata learning of environmental dynamics. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 5177–5182. IEEE, 2012.

[37] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, June 2005.

[38] Y. Chow, M. Pavone, B. Sadler, and S. Carpin. Trading safety versus performance: Rapid deployment of robotic swarms with robust performance constraints. *ASME Journal of Dynamical Systems, Measurements and Control*, 137(3):031005, 2015.

[39] Y. Chow, M. Pavone, B. M. Sadler, and S. Carpin. Trading safety versus performance: Rapid deployment of robotic swarms with robust performance constraints. *Journal of Dynamic Systems, Measurement, and Control*, 137(3):031005, 2015.

[40] F. Ciesinski and M. Größer. On probabilistic computation tree logic. In *Validation of Stochastic Systems*, pages 147–188. Springer, 2004.

[41] C. Cini and A. Francalanza. An ltl proof system for runtime verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 581–595. Springer, 2015.

[42] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Progress on the state explosion problem in model checking. In *Informatics*, pages 176–194. Springer, 2001.

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

[44] C. A. Coello and G. B. Lamont. *Applications of multi-objective evolutionary algorithms*, volume 1. World Scientific, 2004.

[45] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[46] P. Dai and J. Goldsmith. Topological value iteration algorithm for markov decision processes. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1860–1865, 2007.

[47] P. Dai, Mausam, D.S. Weld, and J. Goldsmith. Topological value iteration algorithms. *Journal of Artificial Intelligence Research*, 42(1):181–209, 2011.

[48] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx. Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. *The International Journal of Robotics Research*, 26(5):433–455, 2007.

[49] K. V. Delgado, S. Sanner, and L. N. De Barros. Efficient solutions to factored mdps with imprecise transition probabilities. *Artificial Intelligence*, 175(9):1498–1527, 2011.

[50] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, pages 227–303, 2000.

[51] X. Ding, B. Englot, A. Pinto, A. Speranzon, and A. Surana. Hierarchical multi-objective planning: From mission specifications to contingency management. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3735–3742, 2014.

[52] X. Ding, A. Pinto, and A. Surana. Strategic planning under uncertainties via constrained markov decision processes. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4568–4575, 2013.

[53] X. Ding, S. L. Smith, C. Belta, and D. Rus. LTL control in uncertain environments with probabilistic satisfaction guarantees. In *18th IFAC World Congress*, volume 44, pages 3515–3520, 2011.

[54] X. Ding, S. L. Smith, C. Belta, and D. Rus. Optimal control of Markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control*, 59(5):1244–1257, 2014.

[55] D. V. Djonin and V. Krishnamurthy. Q-learning algorithms for constrained markov decision processes with randomized monotone policies: Application to mimo transmission control. *IEEE Transactions on Signal Processing*, 55(5):2170–2181, 2007.

[56] D. A. Dolgov and E. H. Durfee. Constructing optimal policies for agents with constrained architectures. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 974–975. ACM, 2003.

[57] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *International Conference on Computer Aided Verification*, pages 136–145. Springer, 1990.

[58] K. Etessami, M. Kwiatkowska, M. Y Vardi, and M. Yannakakis. Multi-objective model checking of markov decision processes. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 50–65. Springer Berlin Heidelberg, 2007.

[59] K. Etessami, M. Kwiatkowska, M. Y. Vardi, and M. Yannakakis. Multi-Objective Model Checking of Markov Decision Processes. *Logical Methods in Computer Science*, Volume 4, Issue 4, November 2008.

[60] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for mobile robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2020–2025. IEEE, 2005.

[61] G. E. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[62] Z. Feng and E. A. Hansen. Symbolic heuristic search for factored markov decision processes. In *AAAI/IAAI*, pages 455–460, 2002.

[63] S. Feyzabadi and S. Carpin. Risk aware path planning using hierarchical constrained markov decision processes. In *Proceedings of the IEEE International Conference on Automation Science and Engineering*, pages 297–303, 2014.

[64] S. Feyzabadi and S. Carpin. Risk-aware path planning using hierarchical constrained markov decision processes. In *Proceedings of the IEEE International Conference on Automation Science and Engineering*, pages 297–303, 2014.

[65] S. Feyzabadi and S. Carpin. HCMDP: a hierarchical solution to constrained markov decision processes. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3971–3978, 2015.

[66] S. Feyzabadi and S. Carpin. Planning using hierarchical constrained markov decision processes. *Autonomous Robots*, 2017.

[67] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative multi-objective verification for probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 112–127. Springer, 2011.

[68] V. Forejt, M. Kwiatkowska, and D. Parker. Pareto curves for probabilistic model checking. In *International Symposium on Automated Technology for Verification and Analysis*, pages 317–332. Springer, 2012.

[69] E. Frazzoli, M. A Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129, 2002.

[70] N. Galichet, M. Sebag, and O. Teytaud. Exploration vs exploitation vs safety: Risk-aware multi-armed bandits. In *ACML*, pages 245–260, 2013.

[71] M. Geilen. On the construction of monitors for temporal logic properties. *Electronic Notes in Theoretical Computer Science*, 55(2):181–199, 2001.

[72] R. Geraerts and M. H. Overmars. A comparative study of probabilistic roadmap planners. In *Algorithmic Foundations of Robotics V*, pages 43–57. Springer, 2004.

[73] R. Gerth, D. Peled, M. Y Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing and Verification XV*, pages 3–18. Springer, 1996.

[74] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1):163–223, 2003.

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

[76] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping iwht Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):36–46, 2007.

[77] E. A. Hansen and S. Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1):35–62, 2001.

[78] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[79] W. B. Haskell and R. Jain. Dominance-constrained markov decision processes. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 5991–5996. IEEE, 2012.

[80] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 220–229. Morgan Kaufmann Publishers Inc., 1998.

[81] Human Science Group Health and Safety Laboratory. Collision and injury criteria when working with collaborative robots, 2012.

[82] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 441–444. Springer, 2006.

[83] J. Hoey, R. St-Aubin, A. J. Hu, and C. C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 279–288, 1999.

[84] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson, 2006.

[85] R. A Howard. Dynamic programming and markov processes. 1960.

[86] D. Hsu, L. E. Kavraki, J. C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*, pages 141–154, 1998.

[87] J. Huang, C. Erdogan, Y. Zhang, B. Moore, Q. Luo, A. Sundaresan, and G. Rosu. Rosrv: Runtime verification for robots. In *International Conference on Runtime Verification*, pages 247–254. Springer, 2014.

[88] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge University Press, 2004.

[89] Y. K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Computing Surveys (CSUR)*, 24(3):219–291, 1992.

[90] M. Kalisiak and M. Van de Panne. A grasp-based motion planning algorithm for character animation. *The Journal of Visualization and Computer Animation*, 12(3):117–129, 2001.

[91] L. Kallenberg. Markov decision processes. *URL http://www. math. leidenuniv. nl/˜ kallenberg/Lecture-notes-MDP. pdf*, 2009.

[92] S Karaman and E Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*, 104.

[93] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 2222–2229. IEEE, 2009.

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

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

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

[97] D. Koditschek. Global robot task specification and control via objective functions on the euclidean group: End-point tasks. Yale University, April 1987.

[98] A. Kolobov. Planning with Markov decision processes: An AI perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–210, 2012.

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

[100] J.J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings (ICRA). IEEE International Conference on*, volume 2, pages 995–1001 vol.2. IEEE, 2000.

[101] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Computer aided verification*, pages 585–591. Springer, 2011.

[102] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Compositional probabilistic verification through multi-objective model checking. *Information and Computation*, 232:38–65, 2013.

[103] B. Lacerda, D. Parker, and N. Hawes. Optimal and dynamic planning for markov decision processes with co-safe LTL specifications. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1511–1516, 2014.

[104] M. Lahijanian, S. Almagor, D. Fried, L. E. Kavraki, and M. Y. Vardi. This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction. In *AAAI*, pages 3664–3671, 2015.

[105] M. Lahijanian, S. Andersson, and C. Belta. Control of markov decision processes from pctl specifications. In *Proceedings of the 2011 American Control Conference*, pages 311–316. IEEE, 2011.

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

[107] M. Lahijanian and M. Kwiatkowska. Specification revision for markov decision processes with optimal trade-off. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 7411–7418. IEEE, 2016.

[108] K. Lamine and F. Kabanza. Reasoning about robot actions: A model checking approach. In *Advances in Plan-Based Control of Robotic Agents*, pages 123–139. Springer, 2002.

[109] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

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

[111] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

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

[113] S. R. Lindemann and S. M. LaValle. Current issues in sampling-based motion planning. In *Robotics Research. The Eleventh International Symposium*, pages 36–54. Springer, 2005.

[114] P. Linz. *An introduction to formal languages and automata*. Jones & Bartlett Publishers, 2011.

[115] S. G Loizou and K. J. Kyriakopoulos. Automatic synthesis of multi-agent motion tasks based on ltl specifications. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 1, pages 153–158. IEEE, 2004.

[116] T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE transactions on computers*, 100(2):108–120, 1983.

[117] R. Luna, M. Lahijanian, M. Moll, and L. E. Kavraki. Asymptotically optimal stochastic motion planning with temporal goals. In *Algorithmic Foundations of Robotics XI*, pages 335–352. Springer, 2015.

[118] D. C. MacKenzie, R. C. Arkin, and J. M. Cameron. Multiagent mission specification and execution. In *Robot colonies*, pages 29–52. Springer, 1997.

[119] P. Mellodge and P. Kachroo. *Model abstraction in dynamical systems: Application to mobile robot control*. Springer, 2008.

[120] T. M. Moldovan and P. Abbeel. Risk aversion in markov decision processes via near optimal Chernoff bounds. In *NIPS*, pages 3140–3148, 2012.

[121] C. Nissoux, T. Siméon, and J. P. Laumond. Visibility based probabilistic roadmaps. In *Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on*, volume 3, pages 1316–1321. IEEE, 1999.

[122] A. A. Pereira, J. Binney, G. A. Hollinger, and G. S. Sukhatme. Risk-aware path planning for autonomous underwater vehicles using predictive ocean models. *Journal of Field Robotics*, 30(5):741–762, 2013.

[123] E. Plaku. Path planning with probabilistic roadmaps and co-safe linear temporal logic. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2269–2275. IEEE, 2012.

[124] E. Plaku. Planning in discrete and continuous spaces: From ltl tasks to robot motions. In *Conference Towards Autonomous Robotic Systems*, pages 331–342. Springer, 2012.

[125] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics*, 21(4):597–608, 2005.

[126] E. Plaku and S. Karaman. Motion planning with temporal-logic specifications: Progress and challenges. *AI Communications*, 29(1):151–162, 2015.

[127] E. Plaku, L. E. Kavraki, and M. Y. Vardi. Falsification of ltl safety properties in hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 368–382. Springer, 2009.

[128] M. L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[129] A. Rodríguez, L. Basañez, and E. Celaya. Robot task specification and execution through relational positioning. *IFAC Proceedings Volumes*, 40(3):183–188, 2007.

[130] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages: Volume 3 Beyond Words*. Springer Science & Business Media, 2012.

[131] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

[132] C. Samson, B. Espiau, and M. L. Borgne. *Robot control: the task function approach*. Oxford University Press, 1991.

[133] J.T. Schwartz and M. Sharir. A survey of motion planning and related geometric algorithms. *Artificial Intelligence*, 37(1):157 – 169, 1988.

[134] K. Sen, G. Roşu, and G. Agha. Generating optimal linear temporal logic monitors by coinduction. In *Annual Asian Computing Science Conference*, pages 260–275. Springer, 2003.

[135] M. Sipser. *Introduction to the theory of computation*. Course technology CENGAGE learning, 2006.

[136] M. Spaan. Partially observable markov decision processes. In *Reinforcement Learning*, pages 387–414. Springer, 2012.

[137] I. Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 443–453. IEEE, 2000.

[138] S. Struck, M. Lipaczewski, F. Ortmeier, and M. Güdemann. Multi-objective optimization of formal specifications. In *High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on*, pages 201–208. IEEE, 2012.

[139] C. Sun, E. Stevens-Navarro, V. Shah-Mansouri, and V. W. Wong. A constrained mdp-based vertical handoff decision algorithm for 4g heterogeneous wireless networks. *Wireless Networks*, 17(4):1063–1081, 2011.

[140] C. Sun, E. Stevens-Navarro, and V. W. Wong. A constrained mdp-based vertical handoff decision algorithm for 4g wireless networks. In *2008 IEEE International Conference on Communications*, pages 2169–2174. IEEE, 2008.

[141] P. Thati and G. Roşu. Monitoring algorithms for metric temporal logic specifications. *Electronic Notes in Theoretical Computer Science*, 113:145–162, 2005.

[142] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2006.

[143] P. Ulam, Y. Endo, A. Wagner, and R. Arkin. Integrated mission specification and task allocation for robot teams-design and implementation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4428–4435. IEEE, 2007.

[144] A. Ulusoy, T. Wongpiromsarn, and C. Belta. Incremental controller synthesis in probabilistic environments with temporal logic constraints. *The International Journal of Robotics Research*, 33(8):1130–1144, 2014.

[145] N. A. Vien and M. Toussaint. Hierarchical monte-carlo planning. In *AAAI*, pages 3613–3619, 2015.

[146] E. M. Wolff, U. Topcu, and R. M. Murray. Robust control of uncertain markov decision processes with temporal logic specifications. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 3372–3379. IEEE, 2012.

[147] T. Wongpiromsarn, A. Ulusoy, C. Belta, E. Frazzoli, and D. Rus. Incremental synthesis of control policies for heterogeneous multi-agent systems with linear temporal logic specifications. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 5011–5018, 2013.