UNIVERSITÄT BONN

Inaugural-Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Landwirtschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn
Institut für Geodäsie und Geoinformation

Robot Navigation Beyond Planning the Shortest Path

von

Lorenzo Nardi

aus
Assisi, Italien

Bonn 2021

Referent:
Prof. Dr. Cyrill Stachniss, University of Bonn, Germany

Korreferent:
Prof. Dr. Luca Iocchi, Sapienza University of Rome, Italy

Tag der mündlichen Prüfung:

6. Juli 2020

Angefertigt mit Genehmigung der Landwirtschaftlichen Fakultät der Universität Bonn

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenenen Hilfsmittel angefertigt habe; die aus fremdem Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum	(Unterschrift)

Zusammenfassung

In den letzten zehn Jahren ist die Nachfrage nach autonomen mobilen Robotern kontinuierlich gestiegen. Die Anwendungen reichen von mobilen Manipulatoren bis hin zu selbstfahrenden Autos. Die Fähigkeit autonom in komplexen und dynamischen Umgebungen zu navigieren ist Grundvoraussetzung für diese Anwendungen. Der Einsatz von Robotern in dynamischen Umgebungen mit bewegten Objekten wie beispielsweise Personen, erfordert ausserdem ein sicheres und angemessenes Navigationsverhalten der autonomen Systeme. Weiterhin sind reale Szenarien typischerweise durch Unsicherheiten in den Beobachtungen, in der Ausführung von Aktionen und über den Zustand der Umgebung gekennzeichnet. Traditionelle Ansätze zur autonomen Navigation basieren meist auf der Berechnung des kürzestens Weges in einer statischen, geo-metrischen Repräsentation der Umgebung. Planungssysteme, die auf einer solchen Modellierung für reale Szenen aufsetzen sind oft nicht ausreichend, um in dynamischen Umgebungen zu operieren und können zu einem suboptimalen Navigationsverhalten führen.

In dieser Arbeit präsenterien wir eine Reihe von neuen Ansätzen zur autonomen Navigation von mobilen Robotern in realen Szenarien, die über die Planung des kürzesten Weges hinausgehen. Die vorgestellten Techniken berücksichtigen bei der Navigation sowohl dynamische Objekte als auch Unsicherheiten über den Zustand der Umgebung und nutzen Hintergrundinformationen. Wir verwenden beispielsweise öffentliche, städtische Karten für eine robuste Navigation in urbanen Umgebungen und schätzen erwartete Positionsunsicherheiten. Außerdem nutzen wir bei der Navigation zuvor zurückgelegte Wege, um ein sicheres und vorhersehbares Verhalten der Roboter in der Zukunft zu erzeugen, das den Präferenzen des Benutzers folgt. Zusätzlich stellen wir Methoden für die Navigation in teilweise unbekannten Umgebungen vor. Dabei erzielen wir durch aktiv erfasste Informationen eine automatisierte, sukzessive Verbesserung im Navigationsverhalten. Wir nutzen dafür die Beobachtungen des Roboters, um automatisert neue Pfade zu bestimmen auf denen negative Einflüsse des Geländes geringer sind. Darüber hinaus berücksichtigen wir beobachtete Änderungen in der Befahrbarkeit um eine vorausschauende Navigation mit geringer Anzahl unvorhergesehener Hindernisse zu gewährleisten.

Abstract

Over the last decade, the demand for autonomous mobile robots has been growing continuously. Applications range from mobile manipulators operating on factory floors to autonomous cars driving in urban environments. A common requirement for all these tasks is the capability to autonomously navigate by making sequences of decisions in environments that are complex, dynamic, and uncertain. Mobile robots are often deployed in environments populated by humans or other moving objects and are required to perform safe and compliant navigation. Furthermore, real-world scenarios are typically characterized by uncertainty in the robot's perception, action execution, and belief about the world. Traditional approaches to robot navigation plan and follow the shortest path on static geometric representations of the environment. Such systems are often not adequate to capture the characteristics of real-world environments and may lead robots to perform behaviors that are sub-optimal in practice.

In this thesis, we address robot navigation in different real-world scenarios and investigate a set of approaches that go beyond planning the shortest paths. We present solutions for robot navigation that are able to take into account and reason about the situation in which the robot navigates, the dynamics populating the environment, and the uncertainty about the world. We achieve this by exploiting available background knowledge. For example, we use publicly available maps of urban environments to planning policies for performing robust navigation on road networks under position uncertainty. Whereas, we exploit the paths previously experienced by the robot to generate safe and predictable behaviors that meet the user's preferences. We also present solutions for navigating in partially unknown environments by actively gathering information and by exploiting this knowledge to automatically improve robot navigation over time. We use the onboard robot perception during navigation in outdoor environments to automatically discover paths along which the impact of detrimental factors due to the terrain is lower. Furthermore, we exploit the observations about the traversability changes in an environment to plan anticipatory behaviors that lead the robot to encounter a reduced number of unforeseen obstacles while navigating.

Acknowledgements

My path to the Ph.D. has been very different from the shortest path. I've been wandering around, exploring, and learning a lot throughout this journey. Sometimes, I took longer deviations. But, it is thanks to the many people who helped and supported me along this path that I could write this thesis.

First, I would like to thank Cyrill Stachniss for being a great supervisor. He gave me the opportunity and the time to grow and to find my path through this journey. He has been a brilliant example to follow and to learn from, and I sincerely appreciate his guidance and support. Thank you, Cyrill.

I want to thank my lab mates and friends who shared joy, pain, fun, frustration, and drinks with me. Thanks to Igor Bogoslavskyi and Ignacio Vizzo for being the coolest office mates I could have. Thanks to Olga Vysotska for being there every time I needed support. Thanks to Nived Chebrolu for the interesting and inspiring chats. Thanks to Andres Milioto for being a good friend since the first day we met. Thanks to Taigo Bonanni, Mathias Hans, Susanne Wenzel, Johannes Schneider, Thomas Läbe, Emanuele Palazzolo, Philipp Lottes, Jens Behley, Ribana Roscher, Robert Schirmer, Christian Merfels, Philippe Giguère, Xieyuanli Chen, Jan Weyler, Irvin Aloise, Louis Wiesmann, Federico Magistri. Each of you enriched me and contributed to making this experience unique and unforgettable. I also would like to thank Birgit Klein for all the help through the German world and for being always kind and patient with me. Another thanks goes to Jens for carefully proofreading this thesis, and to Jan and Birgit for helping me with the Zusammenfassung.

I want to thank my mom and family for their unceasing and unconditional love. They have been a solid foundation on which I could always rely on, no matter how far I was. They all contributed to make me the person I am, and I could not be more grateful to them.

Thanks also to my lifetime friends Leonardo Andreucci, Giorgio D'Antoni, Daniele Bagarani, Alessio Mellino, Diego Chitarrini, Luca Puggini, Matteo Angelini, David Shaholli, Luisa Mari, Chiara Manes for the good time spent together. Despite the distance, when we are together, it feels like time has not passed.

Finally, I want to thank Elisa Giannantoni for joining me in this adventure. She always believed in me without ever doubting that I could make it. Thank you

for understanding my ups and downs, my anxiety, the weekends and the evenings at the office. This journey would have been much harder without having you by my side.

The work presented in this thesis is partially supported by the European Commission through the RobDREAM project, H2020-ICT-645403-RobDREAM, and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy, EXC-2070-390732324-PhenoRob.
The financial support of the EC through the RobDREAM project and of the DFG through the PhenoRob cluster is gratefully acknowledged.

Contents

Chapter 1

Introduction

NOWADAYS, autonomous mobile robots are populating the world and are progressively becoming a part of our daily lives. The capability to navigate autonomously allows mobile robotic systems to catch on a wide range of applications both in commercial and in industrial settings. On factory floors, mobile robots are employed to perform tasks such as mobile manipulation or transportation of materials. Outside factories, mobile robots have found many applications such as environmental inspection, surveillance, precision agriculture, and many others. In homes, offices, hospitals, and shopping malls, mobile robots are deployed to perform tasks from vacuuming to escorting people. Furthermore, mobile robotic technology is guiding the development of self-driving cars, which are expected to revolutionize the future of urban mobility. A fundamental requirement that is common to all these tasks is the ability of robots to autonomously make sequences of decisions for navigation.

Most robot navigation systems consist of three primary components: mapping, localization, and path planning. Mapping allows for building a representation of the environment where the robot navigates, whereas localization estimates the position of the robot on this map. Path planning is the problem of deciding how the robot should move to reach a given goal location. Planning is a crucial component of robot navigation as it defines the robot's navigation behavior, including which trajectory it follows and how it reacts to unforeseen obstacles. Given the current location of the robot provided by the localization system and a target location, path planning algorithms compute a sequence of poses or actions in the map of the environment that leads the robot to the goal by avoiding collisions with obstacles.

Since the development of "Shakey, the robot" in the mid-1960s, the path planning problem has been widely studied. Traditional planning approaches stem from the shortest path problem in graph theory. This is the problem of finding the path between two vertices on a weighted graph along which the sum of the

(a) Office populated by dynamic agents.

(b) Environment with different terrains.

Figure 1.1: Mobile robots are deployed in many real-world scenarios ranging from dynamic indoor environments populated by other moving agents (e.g. humans) to outdoor environments presenting different characteristics for navigation (e.g. roughness).

weights (or costs) is minimal. Many robot navigation systems use shortest path planning algorithms, such as Dijkstra's or A*, on static geometrical representations of the environment to compute paths that minimize the travel time, the travel distance, the clearance from the obstacles, or a particular cost function. Planning the shortest path for robot navigation generates minimum cost paths by assuming that the environment is stationary and that the world is perfectly known. Furthermore, it requires an expert to define a specific cost function for navigation in the environment. These assumptions are acceptable if the robot navigates in controlled environments, but they may become inadequate to tackle general real-world scenarios.

Real-world environments are typically complex, dynamic, and uncertain. Therefore, planning and following the shortest path may result in trajectories that are far from being optimal in practice. Consider, for example, a mobile robot that operates in the environments illustrated in Figure 1.1. An indoor environment populated by dynamic agents or humans like the one shown in Figure 1.1(a) changes continuously. Thus, following a once-planned shortest path may lead the robot to encounter blocked passages and unforeseen obstacles, increasing the risk of collisions. Instead, in an outdoor environment consisting of different terrains with diverse characteristics as the one illustrated in Figure 1.1(b), planning the shortest path is not able to capture and incorporate the onboard observations of the robot during navigation. Thus, it may expose the robot to detrimental factors such as strong vibrations or high power consumption.

Robot navigation in real-world environments poses several challenges. A robot navigation system should, first, take into account the changes in the environment and the different dynamic agents populating it to perform safe behaviors. Second,

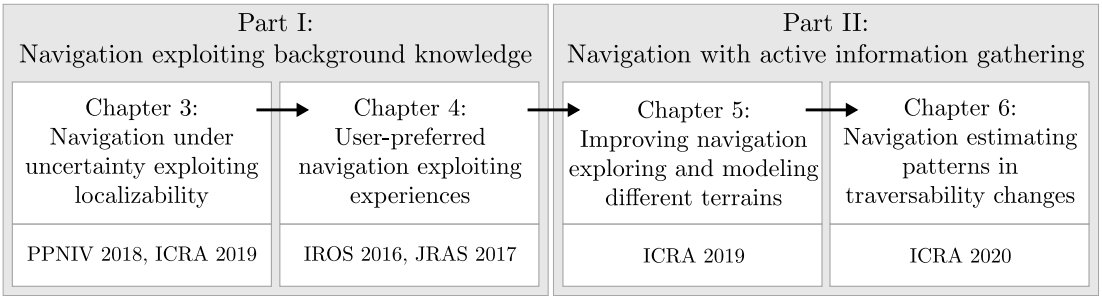


Figure 1.2: Overview of the structure of this thesis. It consists of four chapters divided in two main parts: the first part focuses on navigation exploiting background knowledge, whereas the second one on navigation integrating active information gathering. The last row shows the publications related to each chapter.

it should deal with the uncertainty about the environment representation, the sensor measurements, and the robot's belief about the world to make appropriate decisions for navigation. Third, it should be able to automatically adapt the robot's behavior to the current situation without requiring an expert to tune its parameters manually. In this thesis, we focus on robot navigation systems that aim at solving these challenges by going beyond planning the shortest path on static representations of the environment.

This thesis consists of two main parts, and an overview of its structure is illustrated in Figure 1.2. The first part aims at optimizing navigation by using background information about the environment and the experiences accumulated by the robot during traversal. We present approaches to robot navigation that take advantage of this data to reason about the robot's uncertainty and the user's preferences. Our solutions allow robots to navigate safely in uncertain environments and to perform different behaviors according to the situation and the dynamics in the scene. The second part aims at improving robot navigation in partially known environments over time by actively gathering information during navigation. We present approaches that collect information about the environment, reason about this knowledge to predict the state of the world at unknown locations, and exploit the predictions for navigation. Our solutions lead robots to automatically discover the favorable regions for navigation on different terrains as well as to navigate safely and efficiently in changing environments.

Throughout this thesis, we address robot navigation in different environments with diverse robotic platforms that range from a mobile manipulator operating on a factory floor to a self-driving car driving on a road network. We discuss the robot navigation problem by focusing on path planning and decision-making problems while taking into account their relationship with mapping and localization. The approaches presented in this thesis contributed to an EU-funded project and a DFG cluster and have been published and presented at peer-reviewed international conferences and journals.

1.1 Main contributions

The main contribution of this thesis is a set of solutions to robot navigation that leverage information about the environment, the robot's experiences, and the observations collected during traversal to make decisions for navigating in complex, uncertain, and dynamic environments. First, we provide a summary of the main techniques on which this thesis is based in Chapter 2.

In Part I, we focus on navigation exploiting background knowledge. The first contribution presented in Chapter 3 is a solution for making decisions to navigate on a road network by taking into account the uncertainty about the robot's position and action execution. We aim at reducing the number of wrong directions that the robot takes during navigation with large position uncertainty and, thus, the travel time to its destination. To this end, we consider a compact representation of the robot's belief that discretizes the possible degrees of uncertainty. We propose to exploit a localization prior computed from publicly available maps to estimate how the belief about the robot's position will propagate along the road network. In contrast to most existing methods, our approach generates navigation policies that enable the robot to make decisions according to the degree of uncertainty while being computationally tractable.

The second contribution presented in Chapter 4 is a path planning approach that exploits the previously experienced paths to meet the preferences of a non-expert user. We propose to extract preferences by allowing the user to demonstrate desired behaviors or to provide feedback about the previous experiences of the robot. We store the preferred behaviors and automatically incorporate them into the planning process for successive tasks. Our approach is able to reproduce and generalize desirable navigation behaviors depending on the current situation without requiring experts to hard-code rules or define complex cost functions. We also incorporate a probabilistic approach for predicting the uncertain trajectories of the moving entities that share the workspace with the robot within this framework. This approach allows for planning local deviations to safely avoid dynamic obstacles while still performing foreseeable behaviors.

Part II of this thesis focuses on navigation integrating active information gathering. The third contribution presented in Chapter 5 is an approach to navigate in outdoor environments consisting of terrains with diverse unknown characteristics. We aim at reducing the detrimental phenomena affecting robot navigation, such as strong vibrations or high power consumption, caused by the terrain on which the robot navigates. We learn a place-dependent model of such phenomena by using an aerial image of the environment as a prior and by incorporating the onboard observations of the robot during traversal. We use this model to plan paths that trade off the exploration of unknown promising regions

and the exploitation of known areas where the impact of the detrimental factors on navigation is low. Our approach leads the robot to automatically navigate along paths where it experiences fewer vibrations or energy consumption over time.

The fourth contribution presented in Chapter 6 is an approach for generating strategies to navigate over extended periods of time in indoor environments subject to changes in the traversability that occur by following patterns. We aim at increasing the efficiency of the robot's operations by reducing the number of unforeseen blocked passages that the robot encounters during navigation. To achieve this, we incrementally model the traversability changes in the environment from the robot's observations during traversal. We use a probabilistic model that allows for making predictions at currently unobserved locations. We take the predictions into account to plan paths that trade off the risk to encounter obstacles and the information gain of visiting unknown locations for improving the model and the subsequent predictions. Our approach leads the robot to encounter a reduced number of blocked passages and, thus, to navigate along shorter paths over time than following traditional shortest path methods.

1.2 Publications

Parts of this thesis have been published in the following papers:

- L. Nardi and C. Stachniss. Experience-Based Path Planning for Mobile Robots Exploiting User Preferences. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016
- L. Nardi and C. Stachniss. User Preferred Behaviors for Robot Navigation Exploiting Previous Experiences. *Journal on Robotics and Autonomous Systems (RAS)*, 97, 2017
- L. Nardi and C. Stachniss. Towards Uncertainty-Aware Path Planning for Navigation on Road Networks Using Augmented MDPs. In *10th Workshop on Planning, Perception and Navigation for Intelligent Vehicles at the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018
- L. Nardi and C. Stachniss. Uncertainty-Aware Path Planning for Navigation on Road Networks Using Augmented MDPs. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019
- L. Nardi and C. Stachniss. Actively Improving Robot Navigation On Different Terrains Using Gaussian Process Mixture Models. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019
- L. Nardi and C. Stachniss. Long-Term Robot Navigation in Indoor Environments Estimating Patterns in Traversability Changes. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020

Additional work not included in this thesis:

- A. Ahmadi, L. Nardi, N. Chebrolu, and C. Stachniss. Visual Servoing-based Navigation for Monitoring Row-Crop Fields. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020
- X. Chen, T. Läbe, L. Nardi, J. Behley, and C. Stachniss. Learning an Overlap-based Observation Model for 3D LiDAR Localization. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020

Chapter 2

Basic techniques

In this chapter, we provide a summary of techniques on which the solutions presented in this thesis rely. We describe first some basic methods for state estimation and probabilistic inference. We discuss then different basic approaches to path planning and decision-making under uncertainty.

2.1 State estimation

State estimation is the problem of estimating the state of a system that is not directly observable. For robot navigation, a fundamental state estimation problem is the localization problem that aims at estimating the position of the robot given a map of the environment from noisy sensor measurements.

2.1.1 Extended Kalman filter

The Kalman filter (KF) [66] is a popular state estimation technique used for filtering and computing predictions. It implements a Bayes filter for linear Gaussian systems. The KF allows for estimating the internal state of a system that follows a linear transition and observation model with additive Gaussian noise:

$$\mathbf{x}_t = \mathbf{A}_t \, \mathbf{x}_{t-1} + \mathbf{B}_t \, \mathbf{u}_t + \boldsymbol{\epsilon}_t, \tag{2.1}$$

$$\mathbf{z}_t = \mathbf{C}_t \, \mathbf{x}_t + \boldsymbol{\delta}_t, \tag{2.2}$$

where $\mathbf{x}_t$ represents the state of the system, $\mathbf{u}_t$ is the control applied to the system, and $\mathbf{z}_t$ the observation at time t; the matrices $\mathbf{A}_t$, $\mathbf{B}_t$ and $\mathbf{C}_t$ describe the evolution of the state, how the control input changes the state, and how the observation is generated given the state respectively; the random variables $\boldsymbol{\epsilon}_t$ and $\boldsymbol{\delta}_t$ represent the process and observation noise and are distributed according to a Gaussian distribution with zero mean and covariance $\mathbf{R}_t$ and $\mathbf{Q}_t$ respectively. Given an initial Gaussian belief about the state of the system $bel(\mathbf{x}_{t-1}) = \mathcal{N}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$,

the control $\mathbf{u}_t$, and the sensor's observation $\mathbf{z}_t$, the KF computes a new belief $bel(\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ according to:

$$\begin{aligned}
\overline{\boldsymbol{\mu}}_t &= \mathbf{A}_t\,\boldsymbol{\mu}_{t-1} + \mathbf{B}_t\,\mathbf{u}_t, && (2.3)\\
\overline{\boldsymbol{\Sigma}}_t &= \mathbf{A}_t\,\boldsymbol{\Sigma}_{t-1}\,\mathbf{A}_t^\top + \mathbf{R}_t, && (2.4)\\
\mathbf{K}_t &= \overline{\boldsymbol{\Sigma}}_t\,\mathbf{C}_t^\top\,(\mathbf{C}_t\,\overline{\boldsymbol{\Sigma}}_t\,\mathbf{C}_t^\top + \mathbf{Q}_t)^{-1}, && (2.5)\\
\boldsymbol{\mu}_t &= \overline{\boldsymbol{\mu}}_t + \mathbf{K}_t\,(\mathbf{z}_t - \mathbf{C}_t\,\overline{\boldsymbol{\mu}}_t), && (2.6)\\
\boldsymbol{\Sigma}_t &= (\mathbf{I} - \mathbf{K}_t\,\mathbf{C}_t)\,\overline{\boldsymbol{\Sigma}}_t, && (2.7)
\end{aligned}$$

where Equation (2.3) and (2.4) define the prediction step of the filter, which estimates the new state of the system based on the transition model; Equation (2.6) and (2.7) define the update step, which corrects the prediction according to the observation; the matrix $\mathbf{K}_t$ is the so-called Kalman gain and determines the weight given to the measurement with respect to the predicted estimate.

Most real-world systems are non-linear and the assumptions of linear transition and observation models of the Kalman filter do not hold. The Extended Kalman filter (EKF) relaxes the linearity assumption and allows for considering systems that follow transition and observation models defined as:

$$\begin{aligned}
\mathbf{x}_t &= g(\mathbf{x}_{t-1}, \mathbf{u}_t) + \boldsymbol{\epsilon}_t, && (2.8)\\
\mathbf{z}_t &= h(\mathbf{x}_t) + \boldsymbol{\delta}_t, && (2.9)
\end{aligned}$$

where g and h are non-linear functions. The non-linear functions lead to non-Gaussian distribution and, thus, the KF is not applicable. The key idea of the EKF is to locally linearize the functions g and h by using the first order Taylor expansion. The Taylor expansion offers a linear approximation of the non-linear functions as:

$$\begin{aligned}
g(\mathbf{x}_{t-1}, \mathbf{u}_t) &\approx g(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t) + \mathbf{G}_t\,(\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}), && (2.10)\\
h(\mathbf{x}_t) &\approx h(\overline{\boldsymbol{\mu}}_t) + \mathbf{H}_t\,(\mathbf{x}_t - \overline{\boldsymbol{\mu}}_t), && (2.11)
\end{aligned}$$

where G_t and H_t are the Jacobians consisting of the partial derivatives:

$$\mathbf{G}_t = \frac{\partial g(\mathbf{x}_{t-1}, \mathbf{u}_t)}{\partial \mathbf{x}_{t-1}}, \quad \mathbf{H}_t = \frac{\partial h(\overline{\boldsymbol{\mu}}_t)}{\partial \mathbf{x}_t}. \tag{2.12}$$

Given an initial belief $bel(\mathbf{x}_{t-1}) = \mathcal{N}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$, the EKF computes a Gaussian approximation of the true belief $bel(\mathbf{x}_t) \approx \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ according to:

$$\begin{aligned}
\overline{\boldsymbol{\mu}}_t &= g(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t), && (2.13)\\
\overline{\boldsymbol{\Sigma}}_t &= \mathbf{G}_t\,\boldsymbol{\Sigma}_{t-1}\,\mathbf{G}_t^\top + \mathbf{R}_t, && (2.14)\\
\mathbf{K}_t &= \overline{\boldsymbol{\Sigma}}_t\,\mathbf{H}_t^\top\,(\mathbf{H}_t\,\overline{\boldsymbol{\Sigma}}_t\,\mathbf{H}_t^\top + \mathbf{Q}_t)^{-1}, && (2.15)\\
\boldsymbol{\mu}_t &= \overline{\boldsymbol{\mu}}_t + \mathbf{K}_t\,(\mathbf{z}_t - h(\overline{\boldsymbol{\mu}}_t)), && (2.16)\\
\boldsymbol{\Sigma}_t &= (\mathbf{I} - \mathbf{K}_t\,\mathbf{H}_t)\,\overline{\boldsymbol{\Sigma}}_t. && (2.17)
\end{aligned}$$

Algorithm 1 Discrete Markov localization

1: **procedure** MarkovLocalization ($bel(\mathbf{x}_{t-1})$, $\mathbf{u}_t$, $\mathbf{z}_t$, $\mathcal{X}$)
2: **for** all $\mathbf{x}_t \in \mathcal{X}$ **do**
3: $\overline{bel}(\mathbf{x}_t) \leftarrow \sum_{x_{t-1}} p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t, \mathbf{z}_t, \mathcal{X})\, bel(\mathbf{x}_{t-1})$ ▷ Prediction step.
4: $bel(\mathbf{x}_t) \leftarrow \eta\, p(\mathbf{z}_t \mid \mathbf{x}_t, \mathcal{X})\, \overline{bel}(\mathbf{x}_t)$ ▷ Correction step.
return $bel(\mathbf{x}_t)$

The EKF is a commonly used technique to perform robot localization. EKF localization considers a robot that moves according to a motion model defined similarly as Equation (2.8) and that obtains measurements of the environment according to an observation function defined similarly as Equation (2.9). It uses the motion model and the sensor measurements to recursively estimate the pose of the robot in a map of the environment by representing the belief by its mean and covariance.

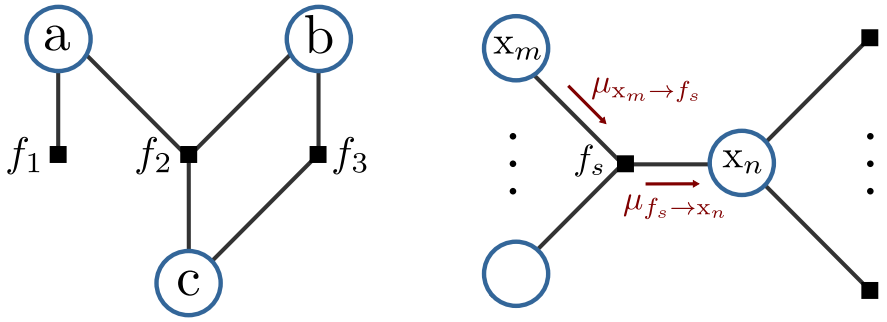
2.1.2 Markov localization

Markov localization [46] implements a Bayes filter for recursively estimating the position of the robot in a given map. It uses a histogram filter and, therefore, does not require to restrict the belief to any particular class of distributions. Although Markov localization is independent from the representation of the state space, we consider discrete grid-based representations of the environment for localization throughout this thesis. A common grid-based representation is occupancy grid maps, which consist of a set of cells that can be free, occupied, or unknown. On an occupancy grid map $\mathcal{X}$, Markov localization approximates the belief about the position of the robot by maintaining a collection of discrete probability values:

$$bel(\mathbf{x}_t) \;=\; \{p_{k,t}\}, \quad \text{with} \quad \sum_k p_{k,t} \;=\; 1, \tag{2.18}$$

where each probability $p_{k,t}$ corresponds to a cell $\mathbf{x}_k \in \mathcal{X}$ and represents the probability that the robot is located at that cell. Therefore, Markov localization represents the belief in form of an histogram defined over $\mathcal{X}$.

Given an initial belief about the position of the robot $bel(\mathbf{x}_{t-1})$, the control $\mathbf{u}_t$, the observation $\mathbf{z}_t$, and the grid map of the environment $\mathcal{X}$, Markov localization computes the belief at time t by using the procedure described in Algorithm 1. For each cell $\mathbf{x}_t \in \mathcal{X}$, it first predicts an estimate $\overline{bel}(\mathbf{x}_t)$ by using the robot motion model (line 3), which specifies the posterior probability on $\mathbf{x}_t$ of executing the control $\mathbf{u}_t$ from $\mathbf{x}_{t-1}$. Then, it corrects the prediction by integrating it over the observation model (line 4), which depends on the sensors used, and normalizes the final probabilities using the normalizer η. The result is a new discrete probability



(a) Factor graph. (b) Belief propagation.

Figure 2.1: Example of factor graph, where the circles are the variable nodes and the squares the factor node, and the message passing procedure of the belief propagation algorithm to perform inference.

distribution $bel(\mathbf{x}_t)$ defined as a histogram over $\mathcal{X}$ that represents the belief about the position of the robot at time t.

2.2 Probabilistic inference

Probabilistic inference is one class of state estimation techniques. It uses data to derive probabilistic models that represent underlying probability distributions over the state. In this thesis, we use probabilistic inference for modeling and predicting partially observable phenomena.

2.2.1 Factor graphs

Probabilistic graphical models [74] are models that allow for representing complex probability densities with a compact graph structure. A factor graph [81] is a probabilistic graphical model that can be used to represent any general factorization of a function. It is structured as a bipartite undirected graph with two kinds of nodes: the factor nodes and the variable nodes. The variable nodes correspond to random variables, whereas the factor nodes represent local functions with the adjacent variable nodes as inputs. A variable node can only be connected to one or multiple factor nodes but not directly to another variable node. A simple factor graph is illustrated in Figure 2.1(a) where the blue circles are the variable nodes and the black squares the factor nodes.

A factor graph allows, in general, for computing a joint distribution over a set of variables $\boldsymbol{X} = \{\mathrm{x}_1, \ldots, \mathrm{x}_n\}$ as the product of factors:

$$p(\mathrm{x}_1, \ldots, \mathrm{x}_n) = \prod_s f_s(\boldsymbol{X}_s), \tag{2.19}$$

where a factor f_s is a local function of the set of variables $\boldsymbol{X}_s \subseteq \boldsymbol{X}$. For example, the factor graph in Figure 2.1(a) expresses the factorization:

$$p(\mathrm{a},\, \mathrm{b},\, \mathrm{c}) \;=\; f_1(\mathrm{a})\, f_2(\mathrm{a},\, \mathrm{b},\, \mathrm{c})\, f_3(\mathrm{b},\, \mathrm{c}). \tag{2.20}$$

Probabilistic graphical models allow for performing inference, i.e., estimating the probability that one or a set of variables takes a certain value. Given the observed values of some variables of a factor graph, the belief propagation algorithm [131] implements a message passing procedure that exploits the graph structure to estimate the maximum-a-posteriori configuration and the marginal posterior distribution of each variable. The message passing procedure of belief propagation is sketched in Figure 2.1(b). Messages are results of partial computations. A message $\mu_{\mathrm{x}_m \to f_s}$ from a variable node x_m to a factor node f_s is the product of all incoming messages except the message from the receiving factor:

$$\mu_{\mathrm{x}_m \to f_s} \;=\; \prod_{f_i \in \mathbf{f} \setminus \{f_s\}} \mu_{f_i \to \mathrm{x}_m}, \tag{2.21}$$

where $\mathbf{f}$ is the set of factor nodes. Whereas, a message $\mu_{f_s \to \mathrm{x}_n}$ from a factor f_s to a variable node x_n is the product of the messages from all other nodes to f_s, marginalized over all variables except the receiving one:

$$\mu_{f_s \to \mathrm{x}_n} \;=\; \sum_{\boldsymbol{X}_s} f_s(\boldsymbol{X}_s,\, \mathrm{x}_n) \prod_{\mathrm{x}_i \in \mathrm{Neig}(f_s) \setminus \{\mathrm{x}_n\}} \mu_{\mathrm{x}_i \to f_s}, \tag{2.22}$$

where $\boldsymbol{X}_s$ is the set of variables connected to x_n via f_s and $\mathrm{Neig}(f_s)$ denotes the variable nodes that are neighbors of x_n. Marginals are computed as the product of all incoming messages from neighbor factors:

$$p(\mathrm{x}_i) \;=\; \prod_{f_j \in \mathrm{Neig}(\mathrm{x}_i)} \mu_{f_j \to \mathrm{x}_i}. \tag{2.23}$$

The belief propagation algorithm performs exact inference on tree-structured graphs without loops, but it is able to approximate inference also in graphs with loops by applying the same procedure until convergence. Although there is no guarantee of convergence, loopy belief propagation has been demonstrated to yield good results in practice for several problems such as visual odometry estimation [94], simultaneous localization and mapping [141], and traffic prediction [52].

2.2.2 Gaussian process regression

A Gaussian process (GP) [142] is a collection of infinitely many random variables that forms a joint multivariate Gaussian distribution. GPs are often used for regression to estimate a distribution over all the possible functions that are consistent with a set of observed training data. GP regression does not assume that

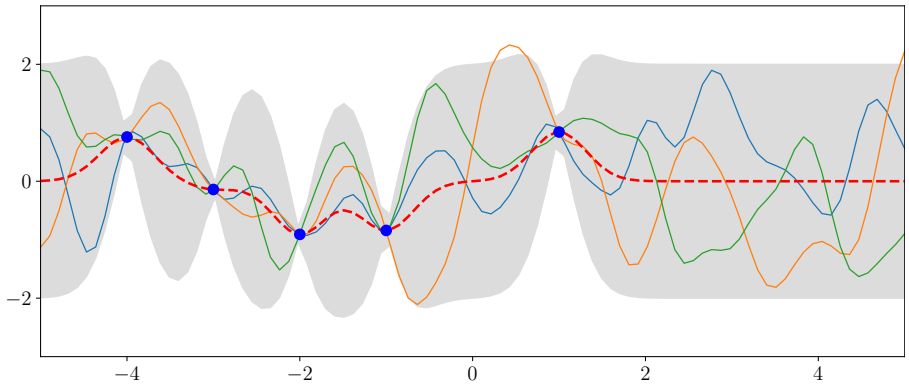


Figure 2.2: Gaussian process posterior computed from observations (blue circles) and functions sampled accordingly (solid colored lines). The red dotted line is the mean of the predictive distribution and the gray area represents its 2σ confidence interval.

the data distribution can be defined with a finite set of parameters and, thus, it is typically referred to as a non-parametric approach.

A GP is defined by a mean function $m(x)$ and a covariance function $k(x, x')$, also called kernel, which defines the similarity between data points and encodes a prior over the model. Given a set of observations $\mathbf{y}$ of a function f for the inputs $\mathbf{x}$, GP regression allows for learning a predictive model of f at the query inputs $\mathbf{x}_*$ by assuming a joint Gaussian distribution over the samples. Therefore, we can write the joint distribution of the target values $\mathbf{y}$ and the function at the query inputs $\mathbf{f}_*$ under this prior as:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} = \mathcal{N}\left(m\begin{pmatrix} \mathbf{x} \\ \mathbf{x}_* \end{pmatrix}, \begin{bmatrix} \mathbf{K}(\mathbf{x}, \mathbf{x}) + \varsigma_n^2 \mathbf{I} & \mathbf{K}(\mathbf{x}, \mathbf{x}_*) \\ \mathbf{K}(\mathbf{x}_*, \mathbf{x}) & \mathbf{K}(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix} \right), \tag{2.24}$$

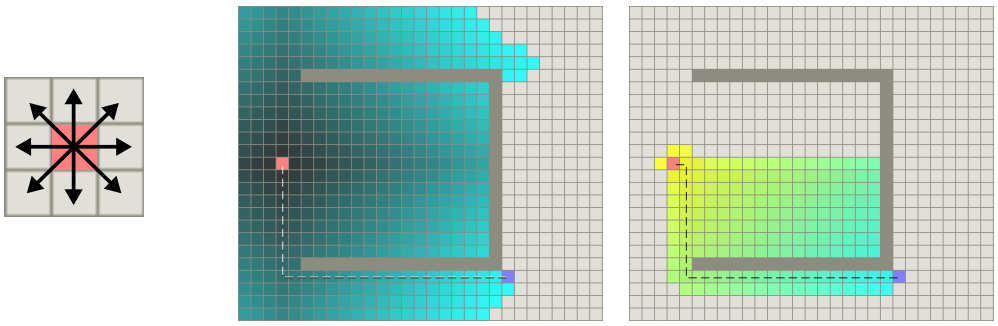
where $\mathbf{K}(\cdot, \cdot)$ are matrices constructed using the covariance function and ς_n^2 is the Gaussian noise variance. The prediction at a test point $x_* \in \mathbf{x}_*$ is computed by conditioning the prior on the observations $\mathbf{y}$ and it is represented by the predictive mean μ_* and variance σ_*^2 defined as:

$$\mu_* = \mathbf{k}_*{}^\top \mathbf{K}_{\mathrm{XX}}{}^{-1} \mathbf{y}, \tag{2.25}$$

$$\sigma_*^2 = k(x_*, x_*) - \mathbf{k}_*{}^\top \mathbf{K}_{\mathrm{XX}} \mathbf{k}_*, \tag{2.26}$$

where $\mathbf{K}_{\mathrm{XX}} = \mathbf{K}(\mathbf{x}, \mathbf{x}) + \varsigma_n^2 \mathbf{I}$ represents the covariance between the training data points, whereas $\mathbf{k}_* = k(x_*, \mathbf{x})$ is the covariance between the test and training data points.

Figure 2.2 shows an example GP computed from observed data points (blue circles). The predictive mean is represented by the red dotted line, the grey area shows the 2 standard deviations range from the mean, and the colored solid lines are possible functions representing the data points sampled from the GP posterior distribution.



(a) N8-neighborhood defining edges on a grid. (b) Dijkstra's algorithm. (c) A* search algorithm.

Figure 2.3: Shortest path planning to navigate from start (red cell) to goal (blue cell) on a grid map. The resulting path is the black dotted line and the colored cells are the expanded ones during the search. Images courtesy of A. Patel [130].

A common choice is to assume that the mean function of the GP is zero, i.e., $m(x) = 0$, and to use the squared exponential covariance function:

$$k(x_i, x_j) \quad = \quad \varsigma_f^2 \cdot \exp\left(-\frac{1}{2}\frac{|x_i - x_j|^2}{\ell^2}\right) + \delta_{ij}\varsigma_n^2, \tag{2.27}$$

where $\boldsymbol{\theta} = \{\ell, \varsigma_f^2, \varsigma_n^2\}$ are the so-called hyperparameters, and represent respectively the length scale ℓ, the variance of the output ς_f^2, and of the noise ς_n^2. Typically, the hyperparameters of a GP are learned from the training data by maximizing the log marginal likelihood:

$$\log p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^\top \mathbf{K}_{\mathrm{XX}}\,\mathbf{y} - \frac{1}{2}\log|\,\mathbf{K}_{\mathrm{XX}}\,| - \frac{n}{2}\log(2\pi). \tag{2.28}$$

GP regression is a popular technique also in geostatistics, where it is known as kriging [107]. Kriging is a spatial interpolation method that is typically used to perform inference for physical quantities in unobserved locations on two or three-dimensional input spaces.

2.3 Path planning

Path planning is the problem of finding a route between two points in the traversable configuration space. For robot navigation, planning defines the trajectory along which the robot should navigate to reach a targeted location without colliding with obstacles.

2.3.1 Shortest path planning

Motion planning algorithms for robot navigation have been widely studied [92]. The most common planners are the shortest path planning algorithms. These

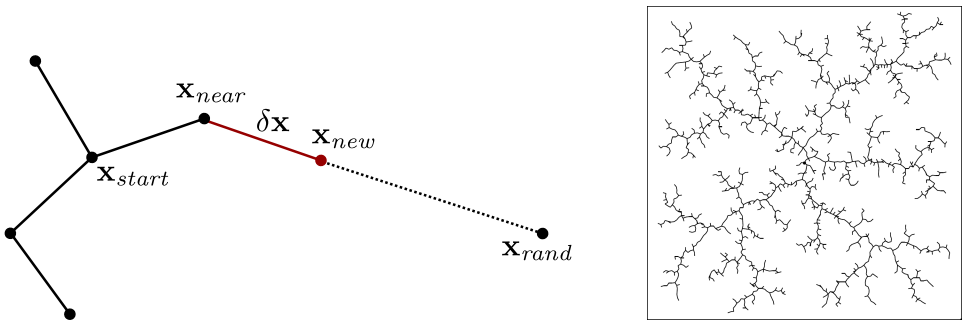
Algorithm 2 Dijkstra's path finding algorithm

```
1:  procedure DIJKSTRA(x_start, x_goal, X)
2:      for each x ∈ X do
3:          gcost(x) ← ∞                          ▷ Initialize data structures.
4:          parent(x) ← ∅
5:          Q.Add(X, gcost)
6:      gcost(x_start) ← 0
7:      while ¬ IsEmpty(Q) do
8:          x ← Pop(Q)                            ▷ Node with minimum gcost.
9:          if IsGoal(x, x_goal) then
10:             path ← ReconstructPath(x, parent)  ▷ Backtrack path.
11:         for each x_neig ∈ neig(x) do
12:             tmp_gcost ← gcost(x_neig)          ▷ Tentative new gcost.
13:             if tmp_gcost < gcost(x_neig) then
14:                 parent(x_neig) ← x             ▷ Update parent.
15:                 gcost(x_neig) ← tmp_gcost      ▷ Update gcost.
```

algorithms stem from the shortest path problem in graph theory. This is the problem of finding the route between two nodes of a weighted graph such that the sum of the weights along the edges is minimized.

The shortest path algorithms are commonly used to find paths for robot navigation on static geometrical representation of the environment, such as a topological graph or a 2D occupancy grid map. Occupancy grid maps can be seen as graphs in which each cell has edges connecting it to its neighbors, as illustrated in Figure 2.3(a). Graph-based shortest path algorithms allow for formulating path planning as an explicit cost minimization problem and to compute paths that minimize travel distance, travel time, or any defined cost to navigate between two locations.

The most popular shortest path algorithm is the Dijkstra's algorithm [35], which is described in Algorithm 2. The algorithm maintains in a priority queue $\mathbf{Q}$ a set of candidate nodes sorted according to the cost of the path from the start to the node itself, referred to as gcost. At each step of the search, it selects the node $\mathbf{x}$ that minimizes the $gcost(\mathbf{x})$ (line 8). If it is the goal, the path is retrieved by recursively backtracking the parent nodes of the goal up to the start (line 10). Otherwise, it attempts to update the gcost for the neighbors of the current node (line 12). If the current node leads to a smaller gcost, it updates its cost and parent node (line 14). The search terminates when the goal has been reached or all of the nodes have been examined. If all edges have positive costs, Dijkstra's algorithm is guaranteed to find the shortest (or minimum cost) path.



(a) RRT extending branch. (b) RRT search tree.

Figure 2.4: RRT planning and its branch extension procedure to explore the configuration space from the start position after a few iterations. Images courtesy of S. LaValle and J. Kuffner [91].

The A* search is a popular algorithm that has been developed by Nilsson for "Shakey the robot" [123], which implements an informed search to speed up the planning process. A* search follows the same procedure as Dijkstra's but selects nodes by using a best-first strategy. It selects the node to expand from the set of candidate nodes based on an evaluation function $feval(\mathbf{x})$ which estimates the cheapest path to the goal through the node $\mathbf{x}$:

$$feval(\mathbf{x}) \quad = \quad gcost(\mathbf{x}) \; + \; hcost(\mathbf{x}), \tag{2.29}$$

where $gcost(\mathbf{x})$ is the cost of the path from start to $\mathbf{x}$ and $hcost(\mathbf{x})$ is a heuristic function which estimates the cost from $\mathbf{x}$ to the goal node. The heuristic function informs the search by injecting domain knowledge. Selecting the node with the lowest feval allows for expanding first the more promising nodes and, therefore, to reach earlier the goal node.

A heuristic function is admissible if it never overestimates the true cost. Therefore, the following condition must hold $\forall\mathbf{x}$:

$$hcost(\mathbf{x}) \quad \leq \quad gcost(\mathbf{x}). \tag{2.30}$$

If the heuristic is admissible, A* search is complete and provides optimal paths. The closer the heuristic approximates the true cost, the more efficient is the search. If the heuristic is set to zero, A* is equivalent to Dijkstra's algorithm. Figure 2.3 illustrates the cells expanded by Dijkstra's and A* for a simple path planning task. As can be seen, A* typically requires to expand substantially fewer cells than Dijkstra's to find the optimal path.

2.3.2 Sampling-based planning

Sampling-based planning is a class of algorithms that allows for computing paths without an explicit representation of the configuration space. Rather than considering a map of the environment where cells are free or occupied, these algorithms

Algorithm 3 Rapidly exploring random trees planning

```
1:  procedure RRT (x_start, x_goal, δx, max_iter)
2:      G ← x_start                                   ▷ Initialize search tree.
3:      repeat
4:          x_rand ← SampleConfig()
5:          x_near ← GetNearestVertex(G, x_rand)      ▷ Extending branch.
6:          x_new ← GetCandidateVertex(x_rand, x_near, δx)
7:          if IsValid(x_near, x_new) then
8:              G ← AddVertex(x_new)                  ▷ Update tree.
9:              G ← AddEdge(x_near, x_new)
10:         i ← i + 1
11:     until HasReached(G, x_goal) ∨ i > max_iter    ▷ Termination conditions.
```

rely on a collision detection module that checks whether a location belongs to the free configuration space or not. Given a start and goal location, sampling-based planning algorithms sample states in the configuration space and connect them locally in order to find a collision-free path.

One of the most popular sampling-based planning algorithms is the rapidly exploring random trees algorithm or RRT. The idea of RRT is to incrementally expand a search tree from an initial configuration by randomly sampling the configuration space. The RRT algorithm is described in Algorithm 3. The search starts by initializing a tree $\mathbf{G}$ rooted in the start configuration (line 2). RRT iteratively explores the configuration space by randomly sampling a configuration $\mathbf{x}_{rand}$ (line 4) and extending the tree towards this state. The branch extension procedure is illustrated in Figure 2.4(a). It queries the tree for the closest vertex to $\mathbf{x}_{rand}$ (line 5), $\mathbf{x}_{near}$, and selects the configuration $\mathbf{x}_{new}$ that lies along the line connecting the two vertices at a fixed maximum distance $\delta\mathbf{x}$ from $\mathbf{x}_{near}$ (line 6). The collision detector checks whether the edge between $\mathbf{x}_{new}$ and $\mathbf{x}_{near}$ is collision-free and, if this is the case, RRT adds the new vertex $\mathbf{x}_{new}$ and the corresponding edge to the search tree $\mathbf{G}$ (line 8). Figure 2.4(b) shows the search tree of RRT after few iterations. The search terminates when the goal configuration $\mathbf{x}_{\text{goal}}$ has been reached or a maximum number of iterations *max_iter* has been performed. A common method to speed-up the search and find a path to the goal is to use a sampling strategy that either randomly samples states from the configuration space or selects the goal with a fixed probability [93].

An efficient variant of RRT is bi-directional RRT. It grows two search trees one rooted in the start and one in the goal configuration. At every iteration, one tree attempts to extend towards the newest vertex of the other tree. The search terminates when the trees are connected and a valid path from start to the goal has been found. Bi-directional RRT is typically faster RRT as it gives

less importance to exploring the configuration space but biases the tree extension to connect the start to the goal configuration.

Sampling-based planning is generally more efficient than shortest path approaches to find valid paths as it does not explore every possible configuration but approximates the search by performing sampling in the configuration space. Another advantage of these approaches is that they do not require to explicitly construct the configuration space and, thus, maintain an efficient representation for planning even in high dimensional spaces. However, RRT typically does not provide optimal paths and is only probabilistic complete, which means that a solution will be provided, if it exists, given an infinite number of iterations.

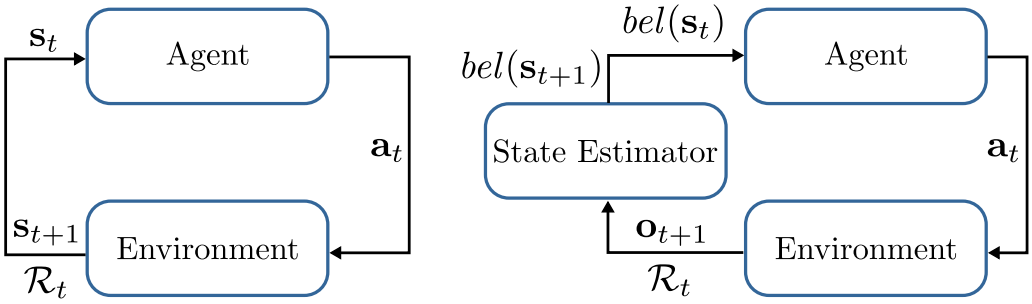
2.4 Decision-making under uncertainty

Traditional path planning algorithms as the ones described in the previous section assume known environments, full observability, and a deterministic world. However, in the real world, the robot's belief about the environment, its perception and its action execution are uncertain. Therefore, the capability to make decisions under uncertainty is crucial for realizing robust robot navigation.

2.4.1 Markov decision process

Markov decision process or MDP [139] is a discrete stochastic process for modeling sequential decision problems. MDPs allow for modeling problems in which the state that characterizes the problem is always known and fully observable but the outcome of the agent's decisions is non-deterministic and is described by a probability distribution over the successor states. MDPs rely on the Markov assumption: the different outcomes of an action do not depend on the full history of the agent's decisions but only on the current state, which is assumed to capture all of the relevant information to make decisions.

Formally, a Markov decision process is a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{T}$ defines a transition model between states, and $\mathcal{R}$ represents the immediate reward for each transition. At each time step t, the process is in a state $\mathbf{s}_t \in \mathcal{S}$ that represents the world, and the agent has to decide which action $\mathbf{a}_t \in \mathcal{A}$ to take. The transition model $\mathcal{T}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ specifies the probability of success of the agent to move to a state $\mathbf{s}'$ by executing an action $\mathbf{a}$ from the state $\mathbf{s}$. Thus, the next state depends only on the current state and on the action taken. For every new state reached by the agent, it receives a reward defined by the reward function $\mathcal{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$. The reward can be positive or negative but must be bounded. The decision-making scheme of MDPs is illustrated in Figure 2.5(a).



(a) MDP. (b) POMDP.

Figure 2.5: Decision-making scheme of fully observable and partially observable Markov decision process. In MDPs the state is always known while, in POMDPs, it only noisy observations of it can be obtained.

Given an MDP, the objective is to find a strategy to make decisions in the world that maximizes the rewards. The solution cannot be a sequence of actions as for path planning because the non-deterministic outcome of the actions may lead the agent in any state different from the goal. Therefore, the solution of an MDP is a policy that specifies which action to take in each of the states of the world. The optimal policy π^* is the one that chooses at each state the action that maximizes the expected utility of the subsequent states:

$$\pi^*(\mathbf{s}) \quad = \quad \underset{\mathbf{a}}{\operatorname{argmax}} \sum_{\mathbf{s}'} \mathcal{T}(\mathbf{s},\, \mathbf{a},\, \mathbf{s}')\, U(\mathbf{s}'), \tag{2.31}$$

where the expected utility $U(\mathbf{s})$ for a state $\mathbf{s}$ is the discounted sum of the rewards obtained by executing the policy π:

$$U(\mathbf{s}) \quad = \quad E\left[\sum_{t=0}^{\infty} \gamma^t\, \mathcal{R}(\mathbf{s}_t, \mathbf{a}, \mathbf{s}_{t+1}) \mid \pi(\mathbf{s}_t) = \mathbf{a}, \mathbf{s}_0 = \mathbf{s}\right], \tag{2.32}$$

where $\gamma \in [0, 1]$ is called the discount factor and describes the preference for immediate rewards over future rewards.

The most popular algorithms to compute optimal solutions for MDPs are value iteration and policy iteration. Value iteration [8] computes the utility of each state by iteratively using the Bellman equation:

$$U(\mathbf{s}) \quad = \quad \max_{\mathbf{a}} \sum_{\mathbf{s}'} \mathcal{T}(\mathbf{s}, \mathbf{a}, \mathbf{s}')\, [\mathcal{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma U(\mathbf{s}')]\,, \tag{2.33}$$

until the utilities converge. It then uses the computed utilities to select an optimal action in each state. Policy iteration [60], instead, starts from a random policy and iteratively computes the utility of each state for the current policy and calculates a new policy based on the computed utilities until convergence. Policy iteration requires fewer iterations that value iteration to converge and, therefore, it is often faster to find an optimal policy.

2.4.2 Partially observable Markov decision process

MDPs assume that the environment is fully observable. However, for many problems such as robot navigation the state of the world is not perfectly known. For example, the localization approaches introduced in Section 2.1 do not provide a deterministic state but estimate the position of the robot by computing a probability distribution over the state space. Partially observable Markov decision process [4], or POMDPs, formalize decision-making problems in which the environment is not directly observable but measurements of it can be obtained. POMDPs assume that the system dynamics are determined by an MDP but allow for modeling problems in which the state is uncertain due to noisy partial observations as well as the actions.

A partially observable Markov decision process is a 6-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \Omega \rangle$ where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{R}$ are respectively the set of states, actions, the transition and the reward functions as in an MDP, $\mathcal{O}$ is the set of observations, and Ω specifies the probability to make an observation in a given state. At each time step t, the agent maintains a belief over the states $\mathcal{S}$ that summarizes its experience. According to the current belief $bel(\mathbf{s}_t)$, the agent decides which action $\mathbf{a}_t$ to take. The state of the world is not fully observable but the effects of the action on the world are captured by an observation $\mathbf{o}_{t+1} \in \mathcal{O}$. The agent uses a state estimator to update the belief state $bel(\mathbf{s}_{t+1})$ by considering the last action $\mathbf{a}_t$, the current observation $\mathbf{o}_{t+1}$, and the previous belief $bel(\mathbf{s}_t)$ as:

$$bel(\mathbf{s}_{t+1}) \;=\; \eta\, \Omega(\mathbf{s}_{t+1},\, \mathbf{o}_{t+1}) \sum_{s} \mathcal{T}(\mathbf{s}_t,\, \mathbf{a}_t,\, \mathbf{s}_{t+1})\, bel(\mathbf{s}_t), \tag{2.34}$$

where η is a normalizing constant. The decision-making scheme of POMDPs is illustrated in Figure 2.5(b).

In POMDPs, the agent has to make decisions based on beliefs. Each belief is a probability distribution over the states and, thus, the belief state space is continuous. This makes POMDPs in general computationally complex and hard to solve. POMDPs have been demonstrated to be PSPACE-complete [128] and, thus, are intractable for most of large real-world problems. In Chapter 3, we will focus on a tailored approximation of POMDPs that allows for planning for robot navigation in real-world scenarios.

Part I

Navigation exploiting background knowledge

Chapter 3

Navigation under uncertainty exploiting localizability

Although most robots use probabilistic algorithms to solve state estimation problems such as localization or mapping, planning systems often compute paths and make decisions by assuming that the world is fully known and, therefore, ignore uncertainty. Uncertainty, however, matters for planning in real-world environments, but considering it often leads to computationally expensive algorithms.

In practice, many planning approaches, such as traditional shortest path planners, compute paths by ignoring the uncertainty about the robot's position provided by the localization system. Ignoring the position uncertainty during planning may be acceptable if the robot is precisely localized but it can lead to sub-optimal navigation decisions if the uncertainty is large.

Consider, for example, a mobile robot navigating on a road network that encounters the situation depicted in Figure 3.1. The robot is following the shortest path to the goal (solid blue), and the current belief about its position is represented by the black shaded area (the darker, the more likely). This belief indicates that the robot could be in the proximity of intersection A or B. These intersections present surroundings with a similar structure, as illustrated on the right side of Figure 3.1. Thus, a localization system based on range sensors might not be able to disambiguate them. Planning algorithms that ignore the position uncertainty typically assume that the robot is at the most likely position that, in this example, is intersection B. Therefore, under this assumption, the robot should turn to the right (dotted orange path) in order to reach the goal along the shortest path. However, if the robot is in reality at intersection A (less likely, but possible), turning right would lead it to a long detour (dash-dotted red path) and, thus, to follow a trajectory that is very different from the planned shortest path.

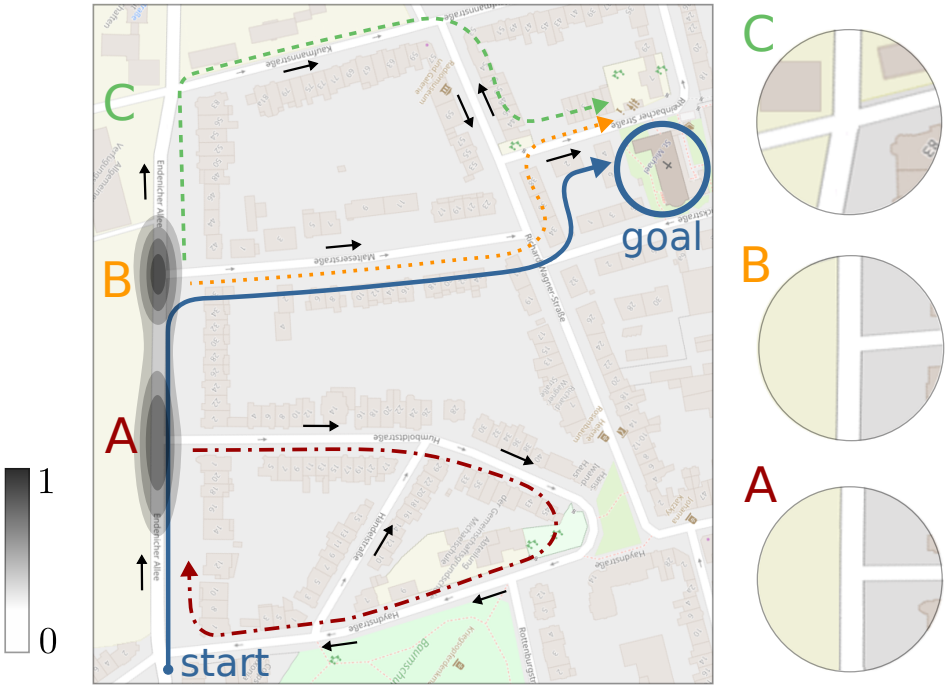


Figure 3.1: Robot navigation on a road network under large position uncertainty. The black shaded area is the belief about the position of the robot (the darker, the more likely). The blue path is the shortest path to the goal. A, B, C are the road intersections that are shown in detail on the right side of the figure. The paths from each intersection are colored respectively in red, orange, and green. The black arrows indicate the roads' directions.

In this chapter, we investigate the problem of planning paths by taking into account the uncertainty in the robot's belief about the world, in its perception, and action execution. Uncertainty-aware plans reduce the risk of making wrong turns when the uncertainty is large. For example, in the situation of Figure 3.1, the robot could navigate towards intersection C, which has a distinctive surrounding and, thus, is a place where the robot is expected to localize better. At C, it can safely turn right towards the goal, thus avoiding the risk of taking longer detours (dashed green path). We introduce in this chapter an approach that reasons about the uncertainty and computes this type of plans.

A general formalization for incorporating uncertainty in the planning problem is the partially observable Markov decision process (POMDP), see Section 2.4.2 for a brief description. POMDPs, however, lead typically to algorithms that are computationally intractable for most real-world applications. We investigate an approximation of POMDPs that is able to take into account the uncertainty while being computationally efficient. We propose the use of an uncertainty-augmented Markov decision process that models the uncertainty as part of the state to approximate the underlying POMDP. We furthermore employ a localization prior called localizability to estimate how the uncertainty propagates along the road network. Solving this process yields navigation policies that allow robots to make

decisions according to the degree of uncertainty. Following these policies reduces the number of mistakes that the robot makes during navigation with large position uncertainty and, thus, the expected travel time to the goal.

3.1 Navigation on road networks

In this chapter, we focus on uncertainty-aware path planning for robots navigating on road networks. Navigation in urban environments has received substantial attention in the robotics community. Over the past decades, several robotic systems for urban navigation have been developed such as Obelix [89], Boss [166], or the Autonomous City Explorer [95]. This trend recently increased even more with the growing interest in self-driving cars.

In the following, we introduce the target robotic platform, the environment representation, and the localization system that we consider throughout this chapter to navigate in urban environments. Furthermore, we describe a baseline approach to plan minimum travel distance routes on road networks ignoring the uncertainty about the position of the robot.

3.1.1 Robotic platform

Our target robotic platform is a wheeled mobile robot such as a self-driving car that navigates in an urban environment by driving along roads and following their directions. Our robot is equipped with a 360° 2D LiDAR sensor and has also access to the readings of the wheel odometry.

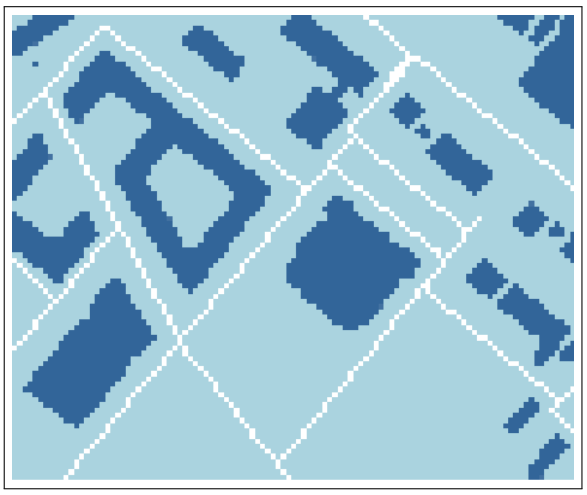
3.1.2 Topo-metric maps

Navigation in urban environments requires an appropriate representation of the environment. While several probabilistic approaches for robot localization rely on occupancy grid maps, topology graphs are an effective representation for planning. We combine these two representations and represent the environment using a topo-metric map, similar to hierarchical maps [75].

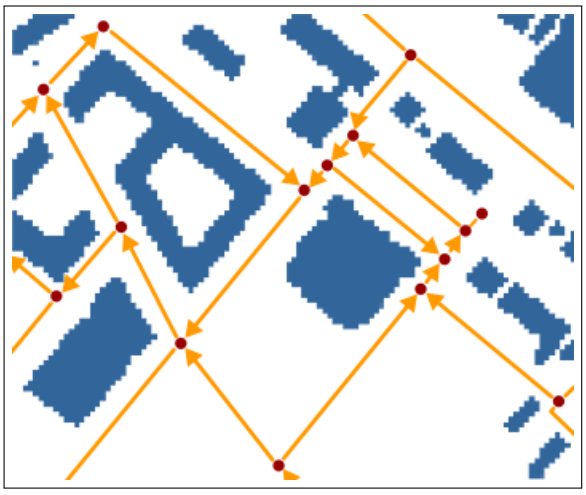
Publicly available map services, such as Google Maps[1] and OpenStreetMap[2], offer an informative representation of most of the urban environments including metric and topological information. We define our environment representation by extracting information about buildings as well as roads and their directions from OpenStreetMap, see for example Figure 3.2(a). We store this information in a 2D grid map $\mathcal{X}$ in which each cell contains information about its traversability,

[1] http://maps.google.com
[2] http://openstreetmap.org

(a) OpenStreetMap representation.



(b) Traversability grid map $\mathcal{X}$.



(c) Road network topology $\mathcal{G}$.

Figure 3.2: Our topo-metric representation of the environment combining a grid map $\mathcal{X}$ and a topological graph $\mathcal{G}$ extracted from OpenStreetMap (a). In (b), the traversable roads are in white, blue refers to non-traversable areas and buildings are in dark blue. In (c), the orange arrows are the roads $\boldsymbol{E}$ and the red dots are their intersections $\boldsymbol{V}$.

as the one illustrated in Figure 3.2(b). As our robot can move only along roads, the traversable area corresponds to the the roads, which are represented as the white cells in Figure 3.2(b). Each traversable cell additionally contains a set of links to the neighboring cells where the robot can move to. Our robot uses such a grid map representation for localization. In addition, we consider a more abstract representation of the environment for planning routes. We define a topological graph $\mathcal{G} = (\boldsymbol{V}, \boldsymbol{E})$ defined over the discretized metric space $\mathcal{X}$, see for example Figure 3.2(c). In $\mathcal{G}$, the vertices $\boldsymbol{V}$ correspond to the cells in $\mathcal{X}$ where two or more roads intersect. The vertices $\boldsymbol{V}$ are connected with oriented edges $\boldsymbol{E}$ that represent the roads and are oriented according to their directions. Therefore, one edge in $\mathcal{G}$ corresponds to a sequence of traversable cells in $\mathcal{X}$.

3.1.3 Markov localization

We estimate the robot position in $\mathcal{X}$ by using a localization system based on Markov localization [46] which relies on the onboard LiDAR and the wheel odometry readings. As the robot moves and acquires a new measurement from the LiDAR, the localization system uses this measurement and the odometry readings to compute a new estimate of the robot's position. The Markov localization algorithm is described in Section 2.1.2. The main difference to most Markov localization systems is that, as we assume the robot to move only along roads, we consider as possible robot locations only the cells of the grid map $\mathcal{X}$ along the roads and not the whole 2D plane.

Note that other approaches for state estimation such as particle filter could be used in the navigation system presented in this chapter without loss of generality. We refer to Markov localization because it is the most general approach for probabilistic estimation of the robot position.

3.1.4 MDP planning

Given our topological representation of the environment $\mathcal{G}$, we can compute the route that leads the robot to the goal along the shortest path by formalizing the planning problem as a Markov decision process, or MDP [139]. We shortly defined MDPs in Section 2.4.1. We can find a policy to reach a desired location by defining an MDP as the 4-tuple $\mathcal{MDP} = \langle \mathcal{S}_{\text{MDP}}, \mathcal{A}_{\text{MDP}}, \mathcal{T}_{\text{MDP}}, \mathcal{R}_{\text{MDP}} \rangle$, where the states are the road intersections:

$$\mathcal{S}_{\text{MDP}} \;=\; \boldsymbol{V}. \tag{3.1}$$

The actions correspond to the directions of the roads $\boldsymbol{E}$ connecting the intersections. We assume that every intersection is a junction of up to 4 roads

corresponding to the cardinal directions. Therefore, the set of action is:

$$\mathcal{A}_{\text{MDP}} = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}. \tag{3.2}$$

If needed, more actions can be added trivially.

The transition function allows for transitions from $\mathbf{v}$ to $\mathbf{v}'$ with $\mathbf{v}$, $\mathbf{v}' \in \mathcal{S}_{\text{MDP}}$ only if a road exists between them in the direction defined by $\mathbf{a} \in \mathcal{A}_{\text{MDP}}$:

$$\mathcal{T}_{\text{MDP}}(\mathbf{v}', \mathbf{a}, \mathbf{v}) = \begin{cases} 1 & \text{, if } \exists\, \mathbf{e} \in \boldsymbol{E} \mid \mathbf{e} \text{ connects } \mathbf{v} \text{ to } \mathbf{v}' \\ & \text{along direction } \mathbf{a}, \\ 0 & \text{, otherwise.} \end{cases} \tag{3.3}$$

The rewards correspond to the negative length of the roads:

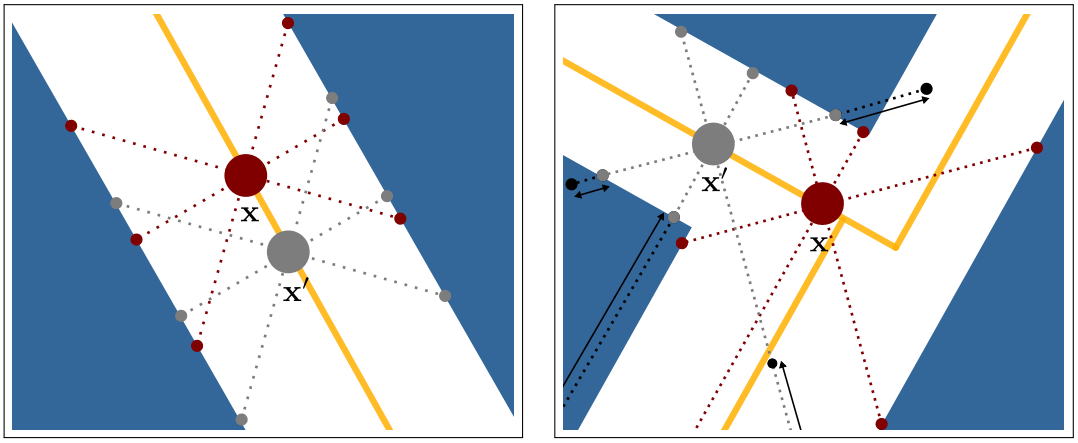
$$\mathcal{R}_{\text{MDP}}(\mathbf{v}, \mathbf{a}, \mathbf{v}') = \begin{cases} r_{\text{goal}} - \ell(\mathbf{v}, \mathbf{a}, \mathbf{v}') & \text{, if } \exists\, \mathbf{e} \in \boldsymbol{E} \mid \mathbf{e} \text{ connects } \mathbf{v} \text{ to } \mathbf{v}' \\ & \text{along direction } \mathbf{a} \text{ and } \mathbf{v}' \text{ is the goal,} \\ -\ell(\mathbf{v}, \mathbf{a}, \mathbf{v}') & \text{, if } \exists\, \mathbf{e} \in \boldsymbol{E} \mid \mathbf{e} \text{ connects } \mathbf{v} \text{ to } \mathbf{v}' \\ & \text{along direction } \mathbf{a}, \\ r_{\text{noroad}} & \text{, otherwise,} \end{cases} \tag{3.4}$$

where $\ell(\mathbf{v}, \mathbf{a}, \mathbf{v}')$ indicates the length of the road from $\mathbf{v}$ to $\mathbf{v}'$ corresponding to action $\mathbf{a}$, $r_{\text{goal}} \geq 0$ is a constant positive reward assigned to the transitions to the goal, whereas $r_{\text{noroad}} \ll 0$ is a negative constant penalty factor assigned if no transition from $\mathbf{v}$ to $\mathbf{v}'$ along the direction defined by $\mathbf{a}$ is possible.

Solving this MDP generates a navigation policy that leads the robot to the goal along the shortest path or, equivalently assuming unary constant velocity, minimum travel time. However, MDPs assume full observability of the state and, thus, that the position of the robot is always known. This is often not the case in robot navigation as most localization systems, including the one adopted in this chapter, provide a probability distribution over the map to describe the position of the robot and not a unique location. Therefore, following an MDP policy in situations with large position uncertainty, the robot may assume to be at the wrong intersection and take a sub-optimal action that results in a longer route to reach the goal.

3.2 Uncertainty-aware planning on road networks

The main contribution of this chapter is a novel approach to planning uncertainty-aware routes on road networks. We propose to improve decision-making at intersections by integrating into the planning process the uncertainty about the



(a) Uninformative location. (b) Informative location.

Figure 3.3: Informativeness of the observations to localize the robot. The buildings' footprints are in blue and the roads are the orange lines. The robot observation at $\mathbf{x}$ is in red, the measurement translated along the road to $\mathbf{x}'$ is in gray, whereas the error between the translated measurement and the map is represented by the black arrows.

position of the robot provided by the localization system. We use an augmented-MDP [146] to efficiently approximate a POMDP by formulating it as an MDP over a discretized belief space. We achieve this by modeling the uncertainty about the position of the robot as a part of the state of the augmented-MDP. In this way, we explicitly consider the state uncertainty while maintaining computational tractability. We define the transition function of an uncertainty-augmented MDP by employing a localization prior to estimate how the belief about the robot's position propagates along the road network.

Our planning approach is able to (ii) take explicitly into account the position uncertainty of the robot to select different actions according to the degree of uncertainty; (i) reduce the number of mistakes that the robot makes during navigation with large position uncertainty; (iii) lead the robot to navigate along routes that, in complex situations, are on average shorter than following a shortest path policy operating under uncertainty but ignoring it.

3.2.1 Localizability

In order to explicitly consider the position uncertainty *in the planning process*, we need to estimate how the belief about the robot's position propagates upon driving along a road and reaching a certain location before moving there. OpenStreetMap provides information about the environment such as the buildings' footprints and the traversable areas. We can exploit this information to estimate in advance how laser scans fired from the robot's LiDAR at a given location will affect localization. A measurement is informative for localizing the robot

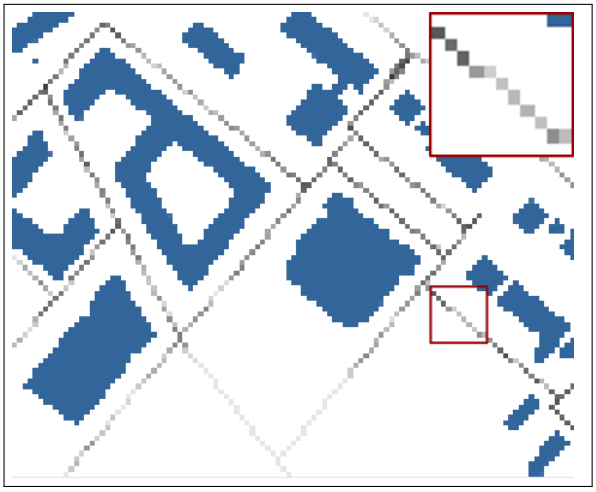


Figure 3.4: Localizability map $\mathcal{Z}$ of the environment illustrated in Figure 3.2. The darker the pixels along the roads, the smaller is the localizability covariance.

if the scans hit the buildings and the observed structure is distinctive, see for example Figure 3.3(b). Whereas, at locations that are too far from buildings or where buildings form a 'corridor' along the road, measurements are not distinctive enough to provide accurate localization, see for example Figure 3.3(a).

Using the information extracted from OpenStreetMap and the sensor model of the LiDAR, we compute a localization prior, called localizability, using the method proposed by Vysotska and Stachniss [169]. This approach estimates how well the robot is expected to localize in a given location based on the capability of the LiDAR measurements to match the footprint of the buildings. To this end, we simulate at each traversable location $\mathbf{x} \in \mathcal{X}$ a virtual laser scan by ray-casting the map. We translate and rotate the virtual measurement along the roads in the local neighborhood and estimate the error between the scan and the map. Considering the decay in the observation likelihood, we compute a covariance matrix $\mathbf{\Sigma}_{\mathcal{Z},\mathbf{x}}$ that estimates how well a scan fired at a position $\mathbf{x}$ matches the map under position uncertainty. At informative locations where the surrounding environment has a distinctive structure, the resulting covariance is small. Whereas, the covariance is large at locations where the surrounding environment is not informative or ambiguous. For example, in Figure 3.3(a), the measurement at position $\mathbf{x}$ (red dotted lines) is not informative because, applying a small translation along the road to the measurement to $\mathbf{x}'$ (grey dotted lines), the translated measurement explains the surrounding as well as the original one. In Figure 3.3(b), the measurement at position $\mathbf{x}$ (red dotted lines) is informative because translating the measurement to $\mathbf{x}'$, the likelihood of the measurement decreases substantially.

We compute such localization prior for each traversable cell in $\mathcal{X}$ offline before starting to navigate and we refer to the resulting representation as to the local-

izability map $\mathcal{Z}$. An example of localizability map is illustrated in Figure 3.4, where the darker the intensity of a cell, the smaller is the maximum eigenvalue of the covariance matrix and, thus, the better the robot is expected to localize.

3.2.2 Uncertainty-augmented MDP

We plan uncertainty-aware routes on road networks by designing an augmented-MDP in which we augment the conventional MDP states with the robot's position uncertainty. Due to the augmented state representation, the transition function and the reward function become more complex than in conventional MDPs. In their final formulation, augmented-MDPs have an analogous representation as MDPs, except for a larger number of states. Therefore, augmented-MDPs can be solved by using the same efficient algorithms as for MDPs. In the following, we define the set of states, the set of actions, the transition function, and the reward function that characterize our uncertainty-augmented MDP for planning routes on road networks.

3.2.2.1 Uncertainty-augmented states

We define the state space of our augmented-MDP by augmenting the states of the MDP formulation in Section 3.1.4, $\mathcal{S}_{\text{MDP}}$, with a representation of the uncertainty about the robot's position. Different statistics can be used to represent uncertainty such as the entropy or the covariance matrix.

In general, the more compact a representation, the more efficient is the planner. Although our localization system can potentially generate any kind of belief, we assume that *during planning*, we can approximate the belief about the position of the robot by a Gaussian distribution with isotropic covariance. This is clearly an approximation, but common planners consider an even stronger assumption by approximating the belief with its expected value and, thus, ignoring the uncertainty at all. A similar representation of the uncertainty has been used by Bopardikar *et al.* [20] as a bound for a state estimator under intermittent sensing. We fully represent a Gaussian distribution with isotropic covariance describing the belief about the position of the robot by its expected position and the variance. This representation augments the MDP state space by considering only one additional dimension and keeps the state space compact, thus avoiding an explosion in complexity for planning. In contrast to that, for example, considering a 2D Gaussian with a general covariance matrix would require three additional dimensions.

Therefore, we define the set of augmented states as:

$$\begin{aligned} \mathcal{S}_{\text{AMDP}} \;=\; & \{(\mathbf{v}_0, \sigma_0^2), (\mathbf{v}_0, \sigma_1^2), (\mathbf{v}_0, \sigma_2^2), \dots, (\mathbf{v}_i, \sigma_j^2), (\mathbf{v}_i, \sigma_{j+1}^2), \dots\}, \\ & \text{with } \mathbf{v}_i \in \boldsymbol{V} \text{ and } \sigma_j^2 \in \boldsymbol{W}, \end{aligned} \tag{3.5}$$

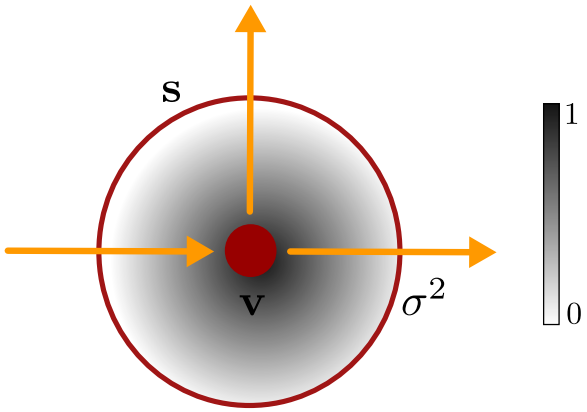


Figure 3.5: Uncertainty-augmented state $\mathbf{s} = (\mathbf{v}, \sigma^2) \in \mathcal{S}_{\text{AMDP}}$, where $\mathbf{v} \in \boldsymbol{V}$ is a road intersection and $\sigma^2 \in \boldsymbol{W}$ defines the associated isotropic covariance. The variance is illustrated by the 2σ circle in red. The probability distribution is represented by the black shaded area such that the darker the more likely.

where $\boldsymbol{V}$ is the set of road intersections and $\boldsymbol{W}$ is a set of variances that discretizes the possible degrees of uncertainty. We can define $\boldsymbol{W}$ by choosing different ranges and discretization degrees depending on the environment and on the desired precision. Each augmented state $\mathbf{s}$ corresponds to the normal distribution $\mathcal{N}(\mathbf{v}, \boldsymbol{\Sigma})$, with $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{smallmatrix}\right]$. An example of uncertainty-augmented state is illustrated in Figure 3.5. We refer to an augmented-state $\mathbf{s} \in \mathcal{S}_{\text{AMDP}}$ defined over the discrete space $\mathcal{X}$ as the probability mass function $p(\mathbf{x} \mid \mathbf{s})$ with $\mathbf{x} \in \mathcal{X}$ or, equivalently, $p(\mathbf{x} \mid \mathcal{N}(\mathbf{v}, \boldsymbol{\Sigma}))$.

3.2.2.2 Actions

Augmenting the state representation with the robot's position uncertainty does not change the actions that the robot can execute. Therefore, the set of actions of our uncertainty-augmented MDP corresponds to the possible directions at a road intersection, identically to the MDP case:

$$\mathcal{A}_{\text{AMDP}} = \mathcal{A}_{\text{MDP}} = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}. \tag{3.6}$$

3.2.2.3 Transitions under position uncertainty

As our augmented-MDP states represent probability distributions, the transition function is more complex to define compared to standard MDPs. The augmented-MDP transition function $\mathcal{T}_{\text{AMDP}}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ takes as input an augmented state $\mathbf{s}$ and an action $\mathbf{a} \in \mathcal{A}_{\text{AMDP}}$ and maps it to a probability distribution of possible end augmented states $\mathbf{s}'$, with $\mathbf{s}, \mathbf{s}' \in \mathcal{S}_{\text{AMDP}},$. Therefore, it defines a probability distribution over probability distributions. We compute $\mathcal{T}_{\text{AMDP}}$ in three steps:

1. We compute the posterior from an intersection $p(\mathbf{x} \mid \mathbf{v}, \mathbf{a})$, with $\mathbf{x} \in \mathcal{X}$, $\mathbf{v} \in \boldsymbol{V}$, that is posterior probability about the robot's position given that it executes $\mathbf{a}$ from $\mathbf{v}$ and navigates up to the next intersection without considering any uncertainty in the input position.

2. We compute the posterior from an augmented state $p(\mathbf{x} \mid \mathbf{s}, \mathbf{a})$, in which the belief about the input position of the robot is represented by the augmented state $\mathbf{s}$, by integrating the posteriors from all possible input intersections.

3. We map the posterior from a state into our augmented-MDP state representation to define the augmented state transition $\mathcal{T}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$.

Posterior from an intersection We compute first the posterior probability about the position of the robot $p(\mathbf{x} \,|\, \mathbf{s}, \mathbf{a})$ over $\mathcal{X}$ upon taking action $\mathbf{a}$ at intersection $\mathbf{v}$ without considering any uncertainty in the input position. To this end, we *simulate* the robot taking $\mathbf{a}$ at $\mathbf{v}$ and moving along the corresponding road according to:

$$\mathbf{s}_t \;=\; g(\mathbf{s}_{t-1}, \mathbf{u}_t) + \epsilon_t, \quad \text{with } \epsilon_t \sim \mathcal{N}(0, \mathbf{R}_t), \tag{3.7}$$

where g is a linearizable function that defines the dynamics of the robot, $\mathbf{u}_t$ is the one-step control corresponding to the action $\mathbf{a}$, and $\mathbf{R}_t$ is the motion noise. We assume that the belief about the position of the robot while navigating along the road can be approximated by a Gaussian distribution. In this way, we can estimate the new belief after a simulated movement by using the prediction step of the Extended Kalman Filter (EKF):

$$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t) \;\approx\; \mathcal{N}(\hat{\boldsymbol{\mu}}_t, \hat{\boldsymbol{\Sigma}}_t), \tag{3.8}$$

$$\text{with} \qquad \hat{\boldsymbol{\mu}}_t \;=\; g(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t) \tag{3.9}$$

$$\hat{\boldsymbol{\Sigma}}_t \;=\; \mathbf{G}_t \, \boldsymbol{\Sigma}_{t-1} \, \mathbf{G}_t^{\top} + \mathbf{R}_t \tag{3.10}$$

where $\mathbf{G}_t$ is the Jacobian of g. For further details about the EKF, refer to Section 2.1.1.

As we are simulating robot navigation along the roads, we do not have actual measurements to correct the predictions as in the typical EKF procedure. However, the localizability information offers an estimate of how informative a measurement in a certain location would be to localize the robot. Therefore, we estimate how the position uncertainty propagates along the road by combining the covariance from the EKF prediction $\hat{\boldsymbol{\Sigma}}_t$ with the localizability covariance $\boldsymbol{\Sigma}_{\hat{\boldsymbol{\mu}}_t, \mathcal{Z}}$ computed at the expected position $\hat{\boldsymbol{\mu}}_t \in \mathcal{X}$:

$$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t, \mathcal{Z}) \;=\; \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t), \tag{3.11}$$

$$\text{with} \qquad \boldsymbol{\mu}_t \;=\; \hat{\boldsymbol{\mu}}_t, \tag{3.12}$$

$$\boldsymbol{\Sigma}_t \;=\; (\hat{\boldsymbol{\Sigma}}_t^{-1} + \boldsymbol{\Sigma}_{\hat{\boldsymbol{\mu}}_t, \mathcal{Z}}^{-1})^{-1}, \tag{3.13}$$

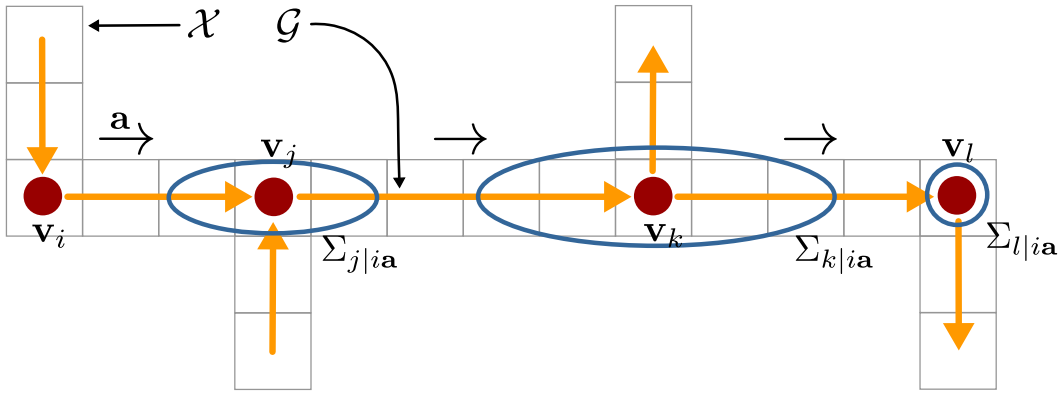


Figure 3.6: Posterior from intersection $\mathbf{v}_i$ to taking action $\mathbf{a} = \rightarrow \in \mathcal{A}_{\text{AMDP}}$ to the intersections $\mathbf{v}_j$, $\mathbf{v}_k$, $\mathbf{v}_l$ represented in the grid map $\mathcal{X}$ underlying the road graph $\mathcal{G}$. The red dots denote the intersections $\boldsymbol{V}$ and the blue ellipses the estimated uncertainty $\boldsymbol{\Sigma}_{\cdot|ia}$ upon reaching the corresponding intersection.

where the localizability imposes a bound to the uncertainty of the EKF prediction step that will otherwise grow infinite independently from the environment. We recursively apply this procedure to compute the posterior probability about the position of the robot while navigating along a road. For example, in Figure 3.6, we estimate the posterior belief about the robot's position of taking action $\mathbf{a} = \rightarrow$ from intersection $\mathbf{v}_i$ to $\mathbf{v}_j$ as the Gaussian distribution $\mathcal{N}(\mathbf{v}_j, \boldsymbol{\Sigma}_{j|i\mathbf{a}})$ computed by recursively applying Equation (3.11) along the cells of $\mathcal{X}$ belonging to the corresponding road.

We explicitly model the possibility that the robot misses an intersection and ends up in a successive one while navigating with large position uncertainty. For example, in Figure 3.6, while navigating from $\mathbf{v}_i$ towards $\mathbf{v}_j$, the robot could miss to detect $\mathbf{v}_j$ and continue navigating along the road ending up in $\mathbf{v}_k$ or in $\mathbf{v}_l$. We compute the probability that the robot detects $\mathbf{v}_j$ such that the smaller is the uncertainty $\boldsymbol{\Sigma}_{j|i\mathbf{a}}$ upon reaching $\mathbf{v}_j$, the higher is the probability to detect it:

$$p_{\text{detect}}(\mathbf{v}_j \mid \mathbf{v}_i, \mathbf{a}) \;=\; p\left(\mathbf{x} = \mathbf{v}_j \mid \mathcal{N}(\mathbf{v}_j, \boldsymbol{\Sigma}_{j|i\mathbf{a}})\right), \tag{3.14}$$

where $p(\mathbf{x} = \mathbf{v}_j \mid \cdot)$ refers to a probability mass function defined over $\mathcal{X}$.

We compute the posterior probability about the robot position $p(\mathbf{x} \mid \mathbf{v}, \mathbf{a})$ of taking action $\mathbf{a}$ at intersection $\mathbf{v}$ and navigating along the corresponding road up to the next detected intersection by considering the probability to end up in each of the reachable intersections:

$$p(\mathbf{x} \mid \mathbf{v}, \mathbf{a}) = \sum_{j=1}^{|J|} \mathcal{N}(\mathbf{v}_j, \boldsymbol{\Sigma}_{j|i\mathbf{a}})\, p_{\text{detect}}(\mathbf{v}_j \mid \mathbf{v}, \mathbf{a}) \prod_{k=1}^{j-1} (1 - p_{\text{detect}}(\mathbf{v}_k \mid \mathbf{v}, \mathbf{a})), \tag{3.15}$$

where J is the ordered set of subsequent intersections that the robot may reach by missing a previous intersection along the road. The probability that the robot

ends up in each of the J intersections decays according to the probability that a previous one has been detected. If no road exists for executing $\mathbf{a}$ at intersection $\mathbf{v}$, we set the posterior to be equal to the input intersection $\mathbf{v}$. Therefore, computing the posterior from an intersection with Equation (3.15) results in a mixture of Gaussian distributions.

Posterior from an augmented state We introduce uncertainty in the input position by considering that the initial belief about the position of the robot is represented by an augmented state $\mathbf{s} \in \mathcal{S}_{\text{AMDP}}$. As the input position is described by a probability distribution, the posterior of taking an action should represent all of the possible transitions that might occur. Therefore, we compute the posterior probability about the robot's position of executing $\mathbf{a}$ from $\mathbf{s}$ by integrating all of the possible posteriors from intersections according to the belief represented by $\mathbf{s}$:

$$p(\mathbf{x} \mid \mathbf{s}, \mathbf{a}) \;=\; \eta \sum_{i=1}^{|\boldsymbol{V}|} p(\mathbf{x} = \mathbf{v}_i \mid \mathbf{s})\, p(\mathbf{x} \mid \mathbf{v}_i, \mathbf{a}), \tag{3.16}$$

where η is a normalization term.

Augmented state transitions Given the posterior from an augmented state, we need to find a correspondence to the augmented states to define the augmented state transitions, i.e., the probability to make a transition from one augmented state to another. We compute this correspondence by using the Bhattacharyya distance [16]. This distance is used in the context of image processing for example to perform feature rejection [30] or to estimate the classification error for feature extraction [26]. Given two distributions p and q defined over the same domain, for example $\mathcal{X}$, the Bhattacharyya distance between the two distributions is defined as:

$$d^{\mathcal{X}}_{\text{Bhatt}}(p,\, q) \;=\; -\ln\left(\sum_{\mathbf{x}\in\mathcal{X}} \sqrt{p(\mathbf{x})\, q(\mathbf{x})}\right). \tag{3.17}$$

The Bhattacharyya distance is not a true metric as it does not satisfy the triangle inequality. However, we prefer it over other approaches such as Kullback-Leibler divergence [87] as it is a symmetric distance.

We define the augmented state transition $\mathcal{T}_{\text{AMDP}}(\mathbf{s},\, \mathbf{a},\, \mathbf{s}')$, with $\mathbf{s},\, \mathbf{s}' \in \mathcal{S}_{\text{AMDP}}$, according to the Bhattacharyya distance over $\mathcal{X}$ between the posterior from $\mathbf{s}$, $p(\mathbf{x} \mid \mathbf{s}, \mathbf{a})$, and the probability distribution represented by $\mathbf{s}'$:

$$h(\mathbf{s},\, \mathbf{a},\, \mathbf{s}') \;=\; d^{\mathcal{X}}_{\text{Bhatt}}\left(p(\mathbf{x} \mid \mathbf{s}, \mathbf{a}),\, p(\mathbf{x} \mid \mathbf{s}')\right), \tag{3.18}$$

and by transforming the distance into the probability space using the softmax function:

$$\mathcal{T}_{\text{AMDP}}(\mathbf{s},\, \mathbf{a},\, \mathbf{s}') \;=\; \frac{e^{-h(\mathbf{s},\, \mathbf{a},\, \mathbf{s}')}}{\sum_{\mathbf{s}_i \in \mathcal{S}_{\text{AMDP}}} e^{-h(\mathbf{s},\, \mathbf{a},\, \mathbf{s}_i)}}. \tag{3.19}$$

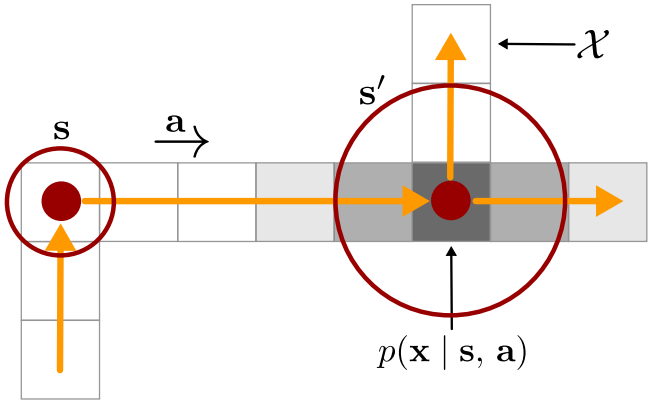


Figure 3.7: We compute the augmented state transition from $\mathbf{s}$ to $\mathbf{s}'$, $\mathcal{T}_{\text{AMDP}}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$, with $\mathbf{s}, \mathbf{s}' \in \mathcal{S}_{\text{AMDP}}$, according to the Bhattacharyya distance over $\mathcal{X}$ between the posterior from $\mathbf{s}$, $p(\mathbf{x} \mid \mathbf{s}, \mathbf{a})$ with $\mathbf{x} \in \mathcal{X}$ and the belief represented by $\mathbf{s}'$.

An illustrative example to compute the augmented state transition from an augmented state posterior is depicted in Figure 3.7.

3.2.2.4 Rewards under position uncertainty

We define the augmented-MDP reward function such that the resulting policy leads the robot to the goal by minimizing the expected travel time (or equivalently the travel distance) while taking into account the uncertainty of the transitions to reduce the risk to go for a long detour during navigation with large position uncertainty. To this end, the rewards should reflect the position uncertainty of the beliefs represented by the input and end augmented states. Therefore, we define the reward of taking action $\mathbf{a}$ from an augmented state $\mathbf{s}$ to the next one $\mathbf{s}'$, with $\mathbf{s}, \mathbf{s}' \in \mathcal{S}_{\text{AMDP}}$, as:

$$\mathcal{R}_{\text{AMDP}}(\mathbf{s}', \mathbf{a}, \mathbf{s}) = \sum_{i=1}^{|\boldsymbol{V}|} p(\mathbf{x} = \mathbf{v}_i \mid \mathbf{s}) \sum_{j=1}^{|\boldsymbol{V}|} p(\mathbf{x} = \mathbf{v}_j \mid \mathbf{s}')\, \mathcal{R}_{\text{MDP}}(\mathbf{v}_i, \mathbf{a}, \mathbf{v}_j). \quad (3.20)$$

where $\mathcal{R}_{\text{MDP}}$ are the rewards of the MDP formulation based on road length. In our uncertainty-augmented MDP, we weight these rewards according to distributions defined by $\mathbf{s}$ and $\mathbf{s}'$.

3.2.3 Solving augmented-MDPs

By solving the augmented-MDP defined above, we obtain a policy to reach the goal that tells the robot the optimal action to take at each intersection of the road network. We look for the optimal policy that selects the actions that maximize the expected rewards of the subsequent states in an infinite time horizon.

Although an augmented-MDP requires more states and more complex models, its formulation is analogous to a conventional MDP. Therefore, we can compute the optimal policy π^* by using the same tools as for MDPs. We compute π^* by using the policy iteration algorithm [60]. Policy iteration has a polynomial bound in the number of states and actions to solve MDPs with fixed discounted rewards [100], but often it is much more efficient in practice.

Solving augmented-MDPs has the same complexity as MDPs but requires a larger number of states, $\mid \mathcal{S}_{\text{AMDP}} \mid = \mid \boldsymbol{V} \mid \cdot \mid \boldsymbol{W} \mid$. On the contrary, POMDPs are PSPACE-complete [128]. Therefore, augmented-MDPs approximate POMDPs with a practically and theoretically more efficient algorithm.

3.2.4 Navigation following augmented-MDP policies

During robot navigation, the localization system computes continuously a belief $bel(\mathbf{x})$ over $\mathcal{X}$ about the position of the robot as described in Section 3.1.3. When the robot detects to be at a road intersection, it has to take an action to decide in which direction to navigate. In order to take actions according to the optimal policy π^* computed by solving our augmented-MDP, we need to determine in which uncertainty-augmented state the robot is. We assume the robot to be in the augmented state $\mathbf{s} \in \mathcal{S}_{\text{AMDP}}$ that presents the minimum Bhattacharyya distance from $bel(\mathbf{x})$ over $\mathcal{X}$:

$$\mathbf{s}_{bel} \;=\; \underset{\mathbf{s} \in \mathcal{S}_{\text{AMDP}}}{\operatorname{argmin}} \; d^{\mathcal{X}}_{\text{Bhatt}} \left(bel(\mathbf{x}),\, p(\mathbf{x} \mid \mathbf{s}) \right). \tag{3.21}$$

Thus, the robot takes the action corresponding to following the optimal policy from this state:

$$\mathbf{a}^* \;=\; \pi^*(\mathbf{s}_{bel}), \quad \text{with } \mathbf{a}^* \in \mathcal{A}_{\text{AMDP}}, \tag{3.22}$$

and keeps navigating along the selected road until it detects the next intersection.

3.3 Experimental evaluation

In this section, we evaluate our planning approach that explicitly takes the uncertainty about the position of the robot into account for robot navigation on road networks. Our experiments are designed to show the capabilities of our approach to (i) make effective decisions based on the position uncertainty during navigation, the environment, and the goal location; (ii) lead to routes that are on average shorter than a shortest path policy operating under uncertainty but ignoring it; (iii) outperform policies that always take the safest actions.

3.3.1 Experimental setup and baseline

All experiments presented here are conducted in a simulated environment containing buildings and road information extracted from OpenStreetMap. The robot navigates along the roads and uses the buildings to simulate LiDAR measurements as well as to compute the localizability map as described in Section 3.2.1. The scans and the odometry are affected by random sensor noise. The localization system implements Markov localization as described in Section 3.1.3. The navigation actions at the intersections are non-deterministic, and the probability of missing an intersection is proportional to the uncertainty about the position of the robot. Our approach and the two baseline methods use the same simulator and localization system.

For comparisons, we consider a shortest path policy similar to the one described in Section 3.1.4. It assumes the robot to be located at the most likely position given the belief provided by the localization system and selects the actions to execute, similarly to the shortest path planners described in Section 2.3.1. We compare our approach also against a safest path policy that considers the localizability information to minimize the expected uncertainty and, thus, selects always safe actions. For illustration purposes, consider again the motivation example illustrated in Figure 3.1. As the most likely position of the robot is B, the shortest path policy would take a right turn to reach the goal (dotted orange path), ignoring the risk of taking a long detour (dash-dotted red path). The safest path policy instead, recognizing that A and B are not distinctive enough, leads the robot to C and makes there a safe right turn to reach the goal (dashed green path), independently from whether the robot is precisely localized or the uncertainty about its position is large.

3.3.2 Situation-aware action selection

The first experiment is designed to show that our approach is able to minimize the travel time to reach the goal while reducing the risk to go through long detours by making appropriate decisions according to the situation and the position uncertainty. We consider the environment depicted in Figure 3.8 and assume that the robot goes for a long detour if it navigates towards intersection O, M, or N. According to the localizability information $\mathcal{Z}$, the robot is expected to localize well along some roads such as $\overline{\mathrm{JK}}, \overline{\mathrm{KC}}$, but finds little structure to localize in others like $\overline{\mathrm{AB}}, \overline{\mathrm{BC}}$ and navigating along them cause a growth in the position uncertainty. Given the initial belief that the robot is at A, B, I, or J (green circle), we show that our approach is able to adapt the action selection depending on the situation. To this end, we sample the initial state with uniform probability and consider, in turn, two different goal locations.

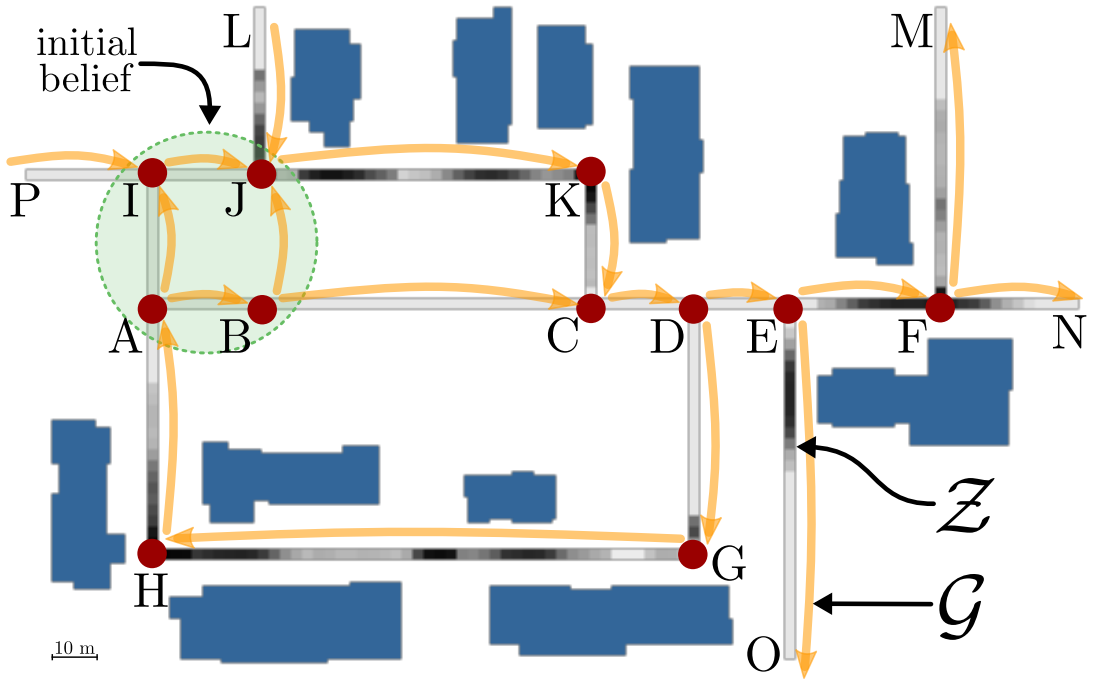
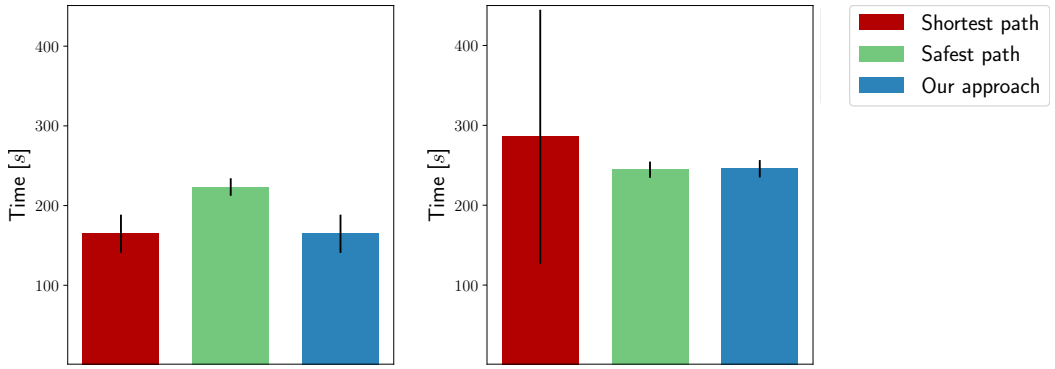


Figure 3.8: Environment considered in the situation-aware action selection experiment discussed in Section 3.3.2. The intersections are the red dots denoted by letters, the roads are the orange arrows, and the buildings are colored in blue. The localizability $\mathcal{Z}$ along roads is represented such that the darker, the better the expected localization. The initial belief about the position of the robot is a uniform distribution over the intersections A, B, I, J highlighted in green.



(a) Goal in F. (b) Goal in G.

Figure 3.9: Mean and standard deviation of the time that the robot takes to travel to the different goals in the environment illustrated in Figure 3.8.

First, we set F as the goal location. The shortest path policy navigates rightwards to reach the goal quickly. Whereas, the safest path policy seeks to go through $\overline{\text{JK}}$ where the robot is expected to localize better. The policy generated by our approach follows a similar strategy as the shortest path. In fact, although the robot cannot localize perfectly along $\overline{\text{AE}}$, it is expected to re-localize along $\overline{\text{EF}}$ and, thus, to reach safely the goal without the risk to taking a longer route even by following a greedy plan. Figure 3.9(a) shows the average travel time of the three policies. Our approach presents the same performance as the shortest path and outperforms the safest path policy.

The situation changes if we set G as the goal. The safest path policy seeks again to go through $\overline{\text{JK}}$ to reduce the uncertainty and take the correct turn at D. Whereas, the shortest path policy leads the robot rightwards to quickly reach D and make there a turn to the goal. However, navigating along $\overline{\text{AD}}$, the uncertainty grows and so the probability that the robot takes the wrong turn or misses intersection D. For this reason, the shortest path policy leads the robot to the goal G in the 60% of the cases and, in the 40% of the cases, to detours at O or N. This behavior results overall into sub-optimal performance, see Figure 3.9(b). As reaching D with large uncertainty may cause the robot to make mistakes and go through long detours, our planner seeks to reduce the uncertainty before making the turn. Thus, in this case, our policy behaves similarly to the safest path policy leading the robot along $\overline{\text{JK}}$ before turning towards the goal at D and outperforms the shortest path policy.

This experiment showcases the ability of our planner to adapt to the situation and the actual uncertainty by picking the best of both the shortest and the safest path worlds.

3.3.3 Uncertainty-aware action selection

The second experiment is designed to illustrate the ability of our approach to deal with different degrees of uncertainty and, also in more complex scenarios, to computes policies that are on average shorter than the ones provided by a planner that ignores the actual uncertainty of the robot during navigation. To this end, we consider a robot that navigates in the environment depicted in Figure 3.10. The robot starts from A, B, or C with different initial position uncertainties and should reach to the goal G.

For shortest path planning algorithms, this is a trivial problem and the shortest path to the goal can be easily found: navigate upwards and make a right turn to the goal at E. If the robot is accurately localized, following this path leads it quickly and safely to the goal. However, as there is little structure in the environment to localize along $\overline{\text{AE}}$, the position uncertainty upon reaching E grows. Reaching E with large uncertainty increases the probability of mismatching the

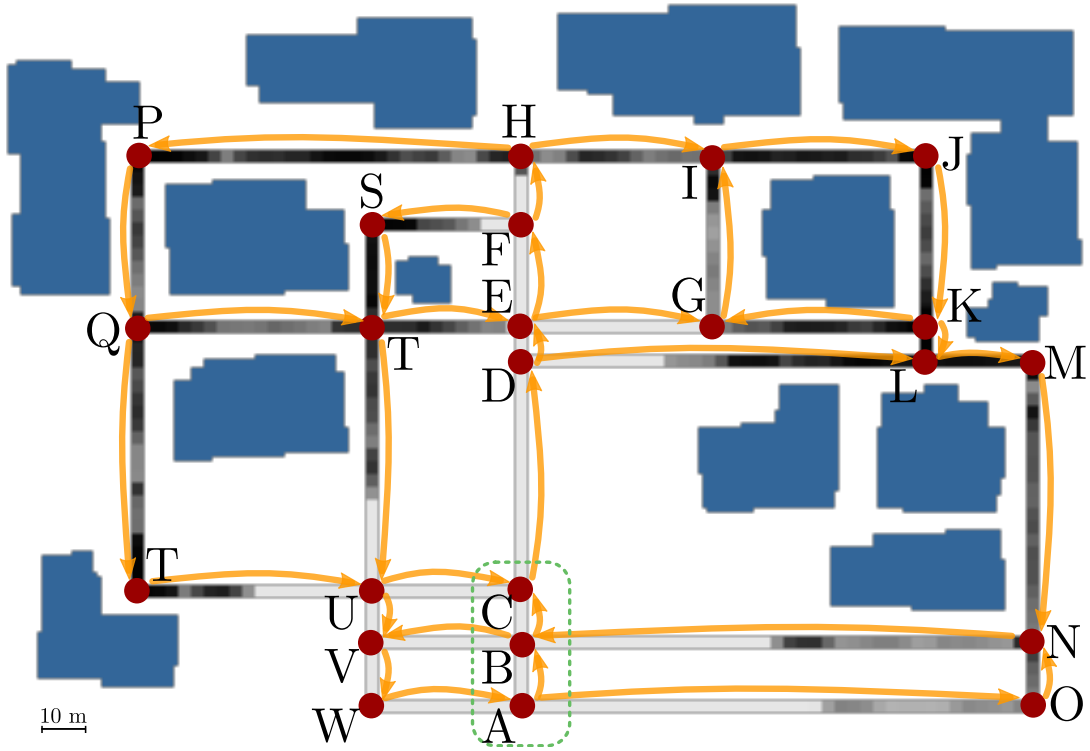
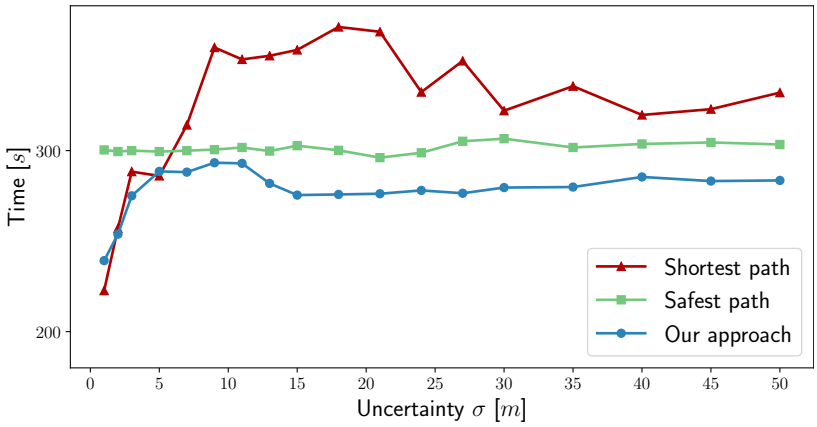


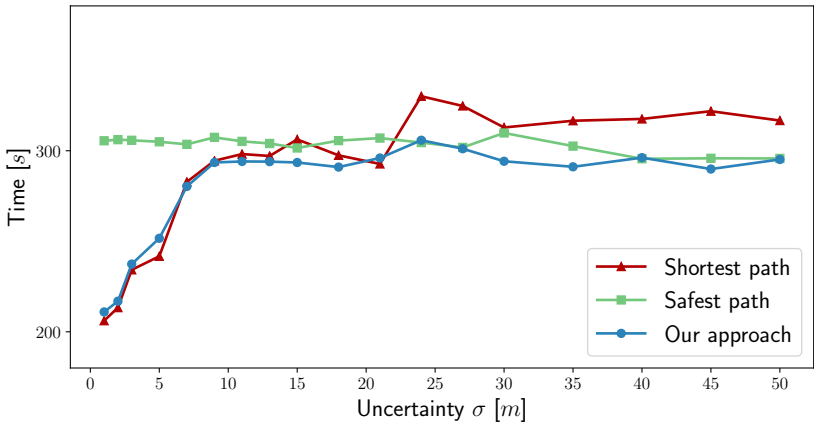
Figure 3.10: Environment considered in the uncertainty-aware action selection experiment discussed in Section 3.3.3. Same notation as in Figure 3.8.

intersections D and E. Thus, the situations in which the expected robot's position is at E while it is actually at D are more likely. In this case, the shortest path policy makes the robot turning right leading it to a long detour through L. Large uncertainty may also cause that the robot misses intersection E or F leading also to detours. For example, if the robot believes to be at F but its actual position is at H, following the shortest path policy, it will make a left turn to S, so that it can reach the goal G by moving through T and E. However, turning left takes the robot to a long detour through P instead. Therefore, as soon as the uncertainty about the robot's position grows, following a greedy strategy increases the probability to take wrong turns and detouring.

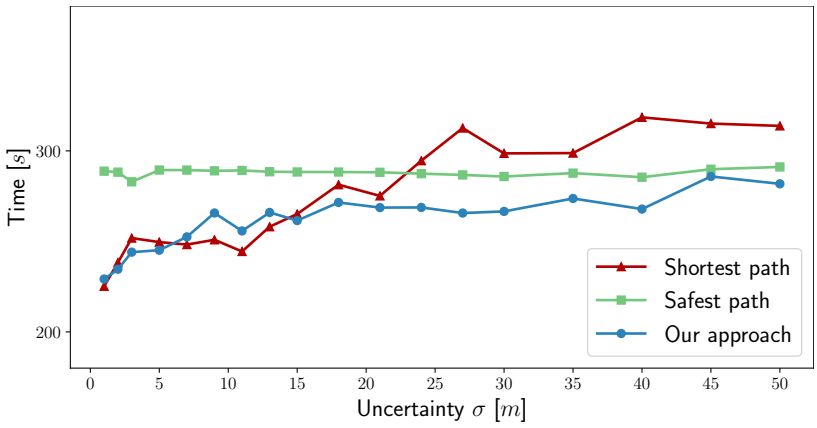
The safest path policy seeks to make safe turns at intersections in which the robot is expected to localize well, for example, at the end of the roads or where the localizability is good. Therefore, to reach the goal, it leads the robot upwards to H, and makes a safe right turn towards I reducing the risk to mismatch intersections. From I, it moves the robot rightwards to J, turns to K and, finally, to the goal G. Making a turn to the goal at K is considered as a safe decision as the robot is expected to localize very well there. However, the safest path policy always makes safe decisions considering how well the robot is expected to localize but ignores the actual position uncertainty during navigation. Therefore, it often follows an overly conservative and longer strategy to reach the goal also



(a) Start in A.



(b) Start in B.



(c) Start in C.

Figure 3.11: Average travel time to reach the goal G starting from A, B, and C with different levels of uncertainty σ in the environment illustrated in Figure 3.10.

in situations in which the uncertainty is small.

Our approach, in contrast, makes decisions by explicitly considering the belief about the position of the robot provided by the localization system during navigation. It selects actions depending on the degree of uncertainty and, thus, leads the robot to the goal by trading off safety and travel time.

The performance of the three policies is illustrated in Figure 3.11. We considered 18 different levels of uncertainty with σ ranging from 1 to 50 meters and performed 200 runs for each initial location and degree of uncertainty. The safest path policy presents on average similar travel time when varying the initial position uncertainty. The shortest path policy shows short travel time when the uncertainty is small but, when the uncertainty grows, it takes on average longer than the safest path to reach the goal. Our approach follows a strategy similar to the *shortest* path policy when the uncertainty is small and thus mistakes are unlikely. However, in tricky situations, when the uncertainty becomes large, our approach makes decisions similar to the *safest* path policy, thereby taking safer actions to avoid making mistakes and ending up in long detours.

This experiment shows that our approach is able to take the appropriate navigation action according to the degree of uncertainty of the robot during navigation, overall outperforming the shortest and safest path policies.

3.4 Towards real-world environments

Although our approach for planning routes considering position uncertainty is more efficient than solving a POMDP, the complexity to compute a policy at city-scale for a real-world environment, where the number of roads and intersections is large, is still high. To the best of our knowledge, currently there exist no approaches which take the position uncertainty explicitly into account that can plan paths at city-scale. Our approach can be seen as a step in this direction. We compute navigation policies offline and, once one is available, it can be executed online by the robot in constant time, without requiring to re-plan. The localizability is also expensive to compute but it has to be computed only once for each environment. Furthermore, we believe that planning under uncertainty is more relevant in practice at a local scale where mistakes at intersections can lead to detours that have a significant impact on the route followed by the robot. Therefore, planning at a city-scale can be more effective by combining, in a hierarchical manner, our approach with a higher level planner that plans paths in a coarse representation, and uses our policy in at a finer level to make appropriate, uncertainty-aware, local decisions where they have the highest impact.

To sum up, even though our evaluation only considers simulated environments, the approach discussed in this chapter takes a step towards planning

under position uncertainty for robot navigation.

3.5 Possible extension to navigation with GNSS

Throughout this chapter, we considered a mobile robot that localizes itself using a 360° 2D LiDAR sensor. However, our planning approach can also be applied similarly to robots that rely on different sensors for localization. Consider for example a robot the localizes itself by using a global navigation satellite system (GNSS). The GNSS signal might be temporarily unavailable because the robot is in a tunnel or in a so-called urban canyon, i.e., roads with tall buildings on both sides creating a canyon-like environment. In these cases, the position uncertainty increases and following the shortest path policy may lead to taking wrong turns and going through long detours. Prior information about the localization accuracy for a GNSS in an urban environment can be computed from the buildings' elevation map. Zimmerman *et al.* [178] compute obstruction adaptive elevation masks for identifying the influence of signal diffraction, NLOS reception and signal distortions, and, thus, improving the positioning of high-precision GNSS. We can use this prior similarly to the localizability information to estimate the propagation of the belief about the robot's position in the road network. In this way, we can compute an augmented-MDP policy that takes into account the localization uncertainty for GNSS-based navigation by using the procedure described in Section 3.2.2.

3.6 Related work

Although the interest in planning under uncertainty is increasing more and more, most robots still rely on some variant of A* to navigate. Several urban navigation robots such as Obelix [89] or the Autonomous City Explorer [95] plan paths by using A* on metric or topo-metric maps. Zahn and Noon [176] provide an extensive evaluation of the traditional shortest path algorithms for pathfinding on real road networks.

The approach presented in this chapter relies on a topo-metric representation of the environment. Topo-metric maps are an effective representation for planning and localization in urban environments characterized by road networks. Kuipers and Byun [85] introduce a topological map representation where nodes correspond to places and edges to actions for moving between places that contain local information for navigation. Konolige *et al.* [75] present an approach to building hybrid maps that combine occupancy grid maps and topological graphs

in a hierarchical manner. Thrun [162] uses artificial neural networks to build grid maps and generates topological maps on top by splitting the metric map into coherent regions. Publicly available map services, such as OpenStreetMap and Google Maps, offer free topo-metric maps of most cities. Such map representations have been used in several robotic systems for path planning, localization, and mapping [44, 56, 169]. We built our environment representation by extracting information from OpenStreetMap.

The Markov decision process allows for optimally solving planning problems in which the state is fully observable but the actions are noisy, see Section 2.4.1. There are many well-studied algorithms to solve MDPs such as value iteration [8] and policy iteration [60]. If the state is not observable, the problem turns into a POMDP, see Section 2.4.2. However, the computational complexity of POMDPs is often too high to provide useful results for real-world problems [128]. There exist many approaches to approximate POMDPs. The POMCP [151] allows for online POMDP planning by using an algorithm based on the Monte Carlo tree search which samples the successive states and runs simulations to estimate the potential reward. The DESPOT [152] is based on a similar idea but focuses the search on a set of randomly sampled scenarios. Luo *et al.* [101] use DESPOT for planing controls for autonomously driving a vehicle among pedestrians by taking into account the uncertainty in pedestrian intentions. Although these approaches do not require solving the full POMDP, they demand significant resources and are effective for planning with short time horizons. In contrast to that, our approach computes global navigation policies that allow the robot to make decisions with an infinite time horizon for reaching the goal.

Roy [145] introduced the augmented Markov decision process to approximate the state space of a POMDP. Augmented-MDPs define MDPs over belief states similarly to Belief MDPs [65]. The belief state representation allows for casting POMDPs into MDPs with fully observable states that can be solved by using the tools of the MDP world. Augmented-MDPs compress the belief space into a compact representation which allows for solving the problem efficiently. Augmented-MDPs have been used in different contexts: Roy *et al.* [146] generate robust trajectories for indoor navigation that minimize the likelihood that the robot becomes lost; Hornung *et al.* [59] plan velocity commands for a mobile robot that minimize the motion blur of its camera; whereas, Kawano [68] controls under-actuated blimps under the effects of wind disturbance. In this chapter, we used augmented-MDPs to plan routes on road networks that take into account the uncertainty about the position of the robot to reduce the risk to take a wrong turn and go through a long detour.

Approaches that incorporate the uncertainty into the planning process are usually referred to as planning in belief space. The belief roadmap [138] is a

variant of the probabilistic roadmap algorithm that plans in belief space for linear Gaussian systems by using a factored form of the covariance matrix. Platt *et al.* [136] assume maximum likelihood observations and use linear quadratic regulation to generate policies. The LQG-MP [13] uses a linear-quadratic controller to estimate a priori the robot's belief along the paths assuming Gaussian uncertainty and, accordingly, selects the best candidate path. These approaches compute a path offline assuming a priori maximum likelihood observations, but they do not consider the sensor or process noise while the robot executes it. Van den Berg *et al.* [14] overcome this assumption by taking as input a feasible trajectory and improving it by computing a locally-optimal POMDP control policy that assumes Gaussian beliefs. Bry and Roy [22] propose to improve a nominal trajectory by constructing incrementally a graph of trajectories through the belief space using RRT*. The FIRM [1, 2] generalizes probabilistic roadmaps over the belief space and assigns a unique belief to each node taking all possible future observations into account, thus breaking the "curse of history" of POMDPs. However, this approach does not consider the possibility that the robot reaches a node with different beliefs. In contrast to that, our approach generates offline a policy that deals with different degrees of uncertainty and selects online the optimal action given the current belief of the robot. Bopardikar *et al.* [20] consider a compact representation of the robot's belief based on the expected maximum eigenvalue of the covariance and demonstrate that using this representation there exists a bound on the performance of a state estimator under intermittent sensing. We use a similar approximation to represent the robot's position uncertainty. Such a compact representation allows us to plan efficiently even in large environments and, thus, to take a step towards real-world applications.

Other related approaches for planning in the belief space are in the context of simultaneous localization and mapping (SLAM). Indelman *et al.* [61] perform exploration in an unknown environment while maintaining localization uncertainty within given bounds. To this end, they model the joint probability distribution over the robot state and the environment to predict the outcome of possible decisions. Valencia *et al.* [167] use SLAM pose graphs as belief roadmaps for planning paths with the lowest accumulated robot pose uncertainty. Fermin-Leon *et al.* [43] plan on pose graphs paths that minimize the expected uncertainty at the goal by considering the probability of re-localization at every step of the trajectory.

In this chapter, we estimated how the robot's belief propagates along the road network by combining the EKF-style prediction step and a localization prior. We compute the localizability along the roads by using the method proposed by Vysotska and Stachniss [169]. This approach was developed to support active localization using OpenStreetMap data. Roy *et al.* [146] proposed one of the first approaches for modeling the information content of each point in the map to nav-

igate a robot by reducing localization uncertainty. Bengtsson and Baerveldt [9] calculate the covariance of the position estimate by building a local geometric map from a scan and matching simulated scans from the surrounding positions to score scan positions for mapping and localization. Schirmer *et al.* [148] use this approach to define a heuristic for planning paths in the belief space that maximizes the safety of a robotic lawn-mower.

3.7 Conclusion

In this chapter, we presented a novel approach for efficient path planning under position uncertainty on road networks. We formulate this problem as an augmented MDP that incorporates the robot's position uncertainty into the state space. This formulation allows for solving the planning problem by using the MDP tools without requiring to solve a full POMDP. We define the transition function of the augmented-MDP by estimating how the robot's belief propagates along the road network by using a localization prior to bound the growth in the predicted uncertainty. During navigation, we match the belief about the position of the robot provided by the localization system with our augmented state representation and select accordingly the optimal action to execute by taking into account the degree of uncertainty.

The key advantage of our approach is that it makes navigation decisions adaptively and adjusts to the current task, configuration, and uncertainty. Taking into account this information, it can trade off safety and travel time to reach the goal. The experiments illustrate that, if the robot's position uncertainty is small, our approach performs similarly to the shortest path policy and outperforms the safest path policy that always selects the action that reduces the uncertainty. Whereas, when the uncertainty is large and the risk of making sub-optimal decisions grows, our approach outperforms the shortest path policy by taking safe decisions that reduce the risk of taking wrong turns and going for long detours. Therefore, our approach is able to perform robust navigation under uncertainty on road networks by picking the best for the shortest and the safest path worlds.

Chapter 4

User-preferred navigation exploiting experiences

In real-world environments, robots perform different tasks and encounter a large variety of scenes during navigation. Therefore, it is fundamental for autonomous robots to be able to navigate by adapting their behaviors to suit the situation. In the previous chapter, we discussed how robots can take into account uncertainty to decide which action to execute in a given situation and, thus, perform robust navigation. Other essential aspects that should be taken into account to navigate in a certain scenario are the criteria imposed by the user, the task, the environment, and the other agents operating in it. For example, in many environments such as factory floors, users require robots to follow specific behaviors during traversal. Furthermore, in environments populated by other moving entities such as humans or other vehicles, robots are required to navigate in a compliant manner by following safe and foreseeable behaviors.

In this chapter, we investigate the problem of robot navigation by following the user's preferences on robot behaviors according to the task, the environment, and the local scene that the robot encounters while navigating. Many approaches attempt to define formal rules and preferences for navigation, for example, Kirby *et al.* [70]. However, formalizing preferences can be difficult, and hard-coded rules are often complex to maintain and update. Furthermore, hard-coded rules may interfere and increase the complexity of switching between preferences or between different users. In the same way, specifying mathematical cost functions to define behaviors as, for example, Haigh *et al.* [55] can be complex and usually requires experts to define such costs. In our work, we aim at implicitly extracting information about the user's preferences from the previous experiences of the robot and at incorporating this knowledge for planning new trajectories.

We introduce a robot navigation system that captures the user's preferences by allowing the user to demonstrate desired behaviors to the robot or to provide

Figure 4.1: A KUKA KMR iiwa mobile manipulator deployed on a factory floor where it shares the workspace with human operators as in the use case envisioned by the RobDREAM project. Image courtesy of KUKA Roboter AG.

feedback about its experiences. This is generally more convenient than hard-coding preferences as even non-expert users can easily teach behaviors to the robot from demonstration, for example, by joysticking it along desired routes in the environment. We store the user's favorite behaviors into a database of experiences and reuse them to guide the planning process for a new but similar navigation task. Several motion planning systems have been developed to reuse previously experienced paths, for example, Berenson *et al.* [12]. Most of these approaches focus on speeding up the planning process rather than implementing specific robot behaviors. Teach-and-repeat approaches, such as Sprunk *et al.* [154], allow robots to precisely repeat trajectories. However, it is typically hard to generalize taught trajectories to different situations and to reproduce them if the environment changes.

We consider a planning approach that is able to compute paths that reproduce and generalize previous successful experiences of the robot, thus meeting the preferences of the user. Our approach generates trajectories by incorporating the user-preferred paths in a sampling-based planning system. In this way, it maintains the flexibility of a general planner and allows for reproducing experiences even when changes occur in the environment. We employ this approach both for planning global paths from start to goal locations of the environment and for planning local deviations to avoid obstacles. For dynamic obstacle avoidance, we combine our planning approach with a probabilistic model for estimating their trajectories. This model makes predictions by taking the uncertainty of the obstacle motion into account. Following our approach, robots can accomplish safe and predictable navigation behaviors that meet the user's preferences by reproducing his preferred experiences in similar situations.

4.1 Navigation on factory floors

Nowadays, the use of robotic systems in manufacturing industries is widespread. Traditional industrial robots are manipulators employed to perform specific tasks at fixed locations. Recently, the demand for flexible robotic solutions that are able to perform different tasks at different locations has grown. These tasks include mobile manipulation as well as navigation to transport materials and tools from one location to another. To operate on factory floors, mobile robots are usually requested to fulfill some requirements such as navigating in restricted areas of the environment or following definite patterns during navigation. Furthermore, when robots operate in environments shared with human workers or forklifts, the reproducibility and the predictability of the navigation behaviors become essential to avoid collisions without limiting the flexibility of the robot operation.

This demand has led to several EU-funded projects on the matter. The work presented in this chapter has been developed in the context of the H2020 project RobDREAM[1]. This project focuses on automatically adapting and improving over time the capabilities of a mobile manipulator working on a factory floor by processing its previous experiences without requiring an expert to tune and optimize its parameters.

4.1.1 RobDREAM use case

Throughout this chapter, we consider a use case that stems from the RobDREAM project to explain and motivate our work: a non-expert operator requests a mobile robot to perform multiple navigation tasks on a factory floor while expecting the robot to behave according to his preferences. We assume that both the robot and the operator know the map of the environment. The operator asks the robot to perform a navigation task by specifying the start and goal poses. In this scenario, the robot may encounter other moving entities such as human workers, forklifts, other robots, etc. during navigation.

The navigation system presented in this chapter enables the robot to accomplish the required navigation tasks by adapting its navigation behaviors according to the user's preferences. To achieve this, we give the possibility to the operator to express preferences on the behaviors experienced by the robot by rating them as good or bad, for example by using a simple GUI in which the user can also replay the robot's behaviors. The operator can also demonstrate good behaviors to the robot by joysticking the platform along the desired trajectories. We store the good experiences into a database that grows over time. Our system exploits such a database of experiences to capture and reproduce the preferences of the

[1] http://robdream.eu/

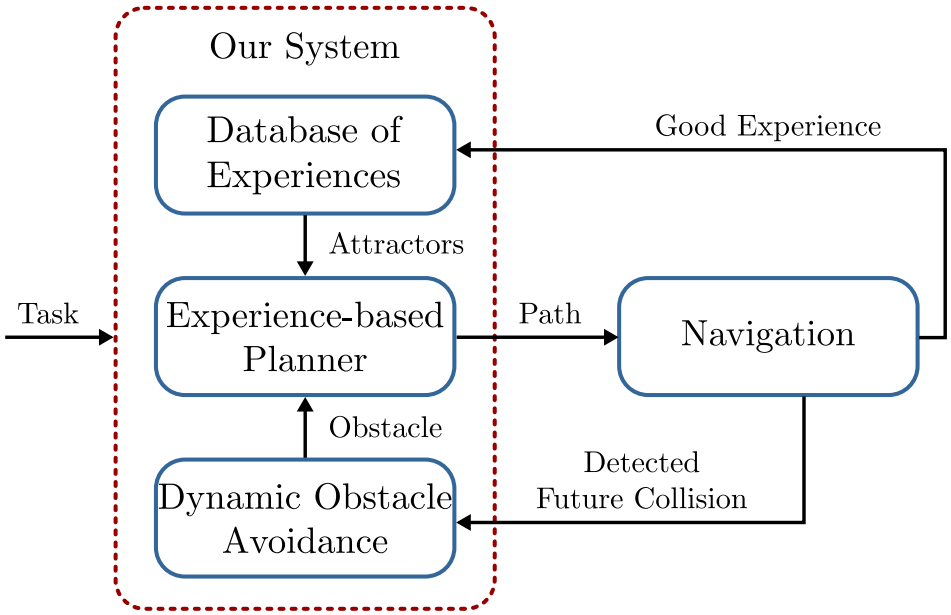


Figure 4.2: Overview of our experience-based robot navigation system. It consists of three main components: a database containing the good paths experienced by the robot, a planner that exploits these experiences for computing new paths, and a collision avoidance module to avoid dynamic obstacles during navigation.

user for planning paths to perform new tasks.

4.2 Experience-based navigation following user preferences

The main contribution of this chapter is a novel robot navigation system that allows for easily instructing robots to follow preferred navigation behaviors. It extracts the user's preferences from the previous experiences of the robot and incorporates them into a path planning algorithm. Our approach allows the robot to: (i) perform navigation behaviors that meet the preferences of the user by reusing the previous experiences of the robot; (ii) reproduce and generalize behaviors over different environments and situations; (iii) perform predictable behaviors providing similar solutions for similar situations; and (iv) navigate in environments populated by other moving objects while still accomplishing foreseeable behaviors.

4.2.1 System overview

Our robot navigation system consists of three main components and an overview is illustrated in Figure 4.2. The remainder of this section briefly introduces these components, which we will describe in detail in the subsequent sections.

Database of experiences We consider a database to store the paths experienced by the robot that have been positively evaluated by the user. Once a new task is assigned to the robot, it retrieves from the database the experience matching the current situation and exploits it for planning a new path.

Experience-based planning Our planning algorithm computes paths by guiding the search to prefer trajectories that match the robot's experiences and thus, leads the robot to reproduce previous successful behaviors. Whereas, when no suitable experience is available, it is able to explore the configuration space.

Dynamic obstacle avoidance If the robot detects a dynamic obstacle during navigation, we use a probabilistic approach to predict its trajectory and the experience-based planner to generate a local deviation that avoids the obstacle while performing a foreseeable behavior.

Our system realizes robot navigation both at the global and the local levels. The global level computes a path from the start to the goal pose in the static map of the environment. At the global level, our system can exploit the previous experiences of the robot in an environment to perform new similar tasks in the same environment, but it does not generalize the exploitation of experiences across different environments. The local level handles collision avoidance with unforeseen obstacles by planning local deviations from the global path. It relies only on the local situation and is largely independent of the map. Therefore, our system can generalize local experiences across different scenes and environments.

4.2.2 Database of experiences

We store the paths experienced by the robot that the user evaluates positively into a database that we call the database of experiences. When the user requires the robot to perform a new task, we search the database of experiences and retrieve a path experienced in a similar situation. We use this path to generate a new trajectory that reproduces the previous experience and, thus, meets the user's preferences. To this end, we introduce a path representation that allows for compactly storing the experiences of the robot while capturing the geometric structure of the paths for reproducing them. We also define a situation descriptor that determines a similarity relationship among navigation tasks and, thus, allows for selecting the previous experiences matching the current task. We define a different path representation and situation descriptor at global and local levels to capture better their functions and properties.

4.2.2.1 Attractor-based path representation

We use a path representation that effectively stores and retrieves previously experienced paths. We represent a path $\mathcal{P}$ as the set of poses:

$$\mathcal{P} \approx \boldsymbol{A} = \{\mathbf{x}_{\text{start}}, \mathbf{a}_1, \ldots, \mathbf{a}_n, \mathbf{x}_{\text{goal}}\}, \tag{4.1}$$

where $\mathbf{x}_{\text{start}}$ and $\mathbf{x}_{\text{goal}}$ are the start and goal poses and $\mathbf{a}_1, \ldots, \mathbf{a}_n$ are the poses along the path that allow for capturing its geometric structure. We call these poses attractors, recalling the concept introduced by Jiang and Kallmann [64]. In Section 4.2.3, we will describe how our experience-based planning algorithm takes advantage of this representation for generating new paths.

Our planning algorithm computes and represents paths as dense sequences of poses in the configuration space. From this representation, we compute the attractors of a path by iteratively considering a window of path points. We initialize the window with the first two path points. If a line fits through the points in the window, we insert the successive point in the window. Otherwise, we identify the last inserted point as a candidate attractor and check whether a straight motion from the previous attractor is valid and collision-free. If this is the case, we include the candidate attractor to the list of attractors and reinitialize the window. Otherwise, we select the previous point in the path as a candidate attractor and perform a new check. We iterate this procedure until all path points are processed. The result is a compact ordered list of poses describing the structure of the path.

Global and local levels of our system rely on different assumptions, so it is necessary to define a different way to store and to reuse the attractors. We maintain the experiences in two distinct databases to which we refer to as $\mathcal{D}^G$ for the global paths and as $\mathcal{D}^L$ for the local ones.

Global experiences do not generalize across different environments. Therefore, we represent the attractors of global paths as:

$$\mathbf{a}_i^G = (x_i, y_i, \theta_i), \tag{4.2}$$

where (x_i, y_i, θ_i) are the coordinates of the attractor in the map frame. For the same reason, we maintain a distinct database of global paths for each environment.

Instead, we aim at generalizing local experiences across different scenes. Therefore, the path representation at the local level should not depend on the environment and be translation and rotation invariant. To this end, we apply a coordinate transformation to the attractors that makes them independent from the map frame and that can be inverted in new situations. We introduce a local coordinate system based on the local situation and the corresponding obstacle,

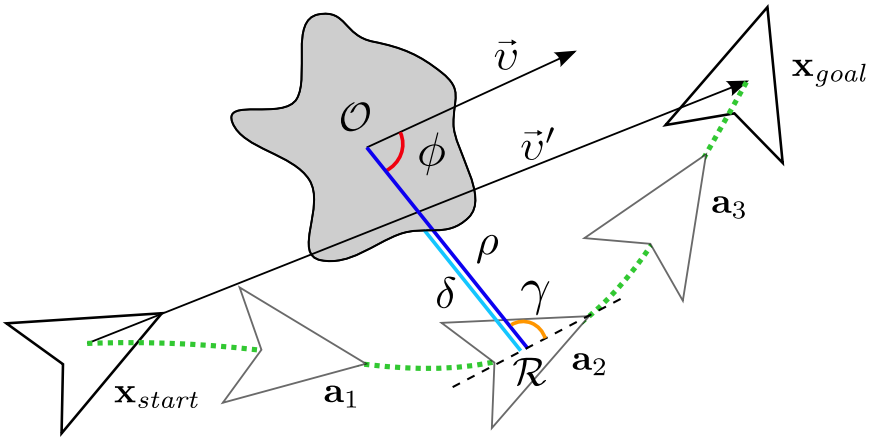


Figure 4.3: Our attractor representation of a local path $\mathcal{P} \approx \{\mathbf{x}_{\text{start}}, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{x}_{\text{goal}}\}$. The green dotted curve is the original path executed to avoid the obstacle $\mathcal{O}$. We represent a local attractor $\mathbf{a}_i$ using the coordinates (δ, ϕ, γ).

as illustrated in Figure 4.3. We represent the attractors of the local paths in the polar coordinate system that has the pole in the center of the obstacle $\mathcal{O}$ and the orientation of the vector $\vec{v}'$, which connects the start to the goal pose. In this frame, we can identify unambiguously a local attractor with as:

$$\tilde{\mathbf{a}}_i^L = (\rho_i, \phi_i, \gamma_i), \tag{4.3}$$

where ρ is the distance of the attractor to $\mathcal{O}$, ϕ and γ are the angles that the line passing by $\mathcal{O}$ and the center of the robot $\mathcal{R}$ form respectively with $\vec{v}$ and the robot axis. This representation relies only on local information and, thus, allows for transferring experiences across different environments. To exploit an experienced local path for a new task, we transform the local attractors in the map frame of the current environment and use them to guide the new plan.

To further generalize experiences over obstacles of different shape and size, we replace ρ with δ, which is the distance of the attractor from the obstacle surface along ρ. Representing local attractors as

$$\mathbf{a}_i^L = (\delta_i, \phi_i, \gamma_i) \tag{4.4}$$

allows for reusing an experienced local behavior even in the cases in which the obstacle encountered by the robot has different dimension, while keeping a safe distance from it.

4.2.2.2 Situation descriptors

To reproduce previous successful behaviors, it is fundamental to identify which experience fits a new situation. We compare navigation tasks using situation

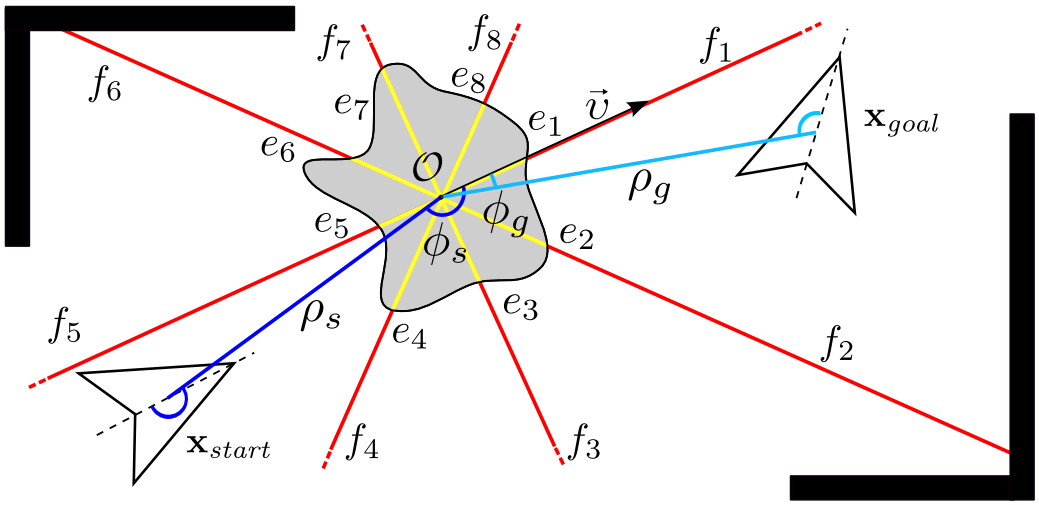


Figure 4.4: Local situation descriptor for avoiding the obstacle $\mathcal{O}$ defined by the local start (dark blue) and goal (light blue) in the local attractor representation, the extent of the obstacle (yellow) and of free space around it (red).

descriptors. A situation descriptor is a feature vector that describes the task and the scene in which the robot performs it. For each experienced path, we compute the corresponding situation descriptor and store it into the database of experiences together with the attractors. We measure the similarity between two tasks by computing the sum of the Euclidean distances of the attributes of their situation descriptors.

As stated by Jetchev *et al.* [62], the ability to generalize experiences to new tasks depends on the description of the situation. Therefore, as for the path representation, we specify two distinct situation descriptors to describe global and local tasks that fit the objectives and the assumptions considered at each level. We define the situation descriptor for global experiences as the vector composed by the start and goal poses of the task expressed in the map frame. Given the task to navigate from $\mathbf{x}_{\text{start}} = (x_s, y_s, \theta_s)$ to $\mathbf{x}_{\text{goal}} = (x_g, y_g, \theta_g)$, the corresponding situation descriptor is:

$$\mathbf{d}^G = [x_s, y_s, \theta_s, x_g, y_g, \theta_g]. \tag{4.5}$$

This definition of descriptor fits the requirements at global level: it allows for multiple queries in one environment and provides no information across different environments. Jiang and Kallmann [64] show that such descriptors are effective in practice for matching situations in static environments.

Combining this definition of situation descriptor with the representation of paths as attractors, we can consider at the global level every sub-path as a distinct example. We store each attractor individually in the database of experiences with a reference to the path it belongs to. Given a new task, we search the database for the pair of robot poses $\{\mathbf{x}_i, \mathbf{x}_j\}$ such that $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same path $\mathcal{P}$

and the sum of the distances to the start and goal configurations of the new task is the smallest. In this way, a new plan can be guided by any of the sub-paths of $\mathcal{P}$.

At the local level, we want to generalize experiences across different environments and obstacles. To this end, we define a situation descriptor that relies only on local information, as illustrated in Figure 4.4. First, similarly, as for the global level, we consider the start and goal pose of the local task expressed in the local coordinates introduced in Section 4.2.2.1 as attributes of the descriptor:

$$\mathbf{d}^L_{\text{task}} = [\rho_s, \phi_s, \gamma_s, \rho_g, \phi_g, \gamma_g], \tag{4.6}$$

where we consider the coordinates defined in Equation (4.3) as ρ is straightforward to compute and δ does not provide any further information to describe the task. This descriptor represents the task with respect to the obstacle, but provides no information about the obstacle itself and the space around it. The geometry of the obstacle and the space around it are fundamental to plan a local deviation. Therefore, we consider two additional components. The first one describes the shape and dimension of the obstacle by computing the extent of the obstacle from its center $\mathcal{O}$ in the 8 directions defined by the $\vec{v}$ axis, each rotated by 45 degrees, as illustrated in Figure 4.4:

$$\mathbf{d}^L_{\text{obstacle}} = [e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8]. \tag{4.7}$$

This component favors the robot to accomplish similar behaviors at obstacles with similar shapes and dimensions. Such obstacles are likely to correspond to similar objects and the user may want the robot to accomplish a specific behavior to avoid a certain class of objects. For example, to avoid moving or potentially moving obstacles the user may prefer a behavior that does not cross their front or their way. The second additional component describes the extent of free space from the obstacle surface to the next obstacle along the same directions considered in $\mathbf{d}^L_{\text{obstacle}}$:

$$\mathbf{d}^L_{\text{freespace}} = [f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8], \tag{4.8}$$

see Figure 4.4 for an illustration. The geometry around an obstacle is a good indicator for deciding on which side of the obstacle the robot should pass to avoid it. Therefore, considering this component will favor paths in the same homotopy for scenes presenting similar geometry around the obstacle.

The resulting local descriptor is:

$$\mathbf{d}^L = [\mathbf{d}^L_{\text{task}}, \mathbf{d}^L_{\text{obstacle}}, \mathbf{d}^L_{\text{freespace}}], \tag{4.9}$$

and it allows for comparing local tasks across different situations, obstacles and environments. We combine different measures of similarity to keep the local descriptor as general as possible. Nevertheless, each component can be parametrized

Algorithm 4 Experience-based navigation to follow user's preferences

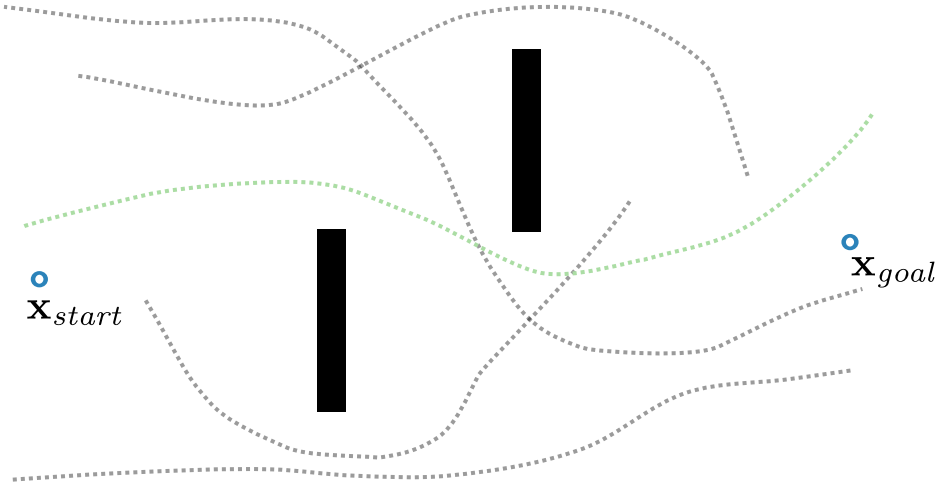
1: **procedure** ExperienceBasedNavigation ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{D}^{G,\mathcal{X}}$, $\mathcal{D}^{L}$)
2: $\quad\mathcal{P}^{G} \leftarrow$ PlanGlobalPath ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{D}^{G,\mathcal{X}}$), $\mathcal{X}$
3: $\quad\mathcal{P}_{\text{actual}} \leftarrow \mathcal{P}^{G}$
4: $\quad$**while** $\mathbf{x}_i \in \mathcal{P}_{\text{actual}} \neq \mathbf{x}_{\text{goal}}$ **do**
5: $\quad\quad\mathcal{O}, \mathcal{P}_{\text{invalid}} = \{\mathbf{x}_j, ..., \mathbf{x}_k\} \leftarrow$ CheckPath ($\mathbf{x}_i$, $\mathbf{x}_{\text{goal}}$, $\mathcal{P}_{\text{actual}}$)
6: $\quad\quad$**if** $\mathcal{P}_{\text{invalid}} \neq \emptyset$ **then**
7: $\quad\quad\quad\mathcal{P}^{L} \leftarrow$ PlanLocalPath ($\mathbf{x}_{j-1}$, $\mathbf{x}_{k+1}$, $\mathcal{O}$, $\mathcal{D}^{L}$, $\mathcal{X}$)
8: $\quad\quad\quad\mathcal{P}_{\text{actual}} \leftarrow$ UpdatePath ($\mathcal{P}_{\text{actual}}$, $\mathcal{P}^{L}$)
9: $\quad\quad$NavigateTo($\mathbf{x}_i$)
10: $\quad\quad i \leftarrow i + 1$
11: $\quad\mathcal{D}^{G,\mathcal{X}}$, $\mathcal{D}^{L} \leftarrow$ CollectExperiences ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{P}_{\text{actual}}$, $\mathcal{P}^{G}$, $\mathcal{D}^{G,\mathcal{X}}$, $\mathcal{D}^{L}$)

to obtain a descriptor biased towards a specific aspect of a local task, for example, to favor similar behaviors to avoid similar obstacles.

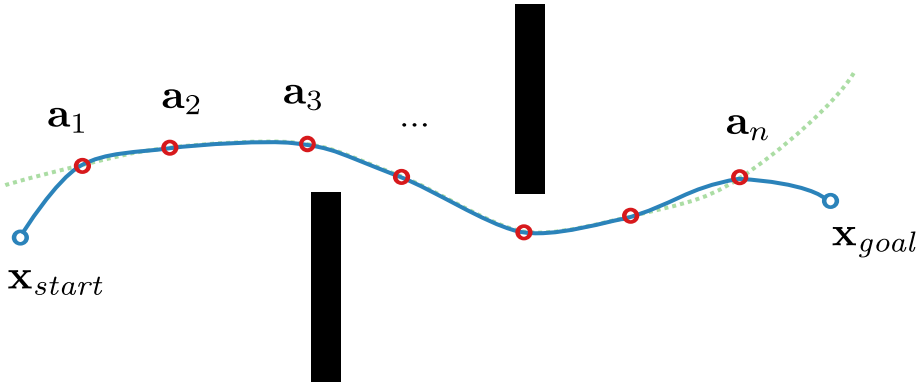
4.2.3 Experience-based path planning

We introduce a path planning algorithm that exploits the robot's previous experiences to generate paths for new tasks. Our algorithm takes advantage of the notions of situation descriptors to identify similar tasks and of attractors to represent the corresponding experienced paths. As an example to illustrate our algorithm, consider the use case introduced in Section 4.1.1 and assume that the operator requires the robot to perform the task illustrated in Figure 4.5: navigate from $\mathbf{x}_{\text{start}}$ to $\mathbf{x}_{\text{goal}}$ in a known environment where the robot has already successfully performed several tasks (dotted lines).

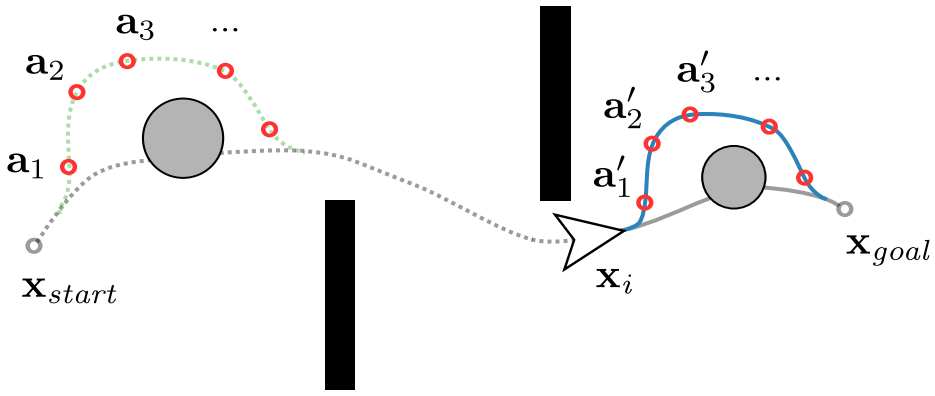
The main procedure of our system is illustrated in Algorithm 4. It takes as input the map of the environment $\mathcal{X}$, the database of global paths experienced in $\mathcal{X}$, $\mathcal{D}^{G,\mathcal{X}}$, the database of local paths $\mathcal{D}^{L}$, the start $\mathbf{x}_{\text{start}}$, and the goal $\mathbf{x}_{\text{goal}}$ pose. First, we compute a path from start to goal in $\mathcal{X}$ (line 2). Once a path is available, the robot starts navigating along it. At each step of the path, we check the remaining poses for invalid configurations (line 5). If an unforeseen obstacle blocks the path (line 6), we plan a local deviation that enables the robot to avoid the obstacle (line 7). Therefore, we update the path with the local deviation (line 8) and the robot can safely navigate towards the next path pose (line 9). We repeat this procedure until the robot reaches the goal. Once the robot completes the task, we ask the user to provide feedback about the robot's behaviors and store the behaviors evaluated positively into the database of experiences (line 11). In the following, we describe in detail how the local and global planning work and how we collect the user feedback.



(a) New task and previously experienced paths in the environment (dotted lines). The green path is the one that best fits the current task.



(b) Global planning guided by the attractors of a similar experience.



(c) Local planning guided by the attractors of a similar experience.

Figure 4.5: Our planning approach exploits similar experienced paths at global and local level to perform a new navigation task.

Algorithm 5 Experience-based global path planning

1: **function** PlanGlobalPath ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{D}^{G,\mathcal{X}}$, $\mathcal{X}$)
2: $\quad\mathbf{d}^G \leftarrow$ ComputeGlobalDescriptor ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$)
3: $\quad\mathbf{d}^G_{\text{exp}} \leftarrow$ GetSimilarExperience ($\mathbf{d}^G$, $\mathcal{D}^{G,\mathcal{X}}$)
4: $\quad$**if** IsEmpty ($\mathbf{d}^G_{\text{exp}}$) **then**
5: $\quad\quad\mathcal{P}^G \leftarrow$ Bi-RRT ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, *uniform_sampling*, $\mathcal{X}$)
6: $\quad$**else**
7: $\quad\quad\boldsymbol{A}^G \leftarrow$ RetrieveAttractors ($\mathbf{d}^G_{\text{exp}}$, $\mathcal{D}^{G,\mathcal{X}}$)
8: $\quad\quad\mathcal{P}^G \leftarrow$ Bi-RRT ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, *guided_sampling*, $\boldsymbol{A}^G$, $\mathcal{X}$)
9: $\quad$**return** $\mathcal{P}^G$

4.2.3.1 Global path planning

The PlanGlobalPath function illustrated in Algorithm 5 implements the navigation at global level of our system. Given the task to navigate from $\mathbf{x}_{\text{start}}$ to $\mathbf{x}_{\text{goal}}$ and the static map of the environment $\mathcal{X}$, the global planner computes a path from $\mathbf{x}_{\text{start}}$ to $\mathbf{x}_{\text{goal}}$ in $\mathcal{X}$. To this end, we first compute the global situation descriptor $\mathbf{d}^G$ corresponding to the current task (line 2), as described in Section 4.2.2.2. We compare the descriptor $\mathbf{d}^G$ with the ones stored in $\mathcal{D}^{G,\mathcal{X}}$ to search for a previous experience performed in a similar situation (line 3). If no similar experience is available, we plan a new path from scratch using standard bi-directional RRT (Bi-RRT) [84] (line 5). Bi-RRT is a sampling-based planning algorithm that searches for a valid path by sampling poses with a uniform probability distribution over the configuration space. See Section 2.3.2 for further details.

Otherwise, if the robot experienced successfully a similar task as in the situation illustrated in Figure 4.5(a) (green dotted line), we retrieve its attractors $\boldsymbol{A}^G$ (line 7) and use them to compute a new path (line 8). To achieve this, we bias the search of Bi-RRT to follow the previous experience by using the attractors $\boldsymbol{A}^G$ to guide the sampling process. We search for a path by iteratively sampling one of the attractors $\mathbf{a}_i \in \boldsymbol{A}^G$ in an ordered fashion. For the tree rooted in the start pose, we select the attractors from the start location to the goal location. For the tree rooted in the goal, in the opposite order. As a search tree reaches an attractor $\mathbf{a}_i$, we expand the tree towards the next one $\mathbf{a}_{i+1}$.

If the environment changed with respect to when the robot performed the selected previous experience, one or more attractors might not be reachable. In this case, we sample a new pose according to a dynamically-updated isotropic Gaussian distribution centered in the unreachable attractor and with covariance growing proportionally to the number of non-valid states sampled around it. We repeat this procedure until we sample a valid state or we reach a maximum

Algorithm 6 Experience-based local path planning

1: **function** PlanLocalPath ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{O}$, $\mathcal{D}^L$, $\mathcal{X}$)
2: $\quad\mathbf{d}^L \leftarrow$ ComputeLocalDescriptor ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{O}$)
3: $\quad\mathbf{d}^L_{\text{exp}} \leftarrow$ GetSimilarExperience ($\mathbf{d}^L$, $\mathcal{D}^L$)
4: $\quad$**if** IsEmpty ($\mathbf{d}^L_{\text{exp}}$) **then**
5: $\quad\quad\mathcal{P}^L \leftarrow$ Bi-RRT ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, *uniform_sampling*, $\mathcal{X}$)
6: $\quad$**else**
7: $\quad\quad\boldsymbol{A}^L \leftarrow$ RetrieveAttractors ($\mathbf{d}^L_{\text{exp}}$, $\mathcal{D}^L$)
8: $\quad\quad\mathbf{a}^L_{\mathcal{X}} \leftarrow$ ToMapFrame ($\boldsymbol{A}^L$, $\mathcal{O}$, $\mathcal{X}$)
9: $\quad\quad\mathcal{P}^L \leftarrow$ Bi-RRT ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, *guided_sampling*, $\mathbf{a}^L_{\mathcal{X}}$, $\mathcal{X}$)
10: $\quad$**return** $\mathcal{P}^L$

number of iterations. In the latter case, the search continues by sampling poses uniformly in the configuration space in the same way as standard Bi-RRT.

If our planner can successfully find a path by following a previous experience, the new path reproduces its structure and, thus, meets the user's preferences, as shown in Figure 4.5(b). Otherwise, if no similar experience is available, our planner is still able to generate a path to the goal from scratch. Another advantage of our approach is that, when a previous experience guides the planning process, the planning time decreases dramatically. We will illustrate this result in Section 4.4.1.

4.2.3.2 Local path planning

The PlanLocalPath function implements the local re-planning procedure of our system and it is described in Algorithm 6. We trigger this function if the robot encounters an unforeseen obstacle while following the global path. It considers the blocking obstacle $\mathcal{O}$ to compute the descriptor of the local situation $\mathbf{d}^L$ (line 2), as described in Section 4.2.2.2. We use the descriptor to query the database of local paths $\mathbf{d}^L$ for a similar experience (line 3). The planning scheme at the local level reproduces the one at the global level. If no similar previous experience is available to be exploited, we search a new local path using standard Bi-RRT (line 5). Otherwise, we retrieve the attractors $\boldsymbol{A}^L$ corresponding to the previous similar experience (line 7) and transform them into the current situation (line 8) to guide the sampling process of the planning algorithm (line 9). If an experience guides successfully the planning process, the resulting path reproduces the previous local behavior in the new situation, thus meeting the user's preferences.

In the example shown in Figure 4.5(c), an obstacle blocks the path and the robot needs to plan a deviation. Along its path, the robot has already successfully avoided an obstacle in a locally similar situation. Therefore, we transform the

Algorithm 7 Collecting examples from user feedback

1: **function** CollectExperiences ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{P}_{\text{actual}}$, $\mathcal{P}^G$, $\mathcal{D}^{G,\mathcal{X}}$, $\mathcal{D}^L$, $\mathcal{X}$)
2: $feedback^G \leftarrow$ GetFeedback ($\mathcal{P}^G$)
3: **if** IsGoodBehavior ($feedback^G$) **then**
4: $\mathbf{d}^G \leftarrow$ ComputeGlobalDescriptor ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$)
5: $\boldsymbol{A}^G \leftarrow$ ComputeAttractors ($\mathcal{P}^G$)
6: StoreExperience ($\mathbf{d}^G$, $\boldsymbol{A}^G$, $\mathcal{D}^{G,\mathcal{X}}$)
7: $deviations \leftarrow$ GetDeviations ($\mathcal{P}_{\text{actual}}$, $\mathcal{P}^G$)
8: **for each** $\{\mathcal{O}, \mathcal{P}^L\}$ **in** $deviations$ **do**
9: $feedback^L \leftarrow$ GetFeedback ($\mathcal{O}$, $\mathcal{P}^L$)
10: **if** IsGoodBehavior ($feedback^L$) **then**
11: $\mathbf{d}^L \leftarrow$ ComputeLocalDescriptor ($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{O}$)
12: $\boldsymbol{A}^L_{\mathcal{X}} \leftarrow$ ComputeAttractors ($\mathcal{P}^L$)
13: $\boldsymbol{A}^L \leftarrow$ ToLocalFrame ($\mathbf{a}^L_{\mathcal{X}}$, $\mathcal{O}$)
14: StoreExperience ($\mathbf{d}^L$, $\boldsymbol{A}^L$, $\mathcal{D}^L$)

attractors of that experience to the new situation and use them to compute the new plan. As a result, the robot avoids the new obstacle by performing a behavior similar to the previously experienced one.

4.2.3.3 Collecting examples from user feedback

The CollectExperiences function (Algorithm 7) enables the user to rate the experienced robot's behaviors and to collect his favorite experiences. To achieve this, we ask first the user to provide feedback for the executed global path (line 3). If the user rates the path positively, we compute the corresponding descriptor (line 4) and attractors (line 5), and store them into the database of global experiences $\mathcal{D}^{G,\mathcal{X}}$ (line 6). Considering the deviations from the global path generated by local re-planning (line 7), the user can also provide feedback for the robot's local behaviors (line 10). If the user evaluates a local behavior as good, we compute the corresponding local descriptor (line 11) and attractors (line 12) as for the global path. At local level, we transform the attractors into the local coordinate frame introduced in Section 4.2.2.1 (line 13) before storing them into the database of local experiences $\mathcal{D}^L$ (line 14). The experiences stored in this process are then made available to guide the planning process for the subsequent tasks.

Our system does not require any initial data to work. Without examples, our approach automatically falls back to Bi-RRT performance. We build the database of experiences by storing good behaviors online, while the robot performs its tasks. The larger the number of feedbacks provided by the user, the better the

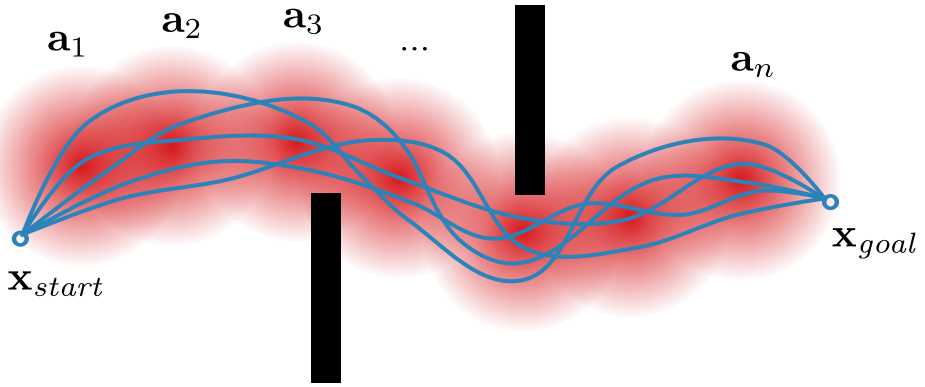


Figure 4.6: Exploration by relaxing the attractors of the experience illustrated in Figure 4.5(b).

system fits his preferences. If the environment changes so that some experiences cannot be exploited anymore, we remove them from the database. In this way, we only keep track of experiences that are still valid and bound the growth of the database. Furthermore, if required, we can easily define distinct databases for different users such that each database contains only the preferences of a specific user.

4.2.3.4 Exploring new behaviors

Our planning approach biases the search for a new path to be as close as possible to a similar previous experience. This allows for reproducing behaviors and generating paths that are predictable and meet the preferences of the user. It comes, however, at the cost of exploring only a limited portion of the configuration space if a similar path exists.

In case the user desires to explore new behaviors, we give the possibility to switch to an exploration mode. Two options are available to explore the space. The first one is the full exploration mode. It does not consider any previous experience but plans new paths using standard Bi-RRT that probabilistically covers the whole configuration space. The second one, which we call attractor relaxation, relaxes the constraints imposed by the attractors by increasing the covariance around one, some, or all attractors during the sampling process in planning. Attractor relaxation does not explore the whole configuration space but allows the user to find an optimal behavior according to his preferences starting from an experienced path. An example in which we relaxed all of the attractors of an experienced path is illustrated in Figure 4.6. The user can evaluate such paths in simulation and, once a path satisfies his preferences, we replace the experience in the database with the new path.

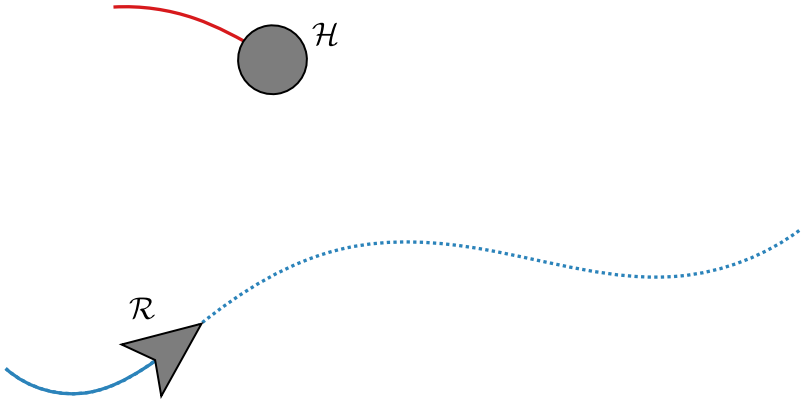


Figure 4.7: Robot $\mathcal{R}$ navigates along the blue dotted path and observes another agent $\mathcal{H}$ moving along the red path.

4.3 Dynamic obstacle avoidance

In the scenario described in Section 4.1.1, the robot operates in environments where it shares the workspace with other moving entities. Therefore, to navigate in such environments, it should be able to avoid collisions with dynamic obstacles. We aim at achieving this by performing safe and predictable behaviors while still reproducing behaviors that meet user's preferences.

Consider for example the situation depicted in Figure 4.7 in which the robot $\mathcal{R}$ initially navigates along the blue dotted path. At the same time, another agent $\mathcal{H}$ is moving nearby. To ensure that $\mathcal{H}$ will not block the robot's path, we first need to check whether a collision between the two agents may occur. We know the robot's path and its motion model so we can easily compute its future poses. However, an entity moving on a factory floor can be a human operator, another robot, a forklift, etc. Thus, we cannot make any specific assumptions about the trajectory and motion model of $\mathcal{H}$.

As the future motion of the other agent presents high uncertainty, we employ a probabilistic approach to predict its trajectory from the observations of the robot. Given the prediction of the agent's trajectory, we introduce a strategy that uses our experience-based planning approach to plan safe local deviations for avoiding dynamic obstacles while meeting the user's preferences. To perform planning efficiently, we do not take time information explicitly into account but replace the notion of time by introducing spatial constraints.

4.3.1 GPs for trajectory modeling

We model the trajectories of the dynamic agents that share the workspace with the robot from the robot's observations by relying on a Gaussian process model [142]. We employ a pair of GPs which takes as input the time $t = 1, 2, 3, \ldots, T$ and

provides as output the trajectory increments along the x and y directions in the environment:

$$\Delta x_t = x_t - x_{t-1}, \tag{4.10}$$

$$\Delta y_t = y_t - y_{t-1}. \tag{4.11}$$

This representation assumes that the movements along x and y are independent. However, as the noise along x and y is correlated, correlation is also induced in the posterior processes. For both GPs, we consider a mean function to be zero. This assumption allows for reducing the number of hyperparameters that characterize our model and it is reasonable assumption as the GPs represent functions of increments. We employ as a covariance function the sum of a Matérn kernel [106] with $\nu = 5/2$ and a noise kernel:

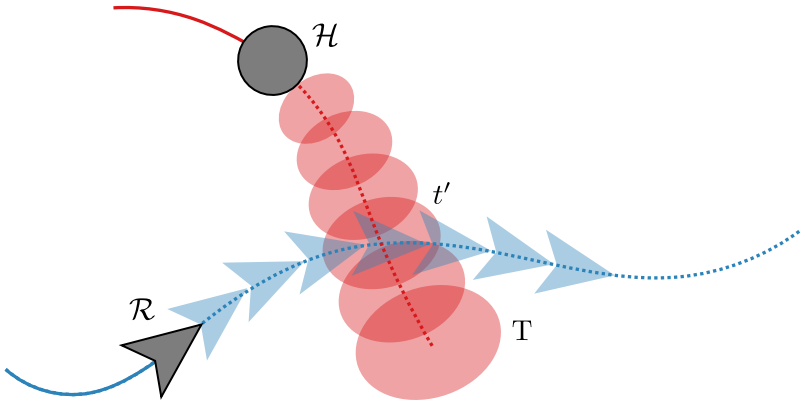
$$\begin{aligned} k(x,\, x') = \quad & \varsigma_f \Big(1 + \frac{\sqrt{5}\,(x - x')}{\ell} + \frac{5\,(x - x')^2}{3\ell^2}\Big) \\ & \exp\Big(-\frac{\sqrt{5}\,(x - x')}{\ell}\Big) + \varsigma_n^2\,\delta(x - x'), \end{aligned} \tag{4.12}$$

where the hyperparameters are the length scale ℓ, the expected variance of the output ς_f and the noise term ς_n. The Matérn kernel includes a large class of kernels and it is often used in real-world applications because of its flexibility. We preferred it over the squared-exponential kernel as the latter assumes high smoothness of the function, which usually does not hold for noisy observations of real trajectories.

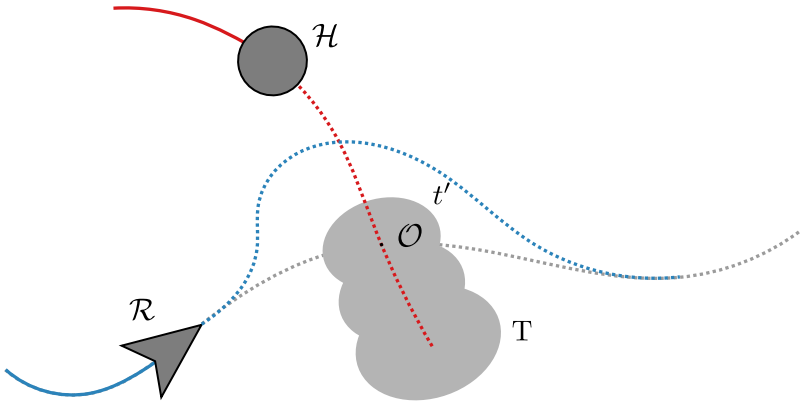
We train the hyperparameters of the GPs by maximizing the log marginal likelihood given the robot's observations during navigation. GPs are in general more flexible estimators than, for example, an EKF for which an explicit motion model needs to be defined, see Section 2.2.2 for further details. Therefore, we can exploit general information about moving entities without the need to train parameters of an explicit parametric motion model. As a result, the trajectory model improves over time as the robot performs more experiences and encounters more obstacles.

4.3.2 Detection of future collisions

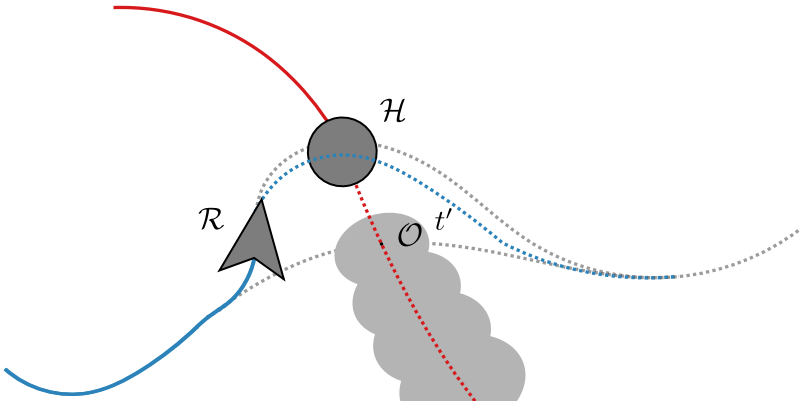
We use this GP model of trajectories to predict the future poses of an agent moving in the surroundings of the robot from its observed positions. In the example in Figure 4.8(a), we predict the future trajectory of the agent $\mathcal{H}$ within a time horizon T. GP models provide the expected positions and the variance of the predictions. The red dotted line represents the predicted mean trajectory. We visualize the predicted 2σ confidence intervals in x, y coordinates as a continuous sequence of ellipsoids, which we call uncertainty areas. In the figure, the



(a) Collision detection at time t' by integrating the robot's footprints over the other agent's predicted uncertainty areas.



(b) Planning a local deviation considering as obstacle the gray region defined by the uncertainty areas for $t' \leq t \leq T$.



(c) Deviation update to smoothly converge to the original path.

Figure 4.8: Planning local deviations to avoid a collision with a dynamic obstacle in the situation illustrated in Figure 4.7.

red ellipsoids represent the uncertainty areas generated by the prediction of $\mathcal{H}$'s trajectory.

To determine whether a collision may occur, we integrate the future footprints of the robot over the uncertainty areas of the trajectory of the other agent corresponding in time. If there is a significant probability that the robot and the agent are in the same area at the same time, we detect a future collision. This approach is equivalent to checking whether the future footprints of the robot overlap with the corresponding uncertainty areas of the agent. For example, in Figure 4.8(a), we detected a future collision at time t'.

4.3.3 Planning for collision avoidance

If a future collision has been detected, the robot needs to plan a local deviation from the global path for avoiding the moving agent and reaching the goal safely. Many path planning algorithms compute local deviations by explicitly taking into account time information, for example, the Timed Elastic Band [69]. However, considering a time dimension in the state space causes the planning process to become more complex and the planning time to increase. The growth in the planning time may result in slow reactions of the robot in the presence of an unforeseen obstacle and, in the worst case, in a collision.

We introduce a strategy for planning local deviations that does not require to explicitly incorporate the time dimension in the planning process. We use the uncertainty areas of the agent's trajectory to replace the notion of time with spatial constraints. We define an artificial obstacle that is centered in the center of the uncertainty area corresponding to the time of the detected future collision t' and that extends over the area delimited by the union of the uncertainty areas for t, $t' \leq t \leq T$, as illustrated in Figure 4.8(b). We exploit such an obstacle for planning a local deviation as described in Section 4.2.3.2. Therefore, if a similar experience is available, we transform its attractors into the current situation and use them to guide the planning process. The resulting local deviation avoids the uncertainty areas corresponding to time $t' \leq t \leq T$, see the blue dotted line in Figure 4.8(b) as an illustration. This may seem like a conservative strategy, however, it ensures safety in the time horizon T. Note that, following our definition, the artificial obstacle expands in the direction in which the other agent is moving. Thus, the re-planning favors local deviations passing on the opposite side.

Every time the robot makes a new observation of the agent's trajectory, we update the prediction and, accordingly, the artificial obstacle. If the obstacle changes, we compute a new local deviation by guiding the planning process with, if possible, the same experience. Using the same experience ensures that the robot does not continuously attempt to execute different conflicting behaviors. As time

goes by, the uncertainty of the agent's trajectory prediction at t' decreases and the corresponding uncertainty area becomes smaller, see for example Figure 4.8(c). Accordingly, also the deviation from the global path decreases. Following this obstacle avoidance strategy, the robot smoothly converges as soon as possible to follow the original global path.

It is straightforward to apply the same strategy to avoid static obstacles. If an obstacle is not moving, the predicted mean is always equal to the current pose and the variance is zero. Therefore, we can exploit indistinctly local experiences to avoid static and dynamic obstacles.

Using our experience-based planning algorithm for computing local deviations that reproduce preferred behaviors for obstacle avoidance has also the advantage of making the robot's behavior foreseeable. Predictability increases safety as the agents sharing the workspace with the robot can easily predict the robot's behavior in certain situations and behave accordingly.

4.4 Experimental evaluation

In this experimental evaluation, we illustrate and discuss the capabilities and performance of our navigation system to: (i) generate navigation behaviors that meet the user's preferences by exploiting the robot's previous experiences; (ii) reproduce and generalize behaviors over different environments and situations; (iii) perform predictable behaviors providing similar solutions for similar situations; (iv) avoid dynamic obstacles while still accomplishing foreseeable behaviors.

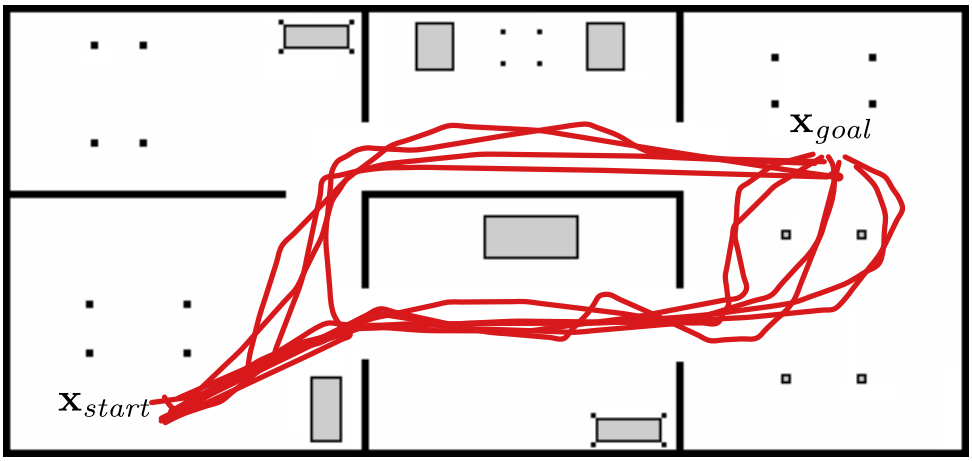
We evaluate our experience-based planning algorithm in Section 4.4.1. Section 4.4.2 focuses on our trajectory prediction model. We test our navigation system in simulation in Section 4.4.3 and we present some tasks performed running our system on a real KUKA Youbot mobile robot in Section 4.4.4.

4.4.1 Experience-based planning

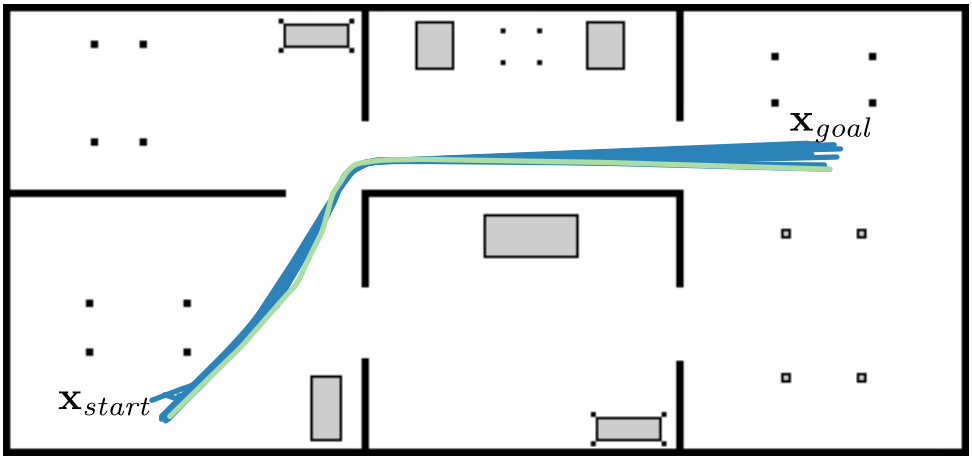
In the first set of experiments, we aim at evaluating the capabilities of our experience-based planning approach. Throughout this section, we consider the scenario of the use case introduced in Section 4.1.1 and use the implementation of Bi-RRT available in Open Motion Planning Library [159] as a baseline.

4.4.1.1 Following user's preferences

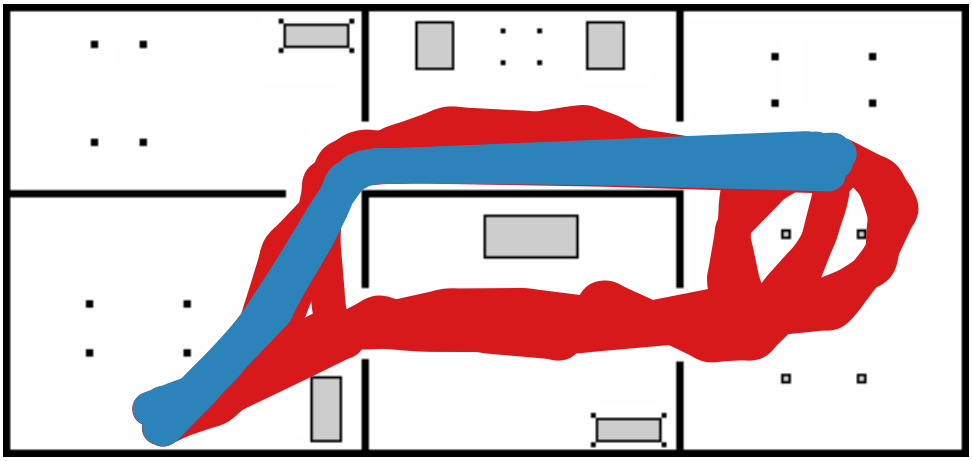
To show the ability of our planner to reproduce previous experiences for meeting the user's preferences, we assume that a user requires the robot to perform the 10 navigation tasks represented in Figure 4.9. These tasks are similar in a global sense, i.e., they have nearby start and goal poses. Figure 4.9(a) shows the paths



(a) Paths generated by using Bi-RRT.

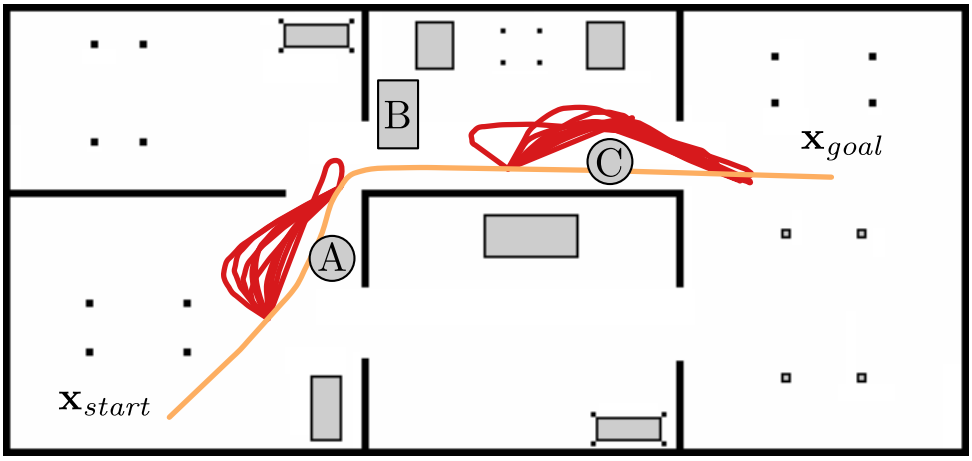


(b) Paths generated by using our approach given the green path as example.

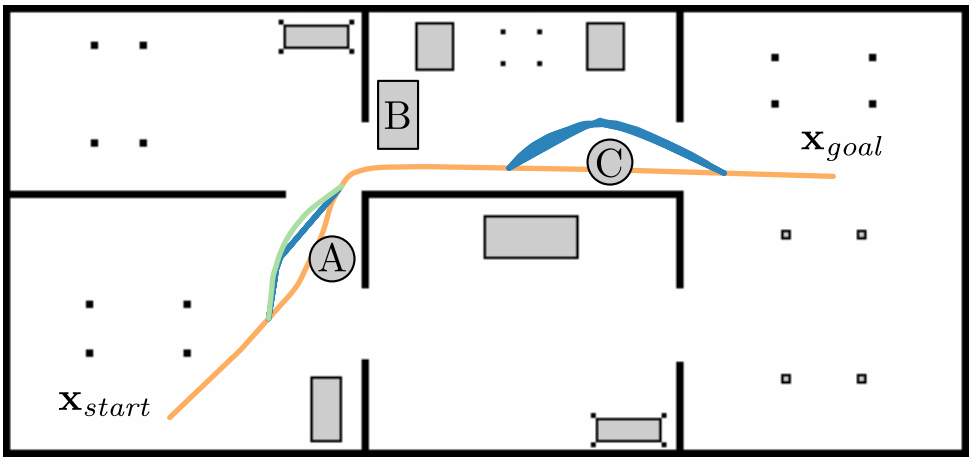


(c) Area covered by a circular robot to navigate along the paths generated by Bi-RRT (red) and by our approach (blue).

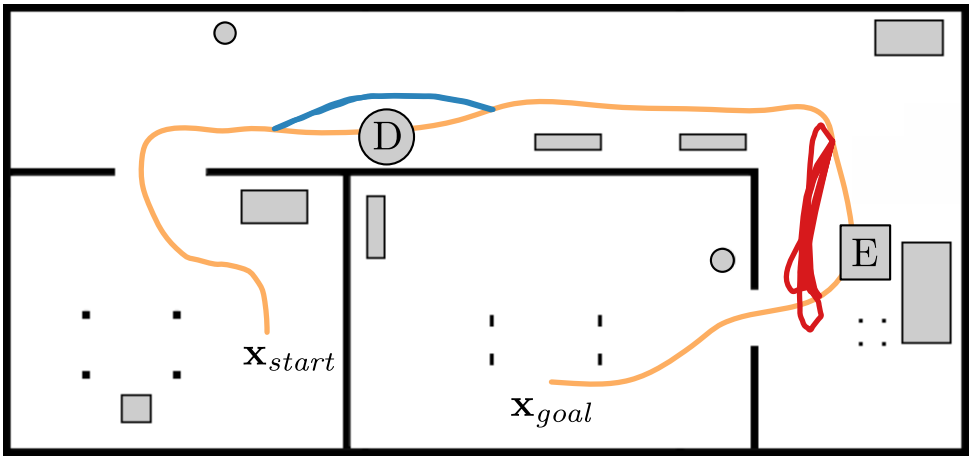
Figure 4.9: Planning global paths for a set of similar navigation tasks.



(a) Paths generated by using Bi-RRT.



(b) Paths generated by using our approach given the green path as example.



(c) Paths generated by using our approach given the green path in Figure 4.10(b) as example.

Figure 4.10: Planning local deviations to avoid unforeseen obstacles.

generated by Bi-RRT for these tasks. Due to the random nature of Bi-RRT, they reveal substantial differences from each other: some pass by the top central room, whereas others pass by the bottom central room. Thus, the user can make only limited predictions about the resulting paths for similar tasks. Furthermore, using such a planning algorithm the user cannot express any preference about where and how the robot should navigate.

Assume that the user prefers that the robot navigates passing by the central top room to perform these tasks. Our system enables the user to express this preference by evaluating positively or demonstrating the green path in Figure 4.9(b). Using this path as an example, our planner generates for the same tasks of Figure 4.9(a) the blue paths in the figure. These paths reproduce the behavior of the example and, thus, meet the user's preferences. Furthermore, using the same example, our planner generates similar solutions for similar tasks and, thus, makes the robot's behaviors predictable.

Our planner generates paths with such properties also for planning local deviations. Assume that the robot is following the orange path in Figure 4.10(a). Along the path, the robot encounters two unforeseen static obstacles A and C that block the path at two different locations. The robot needs to re-plan to avoid the obstacles and to reach its final goal. In Figure 4.10(a), we planned a local deviation 10 times for each obstacle using Bi-RRT. The resulting paths cause the robot to behave differently for distinct runs of the same task. Our planner prevents this result and it furtherly enables the user to express preferences about the robot's behaviors.

Assume that the user rates the green path in Figure 4.10(b) as a good example to avoid obstacle A. Our planning algorithm generates the blue paths exploiting this example path. These paths allow the robot to successfully avoid obstacle A reproducing the example and meeting the user's preferences. As the local situation at obstacle C is similar to the one at obstacle A, our planner transforms the experience at obstacle A into the new situation and generates deviations to avoid obstacle C that reproduce the previous behavior. Therefore, our system can reproduce behaviors and user's preferences across similar local situations.

This property holds also across different environments. Consider the scenario illustrated in Figure 4.10(c) where the orange path is blocked by two obstacles. The situation at obstacle D is similar to the one for which the user provided an example. Thus, our system generates paths that avoid the new obstacle by reproducing the experience in the new situation. Whereas, the local situation at obstacle E is different from the ones experienced so far and, thus, the example cannot be exploited. In this case, our planner generates new paths from scratch in the same way as standard Bi-RRT.

4.4.1.2 Performance analysis

To illustrate the performance of our planning approach, we considered 20 sets of 10 similar global navigation tasks and 20 sets of 10 similar local tasks. First, we plan for these tasks by using Bi-RRT and then by using our planner with 10, 20, 50, 100, and 200 examples randomly selected among the ones generated by Bi-RRT. We run this procedure 10 times for a total of 20,000 planning instances. We consider three measures to evaluate the performance of the planners: the time required for planning, the number of sampled states during the planning process, and the area of the environment covered by a circular robot to perform a set of similar tasks. The latter gives a measure of the similarity of the paths followed by the robot to perform similar tasks. The smaller the total area covered, the more similar are the paths. Figure 4.9(c) illustrates an example of the area covered by a robot when using Bi-RRT (red) and our planning approach (blue).

We illustrate the performance of each planning algorithm at global and local levels in Figure 4.11. The first chart shows the average time required for planning. When only a few examples are available, our system already outperforms Bi-RRT. The planning time decreases with increasing the number of examples up to approximately the 50% in the local case and the 60% in the global case. When the number of examples becomes large, the planning time tends to marginally increase due to the time needed to query the database of experiences.

The second chart of Figure 4.11 shows the average number of states sampled during planning. If a similar experience is available, our planner tends to sample the corresponding attractors instead of attempting to explore the whole configuration space as Bi-RRT. Thus, the larger the number of examples, the smaller is the number of sampled states during planning.

In the third chart, we compare the average percentage of the area of the environment covered by a circular robot to perform a set of similar tasks. Even with a few examples, the area covered by our system is approximately 25% smaller than by using Bi-RRT at the global level and 35% at the local level. The results at the local level are explained by the ability of our approach to generalize experiences across different obstacles and situations. When the number of examples available for each task increases, the covered area grows accordingly as the database of experiences contains one distinct example for each navigation task.

4.4.2 Trajectory prediction

The second set of experiments aims at evaluating our approach to predict the trajectory of moving objects. Our objective is to demonstrate that, within a short time horizon, our approach provides good predictions and that the uncertainty areas capture well the uncertainty of the motion. To this end, we recorded the

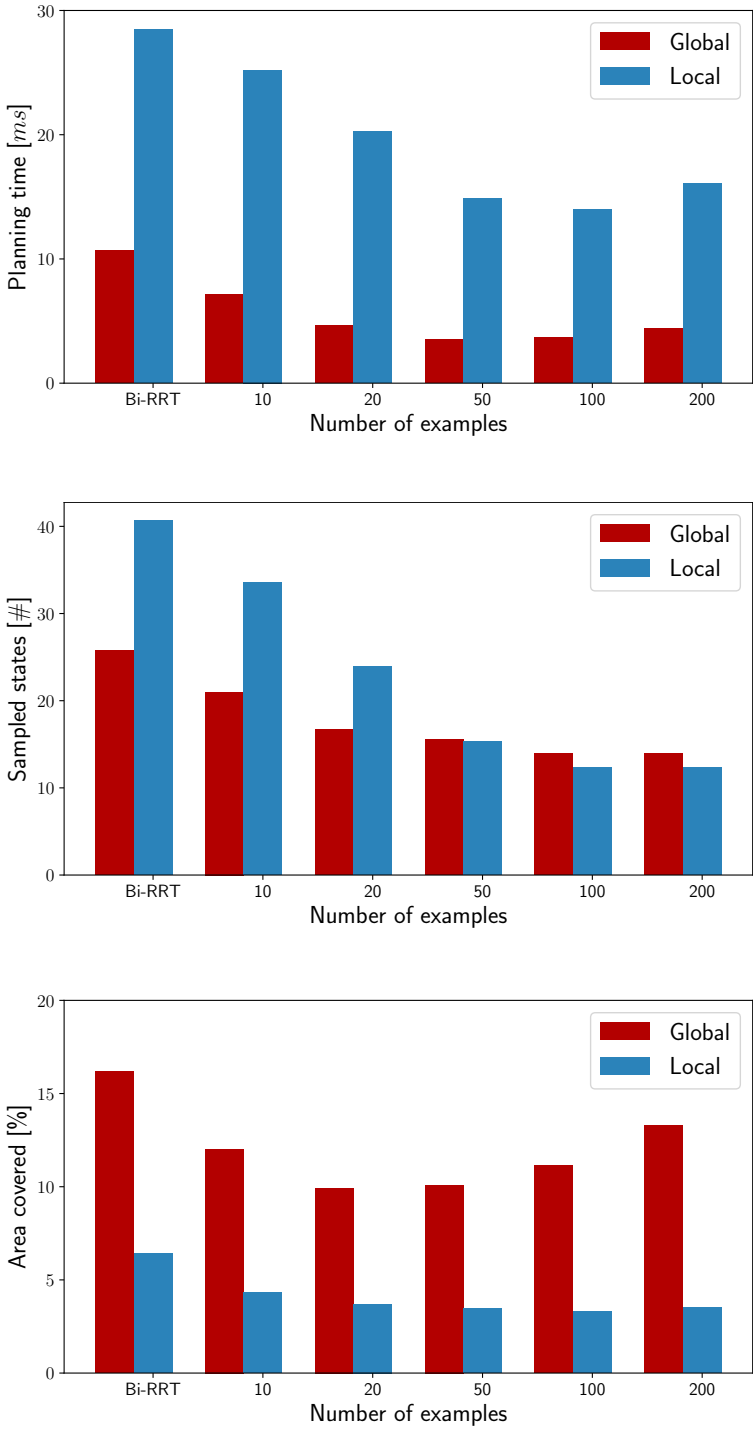
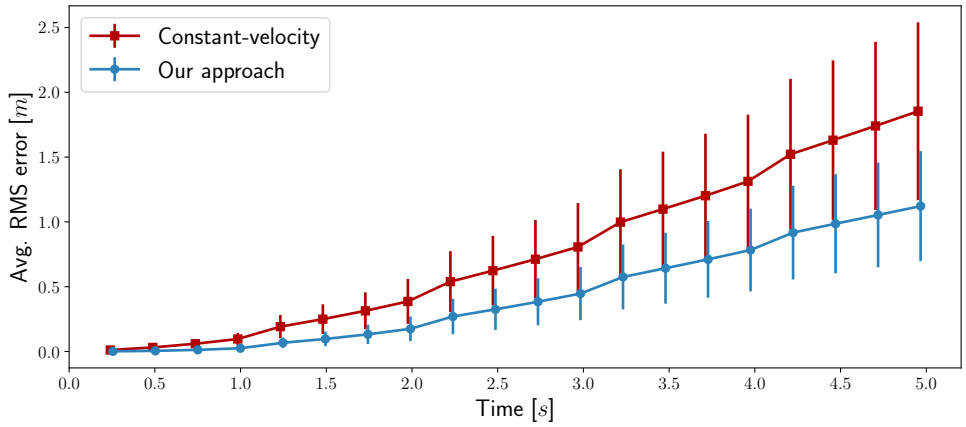
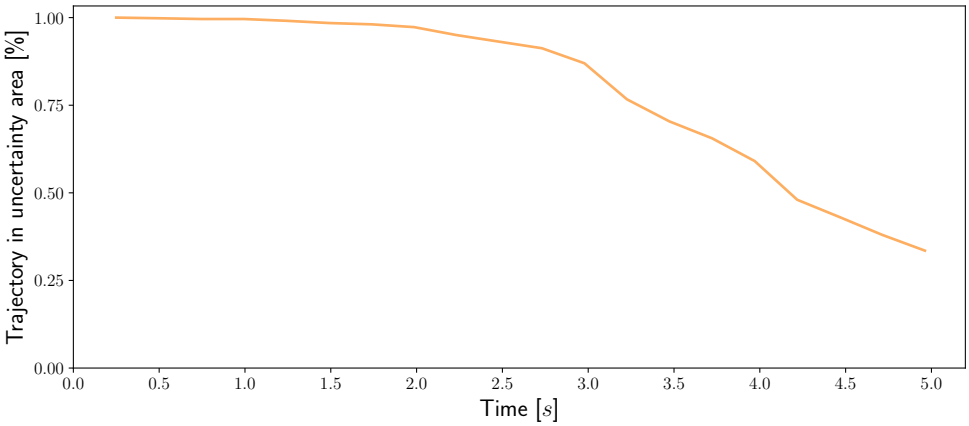


Figure 4.11: Performance comparison for planning using Bi-RRT and our approach with 10, 20, 50, 100 and 200 examples in the database of experiences.



(a) Average RMS error of the prediction.



(b) Percentage of the actual trajectory lying in the uncertainty areas.

Figure 4.12: Performance of our GP-based trajectory prediction model to predict the future trajectory of different people walking in a real-world scenario compared to a linear constant-velocity model.

trajectories of six people walking in our lab for approximately 120 s each by mean of a motion capture system. We considered 30 s of the trajectory of each person to train the hyperparameters of our model. We used the resulting model to predict the future trajectory of the remaining data within the next 5 s every 0.25 s for a total of approximately 2500 predictions.

The performance of our trajectory prediction approach is illustrated in Figure 4.12. We compare our approach with the trajectory predicted by a linear constant-velocity model. The first graph shows the root mean square error of the mean trajectory predicted by our approach with respect to the time of the prediction. Our approach always outperforms the linear model and presents half of the error for predictions up to 3 s. The second graph illustrates in percentage the number of times that the actual trajectory passes on to the corresponding uncertainty area. The percentage is above 90% for predictions up to 3 s. Afterward, it decreases linearly and, at 5 s in the future, the real trajectory is in the corre-

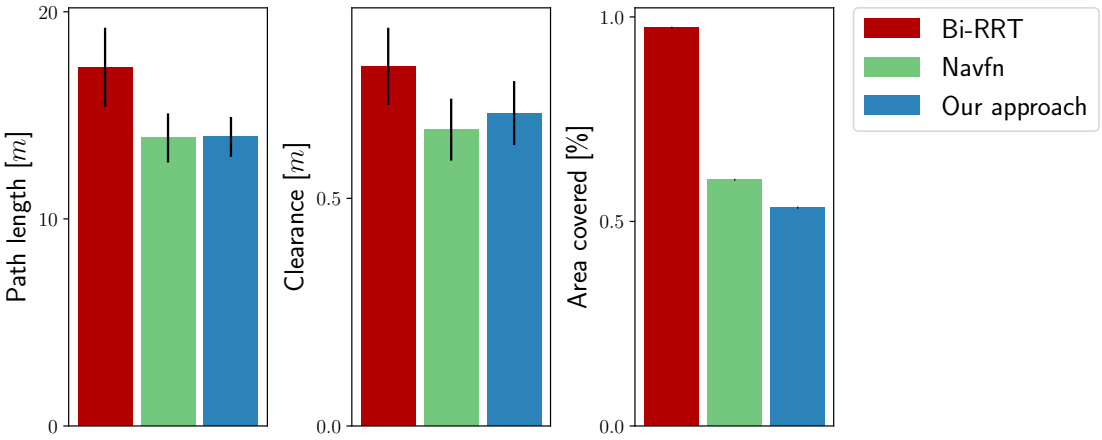


Figure 4.13: Performance of the different robot navigation systems in static simulated environments.

sponding uncertainty area around 30% of the cases. This outcome is partially due to the zero mean assumption made for the Gaussian processes in our model. It means in practice that, for long predicted time, the increments Δx and Δy will tend to zero. We contrast this effect by continuously updating the trajectory prediction and re-planning local deviations while avoiding an obstacle. Still, our approach provides good results for short-time predictions and 3 s is typically a reasonable time to react and avoid a collision with an obstacle moving in an indoor environment. In the next section, we will show that our approach to predict trajectories works well to safely avoiding dynamic obstacles also in practice.

4.4.3 Navigation in simulation

In this experiment, we aim at illustrating and discussing the performance of our navigation system by comparing it to other common approaches. We want to show especially the capability of our system to generate similar robot behaviors for similar tasks. We implemented our system by using C++ and ROS and tested it using the V-REP robot simulator. We considered a simulated KUKA Youbot mobile robot, equivalent to the one used for real-world experiments in the next section, and required it to perform multiple navigation tasks in different simulated indoor environments.

4.4.3.1 Static environments

First, we evaluate the performance of our robot navigation system at the global level. To this end, we require the robot to perform 10 sets of 10 similar navigation tasks in different static environments. To evaluate our approach, we include in the database of global paths one example per set randomly generated using Bi-RRT, so that our system has some experiences available to exploit while planning.

We compare the performance of our system with using standard Bi-RRT and the Navfn planner by ROS as planning algorithms. The Navfn implements a shortest path planner based on Dijkstra's algorithm and it is currently the most commonly used approach in ROS to plan paths for robot navigation. We implement Bi-RRT by using our system without providing any example. We analyze three measures to evaluate their performance: length of the executed path, average clearance from the obstacles during navigation, and area covered by the robot to perform a set of similar tasks.

The results of our experiments are illustrated in Figure 4.13. As expected, the Navfn generates in average the shortest paths, whereas Bi-RRT presents large clearance. Using our approach, the performance depends strongly on the example exploited by the planning algorithm. A first indicator that our approach provides similar solutions for similar tasks is that it presents the lowest standard deviation both for path length and clearance. To further support this, our system covers almost half of the area covered by using Bi-RRT. The Navfn also covers a small area as it attempts to minimize the path length and, by doing this, the paths for similar tasks tend to pass by the same locations. However, it does not allow for easily expressing any preference about which these locations should be.

4.4.3.2 Dynamic environments

We introduce moving obstacles in the simulated environments to evaluate our navigation system at the local level. We defined six situations that differ for: navigation task, obstacle's velocity, trajectory, and type of collision that could occur (frontal, lateral, etc.). We compared our system to a reactive approach as Dynamic Window Approach (DWA) [45], an approach that considers explicitly time as Timed Elastic Bands (TEB) [69], and our system with no examples that is equivalent to Bi-RRT. We used the available ROS implementation of DWA and TEB. For simplicity, we provided only two local examples to our system: one avoiding the obstacle on the left side and one on the right side. We fixed the global path and performed each task 20 times for every navigation algorithm. We considered five measures to evaluate their performance: length of the executed path, deviation from the global path, execution time, clearance from the obstacles, and area covered to perform one task multiple times.

The performance of the different algorithms for this experiment is illustrated in Figure 4.14. Our system leads the robot to navigate along the path with the shortest travel distance on average. The reason for this result is that our approach continuously updates the local path while avoiding an obstacle and, thus, the robot goes back as soon as possible to the global path. This is also revealed by the chart representing the deviation from the global path: our approach is the one that leads to the smallest deviation. Bi-RRT also provides good results

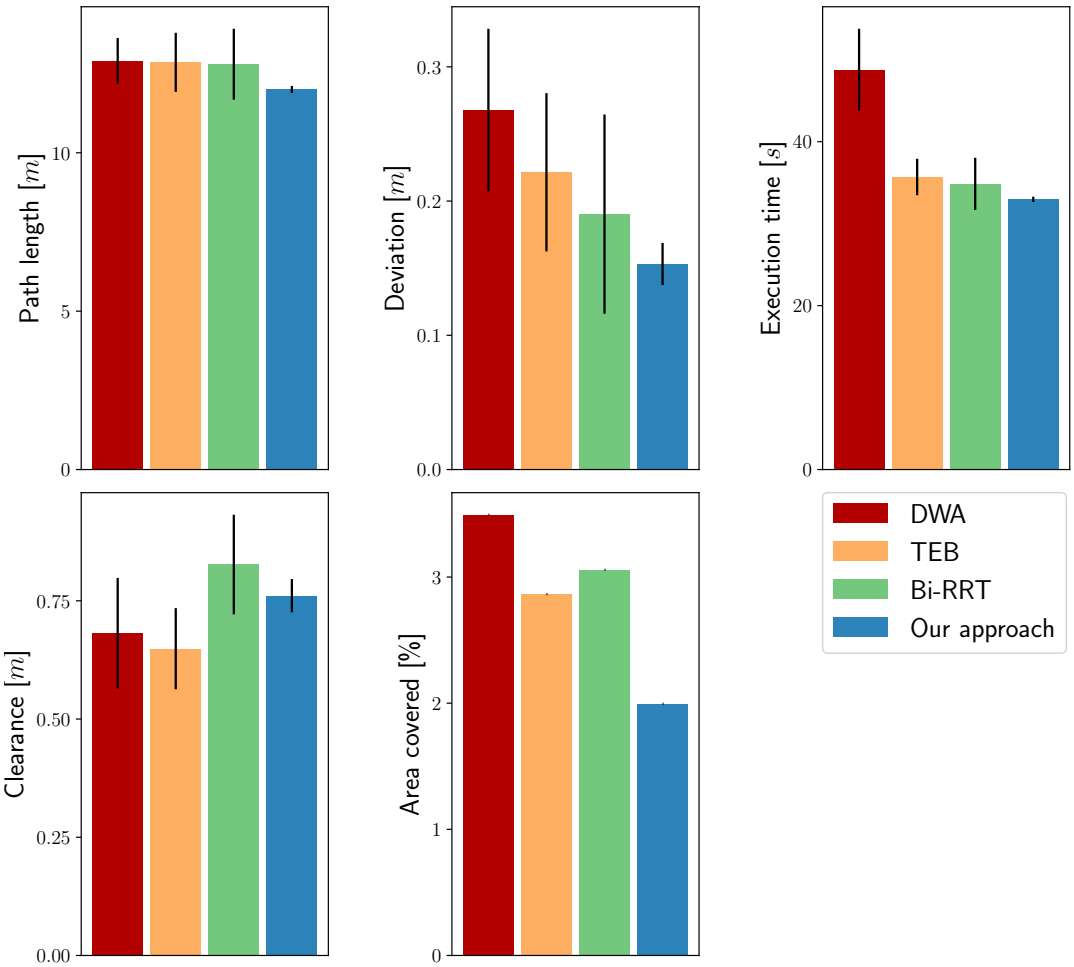


Figure 4.14: Performance of the different robot navigation systems in simulated environments populated by dynamic objects.

	Re-planning frequency [Hz]	Obstacle velocity [m/s]					
		0.4		0.8		1.0	
		coll.	fail.	coll.	fail.	coll.	fail.
DWA	20.0	0.0	0.0	0.38	0.05	0.25	0.0
TEB	5.0	0.0	0.0	0.01	0.0	0.15	0.03
Bi-RRT	20.0	0.0	0.0	0.0	0.0	0.01	0.0
Our approach	20.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 4.1: Statistics of the dynamic obstacle avoidance experiments.

on average as it is implemented within our navigation system, but it presents a large standard deviation due to its random nature. The performance for the execution time reproduces the ones for the path length except for DWA which typically requires a longer time to reach the final orientation. The last chart shows that our system allows the robot to cover the smallest area to perform the navigation tasks 20 times. This result demonstrates the capability of our approach to generating similar behaviors for similar situations also at the local level.

Table 4.1 reports for each algorithm the re-planning frequency, the collision, and failure rates to avoid obstacles with different velocities that we experienced in our experiments. A collision (labeled as "coll.") occurs if the robot collides with an obstacle but it is then able to reach the goal. A failure (labeled as "fail.") takes place if the robot is not able to reach the goal within a certain amount of time. As TEB computes optimal trajectories taking explicitly into account the time dimension, it re-plans at a lower frequency than the other approaches. This is reflected in collision and failure rates: when the obstacle is fast, TEB does not always react fast enough to avoid it. Even if it can re-plan at 20 Hz, DWA may also lead the robot to collisions and failures when the obstacle velocity increases. Our system is, in general, safer. If no example is available (labeled as "Bi-RRT"), it caused collisions only in one case. This episode is due to the random nature of Bi-RRT that for obstacle avoidance may generate completely different local behaviors for successive planning instances. In this case, the robot oscillates and wastes time to react to the obstacle thus causing, in the worst case, a collision. Instead, when the planning process of our system is guided by a previous local example(labeled as "Our approach"), the robot experienced no collisions or failures.

4.4.4 Real robot navigation

The next set of experiments is designed to show that our approach works effectively on real robot navigation in the physical world. The robotic platform used in these experiments is the KUKA Youbot illustrated in Figure 4.15, which is an omnidirectional mobile robot that we equipped with two Hokuyo laser range finders placed at two opposite corners so to have 360° measurements. Using these sensors, the robot localizes itself in a previously built map of the environment by means of a Monte Carlo localization algorithm.

4.4.4.1 User-preferred behaviors

We aim at showing that our system allows the user to easily lead the robot to perform behaviors that may be hard to encode in typical navigation systems. To

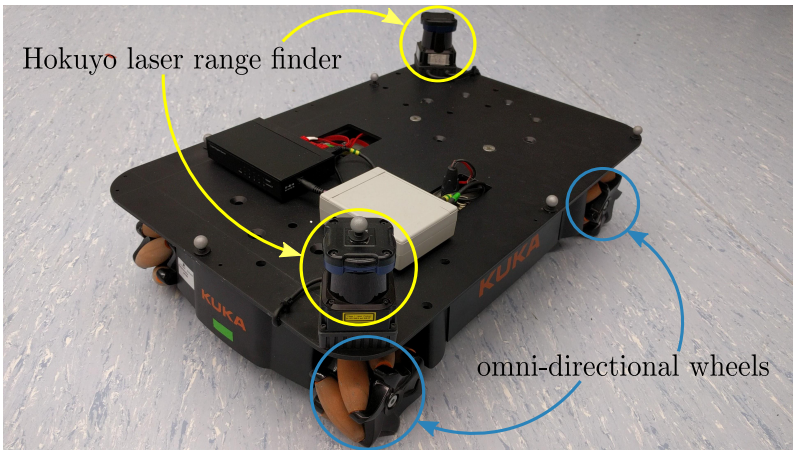


Figure 4.15: Our KUKA Youbot mobile robot equipped with two laser range finders.

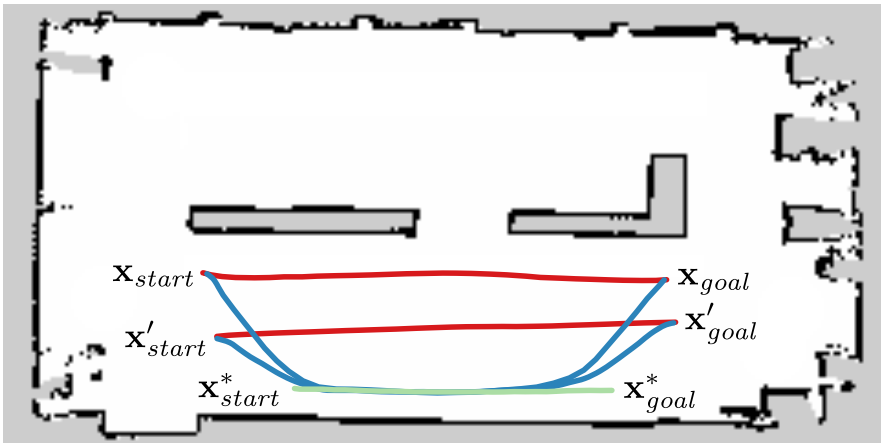
this end, we defined three simple navigation tasks that a user may require the robot to perform by following his preferences.

The first task is illustrated in Figure 4.16(a) and consists of navigating from one side to the other of a hallway. We assume that the user wants the robot to accomplish this task by keeping on the right side of the hallway. If no example is available, the robot navigates straight to the goal (red paths). So, we joysticked the robot along the green path, which passes on the right side of the hallway, and stored it as an example into the database of experiences. By exploiting such an example, our system can lead the robot to navigate from one side to the other of the hallway by keeping on the right (blue paths).

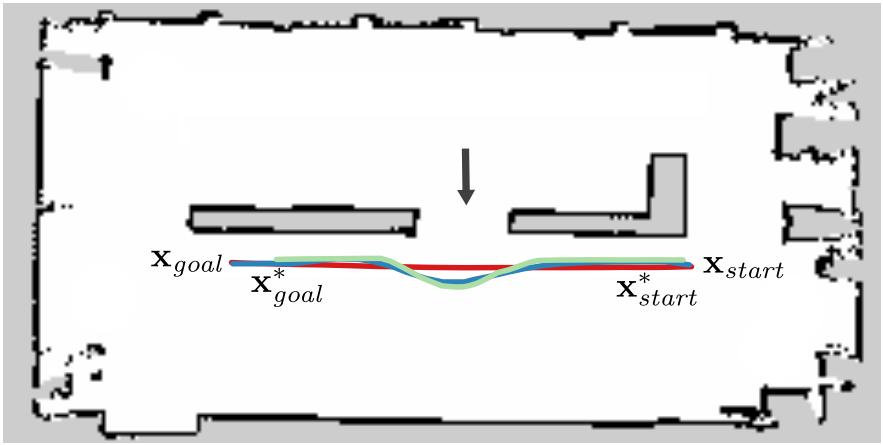
The second task is illustrated in Figure 4.16(b). In this case, we imagine that the user wants the robot to navigate through the hallway without obstructing the open passage. Without any example, the robot follows a straight line to the goal (red path) with the risk to block the passage. As we demonstrate the green example path, which circumvents the passage, the robot navigates along the blue path to reach the other side of the hallway, thus meeting the user's preferences.

In the third task, illustrated in Figure 4.16(c), we required the robot to navigate from one side to the other of a table by avoiding to pass under it. If no example is provided, the robot navigates along the red path which passes under the table, as only the legs of the table are marked as obstacles in the map. When the green path becomes available as an example, the robot executes the blue path which passes on the left of the table to reach the opposite side (blue path).

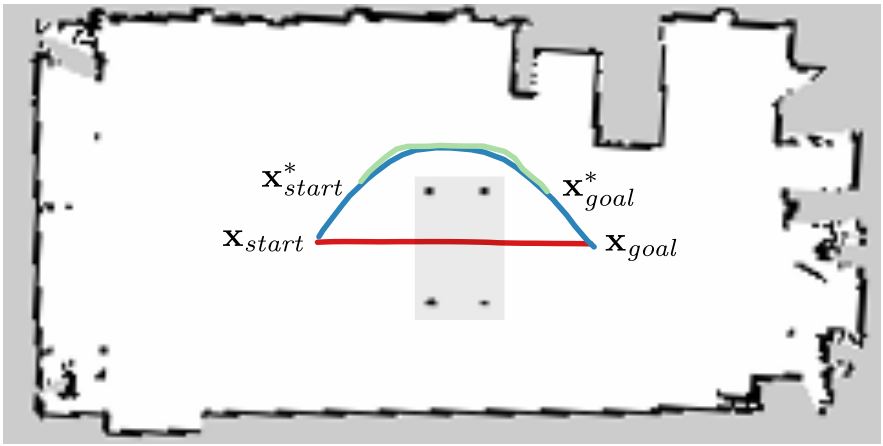
Although the tasks and requirements that we considered in this experiment are very simple, designing a cost function or a rule-based logic that implements these behaviors and integrating it in a standard planner such as A* may be very complex. In contrast to that, our approach allows for easily incorporating such preferences in the robot's behaviors.



(a) Navigation in a hallway keeping on the right.

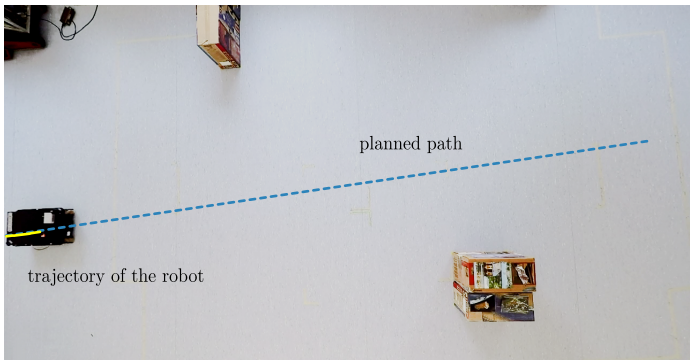


(b) Navigation in a hallway without obstructing the passage.

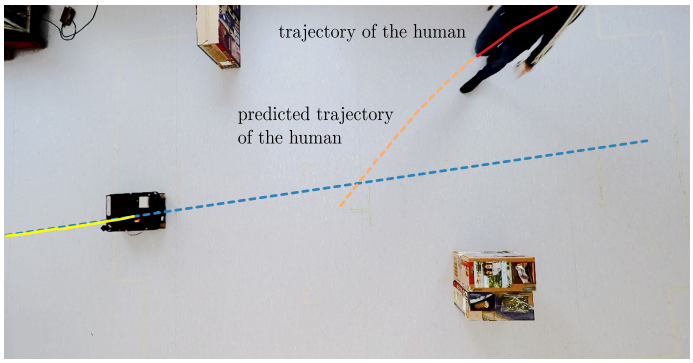


(c) Navigation avoiding to pass under the table.

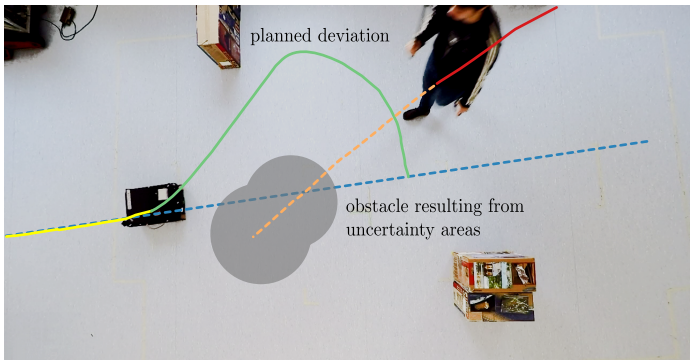
Figure 4.16: Examples of user-preferred behaviors for navigation in real world.



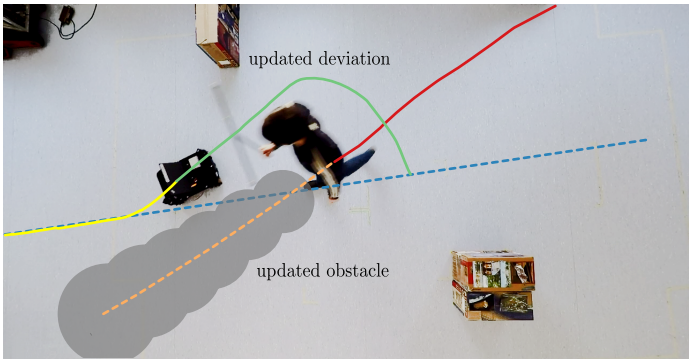
(a) Robot navigates along the planned path.



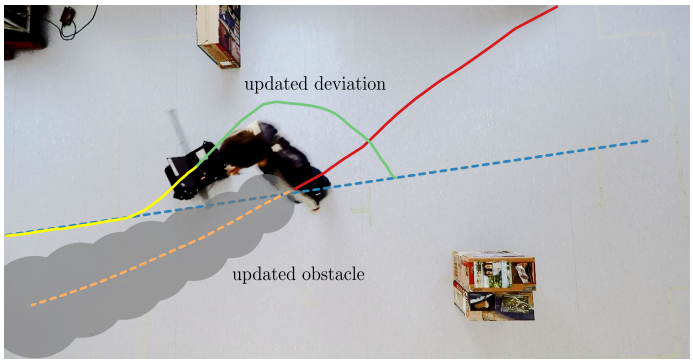
(b) Human walking nearby the robot.



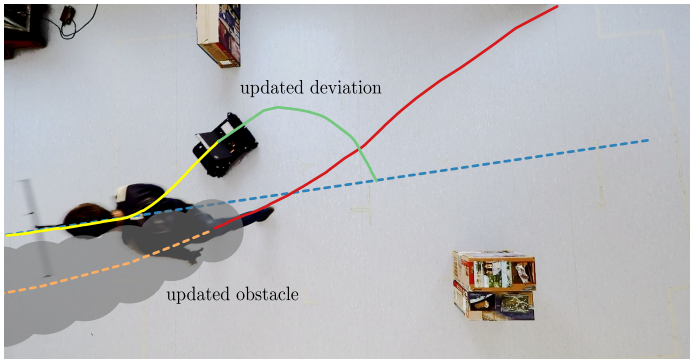
(c) Future collision detection and planning local deviation to avoid the obstacle.



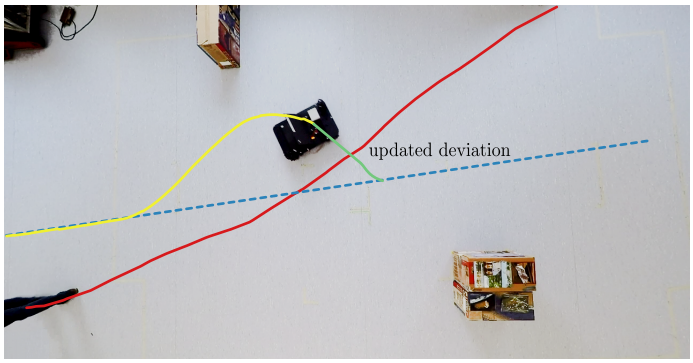
(d) Deviation update (I).



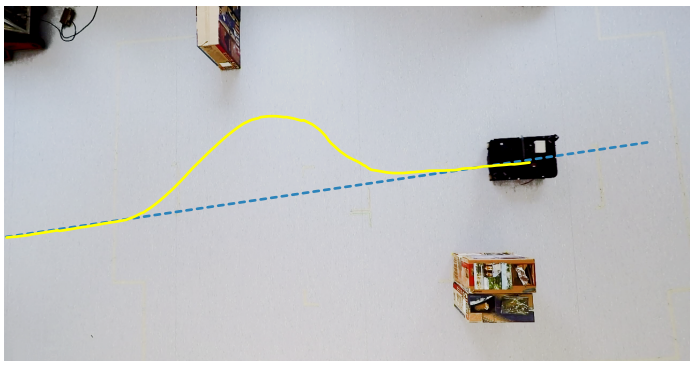
(e) Deviation update (II).



(f) Deviation update (III).



(g) Deviation update and convergence to the original path.



(h) Robot resumes navigation along the original path.

Figure 4.17: Robot navigates along a given path avoiding a human blocking its way.

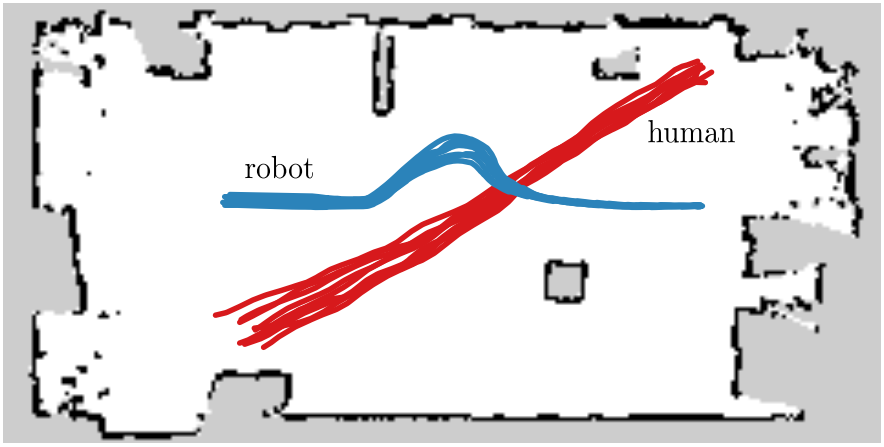


Figure 4.18: Trajectories of robot (blue) and human (red) for similar situations in which the robot avoids collisions with the human with our navigation system.

4.4.4.2 Avoiding collision with people

We furthermore tested the capabilities of our system to avoid walking people who may block the robot's path. We designed different setups in our lab in which the robot and a human could move through, and used a motion capture system to observe their trajectories.

Figure 4.17 illustrates an example in which our experience-based navigation system allows the robot to avoid a human crossing its way. Initially, the robot navigates along the blue dotted path, see Figure 4.17(a). The yellow line shows the actual trajectory followed by the robot. In Figure 4.17(b), a human walks into the scene. The observations of his trajectory are depicted in red, while the orange dotted line represents the predicted mean of his future trajectory. Our system detects a future collision in Figure 4.17(c) and generates the gray artificial obstacle from the uncertainty areas of the predictions. The robot plans the green local deviation, which avoids the obstacle by exploiting a previous experience. As the robot collects more and more observations of the human's trajectory, our system updates the local deviation according to the trajectory prediction, see Figure 4.17(d-f). When the human does not block the original path anymore, our system leads the robot back to following the original global path, as shown in Figure 4.17(g). In Figure 4.17(h), the robot gets back to the global path and continues its navigation along it.

We repeated the same situation 10 times. The trajectories of the robot and the human recorded by the motion capture system are illustrated in Figure 4.18. The figure reveals once again the capability of our system to lead the robot to perform similar behaviors in similar situations.

4.5 Related work

The navigation system presented in this chapter implements a framework that allows for easily incorporating the preferences of non-expert users into the navigation behaviors of a mobile robot.

In the literature, there exist several different approaches that define preferences and behaviors for robot navigation. Some approaches achieve this by defining cost functions over the environment. For example, Haigh and Veloso [55] propose to learn a situation-dependent cost function and to use these costs for planning. Other approaches define preferences as a set of rules or constraints. Kirby *et al.* [70] formulate human social conventions, such as personal space and tending to one side of hallways, as constraints for robot navigation and plan robot's behaviors by solving constraint optimization problems. Whereas, Carlucci *et al.* [23, 114] introduce a framework for explicitly representing norms to plan and execute socially acceptable behaviors. These approaches typically require an expert to define and maintain complex costs, constraints, and norms. In contrast to that, our approach allows for easily incorporating the preferences of a non-expert user by collecting his feedback about the robot's experiences and by reproducing the preferred behaviors in similar navigation tasks.

A general framework for automatic reasoning based on previous experiences is the case-based reasoning. In case-based reasoning, experiences are stored as cases and used to suggest solutions to new problems. In the context of robotics, Likhachev *et al.* [97] uses this approach to select a set of parameters that actively determine the behavior of the robot; Meriçli *et al.* [110] adopt it for mobile push-manipulation tasks; whereas, Ros *et al.* [143] use it for action selection in the robot soccer domain.

Teach-and-repeat approaches allow for reproducing previously taught experiences. Sprunk *et al.* [154] reproduce trajectories with high accuracy by matching laser scans. Furgale *et al.* [51] propose a vision-based approach based on topologically connected sub-maps. Furgale *et al.* [50] extend standard teach-and-repeat by adding a local motion planner to account for dynamic environments. Perea Ström *et al.* [132] use a similar approach for guiding a robot home in case the mapping system fails during an exploration mission. Mazuran *et al.* [108] optimize the demonstrated trajectories within constraints defined according to the user's preferences. In contrast to teach-and-repeat, we incorporate the previous experiences in a new flexible motion planning process. In this way, our approach can reproduce experienced paths even when changes occur in the environment and generalize local behaviors over different situations and environments.

Sampling-based planning algorithms are a class of commonly used planners for robot navigation which includes Rapidly-exploring random tree (RRT) [91]

and its variants such as RRT-Connect [84] and RRT* [67]. These approaches are fast to discover the connectivity of the configuration space but it is difficult to predict the outcoming paths due to their random nature. A short description of sampling-based planning is provided in Section 2.3.2. Many planning approaches proposed to refine the RRT search by biasing the sampling process. For example, Bruce and Veloso [21] extend traditional RRT to reuse cached plans and bias the sampling process towards their way-points. Zucker *et al.* [179] adapt the sampling process by taking into account the features of the configuration space. Jiang and Kallmann [64] introduce the Attractor Guided Planner, a planning algorithm that stores every successful path and biases the sampling for a new plan to reproduce the structure of an experienced path according to a similarity function. Our planning approach is based on a bi-directional variant of RRT that reuses and generalizes the idea of experience-guided sampling to plan paths for robot navigation.

There exist several motion planning approaches that take advantage of previous experiences to speed up the planning time for finding paths in high-dimensional configuration spaces. Fraichard and Delsart [34] deform previous trajectories in the configuration-time space to fit new situations. Phillips *et al.* [135] build a graph of experiences and exploit it for performing repetitive tasks. The Lightning framework [12] repairs previously generated trajectories to fit new planning problems. Coleman *et al.* [29] extend this idea by storing generated paths in a sparse road-map. We store the previous successful robot's behaviors and exploit them in the subsequent planning processes to generate trajectories that reproduce the user's preferences.

Our approach stores the user-preferred robot's behaviors in a database of experiences by considering a path representation that allows for effectively reproducing behaviors and a situation descriptor to identify the previous paths that fit new situations. Jetchev and Toussaint [62] learn a mapping between situations and trajectories and use a descriptor of the situation to transfer and optimize a previous trajectory in a new situation. They also extend this work by using a voxel-grid representation of the environment to generalize trajectories to a wider range of situations [63].

Our navigation system allows the robot to navigate in dynamic environments and to reproduce user-preferred behaviors for avoiding collisions with moving obstacles. There exist several different approaches to dynamic obstacle avoidance. Many of them rely on reactive strategies such as Dynamic Window Approach [45]. It repeatedly selects a velocity yielding to a safe trajectory within a short time interval. Elastic Bands [140] combines a reactive strategy with global planning to avoid obstacles. Rösmann *et al.* [144] and successively Keller *et al.* [69] extend this approach to consider explicitly time information for optimal trajectory planning.

These approaches enable robots to avoid dynamic obstacles but do not allow for expressing preferences about their behaviors. Furthermore, our experimental evaluation reveals that our approach is safer than the other tested methods to avoid collisions with dynamic obstacles.

Another approach to performing compliant robot navigation in populated environments is to explicitly model human motion. Kruse *et al.* [80] conduct a survey about human-aware navigation. Ziebart *et al.* [177] learn a cost function of features to predict people's trajectories and plan paths that avoid hindrances. Bennewitz *et al.* [10] learn collections of trajectories that characterize typical human motion patterns. Kuderer *et al.* [83] as well as Kretzschmar *et al.* [79] model cooperative navigation behaviors of humans to let the robot interact with people in a socially compliant way. Pfeiffer *et al.* [134] propose a navigation model trained on human-human interaction. In the scenario that we consider in this chapter, the robot shares the workspace with different moving objects such as human workers, other robots, or forklifts. Thus, we model the trajectory of the other agents by using a probabilistic approach based on Gaussian processes [142] that does not require to make any specific assumption about the motion model and the trajectories of the obstacles.

GPs have been already successfully employed to model the trajectories of moving objects. Trautman *et al.* [163] introduce Interacting Gaussian processes for navigating through dense human crowds. Such representation describes a probabilistic interaction between multiple navigating entities to accomplish cooperative collision avoidance. Fulgenzi *et al.* [49] use GPs to model typical obstacle trajectories and planned with a variant of RRT that explicitly considers the probability of collision. Whereas, Ellis *et al.* [38] show that GPs allow for explicitly capturing the uncertainty of the human motion.

4.6 Conclusion

The goal of the work presented in this chapter is to build a non-rigid robot navigation system that incorporates the user's preferences, and that can be easily adopted by non-expert users. Following the user's preferences is especially important in environments such as factory floors where the operator often requires a robot to be predictable and to follow specific patterns for navigation. We presented a system that allows the robot to perform navigation behaviors that meet the user's preferences by exploiting its previous experiences. Preferences are implicitly extracted through simple demonstration and feedback about the robot's behaviors compared to writing complex code. We store the preferred behaviors experienced by the robot over time in a database of experiences. Our experience-based planning algorithm computes paths that reproduce and gener-

alize these experiences over situations and environments. This approach leads the robot to accomplish similar behaviors for similar tasks according to the user's preferences. Our navigation system is able to apply the same planning scheme both for planning global paths and for computing local deviations to avoid obstacles. We also define a dynamic obstacle avoidance strategy that combines our experience-based planner with a probabilistic model to predict the trajectories of other moving agents. This approach allows for planning safe and predictable deviations to avoid unforeseen obstacles.

We implemented and evaluated our robot navigation system over an extensive set of experiments comparing it with different approaches commonly employed on mobile robotic systems. We performed tests both in simulation and on an actual mobile robot operating in the real world. The experiments suggest that our system is able to reproduce previous behaviors for robot navigation, thus meeting user's preferences both at global and local levels. Furthermore, it allows the user to easily incorporate commonly demanded robot's behaviors in the planning process that may be hard to encode in typical navigation systems.

Part II

Navigation with active information gathering

Chapter 5

Improving navigation exploring and modeling different terrains

The solutions presented in the first part of this thesis take advantage of the available background information about the environment and the robot's experiences to optimize navigation. However, most robots operating in real-world environments have only partial knowledge about the world. One approach to discover relevant information for navigation is to actively collect informative observations while navigating. The second part of this thesis focuses on solutions that integrate active information gathering to improve robot navigation in an environment over time.

Most robot navigation systems, including the ones presented in Part I, rely on known static representations of the environment such as occupancy grid maps or topological maps for planning paths and navigating to targeted locations. Such maps represent the geometry and the connectivity of the environment but do not capture many other factors that might affect the quality of robot navigation. One of the factors that affect navigation the most is the surface on which the robot drives. This factor becomes especially critical for robots operating in outdoor environments where there exists a variety of terrains characterized by different degrees of roughness and irregularities.

Over the last decade, mobile robots have found many applications in outdoor environments, for example for environmental inspection and surveillance [17, 102, 161], precision agriculture [3, 24, 41, 96], urban navigation [56, 89, 95, 166], and many others. Robots that navigate in the outdoors are exposed to detrimental factors due to the terrains that may cause many undesirable phenomena such as strong vibrations or high power consumption. Therefore, the choice of the terrain on which the robot should move and, thus, of the specific path that it should follow have a significant impact on the way it navigates. Consider, for example, the scenario depicted in Figure 5.1. The red path leads the robot from the start

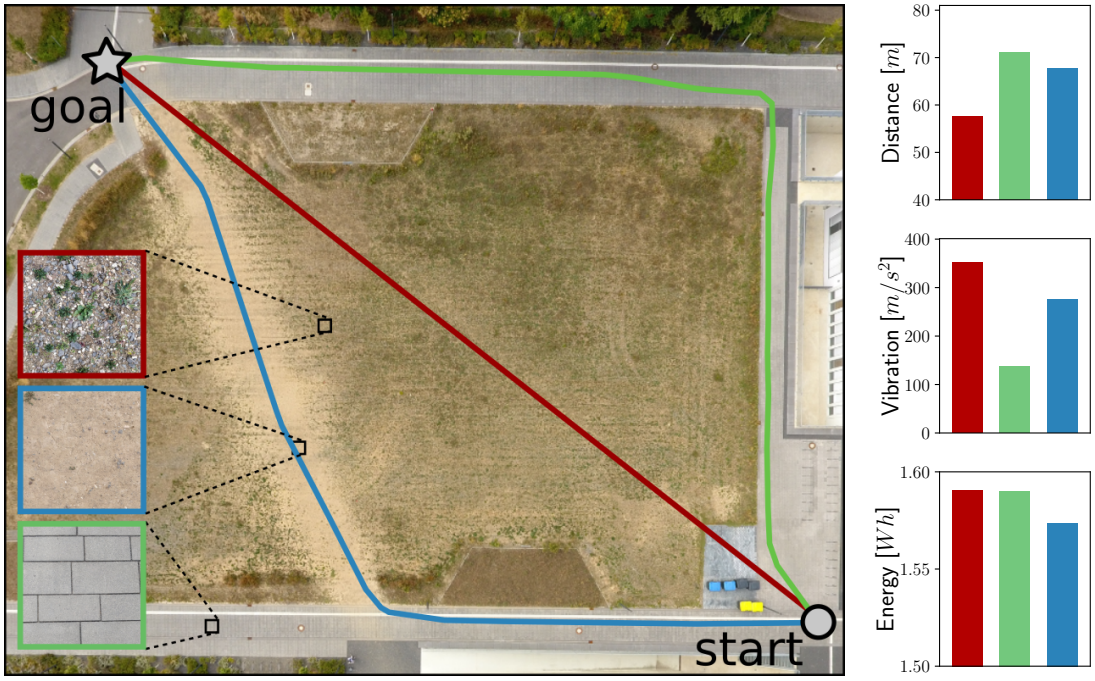


Figure 5.1: Robot navigation to reach a goal location following three different paths on diverse terrains (left), and the travel distance, the amplitude of the vibrations and the energy consumption that the robot experiences to navigate along these paths (right).

to the goal location along the shortest path. Although this path minimizes the travel distance, it might not be the best path to follow as it goes on gravel, which causes the robot to suffer strong vibrations during traversal, as shown on the right of Figure 5.1. Strong vibrations can be problematic for the robot's perception capabilities. They may cause blurred camera images [171] and noisy inertial measurement unit (IMU) data [40] that may affect localization. Furthermore, excessive vibrations might damage or affect the durability of the robot hardware. Another aspect of navigation that is strongly affected by the nature of the terrain is power consumption. This factor becomes particularly critical when deploying the robot over long periods of time or distance [126, 7]. Despite the short travel distance, following the red path, the robot consumes a similar amount of energy as following longer paths. For example, the green path in Figure 5.1, which follows on a paved road, offers significantly lower vibrations than following the red one while requiring a similar energy expenditure. In comparison, the blue path, which follows on a paved and a dirt road, requires less energy while offering a balanced experience in terms of traveled distance and vibrations. Therefore, depending on the task and on the environment, it is important to take these factors into account to achieve the desired navigation behaviors.

In this chapter, we investigate the problem of actively improving robot navigation by planning paths that aim at reducing detrimental factors such as vibrations

or energy consumption during traversal. One approach to achieve this is to associate a cost to these factors and to plan paths that minimize the cost to reach the goal. However, in reality, vibrations and energy consumption of navigating in an environment are unknown a priori. But, these quantities can be observed by the robot during navigation, e.g., by an IMU or an energy monitor. There exist several approaches that classify the terrain types from online perception and arbitrarily assign high costs to uneven grounds, for example, Wurm *et al.* [174]. However, such approaches might not always reflect adequately the physical quantities that the robot actually experiences while navigating. On the contrary, we aim at continuously learning and improving a place-dependent model of the detrimental phenomena from the robot's observations. We exploit this model for planning paths that trade off the exploration of unknown promising regions and the exploitation of known areas where the impact of the detrimental factors on navigation is low. Following our approach, the robot experiences reduced vibration intensities and energy expenditure and, thus, improves its navigation over time.

5.1 Modeling phenomena due to terrains

During navigation, the robot observes many phenomena that affect the navigation and that depend on the terrain on which it navigates, which is unknown a priori to the robot. To plan and navigate along paths on which the effect of detrimental factors is limited, we need to estimate their impact on navigation over the whole environment. To this end, we can learn a place-dependent model of a specific physical quantity over the environment from the robot's observations during navigation. We use this model to make predictions about the intensity of the modeled phenomena at unvisited locations and to plan paths along which its impact on navigation is reduced.

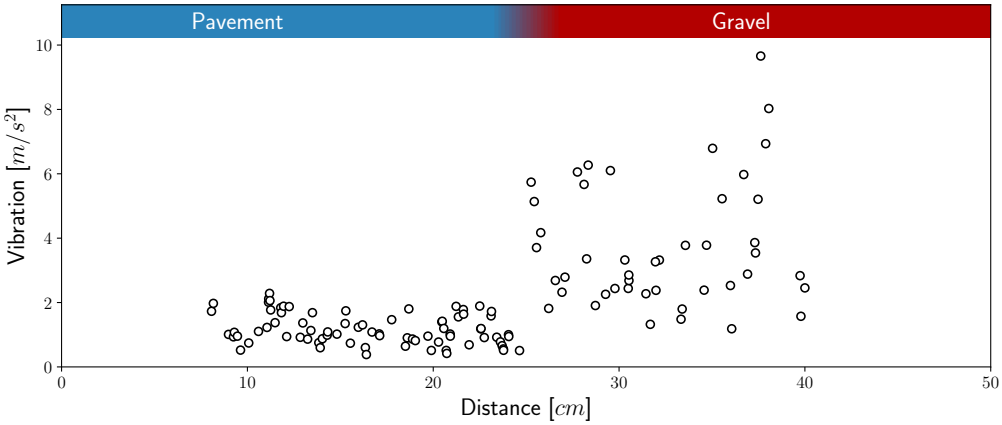
5.1.1 Gaussian process model

An effective method for modeling spatial processes from observations, such as a place-dependent cost function, is GP regression [142]. This is the same technique that we used in Chapter 4 for modeling dynamic obstacles and their occupancy over time.

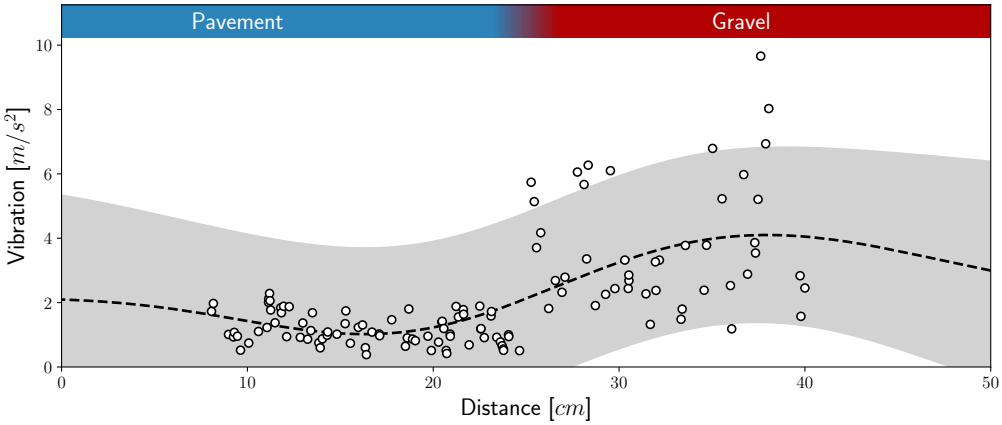
Given a set of observations $\mathbf{z}$ of a physical quantity f observed at the locations $\mathbf{X}$, GP regression allows for learning a predictive model of f at the unvisited location $\mathbf{x}_*$ as:

$$\mu_* = \mathbf{k}_*{}^\top \mathbf{K}_{\mathrm{XX}}{}^{-1} \mathbf{z}, \tag{5.1}$$

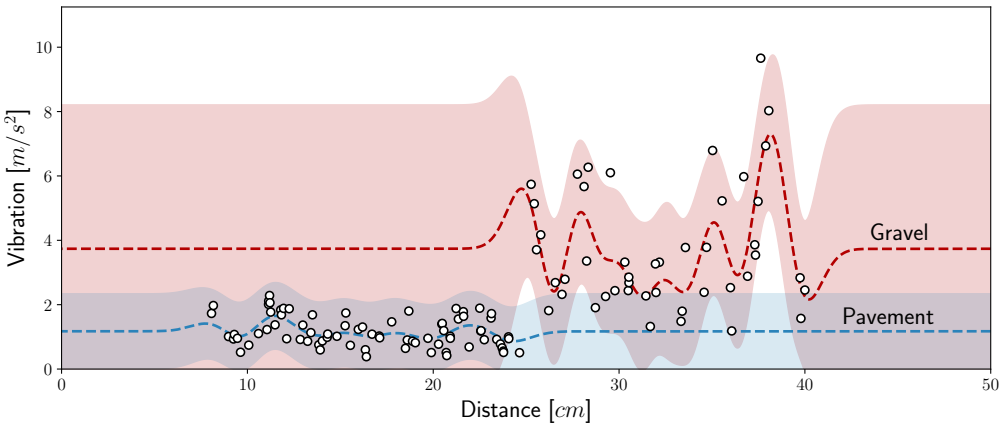
$$\sigma_*^2 = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*{}^\top \mathbf{K}_{\mathrm{XX}} \mathbf{k}_*, \tag{5.2}$$



(a) Observed vibration intensities.



(b) Single GP model.



(c) One GP model for each terrain.

Figure 5.2: Observed vibration intensities during robot navigation on two different terrains (a) and GP models of the vibration intensity in the environment(b)(c). The circles are the observed vibrations, the dotted lines are the mean prediction, the shaded areas correspond to the 2σ confidence bound of the prediction. The blue-red gradient shows the terrain on which the robot navigates.

where $\mathbf{K}_{\mathrm{XX}} = \mathbf{K}(\mathbf{X}, \mathbf{X}) + \varsigma_n^2 \mathbf{I}$, $\mathbf{k}_* = k(\mathbf{x}_*, \mathbf{X})$, and $k(\cdot, \cdot)$ is the covariance function. The predictive distribution of the quantity at $\mathbf{x}_*$ is $\mathcal{N}(\mu_*, \sigma_*^2)$. For a short summary on GP regression, see Section 2.2.2.

Using this predictive model, we can model the vibration intensities due to the terrain experienced by the robot during navigation in an environment. Consider the example illustrated in Figure 5.2 in which, for simplicity, a robot navigates in a 1D environment. The robot moves at first on pavement and then on gravel observing different vibration intensities (white circles). Given the observed intensities and the corresponding locations, GP regression can learn a predictive model of the vibrations over the environment by using Equation (5.1). The resulting GP is illustrated in Figure 5.2(b) where the black dotted line is the predicted mean and the gray area represents the 2σ confidence bound of the predictions. The GP tries to learn a single characteristic that fits all of the observations. However, while navigating on pavement and gravel, the robot experiences vibration intensities characterized by quite different modes. This results in a little informative model that smooths out the peaks and the signal trends, and that, thus, might not capture the reality well. If, for example, a peak corresponds to a stone or a hole in the ground, a smooth GP model may cause the robot to navigate through that location again.

For comparison, consider Figure 5.2(c) where we use an independent GP model for each terrain on which the robot navigates. Each model represents only the vibrations on one terrain but they are more accurate and informative than the single-GP model learned in Figure 5.2(b). This indicates that learning a GP with a single characteristic would be sufficient for modeling a physical quantity over an environment characterized by a unique uniform terrain, but it hardly fits situations in which the robot navigates in outdoor environments that are likely to be populated by terrains with different properties.

5.2 Actively improving navigation on different terrains

The main contribution of this work is a novel approach to improve robot navigation on different terrains over time from the experiences of the robot. We propose to use a Gaussian process mixture model for learning a probabilistic model of the intensity of phenomena such as vibrations or energy consumption. We furthermore leverage an aerial image of the environment for modeling predictions on different terrains, without requiring training data or an explicit terrain classifier. We exploit this GP model to plan paths that aim at reducing the detrimental factors that affect navigation while taking into account the model's uncertainty

in order to achieve an exploration-exploitation trade-off.

In sum, our approach is able to (i) learn an accurate place-dependent probabilistic model of a specific phenomenon that affects navigation on different terrains from the onboard observations of the robot during traversal; (ii) use such a model to plan paths that reduce over time the impact of this phenomenon; (iii) exploit an image prior to improve the model accuracy while requiring a smaller number of observations.

5.2.1 Modeling different terrains

We propose to model the intensity of a particular phenomenon that the robot experiences during navigation on different terrains using a mixture of Gaussian processes [164]. Furthermore, we use an aerial image of the environment such as the satellite images available in Google Earth[1] as a prior to quickly learn an accurate model that accounts for both spatial proximity and appearance similarity, without requiring the robot to visit every location in the environment.

5.2.1.1 GP mixture model

A GP mixture model, first introduced by Tresp [164], is a sum of m Gaussian processes, $\mathrm{GP}_1, \ldots, \mathrm{GP}_m$, weighted according to a so-called gating function $\boldsymbol{\phi} = \{\phi_1, \ldots, \phi_m\}$. The gating function assigns to each data point the probability that it is associated with each component of the mixture. Given the query input $\mathbf{x}_*$, the gating function $\phi_i(\mathbf{x}_*)$ represents the probability that $\mathbf{x}_*$ belongs to the model specified by GP_i. Therefore, the predictive distribution at $\mathbf{x}_*$ is given by $\mathcal{N}(\mu_{\mathrm{mix}*}, \sigma^2_{\mathrm{mix}*})$ such that:

$$\mu_{\mathrm{mix}*} = \sum_{i=1}^{m} \phi_i(\mathbf{x}_*)\, \mu_{i*}, \tag{5.3}$$

$$\sigma^2_{\mathrm{mix}*} = \sum_{i=1}^{m} \phi_i(\mathbf{x}_*) \left(\sigma^2_{i\,*} + (\mu_{i*} - \mu_{\mathrm{mix}*})^2\right), \tag{5.4}$$

where $\mathcal{N}(\mu_{i*}, \sigma^2_{i\,*})$ indicates the predictive distribution of the GP_i component at $\mathbf{x}_*$. We compute the predictive mean μ_{i*} and variance $\sigma^2_{i\,*}$ for each component using GP regression. We incorporate the probability that each data point belongs to the i-th component of the mixture by defining the term $\mathbf{K}_{\mathrm{XX}}$ in Equation (5.1) for the i-th component as:

$$\mathbf{K}^i_{\mathrm{XX}} = \mathbf{K}(\mathbf{X}, \mathbf{X}) + \boldsymbol{\Psi}^i, \tag{5.5}$$

[1] http://earth.google.com

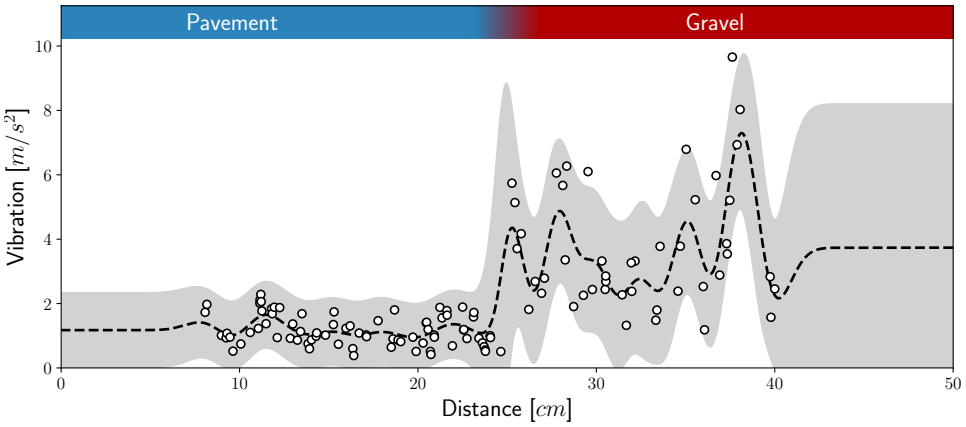


Figure 5.3: GP mixture model of the vibration intensities on two different terrains. It uses two GP models and a deterministic gating function defined according to the terrain on which the robot navigates.

where $\mathbf{\Psi}^i$ is a diagonal matrix such that for all training inputs $\mathbf{x}_j \in \mathbf{X}$:

$$\mathbf{\Psi}^i_{jj} = \frac{\varsigma_n^2}{\phi_i(\mathbf{x}_j)}. \tag{5.6}$$

Using a mixture of GPs allows for computing a more accurate model of the vibration amplitudes on different terrains than using a single-GP model. Consider the example introduced in Section 5.1.1 where the robot experiences vibrations presenting different characteristics. Given the observed vibration intensities, we can learn a GP mixture model by considering the two GP models trained independently on the two different terrains, as illustrated in Figure 5.2(c). We define a deterministic gating function according to the ground on which the robot navigates and compute a mixture of the two GPs using Equation (5.3).

The resulting predictive model is illustrated in Figure 5.3. As the GP mixture model can learn multiple characteristics, this model is able to capture the trends of the vibration intensities on the different terrains without smoothing out the peaks. Such a model allows also for providing better predictions in the regions where the robot has not collected observations: the predicted mean intensity of vibrations is lower on pavement than on gravel. Therefore, using a mixture of GPs improves the accuracy and the expressiveness of the model.

Using a GP mixture model brings advantages also in the computation complexity. Learning a single GP presents a complexity of $\mathcal{O}(N^3)$, where N is the number of observations, as it requires to invert the matrix $\mathbf{K}_{\mathrm{XX}}$. A GP mixture model instead presents only cubic computation in the number of data associated with each component Q, with $Q \leq N$.

5.2.1.2 Estimating the gating function from observations

In this chapter, we assume that the robot has initially no information about the terrain on which it navigates. This is in practice the case for several robotic applications. Therefore, in general, no gating function for the mixture model is available a priori. One approach to compute the gating function is to train a classifier that determines the type of terrain on which the robot navigates as, for example, Murphy *et al.* [112]. However, this approach would require to define a priori a set of terrain types and training data for each of the selected types but also a descriptive input representation of the environment to classify unvisited locations. Another limitation of such an approach is that defining a gating function based purely on the terrain type does not allow for modeling regions with the same terrain type but showing different conditions differently. For example, in the same environment, the surface of one lawn could be hard because it is exposed to the sun, while another one being muddy because it is in the shadow most of the time. A classifier would assign to these regions the same labels but, in practice, they will have a very different impact on the navigation of the robot.

Instead of learning an explicit terrain classifier, we propose to compute the gating function directly from the robot's observations during navigation. We achieve this by using the procedure illustrated in Figure 5.4. First, given the observations $\mathbf{z}_{\text{visited}}$ collected by the robot at the visited locations $\mathbf{X}_{\text{visited}}$ while navigating in the environment $\mathcal{X}$, we group the observations into clusters. We perform clustering by using the mean-shift clustering algorithm [48]. The advantage of using the mean-shift algorithm is that it neither requires prior knowledge of the number of clusters nor imposes constraints on their shape. It computes m cluster centroids by iteratively shifting the mean in the direction that maximizes the density of each cluster.

Given the cluster centroids $\{c_1, \ldots c_m\}$, we define a gating function $\boldsymbol{\phi}$ over the robot observations by computing the probability that a data point $\{\mathbf{x}, z\}$, with $\mathbf{x} \in \mathbf{X}_{\text{visited}}$ and $z \in \mathbf{z}_{\text{visited}}$, belongs to the cluster c_i as:

$$\phi_i(\mathbf{x}) \;=\; \frac{\exp\left(\,\|z \,-\, c_i\|\right)}{\sum_{j=1}^{m} \exp(\|z \,-\, c_j\|)}. \tag{5.7}$$

We use the probability of each observed data point to belong to each of the clusters to compute the gating function over the whole environment. To achieve this, we train a classifier to predict the gating $\boldsymbol{\phi}$ for the non-visited locations. We use m binary classifiers in which the inputs are the visited locations $\mathbf{X}_{\text{visited}}$ and the targets for the i-th component are the probabilities $\phi_i(\mathbf{X}_{\text{visited}})$. For classification, we adopt simple binary discriminative GP classifiers but, in general, the use of other classifiers is also possible. GP classifiers perform classification

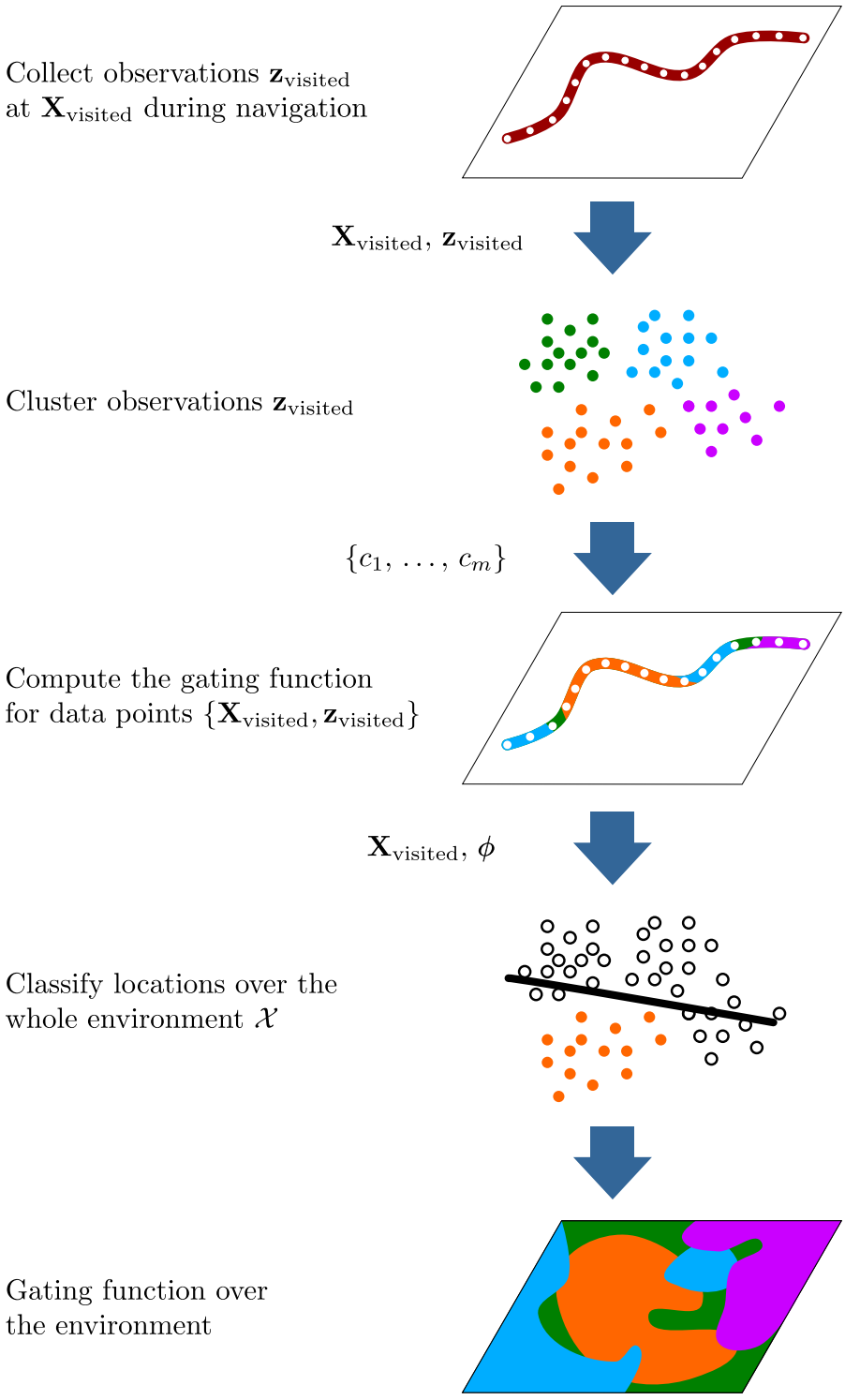


Figure 5.4: Overview of our procedure to compute the gating function of the GP mixture model for modeling different terrains.

(a) Google Earth satellite image. (b) Aerial image taken from an UAV.

Figure 5.5: Examples of aerial images of the environment that we incorporate as prior to speed up the learning process of the gating function.

by 'squashing' the output of GP regression into a class probability using the logistic function. Normalizing the output of these classifiers, we compute for each location a probability distribution over the environment $\mathcal{X}$ to belong to each of the clusters. We use the resulting probabilities as the gating function for our GP mixture model.

5.2.1.3 Incorporating aerial image in gating function

The procedure described in the previous section computes a gating function over the whole environment from the robot's observations by exploiting spatial information deriving from the location coordinates where the observations were collected. However, using only spatial information may require the robot to visit the whole environment for learning an accurate gating function. Even little additional information can help to speed up the learning process of the gating function.

One possible prior information is that terrains with similar visual appearances are likely to affect robot navigation similarly. We propose to incorporate this prior in the gating function by using an aerial image of the environment, for example, a satellite image available from Google Earth data or an aerial image captured by a unmanned aerial vehicle (UAV), as the ones illustrated in Figure 5.5. We assume to know the correspondence between the locations of the environment and the pixels in the aerial image. This is possible, for example, by using geo-referenced images. We incorporate this prior information for learning the gating function by considering as input to the classifiers introduced in the previous section both the coordinates of the visited locations $\mathbf{X}_{\text{visited}}$ and a set of features associated to the corresponding pixels in the aerial image $\mathcal{I}$. In this work, we simply consider the

RGB color space as a feature vector but, in general, we could consider any feature or color space that can be computed for each location in the image. Therefore, we consider as input to the m GP classifiers, which determine the gating function, a vector of features for each data point defined as:

$$\mathbf{f}_{\mathcal{X},\mathcal{I}}(\mathbf{x}) = [\mathbf{x}, \mathcal{I}(\mathbf{x})] \tag{5.8}$$

$$= [x, y, \mathbf{r}(\mathbf{x}), \mathbf{g}(\mathbf{x}), \mathbf{b}(\mathbf{x})], \tag{5.9}$$

where (x, y) are the coordinates of the location $\mathbf{x}$ in the environment, and $\mathbf{r}$, $\mathbf{g}$, $\mathbf{b}$ are the color channels of the image. The resulting vector of features consists of different types of quantities. Therefore, we employ an automatic-relevance-determination squared-exponential covariance function [121]. Automatic relevance determination kernels allow for considering different kinds of features by implicitly determining the importance of each dimension and by multiplying together kernels defined on each individual input in the training process.

Incorporating the aerial image for computing the gating function allows us to make predictions and, thus, to learn a model based not only on the spatial proximity but that takes into account also the appearance of each location. This improves the information value content of the model, especially in regions that have not been visited yet, thus reducing the need to explore the whole environment to obtain an accurate model.

5.2.2 Planning to improve navigation

For actively improving robot navigation, we aim at exploiting the model introduced in Section 5.2.1 for planning paths that reduce the impact of a specific detrimental phenomenon on navigation over time. To this end, we consider a function that maps the intensity of this phenomenon to costs and plan paths that aim at minimizing these costs. We define this correspondence using an identity function but, in general, any other function could be considered, also as a combination of multiple factors.

When the robot starts to navigate in a new environment, no information about the factors affecting the navigation is available and, thus, our predictive model presents large uncertainty everywhere. Therefore, in the initial runs, the robot should explore the regions with large uncertainty and collect informative observations for improving the model. The larger the number of observations, the more accurate will be the model and, thus, the more the robot can navigate through areas where the impact of detrimental factors is likely to be low. We achieve this behavior by designing a tailored cost function and by computing paths that minimize it.

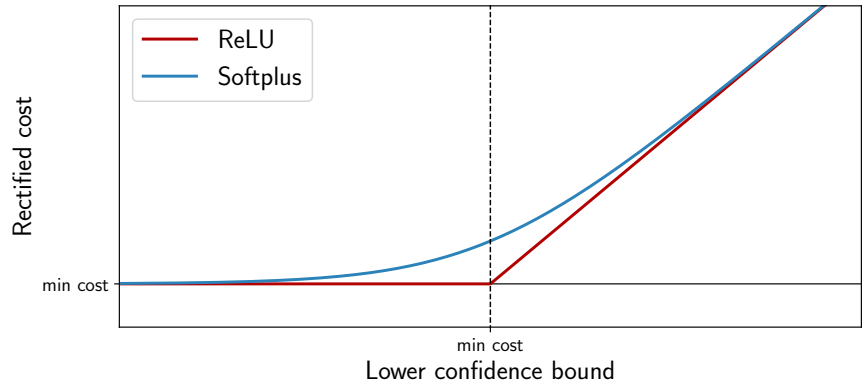


Figure 5.6: ReLU vs. softplus function to rectify the lower confidence bound of the predictive distribution.

5.2.2.1 Trading off exploration and exploitation

We formulate path planning as an explicit cost minimization problem and solve it by using a generic graph search algorithm. To this end, we consider a grid map representation of the environment $\mathcal{X}$ and a corresponding discretization of our model. In this representation, we compute a path to the goal by selecting at each iteration of the search the node that minimizes the cost of the path from the start to the node itself. This can be achieved, for example, by using Dijkstra's or A* algorithms. See Section 2.3.1 for further details about graph search algorithms for planning.

We exploit the probabilistic nature of our predictive model to design a gcost function that trades off the exploration of promising non-visited regions and the exploitation of known low-cost areas. We define the gcost at location $\mathbf{x}$ based on the lower confidence bound of the predictive distribution provided by our model at $\mathbf{x}$, $\mathcal{N}(\mu, \sigma^2)$:

$$gcost(\mathbf{x}) \quad = \quad gcost(\mathbf{x}') \, + \, \|\mathbf{x} - \mathbf{x}'\| \, \cdot \, \text{softplus}\,(\mu \, - \, \lambda\,\sigma)\,, \tag{5.10}$$

where $\mathbf{x}'$ is the previous location from which we expand the search, $\lambda > 0$ defines the range of the confidence bound, and the softplus function [37] is a rectifier.

Planning a path according to Equation (5.10) minimizes at the same time the lower confidence bound of the predictions, $(\mu \, - \, \lambda\,\sigma)$, and the travel distance with the term $\|\mathbf{x} - \mathbf{x}'\|$. The lower confidence bound provides a natural trade-off between exploration and exploitation [31]. The term μ is small at locations with low-mean prediction, thus favoring low-cost paths. Whereas, the term σ represents the possible improvement of the prediction and favors paths through locations with large uncertainty that could lead to the goal through a lower-cost path.

The lower confidence bound may, in general, assume negative values. In this case, minimizing Equation (5.10) may result in paths that lead the robot

Algorithm 8 Actively improving navigation on different terrains

procedure ActiveNavigationOnDifferentTerrains ($\mathcal{X}$, $\mathcal{I}$)
  $\boldsymbol{\mu}_{\mathcal{X}}$, $\boldsymbol{\sigma}^2_{\mathcal{X}} \leftarrow$ InitializeModel()
  $\mathbf{X}_{\text{visited}}$, $\mathbf{z}_{\text{visited}} \leftarrow \emptyset$
  for each $\{\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}\}$ **do**
    $\mathbf{X}_{\text{path}} \leftarrow$ PlanPath($\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{goal}}$, $\mathcal{X}$, $\boldsymbol{\mu}_{\mathcal{X}}$, $\boldsymbol{\sigma}^2_{\mathcal{X}}$)
    $\mathbf{z}_{\text{path}} \leftarrow$ FollowPath($\mathbf{X}_{\text{path}}$)
    $\mathbf{X}_{\text{visited}}$, $\mathbf{z}_{\text{visited}} \leftarrow \mathbf{X}_{\text{visited}}$, $\mathbf{z}_{\text{visited}} \cup \mathbf{X}_{\text{path}}$, $\mathbf{z}_{\text{path}}$
    $c_{1:m} \leftarrow$ ClusterDataPoints($\mathbf{X}_{\text{visited}}$, $\mathbf{z}_{\text{visited}}$)
    for each c_i **do**
      $\phi_i(\mathbf{X}_{\text{visited}}) \leftarrow$ ComputeGatingDataPoints($\mathbf{z}_{\text{visited}}$, c_i)
      $\phi_i(\mathcal{X}) \leftarrow$ ComputeGatingFunction($\mathbf{X}_{\text{visited}}$, $\phi_i(\mathbf{X}_{\text{visited}})$, $\mathcal{X}$, $\mathcal{I}$)
      $\boldsymbol{\mu}_i, \boldsymbol{\sigma}^2_i \leftarrow$ TrainGP($\mathbf{X}_{\text{visited}}$, $\mathbf{z}_{\text{visited}}$, $\phi_i(\mathcal{X})$)
    $\boldsymbol{\mu}_{\mathcal{X}}, \boldsymbol{\sigma}^2_{\mathcal{X}} \leftarrow$ ComputeGPMixtureModel($\boldsymbol{\mu}_{1:m}$, $\boldsymbol{\sigma}^2_{1:m}$, $\boldsymbol{\phi}$)

to explore the whole environment rather than to reach the goal as required. A simple idea to overcome this issue is to use a rectified linear unit (ReLU) as a rectifier. It ensures that the costs are always positive:

$$\text{ReLU}(\mathbf{x}) \quad = \quad \max(\mathit{min\_cost}, \mathbf{x} - \mathit{min\_cost}), \tag{5.11}$$

where *min_cost* is the minimum allowed cost

However, using ReLU may cause that the well-known low-cost locations have the same gcost as locations with very large uncertainty. Instead, we prefer that the robot navigates through regions that are known to be low-cost over the very uncertain ones. Therefore, we employ as rectifier the softplus function that is a smooth approximation of ReLU defined as:

$$\text{softplus}(\mathbf{x}) \quad = \quad \log(1 + \exp(\mathbf{x} - \mathit{min\_cost})) + \mathit{min\_cost}. \tag{5.12}$$

Using the softplus rectifier determines that the robot prefers low-mean and low-uncertainty areas over the medium-mean high-uncertainty ones, and, in turn, prefers these areas over the medium-mean low-uncertainty ones. The softplus and the ReLU functions are illustrated in Figure 5.6.

5.2.2.2 Actively improving navigation

The main procedure of our approach for actively improving robot navigation by exploring and modeling the different terrains which characterize the environment is illustrated in Figure 5.7 and described in Algorithm 8. Our approach requires as input the grid map $\mathcal{X}$ and an aerial image $\mathcal{I}$ of the environment. We initialize

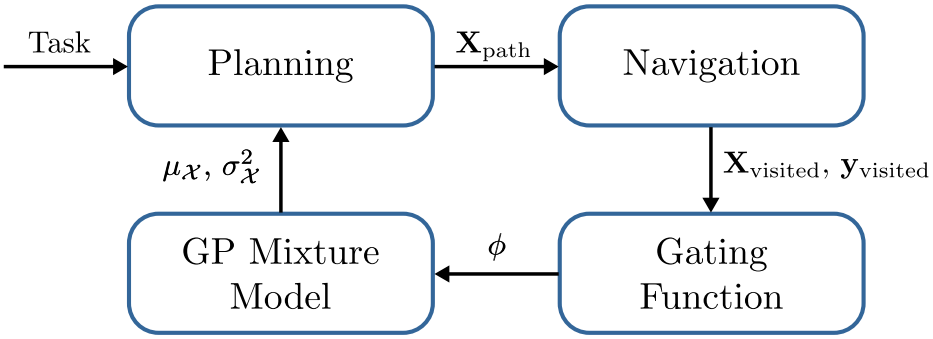


Figure 5.7: Workflow for actively improving robot navigation by exploring and modeling the different terrains in the environment. Given a new task, our planner computes a path using the current model. During navigation, the robot collects observations and updates the gating function and the model accordingly for the next tasks.

our probabilistic model uniformly with the same mean value and large uncertainty over the whole environment (line 2).

For each navigation task that the robot is required to perform, we consider the current model of the environment, denoted as $\boldsymbol{\mu}_{\mathcal{X}}$, $\boldsymbol{\sigma}^2_{\mathcal{X}}$, and plan a path using the approach described in Section 5.2.2.1 (line 5). The robot navigates by following the planned path to the goal (line 6) and collecting the observations $\mathbf{z}_{\text{path}}$ at the visited locations $\mathbf{X}_{\text{path}}$ during traversal. We update the set of observed data points, $\{\mathbf{X}_{\text{visited}}, \mathbf{z}_{\text{visited}}\}$, with the observations collected during the last navigation task (line 7). Given all robot's observations collected so far, $\mathbf{z}_{\text{visited}}$, we run the procedure described in Section 5.2.1.2. Thus, we cluster the data points using the mean-shift algorithm (line 8). For each cluster centroid c_i, we compute the probability that an observation is associated to it using Equation (5.7) (line 10). Using these probabilities, we compute the gating for the i-th component over the whole environment $\mathcal{X}$ (line 11), as described in Section 5.2.1.3. Given the gating function $\phi_i(\mathcal{X})$, we learn the corresponding model using GP regression and Equation (5.5) (line 12). Once we computed all of the GP models corresponding to the clusters, we combine the models of each component using Equation (5.3) (line 13) to obtain an updated GP mixture model that includes the robot's observations during the last navigation task.

Figure 5.7 illustrates the workflow of our approach. It highlights how we continuously refine the model of a specific quantity over the environment and exploit the updated model for improving the paths along which the robot navigates over time.

5.2.2.3 Navigation on different terrains over time

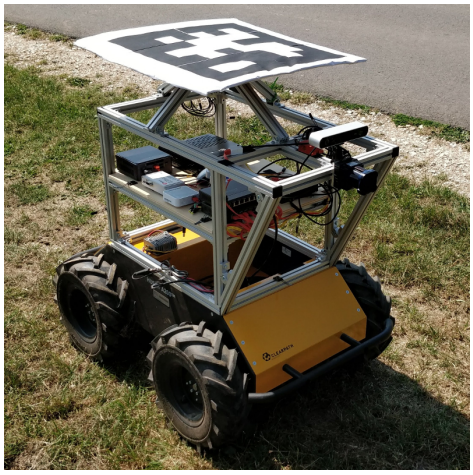
The procedure illustrated in the previous section leads to a navigation behavior that, initially, when no information about the environment is available and the predictive uncertainty of the model is large, aims at exploring the environment. Initially, in the cost function for path planning in Equation (5.10), the uncertainty term σ is more prominent than the mean predicted cost μ, which provides little information. Thus, our planning algorithm generates a path that allows the robot to collect observations at informative locations. At the same time, it minimizes the travel distance to the goal so to avoid that the robot goes for long exploratory detours. As the robot collects more and more observations, the uncertainty σ decreases and the mean predicted cost μ becomes more significant. At this point, our planning algorithm generates paths that guide the robot to prefer low-cost short-distance paths over collecting new observations at unknown less-promising locations. Following this strategy, after a number of navigation tasks, the robot will navigate along paths where the impact of the modeled detrimental factor is low, thus improving robot navigation.

5.3 Experimental evaluation

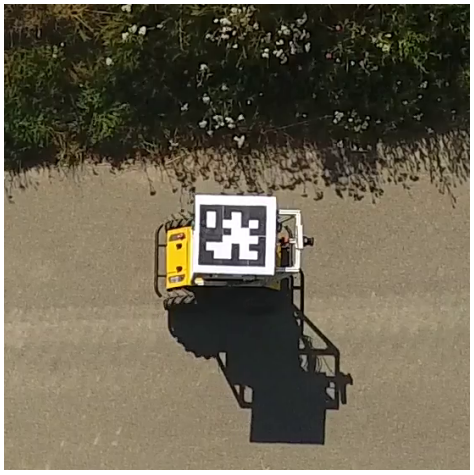
The aim of this experimental evaluation is to analyze our approach for actively improving robot navigation by learning a model of the phenomena affecting navigation from robot's observations and planning paths that minimize their impact over time. Our experiments are designed to show the capabilities of our approach, which are: (i) planning paths that reduce over time the impact of detrimental factors during navigation, (ii) learning an accurate place-dependent model for navigation on different terrains, (iii) improving the predictions of the modeled quantity by using an aerial image of the environment. We implemented our approach and thoroughly tested it using real-world data.

5.3.1 Experimental setup

The experiments we present in this paper use real-world data recorded by our Clearpath Husky mobile robot illustrated in Figure 5.8(a). This robot is designed for outdoor navigation and can drive even on rough terrains. We consider as input to our navigation system the occupancy grid map and an aerial image of the environment that we captured using a UAV. We establish a correspondence between the locations in the grid map and the aerial image to learn a model of the environment that takes into account both the spatial proximity of the observations and the appearance of the locations. We achieve this by placing an AprilTag marker [125] of top of the robot and by continuously tracking its position



(a) Frontal view.



(b) View from the UAV.

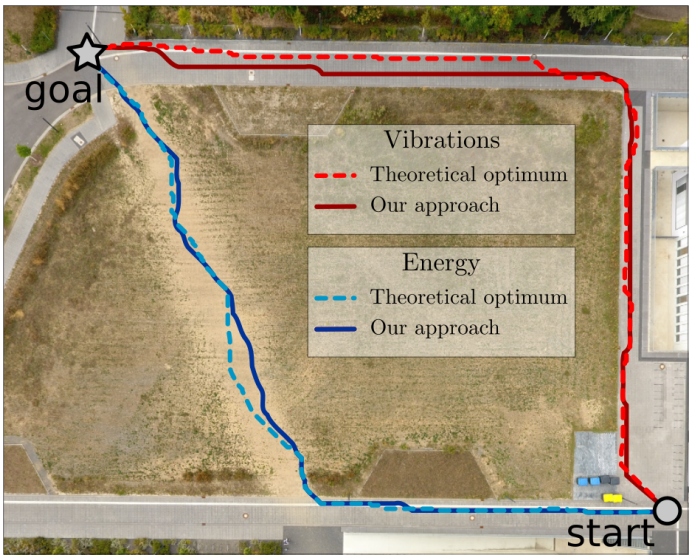
Figure 5.8: Our Clearpath Husky robotic platform that we used to conduct the experiments in real outdoor environments.

in the camera image of the UAV which was flying over the environment during the experiments. See an illustration of the experimental setup in Figure 5.8(b).

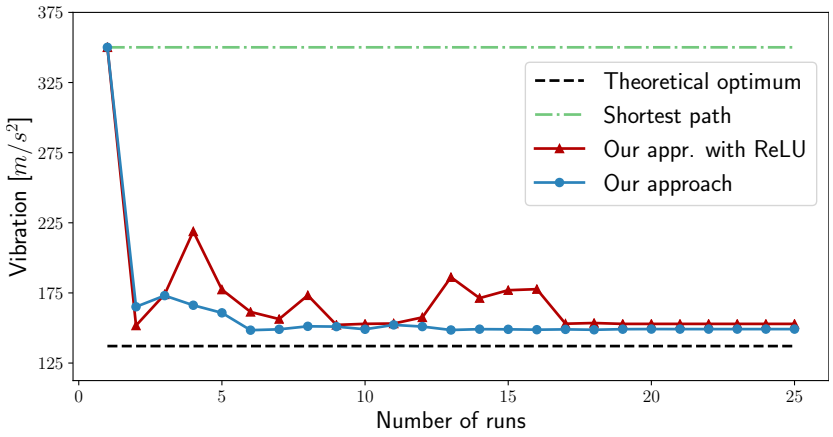
In our evaluation, we consider the vibrations intensity or energy consumption as the factors that we aim at reducing to improve robot navigation. We estimate the intensity of the onboard vibrations by using an XSens MTi-30 IMU installed on the front side of the robot and measuring the acceleration along the z-axis, similarly to Wurm *et al.* [174]. Whereas, we measure the amount of energy consumption during navigation from the data provided by the motor drivers. For evaluation, we drove the robot with a constant cruise velocity through three different environments consisting of different terrains including lawn, gravel, dirt, concrete, pavement, etc. We collected the robot's onboard observations during traversal and built the ground truth models for vibrations and energy consumption. We built the ground truth models by using an average filter to assign an intensity value to each location in the model that corresponds to a cell of the grid map of the environment. In our experiments, the ground truth models are unknown to the robot and no information about vibrations and energy consumption is available a priori. The robot can, however, collect noisy observations in the locations that it visits during navigation.

5.3.2 Improving navigation over time

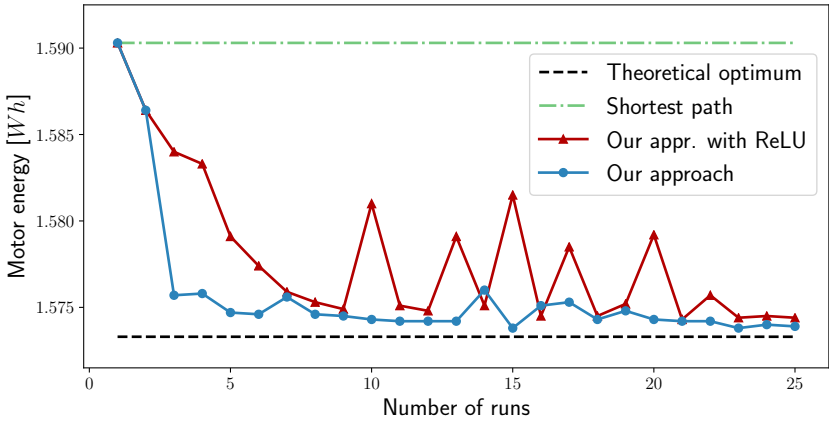
We designed the first experiment to illustrate that our approach is able to improve robot navigation by reducing the detrimental factors that the robot experience during traversal over time. To this end, we consider two scenarios in the real-world environment shown in Figure 5.9(a). In the first scenario, we aim at minimizing



(a) Theoretical optimal paths vs. our approach.



(b) Experienced vibrations intensity.



(c) Experienced energy consumption.

Figure 5.9: Results of our approach improving robot navigation over time. (a) The theoretical optimal paths (dotted lines) and the paths computed by our approach after 25 runs (solid lines) minimizing vibrations (red) and energy consumption (blue). The vibrations intensity (b) and the energy consumption (c) experienced by the robot during navigation in the 25 runs following the different planning approaches.

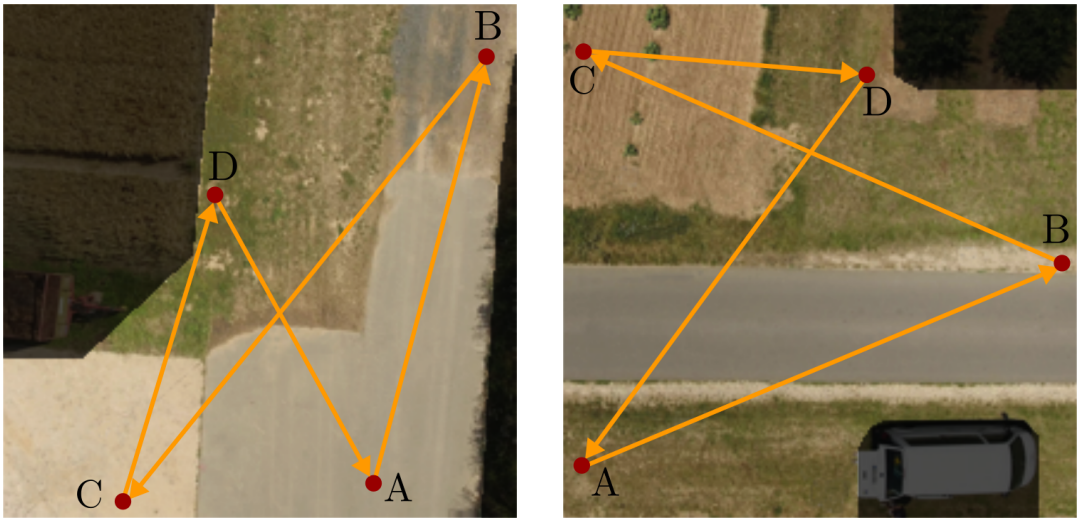
the intensity of the robot's vibrations, whereas, in the second one, the amount of energy it consumes.

For each scenario, we required the robot to navigate from a start to a goal location 25 times. For comparison, we consider a shortest path planner that computes paths by using the A* search algorithm and minimizes the travel distance while ignoring the vibrations and energy consumption. We also compare our planning approach that employs a softplus rectifier in the gcost function with our approach but employing ReLU as a rectifier, as discussed in Section 5.2.2.1. Furthermore, we compare these approaches with the theoretical optimal path computed using the ground truth values for the vibrations intensity and energy consumption in the environment that are unknown in practice to the robot.

The vibrations intensity and the energy consumption that the robot experiences during traversal over time by following each of the planning approaches are illustrated in Figure 5.9(b) and Figure 5.9(c) respectively. These plots reveal that the shortest path planner, which ignores the robot's observations during traversal, computes at every run the same path without showing any improvement over time. Since no information about vibrations and energy consumption is initially available, our approach plans at first paths analogously to the shortest path. In the initial runs, our approach leads the robot to collect observations at different regions of the environment and, after 3-4 runs, it already computes paths along which the robot suffers a dramatically reduced intensity of vibrations and energy consumption. After 25 runs, our approach converges and it is able to drive the robot along paths in which it experiences similar vibrations intensity or energy consumption as if it would follow the theoretical optimal path. Instead, following our approach but using the ReLU rectifier, the robot keeps exploring new regions of the environment rather than exploiting the known low-cost areas. Therefore, our approach takes longer using the ReLU rectifier than using the softplus rectifier to converge to a path along which the robot experiences low-intensity vibrations or energy consumption.

The paths computed by our approach after 25 runs are illustrated in Figure 5.9(a). The path that minimizes the intensity of the onboard vibrations follows on the paved road (solid dark red). Whereas, the path that minimizes the energy consumption follows on a paved road and on dirt (solid dark blue). Figure 5.9(a) shows also that our approach leads the robot to navigate along similar trajectories and on similar terrains as following the theoretical optimal paths (dotted curves).

This experiment showcases the ability of our approach to improving robot navigation by reducing over time the impact of the detrimental factors that the robot experiences during traversal and by leading it to navigate along trajectories similar to the theoretical optimal paths.



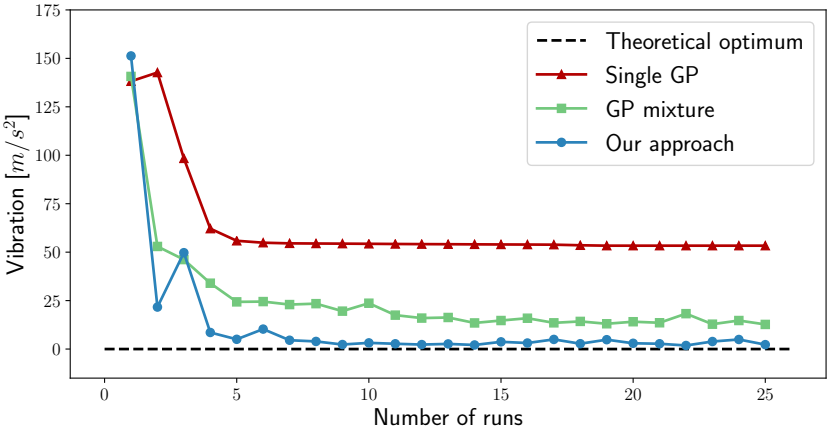
(a) Site 1. (b) Site 2.

Figure 5.10: The environments and the navigation tasks considered for the experiments on learning an accurate model of the environment described in Section 5.3.3.

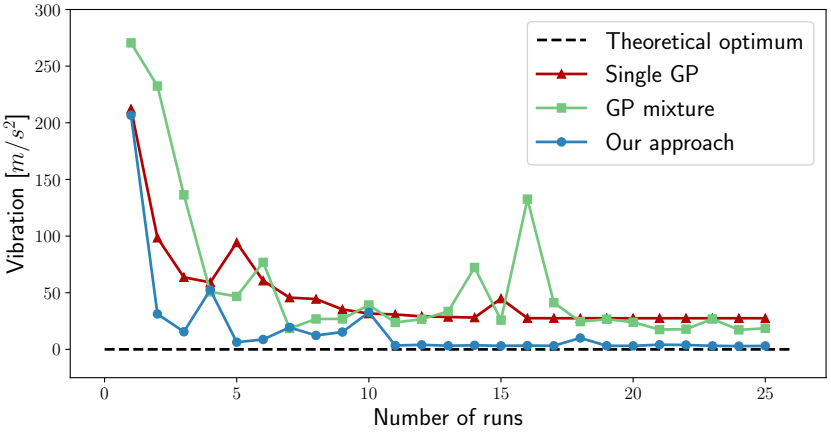
5.3.3 Learning an accurate model of the environment

The second experiment aims at showing that our approach is able to learn a model that provides accurate predictions about the intensity of the phenomena affecting robot navigation on different terrains. To illustrate this, we consider the two environments depicted in Figure 5.10 where we require the robot to navigate through the locations A, B, C, D as shown in the figure. We aim at using our approach to perform these navigation tasks while reducing over time the intensity of the vibrations that the robot experiences during traversal. We also compare our approach, which incorporates an aerial image in a GP mixture model, with two methods that use the same planning algorithm but different approaches to model the vibration intensities in the environment. The first one uses a single GP similar to the approach described in Section 5.1.1, which we denote as Single GP. The second one models the environment by using a mixture of GPs but it does not consider the aerial image of the environment. We refer to this approach as GP mixture. Also in this experiment, we evaluate our approach with respect to the theoretical optimal path computed using the ground truth data of the vibration intensities in the environment.

The vibration intensities experienced over time by the robot in the two environments following the different approaches are illustrated in Figure 5.11. Our approach is able to quickly learn accurate models of the environments by combining spatial proximity and visual appearance information. Such models lead the robot to experience significantly lower vibrations already after few runs in both environments. After 25 runs, following our approach, the robot is able to follow



(a) Site 1.

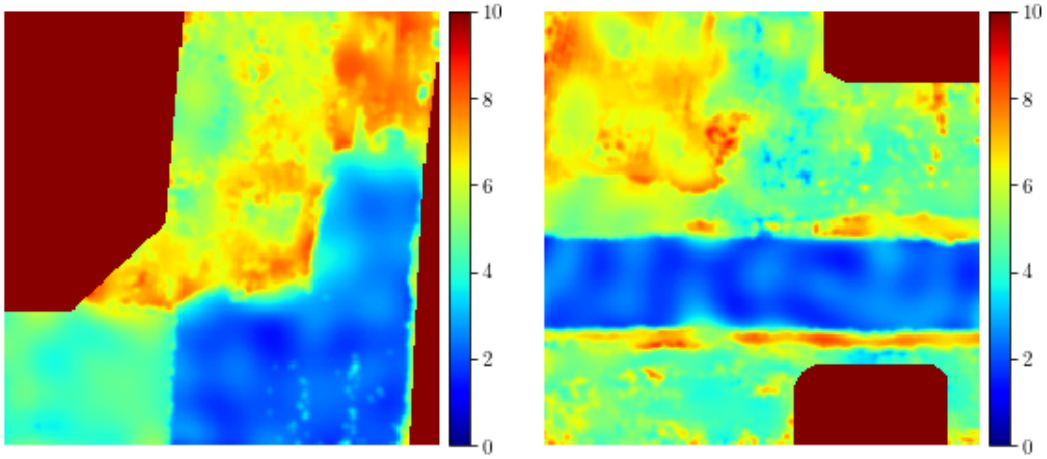


(b) Site 2.

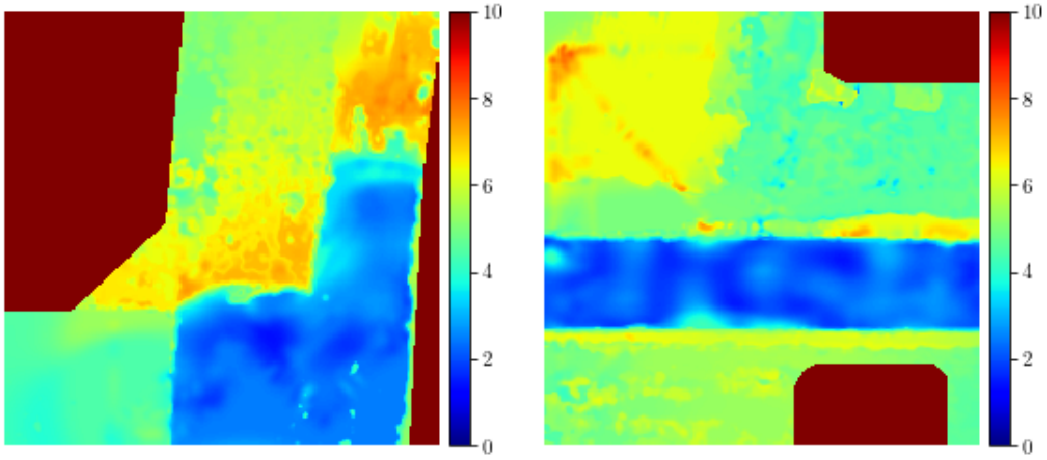
Figure 5.11: Difference of the vibrations intensity experienced by the robot during navigation in the sites illustrated in Figure 5.10 following different approaches to model the environment from the theoretical optimum.

a path along which it experiences similar vibrations as following the theoretical optimal path. In comparison, using a single GP model, the robot stops the exploration phase earlier. The robot converges to navigate along a path that causes lower vibration intensities than in the initial run when no information about the environment is available but still significantly more than by following our approach. Using a GP mixture model, the robot shows better performance than using a single GP in the long run but it takes longer than our approach to finding a low-cost path. The reason for this is that, without taking into account the aerial image of the environment, it relies only on spatial proximity information to make predictions about the intensity of the vibrations at non-visited locations.

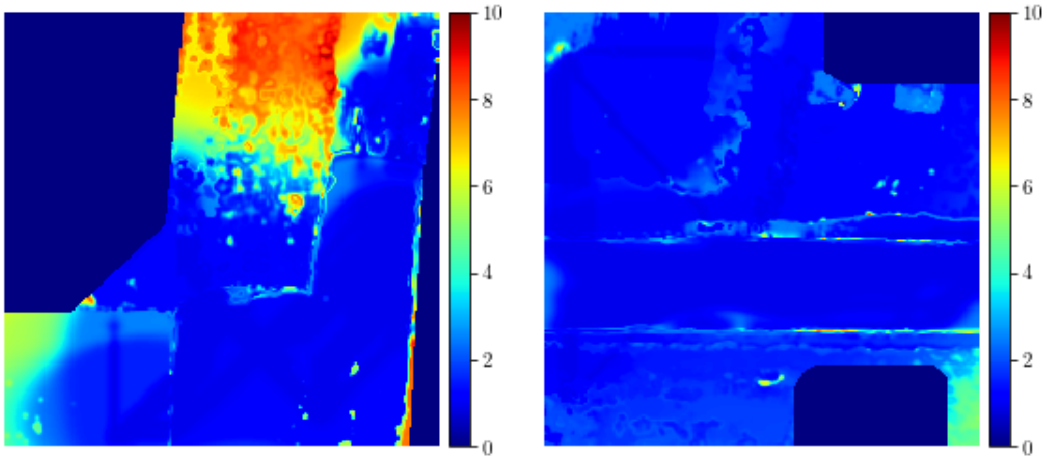
The models of vibration intensities in the two environments learned by our approach after 25 runs are illustrated in Figure 5.12. We compare these models with the ground truth values depicted in Figure 5.12(a) and built as described



(a) True vibration intensities.



(b) Predictive mean of the vibration intensities.



(c) Predictive variance of the vibration intensities.

Figure 5.12: The true vibration intensity models, and the mean and the variance predictions provided by our approach from robot's observations after 25 runs in the two scenarios illustrated in Figure 5.10.

	RMS error	Mahal. dist.	RMS error	Mahal. dist.
Single GP	16.15	265.84	15.364	316.28
GP mixture	9.29	132.52	9.43	**150.24**
Our approach	**4.96**	**79.17**	**6.91**	151.42
	Site 1		Site 2	

Table 5.1: Accuracy of the estimated vibration intensity models resulting from the experiment in Section 5.3.3 and illustrated in Figure 5.12(b) and Figure 5.12(c) with respect to the true models illustrated in Figure 5.12(a).

in Section 5.3.1, which are in practice not known to the robot. The predictive mean of our model in Figure 5.12(b) reveals similar values to the ground truth over most of the environment. In particular, we obtain very accurate models at regions with low-intensity vibrations as the robot prefers those regions for navigation and, therefore, traverses them multiple times. The variance of the predictions is illustrated in Figure 5.12(c) and is mainly low. This indicates that our approach can make reliable predictions over the whole environment. Our model presents a higher variance at borders between different terrains. This can be explained by the fact that, in these regions, the robot observes very different vibrations in adjacent locations.

In Table 5.1, we provide a comparison of the accuracy of the models of the vibration intensity obtained using the different approaches after 25 runs. We compare the root-mean-square (RMS) error of the mean prediction and the Mahalanobis distance between the predictive distributions and the ground truth intensity computed as:

$$d_{\text{Mahal}} = \sqrt{\sum_{i}^{N} \frac{(z_i - \mu_i)}{\sigma_i^2}}, \tag{5.13}$$

where z_i is the ground truth value and $\mathcal{N}(\mu_i, \sigma_i)$ is the corresponding predictive distribution of the model.

In Site 1, our approach presents the lowest root-mean-square error and Mahalanobis distance which is more than three times lower than by using a single GP model. In Site 2, our approach presents a similar Mahalanobis distance as the GP mixture model, but a significantly lower error.

This experiment illustrates the capability of our approach to learn an accurate representation of the detrimental factors that characterize the environment from the robot's observations during navigation and to make effective predictions at non-visited locations.

5.3.3.1 Using aerial image for improving predictions

Our approach is able to quickly learn a model of a specific physical quantity experienced by the robot during navigation thanks to the use of an unlabeled, easy to obtain, aerial image of the environment. Using an aerial image of the environment as a prior allows for learning a more accurate model while visiting a smaller number of locations than not using it. Table 5.1 illustrates that our approach that uses an aerial image as a prior shows better or similar performance than the GP mixture model. However, our approach obtains this performance by visiting around 20% less of the locations. Using an aerial image of the environment helps to identify the promising regions of the environment for navigation, and so to find a low-cost path. For example, a few observations of low-intensity vibrations on a paved road permit to predict low-intensity vibrations in areas with similar appearances. Another advantage of using an aerial image is that borders between different terrains are typically well defined in images. This makes easier to learn a gating function that is sharp at borders and, thus, more accurate models.

5.4 Related work

In outdoor environments, the terrain may cause undesired effects on robot navigation such as strong vibrations or high energy consumption. Many robot navigation systems such as Obelix [89] navigate outdoors by planning paths using A* and by trying to locally avoid rough terrains. There exist several approaches that achieve this by classifying the terrain on which the robot navigates to determine the traversable areas of the environment. For example, Wolf *et al.* [173] use a laser range finders to classify traversable regions based on the roughness of the terrain. Similarly, Suger *et al.* [160] classify the terrain type using a 3D-LiDAR and determine non-traversable regions accordingly. Whereas, Berczi *et al.* [11] use a stereo camera to learn the terrain assessment from human demonstration. Such approaches require to define a priori the types of terrain to classify and demand training data for each terrain type. Other approaches propose to use self-supervised learning techniques. Wurm *et al.* [174] train a self-supervised classifier to identify and avoid vegetated areas by using an IMU to capture the onboard vibrations caused by the terrain. Similarly, Mou and Kleiner [111] classify terrains for online adapting robot velocities by combining vision and laser readings. Otsu *et al.* [127] combine co- and self-training approaches to classify terrains for planetary exploration rovers using onboard imagery and vibration data. Giguere and Dudek [53] propose an unsupervised approach to identify terrains using the readings from accelerometers, gyroscopes, and encoders. Both supervised and unsupervised approaches identify the terrain types and use this knowledge to make decisions for navigation. This is often achieved by assign-

ing a cost for navigating on each specific terrain, e.g., moving on grass is more costly than on asphalt. However, these costs are artificially defined by the user and, thus, might not adequately represent the physical quantities that the robot experiences while navigating. In contrast to that, the solution presented in this chapter learns a place-dependent model of the detrimental quantities that affect robot navigation directly from the robot's observations during traversal.

One common approach to learn spatial processes from observation is Gaussian processes regression [142]. In the context of robotics, GP regression has been used for modeling many different phenomena and physical quantities in the environment. For example, Lang *et al.* [90] use GPs to model the 3D structure of the terrain from the readings of a 3D-LiDAR, while DGPOM [124, 149] builds a spatio-temporal model of the dynamic areas in the environment from laser scans. Tresp [164] introduces an approach for modeling general conditional probability densities by mixing GPs. GP mixture models allow for learning even more complex phenomena that present multiple different characteristics. Stachniss *et al.* [158] model gas distributions by using a mixture of GPs which allows for accurately representing the smooth background signal and the areas with patches of high concentrations. Similarly, we employ a mixture of GPs to model the physical phenomena affecting robot navigation that present different characteristics on diverse terrains.

GP models allow for easily incorporating additional prior information. For example, Cunningham *et al.* [33] model the slippage for slip prediction of planetary rovers by using GP models and incorporating the geometry of visually classified terrain types. A common source to obtain prior knowledge for environmental modeling are satellite and aerial images that, in the last years, became more and more accessible, for example from Google Earth. Murphy *et al.* [112] estimate the traversal costs in outdoor environments by classifying the terrains from overhead imagery and by using GP models to combine spatial information with the classification probability. Silver *et al.* [150] incorporate similar aerial terrain data with human expert's demonstrations in an imitation learning framework for outdoor navigation. Our approach incorporates information from an aerial image of the environment for learning an accurate GP mixture model without requiring any explicit terrain classifier. We also exploit this information for quickly discovering promising regions of the environment for navigation.

GP models offer an attractive representation for many robotic applications. For example, Fentanes *et al.* [41] use a GP for modeling the soil compaction in fields and exploit this representation to guide the exploration of the environment with an agricultural mobile robot. A popular application for GP models is for environmental monitoring. In this context, Krause *et al.* [78] propose a method to optimize the placement of the sensors in the environment that maximizes the

mutual information on a GP model. Binney *et al.* [17] extend this approach to planning paths for a mobile robot to monitor spatio-temporal phenomena such as measuring wireless signal strength on a lake. Ma *et al.* [102] use a similar approach to select regions where to collect observations for performing persistent ocean monitoring tasks. We use a GP model to plan paths that aim at reducing the impact of undesired effects on robot navigation while exploring unknown promising areas.

To build an accurate model of the physical quantities affecting navigation, we collect observations in different regions of the environment. The problem of selecting informative regions to observe for modeling the environment has been studied in the context of active perception [6] and autonomous exploration [172]. Wang *et al.* [170] investigate an active sensing approach that decides where the robot should direct its data collection in indoor environments and that adjusts the navigation strategy based on the previously collected data. Whereas, Girdhar *et al.* [54] propose to explore the environment by using spatio-temporal topic modeling techniques to learn a model of their environment and to identify interesting, information-rich locations to visit.

Given a GP model of the quantities that affect robot navigation, our approach generates paths that trade off the exploration of unknown promising regions and the exploitation of the known areas where the cost for navigation is low. Viseras *et al.* [168] address the problem of planning paths that trade off exploration-exploitation by using a sampling-based algorithm based on the mean entropy that maximizes information gathering while minimizing the path cost. Marchant and Ramos [105] instead use Bayesian optimization to plan continuous informative paths that deal with the exploration-exploitation trade-off by maximizing the upper confidence bound. A similar planning approach has been proposed by Souza *et al.* [153] for planning paths that collect data about the roughness of the terrain while minimizing the robot's vibrations. The GP-UCB [155] algorithm formalizes a utility function based on the upper confidence bound for GP optimization. Tan *et al.* [161] adopt such a utility function to plan paths that adapt the exploration-exploitation trade-off for bathymetry monitoring. We use a similar concept of confidence bounds to plan paths that trade off exploration and exploitation. However, we guide the robot early towards the goal through low-cost paths without requiring it to explore the whole environment.

5.5 Conclusion

In this chapter, we presented a novel approach to actively improve robot navigation on different terrains. We improve navigation by leading the robot to navigate through paths along which the impact of the detrimental factors affecting navi-

gation, such as vibrations or energy consumption, is reduced over time. To this end, we build a place-dependent probabilistic model of these quantities directly from the robot's onboard observations during traversal without requiring any initial training data or explicit terrain classifier. We learn this model by using a Gaussian mixture model that incorporates an aerial image of the environment. It makes predictions of the specific physical quantities at unvisited locations by using spatial proximity and appearance similarity information. We use this predictive model to plan paths that deal with the exploration-exploitation trade-off, and that quickly lead the robot to navigate through low-cost paths. In contrast to the approaches presented in Part I, the robot actively plans to explore the environment for collecting observations at promising locations while, at the same time, attempting to reach the goal by navigating along low-cost paths.

We implemented and evaluated our approach on different real-world outdoor environments. The experiments suggest that we are able to learn models that provide accurate predictions about the intensity of a quantity affecting navigation in the environment and to exploit these models for planning paths that lead the robot to navigate experiencing lower intensity vibrations or energy consumption over time.

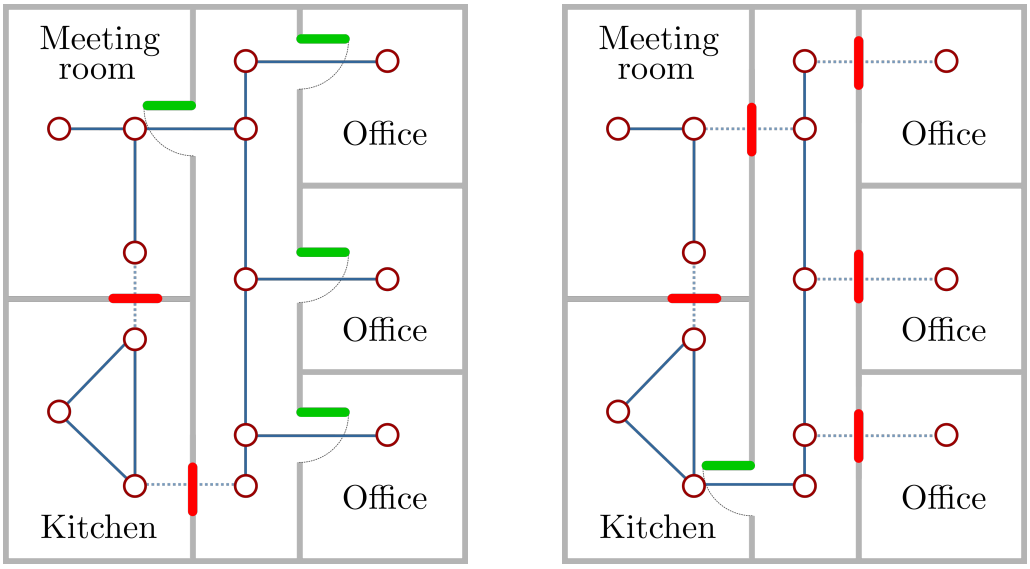
Chapter 6

Navigation estimating patterns in traversability changes

Mobile robots operating in the real world are often employed in dynamic environments that change over time and where other moving agents operate. In the solutions presented in the previous chapters of this thesis, we investigated navigation problems in which the environment is unknown or uncertain, but we assumed it to be mainly static. However, in reality, most environments are subject to continuous changes that affect the connectivity of the configuration space. In these environments, relying on statics maps for path planning can become inadequate to navigate efficiently.

Over the last decade, many mobile robots have been deployed in indoor environments such as offices, hospitals, and shopping malls where they share the workspace with other dynamic agents and/or humans. The presence of other dynamic agents may lead to changes in the environment. For example, while a robot is operating, another agent could close a door or block an aisles potentially affecting the trajectory along which the robot navigates. If the robot encounters an unforeseen obstacle along its path, traditional approaches to robot navigation, for example, Fox *et al.* [45] and many others, perform reactive strategies to avoid the obstacle. Extensions of such mainly static planners predict the short-term motion of obstacles and plan a local deviation for avoiding collisions [116, 88, 156] similarly to the approach described in Chapter 4. If no local deviation can be found, the robot attempts to generate a new path to the goal from scratch. These approaches have been demonstrated to be effective for tackling unexpected and unpredictable obstacles but have no memory about previously experienced situations. Thus, when encountering the same situation multiple times, the robot is not able to exploit previous experiences and may perform every time the same sub-optimal behavior.

In indoor environments, there are many changes in the traversability that



(a) Working state. (b) Coffee break.

Figure 6.1: Traversability change patterns on the topological map of an office. The red circles are the nodes and the blue solid lines are the traversable connections among them.

are correlated or happen following certain patterns. For example, a number of doors in an environment could be frequently open or closed at the same time. In the environment illustrated in Figure 6.1(a), the offices' doors are typically open (green) while people are working. Whereas, if the kitchen is open, it is likely that people are enjoying a coffee and so that the offices' doors are closed (red), see Figure 6.1(b). Similar to that, in a warehouse, two operators could repeatedly work at the same time at shelves containing similar items but located at different locations blocking simultaneously the passage through the corresponding aisles. When deploying a robot in such environments over a longer period of time, it can observe these patterns and exploit this knowledge to avoid encountering obstacles along its paths, thus improving the efficiency of its operations.

In this chapter, we investigate the problem of robot navigation over longer periods of time in dynamic indoor environments where changes occur following certain patterns. We address this problem by learning patterns in the traversability changes and by exploiting this knowledge to realize anticipatory strategies that reduce the risk of encountering blocked passages during navigation. We achieve this by incrementally modeling the patterns of change in the traversability from the robot's observations during traversal. Our traversability model enhances a topological map like the one illustrated in Figure 6.1, where the nodes are the locations of interest and the edges are the traversable passages between these locations. We use such a model to make predictions about the traversability at unknown locations in the environment, and we incorporate the predictions into

a path planning system. Our planning algorithm accounts for the uncertainty of the predictions to plan paths that trade off the risk to encounter an unforeseen obstacle blocking the robot and the information gain of visiting different locations to improve the model. Over time, our approach leads the robot to encounter a reduced number of blocked passages and, thus, to navigate along paths that are shorter than following greedy-reactive strategies on average.

6.1 Navigation with patterns in traversability changes

In this chapter, we consider a robot that navigates over extended periods of time in an indoor environment where the traversability changes according to certain patterns. We aim at modeling these patterns and at exploiting this knowledge to plan strategies for accomplishing safe and efficient robot navigation.

6.1.1 Spatial patterns of change

In an environment, patterns in traversability changes are typically either temporal or spatial. Temporal patterns of change include all those changes that happen periodically in time. For example, in an office environment, the office doors are likely to be closed every night. Temporal patterns of change have been investigated by Fentanes *et al.* [42] that exploit this knowledge to make decisions for robot navigation. The robot navigates along specific routes based on the time of the day to reduce the number of times in which the robot encounters a blocked passage and requires to plan a new path to the goal. In contrast to that, spatial patterns in traversability changes are those changes that happen contemporaneously in different locations of the environment. For example, in our office environment, if the Professor is away and his office door is closed, it is very likely that also Ph.D. students' offices are closed, no matter what is the time. In this chapter, we focus on capturing and modeling spatial patterns of change in the environment without taking into account their periodicity.

6.1.2 Problem definition and assumptions

We assume that, when the robot starts to navigate in an environment, the patterns of change in the traversability are completely unknown. This assumption holds typically true in practice. However, during navigation, the robot can observe which passages are frequently blocked at the same time and incrementally model their traversability patterns. We achieve this by considering a topological map of the environment $\mathcal{G} = (\boldsymbol{V}, \boldsymbol{E})$, where $\boldsymbol{E}$ are the edges representing

possible passages and $\boldsymbol{V}$ are the set of vertices representing locations connected through the passages. An example topology is illustrated in Figure 6.1, where the red circles are the vertices $\boldsymbol{V}$ and the blue lines their connecting edges $\boldsymbol{E}$.

We refer to each navigation task performed by the robot as a run. At every run, the traversability of the environment may change, for example, because of the activity of other dynamic agents operating in the environment. We represent the traversability of an edge $\mathbf{e}_i$ at a run t as the binary random variable e_i^t that is 0 if the edge is blocked or it is 1 if the edge is free. We refer to the state of all the edges of the topology during run t as the environment configuration $\boldsymbol{E}^t = \{\mathrm{e}_1^t, \ldots, \mathrm{e}_{|\boldsymbol{E}|}^t\}$.

In this chapter, we make the following key assumption:

1. when the robot starts a new run, it has no knowledge about the current environment configuration except from its previous observations;

2. *during* each run, the environment configuration does not change. This means that we account only for the low-frequency dynamic changes in the environment;

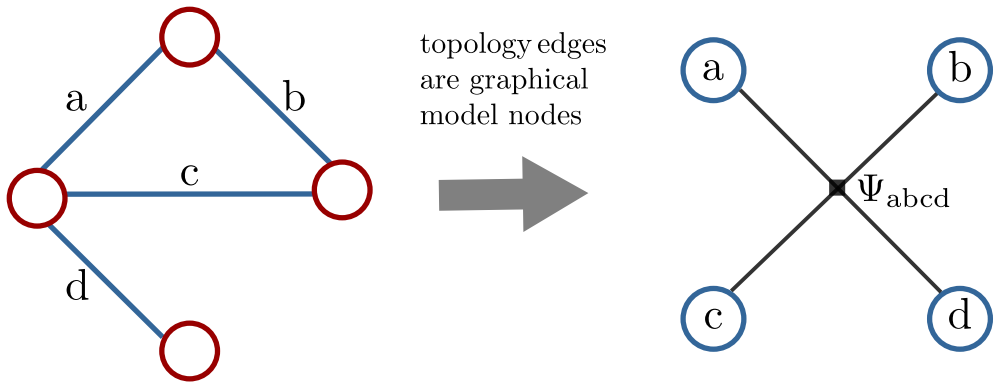
3. the environment configuration is independent of the temporal order of the runs, i.e., the configuration at run t has the same degree of dependence to the configuration at run $t+1$ than to the one at $t+k$, with $|k| > 1$;

4. the robot observes the traversability of an edge when reaching one of the vertices $\mathbf{v} \in \boldsymbol{V}$ which the edge connects to. The traversability detector allows for observing all the adjacent edges to $\mathbf{v}$.

6.1.3 Modeling patterns of change and predicting traversability

To make informed decisions for navigation in changing environments, we aim at learning a model of the patterns of change to predict the traversability at unknown locations. We use the robot's observations during traversal to incrementally learn a probabilistic model that captures the correlation among the edge traversability and that exploits this correlation to make predictions during navigation.

During navigation at run t, the environment presents a configuration $\boldsymbol{E}^t$ of which the robot typically only observes a subset of edges $\boldsymbol{Z}^t \subseteq \boldsymbol{E}^t$. To make informed decisions for navigation, we aim at predicting the current environment configuration $\boldsymbol{E}^t$ and, in particular, the traversability of the unobserved edges $\boldsymbol{U}^t$, with $\boldsymbol{E}^t = \boldsymbol{Z}^t \cup \boldsymbol{U}^t$. We formulate this problem as the problem of estimating the probability of the unobserved edges $\boldsymbol{U}^t$ to be traversable conditioned on the



(a) Example topology.

(b) Factor graph joint probability.

Figure 6.2: Example of topology (a), where the red circles are the nodes and the blue lines are the edges, and the graphical model of the joint probability distribution over the topology edges (b). The topology edges correspond to the nodes of the graphical model.

partial observation of the environment configuration $\boldsymbol{Z}^t$:

$$p(\boldsymbol{U}^t \mid \boldsymbol{Z}^t) = \frac{p(\boldsymbol{U}^t, \boldsymbol{Z}^t)}{p(\boldsymbol{Z}^t)}, \tag{6.1}$$

$$= \eta\, p(\boldsymbol{E}^t), \qquad \boldsymbol{E}^t = \boldsymbol{Z}^t \cup \boldsymbol{U}^t \tag{6.2}$$

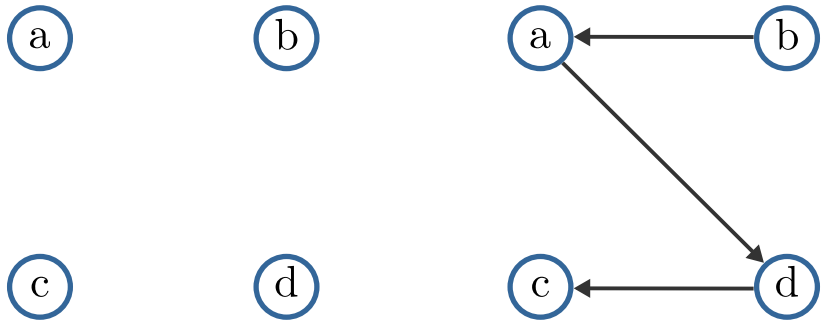
where η is a normalizer given the current observations $\boldsymbol{Z}^t$ and $p(\boldsymbol{E}^t)$ is the joint probability distribution over the traversability of the edges in the environment.

The distribution $p(\boldsymbol{E}^t)$ defines a probability function over the space of possible configurations. It captures the correlation among the traversability of edges and, thus, its knowledge is essential to make predictions about the environment configuration. In general, $p(\boldsymbol{E}^t)$ is a distribution without a special structure and the space required to represent it is exponential in the number of edges $|\boldsymbol{E}|$. Therefore, representing the joint distribution over the edges becomes quickly intractable. A graphical model representation of the full joint probability of the simple example topology in Figure 6.2(a) is illustrated in Figure 6.2(b), where the model nodes are the topology edges and the factor Ψ_{abcd} defines a probability over all the possible combinations of values of a, b, c, and d.

There exist different approaches to compute a tractable approximation of the joint probability distribution. We discuss two common approximations in the following.

6.1.3.1 Independent variables approximation

The simplest approach to approximate the joint probability distribution over the edges is to consider each topology edge to be independent of all others. Under



(a) Independent variables approximation.

(b) Chow-Liu tree approximation.

Figure 6.3: Possible approximations of the joint probability distribution in Figure 6.2(b) over the edges of the topology in Figure 6.2(a).

this assumption, we would approximate the joint distribution as:

$$p(\boldsymbol{E}^t) \approx \prod_{i}^{|\boldsymbol{E}|} p(\mathrm{e}_i^t). \tag{6.3}$$

The graphical model corresponding to this approximation for the example topology in Figure 6.2(a) is illustrated in Figure 6.3(a). This representation is efficient to compute and to store. However, considering edges independent of each other does not allow for learning the correlation among them and, thus, for capturing and predicting the spatial patterns of change in the environment.

6.1.3.2 Chow-Liu tree approximation

A more advanced approach to approximate the joint probability distribution is the Chow-Liu tree approximation. Chow and Liu [27] introduce a polynomial-time algorithm to approximate the joint probability that has been successfully used in the context of SLAM [32]. The Chow-Liu tree approximation is obtained by considering the mutual information graph between the random variables. This is a complete graph where each edge has a weight equals to the mutual information between the variables which it connects to. Chow-Liu algorithm computes the maximum spanning tree over the mutual information graph to select the tree-structured Bayesian network that presents the minimum Kullback-Leibler divergence to the original distribution. A Bayesian network [147] is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a directed acyclic graph. The tree representation implies that each variable is conditioned on at most one other variable. Therefore, we can

approximate the joint probability distribution over the edges of a topology as:

$$p(\boldsymbol{E}^t) \;\approx\; p(\mathrm{e}^t_{\mathrm{root}}) \prod_{i}^{|\boldsymbol{E}|} p(\mathrm{e}^t_i \mid \mathrm{parent}(\mathrm{e}^t_i)), \tag{6.4}$$

where $\mathrm{e}^t_{\mathrm{root}}$ is the root of the tree and parent$(\cdot)$ indicates the parent edge in the tree-structured Bayesian network. The Bayesian network of a possible Chow-Liu tree approximation for the example topology in Figure 6.2(a) is depicted in Figure 6.3(b).

The Chow-Liu tree approximation is attractive as it only requires space quadratic in the number of variables and is able to capture some of the correlation between edges. Yet, this approximation requires an amount of training data and performing offline computation to obtain the Bayesian network representation. In contrast to that, we aim at learning a model online and incrementally without any initial prior knowledge or training data. Using the Chow-Liu tree approximation to this end would require to compute the mutual information graph, the maximum spanning tree, and the parameters of the corresponding Bayesian network for each new robot observation. The robot, however, observes the environment continuously and it is not feasible to compute online an updated model of the edge traversability to make predictions for navigation. Furthermore, in a single run, the robot might observe only a subset of the edges of the topology. If this is the case, it is non-trivial to update the conditional probability tables of the Bayesian network using partial observations. We consider instead an approximation based on a flexible probabilistic graphical model that allows for efficiently updating the model by incorporating incrementally the new partial observations acquired by the robot.

6.2 Improving navigation estimating patterns in traversability changes

The main contribution of this chapter is a novel framework for robot navigation that is capable of capturing spatial patterns of change in an environment and exploiting this knowledge to plan informed strategies in partially unknown configurations. We achieve this by considering a topological map of the environment and incrementally learning a probabilistic model of the patterns in the traversability of the edges from the robot's observations during traversal. This model exploits the correlation among the traversability of the edges to make predictions about the configuration of the environment. We exploit the predicted configurations to plan navigation strategies that account both for the risk to encounter a blocked passage and, at the same time, for the information gain of making observations

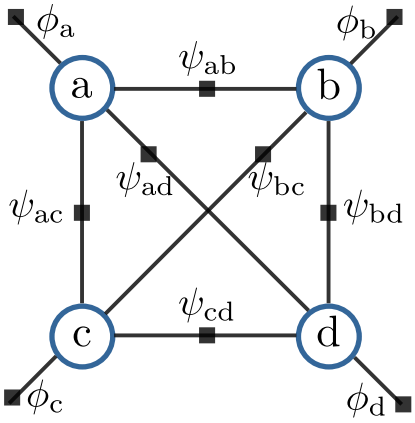


Figure 6.4: Our factor graph model to approximate the joint probability distribution in Figure 6.2(b) over the edges of the topology in Figure 6.2(a).

that improve the model. As a result of that, our approach is able to (i) learn incrementally a model of the patterns of change in the environment traversability from robot's observations; (ii) make predictions about the traversability at unobserved locations; (iii) plan paths that exploit the predictions to make informed decisions for navigation and, thus, to navigate efficiently in the environment.

6.2.1 Estimating patterns in traversability changes

To make predictions about the traversability in the environment from the robot's observations, we aim at learning a model that approximates the joint probability distribution over the edges in Equation (6.2). We require this model to (i) have a tractable representation for real-world indoor environments, (ii) capture the correlation between the traversability changes, and (iii) incorporate incremental and partial observations.

6.2.1.1 Factor graph model

To define a tractable representation of the joint probability distribution over the environment configurations that still captures dependency patterns well, we propose to model and predict the traversability changes in an environment by using a factor graph representation. A factor graph [81] is a probabilistic graphical model that allows for representing a general factorization of a function. A short introduction to factor graphs is provided in Section 2.2.1.

We define a factor graph model in which the variable nodes correspond to the edges $\boldsymbol{E}$ of the topology. As the correlation among the edges is unknown a priori, we consider a factor graph with a general structure that consists of one unary factor node for each edge and one binary factor node for each pair of topology edges. Considering such factors, we are assuming that we can approximate the

joint distribution over the edges as:

$$p(\boldsymbol{E}^t) \;\approx\; p_{\phi\psi}(\boldsymbol{E}^t) \;=\; \eta \prod_i \phi_i \prod_j \psi_{ij}, \tag{6.5}$$

where $\phi_i(\cdot)$ and $\psi_{ij}(\cdot)$ represent respectively the unary and binary factors of the graph, and η is a normalizer. An example of such factor graph model for the example topology in Figure 6.2(a) is illustrated in Figure 6.4.

This factor graph representation allows for approximating the probability over the environment configuration by capturing some of the correlation between the edge traversability while storing only $|\boldsymbol{E}|(|\boldsymbol{E}|-1)/2 + |\boldsymbol{E}|$ low-dimensional factors. Therefore, our model requires only quadratic space in the number of edges, i.e., $\mathcal{O}(|\boldsymbol{E}|^2)$, rather than exponential as in the case of the full joint probability distribution.

6.2.1.2 Learning factors from observations

Given our factor graph representation, we need to define the factors ϕ and ψ such that the model corresponds to the robot's observations collected in the previous runs and we can efficiently update the model in an incremental manner as the robot acquires new observations.

The belief propagation algorithm [131] allows for performing inference on factor graphs. We use the belief propagation algorithm to make predictions from current data (see Section 6.2.1.3) but also to estimate the model parameters from the robot's observations. Furtlehner *et al.* [52] introduce an approach to estimate the factor nodes from the marginal probabilities by using the fixed points of belief propagation. We use this approach to define the unary factors and binary factors of our factor graph model as:

$$\phi_i \;=\; p(\mathrm{e}_i), \tag{6.6}$$

$$\psi_{ij} \;=\; \frac{p(\mathrm{e}_i,\, \mathrm{e}_j)}{p(\mathrm{e}_i)\, p(\mathrm{e}_j)}, \tag{6.7}$$

where $p(\mathrm{e}_i)$ and $p(\mathrm{e}_i, \mathrm{e}_j)$ are respectively the unary and binary joint probabilities of the edges to be traversable or blocked. Note that Equation (6.7) has an analogy with the mutual information between e_i and e_j that is non-zero for $p(\mathrm{e}_i,\, \mathrm{e}_j) \neq p(\mathrm{e}_i)\, p(\mathrm{e}_j)$. This analogy provides an intuition that such definition of the factors allows for modeling the correlation between edges. This definition of factor nodes allows for computing the approximated joint probability distribution in Equation (6.5) as:

$$p_{\phi\psi}(\boldsymbol{E}^t) \;=\; \eta \prod_i^{|\boldsymbol{E}|} p(\mathrm{e}_i) \prod_j^{|\boldsymbol{E}|} \frac{p(\mathrm{e}_i, \mathrm{e}_j)}{p(\mathrm{e}_i)\, p(\mathrm{e}_j)}. \tag{6.8}$$

We compute the unary and binary joint probabilities, $p(\mathrm{e}_i)$ and $p(\mathrm{e}_i, \mathrm{e}_j)$, from the robot's observations in the previous runs $\boldsymbol{Z}^{1:t\text{-}1}$. We achieve this by maintaining a counter of the number of observed occurrences of each unary and binary configurations. To prevent probabilities to take the extreme values of 0 or 1 on a single observation, we initialize them with a uniform prior by assigning to each configuration an equal positive number of occurrences. After each run, we update the counters based on the robot's observations and recompute the probabilities. Dealing with unary and binary joint probabilities allows us to update only the probabilities corresponding to the edges observed at each run. Using this procedure, we can incrementally and efficiently compute the model's parameters from the robot's observations. The other key advantage of this procedure is that it allows for easily incorporating partial observations of the environment. For instance, in the example illustrated in Figure 6.2(a), if the robot observes the edges a and b but not c and d, we update $p(\mathrm{a})$, $p(\mathrm{b})$ and $p(\mathrm{a}, \mathrm{b})$, but not $p(\mathrm{a}, \mathrm{c})$ and $p(\mathrm{b}, \mathrm{d})$.

6.2.1.3 Predicting edge traversability

The factor graph model described above allows us to maintain a tractable approximation of the joint probability distribution over the edges, but it also provides us a tool to make predictions about the traversability of currently unknown edges. The belief propagation algorithm allows for computing the maximum a posteriori configuration and for estimating the marginal probability of each edge to be traversable.

Given the observations obtained during the current run $\boldsymbol{Z}^t$, we predict the traversability of the unobserved edges $\boldsymbol{U}^t$ by fixing the observed edges to the observed values in the factor graph and by performing approximate loopy belief propagation. This procedure allows for estimating $p(\boldsymbol{U}^t \mid \boldsymbol{Z}^t)$ and, thus, to compute a belief about the current environment configuration.

6.2.2 Planning exploiting traversability predictions

Given the predictions about the current environment configuration, we can exploit this knowledge to make informed decisions for navigation. We aim at planning non-myopic strategies that allow the robot to perform anticipatory behaviors so that it encounters a reduced number of unforeseen obstacles over time and, thus, navigates along shorter paths in the long run.

To achieve this, we need to define a path planning algorithm that presents two main characteristics. First, it should explore the belief space of the possible configurations to select short paths to the goal while taking into account the uncertainty of the predictions. Second, as we have no initial knowledge about

the environment configurations, it should lead the robot to make informative observations that improve the model of the edge traversability and, thus, the predictions in the subsequent runs.

6.2.2.1 Canadian traveler's problem

Our planning problem reveals similarities to the so-called stochastic Canadian traveler's problem (CTP) [129]. In contrast to the traveling salesman problem, the CTP describes a traveler who seeks to find the shortest path to a goal location in a road network where a subset of roads is not traversable. Initially, the agent does not know which roads are traversable but it knows the probability of each road to be blocked. The agent observes the actual traversability of the roads only while navigating through the network. Finding an optimal solution for the CTP is PSPACE-complete [47], thus it is intractable for any real-world application. However, there exist different approaches to compute approximate solutions. Eyerich *et al.* [39] proposed a Monte Carlo search algorithm based on the Upper Confidence tree algorithm [71] called CTP-UCT. It allows for computing approximate solutions for CTPs by taking the uncertainty of the current belief about the environment into account.

Given the current belief about the environment configuration, CTP-UCT iteratively performs a sequence of rollouts. A rollout consists of randomly sampling a configuration of the environment according to the current belief and of simulating the agent navigation based on this configuration. During the rollouts, it assumes that the agent has no knowledge about the sampled configuration but it can acquire observations during traversal as if traversing the actual road network. In each rollout, CTP-UCT selects locations for navigation by preferring the ones that led to the goal through short paths and have been selected less often in the previous rollouts. To this end, at each step of the rollouts, it considers a state $\mathbf{s}$ composed by the agent's current location, the set of known traversable and blocked edges, and the set of unknown edges. Let $\boldsymbol{\rho}_i = \{\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_i\}$ be the current sequence of states at the k-th rollout, CTP-UCT selects the next state $\mathbf{s}_{i+1}$ that maximizes the UCT formula:

$$\mathbf{s}_{i+1} = \operatorname*{argmax}_{\mathbf{s}} B\sqrt{\frac{\log R^{k\text{-}1}(\boldsymbol{\rho}_i)}{R^{k\text{-}1}(\boldsymbol{\rho}_{i+1})}} - \mathit{length}(\mathbf{s}_i, \mathbf{s}) - C^{k\text{-}1}(\boldsymbol{\rho}_{i+1}), \tag{6.9}$$

where

- $\boldsymbol{\rho}_{i+1} = \{\boldsymbol{\rho}_i, \mathbf{s}_{i+1}\}$ is the new sequence of states upon selecting $\mathbf{s}'$,
- $B > 0$ is a parameter that balances exploration and exploitation,
- $\mathit{length}(\mathbf{s}_i, \mathbf{s}_{i+1})$ is the travel distance from $\mathbf{s}_i$ to $\mathbf{s}_{i+1}$,

- $R^{k\text{-}1}(\boldsymbol{\rho}_i)$ indicates the number of rollouts among the previous $k-1$ rollouts that start with sequence $\boldsymbol{\rho}_i$,

- $C^{k\text{-}1}(\boldsymbol{\rho}_i)$ indicates the average travel distance of the previous $k-1$ rollouts that start with sequence $\boldsymbol{\rho}_i$ to reach the goal.

After performing a number of rollouts, CTP-UCT selects the path $\mathcal{P}$ to the goal that minimizes:

$$cost_{\text{CTP-UCT}}(\mathcal{P}) \;=\; \overline{length}(\mathcal{P}), \tag{6.10}$$

where $\overline{length}(\mathcal{P})$ is the average distance traveled to the goal during the rollouts by following $\mathcal{P}$.

The main difference between the CTP and our problem definition is that we assume the robot to traverse the environment several times with different configurations and that no informative prior probabilities are initially provided about the environment configurations. Therefore, the robot should plan policies that guide it along short paths but, at the same time, it should collect information for improving the prediction model. Hence, we aim at making decisions that trade off the exploitation of the predictions and the exploration of the environment.

Although CTP-UCT relies on an exploration-exploitation trade-off to search in the belief space, it makes decisions according to Equation (6.10), which minimizes the estimated average distance to the goal, and, thus, follows an optimistic-greedy criterion. We introduce an approach that trades off exploration and exploitation in the locations actually visited by the robot. To achieve this, we extend the CTP-UCT cost function to select paths so that, in the initial runs, the robot collects information about the traversability in the environment for improving the model and the predictions in the subsequent runs.

6.2.2.2 Information-driven exploration

In many robotic applications that aim at exploring or monitoring the environment, the exploration is guided by the mutual information [78]. The mutual information [104] also called expected information gain, quantifies the amount of information obtained about one random variable by observing another random variable. The mutual information between two discrete random variables a and b is defined as:

$$\mathcal{I}(\mathrm{a},\,\mathrm{b}) \;=\; \mathcal{H}(\mathrm{a}) - \mathcal{H}(\mathrm{a} \mid \mathrm{b}), \tag{6.11}$$

$$\;=\; \sum_{a\in\mathrm{a},\, b\in\mathrm{b}} p(a,\,b)\,\log\frac{p(a,\,b)}{p(a)\,p(b)}, \tag{6.12}$$

where $\mathcal{H}(\cdot)$ and $\mathcal{H}(\cdot \mid \cdot)$ are the entropy and the conditional entropy that are defined respectively as:

$$\mathcal{H}(\mathrm{a}) = -\sum_{a \in \mathrm{a}} p(a) \log p(a), \tag{6.13}$$

$$\mathcal{H}(\mathrm{a} \mid \mathrm{b}) = -\sum_{a \in \mathrm{a},\, b \in \mathrm{b}} p(a,\, b) \log \frac{p(a,\, b)}{p(b)}. \tag{6.14}$$

A common approach to collect information about the environment is to make observations at locations that maximize the mutual information about the non-observed regions, for example, Krause *et al.* [77]. We use a similar approach in our problem to bias the robot to make informative observations for improving the edge traversability model. Given the current set of unobserved edges $\boldsymbol{U}^t$, we select the path $\mathcal{P}$ that maximizes the information gain:

$$\mathcal{P} = \underset{\boldsymbol{P} \subseteq \boldsymbol{E}}{\operatorname{argmax}} \quad \mathcal{I}(\boldsymbol{P},\, \boldsymbol{U}^t) = \mathcal{H}(\boldsymbol{P}) - \mathcal{H}(\boldsymbol{P} \mid \boldsymbol{U}^t), \tag{6.15}$$

where $\boldsymbol{P}$ is the set of edges along the path $\mathcal{P}$.

Computing the entropy over the edges $\boldsymbol{P}$ requires the knowledge of their joint probability distribution. However, our factor graph model does not provide direct access to these probabilities. Therefore, we propose to approximate the mutual information $\mathcal{I}(\boldsymbol{P},\, \boldsymbol{U}^t)$ with the sum over the pairwise mutual information between the edges along $\boldsymbol{P}$ and the unobserved ones $\boldsymbol{U}^t$:

$$\mathcal{I}(\boldsymbol{P},\, \boldsymbol{U}^t) \approx \hat{\mathcal{I}}(\boldsymbol{P},\, \boldsymbol{U}^t) = \sum_{\mathrm{u} \in \boldsymbol{U}^t} \max_{\mathrm{p} \in \boldsymbol{P}} \mathcal{I}(\mathrm{p},\, \mathrm{u}), \tag{6.16}$$

where $\mathcal{I}(\mathrm{p},\, \mathrm{u})$ is computed using Equation (6.12). This approximation requires only unary and binary joint probabilities that are directly available from our factor graph model. Furthermore, we make sure that the mutual information for the same edge is not counted multiple times by considering the maximum mutual information for each unobserved edge u.

6.2.2.3 Exploration-exploitation trade-off

Given a prediction about the current configuration of the environment provided by our edge traversability model, we compute a policy to navigate to the goal which aims at minimizing the travel distance while exploring the environment configurations in the initial runs. To achieve this, we perform a sequence of rollouts based on the belief space defined by the prediction. We select successors as in the original CTP-UCT to estimate the travel distance to reach the goal in the partially observed environment. Given the outcomes of the rollouts, we select

paths according to a cost function which combines the information gain of the path and the estimated travel distance to reach the goal:

$$cost_{\text{OUR}}(\mathcal{P}) \;=\; \overline{length}(\mathcal{P}) - \gamma^{\#\text{runs}} \left[\zeta\, \hat{\mathcal{I}}(\boldsymbol{P}, \boldsymbol{U}^t)\right], \tag{6.17}$$

where $\boldsymbol{P}$ are the edges along the path $\mathcal{P}$, $\overline{length}(\mathcal{P})$ is the average travel distance to reach the goal following $\mathcal{P}$ in the rollouts, $\gamma \in [0, 1]$ is a parameter that controls the exploration term, ζ is a constant that normalizes the mutual information with respect to the travel distance, and $\hat{\mathcal{I}}$ is the approximated mutual information computed using Equation (6.16).

The exploratory behavior of the robot is determined by the parameter γ that decays exponentially with the number of runs performed by the robot. This parameter can be interpreted similarly as the discount rate of rewards used in Markov decision processes described in Section 2.4.1. Initially, when few observations are available, γ leads the robot to favor exploratory behaviors for improving the model of the edge traversability. As the robot performs a number of runs and acquires several observations of the environment, the model and its ability to make predictions improve and the exploration behavior becomes less and less prominent. When the learning process of the model converges, our problem becomes similar to a CTP in which the traversability of the edges is correlated. At this point, the exploratory term in Equation (6.17) is weighted low and, thus, the cost function becomes similar to the original CTP-UCT.

6.3 Experimental evaluation

Our experiments are designed to illustrate that our approach is able to (i) model and make predictions about the traversability changes in the environment from the robot's incremental observations; (ii) plan paths that exploit the predictions to make informed decisions for navigation; (iii) navigate along paths that are on average shorter than following greedy shortest path strategies.

6.3.1 Experimental setup

In our experiments, we consider different topologies defined over real-world environments such as the one illustrated in Figure 6.5, which depicts a topological map overlaying the grid map of our office. There exist many approaches to build topological maps of the environment, for example, Kuipers *et al.* [86]. On these topologies, we simulate patterns in the traversability changes. To this end, we first randomly sample M independent template configurations for each environment. Then, we generate a set of N, $N \gg M$, environment configurations by sampling uniformly one of the M templates and applying random noise on edge traversability.

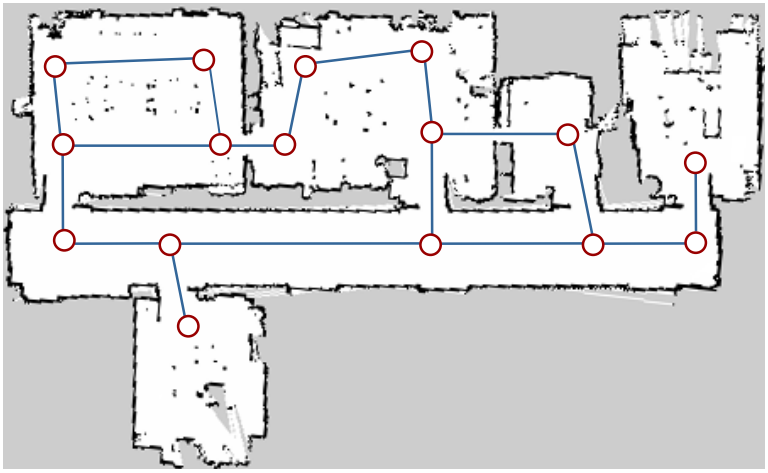


Figure 6.5: A topological map overlaying the grid map of our office used in our experiments. The topology nodes are locations of interest denoted by blue circles, whereas the red lines indicate the traversable passages among the nodes which are the topology edges.

6.3.2 Predicting edge traversability

We designed the first experiment to show the capabilities of our approach to model the traversability changes in the environment and to make predictions about the environment configurations. In this experiment, we consider a relatively small topology composed by 9 nodes and 13 edges and assume that the robot observes at each run the whole environment. These assumptions allow for computing the *full* joint distribution over the edge traversability despite its exponential space complexity and to use it for comparison. We additionally compare our approach with a model that assumes each edge to be independent of all others, as the one discussed in Section 6.1.3.1. To investigate the performance of our approach, we consider two different cases: one in which the environment configurations are highly correlated and one in which the correlation is low.

We compare the capabilities of each model to predict the edge traversability for 10000 partial configurations after 10, 100, 1000, and 10000 observations. In Figure 6.6, we illustrate the RMS error of the predictions to the true configurations for the three approaches. The solid lines indicate the results for the case in which the environment configurations are highly correlated, whereas the dotted lines denote the case in which the correlation is low. In cases in which the configurations are highly correlated, our approach (blue) is able to provide good predictions already after a few observations. The predictions improve incrementally as the robot collects more observations, similarly as if using the full joint probability distribution (green). Instead, assuming the edges to be independent (red), we cannot capture the correlation between edges and leads to worse predictions. In cases in which the configurations have low correlation, all of the three approaches provide similar predictions. Therefore, also in situations with

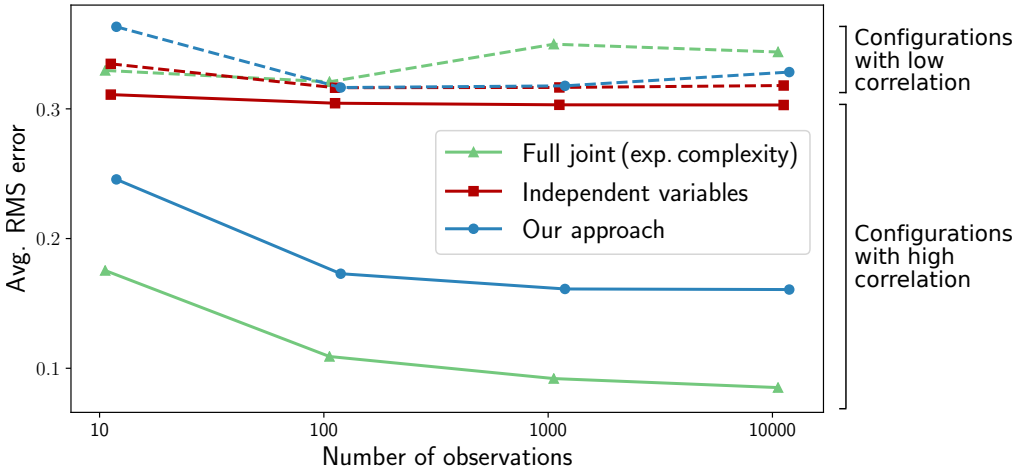


Figure 6.6: Average RMS error of the predictions for low and high correlated environment configurations using different models.

low correlation, our approach shows a similar behavior as the full joint probability distribution and does not reveal worse performance than the model assuming independence among edges.

In sum, the first experiment shows that our approach is able to make accurate predictions of the edges traversability which are in line with the ones considering the full joint probability. It also shows that, as expected, the predictions improve incrementally with the number of robot's observations.

6.3.3 Navigation exploiting predictions

The second experiment is designed to show that our approach is able to exploit the predictions of the environment configurations to plan informed strategies that lead the robot to navigate along shorter paths over time. In this experiment, we consider the four different environments described in Table 6.1: three offices of different dimensions, including the one illustrated in Figure 6.5, and one hospital where we sampled different patterns of change in the edge traversability. To evaluate the improvement over time, we repeat a fixed sequence of 25 navigation tasks and environment configurations for a total of 500 runs in each environment. At every run, the environment configuration can be different and the robot has no initial knowledge about it but from its previous observations. We compare our approach to an optimistic shortest path policy referred to as SPO. This strategy plans paths by using A* and assumes that every edge of the environment is traversable unless the robot observes the opposite and re-plans. The SPO does not take into account the predictions of the environment configuration and, thus, independently from the number of runs, makes the same decisions as if navigating in the environment for the very first time. For comparison, we also consider the theoretical optimal path computed in the ground truth environment configura-

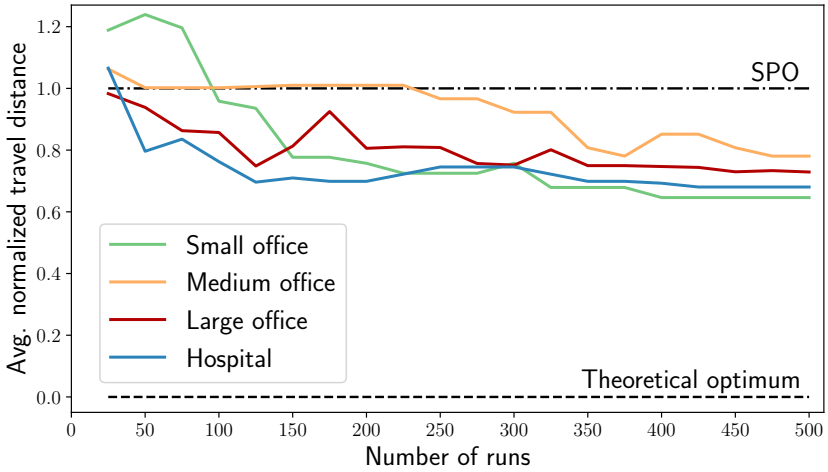


Figure 6.7: Average distance traveled by the robot following our approach in the environments described in Table 6.1 over the number of runs. We normalized the travel distance between the theoretical optimum (black dashed line) and the SPO solution (black dash-dotted line). As can be seen, as new observations of the environment get incorporated into our model, our approach is able to lead the robot to navigate along shorter paths. After 500 runs, the robot follows paths that are on average about 30% shorter than following a greedy shortest path strategy.

Environment	Dimensions	Nodes	Edges
Small office	25 × 20 m	16	18
Medium office	30 × 30 m	20	30
Large office	50 × 30 m	18	37
Hospital	125 × 35 m	40	55

Table 6.1: Description of the environments considered in our experimental evaluation.

tion, which is in practice unknown to the robot. In our approach, we perform 50 rollouts per decision and set the parameter that regulates the exploratory term of the cost function in Equation (6.17) to $\gamma = 0.95$.

The performance of our approach over the number of runs in the four environments is illustrated in Figure 6.7. We evaluate the average difference in the travel distance to the theoretical optimum with ground truth knowledge available (0.0) normalized with respect to the SPO solution (1.0), which leads the robot greedily towards the goal. Initially, when the robot collected little information about the environment, the predictions of the edge traversability are weak. Thus, following our approach, the robot performs similarly as by following SPO. After 100 runs, the robot starts discovering patterns in the traversability changes. Thus, the predictions improve and our approach is able to plan paths that lead the robot to the goal along shorter paths. Over time, when the robot collects more and more observations about the environment, the learning process of the edge traversability model converges. After 500 runs, the robot navigates along paths

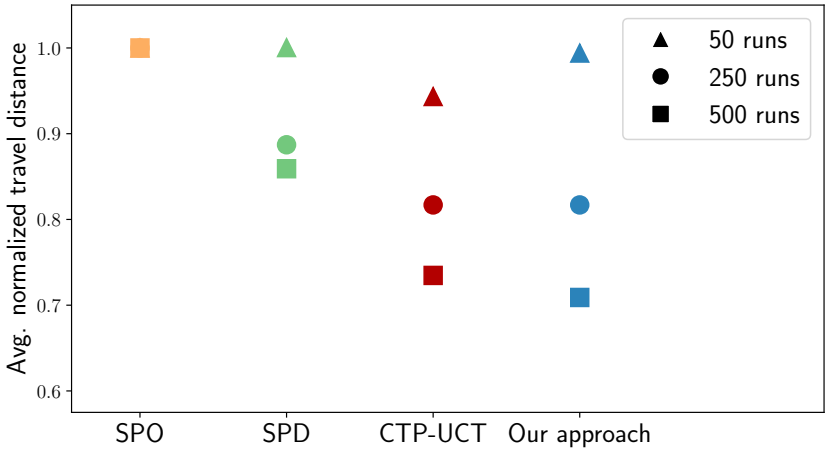


Figure 6.8: Average distance traveled by the robot following different planning approaches in the environments described in Table 6.1 over the number of runs. As can be seen, our approach presents a larger improvement over time and, after 500 runs leads the robot to navigate along shorter paths than following other approaches. The CTP-UCT and the SPD use our factor graph model for computing the predictions.

that are on average about 30% shorter than following an optimistic shortest path strategy.

This experiment showcases the ability of our approach to exploit the predictions of the edge traversability to lead the robot to navigate along shorter paths as new robot's observations about the environment get incorporated into our system.

6.3.4 Planning performance comparison

Besides the baselines discussed in the previous section, we compare the performance of our approach to other planning approaches. For comparison, we consider the original CTP-UCT [39] that searches for the shortest path in the predicted belief without explicitly exploring the environment, as described in Section 6.2.2.1. We compare our approach also to a strategy inspired by Lim *et al.* [99] referred to as the determinized shortest path or SPD. The SPD considers a most likely assumption on the belief about the edge traversability and plans the shortest path using A* on the determinized environment configuration. When the robot makes an observation incompatible with the current determinized configuration, the SPD computes a new prediction and re-plans.

It is important to note that both the CTP-UCT and the SPD do not provide an approach to model and make predictions of the edge traversability in the environment but assume that the belief is provided as prior information. Therefore, in this comparison, we consider the predictions provided by our factor graph model also for these approaches. Otherwise, no comparison would be possible.

We illustrate in Figure 6.8 the performance of the algorithms and their abil-

ity to exploit the predictions for navigating along short paths. We evaluate the average distance traveled by the robot following each of the approaches after 50, 250, and 500 runs for navigating in the environments introduced in the previous section. The SPO (orange) reveals a constant trend over time as it does not take the predictions into account. Considering the predictions of the edge traversability, the SPD (green) leads the robot to navigate along shorter paths over time. However, the determinization of the predicted configurations is a strong assumption that may cause the robot to follow paths that are distant from the optimal ones. The CTP-UCT (red) considers a weaker approximation of the belief about the environment configuration by performing rollouts which allow for taking into account the uncertainty of the predictions. Thus, it can make more informed decisions than SPD and leads the robot to follow shorter paths over time. Our approach (blue) extends the CTP-UCT by considering an exploratory term that allows the robot to collect informative observations. It explicitly explores the environment in the initial runs thus improving the model and the predictions of the traversability of the edges. The exploratory behavior leads initially the robot to slightly longer travel distances than the CTP-UCT. But, in the long run, it allows the robot to navigate along even shorter paths.

6.4 Related work

This work focuses on robot navigation over extended periods of time in environments where the traversability changes by following patterns. Traditional approaches to robot navigation follow the shortest path to the goal and execute reactive strategies if the robot encounters an unforeseen obstacle along it, for example, Fox *et al.* [45]. Algorithms such as Lifelong A* [73] and D* Lite [72] allow for fast re-planning to react to unforeseen obstacles. More advanced approaches predict the short-term motion of obstacles and plan local deviation for collision avoidance [88, 156]. These approaches are effective to tackle unexpected and unpredictable obstacles but may lead the robot to perform the same sub-optimal behavior again and again in environments with recurrent patterns of change.

Del Duchetto *et al.* [36] introduce an incremental learning by demonstration approach to autonomously detect and recover from navigation failures and, thus, to avoid making the same mistake again and again. Fremen [76] enhances a topological map with a spectral model that allows for predicting the traversability of the edges as a function of the time of day. Fentanes *et al.* [42] use this model for planning paths that take into account the temporal periodicity of changes in the environment. This approach is similar to the one described in this chapter but, in contrast to that, we focus on capturing the spatial correlation among the changes in the environment instead of taking explicitly into account the time

information.

Our approach models the traversability patterns from incremental robot's observations during traversal and exploits this knowledge for navigation. Several approaches have been proposed to model changing environments for robot navigation. For example, Stachniss and Burgard [157] map the typical configurations of low-dynamic areas of the environment for improving localization. Conditional transition maps [82] offer a grid-based representation for learning the motion patterns of the objects in the environment. Whereas, dynamic Gaussian process occupancy maps [124] map long-term dynamics with a spatially-continuous representation that provides occupancy estimates.

We capture the traversability patterns in the environment by employing a model that approximates the joint probability over the traversability of all edges in the environment. As discussed in Section 6.1.3, working with the full joint probability is unfeasible in most real-world applications as it requires exponential space in the number of variables. In the context of SLAM, FAB-MAP [32] approximates the joint probability distribution over a set of visual features by using the Chow-Liu tree approximation [27]. We consider instead an approximation based on a factor graph [81] with a fixed but flexible structure described in detail in Section 6.2.1.1. We define the model's parameters according to the approach proposed by Furtlehner *et al.* [52] in the context of traffic prediction from a probe vehicle. This approach is based on the correspondence between the fixed points of the Belief Propagation algorithm used for inference and the stable points of the Bethe free energy [175]. Our representation approximates the joint probability over the edge traversability in the environment with a model that captures the spatial correlation while allowing for incrementally learning the model's parameters from the robot's observations.

Given a prediction about the traversability of an environment, we plan a path to the goal that exploits the prediction to make informed decisions for navigation. Planning paths in the belief space defined by the predictions can be formulated as a partially observable Markov decision process [5]. However, POMDPs are in practice intractable for real-world environments [128]. The Reactive Planning Problem [103, 165] plans policies that guarantee the robot to reach the goal in uncertain environments. Bhattacharya *et al.* [15] maximize the probability of reaching the goal by seeking the most persistent homology for the given probability map. Murphy *et al.* [113] samples the edge costs from a probabilistic costmap, generates a list of paths using A* and selects the most frequent path. RAG search [28] plans risk-aware paths in a graph where the edge costs are unknown by trading off exploration and exploitation. Pereira *et al.* [133] build risk-maps from historical data and use them to find minimum risk paths for autonomous underwater vehicles.

Our planning problem presents a similar formulation to the Canadian traveler's problem [129]. Here, an agent aims at traveling along the shortest path in a road network where some roads, unknown to the agent, are blocked. Lim *et al.* [99] extend the CTP introducing a Bayesian variant in which where the road states may be correlated. The main difference of the problem presented in this chapter to the CTP is that, in our formulation, no probability over the edges is known a priori. However, the robot navigates in the environment several times and, thus, can build a model of the edge traversability from its observations over time. The CTP and its variants are typically computationally intractable for real-world applications. Nikolova and Karger [122] approximate the CTP by considering graphs that consist only of disjoint paths. Eyerich *et al.* [39] introduce the CTP-UCT that is a Monte Carlo search algorithm for computing policies in a CTP by taking the uncertainty of the predictions into account. Bnaya *et al.* [19] extend this approach to a variant of the CTP where a number of agents need to travel to the same goal. Our planning approach that extends the CTP-UCT by computing paths that aim at collecting information about the environment to improve the edge traversability model while minimizing the travel distance.

Planning to collect informative observations about the environment is often referred to as informative path planning. Informative planning is widely used in robotic applications such as exploration and environmental monitoring. For example, Popovic *et al.* [137] use informative planning for active classification with UAVs in the context of precision agriculture, while Hollinger *et al.* [57] for active underwater inspection. A common approach to plan informative paths proposed by Krause *et al.* [77, 109] is to maximize the mutual information between the visited and the unknown locations. Many planning algorithms have been proposed for computing paths that maximize the information gain. For example, Binney *et al.* [18] use a branch and bound method for informative path planning, whereas Holliger *et al.* [58] use a sampling-based planning approach similar to the ones described in Section 2.3.2. Lim *et al.* [98] introduce an adaptive variant of informative path planning where the robot chooses the next sensing location conditioned on all information acquired so far. Our planning approach uses the mutual information to implement a navigation strategy that trades off exploration and exploitation, similarly to the system presented in Chapter 5. Following this approach, the robot can reduce the risk of encountering unforeseen obstacles along its path and, thus, navigates along shorter paths over time.

6.5 Conclusion

In this last technical chapter of the thesis, we formulated the problem of a robot navigating over extended periods of time in environments where changes

in traversability are correlated or occur according to patterns. We presented an approach to solving such a problem by modeling these patterns and exploiting this knowledge for navigation. We consider a topological map of the environment and learn incrementally a model of the traversability of the edges from the robot's observations during navigation. We introduce a probabilistic model based on a factor graph representation that allows for modeling the correlation between traversability changes in the environment while being tractable for real-world-scale topologies. We use this model to make predictions about the traversability of unobserved regions of the environment. We exploit these predictions to make informed decisions for robot navigation. To this end, we employ a path planning approach that searches the shortest path to the goal in the belief space defined by the predictions. We combine this approach with an exploratory behavior that allows the robot to collect informative observations about the environment for improving the traversability model and, thus, the predictions in the subsequent runs.

The experiments suggest that our approach is able to learn spatial patterns of changes in an environment and to provide more and more accurate predictions of the unobserved edges with a larger number of observations. Furthermore, we show that exploiting the predictions we can plan paths that lead the robot along trajectories that are on average shorter than following a myopic optimistic shortest path approach.

Although our approach presents a higher complexity in comparison to traditional planning systems, in environments where the traversability changes following certain patterns, it has the potential to lead robots to automatically navigate along shorter paths over time, thus increasing the efficiency of their operations.

Chapter 7

Conclusion

Autonomous mobile robots are nowadays deployed to perform a variety of tasks in different environments, from mobile manipulation on factory floors to self-driving taxis. In such real-world scenarios, robots are frequently required to navigate in complex, dynamic, and uncertain environments. When operating in these environments, traditional robot navigation systems, which plan and follow the shortest paths computed on static maps, may lead the robot to perform sub-optimal behaviors in practice.

In this thesis, we investigated four distinct problems that address robot navigation in diverse real-world scenarios. We presented different approaches to making decisions for navigation in such environments that go beyond planning the shortest path. We proposed solutions to deal with and reason about the several sources of uncertainty characterizing navigation in the real world. We took into account the uncertainty in the robot's belief about the world, its action execution, and sensor measurements to make decisions for navigation. We addressed navigation in partially observed environments by making predictions at unknown locations and planning in the belief space defined by the predictions. We additionally investigated solutions that allow robots to tackle continuously changing and highly dynamic scenarios by performing safe and predictable behaviors for avoiding collisions. We presented approaches to automatically adapt and improve the robot's behaviors according to the user's preferences, the degree of uncertainty, and the estimated characteristics or configurations of the environments. We achieve this by exploiting background information that is either available a priori, such as satellite images and publicly available maps of the environment, or collected by the robot during navigation, for example, experienced paths. We also proposed approaches to actively gather relevant information for improving navigation over time, such as the observations about the configuration of the environment or the onboard perception of the robot in different regions.

7.1 Short summary of the key contributions

The key contributions of this thesis are novel solutions to perform robot navigation in complex, dynamic, and uncertain environments. We summarize these contributions in the following.

The first contribution of this thesis is a solution for making decisions to navigate on a road network under large position uncertainty. We build a stochastic process that takes explicitly into account the uncertainty about the robot's actions and position while being computationally tractable. We use a localization prior to estimate how the robot's belief propagates into the network and make safe decisions accordingly. Following this approach, the robot is able to select actions based on the degree of uncertainty and, thus, to reduce the number of mistakes it makes when navigating with large position uncertainty.

The second contribution is a solution to generate behaviors that meet the user's preferences for navigating on factory floors. We collect the user's preferences about the robot's experiences and reuse the favorite paths to guide the planning process in new but similar situations. We also incorporate an uncertainty-aware trajectory predictor for avoiding unforeseen dynamic obstacles in this framework. Our approach allows the robot to perform safe and predictable paths by reproducing and generalizing preferred behaviors without requiring experts to define norms or tune its parameters.

The third contribution of this thesis is an approach to automatically reduce the detrimental factors due to the terrains that affect navigation in unknown outdoor environments. We use robot's onboard observations during navigation and an aerial image of the environment to learn a probabilistic model of these factors. We exploit this model to plan paths that trade off the exploration of unknown promising regions and the exploitation of known low-cost areas. Our approach leads the robot to navigate along paths following on terrains on which it experiences reduced vibration intensities or power consumption over time.

The fourth contribution is a solution to navigation over extended periods of time in environments where the traversability changes following patterns. We use the robot's observations during traversal to incrementally model these patterns and to predict the traversability in unobserved regions. We exploit the predictions to plan paths that minimize the travel distance while maximizing the information gain of visiting unknown locations. Following our approach, the robot is able to follow paths along which it encounters a reduced number of blocked passages and, thus, to navigate along shorter paths over time.

7.2 Future work

The work and the promising results presented in this thesis open different directions for future research.

An interesting direction for future work would be to investigate different representations of the uncertainty that are compact, informative, and tractable for planning. In this thesis, we have recourse often to Gaussian assumptions and approximations. However, these representations may be inadequate to describe and model well complex phenomena. Whereas, more involved models require a larger representation space and, thus, may dramatically increase the complexity of the planning problem. One possible idea to investigate is to learn embeddings of distributions that preserve the statistical features by using data-driven approaches.

We investigated approaches to improve navigation in a specific environment where the robot navigates over extended periods of time by meeting the user's preferences, by discovering low-cost regions, or by avoiding blocked passages. One interesting direction is to investigate approaches that transfer the models and the preferences learned in one environment to new similar environments.

Another interesting future work would be to investigate a unique general representation for planning that allows for taking into account seamlessly different sources of uncertainty, exploiting prior information, and actively gather new information when needed.

We presented solutions for making decisions in unknown and uncertain environments. We achieve this by explicitly modeling the observations collected during navigation, making predictions at unobserved regions, and planning paths in the belief space defined by the predictions. A different approach for solving this kind of problems that would be interesting to investigate is reinforcement learning. Reinforcement learning is a framework that formulates these problems as a unique control process and solves them by generating policies that maximize a reward function over time.

An interesting extension of the work presented in this thesis is to use similar solutions for multi-robot navigation systems. Multi-robot systems have the capability to parallelize information gathering and to synthesize observations from all robots. Therefore, using such systems could speed-up the exploration of the environment and, accordingly, the improvement of the robots' navigation behavior.

Bibliography

[1] A. Agha-Mohammadi, S. Chakravorty, and N. M. Amato. FIRM: Feedback Controller-based Information-state RoadMap–A Framework for Motion Planning Under Uncertainty. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[2] A. Agha-Mohammadi, S. Chakravorty, and N. M. Amato. FIRM: Sampling-based Feedback Motion Planning Under Motion Uncertainty and Imperfect Measurements. *Intl. Journal of Robotics Research (IJRR)*, 33(2):268–304, 2014.

[3] A. Ahmadi, L. Nardi, N. Chebrolu, and C. Stachniss. Visual Servoing-based Navigation for Monitoring Row-Crop Fields. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020.

[4] K. J. Astrom. Optimal Control of Markov Processes with Incomplete State Information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965.

[5] H. Bai, D. Hsu, and W.S. Lee. Integrated Perception and Planning in the Continuous Space: A POMDP Approach. *Intl. Journal of Robotics Research (IJRR)*, 33(9):1288–1302, 2014.

[6] Ruzena Bajcsy. Active Perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988.

[7] O. Bartlett, C. Gurau, L. Marchegiani, and I. Posner. Enabling Intelligent Energy Management for Robots Using Publicly Available Maps. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[8] R. Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[9] O. Bengtsson and A. Baerveldt. Robot Localization Based on Scan-Matching – Estimating the Covariance Matrix for the IDC Algorithm. *Journal on Robotics and Autonomous Systems (RAS)*, 44(1):29–40, 2003.

[10] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun. Learning Motion Patterns of People for Compliant Robot Motion. *Intl. Journal of Robotics Research (IJRR)*, 24(1):31–48, 2005.

[11] L. Berczi, I. Posner, and T. Barfoot. Learning to Assess Terrain from Human Demonstration Using an Introspective Gaussian-Process Classifier. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2015.

[12] D. Berenson, P. Abbeel, and K. Goldberg. A Robot Path Planning Framework that Learns from Experience. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 3671–3678, 2012.

[13] J. Van Den Berg, P. Abbeel, and K. Goldberg. LQG-MP: Optimized Path Planning for Robots with Motion Uncertainty and Imperfect State Information. *Intl. Journal of Robotics Research (IJRR)*, 30(7):895–913, 2011.

[14] J. Van Den Berg, S. Patil, and R. Alterovitz. Motion Planning Under Uncertainty Using Iterative Local Optimization in Belief Space. *Intl. Journal of Robotics Research (IJRR)*, 31(11):1263–1278, 2012.

[15] S. Bhattacharya, R. Ghrist, and V. Kumar. Persistent Homology for Path Planning in Uncertain Environments. *IEEE Trans. on Robotics (TRO)*, 31(3):578–590, 2015.

[16] A. Bhattacharyya. On a Measure of Divergence Between Two Statistical Populations Defined by Their Probability Distributions. *Bull. Calcutta Math. Soc.*, 35:99–109, 1943.

[17] J. Binney, A. Krause, and G. Sukhatme. Optimizing Waypoints for Monitoring Spatiotemporal Phenomena. *Intl. Journal of Robotics Research (IJRR)*, 32(8):873–888, 2013.

[18] J. Binney and G. Sukhatme. Branch and Bound for Informative Path Planning. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 2147–2154, 2012.

[19] Z. Bnaya, A. Felner, D. Fried, O. Maksin, and S.E. Shimony. Repeated-task Canadian traveler problem. In *Proc. of the Intl. Symposium on Combinatorial Search (SoCS)*, 2011.

[20] S. Bopardikar, B. Englot, and A. Speranzon. Robust Belief Roadmap: Planning under Uncertain and Intermittent Sensing. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2014.

[21] J. Bruce and M. Veloso. Real-Time Randomized Path Planning for Robot Navigation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, volume 3, pages 2383–2388, 2002.

[22] A. Bry and N. Roy. Rapidly-exploring Random Belief Trees for Motion Planning Under Uncertainty. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 723–730, 2011.

[23] F.M. Carlucci, L. Nardi, L. Iocchi, and D. Nardi. Explicit Representation of Social Norms for Social Robots. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4191–4196, 2015.

[24] N. Chebrolu, P. Lottes, T. Laebe, and C. Stachniss. Robot Localization Based on Aerial Images for Precision Agriculture Tasks in Crop Fields. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[25] X. Chen, T. Läbe, L. Nardi, J. Behley, and C. Stachniss. Learning an Overlap-based Observation Model for 3D LiDAR Localization. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[26] E. Choi and C. Lee. Feature Extraction based on the Bhattacharyya Distance. *Pattern Recognition*, 36(8):1703–1709, 2003.

[27] C. Chow and C. Liu. Approximating Discrete Probability Distributions with Dependence Trees. *IEEE Trans. on Information Theory*, 14(3):462–467, 1968.

[28] J.J. Chung, A. Smith, R. Skeele, and G. Hollinger. Risk-aware Graph Search with Dynamic Edge Cost Discovery. *Intl. Journal of Robotics Research (IJRR)*, 2018.

[29] D. Coleman, I.A. Sucan, M. Moll, K. Okada, and N. Correll. Experience-Based Planning with Sparse Roadmap Spanners. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2015.

[30] G. Coleman and H. Andrews. Image Segmentation by Clustering. *Proceedings of the IEEE*, 67(5):773–785, 1979.

[31] D. Cox and S. John. A Statistical Method for Global Optimization. In *Proc. of the IEEE Intl. Conf. on Systems, Man, and Cybernetics (SMC)*, 1992.

[32] M. Cummins and P. Newman. FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance. *Intl. Journal of Robotics Research (IJRR)*, 27(6):647–665, 2008.

[33] C. Cunningham, M. Ono, I. Nesnas, J. Yen, and W. Whittaker. Locally-Adaptive Slip Prediction for Planetary Rovers Using Gaussian Processes. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[34] V. Delsart and T. Fraichard. Navigating Dynamic Environments with Trajectory Deformation. *Journal of Computing and Information Technology*, 17(1):27–36, 2009.

[35] E. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[36] F. Del Duchetto, A. Kucukyilmaz, L. Iocchi, and M. Hanheide. Do Not Make the Same Mistakes Again and Again: Learning Local Recovery Policies for Navigation from Human Demonstrations. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):4084–4091, 2018.

[37] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia. Incorporating Second-Order Functional Knowledge for Better Option Pricing. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, 2001.

[38] D. Ellis, E. Sommerlade, and I. Reid. Modelling Pedestrian Trajectory Patterns with Gaussian Processes. In *Proc. of the Intl. Conf. on Computer Vision (ICCV) Workshops*, pages 1229–1234, 2009.

[39] P. Eyerich, T. Keller, and M. Helmert. High-quality Policies for the Canadian Traveler's Problem. In *Proc. of the Conf. on Advancements of Artificial Intelligence (AAAI)*, 2010.

[40] J. Farrell. *GNSS Aided Navigation & Tracking: Inertially Augmented Or Autonomous*. American Literary Press Baltimore, MD, 2007.

[41] J.P. Fentanes, I. Gould, T. Duckett, S. Pearson, and G. Cielniak. 3-D Soil Compaction Mapping Through Kriging-Based Exploration With a Mobile Robot. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):3066–3072, 2018.

[42] J.P. Fentanes, B. Lacerda, T. Krajník, N. Hawes, and M. Hanheide. Now or later? Predicting and Maximising Success of Navigation Actions from Long-Term Experience. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1112–1117, 2015.

[43] L. Fermin-Leon, J. Neira, and J. A. Castellanos. Path Planning in Graph SLAM Using Expected Uncertainty. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4594–4601, 2016.

[44] G. Floros, B. van der Zander, and B Leibe. Openstreetslam: Global Vehicle Localization using Openstreetmaps. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2013.

[45] D. Fox, W. Burgard, and S. Thrun. The Dynamic Window Approach to Collision Avoidance. *IEEE Journal of Robotics and Automation*, 4(1):23–33, 1997.

[46] D. Fox, W. Burgard, and S. Thrun. Active Markov Localization for Mobile Robots. *Journal on Robotics and Autonomous Systems (RAS)*, 25(3-4):195–207, 1998.

[47] D. Fried, S. Shimony, A. Benbassat, and C. Wenner. Complexity of Canadian Traveler Problem Variants. *Theoretical Computer Science*, 487:1–16, 2013.

[48] K. Fukunaga and L. Hostetler. The Estimation of the Gradient of a Density Function, With Applications in Pattern Recognition. *IEEE Trans. on Information Theory*, 21(1):32–40, 1975.

[49] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier. Probabilistic Navigation in Dynamic Environment Using Rapidly-exploring Random Trees and Gaussian Processes. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1056–1062, 2008.

[50] P. Furgale, P. Krüsi, F. Pomerleau, U. Schwesinger, F. Colas, and R. Siegwart. There and Back Again–Dealing with Highly-dynamic Scenes and Long-term Change during Topological/Metric Route Following. In *Workshop on Modelling, Estimation, Perception, and Control of All Terrain Mobile Robots*, 2014.

[51] P.T. Furgale and T.D. Barfoot. Visual Teach and Repeat for Long-range Rover Autonomy. *Journal of Field Robotics (JFR)*, 27:534–560, 2010.

[52] C. Furtlehner, J.M. Lasgouttes, and D. de La Fortelle. A Belief Propagation Approach to Traffic Prediction using Probe Vehicles. In *IEEE Trans. on Intelligent Transportation Systems (ITS)*, 2007.

[53] P. Giguere and G. Dudek. Clustering Sensor Data for Autonomous Terrain Identification using Time-dependency. *Autonomous Robots*, 26(2-3):171–186, 2009.

[54] Y. Girdhar and G. Dudek. Modeling Curiosity in a Mobile Robot for Long-term Autonomous Exploration and Monitoring. *Autonomous Robots*, 40(7):1267–1278, 2016.

[55] K. Haigh and M. Veloso. Learning Situation-dependent Costs: Improving Planning from Probabilistic Robot Execution. *Journal on Robotics and Autonomous Systems (RAS)*, 29(2-3):145–174, 1999.

[56] M. Hentschel and B. Wagner. Autonomous Robot Navigation based on Openstreetmap Geodata. In *IEEE Trans. on Intelligent Transportation Systems (ITS)*, 2010.

[57] G. Hollinger, B. Englot, F. Hover, U. Mitra, and G. Sukhatme. Active Planning for Underwater Inspection and the Benefit of Adaptivity. *Intl. Journal of Robotics Research (IJRR)*, 32(1):3–18, 2013.

[58] G. Hollinger and G. Sukhatme. Sampling-based Motion Planning for Robotic Information Gathering. In *Proc. of Robotics: Science and Systems (RSS)*, volume 3, 2013.

[59] A. Hornung, H. Strasdat, M. Bennewitz, and W. Burgard. Learning Efficient Policies for Vision-based Navigation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[60] R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1964.

[61] V. Indelman, L. Carlone, and F. Dellaert. Planning in the Continuous Domain: A Generalized Belief Space Approach for Autonomous Navigation in Unknown Environments. *Intl. Journal of Robotics Research (IJRR)*, 34(7):849–882, 2015.

[62] N. Jetchev and M. Toussaint. Trajectory Prediction: Learning to Map Situations to Robot Trajectories. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, pages 449–456, 2009.

[63] N. Jetchev and M. Toussaint. Trajectory Prediction in Cluttered Voxel Environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 2523–2528, 2010.

[64] X. Jiang and M. Kallmann. Learning Humanoid Reaching Tasks in Dynamic Environments. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1148–1153, 2007.

[65] L. Kaelbling, M. Littman, and A. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[66] R.E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME – Journal of Basic Engineering*, 82:35–45, 1960.

[67] S. Karaman and E. Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. *Intl. Journal of Robotics Research (IJRR)*, 30(7):846–894, 2011.

[68] H. Kawano. Study of Path Planning Method for Under-actuated Blimp-type UAV in Stochastic Wind Disturbance via Augmented-MDP. In *Proc. of the IEEE/ASME Intl. Conf. on Advanced Intelligent Mechatronics (AIM)*, 2011.

[69] M. Keller, F. Hoffmann, C. Hass, T. Bertram, and A. Seewald. Planning of Optimal Collision Avoidance Trajectories with Timed Elastic Bands. *IFAC Proceedings Volumes*, 47(3):9822–9827, 2014.

[70] R. Kirby, R. Simmons, and J. Forlizzi. Companion: A Constraint-optimizing Method for Person-acceptable Navigation. In *Proc. of the IEEE Intl. Symp. on Robot and Human Interactive Communication (RO-MAN)*, pages 607–612, 2009.

[71] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo Planning. In *Proc. of the Europ. Conf. on Machine Learning (ECML)*, pages 282–293, 2006.

[72] S. Koenig and M. Likhachev. Fast Replanning for Navigation in Unknown Terrain. *IEEE Trans. on Robotics (TRO)*, 21(3):354–363, 2005.

[73] S. Koenig, M. Likhachev, and D. Furcy. Lifelong Planning A*. *Artificial Intelligence*, 155(1-2):93–146, 2004.

[74] D. Koller, N. Friedman, and F. Bach. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.

[75] K. Konolige, E. Marder-Eppstein, and B. Marthi. Navigation in Hybrid Metric-topological Maps. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2011.

[76] T. Krajník, J.P. Fentanes, J.M. Santos, and T. Duckett. Fremen: Frequency Map Enhancement for Long-Term Mobile Robot Autonomy in Changing Environments. *IEEE Trans. on Robotics (TRO)*, 33(4):964–977, 2017.

[77] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg. Near-optimal Sensor Placements: Maximizing Information while Minimizing Communication

Cost. In *Proc. of the Intl. Conf. on Information Processing in Sensor Networks*, pages 2–10, 2006.

[78] A. Krause, A. Singh, and C. Guestrin. Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. *Journal of Machine Learning Research*, 9:235–284, 2008.

[79] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard. Socially Compliant Mobile Robot Navigation via Inverse Reinforcement Learning. *Intl. Journal of Robotics Research (IJRR)*, 2016.

[80] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch. Human-aware Robot Navigation: A Survey. *Journal on Robotics and Autonomous Systems (RAS)*, 61(12):1726–1743, 2013.

[81] F. Kschischang, B. Frey, and H.A. Loeliger. Factor Graphs and the Sum-Product Algorithm. *IEEE Trans. on Information Theory*, 47(2):498–519, 2001.

[82] T. Kucner, J. Saarinen, M. Magnusson, and A. Lilienthal. Conditional Transition Maps: Learning Motion Patterns in Dynamic Environments. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1196–1201, 2013.

[83] M. Kuderer, H. Kretzschmar, and W. Burgard. Teaching Mobile Robots to Cooperatively Navigate in Populated Environments. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3138–3143, 2013.

[84] J. Kuffner and S. LaValle. RRT-Connect: An Efficient Approach to Single-query Path Planning. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, volume 2, pages 995–1001, 2000.

[85] B. Kuipers and Y. Byun. A Robust, Qualitative Method for Robot Spatial Learning. In *Proc. of the Conf. on Advancements of Artificial Intelligence (AAAI)*, 1988.

[86] B. Kuipers, J. Modayil, P. Beeson, M. MacMahon, and F. Savelli. Local Metrical and Global Topological Maps in the Hybrid Spatial Semantic Hierarchy. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, volume 5, pages 4845–4851, 2004.

[87] S. Kullback and R. Leibler. On Information and Sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[88] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. A Navigation System for Robots Operating in Crowded Urban Environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2013.

[89] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. Autonomous Robot Navigation in Highly Populated Pedestrian Zones. *Journal of Field Robotics (JFR)*, 2014.

[90] T. Lang, C. Plagemann, and W. Burgard. Adaptive Non-Stationary Kernel Regression for Terrain Modeling. In *Proc. of Robotics: Science and Systems (RSS)*, 2007.

[91] S. LaValle. Rapidly-exploring Random Trees: A New Tool for Path Planning. Technical report, 1998.

[92] S. LaValle. Motion Planning. *IEEE Robotics and Automation Magazine (RAM)*, 18(2):108–118, 2011.

[93] S. LaValle and J. Kuffner. Rapidly-exploring Random Trees: Progress and Prospects. Technical report, 2000.

[94] A. Levin and R. Szeliski. Visual Odometry and Map Correlation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 1, 2004.

[95] G. Lidoris, F. Rohrmuller, D. Wollherr, and M. Buss. The Autonomous City Explorer (ACE) project – Mobile Robot Navigation in Highly Populated Urban Environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2009.

[96] F. Liebisch, J. Pfeifer, R. Khanna, P. Lottes, C. Stachniss, T. Falck, S. Sander, R. Siegwart, A. Walter, and E. Galceran. Flourish – A Robotic Approach for Automation in Crop Management. In *In Proc. of the Workshop für Computer-Bildanalyse und unbemannte autonom fliegende Systeme in der Landwirtschaft*, 2016.

[97] M. Likhachev and R. Arkin. Spatio-temporal Case-based Reasoning for Behavioral Selection. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, volume 2, pages 1627–1634, 2001.

[98] Z.W. Lim, D. Hsu, and W.S. Lee. Adaptive Informative Path Planning in Metric Spaces. *Intl. Journal of Robotics Research (IJRR)*, 35(5):585–598, 2016.

[99] Z.W. Lim, D. Hsu, W.S. Lee, and W. Sun. Shortest Path under Uncertainty: Exploration versus Exploitation. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2017.

[100] M. Littman, T. Dean, and L Kaelbling. On the Complexity of Solving Markov Decision Problems. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 394–402, 1995.

[101] Y. Luo, P. Cai, A. Bera, D. Hsu, W.S. Lee, and D. Manocha. PORCA: Modeling and Planning for Autonomous Driving among many Pedestrians. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):3418–3425, 2018.

[102] K. Ma, L. Liu, and G. Sukhatme. Informative Planning and Online Learning with Sparse Gaussian Processes. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[103] R. MacDonald and S. Smith. Active Sensing for Motion Planning in Uncertain Environments via Mutual Information Policies. *Intl. Journal of Robotics Research (IJRR)*, 38(2-3):146–161, 2019.

[104] D. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2004.

[105] R. Marchant and F. Ramos. Bayesian Optimisation for Informative Continuous Path Planning. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2014.

[106] B. Matérn. *Spatial Variation*, volume 36. Springer Science & Business Media, 2013.

[107] G. Matheron. Principles of Geostatistics. *Economic Geology*, 58(8):1246–1266, 1963.

[108] M. Mazuran, C. Sprunk, W. Burgard, and G.D. Tipaldi. LexTOR: Lexicographic Teach Optimize and Repeat Based on User Preferences. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, Seattle, 2015.

[109] A. Meliou, A. Krause, C. Guestrin, and J. Hellerstein. Nonmyopic Informative Path Planning in Spatio-temporal Models. In *Proc. of the Conf. on Advancements of Artificial Intelligence (AAAI)*, volume 10, pages 16–7, 2007.

[110] T. Meriçli, M. Veloso, and H. L. Akın. A Case-Based Approach to Mobile Push-Manipulation. *Journal of Intelligent and Robotic Systems (JIRS)*, 80(1):189–203, 2015.

[111] W. Mou and A. Kleiner. Online Learning Terrain Classification for Adaptive Velocity Control. In *Proc. of the IEEE Conf. on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–7, 2010.

[112] E. Murphy and P. Newman. Planning Most-Likely Paths from Overhead Imagery. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2010.

[113] L. Murphy and P. Newman. Risky Planning on Probabilistic Costmaps for Path Planning in Outdoor Environments. *IEEE Trans. on Robotics and Automation*, 29(2):445–457, 2013.

[114] L. Nardi and L. Iocchi. Representation and Execution of Social Plans through Human-Robot Collaboration. In *International Conference on Social Robotics*, pages 266–275, 2014.

[115] L. Nardi and C. Stachniss. Experience-Based Path Planning for Mobile Robots Exploiting User Preferences. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[116] L. Nardi and C. Stachniss. User Preferred Behaviors for Robot Navigation Exploiting Previous Experiences. *Journal on Robotics and Autonomous Systems (RAS)*, 97, 2017.

[117] L. Nardi and C. Stachniss. Towards Uncertainty-Aware Path Planning for Navigation on Road Networks Using Augmented MDPs. In *10th Workshop on Planning, Perception and Navigation for Intelligent Vehicles at the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[118] L. Nardi and C. Stachniss. Actively Improving Robot Navigation On Different Terrains Using Gaussian Process Mixture Models. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[119] L. Nardi and C. Stachniss. Uncertainty-Aware Path Planning for Navigation on Road Networks Using Augmented MDPs. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[120] L. Nardi and C. Stachniss. Long-Term Robot Navigation in Indoor Environments Estimating Patterns in Traversability Changes. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020.

[121] R.M. Neal. *Bayesian Learning for Neural Networks*, volume 118. 2012.

[122] E. Nikolova and D.R. Karger. Route Planning under Uncertainty: The Canadian Traveller Problem. In *Proc. of the Conf. on Advancements of Artificial Intelligence (AAAI)*, pages 969–974, 2008.

[123] N. Nilsson. Shakey the Robot. Technical report, SRI International, 1984.

[124] S. O'Callaghan and F. Ramos. Gaussian Process Occupancy Maps for Dynamic Environments. In *Proc. of the Intl. Sym. on Experimental Robotics (ISER)*, pages 791–805, 2016.

[125] E. Olson. AprilTag: A Robust and Flexible Visual Fiducial System. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2011.

[126] P. Ondrúška, C. Gurău, L. Marchegiani, C. Tong, and I. Posner. Scheduled Perception for Energy-Efficient Path Following. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2015.

[127] K. Otsu, M. Ono, T. Fuchs, I. Baldwin, and T. Kubota. Autonomous Terrain Classification with Co-and Self-training Approach. *IEEE Robotics and Automation Letters (RA-L)*, 1(2):814–819, 2016.

[128] C. Papadimitriou and J. Tsitsiklis. The Complexity of Markov Decision Processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[129] C. Papadimitriou and M. Yannakakis. Shortest Paths Without a Map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[130] A. Patel. http://theory.stanford.edu/~amitp/GameProgramming.

[131] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Elsevier, 2014.

[132] D. Perea-Ström, I. Bogoslavskyi, and C. Stachniss. Robust Exploration and Homing for Autonomous Robots. In *Journal on Robotics and Autonomous Systems (RAS)*, 2016.

[133] A. Pereira, J. Binney, B. Jones, M. Ragan, and G. Sukhatme. Toward Risk Aware Mission Planning for Autonomous Underwater Vehicles. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3147–3153, 2011.

[134] M. Pfeiffer, U. Schwesinger, H. Sommer, E. Galceran, and R. Siegwart. Predicting Actions to Act Predictably: Cooperative Partial Motion Planning with Maximum Entropy Models. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2096–2101, 2016.

[135] M. Phillips, B. Cohen, S. Chitta, and M. Likhachev. E-Graphs: Bootstrapping Planning with Experience Graphs. In *Proc. of Robotics: Science and Systems (RSS)*, 2012.

[136] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez. Belief Space Planning Assuming Maximum Likelihood Observations. In *Proc. of Robotics: Science and Systems (RSS)*, 2010.

[137] M. Popovic, G. Hitz, J. Nieto, I. Sa, R. Siegwart, and E. Galceran. Online Informative Path Planning for Active Classification Using UAVs. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[138] S. Prentice and N. Roy. The Belief Roadmap: Efficient Planning in Belief Space by Factoring the Covariance. *Intl. Journal of Robotics Research (IJRR)*, 28(11-12):1448–1465, 2009.

[139] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[140] S. Quinlan and O. Khatib. Elastic Bands: Connecting Path Planning and Control. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 802–807, 1993.

[141] A. Ranganathan, M. Kaess, and F. Dellaert. Loopy SAM. In *Proc. of the Intl. Conf. on Artificial Intelligence (IJCAI)*, 2007.

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

[143] R. Ros, R. L. De Màntaras, J. L. Arcos, and M. Veloso. Team Playing Behavior in Robot Soccer: A Case-based Reasoning Approach. In *Proc. of the Intl. Conf. on Case-Based Reasoning*, pages 46–60, 2007.

[144] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram. Trajectory Modification Considering Dynamic Constraints of Autonomous Robots. In *Proc. of German Conf. on Robotics (ROBOTIK)*, pages 1–6, 2012.

[145] N. Roy. *Finding Approximate POMDP solutions Through Belief Compression*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2003.

[146] N. Roy and S. Thrun. Coastal Navigation with Mobile Robots. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, volume 12, pages 1043–1049, 1999.

[147] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Malaysia, Pearson Education Limited, 2016.

[148] R. Schirmer, P. Biber, and C. Stachniss. Efficient Path Planning in Belief Space for Safe Navigation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[149] R. Senanayake, S. O'Callaghan, and F. Ramos. Learning Highly Dynamic Environments with Stochastic Variational Inference. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[150] D. Silver, A. Bagnell, and A. Stentz. High Performance Outdoor Navigation from Overhead Data Using Imitation Learning. *Proc. of Robotics: Science and Systems (RSS)*, 2008.

[151] D. Silver and J. Veness. Monte-Carlo Planning in Large POMDPs. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, 2010.

[152] A. Somani, N. Ye, D. Hsu, and W. S. Lee. DESPOT: Online POMDP Planning with Regularization. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, 2013.

[153] J. R. Souza, R. Marchant, L. Ott, D. F. Wolf, and F. Ramos. Bayesian Optimisation for Active Perception and Smooth Navigation. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2014.

[154] C. Sprunk, G. D. Tipaldi, A. Cherubini, and W. Burgard. Lidar-based Teach-and-repeat of Mobile Robot Trajectories. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3144–3149, 2013.

[155] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2010.

[156] C. Stachniss and W. Burgard. An Integrated Approach to Goal-directed Obstacle Avoidance under Dynamic Constraints for Dynamic Environments. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 508–513, 2002.

[157] C. Stachniss and W. Burgard. Mobile Robot Mapping and Localization in Non-Static Environments. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, pages 1324–1329, 2005.

[158] C. Stachniss, C. Plagemann, and A.J. Lilienthal. Gas Distribution Modeling using Sparse Gaussian Process Mixtures. *Autonomous Robots*, 26:187ff, 2009.

[159] I. Şucan, M. Moll, and L. Kavraki. The Open Motion Planning Library. *IEEE Robotics and Automation Magazine (RAM)*, 19(4):72–82, 2012.

[160] B. Suger, B. Steder, and W. Burgard. Terrain-Adaptive Obstacle Detection. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[161] Y.T. Tan, A. Kunapareddy, and M. Kobilarov. Gaussian Process Adaptive Sampling using the Cross-Entropy Method for Environmental Sensing and Monitoring. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[162] S. Thrun. Learning Metric-Topological Maps for Indoor Mobile Robot Navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[163] P. Trautman, J. Ma, R. Murray, and A. Krause. Robot Navigation in Dense Human Crowds: The Case for Cooperation. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 2153–2160, 2013.

[164] V. Tresp. Mixtures of Gaussian Processes. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, 2001.

[165] F. Tsang, R. Macdonald, and S. Smith. Learning Motion Planning Policies in Uncertain Environments through Repeated Task Executions. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[166] C. Urmson, J. Anhalt, A. Bagnell, C. Baker, R. Bittner, M.N. Clark, J. Dolan, D. Duggins, T. Galatali, Chris C. Geyer, et al. Autonomous Driving in Urban Environments: Boss and the Urban Challenge. *Journal of Field Robotics (JFR)*, 25(8):425–466, 2008.

[167] R. Valencia and J. Andrade-Cetto. Path Planning in Belief Space with Pose SLAM. In *Mapping, Planning and Exploration with Pose SLAM*, pages 53–87. Springer, 2018.

[168] A. Viseras, D. Shutin, and L. Merino. Online Information Gathering Using Sampling-based Planners and GPs: an Information Theoretic Approach. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[169] O. Vysotska and C. Stachniss. Improving SLAM by Exploiting Building Information from Publicly Available Maps and Localization Priors. *Photogrammetrie – Fernerkundung – Geoinformation (PFG)*, 85(1):53–65, 2017.

[170] R. Wang, M. Veloso, and S. Seshan. Active Sensing Data Collection with Autonomous Mobile Robots. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2016.

[171] S. Weiss, M. Achtelik, S. Lynen, M. Chli, and R. Siegwart. Real-Time Onboard Visual-Inertial State Estimation and Self-Calibration of MAVs in Unknown Environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2012.

[172] P. Whaite and F. Ferrie. Autonomous Exploration: Driven by Uncertainty. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 19(3):193–205, 1997.

[173] D. Wolf, G. Sukhatme, D. Fox, and W. Burgard. Autonomous Terrain Mapping and Classification Using Hidden Markov Models. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2005.

[174] K.M. Wurm, R. Kümmerle, C. Stachniss, and W. Burgard. Improving Robot Navigation in Structured Outdoor Environments by Identifying Vegetation from Laser Data. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[175] J. Yedidia, W. Freeman, and Y. Weiss. Generalized Belief Propagation. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, 2001.

[176] B. Zhan and C. Noon. Shortest Path Algorithms: An Evaluation Using Real Road Networks. *Transportation Science*, 32(1):65–73, 1998.

[177] B. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, A. Bagnell, M. Hebert, A. Dey, and S. Srinivasa. Planning-based Prediction for Pedestrians. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3931–3936, 2009.

[178] F. Zimmermann, C. Eling, and H. Kuhlmann. Empirical Assessment of Obstruction Adaptive Elevation Masks to Mitigate Site-dependent Effects. *GPS Solutions*, 21(4):1695–1706, 2017.

[179] M. Zucker, J. Kuffner, and A. Bagnell. Adaptive Workspace Biasing for Sampling-based Planners. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 3757–3762, 2008.

List of Figures

List of Tables

List of Algorithms