UNIVERSITÄT BONN

ROBOT NAVIGATION IN CLUTTERED ENVIRONMENTS

DISSERTATION

zur Erlangung des Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

von
PETER REGIER
aus
Semipalatinsk, Kasachstan

Bonn, September 2021

Angefertigt mit Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

Erste Gutachterin:	Prof. Dr. Maren Bennewitz Rheinische Friedrich-Wilhelms-Universität Bonn
Zweiter Gutachter:	Prof. Dr. Jürgen Gall Rheinische Friedrich-Wilhelms-Universität Bonn
Tag der Disputation:	22. Dezember 2021
Erscheinungsjahr:	2022

Dedicated to my children, without whom this thesis would have been finished two years earlier. I love you with all my heart.

ABSTRACT

Service robots are designed for non-industrial use to help people at home and in public spaces. Today, service robots perform a variety of tasks that range from distributing medical supply to clean the floor. The wide range of possible application and the complexity of service robots spark major research interest. The goal of the multidisciplinary research is to increase the autonomy of service robots. The capability to navigate in the environment is fundamental for service robots. In this thesis, we present new approaches for robot navigation in challenging indoor scenarios with little space to maneuver, many objects, and crowds of people. We collectively call such scenarios as 'cluttered'. The presented contributions increase the efficiency of robot navigation and allow for new capabilities of the robot to solve challenging problems. Initially, we introduce a method to incorporate clutter in to the navigation process. We extend the state of the art navigation cost function by considering the configuration and quantity of object in the vicinity of the robot. The result is a foresighted robot navigation behavior, that leads around the clutter when it is beneficial for the robot. The second approach predicts the time the robot needs to complete a navigation task, based on a 2D path. The estimation of time is an important feature for service robots to schedule their tasks, e.g., guiding groups in a museum. Unfortunately, due to the lack of a dynamic model the completion time is a priori unknown. Therefore we train a regression model that reliably predicts the completion time based on 2D path features. To achieve human-aware navigation through pedestrian crowds, we apply the social force model (SFM) to control the robot. Our new approach reduces the collision rate with pedestrians of the SFM controlled robot, while maintaining similar velocities. The method considers a set of motion commands and evaluates the outcome, by simulating the corresponding situation into the future. Since such a omniscient robot control is not feasible to use in a real world scenario, we train a network with the best evaluated control command from the simulation. Our method outperforms the standard control with the SFM and the network successfully mimics the improved omniscient behavior, but considers information that is available to the robot in a real world scenario. In the next approach, the robot learns a navigation policy, through reinforcement learning (RL). Self-learning approaches have the potential to reduce the amount of parameter tuning, that is required to operate a robot. The endeavor of tuning is time consuming and know-how-intensive. To reduce the workload in this context, we successfully apply RL for the task of learning a navigation policy from scratch. The learned policy is capable to

reach the target location faster than the state of the art approaches, by optimizing the behavior when navigating close to obstacles. In some situations efficient and collision-free navigation is not enough to reach the goal, e.g., when the path is blocked by an object. To target those scenarios, our final approach combines object classification, fast 2D grid-based path planning, manipulation, and footstep planning to overcome objects by stepping over it or moving it to free the path to the goal. Our method is able to run on a small humanoid robot and finds paths through regions where traditional motion planning methods are not able to calculate a solution or require substantially more time. All navigation techniques presented in this thesis were thoroughly evaluated in various experiments. Our approaches advance the state of the art towards autonomous robot navigation in cluttered scenarios.

KURZFASSUNG

Serviceroboter dienen der Nutzung zu Hause oder an öffentlichen Plätzen und zeichnen sich durch ihre enormen Einsatzmöglichkeiten aus. Schon heute helfen uns Roboter beim Staubsaugen oder Rasenmähen. Sie agieren als mobile Auskunftsplattform in Museen und Einkaufszentren. Ihre Vielfältigkeit und die Komplexität der gestellten Aufgaben wecken das Interesse von Wissenschaftlern weltweit. Im Mittelpunkt der multidisziplinären Forschung steht die Autonomie der Robotern. Damit ein Roboter möglichst autonom mit seiner Umwelt interagiert, ist die eigenständige Navigation fundamental. In dieser Arbeit zeigen wir neue Methoden für die Roboternavigation in problematischen Scenarien. Vor allem untersuchen wir das Navigationsverhalten von Robotern durch Menschenmengen und in engen Umgebungen mit vielen Objekten auf dem Boden. Diese lassen nur wenig Platz zum Manövrieren oder versperren den Weg zum Zielpunkt komplett. Im Folgenden kategorisieren wir solche Scenarien als "cluttered"(Engl. für Gerümpel). In unserem ersten Beitrag erweitern wir die Perzeption von Robotern, um explizit Objektgruppen in der Umgebung während des Navigationsprozesses zu erkennen. Das Ergebnis ist ein vorausschauendes Roboternavigationsverhalten, das schwierig zu befahrende Bereiche umgeht, wenn somit das Ziel schneller erreicht wird. Eine weitere wichtige Eigenschaft von Servicerobotern ist das Abschätzen der Fahrtdauer bis ein Ziel erreicht ist. Die Fahrtdauerschätzung ist besonders wichtig bei der Zeitplanung von sequentiellen Aufgabenstellungen, wie zum Beispiel der Museumsführung und dem Medikamententransport in Krankenhäusern. Normalerweise ist die Fahrtdauer während der Pfadplanungsphase unbekannt, aufgrund von Ungenauigkeiten in der Steuerung des Roboters durch Schlupf der Räder und verrauschte Messungen der Sensoren. In unserer Arbeit werden verschieden Regressionsmodelle anhand von mehreren Pfadmerkmalen gelernt und verglichen. Die Ergebnisse zeigen eine deutlich bessere Schätzung der Ankunftszeit anhand der eingeführten Pfadmerkmale als die Schätzung, die nur auf der Pfadlänge basiert. Ein weiterer Teil dieser Arbeit befasst sich mit der Navigation von Robotern durch Menschenmengen. Wir präsentieren eine Methode, die auf dem Social Force Modell (SFM) basiert. Das SFM wird benutzt, um die Dynamiken, Interaktion und Verhalten von Passanten zu simulieren. Es ist daher besonders gut geeignet, um ein für den Menschen intuitives Navigationsverhalten des Roboters zu realisieren. In unserer Arbeit verbessern wir das Verhalten des Roboters, basierend auf dem SFM. Unsere Methode benutzt ein allwissendes Simulationsverfahren um einen Datensatz mit optimierten

Steuerbefehlen aufzuzeichnen. Anschließend wird der Datensatz genutzt um ein neuronales Netz (NN) zu trainieren. Das trainierte NN übernimmt die Steuerung des Roboters und kann durch die gesammelten Daten aus der Simulation das Verhalten des Roboters hinsichtlich der auftretenden Kollisionen mit Passanten verbessern. Des Weiteren ermöglicht das NN ein Betrieb mit Daten, die dem Roboter in der realen Welt zur Verfügung stehen, und schließt somit die Lücke zwischen den Simulator und der Realität. Im nächsten Arbeitsschritt haben wir selbstlernende Methoden etabliert, um das so genannte Tuning zu umgehen. In der Robotik erfordert das Parameter Tuning Expertise und ist generell ein zeitaufwändiger und mühsamer Prozess. Selbstlernende Methoden, wie das bestärkende Lernen, ermitteln die beste Handlung für den jeweiligen Umgebungszustand durch eine Versuch-Und-Irrtum Vorgehensweise. Somit optimieren diese selbstlernende Methoden, die notwendigen Parameter indirekt. Wir trainieren das neuronale Netz, um die besten Geschwindigkeitsbefehle des Roboters zu erlernen. Die Steuerung des Roboters durch das trainierte Netzwerk erfordert kein Parameter Tuning und zeigt kürzere Fahrtzeiten auf als der Stand der Technik. In manchen Scenarien reicht effektive und kollisionsfreie Navigation nicht aus, um das Fahrtziel zu erreichen, wenn zum Beispiel der Weg durch ein Objekt versperrt ist. Bisherige Verfahren scheitern in solchen Scenarien oder sie sind nicht ausführbar auf einem mobilen Roboter in Echtzeit. Unsere Methode vereint die Objekterkennung, die Robotermanipulation und die Fußschrittplanung mit der zwei-dimensionalen Pfadplanung in einem Navigationsverfahren, um genau solche Situationen zu bewältigen. Somit ist es möglich eine schnelle zweidimensionale Pfadplanung mit den notwendigen Aktionen des Roboters durchzuführen, um ein Ziel zu erreichen. Alle vorgestellten Verfahren wurden eingehend experimentell evaluiert. Die vorliegende Arbeit erweitert den Stand der Technik bezüglich der autonomen Roboternavigation in komplexen und schwierigen Scenarien.

ACKNOWLEDGMENTS

As the famous African proverb says, "it takes a village to raise a child", and, in my case, it took half of Europe to write this thesis.

I would like to express my gratitude to my supervisor and friend Prof. Maren Bennewitz, for always believing in me. She made the Humanoid Robots Lab my second home, by giving opportunity to explore and develop my own ideas and providing guidance and motivation when it was most needed. I am thankful for her help and her crunch time ability, the whole team could rely on and many many other things. I will always remember the good times after the work was done and the countless valuable lessons in scientific writing and presenting.

I am grateful I had the opportunity to travel around Europe and work with so many remarkable people. I would like to thank Federico Boniardi from Freiburg, Richard Bormann from Stuttgart, and Agostino Stilli from London for our discussions and our legendary billiard sessions that made the project meetings so much more fun to attend. My thanks also go to Khiet P. Truong, Jaebok Kim and Cristina Zaga for the hospitality during my time in Twente. For their hospitality in Innsbruck, I thank Philipp Zech and Cornelia Vidovic. Many thanks to Thomas Bächle for the awesome cooperation in the field of media studies, which extend my view on robots, sensors and science in general. Probably one of the greatest and helpful colleagues one could possibly imagine is Stefan Oßwald. I am thankful for all the things I was able to learn from him and for all the help. I am grateful I had the opportunity to build our lab from scratch and to be one of the founding members of the lab together with Stefan and Philipp Karkowski. I would like to thank Nils Dengler, Christopher Gebauer, and especially Stefan Oßwald for proof reading this thesis. Many thanks to my first supervisor Till Schmitte for sparking my curiosity in science during my Bachelor studies and for inspiring me to take this journey.

Finally I want to mention my family. Thank you for always supporting me and having so much patience with me. My parents and my sister with her family are always there when the times got tough and I am really grateful for that. Most importantly I want to express my gratitude to my wife Friederike Regier. Her kindness, patience and love keeps me always going. She is my world that gives meaning to all the things I am doing.

CONTENTS

1 INTRODUCTION

Daily, people perform dull and boring tasks, that range from cleaning the house to running errands through crowds of people. Service robots can potentially reduce the amount of work and time people spend every day to accomplish these tasks, by either aiding humans or acting completely autonomously. The current generations of robots can accomplish a vast variety of jobs and be applied in different scenarios. To illustrate, several successful robotic products are listed in Fig. 1.1. The iRobot Roomba is a robotic vacuum cleaner, that already helps millions of households to keep the floor clean. Toyota developed the Human Support Robot (HSR) for scenarios at home, to help handicapped or ill people. The MiR100 is delivering sterile and medical goods in a hospital, during the COVID-19 pandemic. Finally, the Nao humanoid robot is a multi-purpose platform, that can be used for education, entertainment, and as in our case, for research purpose. All mentioned examples require mobility of the robot and the ability to navigate autonomously in the environment. Autonomous navigation is the robot's capability to reach a target location in the environment with none or minimal human intervention. It is a multidisciplinary field and includes localization, mapping, perception, object and human detection, and motion and path planning.

Robots came a long way since George Devol and Joseph Engelberger developed the first industrial robot in 1959 [1]. Today, more robots are deployed than ever. The International Federation of Robotic [2] reports a positive trend for the industry. In the past decade the stock of operational industrial robots increased almost threefold from 1 million units in 2009 to 2.7 million in 2019. In recent years, service robots entered the market. Service robots are intended for non-industrial use and are employed in households and public spaces. In 2019 23.2 million units were sold and the number is expected to rise to 53.3 million by the year 2023.

Notably, this bright economical prospects and the potential of numerous new applications spark major research interest in the scientific community. Today, teams of scholars can compete against each other in many robot competitions, e.g., the RoboCup@Home league [3] and The European Robotics League Consumer Service Robots competition [4]. Those events bolster the progress towards future service and assistive technology for domestic applications. The goal is to test the abilities and performance of service robots in non-standardized home environments. The following examples show the achievements made in the RoboCup@Home league since the start in 2006 till 2019 [5].

(a) Roomba vacuuming the floor.

(b) HSR supporting people in need.

(c) MiR100 distributing sterile goods.

(d) Nao picking up toys.

Figure 1.1: Robots operating in human environments. a)The iRobot Roomba vacuum cleaner is widely used in many households (source: [6]). b) Human Support Robot (HSR) designed by Toyota for collaborative task in a household environment (source: [7]). c) The MiR100 model developed by Mobile Industrial Robots is already deployed in a hospital for automated logistic tasks (source: [2]). d) The educational and research robot NAO from SoftBank Robotics is also used for entertainment and information purposes (source: [8]).

The robots competition task to navigate from one room to another was first evaluated based on the collisions occurred with the environment. Later every collision meant the failing of the task. Nowadays functional touching while navigating is even encouraged, e.g., to open doors. Initially, object recognition was accomplished by attaching a well recognizable marker to the object and is now realized without markers by using color-segmentation and machine learning methods. While the people recognition task in the past consisted of recognizing a face in front of the robot, it is now evolved to state the gender and the pose of a specific person within a crowd. The difficulty of the manipulation task increased from grasping a simple box on the floor to open a beer can and handing it to a person.

These examples show how constant research and development enables a higher degree of autonomous behavior, and thus pave the way for new robotic applications. Accordingly, our work aims at advancing the state of the art navigation approaches towards a more foresighted and autonomous behavior for service robots. We explore how robots

can efficiently solve navigation tasks in human environments. More precisely, we target overfilled areas with many objects on the floor or narrow hallways with crowds of people and collectively designate this characteristic as 'clutter'. To illustrate the required navigation capabilities, let us assume a service robot, that assists a family. After receiving a task of buying some groceries in the store the robot has to traverse the living area, where objects are occupying the floor. It has now to determine the best way to overcome the obstacles on its way outside, by either getting around the clutter or manipulating the objects to free the path. To deliver the groceries, the robot has to move through human crowds, while being unobtrusive and human-friendly. While doing so, the robot has to update the human at home about the expected time of delivery of the groceries. Since the software to accomplish all the aforementioned challenges has to run preferably on a mobile platform, the algorithms have to be as efficient as possible and real time capable. Furthermore the interfaces and software, to operate the robot, should be usable by customers with any kind of technological know-how.

This thesis presents approaches, to successfully accomplish the outlined steps above. In summary, we present techniques to answer the following questions:

- How can we improve the navigation of the robot in cluttered environments, by considering the clutter distribution?
- How can we predict the robot's travel time based on the global path?
- How can we improve navigation through pedestrian crowds, by using prediction and machine learning techniques?
- How can we optimize the navigation behavior of the robot near obstacles with reinforcement learning?
- How can a robot efficiently reach a navigation goal, that is blocked by obstacles?

1.1 MAIN CONTRIBUTIONS

This thesis presents novel solutions to navigation problems in tough, narrow, and cluttered environments, where there are many objects lying on the floor or pedestrians crossing the robot paths. Our contributions improve the state of the art robot navigation, by reaching faster the goal location or having less collisions. All approaches consider real world scenarios for potential service robots. More importantly, the developed software is capable to run on a mobile robot with limited computational capabilities and narrow field of view (FOV). This section provides an overview of the contributions.

The first contribution in Chapter 3 presents an approach to efficiently navigate in cluttered environments, with many static objects on the floor. We propose an extension of the standard navigation cost, that influences the global path planning, to create faster routes to the goal. The standard navigation cost function computes the navigation cost based on the distance to the closest obstacle. Our approach, instead, considers the complete object distribution in the vicinity of the robot and allows to propagate the navigation cost into unexplored regions, accounting for potential objects beyond the perceived area. The result is a foresighted robot navigation behavior that leads around cluttered areas when it is beneficial for the navigation task and performs equal in the absence of clutter.

In Chapter 4, we introduce a method to predict travel time based on global path characteristic. We compare different regression methods to train an estimator with data gained from the simulation of the actual path execution with a controller that is based on the dynamic window approach (DWA). The prediction of the completion time, the period the robot needs to complete a navigation task, is pivotal for time-efficient planning in different scenarios, e.g., medical supply distribution or rescue situation. All this tasks involve locomotion and the travel time can make up a significant amount of the total task completion time. Furthermore, when multiple path candidates are present, the shortest path is not always the best choice as it may lead through narrow gaps and it may be in general hard to follow due to a lack of smoothness (Chapter 3). The assessment of an estimated completion time is a much stronger selection criterion compared to the bare path length, but due to the lack of a dynamic model in the path computation phase the completion time is typically a priori not known. We show that a non-linear regression model is well suited to reliably predict the completion time of a navigation task from 2D path features.

The third contributions focuses on navigation through pedestrian crowds. A popular tool to simulate human motion and crowd behavior is the social force model (SFM). In Chapter 5, we apply the SFM to control also the robot instead of the DWA, in order to achieve human-aware navigation behavior. The robot's objective is to reach the target location with minimal collisions and as fast as possible, while passing crowds of people. Our approach determines the best acceleration command by simulating the pedestrians as well as the robot motion forward into the future and evaluating the predicted situation. In this phase we collect the relevant environment data to train a network controller to mimic the navigation policy achieved through the omniscient simulation. In extensive experiments using different pedestrian densities, we demonstrate that the controls generated by the learned neural network lead to a significantly reduced number of robot collisions with pedestrians compared to the results of the basic

SFM controller, while achieving similar or even shorter completion times.

In Chapter 6 we also apply a neural network, this time, to learn a navigation policy by deep reinforcement learning (RL). RL applies the trial-and-error principle to determine an optimal policy, that does not need any tuning. Typical navigation systems require manual parameter tuning to achieve a good navigation behavior. This tuning takes a significant amount of time and profound knowledge about the navigation software, the robot hardware, as well as the environment conditions. Furthermore, the parameter setup is not optimal for all situations, but rather aims for a good trade off between time efficiency and safety. Our method combines path planning on a 2D grid with reinforcement learning and does not need any supervision (as opposed to the approach described in Chapter 5). The experiments illustrate that our trained policy can be applied to solve complex navigation tasks. Furthermore, the robot controlled by our learned policy reaches the goal significantly faster compared to the DWA, by closely bypassing obstacles and thus saving time.

So far, the thesis focuses on increasing the efficiency of the robot's navigation behavior assuming a free path to the goal. However, this assumption is not necessary given in a cluttered environment, e.g., when objects block the path to the target location. In Chapter 7 we present a solution for the blocked path scenario by combining fast 2D path planning with 3D footstep planning and object manipulation actions, to overcome obstacles in obstructed regions. In our work we classify the objects in the environment based on color images, to decide whether the object can be pushed away, be carried aside, or in case of a humanoid robot, can even be stepped over. Based on associated actions, we compute the navigation cost and create a cost grid, to perform efficient 2D path planning. The resulting path encodes the necessary actions, that need to be carried out by the robot to reach the goal. As the experiments demonstrate, using our convolutional neural network (CNN) the robot can robustly classify the observed obstacles into the different classes and decide on suitable actions to find efficient solution paths. Our system finds paths also through regions where traditional motion planning methods are not able to calculate a solution or require substantially more time.

In summary, this thesis presents five contributions, that target difficult scenarios regarding robot navigation in clutter.

1.2 PUBLICATIONS

Parts of this thesis have been published in international journals and conference proceedings. The following list gives an overview about the individual publications.

- Chapter 3

  P. Regier, S. Oßwald, P. Karkowski, *et al.*, "Foresighted navigation through cluttered environments," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016

- Chapter 4

  P. Regier, M. Missura, and M. Bennewitz, "Predicting travel time from path characteristics for wheeled robot navigation," in *Proc. of the Europ. Conf. on Mobile Robotics (ECMR)*, 2017

- Chapter 5

  P. Regier, I. Shareef, and M. Bennewitz, "Improving navigation with the social force model by learning a neural network controller in pedestrian crowds," in *Proc. of the Europ. Conf. on Mobile Robotics (ECMR)*, 2019

- Chapter 6

  P. Regier, L. Gesing, and M. Bennewitz, "Deep reinforcement learning for navigation in cluttered environments," in *Proc. of the Intl. Conf. on Machine Learning and Applications (CMLA)*, 2020

- Chapter 7

  - P. Regier, A. Milioto, P. Karkowski, *et al.*, "Classifying obstacles and exploiting knowledge about classes for efficient humanoid navigation," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2018
  - P. Regier, A. Milioto, C. Stachniss, *et al.*, "Classifying obstacles and exploiting class information for humanoid navigation through cluttered environments.," *The Int. Journal of Humanoid Robotics (IJHR)*, 2020

PUBLICATIONS NOT COVERED BY THIS THESIS

The following publications were written while employed as a research associate, but are not covered by this thesis:

- T. C. Bächle, P. Regier, and M. Bennewitz, "Sensor und Sinnlichkeit. Humanoide Roboter als selbstlernende soziale Interfaces und die Oboloszenz des Impliziten," *Navigationen-Zeitschrift für Medien- und Kulturwissenschaften*, 2017

- J. Kim, I. H. Shareef, P. Regier, *et al.*, "Automatic ranking of engagement of a group of children „in the wild "using emotional states and deep pose features," in *Proceedings of the workshop on 'Creating Meaning with Robot Assistants: The Gap Left by Smart Devices' at the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*, 2017

- C. Thimm, P. Regier, I. C. Cheng, *et al.*, "Die Maschine als Partner? Verbale und non-verbale Kommunikation mit einem humanoiden Roboter," in *Die Maschine: Freund oder Feind?* Springer, 2019
- T. C. Bächle, P. Regier, and M. Bennewitz, "Sensoren," in *Technikanthropologie*, Nomos Verlagsgesellschaft mbH & Co. KG, 2020

1.3 COLLABORATIONS

Parts of this thesis are the results of collaborative work with other researchers, hence we consistently use "we" within this thesis. The collaborations on the work in this thesis and my contributions are as follows:

- Chapter 3: This work has been carried out along with Philipp Karkowski and Stefan Oßwald. Philipp and Stefan helped with the real-world experiments set up. Stefan was also closely involved in the automation of the evaluation of the simulation experiments.
- Chapter 4: This approach was supervised additionally by Marcell Missura. Marcell helped with fruitful discussions and ideas during the development of the work.
- Chapter 5: This work was originally developed within the Master's thesis of Ibrahim Shareef. Ibrahim implemented the approach. I co-developed the idea, designed and performed the experiments and evaluation.
- Chapter 6: This work was originally developed within the Master's thesis of Lukas Gesing. I developed the project idea and supervised and advised Lukas during his work.
- Chapter 7: These two publications are the result of the cooperation with Andres Milioto. The project was supervised also by Cyrill Stachniss. Andres contributed the Bonnet framework for Semantic Segmentation in Robotics [9]. He performed the training and evaluation for the object classification part in this work and helped to integrate the Bonnet framework into the navigation scheme.

2

BASIC ROBOTIC CONCEPTS

This chapter summarizes the basic navigation concepts for mobile robots. We describe the deployed sensors and the navigation scheme, that are subject of this research. Furthermore we explain the environmental model, path and motion planning techniques, which are the groundwork for the methods presented in this thesis. We also recommend [10], for a more detailed analysis of robotics concepts in general. This book introduces the reader to the many techniques and algorithms in the field.

2.1 SENSORS

Robots rely on sensors to perceive the world analog to the human senses. In our experiments, we deploy cameras and Light Detection and Ranging (LiDAR) sensors to perform mapping, localization, obstacle detection, and object recognition.

Cameras are one of the most used sensors in robotics. In this context, it is common to name the different types of cameras based on the received sensor data. Accordingly, a RGB camera produces an array of color pixels represented by the primary colors red, green, and blue. In addition to the three color channels, a RGB-D (depth) camera provides also a grayscale image, that can be converted into per-pixel depth information. We use the depth information for localization and the RGB data for object classification. Furthermore, a RGB-D camera enables us to efficiently allocate color pixels to a location in the environment. We use this advantage in Chapter 7 to first detect and classify objects based on the color information and then use the depth information to allocate the object in the map. In general RGB-D sensors, such as the Intel RealSense Depth Camera D455 [11], are very popular in robotics, due to their light weight and affordability. However, it is not ideal to rely only on the RGB-D sensor, since the accuracy and precision might drastically be diminished when working under too much sunlight.

A LiDAR sensor is much better suited to perform localization, due to its wide field of view (FOV) that can cover a huge area in the environment. Additionally, the sparse data structure of a 2D LiDAR sensor speeds up the localization in comparison to the dense data of a depth image. LiDAR sensors in its simple form uses a series of laser pulses and measures the time of flight (TOF) when the corresponding signal returns back to its sender. The TOF is then

scaled to determine the distance to an obstacle. LiDAR sensors have a high precision, wide FOV, and approved safety (DIN EN ISO 13849) that is important for collision avoidance. In this work, we utilize 2D LiDAR sensors that perform measurements in one horizontal plane for computational cheap and fast obstacle detection, mapping, and localization (Chapter 3).

2.2 MAPS

To navigate successfully, many robotic applications require a map to model the environment. Among the most popular maps used in robotics are the feature-based maps and the grid maps [12]–[16].

The compact, but sparse, feature maps represent the location of landmarks and other significant characteristic, e.g., distinctive objects or shapes in the environments. Such type of maps are often used in outdoor navigation, where the robot has to cover a huge operational area. Due to the reduced feature representation of the environment, it is possible to speed up the localization process significantly and to store a huge mapped territory into memory space. However this type of map will underperform in regions with only few features, e.g., corridors and white walls in indoor environments. Moreover the feature computation from the sensor data is often expensive and thus hinders the real time capabilities of mobile robots [16].

A more suited representation of the environment, for indoor navigation and the developed approaches in this thesis, is the grid map. Grid maps are a dense representation of the environment and are widely used in robotics. The mapping approach divides the environment into small fix square cells, containing information about their property. Similar to the cameras (Sec. 2.1), we name the map based on the contained information. For example, an occupancy map cell stores the probability of being occupied by an obstacle. Based on the occupancy map, robots can perform path planning, localization and collision checks. However, considering only the occupancy state of the cell is not enough for some navigation scenarios, when other factors need to be taken into the account, e.g., social distance to pedestrians. Thus, the common practice today is to apply navigation cost to every grid cell, instead of the occupancy value. The cell navigation costs in the, so called, costmap reflect the difficulty to traverse the corresponding area in the environment.

Usually, those costs are determined by an objective function. Depending on the application, a navigation cost function can consider several factors, e.g., ground conditions, distance to obstacles, and social norms. However, a necessary step is the inflation of the map. It describes the process of enlarging every single occupied cell in the map by a certain amount, usually the robot radius. In robot navigation, it is common to reduce the robot footprint to a single cell. The

footprint reduction facilitate collision checks and path planning, by considering only one cell in the grid as the robot. However, to keep the configuration of free and occupied space of the map accordingly to the real world, the obstacles in the maps need to be inflated by the same amount as the footprint was reduced. In Chapter 6, we show how the inflation can be neglected by learning a navigation policy with reinforcement learning.

2.3 NAVIGATION SCHEME

To generate a wide range of intelligent behaviors, the different robotic modules for sensing, localization, planning, and motion execution need to be integrated into one working unit accordingly to a scheme. In this thesis we use a state of the art navigation scheme, that follows a two-step approach (Fig. 2.1). Before the start of the navigation process, we create a map of the environment with a simultaneous localization and mapping (SLAM) method [17], [18]. The map represents the robot's prior knowledge of the environment and is updated based on the on-board sensor information during the navigation process.

In the navigation scheme the first stage is dedicated to plan a spatial global path through the environment from the position of the robot to the goal location. Depending on the distance and the environment this step is usually computationally expensive. Thus, the global path planner operates in low frequency range of 0.5 to 2Hz. To smoothly follow the globally computed 2D path afterwards, one typically employs a local reactive collision avoidance system that efficiently generates motion commands for the robot [19]–[21]. It takes into account the current sensor information and the subgoal provided by the global planner module. Reactive collision avoidance modules work with high frequency up to 50 Hz, to react to unforeseen events, e.g., people crossing the robots path.

In the following sections, we introduce state of the art methods to compute a global path on the grid map and subsequently motion commands to follow the path. In our contributions we show how this method can be improved and extended.

2.4 GLOBAL PATH PLANNING

A common approach to find a global path in the environment is to apply graph traversal algorithms on a grid map. For that reason, the grid map needs to be transformed into a graph, by considering every grid cell as a graph node, that is connected with edges to the adjacent cells.

In this thesis, we deployed the A* search [22], to find a path on a grid map. The A* algorithm manages an *Open list* and a *Closed*

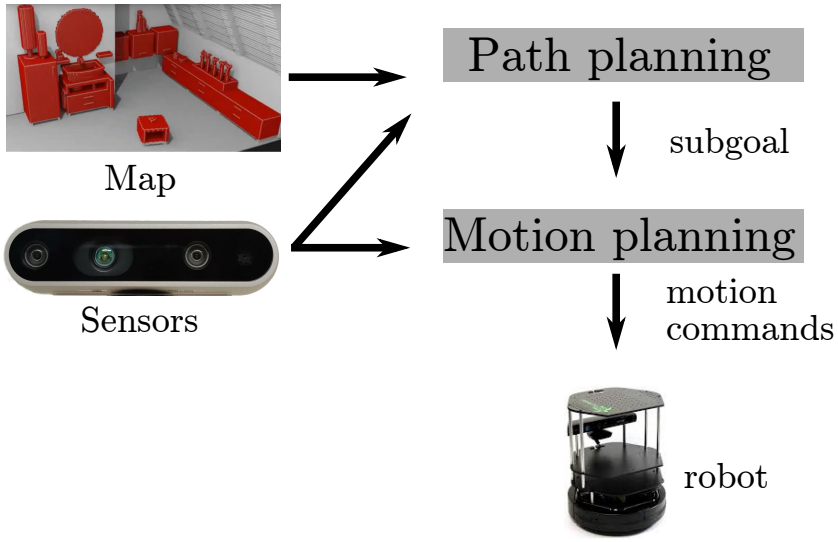


Figure 2.1: Two stage navigation approach. Before starting the navigation, the robot requires a map of the environment, which is continuously updated based on new sensor information. In the first stage, the robot computes a path to a given target location by using the grid map (Sec. 2.4). In the second phase, the robot extracts a subgoal from the global path and determines the motion commands for the next time step, to follow the path (Sec. 2.5).

list, to determine a minimum-cost path from a start node to a goal node. At the beginning of the algorithm, the *Closed list* is empty and the *Open list* is initialized with the start node. In every iteration the algorithm expands the most promising node and moves it into the *Closed list*, while its successor nodes are moved to the *Open list*. The search is complete when the goal node is selected for expansion. The expanding order of the nodes is based on the evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is the best path cost currently known from the start to node n, and h(n) is a heuristic, that estimates the cost from node n to the goal. The heuristic function $h(n)$ determines the performance of A*, by directing the search towards the goal. $h(n)$ is called admissible, when it never overestimates $h^*(n)$, that is the true optimal cost from node n to the goal node. $h(n)$ is consistent, when $h(n) \leq ec(n, n') + h(n') \forall n$ and n', where $ec(n, n')$ is a function, that returns the edge cost between the nodes n and n'. If $h(n)$ is consistent, then the cost of $g(n)$ is guaranteed to be optimal and every node is expanded only once. Note that consistency implies admissibility, and non-admissibility implies inconsistency.

When the heuristic is not consistent, A* can potentially find a better solution for an already expanded node. Then the new costs $g(n)$ of node n need to be propagated to its successors, by fetching the node from the *Closed list* and register it, with the new updated cost, in the *Open list*. In this case, it is possible a node will be expanded several

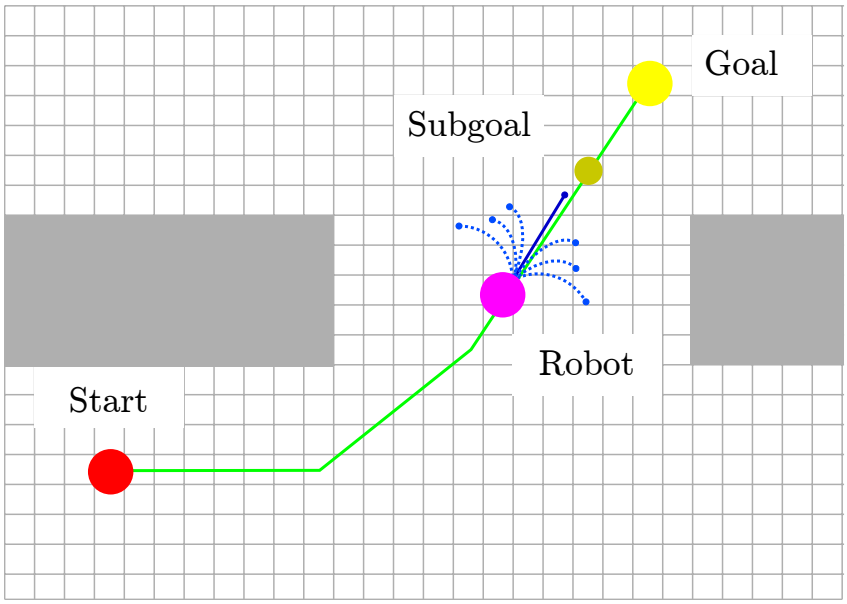


Figure 2.2: Example of the robot (magenta) planning a global path (green) on the grid map from start (red) to the goal (yellow). While following the path to the goal, the robot repeatedly computes the trajectories (dashed blue lines) for a set of motion control commands and chooses the best trajectory (solid blue line) based on the surroundings. In this example, the best trajectory leads the robot close to the subgoal (dark yellow), while keeping distance to obstacles (gray).

times. In Appendix A, we list the general version of the algorithm for non-admissible heuristic (Algorithm 2). To obtain the path after the A* search is terminated, we need to backtrack the *Parent map* from the goal node to the start node (Algorithm 1).

After the path computation, the robot needs to find the right sequence of robots velocity commands to follow the path to the goal. In the next section we introduce a state of the art method to accomplish this task.

2.5 MOTION PLANNING

After the path computation, the robot has to compute a sequence of motion commands, to navigate collision-free and as fast as possible to the target location. Trajectory rollout (TR) [23] algorithms are particularly good at achieving this task. TR algorithms sample motion control commands and predict their out come into the future, while considering the robot's state of motion.

In more detail, the algorithm determines a set of motion control commands, considering the robot's acceleration and velocity limits, as well as the robot's current velocity. Then, the trajectory of the robot is computed, for each control command in the set, assuming

constant control parameters. Afterwards, every sampled trajectory is evaluated by an objective function and the control commands of the best trajectory are applied to the robot for the next time step. This procedure is repeated every time step until the robot reaches the goal.

Several approaches, e.g., [21], [23] follow this pattern to achieve reactive collision avoidance. The Robot Operating System (ROS) [24] provides a tool set for all kinds of robotic application. This includes the aforementioned approaches [25]. In practice, both approaches show comparable results. We deployed the dynamic window approach (DWA) [21], because it showed slightly better performance in cluttered environments. Initially, the DWA assumes a unicycle model and samples linear and angular velocity control commands, but the approach can easily be modified to fit other kinematic models. It also works for holonomic robots by expanding the search and control space and allowing linear motion to the side of the robot. Furthermore, some research [26] propose to apply acceleration commands to control the robot. In this thesis however, we focus on the navigation cost function and machine learning to improve the robots navigation. Fig. 2.2 outlines the global path and motion planning on a grid map, for a better understanding.

2.6 MACHINE LEARNING

The ability to learn allows robot to acquire new skills, to carry out complex tasks, and to obtain problem solutions, that are difficult to formulate. The goal of this section is to introduce the principles and terminology of machine learning. A more detailed coverage of machine learning methods in general can be found in [27]–[29].

When applying machine learning, we want to model the relationship

$$\boldsymbol{y} = \boldsymbol{f}_{\mathcal{M}}(\boldsymbol{x}) \tag{2.1}$$

between the input values $\boldsymbol{x} \in \mathcal{X}$ and the output values $\boldsymbol{y} \in \mathcal{Y}$, where $\mathcal{X}$ is the input space, $\mathcal{Y}$ is the target space, and $\boldsymbol{f}_{\mathcal{M}}$ is a mapping function. The problem of estimating $\boldsymbol{f}_{\mathcal{M}}$ is called regression if $\mathcal{Y}$ is continuous and the relationship between the input and output is deterministic. We apply regression methods in Chapter 4 to estimate the completion time of a navigation task based on path features. However, robots are often influenced by noise and uncertainties, that introduces randomness to the model. In that case, the output $\boldsymbol{y}$ becomes a random variable and the dependency between input and output is given by the conditional probability distribution

$$p(\boldsymbol{y}|\boldsymbol{x}), \tag{2.2}$$

that defines the probabilities of occurrence for possible outcomes of $\boldsymbol{y}$ given $\boldsymbol{x}$. The problem of obtaining the probability distribution over

a finite set of classes k of the target space $\mathcal{Y} = \{1, \ldots, k\}$ is called classification. Often we are interested only in the most likely class, i.e.,

$$\hat{\boldsymbol{y}} = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \; p(\boldsymbol{y}|\boldsymbol{x}). \tag{2.3}$$

In this thesis, we deploy two types of learning, supervised learning and reinforcement learning, to model the functionality between input and output values. Supervised learning algorithms approximate the target function based on a data set of labeled examples. Each example contains the input vector and the desired output vector (label). During the training the parameters of the model are tuned based on the training set in the data set. The goal of the training is to create a model that generalizes its responses and acts correctly for input data not present in the training set, thus, after the training, the model is evaluated based on a test set. Reinforcement learning (RL), instead, tries to find the correct action of a robot for a specific situation, where no labels of an optimal output are given. Hence, RL tries to find the right policy through an trial-and-error process, where the robot's actions are rated by a reward feedback signal. While supervised learning requires a teacher or examples in order to learn $\boldsymbol{f}_{\mathcal{M}}$, RL requires an agent, e.g., a robot that can perceive the state of the environment and take actions to affect that state. Thus, supervised learning can only learn the information contained in the data set, while RL-agents can explore the state space to learn the correct behavior based on the gathered experience. In the following, we chose the learning methods based on the problem we are trying to solve. For example, in Chapter 7 it is convenient to provide a data set with examples of toy classes in an image, because it is an easy task for humans to do. Accordingly, we chose supervised learning to solve this task. While in Chapter 6, it is very difficult to implement a navigation behavior that acts optimal near obstacles. Therefore, it is better to apply RL, to let the robot learn the optimal navigation policy through experience.

3

FORESIGHTED NAVIGATION THROUGH CLUTTER

In this chapter, we introduce an approach to efficient robot navigation through cluttered indoor environments. We propose to estimate local obstacle densities based on already detected objects and use them to predict traversal costs corresponding to potential obstacles in regions not yet observable by the robot's sensors. By taking into account the predicted costs for path planning, the robot is then able to navigate in a more foresighted manner and reduces the risk of getting stuck in cluttered regions. We thoroughly evaluated our approach in simulated and real-world experiments. As the experimental results demonstrate, our method enables the robot to efficiently navigate through environments containing cluttered regions and achieves significantly shorter completion times compared to a standard approach not using any prediction.

3.1 INTRODUCTION

Today, mobile robots are able to localize and navigate in laboratory environments that are nicely tidied up, static, and have an accurate, up-to-date world representation such as a floor plan or a pre-recorded SLAM map. When robots step out of the laboratory, however, they face an ever-changing world filled with piles of objects and moving obstacles that are not contained in the static environment representation. In particular, service robots operating alongside humans in daily-life domestic and office environments need to navigate efficiently and robustly also through cluttered scenes, for example in children's rooms with toys scattered on the floor, in workshops with tools laying around, or in storage rooms with piles of boxes.

Driving through clutter is challenging as it requires accurate sensing of obstacles and precise motion execution. The robot has to drive slower, turn frequently, replan a collision-free path when new obstacles come into its field of view, backtrack and take a detour when it gets stuck in a dead end, execute dangerous maneuvers such as driving backwards, and may eventually even have to give up when it is unable to find a suitable path back out of the cluttered area.

Hence, the shortest path planned on a given floor plan is not necessarily the most efficient path, as taking a detour around cluttered areas right from the beginning may be longer, but faster and safer to

execute. Consider, for example, the scene depicted in Fig. 3.1. If the robot takes the green path, it can drive fast and steadily to the goal without encountering obstacles, which is more efficient than taking the red path and potentially getting stuck.

If the world was fully observable and the distribution of clutter was known, the robot would be able to plan a path that is optimized for efficiency by reasoning about safety distances, allowable velocities, and probabilistic sensing and motion execution errors. While solving this problem is already challenging, accurate planning is impossible when the world is only partially observable. Due to limited sensor ranges and occlusions from the robot's point of view, the true distribution of clutter is unknown to the robot, hence it cannot directly plan and optimize the planned path for efficiency.

Traditional planners can only consider objects that are either registered in the given map of the environment, or have already been observed by the sensors, so these planners would assign equal traversal costs to all unobserved areas in the free space of the given map. However, clutter is typically not spread uniformly. In a children's room, toys often come in piles. In a workshop, tools usually gather around the workbench. In storage rooms, boxes are usually stacked in heaps close to each other. Using knowledge about the distribution of clutter allows for predicting cluttered areas in regions that the robot has not yet observed, which enables the robot to plan its path in a more foresighted manner.

In this chapter, we propose a method to predict the occurrence of obstacles in the space outside the field of view from the information about objects in already observed areas. By increasing the costs for traveling through areas where obstacles are expected, we allow a path planner to avoid cluttered areas, which leads to more foresighted navigation. Since the world is only partially observable, there is no guarantee that the predicted obstacles actually exist and that the planned path is optimal with respect to efficiency, but it seems reasonable to avoid difficult navigation challenges if easier and safer options are available.

We implemented our system in the Robot Operating System (ROS) [24] and thoroughly evaluated it both in simulated and real-world experiments with a Robotino robot from Festo Didactics [30]. As the experimental results demonstrate, our approach enables the robot to react to unexpected objects in a foresighted manner and to navigate efficiently. In comparative experiments with a traditional path planner, our method achieves significantly shorter completion times in various complex scenarios.

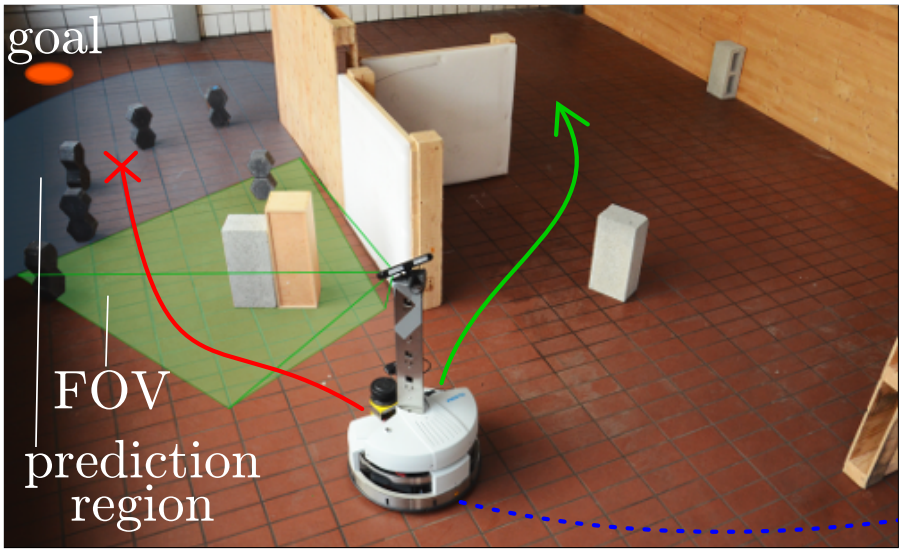


Figure 3.1: A robot navigating through a cluttered environment has to choose between different paths for navigating to a goal. While the red path is shorter, it passes through a cluttered area, putting the robot at risk of getting stuck. The green path is longer, but it is safer. Our approach predicts traversal costs corresponding to potential obstacles outside the field of view (FOV) based on already detected objects. This enables the robot to choose more promising paths that are likely to lead to a shorter completion time.

3.2 RELATED WORK

Lu *et al.* introduced the notion of *layered costmaps* and implemented it in ROS [31]. Instead of maintaining a single costmap, the authors proposed to split the information of the costmap into several layers with different semantics. Each layer represents a different type of obstacle or constraint, such as the static map, caution zones, and the personal space of a human that the robot should not penetrate. This concept has been widely used, such as for path planning or in order to represent the exploration progress. Marder-Eppstein *et al.* [32] suggested to exploit the costmap representation for navigating a PR2 robot through a cluttered real-world office environment containing obstacles of varying shapes and sizes. While their navigation system tries to navigate even through difficult obstacle fields, our approach predicts the scene behind initially detected obstacles and considers taking a detour right from the start if navigating through the clutter does not look promising.

Hornung *et al.* [33] considered 3D environments containing a moderate amount of cluttered objects. The focus was mainly on finding collision-free upper body configurations of the robot for traversing tight passages. For humanoid robots, cluttered environments are particularly challenging as the robots have to adhere to balancing

constraints in addition to the navigation task. Hence, they have to choose their footholds carefully when moving through cluttered areas. Orthey and Stasse [34] as well as Maier *et al.* [35] proposed suitable solutions for this task given observed obstacle locations.

Joho *et al.* [36] presented a technique using nonparametric Bayesian models for learning exact geometric arrangements of objects from large data sets. The authors showed that their unsupervised approach is able to learn how to set the table. In a related approach, Sudderth *et al.* [37] also reasoned about the number of objects and their spatial relation for detecting items in visual scenes. While the authors are interested in finding reproducible patterns in the geometric arrangements, we cannot expect that there are typical structures that can be learned in our scenario. Instead, we propose an efficient approach for predicting traversal costs for unobserved parts of the environment based on cluttered objects already observed.

Henry *et al.* [38] estimated the density of people in the environment to learn human-like navigation behavior of the robot. When navigating alongside humans in crowded environments such as pedestrian areas, robots have to adapt to the habits of humans, for example with respect to crowd flows or personal spaces of humans. While this approach estimates the density and velocity of pedestrians to plan a path, that mostly follows the flow of the crowd, our work aims at reaching the goal as fast as possible, by executing a foresighted navigation behavior.

In particular, we focus on domestic and office environments. Based on detected objects, we predict costs corresponding to potential obstacles in close but not yet observed areas. By making detours around regions that are likely too cluttered for the robot to easily pass through, our robot avoids getting stuck and efficiently navigates to the goal.

3.3 COST MAPS FOR PATH PLANNING IN CLUTTERED ENVIRONMENTS

We assume that a 2D grid map containing the static obstacles in the environment is given (Sec. 2.3). As the robot moves through the environment, it updates the map continuously by marking cells as *occupied* or *free* whenever previously unknown objects appear in the robot's field of view. The robot uses this map to localize itself and to predict costs corresponding to potential cluttered objects in areas that it has not yet observed with its sensors.

For path planning, we apply the same procedure as described in Sec. 2.4. We assign a cost value to each grid cell and use an A*-based planner to find the path with the lowest costs to the given goal location of the robot. In the following, we describe a standard cost function that considers a safety distance to all obstacles. Already, we briefly outlined the practice of inflating obstacles in Sec. 2.2. In the next

section, we cover this concept in more detail, by focusing on the cost function formulation. Afterwards, we introduce two extensions to the standard cost function, by considering the clutter density and configuration of the local surroundings.

3.3.1 Standard Cost Function

The standard cost function sets the costs of all occupied cells to infinity, furthermore, occupied cells are inflated by a safety distance and their neighbor cells get also assigned infinite costs. In that way, the robot keeps a safety distance r to all obstacles. While this results in the computation of collision-free paths, it may lead to undesirable behavior as the shortest path from one room to another would often run close to walls and door posts when entering a room. While some situations require to squeeze through narrow passages or move close to obstacles, the robot should generally prefer to maintain a certain clearance. A commonly used approach to represent such a preference for open space is to add costs to the regions adjacent to inflated obstacles that decay exponentially with growing distance to the nearest obstacle. This approach allows the planner to trade off between path length and obstacle clearance. As this cost term quickly loses influence with increasing distance, it only has to be computed for cells near obstacles and can be neglected for cells further away.

Putting these cost terms together yields the following cost function:

$$C_{base} = \begin{cases} 0 & \text{if } n = 0 \\ \infty & \text{if } \exists i : \, d_i < r \, , \\ C_{max} \cdot \max\limits_i \, e^{k(r-d_i)} & \text{otherwise} \end{cases} \tag{3.1}$$

where r is the robot's safety distance, $d_1, \ldots, d_n$ are the distances to the nearby obstacles, C_{max} defines the highest traversable cost, and k is a constant scaling factor controlling the decay of the costs. The red curve in Fig. 3.2 illustrates this cost function in an example environment with two obstacles.

3.3.2 Cost Function for Cluttered Environments

The cost function described in Eq. (3.1) takes only the nearest obstacle into account and neglects all other obstacles. However, areas with multiple obstacles close to each other are more challenging for the robot, as locomotion in confined spaces is difficult and the sensor view is often obstructed. Increasing factor k would incite the robot to keep a larger distance to all obstacles irrespective of the amount of clutter, leading to unnecessarily long detours around single, free-standing objects. By contrast, we propose to introduce an additional cost term

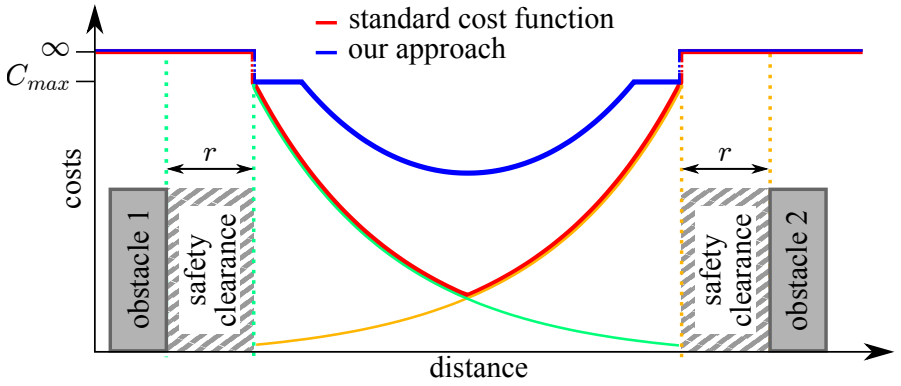


Figure 3.2: Cost function for a simple environment containing two obstacles. The green and yellow curves represent the exponentially decaying costs for obstacle 1 and 2, respectively. These costs encourage the planner to keep a clearance to obstacles in addition to the safety distance. While the standard approach takes the maximum of those curves as the cost function (red), our approach combines the curves according to Eq. (3.2), which leads to higher costs in cluttered areas.

that reflects the amount of clutter in the local surroundings of each grid cell.

The new cost function should meet these requirements:

- The cost value should increase exponentially with the number of objects in the vicinity of the robot, so that the planner will avoid cluttered regions.
- The cost function should have the same value range as the original function Eq. (3.1).
- The cost function should approach zero as the distances go to infinity.
- If there is only one object in the vicinity of the robot, then the cost function should coincide with the original cost function from Eq. (3.1).

The following cost function fulfills these requirements:

$$C_{clut} = \begin{cases} 0 & \text{if } n = 0 \\ \infty & \text{if } \exists i : d_i < r \\ C_{max} \cdot \min\left\{1, \prod_{i=1}^{n} (E_i + 1) - 1\right\} & \text{otherwise} \end{cases} \tag{3.2}$$

where $E_i = e^{k \cdot (r - d_i)}$ is the same exponential decay function as used above in Eq. (3.1). In this formula, we add 1 to E_i so that far-away objects with $E_i \approx 0$ turn into the neutral element in the multiplication.

After the multiplication, we have to subtract 1 again so that the cost function approaches 0 when the distances go to infinity.

In addition to the requirements defined above, the function also has the following properties:

- The product can be computed incrementally as

$$E_{new} = (E_{old} + 1) \cdot (E_j + 1) - 1 \tag{3.3}$$

  when a new object j is observed, which allows for efficient implementation.

- In contrast to the original function Eq. (3.1), it is continuously differentiable in the areas the robot can traverse, which is required by some planning algorithms. Like the original function, it is not differentiable at the borders between free space and obstacles.

Fig. 3.2 shows a comparison of the standard cost function (red) and our approach (blue) in a simple environment containing two obstacles.

3.4 CLUTTER DENSITY AND COST PREDICTION

The cost function defined in the section above can be applied to determine the costs in regions the robot has already observed. However, we would like to also predict costs corresponding to potential objects in adjacent regions that are not yet within the field of view or occluded by other objects. To do so, we estimate local obstacle densities depending on the already observed objects and increase the traversal costs for cells in close-by regions that are not yet visible.

To allow for efficient computation of the object density, we calculate the observed occupation density inside a circle of radius R_j for each cell c_i within the prediction region. We perform this calculation for M circles with different radii as shown in Fig. 3.3. The predicted density ρ_i is then found by summing over all M density estimates, weighted by an exponential factor that decreases with R,

$$\rho_i = \frac{\sum_{j=1}^{M} e^{-R_j} \frac{o_{i,j}}{t_j}}{\sum_{j=1}^{M} e^{-R_j}}, \tag{3.4}$$

where t_j is the total count of cells and $o_{i,j}$ is the count of occupied cells around c_i, both inside the circle with radius R_j. The cost $C_{predict}$ for each cell c_i is then defined as

$$C_{predict} = \alpha \cdot \rho_i, \tag{3.5}$$

where α has to be chosen such that clutter densities that raise navigation difficulties lead to predicted costs close to C_{max}.

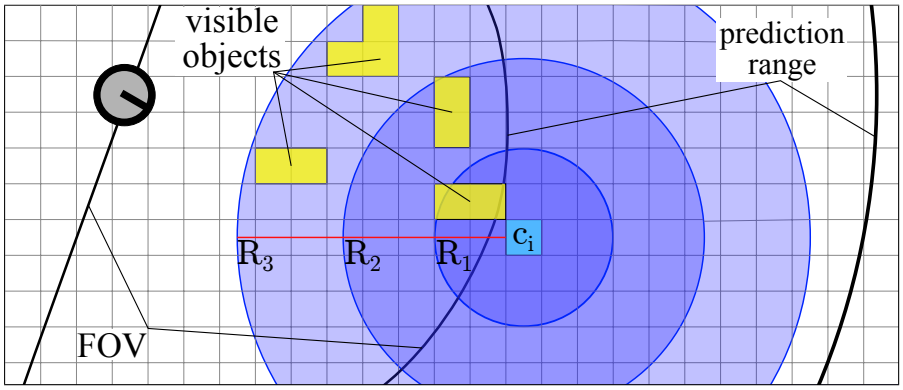


Figure 3.3: Our framework predicts an average clutter density for cells c_i in close but not yet observed areas by calculating densities within the radii R_j around c_i based on occupied cells corresponding to non-static obstacles.

3.5 EXPERIMENTS

We implemented our approach in ROS and evaluated it both in simulation (Gazebo [39]) and in real-world experiments. The practical experiments were performed using a Robotino robot from Festo Didactics with an ASUS Xtion Pro Live fixed to the mounting tower for obstacle detection and a Sick-S300 laser scanner mounted horizontally above the ground for localization. For detecting objects, the robot uses data from the RGB-D camera to build a point cloud of the environment. All points above the ground plane are considered as obstacles. Points that cannot be explained by known obstacles contained in the given environment representation are classified as corresponding to cluttered objects. In our experiments, the prediction region is located from 2.5 m up to 5 m around the front half of the robot. The radii of the circles to compute the occupation density are 0.5 m, 1.0 m, and 1.5 m.

3.5.1 Path Planning and Trajectory Execution

We deploy the navigation setup described in Chapter 2, with a grid-based A* planner to find a global plan using the extended costs of Sec. 3.3.2 and Sec. 3.4. After the path planning the robot executes the trajectory rollout procedure for motion planning (Sec. 2.5). As mentioned before, the motion planner samples velocities in the robot's control space, simulates and evaluates the robot's trajectory in a short lookahead time frame, and sends the highest-rated velocity command to the robot (see [21][23] for details). By sampling and evaluating in the robot's control space, the planner is able to adapt the robot's velocities to the constraints imposed by the environment, to follow the path and subsequently reach the goal. However, the robot will drive

slower in cluttered areas as it has to respect acceleration limits when turning or evading obstacles.

3.5.2 Quantitative Evaluation

We evaluated the impact of the amount of clutter on the completion time in a series of simulation experiments with varying obstacle densities for both the standard technique without modifications and our approach with the modified cost function and prediction. We randomly sampled objects within a rectangular area of size 23×8 m^2. The clutter area is surrounded by free space leaving the robot the possibility to drive around it. As clutter objects, we used boxes with lengths varying from 0.3 to 1.2 m (see Fig. 3.6 for an example map). For the sake of comparison, we define a parameter for the obstacle density D_c in the clutter area as the average number of objects that appear in a region of one square meter.

The task of the robot was to navigate to a goal point at a distance of 28 m at the opposite end of the arena with the cluttered area in between. In this scenario, the robot cannot get stuck in the clutter area as there is always a possible path to the goal, the robot can rotate on the spot and drive back the path that it came from, and the simulator provides error-free sensor measurements. For each D_c, we created 20 randomly cluttered environments with a variety of objects. We averaged the results over two runs with different start and goal positions for each generated environment and evaluated the average travel time for reaching the goal. Fig. 3.4 shows the mean and 95% confidence interval of the overall traveling time in relation to the density of the clutter region.

For the extreme values $D_c = 0$ and $D_c \to \infty$, both approaches yield the same behavior by definition: If no clutter obstacles are present, both approaches will choose the shortest path without being impeded, thus both the mean and variance of the travel time are low. As the clutter density increases, the completion time generally increases as the robot has to take longer paths to avoid obstacles. If the clutter is too dense for the robot to fit through, then both approaches will plan a detour around the whole area, again leading to similar completion times. The approaches behave differently when the clutter is sparse enough for the robot to drive through, but dense enough to impede the robot, i.e., for values of D_c between 0.3 and 0.8. In this case, the robot has to decide whether to drive through the clutter area or to take a detour around it. Driving through the clutter leads to shorter trajectories, but the robot has to drive slower near obstacles and the risk of driving into a dead end and having to backtrack increases. As the results show, our approach finds a good trade-off and achieves shorter completion times in all cases. For the densities marked with (*) and (**), the difference is statistically significant according to a paired

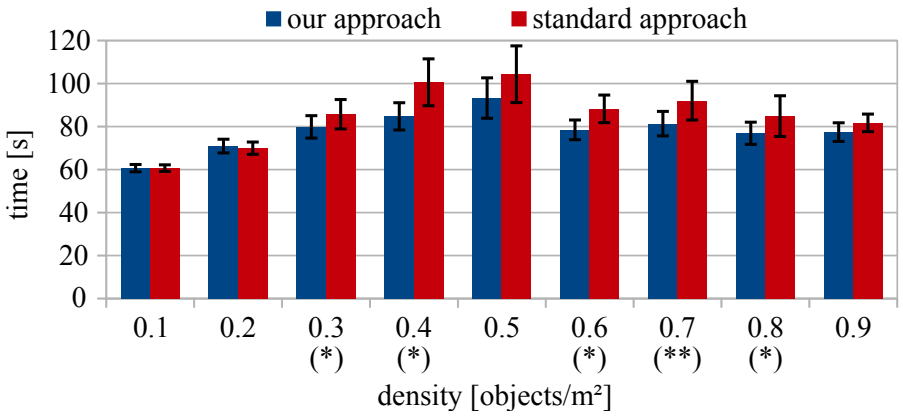


Figure 3.4: Mean and 95% confidence interval of the traveling time in relation to the amount of clutter. For low and high clutter densities, both approaches behave similarly. In the middle range, our approach leads to shorter completion times. For the densities marked with (*) and (**), our approach performs significantly better than the standard approach according to a paired t-test at the 0.05 and 0.001 level, respectively.

t-test at the 0.05 and 0.001 levels, respectively. For densities below 0.5, driving through the clutter is usually the favorable option, while for densities above 0.5 taking a detour around groups of obstacles is often faster. For densities around 0.5, the completion time strongly depends on the geometrical distribution of the clutter, as dead ends and maze-like structures occur frequently, making it hard to predict which path will lead to the goal. Hence, both mean and variance increase for both approaches. While our approach still performs better on average, the difference is not statistically significant due to the high variance. On average, updating the cost map after new sensor readings takes 10.8 ± 4.0 ms for our approach compared to 5.9 ± 2.3 ms for the standard approach.

To assess the contribution of the individual components, we report the results in Fig. 3.5 separately for a planner using only the cost function from Sec. 3.3.2 (green), a planner using only the prediction method from Sec. 3.4 (yellow), and a planner using the combination of both (blue). For better comparison, we normalized the times so that the standard approach matches 100%.

The main effect of the cost function introduced in Sec. 3.3.2 is to increase the costs for squeezing through narrow passages between obstacles. Such navigation maneuvers slow down the robot and they are risky, as sensing errors might lead to collisions or trap the robot in situations where its planner cannot find a valid path anymore.As the results from Fig. 3.5 show, the cost function on its own increases the completion time compared to the standard approach for low clutter densities, because the planner will keep a bigger distance to groups

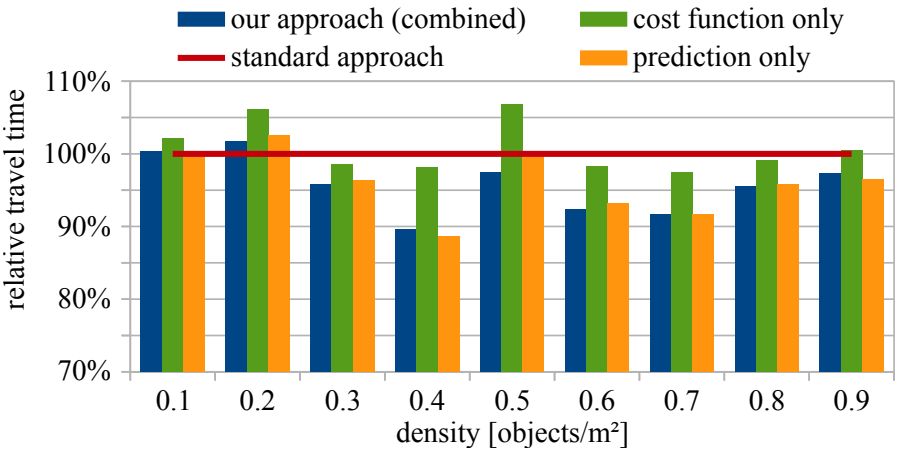


Figure 3.5: Average of the traveling time in comparison to the standard approach broken down by the individual components of our system. Overall, the combination of our cost function and prediction performs best. See Sec. 3.5.2 for a detailed discussion of the results. Note that the vertical axis starts at 70% to improve readability.

of obstacles. In combination with the clutter prediction, however, it decreases the completion time.

Overall, the combination of our cost function and the clutter prediction performs best. In some cases, the prediction alone performs marginally better than the combination, as the cost function increases the clearance between the robot and groups of obstacles, leading to slightly longer trajectories.

In the next section, we will discuss the influence of the individual parts in more detail based on exemplary situations.

3.5.3 Qualitative Evaluation

Fig. 3.6 shows a typical simulation result. The task of the robot was to navigate from the left side to the right side of the room. The center of the room is filled with clutter objects that are not contained in the robot's map. In the depicted situation, the robot has traveled about half the way to the goal. The standard approach without the modified cost function and prediction (top) tries to follow the shortest path to the goal, which leads the robot through dense clutter. As the robot has to respect acceleration limits when turning and evading obstacles, the robot has to slow down. It arrives at the goal after 102 s. Our approach, by contrast, predicts that there are probably more objects behind the obstacles that gradually appear in its field of view while driving. The prediction and the modified cost function increase the costs towards the center of the clutter area as well as in between the objects, inciting the planner to choose the longer, but safer option of making a detour around the obstructed area. Even though the traveled

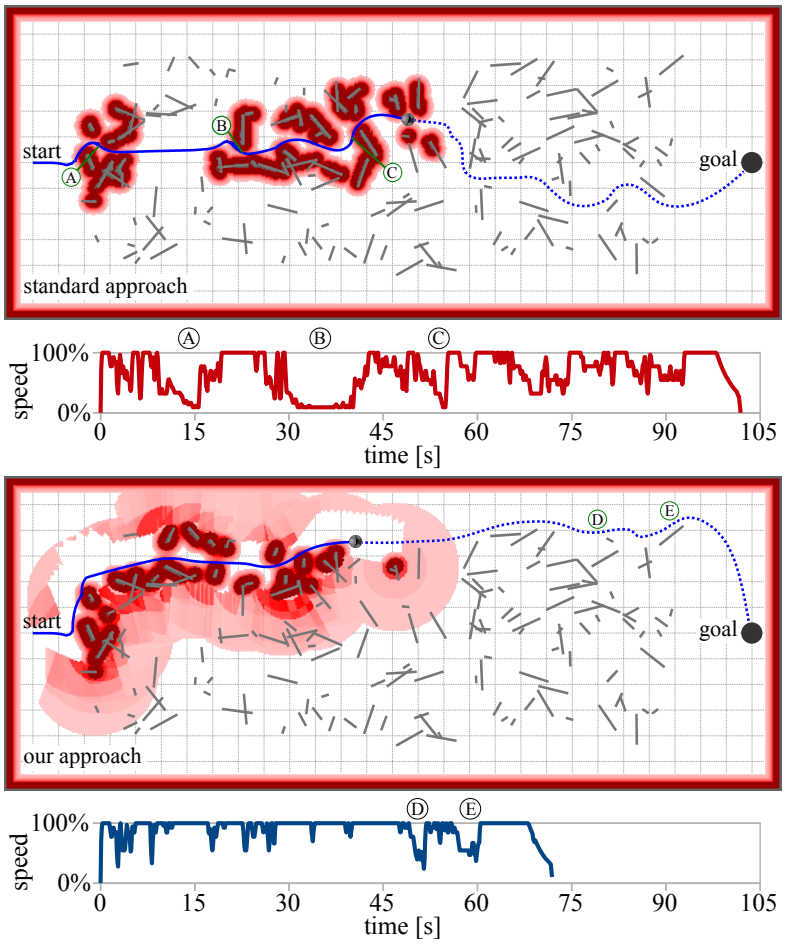


Figure 3.6: Comparison of the two approaches on an example map ($D_c = 0.35$) and velocity profiles in percent of the maximum allowable speed. The standard approach (top) tries to follow the shortest path to the goal, leading the robot through dense clutter where the robot has to slow down to avoid collisions. With our approach (bottom), the costs increase in locations where clutter is predicted, driving the robot around the clutter field and leading to a shorter completion time.

distance of our approach is 11% longer, the robot arrives at the goal 29% earlier after 72 s.

As our algorithm predicts clutter in regions that the robot has not observed yet, it has to update the prediction, based on real measurements, once the robot travels through previously unknown regions and observes the actual scene. Fig. 3.6 also shows how the robot clears the prediction for areas that it has observed as being free space, e.g., in the center top area.

Fig. 3.7 shows the effects of the individual components of our system on a sample map. The standard approach loses time as it has to slow down near obstacles. The prediction incites the planner to drive around dense clutter, but it still drives through narrow passages.

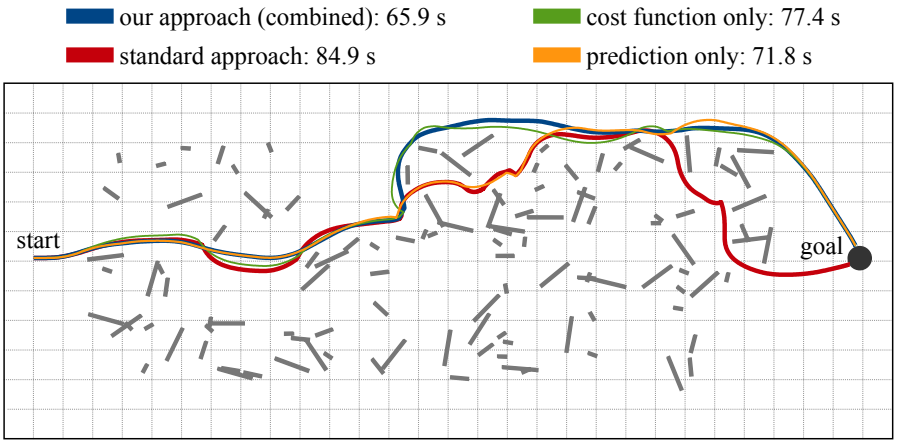


Figure 3.7: Comparison of the individual components of our system on an example map ($D_c = 0.25$). The standard approach loses time by squeezing through narrow gaps. The prediction incites the planner to drive around dense clutter. The cost function prevents the robot from driving through narrow passages. Our combined approach leads to the shortest time.

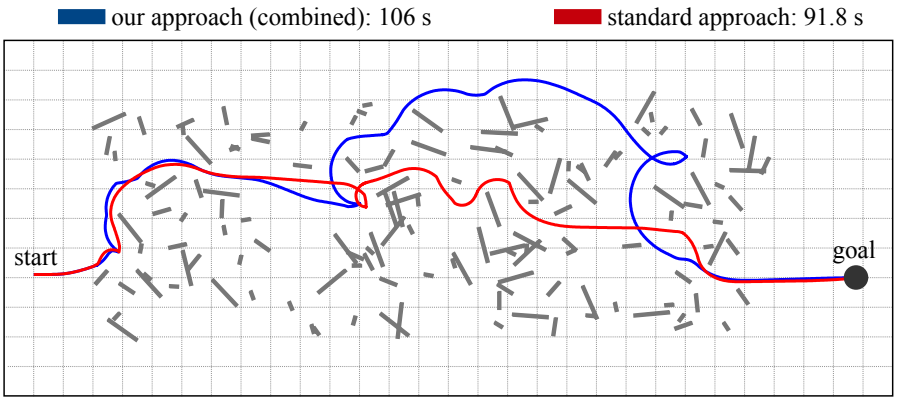


Figure 3.8: Our approach chooses the conservative option of driving around the dense clutter in the center. The standard approach drives through the narrow passage and arrives at the goal earlier.

In the combined approach, the cost function additionally prevents the robot from entering cluttered regions through narrow passages, hence the combined approach yields the shortest completion time.

In Fig. 3.8, our approach evades the dense region in the center of the map and prefers the less dense area in the top. As there is no guarantee that predicted obstacles exist, in this case the clutter avoiding strategy leads to an unnecessary detour. On average, however, the advantages of the prediction outweigh the risk of superfluous detours.

3.5.4 Real-World Experiments

We conducted real-world experiments using the Robotino robot in the environment shown in Fig. 3.9. The environment is surrounded by

walls and contains additional walls separating a corridor and multiple rooms. Additionally, we placed several obstacles on one side of the environment. While the robot has access to a map containing the walls, it does not know the amount or distribution of the clutter objects beforehand.

The standard approach computed a full path through the cluttered region, which is displayed on the left in Fig. 3.9. Although the robot successfully navigated through the clutter, the robot needed to slow down on several occasions to change its orientation and avoid collisions leading to a total navigation time of 45.5 s.

Our approach started with a similar navigation plan, however, once the robot discovered the nearby clutter at the start, the increased predicted costs in the cluttered region immediately led to a replanned path around the corridor. This path is shown on the right in Fig. 3.9. While resulting in a longer path, the total travel time was only 34 s as the robot was able to drive faster due to the free space on the right corridor.

3.6 CONCLUSION

In this work, we proposed a novel solution to efficient navigation through environments containing areas with many cluttered objects. Our approach predicts traversal costs resulting from potential obstacles in regions that are not yet observable by the robot's sensors. The prediction is based on estimated obstacle densities from already detected objects. By considering the predicted traversal costs directly for path planning, the robot navigates foresightedly and avoids regions that are likely to be too cluttered for the robot to easily pass through.

We implemented our system within ROS and represented the predicted traversal costs as a new cost map layer for path planning. As we demonstrate in the experiments with a wheeled robot equipped with a depth sensor, in several situations the resulting navigation behavior significantly outperforms the one generated by a standard path planner that only considers detected objects.

In next chapter we introduce a method to estimate the completion time for wheeled robots based on path characteristic. The predictability of the completion time is a necessary capability for service robots to schedule their work in many situations. Furthermore, the information about the completion time could prevent suboptimal cases as depicted in Fig. 3.8, by comparing the two paths and following the best estimated option.

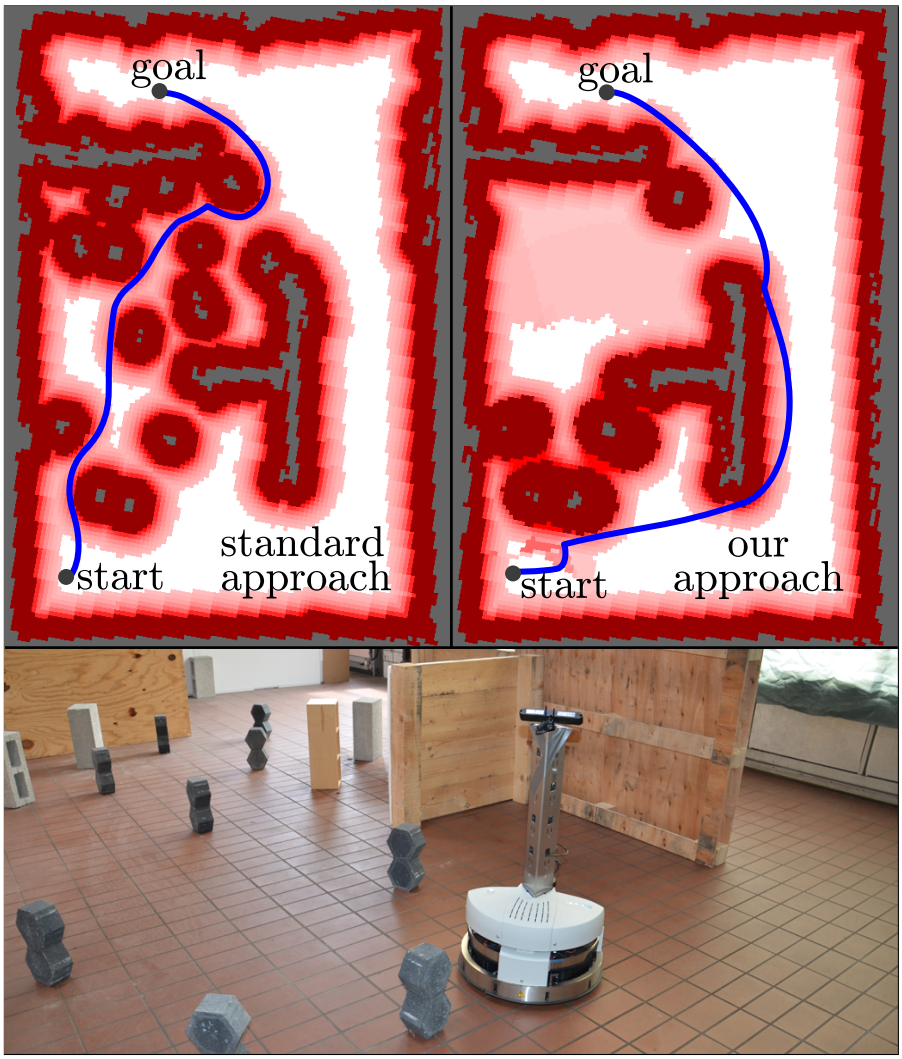


Figure 3.9: Experiment with the real Robotino traveling through a cluttered scene. The standard approach computes a path through the clutter, where the navigation is slowed down. Our approach predicts higher traversal costs leading to an early replan around the cluttered region. The displayed costmaps correspond to the time after the robot has reached the corresponding goals.

4

PREDICTING TRAVEL TIME FROM PATH CHARACTERISTICS

As the experiments in Chapter 3 clearly demonstrate, it is not always the best option for the robot, to choose the shortest path to follow. Since, the shortest path may lead the robot through narrow gaps or be in general hard to execute due to a lack of smoothness. The estimated completion time of a navigation task is a much stronger selection criterion, but is typically a priori not known. We introduce a novel approach to estimate the completion time of a path based on simple, readily available features such as the length, the smoothness, and the clearance of the path. To this end, we apply non-linear regression and train an estimator with data gained from the simulation of the actual path execution with a controller that is based on the dynamic window approach (DWA). As we show in the experiments, our method is able to realistically estimate the completion time for 2D grid paths using the learned predictor and highly outperforms a prediction that is only based on path length.

4.1 INTRODUCTION

Robots nowadays are facing the challenge of having to solve tasks with ever increasing complexity. In several real world applications, the predictability of the completion time of such tasks plays an important role. Generally speaking, the prediction of the task completion time is pivotal for all time-efficient planning scenarios. Such scenarios include cooperative floor-cleaning and household tasks, long term autonomous driving systems [40] that need to estimate their travel time for a more efficient power supply management, and museum tour-guide robots that have to schedule their guiding precisely in order to guarantee sufficient entertainment of the visitors [41].

In Chapter 3 we illustrate scenarios, where several path options are available. Whereby, the fastest option often differs from the shortest or cost-minimal solution. Consider for example Fig. 4.1, where the robot has two options to reach the goal location. The first option (red path) contains narrow passages with several cluttered objects. The second option (green path) traverses solely free space. Taking the red path, the robot needs to drive slowly since it needs accurate sensing and precise motion execution to avoid collisions. At very tight spots the robot even has to stop frequently and rotate in order to adjust its

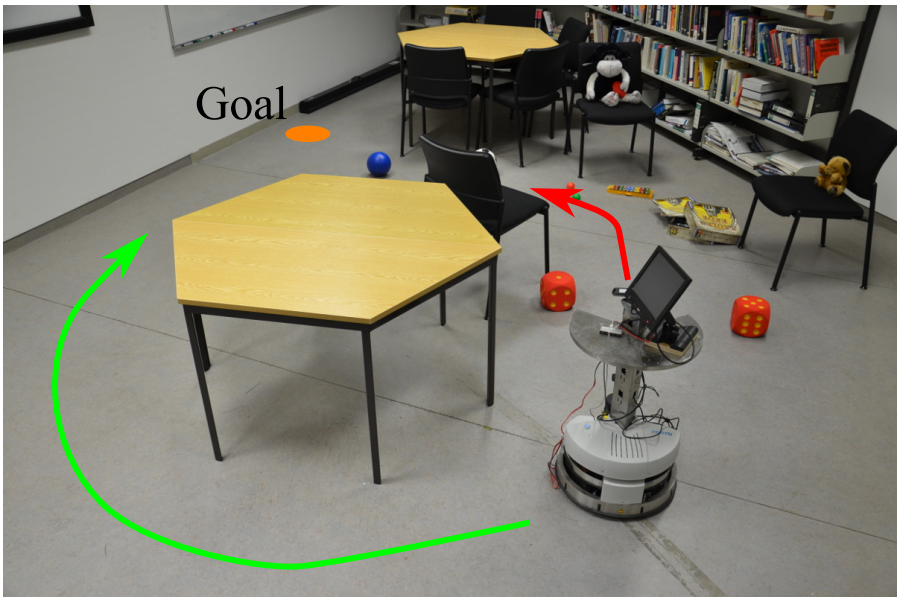


Figure 4.1: Motivation of our approach. The robot can choose between two options to reach the goal location. The shorter path (red) leads through dense clutter where the robot needs to drive carefully and needs accurate sensing and pose estimation to avoid collisions. The second path (green) is longer but leads through wide free space where the robot can drive with a faster velocity profile. The presented approach learns to predict the travel time along 2D paths from training data to decide which path leads to the fastest completion time.

heading. In contrast, the green path leads through free space where the robot can drive with higher velocities. Thus, the robot is faced with the question which route to choose to reach the goal location as quickly as possible.

With two-stage navigation systems where a global path planner is combined with a local motion controller (Chapter 3), the precise outcome of the motion execution is typically difficult or impossible to predict in advance, especially when traversing narrow or cluttered passages. A reactive robot control system such as the dynamic window approach (DWA) (Sec. 2.5) can neither ensure a time-optimal trajectory, nor control stability, nor convergence of the system [42]. Additionally, many sources of noise randomly influence the navigation performance, e.g., the slippage of the wheels, noise in the sensor measurements, and inaccuracies in the localization. The resolution of the grid-based environment representation or the choice of the navigation cost function can also influence the performance significantly.

In this work, we present a novel method to predict the travel time for a mobile robot based on path features that are available ahead of the execution time. This will allow the robot to evaluate different options and choose the path that is predicted to be the most time-efficient.

Given a 2D path in a grid representation of the environment, our method predicts the completion time by means of regression analysis based on general path characteristics such as its length, clearance, and curvature. We extensively evaluated our method in various environments of different complexity. As the experiments show, our method is able to realistically estimate the completion time of 2D grid paths and outperforms a prediction that is solely based on the path length. To the best of our knowledge, this is the first approach that can efficiently predict the completion time of navigation tasks without applying computationally expensive calculations using an exact kinodynamic model of the robot.

4.2 RELATED WORK

Navigation systems, as introduced in Chapter 2, assume that the time-optimal trajectory lies close to the computed 2D path, which is often the case but might not be true in the presence of many obstacles (Chapter 3). In such cases it might actually be better to also take into account different paths with fewer obstacles that need to be passed. Therefore, we present a method that learns the time the robot needs to navigate along a given 2D path based on path characteristics. Based on general features of a 2D path, the robot can then estimate the completion time it would need to follow the path towards the goal location and choose the best option among different possibilities.

Murphy and Newman considered robots operating in large outdoor environments and developed an approach to trade off the risk of planning a path with suboptimal length for planning time and plan over probabilistic costmaps [43]. To create such a probabilistic costmap, one typically needs a priori knowledge about the terrain such as an overhead image of the environment. The work of Murphy and Newman focuses on traversing special types of terrain, whereas our approach is optimized for dealing with challenging indoor environments with mainly flat floor where the terrain properties play a minor role for the performance. Zhu and Qingbao proposed path planning based on a genetic algorithm [44]. The authors introduced functions to describe path characteristics that allow to choose an optimized path from a given set. This approach does not consider the motion control system of the robot. Philippsen [45] used probabilistic navigation functions to trade off the risk of colliding with dynamic obstacles against the length of a detour to avoid those. However, the approach requires tuning and user-defined heuristics and does not involve a trained model.

Lau *et al.* [42] developed an approach to time-optimal control from sparse way points to the goal based on quintic Bézier splines. Starting from a given straight-line path, the trajectory is optimized for smoothness and time taking into account the constraints of the system. In

this work, we consider general navigation in environments of different complexity also containing highly cluttered and narrow passages. Our goal is to estimate the travel time based on simple, readily available features describing the path characteristics and in this way enable the robot to choose the best option, i.e., the path assumed to lead fastest to the goal using a standard DWA-based controller that generates velocity commands in an efficient manner.

4.3 PREDICTING TRAVEL TIME FROM PATH CHARACTERISTICS

A two-layered approach (Sec. 2.5) can generate robust, collision-free motion even in obstacle-laden environments. However, the highly unpredictable nature of the DWA controller as well as the influence of noisy perception and localization makes the estimation of the completion time of a motion task a difficult endeavor. The cheapest path in a costmap is not always the best choice as it may lead through narrow or cluttered passages and it may be in general hard to follow without slowing down and rotating on the spot (Chapter 3).

To illustrate this, we performed two navigation experiments in a large cluttered environment where the robot had the choice to drive through or around the cluttered region. Fig. 4.2 shows this scenario with the two paths and their corresponding velocity profiles. The red profile in Fig. 4.2 shows that driving around a cluttered region allows the robot to navigate at full speed and reach the goal in a shorter time even though the total path length is longer. Driving through the dense clutter, however, leads to higher localization errors due to more frequent rotations, repeated velocity drops in order to avoid collisions and on-spot rotations, which are necessary in regions with very little space to navigate.

Thus, the estimated completion time is in many situations a much stronger selection criterion than the path costs, but due to the lack of a dynamic model in the path computation phase the completion time is typically not known a priori. In the following, we introduce a novel approach to estimate the completion time from path features to enable the robot to choose the most promising path among different possible routes through the environment.

In principle, the only way to predict the completion time is to simulate the path execution and measure the time the robot takes to navigate to the goal. Our idea is to apply a machine learning approach and to train a predictor function for the execution time, based on a small number of generic features that can be efficiently computed from a given global 2D grid path.

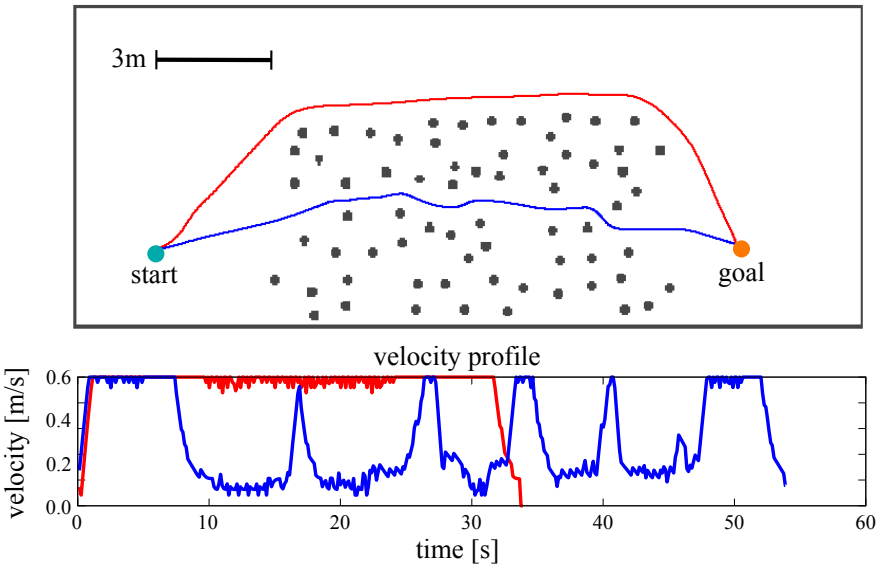


Figure 4.2: Example velocity profiles of a robot driving through and around the cluttered region. Obstacles are displayed in gray. In order to reach the goal, the robot has the choice to either navigate through (blue path) or around (red path) the clutter. The corresponding velocity profiles are displayed to indicate common navigation issues that arise from navigating close to multiple obstacles. Considering the red profile, it is easy to see that the robot can constantly drive close to the maximum velocity and, thus, reaches the goal after only $34s$ with a traveled distance of $18.54m$. The blue profile shows that constant speed drops occur, which are necessary in order to avoid collisions. Additionally, on-spot rotations are performed if too tight directional changes are necessary. This leads to a lower traveled distance of $14.46m$ while the execution time increased to $55.36s$.

4.3.1 Features for Describing Path Characteristics

We define a path $\mathbf{P} = \{\mathbf{p}_0, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ between the current position of the robot and the goal location as a sequence of two-dimensional coordinates (nodes) $\mathbf{p}_i = (x_i, y_i)$, $i \in \{0 \ldots n\}$, as illustrated for an example path in Fig. 4.3. A segment $\mathbf{s}_i$ of the path is then given by the vector $\mathbf{s}_{i+1} = \mathbf{p}_{i+1} - \mathbf{p}_i$. We found out that the length of the path, its clearance, and smoothness are expressive features that can be used to effectively estimate the time the robot needs to travel along the path towards the goal location. These features are described in detail in the following:

1. The *total length* of the path is given by the sum of the lengths of each path segment:

$$L_p = \sum_{i=1}^{n} |\mathbf{s}_i| \tag{4.1}$$

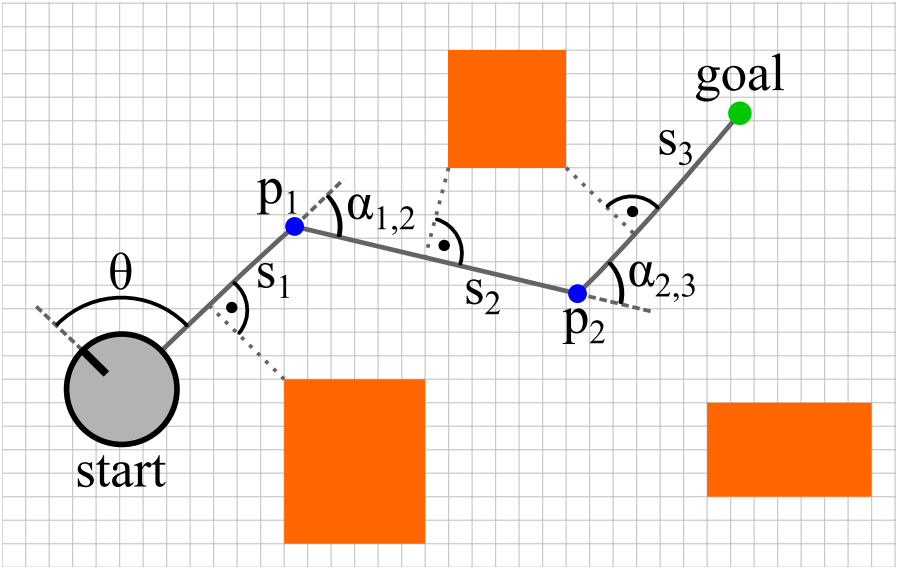


Figure 4.3: Visualization of the features we use for path characterization. The figure shows an example path from the robot's current positions (gray circle) to the goal location through an environment with three obstacles (yellow rectangles). The path consists of three segments and two nodes. The angles $\alpha_{1,2}$, $\alpha_{2,3}$, and θ used in Eq. (4.2) are also shown. The shortest distances between the segments and the obstacle cells are illustrated as black dashed lines and are used in Eq. (4.3).

2. The *average smoothness* of a path expresses its deviation from being a straight line:

$$S_p = \frac{\theta + \sum_{j=1}^{n-1} \alpha_{j,j+1}}{n}, \tag{4.2}$$

where θ is the angle between the initial heading of the robot and the first path segment, and $\alpha_{j,j+1}$ denotes the angle between two path segments $\mathbf{s}_j$ and $\mathbf{s}_{j+1}$. For example, $\alpha_{1,2}$ in Fig. 4.3 denotes the angle between $\mathbf{s}_1$ and $\mathbf{s}_2$.

3. Finally, the *average path clearance* is computed as follows:

$$C_p = \frac{\sum_{i=1}^{n} \max\{D_{max} - D_{min}\left(\mathbf{s}_i, c_{occ}\right), 0\}}{n} \tag{4.3}$$

using the shortest distances $D_{min}\left(\mathbf{s}_i, c_{occ}\right)$ between each path segment $\mathbf{s}_i$ and the occupied cells c_{occ} closer than a threshold $D_{max} > 0$. We assume that obstacles with a distance greater than D_{max} have no effect on the task execution. The clearance is illustrated in Fig. 4.3 as a dashed line from a path segment $\mathbf{s}_i$ to its closest obstacle.

4.3.2 Prediction of Travel Time

Using the three path features defined above, we train a predictor function

$$T_p = \mathcal{F}\left(L_p, S_p, C_p\right) \tag{4.4}$$

that estimates the expected path execution time T_p based on the total length, average smoothness, and average clearance of the path. These features are readily available before the actual path execution starts by a local controller.

4.3.3 Regression Models

Regression is a common tool in statistical analysis to find relationships among variables. The goal of the regression analysis is to find a model that fits well the given data points and thus can be used for prediction afterwards. Different models can map different types of relationships between the variables. Linear regression, for example, is a very fast algorithm, but can only model linear coherences. A special case of linear regression is the simple linear regression that fits the data with a simple regression line. Linear regression, in contrast, models the relationship among several independent variables to predict the requested dependent quantity. For systems with non-linear behavior, linear predictors are often not sufficient. Better results can be achieved with more advanced methods. Support vector machines, for example, are kernel methods that map the data input into a high-dimensional feature space using kernel functions. This kernel trick allows to detect non-linear coherence in data sets.

To find the right regression approach for our problem, we evaluate the simple linear regression, linear regression, and support vector method for the task of completion time prediction based on the path features length, smoothness, and clearance that are described above.

4.4 EXPERIMENTS

In this section, we discuss the data collection process, the regression analysis, as well as the prediction results in different environments.

4.4.1 Data Collection

Our goal is to obtain a single regression model that covers as many scenarios as possible. In order to gather data that is well distributed over the feature space, we performed experiments on a variety of maps such as the Willow office environment and artificially created maps (see Fig. 4.4). One type of artificial maps we used are highly cluttered maps consisting of uniformly distributed or Gaussian distributed

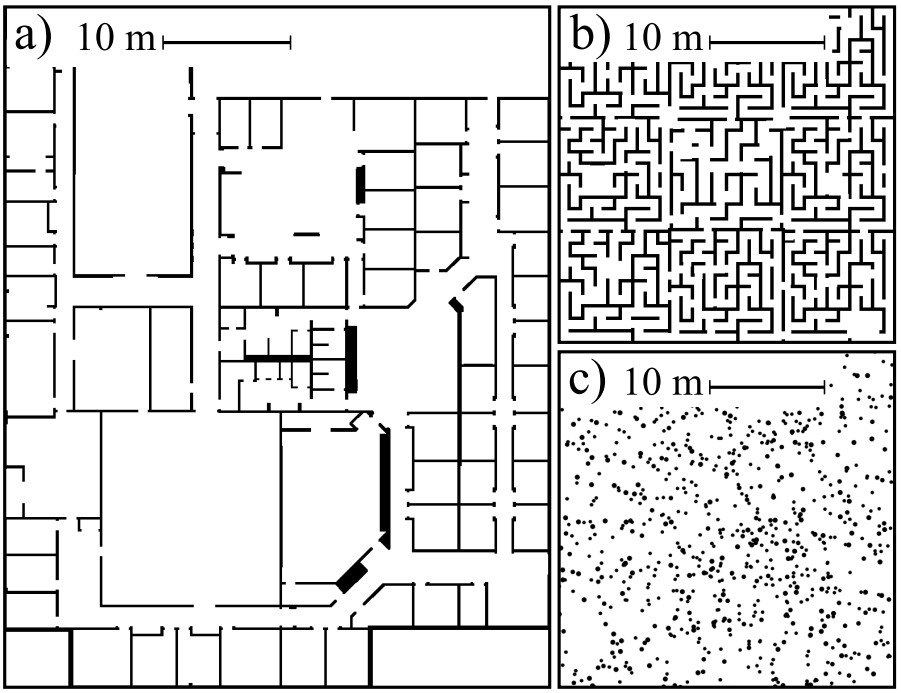


Figure 4.4: Maps used in the experiments. (a) Office environment created by Willow Garage, (b) narrow maze-like environment, and (c) cluttered environment with many randomly distributed obstacles.

pillars, where pillars are randomly generated in varying quantity of 75 pillars per hundred square meters, 50 pillars per hundred square meters, and 25 pillars per hundred square meters with varying radii from 20-60 cm according to the distribution used. Another type of artificially created maps consist of narrow maze-like structures with a corridor width between 0.6 m and 0.9 m. We generated three different artificial maps of both types.

To collect training data, we used the Gazebo simulation environment [39] with a model of the omnidirectional Robotino robot by *Festo Didactics*. We first compute a global path from the current position of the robot to a goal position and then let the robot follow this path with a DWA-controller. To obtain ground truth data, we measure the task completion time when the position of the robot is close to the x-y-coordinates of the goal position. The final heading is not considered. In each experiment, the start position, the initial heading of the robot, and the goal position were chosen randomly. We used an A*-planner for computing the global path. The lengths of the grid-based paths varied between 4 m and 50 m. The A*-planner and the DWA-controller are implemented in the ROS navigation stack [24].

The choice of the parameters of the navigation systems has a pivotal influence on the performance of the robot during the experiments. We found the following parameters to work best in practice. We used a resolution of 5 cm for the global costmaps of the environments and

1 cm for the local map. The frequency of the control loop was set to 8 Hz and the size of the local costmap was chosen to be 1.5 m $\times$ 1.5 m. The maximum linear velocity was set to 0.6 m/s and the maximum rotational velocity was set 0.6 rad/s. The acceleration limits for linear and rotational movement were set to 0.7 m/s^2 and 0.7 rad/s^2, respectively. Naturally, the capabilities of the underlying physical system are instilled into a trained regressor. A deviation from the configuration parameters at a later time may work to some extent, we have not evaluated this in our work so far, but in general it must be assumed that the model is not transferable to a new system with significantly different navigational capabilities. The training must be performed for each individual combination of robot and navigation software.

We created two data sets with each containing 5500 navigation tasks. The first data set was gathered without any sources of noise, i.e., no noise in the sensors and without slippage of the wheels. In particular this also includes a perfect localization. Naturally, this model is not entirely realistic, but it helps to analyze the data with respect to the correlation of the features with the estimate. The second data set was collected from experiments with a localization system that adds noise to the simulation due to faulty pose estimates. Note that in the second data set, the sensors and motion itself are still noise-free. Using these two data sets, we can evaluate our model with noise in comparison to noise-free results and also see how much the noise in the system affects the navigation performance.

4.4.2 Regression Results

In this section, we present the results of our regression analysis. For every data set, we learned an estimator for each of the different approaches. We used a simple linear regression (SLR) method based on the path length alone as computed in Eq. (4.1), a linear regression (LR) model which considers all the features mentioned above (see Sec. 4.3.1), and we trained a support vector machine for regression (SVR), also using all features. For training and testing we used WEKA, a well-established data mining software [46]. To evaluate the different regression models, we performed a 10-fold cross validation on the data set, i.e., during one validation run, 90% of the data set is used for training and the other 10% for testing this specific model. In the next validation round, another subset of 10% is used for testing and we repeated this process 10 times until every subset of the data set has been tested. We computed the average of the root mean square errors (RMSE) of every ten testing runs. As a reference, we additionally computed the RMSE of the constant average estimator over the entire data set.

It is not sufficient to assume a linear distribution, because, in the presence of clutter and narrow gaps, our navigation system exhibits

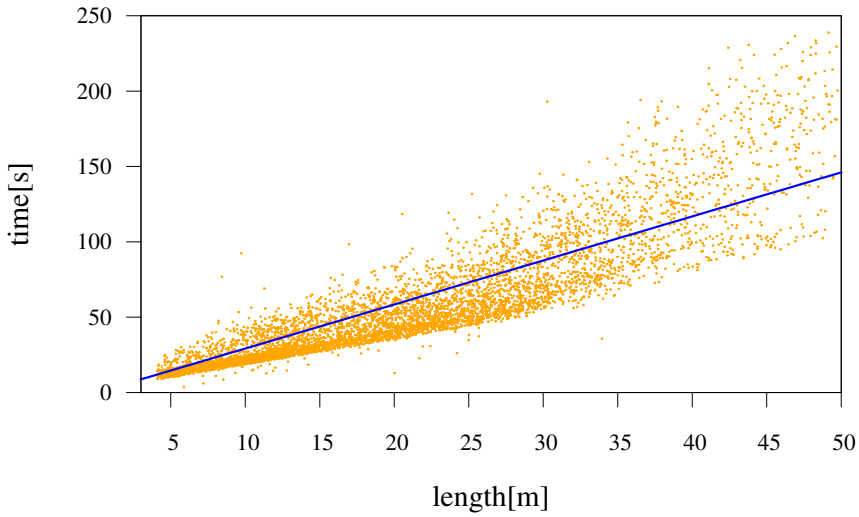


Figure 4.5: Completion time of the simulated execution of the generated paths in the three environments (yellow) over path length. With increasing length, the data spreads broader around the regression line (blue). These results were obtained from the experiments with noisy localization.

a highly non-linear behavior. Nevertheless, linear models are easy to fit and fast to compute and thus serve as a good reference. Fig. 4.5 illustrates the completion time of every run in the data set over the path length (computed according to Eq. (4.1)). Note that the spread of the data points (yellow) around the linear regression line (blue) increases with path length.

The regression results depicted in Fig. 4.6 show that the features introduced in Sec. 4.3 have a substantial influence on the time estimation, as we can see a 14% improvement for both data sets of the LR compared to SLR. A further reduction of the prediction error can be achieved using the non-linear model. Using the SVR results in a RMSE that is further reduced by 15% compared to the LR for the data set with perfect localization and 22% for noisy localization. These results support that the system behaves highly non-linear and a linear regression method is not sufficient if a more accurate estimate is desired.

Comparing both data sets against each other, we can clearly see the influence of the noisy localization which is close to real-world runs. The results also show that some of the noise can be estimated by our approach, since the error reduction from LR to SVR is larger for noisy compared to perfect localization. This improvement stems from the fact that a much higher localization error correlates with certain non-linear behaviors, e.g., rotating fast on the spot or traversing monotone environments with only few features. Thus, the non-linear SVR method is best suited for real-world scenarios.

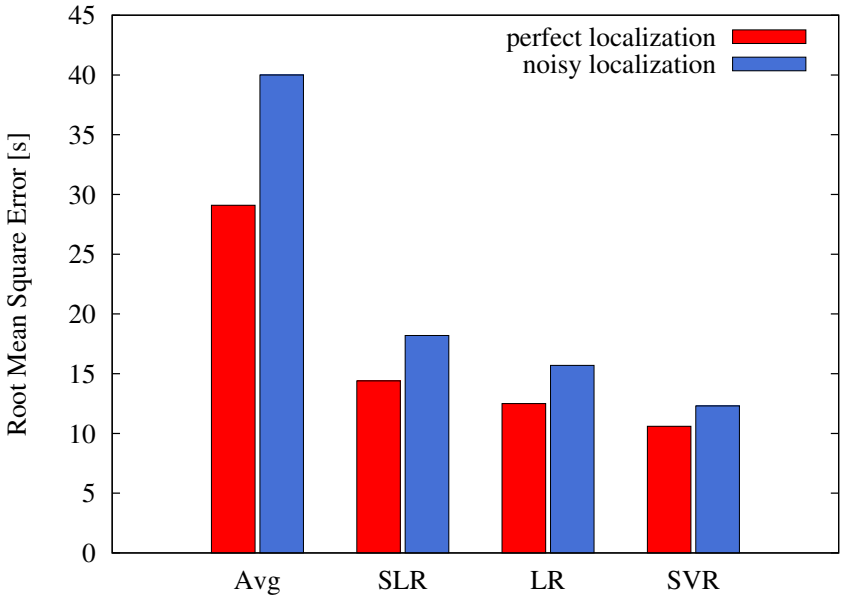


Figure 4.6: Comparison of four different regression methods for perfect (red) and noisy (blue) localization. The regression methods are the constant average estimate (Avg), a simple linear estimate (SLR) based on path length only, linear regression (LR) using all features, and support vector machine regression (SVR) also using all features. As can be seen, the SVR has the smallest RMSE of all approaches. This non-linear model seems to be the best approximation for our robot and controller setup. Furthermore, we see a clear improvement of the LR model in comparison to SLR with the additional independent features.

The evaluation shown in Fig. 4.6 is well suited for a comparison of the different regression approaches. Additionally, we are interested in the relative root mean square error of the estimate, which is defined as follows:

$$\sigma_{est} = \sqrt{\frac{\sum_{i=1}^{N} \frac{(Y_i - Y_i')^2}{Y_i^2}}{N}}, \tag{4.5}$$

where Y_i is the completion time of experiment i, Y_i' is the corresponding estimated value, and N is the number of experiments in a set. As we evaluated a wide variety of scenarios which contained both very short and very long paths, σ_{est} is a better measurement of the relative deviation per experiment, as we first scale every separate squared error by the corresponding completion time. We computed the values of Eq. (4.5) for both the LR and SVR due to the superior performance compared to SLR. For the LR and SVR, σ_{est} evaluates to 0.32 and 0.13, respectively. These results show that the use of SVR not only highly decreases the average deviation, but also shows an improved estimate for the whole spectrum of path lengths. As these results show, our

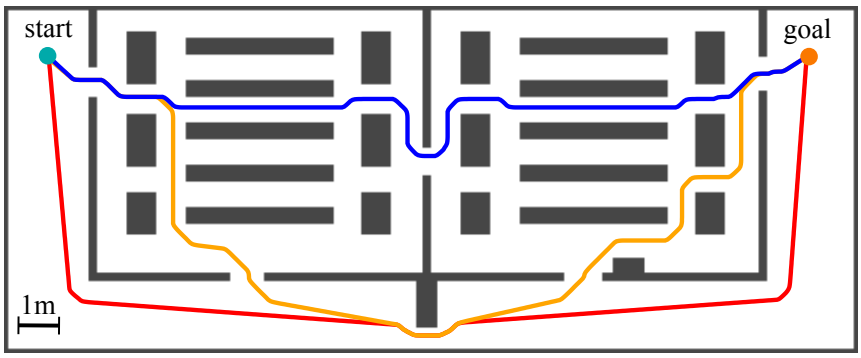


Figure 4.7: An environment with two rooms and a corridor, that was not used for the training data, with three different path options to get from the start to the goal. The longest path (red) leads the robot through wide free-space area. The shortest path (blue) guides the robot through narrow space between obstacles. An alternative path (yellow) leads partly through the narrow passages and through wide-space. The evaluation of the path is shown in Tab. 4.1.

approach is able to predict the path completion time with an error of only 13% in average.

4.4.3 Temporal Gain

To demonstrate the temporal gain when applying our prediction, we performed an experiment on a completely new map (see Fig. 4.7). In this experiment, three different path choices to navigate from start to the goal location exist. The first option is the shortest path (blue), which leads through narrow areas. The red path is the longest, but it is smooth and has a high clearance to obstacles. The third alternative consists of segments of the other two paths. Based on the completion time predicted by our approach, the longest path is chosen as the fastest option followed by the shortest path. The third path is the slowest according to our prediction. By executing all three options in simulation, the actual completion time in Tab. 4.1 confirms the prediction and the path choice. The actual temporal gain when executing the red path in comparison to the execution of the shortest path amounts to 6 s, which is 9.8% of the travel time.

4.5 CONCLUSION

In this work, we presented a technique to estimate the completion time for 2D grid paths. The completion time is in general not known in advance as it strongly depends on the capabilities of the underlying motion controller. Through a low-dimensional categorization of the

Table 4.1: Evaluation of the paths shown in Fig. 4.7

	red	**yellow**	**blue**
length	28.5145 m	26.0843 m	20.9335 m
clearance	0.2685	0.4401	0.5131
smoothness	0.0137	0.0457	0.0483
prediction time	47.373 s	58.756 s	50.392 s
completion time	55.512 s	63.751 s	61.511 s

paths using three generic features—their length, smoothness, and clearance—and the simulation of a large variety of motion tasks on different types of maps, we were able to regress an estimator that predicts the path completion time with a low error of around 10% before motion execution starts. Naturally, as the completion time depends strongly on the navigation performance of the robot, it needs to be trained individually for a specific hardware and motion controller combination.

In the next chapter, we will also use supervised learning, to train a neural network motion controller for navigation in pedestrian crowds. Predicting the completion time of navigation paths works well in semi-dynamic environments, where we can assume the robots trajectory taken to the goal is similar to the path computed at the beginning. However, this assumption is inaccurate when navigating through crowds of pedestrians, where every potential obstacle is also moving to its own destination. In these scenarios it is impossible to predict the final trajectory of the robot, since the surroundings of the robot drastically change for every time step. Reaching the goal, avoiding the incoming pedestrians, and considering the social distance to people are factors of more significance for the navigation performance in the next chapter.

5

IMPROVING SOCIAL NAVIGATION BY SUPERVISED LEARNING

In this chapter, we present a novel, efficient approach to improve the acceleration commands computed by the popular social force model (SFM) for navigation through pedestrian crowds. Our method consists of two stages. In the first phase, we collect training data with a simulated approach. In this step, we modify the steering acceleration commands from the SFM according to a set of discrete alterations and simulate the motion of the robot as well as the pedestrians into the future for each alteration. We rate each resulting trajectory based on an objective function and apply the best steering command to the robot. While controlling the robot in such way, we collect for every time step the input and output training data. In the second stage, we then train a neural network given the collected training data. We use the best acceleration values experienced in the first phase as target values for the neural network and define simple input features describing the local surrounding of the robot. In extensive simulation experiments using different pedestrian densities, we demonstrate that the controls generated by the learned neural network lead to a significantly reduced number of collisions with pedestrians compared to the results of the basic SFM controller, while achieving similar or even shorter completion times.

5.1 INTRODUCTION

The ability to navigate through crowds of people is essential for all service robots operating in human populated environments such as office corridors, hospitals, or shopping malls. However, generating motion commands for navigation among people is a complex task (see Fig. 5.1). To be able to anticipate and avoid collisions, the motion of people needs to be taken into account.

A popular approach to model human-aware navigation is the social force model (SFM) [47], which drives individuals through the combination of repulsive and attractive virtual forces called "social forces". However, such virtual force field methods have limitations, since mobile robots controlled by using force fields can be trapped in local minima and show oscillation behavior in narrow spaces. In addition, encountering dense crowds of pedestrians can lead to a high number of collisions[48], [49].

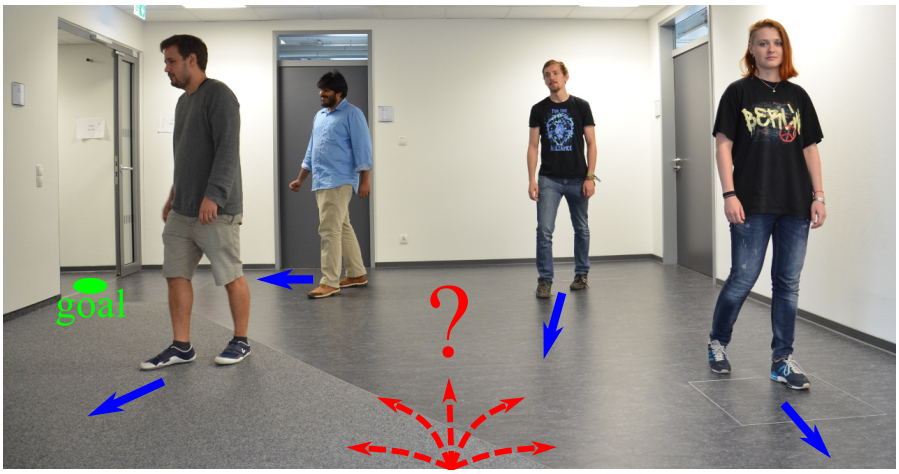


Figure 5.1: Example of a robot navigating through environment populated by humans. To avoid collisions and efficiently reach the goal, the robot has to decide on its control commands. We propose to use a neural network in combination with the popular social force model (SFM) to generate acceleration commands, based on features describing the configuration of the humans in the robot's vicinity.

We therefore propose a two-stage approach to improve the controls generated by a SFM based controller to realize efficient and collision-free robot navigation through pedestrian crowds. The idea of the first phase is to improve the SFM based robot navigation. To do so, we simulate the positions of the robot as well as of the pedestrians in the robot's local environment a few time steps into the future. We hereby assume the positions and velocities of the pedestrians to be known. For the robot's control, we consider a discrete set of incremental alterations of the accelerations commands generated by the SFM based controller. Based on the prediction for a given steering command, our system then evaluates the surroundings of the robot, i.e., the number of pedestrians in the robot's vicinity and the distance to its navigation goal. The best result of the acceleration values is chosen as the control command to be executed by the robot. With this predictive controller, the robot navigation behavior can be substantially improved since it forecasts the situation and reacts accordingly.

However, the simulated prediction is computationally expensive and, furthermore, a real robot will not have knowledge about the velocity and the target destination of all surrounding people. Therefore, we propose to train in a second step a neural network (NN) based on simple features describing the configuration of the pedestrians in the robot's vicinity and the output of the predictive controller. The NN combined with the SFM can then be used for safe and efficient navigation through dense crowds.

As we show in extensive simulation experiments with different densities of pedestrians, the controls generated by the NN lead to a

significantly reduced collision rate of the robot with pedestrians in comparison to the basic SFM controller. Furthermore, the experiments demonstrate that a robot controlled by the NN achieves a speed that is comparable to the one generated by the SFM controller.

The contribution of our work is a controller that is learned in a supervised manner based on simple features describing the pedestrian state around the robot. To generate the training data, we predict the state of the local environment by simulating forward the motions of pedestrians and the robot.

5.2 RELATED WORK

A navigation approach that uses the SFM was presented by Ferrer *et al.* [50]. The authors introduced a metric computed as the weighted sum of the robot's social forces and determined the weights of the different forces using a Markov Chain Monte Carlo Metropolis-Hastings algorithm. In [51], Ferrer *et al.* extend the capabilities of the robot in context of social navigation through additional categories of forces to accompany people side by side. Our approach, in contrast, learns to improve the acceleration commands of the robot provided by the SFM depending on the current situation regarding the completion time of the navigation task. Cao *et al.* [52] perform path planning through crowds using first Voronoi diagrams and Delaunay triangulation to segment the environment and later search for a path through pedestrians within a defined dynamic channel. A great idea is presented by Chen *et al.*[53] to consider human-robot as well as human-human interaction while navigating using a reinforcement learning approach. Unfortunately, it is not mentioned how to incorporate static obstacles within the framework. Also the training and performance of the network was tested only in a circle scenario, where all the participants are placed on a circle line and have to reach the opposite site. It is not clear how the learned behavior would adapt to general scenarios.

Inverse reinforcement learning (IRL) has been the most widely-used approach applied to achieving social navigation. Henry *et al.* [38] utilized IRL to teach a path planner on a 2D grid from example traces of pedestrians. The authors note that a robot agent in such a context would have limited knowledge of its environment and therefore estimate the density and flow of pedestrians around it using Gaussian processes. Gaussian Processes have also been utilized for modeling pedestrian motion by Vemula *et al.* [54]. The authors used real human trajectory data to train their model and propose to discretize each agent's neighborhood and construct an occupancy grid per agent containing all information of its neighbors. This approach is able to predict future trajectories of observed pedestrians using a Monte Carlo sampling approach as well as to compute the path of a robot through

a dense crowd. Kuderer *et al.* [55] and Vasquez [56] used maximum-entropy IRL to learn a behavior model of pedestrians in order to generate socially compliant trajectories. In the work of Vasquez [56], the speed and orientation of the pedestrians in relation to the robot are used as observations for the robot. However, IRL is typically computationally expensive leading to long training times and requires a large training set in order to learn the reward function.

Kim and Pineau [57] proposed to use maximum mean discrepancy as a query metric to determine how far an observation is from the distribution of the training data up to that point. Teaching values are requested only if this metric is above a certain threshold. This approach produces safe trajectories, however, it is rather computationally expensive to train. Recently, Tai *et al.* [58] presented generative adversarial imitation learning that directly learns a policy from expert demonstrations using the so-called trust-region policy optimization without going through the intermediate step of learning a reward function and without needing a map of the environment. The authors make use of the SFM and generate a large set of training examples in order to train their model based on raw depth data. So far, the approach has been only applied to specific basic navigation scenarios.

Recently, long short-term memory (LSTM) have become popular for motion prediction. For example, Everett *et al.* [59] consider motion planning for a group of robots. The authors defined a reward function based on velocity, change in heading direction, and distances to the goal as well as the closest other agent. They use a LSTM cell at the input layer where each agent in the environment subsequently feeds the cell its own state at every decision step. By considering the information of all agents about the state, the LSTM contains a fixed-length, encoded state of the world for the current decision step. The authors then apply reinforcement learning to generate the navigation policy. As each learning episode contains information from several agents the learning is accelerated. However, the requirement of this method to have global knowledge about the states of all agents limits its applicability in a real-world scenario.

Pfeiffer *et al.* [60] use LSTM neural networks to predict the motion of humans, in this case taking explicitly into account static obstacles to improve the prediction in cluttered environments. Also Alahi *et al.* [61] proposed an LSTM for modeling human motions in crowds. The focus in this approach was on trajectory prediction without hand-crafted functions. A Bayesian generative model was developed by Blaiotta [62] to predict multi-agent dynamics. It would be interesting to investigate those recently presented capable prediction methods in context of robot navigation. However, considering the many possibilities to integrate a prediction method into the navigation process, many question still remain open in that area of research. In this work, we favor

the SFM because of the intuitive nature, the simplicity, and straight forward applicability for robot control.

5.3 SOCIAL FORCE MODEL

In the following, we first introduce the social force model (SFM) [47], [63], [64], which is the basis of our approach. The SFM models human motion as the sum of three different *social forces*:

- The *desired force* $\vec{f}_i^0$ reflecting the desire of pedestrian i to reach a particular destination at a particular speed,
- The *obstacle force* $\vec{f}_i^{wall}$ as repulsive force between static obstacles and pedestrian i,
- The *social interaction force* $\vec{f}_{ij}$ as repulsive force between pedestrians i and j.

The desired force $\vec{f}_i^0$ is defined as

$$\vec{f}_i^0 = \frac{\partial \vec{v}_i^0}{\partial t} = v_i^0 \vec{e}_i^0 - \vec{v}_i(t) \ , \tag{5.1}$$

where $\vec{v}_i$ is the current velocity of pedestrian i, $\vec{e}_i^0$ is a unit vector pointing from the pedestrian to the goal and v_i^0 is the desired speed.

The obstacle force $\vec{f}_i^{wall}$ decays with the distance d_w between the pedestrian and a static obstacle:

$$\vec{f}_i^{wall}(d_w) = e^{\frac{-d_w}{0.2}} \tag{5.2}$$

The social interaction force is defined as the sum of two components: f_v that describes the deceleration along the interaction direction $\vec{t}_{ij}$ and f_θ that describes the directional changes along $\vec{n}_{ij}$.

The interaction direction is given by

$$\vec{t}_{ij} = \frac{\lambda(\vec{v}_i - \vec{v}_j) + \vec{e}_{ij}}{\left\|\lambda(\vec{v}_i - \vec{v}_j) + \vec{e}_{ij}\right\|} \ , \tag{5.3}$$

where $\vec{v}_i - \vec{v}_j$ is the relative motion between the pedestrians, $\vec{e}_{ij}$ the unit vector pointing from pedestrian i to j and λ is the relative weight of the two directions.

Let $\vec{n}_{ij}$ be the normal vector of $\vec{t}_{ij}$, oriented to the left. The forces f_v and f_θ are defined as

$$f_v(d_{ij}, \theta_{ij}) = -Ae^{\frac{-d_{ij}}{B-(n'B\theta_{ij})^2}} \tag{5.4}$$

and

$$f_\theta(d_{ij}, \theta_{ij}) = -AKe^{\frac{-d_{ij}}{B-(nB\theta_{ij})^2}} \ , \tag{5.5}$$

where d_{ij} denotes the distance between two pedestrians i and j and θ_{ij} is the angle between $\vec{t}_{ij}$ and $\vec{e}_{ij}$. The parameter K is the sign of the angle θ_{ij} and B is modeled as

$$B = \gamma||\lambda(\vec{v}_i - \vec{v}_j) + \vec{e}_{ij}|| \ . \tag{5.6}$$

The social interaction force $\vec{f}_{ij}$ can now be defined by combining the Equations 5.4 and 5.5:

$$\vec{f}_{ij}(d_{ij}, \theta_{ij}) = -Ae^{\frac{d_{ij}}{B}}[e^{-(n'B\theta_{ij})^2}\vec{t}_{ij} + Ke^{-(nB\theta_{ij})^2}\vec{n}_{ij}] \tag{5.7}$$

Moussaïd [63] defined the parameters $A, B, n, n', \gamma, \lambda$ by fitting the function in Eq. (5.7) to real-world data of pedestrian behavior. The complete equation of motion for a pedestrian, which determines how its velocity changes per unit time, is then defined as the sum of the three forces:

$$\frac{\partial \vec{v}_i}{\partial t} = \vec{f}_i^0 + \vec{f}_i^{wall} + \sum_j \vec{f}_{ij} \tag{5.8}$$

5.4 PEDESTRIAN AND ROBOT MOTION

Since the SFM robot control can still lead to collisions, especially in scenarios with dense crowds of pedestrians, we propose to train a neural network that outputs improved acceleration commands. To learn the NN, we use a simulator that models the pedestrians and robot motion according to the SFM. However, we modify Eq. (5.8), that determines the pedestrian and robot motion, by applying weights F_{df}, F_{of}, F_{sf} to each component:

$$\frac{\partial \vec{v}_i}{\partial t} = F_{df}\vec{f}_i^0 + F_{of}\vec{f}_i^{wall} + F_{sf}\sum_j \vec{f}_{ij} \tag{5.9}$$

By tuning the weights, we make sure that all our experiments show reasonable behavior of the pedestrians without lumping groups or traffic jam effects.

5.4.1 Robot Model

We assume a non-holonomic robot and define the robot state as tuple of the global xy-position and orientation θ. The robot control is determined by v^x and v^θ, which are its linear velocity in the heading direction and the angular velocity around the vertical axis, respectively. Thus, the trajectories for any constant velocity have a curvature with the radius $r = v^x/v^\theta$ and the new state of the robot at time step $t + \Delta t$ is determined as:

$$\begin{pmatrix} x(t+\Delta t) \\ y(t+\Delta t) \\ \theta(t+\Delta t) \end{pmatrix} = \begin{pmatrix} x + r\big(\sin(\theta + \Delta t \cdot v^\theta) - \sin(\theta)\big) \\ y + r\big(\cos(\theta + \Delta t \cdot v^\theta) - \cos(\theta)\big) \\ \theta + v^\theta \Delta t \end{pmatrix} \tag{5.10}$$

The velocities are clamped to reasonable values, i.e., $|v^x| \leq 1.0$ m/s and $|v^\theta| \leq 1.0$ rad/s. Given the linear and angular accelerations a^x and a^θ we can compute the robot's velocities as

$$v^x = v_r^x + a^x \cdot \Delta t \tag{5.11}$$

$$v^\theta = v_r^\theta + a^\theta \cdot \Delta t \tag{5.12}$$

In our simulation, we determine a^θ based on the resulting social force vector and the robot orientation, and the robot rotational velocity.

$$a^\theta = K_p \cdot \frac{\Delta\alpha}{(\Delta t)^2} + K_d \cdot \frac{-v^\theta}{\Delta t} \tag{5.13}$$

where K_p and K_d denote the coefficients and $\Delta\alpha$ is the angle between the social force vector and the heading direction of the robot. To compute the linear acceleration a^x, we project the social force vector onto the heading direction of the robot and take the magnitude of the new vector as value for a^x, if $\Delta\alpha$ is less than or equal to 70°. That means, we only apply forward acceleration when the social force vector is within the field of view of the robot, whereby $\Delta\alpha = 0$ means the maximal acceleration. In case $\Delta\alpha$ is greater than 70°, we set a^x to the maximum possible negative value to slow down or stop the robot.

5.5 PREDICTIVE CONTROLLER

We developed a predictive controller that aims at improving the computed SFM commands. The predictive controller considers a set of adjustment values of the acceleration commands generated by the SFM controller and simulates the pedestrians as well as robot motion some time steps into the future, evaluates the resulting configuration based on an objective function, and chooses the best result. The best acceleration values are then used as teacher values of the neural network described in the next section.

In more detail, the predictive controller has knowledge about the positions and velocities of the pedestrians within a certain range around the robot and tries to improve the acceleration commands computed by the SFM controller at every time step by considering a set of adjustments, defined as the tuple of linear acceleration adjustment δa^x and angular acceleration adjustment δa^θ from a set of discrete possible adjustments tuples. The predictive controller then simulates the motion of the pedestrians as well as the robot for each considered adjustment for time Δt into the future and evaluates the resulting configuration with the following objective function

$$\begin{aligned}\Omega(\delta a^x, \delta a^\theta) = \\ -\alpha d_G(\delta a^x, \delta a^\theta) - \beta ped(\delta a^x, \delta a^\theta) - \gamma col(\delta a^x, \delta a^\theta),\end{aligned} \tag{5.14}$$

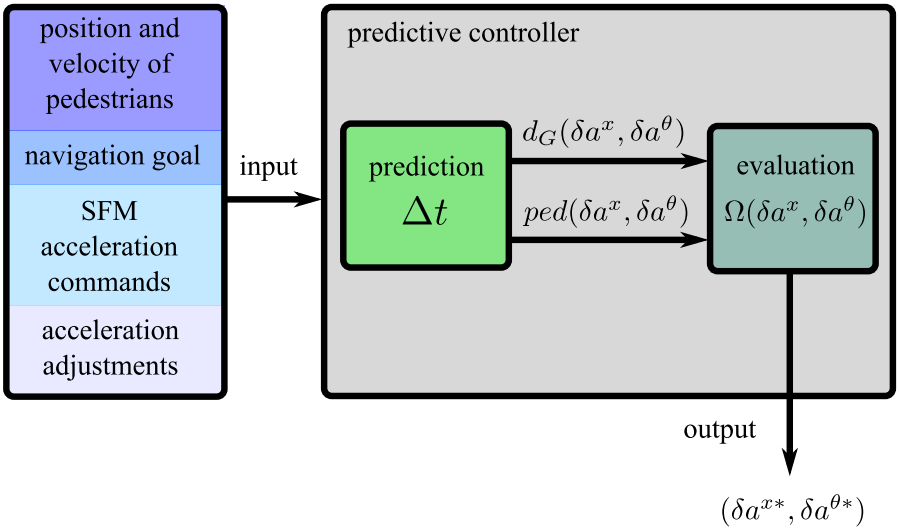


Figure 5.2: Overview of the predictive controller that outputs the best acceleration adjustment $adj^* = \{\delta a^{x*}, \delta a^{\theta*}\}$ for the calculated SFM command. The objective function Ω takes into account the progress towards the goal d_G and the resulting number of pedestrians within a certain range ped. See text for a detailed explanation.

where $d_G(\delta a^x, \delta a^\theta)$ is the resulting Euclidean distance from the robot to its goal, $ped(\delta a^x, \delta a^\theta)$ is the number of pedestrians that are predicted to be in a $3\,m$ radius around the robot, $col(\delta a^x, \delta a^\theta)$ is the number of collisions encountered during the simulation step, and α, β, and γ are weights.

The tuple $adj^* = \{\delta a^{x*}, \delta a^{\theta*}\}$ that achieves the highest evaluation is then added to the acceleration values returned by the SFM and chosen as the command to be executed by the robot. Fig. 5.2 illustrates the different components of the predictive controller.

5.6 TRAINING A NEURAL NETWORK

The predictive controller introduced in the previous section makes observations from its simulated environment which cannot be obtained in the real world. In addition, the calculations are too expensive to be computed on board and in real-time. We therefore propose to learn a neural network that can subsequently be used for a real robot to generate improved acceleration commands based on the SFM. The output of the predictive controller is hereby used as the training data.

We generated a training set by running our simulator and recording at each time step a set of features describing the situation and the adjustment tuple from the predictive controller. The final training set contained approximately 300,000 data pairs of features and acceleration adjustments and was collected with different densities of pedestrians.

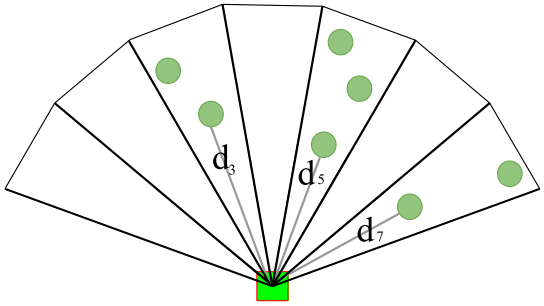


Figure 5.3: The features used as input to the neural network are the robot's observations of the pedestrians within its field of view. The robot's sensor field is divided into seven zones. The feature vector consists of the distance to the closest pedestrian per zone, as well as the number of pedestrians per zone. In this example, the vector would be as follows: [(0,0),(0,0),(d_3,2),(0,0),(d_5,3),(0,0),(d_7,2)]

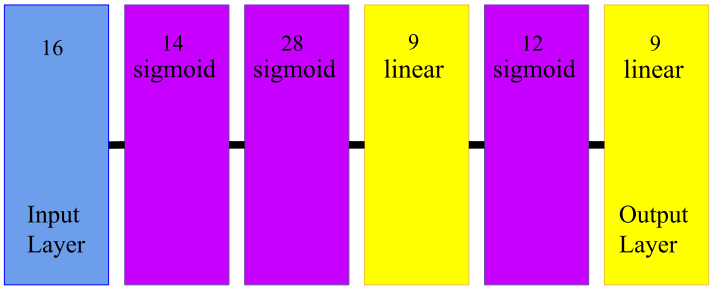


Figure 5.4: The neural network architecture used for the robot controller (input layer in *blue*, layers of linear neurons in *yellow*, and layers of sigmoid neurons in *purple*).

Since typically the velocity of the people cannot be reliably observed, we use a different set of features to describe the configurations of pedestrians around the robot based on the available information. In particular, we define a field of view in front of the robot with an opening angle of 140° with a 3 m radius and divide this area into seven equally sized angular zones as depicted in Fig. 5.3. For each of these zones, we determine the distance of the robot to the nearest pedestrian as well as the number of pedestrians currently within the area. We use this information to describe the local environment around the robot during training and application of the NN.

As a result, the input dimension of the neural network is 16 (2 features for each of the 7 zones and the linear and angular SFM acceleration) and the output, corresponding to the acceleration values $\delta a^{x*}, \delta a^{\theta*}$ given by the predictive controller, has a dimension of 2. An example how the feature vector is computed is provided in Fig. 5.3.

We used the mean-square error as a loss function and optimized training using RMSProp [65]. The network as depicted in Fig. 5.4 achieved the best validation loss rates for all architectures we evalu-

ated, with an overall training time of only 10 minutes with an Intel Core i5 CPU for 300.000 collected data points.

The weights of the neural network are initialized with the Glorot normalization [66]. The output layer of the neural network has two linear neurons corresponding to δa^x or δa^θ. Note that in general it might also be possible to train absolute velocities which, however, would require a much larger data set for training and a more complex network architecture.

5.7 EXPERIMENTS

We performed extensive experiments in simulation to evaluate the performance of the neural network controller in comparison to the basic SFM and the predictive controller in terms of number of collisions and completion time of the navigation task.

5.7.1 Parameters and Setup

For the SFM we used the following parameters: $A = 1.0, n = 2, n' = 3$ in Eq. (5.7). The desired velocity was set to $v_i^0 = 0.8\,\mathrm{m/s}$ and the parameters of Eq. (5.9) were set to $F_{sf} = 2.1$, $F_{df} = 1.0$, $F_{of} = 1.0$.

For the predictive controller, we experimentally determined the range of $\delta a^x, \delta a^\theta \in [-0.3, -0.2, -0.05, 0.0, 0.05, 0.2, 0.3]$ for the adjustment values of the predictive controller and thus considered 49 adjustment tuples. For the objective function in Eq. (5.14), we chose $\alpha = 5, \beta = 1, \gamma = 6$. These values showed the best performance for the predictive controller. The prediction time was set to $\Delta t = 0.4\,s$ and the simulated environment ran with a frequency of 10Hz.

Our experimental environment consists of a corridor with a width of $10\,m$, bounded by two parallel walls. The robot's starting and goal position across all runs was the same with a distance of $21\,m$ between these two points. We performed the evaluation for six different pedestrian densities. For each run, we distributed the pedestrians equally at either end of the corridor and randomly initialized their position in a certain region (Fig. 5.5). The goal positions of the pedestrians are randomly sampled along the width of the corridor at the opposite end.

For each of the six density groups we performed 50 runs using the SFM controller, the predictive controller, and the neural network controller in combination with the SFM.

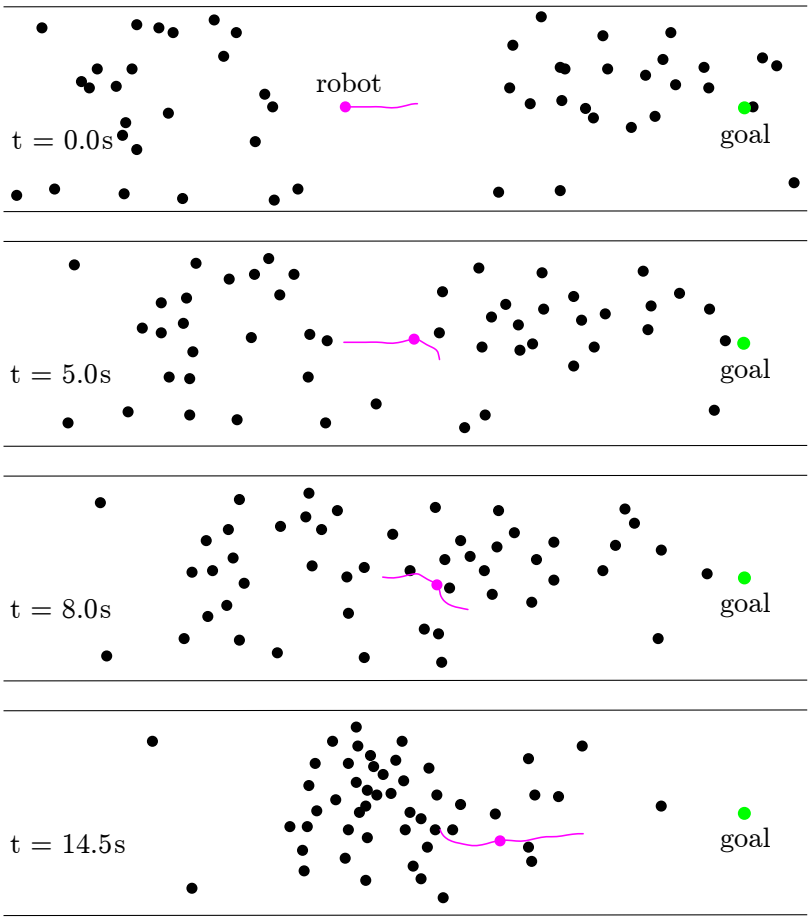


Figure 5.5: Experimental setup with 50 pedestrians (black) and a corridor size of 50x10 meters. The pedestrians on the left side have to reach the right end of the corridor and vice versa. The robot (magenta) starts in the middle of the corridor and has to reach the goal on the right (green). The resulting trajectory is illustrated for five seconds before and after each shown time step.

5.7.2 Average Number of Collisions for Different Pedestrian Densities

We first evaluated the average number of collisions between the robot and pedestrians for all three controllers. We hereby assume a collision to happen when the outer boundaries of the robot and a pedestrian come into contact, where both radii were set to 0.2 m. The results depicted in Fig. 5.6 show that both, the predictive controller and the neural network controller achieve a substantially reduced collision rate for all densities compared to the basic SFM controller. Over all runs, the NN controller shows a reduction of 31% in the number of collisions compared to the SFM controller. Most importantly, we show that the NN controller achieves a performance comparable with the predictive controller, even if it uses simple features instead of the complete

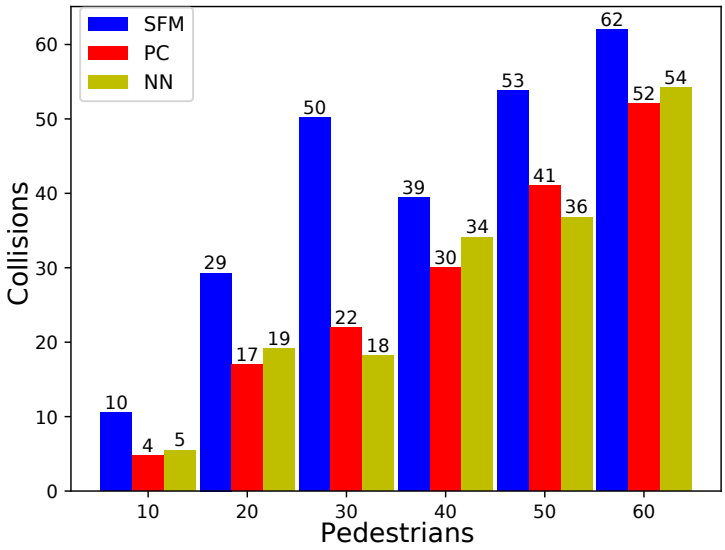


Figure 5.6: Average number of collisions per run for the basic SFM controller (SFM), the predictive controller (PC), and the neural network (NN) for different pedestrian densities. Both, the predictive controller and the neural network considerably outperform the basic SFM for all densities.

information from the simulation. As these results demonstrate, the NN successfully imitates our predictive controller and substantially improves the SFM commands.

5.7.3 Average Completion Time for Different Pedestrian Densities

Furthermore, we compared the completion time of the robot when controlled by the SFM controller, the predictive controller and the neural network to navigate through different pedestrian densities.

The results illustrated in Fig. 5.7 show that the NN controller achieves a completion time that is comparable to the one of the PC controller on average and outperforms the SFM controller up to 50 pedestrians. The evaluation confirms again that our NN is able to imitate the behavior of the predictive controller. For 60 pedestrians the completion time of the NN approach is higher than the time of both, the SFM and the PC controller, suggesting that additional training for very dense crowds is needed.

5.7.4 Qualitative Evaluation

Fig. 5.5 shows how the robot navigates through a pedestrian crowd of 50 people towards its goal location (green). The magenta line illustrates the robot trajectory five seconds before and after the current time step. As can be seen, the robot moves smoothly through the pedestrians by applying the acceleration adjustments of the NN. The reduced number

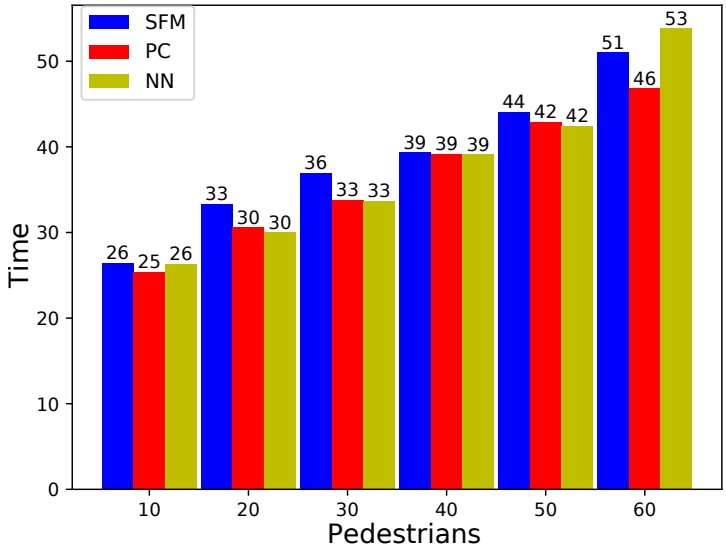


Figure 5.7: Average completion time for the SFM controller, predictive controller (PC), and the neural network (NN) for different pedestrian densities. As can be seen, there is no significant difference between the three controllers up to 50 pedestrians even if the NN and PC controller manage better to avoid collisions with pedestrians (as shown in Fig. 5.6). Only for 60 pedestrians the completion time of the NN controller is larger than the time of the PC controller, suggesting that additional training for very dense crowds is needed. These results demonstrate that the behavior of the predictive controller can be mimicked successfully by the NN controller, which is not using the omniscient simulation for improving control commands.

of collisions of the NN controller results from foresighted behavior in terms of early evasive actions into areas with only few pedestrians (see the example Fig. 5.8).

5.8 CONCLUSION

In this work, we proposed a novel robot controller based on a neural network to improve the navigation behavior of a robot steered by the popular social force model (SFM). To train the neural network, we developed a predictive controller that uses information about the positions and velocities of pedestrians within the vicinity of the robot and simulates their motions a few time steps ahead. For the robot's motions, the predictive controller evaluates a set of incremental adjustments of the SFM computed acceleration commands and determines the best result based on the number of close pedestrians and the progress towards the navigation goal. The best acceleration adjustments values are then used as target values for the neural network. As input, we use simple features describing the configuration of the pedestrians in the robot's vicinity.

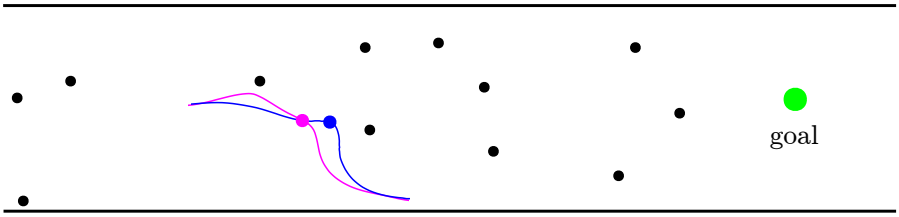


Figure 5.8: Qualitative comparison of the NN controller and the SFM controller for the same pedestrians configuration. The NN controlled robot initiates the evasive action earlier (magenta) than the SFM controlled robot (blue).

The learned network can be efficiently applied on a robot in combination with an SFM based controller and uses only features that can be derived from real observations. As our simulation experimental results with different pedestrian densities show, the motion commands generated by the predictive controller as well as by the neural network lead to a significantly reduced collision rate in comparison to the basic SFM controller while maintaining a comparable velocity.

For the presented approach in this chapter, we chose supervised learning to train a neural network. However, a great deal of work was previously necessary to tune all the parameters of the SFM model [63]. The tuning of system parameters is a common necessity for almost all mobile robots, that is time consuming and know-how-intensive. To circumvent the requirements of tuning, we solve the navigation task in Chapter 6 with methods of reinforcement learning (RL). RL deploys a reward function to train a network based on trial-and-error procedure, instead of using teaching examples from the predictive controller as in this approach. We will show in Chapter 6 how a policy can be learned completely autonomously for static environments based on robot's experience.

6

IMPROVING NAVIGATION BY DEEP REINFORCEMENT LEARNING

Most approaches to collision-free and efficient navigation with wheeled robots require parameter tuning by experts to obtain good navigation behavior. In this chapter, we aim at learning an optimal navigation policy by deep reinforcement learning to overcome this manual parameter tuning. Our approach uses proximal policy optimization to train the policy and achieve collision-free and goal-directed behavior. The output of the learned network are the robot's translational and angular velocities for the next time step. Our method combines path planning on a 2D grid with reinforcement learning and does not need any supervision. Our network is first trained in a simple environment and then transferred to scenarios of increasing complexity. The experiments illustrate that the learned policy can be applied to solve complex navigation tasks.

6.1 INTRODUCTION

To fulfill the prerequisite for mobile robot applications of collision-free navigation, typical solutions apply a two-stage approach as we discussed in Chapter 2. As stated before, the goal of the first stage is to compute a 2D path on a cost grid. In the second stage, the robot uses a low-level motion controller for path tracking and collision avoidance. The low-level controller hereby determines the motion commands for the current time step taking into account the global path and the current robot state as well as the local environment. Typical navigation systems require manual parameter tuning to achieve a good navigation behavior. This tuning requires a significant amount of time and profound knowledge about the navigation software, the robot hardware, as well as the environment conditions, and is a difficult task due to the trade off between time efficiency and safety.

In this work, we present a self-learned navigation controller realizing collision avoidance and goal-directed behavior. Our approach combines grid-based planning with reinforcement learning (RL) and applies proximal policy optimization (PPO) [67] for the learning task. Our framework hereby uses a global planner to obtain a 2D path from the current robot pose to the global goal. The input of the network consists of the robot's translational and angular velocities, local goals determined from the global path, and a patch of the occupancy grid

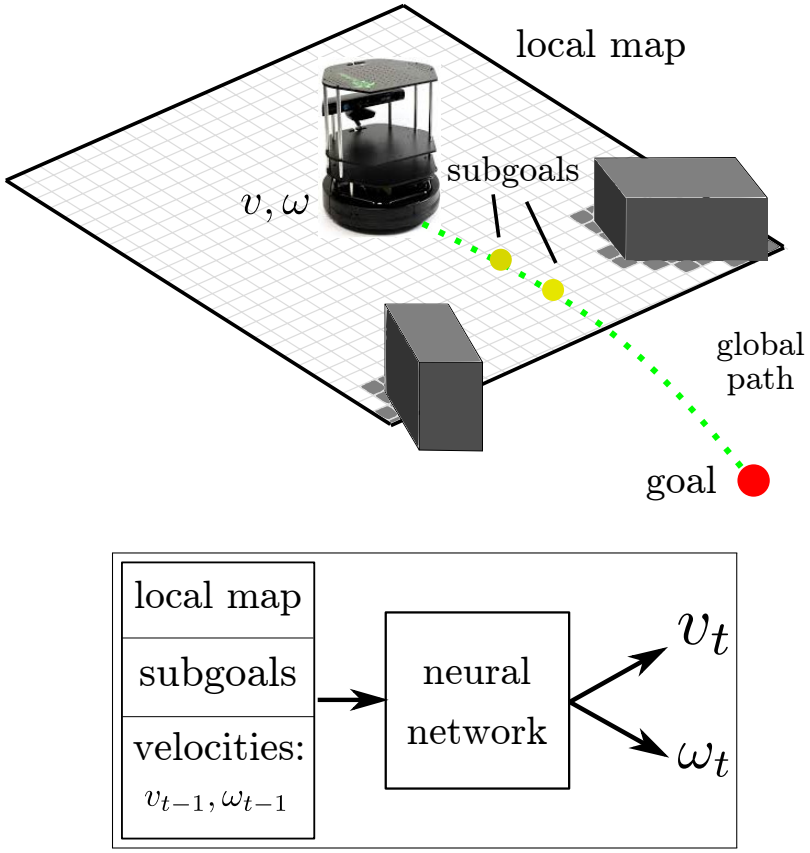


Figure 6.1: Visualization of our approach, which relies on a global path that is recalculated on a 2D occupancy grid every time step. We use subgoals on the path as part of our observation space and input to a neural network. Additionally, we use a local grid map centered around the robot and the current translational and angular velocities v_{t-1} and ω_{t-1} as network input. We train the network to learn a navigation policy that outputs the velocity commands v_t and ω_t for the next time step.

map containing the obstacles in the robot's vicinity. The outputs are the robot's velocity commands for the next time step. Fig. 6.1 shows an example situation and a visualization of our approach.

To the best of our knowledge, we present the first solution that integrates global path planning with deep RL to reach a global goal in the environment. Our framework thereby learns the appropriate distance to obstacles and performs regular recomputation of the global path. As a result, no parameter tuning of the navigation controller and inflation (Sec. 2.2) of the objects in the map is needed to find the best trade-off between completion time and safety distance to obstacles. When the global path leads through narrow passages, where collisions are likely to occur, our system learns to drive around those narrow regions.

We integrated our learned navigation policy as a collision avoidance module that can be used with the Robot Operating System (ROS) [24]

navigation stack [24]. We thoroughly evaluated our approach in simulation and in a real-world experiment and compared the performance with the dynamic window approach (DWA) (Sec. 2.5), which is used in ROS and which is still one of the most popular navigation schemes. As the experimental results show, the robot steered by our learned policy reaches the goal significantly faster than using the DWA. Furthermore, we show that the policy, which is initially trained only on a simple environment, can be transferred to environments of increasing complexity.

6.2 RELATED WORK

In the last few years, several learning approaches for mobile robot navigation have been presented. Sergeant *et al.* [68] proposed to use a state representation based on laser range data and learned translational and angular velocity commands for local obstacle avoidance. The authors trained an autoencoder neural network with human-controlled action commands. Pfeiffer *et al.* [69] presented an end-to-end navigation system learned for simple maps using 2D laser data as input, the velocity commands as output, and a 2D path as teacher. The authors later extended the work by applying subsequent RL training to the learned model [70], thus reducing the training time of RL and avoiding overfitting of the imitation model. Liu *et al.* [71] used a local occupancy map as state representation to learn a navigation policy using a variant of the value iteration networks. Tai *et al.* [58] proposed generative adversarial imitation learning to achieve socially complaint navigation. The authors use depth data to train the network and the social force model to generate a large set of training data. Pokle *et al.* [72] designed a local controller to determine the robot's velocity commands and predicting a local motion plan, while considering the trajectories of surrounding humans. These supervised learning methods all depend on the teacher, e.g., controls provided by humans, a global path planner, or a well-tuned optimization, while the goal of our work is to enable the robot to learn by itself while navigating in the environment.

Gupta *et al.* [73] investigated a mapping and planning navigation network based on visual data that encodes the robot's observations into a birds-eye view of the environment, which makes the method limited to known scenarios. Also the approach presented by Hsu *et al.* [74] was developed for known environments. A CNN processes image data and generates discrete actions to move the robot towards a global goal pose. In contrast to that, we use a binary occupancy grid map as representation, which makes the learned policy applicable to environments not seen in the training data.

Chen *et al.* [75] deployed also PPO for deep RL as we do. The authors rely on height-map observations as state representation for a wheel-legged robot. Due to a high-dimensional robot state, the authors

discretize the action space and use a set of navigation behaviors to deal with obstacles of certain, given shapes.

Tai *et al.* [76] presented a method that utilizes the robot's velocities and target positions as state representation for an actor-critic RL approach. The authors developed a local controller relying on sparse laser-range measurements and trained a mapless motion planner. Fan *et al.* [77] proposed to use a set of subsequent laser scans and apply PPO to learn movement commands for navigation through crowds. Those approaches do not consider global path planning as they are designed for local navigation.

Chiang *et al.* [78] applied AutoRL to learn two different navigation behaviors, i.e., path following and driving to a global goal location. The authors do not combine learning with global path planning but use the global goal coordinate as input to the network. In our experiments, the robot got stuck in local minima while using only the global goal as input. Therefore, we use a subgoal on the regularly recomputed global path as additional input.

6.3 PROBLEM DESCRIPTION

We consider a robot moving according to the unicycle model that has to reach a goal location by executing translational and angular velocities. A path planner computes the 2D path to the goal on a global grid map at every time step using the estimated robot pose from a localization system. The RL learning task is to determine the velocity commands for each time step to navigate collision-free and as fast as possible to the goal.

We model the problem as a partially observable Markov decision process (POMDP) defined as the tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. Here, $s \in \mathcal{S}$ corresponds to the state of the environment including the robot. The state of the environment changes based on the robot's actions $a \in \mathcal{A}$, which are in our case the translational and angular velocity commands (v, ω), and according to the transition probability $\mathcal{T}(s'|s, a)$. The agent cannot determine the state s but has to rely on observations $\mathcal{O}(o|s', a)$. After every state transition the robot receives a reward $\mathcal{R}(s, a)$.

The actor critic approaches approximate the value function (critic) to be able to update the policy (actor) itself. We use a deep neural network as non-linear function approximator to evaluate the state value function V^π, which determines the expected return for state s when following the policy π. The goal of RL is to find a stochastic policy $\pi_\theta(a_t|o_t)$ that maximizes the expected reward

$$\max \mathbb{E}\left(\sum_{k=0}^{T} \gamma^k \mathcal{R}(s_k, a_k)\right),$$

where θ is the set of parameters that specify the function approximator, T is the final time step, and γ is the discount factor.

The critic network is updated based on the advantage value

$$A_t = A(o_t, a_t) = Q^{\pi_\theta}(o_t, a_t) - V^{\pi_\theta}(o_t). \tag{6.1}$$

where $Q^{\pi_\theta}(o_t, a_t) = r_t + \gamma V^{\pi_\theta}(o'_t)$. Here r_t is the immediate reward at time t and $V^{\pi_\theta}(o'_t)$ is the expected return for the observation o'_t. The actor network uses the policy gradient (PG) method to update the network weights θ in order to maximize

$$\max \mathbb{E}(\log \pi_\theta(a_t|o_t) A_t). \tag{6.2}$$

Proximal policy optimization (PPO) [67] substitutes the $\log \pi_\theta$ term for the policy probability ratio $\Psi = {}^{\pi_\theta}/_{\pi_{\theta_{old}}}$, to achieve stability. To avoid large policy updates that can impede and reset the training process, the probability ratio is constrained to the range of $[1-\epsilon, 1+\epsilon]$ via the $clip$ function

$$\eta^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(\Psi \mathrm{A}_t, \mathrm{clip} \left(\Psi, 1-\epsilon, 1+\epsilon \right) \mathrm{A}_t \right) \right]. \tag{6.3}$$

6.4 NEURAL NETWORK APPROXIMATOR FOR LOCAL NAVIGATION

To learn the navigation strategy that takes into account the global path to the goal and the obstacles in the robot's vicinity, we train a deep neural network approximator that provides the robot's translational and angular velocities. The architecture of this network is described in the following.

6.4.1 Observation Space

The observation space consists of three components. The first component is $o_v = (v_{t-1}, \omega_{t-1})$ with v_{t-1} and ω_{t-1} as the robot's current translational and angular velocities computed at the previous time step. The second component is o_m, which corresponds to the $3\,m \times 3\,m$ patch of the 2D occupancy grid map around the robot (see Fig. 6.2). As resolution of the map we use 0.05 m, thus the grid patch size has a dimension of 60×60 cells.

Additionally, we use a representation of local 2D subgoals in the observation. The subgoal at the current time step is calculated as the position on the global path that is $1m$ away from the robot and stored in map coordinates. At time step t, we transform the global coordinates of the subgoals stored at time steps $t-1$ and $t-5$ into the robot frame to get their relative positions, which serve as third observation component $o_g = (p^x_{-5}, p^y_{-5}, p^x_{-1}, p^y_{-1})$. The representation of the local goal p^x_{-1}, p^y_{-1} indicates the robot's progress that was made

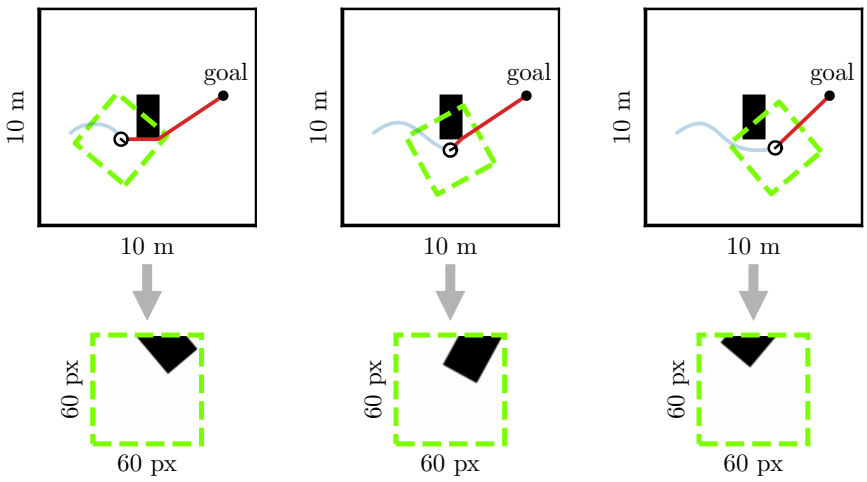


Figure 6.2: Binary image representation used as input to the network. A $3\,m \times 3\,m$ patch (dashed green) around the robot's pose is cropped from the global occupancy grid map. The robot is at the center of the resulting egocentric image and the viewing direction is to the right side. The global path (red) is computed with the A* search in a binary global map. Interpolated values in the cropped image resulting from the rotation are set to occupied as well as regions outside the boundaries of the global map.

towards the goal since the previous time step and is used for the reward calculation. By adding a second subgoal to the observation space $\mathcal{O}$, we noticed an improvement of the navigation policy and speed up of the training. As already noted by Kulhánek *et al.* [79], using information of previous observations helps the system to infer the real state $s \in \mathcal{S}$ of the environment. To summarize, an observation is defined as $o = (o_g, o_v, o_m)$.

6.4.2 Reward

Our reward function considers task completion, the duration, and the progress towards the goal

$$\mathcal{R}(s,a) = \mathcal{R}_{fin}(s,a) + \mathcal{R}_{fix} + \mathcal{R}_{dist}(s,a). \tag{6.4}$$

A navigation task ends if the robot arrives at the goal, a collision occurs, or a maximum number of time steps is reached. Accordingly, the reward $\mathcal{R}_{fin}(s,a)$ is defined as follows:

$$\mathcal{R}_{fin}(s,a) = \begin{cases} b & \text{if the goal was reached} \\ -c & \text{if a collision occurred} \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

$\mathcal{R}_{fin}(s,a)$ is a large positive value if the distance to the final goal is less than $0.3\,m$, a large negative value if the distance between the robot and the nearest obstacle is less than $0.3\,m$, meaning a collision is occurred, and zero otherwise.

$\mathcal{R}_{fix}$ is a fixed negative reward, that penalizes each action to force the robot to finish an episode as fast as possible.

To speed up the training, we use a third reward component

$$\mathcal{R}_{dist}(s,a) = \alpha \cdot \mathcal{D}_{(s,a)}, \tag{6.6}$$

where $\mathcal{D}(s,a)$ is the function computing the distance between the robot and subgoal (p^x_{-1}, p^y_{-1}) and α is a scaling factor.

6.4.3 Neural Network Structure

Our observation space as described in Sec. 6.4.1 is divided based on the representation of the data. Typically, the obstacle grid around the robot is represented as a binary image, while the rest of the observation space provides information about the different components of the robot state. Thus, we propose a network architecture that consist of two branches that split the observation space into scalar values and the binary grid patch (left part of Fig. 6.3). The scalar branch of the network is a single, fully connected neural network layer (green layer in the upper branch in Fig. 6.3) and encodes the subgoals and robot velocities into a high dimensional feature space to process them in the following layers.

The grid patch is processed by separate convolutional neural network (CNN) layers (lower branch in Fig. 6.3), that are well suited for processing 2D data structure, e.g., images. The layers can identify 2D relationships between pixel values and encode obstacles in the robot's vicinity. Max-pooling layers after the first two CNNs reduce the shape and compress the information. This layered design is inspired by the network composition of the well-known VGG-networks for image recognition [80]. The 3D output of the last max-pooling layer is flattened and reduced to a one-dimensional output with another dense layer (shown in blue). Then, we concatenate the outputs of both branches (blue and green) and process them together in an fully connected layer. Finally, we normalize the output, which is a standard technique [81].

The actor and critic estimators share the same connected layers. We found out that the parameter sharing between the actor and critic improves the learning speed, because there are fewer parameters to learn. For the value function estimator $v_\theta(o_t)$, the shared network output is inserted into a last dense layer to get a single real number which represents the critic value. The final output of the actor network is the policy $\pi(a|o)$ modeled by the two Gaussian distributions $\mathcal{N}(\mu_{trans}; \sigma_{trans})$ and $\mathcal{N}(\mu_{ang}; \sigma_{ang})$. The two mean values are

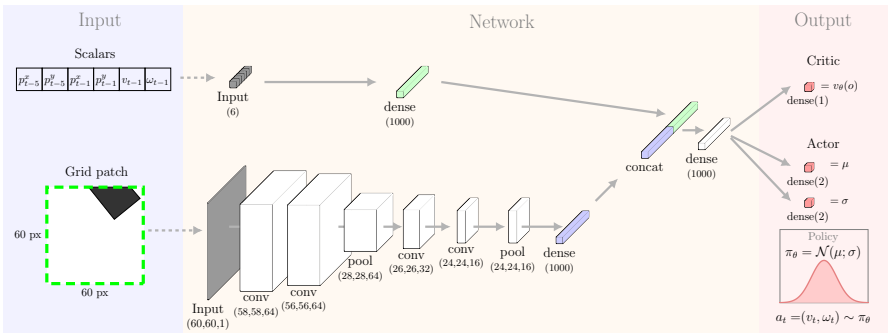


Figure 6.3: Network structure of the actor-critic scheme. The input consists of scalar values and the grid patch. The scalar values are fed into a single, fully connected dense layer. The binary image of the grid patch (see also Fig. 6.2) is handled by multiple CNN layers to distinguish obstacle configurations. Then, both branches are concatenated and assembled in a further dense layer. Finally, the critic value $v_\theta(o)$ corresponding to the value function estimator is computed by a last layer. The policy distribution π is calculated by the mean and standard deviation of two normal distributions from which v_t and ω_t are sampled.

shrunk with an *tanh* activation function. This scaling forces the values to stay between the desired velocity limits ($[0 : 0.7]$ m/s and $[-0.7 : 0.7]$ rad/s). The σ values are the standard deviations of the normal distributions. We apply a sigmoid activation function scaled with 0.5 to guarantee that the bandwidths of the normal distributions do not massively grow.

6.5 EXPERIMENTS

The implementation of our framework is based on several components. As communication backbone, we used ROS and for the RL approach, we created a simulation environment with *Gazebo* [39]. We implemented the RL in *Python* with the *Tensorflow* library [82]. As mobile platform, we use the Robotino robot by Festo [30].

6.5.1 Training

To train the neural network and learn a policy to follow a global path and reach a goal without collisions, we used a simple environment (see Env1 in Fig. 6.4). During the training, we sampled the start and goal positions randomly across the free space, where we chose start-goal configurations with a short Euclidean distance at the beginning and later increased the distance for more challenging scenarios. This helps the robot initially to reach preferable states and learn basic navigation in free space, while longer start-goal configurations force the robot to deal with obstacles, as suggested in [83].

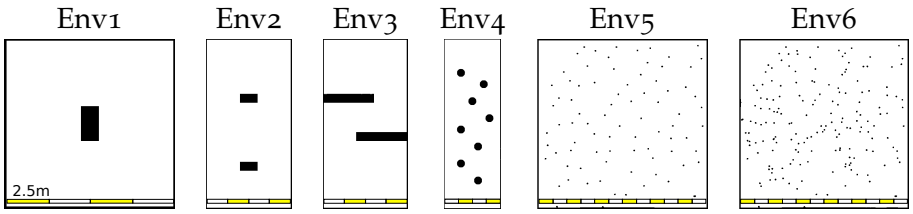


Figure 6.4: Environments used for training and evaluation. The policy was initially trained only in Env1 and evaluated in all other environments. Afterwards, we used further episodes from Env6 to improve the navigation behavior in highly cluttered scenes as in Env5 and Env6. The evaluation results are depicted in Tab. 6.1 and Fig. 6.6.

We used four simultaneously operating robots to ensure our collected data is independent and identically distributed. Each robot was given different start and goal configurations. Every episode was limited to 1000 time steps, the batch size was 32 and the entire training involved 10^6 episodes. In Eq. (6.5), we set the final reward b to 10, c to 50, r_{fix} in Eq. (6.4) to -0.1, and α in Eq. (6.6) had value of 10, as experimentally determined. The controller run with a frequency of 10 Hz during training and testing. The overall training time was about 24 hours using a *Nvidia GeForce GTX 1080*.

6.5.2 Evaluation

After training, we performed experiments in different environments to evaluate the policy learned in Env1 in terms of number of successful runs, which means that the robot reached the goal without collisions, and completion time, both in comparison to the standard ROS navigation stack. The latter uses the DWA [21] to calculate the robot's velocity commands. We configured the DWA with similar restrictions to guarantee similar conditions in terms of acceleration and velocity limits and application of the unicycle robot control. The translational velocity was limited between 0 and 0.7m/s and the angular velocity between $-0.7\,\mathrm{rad/s}$ and $0.7\,\mathrm{rad/s}$. The acceleration limits for translational and angular steering were set to $1\,\mathrm{m/s^2}$ and $1\,\mathrm{rad/s^2}$ for both approaches.

The DWA approach needs an inflation radius around obstacles in the 2D grid map (Sec. 2.2). This corresponds to a general safety distance to prevent collisions that could result, e.g., from the discretization of the environment. The inflation parameters usually need to be tuned to achieve a good trade-off between safety and time performance (Sec. 3.3.1). One advantage of our approach is that it works on a binary map of the environment without any inflation. Our approach

Env1	Env2	Env3	Env4	Env5	Env6	Env5*	Env6*
1.0	1.0	0.99	0.99	0.75	0.22	1.0	0.84

Table 6.1: Success rate of the trained policy. The evaluation consists of 400 runs for each of the environments shown in Fig. 6.4. A successful run means that the robot reaches the goal within a certain time limit without any collision. To improve performance in Env5 and Env6, we continued to train the policy on Env6. As shown in the last two columns (Env5* and Env6*), the results of the re-trained policy were seriously better.

directly learns the appropriate distance to obstacles depending on their local configuration.

Fig. 6.4 depicts the environments we used in the evaluation. Each map introduces a further level of difficulty. Env2 is similar to the training environment Env1 but the length of the room is doubled and an additional obstacle occurs in the center. The large walls of Env3 can lead the robot into local minima if no global path is used. This map is well suited to test the performance of the learned policy in terms of a reduced completion time while avoiding collisions since fast movements on circular arcs around the obstacle corners are needed to achieve a good navigation behavior. Env4 introduces round obstacle shapes not experienced before. Env5 and Env6 consists of several regions with a high obstacle density. In those maps, it is not always possible to follow the global path computed on a map without obstacle inflation, since the path might lead through regions with very close obstacles. Thus, the robot has to learn to bypass the corresponding region by moving away from the global path.

6.5.3 Success Rate

For each environment in Fig. 6.4, we performed 400 runs with the DWA and with our trained policy. The robot's start and goal configurations were sampled randomly for each run but were the same for the two approaches. The DWA controller was able to reach all goals in all environments without any collisions. The success rates of our trained policy are listed in Tab. 6.1. Our approach performs equally well in Env1 to Env4. In Env5 and Env6 the performance decreases due to an insufficient generalization resulting from Env1, that leads to increased collision rates. We discovered that situations in which the robot has to depart from the global path did not occur in Env1 and, thus, the robot could not learn a suitable strategy to handle those situations.

To overcome this limitation, we continued the training process with the so far learned policy parameters θ on Env6. After only 8000 further trained episodes (which corresponds to not even 1% of the initial size

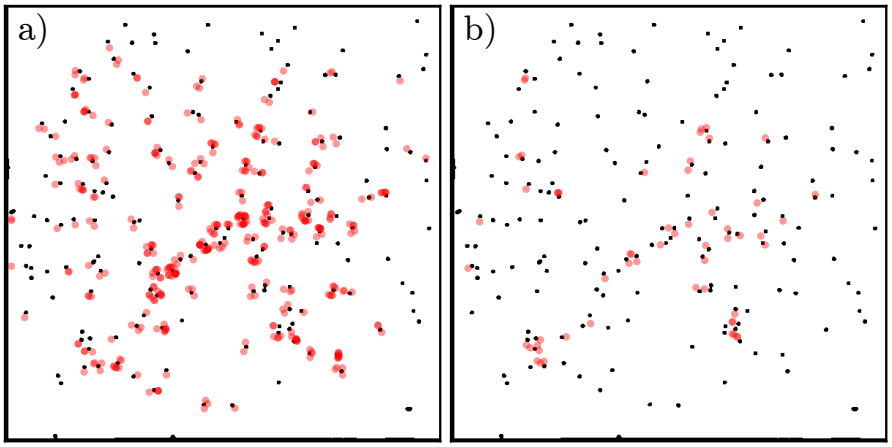


Figure 6.5: Performance improvement in Env6. (a) Collisions in Env6 with the policy trained on Env1. (b) The policy resulting from the additional training in Env6 shows much fewer collisions.

of the training set), the performance improved significantly and the results are shown in the last two columns of Tab. 6.1. In Env5 we could achieve a success rate of 100% with the newly learned policy and in Env6 the robot now reached the goal in 84% of all runs (the results are denoted as Env5* and Env6* in Tab. 6.1).

The left image of Fig. 6.5 visualizes for Env6 the positions where the robot collided with obstacles when following the policy learned on Env1. The right image of Fig. 6.5 shows the collisions after further training on Env6. As can be seen, fewer collisions appear in regions with high obstacle density. The reason is that the robot learned when it is beneficial not to follow the global path into narrow space but rather drive around depending on the obstacle configuration.

6.5.4 Completion Time

Next, we evaluated the completion time of the navigation tasks when using the standard DWA approach and our learned policy. Fig. 6.6 shows the average completion time for the runs from Sec. 6.5.3 that were successfully completed by both approaches. Our approach is 16% faster on average over all evaluated runs. The difference is statistically significant in Env3, Env4, and Env5 according to a *paired* t-test at the 0.05 level. One reason for the faster performance is that the robot learns the best distance to obstacles, which reduces the trajectory length and leads to time savings, especially in Env3 where our policy performs 26% faster than the DWA.

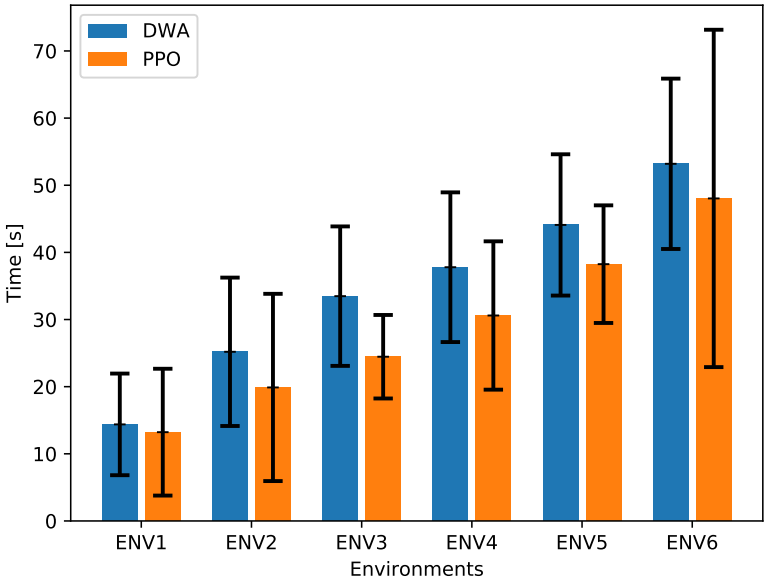


Figure 6.6: Average completion time for the DWA and our learned policy. The box height shows the average completion time and the whiskers illustrate the standard deviation. As can be seen, our trained policy outperforms the DWA in each environment. The difference is statistically significant in Env3, Env4, and Env5 according to a *paired* t-test at the 0.05 level.

6.5.5 Real-World Experiment

Finally, we applied our learned policy on a real robot and compared the performance to the DWA. In the experiment, the robot had to enter an office from the corridor and navigate around an obstacle to reach the global goal (see Fig. 6.7) by following subgoals on the path. An occupancy grid of the environment was mapped before and we applied Monte Carlo localization [84] to obtain the robot pose.

For the evaluation, we performed 10 experiments with similar start and goal configurations for both the standard DWA approach and our trained policy. With both approaches, the robot reached the goal in each run. The DWA approach needed 28.2 *s* on average to reach the goal location while our approach had a reduced average completion time of 25.5 *s*. The difference was statistically significant according to a *paired* t-test at the 0.05 level.

Our approach saves time by driving closer around obstacles while the standard DWA takes into account a general inflation radius around obstacles.

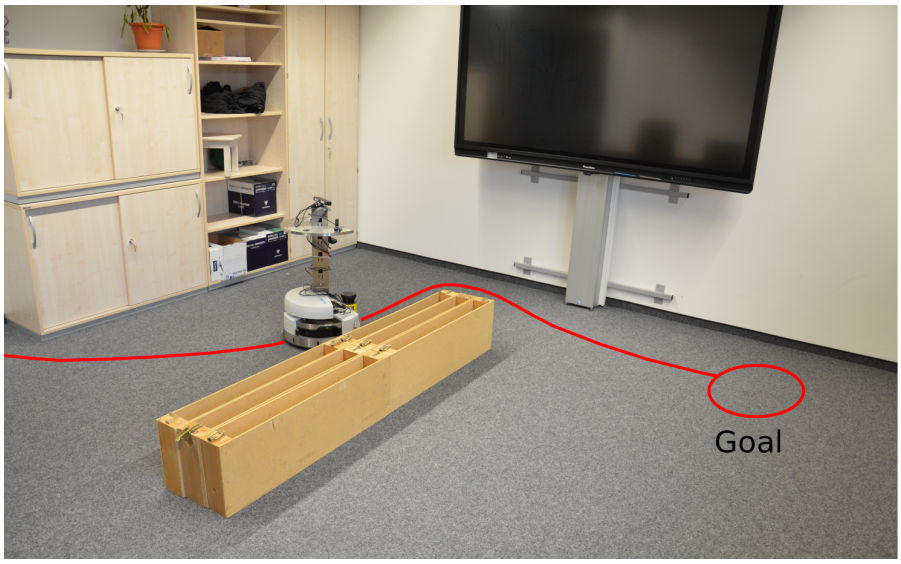


Figure 6.7: Real-world experiment. The robot entered the office from the left. Based on our learned navigation policy, the robot chooses the best translational and angular velocities to reach the global goal quickly while avoiding the obstacle in the center.

6.6 CONCLUSION

In this work, we proposed a new approach to learn a navigation policy for wheeled robots in an unsupervised manner. We use proximal policy optimization for reinforcement learning to train a network that provides the robot's translational and angular velocity commands for the next time step. Our solution combines global path planning with deep RL to navigate collision-free and reach a global goal in the environment.

Our policy was first trained in a simple environment and subsequently evaluated in environments with increasing complexity. The experimental results demonstrate that our network successfully learned collision-free, goal-directed behavior also in cluttered environments. Furthermore, we compared the performance of our trained policy to the popular dynamic window approach (DWA) with respect to completion time of navigation tasks. On average, the robot controlled by our learned policy completed the tasks 16% faster than the DWA of ROS. In our real-world experiment, we experienced similar results, i.e., the robot performs 10% faster than the DWA using our navigation policy. Our learned strategy safes time by keeping a closer distance to obstacles and choosing appropriate velocities. This is a direct result of the optimization of the motion commands based on the local configuration of the obstacles without any parameter tuning for the navigation controller.

So far, all presented approaches improve the navigation in clutter regarding the completion time and collisions. However, those systems can only operate in situations, where a obstacle-free path leads to the goal. When the route to the goal is blocked by an object, then the robot cannot complete the navigation task and reach its goal. Thus, it is required to incorporate robot actions into the navigation scheme, to provide the necessary capacities to overcome impeding objects. In Chapter 7, we present a new navigation approach, that involves phases of manipulation to free the path to the goal and footstep planning to step over objects in an efficient manner.

7

PATH AND ACTION PLANNING TO OVERCOME IMPEDING OBJECTS

Humanoid robots are often supposed to share their workspace with humans and thus have to deal with objects used by humans in their everyday life. Furthermore, they have the potential to execute complex navigation tasks and even overcome impeding scenarios by performing manipulation or advanced footstep planning. However, the robot's high degrees of freedom hinders the real-time computation of motion commands, e.g., the trajectories of the joints and the footstep sequence. In this chapter, we present our novel approach to humanoid navigation through cluttered environments, which exploits knowledge about different obstacle classes to decide how to deal with obstacles and selects appropriate robot actions. To classify objects from RGB images and decide whether an obstacle can be overcome by the robot with a corresponding action, e.g., by pushing or carrying it aside or stepping over or onto it, we train and exploit a convolutional neural network (CNN). Based on associated action costs, we compute a cost grid containing newly observed objects in addition to static obstacles on which a 2D path can be efficiently planned. This path encodes the necessary actions that need to be carried out by the robot to reach the goal. As the experiments demonstrate, using our CNN the robot can robustly classify the observed obstacles into the different classes and decide on suitable actions to find efficient solution paths. Our system finds paths also through regions where traditional motion planning methods are not able to calculate a solution or require substantially more time.

7.1 INTRODUCTION

As humanoid robots are designed to work in human environments, one of the tasks that need to be solved consists of navigating through a cluttered environment where stepping over or onto obstacles or moving an object out of the way might be necessary. Thus, they must be able to avoid or to deal with different types of objects located at random places in the environment. Finding suitable robot motions in environments cluttered with objects imposes a high level of complexity to the motion planning problem and is difficult to solve in a computationally efficient manner.

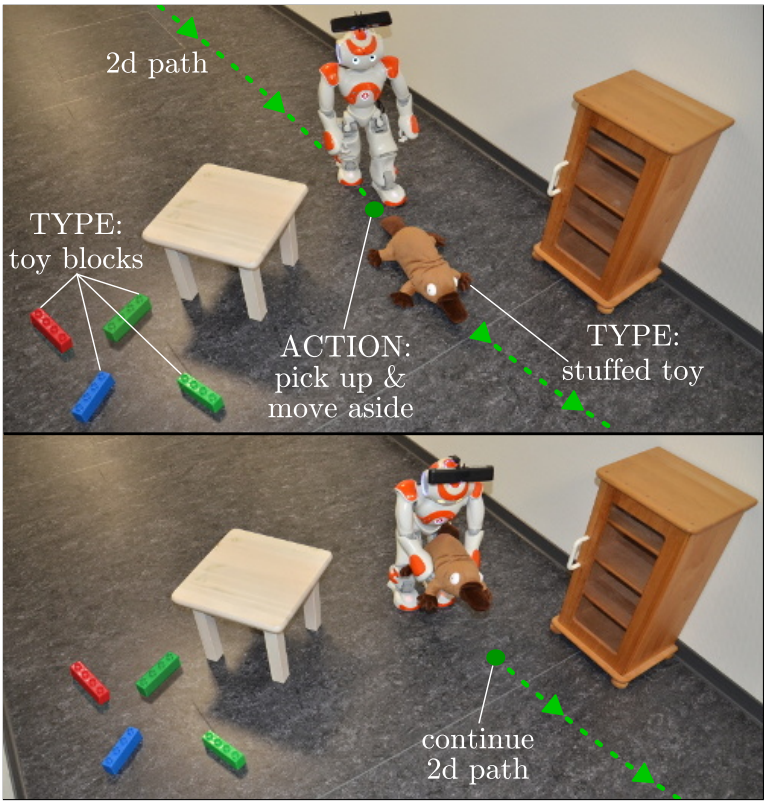


Figure 7.1: Application example, in which the path of the robot is blocked. Based on classified non-static obstacles (in this case a stuffed toy and toy blocks) and their associated actions and costs, our system computes a cost grid on which a 2D path can be efficiently computed. The path also encodes the actions that need to be executed by the robot during navigation to reach the goal. As can be seen, the robot moves the stuffed toy aside to clear its path and can then continue walking along the 2D path.

Common approaches to humanoid navigation in complex environments involve whole-body motion planning and multi-contact planning [85], [86]. These approaches usually take several seconds up to minutes to compute a solution. This may prevent the use on a mobile platform or limits its ability to operate in a reactive way. To limit the computational load, other frameworks employ footstep planning [87]–[90] to avoid computing whole-body motion plans. However, if a blocking object needs to be moved aside to reach the goal location, they often do not yield a solution and thus result in a navigation failure.

In this work, we combine the advantages of the existing planning approaches by exploiting semantic information about objects. We present a novel approach to humanoid navigation that combines fast 2D path planning with 3D footstep planning and object manipulation actions in obstructed regions of the path. We consider an indoor environment from which the robot creates a 2D grid map with static obstacles in the absence of clutter using a standard mapping approach [17].

During navigation, the robot adds information about objects of different classes perceived with its camera to the map. We hereby use a convolutional neural network to classify the objects and to decide whether an obstacle can be *stepped over* or *stepped onto*, or moved out of the way by *pushing* or *carrying* it aside. For the classification task, we use our recently developed real-time CNNs [9] that are capable of segmenting RGB images to detect given object classes. This framework provides reliable information about specific object types and their pixel-wise masks (e.g., books, boxes, toys etc.). We map that information about the object classes to appropriate action types, which allow the robot to navigate across the corresponding area. Our approach adds associated stepping and manipulation costs to a 2D map that is used for path planning. In this way, we greatly simplify the full planning problem, as we split the whole plan into several parts, while each part is solved individually. Fig. 7.1 illustrates a motivating example, where the robot can only reach its goal by manipulating an object. According to the resulting cost map after classifying the obstacles, our planner chooses a path where the robot needs to move the stuffed toy aside.

We implemented our framework in ROS and tested it in various experiments with a Nao and REEM-C humanoid. As the experimental results illustrate, the robot can robustly classify the observed obstacles into different classes and use this information to efficiently find solution paths through passages where objects are blocking the path. The underlying architectures used to extract the semantics from the environment were carefully tailored to this task in order to achieve efficient inference, which allows our approach to run on our small Nao humanoid.

7.2 RELATED WORK

Stilman and Kuffner were the first who considered navigation amongst movable objects where the robot can move objects aside if necessary to reach the goal location [91]. The idea of their approach is to decompose the environment representation into disjunct regions and search for obstacle motions that connect two disjoint regions allowing the robot to transition between them. Stilman *et al.* employed the framework on a real humanoid to navigate through an environment with movable chairs and tables [92] where the world state was observed by an external motion capture system. In contrast to this approach, we perform fast planning on a cost grid based on a 2D map, which can be easily obtained by using standard mapping algorithms [17]. Furthermore, we classify objects perceived by the robot and encode different actions associated to the object classes directly in the navigation costs contained in the cost grid. In this way, we combine the planning on a 2D cost grid with local 3D footstep planning and object manipulation.

A variant of the humanoid locomotion problem, in which the robot can utilize objects to get to locations that are otherwise out of reach for the robot, was approached by Levihn *et al.* [93]. The authors introduced the concept of relaxed-constrained planning where the planner is allowed to violate certain constraints. The violation is then locally resolved by using suitable objects, e.g., to overcome a high step height.

The task of collecting objects and delivering them to designated places while clearing cluttered obstacles out of the robot's way, was addressed by Hornung *et al.* [94]. The authors proposed to apply a high-level planner with integrated perception, world modeling, action planning, navigation, and mobile manipulation. Our navigation framework is orthogonal to that approach and could be integrated into such a higher level task planning framework.

Grey *et al.* recently presented an approach that uses so-called randomized possibility graphs to traverse environments with arbitrary obstacles, in which footstep as well as whole-body motion planning is required [85]. The authors distinguish between passages that are definitely passable by the robot and ones that are definitely impossible to be passed by using approximations of the constraint manifold. So far, their approach has only been tested in simulation with no perception involved. Lin and Berenson considered navigation in uneven terrain using contact planning of palm and foot locations and learned estimates about the traversability of regions [86]. The idea of this approach is to use the learned traversability estimates as a measure how quickly the planner can generate feasible contact sequences. This measure is used in the heuristic function of the contact space planner to guide the search to areas with more contactable regions. In the follow up work [95], the authors speed up the planning by applying fast discrete footstep planning for easy parts of the path and only using the advanced but slow contact space planning method for regions that are difficult to traverse. Dornbush *et al.* recently developed a planning framework that considers adaptive dimensionality by determining which planning dimensions are relevant in each region of the environment [96]. Their idea was to plan with multiple low-dimensional planning representations simultaneously within a multi-heuristic search. While these approaches also aim at speeding up the search for viable solutions paths, the authors do not consider perception and do not take into account the possibility of actively modifying the environment.

The method proposed by Kaiser *et al.* extracts affordances of geometric primitives to support the planning of whole-body locomotion and manipulation actions [97]. Navigation through cluttered passages has not been considered in their scenario. Subsequently, Wang *et al.* [98] applied the ideas of Kaiser to the blocked path scenario as in our work. However, their approach does not integrate navigation cost into the

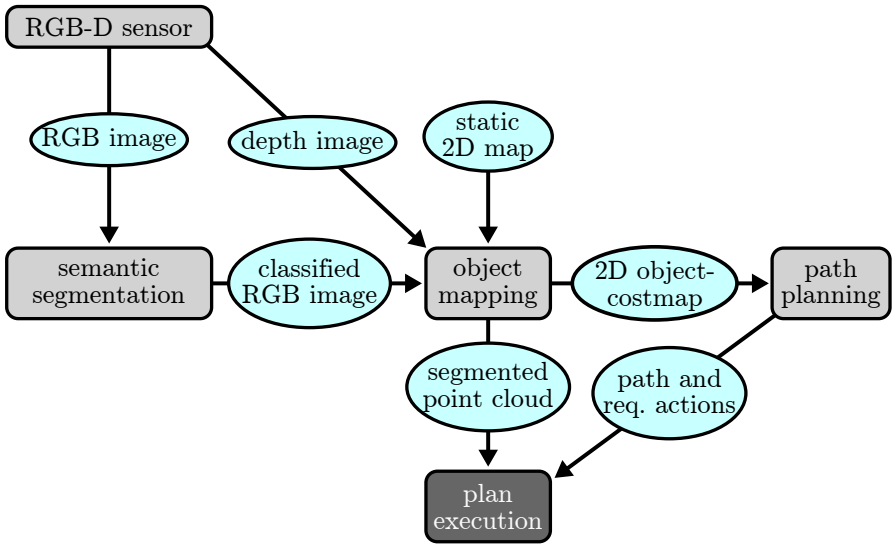


Figure 7.2: Overview of our framework. The different components are summarized in Sec. 7.3.

action planning and only considers replanning when the object can not be moved to free the path. Thus, the method provides no criteria for assessment of different solutions to plan a optimal path containing the best actions.

Although some of the above mentioned approaches provide a robust way to plan through environments containing cluttered regions, they neglect the semantics of the objects. Semantic information, however, can be effectively exploited for humanoid navigation. Current advances in computer vision using deep CNNs to extract semantics of the environment [99]–[101] have made it possible to infer the semantic classes of objects in cluttered scenes with high accuracy. Such approaches allow us to map each object class to a different action type that we can use for planning in an efficient way. Running our classifier in a fast manner is necessary to avoid adding a large computational overhead to our approach. Thus, we build on top of recent work focusing on *real-time* CNNs [9], [102]–[105]. Since training deep CNN models is a data intensive task, we present a way to alleviate this by generating a large-scale data set of our interest clutter classes with minimal effort in a semi-autonomous way by crawling images from the Internet.

7.3 SYSTEM OVERVIEW

Before we describe our approach in detail, we present an overview of the individual components, which are also illustrated in Fig. 7.2. Starting with the raw perceived RGB-D image data, we extract the RGB image for semantic segmentation, to mark the different pixels of objects in the image. The semantically segmented image is then

provided to the object mapping, which additionally uses 3D point cloud data from the input depth image aligned with the RGB data, and a 2D grid representation of static obstacles in the environment. This static 2D map is constructed using a standard mapping system [17] in the absence of further objects. The output of our object mapping is a 2D cost grid, encoding static obstacles as well as the newly detected and classified objects, on which the path planning takes place. The execution of the computed path, which comprises actions corresponding to the detected objects, is done by the plan execution. The plan execution additionally uses the segmented point cloud when necessary, while invoking the required object actions.

During navigation, our system continuously updates the representation of the environment with information about newly sensed obstacles that might be blocking the way of the robot and replans the robot's path toward the goal if necessary.

7.4 SEMANTIC SEGMENTATION

The first key step for our approach is the extraction of the semantics of objects in the robot surroundings. Then our approach aims at inferring possible robot actions from the objects in the environment in real-time. This requires a visual classifier that can recognize individual objects accurately from a dictionary of possible classes, while still running fast on a power- and payload-constrained machine such as a small humanoid robot. The state of the art in object detection and semantic segmentation using CNNs makes the accuracy of such algorithms acceptable for this approach to be feasible, but most CNN pipelines are computationally intensive and require large amounts of training data. In order to make the approach applicable on our robot, we rely on a lightweight architecture to achieve a good runtime vs. accuracy trade-off. To approach the amount of training data needed, we use pre-trained models from Bonnet's library [9], which already provides useful features in the convolutional layers and we create a large data set by mixing images recorded by ourselves and a huge amount of scavenged data from the Internet for refining the pre-trained models.

We opt in favor of a semantic segmentation pipeline, which maps each pixel of the robot's camera images into one of the eight classes: *"balls"*, *"books"*, *"boxes"*, *"cars"*, *"dolls"*, *"stuffed toys"*, *"toy blocks"*, and *"background"*, so that each class has at least one navigation action assigned to it. Note that our approach is not limited to these classes and could easily be extended.

7.4.1 CNN-Based Semantic Segmentation

Fig. 7.3 shows a diagram of the encoder-decoder CNN architecture used in our approach that is build using Bonnet [9]. The chosen ar-

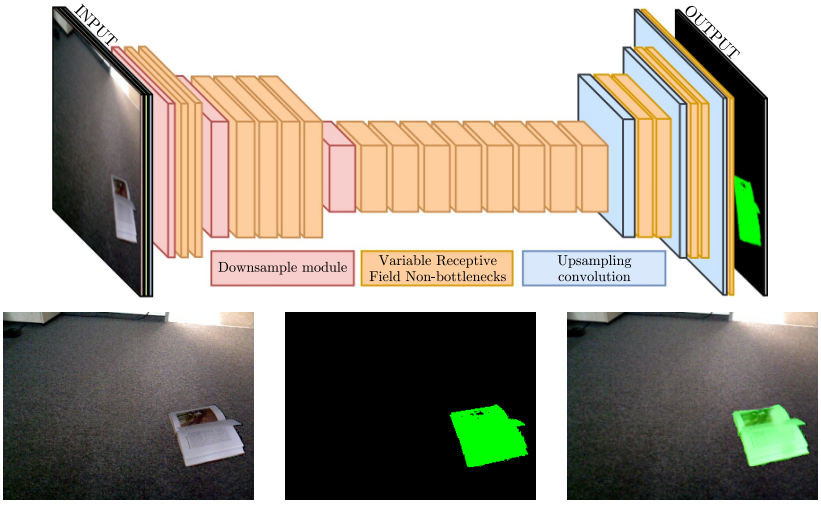


Figure 7.3: Top: encoder-decoder semantic segmentation CNN based on the non-bottleneck concept behind ERFNet[104] inferring an image from our data set. Bottom, left to right: original RGB image, prediction from CNN, and alpha blend for visual qualitative performance assessment. Best viewed in color.

chitecture is based on ERFNet [104], which proposes to change each computational bottleneck introduced in ResNet [100] and ENet [103] with a "separable non-bottleneck" of a variable receptive field. This module uses a set of separable filters of sizes $[1 \times 3]$ and $[3 \times 1]$ and different dilation rates d, which makes each layer effectively wider without increasing computational cost, allowing the network to be more descriptive without affecting runtime. The choice of using different dilation rates allows the network to have a bigger equivalent receptive field in the image space, capturing long-range dependencies, which is key for big objects. By using a model with these properties, and adjusting the number and width of the layers to fit our data, we can achieve a model that is descriptive enough to provide us with accurate semantic labels while running in real time. In order to achieve even further efficiency in inference to allow the approach to run in a resource constrained platform such as a small humanoid, we exploit TensorRT [106] acceleration of our neural network. For this reason, the used ERFNet backbone needed to be modified to fit all supported operators. At the time of writing, there is no efficient support of dilated convolutions within TensorRT, so in order to efficiently infer full semantic segmentation with our proposed architecture, each $[1 \times 3]$ and $[3 \times 1]$ dilated convolution was replaced by a dense convolution of $[1 \times [3 + 2 \times \ d]]$ and $[[3 + 2 \times \ d] \times 1]$. We observed no drop in performance from this modification, and even though more operations are needed for this, the acceleration obtained by the inference optimization provided in TensorRT was superior.

We start from a pre-trained encoder using this modified architecture, which was trained with the COCO data set [107] and therefore provides rich features even before the semantic segmentation task-specific training. We attach a small decoder that is trained from scratch using random weights and fine-tune the model training end-to-end using back-propagation and a pixel-wise cross-entropy loss of the form

$$L_{\text{semantic}} = -\sum_{c=1}^{C} w_c \, y_c \, \log\left(\hat{y}_c\right) \tag{7.1}$$

$$w_c = \frac{1}{\log\left(f_c + \epsilon\right)} \quad \text{with} \quad f_c = \frac{1}{P}\sum_{p=1}^{P} \begin{cases} 1 & \text{if } p = c \\ 0 & \text{if } p \neq c \end{cases}, \tag{7.2}$$

where w_c penalizes class c according to the inverse of its frequency in the ground truth, bounded by a parameter ϵ which is selected by cross-validation and is set to 1.02 in all our experiments.

The training was performed with a data set of 5,000 images containing roughly 20,000 toy instances with their respective dense masks, which we explain in the following section. The retraining was performed by minimizing Eq. (7.1) through stochastic gradient descent using the Adam optimizer, with a batch size of 36, a batch normalization momentum of 0.99, and an initial learning rate of 10^{-4}. The learning rate is then halved every 50 epochs, training for 200 epochs over the whole data. We used a channel dropout rate of 10% in the layer before the linear classifier, and a weight decay of 10^{-5} for regularization during training, and use the model with the best validation error measured after each epoch, a practice commonly called early-stopping.

7.4.2 Data Collection

As previously stated, training deep CNNs requires a large amount of labeled training data to obtain the accuracy required to run other approaches on top of the obtained semantics. This effect is particularly magnified when using a semantic segmentation pipeline, because a holistic knowledge of what each image contains is not enough, and labels are required at a pixel level, increasing the effort required to label each image considerably. Even though using pre-trained model helps to reduce the amount of required labeled data, a particular case which makes the training more data-hungry is having low inter-class distances, like in our case (see Fig. 7.4). To resolve this problem, we collected a data set with 1,000 objects focusing on the hard examples of inter-class distance, i.e., focusing mostly on labeling objects whose appearance is similar, but which have a different semantic label. This is done to train CNN features that are sensitive enough to allow the

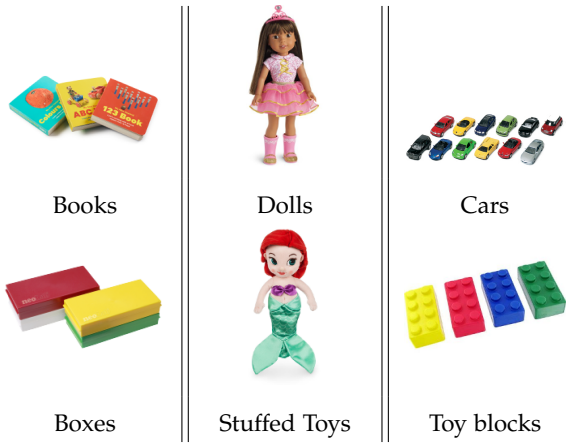


Figure 7.4: Examples of pairs of classes with low inter-class distance, challenging for the CNN, but important for our approach due to different associated object interactions.

Figure 7.5: Examples of backgrounds used to generate the synthetic training images. Backgrounds were also crawled from the internet, with queries designed to match viewpoints of places where the robot is likely to operate, such as home and school environments.

classifier to fit the classification hyper plane effectively. To generate a data set for semantic segmentation, we need pixel-wise masks for each individual object. Since such a labeling is expensive, we collected the data with an RGB-D camera and segmented the objects in the depth channel to obtain the ground-truth mask before feeding them to the CNN as a 3-channel RGB-only image.

To scale up the data set and make it an order of magnitude bigger, we wrote a script to automatically download images from the Internet with properly formatted queries returning images fulfilling the following conditions: (i) the image contains only one of the desired classes in the dictionary, and (ii) the image contains either an alpha channel making the background transparent or has a blank background. Under these restrictions, the script returns roughly 25,000 images using Google Images [108]. We further reduce it to roughly 20,000 images after six hours of supervised cleaning by a human. The supervision consists of navigating quickly through the crawled images and identifying objects that either do not belong to the specific class we are interested in, or whose alpha channel does not fit the object boundary. For our CNN to be usable in realistic environments, the last step in this data set generation method is to generate 5,000 clutter images from

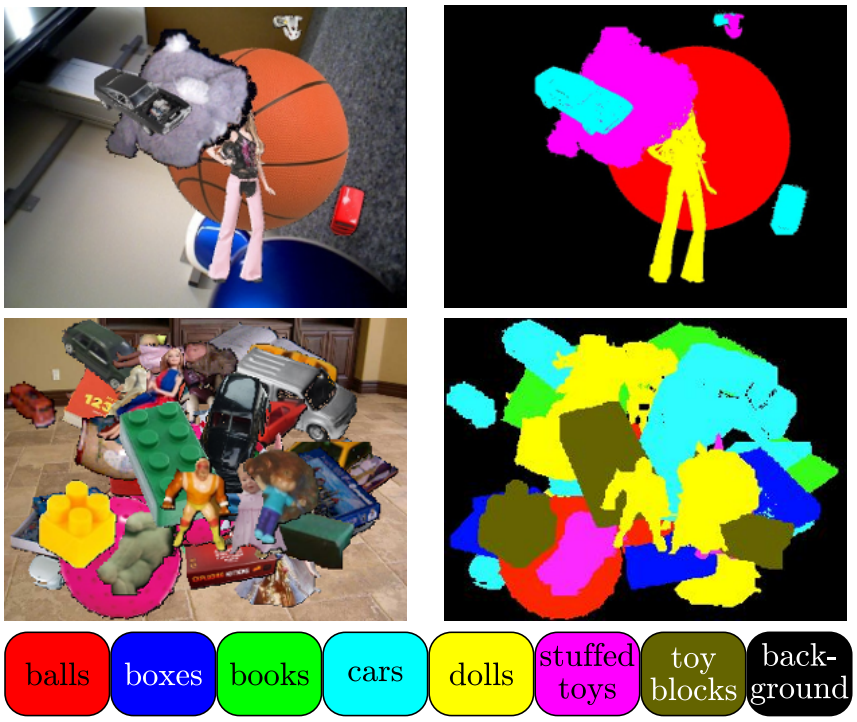


Figure 7.6: Examples of generated clutter images with added background. Left: RGB image, right: ground truth. Best viewed in color.

our raw data, containing one of 300 different backgrounds (Fig. 7.5) and any number of random objects from the database from 0 to 20 objects per image. This can be considered as “synthetic" data, but it is a step closer to the real world, because the images are of real-world objects (Fig. 7.6).

7.5 PATH PLANNING UTILIZING OBSTACLE INFORMATION

In this section, we describe our method for exploiting the semantic information about detected and segmented objects during path and action planning.

7.5.1 Actions for Object Classes

We assume that an expert user assigns for the individual object classes possible actions defining how the robot can overcome such obstacles when necessary. The possible actions for the object types inherently depend on the specific robot that is being deployed. As an example, Tab. 7.1 shows the actions associated with the object classes for a Nao robot. Note that for some object classes several actions are possible, in which case our system selects the least cost action to deal with the

Table 7.1: Example assignment of possible actions to object classes. The actions are chosen by an expert user according to the robotic hardware, in this case a Nao robot.

Object class	Possible actions
balls	push, step over, pick up
cars, toy blocks	step over, pick up
stuffed toys, dolls	pick up
boxes, books	step onto

objects on its path. The robot furthermore analyzes the point cloud of the object to decide if an action may not be executable, for example because the object is to high to be stepped over. Those aspects are described in more detail in Sec. 7.5.3.

For our scenario, we currently use the following actions:

standard walking in case of no objects: If the path does not contain any objects, the robot's walking controller follows a 2D path.

push: If an object needs to be pushed away, the robot follows the 2D path to the last free grid cell on the path and starts the pushing action. Thus, the robot senses the object locally using the segmented point cloud, moves to a target position relative to the object, and pushes the object out of its way in forward direction. The push action is illustrated in Fig. 7.7).

step over and step onto: If the robot needs to traverse an area with objects that need to be stepped over or onto, the robot applies footstep planning in the corresponding region on a height map computed from the point cloud using our previous work [109]. Fig. 7.8 shows the maximum height of a box the Nao can overcome.

pick up: If the robot needs to pick up an object and move it out of the way, the robot again follows the 2D path to the last free cell on the path in front of the object and then identifies the object in the segmented point cloud, finds the target position relative to the object, grabs it, rotates 180°, and puts the object onto the ground, and rotates back 180° . The pickup action is depicted in Fig. 7.9.

7.5.2 Action Costs

We define the costs of an action according to the time the robot needs to perform the action. To determine the execution time, we designed

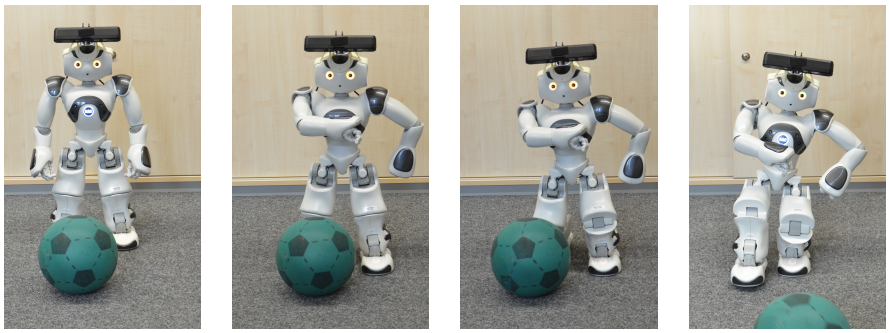


Figure 7.7: Pushing: In our implementation, the push action includes tracking the ball locally, positioning relative to the object, and kicking it in the forward direction.

Figure 7.8: Stepping onto: The step onto action includes stepping onto the object and also stepping down according to a footstep plan.

an experiment in which the Nao robot had to reach a goal one meter in front of it. The experimental setup of the actions is illustrated in Fig. 7.10. We measured the time it took the robot to reach the goal by just walking to the target location and by additionally pushing an object out of the way, stepping onto or over objects, and picking up an object. For each case, we performed ten experiments and selected the average execution time as the cost used for the planner. The averaged times of the experiments are provided in Tab. 7.2. Since the walking is subject to errors induced by the action itself, e.g., shaky walking start

Table 7.2: Execution times of actions.

Action type	**Mean execution time [s]**
walk	12
push	25
pick up	55
step over	40
step onto	61

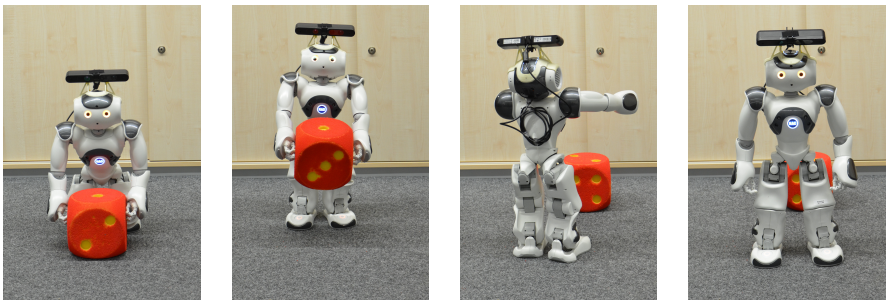
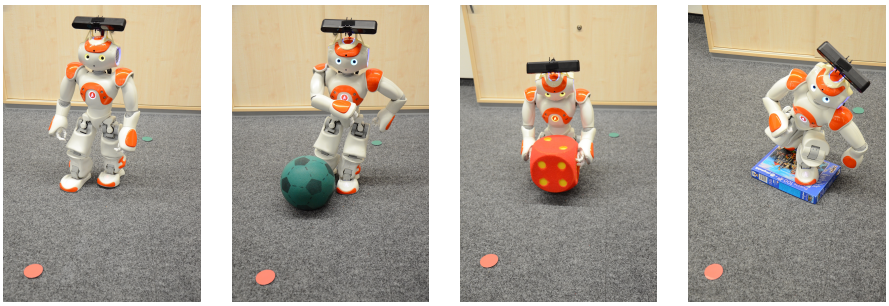


Figure 7.9: Picking up: The pick up action includes the tracking of the object locally, positioning relative to the object, executing the grabbing, performing a 180° turn, dropping the object, and rotating back 180°.



walk push pick up step onto

Figure 7.10: Experimental setup to determine the costs of actions according to the completion time. See Sec. 7.5.2 for a detailed explanation.

after the push action or the difference of the orientation to the goal after the pick up action, it was important to let the Nao robot walk before and after the corresponding action execution to measure the influence of small uncertainties on the time performance.

Other factors, e.g., energy consumption or risk of failure, can be considered as well for the cost computation and added to the determined cost.

7.5.3 Object Mapping

In this subsection, we describe how to map objects detected in the environment using RGB-D data onto the cost grid used for planning. We use the semantic segmentation described in Sec. 7.4 and combine it with the depth information of the RGB-D image to get a segmented point cloud of the corresponding objects (see Fig. 7.11). Afterwards, we project the segmented point cloud onto a 2D grid map representation of the environment containing inflated static obstacles as illustrated in Fig. 7.12a. Inflating all objects with the robot radius is a general concept to prevent the robot from colliding with obstacles in case of

Figure 7.11: Data processing for the RGB-D data. a) The original RGB image containing a doll and toy blocks between two walls. b) Semantic segmentation results using the Bonnet framework. c) Segmented point cloud of the corresponding objects, using the depth image to get the spatial information of each marked pixel.

slight localization errors (Sec. 2.2). We maintain an object database that contains the information about objects in form of object ID, object class, and the set of corresponding 2D grid cells. Note that obstacles that cannot be classified by the CNN and, thus, are considered as background are mapped as static and are not stored in the database of objects that can be manipulated.

Our approach uses a 2D cost grid map for path planning that encodes, in addition to the static obstacles, the costs of the actions corresponding to the observed objects, which are estimated as described in the previous subsection (Sec. 7.5.2).

To choose the action for an object, we consider the segmented point cloud to exclude some possible actions listed in Tab. 7.1 that are predicted to be not executable by the robot depending on the size of the object. We then assign the cheapest action of the remaining actions to each object and, thus, to the cost map. For example, a small ball can either be pushed away (cheapest action), stepped over, or the ball can be picked up in order to free the path, while for a big ball, e.g., a basket ball, neither action would be expected to be executable so that the object would be marked as static in the map. The applied heuristics to suggest a viable action for the Nao humanoid are listed in Tab. 7.3. In general more advance operations for analyzing the geometry of the segmented object could be performed. However, this leads to a trade-off between the accuracy of the suggested action and the time performance of the planning framework.

In Fig. 7.12b, the cost value is represented by the gray level of the object border cells. Hereby, the costs of the overlapping border areas between the inflated segmented objects are the sum of the corresponding action costs of both/multiple objects, which means that in these regions several object actions will be necessary. In this example, the path leads across the toy block. From Tab. 7.1, our planner can chose between the step over and the pick up action and decides to step over the block since this is the cheaper action.

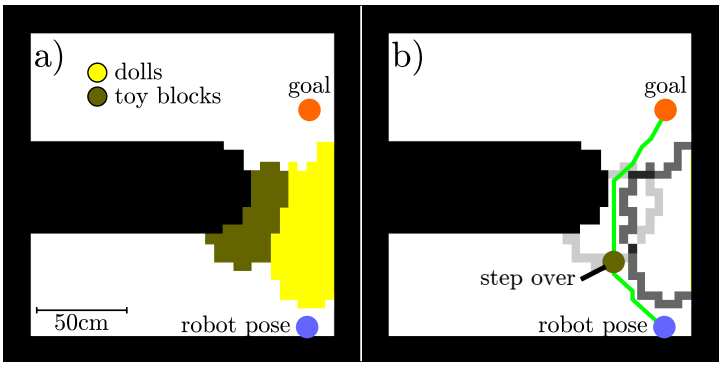


Figure 7.12: a) Visualization of the projection of the objects onto a 2D grid. Inflated static obstacles are represented as black cells and cells with detected objects from the segmented point cloud (see Fig. 7.11c) are color coded. Yellow cells correspond to the doll, brown cells to the toy blocks. b) Resulting 2D cost map for planning with the costs of the object actions encoded in the border cells of the objects. The border cells of the toy blocks (light gray), which can be stepped over, have lower cost than the cells corresponding to the doll (dark gray), which needs to be moved away so that the computed path contains the action to step over the blocks. The overlapping border cells of the inflated objects are the sum of their action costs.

Table 7.3: Heuristics for the viable actions of the Nao robot.

Action type	**Geometric object features**
push	maximum height < 20 *cm*
pick up	longitudinal axis < 30 *cm*
step onto	maximum height < 7 *cm*
step over	maximum height < 6 *cm* and transverse axis < 5 *cm*

Note that our system pauses the mapping while the robot executes a push, step, or pick up action to free as much resources as possible to perform the action. Thus, no object tracking takes place during that time.

7.5.4 Path Planning

Since the cost map contains the information about all obstacles, i.e, static and not static, and encodes the potential object actions and their associated costs, we can efficiently use A* search [22] with the Euclidean distance as the heuristic function to find a path for the robot on the cost grid. If the path leads through any object area, the corresponding class and, thus, the possible actions can be derived from the object database. In the example shown in Fig. 7.11 and Fig. 7.12,

our approach computes the green path with the action to step over the toy blocks.

7.5.5 Updating the Action

Note that the path planner generally provides only suggestions regarding the action types. Should the execution module not find a solution for a proposed action, e.g., due to object-environment configuration or classification errors, the action would be changed for the corresponding object such that our planner would seek a different solution on the updated cost grid. This can also happen in case the footstep planner does not find a sequence of valid footholds for a region containing objects to be stepped over or onto, see Sec. 7.6.3 for an example.

7.6 EXPERIMENTAL EVALUATION

In this section, we present experiments to evaluate the performance of the CNN-based classification framework for the navigation task at hand (Sec. 7.6.1) and to demonstrate the capabilities of our system with respect to efficient planning and navigation in various real-world experiments with a Nao humanoid (Sec. 7.6.2). Additionally, we performed experiments with the REEM-C [110] humanoid (Sec. 7.6.3) in simulation to show a generalization of our approach to other robot platforms that are able to perform more advanced actions. For step over heuristic of the REEM-C, we set the maximum height and transverse axis to 15 cm and 10 cm, respectively. For the pick up action, we designed a one-handed grab procedure and determined 10 cm as maximum value for both minor axes of the object. Throughout this section, the illustrated grid maps are similar to Fig. 7.12a, i.e., they show the inflated object classes (rather than the actual costs) for better visualization. The resolution of the grid maps was set to 5 cm for all experiments. The inflation radius was chosen as half the shoulder width of the considered humanoid (15 cm for the Nao and 25 cm for the REEM-C).

7.6.1 Classification Results

The first set of experiments is designed to show that our semantic segmentation approach is applicable on mobile robot platforms in terms of accuracy, runtime, and computational resources. As detailed in Sec. 7.4.2, we train a semantic segmentation CNN with 5,000 images, generated from a database of 20,000 different object instances, and over 300 backgrounds. The network is then evaluated on a test set containing 1,000 real-world images collected in our lab and pixel-wise annotated. Semantic segmentation approaches which assign a label to

Table 7.4: Classification metrics on validation set for different input image resolutions.

Resolution	mAP	Per-class AP							mIoU
		Ball	Books	Boxes	Cars	Dolls	Stuffed toys	Toy blocks	
320×240	0.792	0.908	0.747	0.733	0.768	0.801	0.828	0.759	0.585
640×480	0.875	0.946	0.878	0.876	0.847	0.874	0.874	0.831	0.715

each pixel in an image are typically evaluated using the mean Jaccard index, also called mean intersection over union (mIoU), defined as

$$mIoU = \frac{1}{C} \sum_{i=1}^{C} \frac{tp_i}{tp_i + fp_i + fn_i}, \tag{7.3}$$

where C is the number of classes, and tp, fp, and fn are the pixel-wise number of true positives, false positives, and false negatives per-class, respectively. For approaches that in the end work on objects, a commonly used measure is the mean average precision (mAP):

$$mAP = \frac{1}{C} \sum_{i=1}^{C} \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} p_i(r), \tag{7.4}$$

where r corresponds to a value of recall in the precision-recall curve for each class, and $p_i(r)$ is the value of precision corresponding to recall r for class i. Predicted instances are defined as a positive detection when they have more than 50% IoU overlap with the ground truth mask.

In our experiments, we found that the quality of the classification depends mostly on the size of the input images, and therefore report in Tab. 7.4 the results for two resolutions of the used sensor (ASUS Xtion PRO). For a resolution of 320×240, we achieve a mAP over all classes of 0.79 and for 640×480 the mAP increases to 0.88. These results are encouraging since they show that starting from pre-trained weights, a network can be trained solely on images crawled from the Internet and few hours of human supervision to clean the data set from improper objects present in the query results and wrong masks in the alpha channel. The limitations of off-line batch training in terms of labeled data collection and pre-definition of the classes are hard to circumvent and currently an open research area in computer vision.

As with the accuracy of the model, the runtime of the CNN is also highly dependent on the input resolution. In Tab. 7.5, we show the runtime of the model in different hardware and using different resolutions. The results show that the approach is usable also in resource-constrained hardware, such as the NVIDIA Jetson TX2, where we achieve a framerate of 4 Hz-11 Hz depending on the image resolution.

Table 7.5: Runtime for segmentation at different image resolutions.

Resolution	Runtime	
	GTX1080Ti	Jetson TX2
320 × 240	10 ms (100 FPS)	89 ms (11 FPS)
640 × 480	33 ms (30 FPS)	245 ms (4 FPS)

7.6.2 Real-World Experiments with a Nao Humanoid

The second set of experiments is designed to show the behavior of our planning framework in different real-world navigation scenarios. The experiments demonstrate the advantages of our method in comparison to pure footstep or whole-body-motion planning systems, as these approaches are not capable of finding a solution to the goal when manipulation actions are required.

We equipped the Nao robot with a ASUS Xtion PRO to show the full capability of our navigation framework. For 6D localization we apply Monte Carlo localization as developed by Hornung *et al.* [111] and extended by Maier *et al.* [112] for depth camera data.

7.6.2.1 *Path Planning with Manipulation Actions*

In the situation depicted in Fig. 7.13a, the robot detected a box and a ball that obstructed the way. To reach the goal location, the robot could either push the ball aside or step onto the box before continuing walking. Based on the computed cost map our planner found a path across the ball. The robot followed the path to the vicinity of the ball and then performed a push action (see Fig. 7.13b). After pushing the ball aside, the path to the goal was free and the robot continued walking (see Fig. 7.13c.)

7.6.2.2 *Replanning the Path During Execution*

The next two experiments are designed to show the replanning capabilities of our framework when the current 2D path does not appear to be optimal anymore according to an updated cost map.

In the first experiment shown in Fig. 7.14, our robot first detected only a single row of blocks and the stuffed toy blocking the way to the goal. Accordingly, stepping over the two detected blocks was the cheaper solution compared to picking up the stuffed toy and putting it aside. While following the path and updating the cost map, the robot subsequently encountered more blocks, so that cheapest path was now leading through the stuffed toy and the plan was changed to include the action to pick up the stuffed toy. Thus, the robot followed the path to the last free cell in front of the object and then executed

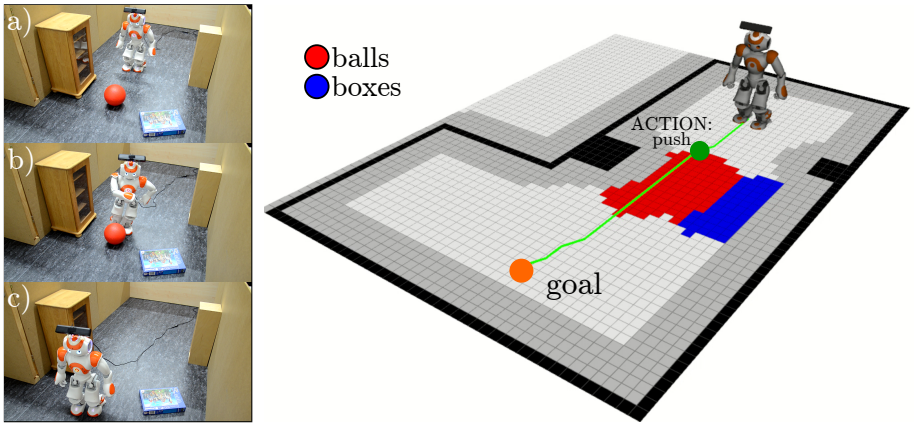


Figure 7.13: Path planning with manipulation actions: Experiment with a Nao robot (left) and segmented objects with inflation radius projected onto the grid map (right). *Left*: a) The robot detects the box and the ball on its way to the goal and decides to push the ball in order to clear the path, since this is the cheapest path on the corresponding cost map. b) After the robot has followed the path close to the object, it performs the pushing action. c) The robot continues walking along the path, which does not contain any further objects.

the pick up action, before it followed the remaining path to reach the goal. A video of this experiment is available online[1].

The next experiment in Fig. 7.15 shows a scenario with two different routes to the goal. Since the balls were too large for the Nao to step over them (Fig. 7.15a), the robot could either push the balls out of the way, pick them up, or take a detour around the large central obstacle. In this case, our planner found a path that included the detour, since interacting with multiple objects had higher associated costs. In Fig. 7.15b another obstructing object, in this case a stuffed toy, was detected, where our planner decided based on the updated cost map to perform a pick up action to clear the path.

7.6.3 Replanning Actions During Execution

The final experiment demonstrates the capabilities of our planning system to replan the action for an object during execution. In this scenario (Fig. 7.16), the robot had to cross a narrow passage with several blocks. The initial plan contained step over actions over two blocks in the middle. However, after approaching the blocks and initiating footstep planning for the step over action, our footstep planner [109] could not find a sequence of valid footholds in order to reach the subgoal and failed. According to the failed action, the cost map was updated with the new action pick up for the first block on the

1 https://youtu.be/WO94iXT3V1I

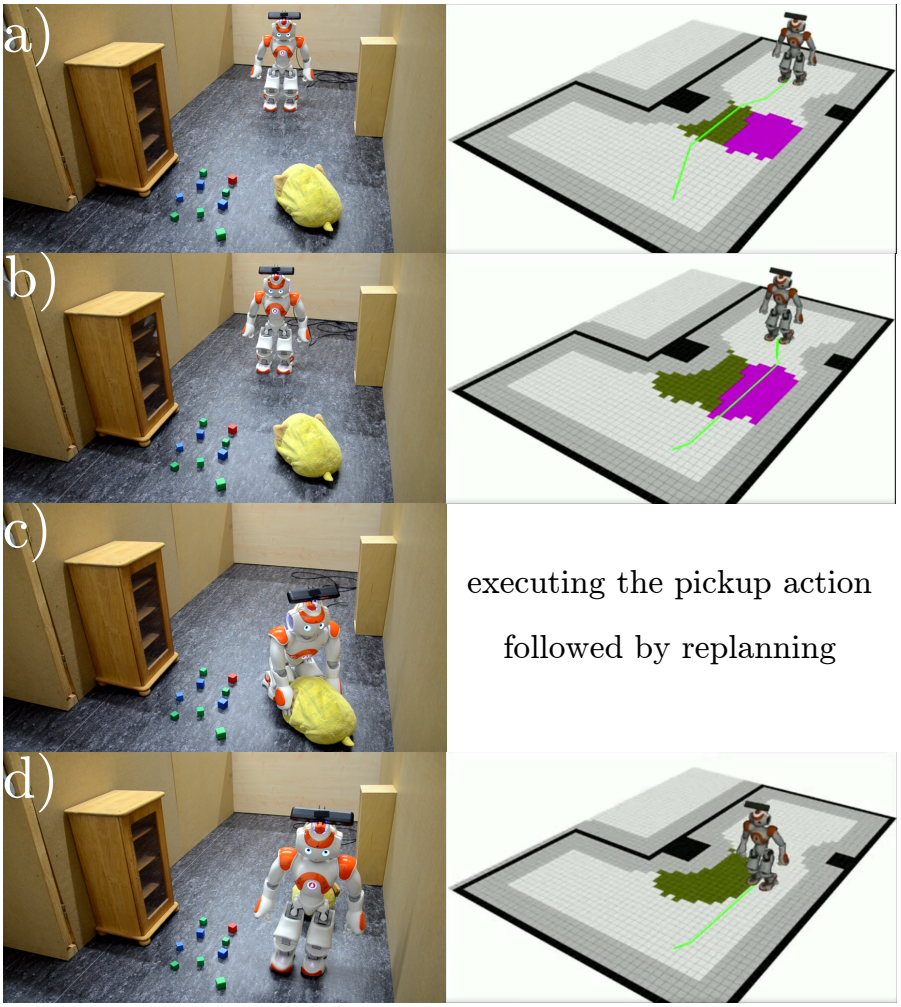


Figure 7.14: Replanning the path on an updated cost grid during execution. a) In this case, the robot initially observes only two toy blocks at the foremost row and the stuffed animal in its field of view. The cheapest solution computed by our planner on the corresponding cost grid leads through the toy blocks, which require footstep planning. b) While moving forward, the robot detects further toy blocks and computes a new path on the updated cost grid. Now the path leads through the stuffed animal, suggesting a pick up action. This solution is cheaper since it contains one manipulation action rather than several step over actions. c) The robot executes the pickup action and puts the object aside. d) Afterwards our planner updates the cost map and replans the path. The robot can now simply follow the path to its goal.

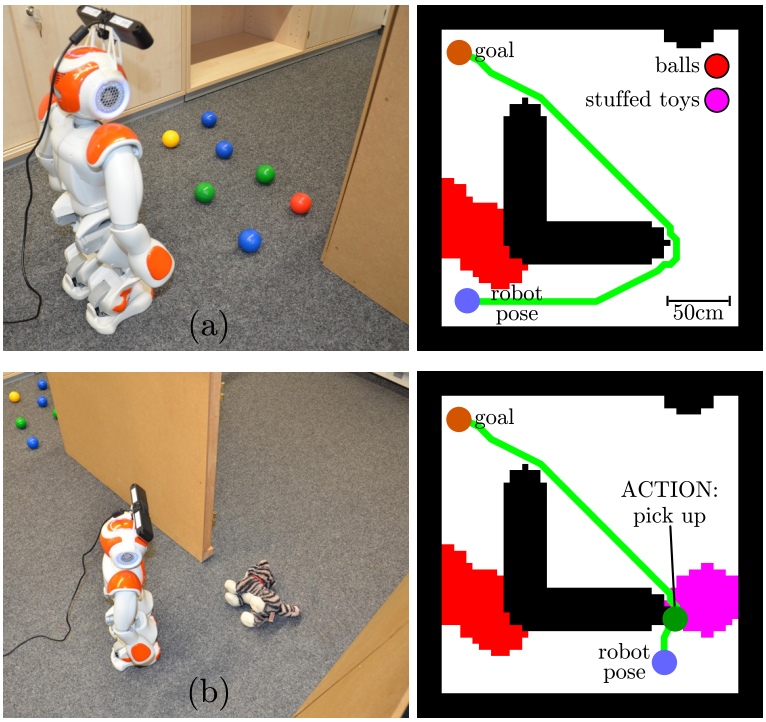


Figure 7.15: Replanning the path during execution: Nao humanoid and segmented objects with inflation radius projected onto the grid map. a) The robot perceives seven objects classified as balls in its way and decides to take the detour around the L-shaped static obstacle since this appears to be the cheapest solution according to the cost map. b) On its way to the goal, the robot detects a further object (in this case a stuffed toy). Our framework now decides to pick up the object to reach the goal location.

path. Replanning on the updated map lead to a solution containing the same objects but with another action for the first object. The robot had to pickup the first block on the path and step over the second one. After performing the pick up action and replacing the block, the path planner found a valid sequence of steps to cross the region.

7.6.4 Summary of the Experiments

In sum, our experimental evaluation shows that semantic information about objects can help humanoid navigation to efficiently plan and effectively execute navigation actions that include handling objects. We demonstrate with our experiments that state of the art semantic segmentation CNNs such as Bonnet [9] are well-suited to solve the associated perception problem of classifying the individual objects in the vicinity of the robot. Based on this information, the robot can combine 2D planning, footstep planning, and an effective object-dependent action selection approach. As a result of that, complex to compute complete whole-body plans or higher-level action plans provided by a symbolic action planner can be avoided. With our

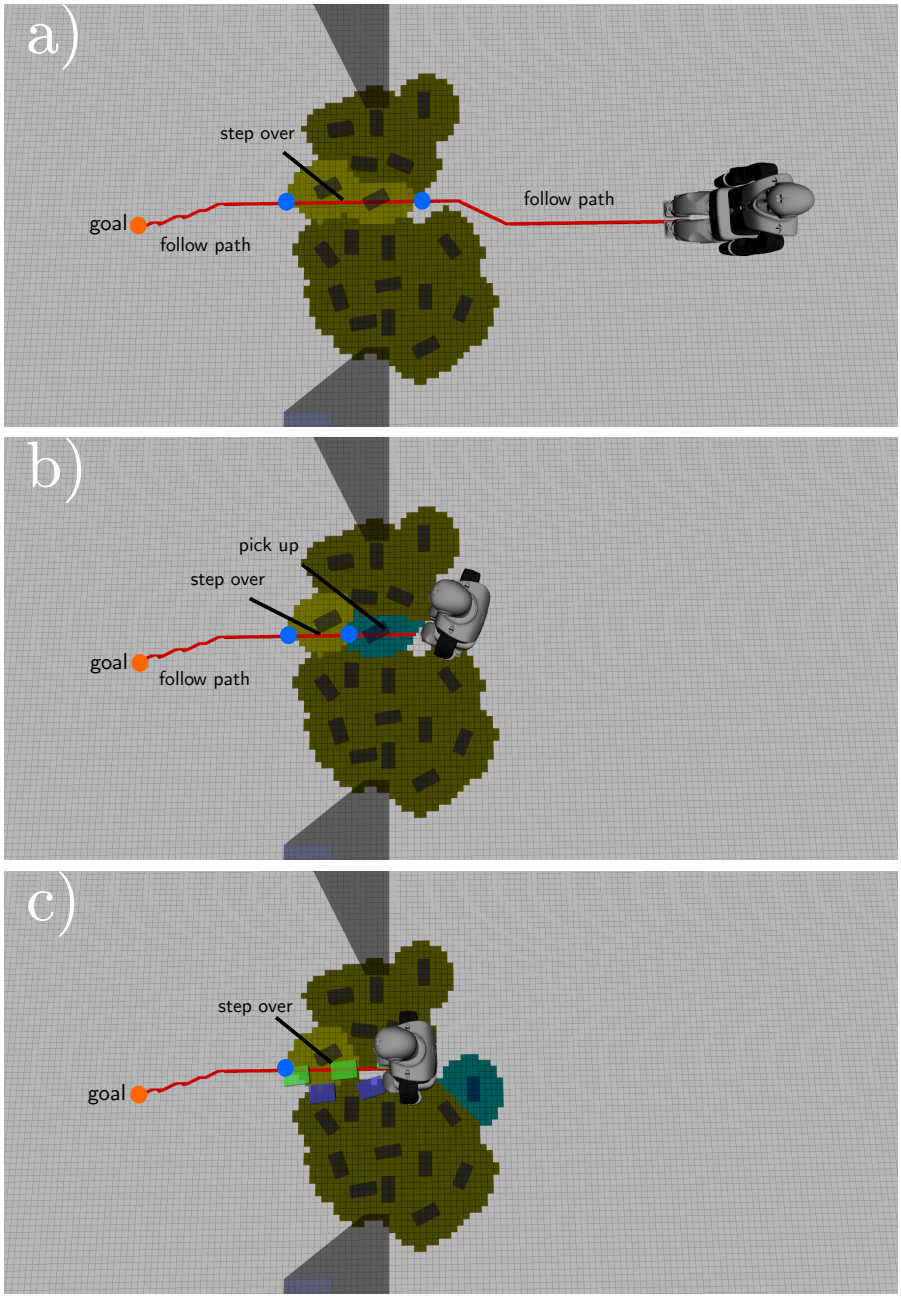


Figure 7.16: Replanning actions during execution: Experiment with a REEM-C[110] robot in simulation. a) Initially, our planner computes a path that suggests to perform the *step over* actions over the two blocks to be computed with a footstep planning algorithm (between the two blue circles). b) However, due to the configuration of the blocks, the footstep planner could not find a valid footstep plan. Therefore, our planner selects an alternative action for the first object on the 2D path. Based on the action table, our framework decides to *pick up* the first object and recomputes the plan. c) After the robot has picked up and moved the object out of the way, the footstep planner finds a path over the remaining region.

framework, the robot can perform planning of complex navigation tasks online, in simulation as well as in the real world. Our experiments demonstrate this for the computation of new plans as well as for online replanning. The combined maximum runtime of the mapping and planning step is 0.2 s for a reasonable number of objects (two to five) on an *Intel Core i7-4710MQ*, without the object classification and segmentation, which runs in another thread with 5 Hz.

7.6.5 Implementation Details

For our experimental setup, we use the Robot Operating System [24] (ROS) as a communication backbone between the different framework components. For the experiments with the Nao robot, we used the ROS packages [113] that contain the drivers for communication with the NaoQI API [114], the robot model, and the walking module. For localization, we also used ROS packages for Monte Carlo localization using Octomap as environment representation [115]–[117].

The experiments with the REEM-C were simulated in Gazebo [39] and visualized with RViz [118]. We integrated our footstep execution with the PAL Robotics repositories [119], [120] available on GitHub that provide the robot model and the communication interface.

7.7 CLASSIFICATION ERRORS

Wrong object classification of the CNN or non-executable actions caused by wrong heuristics for action execution can of course happen. In these cases, the map and the path will be updated when new information becomes available. As a result, the robot's path might not be the optimal one. We experienced the following classification errors in our experiments:

- A single object was perceived as two objects of different classes, especially when detected at the border of the field of view of the sensor, because the object was only partially perceived and/or pixels further away from the robot cover a bigger surface of the environment and thus provide less information for the CNN.
- False positive object detection of the background.

In most cases, the error was compensated by the next sensor measurement and the map as well as the path were updated accordingly.

7.8 CONCLUSION

In this work, we presented a novel framework for humanoid motion planning that exploits the knowledge about obstacle classes during the

planning process for fast planning and efficient humanoid navigation. As one central contribution we propose to train a convolutional neural network to distinguish between different object classes and use this information to construct a cost grid during navigation. The cost grid represents the static obstacles in the environment as well as the costs of actions that need to be carried out by the robot to cross cluttered regions. These cluttered regions typically contain obstacles that can be overcome by the robot, i.e., by actions such as stepping over or onto, pushing, or picking up. The robot then uses the cost grid for efficiently planning its path to the goal location and the 2D path implicitly contains all necessary actions to be executed by the robot. Should replanning be necessary during the execution due to failed actions, the cost grid is updated with the costs of alternative actions and the path is replanned.

As we showed in various experiments, the trained neural network is able to robustly distinguish between the different obstacle classes and thus provides relevant information to the robot's planner. We demonstrated that a humanoid robot can exploit the knowledge about classified obstacles during navigation and efficiently find solution paths that contain appropriate actions to deal with the obstructing objects. Furthermore, we showed that in case the current solution cannot successfully be executed, the robot invokes replanning and our system provides an alternative path.

8 CONCLUSION

8.1 SUMMARY

In this thesis, we presented new contributions to the field of robot navigation. Especially, we investigated challenging real-world scenarios with little space to maneuver, crowds of people, and impeding objects. Our developed approaches focus on mobile robot systems with constrained computation power and limited field of view. The proposed contributions enhance the robot navigation by expanding the environment model, using machine learning to design powerful function approximators, and integrating robot actions into the planning process. As a result the robot navigation improved by arriving faster at the goal and having less collisions.

First, we introduced a method to incorporate the clutter object distribution into the navigation cost function. The new costs predict the occurrence of objects in not detected areas based on the objects in the vicinity of the robot. By integrating the new navigation costs into the grid mapping model, the robot is able to act foresightedly and avoid areas with high densities of objects. As a consequence, the robot is able to drive faster and reaches the goal quicker compared to a standard path planner that only considers detected objects.

The second contribution provides a method to estimate the arrival time of the robot at the goal from a 2D grid path. The estimation of the completion time is vital for real-world scenarios, where time-efficient planning is needed. Those application range from floor-cleaning and household robots to autonomous food delivery vehicles. Moreover, the implemented method allows to choose a path, when several path options are available, since the completion time is a more relevant criterion than, e.g., the length of the path. The completion time is not known in the planning phase, due to the inaccuracies of the motion execution, e.g., slippage of the wheels, noise in the sensor measurements, and errors in the localization. Therefore, we trained a non-linear regression estimator, that shows reliable prediction of the completion time.

In the third contribution, we coped with navigation through pedestrian crowds. The approach improves the robot motion commands determined by the popular social force model (SFM). The SFM is used to simulate crowd dynamics and is very well suited to achieve human-aware navigation behavior of the robot. To improve the steering in terms of collision and the completion time, we used supervised learning to train a network that adjust the original robots commands.

We generated the training data by designing a predictive controller that simulates the crowd and robot movement into the future and evaluates the given situation for all given incremental acceleration adjustments. The best command is then used as a learning example to train a network. We show in our experiments that the learned network controller as well as the predictive controller significantly reduce the collision rate in comparison to the basic SFM controller while maintaining a comparable velocity.

The fourth contribution also deploys machine learning techniques to realize goal-oriented and collision free navigation. This time, we apply reinforcement learning to obtain a self-learned navigation policy. Self-learning approaches optimize the navigation behavior through a trial-and-error procedure, thus no parameter tuning of the navigation system is required. We trained a network that provides the robot's translational and angular velocity commands for the next time step in a simple environment. To evaluate the learned navigation policy, we tested the robots in more complex environments, unknown during the training phase. Furthermore, we compared the performance of our trained policy to the popular dynamic window approach (DWA) with respect to completion time of navigation tasks. The results show, that our learned policy completed the navigation tasks faster on average than the DWA of ROS. The policy gains the time advantage by keeping a closer distance to obstacles and maintaining appropriate velocities.

Lastly, we integrated path planning, object recognition, footstep planning and manipulation actions into a single navigation scheme. The central idea of the approach is to classify the perceived objects by a CNN and subsequently assign actions to every object class. To efficiently overcome the impeding object, we determine the navigation cost for every action and choose the cheapest applicable action according to the object class. By integrating the action costs into the grid map, the robot is able to plan a path, that encodes the necessary actions to perform to reach the goal. Our experiments show the network can successfully classify the detected objects and provide the robot with the necessary information. As a result, the robot can exploit the knowledge about classified obstacles during navigation and efficiently find solution paths that contain appropriate actions to deal with the obstructing objects. Similar state of the art systems, in contrast, fail to reach the goal in comparable scenarios or require significantly more time.

In conclusion, the developed approaches in this thesis could solve the following research questions:

- How can we improve the navigation of the robot in cluttered environments, by considering the clutter distribution in the vicinity of the robot?
- How can we predict the robot's completion time of a navigation task based on path features?

- How can we improve navigation with the social force model through pedestrian crowds, by using prediction and machine learning techniques?
- How can we optimize the navigation behavior of the robot near obstacles with reinforcement learning?
- How can a robot efficiently reach a navigation goal, that is blocked by obstacles?

The presented ideas in this thesis contribute to the progress in robotic development for a more autonomous robot navigation, that can be operated by a non-expert user and can handle complex scenarios in cluttered environments.

8.2 OUTLOOK

At the end of this work, we want to outline several extensions and open research questions for future investigation. As mentioned previously, the time estimation method from Chapter 4 can be integrated with the approach of Chapter 3 and thus improve the navigation performance by providing valid information about the completion time. The idea is to determine two path options by using different navigation cost functions, e.g., the standard cost function and our extended clutter navigation cost function. By estimating the time of both paths, the robot could decide which option is the most promising and follow the best guess. We think that it could greatly improve the completion time and avoid cases where our approach was slower than the standard approach (Fig. 3.8).

With regards to Chapter 7, the presented navigation scheme uses a pipeline, where a network provides information about object classes in the environment of the robot. To receive the information of the required actions, we perform a hand-designed mapping from object class to possible robot actions (Tab. 7.1). Our approach would greatly benefit if we determined the actions of objects directly by using a neural network that can predict affordances [121]. In this context, we understand affordances as the perceived actions of an object in a specific environmental state. For example, a robot could push (action) a toy car (object), but only if the wheels of the car are placed on the ground and the pushing direction is not blocked by another object. The detection of affordances could reduce the amount of engineering required to identify the actions of objects and also reduce false action suggestion of our planner that are only noticed during execution. In the last years, progress has been made that shows the potential of such approaches [122]–[125]. However, the mentioned publications cover only specific scenarios and it is not trivial to apply the presented solutions to the general case. In our trials we discovered insufficient

results regarding the estimation of the affordances, because the network does not only have to reason about the object, but also about its relation to the environment, which introduces several degrees of complexity to the task of classification.

Furthermore, we envision several extensions and advanced robot capabilities for our approach in Chapter 7: The robot should be able to simultaneously tidy up the environment by placing the impeding object at the intended location in the environment. Considering this extended scenario, the robot needs to keep track of the object it discovers and also the object purpose and the location where the object ideally should be placed. Erol *et al.* [126], presented an object tracking system for applications in home environments. The work describes in detail the robotic components to track objects in the environment with a human-robot interaction system. Similar approaches could be integrated into the constraint planning approach of Chapter 7 and expand further the capabilities of our work.

In this thesis we integrated a decision-making process of the robot into the navigation scheme (Fig. 2.1), to, e.g., choose a robot action when the path to the goal is blocked by an object. In Chapter 3 the decision making process is implied into the navigation cost function and allows for a foresighted navigation behavior. However, an open research question in robotics remains, how to implement reasoning in a scalable and domain independent manner. John McCarthy and Patrick J. Hayes [127] developed one of the first knowledge representations for artificial intelligence in the 1960s. The work was the starting point of many discussion of how to represent rational assumptions and common sense of humans, but no ground-breaking solution has emerged yet. We think that major progress in cognitive robotics could be achieved with new theories and algorithms, that enable high-level reasoning, can deal with uncertainties, and allow for greater degree of abstraction to cope with the vast amount of diverse information. The new generation of robot will then be able to make rational decisions, plan for complex tasks and flexibly react to situations, that were not experienced before.

APPENDIX

Algorithm 1 Reconstruct path

```
1: procedure RECONSTRUCT_PATH(P, s_n, g_n)
2:     Path ← ∅
3:     n_current ← g_n
4:     while n_current ≠ s_n do
5:         Path ← [Path, n_current]
6:         n_current ← P[n_current]
7:     end while
8:     return Path
9: end procedure
```

Algorithm 2 A* graph traversal

procedure A_STAR(s_n, g_n)
  $g(s_n) \leftarrow 0$
  $f(s_n) \leftarrow g(s_n) + h(s_n)$
  Open list: $O \leftarrow \{s_n\}$
  Closed list $C \leftarrow \emptyset$
  Parent map: $P \leftarrow \emptyset$
  while O is not empty **do**
    $n_{current} \leftarrow$ the node in O having the lowest f-value
    $C \leftarrow [C, n_{current}]$
    if $n_{current} == g_n$ **then**
      Goal node reached.
    end if
    for $n'_{neighbor} \in neighborhood(n_{current})$ **do**
      $g(n'_{neighbor}) \leftarrow g(n_{current}) + ec(n_{current}, n_{neighbor})$
      Compute $h(n_{neighbor}), f(n_{neighbor})$
      if $n_{neighbor} \in C$ and $cost < g(n_{neighbor})$ **then**
        remove $n_{neighbor}$ from C
      end if
      if $n_{neighbor} \in O$ and $cost < g(n_{neighbor})$ **then**
        remove $n_{neighbor}$ from O
      end if
      if $n_{neighbor} \notin O$ and $n_{neighbor} \notin C$ **then**
        $O \leftarrow [O, n_{neighbor}]$
        $g(n_{neighbor}) = g(n'_{neighbor})$
        $P[n_{neighbor}] = n_{current}$
      end if
    end for
  end while
  Return P
end procedure

ACRONYMS

API	Application Programming Interface
CNN	Convolutional Neural Network
COCO	Common Objects in Context
COVID	Coronavirus Disease
CPU	Central Processing Unit
DWA	Dynamic Window Approach
FOV	Field Of View
FPS	Frames Per Second
HSR	Human Support Robot
IRL	Inverse Reinforcement Learning
ISO	International Organization for Standardization
LR	Linear Regression
LSTM	Long Short-Term Memory
NN	Neural Network
PC	Predictive Controller
PG	Policy Gradient
POMDP	Partially Observable Markov Decision Process
RGB	Red Green Blue (color space)
RGB-D	Red Green Blue and Depth (color space)
RL	Reinforcement Learning
RMSE	Root Mean Square Error
ROS	Robot Operating System
SFM	Social Force Model
SLAM	Simultaneous Localization And Mapping
SLR	Simple Linear Regression
SVR	Support Vector Regression
TOF	Time Of Flight
VGG	Vickrey-Clarke-Groves
WEKA	Waikato Environment for Knowledge Analysis

LIST OF FIGURES

LIST OF TABLES

BIBLIOGRAPHY

[1] A Gasparetto and L Scalera, "From the unimate to the delta robot: The early decades of industrial robotics," in *Explorations in the History and Heritage of Machines and Mechanisms*, Springer, 2019.

[2] International Federation of Robotics (IFR).
[Online]. Available: `http://www.ifr.org`
Accessed on Mar. 2021.

[3] RoboCup@Home.
[Online]. Available: `https://athome.robocup.org/`
Accessed on Mar. 2021.

[4] European Robotics League.
[Online]. Available: `https://www.eu-robotics.net/robotics_league/`
Accessed on Mar. 2021.

[5] M. Matamoros, S. Viktor, and D. Paulus, "Trends, challenges and adopted strategies in robocup@ home," in *Proc. of the IEEE Intl. Conf. on Autonomous Robot Systems and Competitions (ICARSC)*, IEEE, 2019.

[6] iRobot Corporation.
[Online]. Available: `http://www.irobot.com`
Accessed on Mar. 2021.

[7] Toyota.
[Online]. Available: `https://global.toyota/`
Accessed on Mar. 2021.

[8] SQUIRREL Clearing Clutter Bit by Bit.
[Online]. Available: `http://www.squirrel-project.eu`
Accessed on Mar. 2021.

[9] A. Milioto and C. Stachniss, "Bonnet: An Open-Source Training and Deployment Framework for Semantic Segmentation in Robotics using CNNs," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[10] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT Press, 2005.

[11] Intel RealSense Depth Camera.
[Online]. Available: `https://www.intelrealsense.com/depth-camera-d455/`
Accessed on Mar. 2021.

[12] S. Yu, C. Fu, A. K. Gostar, and M. Hu, "A review on map-merging methods for typical map types in multiple-ground-robot slam solutions," *Sensors*, 2020.

[13] J. Guivant, E. Nebot, J. Nieto, and F. Masson, "Navigation and mapping in large unstructured environments," *Intl. Journal of Robotics Research (IJRR)*, 2004.

[14] I. Andersone, "Heterogeneous map merging: State of the art," *Robotics*, 2019.

[15] W. Rone and P. Ben-Tzvi, "Mapping, localization and motion planning in mobile multi-robotic systems," *Robotica*, 2013.

[16] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "An overview to visual odometry and visual slam: Applications to mobile robotics," *Intelligent Industrial Systems*, 2015.

[17] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Trans. on Robotics (TRO)*, 2007.

[18] G. Grisettiyz, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2005.

[19] J. Minguez and L. Montano, "Nearness diagram (ND) navigation: Collision avoidance in troublesome scenarios," *IEEE Trans. on Robotics (TRO)*, 2004.

[20] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 1999.

[21] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine (RAM)*, 1997.

[22] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. on Systems Science and Cybernetics (SSC)*, 1968.

[23] B. P. Gerkey and K. Konolige, "Planning and control in unstructured terrain," in *Proc. of the ICRA Workshop on Path Planning on Costmaps*, 2008.

[24] M. Quigley, K. Conley, B. Gerkey, *et al.*, "ROS: An open-source robot operating system," in *Proc. of the ICRA Workshop on Open Source Software*, 2009.

[25] ROS Local Planner.
[Online]. Available: `http://wiki.ros.org/base_local_planner`
Accessed on Mar. 2021.

[26] M. Missura and M. Bennewitz, "Predictive collision avoidance for the dynamic window approach," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[27] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

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

[29] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[30] M. B. Ralph-Christoph Weber, *Robotino manual*, Festo Didactic GmbH & Co. KG, Denkendorf, Germany, 2010.
[Online]. Available: `https://www.festo-didactic.com/ov3/media/customers/1100/544305_robotino_deen.pdf`
Accessed on Mar. 2021.

[31] D. Lu, D. Hershberger, and W. Smart, "Layered costmaps for context-sensitive navigation," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[32] E. Marder-Eppstein, E. Berger, T. Foote, *et al.*, "The office marathon: Robust navigation in an indoor office environment," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2010.

[33] A. Hornung, M. Phillips, E. G. Jones, *et al.*, "Navigation in three-dimensional cluttered environments for mobile manipulation," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2012.

[34] A. Orthey and O. Stasse, "Towards reactive whole-body motion planning in cluttered environments by precomputing feasible motion spaces," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2013.

[35] D. Maier, C. Lutz, and M. Bennewitz, "Integrated perception, mapping, and footstep planning for humanoid navigation among 3D obstacles," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[36] D. Joho, G. Tipaldi, N. Engelhard, *et al.*, "Nonparametric Bayesian models for unsupervised scene analysis and reconstruction," in *Proc. of Robotics: Science and Systems (RSS)*, 2012.

[37] E. B. Sudderth, A. Torralba, W. T. Freeman, and A. S. Willsky, "Describing visual scenes using transformed objects and parts," *Int. Journal of Computer Vision*, 2008.

[38] P. Henry, C. Vollmer, B. Ferris, and D. Fox, "Learning to navigate through crowded environments," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2010.

[39] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2004.

[40] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA Urban Challenge: Autonomous vehicles in city traffic*. Springer, 2009.

[41] S. Thrun, M. Bennewitz, W. Burgard, *et al.*, "Minerva: A second-generation museum tour-guide robot," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 1999.

[42] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[43] L. Murphy and P. Newman, "Risky planning on probabilistic costmaps for path planning in outdoor environments," *IEEE Trans. on Robotics (TRO)*, 2013.

[44] H. Jun and Z. Qingbao, "Multi-objective mobile robot path planning based on improved genetic algorithm," in *Proc. of the Intl. Conf. on Intelligent Computation Technology and Automation (ICICTA)*, 2010.

[45] R. Philippsen, S. Kolski, K. Macek, and B. Jensen, "Mobile robot planning in dynamic environments and on growable costmaps," in *Workshop on Planning with Cost Maps at the IEEE Intl. Conf. on Robotics and Automation*, 2008.

[46] M. Hall, E. Frank, G. Holmes, *et al.*, "The weka data mining software: An update," *ACM SIGKDD explorations newsletter*, 2009.

[47] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, 1995.

[48] F. Farina, D. Fontanelli, A. Garulli, *et al.*, "Walking ahead: The headed social force model," *PLOS ONE*, 2017.

[49] F. Zanlungo, T. Ikeda, and T. Kanda, "Social force model with explicit collision prediction," *EPL (Europhysics Letters)*, 2011.

[50] G. Ferrer, A. Garrell, and A. Sanfeliu, "Social-aware robot navigation in urban environments," in *Proc. of the Europ. Conf. on Mobile Robotics (ECMR)*, 2013.

[51] G. Ferrer, A. G. Zulueta, F. H. Cotarelo, and A. Sanfeliu, "Robot social-aware navigation framework to accompany people walking side-by-side," *Autonomous Robots*, 2017.

[52] C. Cao, P. Trautman, and S. Iba, "Dynamic channel: A planning framework for crowd navigation," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[53] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[54] A. Vemula, K. Muelling, and J. Oh, "Modeling cooperative navigation in dense human crowds," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[55] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard, "Feature-based prediction of trajectories for socially compliant navigation.," in *Robotics: Science and Systems*, 2012.

[56] D. Vasquez, B. Okal, and K. O. Arras, "Inverse reinforcement learning algorithms and features for robot navigation in crowds: An experimental comparison," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[57] B. Kim and J. Pineau, "Socially adaptive path planning in human environments using inverse reinforcement learning," *Intl. Journal of Social Robotics*, 2016.

[58] L. Tai, J. Zhang, M. Liu, and W. Burgard, "Socially compliant navigation through raw depth inputs with generative adversarial imitation learning," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[59] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[60] M. Pfeiffer, G. Paolo, H. Sommer, *et al.*, "A data-driven model for interaction-aware pedestrian motion prediction in object cluttered environments," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[61] A. Alahi, K. Goel, V. Ramanathan, *et al.*, "Social LSTM: Human trajectory prediction in crowded spaces," in *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[62] C. Blaiotta, "Learning generative socially aware models of pedestrian motion," *IEEE Robotics and Automation Letters (RA-L)*, 2019.

[63] M. Moussaïd, D. Helbing, S. Garnier, *et al.*, "Experimental study of the behavioural mechanisms underlying self-organization in human crowds," *Proc. of the Royal Society of London B: Biological Sciences*, 2009.

[64] M. Moussaïd, N. Perozo, S. Garnier, *et al.*, "The walking behaviour of pedestrian social groups and its impact on crowd dynamics," *PLOS ONE*, 2010.

[65] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, 2012.

[66] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Int. Conf. on Artificial Intelligence and Statistics*, 2010.

[67] J. Schulman, F. Wolski, P. Dhariwal, *et al.*, "Proximal policy optimization algorithms," *arXiv preprint*, 2017.

[68] J. Sergeant, N. Sünderhauf, M. Milford, and B. Upcroft, "Multimodal deep autoencoders for control of a mobile robot," in *Proc. of the Australasian Conf. on Robotics and Automation (ACRA)*, 2015.

[69] M. Pfeiffer, M. Schaeuble, J. Nieto, *et al.*, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[70] M. Pfeiffer, S. Shukla, M. Turchetta, *et al.*, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robotics and Automation Letters (RA-L)*, 2018.

[71] Y. Liu, A. Xu, and Z. Chen, "Map-based deep imitation learning for obstacle avoidance," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[72] A. Pokle, R. Martin-Martin, P. Goebel, *et al.*, "Deep local trajectory replanning and control for robot navigation," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[73] S. Gupta, J. Davidson, S. Levine, *et al.*, "Cognitive mapping and planning for visual navigation," in *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[74] S.-H. Hsu, S.-H. Chan, P.-T. Wu, *et al.*, "Distributed deep reinforcement learning based indoor visual navigation," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[75] X. Chen, A. Ghadirzadeh, J. Folkesson, *et al.*, "Deep reinforcement learning to acquire navigation skills for wheel-legged robots in complex environments," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[76] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[77] T. Fan, X. Cheng, J. Pan, *et al.*, "Crowdmove: Autonomous mapless navigation in crowded scenarios," in *Proc. of the IROS Workshop on From freezing to jostling robots: Current challenges and new paradigms for safe robot navigation in dense crowds*, 2018.

[78] H. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with AutoRL," *IEEE Robotics and Automation Letters (RA-L)*, 2019.

[79] J. Kulhánek, E. Derner, T. de Bruin, and R. Babuška, "Vision-based navigation using deep reinforcement learning," in *Proc. of the Europ. Conf. on Mobile Robotics (ECMR)*, 2019.

[80] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2015.

[81] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint*, 2016.

[82] M. Abadi, A. Agarwal, P. Barham, *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," *arXiv preprint*, 2016.

[83] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2009.

[84] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo localization: Efficient position estimation for mobile robots," *Proc. of the Conference on Advancements of Artificial Intelligence (AAAI)*, 1999.

[85] M. Grey, A. Ames, and C. Liu, "Footstep and motion planning in semi-unstructured environments using randomized possibility graphs," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[86] Y. Lin and D. Berenson, "Humanoid navigation in uneven terrain using learned estimates of traversability," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2017.

[87] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2014.

[88] M. Fallon, P. Marion, R. Deits, *et al.*, "Continuous humanoid locomotion over uneven terrain using stereo fusion," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2015.

[89] A. Hildebrandt, M. Klischat, D. Wahrmann, *et al.*, "Real-time path planning in unknown environments for bipedal robots," *IEEE Robotics and Automation Letters (RA-L)*, 2017.

[90] D. Wahrmann, A.-C. Hildebrandt, T. Bates, *et al.*, "Vision-based 3d modeling of unknown dynamic environments for real-time humanoid navigation," *The Int. Journal of Humanoid Robotics (IJHR)*, 2019.

[91] M. Stilman and J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *Intl. Journal of Robotics Research (IJRR)*, 2005.

[92] M. Stilman, K. Nishiwaki, S. Kagami, and J. Kuffner, "Planning and executing navigation among movable obstacles," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[93] M. Levihn, K. Nishiwaki, S. Kagami, and M. Stilman, "Autonomous environment manipulation to assist humanoid locomotion," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2014.

[94] A. Hornung, S. Boettcher, C. Dornhege, *et al.*, "Mobile manipulation in cluttered environments with humanoids: Integrated perception, task planning, and action execution," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2014.

[95] Y.-C. Lin and D. Berenson, "Humanoid navigation planning in large unstructured environments using traversability-based segmentation," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[96] A. Dornbush, K. Vijayakumar, S. Bardapurkar, *et al.*, "A single-planner approach to multi-modal humanoid mobility," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[97] P. Kaiser, D. Gonzalez-Aguirre, F. Schueltje, *et al.*, "Extracting whole-body affordances from multimodal exploration," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2014.

[98] M. Wang, R. Luo, A. Ö. Önol, and T. Padir, "Affordance-based mobile robot navigation among movable obstacles," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[99] L. Chen, G. Papandreou, I. Kokkinos, *et al.*, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Trans. on Pattern Analalysis and Machine Intelligence (TPAMI)*, 2018.

[100] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[101] H. Zhao, J. Shi, X. Qi, *et al.*, "Pyramid scene parsing network," *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[102] A. Milioto, P. Lottes, and C. Stachniss, "Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in CNNs," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[103] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: Deep neural network architecture for real-time semantic segmentation," *arXiv preprint*, 2016.

[104] E. Romera, J. Alvarez, L. Bergasa, and R. Arroyo, "ERFNet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Trans. on Intelligent Transportation Systems (ITS)*, 2018.

[105] M. Sandler, A. Howard, M. Zhu, *et al.*, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[106] *NVIDIA TensorRT. Programmable Inference Accelerator*.
[Online]. Available: `https://developer.nvidia.com/tensorrt`
Accessed on Mar. 2021.

[107] T. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft COCO: Common objects in context," *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2014.

[108] Google.
[Online]. Available: `https://images.google.com/`
Accessed on May 2021.

[109] P. Karkowski, S. Oßwald, and M. Bennewitz, "Real-time footstep planning in 3D environments," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2016.

[110] *REEM-C. A biped humanoid robot.*
[Online]. Available: `https://pal-robotics.com/robots/reem-c`
Accessed on Mar. 2021.

[111] A. Hornung, K. M. Wurm, and M. Bennewitz, "Humanoid robot localization in complex indoor environments.," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[112] D. Maier, A. Hornung, and M. Bennewitz, "Real-time navigation in 3D environments based on depth camera data," in *Proc. of the IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS)*, 2012.

[113] *ROS: Aldebaran Nao.*
[Online]. Available: https://wiki.ros.org/nao
Accessed on Mar. 2021.

[114] *NAOqi API.*
[Online]. Available: https://doc.aldebaran.com/2-1/naoqi/index.html
Accessed on Mar. 2021.

[115] *ROS: Humanoid localization.*
[Online]. Available: https://wiki.ros.org/humanoid_localization
Accessed on Mar. 2021.

[116] *ROS: OpenNI.*
[Online]. Available: https://wiki.ros.org/openni2_launch
Accessed on Mar. 2021.

[117] *ROS: Octomap.*
[Online]. Available: https://wiki.ros.org/octomap
Accessed on Mar. 2021.

[118] *ROS: RViz.*
[Online]. Available: https://wiki.ros.org/rviz
Accessed on Mar. 2021.

[119] *PAL Robotics*.
[Online]. Available: https://pal-robotics.com
Accessed on Mar. 2021.

[120] *GitHub: PAL Robotics*.
[Online]. Available: https://github.com/pal-robotics
Accessed on Mar. 2021.

[121] P. Zech, S. Haller, S. R. Lakani, *et al.*, "Computational models of affordance in robotics: A taxonomy and systematic classification," *Adaptive Behavior*, 2017.

[122] J. Sawatzky, A. Srikantha, and J. Gall, "Weakly supervised affordance detection," in *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[123] A. Nguyen, D. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis, "Detecting object affordances with convolutional neural networks," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[124] A. Roy and S. Todorovic, "A multi-scale cnn for affordance segmentation in rgb images," in *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, Springer, 2016.

[125] T. Luddecke and F. Worgotter, "Learning to segment affordances," in *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2017.

[126] B. A. Erol, A. Majumdar, J. Lwowski, *et al.*, "Improved deep neural network object tracking system for applications in home robotics," in *Computational Intelligence for Pattern Recognition*, Springer, 2018.

[127] J. McCarthy and P. J. Hayes, "Some philosophical problems from the standpoint of artificial intelligence," in *Machine Intelligence 4*, reprinted in McC90, Edinburgh University Press, 1969.