UNIVERSITY OF CALIFORNIA, MERCED

Probabilistic Constrained Decision Making for Robots Exploring, Mapping, and Navigating Indoor Environments.

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

in

Electrical Engineering and Computer Science

by

José Luis Susa Rincón

Committee in charge:
Professor Stefano Carpin, Chair
Professor Marcelo Kallmann
Professor David C. Noelle

2020

Copyright Notice

The Dissertation of José Luis Susa Rincón is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Marcelo Kallmann

David C. Noelle

Stefano Carpin, Chair

University of California, Merced

2019

”Tú debes ser como una flecha, y nosotros con tu mamá seremos el arco que con toda nuestras fuerzas te ayudaremos a volar lo más alto y lejos que podamos.”

”You have to be an arrow: your mother and I will be the bow that, with all our strength, will make you fly as far and high as possible.”

José Félix Susa Lizarazo

Dedication

Today is a good day to look back, it is different to any other days in the past, I remember the long journey to get here; It took 35 years, and I almost didn't make it because the path had many obstacles. Reflecting on the winding path - I remember being at a multi-robot conference, presenting one of my papers that I never imagined I would write. I remember an internship in Japan where I worked on autonomous cars. I left behind my country and my family to be here - and created a new one; three awesome ladies (including Boo). I never imagined in my wildest dreams that I was going to see all I have seen, do all I have done, and learn what I have learned. However, in my mind, the luxury and luck of surviving, has always given me the best motivation to do more and more, to contribute to this world and enjoy the opportunity I was given. This difficult journey was also tough not just for me but for my parents and family, I believe they never imagined that one day I was going to be a crazy scientist, building and making intelligent machines or, in technical terms, being a robotics Engineer. Today I get to show my family and friends that, even if they are not physically with me, they all share a vital part in this story...a story that is just beginning. Nothing I have done is attributed to me alone, but is in fact a triumph of the many people who have taken this journey with me - and stuck with me on the good and bad days. This dissertation is ultimately a triumph of my parents. With them standing by me, I have made the right choices in life that allow me to write these very words.

This is for my brother Juanca, my mom, my dad, my grandma Ligia, my little baby gordis, my wife, Dashis, Boo and the many others that walked with me.

Contents

List of Symbols ix

List of Figures ix

List of Tables xiii

List of Algorithms xiv

Acknowledgment xv

Curriculum Vita xviii

Abstract xx

1 Introduction 1

2 Literature Review 5

2.1 Planning under Uncertainty 5

2.1.1 Markov Decision Processes - MDP 7

2.1.2 Constrained Markov Decision Processes - CMDP 8

2.2 Context for the Rapid Deployment Problem 10

2.3 Mapping and Exploration 12

2.3.1 Mapping 12

2.3.2 Map Merging 14

2.3.3 Exploration and SLAM 15

2.4 Replicability and R-Articles 16

3 Theoretical Background 19

3.1 Introduction 19

3.2 Markov Decision Process - MDP 19

3.2.1 MDP Formulation 21

3.3 Constrained Markov Decision Process - CMDP 23

3.3.1 CMDP Formulation 23

3.3.2 Solving CMDPs 25

4 Rapid Deployment of Mobile Robots under Constraints 27
4.1 Introduction 27
4.2 Observing Multiple Targets with One Robot 29
4.2.1 Environment model 29
4.2.2 A model based on CMDPs 30
4.3 Solving the rapid deployment problem 34
4.4 Group Assignment Problem 35
4.5 Simulations 38
4.6 Conclusions 40

5 Motion Planning Under Temporal Constraints with Stochastic Motion Primitives: Theory and Practice 42
5.1 Introduction 42
5.2 Preliminary Background 44
5.2.1 Uncertainty Quantification via Probabilistically-Valid Stochastic Models 44
5.2.2 Stochastic Motion Primitives Derivation 46
5.2.3 Extracting the Transition Probabilities 48
5.2.4 Deterministic Model Fitting 52
5.2.5 Stochastic Model Extension 53
5.3 Experimental Setup 55
5.3.1 Environment 56
5.3.2 Infrastructure 58
5.4 CMDP definition 59
5.5 Experimental Validation 63
5.6 Conclusions 65

6 Time-Constrained Exploration Using Toposemantic Spatial Models 68
6.1 Introduction 68
6.2 Replicability 71
6.3 Oriented Topological Semantic Maps 72
6.3.1 OTSM Formulation 75
6.3.2 Incremental Map Construction, Navigation, and Exploration . 78
6.4 Exploration Algorithms 82
6.4.1 Random Strategy Exploration 82
6.4.2 Topological Frontier 82
6.4.3 Topological Frontier with Normalized Distances 83
6.4.4 Semantic: Explore Corridors First 83
6.4.5 Semantic: Complete Corridors First 83
6.5 Experimental Validation 84
6.5.1 Setup 84
6.5.2 Maneuvers 85
6.5.3 Calculating the Failure and Temporal deadline 85
6.5.4 Software Architecture 87

6.6 Results 88
6.6.1 Random Strategy Exploration 90
6.6.2 Topological Frontier Exploration 91
6.6.3 Topological Frontier with Normalized Distances Exploration . 91
6.6.4 Semantic Strategy Exploration 91
6.7 Conclusions 95

7 OTSM Implementation 96
7.1 Introduction 96
7.2 Systems Implementations 97
7.2.1 Intersection Detection System (IDS) using 2D LIDAR 97
7.2.2 Labeling System (LS) using 2D images 103
7.2.3 Loop closure problem 103
7.3 Experimental Setup 105
7.4 Results 107
7.5 Conclusions 108

8 Map Merging of Oriented Topological Semantic Maps 110
8.1 Introduction 110
8.2 Inverse Warrington's Object Recognition Model (IWORM) 112
8.3 IWORM Inspired Map Merging of OTSMs 112
8.3.1 Sources of error 117
8.3.2 Understanding Errors 119
8.4 Simulations 120
8.4.1 Setup 120
8.5 Results 121
8.6 Conclusions 124

9 Conclusions 126
9.1 Summary of Contributions 126
9.2 Future Work 127

Bibliography 128

List of Figures

2.1 Rapidly Exploring Random Trees (RRT) (Taken from: http://mrs.felk.cvut.cz/research/motion-planning. Vojtěch Vonásek). 6

2.2 Motion uncertainty when moving in a 2D map. 8

2.3 Left: Sample terrain map with red areas showing higher risks. Right: Two solutions using a Hierarchical Solution with CMDP for different risk constraints. Taken from [58] 9

2.4 Rapid Deployment Problem with a group of robots trying to reach different targets with multiple observation points. 9

2.5 Map described as a graph with multiple vertices, where the pink triangle is taken as a deployment vertex and the ones with green crosses as goal vertices. Taken from [34]. 11

2.6 (a): Example of a gridMap. Taken from [78]. (b): Example of a Topological map. Taken from [60]. (c): Example of a Semantic Map. Taken from [140] 18

3.1 Example of a graph taken from an environment where the deployment point is named as v_0. For any state there is a chance to fail and end in a sink state $\mathcal{S}$ while going to a target state T. Image taken from [41] 20

4.1 A possible setup for rapid deployment. Ten different targets are outlined by solid thick circles (visibility sets are displayed by thin circles of the same color.) 29

4.2 When a successful observation from state x_i is made, the state goes through an augmented state x_i' and, from there, it executes its only action. Then with probability 1, it goes back to the state x_i but with the corresponding modified bit. 31

4.3 Success probability for different temporal deadlines for a different number of robots in the team for the case where 10 targets are considered. 38

4.4 Average time to solve a single linear program for the SRMTO problem as a function of the number of targets (average taken over all linear programs to be solved for a specific number of targets). 40

4.5 Average cumulative time to solve all linear programs to solve the SRMTO problem as a function of the number of targets (average taken over all temporal deadlines considered for a given number of targets). 41

5.1 Example of data fitting: a) Grey lines correspond to the empirical data. Dashed lines correspond to sample trajectories. Blue line correspond to the mean of the sample trajectories. Red lines correspond to the cone of data. b) Computing the cone of data with variability ellipses ϵ_t centered at the sample mean. Image taken from [85] 44

5.2 The environment with the Vicon system composed for motion capture cameras around the room. 46

5.3 The Duckiebot robot used in this study. 47

5.4 Experimental distribution of the trajectory endpoints (blue for fast maneuvers, and purple for slow maneuvers). Fast maneuvers (plotted in blue color) are those that end at larger displacements along the y axis. The final orientation is indicated by the arrow. Position uncertainty ellipses are shown in thick curves. 49

5.5 Left: Position discretization. Right: Orientation Discretization. . . . 50

5.6 Left: 50 trajectories for 3 Maneuvers (“probabilitic template”). Right: Calculating probabilities for forward trajectories that end at *range* 1. 51

5.7 (a) Example of using the probabilistic template with the states on the map. (b) Example of a small extract from a transitions table. This table can contain thousands of transitions, depending of the discretization and the number of actions. 52

5.8 Example of the forward maneuver with and without initial noise for the position. 54

5.9 Realizations of the stochastic model: (a) FS, (b) FF, (c) CWS, (d) CWF, (e) CCWS, (f) CCWF. 55

5.10 One of the real scenarios that we used for our experiments 56

5.11 The two intial environments used for testing. The overlaid grid shows the discretization used for the CMDP. In the following, the environments are referred to as empty 5.11(a), maze 5.11(b). 57

5.12 The two following environments used for testing. The overlaid grid shows the discretization used for the CMDP. In the following the environments are referred to as two ways 5.12(a) and 5.12(b). 58

5.13 Infrastructure for Robot Duck executing policy from CMDP. 59

5.14 Structure of the CMDP for the deployment problem. Top states in X' correspond to the states obtained by discretizing the environment and with transition probabilities numerically determined as described in the following. State $\mathcal{S}$ is entered when there is a collision. State $\mathcal{T}$ is the only state in M and is always entered at the end of a run, either successfully ending in a goal state x_G or being unsuccessful because of a collision. 60

5.15 For a given state x and maneuver a, multiple trajectories are generated using the generative model. Each trajectory is then checked to see if it is collision free or not. Collision free trajectories, like the red ones, are used to determine the transition probabilities to states like x', x'' and like. Trajectories resulting in a collision, like the blue one, are used to numerically estimate $P(x, a, \mathcal{S})$, i.e., the transition probability into the sink state $\mathcal{S}$. 62
5.16 Trajectories showing failures when going to the goal. Zones marked by a red ellipse show where the trajectories collided with the obstacles. (a) Red lines correspond to Maze a, and blue lines to Maze a1. (b) Red lines correspond to Empty map b and blue lines to Empty b1. . . 64
5.17 Trajectories showing failures when going to the goal. Zones marked by a red ellipse show where the trajectories collided with the obstacles. (a) Two different failing trajectories. (b) One failing trajectory. 65

6.1 On the left: three adjacent vertices along the E-W direction. *c*1*a* is on the left of *c*1*b*, and *c*1*c* is on the right of *c*1*b*. On the right: three vertices along the N-S direction with *c*2*a* on the right of *c*2*b* and *c*2*c* on the left of *c*2*b*. 73
6.2 Three corridors are connected. For corridor 1 we have two adjacent vertices along the E-W direction. *c*1*a* is on the left of *c*1*b*. For corridor 2 we have two adjacent vertices along the N-S direction with *c*2*a* on the right of *c*2*b*. The third corridor have two adjacent vertices along the E-W direction. *c*3*a* is on the left of *c*3*b*. *c*3*b* is on the left of *c*2*b*, and *c*2*a* is on the left of *c*1*b* 74
6.3 Two corridors and one room in the middle. For corridor 1 we have two adjacent vertices along the E-W direction. *c*1*a* is on the left of *c*1*b*. For corridor 2 we have two adjacent vertices along the N-S direction with *c*2*a* on the right of *c*2*b*. The Room 1 is on the left of *c*2*b* and on the left of *c*1*b* 75
6.4 Oriented Topological Semantic Map - OTSM. Vertices with a label Rx are rooms, while corridors have labels of the type Cx. Edges with a R label are purple, while edges with a L label are green. 77
6.5 Examples of OTSM taken from CAD designs from the University of California Merced. 79
6.6 Engineering building at the University of California Merced. 84
6.7 Software Architecture. 88
6.8 Three starting points (A, B, C) and one target 89
6.9 The performance of the five exploration strategies. 90
6.10 An analysis of the failures for the five exploration strategies. 90
6.11 The success percentage for all of the exploration strategies. 93
6.12 The time spent for all of the exploration strategies. 94

7.1 Intersections in the real world and corresponding laser scan as seen in 2D LIDAR. (a):Left Intersection. (b): Right Intersection. 98
7.2 Intersections in the real world and corresponding laser scan as seen in 2D LIDAR. (a): Four-way Intersection. (b): T Intersection. 99
7.3 Visualization breakpoints, line segments obtained from a single laser scan using a 180 degrees LIDAR. Taken from [125] 100
7.4 Example line segments identified and middle points for a) Left intersection, b) Right intersection, c) T intersection, d) Four way intersection. 102
7.5 Last three layers of resNet-101 changed. 103
7.6 Sketch of the floor at the University of California Merced, with intersections (blue zones) and rooms (Orange zones). 105
7.7 Pioneer Robot 3AT mounted with a Laptop, a LIDAR, a RGB camera and two IMU um7. 106

8.1 Diagram of the Warrington's Object Recognition Model inspired by the [180] 113
8.2 Diagram of the Inverse Warrington's Object Recognition Model . . . 114
8.3 Error type 1: A semantic labeling error. 117
8.4 Error type 2: An error with the compass. 118
8.5 Error type 3: A missing edge in a sub-map. 118
8.6 Error type 4: A vertex missing in a sub-map. 119
8.7 Left: start locations. Right: overlapping regions. 121
8.8 Comparison of quality from 20 full maps with different errors when maps do not overlap A+D. 122
8.9 Left: OTSM Error Free No Overlapping. Right: OTSM Errors 1+2+3+4 No Overlapping 123
8.10 Comparison of quality from 20 full maps with different errors when maps overlap A+B+C+D+E. 124
8.11 Left: Final OTSM error-free. Right: Final OTSM with errors 1+2+3+4125

List of Tables

5.1 Average performance over 50 runs. 63

5.2 Comparison of transition probabilities P for different amount of data samples taken from the real robot and generating 500 modeled samples. 66

5.3 Comparison of P distribution for different amount of modeled samples taken from the stochastic model against the 500 modeled samples successfully used in table 5.1 66

7.1 Average performance for the intersection detection system over 10 runs for 8 different type of intersections. 107

7.2 Performance of the accumulative training with multiple steps. It shows the number of images taken, the time to train the Neural Network, the final identified label and the light conditions. 109

List of Algorithms

4.1 Rapid deployment 35

6.1 Map Initialization 78
6.2 Adding a new node to the map 80
6.3 Navigation Algorithm 81
6.4 Global Exploration Algorithm 81

7.1 Compute line segments between any two break-points bp_1, bp_2 101

8.1 Merging OTSMs 115
8.2 IWORM scores the resemblance of a pair of vertices $v_i,\ v_j$ 115

Acknowledgments

It has been a journey, with ups and ups. I cannot say there were downs because everything was for a purpose and each step brought me up here. I have to thank my advisor Stefano Carpin who was the one who made all this possible. He gave me the opportunity to have all I have today, and he gave me advice and showed patience throughout this process and I will always remain in his debt. He believed in me, taking me on as a graduate student without knowing me, a complete stranger from a singular country. He saw potential in me when others could not. Thank you for taking a chance on me; I will always remember my humble origins and the mentor who let me be who I am today. I also have to thank my thesis committee Professor David Noelle, and Professor Marcelo Kallmann for supporting my work and this dissertation. Special gratitude for Sree Harsha, Carlos Diaz, Jose Gonzales, and Victor Gonzalez who helped with some of the experiments and implementations of some of the algorithms. I am also grateful for the partial support from by the Army Research Lab through grant W911NF-08-2-0004 (MAST CTA, task CNC-15-4-4). The Miguel Velez Scholarship, and the California Merced Bobcat Fellowships that allowed me to complete my program successfully.

I cannot avoid recognizing the amazing people behind the school of engineering, who were always there to support me. Without people like Tomiko Hale and Tamika Hankston always helping with any administration process, the department would not be functional. Thank you all for your support and work. I really appreciate being a part of this great university, surrounded by excellent people. I also need to thank Martha Garibay, the beautiful person that works for the janitorial staff who sometimes worked with me until very late at night and brought me cookies and bread for motivation.

I want to recognize some members of this great university, who, during all these years, gave advice and support. I need to write some words for Jesus Cisneros and Eric Cannon for assisting and supporting my outreach initiatives and facilitating the financial support. Becky Mirza for being the head of the international office and always standing by and for us as international students. She always helped us get what we needed to have a happy life in this country. The Graduate Programming and Events Coordinator Jen Quiralte and the GRC Coordinator Cassie Gunter for their time and effort to offer activities and support for all graduate students.

I have many people in my heart and my thoughts, people that gave me some of the best years of my life so far. I enjoy their company, the conversations, the tales and mysteries we shared at some point. Ivan, many years have passed and he is still by my side. He's helped me and my family throughout this journey - thank you so much. Of course, I have to thank Eli! Before I even arrived to this country, she was already on my side. We have lived together for years and I hope we can always be in each others lives, stay close.

I cannot feel comfortable saying thank you to the best of all in the middle of this page. She should be at the beginning and at the end, in the center and on the sides, she should be everywhere!. However, the middle may be exactly where she means to be. Anna, you deserve the best place on this page and in the next pages of my life. I could write a whole new dissertation about each of the little and big things you have done for me and with every moment I had to say thank you. We have faced and overcame many challenges. You have been my companion, my support, my friend, my box of laughs and giggles, my travel buddy, my feelings expert, my wife and the mother of the most precious daughter in the world and the cutest Boo ever. I feel lucky for finding you in this world. We still have a story to write so please accept my apology for only writing this time a very short abstract of our life together!

Last but not least, there are people who deserve my most sincere acknowledgments, as they were an important part of this process. Shams and Shuo were very helpful during my first years. They taught me how to succeed and survive graduate school. Dr. Alain, Dr. Angel, Adytia, Jorge, Jocelyn, Andres, Ana (Simões), of course Dr. Drunkachu, and Himanshu. I am thankful for meeting you and thankful for all the adventures we have shared throughout these past years.

It is a very long process to be formed as a researcher, as a thinker, as an engineer, and as a person; In parallel with the academic work done to complete this dissertation, for the past few years, a group of other graduate students from UC Merced and I have been leading a renewed effort to reach the new generation to grow their passion to learn and study in a new way. I would like to recognize the work that helped me to see robotics as a social disruptor, as a tool we can use to break the status quo. I believe we must make the use of robotics in learning a social priority.

For decades, we have grown with an invariable idea of education. We have hoped to obtain all our knowledge from a person standing on front of us, we have sat hours, and hours listening to different teachers and grasping merely the very surface of the idea that they want us to assimilate. We spend years going through school, to college, obtaining multiple degrees, and diplomas, we also accumulate some grades that become only numbers on paper. However, two questions come to mind: how much of all that time and effort is reflected into our society, and how much of that energy is returned to contribute to the great cause of making a better world.

I would like to recognize the great experience I obtained coordinating the Migrant Education in Science and Technology (MESAT) program in 2016 and the Science, Technology and Arts for Youth (STAY) program in 2017. These programs focused on students being able to play being the scientist/engineer within a structured, yet fun and engaging problem, based in the natural sciences. With my team, we have also given many interactive talks, engaging a similar model of problem-based learning in

a very short amount of time. In each talk, we presented secondary students with a problem based in real life and asked them to go with us through a series of problem-solving techniques. In most cases, students forgot they were “learning” because we provided an atmosphere of play and collaboration.

Robotics is a difficult science, complex, and beautiful at the same time. How we use it will impact the world and people in many ways, I have to be thankful for this opportunity to work in this field, help to make it greater and continue working with the next generation.

Curriculum Vita

Education

- **University of California, Merced**. Merced, CA, USA. (2014 – 2020).
  Ph.D. in Electrical Engineering and Computer Sciences.

- **Ecole Nationale D'Ingénieurs De Brest**. Brest, France. (2005 - 2008).
  M.Sc. in Engineering.

- **Escuela Colombiana de Ingeniería Julio Garavito**. Bogotá, Colombia. (2001 - 2005).
  B.S. in Electronics Engineering.

Research Experience

- **Universidad Central**. Bogotá, Colombia. (2013,2014). Engineer's Coordinator.
- **Escuela Colombiana de Ingeniería**. Bogotá, Colombia. (2008, 2009). Researcher.
- **Ecole Nationale supérieure de techniques Avancées**. Paris, France. (Fall 2007). Cognitive Robotics Intern.
- **"Ecole Nationale D'Ingénieurs De Brest (ENIB)**. Brest, France. (2005-2006). Research Assistant.

Publications

- **J. L. Susa Rincon**, K. Karydis, V. Kumar, S. Carpin. Motion Planning Under Temporal Constraints with Stochastic Motion Primitives: Theory and Practice. Springer Autonomous Robots (Under Review).

- **J. L. Susa Rincon**, S. Carpin. Time Constrained Exploration Using TopoSemantic Spatial Models: a replicable approach. *IEEE Robotics and Automation Magazine, (2019).*

- **J. L. Susa Rincon**, S. Carpin. Map Merging of Oriented Topological Semantic Maps. *IEEE International Symposium on Multi-Robot and Multi-Agent Systems, (2019).*

- **J. L. Susa Rincon**, P. Tokekar, V Kumar, S. Carpin. Rapid Deployment of Mobile Robots Under Temporal, Performance, Perception, and Resource Constraints. *IEEE IEEE Robotics and Automation Letters , (2017).*

- **J. L. Susa Rincon**, A. Garcia. Interactive Video Game - Red Dot [computer software]. *Science Faire Groningen, The Netherlands , (2014).*

- **J. L. Susa Rincon**, A. Garcia. Interactive Video Game – Saving Manzanares River [computer software]. *Science Faire Corferias Sergio Arboleda University. Bogotá, Colombia, (2013).*

- **J. L. Susa Rincon**. Educación en robótica para niños. *Gobernación de Arauca. ISBN: 978-958-99553-1-4. Colombia, (2011).*

- **J. L. Susa Rincon**,D. Ramos. Navigation of an autonomous mobile robot using the combined force field concept. *Escuela Colombiana de Ingeniería Magazine No 78. ISSN 0121-5132. Pag 49. April-June , (2010).*

- **J. L. Susa Rincon**, J.F. Susa Lizarazo. Wartegg Automatic Test Qualifier [computer software]. *Colombia, 2008 v1.0, 2016 v5.1.*

Abstract

Probabilistic Constrained Decision Making for Robots Exploring, Mapping, and Navigating Indoor Environments.

Robots are becoming more of a part of our daily lives. They have become an extension of some our human capabilities and there is a need to develop control algorithms that contribute to the successful deployment of these machines to navigate, explore and map indoor human environments. These robots and their actions, despite our effort to make them as predictable as possible, show stochastic behaviors, as well as motion and sensing uncertainties. We leverage the use of Constrained Markov Decision Processes (CMDP) to balance multi-cost problems with constraints, under the premise of having multiple possible sources of uncertainties. This dissertation engages in solving some of these problems in the following chapters.

Initially, we highlight some of the theoretical background about Markov Decision Process (MDP) and its extension the CMDP. From this point we deal with the problem of multiple robots visiting multiple targets, while we fix temporal and failure probability constraints. We present our solution to expand the state space following a binary sequence that represents successful observations of each of the targets. All this is classified as the rapid deployment problem, which we define and solve for a team of robots.

Closing the gap between reality and theory, we implement a stochastic model that recreates the motion primitives from a robot. We proceed to use these modeled primitives to create modeled trajectories and extract transition probabilities from them. These transition probabilities characterize some of the robot's behavior and we use them with our formulation of a CMDP. Then we calculate a navigation policy to traverse some real scenarios.

We create and implement a new spatial model dubbed Oriented Topological Semantic Map (OTSM). This new type of map can be built in run-time, and together with a CMDP, we assign actionable temporal deadlines to the robot executing an exploration task. We open-sourced a ROS framework that can be downloaded and used to reproduce our results and we published the first Reproducible Article or *R-article* in robotics. Consequently, we implement an OTSM by grouping an Orientation System (OS), an Intersection Detection System (IDS), and a Labeling System (LS), using odometry, accelerometers, a LIDAR, and a residual neural network resNet, to extract the orientation, the topology, and the semantics of an indoor environment.

In the last part of this dissertation we propose a new algorithm to merge together pieces of OTSMs when a group of robots have the task to explore an unknown environment. Then they combine their local maps into a global map. Our solution was inspired from research in cognitive science that focused on object recognition. Applying this theory, we create a two-stage method to compare vertices in different OTSMs and measure their resemblance.

Chapter 1

Introduction

Mobile robots can be seen as an extension of our human capacities. We can extend our limitations using one or multiple robots to solve problems that otherwise would be difficult, dangerous, or less efficient to solve by ourselves. For instance, exploration and mapping of a new environment can be a time-consuming and potentially dangerous task that humans do regularly. However we believe this is one area where robots can help considerably. We aim, with this work, to offer novel solutions for problems like exploring, mapping, and navigating indoor environments with mobile robots. We hope that this work contributes to the overall development of these machines that, one day, will help us with our daily tasks.

Often, a robot is seen as a perfect and deterministic agent that behaves in a certain way repetitively under the same conditions and commands; in real life, however, this kind of ideal robot is impossible to develop. Although the control systems aim to reduce the level of errors and noise to make the robot more predictable, at the end, it is very hard to avoid the fact that the robot is a machine with many components (e.g. sensors, actuators, software, etc), each of which, carry inherent errors. Sensors, for example, are components that can only partially estimate the true aspects of specific physical properties. Most of the measurements we use are an approximation of a physical property, and all the sensors present noise when measuring them. The environment also plays a decisive role in defining the type and the amount of errors included in the system. For a physical environment, high and low temperatures, the direction and the amount of light, the type and material of surrounding objects, and the surface where the robot moves, contribute to the overall noise and error. Hence, we will always face a reality where a robot has multiple sources of errors in a variable environment. Under this premise, we develop all our algorithms, assuming that these robots are faulty machines, with noise and errors, and there are multiple types of ***uncertainties*** that need to be considered.

The purpose of this dissertation is to study how to model and solve different problems in three different areas of robotics, i.e navigation, exploration, and mapping. For all of them we will assume that the robots have motion, sensing, and state uncertainties. In navigation, we consider the problem where a team of mobile robots is tasked with collecting information about a set of stationary targets. There is a temporal deadline to complete the task, and the objective is to determine a control policy maximizing the probability of successfully completing the task within the assigned deadline. Furthermore, there are more targets than robots, so load sharing between robots is necessary. We model this problem using the theory of Constrained Markov Decision Processes (CMDP) and split the solution in two steps. First, policies to observe small subsets of targets are computed, and the proposed model and algorithm allow extracting accurate information characterizing the performance of the computed control policies. In the second stage a subset of the computed policies is assigned to the robots for execution with the objective of maximizing a collective team performance function. We show that the objective function is submodular, and a greedy approximation algorithm can be used to solve this nonlinear assignment problem. Simulations demonstrate how these models can be used in practice to appropriately tune the parameters characterizing this problem, and show how the approach favorably scales with the complexity of the problem. A simulation and the implementation of the solution with a real robot is shown in chapters 4 and 5. We present a methodology that uses grey-box methods to identify stochastic motion primitives from sampled trajectories. This approach yields a generative method with guaranteed performance that allows us to predict stochastic behaviors not observed in the original sampled dataset. Developed primitives are then used by a planning algorithm based on Constrained Markov Decision Processes that computes a motion policy aiming to achieve a target pose while being subject to constraints bounding the time to completion as well as failure probability. This work yields a data-driven, closed-loop trajectory control and planning policy with guaranteed, bounded failure probability. Theoretical findings are demonstrated in practice by experimentation on a differential-drive mobile robot subject to motion uncertainties to navigate through different mazes. Experimental results confirm the validity of the proposed method.

For exploration and mapping, spurred by progress in machine learning, there is a tendency to use novel designs in which robots rely more on visual sensors and less on traditional sensors like range finders. Starting from these premises, our objective is twofold. In chapter 6, we first revisit the classic exploration problem introducing temporal constraints in the task and embracing a new type of map called Oriented Topological Semantic Maps (OTSM) that does not include any metric attribute. To assess strengths and weaknesses of the various exploration methods abstracting from the underlying technical implementation, we perform a set of simulations in ROS using its Gazebo simulation environment. The simulation-based approach leads to the second objective of this contribution, namely presenting a set of findings that are fully replicable by a third party. As replicable robotics gains more and more attention, this work represents a first attempt to present a fully replicable investigation based on the widely used Gazebo simulation environment. In chapter 7 we make use of the replicable framework in a real robot, and we test the different systems to extract topological, semantic and orientation information to create OTSMs.

In chapter 8 we propose a solution for the problem of merging together partial spatial models relying on our recently introduced Oriented Topological Semantic Maps (OTSM). This problem arises when a group of robots cooperatively explore an environment, and each one independently builds a partial map that must be combined with the others into a full map. Our methodology is inspired by the Warrington's Object Recognition Model, a cognitive model hypothesizing two post-sensory categorical stages working together for object recognition. Accordingly, we use two stages to compare different maps and match them together based on their mutual resemblance. Our method is complemented by a scoring system to measure the likelihood that two vertices in different OTSMs correspond to the same vertex, despite possible errors in labeling, orientation, or topological structure. Our methodology is validated in a simulation informed by an ongoing real robot implementation, thus allowing us to perform various experiments with carefully controlled error sources.

Finally, we conclude with chapter 9 summarizing our main contributions; We will see by the end of this dissertation that we can leverage a CMDP to model problems where there is stochasticity, uncertainties, and where we need to find a policy that balances temporal and failure probability constraints. In robotics, such application of CMDP model is still mostly unexplored and we hope this dissertation fills in some of the gaps in the areas of navigation, exploration, and mapping. For example, we offered a solution for efficiently distributing a group of robots to visit stationary targets and making stochastic observations that differs from the initial work of rapid deployment introduced by Carpin [35] and Purohit [141]. In exploration, we showed that using semantic information is sufficient to find a target in an unknown environment, and if we have multiple robots exploring, we are able to combine several pieces of OTSM maps they create. We also conclude, that in using stochastic models we can effectively extract and characterize the robot's motion, and that we can use the proposed method to calculate a realistic policy to control a robot. This is, to the best of our knowledge, the first time the use of these stochastic models with a CMDP is proven experimentally.

One of the main contributions of this dissertation, is the creation and use of the OTSMs, for a single and group of robots. We can relate our work to previous works, such as Quatrinni et al. [143], Luperto and Amigoni [109], Blochliger et al. [20], Varadarajan [174], where semantic, and topological information is used to navigate and explore environments. However, for the first time, we created a map that is navigable by humans and robots, that contains enough information to guide a robot, and that we can use in run-time with a CMDP to constrain the exploration and navigation problem with temporal and failure probability constraints. Also for the first time, we publish in the robotics literature a *Reproducible-Article* that stimulates a standard for future publications to share and standardize replicating results in robotics research.

All in all, we show how it is possible to implement our methods and solutions in real robots, we measure the performance in simulation and run the same algorithms with low cost robots. We use the distributed operation system, ROS, to transfer and run the algorithms and prove their applicability and robustness.

Chapter 2

Literature Review

In this chapter we cover some of the main works that are related and inspired the rest of this dissertation. First, in section 2.1 we present an overview of some methods for discrete planning. This will be the core of the following sections where we use the Constrained Markov Decision Processes (CMDP) to Navigate, Explore and Map indoor environments. Then, in section 2.2 we introduce some works that state the Rapid Deployment Problem that we will solve using CMDPs in chapter 4.

In section 2.3 we present the literature related to exploration and mapping, where we compare the techniques and type of information extracted from different environments in order to create a representation that can be used by the robot to calculate policies and an optimal solution for a navigation path using CMDPs.

The last part of this literature review completes the work of mapping an environment with a multi-robot approach. Different techniques are presented and serve as the base for our chapter 8.

2.1 Planning under Uncertainty

A significant body of work has focused on answering one main question: How do we create planning algorithms that can handle uncertainty? We present below some representative works that support our own developments in this area.

Many traditional motion planning approaches, either combinatorial or sampling based, build upon a deterministic framework whereby the environment is known and the outcome of actions is fully predictable (see [97, 40] for thorough introductions to these topics). However, to be applicable in practice, uncertainty must often be considered. When we talk about combinatorial motion planning algorithms [167] we refer to methods that represent the space as combinatorial structures of a given type in order to plan over a graph. This explicit discretization of the environment creates two main problems. First, modeling the free configuration space C_{free} becomes a complex problem to solve for all sorts of environments. Second, when we study problems with many degrees of freedom, dividing the space to achieve a high resolution creates an exponential growth of the space state that makes the computation of a solution harder to process. Sampling based algorithms also try to extract the connectivity of the free space [96, 88], but this time we use random sampling to discover this connectivity, and create the representation of the configuration space C. For this we require a routine that effectively covers C and makes a clear difference between C_{free} and the forbidden space C_{forb}. One of the most used algorithms is the RRT algorithm shown in figure 2.1.

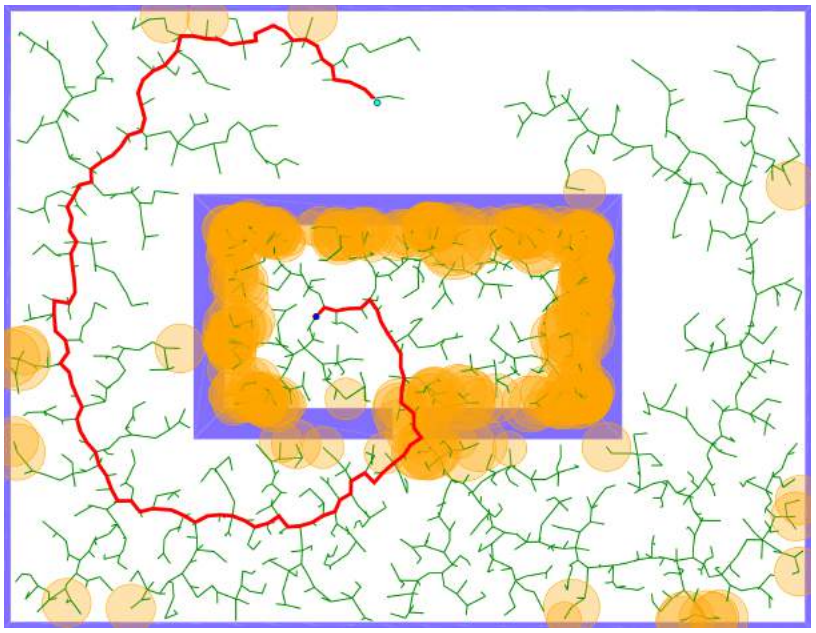


Figure 2.1: Rapidly Exploring Random Trees (RRT) (Taken from: http://mrs.felk.cvut.cz/research/motion-planning. Vojtěch Vonásek).

This leads to two interrelated challenges. The first is how to design algorithms that can handle various sources of uncertainty. The second is to quantify how uncertainty will affect the planning process. Uncertainty may affect various aspects of the system. For example, there may be uncertain knowledge about the environment in which the robot will operate, as well as stochasticity in the execution of its actions and/or in the sensing process used by the robot to track progress during its assigned mission.

Uncertainty quantification has traditionally been a topic in adaptive signal processing [110] and statistical learning theory [173]. Ideas and tools from these areas have helped introduce new models able to capture uncertainty, which are used to derive motion planning strategies. Uncertainty quantification methods can be classified in two main categories: 1) those that employ an underlying first-principles/rule-based model and data from physical processes, and 2) those that employ only data-driven strategies [130, 38]. Examples of the first category include extending deterministic models to stochastic regimes with probabilistic guarantees on the model fidelity [86] and using underlying models as prior information [73] when training a target Gaussian Process model [146]. Data-driven approaches can be broadly categorized into kernel methods (e.g., Gaussian Processes [145], Support

Vector Machines and Principal Component Analysis), artificial neural networks, and function expansion methods [103];[119] offers a thorough presentation on the subject. Examples of the second category include spectral methods [84], kernel methods [76] (such as Volterra models [126]), and more recently deep neural networks [69]. For a general overview the reader is referred to Murphy's work [119].

Compared to purely-data driven approaches, reinforcing existing models with data can perhaps be less general, but it can yield results faster and more accurately. For instance, [46] has shown that using a model to provide prior information when training a target Gaussian Process is more efficient in terms of the required interaction time to achieve a task (e.g. cart-pole balancing). Here we employ the method proposed by [86] to extend deterministic models to stochastic ones with guaranteed degree of fidelity. This approach applies to any user-specified model of a physical process, and determines whether the supplied model can sufficiently[1] reproduce the variability observed experimentally by augmenting model parameters to random variables with appropriate statistics. In a similar way, some recent works also have shown how to transfer control policies for autonomous vehicles into the real world by adding some noise to both state and action to compensate for the mismatch between the model and the real world ([80, 136]). Karydis et al. have shown how stochastic primitives are compatible with the assumptions made in this dissertation and can be generated using a data-driven approach [87].

2.1.1 Markov Decision Processes - MDP

One of the most common approaches to create a planning algorithm capable of handling uncertainty in the execution of actions is provided by Markov Decision Processes (MDPs). MDPs are suitable for stochastic sequential decision-making and have been extensively used in various control and planning scenarios [90, 54, 162]. A good starting point to understand MDPs are the books from Bertsekas [17] and Thrun [163] that have a good overview of what dynamic programming is and how we use the Bellman equation [15] with MDPs.

A simple example is shown in figure 2.2, where a robot is tasked to go to a goal location in a 2D environment and each of its actions embeds a level of uncertainty. When the robot decides to go up toward its target, there is a 10% chance that it falls into the fire/failure_state/sink_state and fail the mission. A MDP takes into account these kind of uncertainties and estimates the best possible policy, i.e. an action for each state that minimizes failure or maximizes future cumulative rewards. In chapter 3 we will extend the explanation of how MDPs work and how we use them in our context of robotics.

One of the tenets of the MDP approach is the assumption that the state is observable and therefore uncertainty in the sensing process does not play a role. Further, MDPs build upon a dynamic programming framework and hinge on the

[1]Sufficiency here is measured in terms of a decision making indicator function; see Section 5.2.1 for more details.

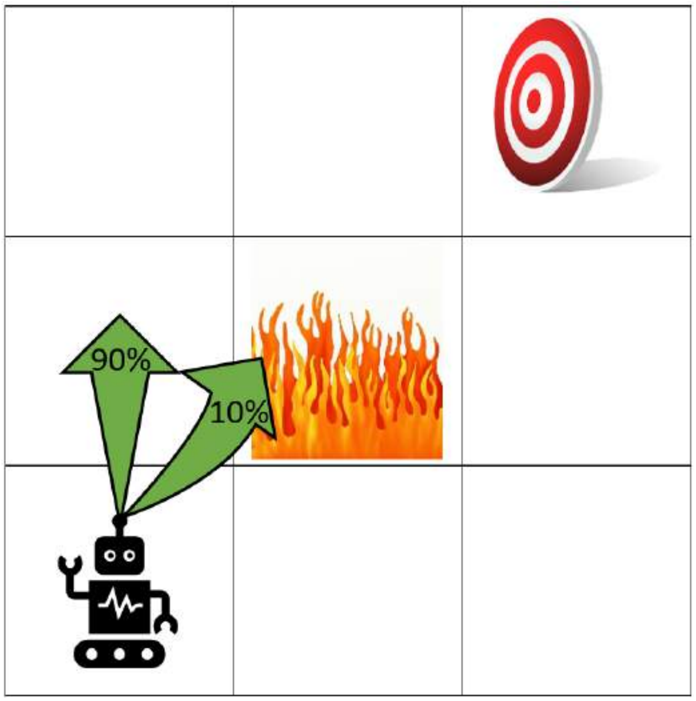


Figure 2.2: Motion uncertainty when moving in a 2D map.

assumption that the parameters characterizing the models are known. When the parameters are not assumed to be known, but are rather estimated through repeated interactions with the environment, reinforcement learning algorithms are used [161, 181, 17]. A good review of MDPs and its applications is the work from Feinberg [57].

2.1.2 Constrained Markov Decision Processes - CMDP

Constrained Markov Decision Processes explicitly considers more than one cost [5]. In a CMDP one cost is minimized while the remaining costs are bounded (all in expectation). Despite their superior modeling power, CMDPs have been scarcely used in robotics, and most applications are relatively recent. Ding et al. [49] introduces a new technique to use a hierarchical approach to solve a multi-objective problem. Similarly Feyzabady et al. use a method to constrain a navigation plan using a hierarchy of partial solutions to solve the problem from a global to a local policy progressively [58]. Figure 2.3 shows a risk map where a hierarchical CMDP evaluates different risk constraints. Planning with constraints has been also studied by [53, 59] who used a CMDP to solve a planning problem. Most recently we presented two different papers where we solve a multi-robot, multi-target problem using CMDPs, that, combined with a optimal solution of the Traveling Salesman Problem (TSP), we can find the best distribution of robots to cover routes that contain multiple observation objectives [160], see figure 2.4. Another example of using CMDP for an iterative topo-semantic map builder algorithm is shown in [159], where a CMDP optimizes a route to follow by a robot that progresively explores and create a graph that describes the environment with topological and semantic information. Both of these last two works will be presented in chapters 4 and 6.

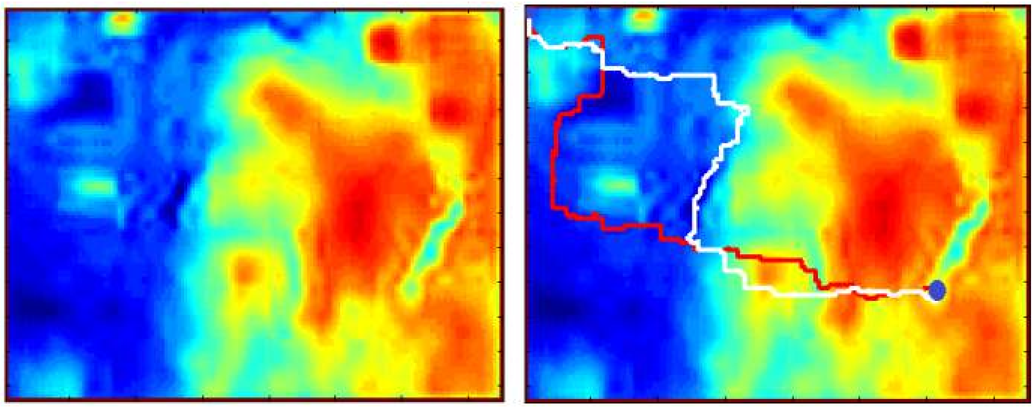


Figure 2.3: Left: Sample terrain map with red areas showing higher risks. Right: Two solutions using a Hierarchical Solution with CMDP for different risk constraints. Taken from [58]

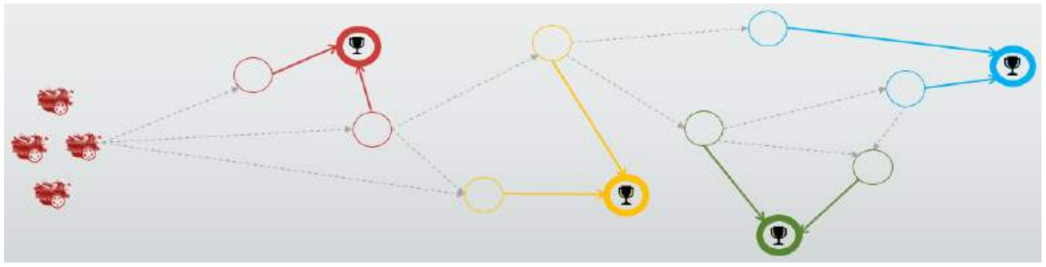


Figure 2.4: Rapid Deployment Problem with a group of robots trying to reach different targets with multiple observation points.

The use of CMDPs has demonstrated the advantages of leading to a policy that balances optimality with multiple constraints. This technique has proven to be a reliable method to deal with different types of uncertainties such as sensing uncertainties, position uncertainties, and even dynamic or kinematic uncertainties of the robot's model.

MDPs and CMDPs work under the hypothesis that the state is observable. A well-known solution for this kind of problem, still using non-deterministic approach, is called Partially Observable MDPs or POMDPs. An introduction to this new type of planners can be found in [128, 155]. A POMP establishes a probability distribution over the set of possible states, using the available observations and observation probabilities, and has a solution that maps beliefs into actions. Despite their inherent computational complexity, numerous variants have been proposed through the years to efficiently tackle special classes of problems. A way to tackle POMDPs' complexity is to make assumptions about the state posterior distribution, rather than leaving it unconstrained. A common choice is to assume a Gaussian distribution, thus bridging classic estimation and control theory with motion planning algorithms [170, 171]. Uncertainty about the state can also be tackled by developing so-called risk-aware planners that explicitly account for uncertainty in the execution of plans [107, 102]. In chapter 6 we deal with the same problem of having an unknown environment, but, instead of increasing the size of the local space state and calculating a policy over the tree of posterior distribution, we reformulate how we create the map and describe the environment. This new representation allows us to significantly reduce the size of the state space and thus, find optimal policies even in run-time.

2.2 Context for the Rapid Deployment Problem

As we previously mentioned, CMDPs can be used to model problems where there are multiple associated costs for each particular action. When we have a single robot solving a planning problem the solution becomes trivial. However, when we think about the same planning problem, but this time using a group of robots, we face a new set of problems and challenges to overcome. In this section we review related work for robot constrained deployment under uncertainty. We can cite Carpin et al. [34] who has addressed the problem of a multirobot team deployment analyzing the trade-off between speed of the robot and the probability of success, despite motion uncertainties, and the number of robots used. This work sets the base to solve navigation problems where the speed of a robot and uncertainties play an important role in the coordination of a team of robots. Figure 2.5 shows the particular problem for a multi-robot team being deployed in a known environment that is represented with a graph. This graph abstraction can be readily extracted from occupancy grid maps [91]. In our previously cited paper [160] we presented a new extension of the fast deployment problem, using, as mentioned, a CMDP for a navigation multirobot and multitarget problem with not just motion uncertainties but also observation uncertainties. A larger number of targets overpass the number of available robots and an efficient distribution of targets among the robots is done using an algorithm that solves the TSP while the robots respect time and probability failure constraints.

The rapid deployment problem is related to the team orienteering problem [39]. Given a weighted graph with values associated to vertices, the problem is to find M paths, from a given start to a goal location, whose length does not exceed a given travel budget B while maximizing the sum of the values for the visited vertices. In this case M is the number of team members. This problem generalizes the orienteering problem [67] that is known to be NP-Hard and also APX-hard to approximate. These problems, however, feature only one cost per edge, whereas in our setting, each edge in the graph is associated with multiple costs, i.e., time to complete the transition and failure probability.

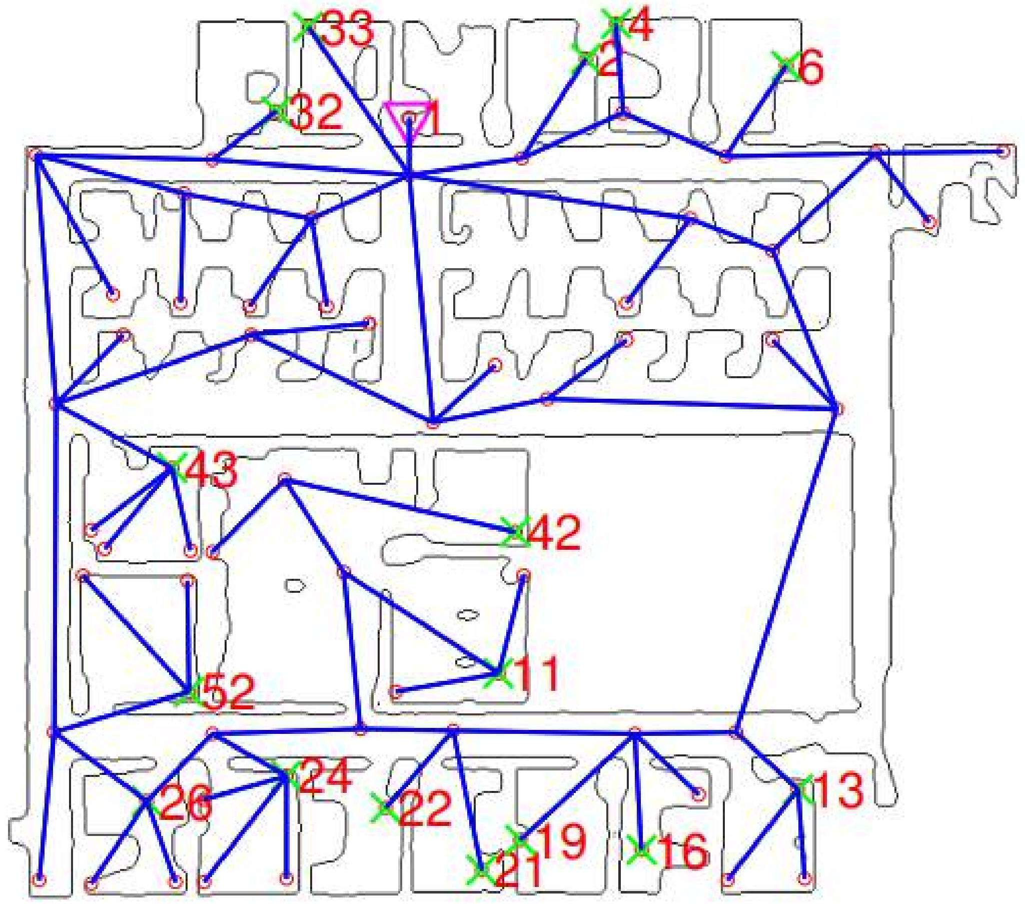


Figure 2.5: Map described as a graph with multiple vertices, where the pink triangle is taken as a deployment vertex and the ones with green crosses as goal vertices. Taken from [34].

Moreover, the problem we consider features stochastic transitions between vertices, whereas in orienteering, and also in the related TSP [12], transitions are commonly deterministic. It then follows that the problem of rapid deployment is significantly more complex and general than orienteering, and therefore, we will have to settle for a suboptimal solution. The problem of finding tours to visit sites with stochastic transitions has been studied extensively in the operations research community [65, 16]. Laporte et al. [95] were among the first to study a variant of TSP known as the vehicle routing problem (VRP) [66] under stochastic travel as well as service times. The service time refers to the amount of time spent at each site. Since then, a number of algorithms for solving many variants of stochastic have been proposed [65, 16] typically based on chance-constrained optimization [122]. These works address not only stochastic times, but also cases where the sites themselves are stochastic [18, 50]. The problem considered in this dissertation differs from these works in that we do not have specific sites that must be visited by the robot. Instead, the algorithm must optimize for the sites to be visited based on the visibility sets of the targets and other input parameters.

Purohit et al. [141] also presented an algorithm to deploy a high number of low-cost, low-complexity Micro Aerial Vehicles at high speed, with a known environment and with motion uncertainty. These works precede further developments like the ones presented by Carpin et al. [41], where a Constrained Markov Decision Process (CMDP) was used to constrain the same problem while still guaranteeing an optimal policy. From this point, different extensions were implemented and successfully proved. In [166] the problem of persistent monitoring with robot teams was studied where a multi-robot system is tasked with monitoring a set of locations larger than the size of the team. Therefore, each robot is assigned multiple targets to observe. However, in [166] they do not consider sensing errors nor stochastic motion models. Finally, temporal deadlines in robotics have been extensively studied for scheduling and coordination tasks. For example, when multiple robots must coordinate their actions so that the relative temporal ordering becomes relevant [68, 71]

2.3 Mapping and Exploration

A map is a representation of the space that uses a specific set of rules and a certain structure to describe the relationship between objects and the free space. For each map there is a specific type of information to display, a different analysis and processing. There are different map representations and exploration techniques, and some of the most relevant works that relate with this dissertation, will be covered by this chapter.

2.3.1 Mapping

The use of LIDARs and distance measurement sensors have promoted the development of grid maps. A grid map discretizes the space of a certain environment with equally distributed square cells. Each of these cells correspond to a specific location with x and y coordinates in the environment. Multiple solutions have been proposed to create these type of maps and use them in robotics. Thrun goes over many of these solutions in his work [165].

While metric maps and occupancy grid maps have been common in multiple applications, new approaches have been developed. Topological and semantic maps have shown the potential to be used in complex applications (see [79] for a relatively recent survey). Vision systems gave the starting point for new algorithms to use visual information to describe the world [105] in a different way using deep neural networks. Tagging and labeling objects became a key task for autonomous systems. Multiple labeling methods and scene recognition [64] have been proposed, with the goal of understanding the objects and their relationships [52, 139]. Mendez et al.

present a novel system to upgrade a Monte Carlo localization algorithm using only labels for walls, doors, windows and eliminating the need of a LIDAR sensor [114].

Specifically for indoor environments we want to mention the work of Quattrini et al. [143], who use semantic information combined with geometric information to improve the exploration of a map that classified areas according to their semantic relevance. These areas are defined by humans as first goals for the robots. This research in particular served as the inspiration to develop a new type of map that can exploit the strength of having semantic and topological information combined. In chapter 6 we will present how a semantic exploration strategy can outperform a random strategy when recurring to the use of labels and semantics. Lowry's work is also a good example of different methods to extract semantic information from an environment, specifically visual place recognition research [106].

Other authors have shown that extracting topology information plays a valuable role when mapping an environment. Sandstr et al. [152] show how a "decomposition map" serves to locate good candidate neighbors that are "topologically relevant", improving the nearest-neighbor time and overall planning. Other methods also propose generating an informative path planning using a topological structure from an information field [113]. Ramaithititima's et al. [144] work also avoids the use of a metric map. During his experiments, the robots use a topological approach with a landmark system, which is scalable and provides a bearing-based controller. Tung et al. propose a method using the visual saliency of objects and the environment and drive the robot towards salient objects using a 3D occupancy map [44].

Extracting topological features from laser scans collected from a range finder is another well-established research area and a number of current algorithms successfully perform this task within the scope of the definition of a topological feature. Extracting information from these laser scans, either 3D or 2D is a key component that enables successful localization and exploration strategies. Corner detection, for example, uses LIDAR data as landmark points [13] for better localization. Shah et al. [153] uses Vornoi-Delauny graphs in order to extract qualitative information regarding two nearby landmarks for path-planning. Delauny triangulation has been used in [111] to provide information regarding the layout of the environment. Line segment extraction from 2D LIDAR data has been widely used in a number of different cases[137, 27, 123, 1, 9, 112]. Pfister et al. [137] use a weighted line fitting algorithm that generates line segments. Brunskill et al. [27] use PCA dimensionality reduction to represent the large dimensional 2D point cloud into a four parameter line segment represented by the end points. Nguyen et al. [123] use a split-and-merge technique to generate line segments from the 2D laser scan data.

The use of deep reinforcement learning to include topological and structural information about a building to improve exploration is a contemporary topic for some researchers like [186]. Luperto [109] also presents a similar approach to build a topological structure and a semantic labeling for indoor environments. Other approaches combine geometric, topological, and semantic information to generate maps for robots and humans [55, 120, 108, 42].

We present in figure 2.6 three examples of maps, 2.6(a) shows a grid map where a maze is divided in square cells and each cell represents a location x,y in the map. We also show in 2.6(b) how a building can be decomposed into connected areas, where we ignore the distances and other metric information. Finally, we show a semantic map in 2.6(c) that gives labels to the locations of an office. Each of the vertices is colored according to the type of location: red for corridors, blue for offices, and green for a meeting room. This particular work also inspired our Oriented Topological Semantic Maps explained in detail in chapter 6.

2.3.2 Map Merging

LIDARs, cameras, radars, and other sensors help robots and other autonomous vehicles [115] to generate maps that extract valuable information from the surroundings in order to make appropriate decisions. While mapping can be done by one robot, one of the main goals in this area is to achieve multi-agent mapping, where individual robots work together to build a whole mapping network, that become more resilient, flexible, and robust. Specifically chapter 8 deals with this problem and presents a new technique to merge pieces of maps together for indoor environments.

Early works in map merging aim at combining multiple grid maps into one global map using metric features. Carpin [31] used Hough features for the bidimensional case and the Radon transform for the three-dimensional case [33]. Blanco et al. [19] use computer vision techniques to extract features and compare them, in what they call multi-hypothesis RANSAC stage, to match them later to the grid map. Paulik [133], on the other hand, merges a hybrid, feature-metric map finding a transformation matrix to match the pieces of maps. A function to measure the overlap between pixels is proposed and used to calculate the quality of the matching. Park et al. [132] use geometric information to match shapes and geometric features to find overlapping points between maps. This approach gives an important advantage when eliminating the need of knowing the relative position of the robots. Another method, implemented by Karpov [82], includes communication between the robots and a landmark system to localize the robots and find matches between the sections of map that each robot builds. For improving vehicle positioning, Rohani et al. [150] merge road maps between vehicles in a VANET network. The GPS error from each individual is taken into account to find the matches, and then a dynamic base station is used to help other vehicles to improve their location when they are not part of the network.

Merging topological maps is often related to the problem of graph merging, because of the underlying graph representation. When trying to merge these graphs, we have some examples of how we can obtain a global map by combining topological with metric information. For example, Bonanni et al. [22] extract a graph from a 3D metric map and try to match the vertices of the graph. This is the opposite of the common approach of applying geometric transformations to each of the map sections to find the match between them. In our work, we do not need to know the global or the relative pose of the robots to merge the maps. Similarly, when merging topological maps for rectilinear worlds, Huang et al. [77] generate hypotheses of how two graphs match depending on the number of outgoing edges. Then, they augment the topological map with metric information (like local distance) while in our work we do not use any concept of distance.

Finally, there are works showing the advantages of using hybrid maps to encapsulate, at once, more information. Shahbandi et al. [154] propose a multi-modal map alignment where multiple maps of different types can be combined into a global model that contains occupancy grid maps, 3D meshes and also semantic information. Dichtl et al. [48] introduce a new type of map called PolyMap, where an environment is decomposed in polygons that stay in the middle of a grid map and a vector map, taking advantage of the strengths of both, and allowing efficient communication and sharing for multi-robot missions.

2.3.3 Exploration and SLAM

Exploration is one of the main challenges for a robot facing a new environment, and there is no consensus about the best strategy to follow. Ultimately, this is application specific and is still an active research topic. There are many ways to explore a certain environment and it is commonly linked to the need of a specific type of information, the robot capabilities, or the goals. For instance, for underwater applications, Vidal et al. recently proposed a two-steps algorithm where a map is used to produce viewpoints, and they are evaluated based on an information measurement [175]. Another example for volcanos is presented by [30], and another for a sewer environment by [179]. Because of this broad set of applications, in this section we will mainly talk about robot exploration for indoor environments.

Exploration can be done autonomously or assisted. An example of the later is the work of Stotko et al. [157] where a novel VR system helps a human to teleoperate a robot. Vilela et al. also present a semi-autonomous exploration method to coordinate robot teams in urban areas [178]. Although tele-operation is a valid way to explore an environment, autonomous robots have been the focus of multiple researchers: Carvalho [36], Albina [3], AlKhawaldah [2] and Ocumura [127] are just few cases of multi-robot coordination to explore different scenarios. A related work for indoor environments, assessing the use of Discrete-Time Markov Process is done by Andre and Bettstetter [10], where they quantify the gain to coordinate robots and determine how varying the group size impacts the results. A survey about multi-robot coordination can be read from [184].

Together with exploration, SLAM is another keyword closely related. Simultaneous Localization and Mapping (SLAM) groups the study of algorithms that build or update a map, while at the same time, it calculates the robot's position or state [29]. The vast majority of SLAM related research embraces a single-robot approach, but when increasing the number of robots, several challenges arise, like estimating relative poses of the robots, uncertainty of the relative poses, concurrent updates of maps and poses, communications, and others. Saeedi et al. [151] present a complete review for multi-robot SLAM research in the past years.

When the environment that is being explored uses a graph to represent the relationship between objects or locations in the space, we commonly know it as graphSLAM: Graph Simultaneous Localization and Mapping [164]. For graphs with millions of vertices and edges, we have to deal with memory, processing delays and limitations. However, few vertices means a reduced amount of information that is sometimes insufficient to finding the optimal solution of an over-discretized environment. When we use visual information and semantic relationships, we refer to VisualSLAM, which is a good way to increase the effectiveness of a small graph. Varadarajan et al. [174] proposed a method to keep a small graph when creating a topological map to represent an environment, but by augmenting the amount of information with visual "semantic affordances" to define what they called "Subject-Object" relationships. This method, for example, manages to solve the loop closure problem and a dynamic environment. Gao and Zhang [62], instead of using object recognition and creating semantic relationships between them, use the leverage of deep neural networks to learn sparse feature representation with a stacked auto-encoder (SDA) for VisualSLAM. Other methods use the well-known Bag of words algorithm to solve the loop closure problem like [56] or a combination of a monocular, stereo and RGB-D sensors [118, 72].

As we mentioned, visualSLAM approaches take advantage of RGB and RGB-D camera sensors. Earlier use of these sensors involved using a marker based approach where QR codes were used to identify a particular place of interest in the environment to provide landmarks for a localization approach [94]. However, a more general approach proposes the extraction of features from the images captured by the camera that are used to estimate the pose of the agent [187, 124, 45, 89]. More recently, the depth data is combined with the 2D images to extract semantic information of the environment to construct hybrid maps [94] and these techniques employ prevalent object identification and segmentation[43] of the scene to achieve this task. Dube et al. [51] generate a SegMap by extracting segments from 3D point clouds by using a CNN that is trained using two loss functions, L_c (softmax cross entropy loss or the classification loss) and L_r (retrieval loss). Grinvald et al. [70] uses mask R-CNNs [74] framework to detect instances of objects and generate a per-pixel segmentation mask that contains semantic information.

2.4 Replicability and R-Articles

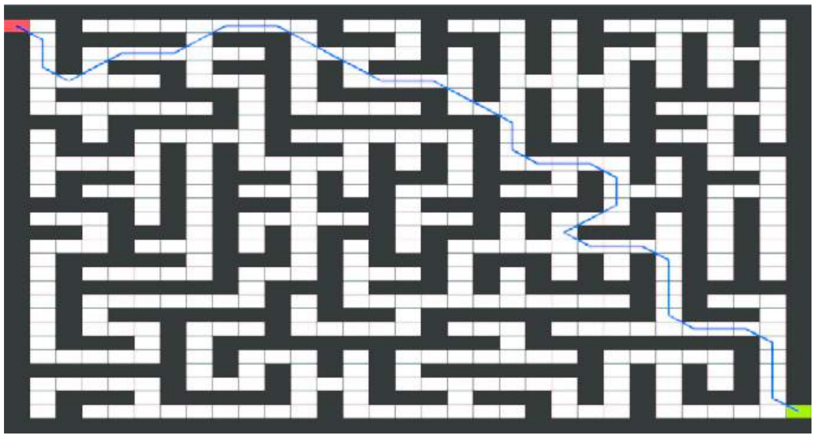
Different efforts in robotics, either in navigation, exploration, mapping fall into the oblivion due to the impossibility of reusing and replicating the results presented to the community. In other fields replicabiliy has been also addressed. For example, for Mining Software Robles [147, 148] makes available a downloadable package, including scripts, and raw data available to replicate their results. In Neuroscience, we can also find some examples of the discussion about replicability like the one from Miłkowski et al. [116] and how four pilars must be respected to reach the replicability goal. Equally, in VGI-related research, Ostermann [129] identifies the need to create a new author-publisher-consumer model in order to reach faster and sustainable research. In the robotics community, in order to overcome this issue, the IEEE has been experimenting a new paradigm for papers in robotics that is finally giving the resources to the scientific community in robotics to share and find new applications. In 2009 and 2015 Bonsignorio [25, 24] presented the need to generate a new standard for papers in robotics where data sets, code identifiers and HW identifiers become a common word for the new papers in robotics. Cervera [37] also investigates the ambiguity, incompleteness, and incorrectness of robotics software used by the most recent papers of the International Conference on Robotics and Automation in 2017. He and Pörtner [138] identify that a possible solution to create papers with reproducible results is to use a Docker container that can be downloaded and used by the community. This issue has been also addressed in the past by Stoelen et al. [156] hoping to share control system for a human-robot system. Amigoni et al. [7, 8] in a more recent work, presents a new approach to improve the replicability of SLAM algorithms and consider the need to create a common software domain to run simulations and SLAM modules.

Parallel efforts are related with the development of Benchmarking Data Bases to test and compare algorithms and solutions. Garcia-Camacho et al. [63] offers three different benchmarks for clothes manipulation, including folding, spreading and dressing. Similarly Mnyusiwalla [117] and Triantafyllou et al. [168] propose a benchmark framework for the very popular task of pick and place. Other benchmarks in robot manipulation can be found from the work of Yang [185], and Leitner [99]. Altough benchmarks for robot manipultors are fairly popular, there are examples of other types of benchmarks aimed to test and compare algorithms for motion planning on roads [4] and posture Control—Human-Inspired methods[185].

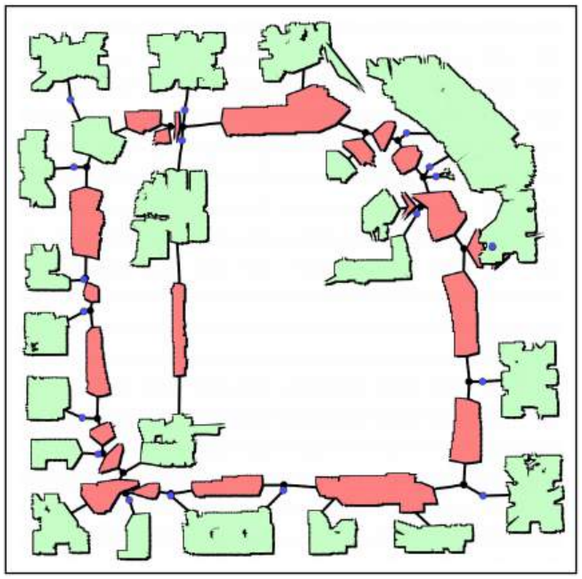
Since 2017 [23] the IEEE Robotics & Automation Magazine has been formally calling for these new types of papers or articles whose experiments and results are fully reproducible by others researchers. These new type articles will be known, from now on, as **R-ARTICLES** [2] and will be the base for a new way to present and accelerate research, with reusable and testable software.

As part of this initiative to create the first R-article and serve as the new standard of all new papers of the same type, we proposed and published [159] the first R-article in robotics [172] and it is discussed in chapter 6.

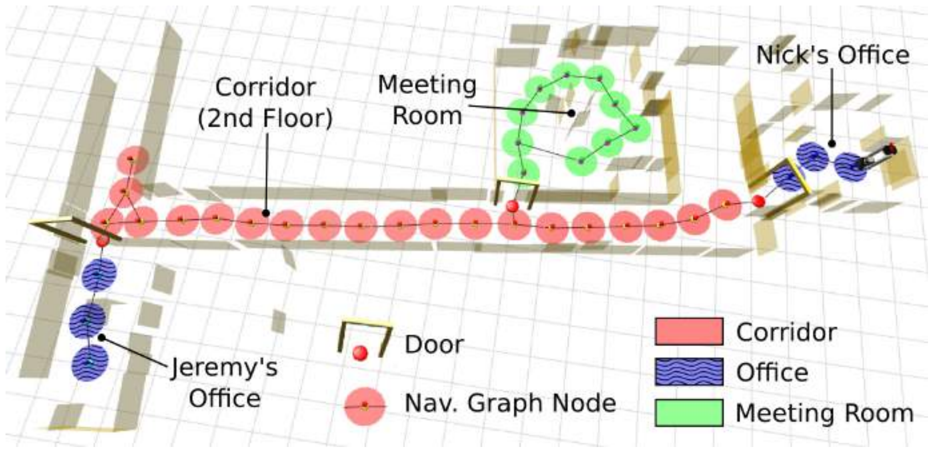
[2]R-Articles - IEEE Robotics and Automation Society, `https://www.ieee-ras.org/publications/ram/information-for-authors-ram/reproducible-articles-r-articles-short-replication-articles-r-articles-reply-articles`, last visit: 2020-02-14



(a)



(b)



(c)

Figure 2.6: (a): Example of a gridMap. Taken from [78]. (b): Example of a Topological map. Taken from [60]. (c): Example of a Semantic Map. Taken from [140]

Chapter 3

Theoretical Background

3.1 Introduction

This chapter provides the general theoretical background for the following chapters in this dissertation. In section 3.2 we will present the theoretical background of Markov Decision Processes (MDPs). Having the base about discrete planning and MDPs we will cover in section 3.3 an extension of this method called Constrained Markov Decision Process or CMDP. We will provide a brief summary of the CMDP formalism and its advantages over MDPs.

3.2 Markov Decision Process - MDP

A Markov Decision Process (MDP) is a tool to solve stochastic sequential decision-making problems. It has been applied in several domains in various control and planning scenarios [17, 163]. This method proposes to find a policy that defines for each state, a single action with a stochastic outcome [17]. The policy uses each of the associated actions to direct the robot through the optimal path to a goal state, where one single function, which depends on the overall trajectory and not just the final state, is minimized or maximized according to the desired objective.

A MDP model applied in robotics consists of mapping between states and actions. Usually the map of an environment is discretized into regions (generally, uniform square cells), and one or multiple cells will serve as goal cells. The objective is to arrive to these cells from any initial location. Each cell will be described later as a node in a graph (Figure 3.1).

The MDP model calculates the best action for each of the empty cells (i.e. state) in a map that leads to the goal location. The mapping between the states and actions is what we call a *policy.* Each possible action will have a different cost/reward and, therefore, only one can be the best to take in order to arrive to the goal. It is important to note the non-deterministic nature of the MDP. The outcomes of the actions are uncertain, and one action can lead to different next states, and for each state and

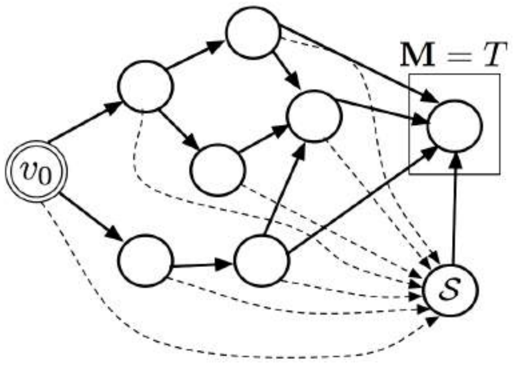


Figure 3.1: Example of a graph taken from an environment where the deployment point is named as v_0. For any state there is a chance to fail and end in a sink state $\mathcal{S}$ while going to a target state T. Image taken from [41]

action we specify a probability distribution over the next states. However, the policy itself is deterministic and there can only be one best action that is executed for one state. There are some works that study non-deterministic policies [54] but they go beyond the scope of this work. Moreover, for the case of finite, discrete MDPs, it is easy to prove that deterministic policies are optimal [161].

Depending on the specific form of the objective function, MDP problems can be categorized into four main classes: finite horizon MDPs, infinite horizon MDPs with discounted cost, average-cost infinite horizon MDPs, and total cost MDPs (or optimal stopping MDPs), where the cost is on an infinite horizon, but the state will eventually enter an absorbing set where no additional costs are incurred. For all the cases, at time t, we will execute an action a_t that may produce a change of the current state x_t (except for the total cost MDPs with an absorbing state).

- Finite horizon: This objective function is determined under a fixed amount of time. For this kind of MDPs we optimize the expected value:

$$\min_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} c(x_t, a_t)\right] \tag{3.1}$$

- Infinite horizon discounted cost/reward: In this case there is no time limit, which will cause the expectation to go also to infinity if a discounted factor is not added. This discounted factor γ, whose values goes from 0 to 1, will limit the expectation and ensure a finite expectation value:

$$\min_{\pi} \mathbb{E}\left[\lim_{T\to\infty} \sum_{t=0}^{T} \gamma^t c(x_t, a_t)\right] \tag{3.2}$$

- Infinite horizon average cost/reward: This type of MDP solves the problem of having an infinite expectation by averaging the sum over the total time.

$$\min_{\pi} \mathbb{E}\left[\lim_{T\to\infty} \frac{1}{T}(\sum_{t=0}^{T} c(x_t, a_t))\right] \tag{3.3}$$

- Infinite horizon total cost/reward: In this type of MDP, there is no discounted factor γ. In order to avoid the unbounded expectation, we set one of the states as an absorbing state, which will have no cost associated for all possible actions. It is also "trapping", which means, after reaching this state no cost will be added and you can't leave the state. This will ensure that the sum of the costs never goes beyond a certain value. For this type we have an optimization function:

$$\min_{\pi} \mathbb{E}\left[\lim_{T\to\infty} (\sum_{t=0}^{T} c(x_t, a_t))\right] \tag{3.4}$$

Due the fact we want to use MDP in a robotics context, it makes sense to use a total cost MDP, which represents a mission with an unknown, but still finite expectation value and amount of time to complete. It is possible to define three types of total cost MDPs: (i) transient MDPs, for which the total expected time spent in each state is finite under any policy, (ii) absorbing MDPs, for which the total expected "life time" of the system is finite under any policy, and (iii) contracting MDPs [5]. One can show that all three types of total cost MDPs are equivalent under the assumption of a finite state and action sets [5].

3.2.1 MDP Formulation

A time invariant MDP can be formulated in the simplest case as a finite quadruple X, A, P, c where:

X is a finite state space with n elements. The state at time t is indicated by the random variable X_t.

A is a collection of n finite sets, one for each state in X. $A(x)$ is the set of actions that can be applied in state x. A_t is the action taken at time t.

P is the transition probability function. We define $P^a_{xy} = P(X_{t+1} = y | X_t = x, A_t = a)$ as the probability that the state transitions from x to y when action a is taken. This probability is assumed to be stationary.

$c : X \times A \to \mathbb{R}_{\geq 0}$ is a cost function. $c(x, a)$ is the cost incurred when applying action a while in state x.

This model is defined in discrete time, with initial time t_0. The *stationary* property indicates that the cost and transition probabilities are independent of time. The base problem that a MDP solves is to find a function called policy π that specifies for each state x the action $a = \pi(x)$. The output of this policy π is then a set of pairs of state-action, indicated as (X_t, A_t).

In order to define a total cost MDP, we need to make the assumption that the MDP is transient, which ensures a finite cost. The state space X can be partitioned into a set X' and a set M set with the following properties:

1. $\sum_{t=0}^{\infty} P^{\pi}(X_t = x) < \infty for\ every\ x \in X'$: Eventually the state will enter set M.
2. $P_{xy}^{a} = 0\ for\ each\ x \in M\ and\ y \in X'$: Once the state enters M it cannot leave it.
3. $c(x, a) = 0\ for\ each\ x \in M, a \in A(x)$: Once the state enters M there is no additional cost.

where $P^{\pi}(X_t = x)$ is the probability that $X_t = x$ when following policy π. The total cost of π is then:

$$c(\pi) = E_{\pi}\left[\sum_{t} c(X_t, A_t)\right] \tag{3.5}$$

E_{π} is the expectation induced by the policy π over the sequence (X_t, A_t).

Using the Bellman Equation [15] we obtain the value function:

$$V^{\pi}(x) = E_{\pi}\left[c_t + V^{\pi}(x_{t+1}|x_t = x)\right] \tag{3.6}$$

$$= \sum_{x' \in X} P(x, a, x')\left(c(x, a, x') + V^{*}(x')\right) \tag{3.7}$$

An optimal policy π^* can be obtained solving the following optimization problem:

MDP Problem. Let $\mathcal{B} = \{X, A, P, c\}$ be an absorbing MDP. Determine a policy π^* solving the following optimization problem:

$$\pi^*(x) = \arg\min_{a \in A} \sum_{x' \in X} P(x, a, x')\left(c(x, a, x') + V^{*}(x')\right) \tag{3.8}$$

This policy can be called a *greedy policy* because it selects the best action for each state using the value V. This policy can be found using an iterative method such as value iteration or policy iteration. The definition of these methods are beyond the scope of this document, see [81], [142] and [17] for a more detailed explanation.

MDPs are formulated considering a single objective function and this may be a limiting factor when robots are tasked with complex missions featuring more than one objective at once. For each action that the robot executes there is only one cost associated with this action. As an example, we can consider driving a car. When we drive a car, we want to know how much gas we will use to reach a certain point and we want to calculate the optimal path to spend the least amount of gas on our trip. Additionally, we are constrained with time and want to know how much time we will spend completing our task or how much time we have to complete our task. In formulating our driving task, we would want to optimize all three of our variables. MDPs limit the solution of these types of problem, and we need to extend its capabilities to a more complete model called Constrained Markov Decision Processes or CMDPs.

3.3 Constrained Markov Decision Process - CMDP

A MDP optimizes a single objective function. For this reason an extension of MDP was implemented and is known as a Constrained Markov Decision Process (CMDP). The CMDP model overcomes this limitation by allowing more than one cost [5]. In the CMDP formulation, one cost is minimized while the remaining costs are bounded (all in expectation). This brings a new tool capable of solving a multi-cost problem, whose different constraints limit the planning.

The CMDP model is used in a similar way as we use the MDP, however the applications of a CMDP are beyond the scope of the MDP, specially when tackling problems where we want to find a policy that guides a robot to a goal, while at the same time, including multiple costs that describe all the limitations of the robot and the task itself. For example, we may want to limit the time to reach the goal, and also set a limit for the accepted rate of failure when executing an observation task, or the global success rate of the mission. All these extra costs and constraints give the CMDP model the ability to solve more complex problems.

3.3.1 CMDP Formulation

We can extend the previous definition of the MDP model by adding additional costs and constraints. We exclusively consider finite, discrete time models. In the following, if X is a finite set we indicate with $\mathbb{P}(X)$ the set of probability mass distributions over X. A CMDP $\mathcal{C}$ is defined by $X, A, P, c, \{c_i\}_{i=1}^{L}, \{B_i\}_{i=1}^{L}, \beta$. The meaning of the components is as follows.

- X is a finite set of n states.
- $A = \bigcup_{i=1}^{n} A(x_i)$ the union of n finite sets, where $A(x_i)$ is the set of actions that can be executed in state x_i, and therefore A is a finite set, too. Starting from X and A, we define the state/action set $K = \{(x, a) \in X \times A \mid x \in X \land a \in A(x)\}$.
- $P : K \rightarrow \mathbb{P}(X)$ is the one step transition kernel defining the probability distribution of the next state for a given state/action pair. In the following we

will also write $P(x, a, x')$ for the probability that the next state is x' when executing action a from state x.

- $c : K \to \mathbb{R}_{\geq 0}$ is the primary objective function. $c(x, a)$ is the cost/reward obtained when executing action a from state x.
- $c_i : K \to \mathbb{R}_{\geq 0}$ are L cost functions. $c_i(x, a)$ is the i-th cost incurred when executing action a from state x.
- B_i are L real constants defining cost bounds. Their role in the definition of a CMDP problem will be clarified after specific cost criteria are introduced in the following.
- $\beta \in \mathbb{P}(X)$ is the initial mass distribution over X, i.e., $\beta(x)$ is the probability that initial state is x.

A Markovian, randomized policy is a function π associating to each state $x \in X$ a probability mass distribution $\mathbb{P}(A(s))$ over the set of actions that can be executed in x. It is well-known that for MDPs, deterministic policies are sufficient to optimize with respect to most cost criteria, but for the case of CMDPs randomization is necessary. Another notable difference between MDPs and CMDPs is that for a given cost criterion, the optimal CMDP policy, in general, depends on the initial distribution β, while for MDPs the optimal policy is independent of the initial state. In the above definitions, by assuming that c and c_i are defined over K, we are implicitly restricting their definitions over legitimate state/action pairs. Similarly, the definition of P is restricted to valid state/action pairs. We define c as a cost/reward and the c_i as costs, so when we define the associated optimization problem, we will aim at minimizing/maximizing the cost/reward function induced by c while being subject to upper bounds for the cost functions defined by the c_i.

As for CMDPs, two discrete time stochastic processes are defined – one for the states and one for the actions. In the following we indicate with X_t the random variable for the state at time t and with A_t the action taken at time t. At time $t = 0$, a robot starts from a state x_0, and executes action a_0. As a consequence of this action, it receives a cost/reward $c(x_0, a_0)$, it incurs in L costs $c_1(x_0, a_0) \ldots c_L(x_0, a_0)$, and it moves to a new state x_1 according to the probability mass distribution $P(x_0, a_0)$. Next, at time $t = 1$, a new action a_1 is executed, and the process repeats. Starting from c and the c_i, different cost/reward or cost functions can be defined. From now on, we focus on the total cost criterion.

For a given Markovian, randomized policy π and an initial probability mass distribution β, the probability of each realization of the stochastic processes for the state and the action can be explicitly computed. Therefore, the following expected cost/reward can be introduced

$$c(\pi, \beta) = \mathbb{E}\left[\sum_{i=1}^{+\infty} c(X_i, \pi(X_i))\right] \tag{3.9}$$

where the expectation is taken with respect to π and β. If one considers c_i instead of c in Eq. (3.9), then L costs $c_i(\pi, \beta)$ can be similarly defined. Without making further assumptions, the cost/reward and the costs may be unbounded, and it is therefore necessary to restrict our attention to special cases for which these functions are bounded. This special class of CMDPs and policies are called absorbing and follow the same properties (2) and (3) that we saw for the absorbing set M in the previous section. Moreover, we assume that this partition is maximal, i.e., M is a subset of X.

However, without making additional assumptions, there is no guarantee that the state will eventually enter M. This is ensured by the definition of a transient policy. If $\mathcal{C}$ is an absorbing CMDP, we say that π is a transient policy if $\sum_{t=0}^{+\infty} P_{x_0}^{\pi}[X_t = x] < +\infty$ for each $x, x_0 \in X'$, where $P_{x_0}^{\pi}[X_t = x]$ is the probability that the state at time t is x given that the initial state is x_0 and policy π is followed. In the following, when considering transient policies we will implicitly assume that the underlying CMDP is absorbing. For a transient policy π, it is immediate to verify that $c(\pi, \beta)$ and $c_i(\pi, \beta)$ are finite for every initial distribution β. Based on these definitions, we can therefore define the CMDP planning problem.

CMDP Planning Problem. Let $\mathcal{C} = \{X, A, P, c, \{c_i\}_{i=1}^{L}, \{B_i\}_{i=1}^{L}, \beta\}$ be an absorbing CMDP. Determine a policy π^* solving the following optimization problem:

$$\begin{aligned} &\pi^* \in \arg\min_{\pi \in \Pi_M} c(\pi, \beta) \\ &\text{s.t.} \;\; c_i(\pi, \beta) \leq B_i \quad 1 \leq i \leq L \end{aligned}$$

where Π_M is the set of randomized, Markovian policies for $\mathcal{C}$.

3.3.2 Solving CMDPs

We stated in section 3.2 that a MDP can be solved using policy or value iteration, however due to the fact that we are optimizing over more than one cost we can no longer use an iterative method. It is necessary to arrange the problem in the form of a linear program in order to solve a CMDP. A well known result in CMDP theory states that solving a CMDP problem is equivalent to solving a constrained linear program over a set of optimization variables known as *occupation measures*. For each $(x, a) \in K$, the occupancy measure $\rho(x, a)$ is defined as

$$\rho(x, a) = \sum_{t=0}^{+\infty} P[X_t = s, \pi(X_t) = a]$$

where $P[X_t = x, \pi(X_t) = a]$ is the probability that at time t the state is x and the taken action is a for a given policy π and initial distribution β (not explicitly indicated to avoid cluttered notation.) Note that unless additional hypotheses are made, an occupancy measure $\rho(x, a)$ is not a probability. For an absorbing CMDP, let K' be the set of state/action pairs restricted to the set X', i.e., $K' = \{(x,a) \in K \mid x \in X'\}$. The following classic result establishes when and how the CMDP planning problem can be solved (see e.g., [6] for more details).

Theorem 3.1. *The CMDP planning problem is solvable if and only if the following linear program is solvable*

$$\begin{aligned}
&\max_{\rho(x,a)} \sum_{(x,a)\in K'} \rho(x,a)c(x,a) \\
&s.t. \quad \sum_{(x,a)\in K'} \rho(x,a)c_i(x,a) \leq B_i \quad 1 \leq i \leq L \\
&\qquad \sum_{y\in X'} \sum_{a\in A(y)} \rho(x,a)(\delta_x(y) - P(y,a,x)) = \beta(x) \\
&\qquad \forall x \in X' \\
&\qquad \rho(x,a) \geq 0 \quad \forall \rho(x,a) \in K'.
\end{aligned}$$

If the linear program in Theorem 3.1 is solvable, its optimal solution $\rho^*(x,a)$ defines the optimal policy π^* as follows:

$$\pi^*(x,a) = \frac{\rho^*(x,a)}{\sum_{a\in A(x)} \rho^*(x,a)}$$

where $\pi^*(x,a)$ is the probability of taking action a in state $x \in X'$. For states in M the policy can be arbitrarily set because it neither contributes to the objective function nor to the constrained costs.

Chapter 4

Rapid Deployment of Mobile Robots under Constraints

4.1 Introduction

In this chapter we will go over one of our first published papers called ”Rapid Deployment of Mobile Robots under Temporal, Performance, Perception, and Resource Constraints” [160]. This work deals with the problem of having a multi-robot, multi-target scenario with different observation goals. This case can simulate the task of monitoring an art gallery where multiple paintings need to be observed from specific locations, and we must coordinate a group of robots to complete this task. To make this scenario more realistic we also added different types of uncertainties: One of these is related to a motion uncertainty where the robot's control may fail and move the position of the robot to an unexpected position and orientation. We also account the fact that the robot may fail observing a target from its specific observation point. Due to these disturbances, there is no guarantee that the task will be successfully completed. For example, robots may crash due to the uncertainty in control, or may fail to observe a target due to erroneous sensor readings. Ideally, given a temporal deadline, the objective is to plan the actions for each robot in the team so that the probability of completing the task within the given deadline is maximized. We consider the case where the number of robots is smaller than the number of targets to observe. Therefore, to observe all targets, it is advantageous to split the task among the robots. This work significantly extends the research [35, 41] where the size of the team is much larger than the number of targets. In such scenario, a swarm approach is possible, i.e., multiple robots are assigned to each target and robustness is obtained through redundancy. Moreover, in [35, 41] the uncertainty in sensing was not considered.

In [166] it was considered a persistent monitoring application where the number of targets to be observed exceeded the number of robots. However, no uncertainty was considered in either the dynamics of the robots or in the sensing process. Moreover, no performance constraints were given. We consider the situation where the number of targets is larger than the number of robots. In this case, uncertainty affects both the dynamics and perception, and the robots must complete the task within a given temporal deadline. For sensing, we assume that the outcome of observations is uncertain. In particular, we suppose that even when a target can be observed, there is some probability of missed detection. This probability is not constant but is a function of the point from which the observation is made. In other words, when the robot observes a target, this can still lead to a failure of the mission due to the sensor's error.

With regards to the dynamics of the system, we consider a scenario where accuracy and speed vary. At each stage the robot is presented with a set of possible motion primitives to choose from. Each primitive is characterized in terms of execution time and success probability. For a wheeled robot one could assume that faster motions are associated with higher failure probabilities, or more risk of collision. On the other hand, for a small-size quadrotor it could also be that too slow motions result in higher failure rates. Our framework can accommodate both these cases, as long as each primitive can be stochastically characterized. At the single robot level, the control policy has to select the actions to execute in each state with the objective of observing a subset of assigned targets within the given temporal deadline. At the team level it is, instead, necessary to split the targets between the robots. We also assume that robots do not communicate while the mission is executed. Therefore, re-planning to re-balance the target assignment on the fly (e.g., to respond to individual failures) is not feasible because each robot is unaware of how the mission unfolds at the team level. In rapid deployment the task is successfully completed if and only if all targets are observed within the deadline, irrespective of which robot observes a given target.

Rapid deployment finds application in various domains, including for example search and rescue where survivors need to be located quickly. In such a case the team of robots is tasked with gathering information (e.g., taking a picture) about a set of relevant locations of interest. The scenario we consider implies that robots in the team will, in general, be tasked with observing more than one target. This assumption requires solving two problems. First, it is necessary to compute a control policy for a robot to observe multiple targets while satisfying the given constraints and accounting for uncertain perception. Second, it is necessary to solve a task assignment problem to allocate subsets of targets to the robots in the team.

The main contributions of this chapter are the following:

- In Section 4.2, we study the case of a single robot and multiple targets. This case is challenging, since it generalizes the orienteering problem [21], that is NP-Hard, by incorporating multiple costs and constraints. We show how the simpler problem of observing preassigned sequences of targets can be solved using the theory of CMDP and present a linear programming formulation to find the optimal solution.

- In Section 4.3 we combine the single robot solutions to solve the rapid deployment problem at the team level.

- In Section 4.4 we consider the problem of splitting the targets among the robots. We formulate a submodular objective function leading to a greedy algorithm achieving a $1 - \frac{1}{e}$ approximation.
- In Section 4.5, we present various simulations studying the CMDP formulation. We assess the performance of our algorithm and study how it scales with the complexity of the problem.

4.2 Observing Multiple Targets with One Robot

First, we present a model for the problem we consider, then show how CMDPs, studied in section 3.3, can be used to solve the problem of observing multiple targets with one robot under temporal and probabilistic constraints, and with uncertain dynamics and sensing.

4.2.1 Environment model

The environment is modeled by a graph $G = (X, E)$ where the vertex set X is a set of locations and an edge $e = (x, y)$ indicates that one or more stochastic motion primitives to move from x to y exists (see Fig. 4.1).

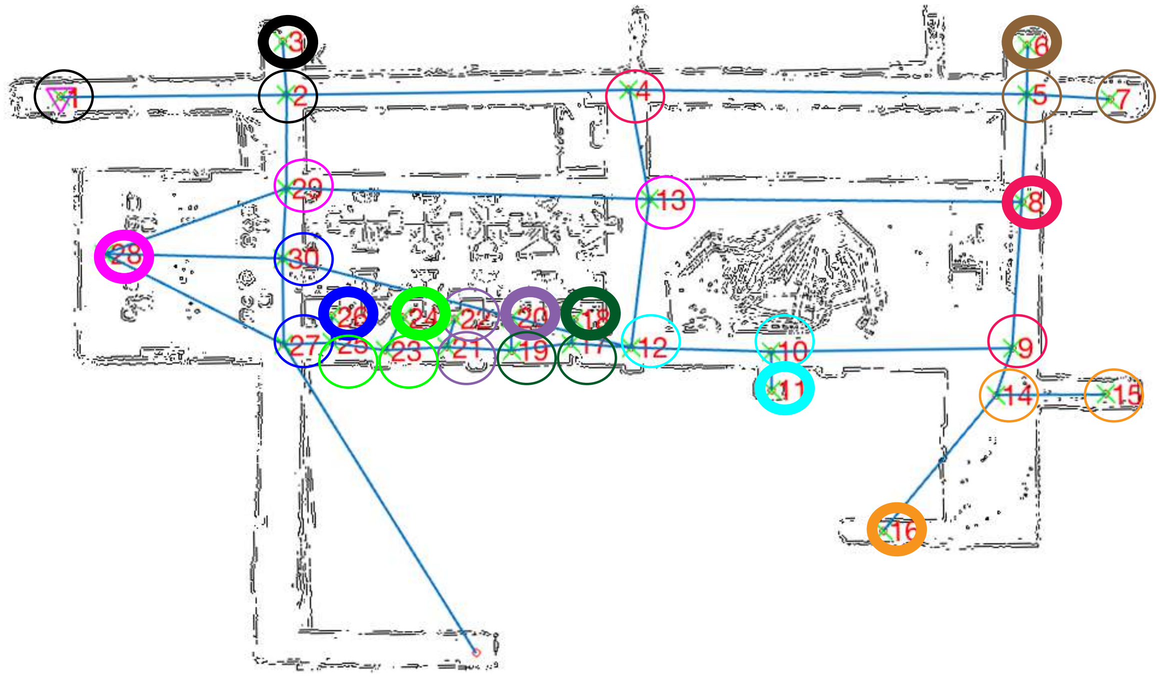


Figure 4.1: A possible setup for rapid deployment. Ten different targets are outlined by solid thick circles (visibility sets are displayed by thin circles of the same color.)

Let a be one of the stochastic motion primitives associated with edge connecting x to y. Motion primitive a will succeed or fail with a certain probability, and will take a certain time to execute. Different primitives between the same vertices x and y are characterized by different success probabilities and time. These primitives model the different ways in which a robot can move between two locations. In the following, $\mathcal{P}^a_{xy}$ is the probability that the motion primitive a succeeds in going from x to y whereas $1 - \mathcal{P}^a_{xy}$ is the probability it fails. Noisy observations are added as follows. Let $T = \{t_1, \ldots, t_M\} \subset X$ be a set of M stationary targets. An observation action performed by a robot can detect these targets. For each target $t \in T$ we define a visibility set $\mathcal{V}(t) \subset X$, i.e., a set of vertices from which the target can be detected if the robot performs the observation action. For simplicity, we assume that the visibility sets are disjoint, i.e., $t_i \neq t_j \Rightarrow \mathcal{V}(t_i) \cap \mathcal{V}(t_j) = \varnothing$. For $x \in \mathcal{V}(t_i)$, let $\mathcal{P}(x, o)$ be the probability that the robot will detect t_i when executing the observation action o in x. Note that since the visibility sets are assumed to be disjoint, this probability is unique and by definition larger than 0, otherwise x would not be in $\mathcal{V}(t_i)$. Note however that this assumption is introduced only to simplify the presentation and that the algorithms do not critically hinge on this hypothesis.

4.2.2 A model based on CMDPs

Throughout this work we assume that the state of the robot is its location in the map, i.e., the vertex in the graph, and that it is observable. Known location is a realistic assumption when robots are operating outdoors and GPS is available. Moreover, if a map of the environment is given, localization can be solved using various existing algorithms [163]. Our solution is then formulated as a feedback policy π mapping the location onto a mass distribution of the set of possible actions. Using a plain MDP model, for a given target t_i, it would be straightforward to compute a policy π determining the primitive to execute for each state, to maximize the probability of eventually observing t_i. However, this approach is inadequate because it does not consider the time taken to complete the observation task, and also because some robots will be necessarily tasked with observing more than one target (as there are more targets than robots). To this end, it is necessary to extend the state space with some memory so that a robot can track the targets it has already seen. One possible approach is to assign a certain subset of targets to the planning algorithm and let the planner determine a policy to visit all of them in an unspecified order. However, it is easy to see that if the robot is assigned k targets to observe, keeping track of those observed already in an arbitrary order would entail an exponential growth of the state space (by 2^k to be precise). To limit the growth in the state space, the planner is therefore, not only given a set of targets to observe, but also given the order in which they should be observed. This assumption implies that given a sequence, if the robot cannot observe the ith target in the sequence, it will not move on to observe the successive ones. This is consistent with our definition of the problem, whereby the task is successfully completed if and only if all targets are observed and therefore skipping one target would lead to failure. Alternatively, if the definition of the problem is changed to allow skipping targets, the same construction we present in the following can incorporate a counter so that the robot will skip a target after a series failed detection attempts. To be specific, if the robot is assigned k targets to be observed in sequence, say $t_1, t_2, \ldots, t_k$, the state space X grows by a factor $k+1$ since to every state we associate a string with k bits [1] indicating which targets have been observed already. Note that k bits are sufficient (instead of 2^k) because the order is specified. For example, if $k = 3$ it means the planner needs to observe 3 targets and the state $(000, x)$ indicates that the robot is in state x and has not seen yet any target. From such state the robot can either remain there, transit to $(000, y)$ for $y \neq x$ or to state $(100, x)$. In the first case it means that it moves to a different state y without making any observation, whereas in the second case it means that it has successfully observed the first target while in x. Observe that moving to state $(010, x)$ is not permitted because this would mean observing the second target before the first one, and this is not consistent with our assumptions. To reward the planner when a target is observed, we introduce augmented states of the form (n, x') where n is a string of k bits satisfying the constraints we formerly described (see Figure 4.2). A reward function taking advantage of these additional states will be introduced later on. When the robot makes an observation from a state x_j belonging to the visibility set of a target, it will detect the target with probability $P(x_j, o)$. In this case it makes the two transitions indicated in Figure 4.2 to mark that it has seen the target. If the robot is in state (n, x_j) and $x_j \in \mathcal{V}(t_k)$ but t_k is not the next target to be observed in the sequence encoded by n, then the observation action cannot be executed.

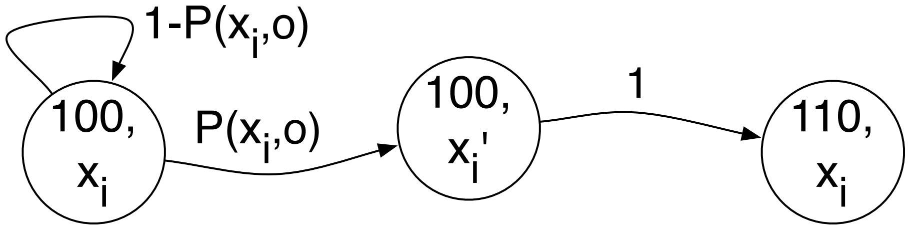


Figure 4.2: When a successful observation from state x_i is made, the state goes through an augmented state x_i' and, from there, it executes its only action. Then with probability 1, it goes back to the state x_i but with the corresponding modified bit.

Starting from $G = (X, E)$ and an ordered sequence of k targets T, we can then build a CMDP as follows. Let $\mathbb{N}_T$ be the set of k bit strings we formerly defined. From X, a state space $X' = \mathbb{N}_T \times X$ with $\mathbb{N}_T \times |X|$ states is built attaching k bits to each state in X. The expanded states are defined as follows. Let, $\mathcal{T} = \cup_{j=1\ldots M} V(t_j)$, i.e., the union of all the vertices from which one of the targets can be seen. Next, we define $\mathcal{T}' = \mathbb{N}_T \times \mathcal{T}$. To distinguish elements of $\mathcal{T}'$ from elements of X', elements

[1] Note that with k targets the growth is by a factor of $k+1$ since each state is expanded with the $k+1$ strings of k bits of the type $00\cdots00, 10\cdots00, \ldots, 11\cdots10, 11\cdots11$.

of the former set will be indicated as (n, x'), whereas elements for the latter will be indicated as (n, x). Finally we introduce a sink state $\mathcal{S}$ and a final state $\mathcal{F}$. The sink state $\mathcal{S}$ is used to represent *failure*, i.e., to model the event of a robot ceasing to correctly function. Consequently, once it is entered, no more actions can be taken by the robot. The full state space is $\mathbf{X} = X' \cup \mathcal{T}' \cup \{\mathcal{S}, \mathcal{F}\}$, and the absorbing set is $\mathbf{M} = \{\mathcal{F}\}$. We next define the action set of each state. For each state of type (n, x) we add an action a if motion primitive a is available in vertex x. Moreover, for states belonging to the visibility sets, we add an action o for the *observe* action, but only if observing a target from that state is compatible with the assigned sequence of targets. These two types of actions apply only to states in X'. To each state in $\mathcal{T}'$ we add a single action going back to the corresponding state with the flipped bit, as we illustrated in Figure 4.2. Finally, to the sink state $\mathcal{S}$, we add a single action $a_{\mathcal{S}}$ going to $\mathcal{F}$. The mass distribution β is set to 1 to the deployment vertex where the robots start from, and 0 elsewhere.

The transition function is defined as follows. When an observation action is executed, it is defined as shown in Figure 4.2, i.e., it remains in the same state with probability $1 - P(x, o)$ (no detection), or it goes to the augmented state with probability $P(x, o)$ (detection). If a is a motion primitive between x and y, then $\mathcal{P}^a_{xy}$ is given by the success probability of the primitive, and we set $\mathcal{P}^a_{x\mathcal{S}} = 1 - \mathcal{P}^a_{xy}$, i.e., if the primitive fails the state goes to the sink state $\mathcal{S}$. Actions for the states $(n, x') \in \mathcal{T}$ move the state back to (n', x) with probability 1, where n' differs from n for the flipped bit. The only action in $\mathcal{S}$ is $\mathcal{P}_{\mathcal{S}\mathcal{F}} = 1$, i.e., it deterministically moves the state to the absorbing state. To conclude, we define two different costs and a reward (hence $L = 2$ in our model.) First, we define a cost $c(x, a)$, such that $c(\mathcal{S}, a_{\mathcal{S}}) = 1$ and is 0 everywhere else. The purpose of this cost is to accurately determine failure probabilities, as it was done in [35, 41]. Next we define a cost $d(x, a)$ for all state actions. d models the time to take an action. If a is a motion primitive, then $d(x, a)$ is the time to complete the primitive. If a is an observation action, we set this to the cost of making an observation, e.g., the robot may have to stop to take a picture. For all other actions the d cost is 0. Finally we introduce a reward function $e(x, a)$. The reward function is a constant, say 1, for all actions associated with states of the type (n, x') and 0 elsewhere. As it will be seen in the following, by posing the question as an optimization problem, the planner will produce policies maximizing the expected number of observed targets in the sequence. Starting from the above definitions, we can then formulate the following single-robot multiple target problem.

Single robot, multiple target observation problem (SRMTO): Given an environment $G = (V, E)$, an ordered sequence of targets $\mathcal{G} \subset \{t_1, \ldots, t_M\}$ with visibility regions $\mathcal{V}(t_i)$, parameters $\mathbf{X}, A, \mathcal{P}, c, d, e, \beta$ as per the above definition, a temporal deadline T_f and a probability bound P_f, determine an optimal randomized policy $\pi^* : \mathbf{X'} \to \mathbb{P}(A)$ mapping state into actions maximizing the expected number of observed targets in the given order, while ensuring the temporal and performance bounds are met in expectation.

The SRMTO formulation, and the following theorem, are an instance of the total cost CMDP problem formulated in Eq. (3.3.1).

Theorem 4.1. *Let $\mathbf{X}' = \mathbf{X} \backslash \{\mathcal{F}\}$ and $\mathcal{K}' = \{(x,a) : x \in \mathbf{X}', a \in A(x)\}$. The following constrained linear program has a solution if and only if the SRMTO problem has a solution and there is a one-to-one correspondence between the solution vector $\rho(x,a)$ and the optimal policy π^*.*

$$\begin{aligned}
\max \sum_{(x,a)\in\mathcal{K}'} & \rho(x,a)e(x,a) \\
s.t. \sum_{(x,a)\in\mathcal{K}'} & \rho(x,a)c(x,a) \leq P_f \\
& \sum_{(x,a)\in\mathcal{K}'} \rho(x,a)d(x,a) \leq T_f \\
& \sum_{y\in\mathbf{X}'} \sum_{a\in A(y)} \rho(y,a)(\delta_x(y) - \mathcal{P}^a_{yx}) = \beta(x) \ \forall x \in \mathbf{X}' \\
& \rho(x,a) \geq 0
\end{aligned}$$

where $\delta_x(y)$ is 1 when $x = y$ and 0 otherwise.

Proof: The proof builds upon the fact [5] that the optimization variables $\rho(x,a)$ are interpreted as occupancy measures, defined as:

$$\rho(x,a) = \sum_{t=0}^{+\infty} \Pr[X_t = x, A_t = a].$$

Because of the way we defined the costs e, these are 1s only on the outgoing arc from states of type (n, x'), i.e., states associated with the observation of a target. Hence the objective function is the expected number of observed targets. Chow et al. [41] have proved that $\sum_{(x,a)\in\mathcal{K}'} \rho(x,a)c(x,a)$ is the expected failure probability and this is bounded by P_f in the first constraint. Similarly, in [41] it was shown that $\sum_{(x,a)\in\mathcal{K}'} \rho(x,a)d(x,a)$ is the expected time to complete a strategy, and this is bounded in expectation by T_f in the second constraint. This establishes that the solution of the CMDP maximizing the occupancy measures $\rho(x,a)e(x,a)$ will produce a policy where the success probability will be also maximized together with the cost e. By construction, all additional costs e follow the sequence of k bits for each succesful observation and, therefore, going through the augmented states x' that will lead ultimately to the final state F. $\square$

Note that an occupancy measure is, in general, not a probability, but it is a probability if it sums the probabilities of mutually disjointed events. For example, the former papers [35, 41] showed that $\rho(\mathcal{S}, a_{\mathcal{S}})$ is the probability of a trajectory going through the sink state and, therefore, by definition, the probability that a robot fails during its mission. Following the same reasoning one can show, for a given target

$t_i \in T$, the solution to the SRMTO problem provides the probability that the target is observed. This probability is

$$p_i = \sum_{x \in \mathcal{V}'(t)} \rho(x, a) \tag{4.1}$$

where $\mathcal{V}'(t)$ is the set of states of the type (n, x') with $x \in \mathcal{V}(t)$. To see why this relationship holds, it suffices to observe that for each target t_i each trajectory will go through only one of the states in $\mathcal{V}'(t)$. We can find the solution vector $\rho(x, a)$ by solving the linear program formulated in Theorem 4.1. The robot can execute the optimal policy using the corresponding occupancy measures (see [35, 41] for details.).

4.3 Solving the rapid deployment problem

In the rapid deployment problem one is given N robots, a set of M targets T, and a temporal deadline T_f. The objective is to split the targets among the robots and compute the associated control policy for each robot so that the team, as a whole, maximizes the probability of completing the task within the target temporal deadline. The technique we presented in the previous section computes the control policy for a single robot under the assumption that the sequence of targets is given, as well as a probability bound P_f. However, both these elements are not part of the input. To fill this gap, an iterative algorithm seeking for the maximum probability of success can be developed, where the optimal value for P_f is determined with a sequential search.

At each iteration, the algorithm determines a set of $K > N$ sequences of targets together with the optimal policy for each sequence, solving the corresponding SRMTO problem. Note that the algorithm may not necessarily find an optimal solution for each of the K sequences. The algorithm then assigns to each robot one of the sequences for which a solution was found, and computes the success probability P' for the whole team. Upon termination, the algorithm returns the policies and the assignment of policies to robots that yields the best group performance. Algorithm 4.1 sketches this approach.

Algorithm 4.1 handles two different probabilities, i.e., P_f and P'. P_f is the failure probability bound used in the CMDP described in Theorem 4.1. This is the bound for the failure probability for each policy computed for the K sequences. Since this is not part of the input, a linear search is performed over the discretized range $[P_f^{MIN}, P_f^{MAX}]$. P_f is the failure probability for a single robot executing one of the policies. Once a policy for all sequences is computed and an assignment is selected, the group success probability P' is computed, and this is a simple algebraic exercise building upon Eq. (4.1). Finally, note that the relationship between P_f and P' is not necessarily monotone, i.e., while increasing the individual robot failure probability P_f, the group failure probability P' may increase or decrease.

Two problems still need to be solved, i.e., generating the sequences of targets (line 1), and assigning policies to the robots (line 4). Generating all target sequences is practically infeasible as soon as T features more than a handful of targets. Therefore,

```
    Data: $G = (X, E)$, set of targets $T = \{t_1, t_2, \ldots, t_M\}$, temporal deadline $T_f$,
          number of robots $N$
    Result: A policy for each robot, and success probability $P'$
  1 Generate $K > N$ sequences of targets $S_1, \ldots, S_K$;
  2 for $P_f \leq P_f^{MIN}$ to $P_f^{MAX}$ do
  3     Solve SRTMO for each sequence with $P_f$ and $T_f$;
  4     Assign one sequence to each robot and compute $P'$;
  5     if $P'$ is best then
  6         Save Assignment and $P'$;
  7 return Assignment and $P'$
```

Algorithm 4.1: Rapid deployment

it is necessary to consider only a small set of sequences. Sequences are generated as follows. First we solve the TSP problem on the complete graph of all M targets. The cost between vertices x and y is the average time to execute all motion primitives going from x to y. In our implementation we use the Concorde approximate solver [11] to quickly determine a tour. Next, we split the tour into subsequences of up to i targets with $3 \leq i \leq M$. It therefore follows that with this strategy the number of generated sequences K bounded by $\sum_{i=3}^{M} = \lceil \frac{M}{i} \rceil$, i.e., K is $\mathcal{O}(M \log M)$. The reason to discard sequences of length 1 and 2 comes after having observed that they rarely contribute to the optimal solution. The last remaining problem is group assignment, i.e., deciding which sequence of targets to assign to each robot. This is discussed in the next section.

4.4 Group Assignment Problem

Let $S_1, \ldots, S_K$ be the groups generated by Algorithm 4.1 for which the SRMTO problem admits a solution. For a solution to the rapid deployment problem to exist, it must be that $\cup_{i=1}^{K} S_i = T$. If this is not the case, then there exists at least one target that cannot be reached.

For each sequence S_j let ρ_j be the vector computed by solving SRMTO for fixed values of T_f and P_f. As formerly stated, the ρ_j vector encodes valuable information. For example, it allows us to determine the expected number of targets seen by the optimal policy π^* solving SRMTO(S_j), as well as the probability that each of the targets in S_j will be observed while executing π^* (Eq. (4.1)). That is to say, that if $t_i \in S_j$, then ρ_j, combined with the underlying model, gives the probability p_{ij} that t_i will be observed by the robot while executing the optimal policy π_j^* returned when solving SRMTO(S_j). More specifically, assuming $t_i \in S_j$, for each $v \in \mathcal{V}(t_i)$ define

$$\mathcal{X}_v = \{(n, x') \in \mathcal{T}' \mid x' = v\}$$

Then it is easy to verify that

$$p_{ij} = \sum_{v \in \mathcal{V}_i} \sum_{y \in \mathcal{X}_v} \rho_j(y, a) \tag{4.2}$$

where $y \in \mathcal{X}_v$ indicates a composite state (n, x'). The validity of this relationship is verified immediately by writing down the explicit formula for occupancy measures considering that states in $\mathcal{T}'$ have just one action, and that each target can be observed just once due to the structure of the underlying state space. If $t_i \notin S_j$, then we set $p_{ij} = 0$.

The objective of the group assignment step is to pick N of the K sets $S_1, \ldots, S_K$. Different objective functions can be considered when making the selection. In the following we cast the problem as an instance of a nonlinear assignment problem related to the "static weapon target assignment" (SWTA) problem (see [131], ch. 3). For each target $t_i \in T$, let V_i be its importance, i.e., a weighting factor indicating how important it is to reach it. If V_i is not provided, than we can set all weights to 1. Moreover, as per Eq. (4.2), let us define the survival rate as $q_{ij} = 1 - p_{ij}$. This is the probability that target t_i will not be observed if group S_j is assigned to one robot for execution. For each of the K groups, we define a binary variable x_j indicating whether S_j is selected or not. The objective function we consider is then

$$\sum_{i=1}^{M} V_i [\Pi_{j=1}^{K} (q_{ij})^{x_j}] \tag{4.3}$$

subject to the constraint that we pick at most N groups. The interpretation of the formula is as follows. For each target t_i and each assignment of variables x_j, $\Pi_{j=1}^{K} (q_{ij})^{x_j}$ is the probability that t_i is not observed by any of the robots. Note that if $t_i \notin S_j$ then $q_{ij} = 1$. The objective function is then the sum of these probabilities for all targets, weighted by the corresponding importance V_j. If all V_js are one, then Eq. (4.3) is the expected number of targets not observed by any robot. We can then formulate the Group Assignment problem.

Group Assignment (GA). Let $S_1, \ldots, S_K$ be K sequences of targets and for each target i and group j, let q_{ij} be the survival rate for i-th target and the j-th group. Select N groups of targets to minimize Eq. (4.3).

SWTA is known to be NP-complete. Therefore, to solve the GA problem one can follow at least two strategies. First, GA could be formulated as the following integer optimization problem:

$$\begin{aligned} \min_{x_1, \ldots, x_K} \quad & \sum_{i=1}^{M} V_i [\Pi_{j=1}^{K} (q_{ij})^{x_j}] \\ \text{s.t.} \quad & \sum_{j=1}^{K} x_j = N \quad x_j \in \{0, 1\} \quad 1 \leq j \leq K. \end{aligned}$$

For problems of moderate size, this can be exactly solved using a branch-and-bound approach and one could then take advantage of commercially available solvers.

Alternatively, and this is the approach we present in the following, an approximate solution can be determined using a greedy approach for the case where an exact method is too onerous.

Let A be a group of sequences from $S_1, \ldots, S_K$, and let us define the function

$$f(A) = -\sum_{i=1}^{M} V_i[\Pi_{j\in A} q_{ij}]. \tag{4.4}$$

The GA problem is then equivalent to the problem of selecting N elements from $S_1, \ldots, S_K$ to maximize $f(S_{i_1} \cup \ldots S_{i_N})$.

Theorem 4.2. *Function f as defined in Eq.* (4.4) *is submodular.*

Proof: Given a finite set N, a function $g : 2^N \to \mathbb{R}$ is submodular if for every $X, Y \subset N$ with $X \subseteq Y$ and every $z \in N \setminus Y$ the following condition holds:

$$g(X \cup \{z\}) - g(X) \geq g(Y \cup \{z\}) - g(Y). \tag{4.5}$$

First note that for $X = Y$ the submodularity condition is trivially verified. Let $X = \{S_{i_1}, \ldots, S_{i_k}\}$ and $Y = \{S_{i_1}, \ldots, S_{i_k}, S_{i_{k+1}}, \ldots, S_{i_{k+p}}\}$ be two subsets of $\{S_1, \ldots, S_K\}$ with $X \subset Y$, and let $S_z \notin Y$. The left hand side of Eq. (4.5) is then written as:

$$\begin{aligned}
&f(X \cup \{S_z\}) - f(X) = \\
&\quad - V_1[q_{1i_1} \ldots q_{1i_k} q_{1z}] \ldots - V_M[q_{Mi_1} \ldots q_{Mi_k} q_{Mz}] \\
&\quad + V_1[q_{1i_1} \ldots q_{1i_k}] \ldots + V_M[q_{Mi_1} \ldots q_{Mi_k}] = \\
&\quad V_1[q_{1i_1} \ldots q_{1i_k}(1 - q_{1z})] + \ldots \\
&\quad + V_M[q_{Mi_1} \ldots q_{Mi_k}(1 - q_{Mz})].
\end{aligned}$$

Similarly, the right side of Eq. (4.5) is

$$\begin{aligned}
&f(Y \cup \{S_z\}) - f(Y) = \\
&\qquad - V_1[q_{1i_1} \ldots q_{1i_k} q_{1i_{k+1}} \ldots q_{1i_{k+p}} q_{1z}] \ldots \\
&\qquad - V_M[q_{Mi_1} \ldots q_{Mi_k} q_{Mi_{k+1}} \ldots q_{Mi_{k+p}} q_{Mz}] \\
&\qquad + V_1[q_{1i_1} \ldots q_{1i_k} q_{1i_{k+1}} \ldots q_{1i_{k+p}}] \ldots \\
&\qquad + V_M[q_{Mi_1} \ldots q_{Mi_k} q_{Mi_{k+1}} \ldots q_{Mi_{k+p}}] \\
&\qquad = V_1[q_{1i_1} \ldots q_{1i_k} q_{1i_{k+1}} \ldots q_{1i_{k+p}}(1 - q_{1z})] + \ldots \\
&\qquad + V_M[q_{Mi_1} \ldots q_{Mi_k} q_{Mi_{k+1}} \ldots q_{Mi_{k+p}}(1 - q_{Mz})].
\end{aligned}$$

The submodularity condition is then verified because the weights V_j are positive and $q_{ij} \leq 1$ for each i, j. □

According to this theorem one can use the well known greedy algorithm due to Nemhauser et al. [121] and obtain an approximated solution with a constant $1 - \frac{1}{e}$ approximation factor.

4.5 Simulations

In this section we present some simulations showing how our solution can not only be used to determine the optimal control policy, but also to infer interesting design and analysis properties. For example it is possible to assess a priori what is the probability of successfully completing a rapid deployment task for a given temporal deadline T_f and a certain number of robots N.

First, for the environment shown in Figure 4.1 we study how the probability of successfully completing the task varies with the number of robots and the temporal deadline T_f. This is shown in Figure 4.3.

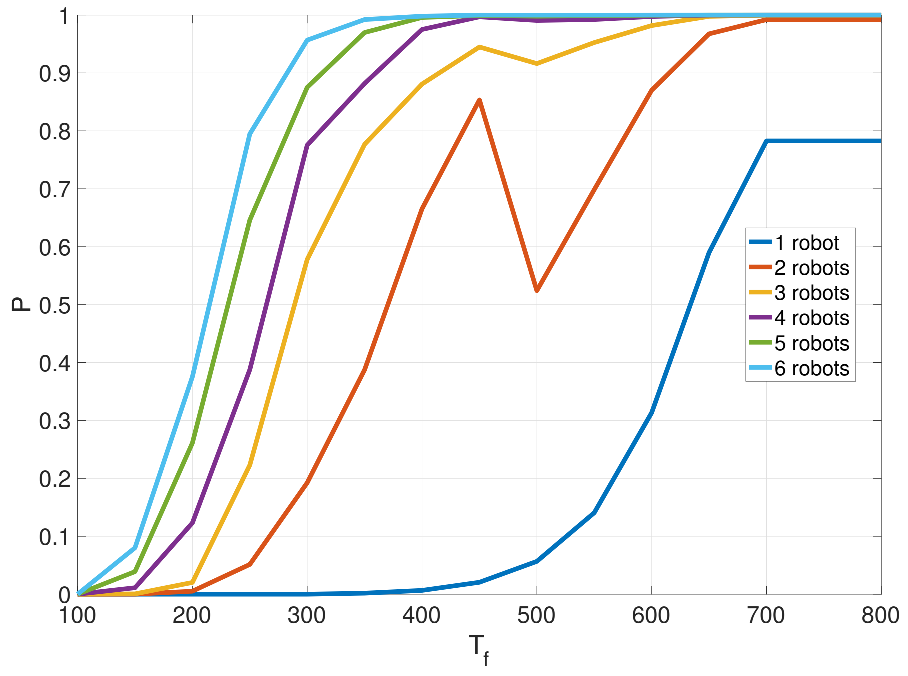


Figure 4.3: Success probability for different temporal deadlines for a different number of robots in the team for the case where 10 targets are considered.

The different curves are not only associated with control strategies, but allow us to anticipate the performance of the team. This way, based on the desired probability of success P', one can either alter the temporal deadline T_f or allocate more robots to the task. A comment is in order to explain why there is a drop in performance for $T_f = 500$. This is particularly evident for the case where two robots are considered. The reason is that for the target assignment problem we rely on a greedy allocation algorithm that is known to be suboptimal. For that specific combination of temporal deadline and number of robots, the greedy strategy picks a suboptimal target assignment that yields a performance drop. The trend of the curves can be used by a human operator (e.g., the incident commander in a search and rescue task) to decide how to allocate resources (how many robots) or how to pick the temporal deadline T_f to achieve a desired confidence level in terms of probability of success.

An important question to ask is how the method scales with the number of robots N and the number of targets M. The algorithm relies on the solution of two subproblems, namely SRMTO (policy calculation) and GA (group assignment). The greedy solution to the GA problem is a function of both the number of targets and the number of robots. In particular, it is $\mathcal{O}(KN^2)$. This bound follows because we have to iterate N times over the K sequences, and it takes $\mathcal{O}(N)$ to evaluate each candidate sequence. Considering that in Section 4.3 we clarified that the number of sequences K is $\mathcal{O}(M \log M)$, it follows that the complexity of the greedy algorithm for GA is $\mathcal{O}(M \log M N^2)$. If instead one opts for the exact method, then it is necessary to solve an integer optimization problem with K binary variables.

The SRMTO problem, is instead, by definition, independent from the number of robots because it deals with the generation of a single robot policy. We, therefore consider only, the dependency between the time to solve SRMTO and the number of targets M. More targets mean more subsequences to consider, with the associated solution of various linear programs to solve the SRMTO problem instance. As the number of targets grows, there is an increase in the size of the linear programs we need to solve because there will be subsequences with more targets. Moreover, the number of subsequences of targets grows as well. Figures 4.4 and 4.5 analyze this growth as a function of the number of targets. In particular, Figure 4.4 displays the average time (with standard deviation bars) to solve a single linear program[2] as a function of the number of targets. As it can be seen, the growth trend is roughly linear. Figure 4.5 instead plots the trend for the sum of the time to solve all the linear programs associated with a specific temporal deadline. The curve shows mean and standard deviation of the trends for different temporal deadlines. The small standard deviation confirms that this is largely independent from the T_f value and unveils a quadratic growth.

[2]For the implementation, we rely on Matlab's function `linprog`.

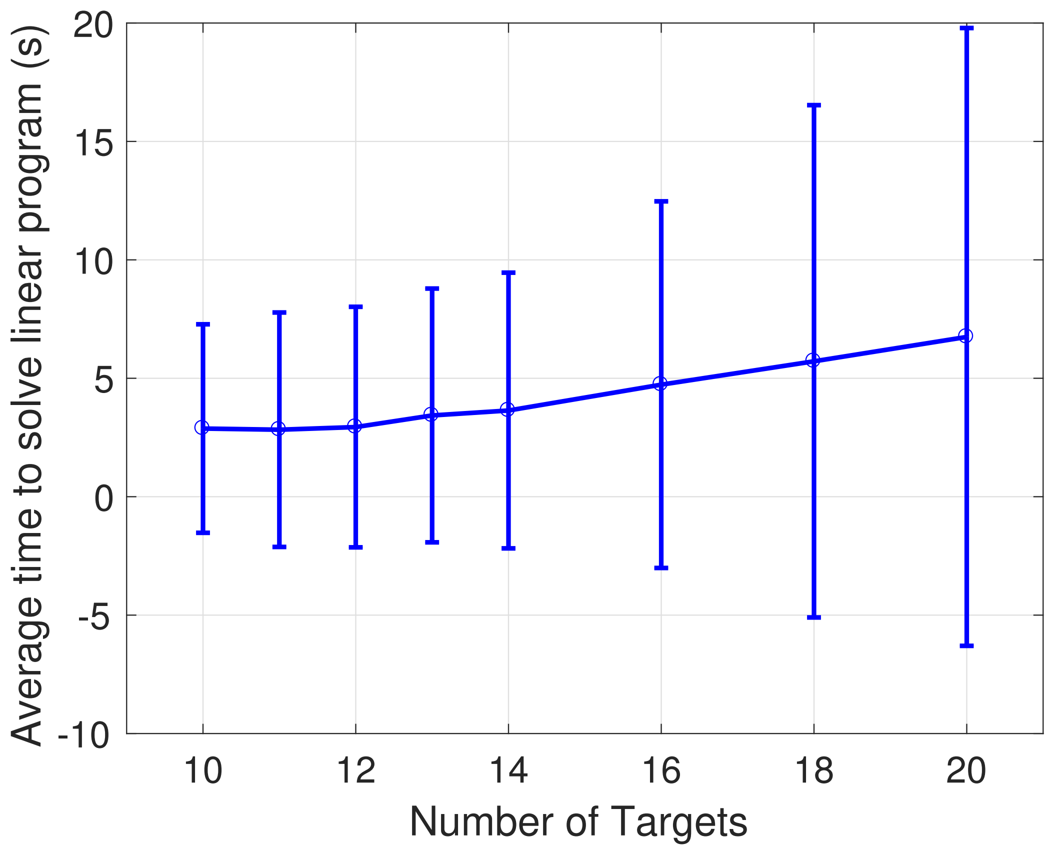


Figure 4.4: Average time to solve a single linear program for the SRMTO problem as a function of the number of targets (average taken over all linear programs to be solved for a specific number of targets).

4.6 Conclusions

We have extended previous work on the rapid deployment problem by considering two important generalizations, namely that the robots are subject to imperfect observations and that there are more targets to observe than robots available. This problem is significantly more complex than the one we considered before and is associated with some hard problems such as orienteering. In particular, it features multiple costs associated for every edge, such as the probability of success and the time to complete a primitive. We have shown that the CMDP framework we formerly formulated can still be extended to tackle a constrained robot where a single robot is tasked with observing a sequence of targets under temporal and probabilistic constraints. This building block can then be used to solve the original problem, i.e., maximizing the probability that the task is solved within a given temporal deadline. The CMDP model, in particular, allows to precisely predict success and failure probabilities at the single robot and at the team level. In

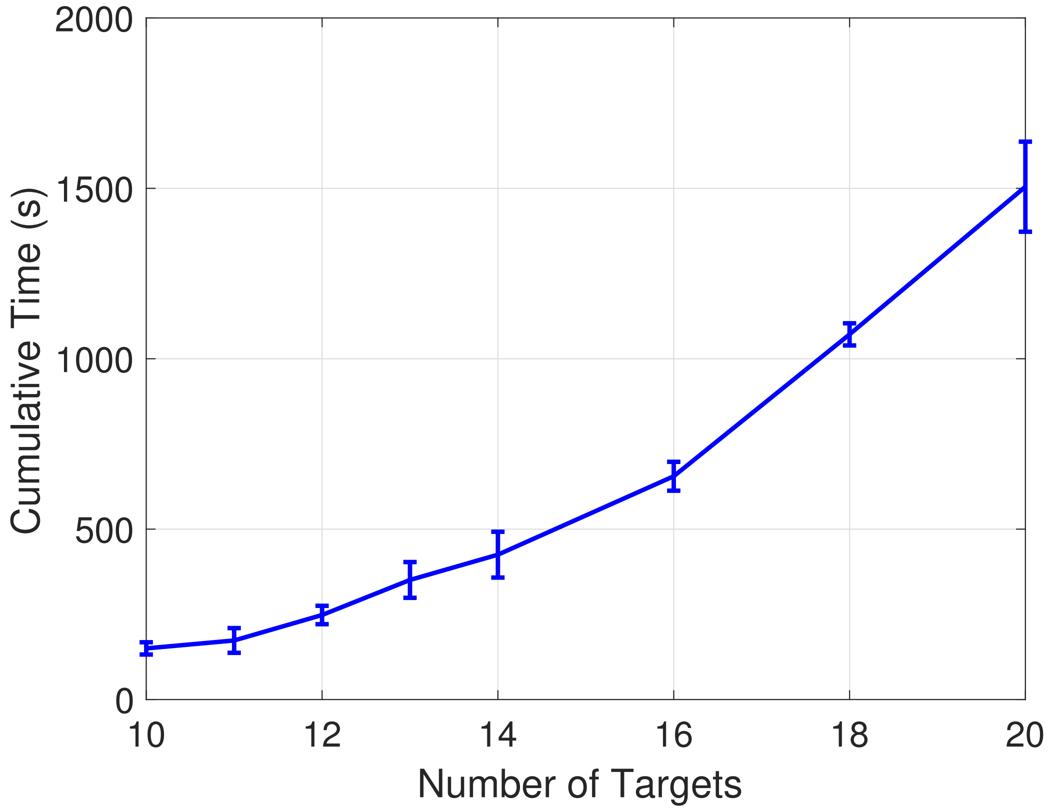


Figure 4.5: Average cumulative time to solve all linear programs to solve the SRMTO problem as a function of the number of targets (average taken over all temporal deadlines considered for a given number of targets).

addition, we have shown that the problem of assigning groups of targets to robots is related to the SWTA problem that can be approximately solved using a greedy algorithm. The simulations show how the results presented in this work can be used by a human supervisor to make informed decisions about how to allocate resources like the number of robots or the time to allocate. The method we proposed scales quadratically with the number of robots and targets.

Chapter 5

Motion Planning Under Temporal Constraints with Stochastic Motion Primitives: Theory and Practice

5.1 Introduction

In the previous chapter we proposed the solution for the "Single robot, multiple target observation problem" (SRMTO), where we modeled a multi-target scenario having multiple robots that needed to be distributed respecting some temporal and failure constraints. For that problem we assumed that the transition probabilities were given. However, in reality this assumption presents some limitations and imprecisions. We now consider the same problem of computing and executing motion policies through composition of a set of preassigned motion primitives while being subject to multiple constraints, such as task completion time and success probability. Now we will experimentally determine the motion stochasticity of our robot and how this translates into the transition probabilities that we use for the CMDP. Derived policies are aimed at and executed by a robot subject to significant motion uncertainties. Robot motion is modeled by a set of stochastic motion primitives. The latter are then composed into feedback plans, i.e. policies mapping states into actions. Our underlying assumption is that each primitive can be characterized in terms of execution time and uncertainty. For instance, consider the case where a mobile robot has to reach a target location within time T_f. Because of the inherent uncertainty affecting the motion primitives, there is no guarantee that an autonomous agent executing the plan will for sure conclude its task within the assigned deadline. Therefore, a maximum failure probability P_f is specified as well, and the objective is to determine a plan that meets the constraints: the temporal deadline T_f and the maximum failure probability P_f. These hypotheses, while useful to lay the theoretical foundations of the work, are not always verified in real world applications. In a separate effort [86], a data driven approach was developed to derive probabilistic valid stochastic models characterizing a set of motion primitives for a multi-legged palm-sized robot. In such work, starting from a set of recorded trajectories, a stochastic model for the vehicle was derived by analyzing its mechanical structure and identifying possible sources of uncertainty. The parameters characterizing these disturbances were then identified from collected data and it was shown that these models could then generate sample trajectories consistent with the available data. The power of this technique is that it can then be used to simulate and predict the stochastic behavior of motion primitives that were not part of the dataset.

In this work, for the first time, we close the loop between theory and practice by connecting these two efforts together and jointly validating them through physical experiments. Specifically, stochastic primitives are derived for differential drive robots. Given the base model and sets of experimental trajectories, stochastically-valid primitives are built. For each primitive, a spatial probability distribution is derived to characterize the anticipated outcome of the maneuver, together with the expected time to complete it. These primitives, in turn, inform a CMPD planner operating on a discretized representation of the environment utilizing uniform grids. Our findings are first demonstrated in simulation and then on a low-cost, error prone differential drive robot.

We presented in section 2.1, the related work on motion planning under uncertainty, and in this chapter the main contributions are the following:

- Section 5.2.2 shows a novel method to extract the transition probabilities from a set of trajectories describing the robot's motion under specific actions.
- Section 5.2 presents the necessary technical preliminaries to understand how a stochastic model can replicate empirical data and expand it to extract transition probabilities and use them with a CMDP.
- Section 5.3 discusses how we created several environments where we discretized the space and use it by the CMDP planner to compute a navigation policy.
- In section 5.4 we define a CMDP model to capture the problem of reaching a target while avoiding obstacles, and still respecting initial constraints.
- We present mazes with walls and narrow corridors to be navigated by a robot without any sophisticated low level control for linear and angular velocities.
- We demonstrate the robustness of calculating an off-line policy, using a novel approach to extract transition probabilities from a stochastic model, that despite motion uncertainties is able to guide a robot from an origin state to a goal state.
- In Section 5.5 experimental results are presented where we studied the performance of the different policies while varying the temporal, and failure probability constraints.

5.2 Preliminary Background

5.2.1 Uncertainty Quantification via Probabilistically-Valid Stochastic Models

This section will summarize the work of Karydis et al. [85] to extract a stochastic model from experimental data.

Having a sequence of experimental data collected $\{w_1, ..., w_I\}, I \in \mathbb{N}$ indicating the position and orientation of a ground robot in a time T.

The model $\mathcal{M}$ is parameterized by $\lambda \in \mathbb{N}$ parameters which are collected in a vector $\xi \in \Xi \subset \mathbb{R}^{\lambda}$. Varying the parameter values we can generate a set of models $\{\mathcal{M}(\xi),\ \xi \in \Xi\}$, whose output is $\mathsf{out}(\mathcal{M}(\xi))$.

The goal is to find the parameters $\bar{\xi} \in \Xi$ that enable the model $\mathcal{M}$ to replicate outputs as close as possible from the experimental data. The closeness will be measured using an appropriate distance metric to the sample mean w^{ave}. Figure 5.1 shows the general idea to fit the experimental data to a cone with some variability ellipses ϵ_t.

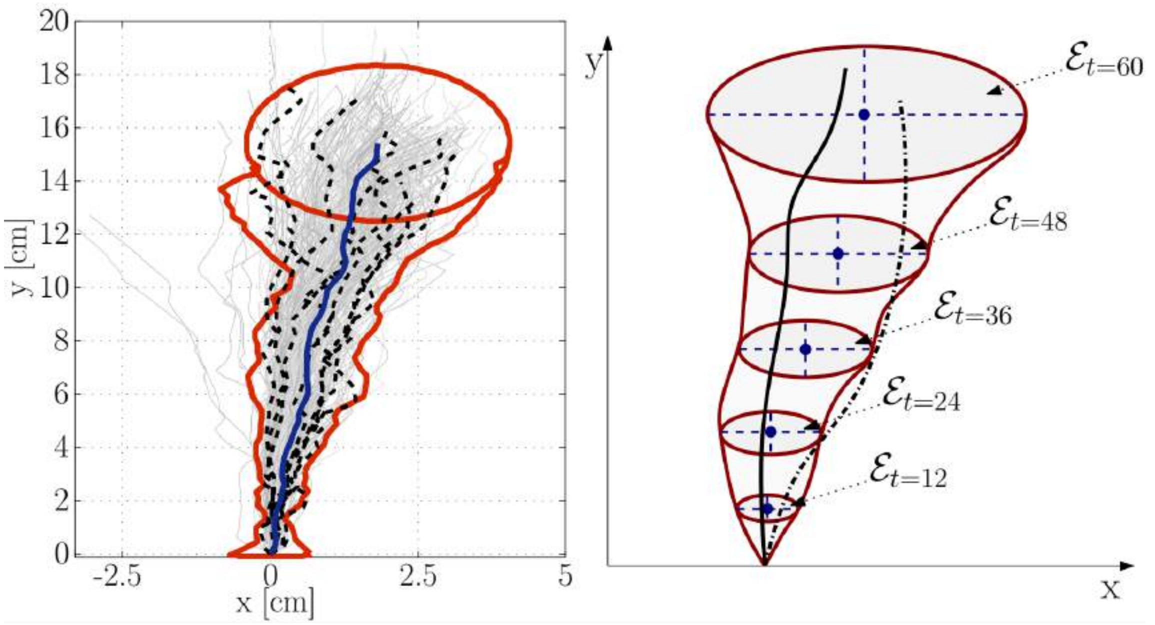


Figure 5.1: Example of data fitting: a) Grey lines correspond to the empirical data. Dashed lines correspond to sample trajectories. Blue line correspond to the mean of the sample trajectories. Red lines correspond to the cone of data. b) Computing the cone of data with variability ellipses ϵ_t centered at the sample mean. Image taken from [85]

To solve this problem, taking into account that we have time series of finite length T, we define a least squares optimization:

$$\bar{\xi} = \arg\min_{\xi \in \Xi} \sum_{t=1}^{T} \|\mathsf{out}(\mathcal{M}(\xi))_t - w^{\mathrm{ave}}(t)\|^2 \ , \tag{5.1}$$

where $||\cdot||$ is the Euclidean norm, and $t \in \{1, \ldots, T\}$. Having identified the model instance with the particular tuple $\bar{\xi}$ of parameter values, the next step is to capture the data variability. This can be done treating the model's parameters as random vectors, $\tilde{\xi}$, instead of tuples of constants. This way, the tuple $\bar{\xi}$ can populate the mean of that random vector, while its higher-order statistics are estimated.

At the same time, it is crucial to define a function that decides if the proposed model actually fits the experimental data or realization of $\tilde{\xi}$. This realization of $\tilde{\xi}$ is denoted by $\xi_n, n = \{1, \ldots, N\}, N \in \mathbb{N}$. N. This function creates a cone of data $\mathsf{cone}_{p,\gamma}(\mathbf{w}_m)$ for a multi-sample $\mathbf{w}_m, m \in \{1, \ldots, M\}$, with $M \in \mathbb{N}$:

$$g(\xi_n, \mathbf{w}_m) := \begin{cases} 0, & \text{if } \mathsf{out}(\mathcal{M}(\xi_n)) \in \mathsf{cone}_{p,\gamma}(\mathbf{w}_m) \\ 1, & \text{otherwise} \end{cases} . \tag{5.2}$$

If the the output of the model is out of the cone, it means that the value does not represent similarity with the experimental data, and therefore helps us to define a probability of violation which is the basis for judging the validity of a (stochastic) model. This probability of violation for each ξ_n, $n \in \{1, \ldots, N\}$ parameter multi-sample is:

$$\hat{P}(\xi_n; \mathbf{W}_M) := \frac{1}{M} \sum_{m=1}^{M} g(\xi_n, \mathbf{w}_m) \ . \tag{5.3}$$

The confidence $1 - \delta$ is defined as:

$$\hat{P}_0 = \max_{n \in \{1,..,N\}} \hat{P}(\xi_n; \mathbf{W}_M) \tag{5.4}$$

and explains the probability of approximating the near maximum to accuracy ε and level α [92, 176, 177].

Having the validity of a model we can calculate the probability distribution of a random vector $\bar{\xi}$. The goal is to maximize the high-order moments to get $\hat{P}_0$ in (5.4) lower than a fixed threshold $\rho \in [0, 1)$. This is done maximizing $\{\sigma_1, .., \sigma_\lambda\}$ such that $\hat{P}_0 \le \rho$ from the covariance matrix $\mathrm{Cov}(\tilde{\xi}, \tilde{\xi}) = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_\lambda^2)$.

The algorithm presented in [85] shows clearly the method to reach a fitting stochastic model following five general steps:

- Select values for the model like a tolerance interval, a confidence value, etc.;
- Record data M and calculate $\mathsf{cone}_{p,\gamma}(\mathbf{w}_m)$ for each multisample;
- Generate the vector $\bar{\xi}$;
- Set values for $\{\sigma_1, .., \sigma_\lambda\}$ and calculate $\hat{P}_0$;
- Vary values of $\{\sigma_1, .., \sigma_\lambda\}$ until $\hat{P}_0 > \rho$.

5.2.2 Stochastic Motion Primitives Derivation

As we discussed in chapter 3, MDPs and CMDPs rely on the fact there is a set P that describes the probabilities of getting to a state x' when executing action a from state x. The assumption that we always have these values, and they are constant over the time and conditions, is very strong.

The purpose of this section is to illustrate a method to extract the transitions probabilities from a set of trajectories showing the robot moving for specific actions or maneuvers. The probabilities-extraction method that we use is a count-based system, where we want to calculate the value that describes the likelihood that the robot ends at one specific pose (position and orientation) after executing a certain action a. In order to obtain this value, we established a protocol and developed a software architecture to record in real time the different trajectories that the robot follows after receiving the command to use a maneuver.

To localize the robot in its operating environment and ensure that the state is observable as required by the CMDP model, we rely on a Vicon motion capture system installed around the working area. The Vicon system can track the robot's pose with sub-centimeter accuracy at a 100 Hz frequency, thus accurately solving the localization problem and providing the three dimensional pose (x, y, ϑ). Desired motion commands (linear and angular velocities) were transmitted wirelessly from an offboard computer workstation.

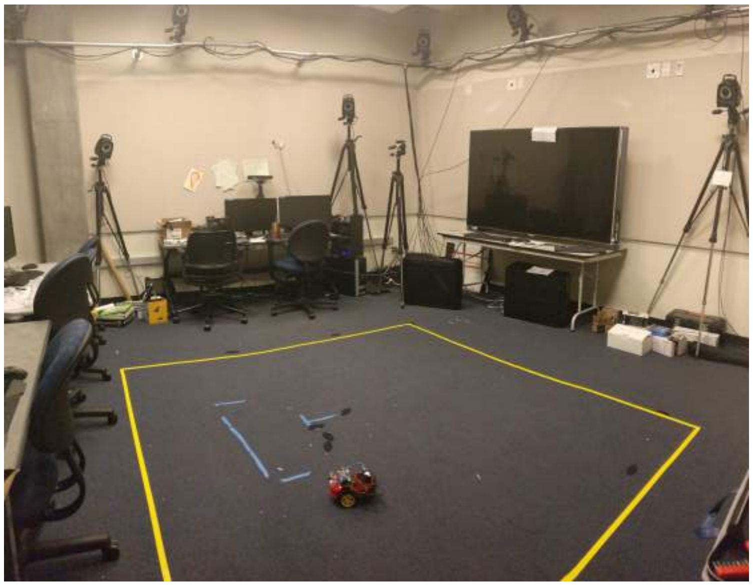


Figure 5.2: The environment with the Vicon system composed for motion capture cameras around the room.

The Robot

The differential-drive robot we used is the *Duckiebot* (see Figure 5.3), an open-source platform developed at MIT as part of an autonomy class [134].[1] The robot features two DC stepper motors, and it is equipped with a RaspberryPi 2 board (4-core 900 MHz ARM processor, 1 GB RAM) that runs ROS, a camera and a USB WiFi module for communications[2]. It is powered through a portable 10400 mAh battery that offers over two hours of continuous operation. The robot is not equipped with encoders or other sensors to measure the rotation and speed of the wheels. Consequently, its motion is rather inaccurate and noisy.

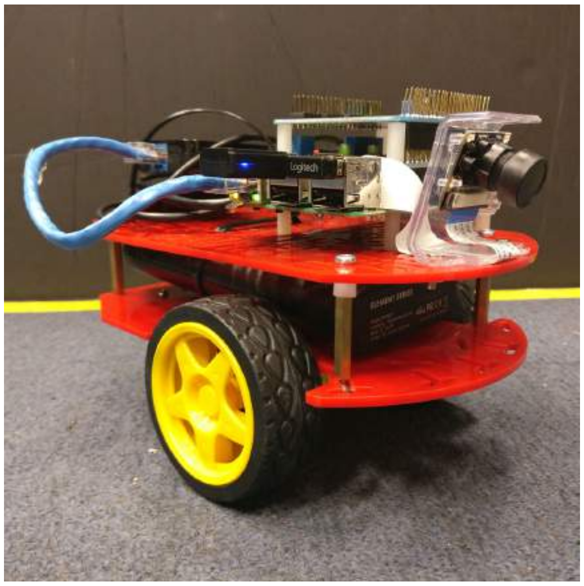


Figure 5.3: The Duckiebot robot used in this study.

Types of Primitives

In this work we use six primitives.[3] The selected primitives are parameterized based on curvature (forward motion, and counterclockwise and clockwise turns) and speed (fast, slow). In the following they are indicated as FF/FS (forward fast/slow), $CCWF/CCWS$ (counterclockwise fast/slow), and CWF/CWS (clockwise fast/slow). The Duckie's firmware accepts linear and angular velocity commands, and the primitives are then obtained by, accordingly, setting angular and linear velocities for a fixed amount of time. The six maneuvers have different durations,

[1]See `https://www.duckietown.org` for details.

[2]The robot is also equipped with a camera that is, however, not used in our setup.

[3]Similar to all works that consider a motion primitives based approach, the number of primitives is a user-selected hyper parameter that tradeoffs complexity and descriptive capacity of the planner. The choice to use the six primitives herein was made empirically, after initial experimentation so as to strike a balance between algorithm complexity, size of experimental datasets, and algorithm performance.

ranging from 0.965s to 1.235s (specific values were determined empirically)[4]. The reason for the different times to execute the maneuvers is because we attempted to generate symmetrical movements when turning left and right, but due to the robot's inconsistency, each maneuver still had some disparity when compared with its counterpart. Therefore, times were tuned and altered to generate a more symmetric set of training data. These primitives enable us to span both spatial and temporal components of the deployment problem at hand, i.e. they enable us to compute policies reaching different parts of the workspace under different temporal constraints.

5.2.3 Extracting the Transition Probabilities

The first step to model the primitives is to gather experimental data samples for each of them. To this end, for each individual primitive, we take a certain amount of sample trajectories. For each trajectory we recorded the three-dimensional pose data (x, y, θ) using the Vicon motion capture system operating at 100 Hz.

After each trajectory was recorded, staring at the position (0,0), the robot was manually placed back to a designated start area. The distribution of the endpoints of the recorded trajectories are shown in Figure 5.4. The plot displays endpoint orientations with arrows. Position uncertainty ellipses using three standard-deviation intervals along x and y are overlaid to the points.

While it is possible to augment the uncertainty ellipses to include the orientation as well, we choose to work here with position uncertainty only.[5] Doing so simplifies the problem in the sense that one needs to specify only desired position waypoints (possibly accompanied with acceptable uncertainty regions as in Figure 5.4), while the feedback nature of our planning approach is able to handle orientation discrepancies. The ability to handle this type of planning problem is useful for rapid deployment in practical, op-tempo cases (e.g., by specifying waypoints on a hand-held device).

[4]Linear velocity to 0.4 and 0.8 (on an 0 to 1 scale), respectively. The angular velocity is regulated by setting two control gains and they are -5.2 for right fast (CWF), 8.2 for left fast (CCWF), -5.4 for right slow (CWS), 6.0 for left slow (CCWS), 0.3 for forward fast (SLF), and 0.6 forward slow (SLS). The duration of all types of primitives are: 1.235 sec. for RF, 1.18 sec. for LF, 1.15 sec. for RS, 1.16 for LS, 1.325 sec. for FF and 0.965 sec. for FS.

[5]If we include uncertainty in the orientation we end up with a more complex distribution that looks like a crescent when sketched on the $x - y$ plane. Such a distribution is called "banana" distribution, and it has been shown that it is essentially a Gaussian distribution in exponential coordinates [104].

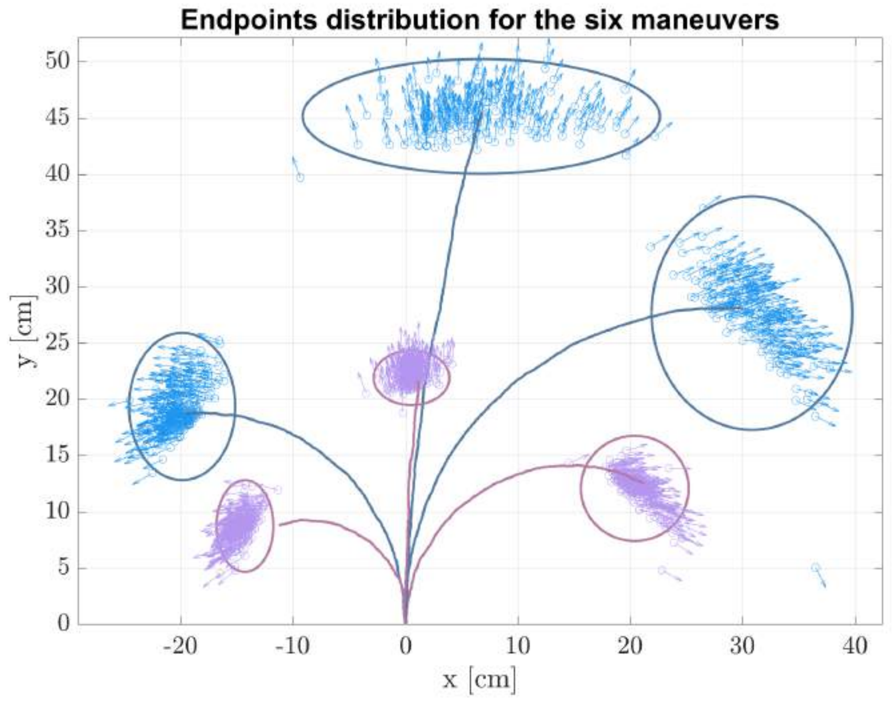


Figure 5.4: Experimental distribution of the trajectory endpoints (blue for fast maneuvers, and purple for slow maneuvers). Fast maneuvers (plotted in blue color) are those that end at larger displacements along the y axis. The final orientation is indicated by the arrow. Position uncertainty ellipses are shown in thick curves.

Data shown in Figure 5.4 give us data-driven motion primitives for the robot. The data-driven primitives can be directly connected to our framework developed in chapter 5. However, such primitives cannot generalize well to predict other possible maneuvers [86, 46]. To do so, we need a model reinforced with data that can help make trustworthy predictions. This is essential for our planning framework and is what we will cover in the following section 5.2.4.

It is important now to define the discretization for the position and the orientation. The environment will be divived in cells of a certain size, and for each of these cells, there will be a number of possible orientations of our robot. We divide the whole circumference in a fixed number of degrees, see figure 5.5. The size of the cells and the number of discretization for the orientation, called *ranges*, will play a main role when trying to calculate a policy: Bigger cells means a poor description of the environment, a smaller cell will correspond to higher resolution and precision of the actual position of the robot. However, the downside of reducing the size of the cells is the exponential growth of the space state X, which, in turn, produces exponential increases of the time to solve the CMDP and obtain a policy. The state for our CMDP will be the defined as (x,y,θ), where (x,y) will correspond to a specific cell on the map.

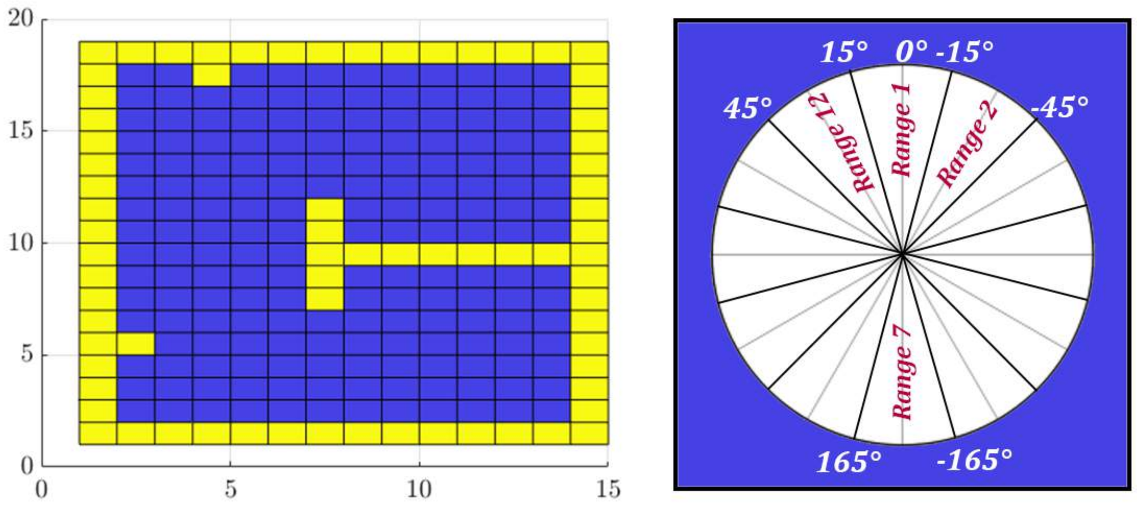


Figure 5.5: Left: Position discretization. Right: Orientation Discretization.

We always set *range* 1 with 0 degrees as the center of it. The *range* defines an interval of degrees/radians (arc). For our case the *range* 1 means having the robot pointing up, which is from 15 to -15 degrees clockwise. *Range* 2 goes from -15 to -45, and so on. The *range* 7 goes from -165 to 165 which means pointing down, then we continue *range* 8 from 165 to 135, until the start again at 15 degrees with *range* 1. After this intervals' distribution, the circle gets divided in two, being the right side for negative angles and left side the positive. The number of intervals can vary and depends on the wished precision. For this case, as an example, we took intervals of 30 degrees which means twelve intervals.

Having the recorded trajectories, and defining the pose discretization, we now proceed to numerically calculate the values of the transition probabilities. Figure 5.6 shows on the left only three different maneuvers that the robot can execute: forward, turning left/right with curve. We can also see 50 trajectory samples taken directly from the robot for each maneuver. We will present the method using only these three maneuvers, to facilitate the understanding and help the visualization of the method, but in reality we will have six maneuvers as we previously mentioned.

We organize each set of trajectories from each maneuver as shown in Figure 5.6 on the right. For our example, we show how, from the maneuver $forward$, we can see 4 trajectories that ended at the same *range* 1 between 15 to -15 degrees. These 4 trajectories out of 50 means an overall probability of 0.08. This also means that the robot can end in 3 different states according to the 3 cells where they landed, and for each we will have different transition probabilities $P(x, a, x)$: 2 trajectories ended at the cell defined between 0cm to -20cm on x and 80cm to 100cm on y, 1 ended on the cell defined between 0cm to -20cm on x and 120cm to 140cm on y, and the

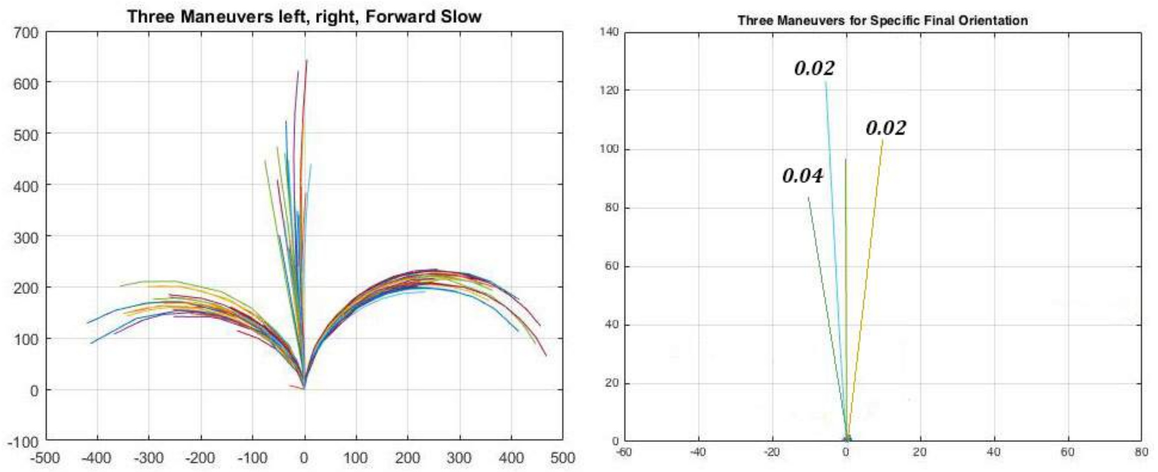


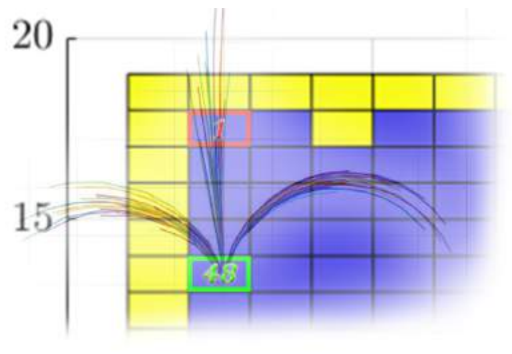
Figure 5.6: Left: 50 trajectories for 3 Maneuvers ("probabilitic template"). Right: Calculating probabilities for forward trajectories that end at *range* 1.

last one ended on the cell defined between 0cm to 20cm on x and 100cm to 120cm on y. These give us the transition probabilities of 0.04, 0.02 and 0.02, for the respective trajectories. We do the same counting for the rest of the 50 trajectories that end with different locations and orientations. At the end, we will be able to describe for the forward maneuver all the transition probabilities. We repeat the process for the other maneuvers and extract the different probabilities from the trajectories.

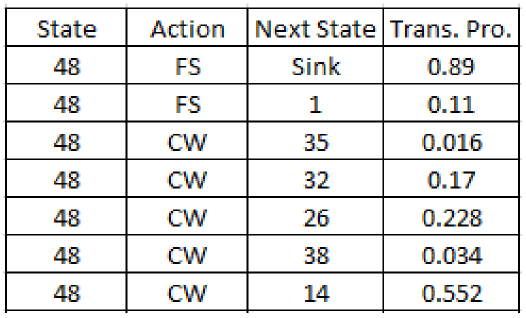
Now that we know how the robot behaves probabilistically after executing each maneuver or action, we can now use it for each of the cells in our map. We will overlap the "probabilistic template" over all the cells and calculate for each one, where the robot may end when executing an action. Figure 5.7 illustrates this process. On the left we see our map being overlapped with the 3 trajectories. As we explained, we used these trajectories to calculate the probabilities of ending in certain regions with certain orientations. For example, state 48 (hypothetical identifier number) will have different transitions and different next states, depending on the maneuver. On the right of the figure 5.6 we can see how when we execute the action forward (FS) from the state 48 we can end with a high probability at the sink state (wall of yellow cells). We can also end with a very small chance at the state 1, which is the cell on the corner of *range*1. We will complete the transitions table for all the cells, for all the maneuvers and for all the initial positions with all the 12 diferent initial orientations.

The transition table is going to be used as an input for our CMDP solver together with the costs of each of the transitions. Normally, in terms of time, there is a high cost for slow actions and low cost for fast actions. In terms of risk, the transition probability will define how risky each transition in the table is. For our example, executing FS from state 48 has a high risk of collision, so we can associate a cost of 1-Transition Probability, which is the by definition of the failure probability.

It is a critical task to extract the transition probabilities in order to solve



(a)



State	Action	Next State	Trans. Pro.
48	FS	Sink	0.89
48	FS	1	0.11
48	CW	35	0.016
48	CW	32	0.17
48	CW	26	0.228
48	CW	38	0.034
48	CW	14	0.552

(b)

Figure 5.7: (a) Example of using the probabilistic template with the states on the map. (b) Example of a small extract from a transitions table. This table can contain thousands of transitions, depending of the discretization and the number of actions.

CMDPs, however, the method to obtain a good amount of data samples can be time-consuming. Because of this, we are proposing a new method to reduce the experimental part when extracting transition probabilities. We developed a model that can reproduce the variability observed in experimental data with a *provable degree of fidelity*. We also have the availability of stochastic motion primitives —to create the associated transition probabilities—that are guaranteed to be valid for a robotic system of interest. We can achieve the construction of a probabilistically-valid stochastic model, and we outline the basic steps below. The interested reader is referred to [86] and [83] for a more detailed presentation on the subject.

5.2.4 Deterministic Model Fitting

We begin by introducing models that capture Duckie behaviors in expectation. A relevant model is that of a differential-drive robot. Let R and L be the wheel radius and distance between the two wheels, respectively. If v_r and v_l are the right and left wheel angular velocities, the following differential-drive model can be derived [97]

$$\begin{aligned} \dot{x} &= \frac{R}{2}(v_r + v_l)\cos\theta \\ \dot{y} &= \frac{R}{2}(v_r + v_l)\sin\theta \\ \dot{\theta} &= \frac{R}{L}(v_r - v_l) \end{aligned} \tag{5.5}$$

or, equivalently,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2}\cos\theta & \frac{R}{2}\cos\theta \\ \frac{R}{2}\sin\theta & \frac{R}{2}\sin\theta \\ \frac{R}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} v_r \\ v_l \end{bmatrix} . \tag{5.6}$$

The next step is to identify nominal wheel angular velocities for each of the primitives at hand. To achieve this we use least-squares constrained optimization. The solver for ordinary differential equations ODE (5.6) is simulated using Matlab's ODE45 routine. For each primitive, the simulation time matches the duration of the experimental one. The cost function penalizes differences at the final pose between the experiment and the one predicted via simulation.

The next step is to capture the observed variability in experimental poses. This we cannot do with a deterministic model; a stochastic one is needed.

5.2.5 Stochastic Model Extension

Our approach to derive a stochastic model is based on the method developed by [86], and introduced in Section 5.2.1. To capture the variability observed in the experiments, we apply the stochastic model extension tools to the differential-drive model described above. The candidate stochastic extension we consider here features zero-mean Gaussian noise that perturbs the system's state, i.e.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2}\cos\theta & \frac{R}{2}\cos\theta \\ \frac{R}{2}\sin\theta & \frac{R}{2}\sin\theta \\ \frac{R}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} v_r \\ v_l \end{bmatrix} + \begin{bmatrix} n_x \\ n_y \\ n_\theta \end{bmatrix} , \tag{5.7}$$

with $n_x \sim \mathcal{N}(0, \sigma_1)$, $n_y \sim \mathcal{N}(0, \sigma_2)$ and $n_\theta \sim \mathcal{N}(0, \sigma_3)$. Through this approach, the goal is to identify σ_1, σ_2 and σ_3 so that the resulting stochastic model produces outputs that capture the variability experimentally observed in the final pose.

For each primitive we have $I = 200$ available training samples. We select $p = 0.95$, $\gamma = 0.95$, $\alpha = 0.32$, $\rho = 0.10$, $\delta = 0.30$, $\varepsilon = 0.33$, and set $K = 5$. With these values $N = 5$ and $M = 20$. We simulate the SDE (5.7) using the Euler-Maruyama method. The simulation time is the same as the duration of each experimentally-collected primitive. We use randomized optimization to increase σ_1, σ_2 and σ_3 until we exceed the selected threshold ρ on the (probably approximate near maximum) probability of violation.

Application of the stochastic model extension approach yields $\sigma_1 = 5.6$, $\sigma_2 = 6.0$, and $\sigma_3 = 0.087$ for all maneuvers. Then, we simulate the derived stochastic model (5.7) by adding random noise at the initial pose, see figure 5.8. This is fixed to plus or less than half of the size of cells and plus or less than half of the discretization angle. Specifically, we will have a shift of the origin +-11.5 and +-5 degrees CW/CCW. Noise is added to emulate the uncertainty about the initial location of the robot. Adding such initial pose perturbations has been proven effective in other works where adding noise to the system's dynamics improves the actual performance of the robot in the real world [136]. There is another way to avoid the use of a random noise. For this we need to record the trajectories, not from a fixed starting point in the middle of a cell, but positioning the robot anywhere within the cell. This way the stochastic model can directly extract the robot's behavior and recreate the variability of the outcomes due to the uncertainty of the initial location. Without taking into account this uncertainty, we will be expecting the robot to always move from the middle of a cell and always end at the middle of another cell, which is practically impossible, unless the cells are small enough to become points in the space. Finally, for the identification process to generate the stochastic model needs to generate a *probability of violation* (probEval) of approximately 90%, which means that the modeled trajectories will be 90% close to the input data.

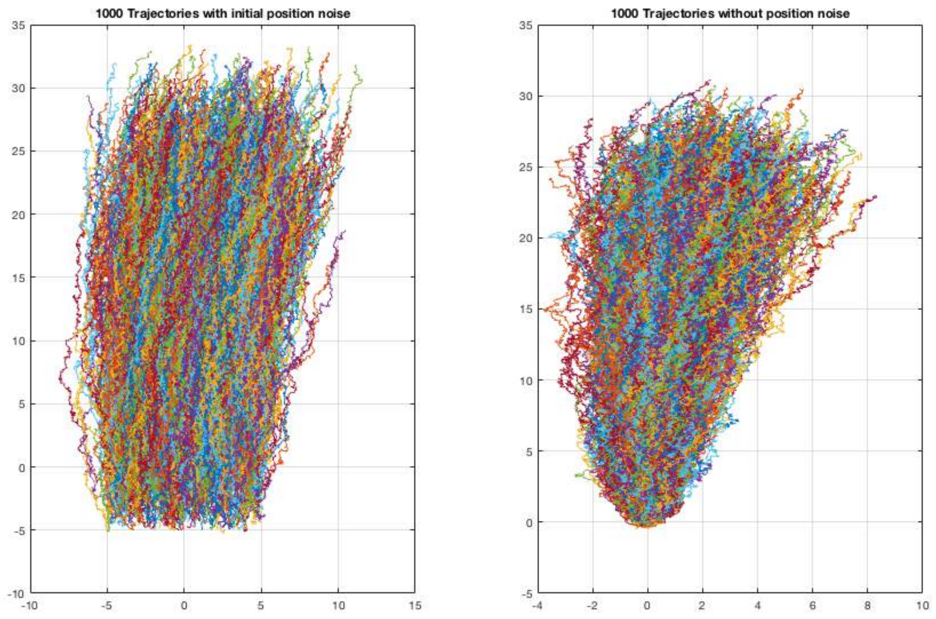


Figure 5.8: Example of the forward maneuver with and without initial noise for the position.

Figure 5.9 shows 500 realizations of each primitive derived with the stochastic model. The resulting first- and second-order statistics at the final pose capture well, the variability observed in experiments. For reference, we show the mean of the data as a thick line.

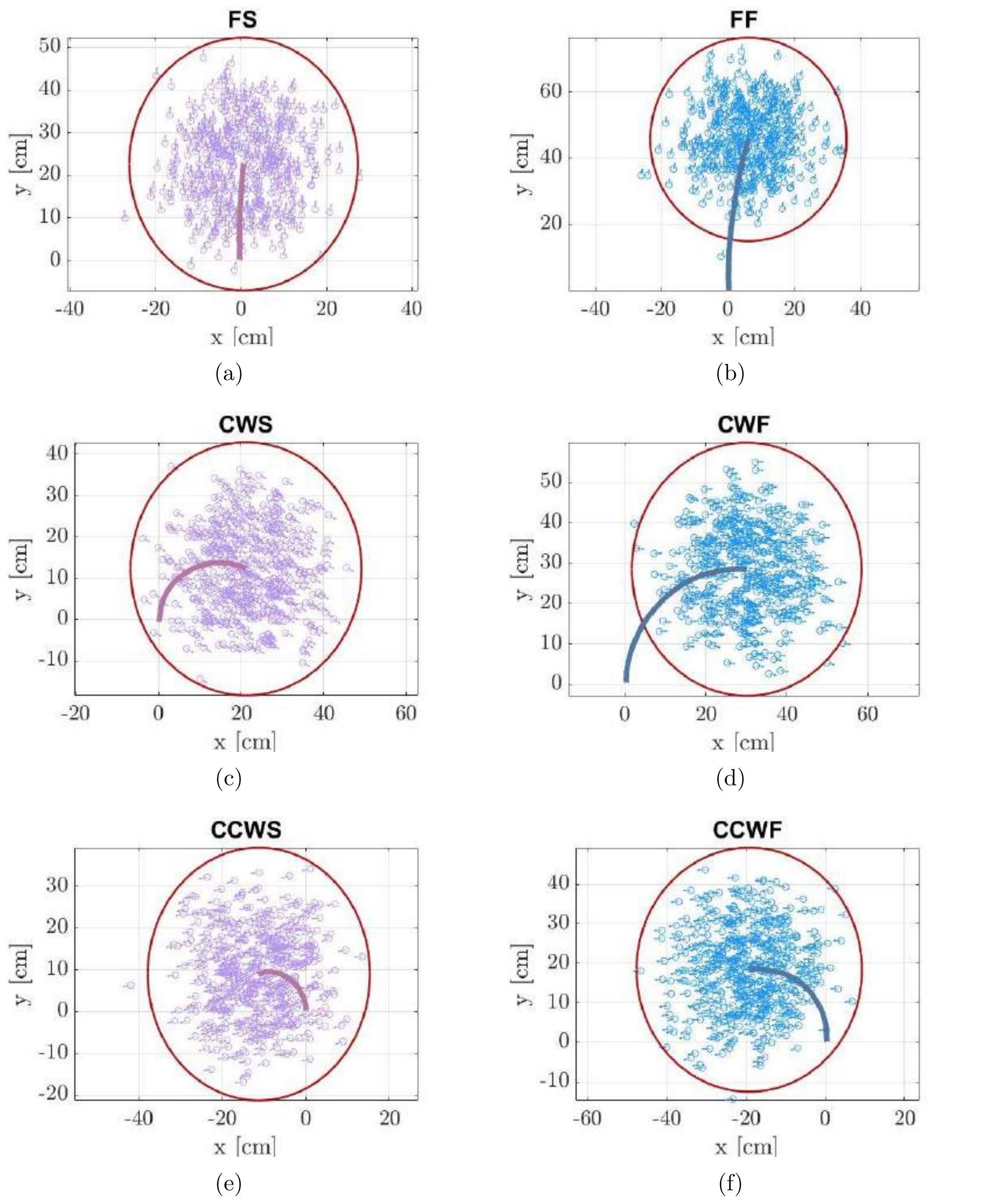


Figure 5.9: Realizations of the stochastic model: (a) FS, (b) FF, (c) CWS, (d) CWF, (e) CCWS, (f) CCWF.

5.3 Experimental Setup

In this chapter we will only focus on the experimental validation of the methods presented in 3.3 for CMDPs and 5.2.2 for generating an stochastic model from experimental data. We discuss how we implement an end-to-end planning system to determine a policy for a rapid deployment mission using a differential drive platform. We will make a description of the environment, the infracstructure we used, the results of all our experiments and finally some conclusions and future work that we believe will increase the understanding of this end-to-end planning system.

5.3.1 Environment

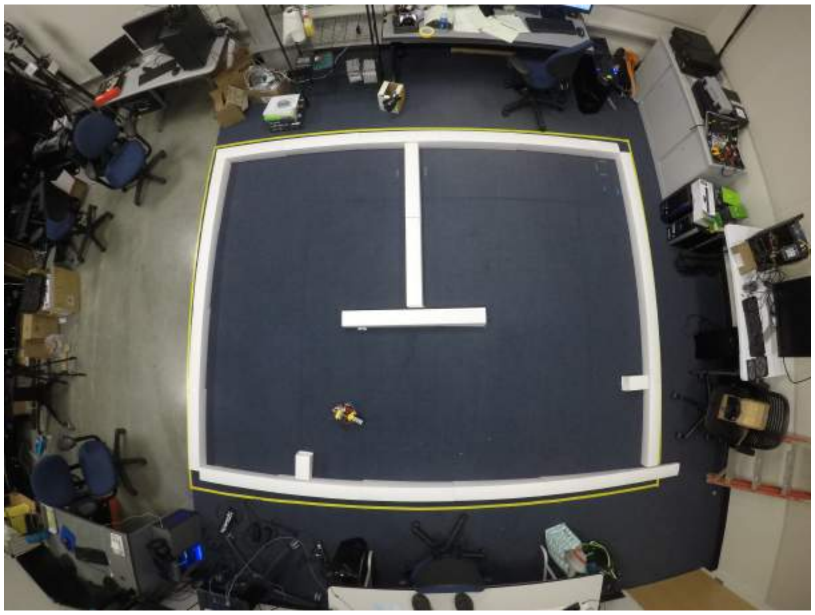
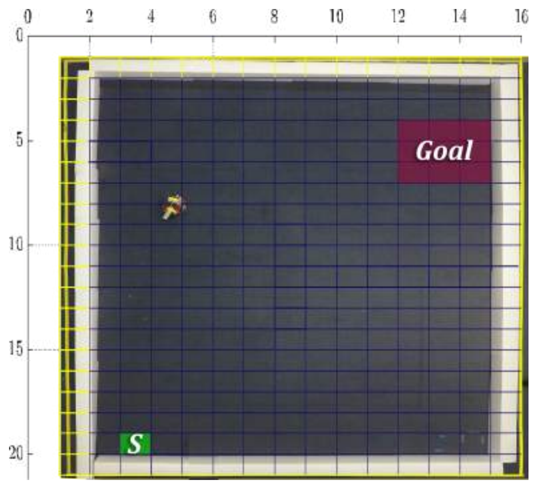
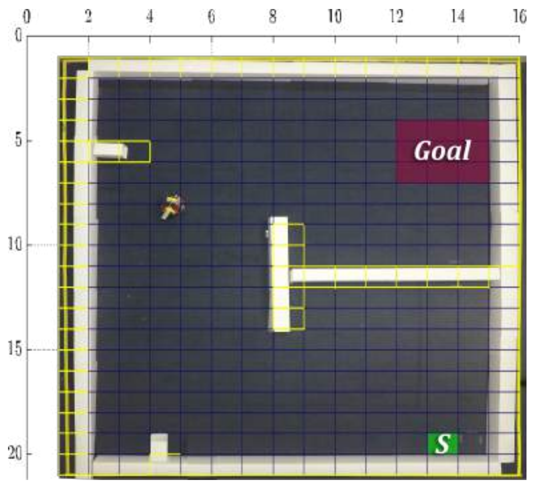


Figure 5.10: One of the real scenarios that we used for our experiments

We consider different planar environments similar to the maze shown in figure 5.10. Figures 5.11 and 5.12 show the four different environments used in the tests, and the overlaid grid the discretization used for the CMDP formulation. Starting from a preassigned known location, the objective is to reach a given goal region, i.e. a set of one or more target cells. The goal condition is satisfied as soon as the robot enters the goal region, irrespective of its orientation. The robot always starts from the locations marked with the letter ‘S’ facing upwards, and has to reach the squares marked in red. If the robot collides with any of the obstacles in the environment, the mission is considered a failure.



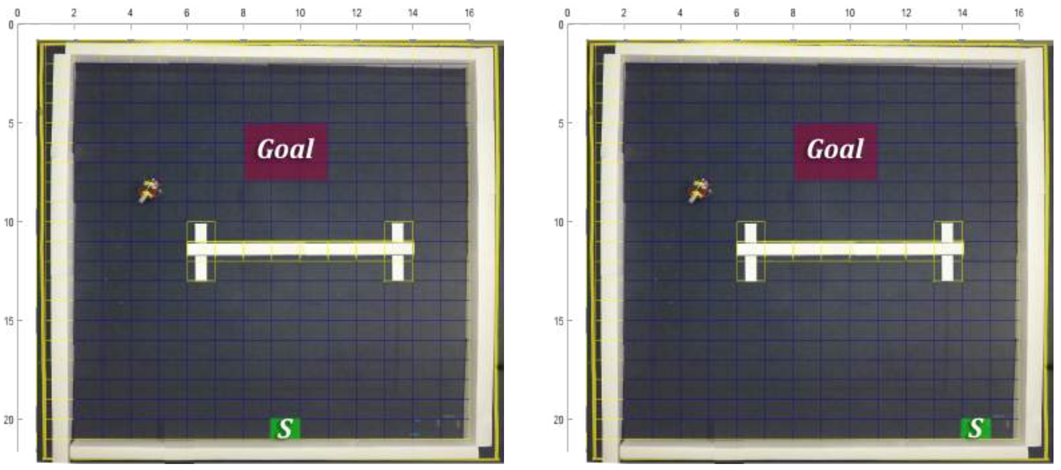
(a)



(b)

Figure 5.11: The two intial environments used for testing. The overlaid grid shows the discretization used for the CMDP. In the following, the environments are referred to as empty 5.11(a), maze 5.11(b).

In the first two maps, shown in figure 5.11, we see two different environments. The empty one was used to test the general performance of the policy without any obstacles and free path choice. Because the policy optimizes the solution to reach the goal, we expect to see that the robot does not go around using valuable time resources, but instead the policy should guide the robot to the shortest path possible with the lowest risk and minimum time. For the second map, the maze , we tried to force the robot to avoid obstacles and have sharp turns. We expect higher failure and slower run-times. Because of the difficulty of the maze, the policy needs to balance the risk with the time in order to reach the goal.



(a) (b)

Figure 5.12: The two following environments used for testing. The overlaid grid shows the discretization used for the CMDP. In the following the environments are referred to as two ways 5.12(a) and 5.12(b).

For the last two maps the goal was different. First, for the environment shown in figure 5.12(a) we wanted to study, with the same environment, what would be the policy when the goal was exactly on the opposite side of the map and where only two ways were possible; One with a very narrow corridor with a very high risk of collision, against a second way with a wider space to maneuver more freely and safely. Again, we were expecting that the policy was "aware" about the risk of the right way, and it should guide the robot to use the left way. For the second map shown in figure 5.12(b), we tried to force the robot to take the risky way, having a starting point closer to the right, the policy could estimate that the risk of taking the narrow passage was worth the shorter time to reach the goal.

5.3.2 Infrastructure

The whole infrastructure is shown in figure 5.13; For a given environment and a set of motion primitives, a policy is computed offline, solving the associated CMDP problem on a desktop computer (Purple box). This solver was implemented in C++ as a ROS node, but any other solver can be used (e.g. linprog from Matlab). This policy associates each state to a specific action/maneuver that will lead to the global target and will be used later for a different ROS node that will search in it. The robot is located at the starting state and a desktop computer queries the motion capture system (Vicon) to determine the pose of the robot, giving the three coordinates (x,y, θ). From this point the state x_0 is calculated and, with it, we can lookup into the policy the corresponding action $\pi(x)$. This action is then sent to the robot via WiFi, and the motion primitive associated with the selected action is then executed by the hardware onboard the robot. The behavior of each action will depend on the robot's kinematics and the parameters previously fixed. This process repeats until the robot reaches a state that is considered the goal or an obstacle, which will define the success or failure of the mission.

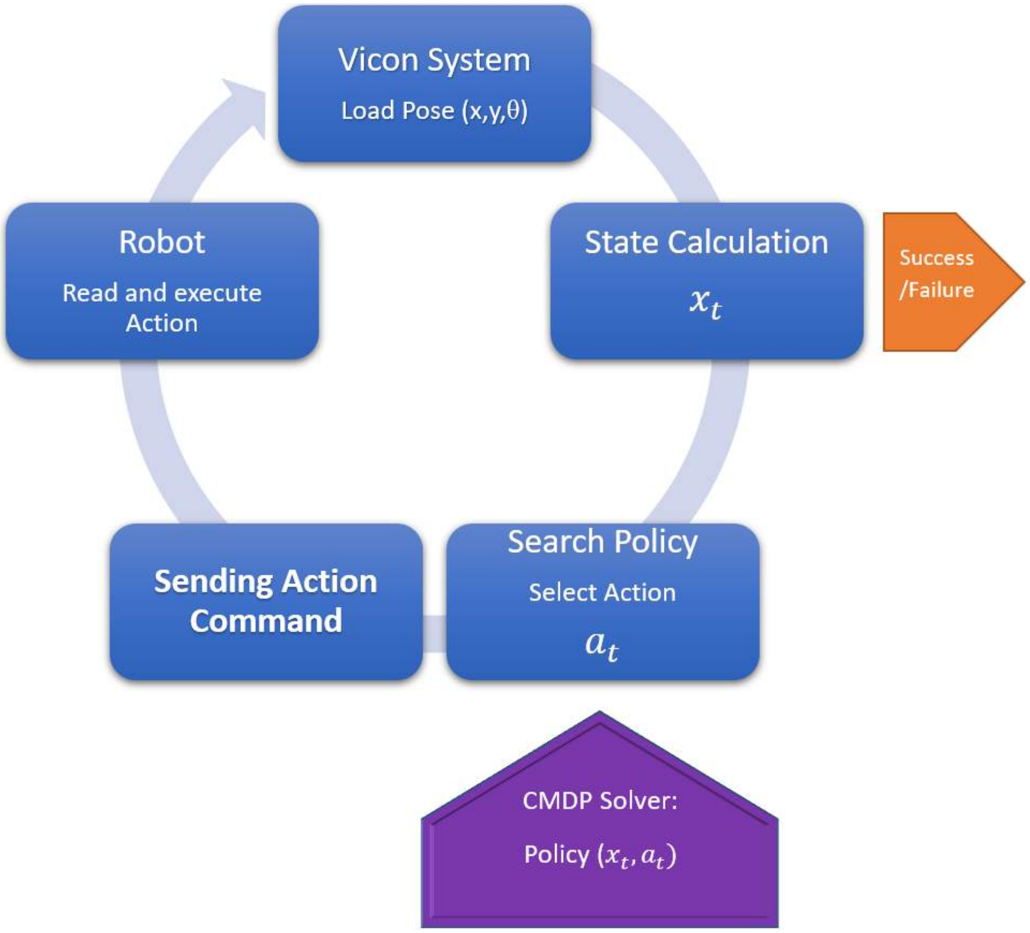


Figure 5.13: Infrastructure for Robot Duck executing policy from CMDP.

For our experiments we sampled 200 trajectories that were used to extract the transition probabilities. (See Section 5.4 below for details on how the CMDP parameters are extracted.). For each trajectory we recorded the three-dimensional pose data (x, y, θ). In principle, all run time computation could take place on the Duckiebot. Our design choices are solely driven by technical convenience and are inconsequential to the validation of the algorithm.

5.4 CMDP definition

As per the definitions given in Section 3.3, various parameters are needed to complete the specification of a CMDP. The finite state space X is given by the discretized robot pose (x, y, θ). To this end, we discretize the environments shown in Figure 5.11 and

Figure 5.12 in squares of 23×23 cm, thus obtaining grids with $16 \times 21 = 336$ cells for the pose. The orientation is discretized into sectors 10 degree wide. Therefore, we will have 36 different values for the orientation. Each of the maps displayed in Fig. 5.11 and Fig. 5.12 has between 9,000 and 10,000 states, that is, less than $16 \times 21 \times 36$ because states inaccessible due to obstacles are removed. All of these states belong to the set of transient states X'. The choice of the discretization angle and the cell size is based on some preliminary tests aiming at determining the appropriate tradeoff between accuracy and speed, as the number of states is directly related to the number of optimization variables in the linear program to solve the CMDP problem.

To solve the rapid deployment problem, two additional states $\mathcal{S}$ and $\mathcal{T}$ are introduced, similarly to [41]. State $\mathcal{S}$ is a sink state representing the failure event, whereby the robot does not accomplish its assigned task because it collides with an obstacle. State $\mathcal{T}$ is a terminal state that is entered when the mission terminates, either with success or failure. The sink state $\mathcal{S}$ belongs to the state of transient states X', whereas the set M includes just $\mathcal{T}$ (see Figure 5.14).

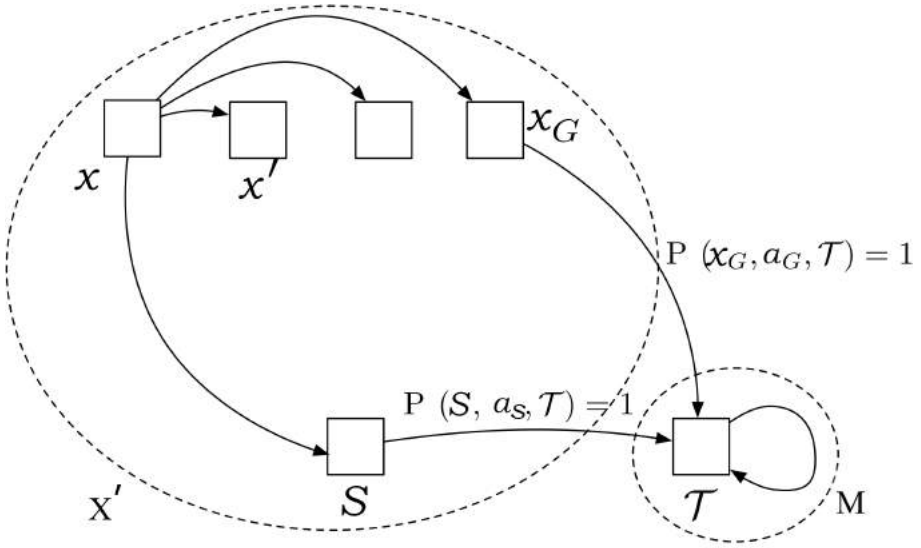

x
x'
x_G
$P\ (x_G, a_G, \mathcal{T}) = 1$
$P\ (S,\ a_S, \mathcal{T}) = 1$
X'
S
$\mathcal{T}$
M

Figure 5.14: Structure of the CMDP for the deployment problem. Top states in X' correspond to the states obtained by discretizing the environment and with transition probabilities numerically determined as described in the following. State $\mathcal{S}$ is entered when there is a collision. State $\mathcal{T}$ is the only state in M and is always entered at the end of a run, either successfully ending in a goal state x_G or being unsuccessful because of a collision.

The average number of state-action pairs is 55,837, yielding, on average, 714,729 possible state/action/state transitions with strictly positive probability. For the maze shown in figure 5.11(b) the number is 9,002 52381 for state-action pairs and 714729 transitions. For the map represented in figure 5.12(a) the number of states is 9146 and has 53245 state-action pairs and 714729 transitions. The number of state-action pairs is equal to the number of variables ρ in the constrained linear system defined in Theorem 3.1.

After defining the states, it is necessary to define the set of actions. For each state $x \in X$ different form $\mathcal{S}$ and $\mathcal{T}$ the action set $A(x)$ includes the six primitives we introduced in Section 5.2.2. For each couple of states x and x', and primitive a, the transition probability $P(x, a, x')$ is determined using a sampling based approach leveraging the generative stochastic models we identified.

For given starting state x and action a we generate 500 trajectories using the generative model associated with the stochastic primitive defined by a. Each sample represents a trajectory obtained following the primitive starting from a point inside x chosen using a uniform distribution. Using the geometrical model of the robot and of the environment, each trajectory is then checked for collisions, similarly to what is done in sampling based path planners such as PRMs and RRTs [88, 98].

If the trajectory is collision-free, then the end point of the trajectory identifies a state x' and this event is recorded to numerically estimate $P(x, a, x')$. If instead the trajectory generates a collision, then this event contributes to the count to determine the probability $P(x, a, \mathcal{S})$ into the sink state(see figure 5.15). For the sink state $\mathcal{S}$ we define a single action $a_{\mathcal{S}}$ with $P(\mathcal{S}, a_{\mathcal{S}}, \mathcal{T}) = 1$, i.e., once the robot enters the sink state $\mathcal{S}$, at the next time step, it will deterministically move to the terminal state $\mathcal{T}$. Similarly, for the goal state(s) x_G we also define just a single action a_G with $P(x_G, a_G, \mathcal{T}) = 1$. These choices are consistent with the former work [35] and allow to accurately estimate the probability of successfully completing a mission. To be precise (see Theorem 1 in [35]), $\rho(\mathcal{S}, a_{\mathcal{S}})$ is the failure probability, i.e. the probability of not completing the task because of a collision with the environment.

To complete the CMDP definition, the remaining elements to introduce are the costs c and c_i and the initial probability distribution β. The primary cost c is defined as $c(x_G, a_G) = 1$ and $c(x, a) = 0$ for all other state/action couples. We consider two additional costs c_1 and c_2. Cost $c_1(x, a)$ is the time of executing primitive a and is determined according to their speed (slow vs. fast). Note that with c_1 defined as the time to execute the maneuver, we are then in the hypotheses of Lemma 5.1, i.e. there is a cost that is strictly positive for each $(x, a) \in K$. Regarding the second cost, $c_2(\mathcal{S}, a_{\mathcal{S}}) = 1$ and $c_2(x, a) = 0$ for all other state/action couples. As shown below, this allows us to formulate the problem with a bound on the failure probability. Finally, since we assumed the robot always starts at a preassigned location and orientation, $\beta(x) = 1$ for the unique initial state and $\beta(x') = 0$ for all other states.

Lemma 5.1. *Let $\mathcal{C}$ be an instance of the CMDP planning problem in which at least one of the c_i costs is strictly positive in X', i.e., for at least one cost $c_i(x, a) > 0$ for each $(x, a) \in K$ with $x \in X'$. Then, every optimal policy for the CMDP planning problem must be transient.*

Proof. Without loss of generality let c_1 be the strictly positive cost. Assume there exist an optimal policy π^* that is not transient. This means that the state remains indefinitely in X' and, consequently, the constraint $c_1(\pi, \beta) \leq B_1$ in the linear program introduced in Theorem 3.1 cannot be satisfied. Therefore π^* cannot be an optimal policy because it does not solve the linear program. □

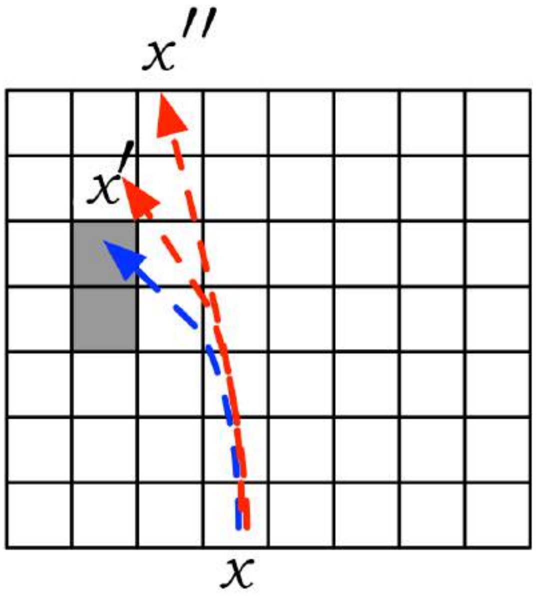


Figure 5.15: For a given state x and maneuver a, multiple trajectories are generated using the generative model. Each trajectory is then checked to see if it is collision free or not. Collision free trajectories, like the red ones, are used to determine the transition probabilities to states like x', x'' and like. Trajectories resulting in a collision, like the blue one, are used to numerically estimate $P(x, a, \mathcal{S})$, i.e., the transition probability into the sink state $\mathcal{S}$.

According to this model, the generic linear program given in Theorem 3.1 takes now takes the form

$$
\begin{aligned}
\max_{\rho(x,a)} & \sum_{(x,a)\in K'} \rho(x,a)c(x,a) \\
\text{s.t.} \quad & \sum_{(x,a)\in K'} \rho(x,a)c_1(x,a) \leq T_f \\
& \sum_{(x,a)\in K'} \rho(x,a)c_2(x,a) \leq P_f \\
& \sum_{y\in X'} \sum_{u\in U(y)} \rho(x,a)(\delta_x(y) - \Pr(y,a,x)) = \beta(x) \\
& \forall x \in X' \\
& \rho(x,a) \geq 0 \quad \forall \rho(x,a) \in K',
\end{aligned}
$$

where T_f is the temporal deadline to complete the task, and P_f is the bound on the failure probability. To see that $\sum_{(x,a)\in K'} \rho(x,a)c_2(x,a)$ is, indeed, the failure probability, it suffices to observe that the state enters $\mathcal{S}$ exactly once, i.e. when it collides with an obstacle, and then, it deterministically moves to $\mathcal{T}$. Therefore, $\sum_{(x,a)\in K'} \rho(x,a)c_2(x,a)$ is indeed the failure probability. The reader is referred to [35] for a more detailed discussion of this derivation.

map	P_f	T_f	Failures	Time (sec)
Empty b	5%	50	4%	30.8542
Empty b1	10%	50	6%	32.7447
Maze a	5%	50	18%	43.2439
Maze a1	10%	50	16%	46.9762
Maze a2	10%	250	10%	43.7111
Two ways c	5%	50	4%	32.4043
Two ways d	5%	50	2%	46.8571

Table 5.1: Average performance over 50 runs.

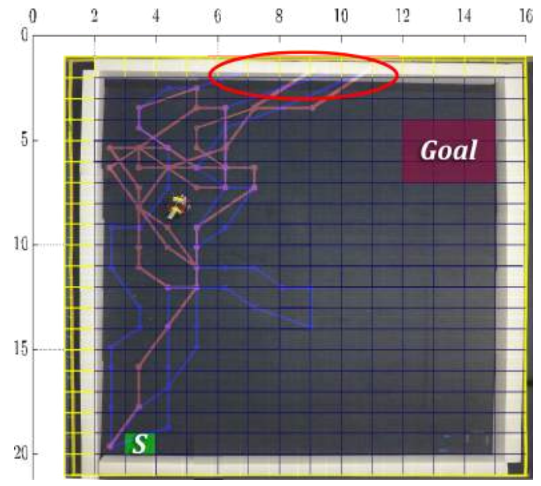
5.5 Experimental Validation

In this section we present various experiments to validate theoretical findings of this chapter. In the first set of experiments we consider the DuckieBot operating in the four environments shown in Fig. 5.11 and 5.12. We considered the four maps discussed earlier with different starting and goal locations, as well as different bounds T_f and P_f for the time to complete the task and the success probability. Table 5.1 shows the results. The parameters used to obtain the current results are not unique and different combinations of values can reach the same or better performance. The study of how to determine and combine these values efficiently will be out of the context of this paper and will be the goal of future research.

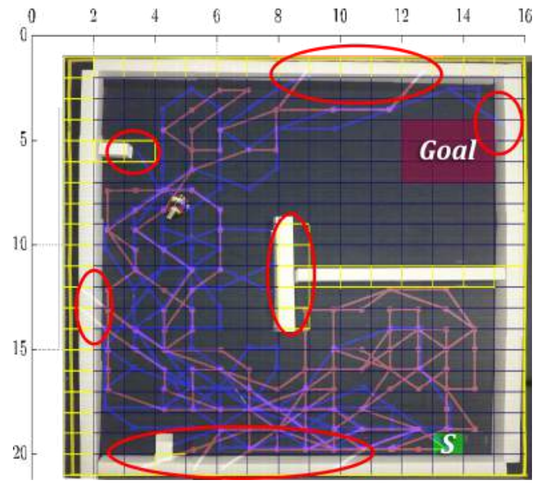
We can see that in all scenarios there is good agreement between the theoretical bounds for the failure probability, the time to completion, and the values observed experimentally. The only outliers are the cases with the maze environment shown in Fig. 5.11(b). In these cases the complexity of the environment makes it hard to experimentally match the expected failure probability. The reason for this discrepancy is due to the fact that in some cases, after executing a maneuver, the robot reaches a state that, according to the model, had a 0 probability to be reached. Such a problem could be mitigated by increasing the number of trajectories generated to assess the transition probability.

To better understand the source of failures, Fig. 5.16 and Fig. 5.17 plots the trajectories that resulted in a collision with the environment. These figures emphasize the regions where the robot hit the obstacles during the experiments with red circles. In all cases, this can be explained from the shape of the maneuvers used to move the robot, i.e. maneuvers of the type CWF or CCWF. The policy tries to reach the goal using fast maneuvers instead of slow and this causes the robot to hit the walls when trying to turn on the top of the map in figure 5.16(a). For the second map, the maze, the robot has more difficulties to reach the goal there are more collision areas, again, mainly when turning.

This problem can be countered by increasing the temporal deadline and allowing the robot to move using slower maneuvers. Indeed, as shown in Table 5.1 for the maze a2 with a higher temporal deadline, we achieve, exactly, the expected P_f. We can also observe that because the robot does not have the possibility to go perfectly straight, it tries to find the path where turning is more suitable to arrive to the goal, but this causes collisions with the walls of the map and other obstacles.



(a)

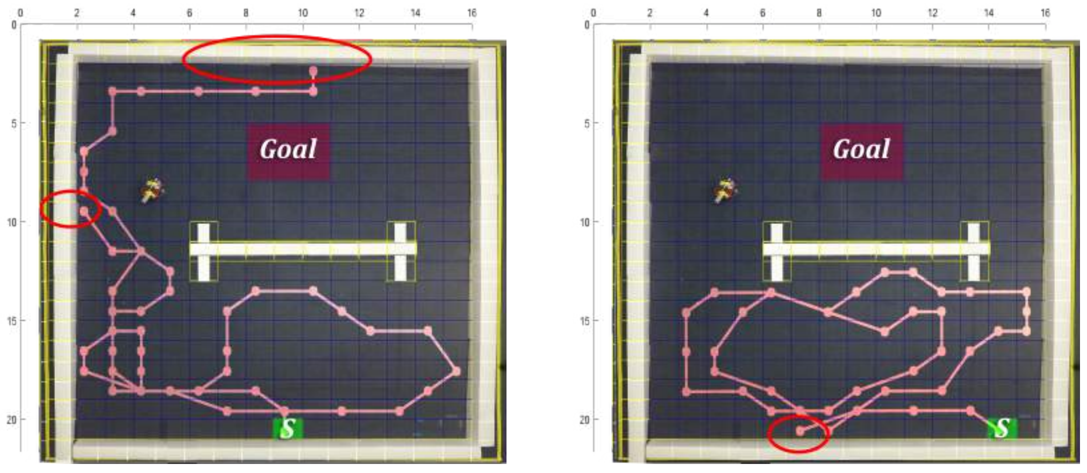


(b)

Figure 5.16: Trajectories showing failures when going to the goal. Zones marked by a red ellipse show where the trajectories collided with the obstacles. (a) Red lines correspond to Maze a, and blue lines to Maze a1. (b) Red lines correspond to Empty map b and blue lines to Empty b1.

In Fig. 5.17(a) and 5.17(b) we see how the same environment with two different starting points and the same settings for T_f and P_f will lead to different failing trajectories. It is interesting to note that for both cases the success rate is higher than most of the other maps. Similarly, important is to see that, despite the possibility to take the shortcut to reach the goal in Fig. 5.12(b) the robot decides to always take the path that is safer with the wider corridor. Our expectations for these two maps were confirmed by the experiments, and the policy shows awareness of the risk of taking the narrowed path over the wide one.

All these results may vary if we change the shape of the maneuvers, for example, including a PID controller to have more precise control over the robot's movement. With this controller we will avoid the amount of discrepancies we see for the same maneuver under the same conditions. We did not include any controller, and instead we had an speed wheel-control open loop system just to show the robustness of the solution and the strength of a policy that used the stochastic primitives from the stochastic model of the robot. Despite having a very noisy set of maneuvers, the robot still manages to reach the goal and predict correctly for each state, the best possible action.



(a) (b)

Figure 5.17: Trajectories showing failures when going to the goal. Zones marked by a red ellipse show where the trajectories collided with the obstacles. (a) Two different failing trajectories. (b) One failing trajectory.

A second set of experiments aims at assessing the sensitivity to the number of samples used to derive the stochastic model (input), and the number of trajectories generated by the model (output). In particular, it is of interest to determine how these values influence the transition probabilities used in the CMDP. In the following, our reference set of transition probabilities are those obtained with 200 data samples (input) and 500 generated trajectories (output) because these are the values used in the experiment described above.

In Table 5.2 we show the results obtained varying the size of the input, while keeping the output size fixed at 500. In Table 5.3 we vary the size of the output while keeping the size of the input fixed at 200. The resulting transition probabilities are contrasted against the reference ones using the KL divergence to assess similarities and differences between the distributions.

We can observe that in terms of KL divergence the sensitivity is modest.

5.6 Conclusions

We presented an integrated method combining theory and practice to compute and execute motion policies by using motion primitives while being subject to multiple constraints. We established how a differential robot can be modeled stochastically, and the same model utilized to solve a multi-objective plan with constrained Markov Decision Processes. The calculated policies for multiple scenarios prove that the pipeline to estimate a functional policy with the real, low-cost robot, is possible and successfully achieved. Through experiments and data analysis, we studied the robustness of the solution and the general performance when varying different

map	input size	KL-div
Maze a	50	2.0630
Maze a	100	2.0851
Empty b	50	2.0548
Empty b	100	2.0797
Two ways c	50	2.0564
Two ways c	100	2.0805

Table 5.2: Comparison of transition probabilities P for different amount of data samples taken from the real robot and generating 500 modeled samples.

map	output	KL-div
Empty b	50	2.1792
Empty b	250	2.0820
Maze a	50	2.1717
Maze a	250	2.0883
Two ways c	50	2.1673
Two ways c	250	2.0855

Table 5.3: Comparison of P distribution for different amount of modeled samples taken from the stochastic model against the 500 modeled samples successfully used in table 5.1

parameters that include a temporal deadline, a failure probability, the number of samples used to generate the stochastic model and the number of realizations or simulated trajectories.

For a future work it, would be interesting to study how multiple parameters can vary and how these variations affect the final performance of the robot.

Chapter 6

Time-Constrained Exploration Using Toposemantic Spatial Models

6.1 Introduction

In this chapter we will go over one of our published papers called ”Time-Constrained Exploration Using Toposemantic Spatial Models: A reproducible approach to Measurable Robotics” [159]. In the previous chapters we studied the problem of rapid deployment, how a CMDP model can help to navigate a known environment, and how to empirically extract the transition probabilities to model the stochastic behavior of a robot. This chapter aims to expand the application of CMDPs for mobile robots, and specifically, it deals with the case of unknown environments. In the past, we assumed that we had an available map of the place we wanted to navigate. In this chapter we drop this assumption to explore and navigate a new unknown environment.

Exploration is a fundamental ability in mobile robotics and has been extensively studied in the last three decades. With the current explosive growth in robotics applications, interest in this area continues to grow, in particular by considering extensions and special cases not studied in the past. As robots become more integrated into our everyday activities in applications such as deliveries, home and hospital care, logistics, and mobility, new paradigms emerge by exploiting novel sensors and robot platforms. Consider a service robot that must deliver an item in an unknown environment, say an office floor it has never entered before, and for which it has no prior knowledge other than the fact that it is an office environment. For example, the robot may have to deliver an envelope in the copy room, or put a package on the desk in Mr. Chairman’s office. Not having preliminary knowledge about where these places are located, the robot should, therefore, explore the environment until it recognizes it has reached the desired place. At that point, it shall deposit the goods it is supposed to deliver, travel its way back to the entrance, and leave. Exploration is an integral component of this task, and if the robot is doing this job as part of a series of deliveries, it is important to complete the task quickly to increase the number of tasks completed in a set amount of time.

Exploration, in its most studied form, aims at building a spatial model of the environment it operates in. Central to the exploration task is the decision process determining "where to move next". A classic early solution to this problem is frontier-based exploration [182], an approach inherently tied to using occupancy grid maps to represent space. In this case, a frontier is defined as the boundary between explored and unexplored space, and the intuition is that by moving towards large frontiers, a robot will manage to complete its task more quickly by discovering more unknown areas. Other possible approaches are based on random exploration, or variations of the frontier based approach. For example, distance to frontiers may be considered to break ties between equally large frontiers. In these approaches the temporal dimension is not explicitly considered. That is to say, that while heuristics are introduced to expedite the exploration process, time is not an explicit metric or constraint.

In this work, we deviate from existing literature in robot exploration by considering three modifications to the basic setting.

- First, we assume the robot does not build a metric model of the environment, like an occupancy grid map, but rather a topological model with semantic annotations. This is called Oriented Topological Semantic Map (OTSM). The characteristics of these maps, how they are built, and used for navigation are presented in section 6.3. There, we formally defined OTSM and set the rules for creating them.

- Second, the objective of the robot is not to build a spatial model per se, but rather to explore the unknown environment until a target location is discovered. The model is functional to this objective, e.g., to avoid revisiting areas already explored. To this end, we assume that the target location is provided in a format compatible with the sensorial capabilities of the robot, so that it can detect when the desired place has been reached.

- Finally, we introduce a temporal constraint, i.e., a time T such that the exploration task is considered not solved if after time T the robot has still not reached the location it is looking for[1].

In this chapter we consider the topic of efficient exploration using Oriented Topological Semantic Maps (OTSM). Efficient, in this case, means that robots are expected to complete their assigned task within a given temporal deadline. To be specific, the goal is to deploy a robot in an unknown environment with the objective of reaching a target location by a given time. For this task to make sense, the robot

[1]One could object that the task we are describing is in fact search, because the robot is looking for a specific location. However, our problem is more akin to exploration because the robot builds a spatial model to avoid revisiting already visited areas and to leave the area once the task is completed. Ultimately, however, it is just a matter of agreeing on the terminology.

has to be able to recognize it has reached the desired place. For example, a delivery robot may be tasked with delivering, within two minutes, a package to the copy room of a previously unvisited office floor. The exploration task is unsuccessful if the robot either collides with the environment or does not reach the desired location by the assigned deadline. To accomplish this task, the robot does not build a metric map, but rather incrementally builds a topological model where the environment is represented by a graph.

We opt for topological maps for various reasons. First, there is evidence that humans make use of topological models for spatial awareness and navigation [93]. By developing robots using comparable models, there is the expectation that more intuitive human-robot interfaces will be possible. Moreover, as robots rely less and less on sensors providing accurate metric information (e.g., laser range finders), there is an interest in moving away from representations such as occupancy grid maps that are tightly integrated with range finders. Finally, topological maps are more compact and less memory intensive.

To introduce temporal deadlines in the exploration strategies, we rely on planning algorithms using CMPDs to consider multiple objectives at once. This approach allows to balance, in a principled way, the unavoidable trade-off between efficiency and safety, whereby a robot moving faster to quickly reach a location is also more likely to incur in motion inaccuracies and collisions. The planner produces a plan by composing maneuvers from a preassigned set of motion primitives characterized by different velocity of execution as well as different success probabilities. Available maneuvers are of the type, traverse corridor slowly, or enter door fast, and the like. In an effort to meet the assigned deadline, the exploration strategies we propose rely on more aggressive maneuvering as the temporal deadline approaches. To the best of our knowledge, this is the first study combining topological exploration with planning under temporal constraints. As the focus of this algorithm is on the algorithmic side, proposed strategies are only studied in simulation, but in the following chapter we transition onto a mobile platform.

The objective of this chapter is twofold. First, we revisit the classic exploration problem, introducing temporal constraints in the task and embracing a toposemantic spatial representation that includes no metric attribute. To assess the strengths and weaknesses of the various exploration methods abstracting from the underlying technical implementation, we perform a set of massive simulations in the Robot Operating System (ROS) using its Gazebo simulation environment. This simulation-based approach leads to the second objective of this contribution, namely, presenting a set of findings that are reproducible by a third party. In light of the increasing attention measurable robotics is attracting, this article represents a first attempt to present a reproducible study based on Gazebo.

The original contributions of this chapter are the following:

- In section 6.3 we present a new spatial model dubbed Oriented Topological Semantic Map (OTSM) that extends classic topological maps in a way that is amenable to implementation by robots with minimal sensor payload;

- In section 6.2 we discuss the creation of the new standard for robotics papers as Reproducible Articles or R-articles.
- We integrate the proposed spatial model with our recently developed planning method using CMDPs to assign actionable temporal deadlines to the robot;
- In section 6.5 we propose and experimentally compare multiple exploration strategies operating on semantic topological oriented maps. More than 3000 runs were executed to assess the relative strength of the proposed methods.
- We present a new framework to enable other researchers to reproduce our study based on ROS.

6.2 Replicability

One of the main contributions of this chapter is in ensuring complete replicability of the proposed methods. In [149] a set of guidelines for good experimental methodologies in robotics is given. According to such guidelines, this work should be characterized as “experimental” and shall be furthermore classified as research in “Autonomy/Cognitive Tasks”. The experimental part of this paper is based on ROS/Gazebo and is therefore particularly suited for being replicated by a third party. Indeed, replicability would be much more challenging if results were obtained on a physical P3AT robot, because of the countless variables influencing the final results. While the robotics community is becoming increasingly aware of the necessity to ensure repeatability in robotics research, “best practices” are still being defined, and standard tools to promote code replicability (e.g., Code Ocean) are not necessarily best suited or ready for all robotics research. This is particularly relevant when considering an end-to-end system relying on multiple external libraries to perform heterogenous operations like mathematical computation, navigation, and more.

Considering these challenges, we have provided an image for a virtual Linux machine including our code, all libraries necessary to run it, and a set of scripts to rerun all of the experiments to produce the results presented in this article in the supplementary materials section on IEEE Xplore: ***Download Sources Here.*** [2] .

The virtual machine image is based on Virtual-Box (www.virtualbox.org), a free software available for a variety of operating systems, including Windows, OsX, and Linux. In addition, we have also developed a technical document with step-by-step instructions to download and boot the virtual image, and run all experiments described in this paper.

As noted in [135], “A study is reproducible if you can take the original data and the computer code used to analyze the data and reproduce all of the numerical

[2] Available on IEEE xplore: `https://ieee-dataport.org/open-access/time-constrained-exploration-using-toposemantic-spatial-models-reproducible-approach`

findings from the study." Accordingly, the resources we provide in the following aim at making our study reproducible. With these premises, the experimental process that led to the results that we present is fully replicable by a third party. It is however, worthwhile to mention that a third-party will not necessarily obtain the same numerical outcomes because of two sources of randomness. First, the algorithms themselves feature various steps relying on randomized choices, e.g., to break ties. Second, the simulation environment is influenced by the underlying platform and will therefore not produce exactly the same results. The best one can expect is that results average over a very large number of trials will converge to a steady value. To the best of our knowledge, this is the first example of a fully replicable study of exploration algorithms.

6.3 Oriented Topological Semantic Maps

Oriented Topological Semantic Maps (OTSMs) can be understood as an extension to the classic definition of topological maps augmented with semantic and orientation information. As we studied in section 2.3, there are multiple approaches to create maps that represent an environment, among them, only few can be used in run-time with a CMDP due to the fact that solving a linear program can be very time-consuming for a high number of optimization variables $\rho(x, a)$. OTSMs enable the use of CMDP in run-time for navigation of an environment.

Topological maps model space as a graph $G = (V, E)$ with vertices representing places and edges modeling the ability to move between the places represented by the corresponding vertices. A "pure"topological map does not include any metric information, like for example the distance between two adjacent vertices, although various extensions adding this or comparable attributes have been proposed. Our model builds upon two assumptions. First, indoor environments are normally subdivided into corridors and rooms arranged along orthogonal directions. This observation was already made and exploited in [143]. Second, we assume that the robot is equipped with a device providing absolute orientation. Such sensor will, in the following, be generically called compass and nowadays there is a variety of inexpensive devices that can implement this functionality. Starting from these two hypotheses, without loss of generality, we assume that the walls and corridors of the building that the robot is exploring are arranged along the four cardinal directions, that will be in the following abbreviated as N (north), S (south), E (east), and W (west). An oriented topological map exploits these assumptions to define the relations "to the right of" and "to the left of" between vertices in the graph. Vertices in a graph may have different degrees:

- Degree 1: rooms, or corridor dead ends.
- Degree 2: corridors or two aligned ways.
- Degree 3: T junction or three way.

- Degree 4: Four way intersection.

Thanks to the compass, each edge can then be labeled as E-W or N-S, depending on the direction faced by the robot while moving along the edge. Accordingly, two adjacent vertices are said to be along the N-S (north-south) direction if the edge connecting them has the N-S label and we similarly define adjacent vertices along the E-W direction. Throughout the remainder of this work we make the following assumption. To define the relationships ”to the right/left of” for elements aligned along the N-S direction, we assume that the robot faces W, whereas for elements along the E-W direction we assume the robot faces N. Figure 6.1 illustrates this approach. The three vertices on the left are along the E-W direction. Based on our assumption that the robot points north to define left/right relationships, vertex $c1a$ is to the left of $c1b$ and $c1b$ is to the right of $c1a$. On the right, $c2b$ is on the right of $c2c$ and on the left of $c2a$.

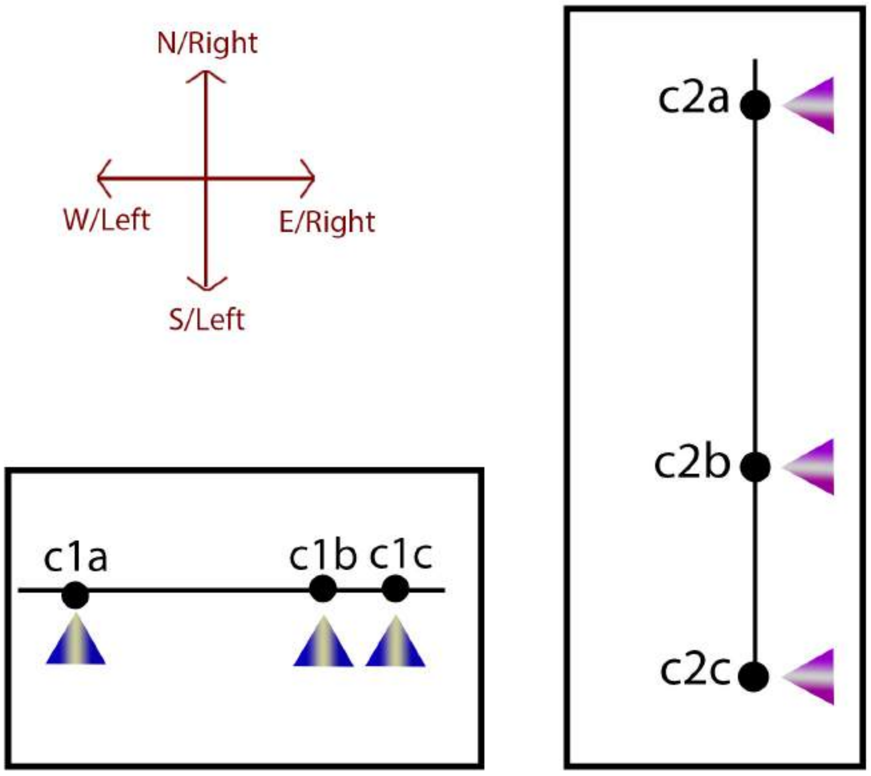


Figure 6.1: On the left: three adjacent vertices along the E-W direction. $c1a$ is on the left of $c1b$, and $c1c$ is on the right of $c1b$. On the right: three vertices along the N-S direction with $c2a$ on the right of $c2b$ and $c2c$ on the left of $c2b$.

For more complex topologies we can also create OTSMs, for example, when two corridors meet on a corner we can still use our convention for the direction to assign it to this vertex, however because some of the vertices are common for both corridors, only one label will be given. Each vertex will have only one semantic label that will describe, in human language, the location, thus the fact that one vertex is shared by two corridors will not affect the navigation. Figure 6.2 presents this case for three corridors $c1$, $c2$, and $c3$. where a OTSM can be defined.

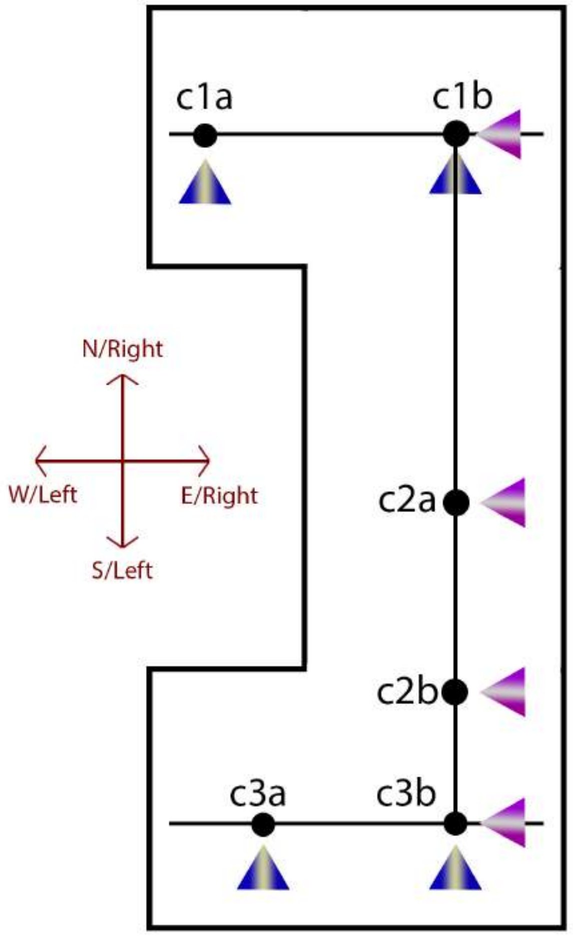


Figure 6.2: Three corridors are connected. For corridor 1 we have two adjacent vertices along the E-W direction. $c1a$ is on the left of $c1b$. For corridor 2 we have two adjacent vertices along the N-S direction with $c2a$ on the right of $c2b$. The third corridor have two adjacent vertices along the E-W direction. $c3a$ is on the left of $c3b$. $c3b$ is on the left of $c2b$, and $c2a$ is on the left of $c1b$

Because buildings have more than corridors, and we are interested in using the robot to find specific rooms or offices, we show in figure 6.3 the case where two corridors are common to one room. Although we know this case is very hypothetical, it serves to illustrate the flexibility of OTSMs for complicated and unusual topologies. The room again has only one label, but in the same way as two corridors, the reference to indicate the direction where a corridor is varies. The edge that connects *Room* 1 to $c1$ will have a direction North, and the edge that connects *Room* 1 to $c2$ will have direction West. If we want to move from the room to the corridor $c2$ we will turn right, and the robot will be pointing North. To go to corridor $c1$, we will point the robot West and we will turn Right. Both corridors are on the right of the room but the reference direction is the key to know where to go correctly.

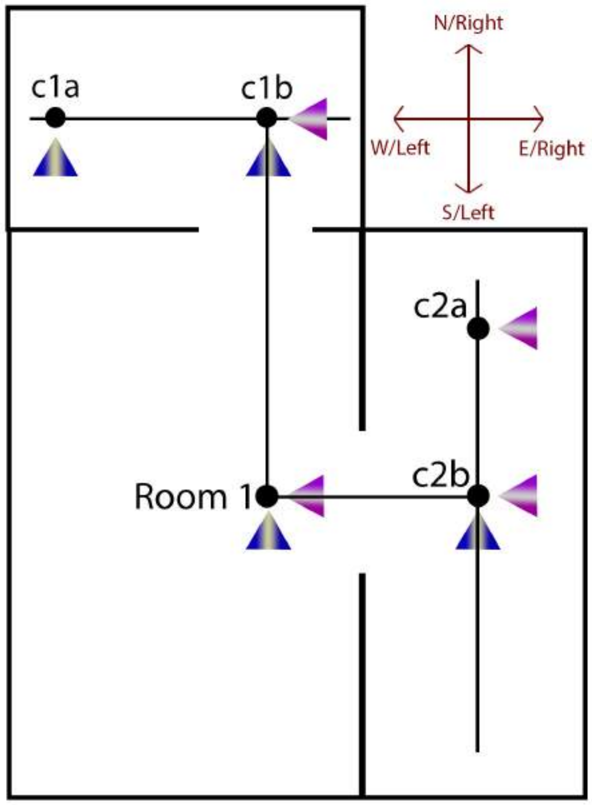


Figure 6.3: Two corridors and one room in the middle. For corridor 1 we have two adjacent vertices along the E-W direction. $c1a$ is on the left of $c1b$. For corridor 2 we have two adjacent vertices along the N-S direction with $c2a$ on the right of $c2b$. The Room 1 is on the left of $c2b$ and on the left of $c1b$

6.3.1 OTSM Formulation

The following definition formalizes the structure of a map.

A semantic topological oriented map is defined as $\mathcal{M} = (V, E, \mathcal{L}, \mathcal{D}, \mathcal{S})$ where:

- (V, E) are the vertices and edges of a directed graph.
- $\mathcal{L} : V \to L$ assigns a unique semantic label to each vertex.
- $\mathcal{D} : E \to \{\text{E-W}, \text{N-S}\}$ is a function assigning a direction to each edge. Note that to the same direction we can associate two orientations, e.g., an edge along the N-S direction can be traversed from North to South or from South to North.
- $\mathcal{S} : E \to \{L, R\}$ is a function assigning a label L (to the left of) or R (to the right of) to each edge.
- If $e = (v_i, v_j) \in E$ is an edge from v_i to v_j and $\mathcal{S}(e) = L$, then $e' = (v_j, v_i) \in E$ and $\mathcal{S}(e') = R$. Likewise, $e(v_i, v_j) \in E \wedge \mathcal{S}(e) = R \Rightarrow e' = (v_j, v_i) \in E \wedge \mathcal{S}(e') = R$.
- For each couple of adjacent vertices, $\mathcal{D}(v_i, v_j) = \mathcal{D}(v_j, v_j)$.

The last two conditions establish two consistency constraints. First, if it is possible to go from v_i to v_j along one direction (say N-S), then it is possible to go from v_j to v_i along the same direction but opposite orientation (say S-N). Second, if there is an edge from v_i to v_j indicating that v_j is to the left of v_i, then there must also exist the opposite edge from v_j to v_i indicating that v_i is to the right of v_j.

To account for the fact that a map $\mathcal{M}$ is incrementally expanded during the exploration stage, we introduce a special vertex $v_\varnothing$. Such a vertex represents an unknown place and is introduced when the robot reaches a place like a four way intersection represented by a vertex v from which there are outgoing edges towards still to be explored locations. These edges are, therefore, between the current vertex v and $v_\varnothing$ and can be considered as unexplored by the exploration algorithm. These edges are also subject to the constraints given above.

One last comment is in order regarding the labeling functions $\mathcal{L}, \mathcal{S}$ and $\mathcal{D}$. Assigning a label $\mathcal{D}$ to an edge is simple because of the compass assumption. Likewise, as we will show in the following, while incrementally building a map $\mathcal{M}$, we will ensure that $\mathcal{S}$ labels are assigned to the edges consistently with the constraints we introduced. For the semantic label $\mathcal{L}$ assigned to vertices, we make two assumptions supported by contemporary algorithms for machine learning and computer vision (see e.g. [52] and [109]). First, it is possible to train an image classifier capable of distinguishing a room from a corridor. If the robot is in a corridor, the image processing algorithm can furthermore, determine if the robot is facing an intersection (three way or four way), or a ninety degree turn (left or right). Second, the classic place revisiting problem typical of SLAM can be solved, i.e., through computer vision the robot is capable of determining if it is visiting a place for the first time, or of it is revisiting a formerly explored area (and which). The algorithms providing $\mathcal{L}, \mathcal{S}$ and $\mathcal{D}$ will be in the following, indicated as Labeling System (LS) and the Intersection Detection System (IDS) respectively. One will provide the labels and the second, the edges for each vertex, but both work as one every time for each vertex. Figure 6.4 shows how an oriented topological semantic map can be used to represent the interior of a building. In this figure we can observe the edges with two different colors, that symbolizes the direction (left/right) to reach the nodes, and we see how each vertex correspond to a room or an intersection between rooms and corridors. The topology is rather simplistic for orthogonal environments like office buildings, but it is still an open question to know how OTSMs can be adapted to other topologies.

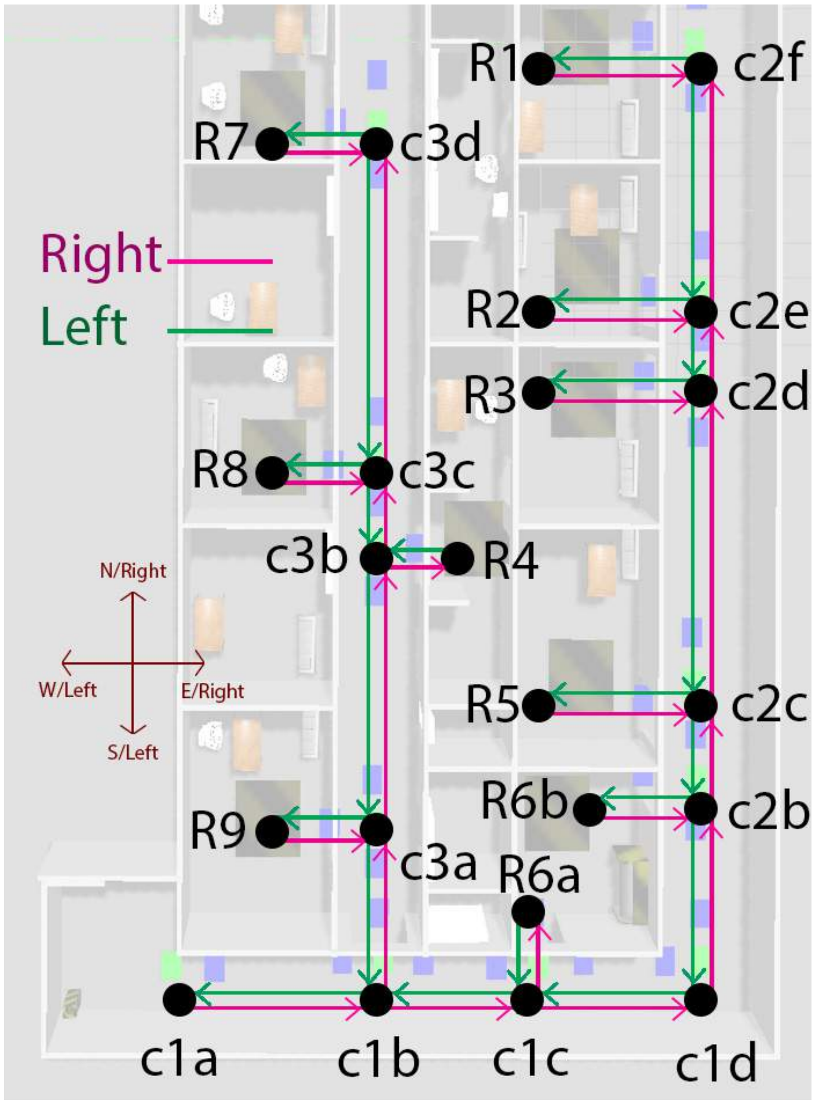


Figure 6.4: Oriented Topological Semantic Map - OTSM. Vertices with a label Rx are rooms, while corridors have labels of the type Cx. Edges with a R label are purple, while edges with a L label are green.

As an example of OTSMs, figure 6.5 shows three different floors from the University of California Merced. From the CAD file we manually created the OTSM as a way to exemplify how they can look in different real scenarios.

6.3.2 Incremental Map Construction, Navigation, and Exploration

We now describe how an oriented topological semantic map $\mathcal{M}$ can be incrementally built and used for navigation by a robot exploring an unknown indoor environment while attempting to reach a destination within a given deadline. Here we provide the way how we include CMDPs with OTSMs. The input provided to the robot consists of two elements: a goal location that can be recognized by the LS module (indicated as $\mathcal{G}$ in the following), and a global temporal deadline $\mathcal{T}$. Here we assume that formerly outlined hypotheses regarding the availability of a compass and the orthogonal structure of the environment hold. The LS module determines when a robot has reached a vertex (either new or formerly visited) and the IDS determines how many outgoing edges are emanating from the vertex. From our former assumptions, this is a number between one and four. The *createNewNode* (sketched in algorithm 6.2) is called every time LS identifies a corridor intersection or a room that was not formerly visited.

Since the robot starts with an initially empty map, the initialization process sketched in Algorithm 6.1 is preliminarily run to bootstrap the exploration and mapping process. The robot randomly moves around until it has K vertices in the graph (in our following experiments, K is fixed to 3).

```
1: **Algorithm** $InitializeGraph()$
2: $\mathcal{M} \leftarrow$ empty map
3: Get vertex $v$ and $edges$ from LS and IDS
4: Insert $v$, $v_\varnothing$ in $V$
5: Insert $edges$ between $v, v_\varnothing$ in $E$
6: **while** size of $|V| < K$ **do**
7:     Select random edge outgoing from current vertex
8:     follow edge until LS return new vertex $v$
9:     createNewNode($v, edges$)
10: **end while**
11: **return** $\mathcal{M}$
```

Algorithm 6.1: Map Initialization

The function $createNewNode(v', edges)$ takes, as parameters, the new vertex v' to be added and the set of edges emanating from it, and it updates the set of vertices and edges assigning the appropriate labels. Unvisited edges, i.e., edges towards parts of the graph still to be explored, are of the type $(v', v_\varnothing)$, to indicate they lead to unexplored areas. In the function, labels for vertices and edges are set to satisfy the constraints listed in subsection 6.3.1.

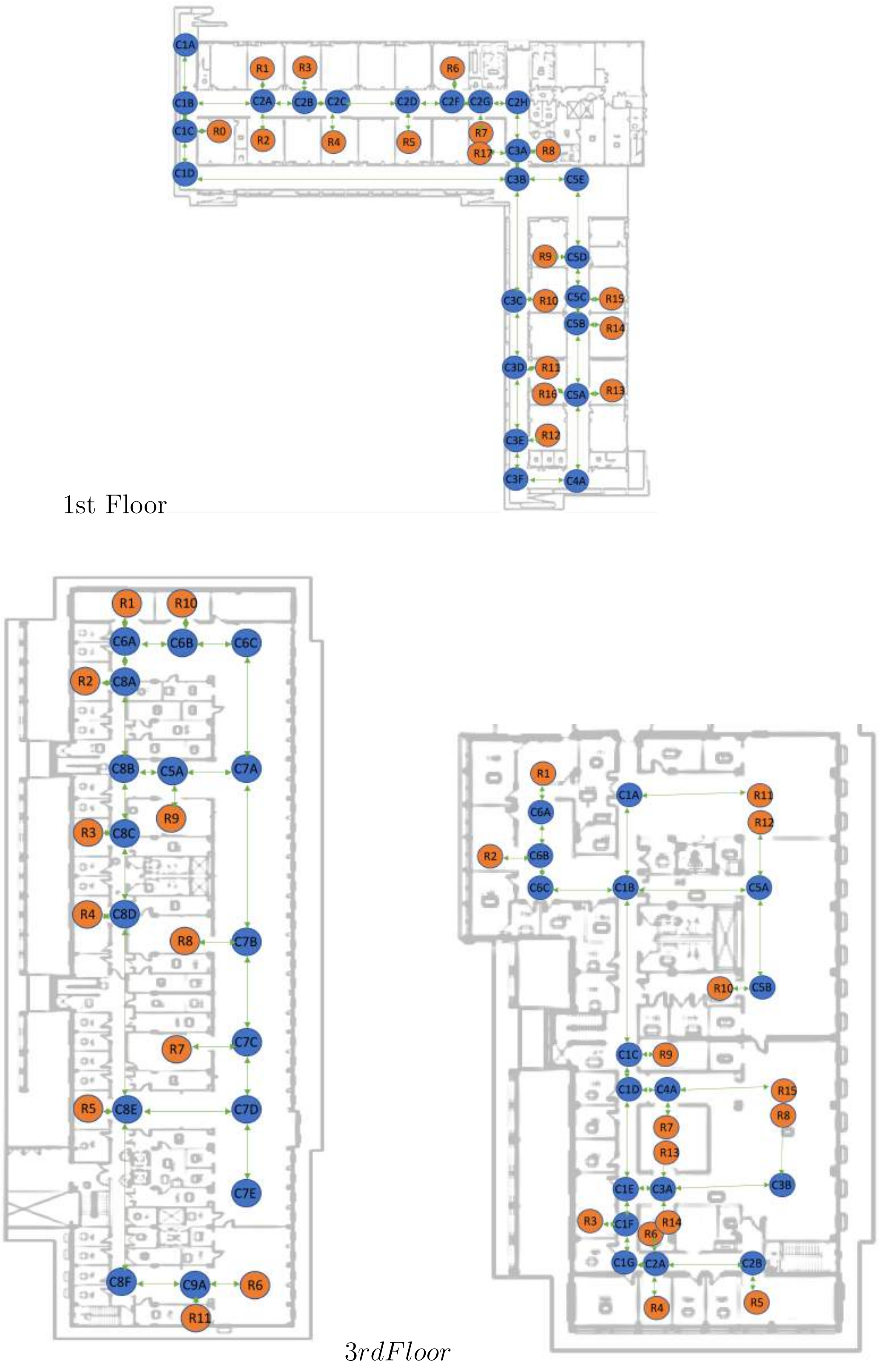


Figure 6.5: Examples of OTSM taken from CAD designs from the University of California Merced.

```
1: **Algorithm** $createNewNode(v', edges)$
2: Let $v$ be the vertex the robot came from
3: $V \leftarrow V \cup \{v'\}$
4: $E \leftarrow E \cup \{(v, v'), (v', v)\}$ using IDS.
5: Set $\mathcal{L}(v')$ as per LS (room or corridor)
6: Set $\mathcal{D}(v', v)$ and $\mathcal{D}(v', v)$ as per compass
7: Set $\mathcal{S}(v', v)$ and $\mathcal{S}(v', v)$ as per compass
8: **for all** unvisited outgoing edges from $v$ **do**
9:     Add edges of type $(v', v_\varnothing)$ and set labels $\mathcal{S}, \mathcal{D}$
10: **end for**
```

Algorithm 6.2: Adding a new node to the map

The algorithm to navigate from the current vertex v to an assigned target vertex v' is sketched in algorithm 6.3. The algorithm takes two parameters, i.e., v' and a temporal deadline t_d. Later on we discuss how these are selected. To navigate to the vertex, a CMDP is built considering the given temporal deadline and a probability of failure P_f (assumed constant in the following). The temporal deadline and the probability of failure are the additional costs to be bounded in the CMDP formulation. The CMDP is then solved, producing an optimal policy π^* that is followed until a termination condition occurs. At each step, the policy π^* is used to determine the primitive a to be executed from the current vertex. Termination may happen because the global temporal deadline expires, and in such a case the robot reports failure because it has failed its overall task. Alternatively, the robot may report a local success when it reaches the given vertex v' but still has not reached the global target vertex $\mathcal{G}$, or global success if it reaches it. It is at this stage that the orientation in the map becomes relevant because it will influence which actions (maneuvers) are selected by the planner.

Finally, in algorithm 6.4, we show how the overall exploration task is solved. After creating the initial map $\mathcal{M}$, the robot enters a loop that will be continued until either the global temporal deadline $\mathcal{T}$ expires, or the robot succeeds in reaching $\mathcal{G}$. At each iteration, the function *ExplorationNext*, returns the vertex v to move to and a temporal deadline t_d, i.e., how much time the robot should spend to reach v. To determine these two quantities, *ExplorationNext* needs the current map, as well as the variable *time*, indicating how much time passed since the exploration task started. As *time* grows, more stringent temporal deadlines t_d will be returned. Details about how vertices and deadlines are computed are discussed in the next section. The robot then attempts to navigate to the assigned vertex v and once there, it randomly picks an outgoing edge and follows it, until LS indicates that a new vertex has been reached. The maneuver chosen to traverse the edge e depends on the relative orientation between the edge and the robot.

In the next section we will present how we used all the previous algorithms to explore an office building using several exploration strategies and exploiting the

```
 1: Algorithm $NavigateToVertex(v, t_d)$
 2: $CMDP \leftarrow BuildCMDP(\mathcal{M}, v, t_d, P_f)$
 3: $\pi^* \leftarrow Solve(CMDP)$
 4: while $time < \mathcal{T}$ do
 5:     $v_c \leftarrow$ current vertex
 6:     if $v_c = \mathcal{G}$ then
 7:         return  Global Success
 8:     end if
 9:     if $v_c = v$ then
10:         return  Local Success
11:     end if
12:     $a \leftarrow \pi^*(v_c)$
13:     execute $a$
14: end while
15: return  Failure
```

Algorithm 6.3: Navigation Algorithm

```
 1: Algorithm $ExplorationTask(\mathcal{T}, \mathcal{G})$
 2: $\mathcal{M} \leftarrow InitializeGraph()$
 3: while $time < \mathcal{T}$ do
 4:     $v, t_d \leftarrow ExplorationNext(\mathcal{M}, time)$
 5:     $flag \leftarrow NavigateToVertex(v, t_f)$
 6:     if $flag$ = Local Success then
 7:         Pick random edge $e$ out of $v$
 8:         follow $e$
 9:         $CreateNewNode(v, edges)$
10:     else
11:         return  $flag$
12:     end if
13: end while
14: return  $Failure$
```

Algorithm 6.4: Global Exploration Algorithm

information provided by OTSMs.

6.4 Exploration Algorithms

As we mentioned, OTSMs contain different type of information, that is available for the robot to exploit in different ways. We can think about using only the topology, or only the semantics, or a combination of both. In this section we present five different strategies that can be used to guide the robot through its exploration task. Each of these strategies represents a different way to select the vertex v returned by the function $ExplorationNext$ used in algorithm 6.4. At the end of the section, we also discuss how we set the temporal deadline.

6.4.1 Random Strategy Exploration

The random exploration strategy is our baseline approach. The function returns a random vertex among all vertices with one or more outgoing unvisited edges, i.e., edges towards $v_{\varnothing}$. If there is more than one vertex with unvisited edges, a uniform probability distribution is used to make a choice. It is easy to show that this strategy will eventually explore the whole environment, provided that the robot does not terminate earlier due to a collision. This exploration approach is the topological equivalent of the classic random exploration when using metric maps or other spatial models.

For this exploration method, after we selected a random node with unvisited edges, we set this node as a $localtarget$, set a temporal deadline and solve the CMDP to obtain the policy to reach it. After navigating and arriving to the $localtarget$ the robot can choose, again, randomly where to go from all the unvisited edges of the current node. This strategy, although very inefficient, will assure exploration of the whole environment or until finding the $GlobalTarget$.

6.4.2 Topological Frontier

Topological frontier is the analogue of the well-known frontier based exploration algorithm [183]. In frontier based exploration, frontiers are defined as the regions on the boundary between explored and unexplored space, and the robot then moves to one frontier to expand the explored space. When multiple frontiers are present, various selection criteria can be used to make a choice. In our topological domain, the boundary between known and unknown parts of the environment are identified by unexplored edges. Accordingly, the topological frontier algorithm sorts all nodes by the number of outgoing unexplored edges, and it randomly selects one among those with the highest number of outgoing edges. In this latter case a uniform distribution among all candidates is used. This technique ensures that the robot always moves towards the largest frontier.

6.4.3 Topological Frontier with Normalized Distances

The next strategy also translates in the topological domain another of the features of basic frontier exploration, i.e., when multiple frontiers are present, the cost to move to a frontier is also factored in when selecting where to go. In the topological case the cost to move is set to the number of edges in the path connecting the current vertex to a target location. The rationale is to choose a closer vertex when one or more equivalent ones are present. Since we need to combine together heterogeneous quantities (number of outgoing unexplored edges and distance), we use a linear combination of normalized quantities (see e.g., [32]) to assign a value to each vertex in the map, and we then select the vertex maximizing this metric. To be specific, let $\mathcal{V} \subset V$ be the set of vertices with one or more outgoing unexplored edge. For each vertex $v \in \mathcal{V}$ we compute the following quantity

$$S(v) = \gamma \frac{\deg(v)}{\max_{v' \in \mathcal{V}} \deg(v')} - (1-\gamma) \frac{\mathrm{d}(v)}{\max_{v' \in \mathcal{V}} \mathrm{d}(v')} \tag{6.1}$$

where $\deg(v)$ is the number of outgoing unexplored edges and $\mathrm{d}(v)$ is the distance in the topological map, defined as the number of edges in the shortest path in the graph. The function then returns the node in $\mathcal{V}$ with the highest value for $S(v)$ with ties arbitrarily broken, if any. In the experiments presented in the following section we set $\gamma = 0.5$. This approach then tries to drive the robot to frontiers that are large and near.

6.4.4 Semantic: Explore Corridors First

The first semantic exploration strategy relies on the labels attributed to vertices in the graph. In particular, it exploits the assumption that we can distinguish between rooms and corridors. Inspired by the work by Quattrini et al. [143], the robot prioritizes corridors when selecting where to go next. That is to say that if among the vertices with unexplored outgoing edges there are both corridors and rooms, the robot always selects corridors first. Rooms are selected only when no coorridor vertices can be selected.

6.4.5 Semantic: Complete Corridors First

The last exploration strategy is somewhat complementary to the previous one. This time, before moving to a different corridor, the robot finishes exploring the corridor it is located in. To accomplish this, the semantic label $\mathcal{L}$ of a vertex and the directions of its adjacent edges are considered. Here, a corridor is defined as a path along the graph $G = (V, E)$ such that all vertices are labeled as corridor vertices by the function $\mathcal{L}$, and all edges connecting the vertices have the same label $\mathcal{D}$, i.e., they are all along the N-S or E-W direction. For example, in figure 3.1 $c1a - c1b - c1c - c1d$ is a corridor along the E-W direction that cannot be further extended because other vertices of type corridor can only be reached traversing edges along the N-S direction. The robot enters a new corridor only once the previous corridor can no longer be extended. This approach further capitalizes one aspect in algorithm 6.3, i.e., once a vertex is reached, a random outgoing edge is picked and traversed, too. This has the effect of exploring rooms connected to the corridor while this is being explored. In this case, too, rooms are selected when all corridor vertices have been explored already.

6.5 Experimental Validation

6.5.1 Setup

As a preliminary step towards evaluating strengths and weaknesses of the proposed approach, we perform an extensive set of tests using ROS and the associated Gazebo simulation environment. The advantage of this initial simulation study is that it allows us to easily perform hundreds of runs with moderate effort. This will provide preliminary insights about weaknesses and strengths of the approach to then perform more meaningful tests on the real platform. Our plan to migrate our code to a real robotic platform is sketched in the final section. The simulated environment is a faithful replica of one of the engineering buildings of the University of California, Merced, and its model in Gazebo was built from its architectural CAD design (see Figure 6.6).

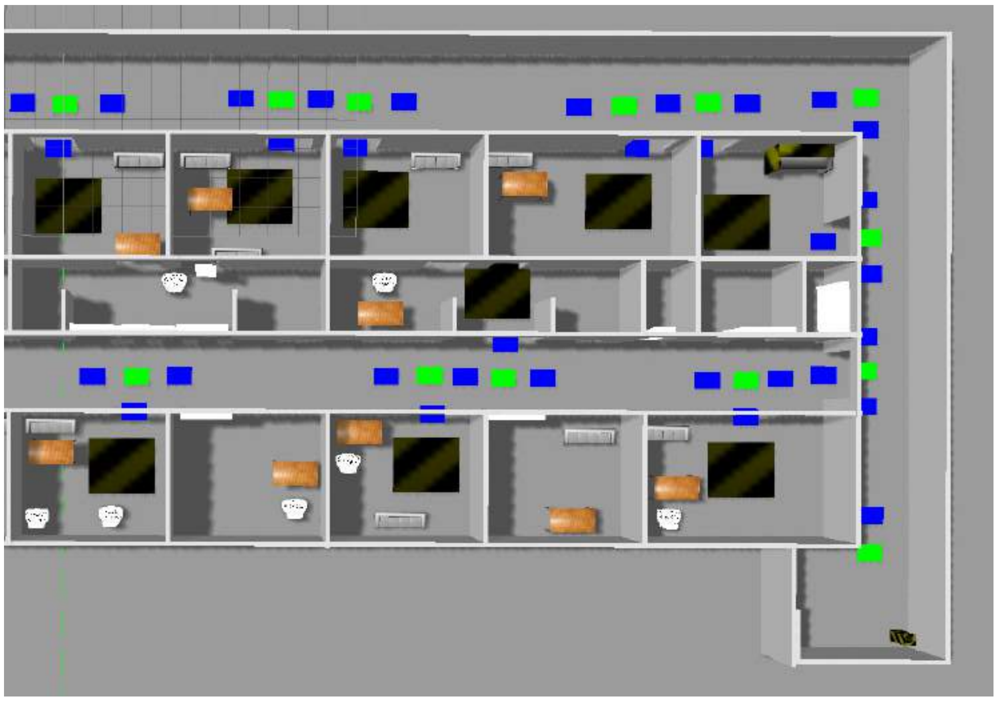


Figure 6.6: Engineering building at the University of California Merced.

The simulated robot is a Pioneer 3AT with limited sensing capabilities compatible with our former assumptions. In particular, the robot is equipped with only a compass, a laser range finder to avoid obstacles, and a logical camera to detect features in the environment. The logical camera is a ROS plugin that returns the position and orientation of any object in its cone of vision with respect to the robot.[3] The logical camera abstracts the implementation of the IDS system based on computer vision. This is obtained by embedding unique features in the environment that can be detected by the camera (these are displayed as green and blue tags in Fig. 6.6). It shall be noted that even though the simulated robot is equipped with a laser range finder, it does not use it to extract any sort of metric information, and it is only used for obstacle avoidance when moving along corridors or entering doors. In fact this could be replaced by other sensors providing proximal information to avoid collisions such as sonars or depth-sensing cameras. To test our navigation system under realistic conditions, a 25% error is added to linear and angular velocity commands, and a 5% error is added to orientation readings.

6.5.2 Maneuvers

To navigate the environment, the action set A for the CMPD includes six maneuvers. Specifically, there are three elementary motions, and each can be executed fast or slow. The first maneuver is go through a corridor. This maneuver moves the robot forward (i.e., keeping the same orientation) while trying to keep an equal distance from the walls on either side. The second maneuver is go through door on the right. This will turn the robot to its right (relative to its current location and orientation) and move through a door, as identified by the IDS system. The third maneuver is the symmetric go through door on the left.

To calculate the transition probabilities for each of the six maneuvers, we ran a series of experiments in which the maneuvers are repeated 500 times while we varied the initial position and orientation. The robot had to go either through a corridor or pass through a door on the left/right. For each trial we recorded the time spent, as well as if the robot was successful or not, with failure defined as a collision with the wall. This gave us the transition probability value to be used with the CMDP for that specific pair maneuver/speed. The variation of position was ± 8 meters and for the orientation of 30 degrees for the initial pose of the robot.

6.5.3 Calculating the Failure and Temporal deadline

Every time the robot takes an action, it will use a certain amount of time until it reaches the next node and this is the time is saved and stored in the structure that contains the semantic topological map. This data structure records the discovered nodes, the visited and unvisited edges of each node, the direction where the edges are located (N,S,W,E), the connected nodes to each edge and finally, the times measured in seconds to reach that specific node from that specific edge. We calculate the

[3]`http://wiki.ros.org/ariac/Tutorials/SensorInterface`

number of nodes between the starting location and the local target using the Dijkstra's algorithm, and with this number, we define the number of actions that are necessary to reach that local target. Because we know that the action was always a slow one during the exploration, we can assume that, for example, if there are four nodes in a path to reach a local target, we will spend a maximum of four slow actions to reach the target. At most, the CMDP will receive a temporal deadline four times the time of a slowest action.

As mentioned, the navigation algorithm uses a CMDP to find the optimal path while balancing two constraints. For the CMDP we need to define a failure state when the robot fails to reach the target. Two cases are considered as a failure. First we have the case when the robot collides with an obstacle. This is considered an immediate failure and during the experiments it will be counted as such. The second case is when a temporal deadline is not respected locally or globally. We define the global temporal deadline experimentally: the robot will use a random exploration strategy, with the lowest speed for each action, to define the baseline of how much time the robot takes to reach the target. The actions, on the other hand are defined as "go through corridor" and "cross a door", each one with two different speeds and two different failure probabilities. The time spent to go from one node to another is variable, depending on the physical distance and the speed. We executed each action 50 times to estimate the average time used to move between nodes. The time we obtained, measured in seconds, for the "fast go through corridor" was 4.8378, for "fast cross a door" was 13.8409, for "slow go through corridor" 9.1266 and finally for "slow cross a door" was 36.6746.

When the global temporal deadline is above 80%, the local temporal deadline is the maximum, which for our example, it would be four times the cost of the slowest action (4x36.6746s). If the global temporal deadline goes lower than 80%, but higher than 50%, then we will use 50% of the maxmimum temporal deadline as a local temporal deadline. If the global time goes lower than 50%, then we will use 30% of the maxmimum temporal deadline as local temporal deadline for the CMDP. The variation of local temporal deadline will force the planner to chose between fast and more risky actions or slow and safer actions. The safety of each action will be calculated with the failure probability determined by the speed of such action. Because we set a very low failure probability constraint of 1%, it is possible that for the minimum temporal deadline, and using the fastest actions a policy cannot be calculated and therefore the robot will report a failed mission. If, by the contrary, the policy is possible, then the planner will return a solution and the robot will execute the solution. Each time the robot executes an action from the policy the time is measured again in seconds and discounted from the established global temporal deadline at the beginning, which will affect the decisions and planning of the next local target.

In summary, one possible way to pick the temporal deadline assigned to reach a given vertex is the following: The rationale is that as the global deadline $\mathcal{T}$ is approaching, the robot should speedup. While there are evidently countless ways to implement this, we currently opt for a very simple schedule. Once a vertex v is selected, we define a preliminary deadline B equal to the number of edges between the current vertex v_c and v multiplied by a constant. Then, if the time available to complete the task is larger than $0.8\mathcal{T}$, we set $t_d = B$. If instead the time to complete the task is between $0.5\mathcal{T}$ and $0.8\mathcal{T}$ we set $t_d = 0.5B$. Finally, if the time to complete the task is less than $0.5\mathcal{T}$, we set $t_d = 0.3B$.

6.5.4 Software Architecture

We used ROS Kinetic to create a modular software easily adaptable, exportable and executable for any device that can run ROS Kinetic. The ROS nodes were created in C++ and a few in python to show the adaptability of the platform. The map building mechanism and the simulation were all developed with ROS Kinetic and Gazebo 7.0. A general idea of how the software is structured is shown on the Figure 6.7. Five general blocks communicate to test different exploration strategies. The first block corresponds to the robot itself. This is a pioneer 3AT robot. The robot is equipped with 3 sensors: a LIDAR, a logical camera, which identifies objects in the simulated environment, and a compass to get the global orientation. To exchange information, we used a folder where text files will be shared by the map manager and the CMDP solver. The topological map is represented as an adjacency matrix with all its nodes and edges, and the solution for the CMDP will be saved and used by the map manager to control the robot based on the calculated policy. Once a node is added to the map, and the chosen exploration strategy decides the next local target, this information is passed through the shared folder to the CMDP solver to obtain the optimal solution while respecting the temporal and failure probability constraints. Then, the CMDP solver will calculate and generate a policy that will guide the robot towards the local target.

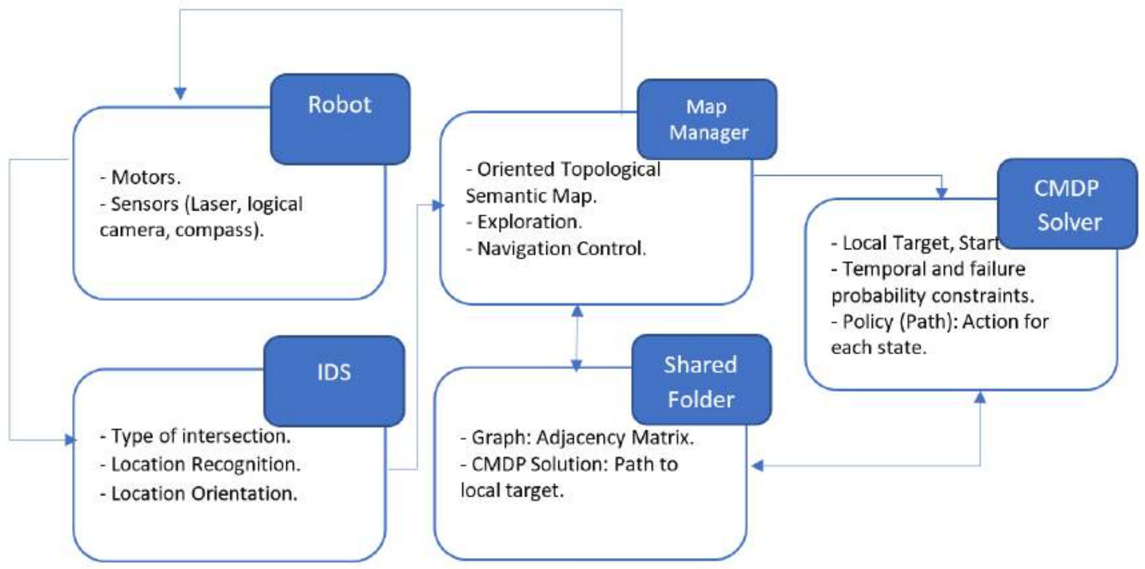

Robot
- Motors.
- Sensors (Laser, logical camera, compass).
Map Manager
- Oriented Topological Semantic Map.
- Exploration.
- Navigation Control.
CMDP Solver
- Local Target, Start
- Temporal and failure probability constraints.
- Policy (Path): Action for each state.
IDS
- Type of intersection.
- Location Recognition.
- Location Orientation.
Shared Folder
- Graph: Adjacency Matrix.
- CMDP Solution: Path to local target.

Figure 6.7: Software Architecture.

6.6 Results

We here compare the five exploration strategies on an exploration task where we vary the complexity of the navigation task, defined as the distance between the robot start location and the target location, and the assigned temporal deadline. The environment is shown in figure 6.8, where the goal location $\mathcal{G}$ is the room marked with the green star, whereas the three different places marked A, B and C identify the different start points considered. Throughout the simulation, the probability of failure when computing the CMDP policy was set to 0.01. For each starting point we consider four different temporal deadlines, and we then execute 500 trials. A trial is considered a failure if the robot has not reached the target location by the given deadline, or collides with the environment while navigating. Overall, 3000 independent tests were executed to evaluate the five exploration strategies.

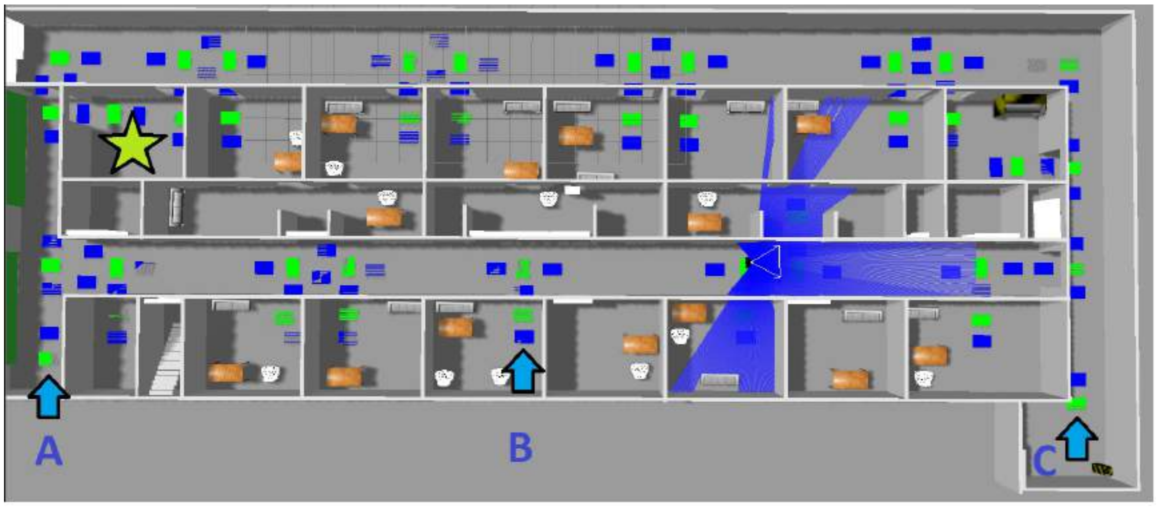
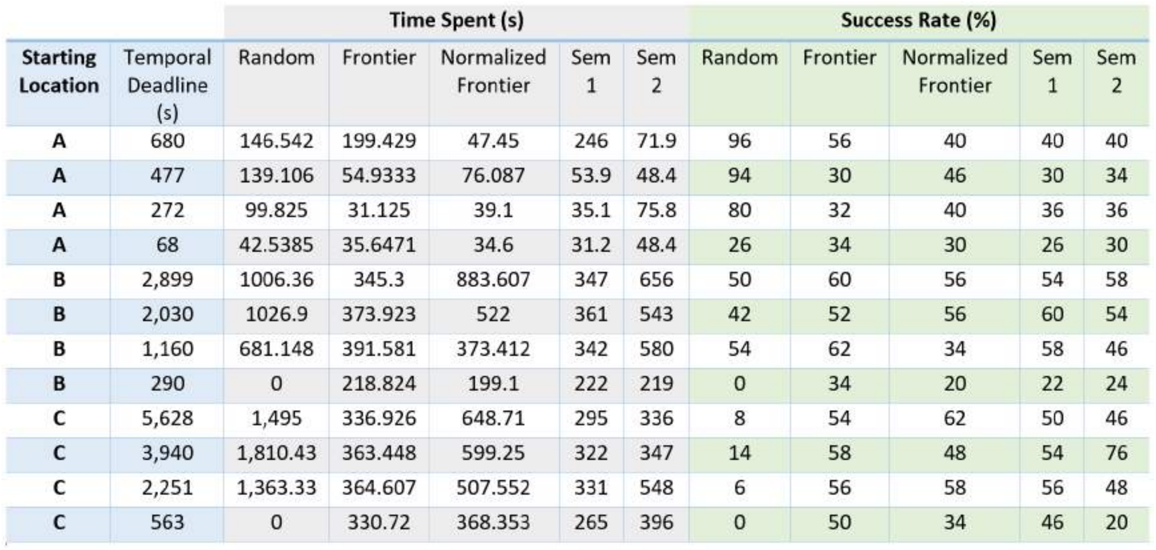


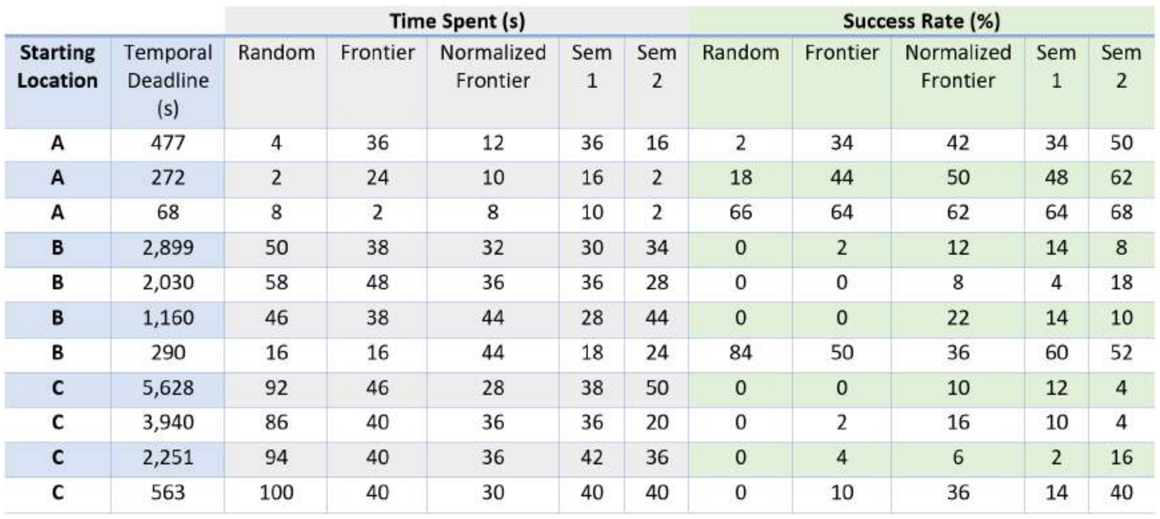
Figure 6.8: Three starting points (A, B, C) and one target

Table 6.9 summarizes the performance for the five exploration methods we propose. The columns give the starting location, temporal deadline expressed in seconds, time spent, and success rate for each type of exploration: Random, Frontier, Normalized Frontier, Semantic Strategy: Explore Corridors First, and Semantic Strategy: Complete Corridors First. For each combination of start location and temporal deadline, we provide the success rate and the average time spent to reach the target. The average time is given for successful runs only, because unsuccessful runs may result from exceeding the temporal deadline or because of collisions with the environment; therefore it would not be meaningful to average over unsuccessful runs, too. Instead, Table 6.10 analyzes, in greater detail, the causes of the failures for each strategy.



		Time Spent (s)					Success Rate (%)				
Starting Location	Temporal Deadline (s)	Random	Frontier	Normalized Frontier	Sem 1	Sem 2	Random	Frontier	Normalized Frontier	Sem 1	Sem 2
A	680	146.542	199.429	47.45	246	71.9	96	56	40	40	40
A	477	139.106	54.9333	76.087	53.9	48.4	94	30	46	30	34
A	272	99.825	31.125	39.1	35.1	75.8	80	32	40	36	36
A	68	42.5385	35.6471	34.6	31.2	48.4	26	34	30	26	30
B	2,899	1006.36	345.3	883.607	347	656	50	60	56	54	58
B	2,030	1026.9	373.923	522	361	543	42	52	56	60	54
B	1,160	681.148	391.581	373.412	342	580	54	62	34	58	46
B	290	0	218.824	199.1	222	219	0	34	20	22	24
C	5,628	1,495	336.926	648.71	295	336	8	54	62	50	46
C	3,940	1,810.43	363.448	599.25	322	347	14	58	48	54	76
C	2,251	1,363.33	364.607	507.552	331	548	6	56	58	56	48
C	563	0	330.72	368.353	265	396	0	50	34	46	20

Figure 6.9: The performance of the five exploration strategies.



		Time Spent (s)					Success Rate (%)				
Starting Location	Temporal Deadline (s)	Random	Frontier	Normalized Frontier	Sem 1	Sem 2	Random	Frontier	Normalized Frontier	Sem 1	Sem 2
A	477	4	36	12	36	16	2	34	42	34	50
A	272	2	24	10	16	2	18	44	50	48	62
A	68	8	2	8	10	2	66	64	62	64	68
B	2,899	50	38	32	30	34	0	2	12	14	8
B	2,030	58	48	36	36	28	0	0	8	4	18
B	1,160	46	38	44	28	44	0	0	22	14	10
B	290	16	16	44	18	24	84	50	36	60	52
C	5,628	92	46	28	38	50	0	0	10	12	4
C	3,940	86	40	36	36	20	0	2	16	10	4
C	2,251	94	40	36	42	36	0	4	6	2	16
C	563	100	40	30	40	40	0	10	36	14	40

Figure 6.10: An analysis of the failures for the five exploration strategies.

6.6.1 Random Strategy Exploration

For the first experiment, we varied the temporal deadline and we kept the failure probability constraint to 1% because, at this time, we only want to establish the base line to know the success of our task without affecting the speed of the maneuvers. Because the random strategy is very inefficient, the robot has to travel, sometimes multiples times, through the same corridor when going to the chosen random node that was selected as a local target. This makes the robot more prone to fail and hit a wall because more maneuvers implies more risk of collisions and mistakes. After 50 tries from each starting point, we obtain the highest success rate of 96% for point A, 54% for point B, and 14% for point C. This supports our initial idea that the robot has more chances to fail when it is located far from the target. We can also observe how the reduction of the temporal deadline has a negative impact in the success rate.

6.6.2 Topological Frontier Exploration

For the topological frontier we observe that, in general, we can always reach the global target despite the starting point. These results differ those from the random strategy when the robot had low success reaching the goal with low temporal deadlines and long distances to the target.

6.6.3 Topological Frontier with Normalized Distances Exploration

The normalized topological frontier did not improved the previous results, and even when we can reach the target from point C, we have a low rate of success when the target is close to the starting point A. In general this strategy seems only adequate when the target is far from the starting point C.

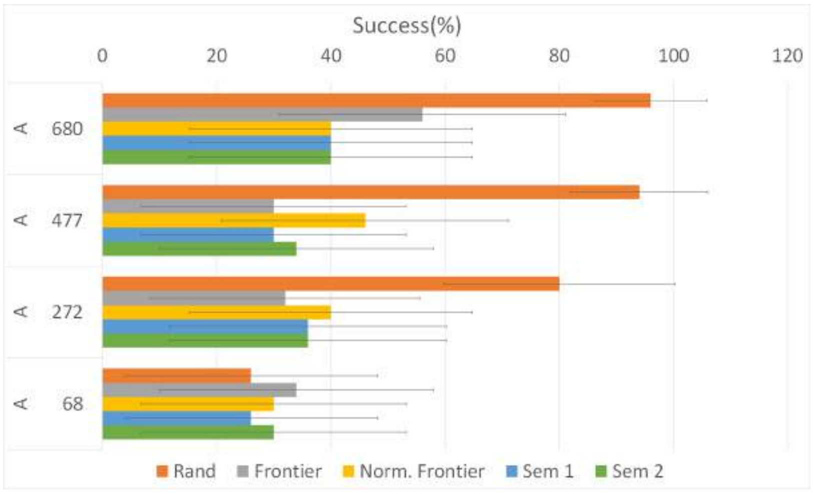
6.6.4 Semantic Strategy Exploration

Unsurprisingly, the random strategy is the most effective when starting from location A, except when the shortest deadline is enforced. This is somewhat expected, given that other strategies tend, instead, to expand the map in a principled way that may push them far from the target location that is relatively close to A. However, in the other cases this strategy is less effective, and performs very poorly for the most challenging case C. All things considered, the topological frontier with normalized distances appears to be most effective when the temporal deadline is not too demanding. However, as the deadline becomes more stringent, its advantage seems to vanish and it becomes more or less comparable to the topological frontier.

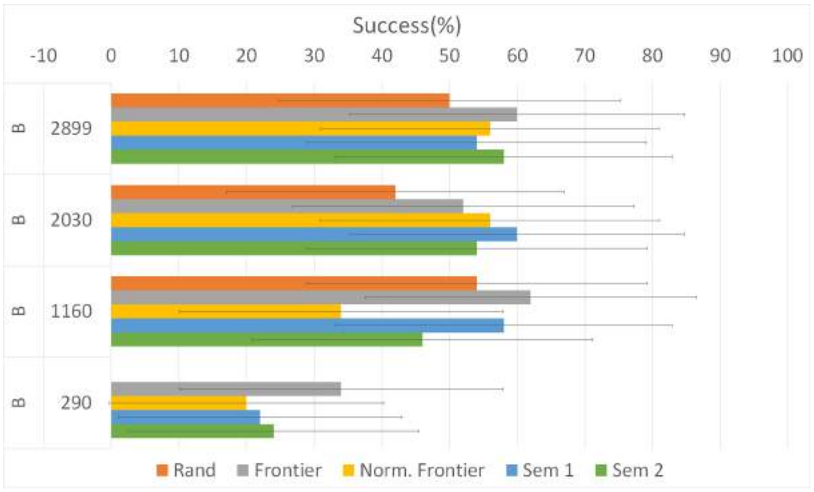
The semantic strategies, on the contrary, appear to have a performance that is less dependent on the start location. None of the two clearly outperforms the other, but the second one seems, in general, better than the first. It remains an interesting question whether by using additional semantic information their performance could be improved. For example, having a prior about typical spatial relationships (e.g., receptionist desks are typically near the entrance), could lead to increased success.

One final comment should be made regarding the success rates, as they may appear to be on the low end. This is due to the fact that the temporal deadlines are strict, and this boosts the failure rates for all of the algorithms. A strict temporal deadline is not only harder to meet but also forces the CMDP planner to utilize more aggressive maneuvers, thus possibly also increasing the number of failures due to collisions with the environment.

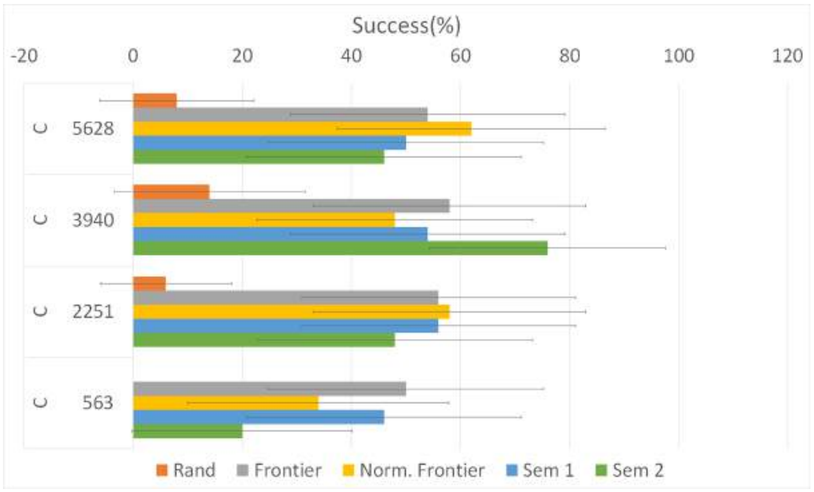
To facilitate the comparison, the results are also visually compared in Figures 6.11 and 6.12 . Unsurprisingly, the random strategy is the most effective when starting from location A, except when the shortest deadline is enforced. This is somewhat expected, given that other strategies tend instead to expand the map in a principled way that may push them far from the target location that is relatively close to A. Moving farther from the target increases the time to complete the mission and, ultimately, the failure rate, as this translates into increased chances to miss the temporal deadline or collide with the environment. However, in the other cases, the random strategy is less effective and performs very poorly for the most challenging case, which is C.



(a)

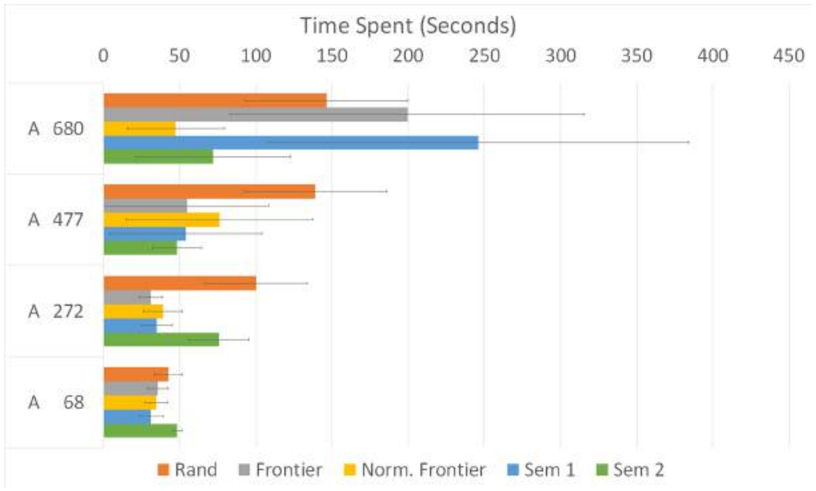


(b)

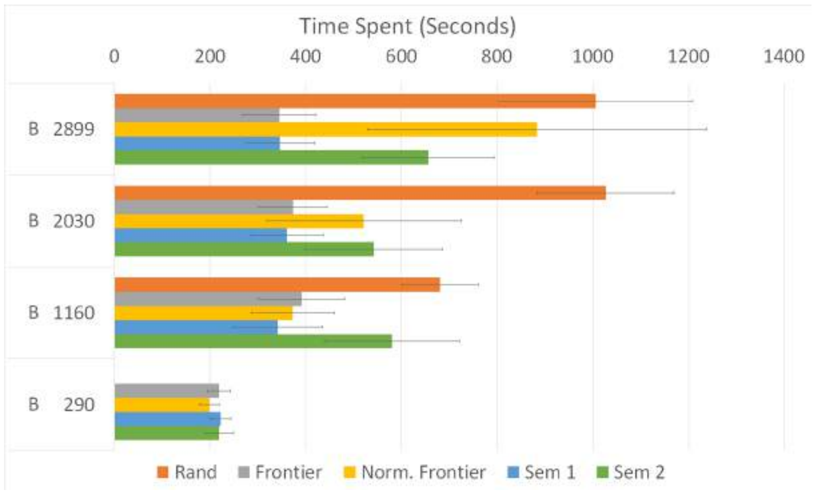


(c)

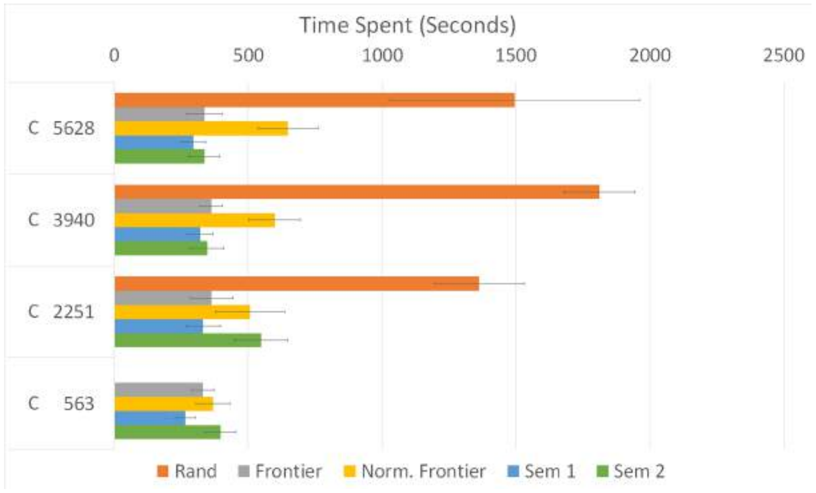
Figure 6.11: The success percentage for all of the exploration strategies.



(a)



(b)



(c)

Figure 6.12: The time spent for all of the exploration strategies.

6.7 Conclusions

In this chapter we have studied an exploration task with temporal constraints. The objective for the robot is to enter an unexplored area and reach a target location within a given temporal deadline. Our system incrementally builds a spatial model called oriented topological semantic map. The model enhances classic topological maps by adding semantic labels (corridor/room) and relationships of the type to-the-right-of and to-the-left-of. Key to our approach is the assumption that walls in the environment are aligned along orthogonal directions. This assumption is very common in most buildings. The proposed spatial model has been coupled with our recently proposed planner based on CMDPs and on top of this we proposed and analyzed five different exploration techniques exploiting the proposed model. Through thousands of simulated runs, it appears that our method dubbed topological frontier with normalized distances works best. In their current formulation, methods on semantic information are slightly less competitive, although it ought to be acknowledged that only very limited semantic information was used. Finally, we have provided all resources to ensure a third party to fully replicate our results – a first in the are of replicable robotics.

In the next chapter we will show the implementation of our method on a Pioneer P3AT robot closely resembling the one used in the simulation. Through a computer vision pipeline exploiting a pre-trained deep constitutional neural network, we are perfecting the software component indicated as IDS. In particular, the network can distinguish corridors from rooms.

Chapter 7

OTSM Implementation

7.1 Introduction

In the previous chapter, we introduced Oriented Topological Semantic Maps (OTSM). We also presented and simulated a solution to explore an unknown indoor structured and orthogonal environment, while we built an OTSM. We also created a framework composed by individual functional blocks that is flexible and reusable by other researchers, and can be used to replicate our own results.

In this chapter, we demonstrate how this open-source framework can be used in a real robot. We focus on how we can implement the three main systems (topological, semantic, and orientation) to create a functional OTSM. In this particular work we present one possible implementation for our three systems, but the proposed framework is sufficiently flexible to allow for components to be swapped. We also tackle the problem of loop closure and test the performance of our solution in a real building from the University of California, Merced. The software and hardware used to implement the solution are the Robotics Operative System *ROS* Kinetic, and a Pioneer Robot 3AT.

There are multiple components to be specified to switch from the simulated world to the real world. We need to be able to obtain each of the elements from the OTMS with the real robot. To obtain the orientation O, we fused odometry and inertial information, from encoders and IMUs. We create a ROS node called *Orientation System (OrS)*, that provides a measurement of robot's rotation and assign this orientation information to the OTSM's. To obtain the building's topology T, as we previously explained, we build a graph where each of the locations and intersections correspond to a vertex. We develop an *Intersection Detection System (IDS)* that is capable of recognizing, in run-time, each of the five different types of intersections.

Because the main algorithm incorporates semantics S to label and describe locations in a building, it was necessary to use an automatic system to learn how to recognize each location and associate a label to them. We defined a *Labeling System (LS)*, that uses a residual neural network resNet[75], to visually learn to recognize a location. Using the same resNet, we were also able to recognize already visited locations and detect loops in the environment.

The contributions of this work are:

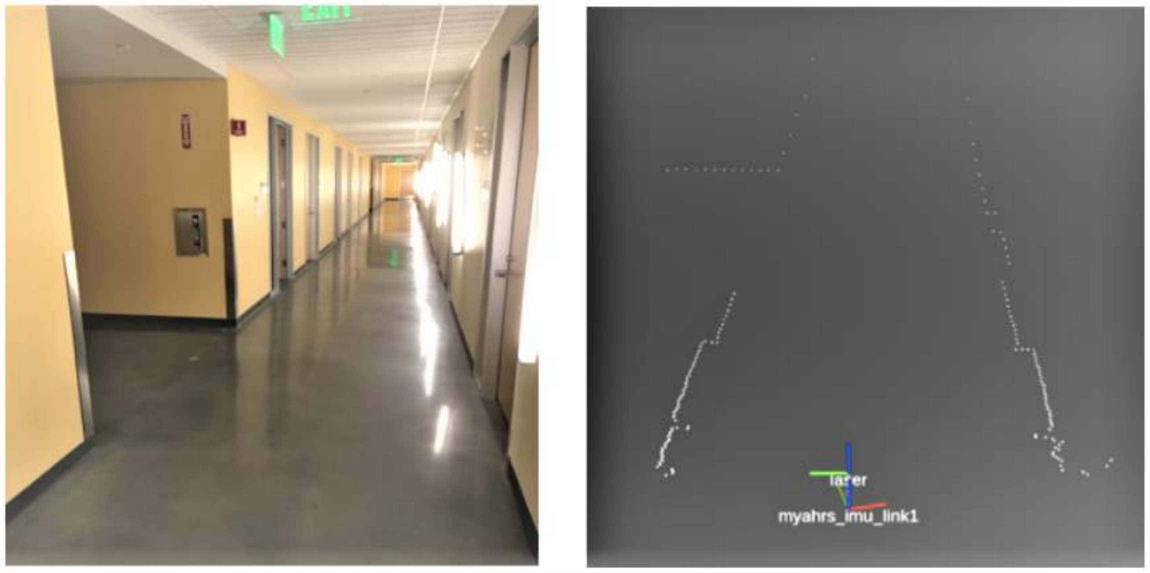
- In section 7.2 we provide an overview of the necessary technical preliminaries to implement each of the systems to extract topological and semantic information from an environment.
- In section 7.2.1 we implement a method to detect intersections using a LIDAR sensor called Intersection Detection System .
- In section 7.2.2 we implement a vision system to learn to recognize locations in a building called Labeling System.
- Section 7.2.3 shows the proof of concept of how to solve the loop closure problem with a regular ResNet network.
- Section 7.3 discusses how we set up the experiments to test the different systems of our robot.
- Experimental results are presented in section 7.4.

7.2 Systems Implementations

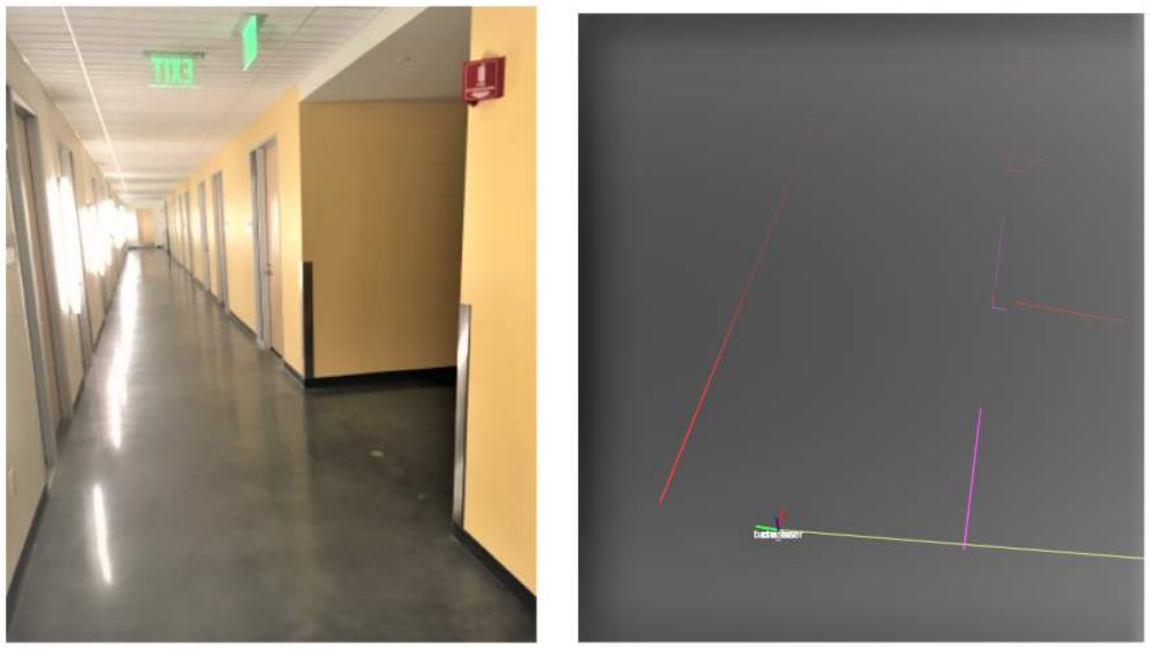
7.2.1 Intersection Detection System (IDS) using 2D LIDAR

As we discussed in the previous chapter, the rooms and locations in a building where the robot can change its trajectory are considered intersections. Depending on the number of possible ways or paths where the robot can go, we define one of the five different types of intersections. To create a system that detects free space, walls and intersections, we proposed the use of a LIDAR sensor together with the method described in [125] and [26] for line segment extraction.

Figures 7.1(a), 7.1(b), 7.2(a), 7.2(b) show the four of the five types of intersections that the Intersection Detection System (IDS) can identify. The fifth intersection is the ”dead end” intersection where there are walls all around the robot. The solution that is presented here is one possible implementation, but ultimately this is a classification problem that can be solved in different ways.

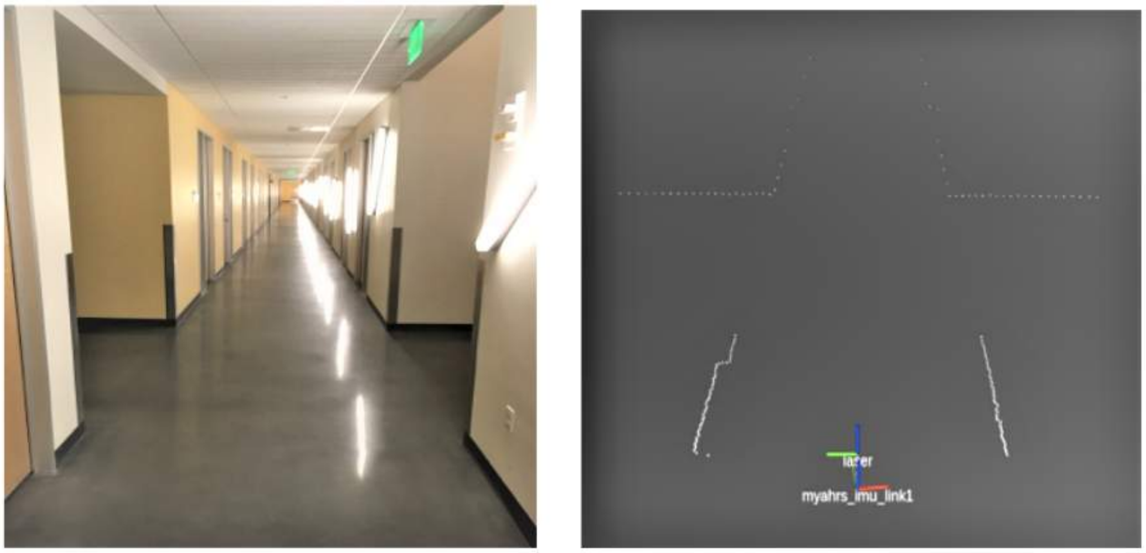


(a)

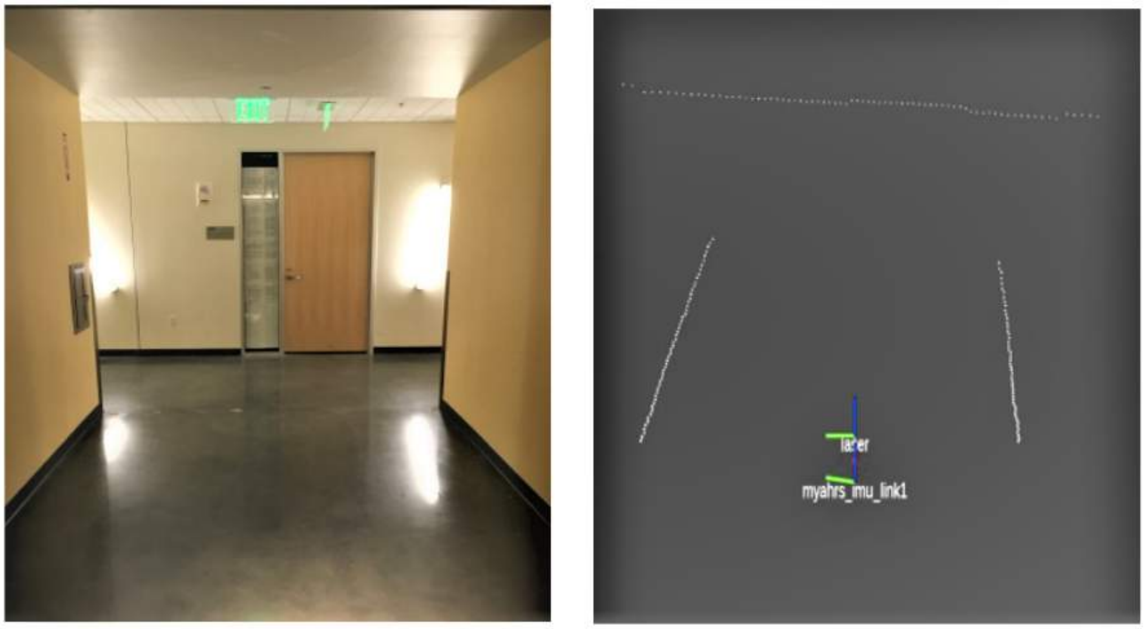


(b)

Figure 7.1: Intersections in the real world and corresponding laser scan as seen in 2D LIDAR. (a):Left Intersection. (b): Right Intersection.



(a)



(b)

Figure 7.2: Intersections in the real world and corresponding laser scan as seen in 2D LIDAR. (a): Four-way Intersection. (b): T Intersection.

The IDS consists of three main functionalities: (a) extracts line segments from a point cloud, (b) establishes relations between different line segments (c) classifies and identifies the midpoint of the intersection. The point cloud is pre-processed by converting the points represented in polar coordinates to Cartesian coordinates and the rupture points are identified.

A point is considered as a rupture point if:

$$r_{i+1} > r_{max} \; and \; r_{min} <= r_i <= r_{max} \tag{7.1}$$

$$r_{min} <= r_i <= r_{max} \; and \; r_{i-1} > r_{max} \tag{7.2}$$

Where r_{max} and r_{min} are fixed thresholds to define the distance between points. All the points between two rupture points are processed to identify the break-point, which is defined as a discontinuity during the laser measurement, and extract the parameters of the line segment (start and ending point of the line segment, see figure 7.3). These break points are identified by comparing two successive laser scans. This means that i and $i-1$ are the break points if:

$$\|(r,\phi)_i - (r,\phi)_{i-1}\| > r_{i-1} fracsin\Delta\phi sin(\lambda - \Delta\phi) + 3\sigma_r \tag{7.3}$$

where $\delta\phi$ is the laser angular resolution, λ is an auxiliary constant parameter and σ_r is the residual variance.

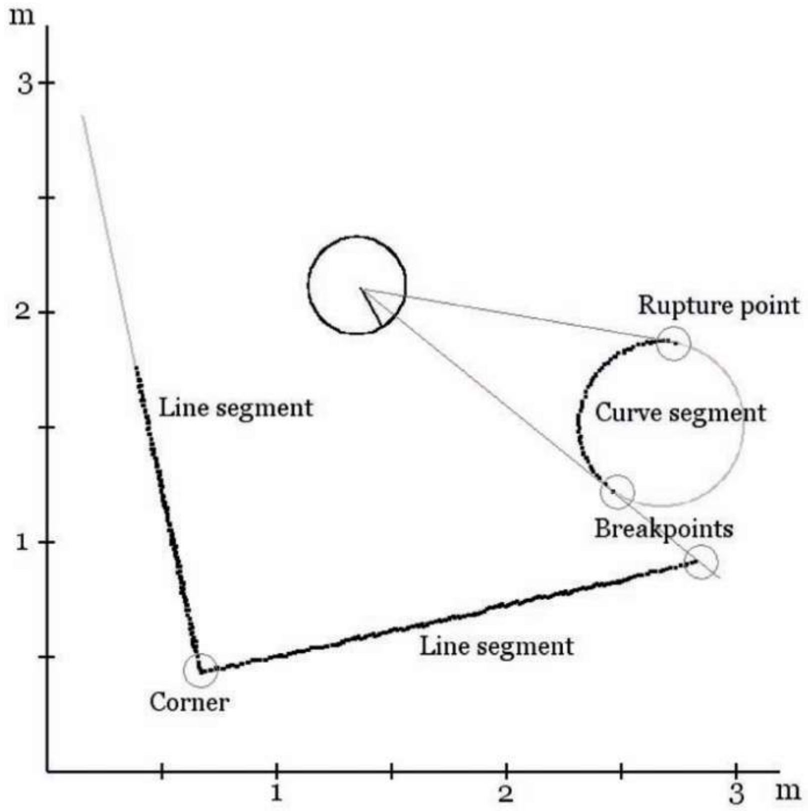


Figure 7.3: Visualization breakpoints, line segments obtained from a single laser scan using a 180 degrees LIDAR. Taken from [125] .

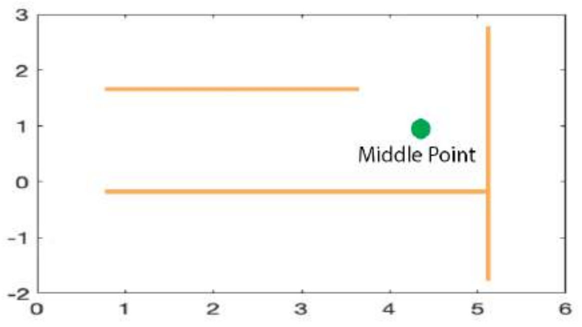
Once the break points are identified, each set of points are again processed to identify if they belong to a line segment using the following algorithm:

```
Result: List of line segments between $bp_1, bp_2$
1  $line_{list} \leftarrow \emptyset$;
2  while $bp_1 \geq i \leq bp_2$ do
3      $K_f[i] \leftarrow 0$ $K_b[i] \leftarrow 0$
4  while $ecd(i, i + K_f[i]) \geq rd(i, i + K_f[i]) - U_k$ do
5      $K_f$++;
6  while $ecd(i, i + K_f[i]) \geq rd(i, i + K_f[i]) - U_k$ do
7      $K_f$++;
8  Compute forward vector $\vec{f} = (x_{i+K_f[i]} - x_i, y_{i+K_f[i]} - y_i) = (f_{x_i}, f_{y_i})$;
9  Compute forward vector $\vec{f} = (x_{i+K_f[i]} - x_i, y_{i+K_f[i]} - y_i) = (f_{x_i}, f_{y_i})$;
10 if $\Theta_i \leq \Theta_{min} || (\Theta_i - \pi) \leq \Theta_{min}$ then
11     $start \leftarrow x_{K_f[i]}, y_{K_f[i]}$;
12     $end \leftarrow x_{K_f[i]}, y_{K_f[i]}$;
13     $line_{list} \leftarrow line_{list} + (beg, end)$;
```

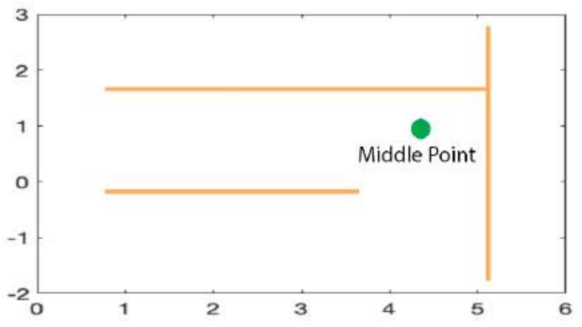
Algorithm 7.1: Compute line segments between any two break-points bp_1, bp_2

The list of line segments extracted is traversed clockwise in the order the laser scan was processed and the relation between the adjacent line segments are established and placed in a de-queue, whose order is used in determining the type of intersection immediately in front of the robot. For the case of the left and right intersection, it will depend on the orientation of our robot. However, we can add one more if we count rooms and no-end locations where there are no outgoing paths but only a dead end.

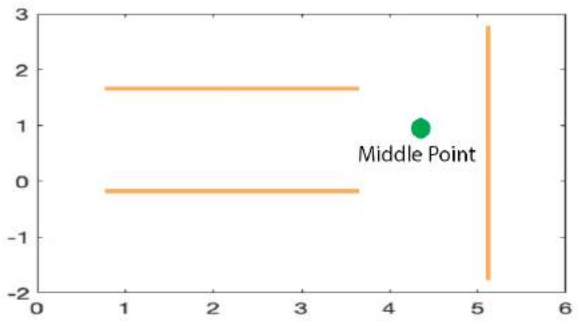
The lines are considered perpendicular to each other if the angle between the two line segments is $\pi/2$, plus a tolerance value, and parallel if the angle is either 0 or π. The adjacent line segments are then checked if they intersect or are separated by a distance. Based on the relation between the line segments, the intersections are classified into various types, whose line segments serve to characterize them. Figure 7.4 shows the four types of intersections with middle points.



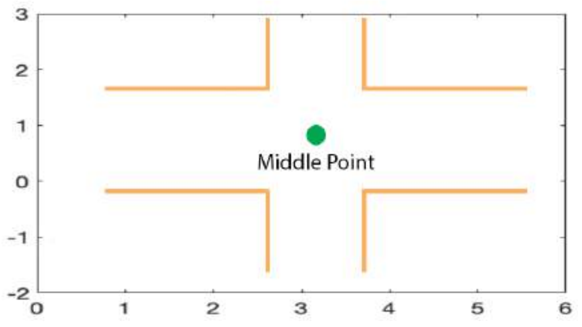
(a)



(b)

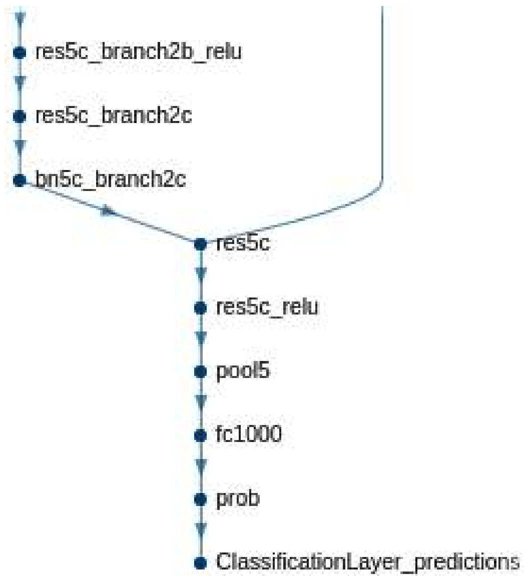


(c)

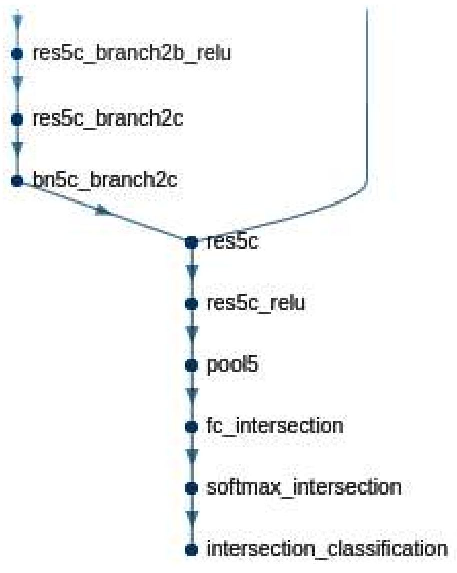


(d)

Figure 7.4: Example line segments identified and middle points for a) Left intersection, b) Right intersection, c) T intersection, d) Four way intersection.



(a)



(b)

Figure 7.5: Last three layers of resNet-101 changed.

The midpoint in the laser frame is computed by using the parameters of an intersection (end point of the first(right) line and the beginning point of the last(left) line) previously identified:

$$(x,y)_{mid} = ((x+\sigma_x, y+\sigma_y)_{rightend} + (x+\sigma_x, y+\sigma_y)_{leftbeg})/2 \qquad (7.4)$$

Here σ_x and σ_y are fixed thresholds that needs to be adjusted based on the average space of the gap/door between the walls. The midpoint, or from now, anchor point, is then transformed into the global frame, and used as the center of the intersection.

7.2.2 Labeling System (LS) using 2D images

Once an intersection and its middle point is identified, it is necessary to learn to visually differentiate this intersection from others. Because we want the robot to be able to learn and identify, using semantic labels, different locations in the building, we use a deep neural network resNet [75] implementation with 101 layers to learn and store the labels of each of these locations. We use the resNet provided in MATLAB as part of the deep learning toolbox. This neural network is pre-trained with the database obtained from [47]. Once the robot reaches the anchor point of the intersection, the robot is rotated in place and starts capturing images using a RGB camera.

7.2.3 Loop closure problem

The loop closure problem, as we mentioned in the section 2.3.3, is an important problem that a majority of the SLAM and VisualSLAM algorithms try to solve.

The objective is to recognize if a certain point in an environment corresponds to a previously-visited location. If this identification fails, we are creating a disconnected map that, with a graph representation, will create new vertices infinitely every time the same loop is visited.

To elaborate in our particular case, let's say the robot starts and reaches a first intersection to be learned. Information about the images of this intersection are collected and the last three layers: fully connected layer, softmax layer and classification layer, are replaced to reflect just one label and classified into this one label. The robot moves further and reaches a second intersection. Then the network is re-trained through the same process and now the network is able to identify two intersections. Note that the first two intersections are always considered new. Once the robot traverses to the third intersection, the same procedure of capturing the images and training the network is performed. If in this case, the confidence or accuracy of the network reduces below a pre-defined threshold, the third intersection is assumed to be one of the previous ones. The images are the input to the old trained network and the newly trained network is discarded, the label provided by the old net is the label for the intersection to which the robot has traversed to. If the confidence is more than this aforementioned threshold, the intersection is considered new and the newly trained network is retained. This provides a functional approach to solve the loop closure problem using 2D images. These thresholds were selected after running a series of preliminary tests and analyzing the thresholds that gave constant success recognizing the locations.

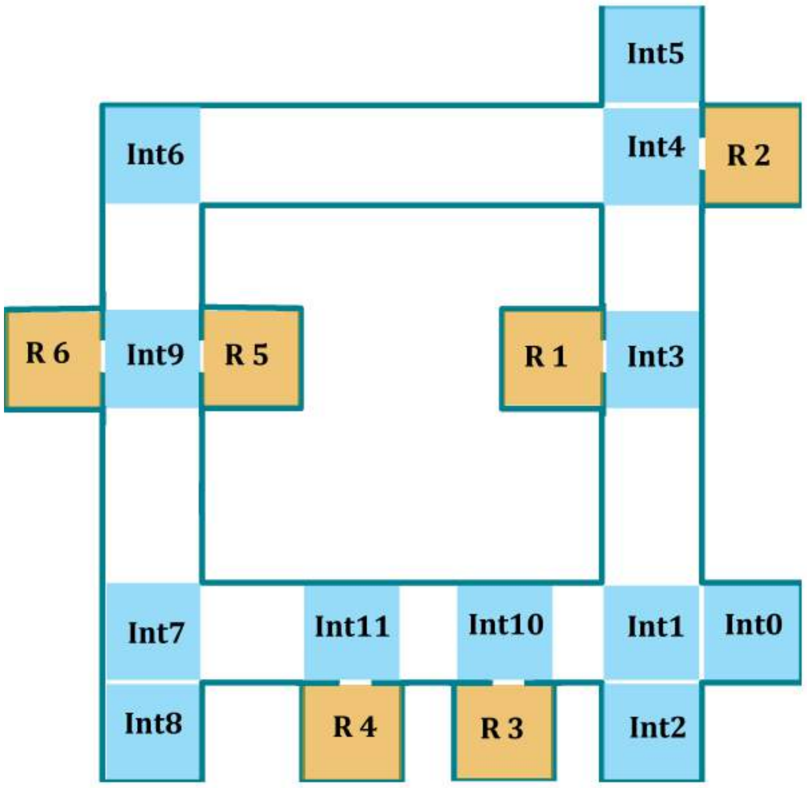


Figure 7.6: Sketch of the floor at the University of California Merced, with intersections (blue zones) and rooms (Orange zones).

Figure 7.6 shows the real case we are dealing with when we have a loop. In this case, after the robot learns the intersections Int 1, Int 4, Int 6 and Int 7, it must recognize them every time the robot visits them again. Otherwise if one of these intersections is labeled incorrectly the loop would break and the topology would be incorrect.

7.3 Experimental Setup

In this section we present how we setup the experiments to test the different systems on our robot. We go over the main characteristics and parameters of the robot, the environment, the semantic, the topological and orientation system.

We use a Pioneer robot 3AT, similar to the one we used in simulation, that is controlled by the ROS 1 package RosAria. This robot has a Lidar, infrared sensors on the front and on the back, a RGB camera, and four wells encoders. We installed a Dell Latitude 7280 with ROS Kinetic to help run many nodes directly from our framework ***Download Sources Here.*** [1] . These nodes control the robot, run the

[1] Available on IEEE xplore: `https://ieee-dataport.org/open-access/`

CMDP solver, the OTSM builder, extract the semantic information, the topology from the environment, and orientations for the vertices and edges. Figure 7.7 shows the robot.

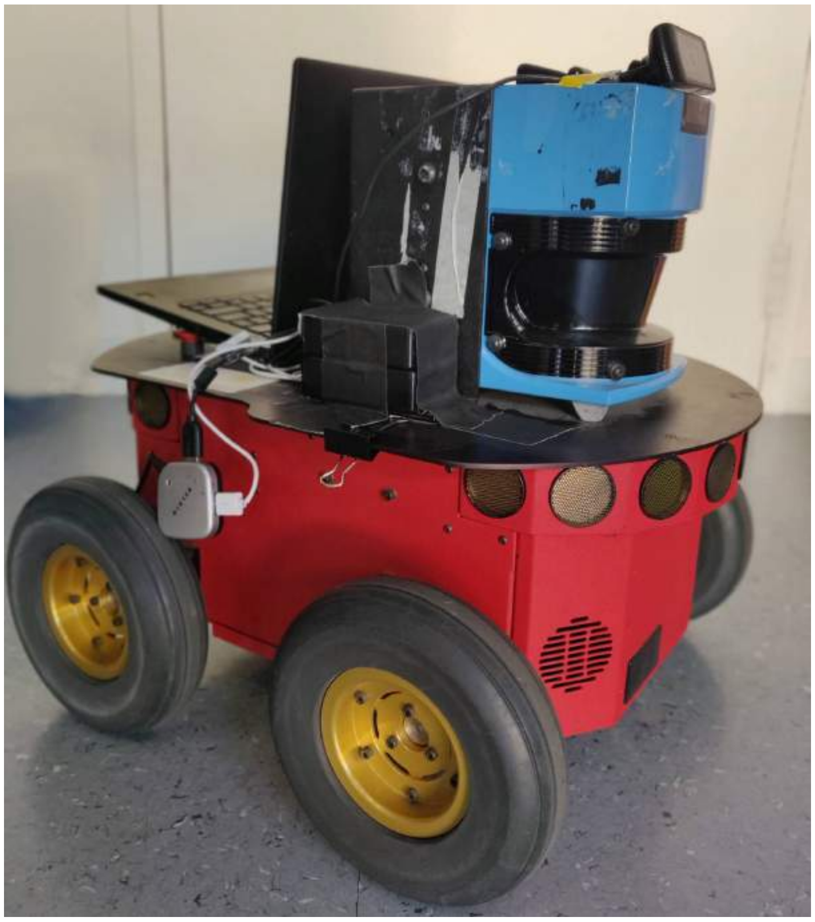


Figure 7.7: Pioneer Robot 3AT mounted with a Laptop, a LIDAR, a RGB camera and two IMU um7.

We chose a real building from our campus at UC Merced, with an orthogonal environment with multiple rooms, and corridors to test our solution. Figure 7.6 shows the sketch of the floor where six rooms were used for the tests in a closed loop with four corridors and all five types of intersections.

We used a standard 1080p Logitech camera to acquire 2D RGB images of a location. In order to get consistent intersection identification results, we used an in-place rotation plan for the robot to collect images. This helped in identifying the features of a particular intersection when the robot enters an intersection in any of the directions pertaining to that location. We also set the learning rate at 0.001 gradient step. Even though the time required for training an intersection was high, a smaller learning rate led to lower accuracy. The number of images collected at each

`time-constrained-exploration-using-toposemantic-spatial-models-reproducible-approach`

Inters	W-E (%)	E-W (%)	N-S (%)	S-N (%)
Int1	100	100	90	100
Int2	NA	100	NA	NA
Int3	100	100	80	NA
Int4	100	100	100	90
Int5	100	NA	NA	NA
Int6	100	NA	NA	NA
Int7	100	100	NA	80
Int8	NA	100	NA	NA

Table 7.1: Average performance for the intersection detection system over 10 runs for 8 different type of intersections.

location was set at 84 with a split of (60/20/20) for the train, validation and test set. Again this provided the desired accuracy of 90% - the threshold that we used to distinguish a location that was identified earlier. We could improve these numbers if one were to spend time fine tuning everything, but this is not the focus of this work.

A 2D LIDAR SICK LMS200 was used for the intersection detection node. Based on the resolution of the LIDAR of 0.25 deg and the average width of the corridor being 0.75m, the threshold used to identify the rupture points was set to $0.1m$, threshold used to identify a breakpoint was set to $0.25m$ and the Θ_{min} used to consider a particular point part of the line was set to 10 deg.

To obtain the orientation for our robot, we use two IMU um7 connected through USB ports and the data is fused together with the odometry from the robot. For the data fusion we use the *robot_localization* package from ROS, that implements an extended Kalman Filter using the *ekf_localization_node*.

7.4 Results

Here we evaluate the performance of our solution. First, we measure the effectiveness of our *Intersection Detection System (IDS)*. For this purpose we tested different intersections in our building section 80 times, varying the orientation of the robot from where the intersection could be identified. Table 7.1 shows the percentage of success recognizing the correct intersection. We reached more than 90% rate of success for all five types of intersections. Because the system is based on a LIDAR that cannot detect transparent surfaces correctly, we had to cover some windows and glass to avoid incorrect detection.

Table 7.2 shows the six different columns that describes the performance of our *Labeling System (LS)* and its ability to differentiate locations already visited and new locations. Two different sequences were produced randomly to analyze the general performance, one with nine steps and a second one with eleven steps. For each step an intersection is analyzed as either visited or unvisited and then, the neural network assigns a new label if it is a new intersection or returns a previously learned label. On the first column we show how incrementally we save pictures for each of the intersections. Because the number of pictures is proportional to the Neural Network training time, the second column shows an increase over time of different steps. The last two columns show the final label and the confidence of this result. We accepted, as a valid label, the result when the confidence level is over 80%. A lower confidence level increases the uncertainty about the label, however, even for lower confidence levels the final label is the one we are expecting.

We observed during the tests that even when the resNet performs relatively well most of the time, it is sensitive to changes in the environment; people walking, changes in the light, or changes of the surrounding objects, creates a higher uncertainty and the confusion matrix spreads the values outside the diagonal. This needs to be addressed when a dynamic environment is used and a different Labeling System must be used. A possible solution would be to use the previously mentioned solution from [174].

Finally, we observed during all the tests that the orientation was always correct. This is due to the fact that we are dealing with a very wide range of angles to define if it is N,S,E,W. The Odometry and compass errors were not significant during the experiments.

We are aware that the size and extension of these experiments can be improved and indeed they were planned and scheduled. However, due to unforeseen circumstances and closure of the campus, we lost access to the facilities.

7.5 Conclusions

In this chapter we presented an implementation with a Pioneer Robot of the different systems showed in simulation in our former work [159]. Our solution is able to recognize intersections with a accuracy of 100% based on the tests. However a wider set of tests could be done to expand these results. We also showed that it is possible to use a modified resNet architecture to classify visually locations in a building and even solving the problem of loop-closure with confidence rates over 75.5%. This last value definitely can be improved in an environment with more unique characteristics at each location. The similarity between locations definitely played against the success of the Neural Network.

Equally, we proved that the architecture used for simulation can be used with a real robot and it is only necessary to modify a few ROS nodes to create an OTSM with any implementation of an IDS, LS or OS. We hope this flexibility can be exploited by the robotics community to try difference approaches and test different vision systems that improve our initial solution. Finally, we expect this framework to be used to test more in depth, different exploration and navigation strategies with real robots that use ROS as their main operative system.

Step	Inters	# Img	Train. Time (min)	Label	Confidence %
1	Int1	84	1.29	Int1	100
2	Int10	84x2	4.21	Int10	100
3	Int7	84x3	4.31	Int7	96.3
4	Int8	84x4	11.18	Int8	100
5	Int7	84x5	14.54	Int7	80
6	Int1	84x5	14.59	Int1	98
7	Int3	84x6	19.34	Int3	93.94
8	Int4	84x7	26.05	Int4	97.14
9	Int3	84x8	39.15	Int3	88.5
1	Int1	84	1.29	Int1	100
2	Int2	84x2	4.41	Int2	100
3	Int1	84x3	7.58	Int1	60.61
4	Int1	84x3	7.23	Int1	78.12
5	Int2	84x3	7.23	Int2	87.88
6	Int2	84x3	7.36	Int2	75.76
7	Int1	84x3	7.23	Int1	100
8	Int7	84x4	10.49	Int7	100
9	Int8	84x5	13.39	Int8	94.23
10	Int7	84x6	17.51	Int7	90.48
11	Int8	84x6	22.15	Int8	77.78

Table 7.2: Performance of the accumulative training with multiple steps. It shows the number of images taken, the time to train the Neural Network, the final identified label and the light conditions.

Chapter 8

Map Merging of Oriented Topological Semantic Maps

8.1 Introduction

In previous chapters, we dealt with multi-robot algorithms for navigation under temporal and failure probability constraints. We also developed our own type of map, OTSM, to be able to use it in run-time with our CMDP model. In this chapter we are interested in expanding the usability of our OTSMs when, instead of one robot building a map, we have a team of robots. In this chapter we will go over one of our published papers called ”Map Merging of Oriented Topological Semantic Maps” [158] that describes the solution for this problem.

Multi-robot research has been on the rise because, in many instances, multiple robots offer inherent advantages over solutions relying on a single robot. Continuing with our exploration task, we can intuit that a group of coordinated robots would explore an unknown environment much faster than a single robot. This is a specific problem where the multi-robot solution introduces new challenges not found in the single robot approach. For individual robots, continuous progress in SLAM research has generated sophisticated solutions and, in some instances, this can be considered to be a solved problem. However, when a group of robots cooperatively explore and map an unknown environment, two approaches can be undertaken to combine the partial results. The robots can either jointly build a spatial model “on the fly,” or they can individually build a single map, and then combine their partial models together a posteriori.

This last problem, known as map merging, is not a direct extension of a single robot problem and is tackled in this chapter. Map merging relates to other problems such as structural graph matching or sub graph isomorphism, depending on the type of model considered [61]. The most common practice is to create metric maps and use different techniques to merge them together [31, 100, 169, 28]. However, when the maps utilize a topological or semantic representation, a different approach is needed. In chapter 6, we introduced a new type of maps called Oriented

Topological Semantic Map (OTSM). The purpose of this chapter, in particular, is to study how multiple partial OTSMs can be combined when a team of robots cooperatively explore a common indoor environment. Our solution is inspired by the Warrington's object recognition model [180], a cognitive model that describes how humans recognize objects using a two layer system of perceptual and semantic categorization.

In their everyday activities, humans do not rely on metric maps to complete their tasks. Instead, a more concise representation to store the structural organization of buildings or other relevant structures is computed quickly and shared, arguably with little effort, with other humans for immediate use. Even in the absence of visual information, humans can still use the same efficient representation and interact with their environment and peers [101]. As we expect robots to operate side-by-side with humans, it makes sense to envision the same capabilities for robots, both for robot-robot inter-operation, and to smooth the human-robot interface.

In the past, some attempts to merge topological maps have been considered, but due to the novelty of OTSMs this is the first bio-inspired model used to merge a these maps. Although, some authors like Huang et al. [77] or Bonanni et al. [22] show similar solutions for merging topological maps, and Erinc et al. [14] studied how to combine hybrid maps including some visual or semantic information, we offer here a method applied to OTSMs, where we expande the topological-semantic map merging with orientation information.

We can classify map merging solutions into two different categories: an online and an offline map merging. In the first case the challenge is to merge two or more maps while they are being built. Complex and memory consuming maps, like metric maps, are difficult to share and many considerations need to be taken into account to assure the correct transmission of the information either as a centralized architecture, where all the robots send their discoveries, or a distributed architecture, where each robot communicates with the rest of the team. The second type of map merging is offline, where the sub-graphs or sub-maps are stitched after the robots complete a distributed exploration of the environment. In this chapter we focus on the second, where we will process partial maps after they were generated by the robots.

The contributions of this chapter are:

- We propose a new two-stage method to compare vertices in different OTSMs and measure their resemblance.
- We present a new merging technique to stitch OTSMs using a semantic and perceptual categorization.
- We study (in simulation) four different types of errors that affect OTSMs and their impact when merging together partial maps.

8.2 Inverse Warrington's Object Recognition Model (IWORM)

The model presented by Warrington [180] inspires our strategy for merging together two or more OTSMs. Warrington hypothesized two post-sensory categorical stages that work together for object recognition (see figure 8.1). This research showed evidence of how different patient groups presented a deficit in recognizing objects because of left-posterior and right-posterior cerebral lesions. These patients were suffering from different levels of visual agnosia, defined as [180], "the inability to recognize or identify common objects that cannot be accounted for by sensory impairment or more generalized cognitive deficits". Warrington's hypothesis of how the human brain processes the visual information proposes that, although both sides of the brain do a visual analysis, the right side of the brain's job is to judge the matching as same or different stimuli, while the left side's function is to match objects to pictorial representation for an a posteriori word matching. Visual stimuli are interpreted by the brain to form a shape/contour of an object. This shape can match with at least one shape already stored in the memory that will eventually receive a semantic label that corresponds to a word that has a meaning or significance.

Starting from this work, we posit that in order to effectively evaluate and measure the similarities between two sub-maps and find their matching vertices, we require a two-sided function that can sequentially process their perceptual and semantic information. Inspired by how our brain finds the match of shapes/contours and assigns semantic labels to them, we propose to match the shapes and labels of different sub-maps to score their resemblance. Then, we will use the quality of the resemblance to then merge them together. Since OTSMs embed semantic information in our graphs, we propose to invert the information flow from a semantic to a perceptual categorization, instead of perceptual to semantic as the original Warrington's work proposes for humans (see figure 8.2). Specifically, we will take the semantic categorization as the comparison of labels and orientations, followed by the perceptual (topological) categorization, that will compare the shape of the graphs, analyze the number of neighbors and their respective connections with other vertices in the map.

8.3 IWORM Inspired Map Merging of OTSMs

When considering spatial models featuring topological components, map merging can be referred to problems like graph structural matching, and sub graph isomorphism. However, for our problem these approaches cannot be applied directly due to the fact that OTSMs extend the basic graph structure embedding semantic information into the topological map. When merging topological maps, the objective is to obtain a complete map of an environment by stitching together multiple sub-maps. We now

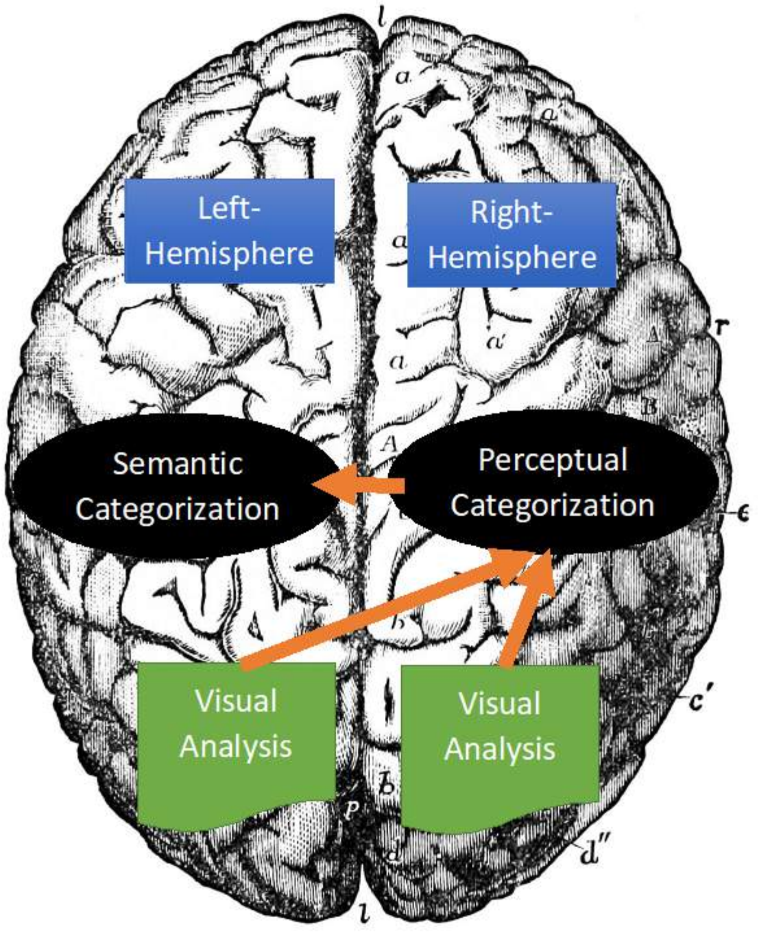


Figure 8.1: Diagram of the Warrington's Object Recognition Model inspired by the [180]

describe how multiple OSTMs can be consolidated into a unique map.

Definition 1. A sub-map of certain environment is represented by a graph $g_i = (V_i, E_i) \subseteq G = (V, E)$, whose vertices V and edges E can be compared with vertices and edges of other sub-maps using a scoring function called IWORM. This will serve to determine the likelihood that a vertex v_i in g_N corresponds to a vertex v_j in g_M (with $N \neq M$).

Without loss of generality, in the following, we consider the case where just two sub-maps must be merged. When three or more maps must be combined, subsequent pairwise mergings can be used. Definition 1 establishes that two sub-maps can be compared and the IWORM function (to be define later) can help to determine correspondences between common vertices. We say that two sub-maps overlap if they share one or more common vertices. These shared vertices are then used to establish a fully connected map. As for other map merging problems, if

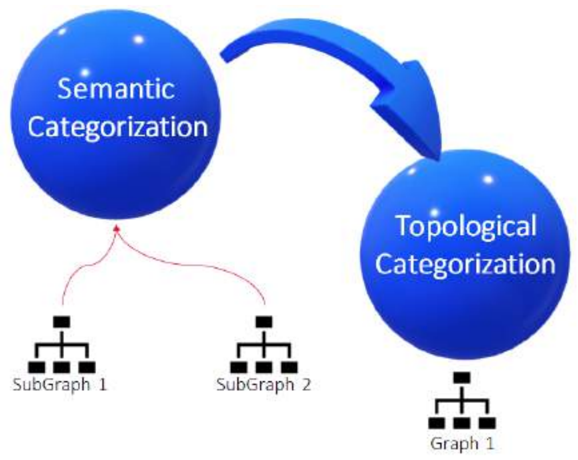


Figure 8.2: Diagram of the Inverse Warrington's Object Recognition Model

there is no overlap between the two sub-maps, then no merging can occur and this situation must be properly detected and handled. While an initial analysis of this case will be presented in the results section, it is still an open question to find a robust way to deal with this problem.

As presented in section 8.2, we aim to find a map matching algorithm following a method inspired by the same cognitive process that our human brain does. IWORM is a pattern classifier that uses two types of inputs to match two given sub-maps. The first layer, called Semantic Categorization, compares two sub-maps g_1 and g_2 in terms of the semantic label and the orientation of each of the vertices. This is a high level matching providing only a limited level of differentiation between vertices. However, as it will be presented later, there are different sources of errors that make it too brittle to rely only on these two features to classify and match vertices. To mitigate this limitation, a second layer is introduced. Perceptual Categorization compares the local structure of the sub-maps. The structure we are looking to match will be the topological properties describing the connections between vertices. Specifically, we examine how many neighbors a vertex has, and how these neighbors, in turn, are connect to others. This structure can be described in terms of degrees of depth, where level one corresponds to the immediate neighbors of a certain vertex, level two corresponds to the neighbors of the immediate neighbors, and so on. Algorithm 8.1 describes the process to merge two sub-maps g_1, and g_2.

The main process iterates over each of the input sub-maps. First, it tries to find correspondences between vertices. To this end, it makes a complete pairwise comparison between all vertices[1]. Note that the first vertices of g_1 are matched against vertices of g_2 and then vertices of g_2 are matched against vertices of g_1. This is because the IWORM function is not symmetric, i.e., for $v_i \neq v_j$ in general $IWORM(v_i, v_j) \neq IWORM(v_j, v_i)$.

[1]This step is typically not time-consuming because topological representations are compact and maps have a small number of vertices.

```
1: Algorithm $mergeGraphs(g_1, g_2)$
2: for all Vertices $v_i$ from $g_1$ do
3:    for all Vertices $v_j$ from $g_2$ do
4:       Get $[matchingPairs_{g_1},\ scoresPairs_{g_1}]$ from $IWORM(v_i, v_j)$
5:    end for
6: end for
7: for all Vertices $v_j$ from $g_2$ do
8:    for all Vertices $v_i$ from $g_1$ do
9:       Get $[matchingPairs_{g_2},\ scoresPairs_{g_2}]$ from $IWORM(v_j, v_i)$
10:   end for
11: end for
12: mergeMaps($g_1$, $g_2$);
```

Algorithm 8.1: Merging OTSMs

```
1: Algorithm $IWORM(v_i,\ v_j)$
   ★ {$Semantic\ Categorization$}
2: $Score_L \leftarrow \mathcal{L}(v_i)$ is semantically similar to $\mathcal{L}(v_j)$
3: $Score_{VD} \leftarrow \mathcal{VD}(v_i)$ is equally oriented to $\mathcal{VD}(v_j)$.
   ★ {$Perceptual\ Categorization$}
4: $Score_{NE} \leftarrow \forall\, e_i \,\exists\, e_j$ Number of edges comparison.
5: $Score_{ED} \leftarrow \forall\, \mathcal{D}(e_i) = \mathcal{D}(e_j)$ Edge's direction comparison.
6: for all Vertices $v_k^i$ do
7:    for all Vertices $v_l^j$ do
8:       if $max_Depth$ is reached then
9:          Create matching pairs with the maximum scores.
10:         return $matchingPairs$, $scoresEdges$
11:      else
12:         Get $scoresEdges$ from $IWORM(v_k^i, v_l^j)$
13:      end if
14:   end for
15: end for
```

Algorithm 8.2: IWORM scores the resemblance of a pair of vertices $v_i,\ v_j$.

Because we assume that in the acquisition of each map there may be errors, the labeling, orientation and neighbors cannot be assumed to be error free. Consequently, we first create pairs of possible matching vertices and for each pair, a score is assigned. To do this, the IWORM function performs the local analysis in the graphs to give a value for each correspondence found when comparing labels, orientation, and the number of neighbors for a certain degree of depth, that will be studied in the numerical validation section. If the semantic label is the same for both vertices, the score will be K, and if the label is different it will be $-K$, where K is a preassigned positive constant (Algorithm 8.2, Line 2). A similar binary score is assigned when comparing the orientation between the two vertices. This will be K when both vertices share the same orientation or 0 otherwise (Algorithm 8.2, Line 3).

In this case a mismatch receives a 0 score, rather than $-K$, because as we established previously, the orientation of each vertex is relative to the robot's orientation and will only be used to add a bias component that will increase the score when two vertices were discovered the same way. For the number of edges outgoing from a vertex, the score is assigned with the following function $Score_{ED}(v_i)$, where diffEdges is the absolute value of the difference between the outdegree of the vertices. The effect of changing the K function will be the subject of future research:

$$\begin{cases} K & \text{if } diffEdges = 0 \\ \left(1 - \frac{diffEdges+1}{K'}\right) \cdot K & \text{if } diffEdges \leq 2 \\ -K & \text{otherwise} \end{cases}$$

Finally, for line 5 of algorithm 8.2, a score of K is assigned if the number of edges and orientations are the same in both vertices. From line 6, for each of the vertices v_i and v_j we score their neighbors recursively, calling the same function IWORM on the neighbors k and l, from vertices v_i and v_j, noted as v_k^i and v_l^j, respectively. The recursive calls to IWORM stop after a fixed number of recursive levels to ensure that the analysis remains local.

After scoring the vertices of each sub-map, the next step is to validate if a label is found in two different locations, understanding the location as where the vertex is in the topological map and its neighbors.

Finally, we need to merge the two sub-maps, in line 8 of Algorithm 8.1. For this purpose, we need to chose which pair of vertices have the same resemblance and, in this case, decide which information to accept. When we obtain the scores matching g_1 to g_2 (lines 2 and 4), and g_2 to g_1 (lines 5 to 7), we are adding each of the individual K values after comparing label, direction and edges of each pair of vertices, that will result in a final score that is compared with the highest possible score of K. If the final score corresponds to more than a threshold γ, which we set to 80% of the highest K, we assume that those two vertices are the same. By the contrary, if a pair of vertices is matched with γ less than 80% of the highest score, it means that the semantic and topological information between those two vertices has some discrepancies (it can be the labels, directions, or the edges). We chose 80% as our threshold to try to evaluate the algorithm with a high error rate of 20%, similar to the one we used for our tests. In the case where the semantic labels are different, we cannot be certain which one is the correct one, so we arbitrarily pick the label from g_1 and save the label from g_2 as an optional label. This optional label list will be used when merging new maps to the existing final map that, perhaps, can confirm which label is most likely the correct one. If the direction differs between vertices, again, we pick the direction from g_1 and save the direction from g_2 as optional. For the edges we combine the information from g_1 and g_2 together, where the outdegree of the vertices from v_1 defines how many possible vertices can be connected. We then complete this list with the available information about the visited vertices that connect with v_1 and v_2.

8.3.1 Sources of error

There are at least four different types of errors that can happen when building OSTMs. Because we are not using metric maps, there are no errors associated with rotation, alignment, or scale. However, we still have a possible translation problem: the location of the same vertex can be different due to an incorrect labeling. To address this issue, we propose to use the score function to calculate the likelihood of where a vertex really is.

1. *Error type 1:* A semantic labeling error, i.e., a vertex is assigned the wrong label when it is visited for the first time, or when it is revisited it is not recognized as the same vertex and assigned a different label (see figure 8.3).

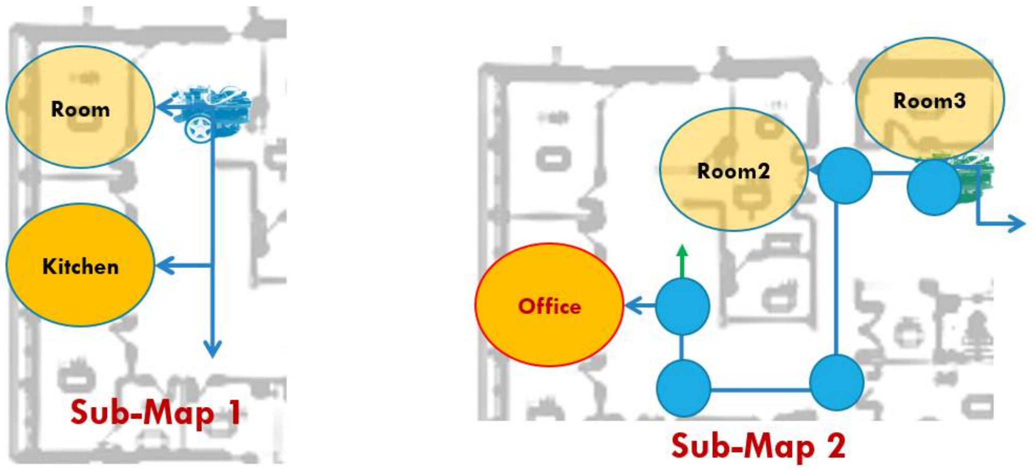


Figure 8.3: Error type 1: A semantic labeling error.

2. *Error type 2:* An error with the compass will lead to an incorrect assignment of the direction of a vertex/edge (see figure 8.4).

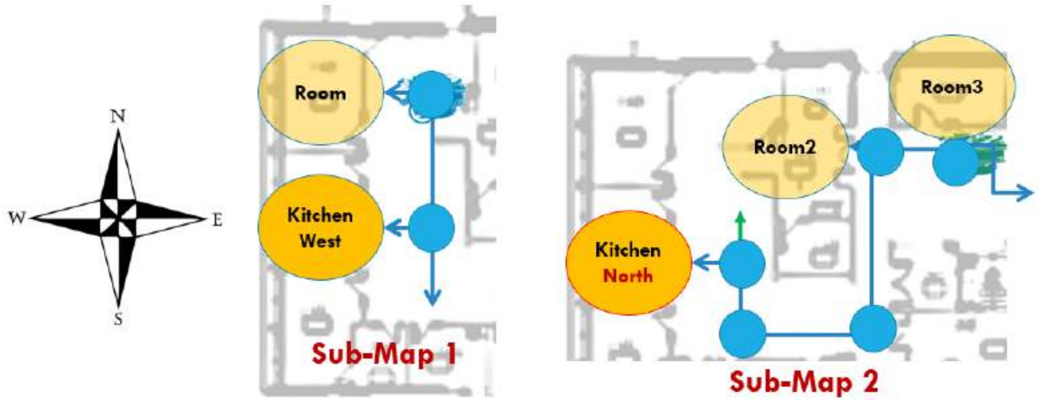


Figure 8.4: Error type 2: An error with the compass.

3. *Error type 3:* A vertex in a sub-map can be mistakenly associated with a wrong degree. For example, the vertex v_1 in g_1 is a four way intersection, but the same vertex in g_2 is detected as a three-way intersection due to a wrong intersection detection (see figure 8.5).

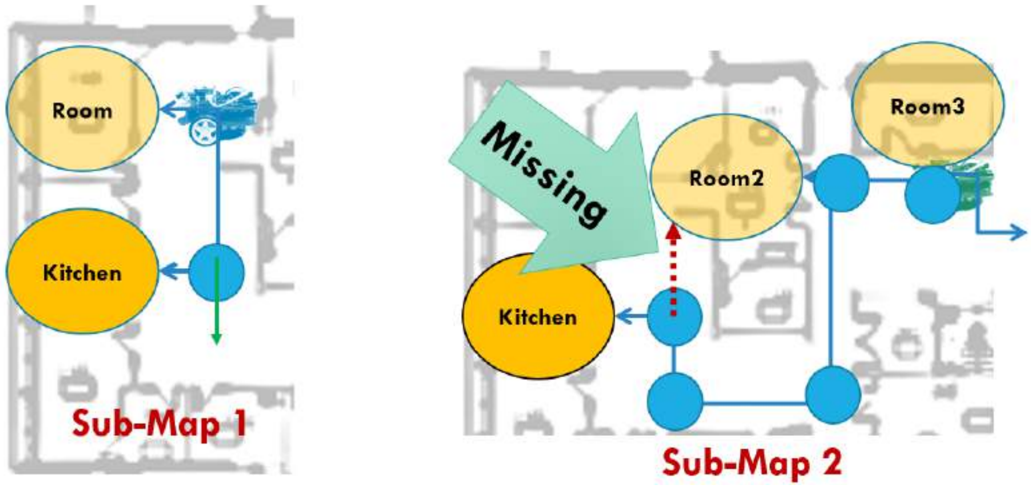


Figure 8.5: Error type 3: A missing edge in a sub-map.

4. *Error type 4:* A vertex v_1 in a sub-map g_1 can be missing, but appear in a second sub-map v_2. This happens when the vertex is recognized like a previous vertex (error type 1), that already exists in the sub-map or because the robot passed by the location and never identified it. This is one of the most serious types of error. If the robot misses a vertex, this means that the neighbors will be connected incorrectly, their directions will not match, or we may have unconnected graphs (see figure 8.6).

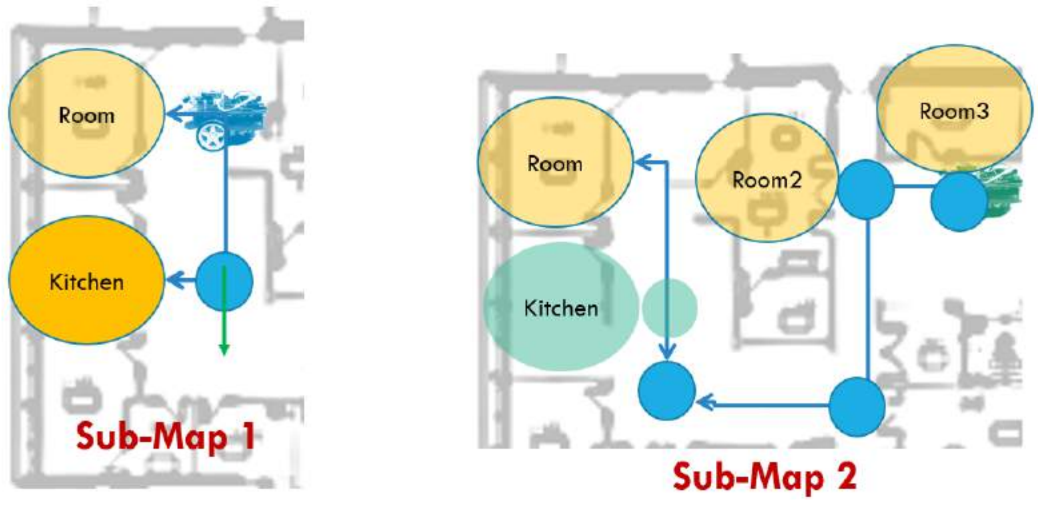


Figure 8.6: Error type 4: A vertex missing in a sub-map.

8.3.2 Understanding Errors

To understand the type of errors, it is useful to discuss how we are implementing the solution on a real robot. While results presented here refer to simulation only, our simulation is setup to mirror the real robot setup. The robot has 2 different algorithms: one to recognize the type of intersection (i.e., the degree of the vertex), and another one to assign a semantic label to each location.

Semantic labeling is performed using a neural network that can be either pre-trained, or incrementally trained as new places are discovered. In our real robot we are using resNet for this task. It is therefore foreseeable that the robot could make the errors we identified earlier on.

- Error type 1: Based on the vision system, the robot mislabels a location, either giving a wrong label from the pretrained list of labels or assigning a label that was previously used for a different location by another robot. Then we will have the same name for two different locations.
- Error type 2: Because the direction is obtained by an IMU orientation errors are possible. However, if a vertex is discovered by two robots from two different directions, this currently does not cause any problem during navigation because the orientation of each vertex only serves as reference to rotate according to the edges directions. A motor, for example, is a common source of this disturbance. Because we will be using a couple of IMUs combined with the robot's odometry and filtered with a Kalman filter, the estimation of the orientation can fail.
- Error type 3: The intersection detection system, is based on a LIDAR sensor, which detects gaps between walls and defines the type of intersection the robot is facing. In the case of the intersection being obstructed by a person or another

object, this will cause a fail, correctly determining the right number of vertices of this intersection.

- Error type 4: When this happens the graphs will not be updated and the new vertex will not be added. There are also cases where the intersection system simply passes by a location and cannot identify it as an intersection, and the robot will just keep moving. This happens when the robot gets too close to one of the walls or its direction is not parallel to the walls and, instead, it is pointing diagonally to the center of an intersection.

8.4 Simulations

8.4.1 Setup

To evaluate the solution we proposed, we studied the problem in Gazebo, so that numerous tests with controlled error conditions could be performed. Consistent with the hypotheses that we operate in an environment with orthogonal walls, we start with the CAD models of one of the buildings in our university (see Figure 8.7). We defined five start locations in the building as shown in figure 8.7. Each one represents an initial position for a robot, and at each location the robot used the exploration algorithm described in our former work [159] to build a partial map. To assess robustness, the formerly identified four different types of errors were introduced while building the partial maps. To evaluate the the IWORM algorithm we chose a very high error rate, where each of these errors had an independent 20% chance of occurring. Consequently, each vertex could be affected by more than one error at once.

At each of the five locations we ran the exploration algorithm 20 times and obtained 20 sub-maps. Since the full map contains 40 vertices in total, and we have 5 different regions, we chose maps with more than 8 vertices in order to have overlapping regions. For our experiments we gave a 8+3 vertex gap to assure the overlap between regions, but we just require any number bigger than 8. At the end the exploration algorithm was stopped when 11 vertices were added. Colored regions in figure 8.7 show the areas within which each robot wandered (red area for red start point, and so on).

For each subgraph we exported seven different versions: one without any errors and four with the four types of errors, plus two with all the errors combined with and without error type 4. A good merging would mean that the topology of the resultant merged map closely resembles the ground truth. Similar to what was done in [31], to assess the quality of a merging, we developed a score function to compare the similarities of ground truth and the merged maps. The function assigns one point to a quality value in four cases:

1. 1 point for each vertex that exists in the ground truth map and also exists in the merged map and both have the topological label;

2. 1 point for each vertex that exists in the ground truth map and also exists in the merged map with the same name and direction;
3. 1 point for each vertex that exists in the ground truth map and also exists in the merged map with the same name and number of edges;
4. 1 point for each vertex that exists in the ground truth map and also exists in the merged map with the same name and the direction of all edges is the same;
5. 0 for every other case.

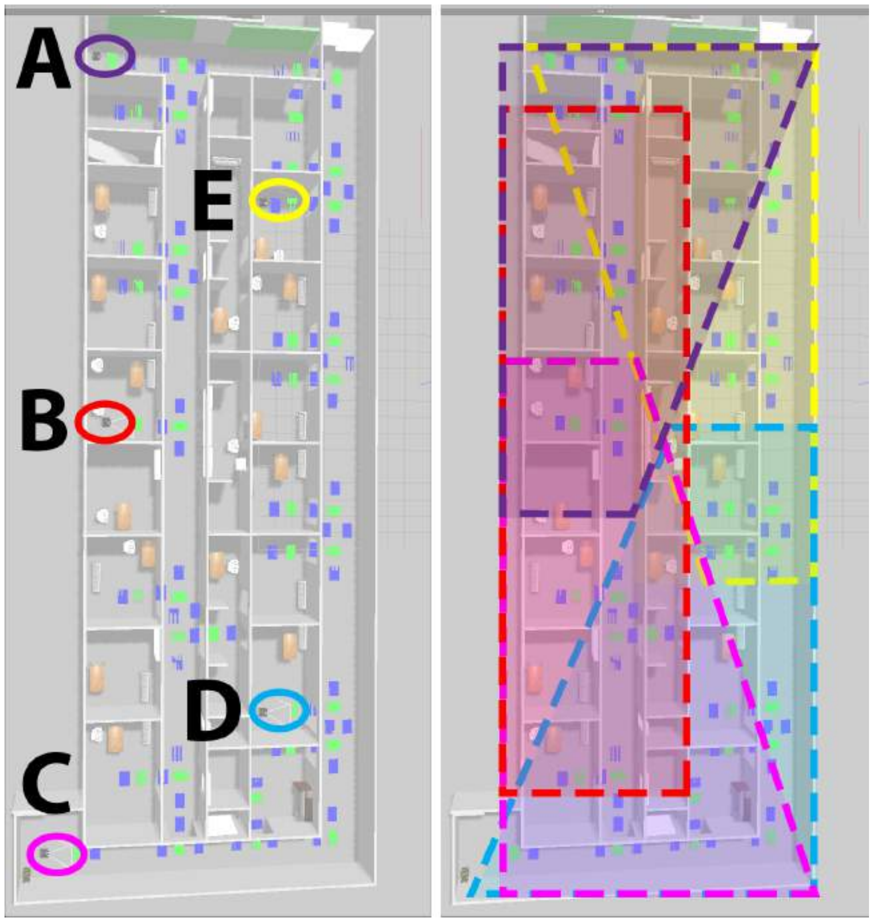


Figure 8.7: Left: start locations. Right: overlapping regions.

8.5 Results

We first tested the merging algorithm with two non-overlapping maps to show how much a map can be affected by the four types of errors when trying to merge them. In this case we took the maps produced by the robot starting in position A and tried to merge them with maps from position D (see figure 8.7 for the non-overlapping footprints). Figure 8.9 compares the final map when merging non-overlapping error-free maps against maps with errors 1,2,3, and 4. Blue and orange circles correspond to discovered corridors and room locations, yellow circles indicate open edges that lead to new undiscovered locations and green arrows correspond to the edges that connect a pair of vertices. After merging a map from region A and D, we show how the merging algorithm kept the correct structure of the sub-maps with little impact on the vertices. On the left of the figure we observe how the algorithm correctly merged both pieces of maps without misplacing, or incorrectly connecting the vertices, and on the right, we can see that, despite adding errors, the final map did not suffer drastic changes. Only 3 vertices were affected by the errors (blue circles with red letters).

Since the final map was qualitatively correct, we quantitatively analyzed the performance of the merged maps for each type of error. Figure 8.8 shows a blue line, labeled Max, representing the maximum score obtained with the error-free maps. As we can observe, the maps with only errors in the direction have a very close quality score to the error-free maps; results that validate the theory about the small impact of this type of error when defining an OTSM that does not affect the topology fundamentally. The maps that contained errors type 2 and 3 remain close to the maximum. However, we see how the label error gives the lowest score, together with the map of combined errors 1+2+3. This is due to how we score the maps based on the label. This also corresponds to the idea that semantics plays a main role when differentiating a vertex from others. It is also expected that the maps with the lowest quality corresponds to the ones with error type 4 and all the errors combined 1+2+3+4.

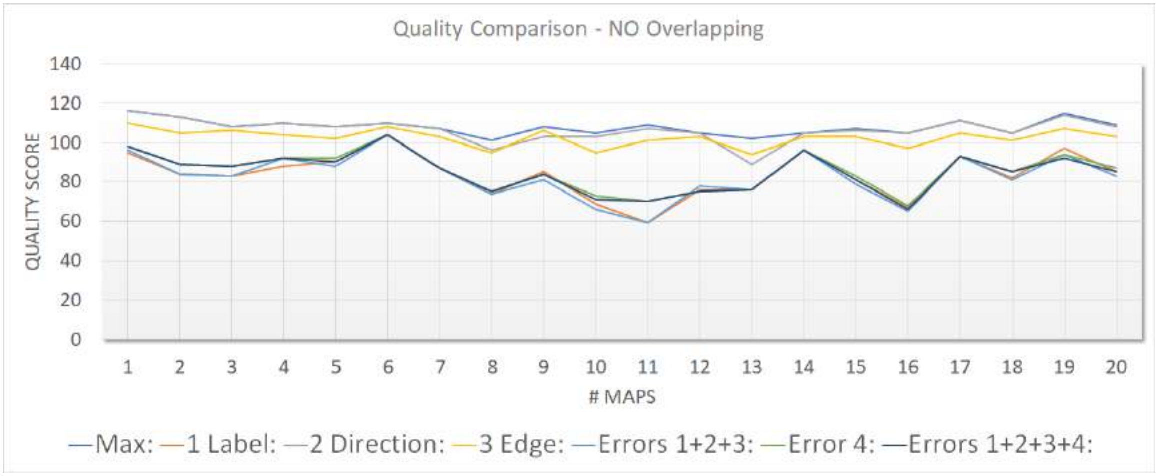


Figure 8.8: Comparison of quality from 20 full maps with different errors when maps do not overlap A+D.

Figure 8.10 shows again the seven versions of the maps, but this time when we combine all the regions. Similarly, as the previous case for no overlapping maps, we observe that for errors 2, and 3 the quality is close to the maximum. The map built in figure 8.11 shows a fully connected map. Once again, for error-free maps, the final

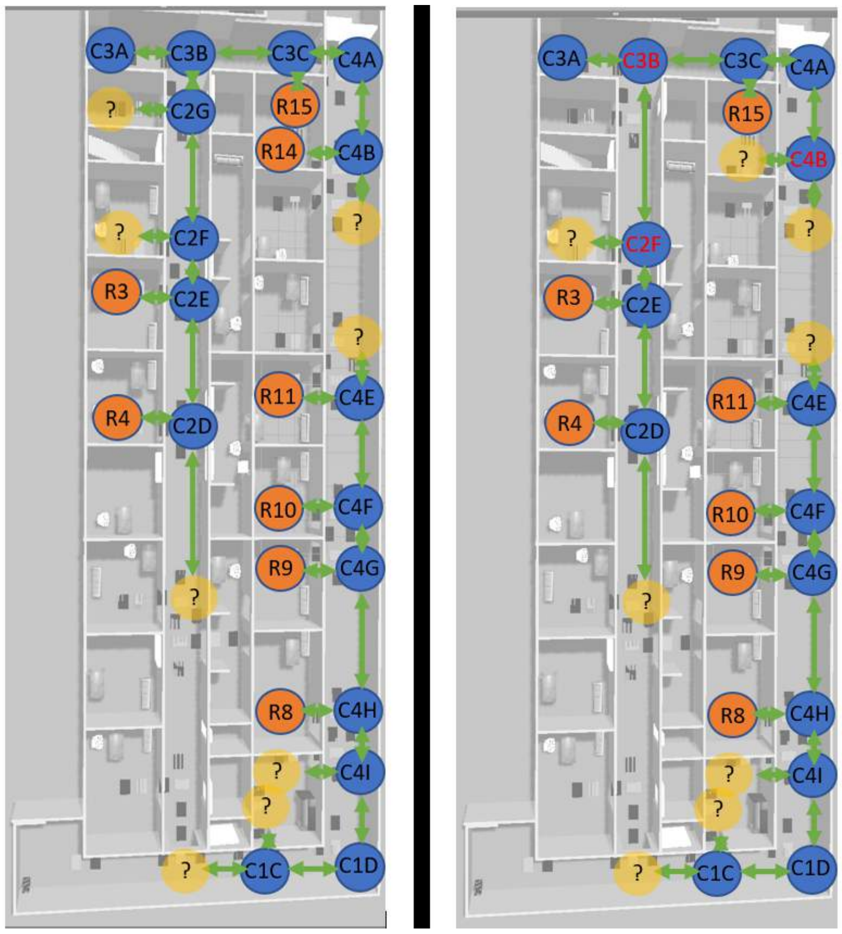


Figure 8.9: Left: OTSM Error Free No Overlapping. Right: OTSM Errors 1+2+3+4 No Overlapping

map is perfectly merged, showing how the IWORM function correctly recognized the overlapping vertices and the matching score adequately served to correctly stitch them together. When trying to merge the maps with all errors, we note that the algorithm can merge the graphs, although as shown in Figure 8.11, there exist some errors in the final map when we compare it with the error-free map. Red arrows show an incorrect edge between two nodes. These issues come mainly from errors type 1 and 4; a missing vertex heavily affects the connection between vertices and this impacts the map's connectivity.

Finally, we did a quantitative analysis when conducting the IWORM from 1 degree up to 4 degrees of depth search. We found that there is no significant difference between degrees 1 and 2, and only 2.19% difference between 2 and 4 for only the maps with all errors. For all our tests we chose degree 3 to prove the intrinsic iterative capacity of our algorithm, even when a degree of 2 would have been enough. However, we believe this degree can play a more significant role when dealing with bigger environments where there are locations in different parts of the map with similar topologies and semantics that require more detailed differentiation.

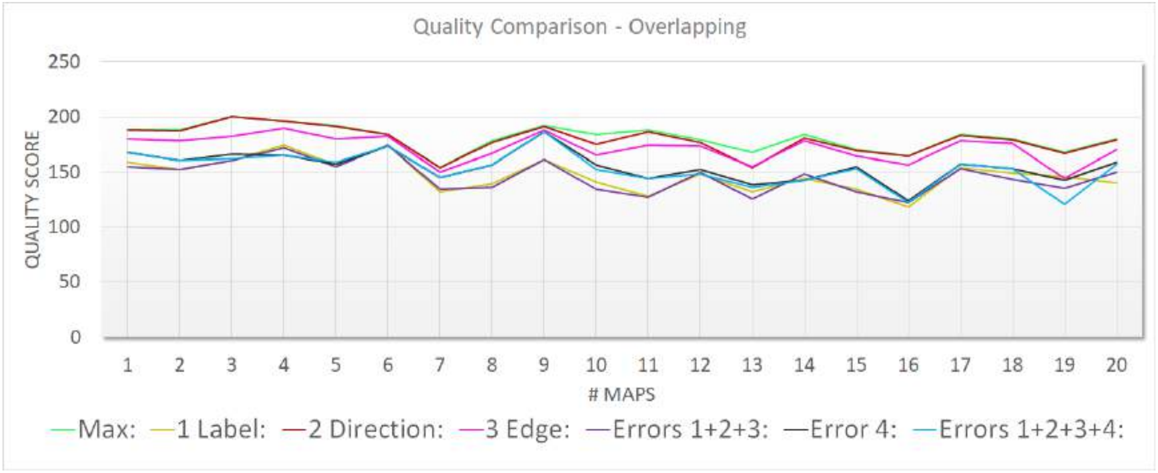


Figure 8.10: Comparison of quality from 20 full maps with different errors when maps overlap A+B+C+D+E.

8.6 Conclusions

In this chapter we presented a new algorithm to merge OTSM that is inspired by the Warrington's object recognition model [180]. The method was tested using overlapping and no-overlapping pieces of a full map that contains four different types of errors and we showed that it is possible to merge maps with errors in the labels, directions, number of edges detected, and missing vertices. The proposed technique proved to be robust to multiple concurrent errors, even when error rates are much higher than what we observed in practice. For future work, we will explore how different exploration strategies, similar to the ones presented, affect the overall quality of the merged maps and how to assess how partial errors in map merging affect the robots' ability to use the combined map for autonomous navigation.

One possible outcome of the presented work will deal with the case of where the sub-maps do not overlap and a new prediction system needs to be used to weigh the probability of closeness between vertices. This will be based on an ordered list of possible matches reinforced by a specific semantic of buildings and expected topological structures of human environments, e.g. offices, houses or factories.

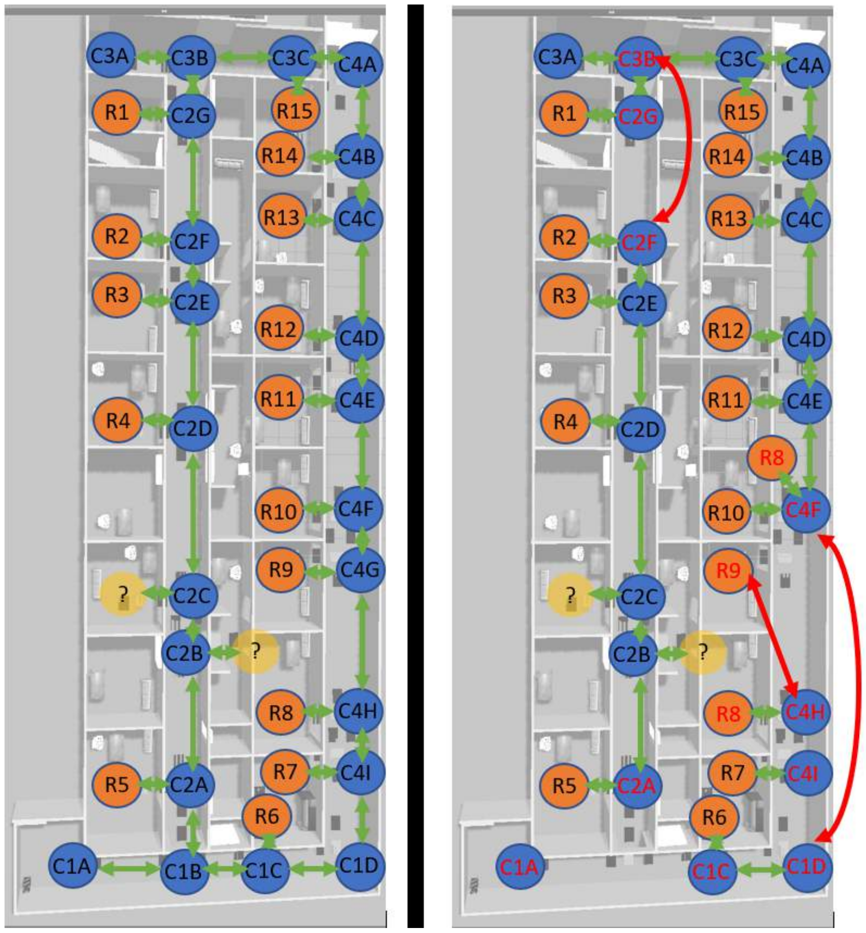


Figure 8.11: Left: Final OTSM error-free. Right: Final OTSM with errors 1+2+3+4

Chapter 9

Conclusions

9.1 Summary of Contributions

Here we present the summary of the multiple contributions of this dissertation. We focused our work on navigation, exploration, and mapping algorithms for indoor robots with uncertainties in motion and sensing. The unifying theme of our work is the use of Constrained Markov Decision Process (CMDP) to compute policies that balance risk and time for the different tasks and problems we studied. We used CMDPs in different scenarios for single and multi robot problems, and we offered new methods to create robust policies to control robots. We proved that having expensive equipment is not always necessary to generate robust policies that compensate the weaknesses of low-cost sensors.

We started studying the case of a single robot visiting multiple targets. This case is challenging, since it generalizes the orienteering problem [21], that is NP-Hard, by incorporating multiple costs and constraints. We showed how the simpler problem of observing preassigned sequences of targets can be solved using the theory of CMDPs. We combined the single robot solutions to solve the rapid deployment problem at the team level. We also considered the problem of splitting the targets among the robots. We formulate a sub-modular objective function leading to a greedy algorithm achieving a $1 - \frac{1}{e}$ approximation.

We closed the gap between reality and theory creating stochastic models to gain a more realistic insight about the performance and behavior of real robots, we demonstrated how a robot's behavior can be extracted empirically, and expanded with this stochastic model. We demonstrated the robustness of calculating an off-line policy, using a novel approach to extract transition probabilities from a stochastic model. Despite motion uncertainties, this stochastic model is able to guide a robot from an origin state to a goal state.

We also proposed a new spatial model dubbed Oriented Topological Semantic Map (OTSM) that extends classic topological maps in a way that is amenable to implementation by robots with minimal sensor payload. We integrated the proposed OTSM model with our recently developed planning method using CMDPs to assign actionable temporal deadlines to the robot. We demonstrated how it was possible to overcome the challenges of implementing such solutions with a real robot. As part of our publication [159] we offered a framework, based on ROS, to enable other researchers to reproduce our results. This work was considered the first Reproducible Articles or *R-articles* in robotics.

Based on the theory about OTSMs, we implemented a method to detect intersections called Intersection Detection System (IDS) using a LIDAR. We combined this IDS with a Labeling System (LS) that uses visual information to recognize locations in a building. We proposed and tested the proof of concept of how to solve the loop closure problem with a ResNet network.

Finally, we completed our research, obtaining inspiration from cognitive sciences and applying it to robotics: solving the problem of map-merging. Our solution was inspired by the human model to perceive objects, and with our experiments showed it effectiveness. We proposed a new two-stage method to compare vertices in different OTSMs and measure their resemblance. Finally, we presented a new merging technique to stitch OTSMs using a semantic and perceptual categorization.

9.2 Future Work

The future of the work presented in this dissertation can be addressed in different directions. We would like see the expansion of the applications of OTSMs for indoor and outdoor scenarios. There is room to measure OTSMs' robustness and flexibility. We would like to use these maps in more complex environments, where the initial assumption of an orthogonal environment is eliminated, and where we could increase the amount of directions, and also a wider set of actions. Similarly, we wish to explore the human aspect of these maps, and implement experiments where both, a robot and a human, interact with each other to accomplish a mission. Collaborative robots is a field that still remains young, and deserves deeper research.

It is possible to use the stochastic model from a robot and extract accurate transition probabilities to solve a CMDP with them. However, more detailed research needs to be conducted to study the variation of the results due to changes in the discretization parameters. There is more to know about how to improve this method including more precise control techniques to reduce motion uncertainty.

For artificial vision there are constant developments that improve the previous state of art; new methods to label and understand the environment can be applied and used together with the presented techniques in this dissertation. New modules developed in ROS or other operative systems can be combined into one whole framework to expand the capabilities of our current robot.

Finally, we hope that our initial contribution to create a reproducible standard for robotics papers can be improved, and become, in the near future, the common method to share knowledge, software, and algorithms. Combining everyone's efforts, we will help each other to reach our goals in robotics more efficiently.

Bibliography

[1] P. Abeles. Robust local localization for indoor environments with uneven floors and inaccurate maps. 2011.

[2] M. Al Khawaldah and S. Livation. Line-of-sight exploration of unknown environment by a pair of mobile Robots. *2018 Advances in Science and Engineering Technology International Conferences, ASET 2018*, pages 1–5, 2018.

[3] K. Albina and S. G. Lee. Hybrid Stochastic Exploration Using Grey Wolf Optimizer and Coordinated Multi-Robot Exploration Algorithms. *IEEE Access*, 7:14246–14255, 2019.

[4] M. Althoff, M. Koschi, and S. Manzinger. CommonRoad: Composable benchmarks for motion planning on roads. *IEEE Intelligent Vehicles Symposium, Proceedings*, (Iv):719–726, 2017.

[5] E. Altman. Constrained Markov decision processes with total cost criteria: Occupation measures and primal LP. *Mathematical methods of operations research*, 43:45–72, 1996.

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

[7] F. Amigoni, V. Castelli, and M. Luperto. Improving Repeatability of Experiments by Automatic Evaluation of SLAM Algorithms. *IEEE International Conference on Intelligent Robots and Systems*, pages 7237–7243, 2018.

[8] F. Amigoni, M. Luperto, and V. Schiaffonati. Toward generalization of experimental results for autonomous robots. *Robotics and Autonomous Systems*, 90:4–14, apr 2017.

[9] S. Y. An, L. Kyoung Lee, and S. Y. Oh. Fast incremental 3D plane extraction from a collection of 2D line segments for 3D mapping. In *IEEE International Conference on Intelligent Robots and Systems*, 2012.

[10] T. Andre and C. Bettstetter. Assessing the value of coordination in mobile robot exploration using a discrete-time Markov process. *IEEE International Conference on Intelligent Robots and Systems*, pages 4772–4777, 2013.

[11] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde tsp solver, 2016.

[12] S. Arora. Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems. *J. ACM*, 45(5):753–782, sep 1998.

[13] A. Bais, R. Sablatnig, and J. Gu. Single landmark based self-localization of mobile robots. In *Third Canadian Conference on Computer and Robot Vision, CRV 2006*, 2006.

[14] B. Balaguer, G. Erinc, and S. Carpin. Classification and regression for wifi localization of heterogeneous robot teams in unknown environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3496–3503, 2012.

[15] R. Bellman. The Theory of Dynamic Programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[16] E. Berhan, B. Beshah, D. Kitaw, and A. Abraham. Stochastic Vehicle Routing Problem: A Literature Survey. *Journal of Information & Knowledge Management*, 13(03):1450022, 2014.

[17] D. P. Bertsekas. *Dynamic Programming & Optimal Control*, volume 1 and 2. Athena Scientific, 2005.

[18] D. J. Bertsimas. A Vehicle Routing Problem with Stochastic Demand. *Operations Research*, 40(3):574–585, 1992.

[19] J. L. Blanco, J. González-Jiménez, and J. A. Fernández-Madrigal. A robust, multi-hypothesis approach to matching occupancy grid maps. *Robotica*, 31(5):687–701, 2013.

[20] F. Blöchliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart. Topomap: Topological Mapping and Navigation Based on Visual SLAM Maps. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3818–3825, 2018.

[21] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation Algorithms for Orienteering and Discounted-Reward TSP. *SIAM Journal on Computing*, 37(2):653–670, 2007.

[22] T. M. Bonanni, B. Della Corte, and G. Grisetti. 3-D Map Merging on Pose Graphs. *IEEE Robotics and Automation Letters*, 2(2):1031–1038, 2017.

[23] F. Bonsignorio. A New Kind of Article for Reproducible Research in Intelligent Robotics [From the Field]. *IEEE Robotics and Automation Magazine*, 24(3):178–182, 2017.

[24] F. Bonsignorio and A. P. Del Pobil. Toward Replicable and Measurable Robotics Research [From the Guest Editors]. *IEEE Robotics and Automation Magazine*, 22(3):32–35, 2015.

[25] F. P. Bonsignorio, J. Hallam, and A. P. del Pobil. Defining the Requisites of a Replicable Robotics Experiment. *Good Experimental Methodology in Robotics. An RSS 2009 Workshop*, 2009.

[26] G. A. Borges and M. J. Aldon. Line extraction in 2D range images for mobile robotics. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 2004.

[27] E. Brunskill and N. Roy. SLAM using incremental probabilistic PCA and dimensionality reduction. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2005.

[28] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider. Coordinated Multi-Robot Exploration. *IEEE Transaction On Robotics*, 21(3), 2005.

[29] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[30] D. Caltabiano, G. Muscato, and F. Russo. Localization and self calibration of a robot for volcano exploration. *Proceedings - IEEE International Conference on Robotics and Automation*, 2004(1):586–591, 2004.

[31] S. Carpin. Fast and accurate map merging for multi-robot systems. *Autonomous Robots*, 25(3):305–316, 2008.

[32] S. Carpin, N. Basilico, D. Burch, T.H. Chung, and M. Kölsch. Variable resolution search with quadrotors: theory and practice. *Journal of Field Robotics*, 30(5):685–701, 2013.

[33] S. Carpin and A. Censi. An experimental assessment of the hsm3d algorithm for sparse and colored data. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3595–3600, 2009.

[34] S. Carpin, T. H. Chung, and B. M. Sadler. Theoretical foundations of high-speed robot team deployment. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2033–2040, 2013.

[35] S. Carpin, M. Pavone, and B. M. Sadler. Rapid multirobot deployment with time constraints. *IEEE International Conference on Intelligent Robots and Systems*, pages 1147–1154, 2014.

[36] F. F. Carvalho, R. C. Cavalcante, M. A.M. Vieira, L. Chaimowicz, and M. F.M. Campos. A multi-robot exploration approach based on distributed graph coloring. *Proceedings - 2013 IEEE Latin American Robotics Symposium, LARS 2013*, pages 142–147, 2013.

[37] E. Cervera. Try to Start It! the Challenge of Reusing Code in Robotics Research. *IEEE Robotics and Automation Letters*, 4(1):49–56, 2019.

[38] B. Chandrasekaran. Models versus rules, deep versus compiled, content versus form: Some distinctions in knowledge systems research. *IEEE Expert*, 6:75–79, 1991.

[39] I-M. Chao, B. L. Golden, and E. A. Wasil. The team orienteering problem. *European Journal of Operational Research*, 88(3):464–474, 1996.

[40] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of robot motion.* MIT Press, 2005.

[41] Y. L. Chow, M. Pavone, B. M. Sadler, and S. Carpin. Trading Safety Versus Performance: Rapid Deployment of Robotic Swarms with Robust Performance Constraints. *ASME Journal on Dynamic Systems, Measurement, and Control*, 137(3):031005.1—-031005.11, 2014.

[42] A. Cosgun and H. I. Christensen. Context-aware robot navigation using interactively built semantic maps. *Paladyn*, 9(1):254–276, 2018.

[43] C. Couprie, L. Najman, L. Najman, and Y. Lecun. Toward Real-time Indoor Semantic Segmentation Using Depth Information. *Journal of Machine Learning Research*, 2014.

[44] T Dang, C. Papachristos, and A. Kostas. Visual Saliency–aware Receding Horizon Autonomous Exploration with Application to Aerial Robotics. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2526–2533, 2018.

[45] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[46] M. P. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, 2011.

[47] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[48] J. Dichtl, L. Fabresse, G. Lozenguez, and N. Bouraqadi. PolyMap: A 2D Polygon-Based Map Format for Multi-robot Autonomous Indoor Localization and Mapping. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10984 LNAI, pages 120–131, 2018.

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

[50] M. Dror, G. Laporte, and P. Trudeau. Vehicle Routing with Stochastic Demands: Properties and Solution Frameworks. *Transportation Science*, 23(3):166–176, 1989.

[51] R. Dubé, A. Cramariuc, D. Dugas, J. Nieto, R. Siegwart, and C. Cadena. SegMap: 3D Segment Mapping using Data-Driven Descriptors. 2018.

[52] M. Durner, M. Brucker, A. Wendt, P. Jensfelt, K. O. Arras, and R. Triebel. Semantic Labeling of Indoor Environments from 3D RGB Maps. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1871–1878, 2018.

[53] M. El Chamie, Y. Yu, and B. Açıkmeşe. Convex synthesis of randomized policies for controlled markov chains with density safety upper bound constraints. In *American Control Conference*, pages 6290–6295, 2016.

[54] M. M. Fard and J. Pineau. MDPs with Non-Deterministic Policies. *Advances in neural information processing systems*, 21:1065–1073, 2009.

[55] X. Fei and S. Soatto. Visual-Inertial Object Detection and Mapping. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11215 LNCS, pages 318–334, 2018.

[56] X. Fei, K. Tsotsos, and S. Soatto. A simple hierarchical pooling data structure for loop closure. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9907 LNCS, pages 321–337, 2016.

[57] E. A Feinberg and A. Shwartz. *Handbook of Markov decision processes: methods and applications*, volume 40. Springer Science & Business Media, 2012.

[58] 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 (CASE)*, pages 297–303, 2014.

[59] S. Feyzabadi and S. Carpin. Planning using hierarchical constrained markov decision processes. *Autonomous Robots*, 41(8):1589–1607, 2017.

[60] S. Friedman, H. Pasula, and D. Fox. Voronoi Random Fields: Extracting the Topological Structure of Indoor Environments via Place Labeling. Technical report, 2007.

[61] B. Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS*, 6:45–53, 2006.

[62] X. Gao and T. Zhang. Unsupervised learning to detect loops using deep neural networks for visual SLAM system. *Autonomous Robots*, 41:1–18, 2017.

[63] I. Garcia-Camacho, G. Alenya, D. Kragic, M. Lippi, M. C. Welle, H. Yin, R. Antonova, A. Varava, J. Borras, C. Torras, and A. Marino. Benchmarking Bimanual Cloth Manipulation. *IEEE Robotics and Automation Letters*, 5(2):1–1, 2020.

[64] S. Garg, N. Suenderhauf, and M. Milford. Don't Look Back: Robustifying Place Categorization for Viewpoint- and Condition-Invariant Place Recognition. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3645–3652, 2018.

[65] M. Gendreau, O. Jabali, and W. Rei. *Chapter 8: Stochastic Vehicle Routing Problems*, pages 213–239. 2014.

[66] B. Golden, S. Raghavan, and E. Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer {US}, 2008.

[67] B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

[68] M. C. Gombolay, R. J. Wilcox, and J. A. Shah. Fast scheduling of robot teams performing tasks with temporospatial constraints. *IEEE Transactions on Robotics*, 34(1):220–239, 2018.

[69] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[70] M. Grinvald, F. Furrer, T. Novkovic, J. J. Chung, C. Cadena, R. Siegwart, and J. Nieto. Volumetric instance-aware semantic mapping and 3D object discovery. *IEEE Robotics and Automation Letters*, 2019.

[71] J. Guerrero and G. Oliver. Swarm-like Methodologies for Executing Tasks with Deadlines. *Journal of Intelligent & Robotic Systems*, 68(1):3–19, sep 2012.

[72] M. Günther, T. Wiemann, S. Albrecht, and J. Hertzberg. Model-based furniture recognition for building semantic object maps. *Artificial Intelligence*, 247:336–351, jun 2017.

[73] J. Hall, C. E. Rasmussen, and J. Maciejowski. Modelling and control of nonlinear systems using gaussian processes with partial model information. In *Proceedings of the 51st IEEE Conference on Decision and Control (CDC)*, pages 5266–5271, 2012.

[74] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[75] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[76] T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel Methods in Machine Learning. *Annals of Statistics*, 36(3):1171–1220, 2008.

[77] W. H. Huang and K. R. Beevers. Topological map merging. *International Journal of Robotics Research*, 24(8):601–613, 2005.

[78] A. Hussein, A. Al-Kaff, A. de la Escalera, and J. M. Armingol. Autonomous Indoor Navigation of Low-Cost Quadcopters. 2015.

[79] Antonios G. Ioannis K. Semantic mapping for mobile robotics tasks: A survey. *Robotics and Autonomous Systems*, 66:86–103, 2015.

[80] K. Jang, E. Vinitsky, B. Chalaki, B. Remer, L. Beaver, A. A. Malikopoulos, and A. Bayen. Simulation to scaled city: Zero-shot policy transfer for traffic control via autonomous vehicles. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '19, pages 291–300, New York, NY, USA, 2019. ACM.

[81] L. Kallenberg. Markov decision processes - Lectures notes. Technical report, University of Leiden, Leiden, 2009.

[82] V. Karpov, A. Migalev, A. Moscowsky, M. Rovbo, and V. Vorobiev. Multi-robot exploration and mapping based on the subdefinite models. In A. Ronzhin, G. Rigoll, and R. Meshcheryakov, editors, *Interactive Collaborative Robotics*, pages 143–152. Springer International Publishing, 2016.

[83] K. Karydis. *A Data-Driven Hierarchical Framework for Planning, Navigation, and Control of Uncertain Systems: Applications to Miniature Legged Robots.* PhD thesis, University of Delaware, 2015.

[84] K. Karydis and M. A. Hsieh. *Uncertainty Quantification for Small Robots Using Principal Orthogonal Decomposition*, pages 33–42. Springer Proceedings in Advanced Robotics. Springer International Publishing, 2017.

[85] K. Karydis, I. Poulakakis, J. Sun, and H. G. Tanner. Probabilistically valid stochastic extensions of deterministic models for systems with uncertainty. *The International Journal of Robotics Research*, pages 1–18, 2015.

[86] K. Karydis, I. Poulakakis, J. Sun, and H. G. Tanner. Probabilistically Valid Stochastic Extensions of Deterministic Models for Systems with Uncertainty. *The International Journal of Robotics Research*, 34(10):1278–1295, 2015.

[87] K. Karydis, D. Zarrouk, I. Poulakakis, R. S. Fearing, and H. G. Tanner. Planning with the STAR (s). In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3033–3038. IEEE, 2014.

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

[89] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR*, 2007.

[90] M.J. Kochenderfer. *Decision Making Under uncertainty.* MIT Lincoln laboratory series, 2015.

[91] A. Kolling and S. Carpin. Extracting surveillance graphs from robot maps. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2323–2328. IEEE, 2008.

[92] V. Koltchinskii, C.T. Abdallah, M. Ariola, P. Dorato, and D. Panchenko. Statistical learning control of uncertain systems: it is better than it seems. Technical Report EECE-TR-00-001, University of New Mexico, 2000.

[93] B. Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119:191–233, 2000.

[94] D. Lang and D. Paulus. Semantic Maps for Robotics. *IEEE International Conference on Robotics and Automation*, 2014.

[95] G. Laporte, F. Louveaux, and H. Mercure. The vehicle routing problem with stochastic travel times. *Transportation science*, 26(3):161–170, 1992.

[96] S. M. Lavalle, J. J. Kuffner, and Jr. Rapidly-Exploring Random Trees: Progress and Prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[97] S.M. LaValle. *Planning algorithms*. Cambridge academic press, 2006.

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

[99] J. Leitner, A. W. Tow, N. Sunderhauf, J. E. Dean, J. W. Durham, M. Cooper, M. Eich, C. Lehnert, R. Mangels, C. McCool, P. T. Kujala, L. Nicholson, T. Pham, J. Sergeant, L. Wu, F. Zhang, B. Upcroft, and P. Corke. The ACRV picking benchmark: A robotic shelf picking benchmark to foster reproducible research. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 4705–4712, 2017.

[100] H. Li and F. Nashashibi. A new method for occupancy grid maps merging: Application to multi-vehicle cooperative local mapping and moving object detection in outdoor environment. *12th International Conference on Control, Automation, Robotics and Vision*, 2012(December):632–637, 2012.

[101] Q. Liu, R. Li, H. Hu, and D. Gu. Building semantic maps for blind people to navigate at home. *Proceedings of the 8th Computer Science and Electronic Engineering Conference*, pages 12–17, 2017.

[102] W. Liu and M.H. Ang. Incremental sampling-based algorithm for risk-aware planning under motion uncertainty. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2051–2058, 2014.

[103] M. Loeve. Probability theory: foundations, random sequences. 1955.

[104] A. W. Long, K. C. Wolfe, M. Mashner, and G. S. Chirikjian. The Banana Distribution is Gaussian: A Localization Study with Exponential Coordinates. In *Proceedings of Robotics: Science and Systems*, 2012.

[105] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[106] S. Lowry, N. Sunderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford. Visual Place Recognition: A Survey. *IEEE Transactions on Robotics*, 32(1):1–19, 2016.

[107] B. Luders, M. Kothari, and J. How. Chance constrained RRT for probabilistic robustness to environmental uncertainty. In *AIAA guidance, navigation and control conference*, 2010.

[108] R. C. Luo and M. Chiou. Hierarchical Semantic Mapping Using Convolutional Neural Networks for Intelligent Service Robotics. *IEEE Access*, 6:61287–61294, 2018.

[109] M. Luperto and F. Amigoni. Predicting the global structure of indoor environments: A constructive machine learning approach. *Autonomous Robots*, pages 1–23, 2018.

[110] Dimitris G Manolakis, Vinay K Ingle, Stephen M Kogon, et al. *Statistical and adaptive signal processing: spectral estimation, signal modeling, adaptive filtering, and array processing.* McGraw-Hill Boston, 2000.

[111] F. Martínez Santa, F.y H. Martínez Sarmiento, and E. Jacinto Gómez. Using the delaunay triangulation and voronoi diagrams for navigation in observable environments. *Revista Tecnura*, 2014.

[112] M. Mazuran and F. Amigoni. Matching line segment scans with mutual compatibility constraints. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2014.

[113] S. Mccammon and G. A. Hollinger. Topological Hotspot Identification for Informative Path Planning with a Marine Robot. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4865–4872, 2018.

[114] O. Mendez, S. Hadfield, N. Pugeault, and R. Bowden. SeDAR - Semantic Detection and Ranging: Humans can localise without LiDAR, can robots? *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 6053–6060, 2018.

[115] M. Milford, S. Anthony, and W. Scheirer. Self-Driving Vehicles: Key Technical Challenges and Progress Off the Road. *IEEE Potentials*, 39(1):37–45, jan 2020.

[116] M. Miłkowski, W. M. Hensel, and M. Hohol. Replicability or reproducibility? On the replication crisis in computational neuroscience and sharing only relevant detail. *Journal of Computational Neuroscience*, 45(3):163–172, 2018.

[117] H. Mnyusiwalla, P. Triantafyllou, P. Sotiropoulos, M. A. Roa, W. Friedl, A. M. Sundaram, D. Russell, and G. Deacon. A Bin-Picking Benchmark for Systematic Evaluation of Robotic Pick-and-Place Systems. *IEEE Robotics and Automation Letters*, 5(2):1–1, 2020.

[118] R. Mur-Artal and J.D. Tardos. ORB-SLAM2 An Open-Source SLAM System for Monocular Stereo.pdf. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[119] K. P. Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012.

[120] L. Naik, S. Blumenthal, N. Huebel, H. Bruyninckx, and E. Prassler. Semantic mapping extension for OpenStreetMap applied to indoor robot navigation. *Proceedings - IEEE International Conference on Robotics and Automation*, 2019-May:3839–3845, 2019.

[121] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming*, 14(1):265–294, 1978.

[122] A. Nemirovski and A. Shapiro. Convex approximations of chance constrained programs. *SIAM Journal on Optimization*, 17(4):969–996, 2007.

[123] V. Nguyen, A. Harati, A. Martinelli, R. Siegwart, and N. Tomatis. Orthogonal SLAM: A step toward lightweight indoor autonomous navigation. In *IEEE International Conference on Intelligent Robots and Systems*, 2006.

[124] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2004.

[125] P. Núñez, R. Vázquez-Martín, J. C. Del Toro, A. Bandera, and F. Sandoval. Feature extraction from laser scan data based on curvature estimation for mobile robotics. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2006.

[126] T. Ogunfunmi. *Adaptive Nonlinear System Identification - The Volterra and Wiener Model Approaches*, volume 8. 2007.

[127] K. Okumura, Y. Tamura, and X. Défago. Amoeba Exploration: Coordinated Exploration with Distributed Robots. *2018 9th International Conference on Awareness Science and Technology, iCAST 2018*, pages 191–195, 2018.

[128] F.A. Oliehoek and C. Amato. *A concise introduction to decentralized POMDPs.* Springer, 2016.

[129] F. O. Ostermann and C. Granell. Advancing Science with VGI: Reproducibility and Replicability of Recent Studies using VGI. *Transactions in GIS*, 21(2):224–237, 2017.

[130] C. C. Pantelides and J. G. Renfro. The online use of First-Principles Models in process operations: review, current status & future needs. Technical report.

[131] P. M. Pardalos and L. Pitsoulis. *Nonlinear Assignment Problems: Algorithms and Applications*, volume 7. Springer Science & Business Media, 2000.

[132] J. Park, A. J. Sinclair, R. E. Sherrill, E. A. Doucette, and J. W. Curtis. Map merging of rotated, corrupted, and different scale maps using rectangular features. *Proceedings of the IEEE/ION Position, Location and Navigation Symposium*, pages 535–543, 2016.

[133] M. J. Paulik, J. Overholt, M. Krishnan, Y. Alnounou, and G. Hudas. Occupancy Grid Map Merging using Feature Maps. In *IASTED Technology Conferences*, pages 10.2316/P.2010.706–074., 2016.

[134] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. Fan Chen, C. Choi, J. Dusek, D. Hoehener, S-Y. Liu, M. Novitzky, I. Franzoni Okuyama, J. Pazis, G. Rosman, V. Varricchio, H.-C. Wang, D. Yershov, H. Zhao, M. Benjamin, C. Carr, M. Zuber, S. Karaman, E. Frazzoli, D. Del Vecchio, D. Rus, J. How, J. Leonard, and A. Censi. Dukietown: an open, inexpensive and flexible platform for autonomy education and research. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1497–1504, 2017.

[135] R. Peng. A Simple Explanation for the Replication Crisis in Science · Simply Statistics, 2016.

[136] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810, May 2018.

[137] S. T. Pfister, S. I. Roumeliotis, and J. W. Burdick. Weighted line fitting algorithms for mobile robot map building and efficient data representation. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2003.

[138] A. Pörtner, M. Hoffmann, S. Zug, and M. König. SwarmRob: A Docker-Based Toolkit for Reproducibility and Sharing of Experimental Artifacts in Robotics Research. *Proceedings - 2018 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2018*, pages 325–332, 2019.

[139] L. F. Posada, A. Velasquez-lopez, F. Hoffmann, and T. Bertram. Semantic Mapping with Omnidirectional Vision. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1901–1907, 2018.

[140] A. Pronobis, O. Martínez Mozos, B. Caputo, and P. Jensfelt. Multi-modal Semantic Place Classification On behalf of: Multimedia Archives can be found at: The International Journal of Robotics Research Additional services and information for Multi-modal Semantic Place Classification. *The International Journal of Robotics ResearchThe International Journal of Robotics Research*, 29(3):298–320, 2010.

[141] A. Purohit, P. Zhang, B. M. Sadler, and S. Carpin. Deployment of swarms of micro-Aerial vehicles: From Theory To practice. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5408–5413, 2014.

[142] M. L. Puterman. *Markov decision processes : discrete stochastic dynamic programming*. Wiley-Interscience, British Columbia, 2005.

[143] A. Quattrini Li, R. Cipolleschi, M. Giusto, and F. Amigoni. A semantically-informed multirobot system for exploration of relevant areas in search and rescue settings. *Autonomous Robots*, 40(4):581–597, 2016.

[144] R. Ramaithititima. Landmark-based Exploration with Swarm of Resource Constrained Robots. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 5034–5041, 2018.

[145] C. E. Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[146] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[147] G. Robles. Replicating MSR: A study of the potential replicability of papers published in the Mining Software Repositories Proceedings. *Proceedings - International Conference on Software Engineering*, pages 171–180, 2010.

[148] G. Robles, U. Rey, J. Carlos, and D. M. Germán. Beyond Replication : An example of the potential benefits of replicability in the Mining of Software Repositories Community. *Victoria*, (February):0–3, 2010.

[149] E. Robotics, N. Noe, F. Bonsignorio, J. Hallam, and A. P. Del Pobil. Good Experimental Methodology GEM Guidelines. Technical report, 2008.

[150] M. Rohani, D. Gingras, and D. Gruyer. A novel approach for improved vehicular positioning using cooperative map matching and dynamic base station DGPS concept. *IEEE Transactions on Intelligent Transportation Systems*, 17(1):230–239, 2016.

[151] S. Saeedi, M. Trentini, M. Seto, and H. Li. Multiple-Robot Simultaneous Localization and Mapping: A Review. *Journal of Field Robotics*, 33(1):3–46, 2016.

[152] R. Sandstr, A. Bregger, B. Smith, S. Thomas, and N. M. Amato. Topological Nearest-Neighbor Filtering for Sampling-Based Planners. *Proceedings of the IEEE International Conference on Robotics and Automation, Brisbane, Australia*, pages 3053–3060, 2018.

[153] D. C. Shah and M. E. Campbell. A qualitative path planner for robot navigation using human-provided maps. *International Journal of Robotics Research*, 2013.

[154] S. G. Shahbandi, M. Magnusson, and K. Iagnemma. Nonlinear Optimization of Multimodal Two-Dimensional Map Alignment With Application to Prior Knowledge Transfer. *IEEE Robotics and Automation Letters*, 3(3):2040–2047, 2018.

[155] M.T.J. Spaan. Partially observable Markov decision processes. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning: State-of-the-art*. Springer, 2012.

[156] M. F. Stoelen, V. F. De Tejada, A. Jardón Huete, C. Balaguer, and F. P. Bonsignorio. Distributed and adaptive shared control systems. *IEEE Robotics and Automation Magazine*, 22(4):137–146, 2015.

[157] P. Stotko, S. Krumpen, M. Schwarz, C. Lenz, S. Behnke, R. Klein, and M. Weinmann. A VR System for Immersive Teleoperation and Live Exploration with a Mobile Robot. pages 3630–3637, 2020.

[158] J. L. Susa Rincon and S. Carpin. Map Merging of Oriented Topological Semantic Maps. In *International Symposium on Multi-Robot and Multi-Agent Systems*, pages 202–208, 2019.

[159] J. L. Susa Rincon and S. Carpin. Time-constrained exploration using toposemantic spatial models: A reproducible approach to measurable robotics. *IEEE Robotics Automation Magazine*, 26(3):78–87, Sep. 2019.

[160] J. L. Susa Rincon, P. Tokekar, V. Kumar, and S. Carpin. Rapid deployment of mobile robots under temporal, performance, perception, and resource constraints. *IEEE Robotics and Automation Letters*, 2(4):2016–2023, Oct 2017.

[161] R.S. Sutton and A.G. Barto. *Reinforcement Learning.* MIT Press, 2018.

[162] S. Temizer, M. J Kochenderfer, L. P. Kaelbling, T. Lozano-Pérez, and J. K. Kuchar. Collision Avoidance for Unmanned Aircraft using Markov Decision Processes *. In *AIAA Guidance, Navigation, and Control Conference*, Toronto, 2010. American Institute of Aeronautics and Astronautics.

[163] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics.* MIT Press, 2006.

[164] S. Thrun and M. Montemerlo. The graph SLAM algorithm with applications to large-scale mapping of urban structures. In *International Journal of Robotics Research*, volume 25, pages 403–429, 2006.

[165] S. Thrun and Others. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, 1(1-35):1, 2002.

[166] P. Tokekar and V. Kumar. Visibility-based persistent monitoring with robot teams. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3387–3394. IEEE, 2015.

[167] L. Trevisan. Combinatorial optimization: exact and approximate algorithms. *Standford University*, 2011.

[168] P. Triantafyllou, H. Mnyusiwalla, P. Sotiropoulos, M. A. Roa, D. Russell, and G. Deacon. A benchmarking framework for systematic evaluation of robotic pick-and-place systems in an industrial grocery setting. *Proceedings - IEEE International Conference on Robotics and Automation*, 2019-May:6692–6698, 2019.

[169] E. Tsardoulias, A. Thallas, and L. Petrou. Metric map merging using RFID tags & topological information. *CoRR*, abs/1711.06591, 2017.

[170] J. van den Berg, P. Abbeel, and K. Goldberg. LQG-MP: optimized path planning for robots with motion uncertainty and imperfect state information. *International Journal of Robotics Research*, 30(7):895–913, 2011.

[171] J. van den Berg, D. Wilkie, S.J. Guy, M. Niethammer, and D. Manocha. LQG-obstacles: feedback control with collision avoidance for mobile robots with motion and sensing uncertainty. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 346–353, 2012.

[172] B. Vanderborght. Decisions, Decisions [From the Editor's Desk]. *IEEE Robotics Automation Magazine*, 26(3):4–13, 2019.

[173] V. N. Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 10(5):988–999, 1999.

[174] K. M. Varadarajan. Topological mapping for robot navigation using affordance features. In *ICARA 2015 - Proceedings of the 2015 6th International Conference on Automation, Robotics and Applications*, pages 42–49, 2015.

[175] E. Vidal, J. D. Hern, and K. Isteni. Optimized environment exploration for autonomous underwater vehicles. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 6409–6416, 2018.

[176] M. Vidyasagar. Randomized Algorithms for Robust Controller Synthesis using Statistical Learning Theory. *Automatica*, 37(10):1515–1528, 2001.

[177] M. Vidyasagar. *Learning and Generalization With Applications to Neural Networks*. Springer-Verlag, London, UK, 2nd edition, 2003.

[178] J. Vilela, Y. Liu, and G. Nejat. Semi-autonomous exploration with robot teams in urban search and rescue. *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2013*, pages 1–6, 2013.

[179] N. G. Villanueva-Chacon and E. A. Martinez-Garcia. Tele-robotic distributed architecture for sewer exploration. *Proceedings of the 6th Andean Region International Conference, Andescon 2012*, pages 220–223, 2012.

[180] E. K. Warrington. The selective impairment of semantic memory. *The Quarterly journal of experimental psychology*, 27(4):635–657, 1975.

[181] M. Wiering and M. van Otterlo, editors. *Reinforcement Learning: State-of-the-art*. Springer, 2012.

[182] B. Yamauchi. A frontier-based exploration for autonomous exploration. *IEEE International Symposium on Computational Intelligence in Robotics and Automation, Monterey, CA*, pages 146–151, 1997.

[183] B. Yamauchi. A frontier-based approach for autonomous exploration. In *International Symposium on Computational Intelligence in Robotics and Automation*, pages 146–151, 1997.

[184] Z. Yan, N. Jouandeau, and A. A. Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10, 2013.

[185] B. Yang, D. Jayaraman, J. Zhang, and S. Levine. REPLAB: A reproducible low-cost arm benchmark for robotic learning. *Proceedings - IEEE International Conference on Robotics and Automation*, 2019-May:8691–8697, 2019.

[186] D. Zhu, T. Li, D. Ho, C. Wang, and M. Q. Meng. Deep Reinforcement Learning Supervised Autonomous Exploration in Office Environments. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 7548–7555, 2018.

[187] D. Zou and P. Tan. CoSLAM: Collaborative visual SLAM in dynamic environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.