Socially Compliant Mobile Robot Navigation

Markus Kuderer

Technische Fakultät
Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

Betreuer: Prof. Dr. Wolfram Burgard

Socially Compliant Mobile Robot Navigation

Markus Kuderer

Dissertation zur Erlangung des akademischen Grades Doktor der Naturwissenschaften
Technische Fakultät, Albert-Ludwigs-Universität Freiburg

Dekan	Prof. Dr. Georg Lausen
Erstgutachter	Prof. Dr. Wolfram Burgard
Zweitgutachter	Prof. Dr. Kai O. Arras
Tag der Disputation	30. April 2015

Zusammenfassung

In den letzten Jahren ging die Entwicklung in der Robotik immer mehr zu flexiblen Robotern, die außerhalb ihrer traditionellen Einsatzgebiete in Fabriken Aufgaben im direkten Umfeld mit Menschen übernehmen. Ein bekanntes Beispiel für diese neue Generation von Robotern sind autonome Staubsauger, die sich frei in Wohnungen bewegen und damit im direkten Kontakt mit Menschen stehen. Darüber hinaus gibt es umfangreiche Forschung zur Entwicklung flexibler Industrieroboter, die ohne Schutzfeld mit menschlichen Angestellten zusammenarbeiten. Auch sollen Roboter in der Pflege eingesetzt werden, um beispielsweise in der Altenpflege die Herausforderungen des demographischen Wandels zu bewältigen.

Arbeiten Roboter im direkten Umfeld mit Menschen, werden andere Ansprüche an ihr Verhalten gestellt. Während traditionell Kollisionsvermeidung, Kostenoptimierung und baubedingte Beschränkungen für die Pfadplanung wichtig waren, sind nun auch die Empfindungen von Menschen als Reaktion auf den Roboter ausschlaggebend. Aus diesem Grund generieren traditionelle Pfadplanungsalgorithmen nicht immer das hier gewünschte Verhalten. Zusätzlich sollte ein Roboter soziale Konventionen verstehen und einhalten, die das tägliche Miteinander zwischen Menschen ermöglichen.

Betrachtet man das Verhalten von Fußgängern, so bewegen sich diese selbst in dicht gedrängten Bereichen meist flüssig und reibungslos. Dies scheint möglich durch soziale und kulturelle Regeln, die von Fußgängern während der Interaktion mit anderen eingehalten werden [45]. Insbesondere sind Menschen dazu in der Lage, das Verhalten von anderen zu einem gewissen Grad vorauszusehen und frühzeitig zu reagieren, um plötzliche, unangenehme Ausweichmanöver zu verhindern. Wir glauben, dass sich ein Roboter, der ein solches Verhalten versteht und selbst anwendet, besser in seine menschliche Umgebung einbringen kann. Das Ziel dieser Arbeit ist es daher, einen Roboter mit einem Modell von menschlichen Navigationsstrategien auszustatten.

Um dies zu ermöglichen, stellen wir in dieser Arbeit Modelle vor, die das natürliche Interaktionsverhalten von mehreren Menschen, oder auch von Menschen und Robotern abbilden. Die zentrale Idee hierbei ist, dass nicht das isolierte Verhalten eines Menschen oder Roboters modelliert wird, sondern explizit ihr gemeinsames Verhalten. Ein Roboter, der mit einem solchen Modell ausgestattet ist, kann das Verhalten von Fußgängern in seiner Umgebung vorhersehen und seine Bewegung frühzeitig daran anpassen. Darüber hinaus kann der Roboter aber auch seine eigenen Pfade mit Hilfe dieses Modells planen. Dies ermöglicht es dem Roboter, die erwarteten Reaktionen von Fußgängern schon in seine Pfadplanung mit einzubeziehen.

Unsere Modelle bilden zum einen diskrete Entscheidungen ab, wie zum Beispiel die Wahl, auf welcher Seite einem Fußgänger ausgewichen werden sollte. Zum anderen modellieren wir stetige Eigenschaften der Trajektorien, die für die Navigation wichtig sind,

wie die Zeit bis zum Ziel, die Distanzen zu Hindernissen, aber auch Geschwindigkeiten und Beschleunigungen. Durch Berücksichtigung dieser Eigenschaften bilden unsere Modelle zielgerichtetes Verhalten ab, so dass die modellierten Agenten ihr Ziel möglichst schnell, aber so angenehm wie möglich erreichen. Ein modularer Ansatz ermöglicht die flexible Anpassung der Modelle an die Eigenschaften der aktuellen Umgebung und an die Aufgabe des Roboters. Insbesondere parametrisieren Gewichte für die einzelnen Eigenschaften unsere Modelle und erlauben somit die komfortable Feineinstellung des Verhaltens.

Die Modellierung diskreter Navigationsentscheidungen basiert auf Trajektorien in verschiedenen Homotopieklassen. Homotopieklassen sind Mengen von Trajektorien, die Hindernisse in der Umgebung auf die gleiche Weise umgehen. In dieser Arbeit stellen wir effiziente Verfahren vor, um Trajektorien in diesen verschiedenen Klassen zu berechnen. Dafür verwendet unser Verfahren Voronoi-Graphen, mit deren Hilfe der Raum abstrakt dargestellt werden kann. Neben der Verwendung zur Modellierung menschlichen Navigationsverhaltens ist dieses Verfahren auch für die effiziente Initialisierung von Trajektorienoptimierungsverfahren einsetzbar.

Die Wahl der Parameter unserer Modelle bestimmt entscheidend die Qualität der Prädiktion sowie die Eigenschaften der geplanten Pfade. Insbesondere die gegensätzlichen Auswirkungen der einzelnen Stellgrößen, wie zum Beispiel der Abstand zu Hindernissen und die erlaubte Zeit zum Ziel, erschweren die manuelle Feineinstellung. Aus diesem Grund stellen wir ein Lernverfahren vor, das die Modellparameter aus observierten Trajektorien lernt. Zum einen ermöglicht dies das Lernen von akkuraten Modellen menschlichen Verhaltens, zum anderen können so Roboter durch Demonstrationen eingelernt werden. Dafür wird der Roboter per Fernsteuerung in seiner Umgebung mit den gewünschten Eigenschaften navigiert, was insbesondere für nicht-technische Benutzer eine bequeme Art der Roboterprogrammierung ist.

Neben dem Einsatz der vorgestellten Modelle für die autonome Roboternavigation, stellen wir in dieser Arbeit weitere Einsatzgebiete unserer Modelle für teilautonome Navigationsaufgaben vor. Zunächst ermöglichen unsere Methoden die Berechnung qualitativ unterschiedlicher Pfade. Ein Benutzer muss somit nur einen der vorgeschlagenen Pfade auswählen, während der Roboter die lokale Pfadplanung übernimmt und sichere, komfortable Trajektorien plant. Ein solches Verfahren ermöglicht es zum Beispiel Rollstuhlfahrern, die feinmotorisch nicht mehr in der Lage sind einen Rollstuhl zu bedienen, sich sicher und unabhängig in ihrer Umgebung zu bewegen. In einer weiteren Art teilautonomer Navigation soll der Roboter in der Nähe eines führenden Menschen bleiben, um ihm beispielsweise Werkzeuge anzubieten oder mit ihm zu interagieren. Der Mensch steuert auch hier nur die globale Navigation, während die lokale Pfadplanung von unseren Verfahren übernommen wird.

Die bisher genannten Anwendungen setzen eine robuste Perzeption, insbesondere der Menschen in der Umgebung, voraus. Während 2D Laser Scanner die Position und Bewegungsrichtung schätzen können, sind in vielen Anwendungen komplexere Modelle der menschlichen Bewegung gewünscht. Wir stellen daher ein Verfahren vor, das vollautomatisches Tracking der Skelettstruktur einer Person auf Basis von Markern ermöglicht. Neben der direkten Interaktion zwischen Menschen und Robotern ermöglicht ein sol-

ches System auch die Aufnahme von Bewegungsdaten für das Erlernen menschlicher Bewegungsmodelle.

Des Weiteren haben wir unsere Modelle im Bereich des autonomen Fahrens auf Autobahnen angewandt. Hier stellen sich andere Herausforderungen, allerdings lassen sich die zentralen Ideen unserer Arbeit auch auf diesen Bereich übertragen. Auch hier gibt es diskrete Entscheidungen, etwa einen Spurwechsel einzuleiten oder auf der aktuellen Spur zu bleiben. Innerhalb jeder dieser Entscheidung gibt es eine stetige Menge von Trajektorien mit unterschiedlichen physikalischen Eigenschaften, die von unseren Modellen repräsentiert werden. Wir stellen auch hier Verfahren vor, um individuelles Fahrverhalten von Demonstrationen zu lernen.

Alle Teile dieser Arbeit werden durch experimentelle Auswertungen evaluiert. Zum einen zeigen diese Experimente die Vorteile unserer Verfahren gegenüber dem aktuellen Stand der Technik. Zum anderen weisen Experimente mit echten Robotern darauf hin, dass unsere Ansätze auch praxistaugliche Roboternavigation ermöglichen. Für die Anwendungen in autonomer oder teilautonomer mobiler Robotik verwenden wir einen elektrischen Rollstuhl, der mit unseren Verfahren gesteuert wird. Im Bereich der Skelettschätzung stellen wir Experimente vor, die unsere Verfahren mit aktuellen, kommerziellen Produkten vergleichen. Darüber hinaus lernen wir individuelles Verhalten sowohl für den autonomen Rollstuhl als auch für autonome Automobile von Trainingsdaten, die mit echten Robotern, bzw. von echtem Fahrverhalten auf Autobahnen aufgenommen wurden. Zusammenfassend stellt diese Arbeit Fortschritte im Bereich der kooperativen, sozial kompatiblen Navigation vor, die uns der Entwicklung einer neuen Generation von Robotern einen Schritt näher bringen.

Abstract

The range of applications for robots currently broadens from performing repetitive, industrial tasks to flexible services in the same environment with humans. Nowadays, vacuum cleaning robots are already widely used domestic robots and we expect more and more services from mobile robotic platforms in the future. Robots that move in our vicinity should navigate in a socially compliant way that does not unnecessarily hinder nearby humans. Traditional path planning methods that compute time-optimal paths for the robot are therefore not always desirable. Instead, robots should understand and comply with social norms that allow the natural navigation behavior of humans. We belief that a robot that has a better understanding of this human-like behavior is more predictable and better accepted by humans.

In this thesis, we present a novel navigation scheme for mobile robots that actively integrates the natural navigation behavior of pedestrians into planning, instead of considering nearby humans as passive obstacles. The robot performs this reasoning in a predictive manner, which gives it the possibility to adapt its behavior early in an encounter to enable smooth evasive movements in contrast to abrupt collision avoidance maneuvers. In particular, we present a novel model of the cooperative, interactive navigation behavior of multiple agents. A mobile robot can use such a model to predict the navigation behavior of nearby pedestrians, and at the same time to plan socially compliant paths to its target location. The intention-driven model captures goal-directed behavior, i.e. the intention to reach a target efficiently but as comfortably as possible. To this end, we use features that model important properties of the navigation behavior such as the time to reach the target, velocities, and the proximity to obstacles. In addition to these continuous properties, our model reasons about discrete navigation decisions such as evading obstacles on the left, or on the right, which we capture in terms of topological classes of trajectories. During navigation, the robot maintains a probability distribution over trajectories in these classes, and chooses the socially most compliant one for navigation. This enables the robot to quickly react to sudden changes in its environment.

The contribution of this thesis is a novel model of the interactive navigation behavior of multiple agents and its application to mobile robot navigation. In addition to socially compliant navigation in the presence of humans, our methods enable shared-autonomy navigation and socially compliant leader following. To allow a robot to observe the behavior of humans it has to interact with, we present an online skeleton tracking method. In addition, we present a learning-from-demonstration approach for autonomous cars that is based on our model. In extensive experiments, carried out in simulation and with real robots, we evaluated the methods proposed throughout this thesis. These experiments suggest that our algorithms outperform traditional path planning methods and that our techniques are applicable to navigate mobile robots in a socially compliant way.

Acknowledgments

This thesis would never have been possible without the help and inspiration of many people to whom I want to express my gratitude. First of all, I thank Wolfram Burgard for his guidance during the time in the Autonomous Intelligent Systems lab at the University of Freiburg. He contributed great ideas for the general direction of my research and was also willing to discuss practical details whenever I needed it. In addition, he was a great teacher in scientific writing. I remember countless late night writing sessions during which he helped transforming our work into papers worthy of submitting to conferences. The many international conferences that he sent me to were a great experience and an invaluable source of insights about the current research in robotics. Wolfram Burgard also established the contact to the Hans Lutz Merkle Scholarship, which supported my work as a PhD student. In addition to their financial support, yearly presentation events and meetings with other HLMS scholars were a valuable complement throughout the time as a PhD student. Here, I especially want to thank my HLMS mentors Lothar Baum and Yasser Jadidi, who also frequently invited me to their departments for presentations and fruitful discussions with Bosch employees.

I would like to give special thanks to Henrik Kretzschmar. From him I learned a great deal about programming, writing scientific papers, and research as such. Even more importantly, the joint work with him over the years was a great pleasure. Late hours in the cold experimental hall with robots that won't work when they are needed the most inevitably binds together.

Overall, I can only thank all members of the Autonomous Intelligent Systems lab for a great working environment. It is a great experience to have lots of experienced and intelligent people around, where I could always find a sympathetic ear for any kind of problems. I would like to single out Christoph Sprunk who was always willing to contribute with his inexhaustible experience and knowledge. Thanks to him and to Nichola Abdo for being the best office mates one can wish for. Furthermore, I would like to express my gratitude for many fruitful discussions with my colleagues Andreas Wachaja, Axel Rottmann, Barbara Frank, Benjamin Suger, Boris Lau, Felix Endres, Jörg Müller, Jörg Röwekämper, Jürgen Hess, Maximilian Beinhofer, and Tobias Schubert.

During my work in the AIS lab I had the chance to supervise student projects during which it has been a pleasure to work together with Alexander Thiemann, Daniel Stratz, Heinke Hihn, Johannes Meyer and Philipp Ruchti. I also want to thank Achim Leydecker and Mario Harter who did a great job in their student assistant jobs.

Besides the academic work I also enjoy remembering events such as playing the saxophone in our AIS band, which performed an annual concert at the AIS Christmas celebrations. For this experience I would like to thank my fellow musicians Andreas Eitel, Andreas Wachaja, Benjamin Suger, Felix Endres, Jonas Rist, Maximilian Beinhofer,

Michael Herman, and Wolfram Burgard.

In the last year of my time as a PhD student, I visited the Bosch research lab in Palo Alto as a guest researcher. In this time, I had the chance to apply methods that I developed previously in the AIS lab to autonomous driving. I would like to thank Shilpa Gulati for supervising my work at Bosch RTC.

I want to express my gratitude to my friends and family for their support also in challenging times. Thanks to my father, who early in my life aroused my interest in tinkering with electronics, dismantling all kinds of devices and playing with computers, which lay the foundation for my further studies in the area of computer science and robotics. Finally, and most importantly, Carla never ceased to encourage me in what I was doing, and reminded me of a world outside of robotics.

Contents

Chapter 1

Introduction

Mobile robots are envisioned to provide more and more services in a shared environment with humans. The applications for such robots range from robotic co-workers in factories over domestic service robots to assistive robots in healthcare. For tasks that require robots to provide services at different locations, they need reliable methods to compute safe navigation paths. However, in a shared environment with humans the navigation behavior should not only be save but also socially compliant, i.e., the robots should behave in a way that is comfortable for nearby humans. Therefore, traditional approaches to mobile robot navigation, such as moving on time-optimal paths, are not always desirable. Instead, the robots should understand and comply with social norms that allow humans to navigate even in crowded environments.

The effortless navigation of pedestrians in crowded environments seems to be possible due to mutually accepted rules that we follow when interacting with other pedestrians [45]. In particular, humans are able to predict the intent of others to some extent and also to reason about the reaction of others to their actions. This allows pedestrians to smoothly adjust their movements at an early stage of an encounter, avoiding sudden and uncomfortable evasive movements. Our goal is to let mobile robots participate in such a natural navigation behavior in order to enable socially compliant human-robot interaction.

To this end, we propose to endow mobile robots with a model of the natural navigation behavior of interacting agents. The model we propose in this thesis captures the interactive behavior of multiple agents in a navigation task in terms of a distribution over their future trajectories. A mobile robot can not only use such a model to predict the navigation behavior of nearby pedestrians, but at the same time to plan its own navigation trajectories. A key aspect of the approach we present in this thesis is that we model the joint behavior of the robot and the pedestrians. This allows the robot to incorporate the reactions of the pedestrians to its own actions into planning.

In particular, the proposed model captures discrete navigation decisions such as on which side to evade an obstacle or a pedestrian, as well as continuous navigation decisions that affect the resulting trajectories in terms of the travel time, distances to obstacles and higher-order dynamics such as velocities and accelerations. We present a set of features that capture these important properties for natural navigation behavior. As a result, the model is intention-driven, i.e., we model the goal-directed behavior of the agents to reach a target position as comfortable as possible without colliding with obstacles. Feature weights determine the importance of each feature, which serves as an intuitive way to

adapt the model to the dedicated task of the robot.

To capture the discrete navigation decisions, we use homotopically distinct navigation paths, which correspond to the different possibilities to evade obstacles on the left, or on the right. In this thesis, we present methods to efficiently compute these different trajectory classes. To this end, we use the Voronoi graph of the environment and compute an abstract graph that captures the connectivity of the free space of the environment. In this graph, we can efficiently explore the distinct homotopy classes. In addition to the application to socially compliant mobile robot navigation, we show how this technique is beneficial for parallel initialization of trajectory optimization schemes.

The behavior of nearby pedestrians, as well as the desired behavior of the robot, depends on the environment the robot is employed in. Therefore, we present methods that allow us to learn the model parameters, i.e., the feature weights, from demonstrated trajectories. As a result, the robot can infer the model parameters that best fit the behavior of observed pedestrians. Furthermore, if the robot should not simply replicate human navigation strategies, it facilitates teaching the robot by tele-operation. Teaching by demonstration is an intuitive way to program a robot, especially for non-experts.

During navigation, the robot utilizes the proposed model to maintain a probability distribution over the trajectories of the pedestrians, and corresponding trajectories for the robot itself. The robot then efficiently computes the most likely interaction by optimizing trajectories with respect to the probability density function. The resulting most likely interaction encodes the best guess of the future behavior of the pedestrian, as well as a corresponding socially compliant plan for the robot. This plan is a valid trajectory that the robot can directly apply for navigation. The proposed efficient techniques allow the robot to replan continuously during navigation, to account for changes in the environment.

In addition to a comprehensive discussion of the underlying model, we present a set of applications in which a mobile robot benefits from the proposed model. It allows a mobile robot to navigate to its goal location in populated environments while smoothly evading pedestrians in a socially compliant manner. Furthermore, our model is capable of computing a set of qualitatively distinct paths, which a robot can utilize for shared autonomy navigation. Here, a user selects the desired path on a high level, whereas the robot takes over the low level control. A further application is the task of following a human leader in a socially compliant way, in which the robot should stay close to a desired relative position with respect to the human. For all these tasks it is beneficial for the robot to have a better understanding of the intent of the pedestrians, which allows it to predict their navigation behavior and to act in an appropriate manner.

In the abovementioned applications, the robot needs to perceive its environment, especially the behavior of nearby pedestrians. Whereas in many applications the estimated position from a 2D laser-based tracking system suffices, it can be beneficial to have more information about the full body posture, for example to perceive the body language of humans. To this end, we propose a robust technique for fully automatic skeleton tracking in optical motion capture. A mobile robot can use such a technique to observe humans during online interaction tasks, or to record datasets for learning navigation behavior models.

We furthermore applied our approach to learning individual styles for autonomous cars.

While highway driving poses different properties and challenges, the fundamental idea of our approach does still apply. Here, we also have distinct decisions such as staying in lane behind an other car or overtaking, and continuous properties of the trajectories within such a behavior class. In the context of highway driving, we use our method to learn individual driving styles from demonstration, which an autonomous car can then use to replicate the desired behavior in autonomous mode.

Extensive experiments in all parts of this thesis, including quantitative comparisons to related methods, suggest that our approaches outperform state-of-the-art techniques. For perception, we show that our method is better able to track a human skeleton compared to a current, commercial software. The generation of homotopically distinct navigation paths outperforms existing methods by an order of magnitude in run-time, and further experiments aim to show the advantages of our novel method for socially compliant mobile robot navigation over traditional path planning techniques in different applications. We furthermore present experiments in which we teach a desired behavior by demonstration, both for a robotic wheelchair as well as for autonomous cars. We implemented our approaches on real robots and evaluated their performance in real-world scenarios. With this thesis, we contribute methods towards a new generation of mobile robots that are able to navigate socially compliantly in human environments.

1.1 Contributions

The key contribution of this thesis is a framework for socially compliant mobile robot navigation. The goals for a mobile robot that provides services in the direct presence of humans are different from traditional path planning methods. For such mobile robots, we propose methods that allow a robot to better understand the natural navigation behavior of pedestrians, and to behave accordingly. In particular the key contributions are:

- A method for online tracking of a human skeleton in motion capture. This method allows a mobile robot to record training data for learning models of human navigation behavior, or to observe humans during interaction. (Chap. 3)
- An online method for generating a set of homotopically distinct navigation paths, which serves as a basis for the models we propose in this thesis. (Chap. 4)
- A model of the natural, cooperative navigation behavior of multiple agents. In particular, this model includes discrete as well as continuous navigation decisions that capture the navigation behavior of humans. (Chap. 5)
- Methods for teaching a mobile robot by tele-operation. (Chap. 6)
- Applications of the proposed models to socially compliant leader following and shared autonomy. (Chap. 7 and Chap. 8)
- Learning individual navigation styles for highway driving from demonstration. (Chap. 9)

1.2 Publications

This thesis is based on our previous work presented in the following conference proceedings and workshop proceedings.

- M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard. Feature-based prediction of trajectories for socially compliant navigation. In *Proceedings of Robotics: Science and Systems (RSS)*, Sydney, Australia, 2012.

- M. Kuderer, H. Kretzschmar, and W. Burgard. Teaching mobile robots to cooperatively navigate in populated environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 2013.

- H. Kretzschmar, M. Kuderer, and W. Burgard. Predicting human navigation behavior via inverse reinforcement learning. In *The 1st Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, Princeton, NJ, USA, 2013.

- H. Kretzschmar, M. Kuderer, and W. Burgard. Learning navigation policies from human demonstrations. In *Proceedings of the Workshop on Inverse Optimal Control & Robotic Learning from Demonstration at Robotics: Science and Systems (RSS)*, Berlin, Germany, 2013.

- H. Kretzschmar, M. Kuderer, and W. Burgard. Inferring navigation policies for mobile robots from demonstrations. In *Proceedings of the Autonomous Learning Workshop at the IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013.

- J. Meyer, M. Kuderer, J. Müller, and W. Burgard. Online marker labeling for automatic skeleton tracking in optical motion capture. In *Workshop on Computational Techniques in Natural Motion Analysis and Reconstruction at the IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013.

- H. Kretzschmar, M. Kuderer, and W. Burgard. Learning to predict trajectories of cooperatively navigating agents. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Hong Kong, China, 2014.

- M. Kuderer, C. Sprunk, H. Kretzschmar, and W. Burgard. Online generation of homotopically distinct navigation paths. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Hong Kong, China, 2014.

- J. Meyer, M. Kuderer, J. Müller, and W. Burgard. Online marker labeling for fully automatic skeleton tracking in optical motion capture. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Hong Kong, China, 2014.

- M. Kuderer and W. Burgard. An approach to socially compliant leader following for mobile robots. In *Social Robotics*, volume 8755, Sydney, Australia, 2014.
  Best Presentation Award

- M. Kuderer, S. Gulati, and W. Burgard. Learning driving styles for autonomous vehicles from demonstration. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Seattle, USA, 2015.
  Accepted for publication

The following publications that are not included in this thesis also originate from the author's work at the research group.

- S. Ito, F. Endres, M. Kuderer, G. D. Tipaldi, C. Stachniss, and W. Burgard. W-RGB-D: Floor-plan-based indoor global localization using a depth camera and wifi. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Hong Kong, China, 2014.

- E. Demeester, E. V. Poorten, A. Hüntemann, J. D. Schutter, M. Hofmann, M. Rooker, G. Kronreif, B. Lau, M. Kuderer, W. Burgard, A. Gelin, K. Vanopdenbosch, P. V. der Beeten, M. Vereecken, S. Ilsbroukx, A. Fossati, G. Roig, X. Boix, L. V. Gool, H. Fraeyman, L. Broucke, H. Goessaert, and J. Josten. Robotic ADaptation to Humans Adapting to Robots: Overview of the FP7 project RADHAR. In *International Conference on Systems and Computer Science (ICSCS)*, Villeneuve d'Ascq, France, 2012.

1.3 Collaborations

Parts of this thesis are the result of collaboration with other researchers. Chap. 3 on skeleton tracking techniques extends the Bachelor thesis of Johannes Meyer, which the author of this thesis supervised together with Jörg Müller. Chap. 4 on generating homotopically distinct navigation paths was joint work with Christoph Sprunk. The computation of the abstract graph that captures the free-space of the environment is based on incrementally updatable Voronoi maps, which is previous work of Christoph Sprunk. The model of the natural navigation behavior of interacting agents that we present in Chap. 5 was developed in collaboration with Henrik Kretzschmar. Based on the joint work on the underlying model, Henrik Kretzschmar's main contribution was the development of machine learning algorithms that we use for learning-from-demonstration applications (Chap. 6), whereas the author of this thesis focused on socially compliant mobile robot navigation. The approach for navigating autonomous cars on highways and for learning individual navigation styles was developed during an internship with Bosch RTC in Palo Alto, which was supervised by Shilpa Gulati. Wolfram Burgard was the supervisor of this thesis and contributed ideas to all its parts.

Chapter 2

Background

The focus of this thesis is robot navigation in the presence of humans. To navigate, a mobile robot needs path planning algorithms that compute the control commands leading the robot to its target location. Since the ability to navigate in its environment is a basic requirement for mobile robots, research on path planning has been extremely active in the last decades. In this chapter, we introduce a number of path planning techniques that are relevant in the following chapters and describe methods that we leverage, or compare against, throughout this thesis. We briefly discuss different methods to find paths from the current position of the robot to a target location and describe two methods that capture the interactive behavior of multiple agents. These interaction models apply to control multi-robot systems, or to model the navigation behavior of pedestrians.

2.1 Shortest Paths in a Graph

In many applications for mobile robot navigation, we can represent the environment of the robot as a graph, for example as 2D grids, topological maps, or other structures that capture the connectivity of the environment. In general, a graph $G = (V, E)$ is a set of vertices V with a set of edges E that connect the vertices. In graphs used for navigation, each edge has an assigned cost that corresponds to the expected travel time, the distance, or other properties that are important for the navigation task. The goal is to find the best path from the current state of the robot to a goal state, i.e., the path in the graph connecting the two corresponding vertices that has the least accumulative cost.

In 1959, Dijkstra [36] proposed an algorithm to find the shortest path in a graph with non-negative edge costs in run time $O\left(|V|^2\right)$. The idea of this algorithm is to iteratively expand vertices by considering all so far unvisited neighbors of the current vertex. During this procedure, the algorithm keeps track of the tentative cost, i.e., the cost of the currently shortest known path from the start to any of the visited vertices. In each step, the algorithm expands the unvisited vertex with the least tentative cost next. After the goal vertex has been visited, the algorithm returns the path with the least cost.

The performance of Dijkstra's algorithm can be improved by using heuristics that estimate the cost from the current vertex to the goal. An extension of Dijkstra's algorithm, known as the A* algorithm [50], computes optimal paths by using an admissible heuristic of the path cost between any two vertices. Admissible in this context means that the heuristic never overestimates the cost. For example, in a 2D grid where the cost function

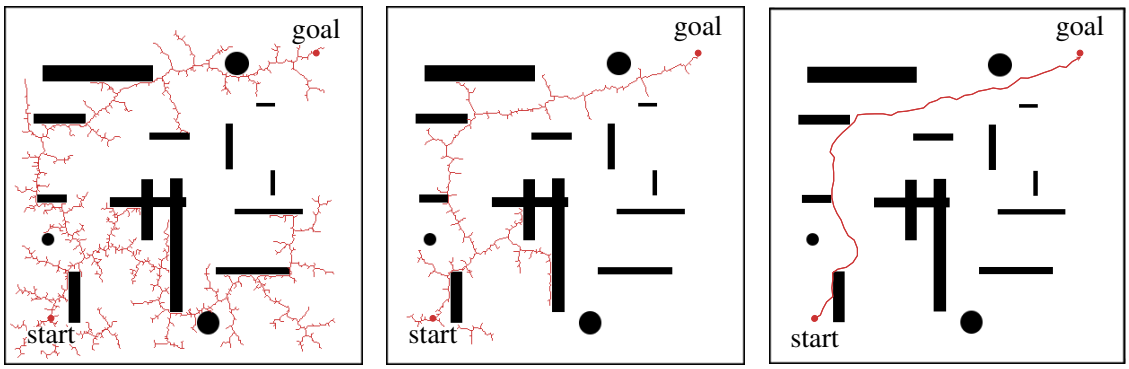


Figure 2.1: Illustration of the RRT path planning algorithm. The search starts in the bottom left corner and the goal is in the top right corner. In each iteration the RRT algorithm attempts to grow the tree towards a sampled position using short straight line edges. Left: search tree where the goal location was sampled with a probability of 0.001. Middle: search tree where the goal location was sampled with a probability of 0.1. Right: resulting trajectory from the RRT tree in the middle image.

captures the Euclidean distance, the linear distance between two vertices is an admissible heuristic. In contrast to Dijkstra's algorithm that greedily selects the vertex with the currently least cost, A* selects in each step the vertex with the least estimated cost to the target, i.e., the sum of the currently least cost to the vertex and the estimated cost from this vertex to the goal, given by the heuristic. As Dijkstra's algorithm, A* guarantees to return the path with the least cost after the goal vertex has been selected.

The resulting path of a graph-based search is often not directly feasible for navigation. For example, a car-like robot cannot follow a path consisting of linear segments in a 2D grid. A typical solution to this problem is to follow the planned path reactively using local methods such as the dynamic window approach [41]. An alternative solution is to plan paths that are directly suitable for robot navigation, which we discuss in the following.

2.2 Rapidly Exploring Random Trees

Graph search reaches its limit when considering continuous configuration spaces, for example when including higher order properties such as velocities and accelerations into planning. In general, we can discretize these properties and extend the state space to also capture velocities and accelerations. However, in practice graph search becomes infeasible at high dimensions due to the large search space.

Sampling-based algorithms overcome these problems by incrementally growing a search tree in the continuous state space. A popular sampling-based method is the rapidly exploring random tree (RRT) presented by LaValle and Kuffner [94]. It iteratively builds a tree by repeatedly sampling configurations in the continuous space. The algorithm then attempts to extend the closest vertex towards the direction of the sampled configuration, mostly with a fixed maximum length of the extension. If the extension succeeds, i.e., it passes free space and satisfies further constraints, the algorithm adds a new vertex at the end point of the extension, and a new edge connects the tree to this vertex. In this way,

the tree efficiently explores the state space.

The convergence of the algorithm is heavily affected by the probability distribution from which the samples are drawn. Adding the goal state with a higher probability to the distribution increases the performance considerably in practice. Fig. 2.1 shows two different runs of a RRT algorithm, which drew the goal state with a probability of 0.001, and 0.1, respectively. The first run produces a much larger tree, exploring more parts of the search space compared to the more goal-directed second run.

Karaman and Frazzoli [68] present RRT*, an extension to the RRT algorithm for asymptotically optimal motion planning. This algorithm continues refining the path after it has found the initial solution. However, this approach assumes that it is possible to connect two arbitrary states in the free space with an optimal trajectory. While straight lines satisfy this requirement for distance-minimizing holonomic robots, it is a non-trivial problem for non-holonomic robots, or kinodynamic cost functions. Furthermore, the convergence time may restrict the application of such algorithms in online path planning scenarios.

2.3 Optimizing Trajectories

An alternative to sampling based path planning is the optimization of an initial guess with respect to a navigation cost function. For some special cost functions, it is possible to compute the optimal trajectory in closed form. Examples for this class of problems are the trivial case of straight lines for minimal-distance paths, or finding trajectories that minimize the integrated jerk [91] in an environment without obstacles.

Real world mobile robot approaches typically employ cost functions that capture a trade-off between a variety of desired properties such as the travel time, velocity and acceleration constraints, and proximity to obstacles. In general, it is not feasible to compute closed form solutions given such complex cost functions. However, optimization techniques provide tools to find optimal or near-optimal trajectories.

An important part of optimization methods is a suitable finite dimensional trajectory representation. Examples for such representations are sequences of points along the trajectory [66], or splines [136], which are piecewise-polynomial functions. Together with a cost function that maps the parameters of the trajectory model to a real value, the navigation task translates to an optimization problem.

The goal is to find the values of the model parameters that yield the trajectory with minimum cost. If the navigation task can be stated in a way that the resulting optimization problem is convex, there exists one unique minimum, which simplifies the optimization considerably [149]. For general, non-convex navigation problems different optimization techniques converge to local minima, but it is difficult to find the global minimum. Approaches to solve this problem are sampling based optimization techniques [66], or multiple initialization as we propose in Chap. 4.

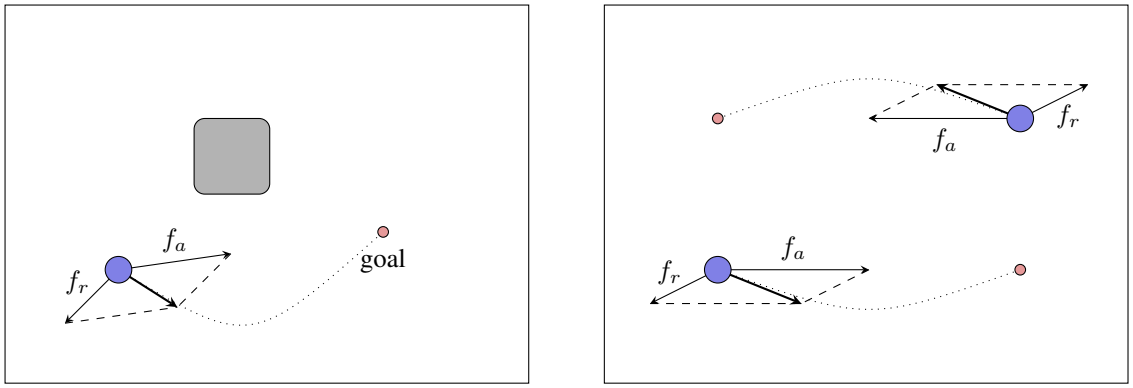


Figure 2.2: Illustration of social forces. Left: The goal yields an attractive force f_a, the static obstacle a repulsive force f_r. The algorithm adapts the agent's motion with respect to the resulting force $f_a + f_r$. Right: social forces in a scenario with two agents. Each agent gets attracted by its goal location. In addition, the agents repel each other. The resulting force used to modify the agents' velocity is the sum of the forces for each agent, which yields an evasive movement shown by the dotted lines.

2.4 Social Forces

In contrast to the previously discussed path planning algorithms that compute paths from the current location of the robot to the goal, reactive approaches map the current state of the environment directly to a control command of the robot. One class of such reactive approaches is based on artificial potential fields [70].

In artificial potential field methods, the environment in which the robot moves has a minimum potential at the goal location, and high potentials in the area of obstacles. Intuitively, the environment resembles a landscape with mountains, where the robot always travels downhill. The negative gradient of the potential, which is a sum of the attractive force f_a and of repulsive forces f_r points towards the target and away from obstacles. Fig. 2.2 (left) illustrates the resulting forces for an environment with one agent and a static obstacle. For navigation, the robot follows the negative gradient direction by changing its velocity according to the resulting force $f_a + \sum f_r$.

Helbing and Molnar [53] applied the principle of attractive and repulsive forces to model the interaction between multiple agents such as pedestrians. Here, the agents additionally exert repulsive forces to each other, modeling the intent of all agents to avoid collisions. As a result, the agents mutually evade each other during navigation, similar to the natural navigation behavior of pedestrians. Fig. 2.2 (right) illustrates social forces in an example with two agents that are attracted by their goal locations. The repulsive force pushes the agents apart, resulting in an evasive maneuver.

Social forces have been successfully applied to large scale pedestrian simulations. However, as other potential field methods, they have inherent limitations when applied to mobile robot navigation, as pointed out by Koren and Borenstein [77]. Most importantly, the locally computed forces are prone to get stuck in local minima of the environment and have difficulties to navigate narrow passages between obstacles. Furthermore, the forces may lead to oscillating behavior near obstacles.

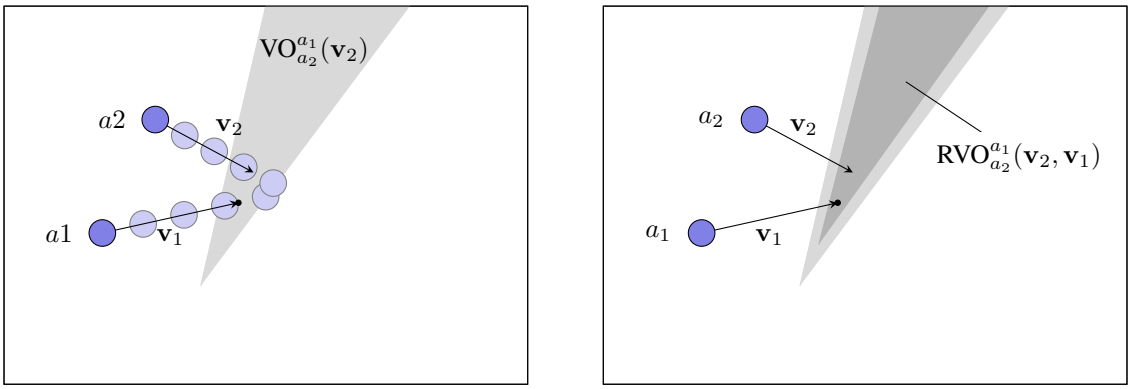


Figure 2.3: Illustration of the reciprocal velocity obstacles method. Left: the light gray area shows the velocity obstacle of agent a_1 with respect to agent a_2. It contains all velocities that lead to a collision at some point in time if a_2 does not change its velocity. Right: extension to reciprocal velocity obstacles. The figure shows in dark gray the reciprocal velocity obstacle for agent a_1 with respect to the other agent. If each agent selects a velocity outside its reciprocal velocity obstacle, they are guaranteed to pass each other without collision.

2.5 Reciprocal Velocity Obstacles

A further navigation algorithm that models the interactive behavior of multiple agents is the reciprocal velocity obstacles (RVO) approach by van den Berg et al. [143]. This approach builds on top of velocity obstacles, i.e., the set of velocities that would lead the agent into collision.

The velocity obstacle $\text{VO}_{a_2}^{a_1}(\mathbf{v}_2)$ is the set of all velocities $\mathbf{v}_1$ of agent a_1, which lead to a collision at some point in time if agent a_2 continues with velocity $\mathbf{v}_2$. Fig. 2.3 (left) illustrates the velocity obstacle for two agents a_1 and a_2. Given the current velocity $\mathbf{v}_2$ of agent a_2, the two agents collide at some point if agent a_1 continues with its velocity $\mathbf{v}_1$. Therefore, this velocity is inside the velocity obstacle, i.e., $\mathbf{v}_1 \in \text{VO}_{a_2}^{a_1}(\mathbf{v}_2)$. The gray area comprises all such velocities $\mathbf{v}_1$ for which the agents would collide at some point.

The RVO approach is an extension of VO, which takes into account mutual evasive maneuvers. Geometrically, the $\text{RVO}_{a_2}^{a_1}(\mathbf{v}_2, \mathbf{v}_1)$ is $\text{VO}_{a_2}^{a_1}(\mathbf{v}_2)$, shifted by $\frac{\mathbf{v}_1 - \mathbf{v}_2}{2}$, where $\mathbf{v}_1$ is the current velocity of agent a_1. This set contains all velocities for agent a_1 that are the mean of the current velocity $\mathbf{v}_1$, and a velocity inside the velocity obstacle $\text{VO}_{a_2}^{a_1}(\mathbf{v}_2)$. Fig. 2.3 (right) illustrates the reciprocal velocity obstacle for the same example as before.

During navigation, two agents can use RVO for collision-free motions. To this end, both agents compute the reciprocal velocity obstacle with respect to the other agent. Then both agents continuously choose a velocity outside this set, i.e., agent a_1 chooses a velocity $\mathbf{v}_1 \notin \text{RVO}_{a_2}^{a_1}(\mathbf{v}_2, \mathbf{v}_1)$, and agent a_2 chooses a velocity $\mathbf{v}_2 \notin \text{RVO}_{a_1}^{a_2}(\mathbf{v}_1, \mathbf{v}_2)$. If they choose the velocity that satisfies this constraint and that is closest to their current velocity, the mutual behavior is guaranteed to be collision- and oscillation free.

The concept of RVO can also be generalized to multiple agents by computing the union of the pairwise computed reciprocal velocity obstacles. In this way, we could coordinate a multi-robot swarm, or model the behavior of a group of pedestrians. More details on this approach can be found in the work of van den Berg et al. [143].

Chapter 3

Fully Automatic Skeleton Tracking in Optical Motion Capture

Mobile robots that operate in populated environments need methods to perceive the behavior of humans in their vicinity. Such methods allow a mobile robot to directly interact with the human, or to observe human behavior in order to learn a model of their behavior. Accurately tracking the human posture is furthermore important in applications such as animation, interaction, orthopedics, and rehabilitation. For marker-based systems major challenges are to associate the observed markers with skeleton segments, to track markers between consecutive frames, and to estimate the underlying skeleton configuration for each frame. Existing solutions to this problem often assume fully labeled markers, which usually requires labor-intensive manual labeling, especially when markers are temporally occluded during the movements. In this chapter, we propose a fully automated method to initialize and track the skeleton configuration of humans from optical motion capture data without the need of any user intervention. Our method applies a flexible T-pose-based initialization that works with a wide range of marker placements, robustly estimates the skeleton configuration through least-squares optimization, and exploits the skeleton structure for fully automatic marker labeling. We demonstrate the capabilities of our approach for online skeleton tracking and show that our method outperforms solutions that are widely used and considered as state of the art.

3.1 Introduction

Social robots that directly interact with humans rely on a perception system that provides an accurate estimation of the humans' behavior. In a variety of applications we are interested not only in the position of the humans, but also in their full body posture. For example, the head direction, or the pose of the arms are valuable information during human-robot interaction. The methods thereby range from estimating human body

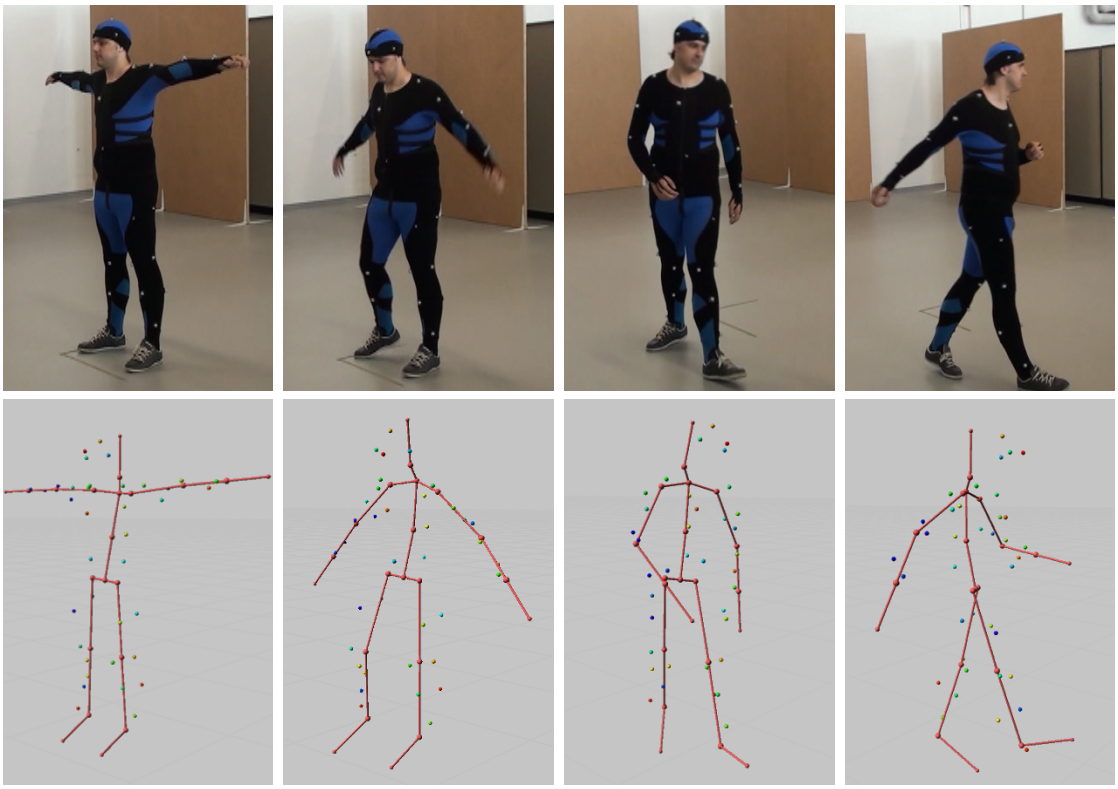


Figure 3.1: Skeleton pose estimation. Top: Human with passive optical markers attached to the body. Bottom: Observed markers and the skeleton configuration as estimated by our fully automated method.

postures in single images [115], from depth images [133] or based on artificial markers. The advantages of marker-based optical motion capture systems are that they provide automatic calibration procedures and precise position information about the markers at a high frame rate. Compared to marker-less approaches, marker-based methods are typically more accurate and at the same time are more robust against occlusions [111]. Accordingly, they are perfectly suited to accurately capture even fast movements of people.

However, inferring the body pose in terms of the *skeleton configuration*, i.e., the global pose and the joint angles of the underlying skeleton, from raw 3D marker position data is a challenging task. Although the camera system provides accurate 3D marker positions, their association to the individual markers placed on the person is initially unknown. This data association problem is called *labeling* and is usually solved by analyzing the geometric structure of the set of detected 3D marker positions, which requires tedious and time-consuming manual work even with state-of-the-art software. Furthermore, markers are occasionally occluded by parts of the body or objects around the tracked person, which makes the labeling of the remaining and especially the re-appearing markers even more challenging. Finally, given the labeling of the markers, inferring the skeleton configuration requires to take into account that the markers are only attached to the skin or to clothes. Hence, the skin movement causes the markers to slightly move with respect to the bones during the activity, so that the skeleton configuration needs to be inferred in a robust way.

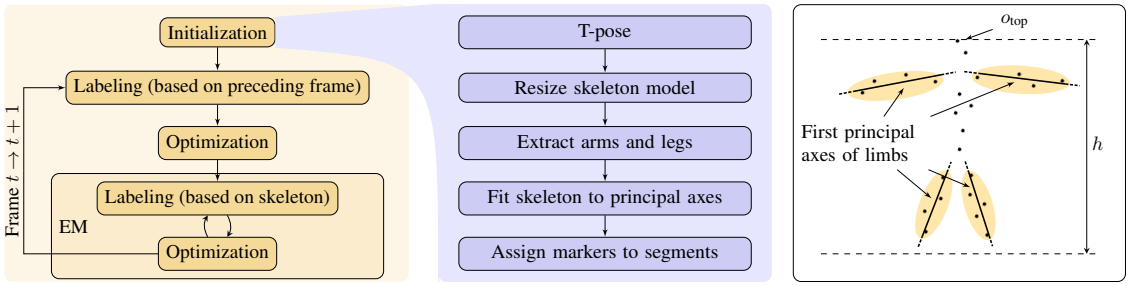


Figure 3.2: Overview of the proposed method for automated skeleton estimation. Based on the observed markers in the first frame our method adjusts the skeleton model to the estimated height and fits the skeleton into the observed point cloud using the principal axes of arms and legs (middle and right). Furthermore, it initializes the marker labeling and their association to skeleton segments. Left: In each successive frame, the most observed points are labeled based on the preceding frame by nearest neighbor association. Repeated optimization of the skeleton configuration and association based on the current skeleton estimate robustly labels the remaining points.

In this chapter, we propose a novel fully automatic skeleton tracking technique. Our method is flexible and includes the labeling of passive markers, which we can place on arbitrary positions on the human body, without any manual effort. We propose to use a parameterized standard human skeleton model, which we automatically adapt to humans of different size, and assume that each marker is attached to one of the limbs of this skeleton. Our approach to skeleton tracking jointly estimates the labeling of the observed markers and the skeleton configuration. In contrast to other approaches, our method exploits the information about the human skeleton during the labeling step and therefore provides a more informed data association.

Our approach initializes the skeleton tracking based on a T-pose executed by the person being tracked (see Fig. 3.1). Our method uses this initialization step to scale the skeleton to the person's size and aligns the skeleton to the person's limbs. During tracking, it then in an alternating fashion labels the observed markers and optimizes the skeleton configuration in an expectation-maximization-like procedure [33]. Thereby, it exploits both the marker positions of the most recent frame and the current skeleton configuration to obtain a consistent labeling and to reliably identify re-appearing markers that were temporally occluded.

We implemented our approach and evaluated it in extensive experiments. The results demonstrate the effectiveness and reliability of our approach and show that it outperforms *Cortex*, a state-of-the-art commercial solution for tracking articulated objects. Furthermore, we present results indicating that our method is highly efficient and enables online skeleton tracking on a standard desktop computer.

3.2 Basics

The goal of this section is to give a formal problem definition and to present the foundations of online marker-based skeleton tracking.

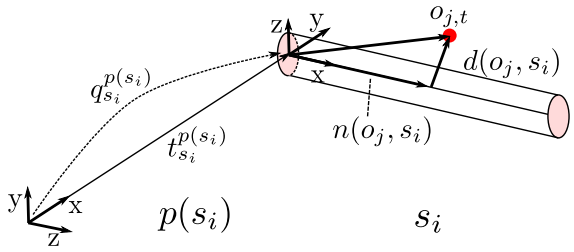


Figure 3.3: This figure illustrates two linked joints s_i and its parent segment $p(s_i)$, their local coordinate systems and the transformation between the two segments. Furthermore, it illustrates an observed point and its normalized projection and distance to the segment s_i, which we use to compute the likelihood for the assignment of the observed marker to a segment.

3.2.1 Problem Definition

At each discrete time step t, we assume to receive a frame of data F_t that is a set of unlabeled 3D points $\{\mathbf{o}_{i,t}\}$. Each point $\mathbf{o}_{i,t} \in F_t$ is the observed 3D position of a marker attached to one of the limbs of a person. The goal of our method is to estimate the *skeleton configuration* C for each frame, i.e., to estimate the global pose and the joint angles of the underlying skeleton. In particular, we use a human skeleton model that consists of 14 connected segments $s_i \in S$ having overall 45 degrees of freedom and that is based on medical data [27]. However, the 3D observations are subject to measurement noise and the markers are usually placed on the skin. During the movements of the body this induces non-deterministic variations of 3D position of the markers also with respect to the segments of the skeleton. Therefore we consider the full posterior probability $p(C \mid F)$ and estimate the maximum likelihood skeleton configuration of the skeleton configuration given the observations. This entails to *label* the observed points, i.e., to find the association function $\chi_t : F_t \rightarrow M$ that assigns each point to a marker label $m_j \in M$, which is one of the key challenges of the method presented in this chapter.

3.2.2 Skeleton Model

In this work, we use a hierarchical skeleton model, where the pose of each segment s_i is defined by its local pose with respect to the coordinate system of its parent segment $p(s_i)$. In particular, we denote the quaternion that describes the orientation of segment s_i in its parent's coordinate system at time step t as $\mathbf{q}_{s_i}^{p(s_i)}(t)$ and the position as $\mathbf{t}_{s_i}^{p(s_i)}(t)$, respectively. Fig. 3.3 shows an example of a segment s_i and its parent segment $p(s_i)$. Consequently, the global orientation $\mathbf{q}_{s_i}^{g}(t)$ of segment s_i at time step t is recursively given by

$$\mathbf{q}_{s_i}^{g}(t) = \mathbf{q}_{p(s_i)}^{g}(t) \odot \mathbf{q}_{s_i}^{p(s_i)}(t) \ , \tag{3.1}$$

where $\odot$ is the quaternion product. Similarly, the global position $\mathbf{t}_{s_i}^{g}(t)$ of segment s_i is recursively given by

$$\mathbf{t}_{s_i}^{g}(t) = \mathbf{t}_{p(s_i)}^{g}(t) + \mathbf{q}_{p(s_i)}^{g}(t) \odot \mathbf{t}_{s_i}^{p(s_i)}(t). \tag{3.2}$$

The position $\mathbf{t}_{s_0}^{p(s_i)}(t) = \mathbf{t}_{s_0}^{g}(t)$ and orientation $\mathbf{q}_{s_0}^{p(s_i)}(t) = \mathbf{q}_{s_0}^{g}(t)$ of the root segment, which is the hip segment in our experiments, describes the global pose of the skeleton. The positions $\mathbf{t}_{s_i}^{p(s_i)}(t)$ of the remaining segments correspond to their parent's lengths and are therefore static and specified by the skeleton model. As a result, the *skeleton configuration* C consists of 14 orientations that correspond to the joint angles and the global position of the skeleton. This results in overall $14 \cdot 3 + 3 = 45$ degrees of freedom of C.

We assume each marker m_j to be rigidly attached to one of the segments of the skeleton and introduce the function $\xi : M \to S$ that maps each marker to a segment. The local position of a marker m_j in the coordinate system of its corresponding segment is denoted as $\mathbf{p}_{m_j}^{\xi(m_j)}$. Thus, the position of marker m_j in the global coordinate system at time step t is given by

$$\mathbf{p}_{m_j}^{g}(t) = \mathbf{t}_{\xi(m_j)}^{g}(t) + \mathbf{q}_{\xi(m_j)}^{g}(t) \odot \mathbf{p}_{m_j}^{\xi(m_j)} \,. \tag{3.3}$$

3.3 Probabilistic Marker-Based Skeleton Tracking

In general skeleton tracking, one aims at estimating the skeleton configuration $C_{1:t}$ from time step 1 to t given the frame of unlabeled, noisy 3D observations of markers $F_{1:t} = \{\mathbf{o}_{i,1:t}\}$. In particular, we consider the likelihood $\mathcal{L}(C_{1:t} \mid F_{1:t})$ of the skeleton configuration given the observations. However, the association of markers to segments $\xi_{1:t}$ and the labeling of the observations $\chi_{1:t}$ are latent variables in our observation model. Thus, to compute the likelihood

$$\begin{aligned} \mathcal{L}(C_{1:t} \mid F_{1:t}) &= p(F_{1:t} \mid C_{1:t}) \\ &= \sum_{\chi_{1:t}, \xi_{1:t}} p(F_{1:t}, \xi_{1:t}, \chi_{1:t} \mid C_{1:t}) \end{aligned} \tag{3.4}$$

we marginalize over these latent variables. Since the maximization of Eq. (3.4) is infeasible in practice, we rely on the popular EM algorithm, which iteratively determines the maximum likelihood skeleton configuration

$$C_{1:t}^{*} = \underset{C_{1:t}}{\operatorname{argmax}} \sum_{\chi_{1:t}, \xi_{1:t}} p(F_{1:t}, \xi_{1:t}, \chi_{1:t} \mid C_{1:t}) \,. \tag{3.5}$$

In particular, the EM algorithm consists of two steps. First, the E-step computes the expectation value of the latent variables $\mathbb{E}(\chi_{1:t}, \xi_{1:t} \mid C_{1:t}^{(k)}, F_{1:t})$ given the configuration $C_{1:t}^{(k)}$. Second, the M-step computes the configuration $C_{1:t}^{(k+1)}$ that maximizes the likelihood under these expectations.

However, evaluating all possible marker and segment associations over all frames is not feasible in practice. Therefore, we propose the following approximations:

1. We assume the association ξ to be static and only compute it once in the initialization phase.

2. We consider online skeleton tracking and therefore recursively compute the likelihood. Hence, we assume the latent variables of the preceding frame to be known so that the recursive E-step reduces to computing the expectation value $\mathbb{E}(\chi_t \mid C_{1:t}, F_{1:t}, \chi_{1:t-1})$ and the recursive M-step computes $C_t^{(k+1)}$.

3. We apply the Hungarian method for optimal assignment to efficiently approximate the expectation value of the latent variables in the E-step. This maximum likelihood assignment technique is also known as hard EM in the literature.

Fig. 3.2 illustrates the resulting algorithm for online skeleton tracking. In the initialization phase, we determine the initial skeleton configuration as well as the association function ξ. For every incoming frame at time F_t, we first label the observations based on the labeling of the preceding frame and optimize the skeleton configuration to obtain an initial guess for the EM skeleton estimation. We then iteratively find the most likely labeling χ_t given the current skeleton configuration (E-step) and optimize the configuration C_t (M-step). In the following, we describe the individual steps of our algorithm for automatic skeleton tracking in detail.

3.4 T-Pose Initialization

In this section, we present the initialization step of our method to estimate the skeleton configuration of a person given the 3D position of observed markers that are attached to the body, as illustrated in Fig. 3.2.

The goal of the initialization step is to estimate the initial skeleton configuration C_0, to determine the initial labeling χ_0, i.e., to assign a marker label to each observed point in the first frame of observations F_0, and to determine ξ_0, i.e., to assign each marker label to one of the skeleton segments. We assume that the person initially stands in the T-pose, in which the person stands on the floor, stretches both arms and legs, and holds the arms approximately horizontally sidewards as shown in Fig. 3.1. Furthermore, we assume that there is one marker placed on top of the person's head. Fig. 3.2 (middle) illustrates the initialization process.

First, we obtain the height of the person by extracting the uppermost marker observation. Given the height, we scale the skeleton model, i.e., the local position vectors $\mathbf{t}_{s_i}^{p(s_i)}$ of all segments $s_i \in S \setminus \{s_0\}$, to match the size of the observed person. By considering the anatomy of humans, we identify the subset of points that belong to the legs and to the arms and compute their first principal axes, as illustrated in Fig. 3.2 (right). We align the skeleton model to these axes through least-squares optimization to obtain the initial skeleton configuration C_0. Our initialization method is robust against deviations from the perfect T-pose due to the optimization-based alignment.

We initialize the bijective association function $\chi_0 : F_0 \to M$ by assigning each observed point $\mathbf{o}_{i,0}$ to the marker m_i. For each observed point $\mathbf{o}_j$, we compute the projection $n(\mathbf{o}_j, s_i)$ and distance $d(\mathbf{o}_j, s_i)$ to the corresponding segment $s_i = \chi_0(\mathbf{o}_j)$, normalized by the segment's length and radius, respectively. Thus, as illustrated in Fig. 3.3, points with the value of n and d in the interval $[0, 1]$ form a tube that has approximately

the size of the corresponding body part. We define the likelihood of a point $\mathbf{o}_i$ to belong to a segment s_i as

$$\mathcal{L}(s_i \mid \mathbf{o}_i) = f(n(\mathbf{o}_j, s_i))\, f(d(\mathbf{o}_j, s_i)) \;, \tag{3.6}$$

where f is a function that has high values in the interval $[0, 1]$ and converges to 0 outside this interval. As a result, we assign high likelihood to points located inside the tube that represents the body part corresponding to segment s_i. In particular we choose

$$f(x) = \phi((x + \theta_1)\theta_2)\, \phi((1 - x + \theta_1)\theta_2) \tag{3.7}$$

with

$$\phi(x) = \frac{0.5x}{\sqrt{1 + x^2}} + 0.5. \tag{3.8}$$

The parameters θ_1 and θ_2 determine the convergence properties of f. Then, we assign each point to the segment it is most likely attached to:

$$\xi(m_j) = \underset{s_j \in S}{\operatorname{argmax}}\, \mathcal{L}\left(s_j \mid \chi_0^{-1}(m_j)\right) \;. \tag{3.9}$$

Finally, we compute the initial estimate of the relative position $\mathbf{p}_{m_j}^{\xi(m_j)}$ of all markers with respect to their corresponding segments.

To resolve the ambiguity between the two possible headings of the person in T-pose, we maintain both hypotheses at first and dismiss one hypothesis as soon as it gets unlikely due to the joint limits of the skeleton (Sec. 3.5.3).

3.5 Joint Labeling and Skeleton Estimation

Skeleton tracking aims at maximizing the likelihood of the skeleton configuration C_t given the unlabeled, noisy observations of markers in an online fashion. As illustrated in Fig. 3.2, we initialize the labeling based on the preceding frame in a nearest neighbor association. Given the initial skeleton configuration, we simultaneously perform the labeling of marker observations and the estimation of the skeleton configuration through an EM procedure, which alternately updates the labeling based on the skeleton configuration and estimates the skeleton configuration given the labeling.

3.5.1 Labeling Based on the Preceding Frame

We initialize the labeling of every incoming frame of observations based on the labeling and the skeleton configuration of the preceding frame. Due to the high frame rate of motion capture systems usually most of the markers only move a short distance between two consecutive frames. In our approach, we initially label these markers through nearest neighbor association given the preceding frame.

In particular, we optimally associate the labeled observations of frame F_{t-1} to the unlabeled observations of frame F_t given the spacial distance as a cost function. Additionally, we define the threshold $\theta_{\text{max}}^{\text{PF}}$ as an upper limit for a valid association to avoid

a wrong labeling in cases where one marker disappears and another marker appears at the same time. We determine the optimal one-to-one assignment using the Hungarian method [89]. This method assigns each observation of F_t to one observation of F_{t-1} in a way that minimizes the sum of the distances. The resulting labeling function is $\chi_t(\mathbf{o}_{i,t}) = \chi_{t-1}(\mathbf{o}_{j,t-1})$ for each pair $(\mathbf{o}_{i,t}, \mathbf{o}_{j,t-1})$ that has been assigned by the Hungarian method. Note, that we can adapt the thresholds $\theta^{\text{PF}}_{\text{max}}$ online for each marker individually based on statistics over the previous frames to account for a changing velocity of the markers.

3.5.2 Labeling Based on the Skeleton Configuration

If a marker was occluded during some frames and reappears, or if it moves more than the threshold $\theta^{\text{PF}}_{\text{max}}$, we cannot label it based on the preceding frame. However, the current skeleton configuration C_t provides a prediction of the position of each marker, given by the global marker positions $\mathbf{p}^g_{m_j}(t)$. According to Sec. 3.5.1, we use the Hungarian method [89] to assign all remaining observations to markers that have not already been labeled based on the preceding frame. Here, we assign only observed points to markers in a radius of $\theta^{\text{S}}_{\text{max}}$ to be more robust against outliers.

3.5.3 Skeleton Estimation

Given the full or partial labeling χ_t of the marker observations of the current frame F_t and the relative position $\mathbf{p}^{\xi(m_j)}_{m_j}$ of each marker $m_j \in M$ with respect to its corresponding segment, we estimate the skeleton configuration C_t. In particular, we estimate the maximum likelihood skeleton configuration of $p(C_t \mid F_t)$ taking into account the uncertainty of observations and the joint limits of the skeleton. Due to the optical measurement process and skin effects, we assume Gaussian noise on the 3D observations of markers with respect to the segments of the skeleton. Thus the M-step translates to optimizing the skeleton configuration with respect to the mean squared error of the marker distances and the joint limits of the skeleton.

For each observation $\mathbf{o}_{i,t}$, we compute the reprojection, i.e., the estimated global position given C_t, of the marker assigned to this observation $\mathbf{p}^g_{\chi_t(\mathbf{o}_{i,t})}(t)$ by evaluating Eq. (3.3). We aim to find the configuration of the skeleton that minimizes the quadratic reprojection error, which is the quadratic distance between this reprojection and the actual observation. To improve the performance of the skeleton pose estimation, we include penalties for joint configurations that are outside of certain limits defined by considering the natural configurations of the skeleton, forcing for example the knee to move in one degree of freedom. Thus, the overall optimization function at time step t

$$f(C_t) = \sum_{i \in I_t} \left\| \mathbf{p}^g_{\chi_t(\mathbf{o}_{i,t})}(t) - \mathbf{o}_{i,t} \right\|^2 + l(C_t)\,, \tag{3.10}$$

sums over the set I_t of labeled marker observations and takes into account the joint limit cost function $l(C_t)$.

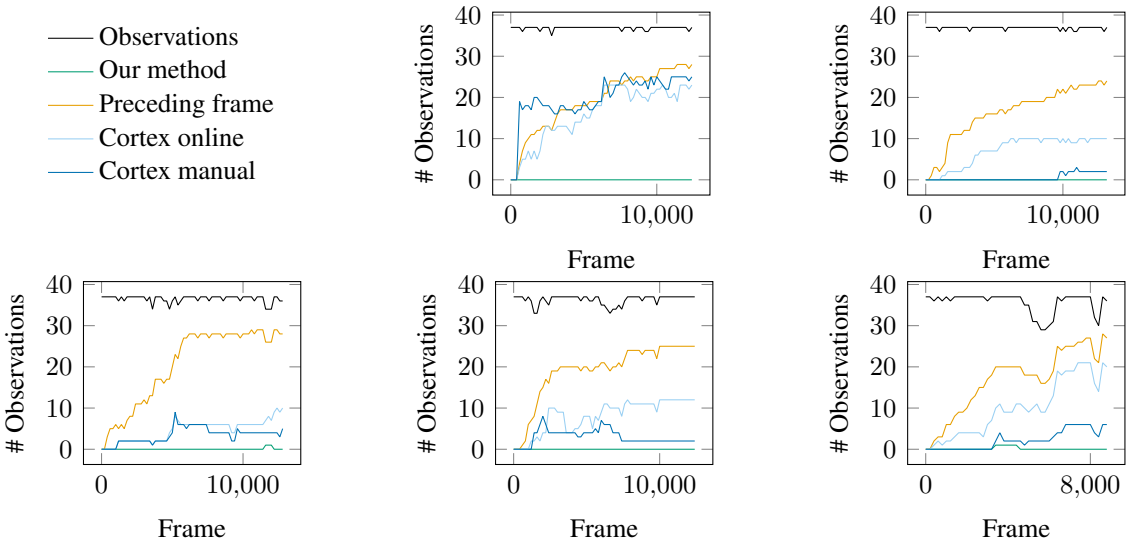


Figure 3.4: The number of observed markers (black) and the number of wrongly or not labeled markers (green) of our proposed method compared to the commercially available software Cortex and to a baseline method.

We determine the skeleton configuration C_t using the optimization framework g^{2}o [90], which provides modular and flexible gradient descent optimization. Particularly, we formulate the skeleton estimation problem as a graph, which contains the skeleton configuration in a variable node, the relative marker positions in fixed nodes, and the components of the error function $f(C_t)$ as unary and binary edges. The gradient of $f(C_t)$ with respect to the skeleton parameterization C_t is computed by means of numerical differentiation. This optimization procedure usually converges after a few iterations, even during fast movements and when initialized with the skeleton configuration C_{t-1} of the preceding frame.

3.6 Experimental Results

We evaluated the presented algorithm on various motion capture recordings of different test subjects and marker sets. Each data set was recorded with a Motion Analysis motion capture system with ten Raptor-E cameras at 100 Hz frame rate. Our method was able to process data online at 100 Hz on an Intel® Core™ i7-2600K CPU with 3.40 GHz.

3.6.1 Initial Association of Markers to Limbs

In a first set of experiments, we evaluated the initialization method (see Sec. 3.4) that associates observed markers to the skeleton segments. Therefore, we compared the association of our method to manually labeled ground truth data. For markers that were located in the joint area between two segments, for example the elbow or the knee, we allowed the association to both segments. As parameter for the association of markers

to limbs in Eq. (3.7) we chose $\theta_1 = 5$ and $\theta_2 = 0.2$. In 24 experiments, our approach associated 99.1% of overall 969 markers to the correct segment.

3.6.2 Performance of the Marker Labeling

A key challenge during the estimation of the skeleton pose of a human is to associate the observed points to the correct marker labels in each frame. To evaluate and compare the labeling performance of our method, we manually annotated five publicly available datasets of overall $10\,\mathrm{min}$ motion capture data, including walking, sitting on a chair, stretching arms and legs, and jumping. Fig. 3.4 shows for each dataset and for our method as well as the following three alternatives the number of observed markers (*Observations*) and the number of markers that were either not or falsely labeled:

- *Preceding frame:* Our method with labeling based on the preceding frame only.
- *Cortex online:* Motion Analysis Cortex without manual post processing.
- *Cortex manual:* Motion Analysis Cortex where each dataset was manually post-processed for $1\,\mathrm{h}$.

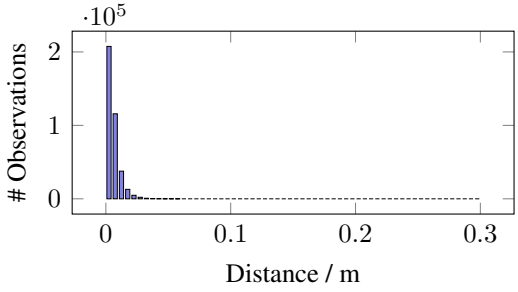
Overall, our approach was able to correctly label 99.6% of all markers, whereas *Cortex online* correctly labeled only 79.8% of the data without manual post processing. After one hour of manual post processing for each dataset, *Cortex manual* correctly labeled 93.6% of the data. Obviously, labeling of markers based on the preceding frame only is not sufficient, as occluded markers are not correctly labeled when they reappear. Thus, our approach provides the best performance in fully automatic marker labeling for skeleton tracking.

3.6.3 Data Association Threshold Experiments

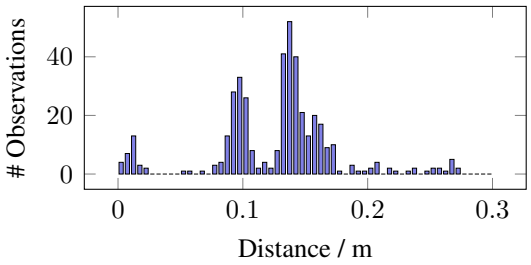
Labeling based on the preceding frame is sensitive to the parameter $\theta^{\mathrm{PF}}_{\mathrm{max}}$, which determines the maximum distance for optimal data association in the Hungarian method. We can adjust this parameter for each marker online using statistics over the previous frames. Thus, we can adapt the threshold to changing velocities of individual markers. To initialize the thresholds, we evaluated the movements of markers in a typical capture of natural movements. Fig. 3.5 shows a histogram of correctly (left) and incorrectly (right) associated markers based on the preceding frame with $\theta^{\mathrm{PF}}_{\mathrm{max}}$ set to ∞. Note, that the scales of the two figures deviate largely, which indicates that most of the markers are labeled correctly. All correctly associated markers moved less than $5\,\mathrm{cm}$ compared to the preceding frame. The incorrectly labeled observations are mostly due to occlusions. Therefore, in all of our experiments we chose the initial value $\theta^{\mathrm{PF}}_{\mathrm{max}} = 5\,\mathrm{cm}$.

3.6.4 Ambiguity of the Initial Pose

As described above, our method initially tracks both potential headings of the person until the constraints in the joints and especially in the knee joints disambiguate the situation.



(a) correct associations



(a) wrong associations

Figure 3.5: Histogram of the movement of markers between two consecutive frames for correctly associated markers (a) and incorrectly associated markers (b) when assigned by the Hungarian method in a typical capture. Note the substantially distinct scales of the y-axes, which shows that the bulk of the markers are correctly associated based on the preceding frame.

To evaluate the robustness of our approach to determine the heading we evaluated its performance on 24 datasets with two different humans and six different marker setups. In all of the 24 datasets, our method chose the correct heading after $1.78\,\mathrm{s}$ in average, with a standard deviation of $0.93\,\mathrm{s}$.

3.7 Related Work

The problem of inferring the structure and motion of objects based on the observation of markers attached to these objects has been studied intensively in the past. A common approach is to detect connected rigid bodies and to estimate the structure of the underlying skeleton in terms of the joints connecting these segments. Ringer and Lasenby [128] cluster observed 3D markers using the variance of the pairwise distance over all frames. Given the association of all markers to the segments, they determine their offsets, similar to our work. Similarly, de Aguiar et al. [30] and Kirk et al. [73] propose a method to cluster the observed markers to rigid bodies and estimate the center of rotation between the resulting limb segments.

Several authors estimate the joint rotation centers and the joint angles of a human skeleton from clusters of labeled 2D and 3D marker positions while taking into account skin movement artifacts [4, 20, 75]. Cerveri et al. [20] and Klous and Klous [75] even infer the skeleton structure and its configuration by calculating statistics over all frames, which allows only offline processing of the captured data. In contrast, the method proposed by Aristidou and Lasenby [4] as well as our approach are able to track the skeleton online frame by frame without observing all frames first.

In many applications, for example when tracking humans, the underlying skeleton structure is known in advance. In particular, Contini [27] describes a standard human skeleton model, which we also use in the work described here. Zordan and Van Der Horst [163] propose a force based approach to fit such a human skeleton model to observed markers. Similar to our work, Xiang et al. [152] estimate the configuration of a human skeleton and utilize the knowledge about joint limits. One can either use medical studies [54]

to determine these joint limits or learn them for individual humans by observing their motions [56].

All the approaches described above assume known marker labels to estimate the underlying skeleton model, or they solve the marker labeling independently of the skeleton estimation process. In contrast to these methods, we propose a marker labeling technique that exploits the knowledge about the current skeleton configuration. Similarly, Herda et al. [55] present a method to estimate the labeling of the markers in each frame and Yu et al. [156] aim to track multiple targets by fitting rigid bodies into the observed point cloud. Both approaches are able to deal with occluded markers but they require to manually label the markers in the first frame. In contrast, our approach performs labeling and skeleton pose estimation automatically without any user assistance.

In addition to the academic approaches described above, there are commercial solutions available for tracking articulated objects. We compare our work to *Cortex* developed by *Motion Analysis* [106] that provides online labeling of objects after a prior manual initialization and model training procedure. However, as opposed to our method, Cortex entirely separates the labeling from the skeleton tracking and therefore requires tedious manual post-processing to achieve results that are comparable to those obtained with our system.

In the context of mobile robots, Ziegler et al. [162] match the information of a laser that is mounted on the robot with a full body posture estimation of a human. This allows the robot to robustly track and follow the pedestrian for longer periods of time. Yamane and Hodgins [153] imitate the posture of a human tracked in a motion capture studio with a humanoid robot, by simultaneously satisfy balance conditions of the robot. Mitsunaga et al. [103] propose to use the information of a marker-based motion capture system to adapt the behavior during interaction. Our system provides online skeleton tracking, which makes it applicable to interaction scenarios with mobile robots.

3.8 Conclusions

In this chapter we presented a fully automatic method to estimate the underlying skeleton configuration of a human based on the position of markers perceived in a motion capture system and freely attached to the human. Our method uses the popular EM algorithm to compute the most likely skeleton configuration by estimating the unknown association of observations to marker labels in each frame. In contrast to existing approaches, which typically require tedious manual post processing, our method solves the estimation of the marker labeling without any user intervention. In an extensive set of experiments we demonstrate that our method outperforms even a commercially available and state-of-the-art method for skeleton pose estimation. The methods towards socially compliant mobile robot navigation we propose in the next chapters of this thesis rely on pedestrian tracking techniques. In our experiments, we use the position and the velocity of the pedestrians. However, considering also more complex body language using the method proposed in this chapter could further improve the interaction between the pedestrian and the robot.

Chapter 4

Online Generation of Homotopically Distinct Navigation Paths

Obstacles partition the space of navigation paths for mobile robots into homotopy classes, which are topologically distinct ways to avoid the obstacles. In motion planning there are many applications in which it is beneficial to consider and identify these different classes. For example, for tethered robots, or surveillance UAVs it may be important in which order and direction they circumvent objects in the environment. Furthermore, we can use homotopically distinct trajectories as an informed initialization for an optimization-based navigation system. Since gradient based optimization approaches are typically unable to 'jump' over obstacles, the homotopy class of the trajectories does not change during optimization. Initializing optimization in different homotopy classes therefore increases the chance of finding the globally optimal path. In later chapters, we will use the techniques presented here for socially compliant mobile robot navigation. When using homotopically distinct paths online during mobile robot navigation, it is crucial to compute these paths fast and efficiently. To this end we propose a method to compute an abstract graph representation of the environment from a Voronoi diagram in which we efficiently search for homotopically distinct navigation paths. In addition, we present an efficient method to identify distinct classes of trajectories. An experimental evaluation suggests that our approach outperforms state-of-the-art methods for computing paths in different homotopy classes.

In topology, homotopy classes are defined as a set of paths that can be continuously transformed into each other. In terms of motion planning, this means that we can incrementally transform two homotopic trajectories into each other without moving over obstacles. As an illustration, Fig. 4.1 shows a set of trajectories in three different homotopy classes in an environment with two obstacles.

In the context of motion planning, there is a wide range of application in which it is important to reason about homotopy classes of trajectories. For winding constrained prob-

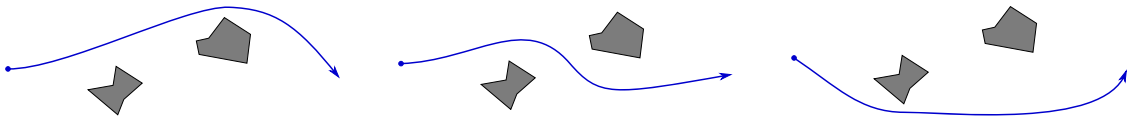


Figure 4.1: Example of three homotopically distinct composite trajectories in the same situation. The agent depicted in blue passes the solid obstacles on different sides. With fixed start- and end points these trajectories cannot be continuously transformed into each other without colliding with obstacles.

lems it is important in which way the robot circumvents the obstacles in the environment. For example, tethered robots are constrained in their homotopy class due to the attached cable [60], and surveillance UAVs may have the task to circumfly the monitored objects a specific number of times [146].

Another application of trajectories in different homotopy classes is an informed initialization for optimization-based navigation systems. Approaches to mobile robot navigation often employ a cost function or constraints to encode desirable trajectory properties. If the employed cost function accounts for higher order properties such as velocities and accelerations, it is in general difficult to find the globally optimal solution due to the high dimensionality of the search space. One way to address this problem is an initialization computed in a lower-dimensional space such as a 2D grid. Approaches following this principle exist in the spectrum between reactively following a guidance path [41] and initializing trajectory optimization regimes with a lower-dimensional path [136]. Typically, the navigation cost function includes penalties for increasing closeness to obstacles. As a result, gradient-based optimization approaches can typically only find solutions that are in the same homotopy class as their initialization since they are unable to 'jump' over obstacles [93]. Unfortunately, for most applications there are no methods available that provide an initialization leading to the global minimum. To solve this problem, many approaches resort to sampling techniques to increase the chance of finding the global minimum [66]. However, existing methods generate a substantial number of initial paths from the same homotopy class, which increases their run-time. In contrast, initializing with trajectories of different homotopy classes leads to distinct local minima after optimization. In this way, we efficiently solve multiple solutions in parallel and select the best result for navigation.

In this chapter, we propose an online method to explicitly compute a set of homotopically different paths to the goal, i.e., a set of paths that cannot be smoothly transformed into each other without colliding with obstacles. We first compute a Voronoi diagram in the environment using an efficient, incrementally updatable method [92]. From this, we compute an abstract graph representation that captures the connectivity of the free space of the environment. In this graph, we efficiently search for the k best paths, or for paths that satisfy certain winding constraints. Subsequently, we convert the 2D paths into trajectories and optimize them with respect to a navigation cost function that captures higher order properties.

In addition, we describe a method to efficiently compute the winding numbers of a given trajectory. The vector of winding numbers uniquely identifies the homology class of a trajectory. Homology is a topological concept related to homotopy, which serves

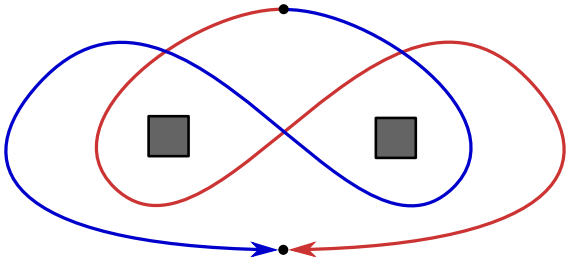


Figure 4.2: This figure shows an example of two paths that are homotopically distinct, but that are in the same homology class. The trajectories cannot be transformed into each other without moving over either of the two gray obstacles.

as a valid approximation for homotopy for navigation tasks. Identifying the class of a given trajectory allows us to re-use optimized trajectories during navigation, instead of discarding the remaining optimized trajectories after selecting the best one. This allows robots to instantly switch to readily available fallback trajectories should the previously selected solution become more costly or even infeasible due to changes in the environment.

In later chapters of this thesis we utilize the techniques described in this section for socially compliant mobile robot navigation. Specifically, in Chap. 5 we present a model of the interactive navigation behavior of multiple agents. This model also takes into account discrete navigation decisions that correspond to the homotopy classes of the environment. Furthermore, the techniques described in this chapter enable human-in-the-loop or shared autonomy applications that provide qualitatively different trajectories to the user, which we discuss in Chap. 7.

In the remainder of this section we first introduce the topological concepts of homotopy and homology. We then describe our method to efficiently compute a set of homotopically distinct navigation paths and a method to compute the winding numbers of trajectories. In an experimental section, we show that our method outperforms state-of-the-art methods for computing homotopically distinct navigation paths in run-time, which makes our method applicable to online navigation tasks.

4.1 Homotopy vs. Homology

There are two related topological concepts for classifying paths, homotopy and homology. Homotopy captures whether two paths can be continuously deformed into each other. Applied to paths in a 2D environment with obstacles, two paths that are homotopic can be transformed into each other in infinitesimal steps without contact to the obstacles. On the other hand, the concept of homology captures how often a given curve winds around a certain point, or an obstacle in our case. Informally, for homotopy the order in which the curves winds around points matter, in contrast to homology. Thus, homotopy is a stronger concept in the sense that any two homotopic curves are homologous, but not necessarily vice versa.

Our motivation is to find homotopic distinct paths that cannot be continuously deformed into each other. However, classifying the homotopy class of a path turns out to be much

harder than to classify homology. Therefore, in this work we propose to use an identifier that captures homology, which serves as a valid approximation to homotopy in the most practical situations.

4.1.1 Homotopy

We define paths in a 2D environment as functions $\mathbf{p} : [0,1] \to \mathbb{R}^2 \setminus \{\mathbf{o}_0, \ldots, \mathbf{o}_m\}$, where the points $\mathbf{o}_i$ correspond to obstacles in the environment. Two paths with start point $\mathbf{s}$ and end point $\mathbf{e}$ are defined as homotopic if they can be continuously transformed into each other. More formally, the paths $\mathbf{p}_1$ and $\mathbf{p}_2$ are homotopic if and only if there exists a continuous function $H : [0,1] \times [0,1] \mapsto \mathbb{R}^2 \setminus \{\mathbf{o}_0, \ldots, \mathbf{o}_m\}$ for which the following conditions hold

$$H(0,\gamma) = \mathbf{s} \quad \forall \gamma \in [0,1] \tag{4.1}$$
$$H(1,\gamma) = \mathbf{e} \quad \forall \gamma \in [0,1] \tag{4.2}$$
$$H(t,0) = \mathbf{p}_1(t) \quad \forall t \in [0,1] \tag{4.3}$$
$$H(t,1) = \mathbf{p}_2(t) \quad \forall t \in [0,1]. \tag{4.4}$$

The first two conditions fix the start point and the end point. The last two conditions state that as we change the second argument of H from 0 to 1, H transforms the path $\mathbf{p}_1$ smoothly into the path $\mathbf{p}_2$.

Fig. 4.2 illustrates two paths that are not homotopic. If we think of the red path as a rubber band fixed at the start and the end, it is not possible to reshape it to the blue path without moving it over one of the obstacles, which corresponds to the definition of homotopy.

For navigation tasks, we are interested in homotopically distinct navigation paths. However, it turns out to be difficult to efficiently determine the homotopy class of a given trajectory. Therefore, we rely on identifying homology classes, which is a related concept that we introduce in the following.

4.1.2 Winding Numbers and Homology

A circle sector $\circlearrowleft$ is the area of a disk with center at $\mathbf{o}$ that is bounded by two radii and an arc. We can define a function $\alpha_{\mathbf{o}} : \circlearrowleft \to \mathbb{R}$ that maps any point $\mathbf{x}$ in the sector to the angle between $(1,0)^T$ and $(\mathbf{x} - \mathbf{o})$, as illustrated in Fig. 4.3 (left).

For any continuous path $\mathbf{p} : [0,1] \to \mathbb{R}^2 \setminus \{\mathbf{o}\}$, we adopt the definition of winding numbers of Fulton [43]. They define the winding number by subdividing the interval $[0,1]$ into subintervals $[a_i, a_{i+1}]$ with $a_0 = 0 < a_1 < \cdots < a_n = 1$ in a way that each of the subsegments $\mathbf{p}\,|_{[a_i,a_{i+1}]} : [a_i, a_{i+1}] \to \mathbb{R}^2 \setminus \{\mathbf{o}\}$ are contained in a circle sector with vertex at $\mathbf{o}$. For each such segment, we compute the angle difference $\alpha_{\mathbf{o}}(\mathbf{p}(a_{i+1})) - \alpha_{\mathbf{o}}(\mathbf{p}(a_i))$, which is the change of the angle along the path. The winding number $\omega_{\mathbf{o}}(\mathbf{p})$ is the sum of

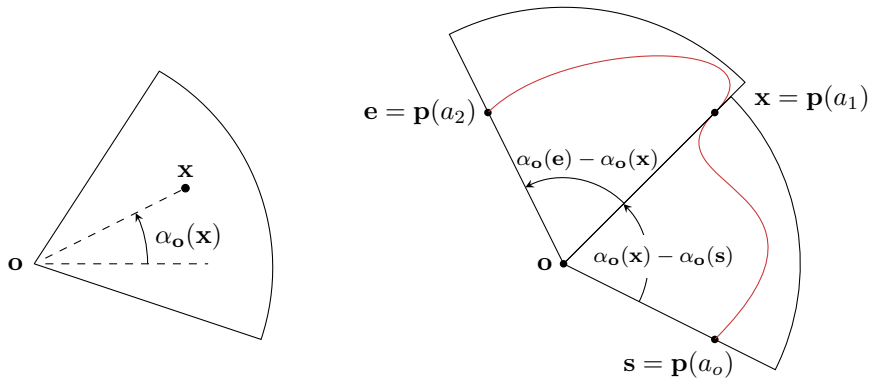


Figure 4.3: Computation of the winding number. Left: for each circle sector we can define an angle function that returns the angle to the x-axis for each point within the slice. Right: The winding number of a path is defined as the sum of angle differences of segments, where each of the segments is contained in a circle sector.

all these angle differences

$$\omega_{\mathbf{o}}(x) = \frac{1}{2\pi} \sum_{i=0}^{n-1} (\alpha_{\mathbf{o}}(\mathbf{p}(a_{i+1})) - \alpha_{\mathbf{o}}(\mathbf{p}(a_i))), \tag{4.5}$$

normalized such that a winding number of 1 corresponds to a full circle around $\mathbf{o}$. Fig. 4.3 (right) illustrates the subdivision of a path into two segments, and their corresponding angle differences. The idea behind the subdivision is to capture all windings around the obstacle. If we only computed the angle difference between the start point and the end point, we would possibly not capture the winding direction, and furthermore we would miss turns around an obstacle.

For an environment with more than one obstacle, we denote for a continuous path $\mathbf{p} : [0, 1] \to \mathbb{R}^2 \setminus \{\mathbf{o}_1, \ldots, \mathbf{o}_m\}$ the vector of winding numbers consisting of $\omega_{\mathbf{o}_i}(\mathbf{p})$ as $\boldsymbol{\omega}$. Two paths are homologous if and only if all winding numbers in $\boldsymbol{\omega}$ are equal [43].

Fig. 4.2 shows an example of two paths that are in the same homology class. Both paths wind counter-clockwise around the left obstacle, and clockwise around the right obstacle. As a result, the winding numbers of both paths are $\boldsymbol{\omega} = (\frac{1}{2}, -\frac{1}{2})$, which makes them homologic according to the definition above. This example shows that there are paths for which the concepts of homotopy and homology differ. However, in most practical situations identifying homology classes serves as a reasonable approximation to homotopy. In the following, we present a method that efficiently generates trajectories in different homotopy classes. Subsequently, we present our method for identifying the homology class of trajectories efficiently for online navigation tasks.

4.2 Generating Homotopically Distinct Trajectories

To compute paths in different homotopy classes, our method computes a graph representation of the Voronoi diagram from a grid map of the environment. For any given path, the

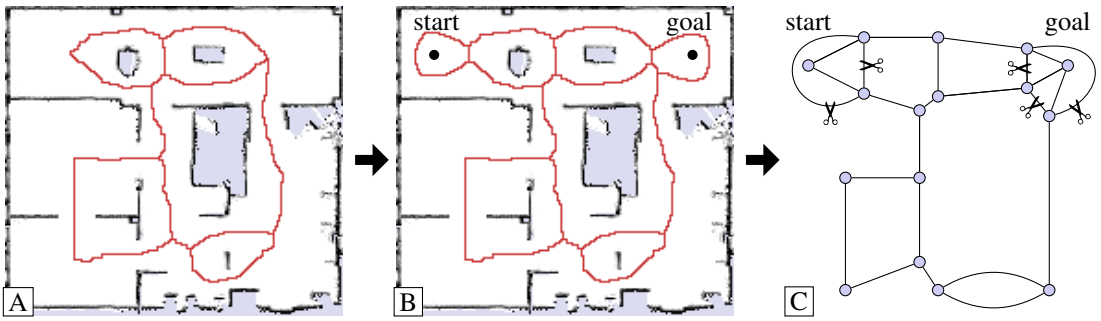


Figure 4.4: Overview of the proposed method to generate an abstract graph of the environment from the Voronoi Diagram. (A) Voronoi diagram (red) of an office environment. (B) We add Voronoi cells around the start and the goal location for a robust connection to the diagram. (C) Graph representation of the Voronoi diagram with connected vertices for the start and goal location.

Voronoi diagram contains a corresponding path in the same homotopy class [9]. Hence, our graph representation faithfully captures the information about the different homotopy classes, enabling efficient exploration of homotopic alternatives.

4.2.1 Discretized Voronoi Diagram

The generalized Voronoi diagram is defined as the set of points in free space to which the two closest obstacles have the same distance [25]. We compute a discretized form on an obstacle grid map bounded by occupied cells and represent it as a binary grid map VD in which a cell $(x, y) \in \mathbb{N}^2$ can either belong to the Voronoi diagram or not, i.e., $VD(x, y) \in \{\text{true}, \text{false}\}$. Fig. 4.4 (A) shows such a discretized Voronoi diagram over the obstacle map and depicts the cells for which $VD(x, y) = \text{true}$ in red.

Our approach builds upon the approach of Lau et al. [92] for an efficient, incrementally updatable computation of the Voronoi diagram. This method employs a wavefront algorithm to calculate distance transforms. It simultaneously starts wavefronts from each obstacle cell that propagate distances to the closest obstacle over the grid map. Wherever these wavefronts meet, it considers cells as candidates for the discretized Voronoi diagram as they are approximately equidistant to at least two obstacle cells. In general, a cell in a grid map will not exactly meet the Voronoi condition of being equidistant to two obstacle cells. Therefore, if two cells are at the boundary of two wavefronts it inserts the one that least violates the condition. It also takes thresholding measures to prevent Voronoi lines from appearing between neighboring obstacle cells and performs pattern matching-based post-processing to reduce discretization artifacts.

In this work, we add a further post-processing method that removes 'loose ends' of the Voronoi diagram that typically reach into concave structures like room corners. We detect such loose ends via pattern matching and process them until we reach a branching point on the Voronoi diagram. As a result, we obtain a discretized Voronoi diagram as shown in Fig. 4.4 (A): the Voronoi lines are sparse (no double lines) and four-connected, i.e., each cell with $VD(x, y) = \text{true}$ has up to four neighbors that are also contained in the Voronoi diagram, see also the magnified part in Fig. 4.5 (right).

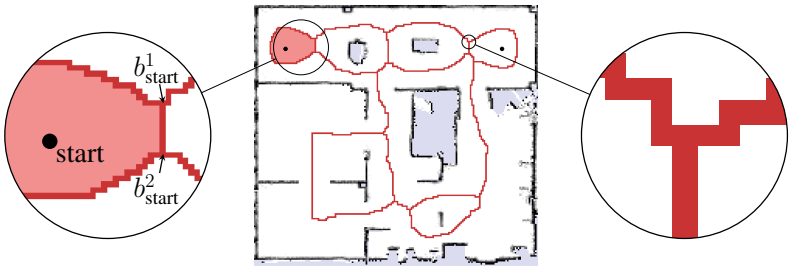


Figure 4.5: *Left:* area in the Voronoi diagram that contains the start point. Highlighted are the branching cells $b^1_{\text{start}}, b^2_{\text{start}}$ connected to this area. The start vertex of the resulting graph is connected to these vertices. *Right:* branching cells are connected to at least three neighbor cells.

We also employ the 'bubble technique' proposed by Lau et al. [92] for Voronoi diagrams. Here, dummy obstacles at the start and the goal provide a robust way to connect to the Voronoi diagram, see Fig. 4.4 (B) vs. (A).

4.2.2 Abstract Graph Representation of the Voronoi Diagram

From the discretized representation of the Voronoi diagram described in the previous section, we build a graph that captures the connectivity of the free space. This reduces the number of states compared to the original grid map representation of the Voronoi diagram. Fig. 4.4 shows the Voronoi diagram in an office environment (A,B) and the corresponding abstract graph (C).

In particular, vertices V in the graph $G = (V, E)$ represent the branching cells of the Voronoi diagram, also known as Voronoi vertices in the literature. In branching cells more than two lines meet. They thus correspond to locations that are equidistant to more than two obstacles. We can easily identify such branching cells since they have at least three neighbors in the Voronoi diagram. Fig. 4.5 (right) shows a close-up of such a branching point. The edges E capture the connectivity between the branching cells, i.e., each edge represents a connected line of cells that belong to the Voronoi diagram.

In addition, we need to connect the start and goal position to the graph. As described in the previous section, we insert dummy obstacles at the start and goal location such that they effectively become enclosed by 'bubbles' which are obstacle free by construction of the Voronoi diagram. Using a floodfill algorithm we mark all cells inside these areas and identify the attached branching cells $\{b^i_{\text{start}}\}$ and $\{b^i_{\text{goal}}\}$, as illustrated in Fig. 4.5 (left). Then, we connect the corresponding graph vertices to the start and the goal vertex, respectively. Finally, we remove the edges that connect the vertices corresponding to $\{b^i_{\text{start}}\}$ and $\{b^i_{\text{goal}}\}$, as illustrated by scissor symbols in Fig. 4.4 (C). This process removes edges that result from the dummy obstacles to ensure that each simple path in the graph corresponds to one unique homotopy class. Finally, we set the weights of the edges according to the length of the lines in the Voronoi diagram. The k best paths in the graph then correspond to the k shortest paths in the Voronoi diagram.

The abstract graph grows linearly with the number of obstacles in the environment [6]. This follows from viewing the Voronoi graph as a planar graph where the number of faces

Figure 4.6: Computing the k best paths from the Voronoi diagram. (D) Graph representation of the Voronoi diagram. Each path in this graph corresponds to a homotopically distinct path from start to goal. (E) Trajectories generated from the three shortest paths. (F) Optimized trajectories.

f corresponds to the number of obstacles. Since each vertex in the Voronoi graph has a minimum degree of three, the sum over the degrees of all vertices $\sum_{v \in V} \deg(v)$ is at least three times the number of vertices $|V|$. Furthermore, any undirected graph satisfies $\sum_{v \in V} \deg(v) = 2|E|$. Hence, we have $2|E| \geq 3|V|$. Combining this with the Euler relation $|V| - |E| + f = 2$ for planar graphs leads to $|E| \leq 3f$ and $|V| \leq 2f$, i.e., the number of edges and vertices is linear in the number of obstacles.

4.2.3 Finding the Shortest Simple Paths in a Graph

In the graph introduced in the previous section different paths always correspond to different homotopy classes in the environment. Therefore, searching for the k best homotopically different simple paths in this graph is equivalent to searching for the k best simple paths. The best known algorithm [69] for this problem has a runtime complexity in

$$O(k(|E| + |V| \log |V|)), \tag{4.6}$$

which follows from the complexity of

$$O(|E| + |V| \log |V|) \tag{4.7}$$

of Dijkstra's algorithm [36]. As the number of vertices and edges in our graph depends linearly on the number of obstacles m, it follows that our algorithms to extract the k best homotopically different simple paths has a complexity in

$$O(k(m \log m)). \tag{4.8}$$

To find these k best paths from the start vertex s to the goal vertex g, we adapt an extension of Dijkstra's algorithm [31]. In essence, this algorithm is the standard Dijkstra's algorithm that keeps expanding nodes until the goal node has been expanded not for the first but for the kth time, as outlined in Alg. 1. In the worst case, each node will thus be expanded k times.

The algorithm uses a priority queue B, which orders paths according to their costs. Furthermore, we count the number c_v of the currently expanded paths to each vertex v

Algorithm 1 Computing the k best paths in a weighted graph $G = (V, E)$ from start $s \in V$ to goal $g \in V$
Input: $G = (V, E)$, s, g, k
Output: Paths P

```
1:  B ← {⟨s⟩}                                  // priority queue containing paths ordered by their costs
2:  P ← ∅                                      // set of k best paths from s to g
3:  c_v ← 0 for all v ∈ V                      // counter for each vertex
4:  while B ≠ ∅ and |P| < k do
5:      ⟨v_1, ..., v_n⟩ ← B.pop()              // get path with lowest costs in B
6:      c_{v_n} ← c_{v_n} + 1
7:      if v_n = g then
8:          P ← P ∪ {⟨v_1, ..., v_n⟩}
9:      else if c_v < k then
10:         for all v' with (v_n, v') ∈ E do
11:             q ← ⟨v_1, ..., v_n, v'⟩        // append edge to path
12:             if q contains no loop then     // optional check
13:                 B.push(q)
14:             end if
15:         end for
16:     end if
17: end while
```

of the graph. While the number of paths to the goal vertex g is less than k (line 4), we extract the path that has the lowest costs from the priority queue B (line 5). If the last vertex v_n of this path is the goal vertex (line 7), we add it to the set of k best paths P (line 8). Otherwise, if the number of paths to v is less than k, we expand the path by all vertices that are adjacent to v and add it to B (lines 10–13). If we are only interested in simple paths, we check for this condition in line 12. After termination of the algorithm, P contains the k best (simple) paths from s to g.

4.2.4 From Discrete Paths to Trajectories

After computing the k best paths in the abstract graph, we need to generate the corresponding path in the original 2D environment. To this end, we store for each edge $v \in V$ in the graph G the corresponding cells of the 2D grid map. Then, we can simply concatenate the cells of all edges to retrieve the path in the grid map that corresponds to a path in the abstract graph, as illustrated in Fig. 4.6 (D,E).

Depending on the application, it may be important to not only compute a geometrical path, but a trajectory that also captures time information, i.e., a continuous mapping

$$\mathbf{p} : [0, T] \to \mathbb{R}^2 \tag{4.9}$$

that represents a 2D position at each time step t. Such a representation allows us for example to optimize trajectories with respect to higher order cost functions that represent

for example accelerations. In general, however, such a function has infinite dimensionality. We propose to use splines as a finite-dimensional representation of the trajectories. Splines are piecewise defined polynomials with a finite degree of freedom. Splines are widely used in motion planning since they satisfy certain smoothness conditions that are crucial for navigation tasks. Find a more detailed introduction to splines in Sec. 5.1.1.

To compute a spline from a list of cells in a 2D grid, which is the result of the method described above, we apply spline fitting techniques. In particular, a heuristic assigns the internal spline parameter for each point by connecting the points by straight lines. Subsequently, it is possible to compute the spline parameters that minimize the sum of the squared distances from the spline to each point efficiently in closed form. Fig. 4.6 (F) shows the trajectories that correspond to the three shortest paths in the Voronoi diagram after optimization.

4.3 Identifying Homology Classes

If the navigation task involves winding constraints, we need a method to identify the corresponding class of a trajectory. Furthermore, identifying the topological class of a trajectory is useful for re-using previously optimized trajectories, instead of repeatedly initializing an optimization with the path on the Voronoi graph. To maintain a set of optimized trajectories, we need to efficiently decide whether two given trajectories belong to the same class.

For efficiency reasons, we identify the homology class of trajectories instead of the homotopy class, which is a suitable substitute for the concept of homotopy in practically relevant scenarios [11]. According to the definition above, the vector of winding numbers

$$\boldsymbol{\omega}(\mathbf{p}) = \langle \omega_1(\mathbf{p}), \ldots, \omega_i(\mathbf{p}), \ldots, \omega_m(\mathbf{p}) \rangle \tag{4.10}$$

is invariant for all trajectories of one homology class. Here, the definition of winding numbers for trajectories with time information is analogous to the definition of winding numbers for geometrical paths. We compute the winding number for each obstacle in the environment as illustrated in Fig. 4.7.

For each obstacle, we need to identify one representative point within the obstacle. To determine this point for each obstacle, we use a flood fill algorithm to determine the cells in each enclosed region in the Voronoi diagram. Since each of these regions correspond to one connected obstacle, we take an arbitrary obstacle grid cell within the region as a representative point for this obstacle. Then, we evaluate the trajectory at discrete time steps and sum up the angle differences $\Delta\omega$ according to Eq. (4.5). Fig. 4.7 illustrates the computation of the winding number corresponding to one obstacle. For efficiency reasons, we adapt the step size dynamically as we walk along the trajectory to compute the winding numbers. If the angle difference between two steps falls below a threshold, we increase the step size, and decrease it if the difference is above a threshold. During navigation, we need to compute the angles each time the environment changes, since homotopy classes can emerge or fall together.

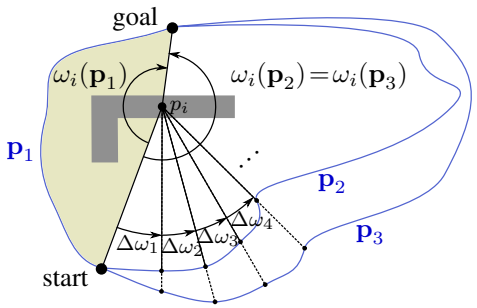


Figure 4.7: Computation of the winding numbers with respect to obstacles. The figure shows three paths $\mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{p}_3$ that bypass the obstacle i. The infinitesimal angles $\Delta\omega$ sum up to the winding number ω_i around the representative point p_i of the obstacle. The two paths on the right yield the same winding number $\omega_i(\mathbf{p}_2) = \omega_i(\mathbf{p}_3)$ in contrast to the path on the left.

4.4 Experiments

In this section we present experiments that compare our method to alternative approaches for computing paths in topologically distinct classes. We compared our approach with a method that computes paths by extending the state space with the H-signature, which is similar to the concept of winding numbers. A second experiment aims to show that our approach is suitable for computing the globally optimal trajectory in an environment by informed initialization more efficiently than related methods. Furthermore, we compared initialization in homotopically distinct paths with sampling-based initialization. Our experiments suggest that our method outperforms state-of-the-art methods in run-time, which makes it applicable to online navigation tasks.

4.4.1 Runtime Evaluation

In a first experiment we compared the efficiency of our method for online computation of paths in different homotopy classes with an existing method. Bhattacharya et al. [11] propose a method to compute paths of distinct homotopy classes. They perform an A* search on an arbitrary graph representation of the environment that they augment with the H-signature to capture topological information. In contrast to the abstract Voronoi graph we use in our approach, their graph may contain multiple paths to the goal within the same homotopy class, which considerably increases the complexity of the graph search.

To compare our method with the one presented by Bhattacharya et al. [11], we used the same environment as they used in their experiments, a $1,000 \times 1,000$ discretized environment with circular and rectangular obstacles, as illustrated in Fig. 4.9. For our method, we used an Intel® Core™2 Duo with $2.6\,\mathrm{GHz}$, which seems comparable to the computer used by Bhattacharya et al. [11]. As Fig. 4.8 shows, our method outperforms their approach by an order of magnitude in terms of run-time for the task of computing trajectories in the k best homotopy classes. Our method needs to initialize only once and can then answer queries for multiple start and goal positions incrementally. In our experiments, our algorithm found all 249 possible simple paths on the Voronoi diagram in $0.7\,\mathrm{s}$, including map initialization.

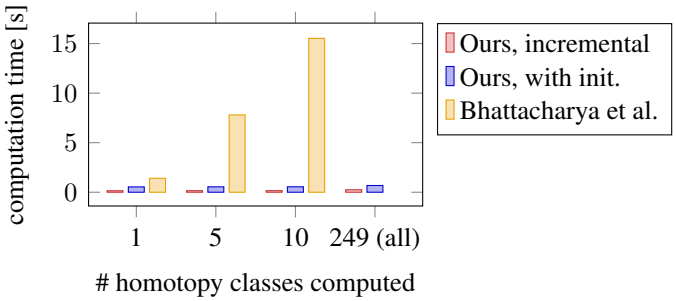


Figure 4.8: Comparison of our approach to the one presented by Bhattacharya et al. [11] for the environment shown in Fig. 4.9. As can bee seen, our approach is substantially faster. While the method proposed by Bhattacharya et al. [11] computes the 10 best paths in 15.5 s, our approach finds all 249 possible simple paths in 0.7 s, including map initialization.

The objectives of the compared algorithms are slightly different. Bhattacharya et al. [11] aim to find the k best paths with respect to the A* cost function that differ in their homotopy class, whereas our algorithm explores paths that lie on the Voronoi diagram. See Fig. 4.9 for the differences between the resulting paths in corresponding homotopy classes. However, the 20 best paths returned by both methods have an overlap of 17 homotopy classes for the scenario shown in Fig. 4.9. In any case, when using the paths for initializing optimization methods, we are not interested in the exact shape of the initial path but only in the corresponding homotopy class.

4.4.2 Optimizing Trajectories

In a second experiment we evaluated the applicability of our method to initialize trajectory optimization methods. The goal is to optimize trajectories of the form $\mathbf{p} : [0, T] \to \mathbb{R}^2$ with respect to a cost function that also captures dynamical constraints in all homotopy classes. Here, we compared our method with the approach of Kim et al. [71], who compute optimal trajectories within a particular homology class using a mixed-integer quadratic program. Similar to Bhattacharya et al. [11], they utilize the H-signature to identify homology classes.

In this experiment, we generated initial paths in all generated homotopy classes and converted these paths into a spline-based trajectory representation, similar to Kim et al. [71]. We also used six spline segments to represent the trajectory, as they did in their experiments. They used a representation of nine basis functions for each segment, whereas we used cubic splines in our experiments.

Kim et al. [71] propose a cost function consisting of the integrated acceleration along the trajectory

$$c(\mathbf{p}) = \int_0^T \| \frac{d^2\mathbf{p}(t)}{dt^2} \|^2 dt = \int_0^T \|\ddot{\mathbf{p}}(t)\|^2 dt. \tag{4.11}$$

They use constrained optimization, which prohibits intersection of the trajectories with a set of convex obstacles. In contrast, we use RPROP [126], a gradient based optimization method that performs unconstrained optimization. We therefore additionally introduce a

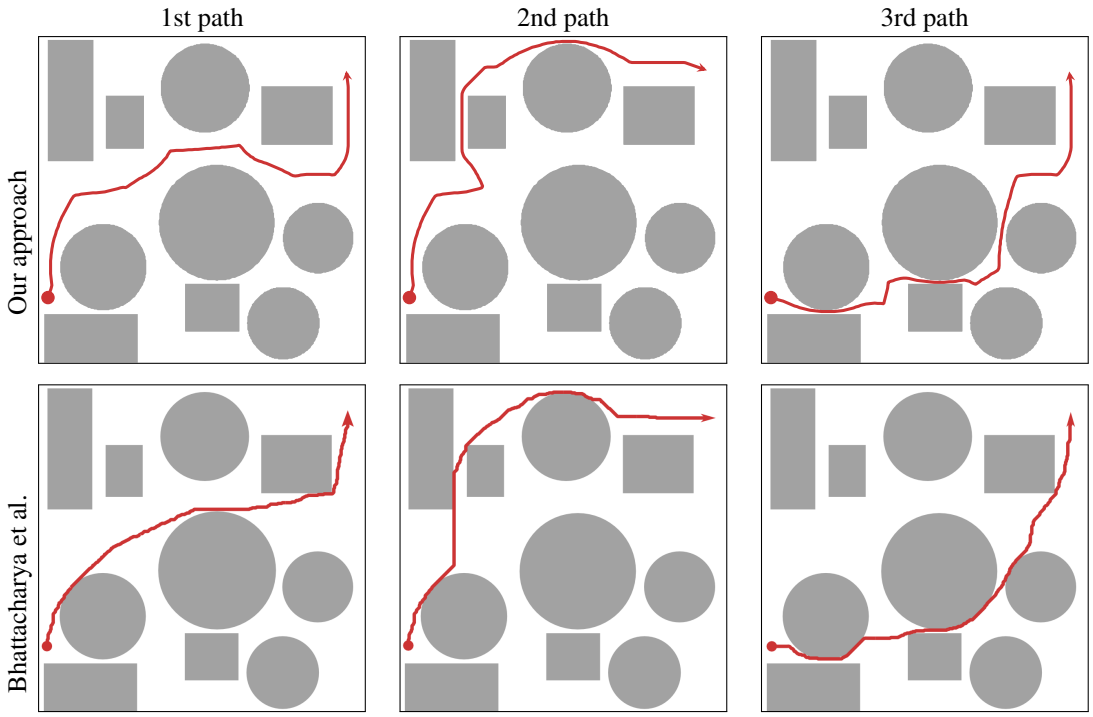


Figure 4.9: The first three homotopically different paths computed by our method (top) and the corresponding paths as computed by the method proposed by Bhattacharya et al. [11] (bottom). Our method computes paths using the Voronoi diagram, therefore the paths have higher distance to obstacles.

penalty for proximity to obstacles. The resulting cost function for our approach is thus

$$c(\mathbf{p}) = \theta \int_{t=0}^{T} \frac{1}{\| \mathbf{p}(t) - \mathbf{o}_{\text{closest}}(t) \|^2} \, dt + \int_{t=0}^{T} \| \ddot{\mathbf{p}}(t) \|^2 \, dt, \tag{4.12}$$

where $\mathbf{o}_{\text{closest}}(t)$ is the position of the closest obstacle to the trajectory at time t. We use a small value for θ, which allows the trajectories to closely approach the obstacles. However, with the distance to the closest obstacle in the denominator the limit of the cost function is infinity as the distance approaches zero, which prohibits intersection with obstacles. For both approaches the total travel time T is fixed.

Fig. 4.10 shows the optimal trajectories found by Kim et al. [71], the trajectories of our approach after convergence, as well as the trajectories of our approach when stopping the optimization after $0.2\,\text{s}$. The optimal trajectories in the first row appear similar to the optimized trajectories of our approach after convergence. The differences originate from the slightly different representation of the trajectories themselves, and the difference in the cost function. However, our approach was able to compute the trajectories orders of magnitude faster than the approach of Kim et al. [71]. Our optimization converged in less than $0.7\,\text{s}$ for all classes, compared to $187\,\text{s}$. Even when stopping the optimization after $0.2\,\text{s}$, our approach delivered reasonable trajectories that are suitable for a navigation system that computes trajectories with $5\,\text{Hz}$.

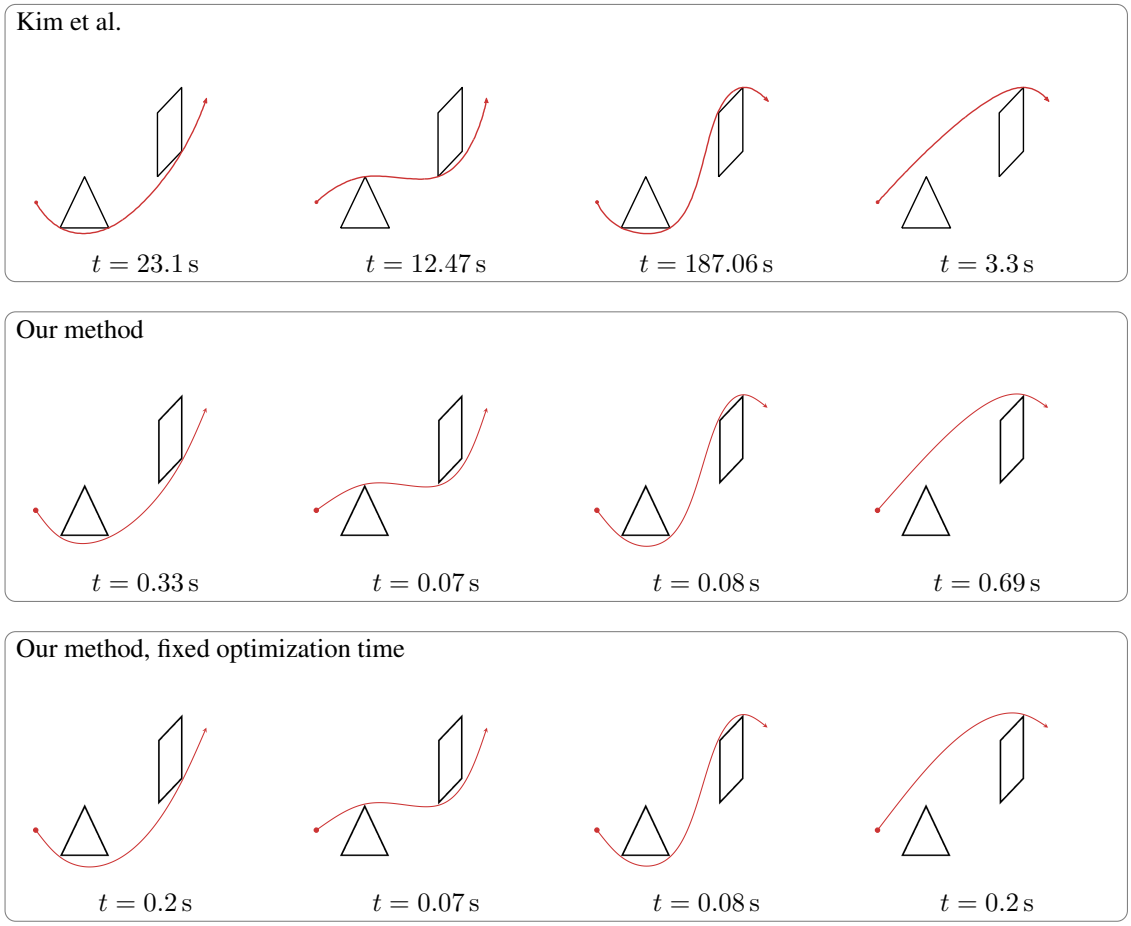


Figure 4.10: Comparison of our method with the approach presented by Kim et al. [71]. It shows that our trajectory optimization converges faster to the local optimum within each homotopy class compared to the existing method. Additionally, it shows that our method generates valid navigation paths even with a fixed optimization time suitable for online navigation. Top: optimal trajectories with minimal acceleration for an environment with two convex obstacles, computed by the method of Kim et al. [71]. Middle: trajectories after convergence computed by our method, where we initialize the optimization with paths in all homotopy classes. Bottom: trajectories computed by our method when stopping the optimization after $0.2\,\mathrm{s}$. For each trajectories the figure shows the computation time in seconds.

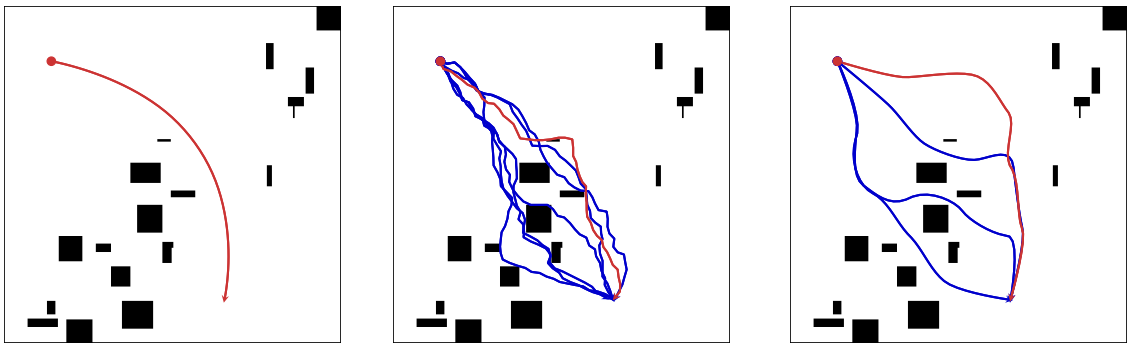


Figure 4.11: Comparison of the initialization with RRT sampling and our method on a randomly generated map. Left: the red trajectory shows the globally optimal path. Middle: initial trajectories from an RRT method. The seventh sample, shown in red, is in the homotopy class of the globally optimal trajectory. Right: initial trajectories on the Voronoi path. The fourth shortest path on the Voronoi graph, shown in red, is in the homotopy class of the globally optimal trajectory.

4.4.3 Comparison to RRT Initialization

The final experiment in this section aims to show the advantage of using homotopy classes for initializing optimization instead of using a sampling based initialization. In this experiment, we used a cost function similar to the experiment above, where we in addition optimized the total travel time. As a result, the overall cost function is a weighted sum of the total travel time, the distance to obstacles, and the acceleration along the trajectory:

$$c(\mathbf{p}) = \theta_0 T + \theta_1 \int_{t=0}^{T} \frac{1}{\| \mathbf{p}(t) - \mathbf{o}_{\text{closest}}(t) \|^2} \, dt + \theta_2 \int_{t=0}^{T} \| \ddot{\mathbf{p}}(t) \|^2 \, dt. \tag{4.13}$$

As a finite dimensional representation of trajectories, we used cubic splines with a discretization of $1\,\text{s}$. See section Sec. 5.1.1 for a detailed description of splines.

To compare the algorithms, we randomly generated 100 different navigation tasks on a grid map of $4 \times 4\,\text{m}$. In each map, we randomly sampled the size and position of 20 obstacles, and the start- and goal position of the agent. Fig. 4.11 shows an example of a randomly generated map.

For each of these tasks, we computed initial paths using our proposed method in all distinct loopless homotopy classes, and sorted them with respect to their path length. We then converted the paths into a spline-based trajectory representation and optimized them using RPROP [126], a gradient based optimization method. After convergence, we assume that we found a unique local minimum within each homotopy class, and therefore the globally optimal trajectory by selecting the optimized trajectory with the lowest cost. In general, we cannot guarantee that there are not multiple local minima within a homotopy class. However, in our experiment different initializations in the same homotopy class always converged to the same trajectory given the cost function in Eq. (4.13). Fig. 4.11 shows the globally optimal path on the left.

As a baseline method, we used the RRT [94] method to sample paths that can be used to initialize optimization. For the RRT method, we sampled a position from a uniform distribution over the map with a probability of 0.9, and the goal position with a probability

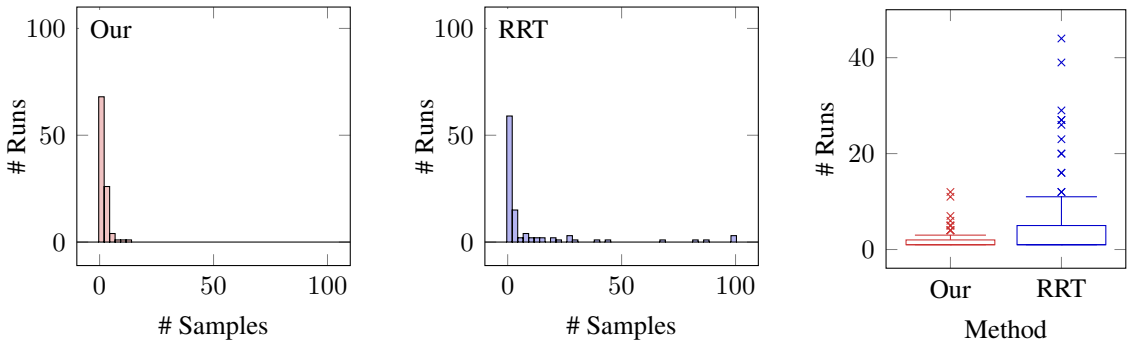


Figure 4.12: Comparison or our approach with RRT sampling to initialize an optimization method. The figures show the number of samples needed to find a trajectory in the homotopy class of the globally optimal trajectory. Left/Middle: histogram of the number of initialization needed for our method and for RRT in 100 runs. Right: Box plot comparing the two methods.

of 0.1. This guides the RRT towards the goal position, while allowing the search tree to explore the whole state space. For each sample we extended the closest node of the search tree by a straight line of $3\,\mathrm{cm}$. For each randomly generated map, we generated a maximum number of 100 initial paths until the globally optimal homotopy class was found. Fig. 4.11 shows an example of an environment where the seventh RRT path (middle), and the fourth shortest Voronoi path (right) was in the desired homotopy class.

To compare our method with RRT, we evaluated the number of samples needed until the homotopy class that contains the globally optimal path was found. In our experiments, the worst case in all 100 sampled environments for our method was that the 14th shortest path on the Voronoi diagram represented the desired homotopy class. In comparison, the RRT method did not find a single path in the desired class in three cases. Fig. 4.12 shows the histogram for both methods, and a box plot that shows the statistics of the number of samples needed in comparison.

As a result, this experiment suggests that our approach outperforms a state-of-the-art sampling method in initializing trajectory optimization methods. Our method needs less samples compared to the RRT method to find an initial trajectory in the homotopy class that contains the globally optimal trajectory. Furthermore, we can guarantee that we explore all classes, which is not the case for sampling based methods with a finite number of samples.

4.5 Related Work

Our model considers trajectories in distinct homotopy classes, which result from the different choices of how to bypass obstacles or other agents. Other authors also investigated methods to compute homotopically distinct paths for mobile robot navigation. Bhattacharya et al. [11] introduce the concept of the H-signature, which is a unique identifier for the homology class of a path, and propose to augment a graph representation of the environment with this signature. They then perform an A* search on the augmented graph to find the optimal path within a given homology class. For example, they apply

their method to an eight-connected grid map of the environment. The resulting graph representation contains multiple paths to the goal of the same homology class, which increases the complexity of the A* search. To lower the computational burden, we compute an abstract graph in which each path corresponds to a unique homotopy class.

To find optimal trajectories with respect to cost functions that also capture higher-order properties such as accelerations, Kim et al. [71] propose to cast the trajectory generation problem as a mixed-integer quadratic program. They represent trajectories using a linear combination of basis functions and utilize the H-signature to find optimal trajectories of a particular homology class. With our work we also aim to optimize trajectories with respect to higher-order cost functions. However, we propose to initialize optimization in different homology classes instead of augmenting the trajectory space.

Vernaza et al. [148] point out that the vector of winding numbers around each obstacle is equivalent to the H-signature used in the abovementioned work, and is therefore an invariant for all trajectories of a given homology class. They present an algorithm that computes paths that satisfy certain winding constraints. For example, their method is applicable to compute optimal paths for a UAV that winds exactly twice around each object in a surveillance task. We also use the vector of winding numbers to compute a unique identifier that fully describes the homology class of a composite trajectory. Igarashi and Stilman [60] apply homotopic path planning for tethered robots that are constrained due to the cable that is attached to a basis.

Similar to our work, Banerjee and Chandrasekaran [9] generate an abstract graph representation from a Voronoi diagram and perform graph search on this abstract graph. They use the abstract representation of the free space for military applications such as rapid re-routing should a path be suddenly blocked. Vela et al. [146] propose to use Voronoi paths for the computation of potential routes for aircrafts, where the obstacles in this case correspond to areas of bad weather that the aircraft should avoid. The previous two approaches describe the general idea of using abstract graphs that correspond to the Voronoi diagram for navigation tasks, whereas we present algorithms that efficiently compute the Voronoi diagram, its abstract graph representation and the corresponding trajectories in the 2D environment. In addition, we evaluate the applicability of our approaches to online robot navigation in experiments.

As an alternative to Voronoi diagrams Demyen and Buro [34] propose Delaunay triangulation to generate an abstract graph that represents the environment. Similar to our work, resulting paths in this graph are then mapped back to a trajectory in the original 2D environment. Whereas Demyen and Buro [34] assume a polygonal representation of the environment, we allow arbitrary obstacles on grid maps, which enables us to incorporate real-world sensor data.

Sprunk et al. [136] perform trajectory optimization using a spline-based representation of the trajectories that also captures higher-order properties. In contrast to our method, they use heuristics to find admissible initial paths in a lower dimensional space. However, the trajectory after optimization is not guaranteed to be in the optimal homotopy class. By parallel initialization and optimization in different homotopy classes, we can overcome this problem.

4.6 Conclusion

In this chapter we presented a method to efficiently compute trajectories in homotopically different classes. We described algorithms that efficiently compute the Voronoi diagram of an environment, generate an abstract graph representation of this environment, and find the shortest paths in this abstract representation. Each path in the abstract graph corresponds to a homotopically distinct path in the environment. These paths can be used for informed initialization of trajectory optimization schemes. Furthermore, we can utilize our method for winding number constrained problems in navigation tasks. To identify the class of a given trajectory, we presented methods to efficiently compute the set of winding numbers, which is also important to re-use optimized trajectories during navigation. We presented a set of experiments that suggest that our method outperforms related methods in terms of runtime, which makes it applicable to online mobile robot navigation. In the following chapter, we present a model of the interactive navigation behavior of multiple agents. This model captures discrete navigation decisions, which correspond to the different homotopy classes. Therefore, the techniques presented in this chapter are a basis for the following approaches to socially compliant mobile robot navigation.

Chapter 5

Socially Compliant Mobile Robot Navigation

The range of applications for robots extends more and more from gated factory halls to services in direct contact with humans. To interact with humans in a socially compliant way, these robots need to understand and comply with mutually accepted rules. In this chapter, we present a novel approach to model the cooperative navigation behavior of interacting agents. This model is able to predict the behavior of nearby pedestrians, as well as to plan suitable trajectories for the robot itself. In particular, we unify prediction and planning, taking into account the expected reactions of the humans to the actions of the robot. Our model captures both, discrete navigation decisions such as on which side to evade a pedestrian, as well as the continuous form of trajectories with desired properties in velocities, accelerations and the proximity to obstacles. During navigation, the robot continuously considers multiple possible trajectories and selects the one that is socially most acceptable in the current situation. We implemented our approach on a real mobile robot and demonstrate that it is able to successfully navigate in an office environment in the presence of humans. Our experiments furthermore suggest that our method outperforms traditional path planning methods for navigating a mobile robot in populated environments.

In the future, more and more robots are expected to perform services in the direct vicinity of humans. Instead of gated factory halls, these robots will work in a shared environment with humans. Especially mobile service robots, or robots that conduct transportation tasks in populated environments, should navigate in a socially compliant way to not unnecessarily hinder nearby pedestrians.

Pedestrians navigate with ease through complex environments, even in densely crowded areas. This seems to be possible due to mutually accepted rules [45], which becomes apparent when encountering pedestrians of different cultures where the learned navigation rules suddenly fail. For example, pedestrians in different cultures show a particular side preference when evading each other [107]. Mobile robots that understand and follow such

mutually accepted rules are able to navigate through populated environments in a socially more compliant way. We believe that such robots are less hindrance to nearby pedestrians and that this increases the acceptance of robots in human environments. Therefore, the goal of our work is to endow mobile robots with a model of the natural navigation behavior of pedestrians.

In this chapter, we present a novel approach to model cooperative navigation behavior of multiple agents. The model uses features that capture the intent of pedestrians to reach a target in the most comfortable way, in accordance with psychological studies. We model the behavior in terms of a mixture distribution that captures both, the discrete navigation decisions as well as the natural variance of the trajectories. To account for the discrete choices to evade obstacles or other pedestrians on either side, we utilize our method for online generation of homotopically distinct paths, which we described in the previous chapter. Our model yields a probability distribution over composite trajectories, i.e., the trajectories of all agents involved in an interaction. The exact shape of this distribution depends on a set of features that capture important properties of natural navigation behavior and on their weights. These weights provide a convenient way to adapt the behavior to the specific requirements of the environment the robot is employed in. In Chap. 6 we will present methods to learn the feature weights from demonstration.

A model as described above allows a mobile robot to reason about the natural navigation behavior of nearby pedestrians and to predict their trajectories. However, predicting the behavior of the environment is only one part of successful mobile robot navigation. Obviously, the crucial step is to plan and follow trajectories that lead the robot to its goal position. As pointed out by Trautman and Krause [141], mobile robot navigation fails in densely populated environments unless the robot takes into account the interaction between itself and the humans. Nearby humans will react to the actions of the robot, which is why the robot has to adapt its behavior, which in turn will affect the humans. To break up this infinite loop, we propose to unify prediction and planning and simultaneously predict the movement of pedestrians and plan the path of the robot.

During navigation, our model allows a mobile robot to consider the interaction between itself and a set of nearby pedestrians. The robot continuously computes the most likely cooperative behavior that allows all agents involved in the navigation task to reach their goal position as smoothly as possible. To this end, our method maintains a set of optimized composite trajectories for each homotopy class, for which we describe an efficient gradient-based optimization scheme. Each optimized composite trajectory in this set represents a locally optimal behavior that corresponds to a specific discrete navigation choice for each of the agents involved. Our model assigns probabilities to each of these discrete choices. Selecting the most likely choice results in the socially most acceptable solution for the given situation.

The most likely trajectory the robot chooses in a particular situation contains a prediction for all other agents, as well as a plan for the motion from its current position to its goal position. In this way, the robot is able to adapt its movement early on in an encounter, avoiding sudden and jerky evasive movements. This is in contrast to reactive methods, which evade dynamic obstacles in a purely reactive manner and do not generate long-term plans for the robot.

We implemented our approach on a real autonomous wheelchair and demonstrate that it is able to successfully navigate in an office environment in the presence of humans. Furthermore, a comparison suggests that our method outperforms traditional path planning methods for navigating a mobile robot in dynamic environments.

The remainder of this chapter is organized as follows. First, in Sec. 5.1, we present a model of the interactive navigation behavior of multiple agents. Sec. 5.2 focuses on the application of this model to mobile robot navigation. In Sec. 5.3 we describe the setup of the autonomous wheelchair that we use for our experiments, which we present in Sec. 5.4. In Sec. 5.5 we discuss related work and conclude the chapter in Sec. 5.6.

5.1 Modeling Interactive Navigation Behavior

In this section, we present a model that captures the natural navigation behavior of pedestrians and at the same time the planned trajectories of mobile robots. This model assigns probabilities to composite trajectories, i.e., the behavior of all agents involved in the navigation task.

5.1.1 Trajectories

We represent the planar movements of each agent in terms of its trajectory

$$\mathbf{p}(t) = \begin{pmatrix} p_x(t) \\ p_y(t) \end{pmatrix} \in \mathbb{R}^2 \tag{5.1}$$

that continuously maps time t to the continuous configuration space $\mathbb{R}^2$ of the agent. This trajectory defines the position of an agent at time t. Our model uses trajectories to predict the movements of pedestrians and to represent planned paths for a mobile robot. To efficiently reason about trajectories, we need a finite-dimensional representation. Furthermore, especially for planning paths for mobile robots it is important that the trajectories satisfy certain smoothness conditions.

We found splines to provide a suitable representation of trajectories that meet these requirements. Splines are piecewise polynomial functions with smoothness conditions at the nodes where the pieces connect. Specifically, a cubic spline with m segments represents for each interval $[t_j, t_{j+1}]$ with $t_0 \leq t_j < t_m$ the restriction $\mathbf{p}\,|_{[t_j,t_{j+1}]}(t)$ of the trajectory to the interval $[t_j, t_{j+1}]$ as a two-dimensional polynomial of degree three

$$\mathbf{p}|_{[t_j,t_{j+1}]}(t) = \begin{pmatrix} a_{x,j}t^3 + b_{x,j}t^2 + c_{x,j}t + d_{x,j} \\ a_{y,j}t^3 + b_{y,j}t^2 + c_{y,j}t + d_{y,j} \end{pmatrix}. \tag{5.2}$$

Instead of directly specifying the set of coefficients $\{a_{x,j}, b_{x,j}, \dots\}$, there are more convenient ways to parameterize such a spline. The parameters of Bézier splines and B-Splines are a set of 2D control points where the curve is completely contained in their convex hull. They provide an intuitive way to modify the form of the spline, which is why they are widely used in computer graphics [14]. In contrast, Catmull-Rom splines pass through

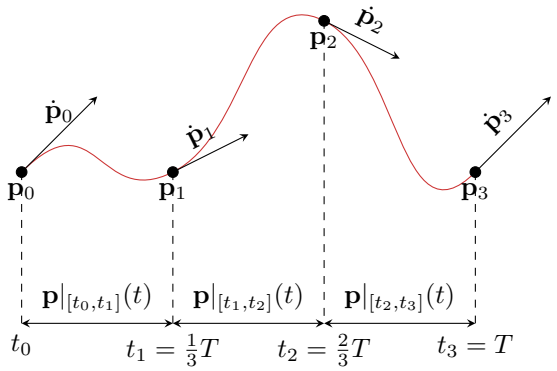


Figure 5.1: Spline representation. Each control point represents the position and velocity at a discrete point in time. This fully defines the spline segments, which are cubic 2D polynomials. We use equidistant control points by dividing the total travel time T into segments that represent equal time intervals each.

all their control points [19], which is a convenient property when interpolating between points.

In this work, we use cubic Hermite splines as a finite-dimensional representation of trajectories. The parameters for a cubic Hermite spline are the position and the first derivative

$$\mathbf{p}(t_j) =: \mathbf{p}_j = \begin{pmatrix} p_{x,j} \\ p_{y,j} \end{pmatrix} \quad \text{and} \quad \dot{\mathbf{p}}(t_j) =: \dot{\mathbf{p}}_j = \begin{pmatrix} \dot{p}_{x,j} \\ \dot{p}_{y,j} \end{pmatrix} \tag{5.3}$$

at the end points of each spline segment, as illustrated in Fig. 5.1. This parameterization has inherent C^1 continuity, since adjacent spline segments share control points.

The x-component of the trajectory in the interval $[t_j, t_{j+1}]$ is given by

$$p_x|_{[t_j,t_{j+1}]}(t) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} 2p_{x,j} + \dot{p}_{x,j}\Delta t - 2p_{x,j+1} + \dot{p}_{x,j+1}\Delta t \\ -3p_{x,j} - 2\dot{p}_{x,j}\Delta t + 3p_{x,j+1} - \dot{p}_{x,j+1}\Delta t \\ \dot{p}_{x,j}\Delta t \\ 3p_{x,j} \end{pmatrix}, \tag{5.4}$$

where $\Delta t = t_{j+1} - t_j$ is the time duration of this segment and

$$u = \frac{t - t_j}{t_{j+1} - t_j} \tag{5.5}$$

is the normalized spline segment parameter with $0 \leq u \leq 1$. Since in our representation the trajectory $\mathbf{p}(t)$ is a function over time t, the velocity $\mathbf{v}(t)$ coincides with the first derivative of the spline. Therefore, the x-component of the velocity is the first derivative of Eq. (5.4) with respect to time t, i.e.,

$$v_x|_{[t_j,t_{j+1}]}(t) = \frac{1}{\Delta t} \begin{pmatrix} u^2 & u & 1 \end{pmatrix} \begin{pmatrix} 6p_{x,j} + 3\dot{p}_{x,j}\Delta t - 6p_{x,j+1} + 3\dot{p}_{x,j+1}\Delta t \\ -6p_{x,j} - 4\dot{p}_{x,j}\Delta t + 6p_{x,j+1} - 2\dot{p}_{x,j+1}\Delta t \\ \dot{p}_{x,j}\Delta t \end{pmatrix}. \tag{5.6}$$

Similarly, the x-component of the acceleration is the second derivative, i.e.,

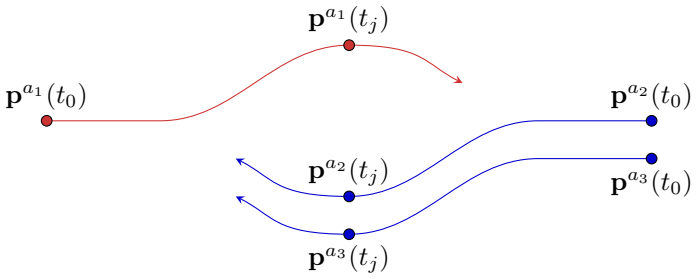


Figure 5.2: This figure shows a composite trajectory that comprises the 2D navigation behavior of three agents. The points highlight the position of the three agents at time t_0 and at time t_j, respectively.

$$a_x|_{[t_j,t_{j+1}]}(t) = \frac{1}{\Delta t^2}\,(u\ 1)\begin{pmatrix} 12p_{x,j} + 6\dot{p}_{x,j}\Delta t - 12p_{x,j+1} + 6\dot{p}_{x,j+1}\Delta t \\ -6p_{x,j} - 4\dot{p}_{x,j}\Delta t + 6p_{x,j+1} - 2\dot{p}_{x,j+1}\Delta t \end{pmatrix}. \tag{5.7}$$

The y-components of the position, velocity and acceleration can be computed analogously. The closed form computation of these properties allow us to efficiently evaluate the feature values that we introduce in Sec. 5.1.4 and Sec. 5.1.6.

The position of the last control point $\mathbf{p}_m$ represents the target position of the agent, which the agent reaches at time t_m. To control the time the agent needs to reach the target position, we allow the total travel time $T = t_m - t_0$ to vary and distribute the time of all other control points uniformly along the trajectory, i.e., $t_j = t_0 + T\frac{j}{m}$.

As a result, a trajectory for agent a_i with m segments has $4(m+1)+1$ degrees of freedom in total, with parameters

$$P^{a_i} = (p_{x,0}, p_{y,0}, \dot{p}_{x,0}, \dot{p}_{y,0}, \ldots, p_{x,m}, p_{y,m}, \dot{p}_{x,m}, \dot{p}_{y,m}, T). \tag{5.8}$$

5.1.2 Composite Trajectories

In our model, we consider the behavior of a set of agents $a_i \in A$ with $i \in \{0, .., n\}$. In the context of mobile robot navigation, this may be a robot or a team of robots, and pedestrians in the vicinity of the robot. To represent the behavior of all agents $a_i \in A$, we use composite trajectories

$$\mathbf{x}(t) = (\mathbf{p}^{a_1}(t), \ldots, \mathbf{p}^{a_n}(t)) \in \mathcal{X}, \tag{5.9}$$

where $\mathbf{p}^{a_i}$ are trajectories as defined above. A composite trajectory encodes the interactive navigation behavior in a certain situation, i. e., the state of each agent over time. Fig. 5.2 illustrates a composite trajectory for three agents that evade each other, and the position of all agents at a specific point in time t_j. In the following section we introduce a probabilistic model that represents distributions over these composite trajectories. In this way we model the cooperative behavior of all agents involved in the interaction process.

The parameterization of a composite trajectory is thus the combination of parameters for the trajectories of each agent, i.e.,

$$P = (P^{a_1}, \ldots, P^{a_n}). \tag{5.10}$$

5.1.3 Modeling Continuous Navigation Behavior

We model the natural navigation behavior of interacting agents using a distribution over composite trajectories. This distribution captures goal-directed navigation, i.e., it assigns high probability to trajectories that lead the agents fast but safely and comfortably to their goal positions. Since comfort is subjective and the requirements for the trajectories of the robot change with the environment, we use an adjustable cost function to represent the navigation goals. This cost function is a linear combination of features that capture important properties of the navigation behavior. As a result, likely trajectories are a trade-off between different navigation goals that might have antagonistic effects, such as reaching a target as fast as possible and minimizing accelerations along the path.

The features that capture these navigation goals are functions

$$f : \mathcal{X} \to \mathbb{R} \tag{5.11}$$

that map composite trajectories $\mathbf{x} \in \mathcal{X}$ to real numbers $f(\mathbf{x})$. We denote the vector that comprises all features as $\mathbf{f}$. The cost function

$$\boldsymbol{\theta}^T \mathbf{f}(\mathbf{x}) = \sum_i \theta_i f_i(\mathbf{x}) \tag{5.12}$$

that encodes the desired navigation behavior consists of a weighted sum of feature values, where $\boldsymbol{\theta}$ is a weight vector that provides a convenient way to adjust the modeled behavior. Based on this cost function, we model the interactive navigation behavior as an exponential family distribution

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(-\boldsymbol{\theta}^T \mathbf{f}(\mathbf{x})\right), \tag{5.13}$$

where $Z(\boldsymbol{\theta})$ is a normalization factor that enforces the distribution to integrate to one. Therefore, we assume non-deterministic, utility optimizing agents that choose trajectories with lower cost exponentially more likely. A distribution of this form is also the result of a maximum entropy assumption when learning from demonstration, as we will describe in detail in Chap. 6.

The fact that we model the interactive behavior of all agents using a joint distribution over composite trajectories implies that we assume cooperative agents, i.e., the goal of all agents is to minimize a common cost function, which is also known to all of the agents. While we assume cooperative behavior, the individual behavior of agents can still differ. For example, our model is able to prioritize one agent such that all other agents clear the way of this agent, as illustrated in Fig. 5.3. Increasing the weight for the feature that captures the travel time for the agent depicted in red yields a distribution in which trajectories with a long travel time for this agent have lower probability. As a result, the most likely composite trajectory changes. However, we still assume that all agents agree on this common cost function, i.e., they all agree to prioritize one agent.

The behavior resulting from the model described above strongly depends on the form of the features. In the following, we propose a set of features that model natural navigation behavior of pedestrians and that also yield trajectories that are suitable for mobile robot navigation.

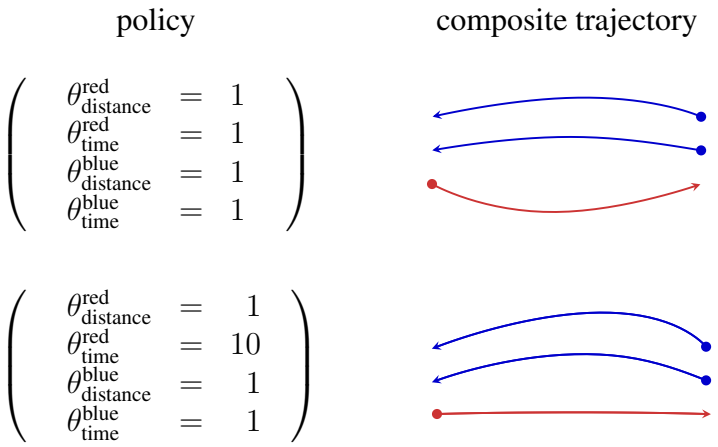


Figure 5.3: This figure illustrates the effect of changing feature weights to the output of our model. We show for two different weight settings the most likely trajectory. Changing the feature weight that corresponds to the time of the red agent from one to ten leads to a behavior where the blue agents clear the way.

5.1.4 Features Capturing Continuous Behavior

According to recent studies, pedestrians seem to consider various criteria for navigation, which we model using a set of features [57]. In particular, we propose to use features that model an intent to reach the target positions energy efficiently, taking into account velocities and proximity to other pedestrians and obstacles. These characteristics also apply to the desired motion of a mobile robot that navigates through populated environments.

Each feature may capture the behavior of one agent or a group of agents $G \subset A$. This allows us to conveniently model groups of homogeneous agents, such as a standard model for pedestrians, but also to give distinct properties to individual pedestrians or robots. For example, Fig. 5.3 shows a policy where the group of agents depicted in blue share a common behavior.

Time

The incentive of a group G of agents to reach a certain target position as fast as possible [104] is captured by the feature

$$f^G_{\text{time}}(\mathbf{x}) = \sum_{a \in G} \int_{t=0}^{T^a} 1 \; dt = \sum_{a \in G} T^a, \tag{5.14}$$

which is the sum of the elapsed time of all agents in the group A to reach their respective target positions.

Acceleration

Pedestrians typically aim to walk efficiently, avoiding unnecessary accelerations [57, 104]. Integrating the squared acceleration over the trajectory yields the feature

$$f^G_{\text{acceleration}}(\mathbf{x}) = \sum_{a \in G} \int_{t=0}^{T^a} \|\ddot{\mathbf{p}}^a(t)\|^2 \, dt. \tag{5.15}$$

The x and y component of the acceleration are polynomials (Eq. (5.6)), therefore also the squared acceleration is a polynomial and we can compute this integral in closed form.

Velocity

We model the goal to maintain a certain velocity that is uncomfortable to exceed [53] using the feature

$$f^G_{\text{velocity}}(\mathbf{x}) = \sum_{a \in G} \int_{t=0}^{T^a} \|\dot{\mathbf{p}}^a(t)\|^2 \, dt \tag{5.16}$$

that integrates the squared velocity over the trajectories. Similar to the acceleration, we can also compute this feature in closed form.

Clearance to other agents

Pedestrians tend to evade other pedestrians, a behavior which we also expect from a cooperative mobile robot. We assume that the evasive maneuvers depend on the distances between the agents, yielding the feature

$$f^G_{\text{distance}}(\mathbf{x}) = \sum_{a \in G} \sum_{\substack{b \in A \\ b \neq a}} \int_{t=0}^{T^a} \frac{1}{\|\mathbf{p}^a(t) - \mathbf{p}^b(t)\|^2} \, dt. \tag{5.17}$$

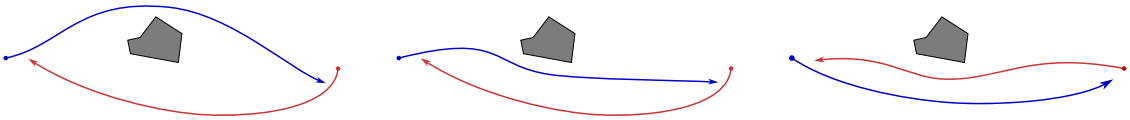
Unfortunately, it is not possible to compute this feature in closed form due to a polynomial in the denominator of the integrand. We therefore use numeric integration to evaluate f^G_{distance}. For efficiency reasons, we vary the intervals Δt of the numeric integration based on the distance between two agents. We decrease the intervals Δt when two agents are close to each other and allow larger intervals when they have a greater distance. In this way, we assure that we accurately compute this feature when two agents are close to each other.

Clearance to static obstacles

In addition to avoiding other agents, we model the proximity to walls and other static obstacles with the feature

$$f^G_{\text{obstacle}}(\mathbf{x}) = \sum_{a \in G} \int_{t=0}^{T^a} \frac{1}{\| \mathbf{p}^a(t) - \mathbf{o}^a_{\text{closest}}(\mathbf{p}^a(t))\|^2} \, dt, \tag{5.18}$$

where $\mathbf{o}^a_{\text{closest}}(\mathbf{p})$ is the position of the closest obstacle to position $\mathbf{p}$. Similar to f^G_{distance}, we apply numeric integration with variable sampling density to compute this feature value.



(a) (b) (c)

Figure 5.4: Example of three homotopically distinct composite trajectories in the same situation. (a vs. b): the blue agent passes the solid obstacle on opposite sides. (b vs. c): the two agents pass each other on different sides.

5.1.5 Modeling Discrete Navigation Decisions

In the previous section we proposed a model that yields a distribution over composite trajectories, where smooth and comfortable trajectories are more likely. In addition to such continuous properties, the decision process of interacting agents often also comprises discrete decisions, such as on which side to evade an other agent or whether to move through a group of people that belong together. We can capture such characteristics in terms of homotopy classes of the trajectories.

In addition to homotopy classes originating from static obstacles, which we introduced in Chap. 4, we also consider homotopy classes that result from the interaction of multiple agents. We define two composite trajectories as homotopic if the trajectories of each agent can be continuously transformed into each other without intersecting with static obstacles or the trajectories of other agents. In addition to the distinct ways to navigate around static obstacles, this also captures the possibilities of each pair of agents to evade each other either on the left side, or on the right side. Fig. 5.4 illustrates three distinct composite trajectories. The composite trajectories depicted in (a) and (b) differ in one of the agent's decision to bypass the static obstacle on the other side. In (b) and (c), the agents pass each other on distinct sides. Hence, the depicted trajectories (a), (b), and (c) are non-homotopic. We denote the homotopy classes for the space of composite trajectories as $\psi \in \Psi$.

To identify the classes, we utilize the winding numbers of the trajectories. As described in Sec. 4.3, the winding numbers serve as an approximate identifier of homotopy. For static obstacles, the winding number captures the rotation of the vector from the position of the agent to a representative point of the obstacle along the trajectory. We apply the same principle to pairs of agents. For the winding number of a pair of agents, however, the representative point moves along the trajectory of the second agent. In this case, we effectively compute the rotation of the vector $\mathbf{p}^b(t) - \mathbf{p}^a(t)$ for a pair of agents a and b along their trajectories.

Given a set of homotopically distinct classes, we model the behavior of multiple agents as a two-stage process. First, the agents choose a discrete class $\psi \in \Psi$. Second, they pick a certain manifestation of a composite trajectory within this class. We model the second stage in terms of distributions over composite trajectories that we described in the previous section. However, instead of one distribution over the space of all composite trajectories $\mathbf{x} \in \mathcal{X}$, we have a distribution $p^{\psi}_{\boldsymbol{\theta}}(\mathbf{x})$ with $\mathbf{x} \in \psi$ for each homotopy class ψ.

We also model the first stage, i.e., the discrete navigation decisions, with an exponential

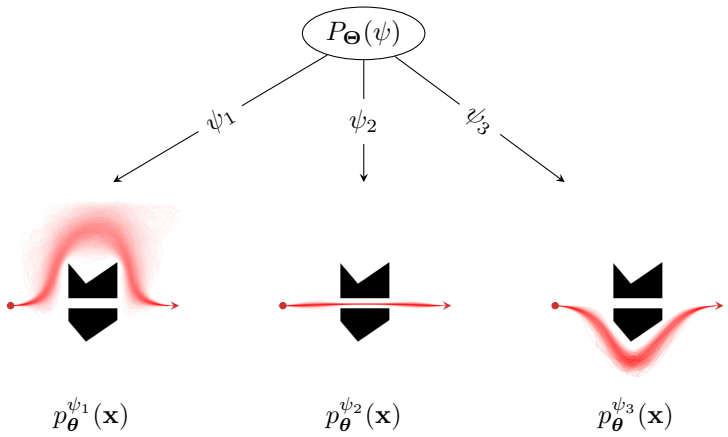


Figure 5.5: We use a mixture distribution to model cooperative navigation behavior. The discrete probability distribution $P_{\boldsymbol{\Theta}}(\psi)$ describes the choice of homotopy classes, whereas the continuous distributions $p_{\psi}(\mathbf{x})$ describe which trajectories of the class the agents are likely to choose. The images show projections of the probability over trajectories from the left to the right side of the obstacle.

family distribution that depends on features. Instead of a distribution over composite trajectories,

$$P_{\boldsymbol{\Theta}}(\psi) = \frac{1}{Z(\boldsymbol{\Theta})} \exp\left(-\boldsymbol{\Theta}^T \mathbf{g}(\psi)\right) \tag{5.19}$$

is a distribution over the homotopy classes $\psi \in \Psi$ where the feature vector $\mathbf{g}$ is a function over homotopy classes, and $\boldsymbol{\Theta}$ is a weight vector.

Fig. 5.5 illustrates the two-staged decision process of our model. This example gives rise to a distribution over three homotopy classes that result from static obstacles. For each discrete choice, there is a continuous distribution over trajectories. In the following, we propose features that allow us to model, for instance, cultural preferences of passing other agents on the left or on the right, or how acceptable it is to split groups of people that belong together.

5.1.6 Features Capturing Discrete Decisions

To represent important properties of natural navigation behavior in terms of homotopy classes, we use features

$$\mathbf{g} : \Psi \to \mathbb{R} \tag{5.20}$$

that map homotopy classes $\psi \in \Psi$ to real values. The following features use the winding numbers ω, which capture the sides on which an agent passes an obstacle, or the sides on which a pair of agents passes each other. For static obstacles, $\omega_a^{o_i}$ denotes the winding number of agent a with respect to obstacle o_i. Similarly, ω_a^b denotes the winding number for the interaction between agent a and agent b. For example, two agents that pass each other on the right yield a positive ω_a^b, in contrast to a negative value for passing on the left. The vector $\boldsymbol{\omega}$ that contains all winding numbers is an invariant for all composite

trajectories of one homology class with fixed start- and goal positions. As a result, the values of features that depend only on these winding numbers are invariant within a homology class. In the following, we use winding numbers as a function of classes ψ, which corresponds to the winding numbers of an arbitrary composite trajectory within this class.

Passing left vs. passing right

To capture the decisions to avoid other agents or obstacles on the left or on the right, we consider the feature

$$g^G_{\text{lr}}(\psi) = \sum_{a \in G} \sum_{\substack{b \in A \\ b \neq a}} \omega_a^b. \tag{5.21}$$

The value of this feature increases when two agents evade on the right instead of on the left. For example, it yields a value of 1 for two agents that swap their positions on the right, and a value of -1 for agents evading each other on the left. Therefore, a positive weight for this feature introduces a bias to all composite trajectories in which the agents of the respective group evade each other on the left. The same concept also applies to static obstacles. If we want to model a bias for agents to pass static obstacles on a specific side, we can easily add a feature that represents the winding numbers for static obstacles.

Group behavior

The following feature indicates if the agents of group G move in between a group of agents that belong together. An agent that passes two members of a group on different sides moves through the corresponding group. Therefore, we have

$$g^G_{\text{group}}(\psi) = \sum_{a \in G} |\{B \in \mathcal{B} \mid \exists b, c \in B : b, c \neq a \wedge \omega_a^b \omega_a^c < 0\}|, \tag{5.22}$$

where $\mathcal{B}$ is the set of groups of agents. In particular, this feature is a counter that increases whenever an agent of group A moves in between the members of any of the groups $B \in \mathcal{B}$. A nonzero, positive weight of this feature causes the robot to avoid trajectories that lead it for example in between parents and their children, or a group of visitors in a museum tour.

Cost of the most likely composite trajectory

Furthermore, we allow the features to depend on the distribution over composite trajectories of the corresponding homotopy class. For example, the feature

$$g_{\text{ml\_cost}}(\psi) = \min_{\mathbf{x} \in \psi} \boldsymbol{\theta}^T \mathbf{f}(\mathbf{x}) \tag{5.23}$$

captures the cost of the most likely composite trajectory $\mathbf{x}$ of homotopy class ψ, which allows the model to reason about the homotopy class the agents choose in terms of the cost of the composite trajectory that is most likely according to the distribution $p^{\psi}_{\boldsymbol{\theta}}(\mathbf{x})$.

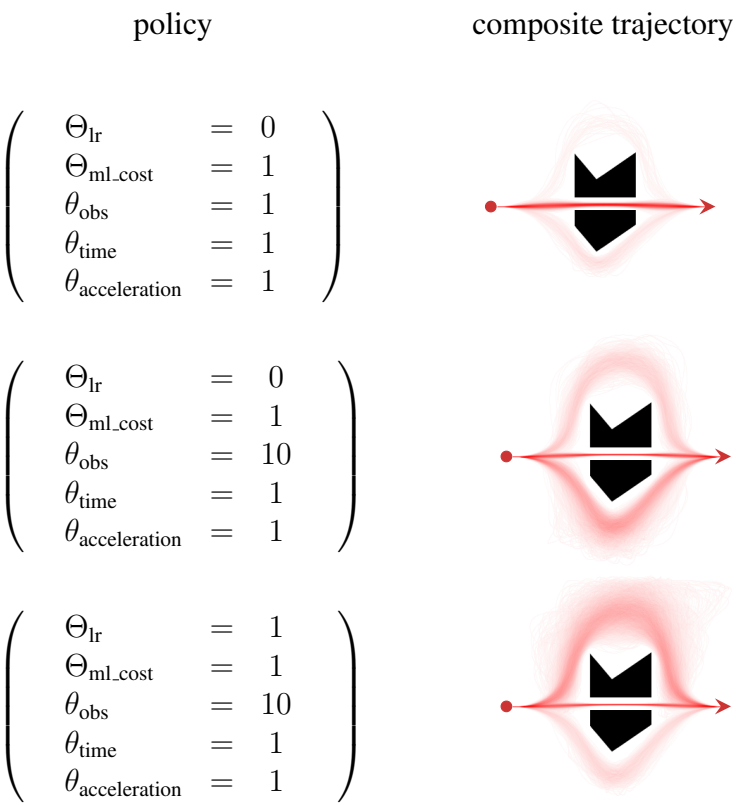


Figure 5.6: This figure illustrates the effect of changing feature weights to the probability distribution of our model. We show the distribution over trajectories of a single agent for three different weight settings, where we project the trajectories to the 2D area. Increasing the feature weight that corresponds to the obstacle distance gives lower probability to trajectories that are close to obstacles. A positive weight of the feature that corresponds to the winding numbers around static obstacles introduces a bias to the left. The images show projections of the distribution over trajectories from the left to the right of the obstacle.

Fig. 5.6 illustrates the resulting probability distribution from the two-staged process for one agent in an environment with two static obstacles. With a low penalty for proximity to obstacles, the most likely trajectories move straight through the narrow passage. When increasing the weight of the feature corresponding to obstacles, evading the passage on the outside becomes more likely, even if this causes longer trajectories. Furthermore, the trajectories of all homotopy classes keep a greater clearance to the obstacles. In the last example, we add a bias for evading obstacles on the left, which shows in the distribution over the trajectories.

5.2 Socially Compliant Mobile Robot Navigation

In the previous section we described a probabilistic model to capture the cooperative behavior of a group of agents. A mobile robot can use this model to predict the behavior of nearby pedestrians and at the same time to plan socially compliant trajectories. In

the following, we present methods to unify prediction and planning for mobile robot navigation. Furthermore, we discuss practical considerations necessary for efficient mobile robot navigation in complex environments.

5.2.1 Unify Prediction and Planning

The model we introduced above is able to predict the cooperative behavior of a group of agents. Let us assume that the model accurately describes the natural navigation behavior of pedestrians, and that we want a mobile robot to imitate human navigation behavior. In this case, we can use the model during navigation to predict how a pedestrian would react in the situation of the robot, including the interaction with other pedestrians. This 'prediction' is what the robot should execute to imitate human navigation behavior. As a result, the robot navigates in a human-like way.

In practice, it might not always be desirable for the robot to imitate human navigation behavior. However, the same principle also applies to a desired behavior of the robot that is different from typical pedestrian behavior. By modifying the feature weights, the robot 'predicts' trajectories for itself that correspond to the desired behavior, still modeling the behavior of nearby pedestrians and their reaction to the actions of the robot. Since the features are meaningful properties of the navigation behavior, changing feature weights is a convenient way to change, for example the desired velocity, the accelerations, or the desired clearance to obstacles.

For a given situation, our model yields a probability distribution over composite trajectories. From this distribution, the robot has to commit to one specific trajectory that it follows. There are different ways of choosing such a trajectory. The robot could either sample from the distribution, which would yield a non-deterministic behavior. For this, the robot would apply ancestral sampling, in accordance to the two-stage process that we described above, first sampling from the discrete set of homotopy classes and subsequently selecting a composite trajectory from the chosen class, as in the example illustrated in Fig. 5.5. However, it is not feasible to sample from the distribution online during navigation. Furthermore, we would have to include a mechanism to avoid hysteresis, i.e., to stick to one plan and not to continuously change it.

We propose an alternative method to utilize our model for efficient mobile robot navigation. During navigation, we choose the most likely state in both stages, selecting the most likely homotopy class

$$\hat{\psi} = \underset{\psi \in \Psi}{\operatorname{argmax}} \exp\left(-\boldsymbol{\Theta}^T \mathbf{g}(\psi)\right) = \underset{\psi \in \Psi}{\operatorname{argmin}} \boldsymbol{\Theta}^T \mathbf{g}(\psi) \tag{5.24}$$

with respect to Eq. (5.19), and subsequently the most likely composite trajectory within this homotopy class

$$\hat{\mathbf{x}}^{\hat{\psi}} = \underset{\mathbf{x} \in \hat{\psi}}{\operatorname{argmax}} \exp\left(-\boldsymbol{\theta}^T \mathbf{f}(\mathbf{x})\right) = \underset{\mathbf{x} \in \hat{\psi}}{\operatorname{argmin}} \boldsymbol{\theta}^T \mathbf{f}(\mathbf{x}) \tag{5.25}$$

that ist given by Eq. (5.13). However, to evaluate the features $\mathbf{g}(\psi)$ that we described in Sec. 5.1.6, we need to compute the cost of the most likely trajectory within each

homotopy class. Computing $\hat{\mathbf{x}}^{\psi}$ for all classes ψ enables us to evaluate the probability distribution over homotopy classes, and subsequently to select $\hat{\mathbf{x}}^{\hat{\psi}}$ that corresponds to the most likely homotopy class $\hat{\psi}$. If our model accurately captures socially compliant behavior, this method results in the a best guess for the behavior of the pedestrians and the socially most compliant plan for the robot.

In practice, due to the variability of human navigation behavior, the actual actions of pedestrians may differ from the predictions captured in $\hat{\mathbf{x}}^{\hat{\psi}}$. Furthermore, the environment might change, such as doors could be closed that were previously open. Therefore, it is important to update the plans continuously during navigation. As a result, it is crucial for our method to optimize the trajectories online efficiently with a high frequency.

5.2.2 Efficiently Optimizing Trajectories

As outlined previously, we need to optimize the composite trajectories of all considered homotopy classes online during navigation. We propose gradient-based optimization techniques to efficiently optimize the parameters P in Eq. (5.10) of an initial composite trajectory $\mathbf{x}$ with respect to its probability density as given in Eq. (5.25). To this end, we compute the derivative of the feature vector $\mathbf{f}$ with respect to the spline control points P of $\mathbf{x}$ using a combination of analytical and numerical differentiations. To optimize the control points of composite trajectories using this gradient, we use the optimization algorithm RPROP [126]

The gradients that lead to the most likely trajectory (Eq. (5.25)) are given by

$$\frac{\partial \boldsymbol{\theta}^T \mathbf{f}(\mathbf{x})}{\partial(p_{x,0}^{a_1}, p_{y,0}^{a_1}, \dot{p}_{x,0}^{a_1}, \dot{p}_{y,0}^{a_1}, \dots, p_{x,m}^{a_1}, p_{y,m}^{a_1}, \dot{p}_{x,m}^{a_1}, \dot{p}_{y,m}^{a_1}, T^{a_1}, \dots)} \tag{5.26}$$

$$= \sum_i \boldsymbol{\theta}_i \frac{\partial f_i(\mathbf{x})}{\partial(p_{x,0}^{a_1}, p_{y,0}^{a_1}, \dot{p}_{x,0}^{a_1}, \dot{p}_{y,0}^{a_1}, \dots, p_{x,m}^{a_1}, p_{y,m}^{a_1}, \dot{p}_{x,m}^{a_1}, \dot{p}_{y,m}^{a_1}, T^{a_1}, \dots)} \tag{5.27}$$

$$=: \sum_i \boldsymbol{\theta}_i \nabla f_i(\mathbf{x}). \tag{5.28}$$

For the features accounting for acceleration and jerk we can compute the derivatives in closed form, since these features are integrals over polynomials. For more complex features for which we cannot compute the gradients analytically, such as the inverse distance to obstacles, we apply a combination of numerical integration and analytical derivatives. In more detail, applying numerical integration to features

$$f(\mathbf{x}) = \int_t c(\mathbf{x}, t) dt \tag{5.29}$$

$$\approx \sum_{t_k} \frac{(t_{k+1} - t_k)}{2} (c(\mathbf{x}, t_k) + c(\mathbf{x}, t_{k+1})) \tag{5.30}$$

allows us to compute the derivatives at each sampling point t_k

$$\nabla f(\mathbf{x}) \approx \sum_{t_k} \frac{(t_{k+1} - t_k)}{2} (\nabla c(\mathbf{x}, t_k) + \nabla c(\mathbf{x}, t_{k+1})). \tag{5.31}$$

For the features listed in Sec. 5.1.4 it is feasible to compute the derivatives of the inner function $c(\mathbf{x}, t_{i+1})$ in closed form and therefore to efficiently compute the feature gradients using numerical integration.

Since we want to compute the most likely trajectory within each homotopy class, it is important to make sure that the homotopy class does not change during optimization. Due to the features f^G_{distance} and f^G_{obstacle} (Sec. 5.1.4), the cost increases when the trajectories approach other agents or static obstacles. As a result, the gradient-based optimization tends to stay within the homotopy class of the initial guess. However, depending on the step size of the gradient-based optimization, we cannot guarantee that the homotopy class does not change. To overcome this limitation, we compute after each optimization step the vector of winding numbers $\boldsymbol{\omega}$ with respect to static and dynamic obstacles. If this vector changes within two optimization steps, we dismiss this step and proceed with a lower step size.

The optimization of the composite trajectories in the different homotopy classes are independent. We can therefore parallelize the optimization process to enable efficient computation. However, in practice the resources in terms of available processor kernels for the navigation algorithm are limited. In the following, we outline how to maintain a set of relevant homotopy classes instead of exploring the space of homotopy classes from scratch.

5.2.3 Maintaining a Set of Relevant Homotopy Classes

During navigation, the robot constantly optimizes composite trajectories of different homotopy classes. The number of homotopy classes increases exponentially with the number of pedestrians and obstacles in the environment. In practice it is therefore not possible to consider all homotopy classes. To enable efficient online computation, we propose to use heuristics that prune the homotopy classes that we consider during navigation.

To this end we rely on the method to generate homotopically distinct paths that we described in Chap. 4. This method returns the k-shortest trajectories on the Voronoi graph. In this way, we rule out long trajectories or detours in the first place. Furthermore, this method filters out trajectories containing loops. From these k-shortest trajectories for all agents we generate all combinations, which results in composite trajectories in different homotopy classes. However, they do not contain variants from the pairwise interactions between agents. In the following we describe an algorithm that iteratively generates relevant homotopy classes.

Alg. 2 describes this method in detail. The idea behind the algorithm is to consider different outcomes of an interaction between two agents only when agents are likely to interact in the first place, i.e., when they are close to each other at some point along their trajectories. In this way, the algorithm ignores interactions between agents that are highly unlikely. First, we initialize the set Ψ' with composite trajectories that arise from static obstacles. For each element $\mathbf{x}$ in this set, our algorithm identifies a potential evasive maneuver when two agents a and b come close to each other at some point along $\mathbf{x}$. For such a potential evasive maneuver, we want to reason about both possible outcomes,

Algorithm 2 Finding the subset of relevant homotopy classes given a set of interacting agents

$\Psi' \leftarrow \Psi'_{\text{static}}$
while `unresolvedCollisions`(Ψ') **do**
  for all $\mathbf{x} \in \Psi'$ **do**
    for all a, b with $a \neq b$ **do**
      if a and b interact in $\mathbf{x}$ **and** $\mathbf{x}_{\pm 2\pi^{ab}} \notin \Psi'$ **then**
        $\Psi' \leftarrow \Psi' \cup \mathbf{x}_{\pm 2\pi^{ab}}$
      end if
    end for
  end for
end while

i.e., passing left or passing right. To this end, we generate a composite trajectory $\mathbf{x}_{\pm 2\pi^{ab}}$ in which the agents a and b pass on the other side compared to the original composite trajectory. Our algorithm repeatedly looks for such potential evasive maneuvers as described above until there are no unresolved collisions, or the maximum number of allowed homotopy classes has been reached. The algorithm resolves collisions that occur earlier first. In this way, even when limited to a small number of classes, we are able to react to potential immediate interactions. During navigation, we repeat the method as described above continuously, to always maintain a set of homotopy classes that captures the likely navigation choices of the agents.

In all generated classes, we optimize the composite trajectories in each planning cycle. The convergence time of optimization-based techniques typically decreases when the initial guess is already close to the optimum. Thus, during navigation, it is desirable to re-use previously optimized trajectories. In particular, from one planning cycle to the next the current position of the robot as well as the environment does not change substantially in most situations. Therefore, we propose to maintain a set of optimized composite trajectories during navigation. Whenever a new homotopy class emerges due to changes in the environment, we need to add a corresponding composite trajectory to the set. On the other hand, when an obstacle vanishes, two homotopy classes merge. In this case, we have two trajectories in the same homotopy class and we can delete one of them, since both classes lead to the same composite trajectory after optimization. For both cases, we identify the homotopy class of a given composite trajectory, for which we rely on the winding numbers that we described in Sec. 4.1.2.

5.2.4 Integration with Global Path Planning

The time to optimize composite trajectories increases with the travel time of the agents. Furthermore, the uncertainties in predicting cooperative behavior grow with the prediction horizon. Therefore, we employ our method to represent the immediate behavior of all the agents in the near future and represent more distant behavior by globally planned paths independently for each agent. This enables a mobile robot to apply the proposed method

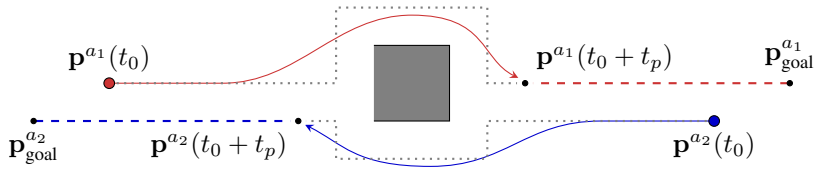


Figure 5.7: This figure shows an example for the global A* plan, the intermediate target positions and a resulting composite trajectory for an environment with two agents and one static obstacle. The dashed lines correspond to the A* plan, the solid lines to the composite trajectory that captures the interactive behavior in the time interval from t_0 to $t_0 + t_p$.

even in large environments.

At time t_0, we apply our model of socially compliant navigation to compute trajectories within the time interval $[t_0, t_0 + t_p]$ and represent the behavior in the time interval $(t_0 + t_p, t_{\text{end}}]$ using the output of a standard path planning algorithm for each agent. For each agent a, i.e., the robot and all pedestrians that are detected by the robot, we generate an initial trajectory $\mathbf{p}^a_{\text{global}}$ to a global target position using A* search. In our experiments, we assume known target positions for the agents. However, there exist approaches that we could use to estimate the target position based on the observed trajectories [10, 160]. As a cost function for global path planning we use the time to reach the target as well as the distance to obstacles. We augment the A* path with a constant velocity profile for each agent, which allows us to compute intermediate target positions given by $\mathbf{p}^a_{\text{global}}(t_0 + t_p)$. Subsequently, we use our model to compute a probability distribution of the composite trajectory of all agents from their current positions to these intermediate target positions.

In our experiments, we typically set the planning horizon to $t_p = 10\,\text{s}$. Fig. 5.7 illustrates an example of the integration with global path planning for an environment with two agents and a static obstacle.

5.2.5 Online Path Planning

During navigation, it is necessary to update the current plan continuously to account for changes in the environment, and for deviations from the prediction and the real behavior of the pedestrians. In each planning cycle, the robot first updates a grid map of the environment given the current sensor readings as well as the estimated position and velocity of the pedestrians, as described in Sec. 5.3.1. Then, the robot computes a trajectory and sends it to the controller for execution.

However, since we apply optimization, the trajectory generation consumes some time. If the robot planned a trajectory with start point at the current state, it would already be at a different state at the time the trajectory is optimized and ready for execution. As a result, there would be a discontinuity in the trajectories the controller receives. To overcome this problem, we use at each planning cycle the expected state of each agent at the time the optimization is finished as start point for the path planning.

Fig. 5.8 illustrates this method. At time t_i, the robot has just sent the red trajectory to the controller. The robot r uses this trajectory to predict the position $\mathbf{p}^r(t_{i+1})$ of the robot at time t_{i+1}, where $\Delta t = t_{i+1} - t_i$ is the maximum allowed optimization time. It

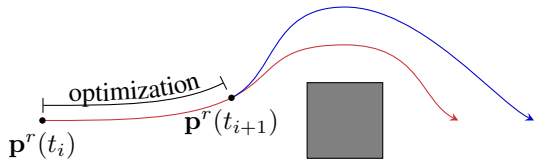


Figure 5.8: Online path planning. At time t_i, the robot sends the red trajectory for execution to the controller and has to initiate the next optimization. Therefore, it uses the sent trajectory to predict the robot's pose at time t_{i+1} and sets the start pose of the trajectory to be optimized to $\mathbf{p}^r(t_{i+1})$. After the optimization has finished, the new trajectory (blue) connects smoothly to the previous trajectory (red).

subsequently sets the start point of all trajectories to $\mathbf{p}^r(t_{i+1})$. By starting the optimization at time t_i, the new trajectory is ready at time t_{i+1}. This method guarantees that the new trajectory that we send to the controller smoothly connects to the previously executed trajectory without discontinuities.

Obviously, this method is only applicable to the robot itself, for which we trust the controller that it keeps the robot on the desired trajectories. For the pedestrians, we use a constant velocity prediction within the time interval Δt. In our experiments, we used a planning rate of $5\,\mathrm{Hz}$. Therefore, the prediction error due to the constant velocity prediction is relatively small.

5.3 Setup of a Robotic Wheelchair

For our experiments we equipped an electronic Permobil C500 wheelchair with controls and sensors that enable the wheelchair to operate autonomously. A custom-made control board interfaces the internal control to send virtual joystick commands to the wheelchair. For odometry, we equipped each of the front wheels with wheel encoders. In addition, we mounted two Hokuyo UTM-30LX laser scanners at the front of the wheelchair, as illustrated in Fig. 5.9. For perception, localization and planning we mounted a desktop computer with an Intel® Core™ i7-4770S $3.9\,\mathrm{GHz}$ processor to the back of the wheelchair. The wheelchair is able to operate fully autonomously, not relying on external sensors or processing devices. In the following, we describe the perception as well as the control components of the wheelchair.

5.3.1 Perception

In order to perceive the environment, we localize the robot in a static grid map using laser-based Monte Carlo Localization (MCL) [140]. This method maintains a probability distribution over the pose of the robot based on the sensor readings of the wheel encoders and the range measurements of the laser scanners. In particular, MCL represents the distribution in terms of a set of particles that are samples drawn from the distribution. In contrast to parameterized representations, such as Gaussian distributions, this method can represent arbitrary, possibly multi-modal distributions.

The localization method returns the pose of the most likely particle as the current pose estimate. Using this estimate, we integrate the laser scans into the static grid map in each

Figure 5.9: Autonomous wheelchair used for the experiments. We equipped the Permobil C500 wheelchair with wheel encoders, two laser range finders at the front and a control interface to send joystick commands to the wheelchair. For processing the navigation algorithms, we mounted a small desktop computer at the back of the wheelchair.

planning cycle. This allows our path planning method to also react to dynamic obstacles that are not contained in the static map.

In addition, we estimate the position and velocity of pedestrians based on laser information, since we treat pedestrians differently from other obstacles. For pedestrians, we predict the behavior using our method to socially compliant robot navigation, in contrast to other obstacles that we integrate into the obstacle grid map. In a first step, we discard the laser beams that correspond to obstacles in the static map. After this background subtraction only obstacles that were not originally mapped remain in the laser image. Subsequently, we extract from the image objects that show typical characteristics of pedestrians. In particular, we filter out objects that are too small, or too large to originate from a potential pedestrian.

In each cycle, we then assign these observations to existing tracks of pedestrians, or add a new track if the observation cannot be assigned to any of the existing tracks. Our method classifies a given track as a pedestrian if it satisfies certain conditions such as smooth movement and velocity limits. By smoothing the history of the track we also estimate the current velocity of each pedestrian.

5.3.2 Controller

To navigate the wheelchair along a given trajectory, we apply a cascaded control scheme. Since we only have access to the joystick input of the wheelchair, which does not directly correspond to velocities, we implemented a PID controller for the velocity commands. This controller takes as input the desired translational and angular velocity as well as the current odometry measurements.

On top of this first controller, we rely on the dynamic feedback controller presented

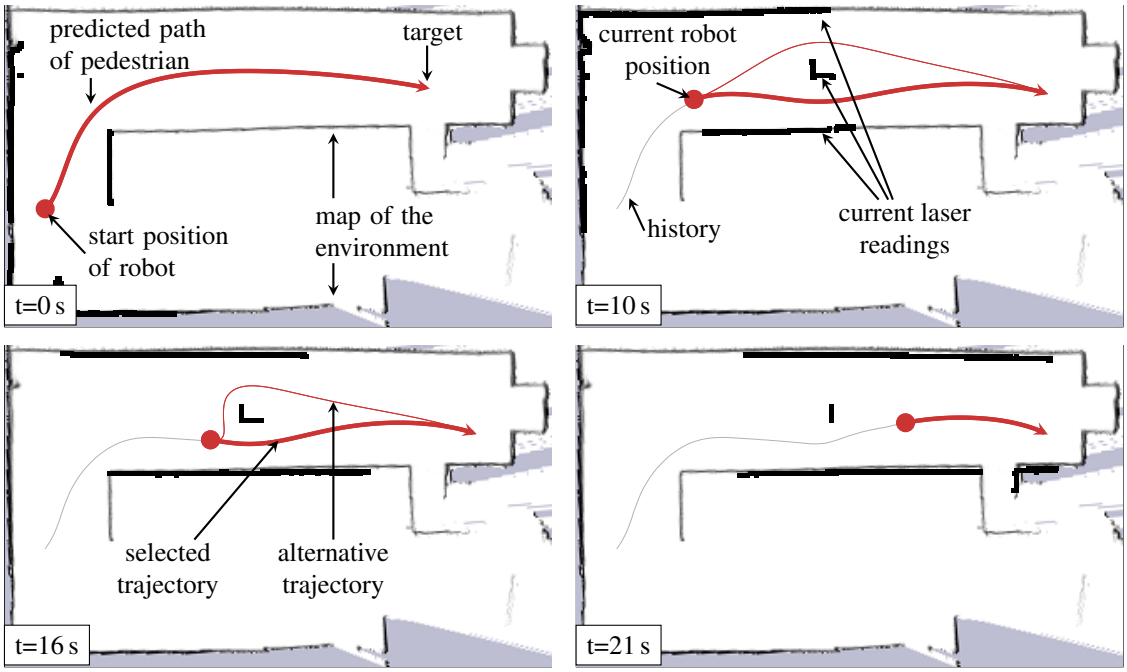


Figure 5.10: A robot controlled by our method navigates from its current position to its target position and avoids a static obstacle in the process. The red dot illustrates the robot's position at four individual time steps, and the gray line visualizes the trajectory that the robot has already driven. When the robot detects the unmapped static obstacle, it computes both possible homotopy classes and behaves according to the most likely trajectory, depicted as a thick red line.

by Oriolo et al. [113] to steer the wheelchair on a given trajectory. It takes as input the desired trajectory and odometry readings as error feedback. Our path planning algorithm continuously replans and sends updated trajectories to the controller. Therefore, the controller always executes the latest available trajectory data for a given time step. The output of this controller is the translational and angular velocity, which it sends to the low-level controller described above.

For safety reasons, we apply a collision avoidance system that can override any commands from the trajectory planning module when necessary. Therefore, we constantly extrapolate the current velocity and check the time to collision with any obstacle currently detected by the laser scanners. Whenever the wheelchair reaches a critical point where the deceleration necessary to avoid collision exceeds a threshold, the collision avoidance module interferes and decreases the velocity commands.

5.4 Experimental Evaluation

The goal of this section is to demonstrate that our approach allows a mobile robot to autonomously navigate in the presence of humans in a socially compliant way. In the following experiments we used the wheelchair system as described above. For the pedestrians, we assume known target positions.

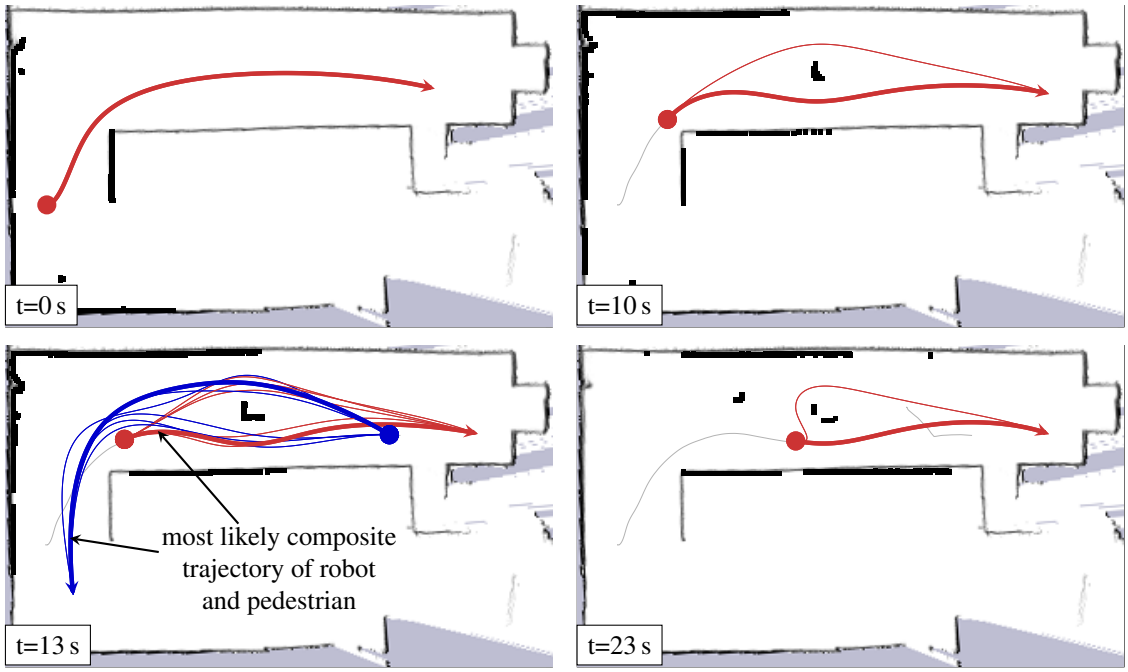


Figure 5.11: A robot controlled by our method avoids a static obstacle and a pedestrian. As soon as the robot detects the pedestrian, it computes the possible cooperative behavior of the pedestrian (blue) and the robot (red). The robot predicts for the pedestrian to evade the obstacle on the right. The pedestrian behaves according to the prediction of the robot and thus the robot proceeds to follow its plan.

5.4.1 Robot Navigation in the Presence of Unmapped Static Obstacles

In a first experiment, the robotic wheelchair controlled by our method navigated through a static environment. Fig. 5.10 visualizes the belief of the robot during the navigation task in terms of the most likely trajectories for each homotopy class. The figure highlights the most likely trajectories that are selected for navigation in thick red in four consecutive time steps. The first image shows the trajectory from the start position of the robot to its target position in the static map of the environment. In the second image, after having traveled around the corner, the robot perceived a static obstacle in the middle of the corridor, which is not part of the static map. As a result of that, the robot started reasoning about the resulting homotopy classes, i.e., trajectories that pass this obstacle on the top (left side) and about trajectories that pass this obstacle on the bottom (right side). The robot preferred to pass the obstacle on the bottom since this trajectory has higher likelihood. The third and the fourth figure show the robot pursuing the selected trajectory moving to the target position. The gray lines show for each time step the trajectory driven by the robot so far.

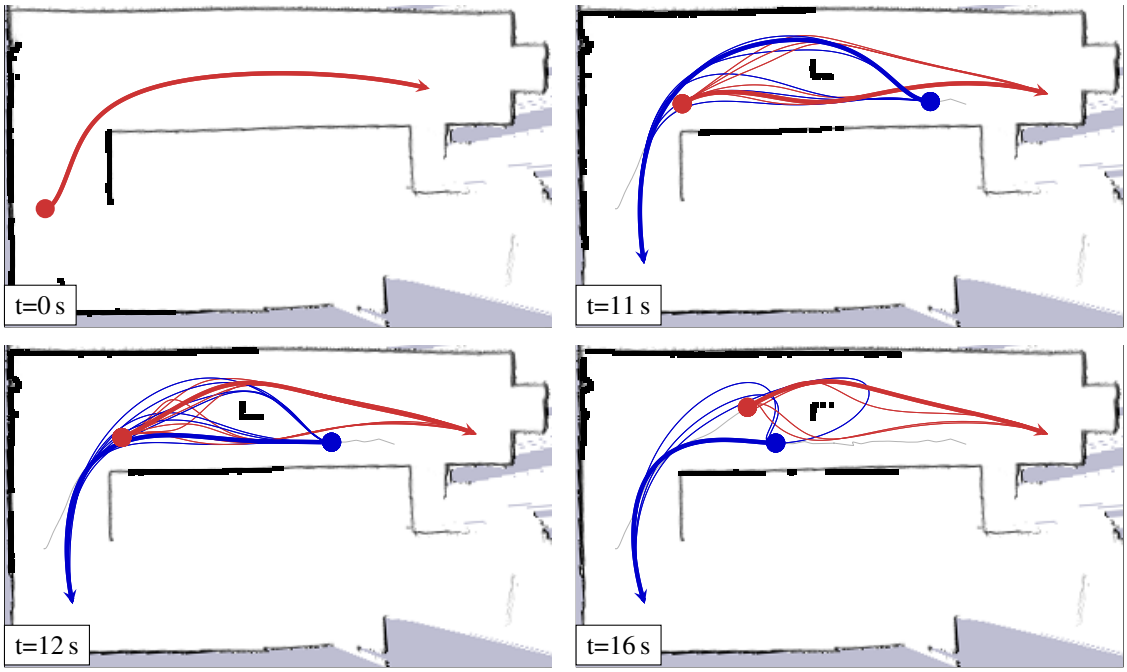


Figure 5.12: Example interaction of the robot (red) with a pedestrian (blue) in the presence of a static obstacle in the corridor. In the top left image, the robot predicts for the pedestrian to evade above the obstacle (thick blue line). However, the pedestrian insists on passing on the other side. Thus, the robot reevaluates the situation, and changes its plan to pass the obstacle on the left side, as illustrated in the two bottom images.

5.4.2 Robot Navigation in the Presence of Unmapped Static Obstacles and Humans

Fig. 5.11 visualizes a second experiment, where the robot navigated through the same environment. In the third image, however, the robot suddenly encountered a pedestrian moving in the opposite direction. Assuming cooperative behavior, the robot started reasoning about composite trajectories comprising itself and the pedestrian. In other words, the robot jointly reasoned about its own trajectory and the trajectory of the pedestrian. As can be seen in the third and fourth image, the robot concluded that the pedestrian most likely passes the static obstacle on the top, since this behavior has the highest likelihood. As a result, the robot choose to pursue its original plan and passed the obstacle on the bottom. In the fourth image the robot has lost track of the pedestrian since it was occluded by the static obstacle. The gray line at the right side of the picture corresponds to the pedestrian's trajectory according to the perception of the robot.

Fig. 5.12 visualizes a third experiment in the same environment. The robot also encountered the static obstacle and a pedestrian that moved in the opposite direction. In the second image, the robot assigned highest likelihood to the homotopy class in which the pedestrian passed the obstacle on the left, similar to the previous experiment. However, in this experiment the pedestrian insisted on passing the static obstacle on the right, which does not match the robot's prediction, as illustrated in the third figure. Since the robot

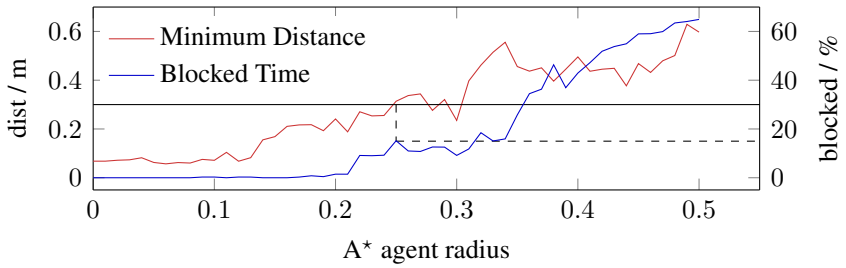


Figure 5.13: Results of the A⋆ planning method. The left axis corresponds to the minimum distance between two agents in all runs and the black solid line indicates the safety distance of $0.3\,\mathrm{m/s}$ that our method was able to satisfy. At the point where the radius setting is high enough to allow the A⋆ planner to satisfy this distance, the robot has to stop in approximately 15% of the time as indicated by the dashed line.

constantly updated the probability distribution to the current state of the environment, it was able to adapt the prediction of the cooperative behavior. As a consequence, the robot changed its plan, decided to give way to the pedestrian and passed the obstacle on the left.

5.4.3 Comparison to a Traditional Path Planner

We conducted a set of experiments in which we evaluated the performance of our method in comparison to previous approaches. As a representative of traditional path planning, we implemented an A⋆ path planner in configuration-time space that uses a constant velocity prediction of the motion of the pedestrians. The A⋆ cost function comprises penalties for proximity of obstacles and the length of the path. We set the parameters of both the A⋆ planner as well as of our method to a desired velocity of the robot of $0.5\mathrm{m/s}$. Whenever the path planner could not compute a valid path to the target position at any point in time, we stopped the robot. In addition we stopped the robot whenever the distance between the robot and the human fell below a safety distance.

Comparison in simulation

In a first experiment we acquired data of realistic evasive movements of two pedestrians that evaded the wheelchair robot in a corridor. To this end, we manually steered the wheelchair with a maximum velocity of $0.5\,\mathrm{m/s}$ ten times in an encounter with two pedestrians. Subsequently, we used these recorded trajectories for a comparison of our method and the A⋆ path planner. To allow for a fair comparison, we fixed the recorded trajectories of the pedestrians, and used the method proposed in this chapter as well as the A⋆ planner to control the robot in simulation, where we set the target position of the robot and the pedestrians to the last position observed in the original recordings. The minimal distance during the manual runs was $0.3\,\mathrm{m}$, which we in the following consider as the safety distance the robot should always respect.

Our method was able to successfully navigate the simulated wheelchair to the target position in all runs. At each point in time during the simulations, our method was able to compute a path to the target position by modeling cooperative behavior of the pedestrians. Furthermore, the distance between the robot and the pedestrians never fell below the

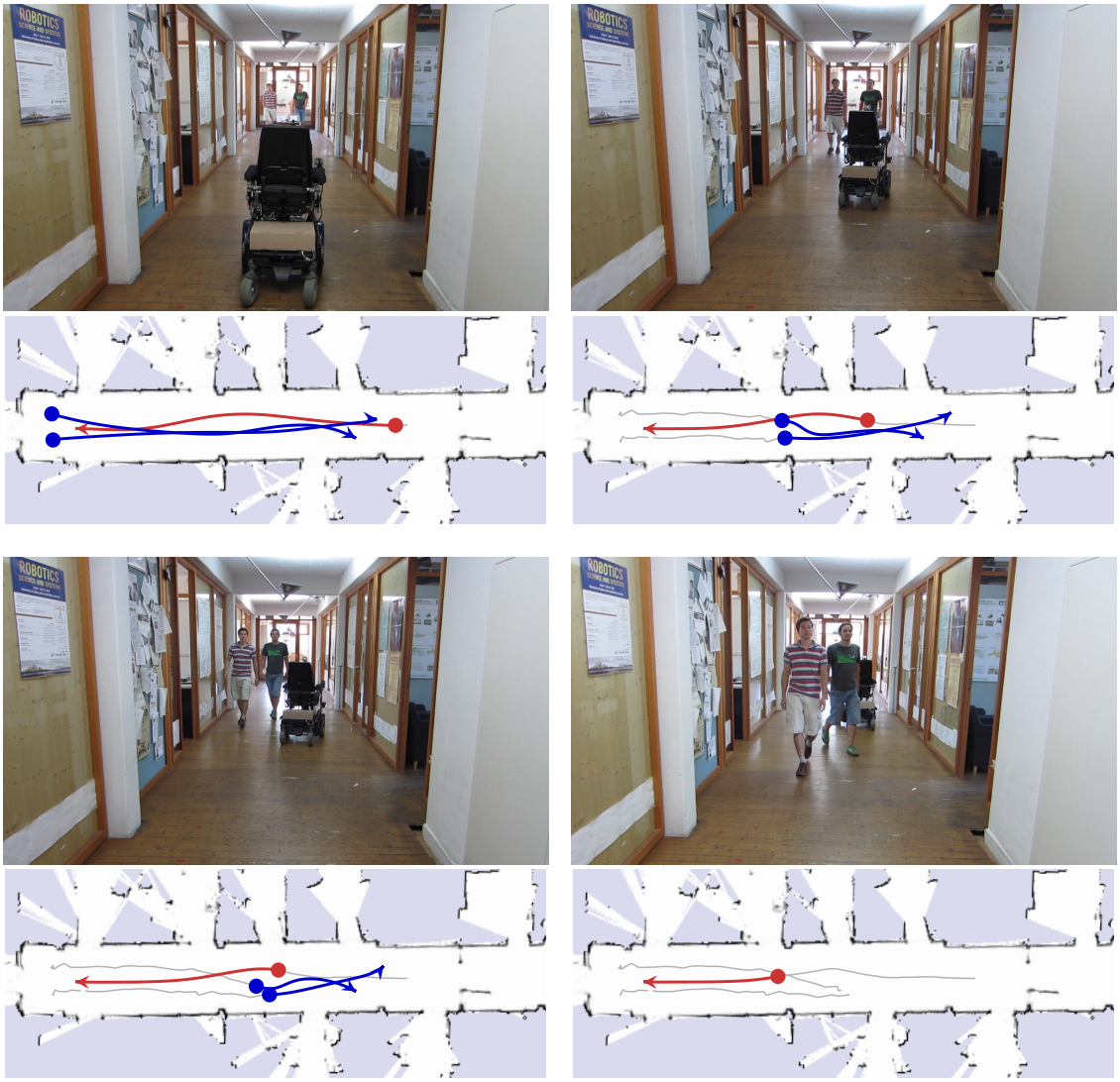


Figure 5.14: Autonomous mobile robot navigation where a robotic wheelchair passes two pedestrians in a hallway using the proposed socially compliant navigation approach. The bottom images depict the driven trajectories (gray) and the interaction of the robot (red) with the pedestrians (blue) predicted by the robot. At first, the pedestrians block the hallway such that a traditional path planning algorithm would be unable to find a path to the target position. In contrast, our method expects the pedestrians to cooperatively engage in joint collision avoidance and is able to find a path to the target position.

safety distance of $0.3\,\mathrm{m}$. As a result, the robot was never blocked, i.e., our method never stopped the robot during navigation and was able to smoothly evade the pedestrians.

In contrast, Fig. 5.13 shows the results of the A* planning method. The x-axis of the plot corresponds to different settings of the radius of the humans and the robot. The A* planner returns a path only if it is guaranteed to be collision free according to the agents' radius along the whole path. For the prediction of the movement of the pedestrians we used a constant velocity model. In this experiment we predicted the pedestrians movement over a time horizon of $10\,\mathrm{s}$. Fig. 5.13 shows that for low values of the radius the robot

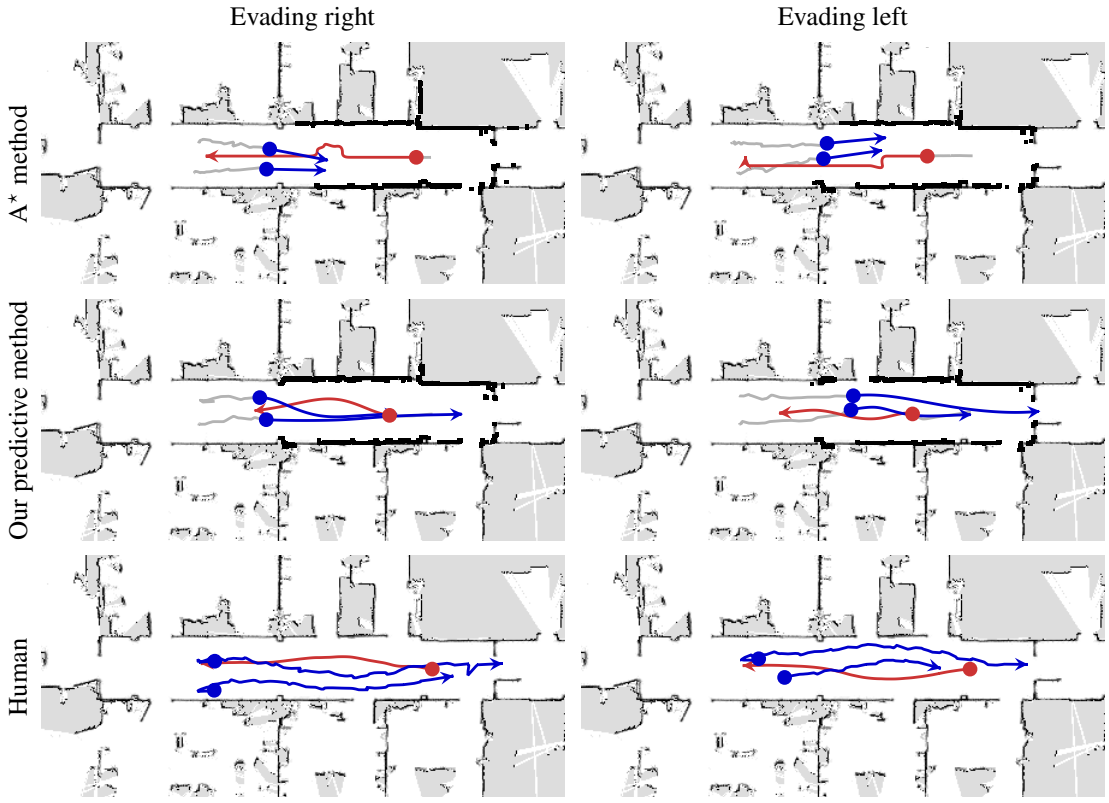


Figure 5.15: This figures show the belief during planning in different situations for different methods. The top row shows the dynamic A* planner using a constant velocity prediction. The middle row shows our method, which predicts the cooperative behavior. The bottom row shows how a human driver resolves the situation.

was never blocked, i.e., the planner was always able to compute a valid path to the target position. However, as a result the minimum distance along the path was also very low, since the low value for the radius allowed the A* planner to compute trajectories that have a low clearance to obstacles. At the point where we set the radius high enough to always satisfy the safety distance of $0.3\,\mathrm{m}$ in all runs, the robot was blocked in approximately 15% of the time. In practice, this means that the autonomous wheelchair would often suddenly stop during execution, which increases the time to reach the target and, more importantly, leads to an uncomfortable navigation behavior. This problem is referred to as the 'freezing robot problem' by Trautman and Krause [141].

This experiment demonstrates the shortcomings of A*-like path planners when navigating in the presence of humans. These approaches rely on computing a path to the target position, which may be blocked by pedestrians at any point in time. In contrast, the approach presented in this chapter predicts cooperative behavior of pedestrians and is therefore able to navigate a robot efficiently and in a socially compliant way in the presence of humans. In this experiment, we used fixed trajectories of the pedestrians to provide the exact same conditions for both methods. However, this does not evaluate the full interactive behavior of the methods. Humans react to the behavior of the robot,

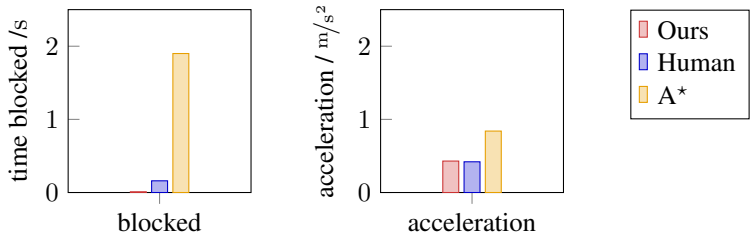


Figure 5.16: Comparison of our approach to socially compliant mobile robot navigation, a traditional path planner, and manual steering. Our method is blocked considerably less compared to the A* planner, similar to a human wheelchair user. This also results in smoother trajectories with less acceleration.

therefore only real robot experiments, which we present in the following, can evaluate the interaction behavior.

Comparison with a real robot

To compare our method in a realistic encounter with humans, we conducted similar experiments as described above in a hallway, this time with a real robot and real interaction with humans. The two pedestrians as well as the robot always started at the same positions in ten runs for each method. Fig. 5.14 depicts four successive time steps and the predictions of our method during such an encounter. First, the pedestrians walked side by side, blocking the corridor. Our method expected the humans to cooperatively engage in cooperative collision avoidance. During the encounter, the robot repeatedly computed the most likely cooperative interaction with the pedestrians, which allowed the wheelchair to engage in a joint collision avoidance.

As a comparison, we also applied the A* planner to control the robotic wheelchair in these encounters. Fig. 5.15 shows the planned path of our method, the A* planner, and the observed path when manually steering the wheelchair, in comparable situations. The figure shows one instance where each method choose to evade on the right side, and one on the left side, respectively. The top row shows the prediction and the planned path for the A* method. It shows that the A* method approached the pedestrians until the safety distance has been reached, and started only then to evade the pedestrian. In contrast, our method, depicted in the middle row, predictively computed a trajectory that lead the robot smoothly around the pedestrians. As a result, the robot started moving to one side at an early stage of the encounter, avoiding sudden and jerky evasive movements. Furthermore, the figure shows the prediction of the pedestrians' behavior. Our method predicted that the pedestrians cooperatively evade the robot, which we can also observe from real pedestrians, as shown in the bottom row.

Fig. 5.16 shows quantitative results averaged over the ten runs for both methods, and for the observed behavior of pedestrians, respectively. It shows that the A* method stopped on average $2\,\mathrm{s}$ in each run since the planner was not able to compute a collision free path to the target position. In contrast, a human steering the wheelchair as well as our method were able to proceed to the goal position smoothly in all runs. This is possible due to their predictive abilities which allows the wheelchair to evade the oncoming pedestrians

early in the encounter. The smooth behavior without stopping also reflects in the average acceleration along the trajectories, which is an indicator for the comfort of the navigation behavior. This experiment illustrates the advantages of our method over traditional path planning methods in navigating a mobile robot in the presence of humans.

5.5 Related Work

Understanding the natural navigation behavior of pedestrians is of interest in psychological research, but also in a variety of other fields such as animation, evacuation studies, and robotics. Therefore, many authors proposed different approaches to model the typical movements of pedestrians [13, 26, 49, 154, 155], which can be classified into steering and optimization models [157].

Steering models regard humans as reactive agents that perform actions depending on the current state of their environment. A popular method that falls in this category is the social forces model by Helbing and Molnar [53]. Social forces consist of an attractive force that pulls the agents towards the target position and repulsive forces that push the agents away from obstacles. This models goal-directed behavior of pedestrians who avoid collisions with obstacles and other pedestrians. To fit the shape of the forces to real behavior of pedestrians, Helbing and Johansson [52] and Johansson et al. [65] propose machine learning methods to learn their parameters from observations. Also learning from observations, Lerner et al. [95] build a database from observed navigation behavior, and select the nearest neighbor in this database to predict the navigation behavior of interacting pedestrians. Hall [49] introduced the concept of proxemics, which defines distances up to which people comfortably approach each other during an interaction. However, Luber et al. [99] present experiments in which they show that during walking pedestrians regularly intrude into the personal space of others. To obtain a more accurate model of human interaction, they learn motion prototypes from observations of pairwise interactions. Similar to the methods we present in this thesis, they use their model of human-human interaction for socially-aware robot navigation. Other authors explicitly investigated the interaction behavior between robots and pedestrians. Pacchierotti et al. [114] describe an experiment in which pedestrians repeatedly encounter a mobile robot in a hallway, whereas Müller et al. [108] present an approach in which a robot makes use of nearby pedestrians during navigation. They propose to have the robot select a pedestrian that walks in the desired direction and to follow this person. In contrast to these methods, we model humans and robots as utility-optimizing agents that prefer trajectories with lower cost.

Other authors proposed different cost functions that capture human-like navigation behavior. Hoogendoorn and Bovy [57] assume agents that minimize the walking discomfort, which they model as a linear combination of accelerations and the distance to obstacles. Pham et al. [116] propose to maximize the smoothness of the trajectories, and Arechavaleta et al. [2] penalize the first derivative of the curvature. Mombaur et al. [104] use optimization techniques to have a humanoid robot imitate human navigation behavior. Similarly, to enable human-like navigation of a mobile robot, Kirby et al. [72] cast social

conventions such as personal space, and a bias for evading pedestrians on the right, as constraints in a global path planner. In our model, we use features that capture important properties of natural navigation behavior, and optimize trajectories with respect to a cost function that consists of a linear combination of these features. In contrast to the above-mentioned approaches, we additionally model the behavior of multiple agents. This allows a robot to capture natural interactions between pedestrians, and between pedestrians and the robot itself.

Especially when navigating through populated areas it is crucial for a mobile robot to assume cooperative behavior of the pedestrians to some extent. If the robot considers pedestrians as obstacles that will not yield, it may not be able to plan a path through these environments, which is referred to as the freezing robot problem by Trautman and Krause [141]. In their work, they develop interactive Gaussian processes to allow a robot to engage in joint collision avoidance, i.e., to plan paths by expecting other agents to behave cooperatively to some extent. Social forces are also able to model cooperative navigation behavior, since the modeled agents all follow the same rules. Similarly, van den Berg et al. [143] present reciprocal velocity obstacles (RVO), which is a set of velocities that potentially lead the agents into collision. If all of the agents select a velocity outside of their reciprocal velocity obstacles, this approach guarantees collision free navigation of all agents. This algorithm assumes that all agents take over half of the effort of evading each other. Guy et al. [48] extend RVO to achieve a more human-like behavior.

Knepper and Rus [76] also present a method for cooperative collision avoidance. They propose an approach inspired by the psychological aspect of civil inattention. An interaction begins when the robot perceives a possible future collision with an agent such as a pedestrian. The robot then decides whether a person is engaged, i.e., pays attention. If this is the case, the robot makes a visible move to correct its trajectory by about half of the amount required to fully avoid collision with the pedestrian, similar to the RVO approach. Then, the robot sticks to this plan, showing civil inattention to avoid oscillating behavior while each agent reacts to another agent's reaction. We propose to solve this problem by predicting the cooperative behavior of all agents involved in the navigation task.

A further class of algorithms for predicting the navigation behavior of agents focuses on learning a spacial distribution over the typical paths that the agents follow. Bennewitz et al. [10] use expectation maximization methods to cluster trajectories of pedestrians that were observed beforehand with a laser scanner. Similarly, Hu et al. [58] hierarchically cluster trajectories of a visual tracking system. Vasquez Govea et al. [144] use growing hidden Markov models to learn and update a distribution over trajectories based on the construction of a topological map. Ikeda et al. [61] propose an algorithm to estimate the position of subgoals that best describe observed trajectories. In contrast to these approaches, we propose a behavior model that is independent of a fixed map but generalizes to different environments.

The optimization of trajectories that we perform is similar to the elastic bands approach, which was introduced by Quinlan and Khatib [123] and also considers continuous trajectories to the goal position. Elastic bands aim to compute collision free, smooth paths by gradually deforming an initially coarse path to the goal. Fraichard and Delsart [42] extend

elastic bands to deform the entire trajectory in the configuration-time space, similar to our optimization approach, although they assume constant velocity models for all obstacles instead of predicting the cooperative behavior of all agents.

For mobile robot navigation we utilize our novel model of interactive navigation behavior. In particular, we repeatedly compute the most likely trajectory in each homotopy class using gradient-based optimization. Many authors presented approaches to find optimal trajectories with respect to a given cost function in the context of path planning for mobile robots. For instance, Rios-Martinez et al. [129] minimize the risk of collision and the risk of disturbing nearby pedestrians using rapidly exploring random trees [94]. Sprunk et al. [136] use a spline-based representation of the trajectories and optimize the corresponding control points to find time-optimized, curvature continuous trajectories that obey acceleration and velocity constraints. Similarly, Gulati et al. [47] optimize trajectories for an assistive mobile robot with respect to user comfort. Ratliff et al. [125] present a general framework for trajectory optimization, which they apply to high-dimensional motion planning for robots. It is well-known that such gradient-based optimization methods often fail to find globally optimal solutions since they are prone to get stuck in local minima. Kalakrishnan et al. [66] propose to use stochastic trajectory optimization to overcome these local minima. However, large state spaces due to complex settings make it infeasible to efficiently find globally optimal solutions by uniformly sampling trajectories. In contrast to that, our model explores the state space by simultaneously searching regions that belong to different homotopy classes, which often correspond to local minima.

5.6 Conclusion

In this chapter we presented a novel approach that allows a mobile robot to navigate in the presence of pedestrians in a socially compliant way. The key aspect of our approach is that we capture the cooperative behavior of all agents involved in the navigation task. We unify prediction and planning by computing at the same time the most likely behavior of nearby pedestrians, and a corresponding plan for the robot. In addition, we compute trajectories to a distant goal point, which allows the robot to adapt its behavior early in a predictive manner. This is in contrast to existing reactive methods for mobile robot navigation and avoids sudden and jerky evasive movements. Our method relies on a model of the interactive behavior of multiple agents that yields a probability distribution over composite trajectories. An important part of this model is that it captures discrete navigation decisions that correspond to homotopy classes of the environment. Therefore, we leverage our approach for online computation of homotopically distinct paths that we presented in the previous chapter. Our approach assumes cooperative agents, i.e., the agents behave in a way that allows all involved agents to reach their target as comfortably as possible. For example in evacuation scenarios, this assumption would fail. In this case, it would be necessary to introduce game-theoretic aspects, where each agents tries to increase its own utility. However, our experiments suggest that the assumption of cooperative agents is reasonable for navigation under normal circumstances, such as navigating in an office environment. We furthermore conducted experiments that

compare our method to traditional path planning methods that show the advantage of considering cooperative behavior when planning in populated environments. In the following chapters, we present further applications of our model of natural navigation behavior of cooperatively navigating agents.

Chapter 6

Teaching Mobile Robots by Demonstration

Mobile service robots are envisioned to operate in environments that are populated by humans and therefore ought to navigate in a socially compliant way. Since the desired behavior of the robots highly depends on the application, we need flexible means for teaching a robot a particular navigation policy. We present an approach that allows a mobile robot to learn how to navigate in the presence of humans while it is being tele-operated in its designated environment. Furthermore, our approach enables a robot to learn a model of human navigation behavior by observing pedestrians. Our method uses the feature-based model of the interactive navigation behavior of multiple agents that we presented in the previous chapter. This model maintains a probability distribution over the trajectories of all the agents that allows the robot to cooperatively avoid collisions with humans. We use inverse reinforcement learning techniques to learn model parameters that capture the observed trajectories, which allows the robot to imitate demonstrated behavior. In an experimental section we present a Turing test comparing how human-like the trajectories of different methods appear. Furthermore, we present experiments in which we teach a robot by tele-operation.

6.1 Introduction

In the previous chapter, we described a model of the interactive navigation behavior of multiple agents and its applications to mobile robot navigation. In this model, feature weights parameterize the behavior of each agent and their interaction. We need to adjust these feature weights to achieve accurate predictions of the pedestrian behavior on the one hand, and to generate a desired behavior for the robot on the other hand. However, the behavior of nearby pedestrians and also the desired navigation behavior of a robot highly depends on the application at hand. For example, a cleaning robot should be unobtrusive and not unnecessarily hinder people, whereas a transportation robot that supplies an

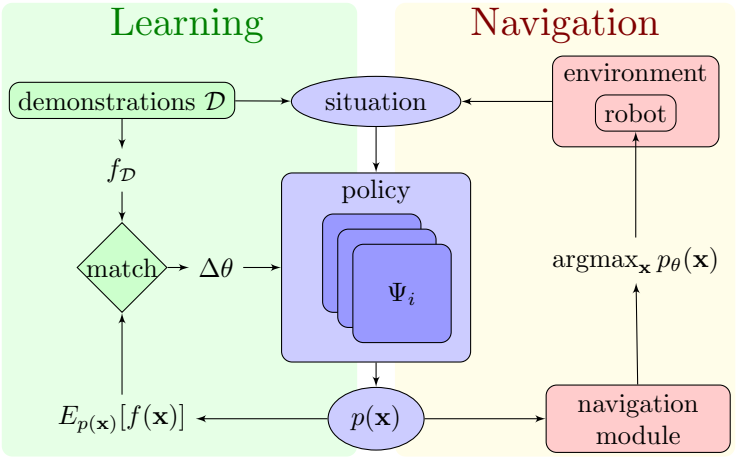


Figure 6.1: Overview of the proposed method. The left side shows the learning process. The inverse reinforcement learning method iteratively matches the empirical feature values from a set of demonstrations $\mathcal{D}$ with expected feature values from the model. After convergence, the policy yields feature values similar to the demonstrated behavior. A mobile robot can use such a model during navigation. It therefore continuously maintains a probability distribution over trajectories and selects the most likely trajectory for navigation. In this way, the robot replicates the learned behavior.

emergency room in a hospital must not delay its task by being overly cautious.

As an alternative to manual tuning of the model parameters for each application, we propose a learning by demonstration approach. Learning by demonstrations gives us flexible means to teach a robot how to navigate in its particular application, especially for non-experts. In particular, our method allows a mobile robot to learn an appropriate navigation policy while being tele-operated in its designated environment. During the teaching phase, the robot learns the desired behavior in terms of the features that we proposed in the previous chapter. Therefore, the robot learns physical properties such as accelerations, velocities and distances to obstacles, as well as topological properties such as a direction bias when evading others.

In addition to observing its own behavior, the robot can also observe nearby humans. This includes the typical navigation behavior of individual pedestrians, but also the interaction behavior between pedestrians, and their reaction to the robot itself. From these observations, our learning method allows the robot to learn the feature weights that best capture the natural navigation behavior of the pedestrians in the dedicated environment of the robot. As we highlighted in the previous chapter, a mobile robot can utilize such a model to predict the behavior of pedestrians and to react accordingly for a smooth, socially compliant navigation behavior.

Our approach applies inverse reinforcement learning (IRL) to learn the feature weights of the proposed model from demonstrations. In general, inverse reinforcement learning means finding a cost function that explains observed behavior. The model we presented in the previous chapter represents interactive behavior of humans and robots in terms of a probability distribution over their trajectories. In addition, the model relies on features that capture important properties of the behavior. This model allows us to compute expected feature values of the distribution over trajectories in a given situation. Furthermore,

the observed behavior yields empirical feature values, i.e., the feature values from the observed trajectories. During learning, our goal is to adapt the model parameters in a way such that the expected feature values match the empirical feature values. In other words, we find the model which explains the observations in a sense that the observations could be samples drawn from this distribution. Fig. 6.1 gives an overview of the proposed approach. The left hand side shows the feature matching that results in a policy that captures the observed behavior. A mobile robot can then use this policy for navigation by computing the most likely interaction in each situation, as discussed in the previous chapter.

The approach we introduced in the previous chapter models the distributions over homotopy classes, as well as the distributions over composite trajectories in terms of exponential family distributions. This form of a distribution is also the direct consequence of feature matching with a maximum entropy assumption. Assuming maximum entropy means that we want to find the 'most general' distribution that matches the features, without implying any further assumptions. In the following, we outline how we can derive the resulting distribution from first principles, and how we can learn the parameters using gradient based optimization.

6.2 The Principle of Maximum Entropy and Feature Matching

In general, the problem of learning from demonstration is to find a model that explains the observed demonstrations and that generalizes to new situations. We model the behavior of the observed agents in terms of probability distributions, therefore learning translates to finding the distribution from which the observed samples $\mathbf{x}_k \in \mathcal{D}$ are drawn. We capture the relevant properties of the behavior in terms of a feature vector

$$\mathbf{f} : \mathcal{X} \to \mathbb{R}^n \tag{6.1}$$

that maps states $\mathbf{x} \in \mathcal{X}$ to a vector of real values. This allows us to compute empirical feature values $\mathbf{f}_\mathcal{D}$ of the demonstrations

$$\mathbf{f}_\mathcal{D} = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x}_k \in \mathcal{D}} \mathbf{f}(\mathbf{x}_k), \tag{6.2}$$

that encode the properties of the observed behavior we want to learn. Following Abbeel and Ng [1], we aim to find the distribution $p(\mathbf{x})$ that matches these empirical feature values in expectation:

$$\mathbb{E}_{p(\mathbf{x})}[\mathbf{f}(\mathbf{x})] = \mathbf{f}_\mathcal{D}. \tag{6.3}$$

In general, however, there is not a unique distribution that matches the features. Ziebart et al. [159] resolve this ambiguity by applying the principle of maximum entropy [64], which states that the distribution with the highest entropy represents the given information best since it does not favor any particular outcome besides the observed constraints.

Following Ziebart et al. [159], we are interested in the distribution that matches the feature expectations, as given in Eq. (6.3), without implying any further assumptions. In this section, we outline how to apply their approach to continuous spaces. The principle of maximum entropy states that the desired distribution maximizes the differential entropy

$$\underset{p}{\operatorname{argmax}}\, H(p) = \underset{p}{\operatorname{argmax}} \int_{\mathbf{x}} -p(\mathbf{x}) \log p(\mathbf{x}) d\mathbf{x}, \tag{6.4}$$

subject to the constraint

$$\int_{\mathbf{x}} p(\mathbf{x})\, d\mathbf{x} = 1 \tag{6.5}$$

that enforces the distribution to integrate to one, and subject to the constraints

$$\forall i\; f_{i\mathcal{D}} = \mathbb{E}_{p(\mathbf{x})}[f_i(\mathbf{x})] = \int_{\mathbf{x}} p(\mathbf{x}) f_i(\mathbf{x}) d\mathbf{x} \tag{6.6}$$

that enforce the empirical features values to match the expected feature values for all features f_i. Introducing Lagrangian multipliers α and θ_i for these constraints yields the maximization problem

$$p^\star, \alpha^\star, \boldsymbol{\theta}^\star = \underset{p,\alpha,\boldsymbol{\theta}}{\operatorname{argmax}}\, h(p, \alpha, \boldsymbol{\theta}), \tag{6.7}$$

where

$$\begin{aligned} h(p, \alpha, \boldsymbol{\theta}) = \int_{\mathbf{x}} & -p(\mathbf{x}) \log p(\mathbf{x})\, d\mathbf{x} \\ & - \alpha \left(\int_{\mathbf{x}} p(\mathbf{x})\, d\mathbf{x} - 1 \right) - \sum_i \theta_i \left(\int_{\mathbf{x}} p(\mathbf{x}) f_i(\mathbf{x})\, d\mathbf{x} - f_{i\mathcal{D}} \right). \end{aligned} \tag{6.8}$$

Applying the Euler-Lagrange equation from the calculus of variations [40] to Eq. (6.8) implies that the probability distribution $p^\star(\mathbf{x})$ has the structure

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{Z(\boldsymbol{\theta})} \exp -\boldsymbol{\theta}^T \mathbf{f}(\mathbf{x}), \tag{6.9}$$

where $Z(\boldsymbol{\theta})$ is a normalization factor to satisfy Eq. (6.5). Thus, the structure of the distribution that maximizes entropy under the constraint of feature matching depends only on the features. However, the parameter vector $\boldsymbol{\theta}^\star$ depends on the training samples $\mathbf{x}_k \in \mathcal{D}$. Unfortunately, it is not feasible to compute $\boldsymbol{\theta}^\star$ analytically, but we can apply gradient-based optimization techniques to determine $\boldsymbol{\theta}^\star$. The gradient is given by the derivative of Eq. (6.8) with respect to the parameter vector:

$$\frac{\partial}{\partial \boldsymbol{\theta}} h(p, \alpha, \boldsymbol{\theta}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}[\mathbf{f}(\mathbf{x})] - \mathbf{f}_{\mathcal{D}}. \tag{6.10}$$

There is also a different point of view that leads to the same result. If we assume an exponential family distribution, as given in Eq. (6.9), the log-likelihood of the observed behavior $\mathcal{D}$ is given by

$$L_{p_{\boldsymbol{\theta}}}(\mathcal{D}) = \log \frac{1}{Z(\boldsymbol{\theta})} \exp -\boldsymbol{\theta}^T \mathbf{f}_{\mathcal{D}}, \tag{6.11}$$

and its derivative with respect to $\boldsymbol{\theta}$ is given by

$$\frac{\partial}{\partial \boldsymbol{\theta}} L_{p_{\boldsymbol{\theta}}}(\mathcal{D}) = \int_{\mathbf{x}} p_{\boldsymbol{\theta}}(\mathbf{x}) \mathbf{f}(\mathbf{x}) \, d\mathbf{x} - \mathbf{f}_{\mathcal{D}} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}[\mathbf{f}(\mathbf{x})] - \mathbf{f}_{\mathcal{D}}. \tag{6.12}$$

Consequently, the problem of finding the maximum entropy distribution subject to feature matching is equivalent to maximizing the likelihood of the training data when assuming an exponential family distribution [64].

To summarize, our goal is to find the distribution that matches, in expectation, the feature values of a set of demonstrations. By applying the principle of maximum entropy, it follows that this distribution is an exponential family distribution. Therefore, finding the desired distribution translates to computing the parameter vector $\boldsymbol{\theta}^\star$ that leads to feature matching. Computing $\boldsymbol{\theta}^\star$ analytically is not feasible, but we can compute the gradient with respect to these parameters and, consequently, apply gradient-based optimization.

We apply IRL to the model of interactive navigation behavior that we described in the previous chapter. This model represents a two-staged process in which the possible outcomes in each stage underly an exponential family distribution. The first stage represents a distribution over homotopy classes, whereas the second stage represents a distribution over composite trajectories. To learn the shape of the distribution in both stages, we apply IRL.

First, we compute the empirical feature values $\tilde{\mathbf{f}}$ for all features that capture continuous behavior (Sec. 5.1.4). For the observed situations, i.e., for the same environment and for the same start and end positions, we compute the expected feature values $\mathbb{E}_{p^{\psi}_{\boldsymbol{\theta}}(\mathbf{x})}[\mathbf{f}]$ within the homotopy class of the corresponding observation. Here, we average over all training examples, assuming identical continuous behavior over all homotopy classes. Given the empirical and the expected feature values, Eq. (6.12) computes the gradient for the feature weights $\boldsymbol{\theta}$, which we use for optimization. This IRL optimization eventually converges to the feature weights $\boldsymbol{\theta}^\star$ that explain the demonstrations on a continuous level. The result of this first step are the feature weights that shape the distributions $p^{\psi}_{\boldsymbol{\theta}^\star}(\mathbf{x})$ over composite trajectories.

Subsequently, we learn the feature weights for the features $\mathbf{g}$ that affect the discrete decisions (Sec. 5.1.6). Given the distributions $p^{\psi}_{\boldsymbol{\theta}^\star}(\mathbf{x})$, we can compute the empirical feature values $\tilde{\mathbf{g}}$ as well as the expected feature values $\mathbb{E}_{\mathrm{p}_{\boldsymbol{\Theta}}(\psi)}[\mathbf{g}]$. This allows us to apply IRL similar to the procedure described above, which results in the feature weights $\boldsymbol{\Theta}^\star$ that complete the model.

6.3 Computing Feature Expectations

The process described above requires repeated computation of the feature expectations $\mathbb{E}_{p^{\psi}_{\boldsymbol{\theta}}(\mathbf{x})}[\mathbf{f}(\mathbf{x})]$ of the resulting probability distribution $p^{\psi}_{\boldsymbol{\theta}}(\mathbf{x})$. In general, however, inference for distributions over continuous trajectories is not analytically tractable. Monte Carlo sampling methods provide means to approximate the expectations using a set of sample trajectories drawn from the distribution. In particular, Markov chain Monte Carlo (MCMC) methods [12] allow us to obtain samples from high-dimensional distributions. These methods aim to explore the state space by constructing a Markov chain whose equilibrium distribution is the target distribution.

Most notably, the widely-used Metropolis-Hastings algorithm [51] generates a Markov chain in the state space using a proposal distribution and a criterion to accept or reject the proposed steps. The choice of the proposal distribution strongly influences the acceptance rate and thus the mixing time, which is the number of steps until the samples sufficiently approximate the target distribution. In general, it is difficult to design a proposal distribution that leads to satisfactory mixing. As a result, efficient sampling from complex high-dimensional distributions is often not tractable in practice.

Our approach exploits the structure of the distributions over composite trajectories to enable efficient sampling. First, the navigation behavior of physical agents shapes the trajectories according to certain properties such as smoothness and goal-directed navigation. As a result, the distributions over the composite trajectories of the same homotopy class are highly peaked. Exploiting the gradient of the probability densities allows us to guide the sampling process towards these regions of high probability.

To this end, we use the Hybrid Monte Carlo algorithm [37], which takes into account the gradient of the density to sample from the distributions $p^{\psi}_{\boldsymbol{\theta}}(\mathbf{x})$. The algorithm considers an extended target density $p^{\psi}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{u})$ to simulate a fictitious physical system, in which $\mathbf{u} \in \mathbb{R}^n$ are auxiliary momentum variables. The method constructs a Markov chain by alternating Hamiltonian dynamical updates and updates of the auxiliary variables, utilizing the gradient of the density $p^{\psi}_{\boldsymbol{\theta}}(\mathbf{x})$ with respect to $\mathbf{x}$. After performing a number of these steps, Hybrid Monte Carlo relies on the Metropolis-Hastings algorithm [12] to accept or reject the candidate samples.

In theory, the Markov chain eventually explores the entire state space, independently of the initial state. In practical applications with limited sampling time, however, the initial guess considerably influences the sampling quality. We compute the most likely composite trajectory within the class of each observation as described in Sec. 5.2.2, and use it as a start point for the Markov chain. The regions of smooth, collision-free trajectories are surrounded by regions of low probability, for example where two agents get close to each other. The Markov chain is therefore unlikely to leave the homotopy class ψ of the initial guess. However, to guarantee sampling within the correct homotopy class, we reject samples outside the homotopy class ψ.

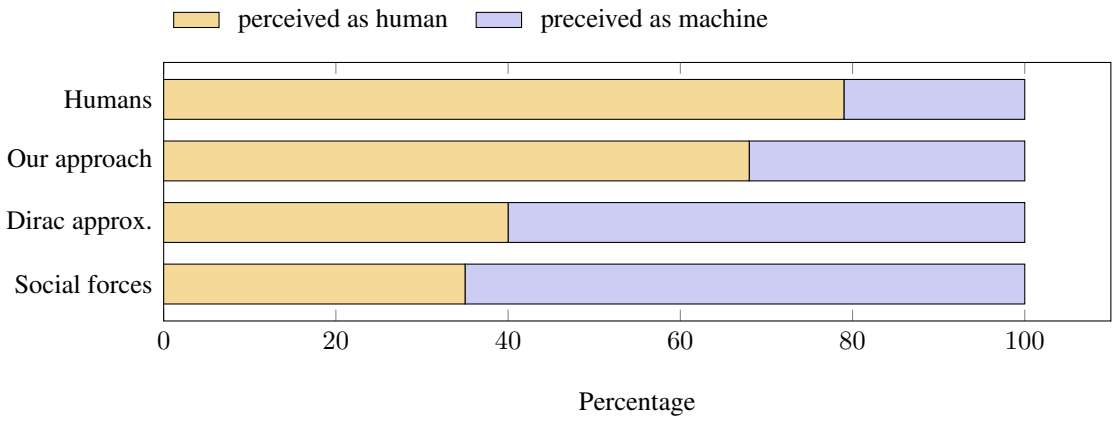


Figure 6.2: Turing test to evaluate whether the behaviors induced by our approach, an IRL approach with Dirac approximation, and the social forces model by Helbing and Molnar [53] qualify as human. The results suggest that the behavior induced by our approach most resembles human behavior.

6.4 Experiments

The learning method presented in this chapter serves two purposes. First, it allows a robot to learn a model of pedestrian navigation behavior by observing their trajectories. Second, it facilitates teaching-by-demonstration, where a robot learns a desired behavior by teleoperating it in the designated environment. In this section, we present two experiments that evaluate both parts. In a first experiment we present a Turing test that evaluates the ability of our method to generate human-like trajectories, compared to previous methods. In a second experiment we taught a real robot different behavior styles by demonstration and applied the learned policy to autonomous navigation.

6.4.1 Learning Pedestrian Navigation Behavior – Turing Test

In a first experiment, we applied our approach to the problem of learning a model of pedestrian behavior. Our goal was to learn a generative model that is able to compute human-like trajectories. The dataset we used for learning comprises interactions of three persons that we recorded in a motion capture system. To distract the persons from the navigation task, we made them read and memorize newspaper articles at different locations that were consecutively numbered. At a signal, they simultaneously walked to the subsequent positions, which repeatedly gave rise to situations in which the participants had to evade each other.

Based on this dataset, we learned a model of natural navigation behavior using the inverse reinforcement learning technique proposed in this chapter. Our learning method aims at minimizing the distance between the observed feature values and the expected feature values of the model. However, reproducing feature values does not sufficiently indicate if we actually capture natural navigation behavior. To evaluate how human-like the resulting trajectories actually appear, we carried out a Turing test in which we compare the trajectories generated by our approach to other methods. A Turing test is an experiment

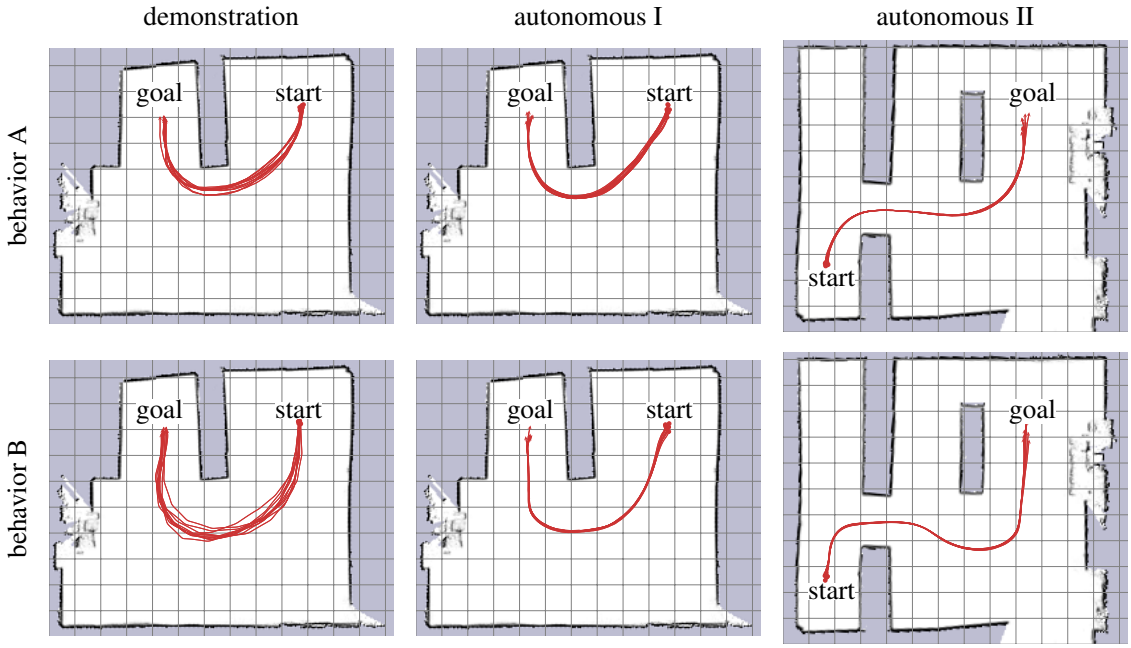


Figure 6.3: Teaching a robot distinct behavior styles by tele-operation. The images on the left show the demonstrated trajectories for each of two different behavior styles. During demonstrating behavior A, we manually steered the wheelchair slowly, but close to obstacles. When demonstrating behavior B, we choose the same start and goal position, but increased the velocity as well as the clearance to obstacles. The underlying grid in the images correspond to one meter in the real environment. The middle column shows the same environment, start and goal position. The depicted trajectories, however, were recorded during an autonomous run of the robot using the learned policies. The right column shows another set of autonomous runs that suggest that the learned model also generalizes to different environments. Each of the images shows trajectories of ten individual runs.

in which humans evaluate the ability of a machine to exhibit human-like behavior, which is non-distinguishable from actual human behavior at best. We conducted a Turing test in which we asked ten human subjects to distinguish recorded human trajectories from trajectories generated by a machine.

For each of the recorded interactions of our dataset, we used different methods to predict the interaction in the same situation. In particular, we predicted the behavior using the learned model described in this chapter, a previous learning method proposed by the author of this thesis [85], and the social forces method [53]. The previous method presented in [85] is an IRL method that computes the feature expectations by means of a Dirac approximation, which is in contrast to the sampling technique proposed in this chapter. In Chap. 2.4 we describe the social forces method in more detail.

During the experiment, we presented animations of 40 runs that were randomly drawn from the set of observations and predicted interactions to each of the participants of the survey. For each run, we asked the participants whether they believe that the interaction was recorded from real humans, or generated by a machine. Fig. 6.2 summarizes the results of the survey. The human subjects correctly identified $79\,\%$ of all the human demonstrations, but they mistook $68\,\%$ of the predictions of our approach, $40\,\%$ of the

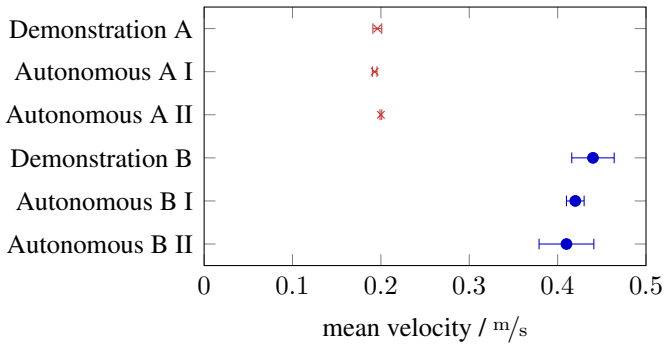


Figure 6.4: Mean velocity averaged over ten runs of the demonstrated and autonomous runs. Red: mean velocity of the demonstrated behavior A, of the autonomous runs in the same environment (A I), and the autonomous runs in a different environment (A II), using the learned policy. Blue: mean velocity of the demonstrations and the autonomous runs for behavior style B. The error bars show the standard deviation over ten runs each.

predictions of the Dirac approximation, and $35\,\%$ of the predictions of the social forces algorithm for human behavior. In summary, the results of this Turing test indicate that the behavior induced by our approach is significantly more human-like than the behavior induced by the other two methods according to a one-sided paired sample t-test at a 95% confidence level.

6.4.2 Teaching a Robot by Tele-Operation

In a second experiment, we used our method to teach a robot two distinct navigation behaviors. To demonstrate the behavior, we manually navigated the wheelchair described in Sec. 5.3. First, we steered the wheelchair close to the obstacles from the start to the goal position at a low velocity, which we refer to as behavior style A in the following. Second, we demonstrated behavior style B, for which we navigated the wheelchair with a higher velocity further away from the obstacles. Fig. 6.3 (left) shows trajectories of the two sets of demonstrations, and Fig. 6.4 shows the mean velocity of both runs.

Based on these demonstrations, we learned two behavior models using the IRL technique described previously in this chapter. During learning, our method adapts the feature weights of the model such that the expected feature values correspond to the observed values. As a result, we have two sets of feature weights, corresponding to the two distinct behavior styles A and B.

We used these two models to autonomously navigate the wheelchair in the same environment to be able to directly compare the resulting trajectories. Fig. 6.3 (middle) shows that the characteristics of the demonstrated behavior also transfers to the autonomous navigation. Whereas in the autonomous runs using model A the distance to the obstacle was about $1\,\mathrm{m}$, model B results in a distance of about $2\,\mathrm{m}$, corresponding to the demonstrations. In addition, our method captured the velocity accurately, as shown in Fig. 6.4.

To demonstrate the ability of the learned models to generalize, we additionally performed experiments in a different environment. Fig. 6.3 (right) shows the distinct behavior

especially when navigating around the second obstacle. However, during the narrow passage near the start position, both models navigate the wheelchair through the middle of the passage, since there is no way around this passage. This shows that our method learns a flexible behavior that adapts to the characteristics of the environment. This is fundamentally different from learning for example a minimal allowed distance, which would not allow the robot to navigate the narrow passage. Also when navigating in this environment, the velocity profiles in Fig. 6.4 suggest that our method is able to imitate characteristics from the demonstrations.

This experiment shows that our method is able to learn important navigation characteristics from demonstration. In particular, this allows for convenient teaching of a mobile robot by tele-operation, which is an interesting alternative to program a robot for non-expert users. The learned models capture the navigation style in the environment in which the behavior was demonstrated, but also generalize to different environments.

6.5 Related Work

There is a wide range of literature on learning policies from demonstrations [3]. To imitate observed behavior, Atkeson and Schaal [5] developed novel approaches to map features over the state space to actions. To allow for learning a more general policy from demonstration, Ng and Russell [109] proposed inverse reinforcement learning (IRL), which aims to find a cost function that explains the observed behavior. Similar to our approach, Abbeel and Ng [1] suggest to model the cost function as a combination of features that capture relevant aspects of the observed behavior. However, in general there are infinitely many cost functions that lead to feature matching. To resolve this ambiguity, Ziebart et al. [159] present maximum entropy IRL that relies on the principle of maximum entropy [64] and, hence, aims at finding the policy with the highest entropy subject to feature matching. Ziebart et al. [159] use a low dimensional, discrete state space to model the observed behavior. In contrast, we learn trajectories in continuous spaces, which allows us to take into account higher-order dynamics such as velocities and accelerations.

The applications of similar learning-from-demonstration approaches include learning pedestrian navigation behavior by Ziebart et al. [160], modeling full body posture by Vernaza and Bagnell [147], and route planning for outdoor mobile robots by Ratliff et al. [124]. In particular, Ziebart et al. [161] use maximum entropy IRL to predict the trajectories of pointing devices such as computer mouses. Furthermore, Kitani et al. [74] propose a method that leverages maximum entropy IRL to learn the preferences of pedestrians using scene features such as sidewalks. Our method is inspired by the maximum entropy IRL approach by Ziebart et al. [159]. However, in addition to the extension to continuous state spaces, we consider the interaction of multiple agents, instead of predicting the trajectory of a single agent. This allows a mobile robot to include the expected reaction of nearby pedestrians into planning.

Computing the feature expectations is important for a variety of IRL methods. Especially in continuous, high-dimensional state spaces this is a challenging problem. Boularias et al. [16] use importance sampling to compute the gradient for model-free IRL.

Other authors propose certain assumptions to overcome this problem. Whereas Kalakrishnan et al. [67] assume locally optimal demonstrations and add Gaussian noise to the model parameters, Vernaza and Bagnell [147] assume a restricted low-dimensional form of the feature functions. A previous work of the author of this thesis, to which we compare the proposed method in the Turing test, uses a Dirac approximation to compute the expected feature values. This approximation does not handle non-optimal demonstrations well. In this chapter, we therefore proposed a sampling method that efficiently estimates feature expectations by Hybrid Monte Carlo sampling [37]. Our method allows arbitrary features for which we can compute the gradient with respect to the trajectory parameters.

6.6 Conclusion

In this chapter, we presented an approach that allows a robot to learn the parameters of its navigation model from demonstration. This is useful for learning models of the natural navigation behavior of pedestrians, which is an important part of socially compliant mobile robot navigation. Furthermore, such a method allows us to teach a robot by tele-operation. Our method relies on maximum entropy inverse reinforcement learning. During learning, the proposed algorithm adapts feature weights to shape a probability distribution over the trajectories of all the agents, i.e., the pedestrians and the robots. As a result, the expected feature values of the learned model match the empirical feature values of the observations. In an experimental evaluation, we conducted a Turing test to compare our learning method to existing approaches. The Turing test suggests that our model captures properties of the navigation behavior that are important for human-like appearance. As outlined in the previous chapter, a robot that is endowed with a model that accurately captures human-like behavior is able to predict the trajectories of pedestrians, and to react in a socially compliant way. Furthermore, we conducted an experiment in which we taught a robot two distinct behavior styles, which also generalize to different environments. Such a method is particularly suited for non-technical users to conveniently adjust the desired behavior of a mobile robot to their needs. Accurate models of the interactive navigation behavior of multiple agents are also the basis of the applications presented in the following chapters.

Chapter 7

Shared Autonomy Navigation

As outlined in the previous chapters, alternative ways to navigate around obstacles typically correspond to different local minima of a navigation cost function. In this chapter we make use of these local minima to quickly compute a set of qualitatively different trajectories. With a cost function that also captures safety and comfort of the trajectories, the different alternatives are valid options for navigation. We propose to utilize this set of trajectories in a shared autonomy navigation system. Using such a system, a user is not required to provide the low level steering commands for a mobile robot, but only a high level selection of one of the alternative trajectories. The low information content of such a decision makes the proposed system also applicable for noisy high level user interfaces such as brain computer interfaces. In experiments, we demonstrate that our approach is able to operate a robotic wheelchair in shared autonomy mode, where a user can quickly switch between topologically different trajectories.

7.1 Introduction

In shared autonomy navigation scenarios, an artificial intelligent system takes over a certain part of the task that is necessary to fulfill the ultimate goal, typically leaving only high level decisions to the human. In this way, tedious or computationally complex parts of the process do not unnecessarily burden the user, who can focus on the meaningful decisions.

Shared autonomy systems already exist in a wide range of areas. Surgical robots, for example, support surgeons by mapping movements of the controller to tool tips that are typically attached to an endoscope. The robot thereby restricts the tool to safe areas, scales the surgeons movement, and suppresses physiological tremor [28]. Shared autonomy has also received attention in mobile manipulation tasks, where a human user chooses a high level task, such as gripping an object, and the robot takes care of the motion planning of the arms [118]. The same concept is applicable to shared autonomy navigation, which is beneficial for complex navigation tasks, such as accurately controlling a quadrotor in demanding scenarios [131]. Furthermore, shared autonomy navigation enables controlling

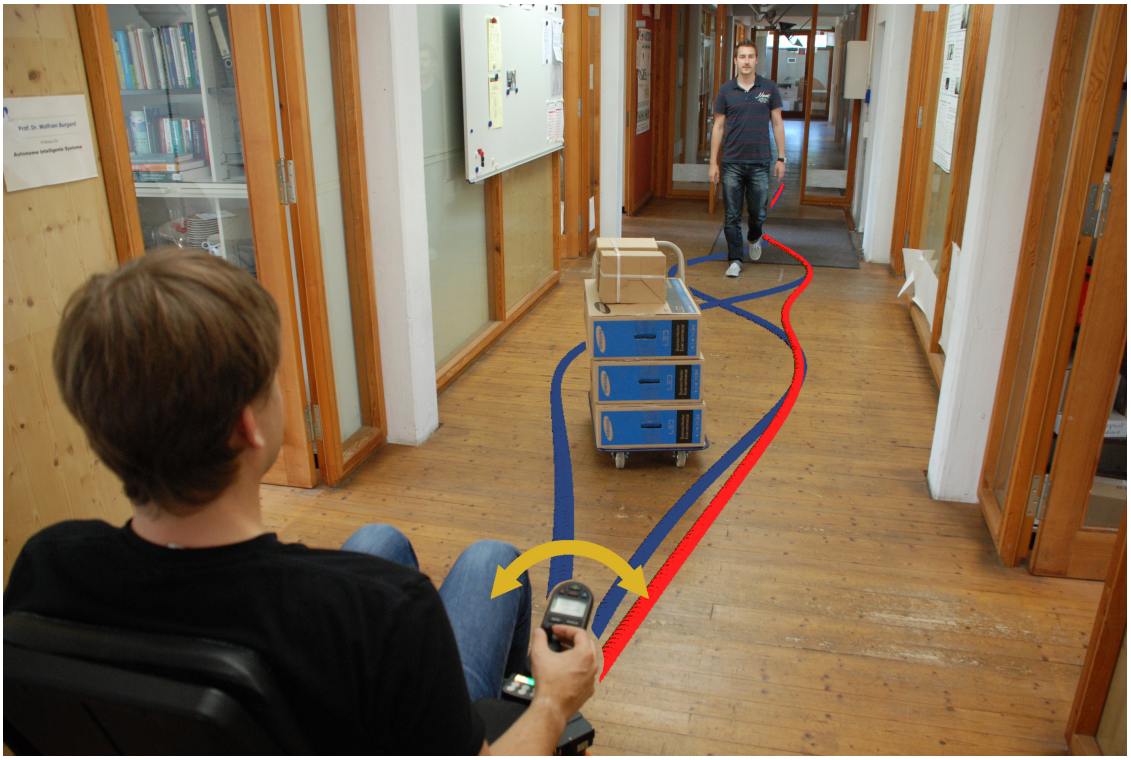


Figure 7.1: Application of our approach in a shared autonomy wheelchair navigation scenario. Our method quickly provides a set of homotopically different trajectories, as illustrated by the projected trajectories in the image. The user can easily bias the selection of the trajectory used for navigation by high-level direction preferences, such as moving the joystick left, or right.

a mobile robot with a slow and noisy interface, such as a brain computer interface, or controlling the robot by tele-operation.

In this chapter, we propose a system for 2D shared autonomy navigation of a mobile robot. In particular, the autonomous system takes care of the low level control by optimizing trajectories with respect to a navigation cost function. The user is only required to choose from a small set of trajectories that correspond to feasible alternatives in the current situation. For this, we utilize the proposed method to compute homotopically distinct navigation paths in combination with parallel trajectory optimization, as introduced in Chap. 5. In the following, we present two different modes of shared autonomy navigation. First, we assume that we know the global goal location. The user is then able to select different paths that lead to this target online during navigation by giving an orientation bias. If there is no goal position known, we propose to provide local paths to nearby subgoals. In each step, the user selects a trajectory from a discrete set, which the robot uses for navigation. From this subgoal, the robot then generates a new set of navigation choices, which gives the user the freedom to navigate the area as desired without the need for low level control commands.

We apply our approach to shared autonomy wheelchair navigation, where the user can influence which trajectory the system follows by providing high-level control input. This enables handicapped users to control the wheelchair despite limited fine-motor skills.

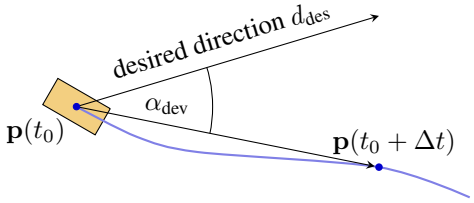


Figure 7.2: This figure shows the computation of the angle α_{dev} that is used as a parameter for the feature modeling the preferred user direction.

Fig. 7.1 illustrates such an application in which the user can bias the trajectory selection by deflecting the joystick left, or right.

7.2 Shared Autonomy Navigation with Known Targets

In Chap. 5 we described a method that allows a mobile robot to compute multiple composite trajectories in parallel, and select the most appropriate for the current situation online. This method maintains a set of composite trajectories that are locally optimized with respect to a navigation cost function. Each of these trajectories corresponds to a specific homotopy class of the environment. The robot is endowed with a feature-based probability distribution over the homotopy classes, where the features capture important properties of socially compliant behavior. For navigation, the robot selects the optimized composite trajectory in the most likely class, with respect to this feature-based distribution.

We realize shared autonomy navigation for mobile robots by adding features that incorporate user preferences online during navigation. The robot follows the trajectory that has lowest cost according to a trade-off between these user preferences and the parts of the cost function that penalize properties such as high velocities or closeness to obstacles. Similar to the approaches presented in previous chapters, the feature weights provide a convenient way to adjust the behavior, i.e., to decide how strongly the robot should follow the user preferences.

Let us assume a wheelchair scenario in which the handicapped user is only capable of issuing high-level commands rather than low-level controls. For example, such a user might want to express navigation preferences by head posture [100], or by joystick deflection. To achieve this with our approach, we introduce a feature f_{dir} that penalizes the deviation α_{dev} of a trajectory from the preferred user direction:

$$f_{\text{dir}}(\mathbf{p}) = \alpha_{\text{dev}}^2 = \arccos^2\left(\frac{d(\mathbf{p}) \cdot d_{\text{des}}}{\|d(\mathbf{p})\|\|d_{\text{des}}\|}\right), \tag{7.1}$$

where $d(\mathbf{p})$ is the direction of the trajectory and d_{des} is the direction selected by the user, as illustrated in Fig. 7.2. To compute $d(\mathbf{p})$ we use the location of trajectory $\mathbf{p}$ at the time Δt in the future, i.e., $d(\mathbf{p}) = \mathbf{p}(t_0 + \Delta t) - \mathbf{p}(t_0)$.We use this feature only for evaluating the costs of the optimized trajectories in the selection process, not for the optimization itself.

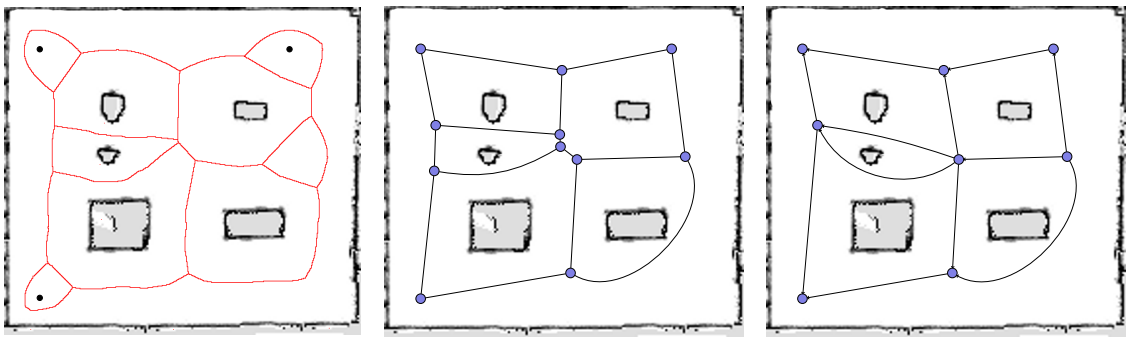


Figure 7.3: Abstract graph computation for local shared autonomy control. Left: Voronoi diagram of an environment with five static obstacles and three additional start- or goal locations. Middle: Abstract graph from the Voronoi Diagram, as described in Sec. 4.2.2. Right: pruned graph used to compute paths for local shared autonomy control.

During navigation, the robot maintains a set of trajectories to the target position, similar to the method we proposed in Sec. 5.2.3. For navigation, the robot selects the most likely trajectory, i.e., the trajectory with the least cost. However, in contrast to the method for fully autonomous socially compliant mobile robot navigation, the user can influence the distribution over the trajectories by means of the new feature introduced above.

7.3 Shared Autonomy Navigation with Local Targets

In the previous section, we assumed that we know the global target of the robot, and let the user bias the trajectory selection towards this target online during navigation. In this section, we present a method that is applicable for scenarios where the goal is not known beforehand, or changes during the navigation task. Since this method presents to the user a fixed set of trajectories at a time, also interfaces that only allow binary selection, such as brain computer interfaces, are applicable. Fig. 7.4 shows examples of a possible user interface. Here, a binary selection could consist of the options *next* and *select*. In addition to scenarios in which the user is located near the robot, or even in the robot such as in an autonomous wheelchair, our method is also suitable for tele-presence applications.

The key idea of this method is that we present to the user a small set of qualitatively different routes in the environment. After selection, the robot moves to the desired subgoal. During the navigation to this position, or after the robot has arrived at the subgoal, the user gets a prompt with a new set of valid trajectories. These routes should have the following properties. First, they should be valid navigation trajectories, i.e., they should be collision free and all the constraints of the robot should be met. Second, they should be 'local' enough to give a user the freedom to steer the robot to wherever he or she chooses.

We found Voronoi diagrams to be a suitable basis to compute such paths since it captures the connectivity of the free space. Therefore, the vertices where two or more Voronoi lines meet encode positions in the environment where there are multiple possible further directions. To compute the different alternatives, we first compute the abstract graph of the environment, as described in Sec. 4.2.2. First, we add as virtual obstacles

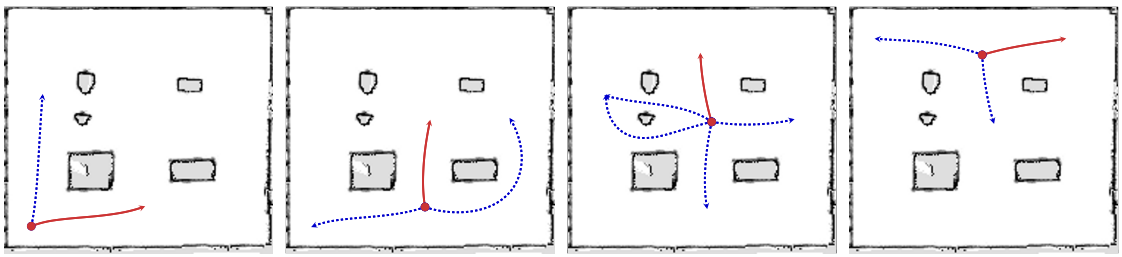


Figure 7.4: Example path through the environment. In each step, the user selects one of the presented trajectories. During the robot moves to this intermediate goal, the next selection appears. Such a system gives a user more freedom compared to only a selection of final goal positions.

the current position of the robot as well as possible additional target positions into the map and then compute the Voronoi diagram. See an example environment with five static obstacles, and three additional positions in Fig. 7.3 (left). In the resulting abstract graph there are often many connecting vertices close to each other, as visible in the center area of Fig. 7.3 (middle). We propose to prune this abstract graph by successively merging vertices that are closer than a threshold. Fig. 7.3 (right) shows the resulting graph in the example after pruning. In this graph, the current position of the robot is one of the vertices by construction. From this vertex, we select the connecting edges and project them back to trajectories in the 2D environment. Then, we optimize each trajectory with respect to our proposed navigation cost function.

In a shared autonomy setting, we present the set of optimized trajectories to the user, who selects one of them using the interface. As soon as the robot receives the selection, it moves along these paths to the next subgoal. During the time the robot moves to this location, it updates the Voronoi Diagram to account for changes in the environment. We also include the current subgoal for the next Voronoi computation, to guarantee that the new position of the robot is a vertex in the abstract graph that will be presented to the user next.

Fig. 7.4 shows an example path from the lower left corner of the environment to the top right corner with three decision steps. The red trajectory corresponds to the selected trajectory, the blue lines show the alternative trajectories the user could select in the respective steps.

7.4 Experimental Evaluation

In this section we evaluate the performance of our method in shared autonomy navigation with a real robot. Therefore, we implemented our method on the autonomous wheelchair described in Sec. 5.3, using the joystick of the wheelchair as user interface.

7.4.1 Shared Autonomy Wheelchair Control

During this experiment, we navigated the autonomous wheelchair in a corridor with obstacles, using the method described in Sec. 7.2 for shared autonomy with known targets.

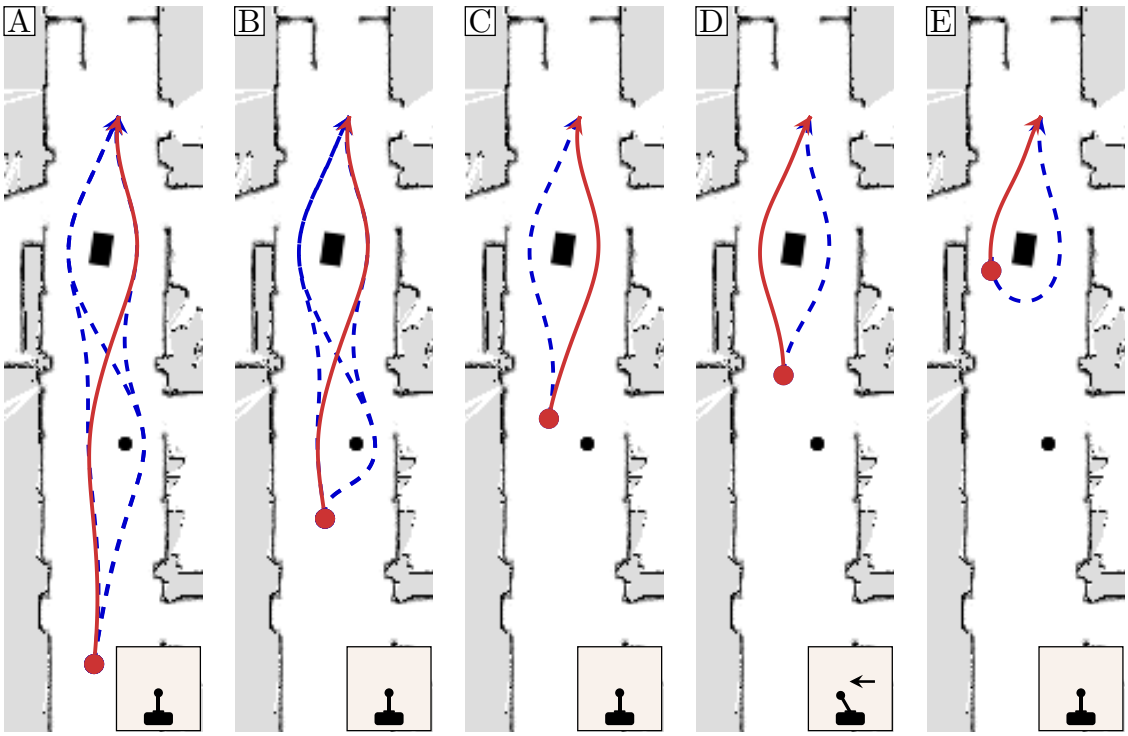


Figure 7.5: Shared autonomy control of a wheelchair in a corridor with two static obstacles. The figure shows the best trajectory according to the cost function, which the wheelchair executes (red) and an additional set of optimized trajectories in different homotopy classes (blue, dashed). The joystick in the lower right indicates the direction preference of the user.

Our method computes trajectories in different homotopy classes and selects the most likely for execution. We allowed the user to bias the system by adding the feature f_{dir} to the cost function. During the experiment, we used the current deflection of the joystick as the direction preference d_{des}.

Fig. 7.5 shows how the wheelchair navigated the corridor in which we placed two additional static obstacles: (A) Our system has computed a set of optimized trajectories in different homotopy classes; (B) Without user preferences, the system selects and follows the red trajectory since it has lowest costs with respect to the cost function that captures the time to reach the target, velocity and acceleration constraints, and distances to obstacles; (C) The wheelchair has discarded the trajectory alternatives that require turning around since their cost has exceeded a threshold. The robot decides to evade the obstacle on the right; (D) In front of the second obstacle, the user deflected the joystick to the left, which biases the costs of the trajectories. As a result, the wheelchair selects the left trajectory for navigation; (E) The wheelchair follows the left trajectory since it has the lowest cost in this situation.

This experiment shows that our method is able to compute different alternatives to the goal position online during navigation. The additional feature allows shared autonomy that biases the choice of the selected trajectory. Such a system could allow a user with fine motoric disabilities to safely navigate a wheelchair.

7.5 Related Work

In this chapter, we utilize path alternatives that are based on the Voronoi diagram of the environment for shared autonomy applications. We discussed related approaches in the context of homotopically distinct paths and the usage of Voronoi diagrams in Sec. 4.5.

Shared autonomy is a suitable approach for tasks in which the manual control of all degrees of freedom would be too complex for a human operator. Sa and Corke [131] developed a shared autonomy method for a visual inspection task. It allows an unskilled operator to steer a quadrotor close to the target object with only high level commands. Okada et al. [112] describe a vehicle that is able to traverse rough terrain by adjusting the robot's frame to the shape of the terrain. Controlling all degrees of freedom is infeasible in practice, therefore, the authors propose a shared autonomy navigation in which the vehicle autonomously controls its frame. Pitzer et al. [118] present an approach towards shared autonomy for robotic mobile manipulation. The system integrates the user into the loop through collaborative object selection, whereas the robot autonomously performs the grasping itself.

Mobile manipulation has also gained increasing interest in health care to support disabled users. Chen et al. [22] describe an assistive mobile robot that performs manipulation tasks together with a quadriplegic patient. To provide high level control, the patient moves a cursor on a 2D screen using a head tracker. The investigated scenarios include fetching items from drawers and even manipulation near to the patient's body such as scratching and shaving. A further assistive manipulation robot was proposed by Soyama et al. [135], who describe a meal-assistance robot that optionally operates in a semi-automatic mode.

In healthcare, semi-autonomous systems are also used to reduce cognitive load on the user. Many authors presented systems to assist wheelchair users. Wang et al. [150] and Nguyen et al. [110] present a shared control scheme for autonomous wheelchairs. Similar to our approach, they use the general direction of the user input, whereas the wheelchair itself takes care of the low level commands. Based on the current information of the laser sensors, the robotic wheelchair adapts the commands to actively avoid collisions with obstacles. Whereas they reactively compute commands that avoid collisions, we propose predictive planning by computing alternative trajectories to the goal location.

Other authors propose to substitute the joystick by brain computer interfaces (BCI). Carlson and Millán [17] describe a BCI controlled autonomous wheelchair where the user selects from the classes *forward*, *left*, and *right*. The wheelchair takes care of the low level control to translate these signals to velocity commands. Similarly, Philips et al. [117] describe a BCI controlled wheelchair also with the three classes *forward*, *left*, and *right*. Every $0.5\,\mathrm{s}$ their interface sends a probability distribution to the shared control system. To alter the distribution the system takes into account the obstacles in the environment as well as a goal location. This is similar to the first mode of shared autonomy that we proposed in this chapter. In our work we propose to maintain a probability distribution over different goal-directed trajectories that we bias with respect to a preferred direction.

Bonarini et al. [15] also describe a project that aims at developing a shared autonomy wheelchair platform. They highlight that BCI interfaces are prone to misclassify user intent which is why they propose an additional confirmation step in the interface. This

suggests that the required information from the user should be as low as possible especially for BCI interfaces. In contrast to the abovementioned approaches, we use the user input to select trajectories to a (sub)goal location, instead of directly mapping the desired direction to movements. Our goal is to exploit the free space representation of Voronoi diagrams to provide meaningful alternatives to the user even with a limited number of discrete choices. Mandel and Frese [100] compare two different paradigms for semi-autonomous wheelchair navigation. Their first method allows the user to select by speech a route from a discrete set of paths on a Voronoi graph, similar to our method. Their second method directly maps head poses to steering commands.

For tele-presence robots, similar techniques apply to overcome slow and noisy user input. Robert et al. [130] describe a BCI controlled tele-presence robot with a two-class interface. The same robotic system is presented by Carlson et al. [18], with an additional class that instructs the robot to proceed with the default behavior of driving in the current direction and avoiding obstacles. Geerinck et al. [44] present a tele-operation robot platform that can either be steered manually using a joystick, or by several levels of autonomy to reduce the human operator's workload. In shared control mode, the robot follows the general direction as specified by the user, but takes the initiative to choose its own low level path to avoid obstacles.

In contrast to many of the aforementioned approaches that implement shared autonomy in terms of local collision avoidance, we let the user select trajectories to (sub)goals. A further advantage of this approach is that our method is suitable to show the alternative trajectories to the user via a GUI, which increases the predictability of the robotic system.

7.6 Conclusion

In this chapter, we applied the methods described in previous chapters to shared autonomy navigation. The key idea of our approach is to compute a set of optimized trajectories that are valid for navigation, but which are qualitatively distinct. Our goal is to maintain a high level of autonomy for the user and at the same time keep the cognitive burden low. To this end, we proposed two different shared autonomy modes. First, we assume a known target position and maintain a probability distribution over different trajectories, which can be continuously biased by a direction preference of the user. Second, when no global goal location is known to the system, we propose to present a discrete set of path alternatives to local subgoals to the user. This allows also the usage of slow and noisy interfaces such as BCI, or tele-operation. In experiments with a real robotic wheelchair we demonstrated that our method is suitable for navigating real robots in shared autonomy mode.

Chapter 8

An Approach to Socially Compliant Leader Following for Mobile Robots

A wide range of tasks for mobile robots demand that the robot follows a human leader. This includes robotic co-workers in factories, autonomous shopping carts, or robotic wheelchairs that autonomously navigate next to an accompanying pedestrian. Many authors proposed follow-the-leader approaches for mobile robots, which have also been applied to the problem of following pedestrians. Most of these approaches use local control methods to keep the robot at the desired position. However, they typically do not incorporate information about the natural navigation behavior of humans, who strongly interact with their environment. In this chapter, we propose to use a learned, predictive model of interactive navigation behavior that enables a mobile robot to predict the trajectory of its leader and to compute a far-sighted plan that keeps the robot at its desired relative position. Extensive experiments in simulation as well as with a real robotic wheelchair suggest that our method outperforms state-of-the-art methods for following a human leader in a wide variety of situations.

8.1 Introduction

In Chap. 5 we presented a model of the natural navigation behavior of pedestrians, and a method to learn its parameters in Chap. 6. In this chapter, we present an application for this model to allow a mobile robot to follow a human leader. There is a wide range of applications for mobile robots for which it is desirable that the robot follows a human leader. For example a robotic co-worker that provides tools to a human in a factory needs to stay in a position where the human can reach the robot. Similarly, a mobile shopping cart should always stay in a position where the human is able to place objects into it. A further application is a robotic wheelchair that stays side by side to an accompanying pedestrian, allowing interaction with the pedestrian during the navigation task.

When following a human leader, it is beneficial for the robot to reason about the natural navigation behavior of pedestrians. During navigation, pedestrians interact with their

environment, which includes obstacles, other nearby humans and also the robot itself. A robot that has a better understanding of this interactive behavior is able to fulfill its task in a socially compliant way, i.e., in a way that does not unnecessarily hinder nearby pedestrians. Such a robot is able to predict the behavior of the humans and to plan far-sighted trajectories that keep the robot close to its desired position in the long run.

There has been a wide range of research on controlling a group of robots in formation, which have also been applied in the context of social robotics [105, 121]. Many of these approaches utilize control-theoretic methods to steer the robot towards a virtual target that moves along with the leader [29, 121]. However, these methods mostly neglect information about the more complex navigation behavior of pedestrians that strongly depends on the environment.

In this chapter, we propose to utilize the feature-based model of human navigation behavior that we presented in Chap. 5 to predict the path of the leading pedestrian. This model accounts for the intention of a human to reach a certain goal while maintaining a comfortable velocity, avoiding strong accelerations, and staying clear of obstacles. We can learn the individual characteristics of different pedestrians, or distinct behavior in different environments from observation, as shown in Chap. 6.

The novelty of the approach presented in the following is a method that simultaneously predicts the most likely trajectory of the pedestrian and computes the trajectory for the robot that minimizes the distance to its desired relative position along the whole trajectory in a forward-looking manner. Such a predictive planning method leads to a socially more compliant behavior of the robot. In addition, planning long-term trajectories mitigates the problem of local minima in a local control function, especially in the presence of arbitrary, non-convex obstacles in the environment. We conducted a simulated comparison of our method to related approaches as well as experiments with a real robot. Our results show the applicability of the proposed approach to navigate a robotic wheelchair next to an accompanying pedestrian.

8.2 A Socially Compliant Follow-the-Leader Approach

A better understanding of the natural navigation behavior of pedestrians enables a mobile robot to follow a human leader in a socially more compliant way. In this section, we first formalize the problem of following a leader. We consider the navigation task to stay close to a fixed relative position with respect to its leader. To solve this task, we propose an approach that predicts the trajectory of the pedestrian and at the same time computes a forward-looking trajectory that minimizes the deviation to the desired position.

8.2.1 Problem Definition

In this work, we consider the 2D navigation behavior of a mobile robot and a leading pedestrian. A trajectory $\mathbf{p}^h$ of the human and $\mathbf{p}^r$ of the robot are mappings $\mathbf{p} : \mathbb{R} \to \mathbb{R}^2$ from time to a 2D position. As discussed in Sec. 5.1, the position of the robot or the pedestrian at time t is thus given by $\mathbf{p}(t)$ and their velocity by $\dot{\mathbf{p}}(t)$. We assume a mobile

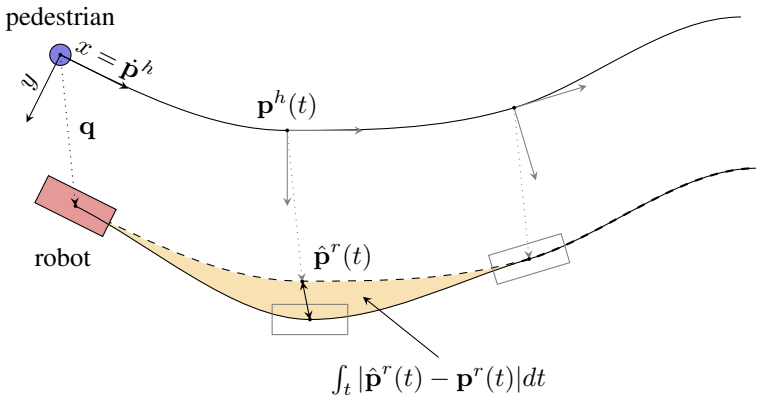


Figure 8.1: The desired position of the robot is a fixed location in the local coordinate system of the pedestrian. The dashed curve illustrates the desired trajectory of the robot $\hat{\mathbf{p}}^r$ given the predicted trajectory $\hat{\mathbf{p}}^h$ of the pedestrian. Deviation from the desired trajectory yields an additional cost integrated along the trajectory, as illustrated by the shaded area.

robot with a differential drive that is always oriented in driving direction. Similarly, we assume that the pedestrian is always headed in walking direction. Thus, the orientation $\alpha(t)$ at time t is the direction of the vector $\dot{\mathbf{p}}(t)$.

We define the desired position of the robot by a fixed position $\mathbf{q} = (q_x, q_y)^T$ in the local coordinate system of the pedestrian, i.e., the robot is supposed to always maintain the same position relative to the human. Given the trajectory $\mathbf{p}^h(t)$ of the human, we can compute the desired trajectory of the robot

$$\hat{\mathbf{p}}^r(t) = \mathbf{p}^h(t) + \boldsymbol{R}(\alpha^h(t))\mathbf{q}, \tag{8.1}$$

where $\boldsymbol{R}(\alpha^h(t))$ is the rotation matrix of the human at time t. In practice the robot cannot always follow this desired trajectory due to obstacles in the environment, or other dynamic constraints. We cast the resulting navigation goal in a utility-optimizing manner, where the cost function is a linear combination of the squared norm of the deviation from the desired trajectory and an additional term $g_{\text{nav}}(\mathbf{p}^r, t)$ that comprises acceleration and velocity bounds and clearance to obstacles. Therefore, the desired trajectory minimizes the navigation cost function

$$c(\mathbf{p}^r) = \int_{t=0}^{T} \left(\theta_1 |\mathbf{p}^r(t) - \hat{\mathbf{p}}^r(t)|^2 + \theta_2 g_{\text{nav}}(\mathbf{p}^r, t)\right) dt, \tag{8.2}$$

where the weights θ_1 and θ_2 are model parameters to adjust the behavior to the given application. Fig. 8.1 illustrates the predicted trajectory of the pedestrian, the offset in the local reference frame of the pedestrian and the resulting desired trajectory of the robot. The challenge of this approach is to predict the trajectory of the human, which determines the desired trajectory of the robot $\hat{\mathbf{p}}^r(t)$. To this end we utilize a predictive model of natural human navigation behavior, which we presented in Chap. 5.

8.2.2 Unifying Prediction and Planning

Our approach relies on an accurate model of human navigation behavior that allows the robot to predict the movements of the leading pedestrian. To achieve a socially compliant behavior of the robot, we want to explicitly model the fact that the human is also aware of the robot and reacts to the actions of the robot.

Chap. 5 introduces a probabilistic model of such an interactive navigation behavior. For given start and goal positions, the proposed model yields a distribution over the joint space of the trajectories of each agent involved in the navigation process. This probability distribution depends on a weighted sum of features $\mathbf{f}$ that capture important properties of human navigation behavior. Each feature is a function that maps a composite trajectory, i.e., the set of trajectories for all agents, to a real value. In Sec. 5.1.4 we proposed features that describe the individual properties of each trajectory, such as the integrated velocity and acceleration along the trajectory, and the time to reach the target. In addition, we proposed features that describe interaction between the agents, such as their mutual distance. A weight vector $\boldsymbol{\theta}$ parameterizes the model and describes the importance of each feature in the feature vector $\mathbf{f}$.

In the special case of two agents h and r, the model yields the distribution

$$p_{\boldsymbol{\theta}}(\mathbf{p}^h, \mathbf{p}^r) \propto \exp(-\boldsymbol{\theta}^T \mathbf{f}(\mathbf{p}^h, \mathbf{p}^r)), \tag{8.3}$$

where $\mathbf{p}^h$ and $\mathbf{p}^r$ are the trajectories of the two agents, as introduced in the previous section. One can interpret $\boldsymbol{\theta}^T \mathbf{f}(\mathbf{p}^h, \mathbf{p}^r)$ as a cost function. The agents are thus exponentially more likely to select a trajectory with lower cost. To adapt the model to the individual navigation behavior of different pedestrians or to a certain environment, we can learn the feature weights $\boldsymbol{\theta}$ from observed data, such that the predicted trajectories accurately resemble the navigation behavior of real humans in the designated environment. We described the learning technique in Chap. 6.

In this chapter, we utilize the model described above to predict the trajectory of the pedestrian, and to plan a trajectory for the robot next to the pedestrian at the same time. In particular, we adopt the proposed features that capture accelerations, velocities, distances to obstacles and the time to reach the target to predict the natural navigation behavior of the pedestrian. In addition, we introduce the feature

$$f_{\text{deviation}}(\mathbf{p}^h, \mathbf{p}^r) = \int_{t=0}^{T} |\mathbf{p}^r(t) - \hat{\mathbf{p}}^r(t)|^2 dt, \tag{8.4}$$

that describes the squared deviation from the desired position of the robot along the trajectory, and

$$f_{\text{nav}}(\mathbf{p}^h, \mathbf{p}^r) = \int_{t=0}^{T} g_{\text{nav}}(\mathbf{p}^r, t) dt, \tag{8.5}$$

to account for further navigation constraints of the robot, as described in Sec. 8.2.1. During navigation, we compute the most likely composite trajectory $(\mathbf{p}^h, \mathbf{p}^r)$ with respect to the probability distribution given by Eq. (8.3). Due to the additional features $f_{\text{deviation}}$ and f_{nav}, this most likely composite trajectory not only predicts the trajectory of the

pedestrian, but also computes the trajectory of the robot that minimizes the navigation cost function of the robot (Eq. (8.2)). In particular, this method accounts for the effect that the pedestrian interacts with the robot, i.e., that the pedestrian behaves cooperatively and navigates in a way that helps the joint navigation goal. By adjusting the weights of the features we can adapt the level of cooperative behavior that we ascribe to the human. Fig. 8.3 and Fig. 8.4 illustrate the predicted trajectory of the pedestrian and the planned trajectory for the robot in two different scenarios.

In addition, the predictive model is beneficial in situations where the leading pedestrian is not in the field of view of the robot's sensors for some time. Instead of stopping the navigation task, the robot is able to predict the trajectory of the pedestrian and to continue its plan. When the human reappears in the observation of the robot, the people tracker can use the prediction to solve the data association problem, i.e., to select the correct pedestrian as leader.

The predictive model yields predictions of trajectories to known target positions. However, the final target position of the pedestrian is not known in general. In our experiments, we interpolate the observed trajectory of the pedestrian to estimate its target position. In environments where prior information of the typical paths of pedestrians is available, we can also use more sophisticated methods to estimate their target position [10, 160].

8.3 Experiments

In this section, we describe a set of experiments using a real robotic wheelchair that suggest that our method is applicable to successfully navigate alongside an accompanying pedestrian in the presence of obstacles. Furthermore, we compare our approach in simulation to two related methods. These experiments intend to show the advantages of our predictive planning approach over local control methods, especially in situations where the environmental conditions hinder the robot to stay close to its desired position. During the navigation task, our method continuously computes the most likely composite trajectory by optimizing its probability at a rate of $5\,\mathrm{Hz}$.

8.3.1 Real Robot Experiments

In the following experiments, we use the method proposed in this chapter to navigate a robotic wheelchair next to a pedestrian at a distance of $1\,\mathrm{m}$. The wheelchair robot relies on on-board sensors only, as we described in Sec. 5.3. It localizes itself in the environment using Monte Carlo localization [140] and tracks the pedestrian using a laser based people tracker. Fig. 8.2 shows the paths of the wheelchair and the pedestrian as observed by the wheelchair in two different scenarios.

In the first run (Fig. 8.2 left), the robot's desired position is on the left of the pedestrian. It starts moving alongside the pedestrian, falls back behind the pedestrian during passing the passage and catches up afterwards. Fig. 8.3 shows the predictions of the wheelchair during the navigation task in the same run. As soon as the pedestrian starts to move, the robot estimates the target position of the pedestrian by interpolating the current velocity

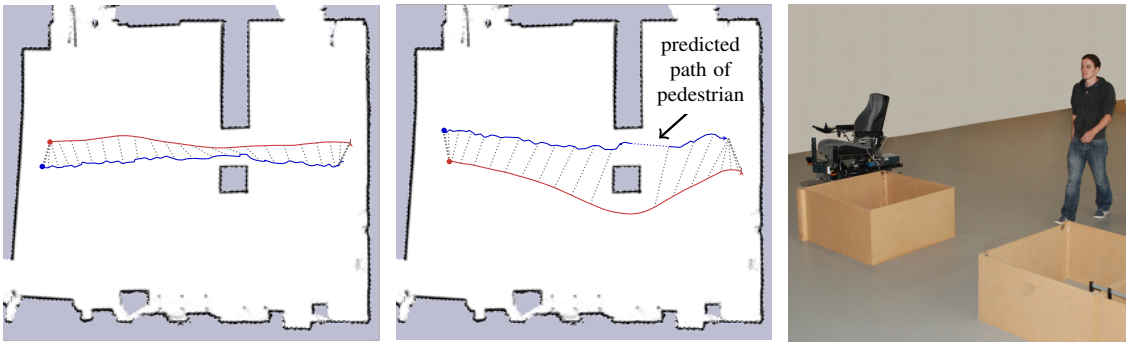


Figure 8.2: Left: observed trajectories of the robot (red) and the human (blue) during navigation. The robot falls back behind the pedestrian in the narrow passage. Middle: observed trajectories in an experiment where the robot bypasses the obstacle on the lower side to meet the pedestrian after the passage. Right: Experimental setup.

10 s into the future. We then use the method described above to compute the most likely composite trajectory of the robot and the pedestrian. Right after the pedestrian starts to move, the robot predicts a target position before the obstacle, and plans to stay next to the pedestrian along the whole planning interval (A). In the second image (B), the predicted target positions are behind the obstacle. Here, the robot plans to stay behind and to regain the position to the left of the pedestrian after the narrow passage. The robot follows this plan, as shown in the third (C) and fourth (D) image.

In the second run (Fig. 8.2 middle), we swap the position of the pedestrian and the robot, which is supposed to keep its position on the right hand side of the pedestrian. The figure shows that the robot stays next to the human at first, but decides to pass the obstacle on the right side. Our system selects this variant that has a lower cost $\boldsymbol{\theta}^T \mathbf{f}$ compared to following the human through the narrow passage. Since the cost function models the task of the robot, this means that the robot expects to better fulfill its task of staying close to the human in addition to further navigation constraints by passing the obstacle on the right.

Fig. 8.4 shows the predictions of the robot during this second run. As soon as the pedestrian starts to move, the robot updates its plan to stay right next to the pedestrian along the whole planned path (A). After the pedestrian approached the obstacle, the robot needs to plan a path around the obstacle. Due to the geometry of the environment, there is enough space on the lower side of the obstacle. Therefore, the robot decides to pass the obstacle on this side, which allows the robot to stay at the human's side as long as possible (B, C). While the pedestrian is in the passage, the obstacle blocks the laser scanner and the robot cannot observe the pedestrian (D). However, since the robot maintains predictions about the movement of the pedestrian, it is able to follow its planned path until the tracker observes the pedestrian again.

These experiments suggest that our method is applicable to navigate a real robot in realistic scenarios. The autonomous wheelchair is able to predict the pedestrian's behavior, and to compute a suitable plan for the wheelchair online during navigation. Such a system would allow a wheelchair user who is not able to steer the wheelchair him or herself to

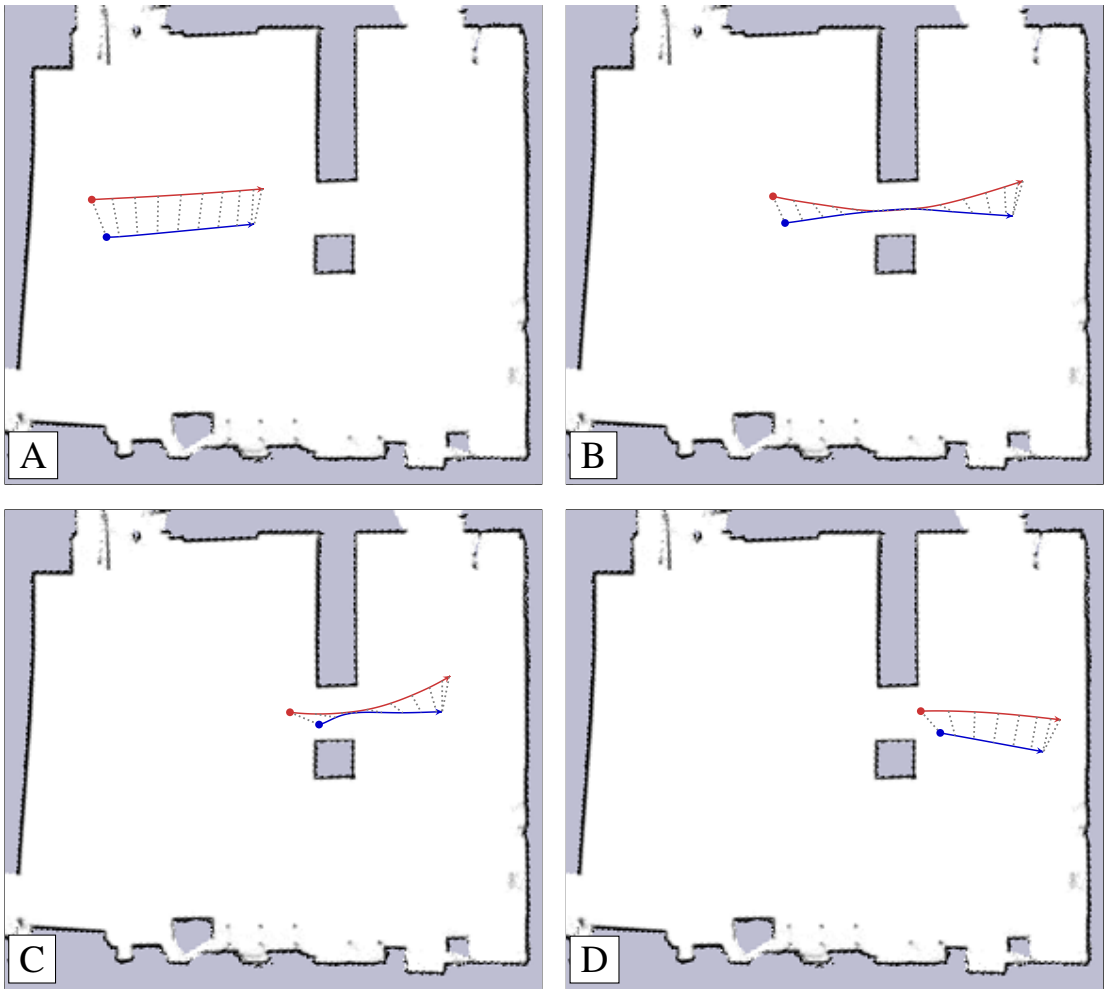


Figure 8.3: Predictions computed by the wheelchair at four successive time steps. The robot predicts the human to pass the passage. Since the passage is too narrow for the robot (red) and the human (blue) to pass it at the same time, the robot leaves its desired position and lets the human pass first. After the passage, the robot resumes its desired position.

interact with an accompanying pedestrian during navigating to their goal location. In the following, we present a comparison to related methods, which show the advantage of predictive planning over reactive approaches.

8.3.2 Comparison

Fig. 8.5 and Fig. 8.6 show a comparison of our method in simulation to a social forces (SF) based approach [8] and a velocity obstacles (VO) approach, similar to the method proposed by Prassler et al. [121]. In their paper, they propose a velocity obstacles approach to navigate an autonomous wheelchair. To allow for a fair comparison of the methods, we scripted the pedestrian's path on a rectangular path with a velocity of $0.5\,\mathrm{m/s}$. The desired position of the robot is $1\,\mathrm{m}$ to the left of the pedestrian for all experiments. We set the parameters of all approaches such that the robot always kept a safety distance of at least

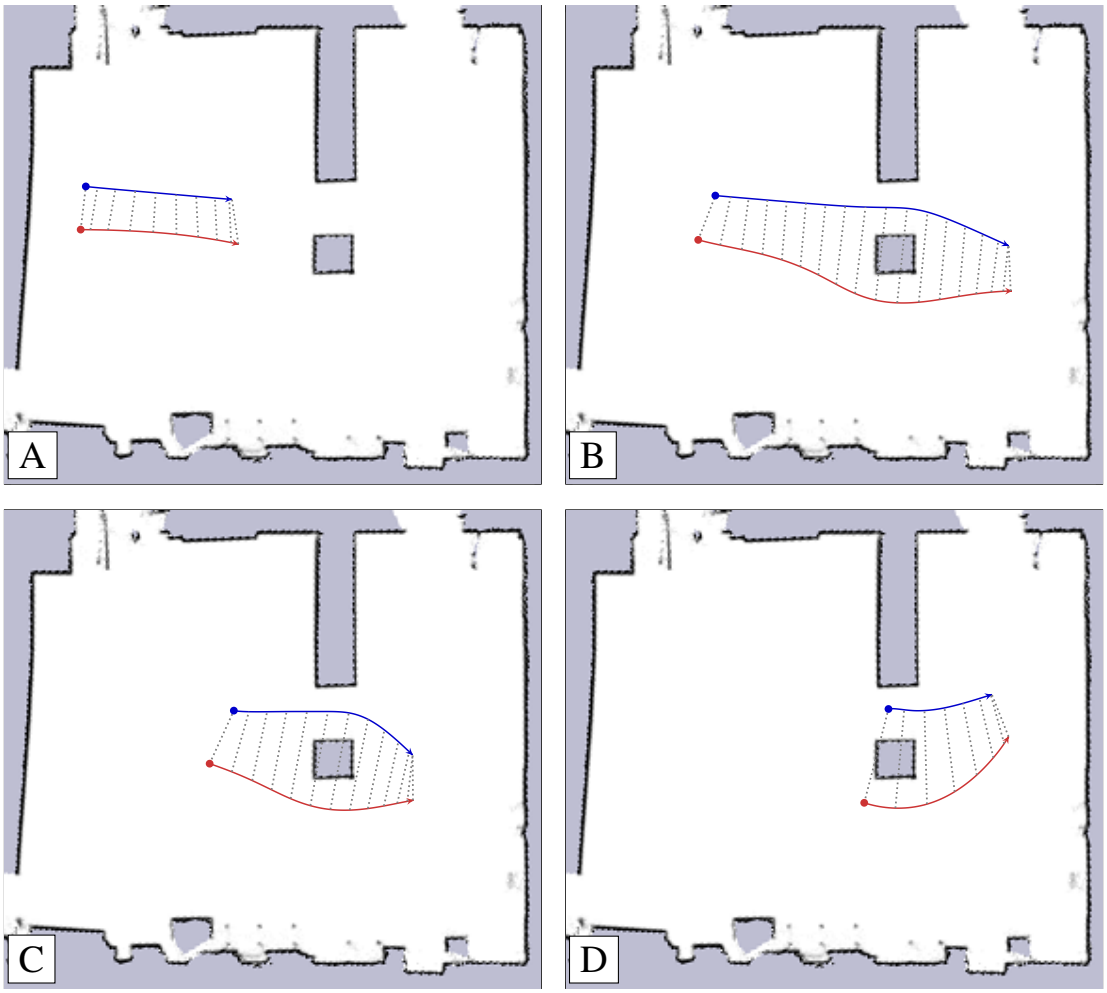


Figure 8.4: Prediction computed by the wheelchair at four successive time steps. In this experiment, the desired position of the robot is on the right hand side of the human. The robot stays at the human's side as long as possible. It then evades the obstacle on the right side and continues to move to its desired position relative to the human.

$0.25\,\mathrm{m}$ to the pedestrian, as well as to obstacles in the environment.

Both, SF as well as VO compute control commands towards a virtual target position. To compute this position, we adopt the method proposed by Prassler et al. [121]. They linearly extrapolate the current velocity of the pedestrian in a small time horizon Δt to avoid that the robot lags behind the desired position. We adjust Δt for both methods such that the robot converges to the desired position when the pedestrian moves on a straight line with $0.5\,\mathrm{m/s}$.

In the test environments, the challenge for the robot is to catch up to the desired position after the pedestrian takes turns on its path. Furthermore, there is a narrow passage in which the robot cannot keep its desired position. Fig. 8.5 shows that all methods manage to pass the passage. However, the bar plot on the right shows that our method is able to stay closer to the desired position on average along the trajectory. This is due to the fact that our method predicts the trajectory of the pedestrian and computes the trajectory of

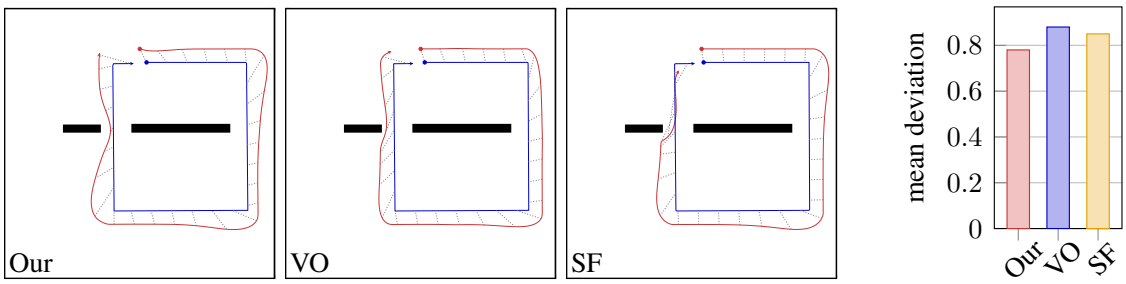


Figure 8.5: Comparing our method to velocity obstacles (VO) and social forces (SF) in simulation. The desired position of the robot (red) is one meter to the left side of the human (blue). The bar plot shows that our method stays closer to the desired position on average.

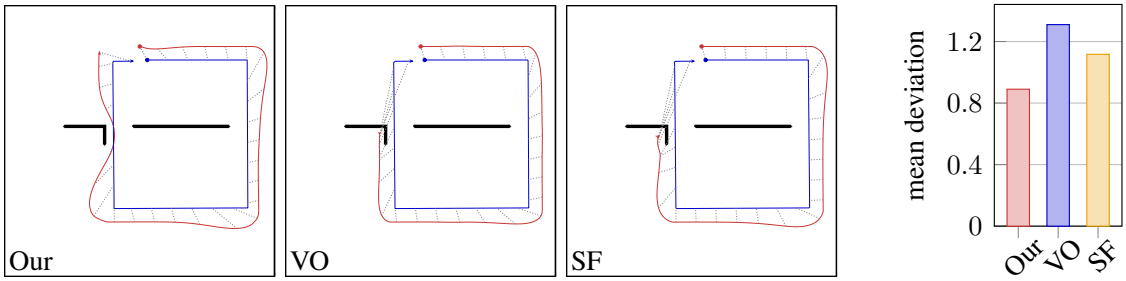


Figure 8.6: Comparison to VO and SF that illustrates the advantages of our method over local control methods. The trajectory of the simulated pedestrian is identical to the previous experiment, illustrated in Fig. 8.5. However, the obstacles have a non-convex shape in this experiment. While our method predictively plans a trajectory through the narrow passage, both VO and SF get stuck in the non-convex obstacle.

the robot that minimizes the deviation along the whole path, while also incorporating properties of the robot, such as limited acceleration or velocity constraints. Such a long term planning is better suited to accomplish the navigation task compared to greedily approaching the desired position.

Fig. 8.6 shows a similar experiment with an additional obstacle that resembles an open door in a typical indoor environment. The first image shows that our approach is able to negotiate the passage in a similar way as in the first setup. The robot falls back behind the pedestrian and catches up afterwards. Both SF as well as VO, however, get stuck behind the open door, since there is a local minimum in their local control functions. The bar plot reflects the advantage of the predictive planning in this experiment. Whereas our method shows a similar mean deviation from the desired position as in the first experiment, SF and VO gain a higher deviation whilst stuck in the local minima.

8.4 Related Work

In the past, many authors proposed methods to navigate a group of robots in formation. Liu et al. [98] cast the joint path planning task of a robot formation as a linear programming problem. Similar to our approach, they plan the trajectories to the target position of each robot. However, Liu et al. control the group of robots in a central manner and each robot

executes the optimal trajectory. Balch and Hybinette [8] propose to use social potential fields that pull the robots towards attraction points to achieve a certain formation. Our experiments include a comparison to a social potential-based approach.

A different problem arises when the task of the robot is to follow a leader whose goal is unknown. Chiem and Cervera [24] and Huang et al. [59] propose to compute a cubic Beziér curve between the leading robot and the follower. The follower then navigates along this trajectory, using a velocity controller. In addition, if the robot's task is not only to follow the same path but to stay in a certain formation, they propose to compute virtual targets for each of the robots and compute Beziér curves to these target positions. However, they follow the leading robot without active obstacle avoidance. Desai et al. [35] and Das et al. [29] use control theoretic approaches to keep each robot close to its designated position within the formation, also considering obstacles in the environment. If the desired shape of the formation changes, they introduce control graphs to assign the robots to their new position in the formation. Qin et al. [122] use artificial forces to navigate each robot close to the desired position in a formation. Similarly, Tanner and Kumar [138] propose to use navigation functions to keep a group of robots in a certain formation. Navigation functions also lead the robot along the gradient of a smooth function, similar to artificial forces, but there are no local minima allowed, except of the target position. In general, however, it is difficult to design such a function for arbitrary environments [93]. Chen and Wang [23] provide a survey on different approaches to robot formation control.

The abovementioned approaches use local control methods to steer the robot either directly to the desired position in the formation, or to some local virtual target position. In contrast, we predict the trajectory of the leader based on its current state and the state of the environment. At the same time, we compute the trajectory that minimizes the distance to the desired relative position along this trajectory while satisfying further constraints. This prevents the robot from getting stuck in local minima of the cost function and allows it to adapt the planned trajectories to the environment early on.

Similar methods have also been used to enable a robot to follow a human leader. Pradhan et al. [120] utilize a navigation function method and set the tracked positions of the pedestrian as virtual target positions. Therefore, the robot is only able to follow the person, but not to stay at a fixed relative position. Prassler et al. [121] aim at coordinating the motion of a human and a robot and also apply it to a robotic wheelchair. They propose to use the velocity obstacles approach [39] to guide the robot to a local virtual target. We compare our method to a similar approach in our experimental section. Most similar to our approach is the work of Morales et al. [105]. They optimize a utility that encodes the desired relative position as well as the walking comfort of the pedestrian. However, they optimize the planned trajectory locally, whereas we optimize future trajectories to a distant subgoal, which allows the robot to adapt its behavior to the environment in a predictive manner.

Müller et al. [108] and Stein et al. [137] proposed methods to select a pedestrian which the robot follows. They aim to improve the navigation of a mobile robot by following a pedestrian who walks in the same direction. In contrast, our goal is to stay close to a desired position relative to a fixed pedestrian throughout the whole navigation task.

8.5 Conclusion

In this chapter, we used our model of interactive navigation behavior to allow a mobile robot to follow a leading person in a socially compliant way. Our approach uses the feature-based model of natural navigation behavior to predict the trajectory of the leading human. In contrast to previous approaches, our method allows the robot to compute far-sighted plans that minimize the long-term deviation from the desired trajectory. In addition to features that describe natural intents of navigating pedestrians, our method uses features that capture the navigation goals of the robot. The resulting model thus unifies predictions of the human's behavior with path planning for the robot. In several experiments with a robotic wheelchair we demonstrated that the proposed model is applicable to real world scenarios such as navigating alongside an accompanying person in the presence of obstacles. A comparison in simulation suggests that our method outperforms previous models that rely on local control strategies.

Chapter 9

Learning Behavior Styles for Autonomous Cars

It is expected that autonomous vehicles capable of driving without human supervision will be released to market within the next decade. For user acceptance, such vehicles should not only be safe and reliable, they should also provide a comfortable user experience. However, individual perception of comfort may vary considerably among users. While some users might prefer sporty driving with high accelerations, others might prefer a more relaxed style. Typically, a large number of parameters such as acceleration profiles, distances to other cars, speed during lane changes, etc., characterize a human driver's style. Manual tuning of these parameters may be a tedious and error-prone task. Therefore, we propose a learning-from-demonstration approach that allows the user to simply demonstrate the desired style by driving the car manually. We model the individual style in terms of a cost function and use feature-based inverse reinforcement learning to find the model parameters that fit the observed style best. Once the model has been learned, a vehicle can use it to efficiently compute trajectories in autonomous mode. We show that our approach is capable of learning cost functions and reproducing different driving styles using data from real drivers. Furthermore, we apply the learned policy for navigating an autonomous car in simulation.

9.1 Introduction

In the previous sections, we described methods that allow mobile robots to navigate in populated environments and to learn models from demonstrations. However, the focus of these methods was on domestic robots that move in the direct vicinity of pedestrians. The same approaches can also be applied in the context of self-driving cars. While highway driving poses different properties and challenges, the main ideas of our methods still apply. During navigation, there are also discrete decisions, such as staying in lane or to overtake, and continuous properties of the trajectories within such a behavior class. Furthermore,

Figure 9.1: A Bosch highly automated driving development vehicle.

especially for high-speed highway driving, it is important to plan predictively and adapt the behavior of the car early in order to avoid dangerous situations. In this chapter, we present a framework to model the decision process for navigating an autonomous car on highways, and a method to learn individual driving styles from demonstrations.

Recent studies indicate that the pace of innovation and investment in self-driving vehicles is accelerating and that consumers are open to the idea of such vehicles [78, 79]. Some of the key factors in user acceptance of autonomous vehicles are safety, reliability, and comfort. Comfort is subjective and can be influenced by various factors including the driving style, which is the way in which a driver habitually drives [38] and is a trade-off between features such as speed, acceleration, jerk, distance to other vehicles, etc. Studies suggest that driving styles vary across users [139]. To be comfortable for different users, an autonomous vehicle should adapt its driving style according to user preferences in addition to maintaining safety.

We can achieve different driving styles for an automated vehicle by varying the model parameters of its motion planning algorithm. However, manual tuning of these parameters is generally difficult because of the large number of parameters that may have antagonistic effects. If at all possible, manual tuning is likely to be a tedious and time-consuming process. Therefore, we propose a learning-from-demonstration approach to learn the model parameters for each user from their observed driving style. We assume that the desired driving style maximizes some notion of a reward, i.e., that there is a cost function that explains the driver's style. The challenge is to find the cost function that best explains

the observed style and that also generalizes to different situations.

In this chapter, we use the feature-based inverse reinforcement learning method that we presented in Chap. 6 to learn driving styles from demonstrations. The key parts of the model that we used in the previous chapters also apply in the context of navigating an autonomous car. Similar to the properties of the desired navigation behavior of social robots, we use features that capture important properties of the driving style that we want to reproduce. Our model comprises a cost function that is a linear combination of these features. To find the feature weights that best fit the observed style we use the proposed learning method.

Furthermore, the task of navigating a car consists of a combination of discrete and continuous decisions. When driving on highways, there arise a number of discrete decisions such as staying in lane, or overtake. For the trajectories within each of these classes, it is important to also capture higher-order properties such as velocity, acceleration and jerk, since acceleration and jerk have a strong influence on the comfort of the passengers [63]. To capture these properties, we propose to use a continuous representation of trajectories as proposed in Chap. 5.

Once the parameters have been learned, we use the resulting model to compute trajectories online during autonomous driving tasks. To quickly react to changes in the environment, it is important to efficiently reason about the trajectories. Thus, we exploit the techniques for efficient trajectory optimization developed in the previous chapters.

In the remainder of this chapter we describe the state representation of trajectories for autonomous cars and describe the inverse reinforcement learning approach. Furthermore, we propose a set of features that are relevant to the task of highway driving and present experiments that suggest that our method is suitable for learning individual driving styles from demonstrated trajectories.

9.2 A Model for Autonomous Highway Driving

For representing trajectories of the autonomous car and learning individual navigation behavior, we apply similar methods as those described in Chap. 5 and Chap. 6. In particular, we model the navigation behavior as a two-stage process. The first stage corresponds to a discrete navigation choice. The second stage selects a specific trajectory from a continuous distribution within this class. We also learn individual navigation behavior using feature-based IRL. However, different conditions and challenges arise in the context of highway navigation at a high speed, compared to mobile robot navigation in office environments. In the following, we describe a model that captures the navigation behavior of autonomous cars on highways and highlight similarities and differences to the methods proposed in the previous chapters.

9.2.1 Trajectory Representation

We represent the trajectories of cars using splines, similar to the representation described in Chap. 5. However, for high-speed navigation it is important to satisfy curvature continuity

of the trajectories. A trajectory with discontinuities in the curvature yields instantaneous changes of the steering angle, which is obviously not possible for a real car. Using such a trajectory for the control of a car could therefore lead to unexpected, dangerous behavior.

The curvature $\kappa(t)$ of a trajectory at time t is a function of the acceleration and the velocity of the trajectory

$$\kappa(t) = \frac{a_y(t)v_x(t) - v_y(t)a_x(t)}{(v_x(t)^2 + v_y(t)^2)^{\frac{3}{2}}}. \tag{9.1}$$

Therefore, to achieve curvature continuity we require the acceleration to be a continuous function over time. Using splines, this implies to increase their degree to five, instead of cubic splines with a degree of three that we used for the experiments in the previous chapters. For quintic splines, the restriction $\mathbf{p}|_{[t_j,t_{j+1}]}(t)$ of the trajectory to a time interval $[t_j, t_{j+1}]$ is a two-dimensional polynomial of degree five:

$$\mathbf{p}|_{[t_j,t_{j+1}]}(t) = \begin{pmatrix} a_{x,j}t^5 + b_{x,j}t^4 + c_{x,j}t^3 + d_{x,j}t^2 + e_{x,j}t + f_{x,j} \\ a_{y,j}t^5 + b_{y,j}t^4 + c_{y,j}t^3 + d_{y,j}t^2 + e_{y,j}t + f_{y,j} \end{pmatrix}. \tag{9.2}$$

Similar to cubic splines, we parameterize the spline using the position, velocity, and in addition the acceleration

$$\mathbf{p}(t_j) =: \mathbf{p}_j = \begin{pmatrix} p_{x,j} \\ p_{y,j} \end{pmatrix}, \dot{\mathbf{p}}(t_j) =: \dot{\mathbf{p}}_j = \begin{pmatrix} \dot{p}_{x,j} \\ \dot{p}_{y,j} \end{pmatrix} \quad \text{and} \quad \ddot{\mathbf{p}}(t_j) =: \ddot{\mathbf{p}}_j = \begin{pmatrix} \ddot{p}_{x,j} \\ \ddot{p}_{y,j} \end{pmatrix} \tag{9.3}$$

at the end points of each spline segment. Since two spline segments share control points, the resulting trajectories are C^2 continuous, i.e., the acceleration and thus the curvature are continuous.

For this representation, the x-component of the trajectory in the interval $[t_j, t_{j+1}]$ is given by

$$p_x|_{[t_j,t_{j+1}]}(t) = \begin{pmatrix} u^5 \\ u^4 \\ u^3 \\ u^2 \\ u \\ 1 \end{pmatrix}^T \begin{pmatrix} -n_0 + 5n_1 - 10n_2 + 10n_3 - 5n_4 + n_5 \\ 5n_0 - 20n_1 + 30n_2 - 20n_3 + 5n4 \\ -10n_0 + 30n_1 - 30n_2 + 10n_3 \\ 10n_0 - 20n_1 + 10n_2 \\ -5n_0 + 5n_1 \\ n_0 \end{pmatrix}, \tag{9.4}$$

where u is the normalized spline segment parameter

$$u = \frac{t - t_j}{t_{j+1} - t_j}. \tag{9.5}$$

The variables n_0 to n_5 are defined as follows:

$$\begin{aligned}
n_0 &= p_{x,j} \\
n_1 &= 0.2\dot{p}_{x,j}\Delta t + p_{x,j} \\
n_2 &= 0.05\ddot{p}_{x,j}\Delta t^2 + 0.4\dot{p}_{x,j}\Delta t + p_{x,j} \\
n_3 &= 0.05\ddot{p}_{x,j+1}\Delta t^2 - 0.4\dot{p}_{x,j+1}\Delta t + p_{x,j+1} \\
n_4 &= -0.2\dot{p}_{x,j+1}\Delta t + p_{x,j+1} \\
n_5 &= p_{x,j+1},
\end{aligned}$$

where $\Delta t = t_{j+1} - t_j$ is the time duration of this segment.

The first derivative of the x component of the spline, which corresponds to the velocity is given by

$$v_x|_{[t_j,t_{j+1}]}(t) = \begin{pmatrix} u^4 \\ u^3 \\ u^2 \\ u^1 \\ 1 \end{pmatrix}^T \begin{pmatrix} 5(-n_0 + 5n_1 - 10n_2 + 10n_3 - 5n_4 + n_5) \\ 4(5n_0 - 20n_1 + 30n_2 - 20n_3 + 5n4) \\ 3(-10n_0 + 30n_1 - 30n_2 + 10n_3) \\ 2(10n_0 - 20n_1 + 10n_2) \\ -5n_0 + 5n_1 \end{pmatrix}, \tag{9.6}$$

and the acceleration is given by

$$a_x|_{[t_j,t_{j+1}]}(t) = \begin{pmatrix} u^3 \\ u^2 \\ u^1 \\ 1 \end{pmatrix}^T \begin{pmatrix} 20(-n_0 + 5n_1 - 10n_2 + 10n_3 - 5n_4 + n_5) \\ 12(5n_0 - 20n_1 + 30n_2 - 20n_3 + 5n4) \\ 6(-10n_0 + 30n_1 - 30n_2 + 10n_3) \\ 2(10n_0 - 20n_1 + 10n_2) \end{pmatrix}. \tag{9.7}$$

In addition, the jerk is the third derivative

$$j_x|_{[t_j,t_{j+1}]}(t) = \begin{pmatrix} u^2 \\ u^1 \\ 1 \end{pmatrix}^T \begin{pmatrix} 60(-n_0 + 5n_1 - 10n_2 + 10n_3 - 5n_4 + n_5) \\ 24(5n_0 - 20n_1 + 30n_2 - 20n_3 + 5n4) \\ 6(-10n_0 + 30n_1 - 30n_2 + 10n_3) \end{pmatrix}. \tag{9.8}$$

The y-components of the position, velocity, acceleration and the jerk are defined analogously.

The behavior of the autonomous car depends also on the behavior of other vehicles on the road. We therefore capture a set of trajectories for all vehicles as a composite trajectory

$$\mathbf{x}(t) = (\mathbf{p}^{a_1}(t), \ldots, \mathbf{p}^{a_n}(t)) \in \mathcal{X}, \tag{9.9}$$

where $\mathbf{p}^{a_i}(t)$ is the trajectory of car a_i.

9.2.2 Modeling Highway Navigation

The representation of the environment in the context of highway-driving is a set of lanes, in comparison to a grid-based representation presented in the previous chapters. The lanes

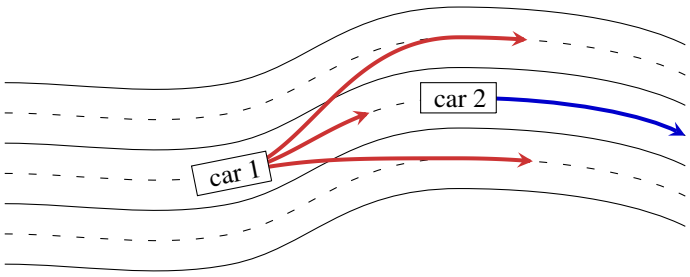


Figure 9.2: Illustration of three distinct discrete choices for *car 1*: Stay in lane behind *car 2*, move to left lane, move to right lane. The blue line illustrates the predicted movement of *car 2*, the red lines the best trajectory within each of the three classes.

are defined as a series of waypoints in the middle of each lane, to which we refer as center line. In our experiments, the lane information results from a previously mapped road network. However, our algorithm is independent of the source of the lane information. We could therefore also apply a lane classification method that runs online on the autonomous car [142, 145, 151, 158]. In contrast of representing the robots as points, we model the 2D shape of the cars to guarantee collision-free trajectories. For the autonomous car as well as all other vehicles that are currently detected on the road, we use a polygon to represent their 2D shape.

In the following, we apply the model that we presented for mobile robot navigation in Chap. 5 to autonomous cars. This model comprises a two-stage decision process. The first stage corresponds to discrete navigation choices, for which we used homotopically distinct paths. In the context of autonomous cars, we consider slightly different discrete decisions. Here, we consider the choices of staying in lane, or changing lanes to the left, or to the right. When changing lanes there may also arise further discrete decisions, for example between which cars we should merge.

For each discrete choice, there is exists a continuous distribution over trajectories within the corresponding class. Each of these continuous trajectories captures one possible behavior of the car for a certain time interval. We model this distribution in terms of an exponential family distribution that depends on features. Fig. 9.2 illustrates an environment with three lanes and two cars, where our method controls *car 1*. The red trajectories show the optimal trajectories within each of the three discrete choices to stay in lane, change to the left lane, of change to the right. The blue trajectory shows the predicted trajectory for *car 2*. For robot navigation in populated environments, we considered the cooperative navigation behavior of all agents. When navigating on a highway, it could be dangerous to always assume cooperative behavior. For example, when we approach an other car that has a lower velocity on the same lane, the optimal, cooperative behavior would be for the other car to yield and change lanes. However, we cannot plan with this cooperative assumption in practice, since it could potentially lead to a fatal accident if the other car does not follow this joint plan. In our experiments we apply a constant velocity model for all other vehicles. In particular, we assume that all other vehicles stay in their lane and keep the current velocity. An additional intention classification, which detects potential lane changes of other cars would be necessary for

safe navigation with our method in practice.

A further difference to the model used in the previous chapters lies in the degrees of freedom of the trajectories that we optimize. In the previous chapters, we set a specific target for the robot and all other agents. During optimization, we fixed the first spline control point as well as the position of the last control point for all trajectories. All other control points as well as the total travel time of the agents were variables that we optimize. This ensured that the agents eventually reached their target position. For navigation on highways we have a more constrained environment, since all cars are supposed to stay close to the center line of one of the lanes on the highway and maintain a given direction at a certain desired velocity. Therefore, we let our optimization method also change the position of the last control point but fix the total travel time. This results in a constant planning time interval during navigation. Using features that penalize the distance to the desired center line and that capture the desired velocity, the optimized trajectories stay close to the desired lane.

9.2.3 Features

In the following, we propose features that capture relevant properties of driving styles such as velocities, acceleration and distances to other agents, the distance to the desired lane, and the desired speed. Each feature takes as input a composite trajectory that represents the trajectories of all vehicles in a certain time interval, but computes statistics for a specific vehicle a.

Acceleration

The acceleration of the car has an important influence on the subjective feeling of comfort. Integrating the squared acceleration over the trajectory yields the feature

$$f^a_{\text{acc}}(\mathbf{x}) = \int_t \|\ddot{\mathbf{p}}^a(t)\|^2 \; dt. \tag{9.10}$$

Normal acceleration

Experimental studies showed that passengers of vehicles react very sensitive to lateral accelerations [63]. Therefore, we use a feature that represents the acceleration perpendicular to the direction of the lane.

$$f^a_{\text{normal\_acc}}(\mathbf{x}) = \int_t \left(d_x(\mathbf{p}^a(t))\ddot{p}^a_y(t) - d_y(\mathbf{p}^a(t))\ddot{p}^a_x(t)\right)^2 \; dt, \tag{9.11}$$

where $\mathbf{d}(\mathbf{p})$ is the direction vector of the lane segment closest to the position $\mathbf{p}$.

Jerk

For the comfort of passengers the jerk along the trajectory is also an important quantity:

$$f^a_{\text{jerk}}(\mathbf{x}) = \int_t \|\dddot{\mathbf{p}}^a(t)\|^2 \; dt. \tag{9.12}$$

Normal jerk

Similar to a feature for the normal acceleration, we also capture the lateral jerk perpendicular to the direction of the lane

$$f^a_{\text{normal\_jerk}}(\mathbf{x}) = \int_t \left(d_x(\mathbf{p}^a(t)) \dddot{p}^a_y(t) - d_y(\mathbf{p}^a(t)) \dddot{p}^a_x(t) \right)^2 \, dt. \tag{9.13}$$

Curvature

Since a real vehicle has typically a limited turning circle, we additionally introduce a feature that represents the squared curvature

$$f^a_{\text{curvature}}(\mathbf{x}) = \int_t \|\kappa^a(t)\|^2 \, dt. \tag{9.14}$$

Desired speed

To represent the deviation from a desired speed, we use the feature

$$f^a_{\text{velocity}}(\mathbf{x}) = \int_t \| v^a_{\text{des}} \mathbf{d}(\mathbf{p}^a(t)) - \dot{\mathbf{p}}^a(t) \| \, dt, \tag{9.15}$$

where v^a_{des} is the desired velocity of vehicle a. The desired speed could either be the speed limit on the given highway, or a lower individual speed that is comfortable for the user.

Lane

When driving on highways, the vehicle should drive close to the center of the desired lane. We represent this property by the feature

$$f^a_{\text{lane}}(\mathbf{x}, k) = \int_t \| \mathbf{l}(\mathbf{p}^a(t), k) - \mathbf{p}^a(t) \| \, dt. \tag{9.16}$$

This feature is parameterized with a lane index k. The function $\mathbf{l}(\mathbf{p}, k)$ returns the point on the centerline of lane k that is closest to point $\mathbf{p}$.

Collision avoidance

Obviously the distance to other vehicles is important to avoid collisions. Therefore, we introduce the feature

$$f^a_{\text{distance}}(\mathbf{x}) = \int_t \frac{1}{\|\mathbf{c}^a(t)\|^2} \, dt, \tag{9.17}$$

where $\mathbf{c}^a(t)$ is the closest distance of vehicle a to an other vehicle at time t. The value of this feature increases as the car gets closer to any other vehicle. To compute $\mathbf{c}^a(t)$, we determine the closest distance between the polygon shaped 2D outline of a car a and all other vehicles in the vicinity. Here, we can exploit the fact that the relative position between two vehicles does typically not change much from one sampling point to another.

We apply an adaptation of the *closest feature tracking* method [97] to 2D. This technique computes the edge or vertex of a polygon that is closest to another polygon and re-uses the previous results iteratively, which improves the run-time compared to the brute force approach.

Following distance

When following other vehicles, we aim at keeping a safety distance which is greater than the minimum allowed distance between two vehicles perpendicular to the lanes. To account for the distance to the following vehicle on the same lane, we propose the feature

$$f^a_{\text{follow}}(\mathbf{x}) = \int_t \max\left(0, \hat{d}^a - d^a(t)\right)\, dt, \tag{9.18}$$

where $d(t)$ is the current distance to the next vehicle along the lane and $\hat{d}$ the desired following distance.

In addition, in accordance with the model described in Chap. 5, we use propose a set of features for the discrete navigation choices.

Cost of the most likely composite trajectory

We use a feature accounting for the cost of the most likely trajectory within each class

$$g_{\text{ml\_cost}}(\psi) = \min_{\mathbf{x}\in\psi} \boldsymbol{\theta}^T \mathbf{f}(\mathbf{x}), \tag{9.19}$$

where the classes $\psi \in \Psi$ correspond to lane changes in the context of highway navigation.

Changing lanes

Furthermore, we add a feature that represents a penalty for changing lanes:

$$g_{\text{change}}(\psi) = \begin{cases} 0, & \text{if } \psi = \psi_{\text{current}} \\ c, & \text{else} \end{cases} \tag{9.20}$$

The two features bias the choice of discrete navigation decisions towards the class that contains the most likely trajectory, i.e., the class in which the optimized trajectory has the lowest cost $\boldsymbol{\theta}^T \mathbf{f}(\mathbf{x})$. However, it avoids hysteresis by adding a penalty for changing lanes.

9.3 Learning Individual Driving Styles

For learning navigation styles from demonstration, we apply IRL, similar to the techniques that we presented in Chap. 6. For convenience, we shortly recap the principle of maximum entropy and feature matching in the context of car navigation.

9.3.1 Maximum Entropy Inverse Reinforcement Learning

Given a set of N observed trajectories $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_N$ our goal is to learn a model that explains the observations and that is capable of generating trajectories of similar characteristics in other situations. However, it is not obvious what *similar* means in this context. For example, if we have training examples from a certain highway, an unbiased learning algorithm without any assumptions might learn that it is important to drive on this exact highway. However, we rather want a system that learns acceleration profiles or local obstacle avoidance behavior and that is able to generate similar trajectories on any highway.

To introduce such domain knowledge, we propose a feature-based learning algorithm. Each feature f_k is a function that maps a trajectory to a real value, capturing some relevant characteristic. The vector of all features $\mathbf{f}$ is thus a function that maps trajectories to a real vector

$$\mathbf{f} : \mathbf{x} \mapsto (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})) \in \mathbb{R}^n. \tag{9.21}$$

In this way, the empirical feature values of the observed trajectories $\tilde{\mathbf{f}} = \sum_i^N \mathbf{f}(\tilde{\mathbf{x}}_i)$ encode relevant properties of the demonstrations, such as accelerations, velocities, or distances to other vehicles.

Our goal is to find a generative model that yields trajectories that are similar to the observations, where the features serve as a measure of similarity. More specifically, given a probabilistic model that yields a probability distribution over trajectories $p(\mathbf{x} \mid \boldsymbol{\theta})$, our goal is to find the parameters $\boldsymbol{\theta}$ such that the expected feature values match the observed empirical feature values, i.e.,

$$\mathbb{E}_{\mathrm{p}_{\boldsymbol{\theta}}(\mathbf{x})}[\mathbf{f}] = \tilde{\mathbf{f}}\,. \tag{9.22}$$

In general, there are many distributions with this property. Within the class of all distributions that match features, Ziebart et al. [159] propose to select the one that maximizes the entropy. Following the principle of maximum entropy, this distribution is least biased estimation on the given data. The solution of the constrained optimization problem of optimizing the entropy given the constraints in Eq. (9.22) has the form

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \exp\left(-\boldsymbol{\theta}^T \mathbf{f}(\mathbf{x})\right). \tag{9.23}$$

One can interpret $\boldsymbol{\theta}^T \mathbf{f}(\mathbf{x})$ as a cost function, where agents are exponentially more likely to select trajectories with lower cost. Unfortunately, it is not possible to compute $\boldsymbol{\theta}$ analytically, but we can compute the gradient of the Lagrange function of the constraint optimization problem with respect to $\boldsymbol{\theta}$. It can be shown that this gradient is the difference between the expected and the empirical feature values

$$\mathbb{E}_{\mathrm{p}_{\boldsymbol{\theta}}(\mathbf{x})}[\mathbf{f}] - \tilde{\mathbf{f}}\,. \tag{9.24}$$

There is an intuitive explanation for this gradient: when the expected value $\mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})}[f_k]$ for a feature f_k is too high, we should increase the corresponding weight θ_k, which in turn assigns lower likelihood to any trajectories with high feature values $f_k(\mathbf{x})$. As a result, the expected feature value $\mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})}[f_k]$ decreases. Sec. 6.2 provides more details about the derivation of this learning algorithm.

9.3.2 Maximum Likelihood Approximation

The main challenge of our learning approach is to compute the expected feature values $\mathbb{E}_{\mathrm{p}_{\boldsymbol{\theta}}(\mathbf{x})}[\mathbf{f}]$. For high-dimensional distributions it is generally not possible to compute the integral

$$\mathbb{E}_{\mathrm{p}_{\boldsymbol{\theta}}(\mathbf{x})}[\mathbf{f}] = \int p(\mathbf{x} \mid \boldsymbol{\theta}) \mathbf{f}(\mathbf{x}) d\,\mathbf{x} \tag{9.25}$$

analytically. In Chap. 6 we described how to sample trajectories from the high-dimensional space of trajectories using a Hamiltonian Markov chain Monte Carlo method. However, this sampling method is computationally very expensive. Instead of computing the expectations by sampling [85], a possible approximation of the expected feature values is to compute the feature values of the most likely trajectory:

$$\mathbb{E}_{p}[\mathbf{f}] \approx \mathbf{f}(\arg\max_{\mathbf{x}} p(\mathbf{x} \mid \boldsymbol{\theta})). \tag{9.26}$$

The resulting learning method is also known as *inverse optimal control*. With this approximation, we assume that the demonstrations are in fact generated by minimizing a cost function. This is in contrast to the assumption that demonstrations are samples from a probability distribution. Our experiments suggest that this approximation is suitable in the context of learning individual driving styles on highways.

9.3.3 Learning Individual Navigation Behavior

In this section, we outline the learning process when approximating the expectations using maximum likelihood trajectories, as described in Sec. 9.3.2. Given a set of demonstrated composite trajectories $\{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_N\}$, the following steps lead to the desired feature weights $\boldsymbol{\theta}$ that capture the continuous navigation behavior.

1. Predict the movement of other vehicles on the road for all composite trajectories $\tilde{\mathbf{x}}_i$.
2. Compute the empirical feature vector $\tilde{\mathbf{f}} = \sum_i \mathbf{f}^{\mathrm{a}}(\tilde{\mathbf{x}}_i)$ for all demonstrated trajectories, where a is the car from which we want to learn the navigation style.
3. Initialize the weight vector $\boldsymbol{\theta}$ with arbitrary values.
4. For each demonstrated trajectory, fix the position, velocity and acceleration at the first control point of $\mathbf{p}^a$, as well as all the control points of trajectories $\mathbf{p}^{b \neq a}$. Set the desired lane k for feature $f^a_{\text{lane}}(\mathbf{x}, k)$ to the lane closest to the last control point of the observed trajectory $\mathbf{p}^a$. Then optimize all demonstrated trajectories with respect to the cost function $\boldsymbol{\theta}^T \mathbf{f}(\mathbf{x})$. We denote the optimized trajectories by $\{\mathbf{x}^{\boldsymbol{\theta}}_1, \ldots, \mathbf{x}^{\boldsymbol{\theta}}_N\}$.
5. Compute the approximated expected feature values by evaluating the feature functions for all optimized trajectories $\mathbb{E}_{\mathrm{p}_{\boldsymbol{\theta}}(\mathbf{x})}[\mathbf{f}] \approx \mathbf{f}^{\boldsymbol{\theta}}_{\text{ML}} := \sum_i^N \mathbf{f}^a(\mathbf{x}^{\boldsymbol{\theta}}_i)$.
6. The gradient for the optimization is given by $\mathbf{f}^{\boldsymbol{\theta}}_{\text{ML}} - \tilde{\mathbf{f}}$, as stated in Eq. (9.24). Use this gradient to update the current estimation of the weight vector $\boldsymbol{\theta}$.
7. Repeat from step 4 until convergence.

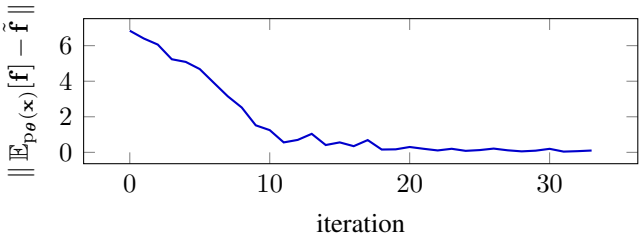


Figure 9.3: Evolution of the norm of the difference between the empirical feature values and the expected feature values during learning from a dataset of 20 observed trajectories.

9.4 Navigating an Autonomous Vehicle Using the Learned Model

We integrated our method into a planning framework for autonomous vehicles. At each planning cycle, we update the state of other perceived vehicles on the highway given the current sensor readings. We predict the future behavior of these vehicles assuming constant speed along their respective lanes. Based on this prediction, we use the method described above to optimize the trajectory of the autonomous vehicle based on the learned cost function.

We propose to maintain a set of optimized trajectories within each class, similar to the approach presented in Chap. 5. In the context of highway navigation, each class corresponds to either staying in lane, or a lane change maneuver. If we do not have a representative for one of these classes, we initialize this class with an initial guess, which is a trajectory from the current position of the car to the desired lane. At each planning cycle we fix the position $\mathbf{p}_0^a$, velocity $\dot{\mathbf{p}}_0^a$, and acceleration $\ddot{\mathbf{p}}_0^a$ of the first control point in all classes according to the trajectory that we sent to the controller in the previous planning cycle. (Find details on online path planning in Sec. 5.2.5). The remaining spline control points $\mathbf{p}_j^a$, $\dot{\mathbf{p}}_j^a$, and $\ddot{\mathbf{p}}_j^a$ for $j \in \{1, \ldots, m\}$ are variables for the optimization using the learned cost function.

In particular, the feature $f_{\text{lane}}^a(\mathbf{x}, k)$ specifies to which lane a trajectory converges during optimization. We set the value of k of this feature individually for each class. We optimize the trajectories in each class in parallel, using the optimization algorithm RPROP [126] and select the class that has the lowest cost $\boldsymbol{\Theta}^T \mathbf{g}(\psi)$ for navigation.

The structure of the leaned cost function yields smooth, comfortable trajectories which keep a safe distance to obstacles. When the navigation system detects a slower vehicle ahead, our system predictively computes trajectories for different maneuvers. Depending on the cost of the different variants, the autonomous car either decelerates and keeps a safe distance, or it changes the lane to overtake the slower vehicle. In this case, the features that capture the curvature, normal acceleration, and normal jerk guarantee a smooth lane change that shows characteristics as demonstrated before.

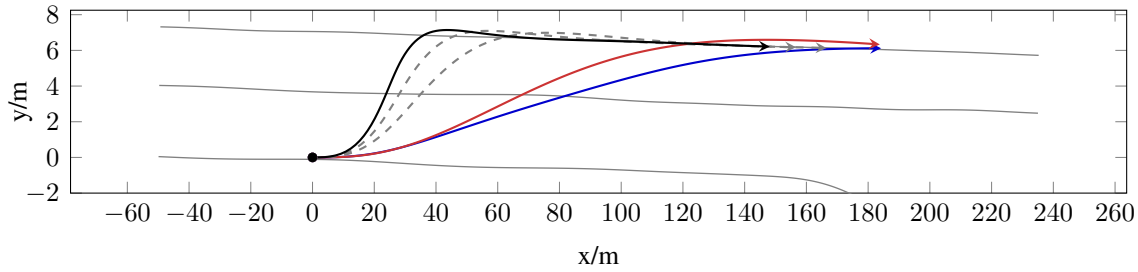


Figure 9.4: Demonstrated trajectory (blue), the initial guess (black), and the optimized trajectory with the final learned policy (red). The trajectory shows a change of two lanes to the left. The dashed lines illustrate intermediate policies during the learning phase.

9.5 Experiments

In this section we evaluate our method for learning autonomous navigation behavior from demonstration. In particular, we present experiments in which we learn an individual driving style from behavior recorded during real highway driving. Furthermore, we show the applicability of our method to navigating an autonomous car in a realistic highway simulation framework.

9.5.1 Data Acquisition

To record training data, we used the car shown in Fig. 9.1 that is equipped with a range of sensors, which allow us to localize in an existing 3D map of the environment. Furthermore, the car observes and tracks other vehicles in its vicinity. For the experiments in this section, we recorded the driving style of a driver on a US highway in normal traffic. From this dataset we manually selected 20 maneuvers of about 10 s length each as training observations, including lane changes and changes of speed with a variety of different start- and target velocities. Each observation consists of the trajectory of the car, the trajectories of all other vehicles in the vicinity, and the lane information. For each sample, we set the desired speed according to the last observed speed. Similarly, we set the desired lane to the closest lane at the end of the observed trajectory. To avoid manual pre-selection of the training data, we could also obtain samples in an interactive training session, where the car issues commands such as “perform a lane change to the left lane now”.

9.5.2 Learning Individual Navigation Styles

The goal of the learning method is to find a weight vector $\boldsymbol{\theta}$ of a policy that explains the observed trajectories. Fig. 9.3 shows the deviation of the empirical features and the approximated, expected feature values during learning from the set of observed trajectories. After about 30 iterations, the feature weights converge.

Fig. 9.4 shows an observed trajectory of changing two lanes to the left. In addition to the observed trajectory, the figure shows the initial guess as well as the trajectory optimized with the learned policy. The learned policy yields a trajectory with similar

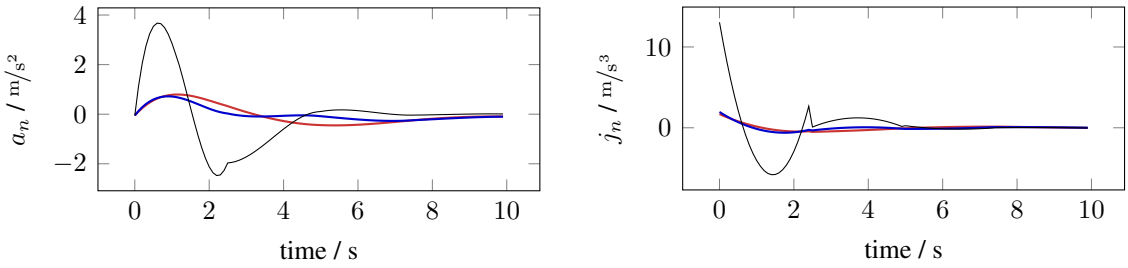


Figure 9.5: Normal acceleration and normal jerk of a demonstrated trajectory (blue), the initial guess (black), and the optimized trajectory with the final learned policy (red).

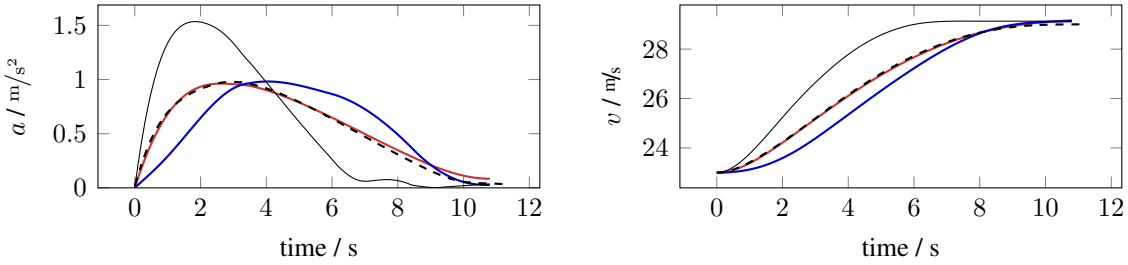


Figure 9.6: Speed (left) and acceleration (right) of a demonstrated trajectory (blue), the initial guess (black), and the optimized trajectory with the final learned policy (red). The dashed line shows the trajectory of a car in a realistic simulation that accelerates from $23\,\mathrm{m/s}$ to $29\,\mathrm{m/s}$ using the learned policy.

characteristics compared to the observation. This is also shown in Fig. 9.5 which plots the normal acceleration and the normal jerk during the lane change maneuver.

Fig. 9.6 shows the velocity and acceleration profiles for a different trajectory, where the car accelerates from an initial velocity of $23\,\mathrm{m/s}$ to a desired velocity of $29\,\mathrm{m/s}$. The plots show the observation, the initial guess, and the final learning result. Similar to the lane change behavior, the learned method better replicates the behavior compared to the initial guess.

In our experiments, the resulting trajectories after optimization do not fit the observation perfectly. The reason for this is twofold. First, the learned policy is an average over the set of different sample trajectories. Second, the observations do not exactly meet the optimality criteria for any cost function that is a linear combination of the features we use. However, the experimental results suggest that our algorithm learns the magnitude of the quantities that contribute to the comfort of the users.

9.5.3 Autonomous Driving

We applied the learned policy to autonomous navigation, as described in Sec. 9.4. Our system continuously computes the trajectory with lowest cost with $5\,\mathrm{Hz}$ and uses the trajectory to control a car in a realistic simulation environment. To show that the learned characteristics also apply to online navigation, we changed the desired velocity of the simulated car from $23\,\mathrm{m/s}$ to $29\,\mathrm{m/s}$ during navigation. This maneuvers is similar to the demonstration sample shown in Fig. 9.6. This parameter change causes the car to accelerate to the new desired velocity. The dashed line in Fig. 9.6 shows the acceleration

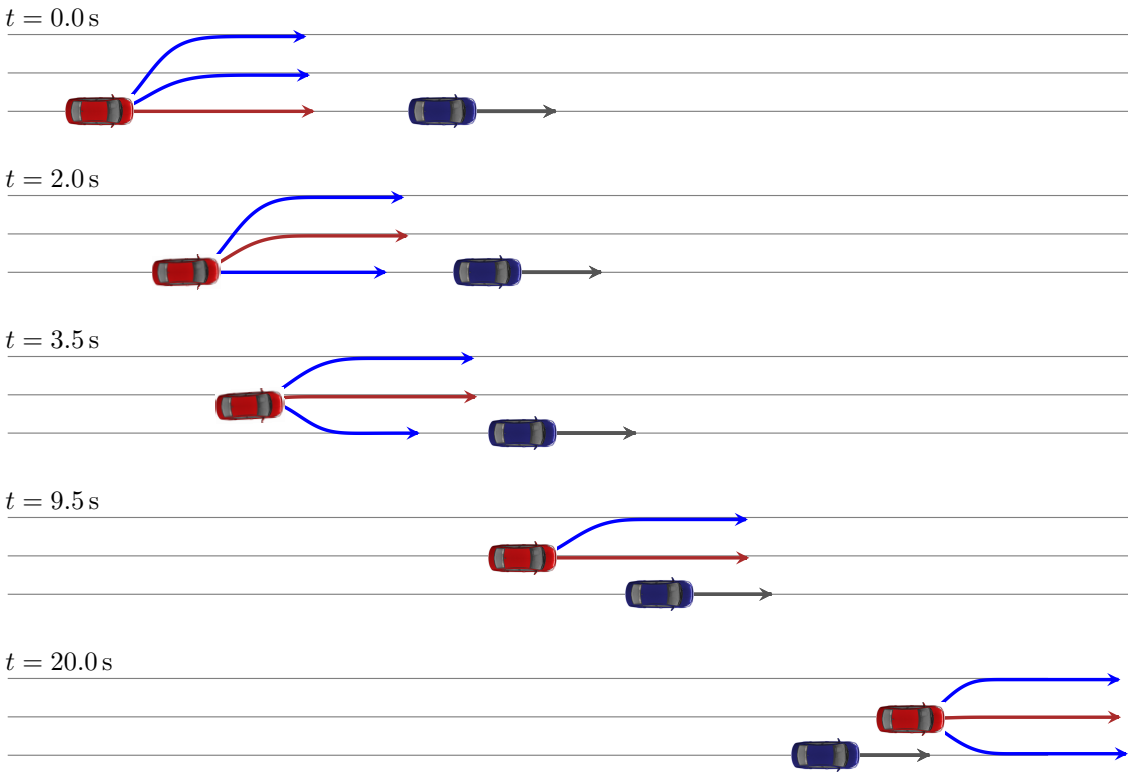


Figure 9.7: Simulation experiment with two cars on a highway with three lanes. The gray lines correspond to the centerlines of the lanes. The car controlled by our method (red) decides to stay in lane as long as the predicted trajectory is not affected by the slower car in front (blue). As soon as the prediction for staying in the current lane implies a deceleration, our method initiates a lane change. During the passing maneuver, moving to the right is not viable option. The corresponding trajectory for a lane change to the right is therefore only considered after finishing the passing maneuver. The gray arrow shows the constant velocity prediction for the blue car.

of the simulated car during this maneuver. The acceleration profiles coincide with the trajectories optimized offline for the same acceleration task. This experiment suggests that the learned policy is suitable to autonomously control a car with similar characteristics as observed from real drivers.

Fig. 9.7 shows an illustration of the navigation behavior on a highway with multiple lanes, where a slower car blocks the current lane. The figure shows the optimized trajectories for each of the three classes that correspond to the lanes of the highway. Our algorithm selects the red trajectory for navigation since it has the lowest assigned cost. First, the car in front does not interfere with the planned trajectory over the whole planning horizon, causing the car controlled by our method to stay in lane. As soon as our method predicts a necessary deceleration on the current lane, the cost distribution shifts and the lane change becomes the best option. During the overtaking, merging to the right has a very high cost and is not considered in this situation. After the passing maneuver, the third option gets viable again, and is added to the set of variants as a result. Fig. 9.8 shows a snapshot of the realistic highway navigation during the lane change maneuver.

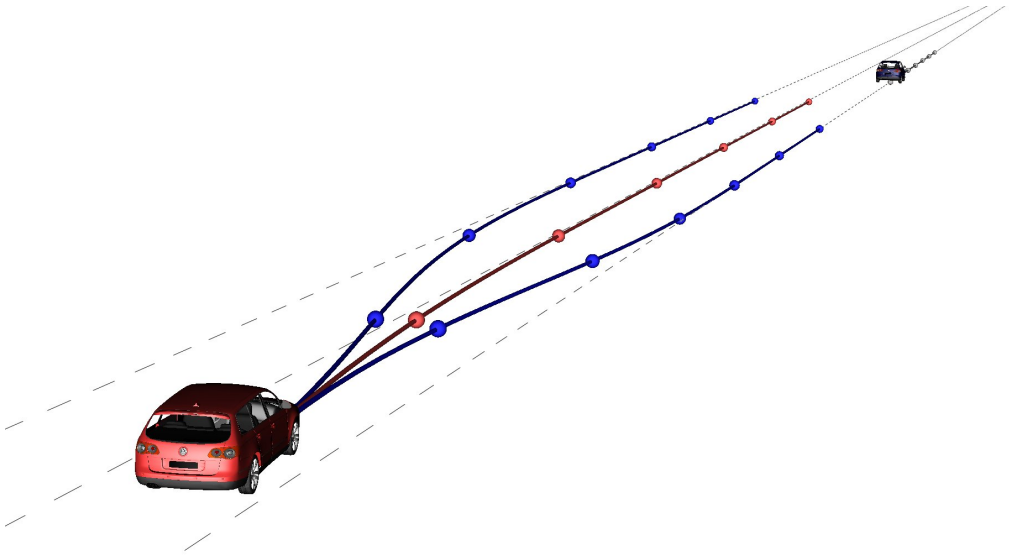


Figure 9.8: Snapshot of our realistic highway simulation. The figure shows the three optimized trajectories in distinct classes that our method computed for the red car. The snapshot corresponds to $t = 2\,\mathrm{s}$ in Fig. 9.7. The spheres along the trajectories illustrate the spline control points.

9.6 Related Work

In the past, machine learning techniques in various fashions have been used to improve the performance of autonomous vehicles. In 1991, Pomerleau [119] used neural networks to learn to steer a vehicle on a highway by observing a person drive. The network receives live input from a camera on the vehicle and learns a control function of the steering angle to keep the vehicle on track. Riedmiller et al. [127] use reinforcement learning to learn a steering controller from scratch. The input is the deviation from the track and the speed of the vehicle. Their approach learns a controller that is able to navigate the vehicle on the track while driving a real car for $25\,\mathrm{min}$. In this work, our goal is to learn a more complex behavior that does not only allow the vehicle to stay in the current lane but to maintain a desired speed, keep a safe distance to other vehicles, and also change lanes.

Abbeel and Ng [1] motivate the use of IRL for learning driving styles: "When driving, we typically trade off many different desiderata [...] To specify a reward function for the driving task, we would have to assign a set of weights stating exactly how we would like to trade off these different factors. Despite being able to drive competently, the authors do not believe they can confidently specify a specific reward function for the task of *driving well*." Abbeel and Ng apply their method to learn different behavior styles on a highway simulation with three lanes and five discrete actions.

Babes et al. [7] introduced maximum likelihood inverse reinforcement learning. Similar to Ziebart et al. [159] they maximize the likelihood of the data assuming an exponential family distribution. They furthermore introduce a method to automatically cluster observed behavior styles and learn individual feature weights for each cluster. Chen et al. [21] propose a method to automatically discover a set of relevant features in IRL and Levine et al. [96] use Gaussian processes to learn the reward as a nonlinear function.

In these approaches, the respective IRL algorithms are used to learn behavior patterns on a discrete highway simulator with a small number of states and actions, similar to Abbeel and Ng [1]. Silver et al. [134] apply maximum margin planning (MMP) to learn more complex behavior for autonomous vehicles and show experiments in which learning by demonstration significantly outperforms manual tuning of the parameters. MMP is an IRL variant that aims to minimize the margin between the observations and the optimal trajectory in an MDP by adjusting the cost of discrete state-action pairs.

In contrast to a discrete state and action space, we consider trajectories in continuous state spaces. This allows our method to learn higher order properties such as lateral jerk, which is important for the comfort of users. Furthermore, the continuous trajectories computed by our approach are suitable for online control of an automated car.

The method we utilize in this chapter builds on previous approaches for learning pedestrian navigation behavior as presented in Chap. 5 and Chap. 6. However, in the context of highway driving we encounter different preconditions and challenges. For example, in contrast to free movement in all directions, vehicles need to follow their respective lanes. Furthermore, continuous acceleration and bounded jerk are necessary for driving comfort especially when driving at a high speed.

Schwarting and Pascheka [132] propose a system to recursively resolve conflicts, by cooperative decision making that is based on a cost function similar to ours. In our experiments, we use a constant velocity for the other vehicles on the road. However, our framework would also allow to integrate cooperative behavior, i.e., to also predict the behavior of others based on the underlying cost function as presented in Chap. 5.

The approach presented in this chapter relies on a finite-dimensional representation of trajectories. We choose to encode the 2D position along the trajectories using quintic splines, similar to Sprunk et al. [136]. They optimize trajectories with respect to a user-defined cost function for navigating a holonomic mobile robot. Gulati [46] proposes an alternative trajectory representation. Instead of the positions along the trajectory, Gulati parametrizes a trajectory using orientation and speed. This allows boundary conditions with zero speed. However, in contrast to an efficient closed form representation, this parametrization requires numerical integration to compute the position along the trajectory. The learning approach proposed in this chapter is independent of the representation of trajectories, as long as it allows computation of the features and their derivatives.

In our experiment, we use pre-build maps of the environment for localization and lane detection. Other authors proposed to extract lane markings using the on-board sensors of the car. Vacek et al. [142] use a particle filter to detect lanes given measurements from virtual scan lines in a 2D camera image. Wang et al. [151] fit a spline-based lane model into the road image. Zhao et al. [158] extend the method of Wang et al. by implementing a Kalman filter to track the control points of the spline. Veit et al. [145] presents a comparison on different features for extracting road markings from images.

9.7 Conclusion

In this chapter, we presented an inverse reinforcement learning method to learn individual driving styles for self-driving cars from demonstration. To this end, we adapted the feature-based approaches for mobile robot navigation described earlier in this theses. In particular, to capture the relevant properties of highway driving, we proposed a set of features that model distances to other vehicles, the distance to the desired lane as well as higher order properties such as velocities and accelerations. By matching observed empirical feature values with the expected feature values of the model, the proposed method allows us to learn a policy that represents an individual driving style. We model discrete navigation decision as well as continuous properties, which allows a car to navigate autonomously on highways with several lanes. Experiments carried out with observed trajectories from a real car suggest that our method is able to reliably learn policies from demonstration. Furthermore, experiments using a realistic highway simulation demonstrate the applicability of the learned policies for autonomous navigation.

Chapter 10

Discussion

In this thesis we presented methods that allow a mobile robot to navigate in a socially more compliant way compared to traditional path planning methods. To this end, we proposed models that give a robot a better understanding of the natural navigation behavior of interacting agents. This allows the robot to predict the behavior of nearby pedestrians, and to plan a socially compliant path to a goal location at the same time. The key idea of our novel model is that we explicitly integrate into the planning process the reactions of pedestrians to the actions of the robot. Furthermore, during navigation a mobile robot can use the models to predictively plan trajectories, which enables it to adapt its behavior at an early stage of the interaction, in contrast to reactive planning methods.

After introducing traditional path planning methods, we presented a method for online skeleton tracking of humans. Such a method enables a robot to estimate the pose of humans, for learning their behavior, or for direct interaction. In contrast to existing methods, our approach is fully automatic and does not need any user input such as labeling the optical markers. In comparison experiments, we showed that our method outperforms state-of-the-art commercial products.

As a basis for the models of interactive navigation behavior presented in this thesis, we introduced the topological concept of homotopy, and outlined its application to path planning. In particular, we presented an algorithm that efficiently computes homotopically distinct paths using an abstract graph representation of the environment. Our models utilize trajectories in different homotopy classes to capture discrete navigation decisions such as evading an obstacle on the left, or on the right. Within each of these classes, we capture the interactive behavior of multiple agents in terms of a probability distribution over their trajectories. This exponential family distribution depends on a set of features that capture important properties of an intention-driven behavior, i.e., reaching a goal position quickly but comfortably.

To model the navigation behavior of interacting agents, we described a two-stage decision process, where the agents first select a discrete class, and in a second step a manifestation of the behavior within this class in the form of a continuous trajectory. A mobile robot can use such a model to predict the behavior of nearby pedestrians, and at the same time to plan its own paths. In contrast to traditional path planning approaches, this allows the robot to reason about the effect of its actions to the pedestrians, and to adapt its behavior appropriately. In an extensive set of experiments, also including a real autonomous wheelchair, we evaluated our novel methods and compared them to related

approaches.

Our models allow us to conveniently adapt the behavior to the application at hand by changing the individual weights of the features. In many situations, however, manual tuning of the feature weights may not be desirable. Therefore, we proposed an approach to teach a robot a desired behavior by demonstration. Using this method, the robot can learn the characteristics of its desired behavior by tele-operation, which is a convenient way of programming a robot, especially for non-experts.

Our models are also applicable to shared autonomy navigation schemes. These methods allow users to control the robot on a high level, leaving the low level control to the autonomous system, which facilitates controlling complex systems while reducing the cognitive burden to the user. We presented approaches in which the user influences the navigation behavior by choosing from a set of qualitatively distinct navigation paths. Furthermore, our methods allows a mobile robot to follow a human leader, who can freely walk in the environment. In contrast to existing methods, we integrate a prediction of the human's behavior into planning, which allows the robot to fulfill its task more accurately without hindering the leading human.

Our method is not only applicable to domestic robots. The general idea of continuous, interactive navigation behavior combined with discrete decisions also applies to self-driving cars. We described a model of optimization-based autonomous navigation on highways. Furthermore, our approach also enables us to teach a car an individual behavior. In this way, a user can teach the car by driving manually, instead of tedious parameter tuning by an expert.

Throughout this thesis, we presented a theoretical description of the proposed methods, a discussion of related work and extensive experiments that back up the claims made in the different chapters. The contribution of this thesis is a set of techniques that enable mobile robots to navigate in a socially compliant way. The key idea is that we capture not only the isolated behavior of the robot, but also its effects on the humans. Furthermore, we predict the trajectories of the humans, and plan trajectories for the robot over the full planning time horizon. This enables the robot to plan far-sighted, in contrast to reactive methods that are prone to sudden, uncomfortable evasive maneuvers. The methods proposed in this thesis contribute to a new generation of robots that are able to take over more and more services in the direct vicinity of humans.

List of Figures

Bibliography

[1] P. Abbeel and A. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, Banff, Alberta, Canada, 2004.

[2] G. Arechavaleta, J.-P. Laumond, H. Hicheur, and A. Berthoz. An optimality principle governing human walking. *IEEE Transactions on Robotics (T-RO)*, 24(1): 5–14, 2008.

[3] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[4] A. Aristidou and J. Lasenby. Real-time marker prediction and CoR estimation in optical motion capture. *The Visual Computer*, 29:7–26, 2013.

[5] C. G. Atkeson and S. Schaal. Robot learning from demonstration. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1997.

[6] F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

[7] M. Babes, V. N. Marivate, K. Subramanian, and M. L. Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2011.

[8] T. Balch and M. Hybinette. Social potentials for scalable multi-robot formations. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2000.

[9] B. Banerjee and B. Chandrasekaran. A framework of Voronoi diagram for planning multiple paths in free space. *Journal of Experimental & Theoretical Artificial Intelligence*, 25(4):457–475, 2013.

[10] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun. Learning motion patterns of people for compliant robot motion. *International Journal of Robotics Research*, 24(1):31–48, 2005.

[11] S. Bhattacharya, M. Likhachev, and V. Kumar. Topological constraints in search-based robot path planning. *Autonomous Robots*, 33(3):273–290, 2012.

[12] C. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[13] S. Bitgood and S. Dukes. Not another step! Economy of movement and pedestrian choice point behavior in shopping malls. *Environment and Behavior*, 38(3):394–405, 2006.

[14] W. Böhm, G. Farin, and J. Kahmann. A survey of curve and surface methods in CAGD. *Computer Aided Geometric Design*, 1(1):1 – 60, 1984.

[15] A. Bonarini, S. Ceriani, G. Fontana, and M. Matteucci. Introducing LURCH: A shared autonomy robotic wheelchair with multimodal interfaces. In *Proceedings of IROS 2012 Workshop on Progress, Challenges and Future Perspectives in Navigation and Manipulation Assistance for Robotic Wheelchairs*, 2012.

[16] A. Boularias, J. Kober, and J. R. Peters. Relative entropy inverse reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[17] T. E. Carlson and J. d. R. Millán. Brain-controlled wheelchairs: A robotic architecture. *IEEE Robotics & Automation Magazine*, 20(1):65–73, 2013.

[18] T. E. Carlson, L. Tonin, S. Perdikis, R. Leeb, and J. d. R. Millán. A hybrid BCI for enhanced control of a telepresence robot. In *Proceedings of Annual International Conference of the Engineering in Medicine and Biology Society*, 2013.

[19] E. Catmull and R. Rom. A class of local interpolating splines. *Computer Aided Geometric Design*, 74:317–326, 1974.

[20] P. Cerveri, A. Pedotti, and G. Ferrigno. Kinematical models to reduce the effect of skin artifacts on marker-based human motion estimation. *Journal of Biomechanics*, 38(11):2228 – 2236, 2005.

[21] S. Chen, H. Qian, J. Fan, Z. Jin, and M. Zhu. Modified reward function on abstract features in inverse reinforcement learning. *Journal of Zhejiang University SCIENCE C*, 11(9):718–723, 2010.

[22] T. Chen, M. Ciocarlie, S. Cousins, P. M. Grice, K. Hawkins, K. Hsiao, C. Kemp, C.-H. King, D. Lazewatsky, A. E. Leeper, H. Nguyen, A. Paepcke, C. Pantofaru, W. Smart, and L. Takayama. Robots for humanity: A case study in assistive mobile manipulation. *IEEE Robotics & Automation Magazine, Special Issue on Assistive Robotics*, 20, 2013.

[23] Y.-Q. Chen and Z. Wang. Formation control: A review and a new consideration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.

[24] S. Y. Chiem and E. Cervera. Vision-based robot formations with Bézier trajectories. In *Intelligent Autonomous Systems*, 2004.

[25] H. Choset and J. Burdick. Sensor-based exploration: The hierarchical generalized Voronoi graph. *International Journal of Robotics Research*, 19(2):96–125, 2000.

[26] H. Christensen and E. Pacchierotti. Embodied social interaction for robots. In *Proceedings of the 2005 Convention of the Society for the Study of Artificial Intelligence and Simulation of Behavior*, Herfordshire, 2005.

[27] R. Contini. Body segment parameters II. *Artificial Limbs*, 16(1):1–19, 1972.

[28] P. Dario, B. Hannaford, and A. Menciassi. Smart surgical tools and augmenting devices. *IEEE Transactions on Robotics and Automation*, 19(5):782–792, 2003.

[29] A. Das, R. Fierro, V. Kumar, J. Ostrowski, J. Spletzer, and C. Taylor. A vision-based formation control framework. *IEEE Transactions on Robotics and Automation*, 18 (5):813–825, 2002.

[30] E. de Aguiar, C. Theobalt, and H.-P. Seidel. Automatic learning of articulated skeletons from 3D marker trajectories. In *International Symposium on Advances in Visual Computing (ISVC)*, 2006.

[31] E. de Queirós Vieira Martins, M. M. B. Pascoal, J. Luis, and J. L. E. dos Santos. The k shortest paths problem. Technical report, Universidade de Coimbra, Portugal, 1998.

[32] E. Demeester, E. V. Poorten, A. Hüntemann, J. D. Schutter, M. Hofmann, M. Rooker, G. Kronreif, B. Lau, M. Kuderer, W. Burgard, A. Gelin, K. Vanopdenbosch, P. V. der Beeten, M. Vereecken, S. Ilsbroukx, A. Fossati, G. Roig, X. Boix, L. V. Gool, H. Fraeyman, L. Broucke, H. Goessaert, and J. Josten. Robotic ADaptation to Humans Adapting to Robots: Overview of the FP7 project RADHAR. In *International Conference on Systems and Computer Science (ICSCS)*, Villeneuve d'Ascq, France, 2012.

[33] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[34] D. Demyen and M. Buro. Efficient triangulation-based pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2006.

[35] J. Desai, J. Ostrowski, and V. Kumar. Modeling and control of formations of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 17 (6):905–908, 2001.

[36] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[37] S. Duane, A. Kennedy, B. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222, 1987.

[38] J. Elander, R. West, and D. French. Behavioral correlates of individual differences in road-traffic crash risk: An examination of methods and findings. *Psychological Bulletin*, 113(2):279, 1993.

[39] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, 17(7):760–772, 1998.

[40] C. Fox. *An Introduction to the Calculus of Variations*. Courier Dover Publications, 1987.

[41] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[42] T. Fraichard and V. Delsart. Navigating dynamic environments with trajectory deformation. *Journal of Computing and Information Technology (CIT)*, 17(1): 27–36, 2009.

[43] W. Fulton. *Algebraic Topology: A First Course*. Springer, 1991.

[44] T. Geerinck, E. Colon, S. A. Berrabah, K. Cauwerts, and H. Sahli. Tele-robot with shared autonomy: Distributed navigation development framework. *Integrated Computer-Aided Engineering*, 13(4):329–345, 2006.

[45] E. Goffman. *Relations in Public: Microstudies of the Public Order*. New York: Basic Books, 1971.

[46] S. Gulati. *A framework for characterization and planning of safe, comfortable, and customizable motion of assistive mobile robots*. PhD thesis, University of Texas, 2011.

[47] S. Gulati, C. Jhurani, B. Kuipers, and R. Longoria. A framework for planning comfortable and customizable motion of an assistive mobile robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.

[48] S. Guy, M. Lin, and D. Manocha. Modeling collision avoidance behavior for virtual humans. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2010.

[49] E. Hall. *The Hidden Dimension*. Doubleday, 1966.

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

[51] W. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[52] D. Helbing and A. Johansson. Pedestrian, crowd, and evacuation dynamics. In *Encyclopedia of Complexity and Systems Science*, pages 1–28. Springer New York, 2014.

[53] D. Helbing and P. Molnar. Social force model for pedestrian dynamics. *Physical Review E (PRE)*, 51(5):4282–4286, 1995.

[54] R. Hepp and R. Debrunner. *Orthopädisches Diagnostikum*. Thieme, 7 edition, 2004.

[55] L. Herda, P. Fua, R. Plänkers, R. Boulic, and D. Thalmann. Using skeleton-based tracking to increase the reliability of optical motion capture. *Human Movement Science*, 20(3):313–441, 2001.

[56] L. Herda, R. Urtasun, P. Fua, and A. Hanson. Automatic determination of shoulder joint limits using quaternion field boundaries. *International Journal of Robotics Research*, 22(6):419–436, 2003.

[57] S. Hoogendoorn and P. Bovy. Simulation of pedestrian flows by optimal control and differential games. *Optimal Control Applications and Methods*, 24(3):153–172, 2003.

[58] W. Hu, X. Xiao, Z. Fu, D. Xie, T. Tan, and S. Maybank. A system for learning statistical motion patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1450–1464, 2006.

[59] J. Huang, S. Farritor, A. Qadi, and S. Goddard. Localization and follow-the-leader control of a heterogeneous group of mobile robots. *IEEE/ASME Transactions on Mechatronics*, 11(2):205–215, 2006.

[60] T. Igarashi and M. Stilman. Homotopic path planning on manifolds for cabled mobile robots. In *In Proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2010.

[61] T. Ikeda, Y. Chigodo, D. Rea, F. Zanlungo, M. Shiomi, and T. Kanda. Modeling and prediction of pedestrian behavior based on the sub-goal concept. In *Proceedings of Robotics: Science and Systems (RSS)*, Sydney, Australia, 2012.

[62] S. Ito, F. Endres, M. Kuderer, G. D. Tipaldi, C. Stachniss, and W. Burgard. W-RGB-D: Floor-plan-based indoor global localization using a depth camera and wifi. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Hong Kong, China, 2014.

[63] I. D. Jacobson, L. G. Richards, and A. R. Kuhlthau. Models of human comfort in vehicle environments. *Human factors in transport research*, 2:24–32, 1980.

[64] E. Jaynes. Where do we stand on maximum entropy. *Maximum Entropy Formalism*, pages 15–118, 1978.

[65] A. Johansson, D. Helbing, and P. Shukla. Specification of the social force pedestrian model by evolutionary adjustment to video tracking data. *Advances in Complex Systems (ACS)*, 10(2):271–288, 2007.

[66] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2011.

[67] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal. Learning objective functions for manipulation. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2013.

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

[69] N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for k shortest simple paths. *Networks*, 12(4):411–427, 1982.

[70] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, volume 2, 1985.

[71] S. Kim, K. Sreenath, S. Bhattacharya, and V. Kumar. Optimal trajectory generation under homology class constraints. In *CDC*, 2012.

[72] R. Kirby, R. Simmons, and J. Forlizzi. COMPANION: A constraint optimizing method for person-acceptable navigation. In *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2009.

[73] A. Kirk, J. O'Brien, and D. Forsyth. Skeletal parameter estimation from optical motion capture data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[74] K. M. Kitani, B. D. Ziebart, D. Bagnell, and M. Hebert. Activity forecasting. In *European Conference on Computer Vision (ECCV 2012)*, 2012.

[75] M. Klous and S. Klous. Marker-based reconstruction of the kinematics of a chain of segments: A new method that incorporates joint kinematic constraints. *Journal of Biomechanical Engineering*, 132(7):074501, 2010.

[76] R. A. Knepper and D. Rus. Pedestrian-inspired sampling-based multi-robot collision avoidance. In *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2012.

[77] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 1991.

[78] KPMG. Self-driving cars: The next revolution. http://www.kpmg.com/US/en/IssuesAndInsights/ArticlesPublications/Documents/self-driving-cars-next-revolution.pdf, 2012. Online, accessed November 26, 2014.

[79] KPMG. Self-driving cars: Are we ready? http://www.kpmg.com/US/en/IssuesAndInsights/ArticlesPublications/Documents/self-driving-cars-are-we-ready.pdf, 2013. Online, accessed November 26, 2014.

[80] H. Kretzschmar, M. Kuderer, and W. Burgard. Inferring navigation policies for mobile robots from demonstrations. In *Proceedings of the Autonomous Learning Workshop at the IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013.

[81] H. Kretzschmar, M. Kuderer, and W. Burgard. Predicting human navigation behavior via inverse reinforcement learning. In *The 1st Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, Princeton, NJ, USA, 2013.

[82] H. Kretzschmar, M. Kuderer, and W. Burgard. Learning navigation policies from human demonstrations. In *Proceedings of the Workshop on Inverse Optimal Control & Robotic Learning from Demonstration at Robotics: Science and Systems (RSS)*, Berlin, Germany, 2013.

[83] H. Kretzschmar, M. Kuderer, and W. Burgard. Learning to predict trajectories of cooperatively navigating agents. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Hong Kong, China, 2014.

[84] M. Kuderer and W. Burgard. An approach to socially compliant leader following for mobile robots. In *Social Robotics*, volume 8755, Sydney, Australia, 2014.

[85] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard. Feature-based prediction of trajectories for socially compliant navigation. In *Proceedings of Robotics: Science and Systems (RSS)*, Sydney, Australia, 2012.

[86] M. Kuderer, H. Kretzschmar, and W. Burgard. Teaching mobile robots to cooperatively navigate in populated environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 2013.

[87] M. Kuderer, C. Sprunk, H. Kretzschmar, and W. Burgard. Online generation of homotopically distinct navigation paths. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Hong Kong, China, 2014.

[88] M. Kuderer, S. Gulati, and W. Burgard. Learning driving styles for autonomous vehicles from demonstration. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Seattle, USA, 2015.

[89] H. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1):83–97, 1955.

[90] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2011.

[91] K. Kyriakopoulos and G. Saridis. Minimum jerk path generation. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 1988.

[92] B. Lau, C. Sprunk, and W. Burgard. Efficient grid-based spatial representations for robot navigation in dynamic environments. *Robotics and Autonomous Systems*, 61 (10):1116–1130, 2013.

[93] S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006. Section 14.7.

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

[95] A. Lerner, Y. Chrysanthou, and D. Lischinski. Crowds by example. *Computer Graphics Forum*, 26(3):655–664, 2007.

[96] S. Levine, Z. Popovic, and V. Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, 2011.

[97] M. Lin and J. Canny. A fast algorithm for incremental distance calculation. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, volume 2, 1991.

[98] S. Liu, D. Sun, and C. Zhu. Coordinated motion planning of multiple mobile robots in formation. In *World Congress on Intelligent Control and Automation*, 2010.

[99] M. Luber, L. Spinello, J. Silva, and K. O. Arras. Socially-aware robot navigation: A learning approach. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, 2012.

[100] C. Mandel and U. Frese. Comparison of wheelchair user interfaces for the paralysed: Head-joystick vs. verbal path selection from an offered route-set. In *European Conference on Mobile Robots*, 2007.

[101] J. Meyer, M. Kuderer, J. Müller, and W. Burgard. Online marker labeling for automatic skeleton tracking in optical motion capture. In *Workshop on Computational Techniques in Natural Motion Analysis and Reconstruction at the IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013.

[102] J. Meyer, M. Kuderer, J. Müller, and W. Burgard. Online marker labeling for fully automatic skeleton tracking in optical motion capture. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Hong Kong, China, 2014.

[103] N. Mitsunaga, C. Smith, T. Kanda, H. Ishiguro, and N. Hagita. Adapting robot behavior for human–robot interaction. *IEEE Transactions on Robotics (T-RO)*, 24 (4):911–916, 2008.

[104] K. Mombaur, A. Truong, and J.-P. Laumond. From human to humanoid locomotion – an inverse optimal control approach. *Autonomous Robots*, 28(3):369–383, 2010.

[105] Y. Morales, S. Satake, R. Huq, D. Glas, T. Kanda, and N. Hagita. How do people walk side-by-side? Using a computational model of human behavior for a social robot. In *ACM/IEEE International Conference on Human-Robot Interaction*, 2012.

[106] Motion Analysis Corp. *Cortex*, 2013. URL `http://www.motionanalysis.com/html/industrial/cortex.html`. Version 4.0.0.1387.

[107] M. Moussaïd, D. Helbing, S. Garnier, A. Johansson, M. Combes, and G. Theraulaz. Experimental study of the behavioural mechanisms underlying self-organization in human crowds. *Proceedings of the Royal Society B: Biological Science*, 276(1668): 2755–2762, 2009.

[108] J. Müller, C. Stachniss, K. Arras, and W. Burgard. Socially inspired motion planning for mobile robots in populated environments. In *Proceedings of the International Conference on Cognitive Systems (COGSYS)*, 2008.

[109] A. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2000.

[110] A. V. Nguyen, L. B. Nguyen, S. Su, and H. T. Nguyen. Shared control strategies for human-machine interface in an intelligent wheelchair. In *Proceedings of Annual International Conference of the Engineering in Medicine and Biology Society*, 2013.

[111] S. Obdrzalek, G. Kurillo, F. Ofli, R. Bajcsy, E. Seto, H. Jimison, and M. Pavel. Accuracy and robustness of kinect pose estimation in the context of coaching of elderly population. In *Proceedings of Annual International Conference of the Engineering in Medicine and Biology Society*, 2012.

[112] Y. Okada, K. Nagatani, K. Yoshida, T. Yoshida, and E. Koyanagi. Shared autonomy system for tracked vehicles to traverse rough terrain based on continuous three-dimensional terrain scanning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[113] G. Oriolo, A. D. Luca, and M. Vendittelli. WMR control via dynamic feedback linearization: Design, implementation, and experimental validation. *IEEE Transactions on Control Systems Technology*, 10(6):835–852, 2002.

[114] E. Pacchierotti, H. Christensen, and P. Jensfelt. Embodied social interaction for service robots in hallway environments. In *Field and Service Robotics*, volume 25 of *Springer Tracts in Advanced Robotics*, 2006.

[115] V. Parameswaran and R. Chellappa. View independent human body pose estimation from a single perspective image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2004.

[116] Q.-C. Pham, H. Hicheur, G. Arechavaleta, J.-P. Laumond, and A. Berthoz. The formation of trajectories during goal-oriented locomotion in humans. II. A maximum smoothness model. *European Journal of Neuroscience*, 26(8):2391–2403, 2007.

[117] J. Philips, J. d. R. Millán, G. Vanacker, E. Lew, F. Galán, P. W. Ferrez, H. V. Brussel, and M. Nuttin. Adaptive shared control of a brain-actuated simulated wheelchair. In *Proceedings of the IEEE International Conference on Rehabilitation Robotics*, 2007.

[118] B. Pitzer, M. Styer, C. Bersch, C. DuHadway, and J. Becker. Towards perceptual shared autonomy for robotic mobile manipulation. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2011.

[119] D. A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[120] N. Pradhan, T. Burg, S. Birchfield, and U. Hasirci. Indoor navigation for mobile robots using predictive fields. In *American Control Conference*, 2013.

[121] E. Prassler, D. Bank, B. Kluge, and M. Hagele. Key technologies in robot assistants: Motion coordination between a human and a mobile robot. *Transactions on Control, Automation and Systems Engineering*, 4(1):56–61, 2002.

[122] L. Qin, Y. Zha, Q. Yin, and Y. Peng. Formation control of robotic swarm using bounded artificial forces. *The Scientific World Journal*, 2013.

[123] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 1993.

[124] N. Ratliff, J. Bagnell, and M. Zinkevich. Maximum margin planning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2006.

[125] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2009.

[126] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE International Conference on Neural Networks*, 1993.

[127] M. Riedmiller, M. Montemerlo, and H. Dahlkamp. Learning to drive a real car in 20 minutes. In *Frontiers in the Convergence of Bioscience and Information Technologies (FBIT)*, 2007.

[128] M. Ringer and K. Lasenby. A procedure for automatically estimating model parameters in optical motion capture. In *Proceedings of the British Machine Vision Conference*, 2002.

[129] J. Rios-Martinez, A. Spalanzani, and C. Laugier. Understanding human interaction for probabilistic autonomous navigation using risk-RRT approach. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[130] L. Robert, S. Perdikis, L. Tonin, A. Biasiucci, M. Tavella, M. Creatura, A. Molina, A. Al-Khodairy, T. Carlson, and J. d. R. Millán. Transferring brain-computer interfaces beyond the laboratory: Successful application control for motor-disabled users. *Artificial Intelligence in Medicine*, 59(2):121–132, 2013.

[131] I. Sa and P. Corke. Vertical infrastructure inspection using a quadcopter and shared autonomy control. In *Field and Service Robotics*, 2014.

[132] W. Schwarting and P. Pascheka. Recursive conflict resolution for cooperative motion planning in dynamic highway traffic. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2014.

[133] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.

[134] D. Silver, J. A. Bagnell, and A. Stentz. Learning autonomous driving styles and maneuvers from expert demonstration. In *Experimental Robotics*, 2013.

[135] R. Soyama, S. Ishii, and A. Fukase. The development of meal-assistance robot "MySpoon". In *Proceedings of the IEEE International Conference on Rehabilitation Robotics*, 2003.

[136] C. Sprunk, B. Lau, P. Pfaff, and W. Burgard. Online generation of kinodynamic trajectories for non-circular omnidirectional robots. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2011.

[137] P. Stein, A. Spalanzani, V. Santos, and C. Laugier. On leader following and classification. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[138] H. G. Tanner and A. Kumar. Formation stabilization of multiple agents using decentralized navigation functions. In *Proceedings of Robotics: Science and Systems (RSS)*, 2005.

[139] O. Taubman-Ben-Ari, M. Mikulincer, and O. Gillath. The multidimensional driving style inventory – scale construct and validation. *Accident Analysis & Prevention*, 36(3):323–332, 2004.

[140] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1):99–141, 2000.

[141] P. Trautman and A. Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[142] S. Vacek, C. Schimmel, and R. Dillmann. Road-marking analysis for autonomous vehicle guidance. In *EMCR*, 2007.

[143] J. van den Berg, S. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In *Proceedings of the International Symposium of Robotics Research (ISRR)*, 2009.

[144] D. A. Vasquez Govea, T. Fraichard, and C. Laugier. Growing hidden Markov models: An incremental tool for learning and predicting human and vehicle motion. *International Journal of Robotics Research*, 28(11-12):1486–1506, 2009.

[145] T. Veit, J.-P. Tarel, P. Nicolle, and P. Charbonnier. Evaluation of road marking feature extraction. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2008.

[146] P. Vela, A. Vela, and G. Ogunmakin. Topologically based decision support tools for aircraft routing. In *IEEE/AIAA Digital Avionics Systems Conference (DASC)*, 2010.

[147] P. Vernaza and D. Bagnell. Efficient high dimensional maximum entropy modeling via symmetric partition functions. In *Advances in Neural Information Processing Systems*, 2012.

[148] P. Vernaza, V. Narayanan, and M. Likhachev. Efficiently finding optimal winding-constrained loops in the plane. In *Proceedings of Robotics: Science and Systems (RSS)*, 2012.

[149] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Transactions on Automatic Control*, 54(10):2318–2327, 2009.

[150] C. Wang, A. Matveev, A. Savkin, T. Nguyen, and H. Nguyen. A collision avoidance strategy for safe autonomous navigation of an intelligent electric-powered wheelchair in dynamic uncertain environments with moving obstacles. In *European Control Conference (ECC)*, 2013.

[151] Y. Wang, D. Shen, and E. K. Teoh. Lane detection using spline model. *Pattern Recognition Letters*, 21(8):677–689, 2000.

[152] Y. Xiang, S. Rahmatalla, J. S. Arora, and K. Abdel-Malek. Enhanced optimisation-based inverse kinematics methodology considering joint discomfort. *International Journal of Human Factors Modelling and Simulation*, 2(1):111–126, 2011.

[153] K. Yamane and J. Hodgins. Simultaneous tracking and balancing of humanoid robots for imitating human motion capture data. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.

[154] M. Yoda and Y. Shiota. Analysis of human avoidance motion for application to robot. In *Proceedings of the 5th IEEE International Workshop on Robot and Human Communication*, 1996.

[155] M. Yoda and Y. Shiota. The mobile robot which passes a man. In *Proceedings of the 6th IEEE International Workshop on Robot and Human Communication*, 1997.

[156] Q. Yu, Q. Li, and Z. Deng. Online motion capture marker labeling for multiple interacting articulated targets. *Computer Graphics Forum*, 26(3):477–483, 2007.

[157] D. Zambrano, D. Bernardin, D. Bennequin, C. Laschi, and A. Berthoz. A comparison of human trajectory planning models for implementation on humanoid robot. In *Proceedings of the IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, 2012.

[158] K. Zhao, M. Meuter, C. Nunn, D. Muller, S. Muller-Schneiders, and J. Pauli. A novel multi-lane detection and tracking system. In *IEEE Intelligent Vehicles Symposium (IV)*, 2012.

[159] B. Ziebart, A. Maas, J. Bagnell, and A. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2008.

[160] B. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. Bagnell, M. Hebert, A. Dey, and S. Srinivasa. Planning-based prediction for pedestrians. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.

[161] B. Ziebart, A. Dey, and J. Bagnell. Probabilistic pointing target prediction via inverse optimal control. In *Proceedings of the ACM International conference on Intelligent User Interfaces*, 2012.

[162] J. Ziegler, H. Kretzschmar, C. Stachniss, G. Grisetti, and W. Burgard. Accurate human motion capture in large areas by combining IMU- and laser-based people tracking. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[163] V. Zordan and N. Van Der Horst. Mapping optical motion capture data to skeletal motion using a physical model. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003.