Techniques for Robot Navigation in Dynamic Real-World Environments

Boris Lau

UNI FREIBURG

Dissertation zur Erlangung des Doktorgrades der Technischen Fakultät der Albert-Ludwigs-Universität Freiburg im Breisgau

Betreuer: Prof. Dr. Wolfram Burgard

Dezember 2013

Techniques for Robot Navigation in Dynamic Real-World Environments

Boris Lau

Dissertation zur Erlangung des Doktorgrades der Technischen Fakultät
der Albert-Ludwigs-Universität Freiburg im Breisgau

Dekan:	Prof. Dr. Yiannos Manoli
Erstgutachter:	Prof. Dr. Wolfram Burgard
Zweitgutachter:	Prof. Dr. Maren Bennewitz
Tag der Disputation:	09.12.2013

Abstract

Many real-world applications in mobile robotics require the operation of robots in dynamic environments that contain unmapped obstacles, such as moving people. In three independent parts, this thesis studies robotic problems that have often been addressed by approaches not geared towards such environments. For example, they might assume simplified or static environments, or have computational requirements that prevent an online application.

The first part of this thesis deals with a number of grid-based spatial representations derived from occupancy grid maps, namely distance maps, Voronoi diagrams, and configuration space maps. While previous approaches to compute and update these representations mostly assume static environments, this thesis presents algorithms for updating them incrementally. Since these algorithms only process cells affected by changes in the environment, they can be applied online even on large maps. The proposed update method for distance maps and Voronoi diagrams is based on a dynamic variant of the brushfire algorithm, which propagates changes in wavefronts starting at newly occupied or freed grid cells.
The computation of collision maps in the configuration space of mobile robots usually requires the convolution of a map with the footprint of the robot. The proposed approach updates the summands of the convolution that correspond to newly occupied or emptied cells in the map. By combining the dynamic brushfire method with the incremental configuration space update, this thesis further proposes distance maps and Voronoi diagrams in the configuration space of mobile robots, and provides suitable update algorithms. Experiments on real-world data sets demonstrate the effectiveness of the proposed methods in 2D and 3D environments.

In the second part, this thesis addresses the problem of kinodynamic motion planning. Smooth and precise trajectory execution on a real robot requires the generation of curvature continuous trajectories that obey the dynamic constraints of the hardware platform. This thesis discusses suitable path representations and proposes a novel path model that represents curvature continuous paths with a small number of parameters. The model is based on quintic Bézier splines, which provide additional degrees of freedom in comparison to the more common cubic splines. This facilitates curvature continuous junctures of individual path segments as found in B-Splines, but without losing the property that the curve passes through its waypoints. By using heuristics to set the orientation of tangents and part of the second derivatives of the spline, the number of free parameters is kept small. Based on this model, this thesis further presents

a method to compute a feasible velocity profile for a given initial path, and to optimize the resulting trajectory with respect to the required time of travel. The system can alter the planned trajectory while moving, and is therefore able to avoid unmapped obstacles, such as moving people. Several experiments tested and evaluated the proposed motion planning system on real robots in different complex and populated environments. The results show that the generated trajectories can be executed with high precision, and that the system can evade unexpected obstacles with smooth trajectory changes.

The third part of this thesis deals with tracking groups of people with the onboard sensors of a mobile robot. In crowded environments, people often move in dense groups. Due to occlusions and ambiguous data association, tracking individuals can become infeasible in such cases. Tracking a joint group state rather than individuals reduces the complexity of the data association problem. This requires suitable detection methods and a mechanism to model and track group splits and merges. This thesis introduces the concept of a group track that estimates not only the position and velocity of the group centroid, but also the number of people in the group and the shape of the group as perceived by the sensor. Using the minimum average Hausdorff distance, the shape information is also accounted for when calculating association probabilities. Based on the group track representation, this thesis proposes an extended Multi-hypothesis tracking framework that considers group formation models and hypothesizes over continuation, splits, and merges of group tracks. The experiments on a moving robot with up to 20 people demonstrate that the system is able to robustly track groups of people in complex group formation processes and can estimate the number of people in groups with high accuracy.

Zusammenfassung

Viele praktische Anwendungen in der mobilen Robotik erfordern den Einsatz von Robotern in dynamischen Umgebungen, die nicht kartierte Hindernisse wie zum Beispiel Personen enthalten können. In drei unabhängigen Teilen betrachtet diese Arbeit verschiedene Probleme der mobilen Robotik, die oft mit Ansätzen angegangen wurden, die nur bedingt für den Einsatz in derartigen Umgebungen geeignet sind. Manche Lösungen nehmen vereinfachte oder statische Umgebungen an, oder sind durch ihren Rechenaufwand nicht für Online-Anwendungen geeignet.

Der erste Teil dieser Arbeit behandelt verschiedene räumliche Repräsentationen, die auf Gitterkarten basieren, nämlich Distanzkarten, Voronoi Diagramme und Hinderniskarten im Konfigurationsraum mobiler Roboter. Im Unterschied zu existierenden Ansätzen, die größtenteils statische Umgebungen annehmen, stellt diese Arbeit Algorithmen vor, welche die genannten Repräsentationen auf inkrementelle Weise aktualisieren. Da die Algorithmen nur Gitterzellen bearbeiten, die von Änderungen in der Umgebung betroffen sind, können sie auch auf großen Karten online arbeiten. Die Aktualisierung von Distanzkarten und Voronoidiagrammen wird mit einer neuen dynamischen Variante des Brushfire-Algorithmus realisiert. Die vorgestellte Methode propagiert Aktualisierungen wellenförmig, ausgehend von neu eingefügten oder entfernten Hindernissen.
Die Berechnung einer Hinderniskarte im Konfigurationsraum eines mobilen Roboters erfordert in herkömmlichen Verfahren die mathematische Faltung der Umgebungskarte mit einer Repräsentation der Grundfläche des Roboters. Das vorgestellte Verfahren aktualisiert gezielt die Summanden der Faltung, die zu neu besetzten oder freigewordenen Zellen der Umgebungskarte gehören. Durch die Kombination der dynamischen Brushfire-Methode mit der Aktualisierung von Hinderniskarten im Konfigurationsraum mobiler Roboter führt diese Arbeit außerdem Distanzkarten und Voronoi Diagramme im Konfigurationsraum ein und liefert entsprechende inkrementelle Aktualisierungsalgorithmen. Experimente mit mehreren Datensätzen unterschiedlicher Sensoren zeigen die Effektivität der vorgestellten Methoden in 2D und 3D Umgebungen.

Im zweiten Teil behandelt diese Arbeit das Problem der kinodynamischen Pfadplanung. Um geplante Trajektorien mit einem Roboter sanft und präzise abfahren zu können, müssen sie krümmungskontinuierlich sein und die dynamischen Beschränkungen der Hardware berücksichtigen. Diese Arbeit diskutiert geeignete Pfadrepräsentationen und stellt ein neues Pfadmodell vor, das krümmungskontinuierliche Pfade mit einer geringen Anzahl an Parametern

repräsentiert. Die vorgestellte Pfadrepräsentation basiert auf quintischen Bézier Splines, die im Gegensatz zu den häufig verwendeten kubischen Splines zusätzliche Freiheitsgrade zur Verfügung stellen. Dadurch lässt sich die Krümmung der Kurve wie bei B-Splines auch an Segmentgrenzen kontinuierlich fortführen, ohne jedoch die Eigenschaft zu verlieren, dass die Kurve ihre Wegpunkte durchläuft. Durch Heuristiken die z.B. das Verhalten der Tangenten und der zweiten Ableitung des Splines kontrollieren, wird die Anzahl der freien Parameter klein gehalten. Weiterhin wird eine Methode vorgestellt um für einen gegebenen Pfad ein Geschwindigkeitsprofil zu bestimmen, und die dadurch definierte Trajektorie mit Hinblick auf die erforderliche Fahrzeit zu optimieren. Das System kann die geplante Trajektorie während der Fahrt ändern, und ist dadurch in der Lage unerwarteten Hindernissen auszuweichen. Mehrere Experimente testen und evaluieren das vorgestellte Pfadplanungssystem in verschiedenen komplexen und belebten Umgebungen. Die Ergebnisse zeigen, dass die erzeugten Trajektorien mit hoher Genauigkeit abgefahren werden können, und dass das System unkartierten Hindernissen durch sanfte Trajektorienänderungen ausweichen kann.

Der dritte Teil dieser Arbeit behandelt die Spurverfolgung (das Tracken) von Menschengruppen mit den Bordsensoren eines mobilen Roboters. In belebten Umgebungen bewegen sich Menschen oft in Gruppen. In solchen Fällen kann die Datenassoziation durch Verdeckungen und Mehrdeutigkeiten zu einem sehr aufwändigen oder sogar unlösbaren Problem werden. Das Tracken von kompletten Gruppen im Gegensatz zu Individuen reduziert die Komplexität der Datenassoziation. Dies erfordert jedoch entsprechende Detektionsmechanismen und die Möglichkeit das Aufteilen oder Vereinigen von Gruppentracks zu modellieren. Diese Arbeit führt das Konzept des Gruppentracks ein, der die Position und Geschwindigkeit des Mittelpunkts einer Gruppe schätzt, sowie die Anzahl der Personen in der Gruppe und die durch Sensoren wahrgenommene Form der Gruppe. Durch Benutzung der Minimum-Average-Hausdorff-Distanz wird die Forminformation auch in die Berechnung von Assoziationswahrscheinlichkeiten einbezogen. Basierend auf den Gruppentracks stellt diese Arbeit ein erweitertes Multi-Hypothesen Trackingsystem vor, das Modelle der Gruppenformation verwendet, und dadurch über die Fortführung, Aufteilung oder Verbindung von Gruppentracks hypothetisieren kann. Die Experimente mit bis zu 20 Personen in der Umgebung eines fahrenden Roboters zeigen, dass das System Personengruppen in komplexen Formierungsprozessen robust tracken und die Gruppengröße akkurat schätzen kann.

Acknowledgements

During my time in the Autonomous Intelligent Systems lab at the University of Freiburg, I was accompanied and supported by many wonderful people, to whom I want to express my gratitude. Especially, I want to thank my advisor Wolfram Burgard. He always provided great ideas, new points of view, and exciting challenges. I enjoyed the way he cares for his people, helping them to set and reach high expectations, while giving them the freedom to pursue their own ideas. He was the best teacher in diplomatic writing I ever had – thank you!

Christoph Sprunk was with me through all highs and lows – as a student, colleague, co-author and friend. His structured way of thinking, strong commitment to everything he does, and his pedantic perfectionism are properties that I really appreciate. And it was fun! Especially before deadlines, my office turned into my second home, shared with my long-term office mates Axel Rottmann and Kai Wurm, followed later by Markus Kuderer and Christoph. They were always up for inspiring discussions and helped when things did not work as planned. Thanks for the good times!

The post-docs were always there to answer all sorts of scientific, technical, or procedural questions. Kai Arras, Giorgio Grisetti, Cyrill Stachniss, Luciano Spinello, and Gian Diego Tipaldi, thank you for your time and support. I also associate pretty much everyone in the lab with something they really helped me out with – software, math, tools, ideas, whatever. Thank you all! During the work on the INDIGO and the RADHAR project, I also shared quite some time with researchers from other labs. Thank you for working with me all over Europe! Kristine Haberer, Bettina Schug, Susanne Bourjaillat, and Michael Keser always assisted with administrative or infrastructural issues. Thank you very much!

I really want to thank the developers of LaTeX and BibTeX for making the typesetting of scientific documents such a pleasure. Thanks also go to Christian Plagemann for providing his nice TOC formatting. And a big thank you to all proofreaders, critics, and commentators!

Well, there were also things in life besides research, and they helped keeping me sane and happy. Thanks to all my fellow musicians, thanks to my friends, and thanks to my family, Ursel, Richard and Hannes. During my time in Freiburg, Inga became my wife and our sweet Naima was born. It is hard to express my gratefulness, you make my day, again and again!

Last but not least: this work has partly been supported by the European Commission under grant agreement numbers FP6-IST-045388 (INDIGO) and FP7-248873-RADHAR, and by the German Research Foundation (DFG) through the Gottfried Wilhelm Leibniz Program. Their support is gratefully acknowledged.

Contents

1 Introduction

Stationary robots have become a common tool in industrial settings, especially when tasks are highly repetitive or potentially dangerous for humans to execute. The advent of mobile robots opened up an even wider field of potential robotic applications such as transportation, building maintenance, household tasks, or entertainment.

For several decades now, mobile robotics is an active field of research and development. In this context, many research problems are related to perception and autonomous navigation under uncertainty: mobile robots often have to estimate their own position and orientation with on-board sensors, handle limited and noisy sensor data, and deal with inaccurate or incomplete representations of the environment. In many real-world applications, an additional complexity originates from dynamic obstacles like moving people. They are often unexpected and hardly predictable for the robot, and thus add to the unknown part of the environment.

Today, a variety of mobile robots is available for applications in private, commercial, and public domains. For some tasks like lawn mowing, floor cleaning, or surveillance of empty buildings, the navigation system of a robot can assume its operational area to be static, i.e., free of moving objects. Other systems like transportation platforms, however, are often deployed in environments such as libraries, hospitals, warehouses, or factory floors. Here, a robot has to deal with moving obstacles like other robots or people in its vicinity. Robotic applications that involve direct interaction with human users naturally require a special focus on the dynamic parts of the environment. For example, besides the interaction with actual users, tour guide or entertainment robots in exhibitions or trade shows can be challenged by crowded environments or people that purposely interfere with their plans of action.

Several fundamental approaches and solutions that have been proposed in the field of mobile robotics are not ideal for application in such scenarios. For example, they might assume all obstacles to be known and static, rely on simplified environments, or have computational requirements that prevent their online application on a moving robot. This thesis focuses on a set of well-studied problems in mobile robotics that have mainly been addressed by approaches that are subject to such restrictions. By proposing novel techniques that can be applied online and in real environments, it contributes to the topics described below.

Grid-based spatial representations derived from occupancy grid maps are important building blocks for many different robotic applications. Distance maps, Voronoi diagrams, and configuration space maps are for example used to speed up techniques for path planning, collision avoidance, and localization. Among moving obstacles, these representations need to be updated frequently to reflect changes in the environment. Most of the existing approaches compute such representations for complete maps and do not support incremental changes, which often prevents updating them at frame rates required for a moving robot.

Kinodynamic motion planning refers to the generation of trajectories that specify the position and velocity of a robot as a function of time. To allow smooth and precise execution by a real robot, such trajectories should be curvature continuous and obey the kinematic and dynamic constraints of the hardware platform. For application in dynamic environments it is also crucial to be able to modify a trajectory while moving, for example, to recover after temporary localization errors or to avoid unexpected obstacles. Existing approaches often disregard these aspects and are therefore not directly applicable under real-world conditions.

People tracking is an enabling technology for most applications that involve human-robot interaction. It comprises estimating the position and velocity of people moving around the robot based on sensor data, and storing these estimates in individual tracks. A major challenge in people tracking is to determine the correct associations between sensor readings and existing tracks, especially if people are walking close to another or occlude each other. Since the number of possible combinatorial associations grows quickly with the number of tracked people, the computational requirements per frame can prevent tracking in real-time if many people are present at the same time, e.g., when moving in groups.

1.1 Scientific Contributions

With this thesis, we contribute novel approaches to the aforementioned problems. To perform efficient updates of Euclidean distance maps and Voronoi diagrams, we propose a dynamic variant of the brushfire algorithm that visits only the map cells that have been affected by changes in the environment. Additionally, we propose an incremental update algorithm for configuration space maps that can be applied to two- and three-dimensional robot and obstacle models. By combining the dynamic brushfire algorithm with the incremental configuration space maps, we obtain distance maps and Voronoi diagrams in the configuration space of mobile robots, and we demonstrate their use for efficient path planning.

An essential aspect of trajectory-based motion planning systems is the representation used to model trajectories. We discuss a variety of parametric curve types and their applicability to motion planning, before proposing a novel path model based on curvature-continuous quintic Bézier splines. We further present a method to efficiently compute velocity profiles for given paths that obey constraints of the hardware platform. From a given set of waypoints, we construct feasible initial trajectories, which are subsequently optimized to reduce the time required to reach the goal. The proposed motion planning system operates in an anytime fashion and allows for replanning trajectories while the robot moves.

Our contribution to people tracking in populated environments is an approach to track groups of people rather than individuals. We introduce the concept of group tracks represented by a joint group state, and present a method to estimate the number of people in each group. To track the group formation process over time, we propose models that account for the continuation, splitting, or merging of existing group tracks. Our multi-model multi-hypothesis tracker integrates group tracks and group formation models into a probabilistic tracking framework. Since the computational complexity scales with the number of groups rather than the number of people, our system is more efficient than previous approaches. Additionally, it provides information about the social relation of the tracked people.

For all presented approaches, we conduct experiments using real robots in real-world environments, and evaluate the computational requirements to ensure their online applicability.

1.2 Publications

Parts of the work presented in this thesis have been published in the form of peer-reviewed conference proceedings, journal articles, or book chapters. The following list presents them in chronological order, grouped by main topic.

Efficient Grid-Based Spatial Representations

B. Lau, C. Sprunk, and W. Burgard. Improved updating of Euclidean distance maps and Voronoi diagrams. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, 2010.

B. Lau, C. Sprunk, and W. Burgard. Incremental updates of configuration space representations for non-circular mobile robots with 2D, 2.5D, or 3D obstacle models. In *Proc. of the European Conference on Mobile Robots (ECMR)*, Örebro, Sweden, 2011.

B. Lau, C. Sprunk, and W. Burgard. Efficient grid-based spatial representations for Robot Navigation in Dynamic Environments. In *Robotics and Autonomous Systems*, 2013, in press.

Kinodynamic Motion Planning

B. Lau, C. Sprunk, and W. Burgard. Kinodynamic motion planning for mobile robots using splines. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, St. Louis, USA, 2009.

C. Sprunk, B. Lau, P. Pfaff, and W. Burgard. Online generation of kinodynamic trajectories for non-circular omnidirectional robots. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, Shanghai, China, 2011.

C. Sprunk, B. Lau, and W. Burgard. Improved non-linear spline fitting for teaching trajectories to mobile robots. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, St. Paul, MN, USA, 2012.

Laser-Based Tracking of People in Groups

B. Lau, K. O. Arras, and W. Burgard. Tracking groups of people with a multi-model hypothesis tracker. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, Kobe, Japan, 2009.

B. Lau, K. O. Arras, and W. Burgard. Multi-model hypothesis group tracking and group size estimation. *International Journal of Social Robotics, Springer Netherlands*, 2(1):19–30, 2010.

K. O. Arras, B. Lau, S. Grzonka, M. Luber, O. Martinez Mozos, D. Meyer-Delius and W. Burgard. Range-based people detection and tracking for socially enabled service robots. In *Towards Service Robots for Everyday in Environments, Springer Tracts in Advanced Robotics (STAR)*, 76:235–280, 2012.

Further appearances of the work contained in this thesis comprise workshop contributions and public demonstrations.

1.3 Software Releases

The following parts of the source code developed in the context of this thesis have been released to the public or to commercial users:

- Our incremental implementations of distance maps, Voronoi diagrams, and configuration space maps have been released as self-contained open-source library under `http://www.informatik.uni-freiburg.de/~lau/dynamicvoronoi`
- The two-dimensional incremental distance maps and Voronoi diagrams are also available as a package for the open source Robot Operating System (ROS) under `http://www.ros.org/wiki/dynamicvoronoi`

- The three-dimensional version of our incremental distance maps has been integrated into the OctoMap software package developed by Wurm *et al.* [2010], available as open source under `http://octomap.github.io`
- The software for the proposed motion planning system has been integrated into a library used to control mobile platforms built by KUKA Laboratories GmbH.

1.4 Collaborations

Parts of the work presented in this thesis have been done in collaboration with other researchers as stated in this section.

The spatial representations in Part I contain joint work with Christoph Sprunk. Besides giving fruitful advice and ideas, he implemented previous approaches and realized a benchmarking suite for evaluation and comparison. He is also responsible for the realization and release of the open source incremental 3D distance maps.

The motion planning system in Part II also contains joint work with Christoph Sprunk. He realized the heuristics of our path model, the parameter optimization, and the replanning routines as part of his student project under supervision of the author. Parts that he conceived independently are summarized in this thesis and properly cited.

The group tracking work in Part III has been done in collaboration with Kai O. Arras. Matthias Luber and Slawomir Grzonka provided their experience and source code for tracking individual people with an MHT framework.

As supervisor, Wolfram Burgard contributed thoughts and ideas to all parts of this thesis.

1.5 Notation

This table summarizes the standard notation of mathematical and algorithmic symbols used throughout this work.

Symbol	Meaning
a, b, x, ...	scalar values
A, H, Q, ...	matrices
$\mathbf{u}$, $\mathbf{x}$, $\mathbf{z}$, ...	column vectors
$\mathcal{C}$, $\mathcal{L}$, $\mathcal{P}$	sets
$(\dots)$	row vector or matrix composed of the given values
$\langle \dots \rangle$	tuple composed of the given entities
$\{\dots\}$	set composed of the given elements
$\lvert a \rvert$	absolute value of a scalar
$\lVert \mathbf{x} \rVert$	Euclidean norm (length) of a vector
$\lvert A \rvert$	determinant of a matrix
$\lvert \mathcal{P} \rvert$	number of elements in a set
$\emptyset$	empty set or empty queue
$\mathbf{x}^T$, A^T	transpose of a vector or matrix
$\mathcal{N}(\mu, \Sigma)$	Gaussian distribution with mean μ and covariance Σ
$M(x, y)$, $M(s)$	grid map, cell value for coordinates x, y or tuple s
$C(x, y, \theta)$	3D c-space map, cell value for given robot pose x, y, θ
$C_\theta(x, y)$	2D layer of $C(x, y, \theta)$ for orientation θ
$\mathbf{Q}(u)$, $\mathbf{Q}_x(u)$, $\mathbf{Q}_y(u)$	Parametric curve (column vector) and its components
$\mathbf{Q}'(u)$, $\mathbf{Q}''(u)$, $\mathbf{Q}^{(k)}(u)$	First, second, and k-th derivative of $\mathbf{Q}(u)$
$\mathbf{P}_i$, $\mathbf{W}_i$, $\mathbf{T}_i$, $\mathbf{A}_i$, $\mathbf{J}$	control points (column vectors)
$\mathbf{x}(k\|k)$	track state vector in time step k for evidence of time step k
$\mathbf{x}(k+1\|k)$	state prediction in time step $k+1$ for evidence of time step k
$\{0\}$, $\{0-0\}$, $\{0-1\}$	label of a group and its two subgroups after a split
s, $D(s)$, $obst(s)$	variables in pseudo-code and text
insert(.), pop(.)	function calls in pseudo-code
$\mathbf{insert}(.)$, $\mathbf{pop}(.)$	function names in text

1.6 Outline

This thesis is divided into the main parts I–III that deal with incremental updates of spatial representations, kinodynamic motion planning, and laser-based tracking of people in groups, respectively. Each part is organized as a self-contained document and has its individual introduction, discussion of related work, technical chapters, experiments, and a conclusion. The individual parts are structured as described in the following.

Part I introduces the examined spatial representations and discusses the problem of updating them incrementally in Chapter 2. After reviewing the related work in Chapter 3, we present our update algorithms for distance maps and Voronoi diagrams in Chapter 4 and Chapter 5, respectively. Chapter 6 proposes incrementally updatable collision maps in the configuration space of mobile robots, combines them with our methods for dynamic distance maps and Voronoi diagrams, and presents a path planning approach on the resulting configuration space Voronoi diagrams. Chapter 7 shows experiments, before Chapter 8 concludes the first part of the thesis.

Part II discusses the problem of kinodynamic motion planning in Chapter 9 and reviews related work in Chapter 10. After analyzing different path representations in Chapter 11, we propose a novel path model in Chapter 12 and corresponding velocity profiles in Chapter 13. In Chapter 14 we describe an approach to generate, optimize, and execute trajectories using our path model, and present experiments in Chapter 15. After discussing extensions and applications of our approach in Chapter 16, we conclude the second part in Chapter 17.

Part III proposes the idea of tracking groups of people rather than individuals in Chapter 18 and discusses related work. We propose the concept of group tracks in Chapter 19 and discuss their definition, detection, and representation. In Chapter 20, we introduce group formation models and discuss our multi-model multi-hypothesis group tracker, which is evaluated in Chapter 21.

Part IV closes the thesis with a final discussion and outlook. Lists of figures, tables, and algorithms as well as the bibliography for all parts are located at the end of the document.

Part I

Efficient Grid-Based Spatial Representations

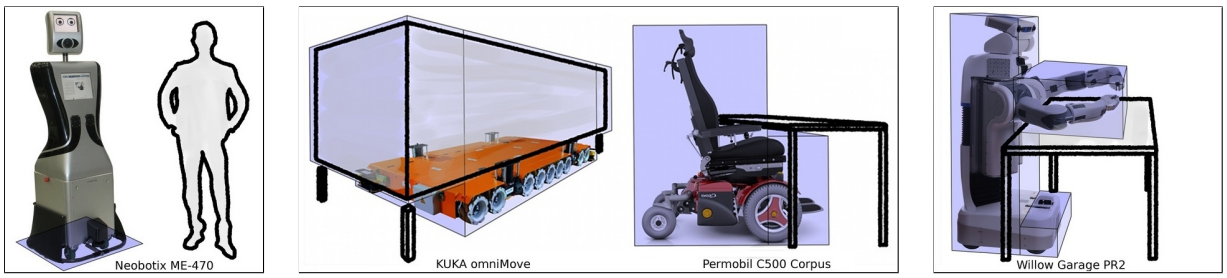
2 Introduction

Many approaches in robot navigation rely on occupancy grid maps to encode the obstacles of the area surrounding a robot. These maps can be learned from sensor data, they are well suited to be used in path planning, collision avoidance, or localization, and they can easily be updated upon sensory input. In the past, several spatial representations have been developed that can be derived from occupancy grid maps, e.g., distance maps, Voronoi diagrams, and configuration space maps. These representations are appealing building blocks for robot navigation systems, since they can be used to speed-up algorithms that solve the aforementioned problems. In this thesis, we propose incremental update algorithms to facilitate the online use of these representations in dynamic environments.

The Generalized Voronoi diagram (GVD) is defined as the set of points in free space to which the two closest obstacles have the same distance [Choset and Burdick, 2000]. It is a discrete form of the Voronoi graph, which has been widely used in various fields [Aurenhammer, 1991]. In the context of robotics, Voronoi graphs are a popular cell decomposition method for solving navigation tasks. Their application as roadmaps is an appealing technique for path planning, since they are "sparse" in the sense that different paths on the graph correspond to topologically different routes with respect to obstacles. This significantly reduces the search problem and can be used for example to generate the n-best paths for offering route alternatives to a user [Mandel and Frese, 2007]. Also, moving along the edges of a Voronoi graph ensures the greatest possible clearance when passing between obstacles. When Voronoi graphs are discretized and stored as a map, they can lose their sparseness property due to erroneous interconnections between neighboring Voronoi lines. Our method to compute GVDs overcomes this problem with additional pruning and conditions that ensure the sparseness of the generated GVDs.

The cells in a distance map (DM) encode the distance to the closest cell that is occupied according to the corresponding occupancy map. Since a cell lookup only requires constant time, DMs provide efficient means for collision checks, to compute traversal costs for path planning, and for robot localization with likelihood fields [Thrun, 2001]. Since the computation of this transform is carried out without considering the shape of the robot, the direct application of plain DMs is restricted to circular approximations of the robot's footprint.

For non-circular robots in passages narrower than their circumcircle, however, circularity is too crude an assumption, and collisions have to be checked for in the three-dimensional configura-



(a) 2D robot model (b) 2.5D robot model (c) 3D robot model

Figure 2.1: For some applications, representing obstacles and robots by their 2D footprints can be sufficient (a). For overhanging parts of robots, their load, or obstacles, 2.5D representations are needed (b), whereas interaction tasks can also require actual 3D obstacle and robot models (c). Robot shape approximations as used in our experiments are depicted in blue.

tion space (c-space) of robot poses. Also, even for robots moving on a plane as considered in this thesis, 3D obstacles and collisions can be important: applications such as robotic transporters, wheelchairs, or mobile manipulators can require the robot to partially move underneath or above obstacles as shown in Figure 2.1. In these cases, collision checks easily become a dominant part of the computational effort in path planning. However, by convolving a map with the discretized shape of the robot, one can precompute a collision map that marks all colliding poses. With such a map, a collision check requires just a single lookup, even for 3D obstacle representations.

In changing environments, precomputed GVDs, DMs, and c-space maps have to be updated regularly to always reflect the current state of the corresponding occupancy map. These changes can be caused by moving people or vehicles, newly explored areas during mapping, or when correcting a map after closing a loop in SLAM. In this thesis, we present efficient methods to compute and update these representations. Since our algorithms perform all updates in an incremental way, i.e., recomputing only parts affected by changes, they can be applied online even on large maps or with more than two dimensions. In comparison to previous approaches, our methods require less computational effort, are easy to implement, and work in both indoor and outdoor environments.

Additionally, we combine DMs and GVDs with c-space collision maps, and propose distance maps and Voronoi diagrams in the configuration space of non-circular mobile robots. Using our algorithms, these representations can be updated in an incremental way as well, and thus be used to speed-up path planning or collision avoidance in dynamic environments. This work extends our previous publications on these topics [Lau *et al.*, 2010; 2011] and includes additional experiments for 3D distance maps and for incremental updates of distance maps and Voronoi

diagrams in the context of simultaneous localization and mapping (SLAM).

After discussing related work in Chapter 3, we present our dynamic DM and GVD algorithms in Chapters 4 and 5. In Chapter 6 we propose dynamic c-space collision maps, which we combine with our dynamic DM and GVD algorithms to obtain c-space DM and c-space GVD representations. Our experiments are presented in Section 7.

3 Related Work

In the past, many different approaches have been proposed to compute DMs, GVDs, and c-space collision maps. With the goal of applying them online in dynamic environments, a lot of effort has been spent on developing more efficient algorithms. However, unlike ours, most of these approaches do not exploit the potential of incremental updates. The remainder of this chapter presents related work and discusses the contribution of our methods.

3.1 Distance Maps

Many different approaches have been proposed to compute static two-dimensional DMs, e.g., linear image traversal [Borgefors, 1986], dimensional decomposition [Fabbri *et al.*, 2008], and distance propagation with the brushfire algorithm [Verwer *et al.*, 1989], which is discussed in Section 4.1. For a comparative survey, please refer to [Fabbri *et al.*, 2008].

Whenever a cell in a grid map is newly occupied or vacated, the corresponding DM has to be updated to reflect this change. A trivial method is to recompute distances for patches within $\hat{d}$ around all changed cells, where $\hat{d}$ is an upper bound on the minimum obstacle distance in the environment. However, this method usually updates substantially more cells than necessary, e.g., if $\hat{d}$ is high due to large open spaces or if the changed cells cover a wide area.

Kalra *et al.* proposed a dynamic brushfire algorithm that incrementally updates DMs and GVDs by propagating wavefronts starting at newly occupied or vacated cells [2009]. While their method is based on the incremental path planning algorithm D* by Stentz [2004], the algorithm proposed here is directly derived from the brushfire algorithm and requires substantially less computational time for the same task due to a reduced number of cell visits.

The wavefronts of Kalra *et al.* accumulate 8-connected grid steps to approximate obstacle distances [2009]. This overestimates the true Euclidean distances by up to 8.0% [Danielsson, 1980], which for a robot implies either a collision risk or overly conservative movements. Scherer *et al.* adopted and corrected Kalra's algorithm for their DM update method [2009]. They propagate obstacle locations rather than grid step counts to determine Euclidean distances, which reduces the absolute overestimation error below an upper bound of 0.09 pixel units [Danielsson, 1980].

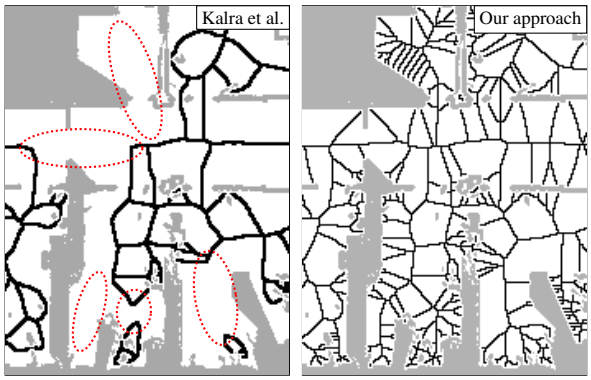


Figure 3.1: GVD of an indoor map, computed by the approach of Kalra *et al.* and our method. The ellipses mark missing Voronoi lines, see Section 3.2. Our approach generates thin Voronoi lines, such that different paths on the GVD correspond to topologically different routes.

According to Cuisenaire and Macq, the shortest distance at which this propagation error can occur is 13 pixels [1999], which yields a maximum relative error of 0.69%. Our approach follows the same principle, but requires substantially shorter computation times for the same task due to a reduced number of cell visits. By propagating obstacle references, our representation can provide the location of the closest obstacle rather than just the distance to it, which can be appealing for collision avoidance methods. In a recent publication, Scherer *et al.* build on our original method for DM updates and combine it with their approach to map scrolling [2012].

This work extends our DMs presented in [Lau *et al.*, 2010] to 3D by adding the possibility to limit the propagated distances to maintain online feasibility in large open spaces and outdoors as proposed by Scherer *et al.* [2009]. Additionally, we describe how to further reduce the number of visited neighbor cells, which increases the efficiency for 3D DMs, and we present additional experiments.

3.2 Voronoi Diagrams

Traditional Voronoi algorithms compute parametric lines or curves that separate singular obstacle points or line segments represented in continuous space. There are approaches to update such Voronoi graphs, e.g., for newly discovered obstacles during exploration [Tao *et al.*, 2011; Rao *et al.*, 1991], moving input points [Gold *et al.*, 1997], or points that have been inserted or deleted [Lee and Gahegan, 2002]. However, these analytic methods are not directly applicable to Voronoi algorithms working on grids.

Several approaches exist to incrementally construct grid-based Voronoi diagrams, for example, the approaches by Guibas *et al.* [1992] and Choset *et al.* [2000]. However, most of them are not suitable for dynamic environments or incremental mapping with SLAM, since they do not support clearing previously occupied map cells, which is necessary to handle dynamic environments and map corrections caused by loop-closures in SLAM.

Tao *et al.* propose line fitting to overcome this problem for a SLAM application [2011]. Their algorithm can incrementally construct analytic Voronoi graphs during exploration, which comprises the addition of obstacles, but not their removal. Furthermore, one would have to update the line fitting as well, which can cause sudden changes in the Voronoi graph.

The approach for updating GVDs proposed by Kalra *et al.* directly operates on grid maps, but introduces obstacle identifiers that are uniquely assigned to a compound of connected obstacle cells [Kalra *et al.*, 2009]. If two adjacent cells have different closest obstacles according to their identifier, both cells are added to the GVD. This condition however generates two-cell-wide lines that violate the sparseness property of the GVD. Additionally, it does not generate Voronoi lines in the interior of concave obstacle compounds like rooms or corridors as shown in Figure 3.1 (left). This destroys the connectivity of the GVD and is problematic for path planning, especially in indoor environments. Furthermore, since Kalra *et al.* use 8-connected step distances for the distance maps, the Voronoi lines also follow this metric and thereby only approximate the GVD.

In this thesis we describe our condition-based approach to incrementally update GVDs, first published in [Lau *et al.*, 2010]. It considers actual Euclidean distances and uses a new criterion that determines if a cell is part of the GVD or not, without requiring obstacle identifiers as the approach by Kalra *et al.* [2009]. Furthermore, our approach correctly handles indoor environments and generates thin Voronoi lines that can be used for the n-best computation of topologically different paths as shown in Figure 3.1. Additionally, it benefits from the speed-up of our dynamic brushfire algorithm described above.

3.3 Configuration Space Maps

Configuration space (c-space) maps encode if a given robot pose leads to a collision with the environment or not. Algorithms for efficient collision checking between three-dimensional objects continue to be an active area of research. Being in an overlap area between motion planning

and computer graphics, most approaches represent the environment and the obstacles with polygon meshes. For example, Tang *et al.* recently proposed a connection collision query algorithm that detects collisions of triangle meshes moving between given states [2011]. Hence, it can be used for sampling-based path planning. For online feasibility, Pan and Manocha use multi-core GPUs for collision queries [2011]. Still, the cost per collision check depends on the number of polygons used to represent the tested objects.

For a collision avoidance system, Schlegel proposed to precompute collision distances for circular arc trajectories as a function of relative obstacle location and curvature [1998]. Thus, the kinematic analysis is done offline and collision distances can be obtained with one lookup per obstacle. Instead, precomputing c-space representations further reduces the online effort for collision checks to a single lookup. Since the publication of the seminal paper by Lozano-Perez on c-space planning among static polyhedric obstacles [1983], many approaches were proposed to reduce the cost for computing c-space obstacles, see for example the survey by Wise and Bowyer [2000]. Linan and Zhenmin for example proposed a method to incrementally grow polygonal c-space obstacles for multiple robots, but did not consider changes other than ongoing exploration [2005]. Because of the relevance of this problem in dynamic environments, researchers are still working on improving the efficiency [Behar and Lien, 2010].

Convolving a grid map of a robot's environment with an image of its footprint yields a discrete c-space map. In order to reflect the current state of previously unknown or moving obstacles at all times, these maps need to be updated regularly. Kavraki proposed to use the fast Fourier transform (FFT) to reduce the computational cost of the convolution [Kavraki, 1995], and Therón *et al.* added parallelization for an additional speed-up [2003]. Later, the same group proposed a multi-resolution approach to reduce memory and computational load in large workspaces [Blanco *et al.*, 2005]. To speed up path planning for an autonomous car, Ziegler and Stiller decompose the shape of the vehicle into circular discs [2010].

As a first dynamic approach for changing environments, Wu *et al.* proposed to precompute colliding robot poses for each potentially occupied cell in the work space of a manipulator [Wu *et al.*, 2006]: taking the union of the colliding poses for a given set of occupied cells yields the c-space collision map without further recomputation. For mobile robots, however, the size of the operational area can render the database storage or the online computation of the union infeasible. In contrast, our method for updating c-space collision maps is truly incremental: it executes an initial map convolution in an offline phase, and during online application only updates the cells affected by changes in the environment [Lau *et al.*, 2011].

For path planning with circular robots, 2D Voronoi diagrams are appealing roadmaps since they cover all topologically different paths in a map with a small number of cells. For rectangular robots however, 2D Voronoi planning loses its completeness property, which requires repairing paths in narrow areas where following the Voronoi diagram leads to collisions, e.g., by using rapidly-exploring random trees (RRTs) as proposed by Foskey *et al.* [2001]. In this thesis we combine dynamic distance maps and Voronoi diagrams with our novel incrementally updatable c-space collision maps. In this way, we overcome the aforementioned problem and can perform complete Voronoi planning in the configuration space of non-circular robots. Due to the ability to perform incremental updates, the resulting systems are suitable for online application in dynamic environments.

Although we use A* planning as an example application, our approach could be combined with other planners, e.g., D* Lite [Koenig and Likhachev, 2002], or RRTs with Voronoi-biased sampling [Lindemann and LaValle, 2004; Zhang and Manocha, 2008].

4 Dynamic Euclidean Distance Maps

This chapter presents our approach to incrementally update Euclidean distance maps for application in dynamic environments. To only update cells affected by changes, it uses a novel dynamic variant of the brushfire algorithm. This method propagates information in wavefronts on a grid, and is therefore also known as wavefront algorithm. In the following, we first review the static brushfire algorithm and then present our modifications to facilitate incremental changes.

4.1 Static Brushfire Algorithm

The brushfire algorithm, as described by Verwer *et al.*, computes static distance maps with a shortest path search similar to Dijkstra's algorithm with multiple sources [Verwer *et al.*, 1989]. By using a priority queue that orders the expansion of cells by the distance to their closest obstacle, the propagation spreads in wavefronts that start at the location of obstacles as shown in Figure 4.1.

The pseudo-code of the static brushfire method is given in Algorithm 4.1. As discussed in Section 3.1, we have modified it to store the location of the closest obstacle of each visited cell in the obstacle reference map $obst(s)$. Given a map $M(s)$, it initializes each free cell s of the distance map $D(s)$ with infinite distance (line 6). The occupied cells are initialized with zero distances and then inserted into the priority queue $open$: the function $\mathbf{insert}(open, s, d)$ inserts s into the queue with distance d, or updates the priority if s is already enqueued (line 5).

As long as this queue contains cells, the algorithm iteratively calls $\mathbf{pop}(open)$, which returns the cell s with the lowest enqueued distance and removes it from the queue (line 8). It then updates the cells in the 8-connected neighborhood $\text{Adj}_8(s)$ of s: if the distance d from a neighbor n to the closest obstacle of s as specified by $obst(s)$ is smaller than the current value $D(n)$ (line 13), the distance value and closest obstacle of n are updated with the obstacle of s (lines 14–15). Furthermore, each updated neighbor cell n is inserted into the priority queue with its new distance value to continue the propagation (line 16). Thus, each obstacle induces a propagation

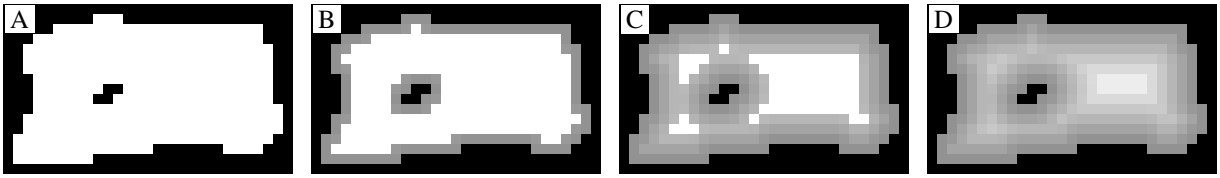


Figure 4.1: Computing a distance map with the static brushfire algorithm. The distance values are initialized with 0 (black) for occupied and infinity (white) for empty cells (A). The occupied cells initiate wavefronts that propagate the increasing distances, denoted by increasing brightness in (B) and (C). A wavefront stops if no further distance values can be lowered. After all wavefronts have stopped, the distance map is complete (D).

Algorithm 4.1 Static Brushfire algorithm for computing Euclidean distance maps

computeDistanceMap()
1: **for all** s **do**
2: **if** $M(s) = 1$ **then**
3: $D(s) \leftarrow 0$
4: $obst(s) \leftarrow s$
5: insert$(open, s, 0)$
6: **else** $D(s) \leftarrow \infty$
7: **while** $open \neq \emptyset$ **do**
8: $s \leftarrow \text{pop}(open)$
9: lower(s)
10: **return** D

lower(s)
11: **for all** $n \in \text{Adj}_8(s)$ **do**
12: $d \leftarrow \|obst(s) - n\|$
13: **if** $d < D(n)$ **then**
14: $D(n) \leftarrow d$
15: $obst(n) \leftarrow obst(s)$
16: insert$(open, n, d)$

wavefront that expands in circles, and updates the distance values of the visited cells to the Euclidean distance to the obstacle. Since this update can only reduce the distance value associated with a cell, we call this type of propagation a "lower" wavefront.[1]

If a wavefront cannot lower the distance of any neighbor of the visited cells, no further cells are enqueued. After all wavefronts came to a halt this way, the priority queue is empty and the algorithm returns the distance map.

4.2 Dynamic Brushfire Algorithm

Movement, insertion, or deletion of objects causes individual cells in a binary occupancy grid map M to flip their state from free (0) to occupied (1) or vice versa. This section presents an approach to update Euclidean distance maps to reflect such changes using a dynamic variant of the brushfire method. After registering an arbitrary set of newly occupied and newly freed cells,

[1] This corresponds to the nomenclature used by Kalra *et al.* [2009].

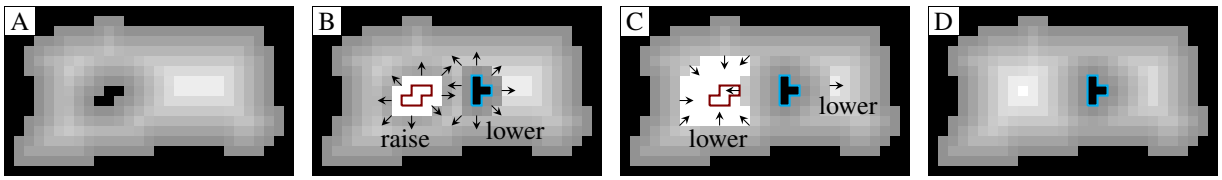


Figure 4.2: The dynamic brushfire algorithm is used to update the distance map (A). To propagate the changes (B), a raise wavefront starts to delete the invalid values for a removed obstacle (marked red), and a lower wavefront propagates the new distances for an inserted obstacle (blue). Where the raise wavefront hits cells with a different (valid) closest obstacle, it halts and initiates a new lower wavefront to restore the invalidated distance values (C). After all wavefronts came to a halt, the update is completed (D).

the algorithm performs the update in an incremental way, i.e., exploiting its previous results. As discussed before in Chapter 3, our approach is directly derived from the brushfire algorithm presented above, unlike the method by Kalra *et al.*, which is derived from D* [2009].

The example in Figure 4.2 shows how the DM from Figure 4.1 is updated after removing an obstacle and inserting a different one. Frame (A) shows the initial state, which is equivalent to the final state (D) of Figure 4.1.

When performing the update, newly occupied cells (blue outline) initiate "lower" wavefronts (B) that update the closest obstacle distance of affected cells similarly to the static variant of the algorithm. These wavefronts are propagated up to the point where a different obstacle is closer (C). In addition, "raise" wavefronts start at newly freed cells (red outline) and clear the distance entries of all cells whose closest obstacle was the deleted one (B). When they come to a halt at cells with a different closest obstacle, they initiate new "lower" wavefronts that recompute the distances for the cleared cells on the basis of the remaining obstacles (C).

Both the raise and lower wavefronts propagate themselves by enqueueing the neighbors of a processed cell into the same priority queue. Since the queue sorts its elements by distance, the processing of raise and lower wavefronts is interwoven. After all wavefronts have stopped, the queue is empty and the update is completed (D).

The pseudo-code of the dynamic brushfire algorithm is given in Algorithm 4.2. The algorithm is initialized with a given distance map $D(s)$ and an obstacle reference map $obst(s)$ for a corresponding occupancy grid map $M(s)$. If a cell s is occupied according to $M(s)$, it has a distance of $D(s) = 0$ and refers to itself as the closest obstacle location, i.e., $obst(s) = s$. Otherwise, $D(s)$ is the distance value to the closest occupied cell, whose location is stored in $obst(s)$. A change in the occupancy of a cell s is registered by calling the function $\mathbf{setObstacle}(s)$ or

Algorithm 4.2 Dynamic Brushfire algorithm to incrementally update distance maps

setObstacle(s)
17: $obst(s) \leftarrow s$
18: $D(s) \leftarrow 0$
19: insert$(open, s, 0)$

removeObstacle(s)
20: clearCell(s)
21: $toRaise(s) \leftarrow$ true
22: insert$(open, s, 0)$

updateDistanceMap()
23: **while** $open \neq \emptyset$ **do**
24: $s \leftarrow$ pop$(open)$
25: **if** $toRaise(s)$ **then**
26: raise(s)
27: **else if** isOcc$(obst(s))$ **then**
28: $voro(s) \leftarrow$ false
29: lower(s)
30: **return** D

raise(s)
31: **for all** $n \in \text{Adj}_8(s)$ **do**
32: **if** $(obst(n) \neq$ cleared $\wedge \neg toRaise(n))$ **then**
33: insert$(open, n, D(n))$
34: **if** $\neg$isOcc$(obst(n))$ **then**
35: clearCell(n)
36: $toRaise(n) \leftarrow$ true
37: $toRaise(s) \leftarrow$ false

lower(s)
38: **for all** $n \in \text{Adj}_8(s)$ **do**
39: **if** $\neg toRaise(n)$ **then**
40: $d \leftarrow \|obst(s) - n\|$
41: **if** $d < D(n)$ **then**
42: $D(n) \leftarrow d$
43: $obst(n) \leftarrow obst(s)$
44: insert$(open, n, d)$
45: **else** checkVoro(s, n)

removeObstacle(s), which updates s and inserts it into a priority queue. Thereby, the function **clearCell**(s) resets s to $D(s) = \infty$ and $obst(s) =$ cleared.[2] An additional flag $toRaise$ is used to ensure proper processing of cells in the wavefronts, in particular where raise and lower wavefronts meet. It indicates for each cell, whether it has to process its neighbors with a raise wavefront (true) or not (false).

After registering a set of changes using **setObstacle**(s) and **removeObstacle**(s), the priority queue $open$ contains the updated cells. Calling **updateDistanceMap**() performs the actual update by propagating the changes to all affected cells. While the priority queue is not empty, it repeatedly retrieves the next unprocessed cell s (lines 23–24). If s has still to propagate a raise wavefront, the function **raise**(s) is called (lines 25–26). If this is not the case and if s has a valid closest obstacle, the function **lower**(s) is called to propagate the lower wavefront (lines 27–29). The function **isOcc**(s) tests if a cell s is occupied by checking if $obst(s) = s$.

The function **raise**(s) processes each cell n in the 8-connected neighborhood $\text{Adj}_8(s)$ of s that has not been raised and still refers to a closest obstacle $obst(n)$ (lines 31–32).

[2] Without a precomputed distance map, e.g., when mapping a new area, the algorithm can also be initialized by clearing all cells and registering occupied cells with **setObstacle**(s).

The cell n is inserted into the priority queue with its old distance value (line 33). If the cell referenced by $obst(n)$ is no longer occupied, n is cleared and marked to propagate the raise wavefront (lines 34–36). Otherwise, the raise wavefront comes to a halt at n, leaves n unchanged, and propagates a lower wavefront, as shown in Figure 4.2 (C). After processing the neighbors, the raise update of s is completed and $toRaise(s)$ is set to false (line 37).

The function $\mathbf{lower}(s)$ considers each cell n in the 8-connected neighborhood $\text{Adj}_8(s)$ of s. If a cell n is not marked to be part of a raise wavefront (lines 38–39), it is updated as in the static version of the algorithm: the Euclidean distance from n to the closest obstacle of s is compared to the current closest obstacle distance of n (lines 40–41). If it is smaller, the values for distance and closest obstacle of n are updated to reflect that $obst(s)$ is now the closest obstacle of n as well. Also, n is inserted into the priority queue to propagate the lower wavefront (lines 42–44). To avoid superfluous raise wavefronts where they would overlap with lower wavefronts, the condition in line 41 can be extended to also overwrite cells with equal distance that refer to a deleted obstacle. With this modification the line reads

if $d < D(n) \vee (d = D(n) \wedge \neg\text{isOcc}(obst(n)))$ **then**.

The lines 28 and 45 are hooks to incrementally update a Voronoi diagram on the fly during the update of the distance map (see Chapter 5). If only distance maps are required, these lines can be omitted.

4.3 Implementation Details

The distance map algorithm described above computes and compares real-valued Euclidean distances stored in $D(s)$. As previously done by Scherer *et al.* and others, we resort to integer squared distances in practice which saves the computational expenses for the square-root. Due to the strict monotony of the square root function for positive inputs, this does not change the behavior of the algorithm.

A central data structure in our algorithm is the sorted priority queue $open$. Such queues are often implemented using search on a binary tree, which yields a complexity of $O(\log n)$ for the insert operation where n is the number of enqueued elements. Since the processing of cells is ordered by distances and cells only enqueue their direct neighbors, many elements in the priority queue have identical distance values. We exploit this by implementing the queue using the bucketing

Algorithm 4.3 Improved expansion of lower wavefronts in 3D,
replaces line 38 in Algorithm 4.2

```
46: w ← (s − obst(s))
47: for all n ∈ Adj_26(s) do
48:     Δ ← (n − s)
49:     if ∃c ∈ {x, y, z} : w_c·Δ_c < 0 then continue
```

technique presented by Cuisenaire and Macq [1999], where cells with the same distance are grouped in unsorted lists.

To implement priority queues with unique entries and increasable priorities, we actually insert the elements whenever they are updated, and carry a Boolean flag $toProcess$ for each cell s. It is set to true by $\mathbf{insert}(open, s, d)$ and reverted to false by $\mathbf{pop}(open)$. The function $\mathbf{pop}(open)$ iteratively dequeues elements until it reached an element s with $toProcess(s) =$ true, and thus discards duplicated entries.

4.4 Extension to Higher Dimensions

In the context of mobile robot navigation, distance transforms have mostly been applied to two-dimensional maps. However, for flying robots or manipulators, 3D distance maps are also very appealing. Our dynamic brushfire algorithm can directly be extended to 3D. Obviously, the obstacle locations, the obstacle reference map, and the distance map itself have to be 3D vectors and arrays in this case. Each cell on a three-dimensional grid has 26 neighbors, so Adj_8 is replaced by Adj_{26}.

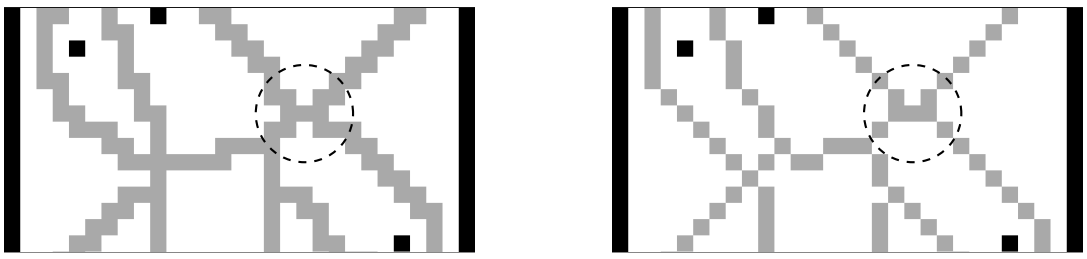
Depending on the map size and the amount of changes in the environment, maintaining a complete 3D distance map may not be feasible even with an incremental update algorithm. As proposed by Scherer *et al.* [2009] we introduce an upper bound d_{max} on the distances that we propagate. Whenever the increasing distances in a lower wavefront reach this threshold, the propagation is stopped. The appropriate value for d_{max} depends on the application and the available computational resources. Our experiment in Section 7.3 demonstrates on real data how the choice of d_{max} influences the required computation time of our algorithm. To implement the distance bound in our algorithm, the condition $d < d_{\text{max}}$ has to be added as an additional requirement in line 41. During initialization, the distance values in the empty cells are set to d_{max} rather than infinity.

The efficiency of the lower wavefront can be improved by reducing the size of the neighborhood that is expanded for each cell s: the neighbors that lie in the inverse direction of the wavefront's propagation can be skipped. The pseudo-code for this modification is shown in Algorithm 4.3. The direction of the wavefront at s can be determined from $s - obst(s)$ (line 46), and the direction of the potential expansion to neighbor n by $n - s$ (line 48). If these vectors have opposing signs in any component x, y, z, the expansion of n can be skipped (line 49). In theory, this modification could be applied in 2D as well. In this case, however, the computational overhead exceeds the benefit of the reduced cell visits.

5 Dynamic 2D Voronoi Diagrams

In continuous space, a point is part of the Voronoi graph if the distances to its two closest obstacles are identical. For discrete GVDs, this condition cannot directly be applied to the discretized cell coordinates. Instead, the GVD is the set of cells that would contain continuous Voronoi lines in their associated area. Furthermore, the implicit grouping of occupied cells to obstacles plays an important role: treating each occupied cell as a single obstacle would cause the GVD to be cluttered, since a line would be inserted between each pair of adjacent occupied cells. In contrast, treating all connected occupied cells as a single obstacle causes missing Voronoi lines in indoor environments as shown in Figure 3.1 (left).

Voronoi graphs in continuous spaces consist of infinitely thin lines and curves. Since GVDs are represented on discretized grids, artifacts in the form of erroneous connections can occur. Firstly, a pair of nearby Voronoi lines that pass through adjacent cells becomes connected and thus creates erroneous circles and interconnections in the graph. Secondly, a single Voronoi line that lies between two discrete cell locations in continuous space causes double lines in the GVD. In both cases, the GVD loses the sparseness property of the Voronoi graph, i.e., the paths in the GVD no longer correspond to topologically different routes with respect to obstacles. When using an 8-connected grid model, the GVD appears to be thinner by visual inspection. However, the additional connections often create additional path variations in adjacent cells. Thus, 8-connected GVDs often violate the sparseness condition, which is not the case for 4-connected ones (see Figure 5.1 for an example).



(a) 4-connected (b) 8-connected

Figure 5.1: Voronoi diagrams on 4- and 8-connected grids. In the 4-connected GVD, different paths correspond to topologically different routes with respect to obstacles. The 8-connected Voronoi lines might appear thinner on visual inspection, but create interconnections (encircled) with multiple paths.

Algorithm 5.1 Evaluation of the Voronoi condition

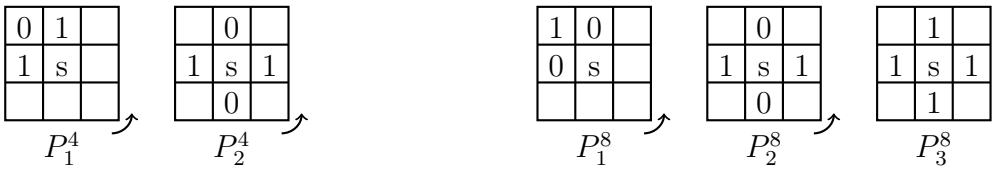
checkVoro(s, n)
50: **if** $(D(s)>1 \vee D(n)>1) \wedge obst(n) \neq$ cleared
$\wedge\ obst(n) \neq obst(s) \wedge obst(s) \notin \mathrm{Adj}_8(obst(n))$ **then**
51: **if** $\|s-obst(n)\| \leq \|n-obst(s)\|$ **then** $voro(s) \leftarrow$ true
52: **if** $\|n-obst(s)\| \leq \|s-obst(n)\|$ **then** $voro(n) \leftarrow$ true

We present a set of conditions to generate GVDs that are fully connected, and at the same time have no neighboring Voronoi lines that touch each other, as shown in Figure 3.1 (right). Although we focus on 4-connected GVDs, our method can generate 8-connected ones as well. An additional pruning step deals with artifacts due to discretization, i.e., double lines and erroneous connections, and thus ensures the sparseness of the GVD. The algorithm is directly integrated with our method for updating distance maps and is easy to implement.

5.1 Incremental Update of Voronoi Diagrams

We represent the GVD by a binary map $voro(s)$, which specifies for each cell s if it is part of the GVD ($voro(s) =$ true) or not ($voro(s) =$ false). The update of the GVD directly integrates with the update of DMs given by Algorithm 4.2 in Chapter 4: lower wavefronts remove all visited cells from the GVD (line 28), and potentially add the cells where they come to a halt. If the lower wavefront propagated by a cell s finds an adjacent cell n whose distance cannot be lowered by adopting $obst(s)$ as closest obstacle, both cells are candidates for the GVD (line 45), and are potentially added after checking our additional conditions in **checkVoro**(s, n) according to Algorithm 5.1. This function tests if at least one of the candidate cells s and n is not adjacent to its closest obstacle (line 50). Furthermore, the neighbor n has to have a valid closest obstacle that is different and not adjacent to the closest obstacle of s. If these conditions are fulfilled, a Voronoi line passes between the cell centers in sufficient distance to the next one, and both cells are candidates for the discrete GVD.

To avoid double lines, the function only adds the cell $c \in \{s, n\}$ that violates the continuous Voronoi condition to the lesser degree, i.e., the one with the smaller distance increase when switching from its own closest obstacle to the one of the competing neighbor. If both have the same increase, both cells are inserted (lines 51–52). To obtain 8-connected GVDs, the "$\leq$" in these lines are replaced by "$<$" for diagonal neighbors s and n, since then no cells need to be inserted in the case of equal increase.



(a) 4-connected (b) 8-connected

Figure 5.2: Image operator patterns used to test the connectivity of a GVD. Arrows indicate application of rotated copies.

5.2 Pruning

As discussed before, different paths on the Voronoi graph correspond to topologically different routes in the environment. To preserve this property for GVDs on grid maps, thin Voronoi lines, i.e., being one pixel wide, are desired. Previous work on dynamic GVDs by Kalra *et al.* however regularly generates Voronoi lines that are two or three pixels wide. Our optional pruning step erodes 2-pixel-wide Voronoi lines that occur where a continuous Voronoi line would pass exactly between two cells. Therefore, all new Voronoi cells are inserted into a priority queue and processed by the pruning stage.

The image operator patterns shown in Figure 5.2 match whenever the center cell s provides connectivity for one or more of its adjacent cells. The left side shows the two patterns required for ensuring 4-connectedness, the three patterns on the right side correspond to 8-connectedness. In any pattern, a "1" matches $voro(s) = \text{true}$, while "0" stands for $voro(s) = \text{false}$, and empty fields are ignored. Where indicated by arrows, the same pattern is applied in all unique 90 degree rotations.

In a first phase, the pruning algorithm merges Voronoi lines that are erroneously connected due to the finite map resolution. This is done using the matching pattern P_3^8, which detects cells that are enclosed by Voronoi cells. If such a cell is free and not part of the GVD, it is added at this point. This merges Voronoi lines that are too close to be separated given the map resolution. Together with the following pruning step, this prevents erroneous connections and ensures that the generated GVD is sparse.

The second phase implements the actual pruning step. In increasing order of distance, the enqueued cells are iteratively popped from the priority queue. If such a cell has more than one neighbor on the GVD and is not required to keep the GVD connected, it can be removed from the GVD without affecting its topology. Again, this is tested using the image operator patterns

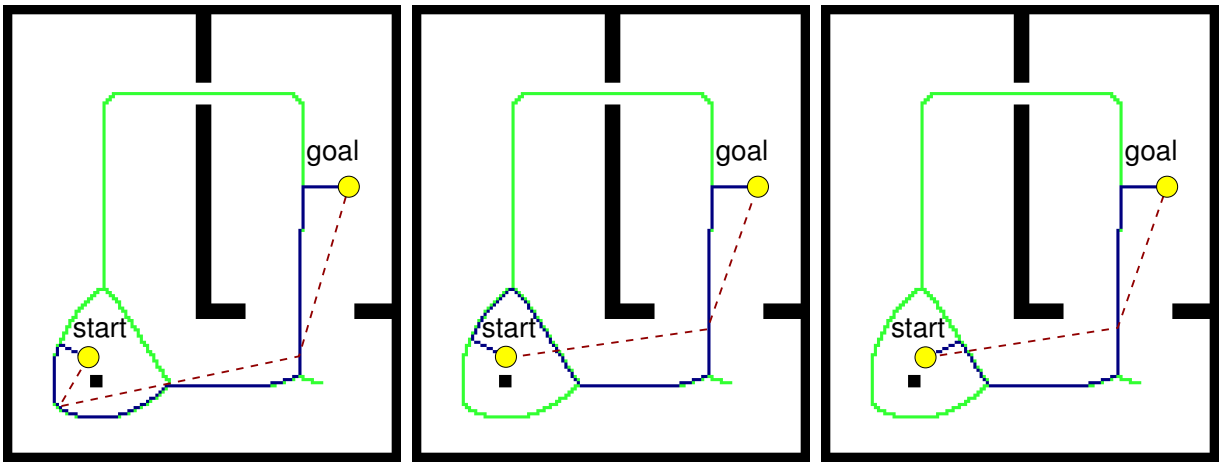
(4- or 8-connected): if none of the connectivity patterns match at the cell location, the cell is not required in the GVD.

5.3 Path Planning on Voronoi Diagrams

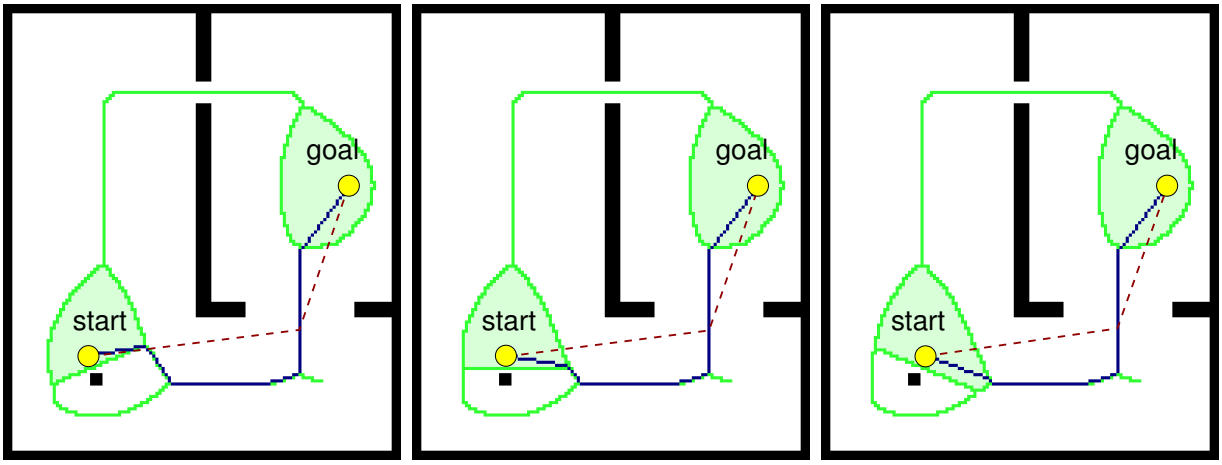
As mentioned before, a GVD is a cell decomposition method that is appealing for path planning. This section details on important aspects of Voronoi planning in dynamic environments. In general, the start and goal locations of a planning problem are not part of the GVD. Straightforward approaches search for the closest Voronoi cell at both locations, and connect them with straight lines to the graph, see for example the work by Geraerts and Overmars [2004]. This is problematic in practice, since a small change of the start pose can substantially change the planned path as shown in Figure 5.3a. On the other hand, running a goal-driven search up to the first Voronoi cell can easily expand a big part of the space that is not on the GVD.

Our approach given by Algorithm 5.2 overcomes these problems. First, we insert virtual obstacles at the start and goal location (lines 54–55). After updating the distance map and the GVD with our incremental algorithm (line 56), these locations become enclosed by Voronoi lines that form "bubble"-like areas as shown in Figure 5.3b. With a simple brushfire expansion, we mark all cells in the bubbles up to the enclosing Voronoi lines (lines 57–58). Now we can start a goal-directed search that is restricted to cells that are either marked or belong to the GVD. In this way, the search expands from the start onto the Voronoi graph, follows Voronoi lines, and then connects to the goal when reaching the goal bubble. Since the whole path is the result of goal-directed graph search, the consecutive paths planned for a moving robot are very similar to each other and do not change abruptly (see Figure 5.3). After the shortest path is computed, we undo the changes to the GVD by removing the virtual obstacles and performing another update (lines 63–65).

In order for the algorithm to be truly incremental, the markers should be stored in a hash map structure rather than in a binary grid that has to be cleared after each frame. Additionally, one can employ an incremental replanning algorithm like D* rather than A*.



(a) Connecting the start and goal to closest Voronoi cells



(b) Our method: Voronoi "bubbles" at start and goal

Figure 5.3: Connecting start and goal to the Voronoi graph (green) during planning: using the shortest connection (a), the planned path (blue) can change abruptly for small changes of the start configuration, even for sightline-pruned paths (dashed). We create Voronoi bubbles around start and goal (b), and use goal-directed search therein, which yields more stable paths (bottom).

Algorithm 5.2 "Bubble"-technique for path planning on a GVD

planPath(*start*, *goal*)
53: **if** $M(start)=1 \vee M(goal)=1$ **then return**
54: setObstacle(*start*) // create Voronoi bubbles
55: setObstacle(*goal*)
56: updateDistanceMap()
57: brushfireMark(*start*) // mark bubbles as searchable
58: brushfireMark(*goal*)
59: push(*astarqueue*, *start*)
60: **while** *astarqueue* $\neq \emptyset$ **do** // A* on Voronoi and bubbles
61: $s \leftarrow$ pop(*astarqueue*)
62: **if** $s = goal$ **then**
63: removeObstacle(*start*)
64: removeObstacle(*goal*)
65: updateDistanceMap()
66: **return** *path from start to goal*
67: **for all** $n \in \mathrm{Adj}_4(s)$ **do**
68: **if** $voro(n) = \text{true} \vee marked(n) = \text{true}$ **then**
69: *A* update for costs and heuristic of n*
70: push(*astarqueue*, n)

brushfireMark(s)
71: push(*unsortedqueue*, s)
72: **while** *unsortedqueue* $\neq \emptyset$ **do**
73: $s \leftarrow$ pop(*unsortedqueue*)
74: $marked(s) = \text{true}$
75: **for all** $n \in \mathrm{Adj}_4(s)$ **do**
76: **if** $M(n)=1$ **then continue**
77: **if** $voro(n) = \text{false} \wedge marked(n) = \text{false}$ **then**
78: push(*unsortedqueue*, n)

6 Dynamic C-Space Representations

This chapter presents our approach to compute and incrementally update collision maps in the configuration space of mobile robots moving on a plane. By combining these maps with our dynamic distance maps and Voronoi diagrams, we also generate c-space distance maps and c-space Voronoi diagrams.

6.1 Dynamic C-Space Collision Maps

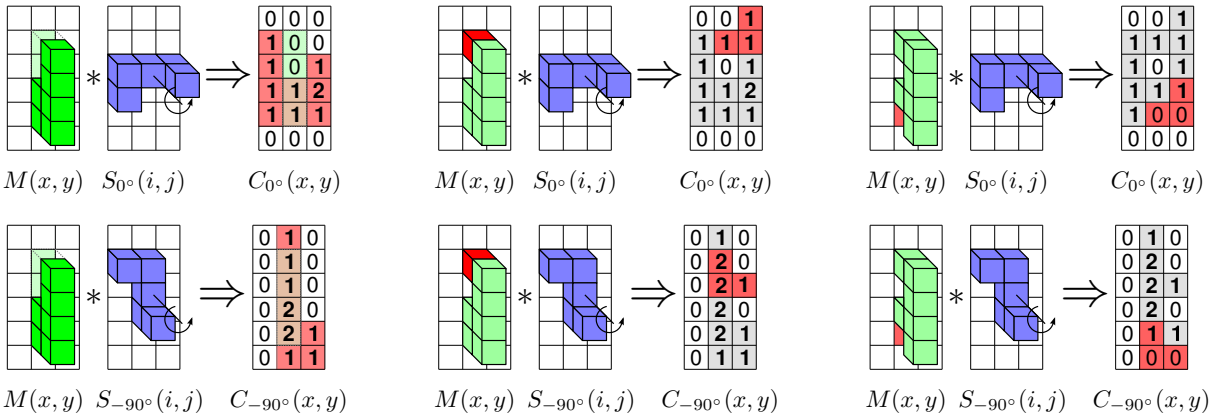
As discussed in Section 3.3, a configuration space (c-space) collision map encodes for each discretized robot configuration whether it causes a collision with obstacles in the environment or not. Non-circular robots moving on a plane have a three-dimensional c-space, since their poses $\langle x, y, \theta \rangle$ are given by their 2D position on the ground and their orientation θ.

Computing a c-space map usually requires convolving a map with the shape of the robot for each orientation. Recomputing these convolutions to reflect changes in dynamic environments is often not feasible at frame rates required for online applications. We present a method to efficiently update c-space collision maps with the obstacle models shown in Figure 2.1. For the sake of clarity, we first describe our algorithm for a 2.5D representation with overhanging obstacles, and discuss the adaptation to other obstacle models later.

Let $M(x, y)$ be a grid map that represents the vertical clearance, i.e., the height of free space above the floor, with zeros for completely occupied cell columns. Consider a robot moving on the floor with continuous orientation $\tilde{\theta}$ with respect to the map coordinate system. We represent the discretized shape of the robot for a given orientation $\tilde{\theta}$ by a map $S_{\tilde{\theta}}(i, j)$, that stores the height of the robot for every cell of its footprint. $S_{\tilde{\theta}}$ has the same resolution and orientation as the map M, whereas its origin $S_{\tilde{\theta}}(0, 0)$ is located at the center of the robot.

A convolution-type conjunction of M and $S_{\tilde{\theta}}$ yields a count map $C_{\tilde{\theta}}(x, y)$ as shown in Figure 6.1a. Each cell $\langle x, y \rangle$ in $C_{\tilde{\theta}}$ stores the number of cells the robot collides with when located there, and is computed according to

$$C_{\tilde{\theta}}(x, y) = \sum\nolimits_i \sum\nolimits_j \text{eval}\{M(x+i, y+j) \leq S_{\tilde{\theta}}(i, j)\} \ , \tag{6.1}$$



(a) Initial convolution of map (green) and robot (blue)

(b) Updating C after a cell in M has been occupied

(c) Updating C after a cell in M has been cleared

Figure 6.1: Convolving a map $M(x, y)$ with a representation of the robot's shape $S_\theta(i, j)$ for orientation θ yields a collision map $C_\theta(x, y)$. Each cell $\langle x, y\rangle$ in C_θ counts the cells in the robot footprint that collide with occupied cells in M, given the robot pose $\langle x, y, \theta\rangle$. A newly occupied or emptied cell in the map M (red) increments or decrements the affected collision counts in C, respectively. This updates the collision map (red cells) without recomputing values for unaffected cells (gray).

where $\text{eval}(\text{true}) = 1$ and $\text{eval}(\text{false}) = 0$. If we discretize $\tilde{\theta}$ and stack the $C_\theta(x, y)$ for all discrete θ, we obtain the robot's c-space collision count map $C(x, y, \theta)$ for M. Clearly, by testing $C(x, y, \theta) > 0$ we can check if the discretized pose $\langle x, y, \theta\rangle$ is colliding. By storing collision counts instead of just binary values as in regular c-space maps, we can update the c-space map incrementally as described below.

6.2 Incremental Update of the C-Space Map

Unknown or moving obstacles cause changes in the map of a robot's environment. For the 2.5D obstacle model, a change is given by an updated vertical clearance v_{new} for a cell $\langle x, y\rangle$ in M. To refresh C incrementally rather than computing it from scratch, we only update the affected parts of the sum in Equation (6.1) according to Algorithm 6.1. See the sequence of Figures 6.1b-6.1c for an illustration.

The algorithm separately updates the θ-layers of C, and can thus be parallelized (line 81). For each cell $\langle i, j\rangle$ of the robot shape $S_\theta(i, j)$ we visit the robot position $\langle x', y'\rangle$ that lets $\langle i, j\rangle$ fall on $\langle x, y\rangle$ (line 82). These cells can efficiently be selected using standard drawing algorithms for rasterized images.

Algorithm 6.1 Dynamic update of C-space collision maps

updateVerticalClearance(x, y, v_{new})

79: $v_{\text{old}} \leftarrow M(x, y)$
80: $M(x, y) \leftarrow v_{\text{new}}$
81: **for all** θ **do**
82: **for all** $\langle x', y' \rangle \in \{\langle x-i, y-j \rangle \mid S_\theta(i, j) > 0\}$ **do**
83: **if** $v_{\text{new}} \leq S_\theta(i, j) \wedge v_{\text{old}} > S_\theta(i, j)$ **then**
84: $C(x', y', \theta) \leftarrow C(x', y', \theta) + 1$
85: **if** $C(x', y', \theta) = 1$ **then** newOccupied(x', y', θ)
86: **else if** $v_{\text{new}} > S_\theta(i, j) \wedge v_{\text{old}} \leq S_\theta(i, j)$ **then**
87: $C(x', y', \theta) \leftarrow C(x', y', \theta) - 1$
88: **if** $C(x', y', \theta) = 0$ **then** newEmpty(x', y', θ)

If the new vertical clearance v_{new} in $\langle x, y \rangle$ causes a collision with $S_\theta(i, j)$ while v_{old} did not, the collision counter of $\langle x', y' \rangle$ is incremented (line 83), since this represents a new collision candidate cell. Vice versa, if v_{new} is collision-free whereas v_{old} collided, the counter is decremented (line 86), since a collision candidate was removed. Whenever the count changes from 0 to 1 or from 1 to 0, the pose $\langle x', y', \theta \rangle$ is newly occupied (line 85) or emptied (line 88), respectively. These events can be used to trigger further computation, e.g., to update the c-space distance map and Voronoi diagram discussed in Section 6.5.

6.3 Discretization of Orientations

An appropriate discretization of $\tilde{\theta}$ ensures that if two adjacent poses $\langle x, y, \theta_i \rangle$ and $\langle x, y, \theta_{i+1} \rangle$ are collision-free according to C, intermediate orientations in $[\theta_i, \theta_{i+1}]$ are collision-free as well. Under this constraint we seek to discretize $\tilde{\theta}$ as coarse as possible to keep the number of θ-layers in C small.

In occupancy grid maps, the actual location of obstacles can be anywhere in the cells they occupy. Therefore, one usually assumes an additional safety margin m around the robot, e.g., of $m = 1$ pixel unit. Given this margin, we can formulate a bound on the angular resolution for the discretization of $\tilde{\theta}$ as follows: if the robot rotates from θ_i to θ_{i+1}, each point on the robot moves along an arc. The maximum arc length occurs at the outmost point of the robot, which is the radius r (in pixels) of the circumcircle around its center of rotation. By choosing a resolution of $|\theta_i - \theta_{i+1}| = m/r$, we ensure that even in the worst case an obstacle collides only with the safety margin but not with the actual robot. Depending on the shape of the robot, less conservative bounds on the discretization can be formulated.

6.4 Adaptation to Other Obstacle and Robot Models

Up to this point, we assumed overhanging obstacles and a robot on the floor that can move underneath obstacles as in Figure 2.1b. By reversing the comparisons of robot height and vertical clearance in Equation (6.1) and Algorithm 6.1 (lines 83 and 86), this can easily be adapted to obstacles elevating from the floor and robots with overhanging load or parts as in the figure. For plain 2D robot and obstacle models, the heights v_{new} and v_{old} are binary values that encode occupied (true) and free (false). In that case, the conditions for determining newly occupied cells in line 83 are given by

"**if** $v_{\mathrm{new}} = \mathrm{true} \wedge v_{\mathrm{old}} = \mathrm{false}$ **then**",

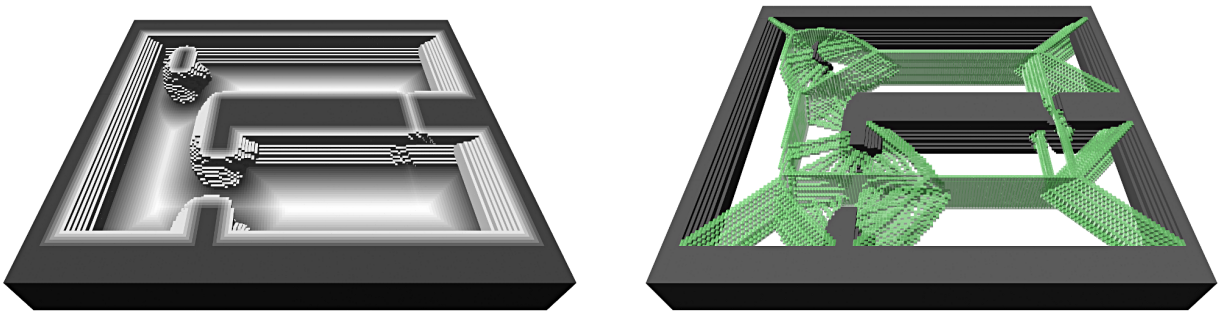
and for newly vacated cells in line 86 by

"**if** $v_{\mathrm{new}} = \mathrm{false} \wedge v_{\mathrm{old}} = \mathrm{true}$ **then**".

For some applications, the obstacles and the robot have to be represented in full 3D as in Figure 2.1c. The height comparisons in Algorithm 6.1, lines 83 and 86 then have to consider lists of obstacle heights. If the robot shape is approximated by a set of vertical columns with a given upper and lower end as in Figure 2.1, one can also use a separate shape map for each column. If line 82 is adapted to only consider the columns that potentially collide with a given new obstacle, the c-space collision map can be efficiently updated.

In applications like mobile manipulation or autonomous transport, the shape of the robot and its payload can vary over time. To use our method in these cases, one can group the changing parts of the robot to additional models, and maintain separate c-space maps for them. Then, one can immediately switch between different configurations of the robot by using different sets of models in the collision check. If highly accurate collision checks are required, one can also create shape models for an inner and an outer approximation. Only if the outer (larger) approximation collides while the inner (smaller) one does not, more complex methods like mesh queries are required. Otherwise, the approximations are sufficient.

Robots with a symmetric shape with respect to their center cause a part of the θ-layers in $C(x, y, \theta)$ to be redundant. For example, a rectangular robot rotating around its center causes the same c-space obstacles at orientation $180°$ as at $0°$. Omitting the respective layers when iterating over θ in Algorithm 6.1 (line 81) saves a substantial part of the computational effort and memory consumption.



(a) C-space distance map (b) C-space Voronoi diagram

Figure 6.2: C-space distance map and Voronoi diagram for a rectangular robot obtained by stacking layers computed in 2D for different robot orientations θ. For readability, only half of the layers are shown, the other half is identical due to the symmetry of the robot. In the visualization at the top, cells above the bottom layer have a different color scaling and were removed when exceeding a distance threshold.

6.5 C-Space Distance Maps and Voronoi Diagrams

This section describes how to compute DMs and GVDs presented in Chapters 4 and 5 for the c-space collision map in Section 6.1. Since we provide incremental update algorithms for all these representations, these combinations are suitable for online applications as well, and thus open new possibilities for collision checking and path planning in the configuration space of mobile robots.

Given a three-dimensional c-space collision map $C(x, y, \theta)$ as described above, we can compute a 3D distance map in this space that uses a 3D distance measure which combines Euclidean distances and the angle of rotation. Therefore, one has to consider the angle wrap-around of the θ component for the expansion of neighborhoods. The resolution of the θ discretization specifies how to balance Cartesian and angular distances. With the methods presented in Section 4.4, such a c-space DM can be updated incrementally, and can for example be used to efficiently perform collision checks for non-circular robots on long trajectories.

As discussed by Canny [1985], it is also appealing to only consider 2D Euclidean distances per θ-layer of the c-space map. Therefore, we stack Euclidean distance maps $D_\theta(x, y)$ computed for every c-space map layer $C_\theta(x, y)$, yielding the c-space distance map $D(x, y, \theta)$ as shown in Figure 6.2 (top).

In 2D, GVDs are the union of points whose two closest obstacles are at the same distance. Just as for the DMs, we compute a GVD $voro_\theta(x, y)$ for every c-space map layer $C_\theta(x, y)$. Stacking

these Voronoi diagrams results in a c-space Voronoi diagram $voro(x, y, \theta)$ as shown in Figure 6.2 (bottom), which is fundamentally different from computing the 3D generalized Voronoi diagram for $C(x, y, \theta)$. If θ is discretized according to Section 6.3, the Voronoi lines in neighboring layers connect to Voronoi surfaces.

To update the layers D_θ and $voro_\theta$, we first update the underlying c-space collision map. The events $\mathbf{newOccupied}(x', y', \theta)$ and $\mathbf{newEmpty}(x', y', \theta)$ in Algorithm 6.1 are used to call $\mathbf{setObstacle}(x, y)$ and $\mathbf{removeObstacle}(x, y)$ in the respective θ-layer to register newly occupied or vacated cells. After the update of the c-space map is completed, we call the function $\mathbf{updateDistanceMap}()$ for each θ-layer, which completes the update of the c-space DM and GVD. Since their θ-layers are independent, this can be parallelized on multi-core CPUs.

6.6 C-Space Voronoi Path Planning

Given a layered c-space GVD as described in the previous section, one can perform deterministic and complete path planning for non-circular mobile robots without a non-holonomic constraint. The search on the c-space GVD is very similar to 2D Voronoi planning. The major difference is the added dimension of the orientation with its cyclic nature, which has to be considered in the neighborhoods during expansion. The bubble planning method presented in Section 5.3 can easily be adapted for these purposes. Therefore, the creation and brushfire expansion of the Voronoi bubbles (lines 54–58) and the corresponding removal (lines 63–65) have to be executed for each θ-layer independently, which can be parallelized on multi-core CPUs. Obviously, the A* algorithm has also to be modified to search the subspace of the three-dimensional c-space, which is given by the GVD and the start/goal bubbles.

When using 8-connected GVDs, the brushfire expansion has to be run using a 4-connected neighborhood to ensure that the expansion is contained in the start and goal bubbles. If the GVD is 4-connected as in our examples, the brushfire expansion may also use 8-connectedness.

7 Experiments

This chapter presents experiments conducted to test our algorithms on real-world data. We analyze the computational requirements and show examples of generated output. The tests were done using our C++ implementation of the algorithms, running on an Intel® Core™ i7 2670 MHz. The source code of our algorithms and a big part of the employed data sets are available online [Lau *et al.*, 2012].

7.1 2D DMs and GVDs in Dynamic Environments

For this set of experiments we used a Pioneer robot equipped with a SICK LMS291 laser range finder. To record data, it was moving in environments where walking people heavily affected the traversable space (see Figure 7.1). The sequence "FR079" consists of 369 frames recorded in an office building, and "FR101" contains 400 frames recorded in a large foyer space. The update radius around the robot was only limited by the maximum range of the laser scanner (80 m). Due to the maximum room size in the environments, the maximum closest obstacle distance in these two maps, i.e., the radius of the largest circular unoccupied area, is 29 cells (1.45 m) and 97 cells (4.85 m), respectively. The 400 frames of "Factory" were simulated by randomly inserting 200 obstacles per frame into a grid map of a large factory floor with a maximum obstacle distance of 44 cells (2.2 m). Similar to the work by Kalra *et al.* [2009], the random obstacles were placed within 5 m radius around a moving center, which simulates a moving observer with limited perception.

To demonstrate the computational benefit of dynamically updatable DMs, we compared our algorithm with state-of-the-art static methods implemented by Fabbri *et al.* [2008], namely the algorithms by Cuisenaire and Macq [1999] and Maurer *et al.* [2003]. These approaches are highly efficient, but recompute the whole distance map in every frame. We further compared our method to the recent approach by Scherer *et al.* [2009]. Since no source code was available, we implemented this algorithm in C++ with the assistance of Scherer. Our GVD approach is compared to the static EVG-Thin method implemented by Beeson [2006] and the dynamic approach by Kalra *et al.* [2009].

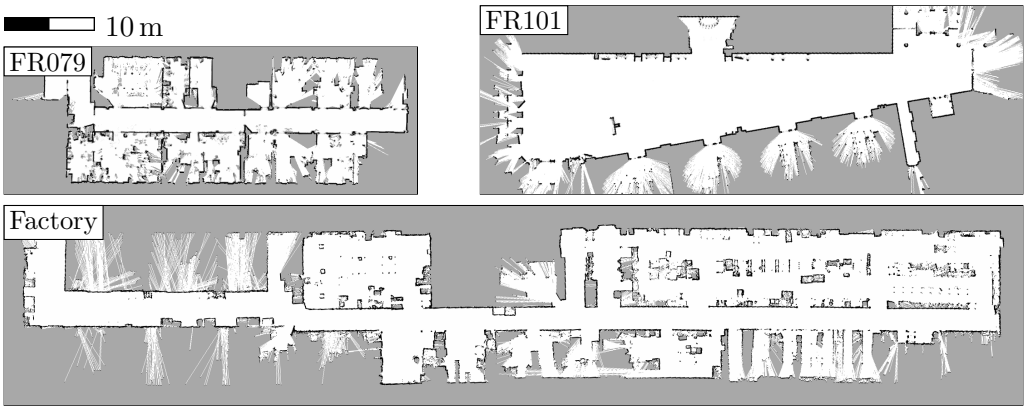
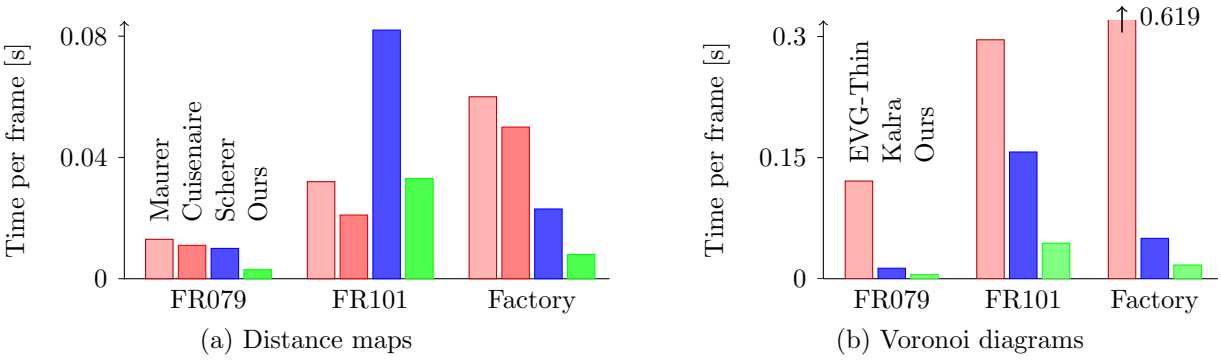


Figure 7.1: Maps of the environments where we recorded 2D laser range data.



(a) Distance maps

(b) Voronoi diagrams

Figure 7.2: Performance of our algorithms for updating distance maps and Voronoi diagrams compared to related work. The plots show average computation times per frame.

In the first frame of each sequence, the algorithms were initialized with the corresponding grid map shown in Figure 7.1, using a resolution of 0.05 m per grid cell. The performance is visualized in Figure 7.2 and presented by numeric results in Table 7.1. For each sequence, the table provides the computation time and cell visits per frame with their mean, minimum, and maximum values. When repeating the measurements 10 times for each sequence, the standard deviations between the runs were well below 1% of the reported means.

In general, the dynamic methods are considerably faster than the static approaches by Cuisenaire and Macq [1999] and Maurer *et al.* [2003], except for the distance maps in the open space of FR101, where most updates affect a large fraction of the map. In all frames of all sequences, our dynamic distance map algorithm visits $60-70\%$ fewer cells and requires $60-70\%$ less computation time than the dynamic approach by Scherer *et al.* [2009]. This can be mainly attributed to the raise function of their algorithm which expands the adjacent cells of the neighbors of a cell s, whereas our algorithm tests only the direct neighbors (line 38). Note that the cell visits performed by the static methods are not directly comparable to the dynamic ones due to the

Table 7.1: Update performance of distance maps and Voronoi diagrams

			Time per frame [s]			Cell visits per frame		
Map & Approach			mean	min	max	mean	min	max
Distance Maps	FR079	Maurer	0.013	0.013	0.013	1,393,286	1,392,450	1,394,272
		Cuisenaire	0.011	0.010	0.011	302,513	301,022	304,138
		*Scherer	0.010	0.006	0.019	250,403	77,277	657,903
		*Ours	0.003	0.001	0.005	99,761	29,340	190,998
	FR101	Maurer	0.032	0.032	0.033	3,299,105	3,296,201	3,302,497
		Cuisenaire	0.021	0.021	0.021	572,345	562,474	581,158
		*Scherer	0.082	0.026	0.148	3,338,297	792,054	6,190,219
		*Ours	0.033	0.011	0.051	1,264,488	427,176	1,929,690
	Factory	Maurer	0.060	0.060	0.060	5,954,325	5,954,259	5,954,447
		Cuisenaire	0.050	0.050	0.052	959,484	957,735	961,709
		*Scherer	0.023	0.003	0.032	976,292	80,262	1,307,630
		*Ours	0.008	0.002	0.011	319,871	80,262	423,315
Voronoi Diagrams	FR079	EVG-Thin	0.121	0.120	0.122	10,030,438	10,001,973	10,059,575
		*Kalra	0.013	0.006	0.030	483,242	281,126	933,410
		*Ours	0.005	0.003	0.008	113,803	31,824	215,131
	FR101	EVG-Thin	0.296	0.282	0.310	19,892,173	19,798,905	20,005,447
		*Kalra	0.157	0.046	0.284	4,363,678	1,537,112	7,558,395
		*Ours	0.044	0.018	0.066	1,372,060	475,163	2,087,083
	Factory	EVG-Thin	0.619	0.592	0.632	35,540,331	35,525,379	35,551,489
		*Kalra	0.050	0.005	0.068	1,462,930	167,553	1,970,182
		*Ours	0.017	0.009	0.021	391,555	113,889	515,343

*dynamic method that only updates the affected parts of the map in each frame

different amount of computation per visit.

The comparison of the GVD update algorithms shows similar results: the dynamic methods clearly outperform the static method EVG-Thin in all tested environments. In addition, our approach can reduce the runtime considerably compared to the previous dynamic approach by Kalra *et al.* [2009], since this method uses the same update strategy as Scherer's approach for DMs and thus visits more cells. In all tested environments, the average frame rate achieved by our approaches was well above 20 fps which allows for online application of both distance maps and GVDs.

The distance maps generated by our method are equal to the ones generated by the compared methods, up to the inherent overestimation errors of 0.09 pixel units, as discussed in Chapter 3. Exemplary outputs of our 4-connected GVD algorithm and the method by Kalra *et al.* are shown in Figure 3.1: while Kalra's GVD misses Voronoi lines inside rooms and corridors, our approach captures the connectivity of the floor plan completely. Furthermore, our method generates thin

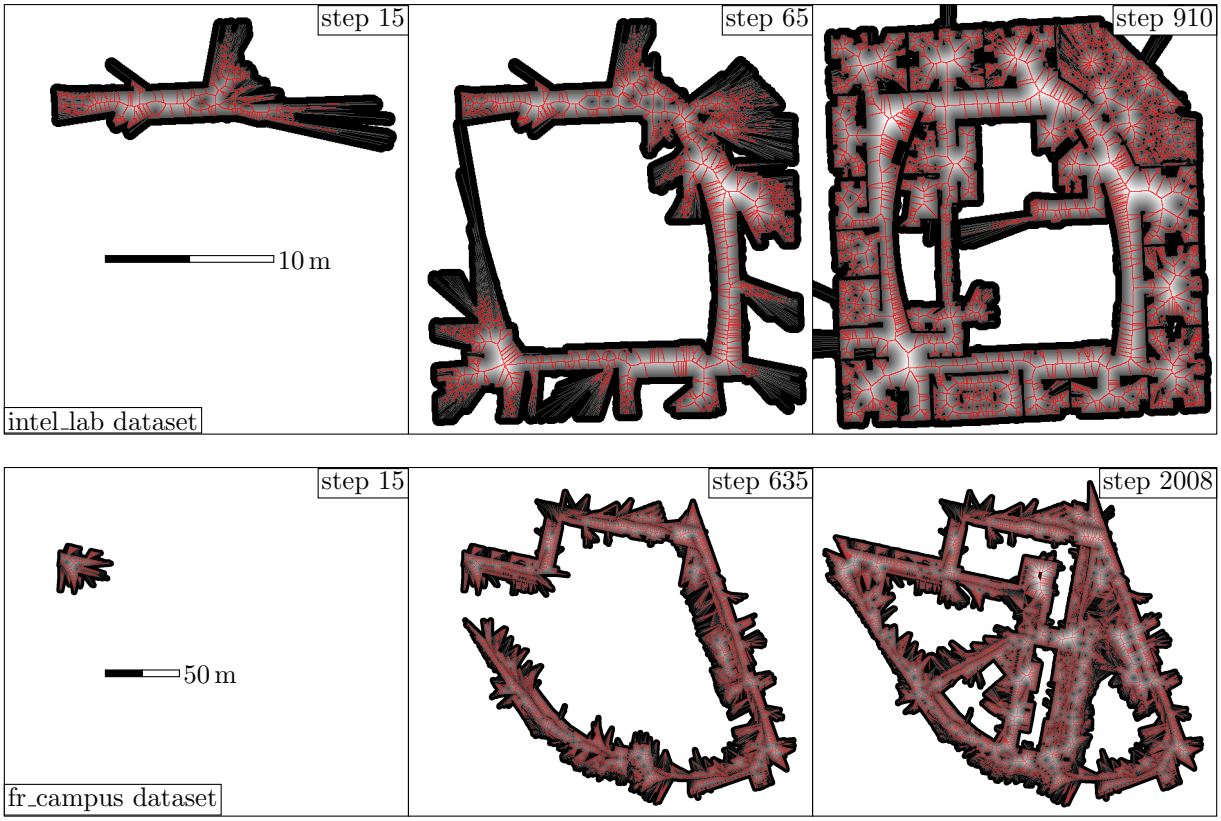


Figure 7.3: Incremental construction of a distance map and a Voronoi diagram during SLAM for two datasets. Pure local mapping only causes changes if new parts of the map are uncovered. Loop-closures, however, can affect large parts of the map, since individual parts often move with respect to the map coordinate system.

Voronoi lines, such that different paths between a given start and goal are topologically different with respect to obstacles.

7.2 2D DMs and GVDs during SLAM

To demonstrate the suitability of our incrementally updatable DMs and GVDs for SLAM applications, we used the GMapping SLAM package [Grisetti *et al.*, 2007; Stachniss *et al.*, 2012] to construct maps of the datasets "intel_lab" (indoor) and "fr_campus" (outdoor) [Lau *et al.*, 2012], with a resolution of 0.05 m and 0.2 m per cell, respectively. We set the parameters to integrate a new scan after the robot has moved 0.5 m or rotated 0.5 radians, and used 100 particles. After each integration step, we use the map associated with the current best particle to determine the newly occupied and freed cells with respect to the previous step.

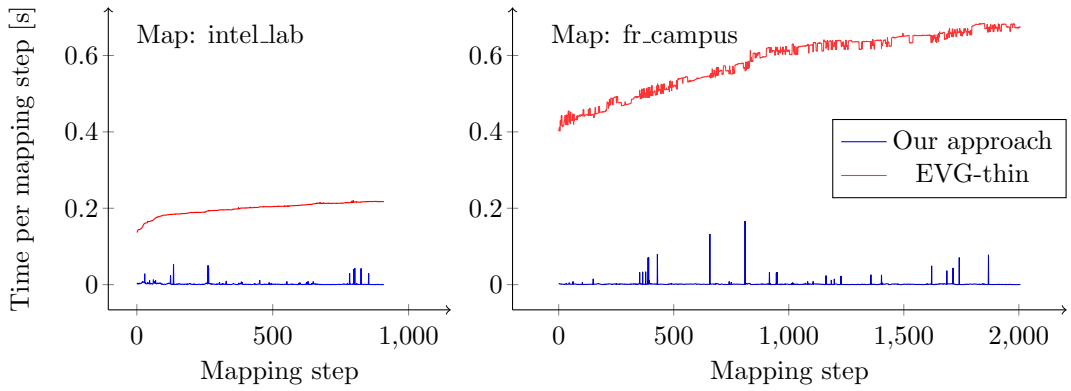


Figure 7.4: Computation time required to perform the updates between each mapping step for the datasets shown in Figure 7.3. Our incremental methods clearly outperform the computation of the full GVD in every step as done by EVG-thin. Peaks in the computation time of our method correspond to the loop closures in SLAM.

Based on these changes, we incrementally update a DM and GVD in every step as shown in Figure 7.3. The computation time taken by our algorithm to perform these updates is shown in Figure 7.4. For comparison, we also plot the computation required by EVG-thin to compute a GVD approximation for the grid map of each step.

Since the maps are incrementally growing during the mapping process, the computational effort for EVG-thin grows with increasing step numbers. The dynamic updates of our algorithm are substantially faster, since the number of changes between consecutive SLAM steps is rather small. The curve shows peaks of increased effort whenever the best particle changes, for example after closing a loop. In our experiments, the average computation time required to update the DM and GVD per mapping step was below 0.002 s and never exceeded 0.2 s even after loop-closures, which allows frame rates above 5 fps.

7.3 Three-dimensional DMs

For the experiments with three-dimensional DMs we recorded 3D maps in a lab room and in a large hall using the depth measurements obtained from a Microsoft Kinect 3D camera (see Figure 7.5). The camera poses were determined using a MotionAnalysis motion capture system with 9 Raptor-E cameras.

The maps were constructed and stored with a resolution of 0.05 m per voxel using the probabilistic occupancy mapping routines in the OctoMap software package [Wurm *et al.*, 2010]. The map sizes and total number of cells are given in the left column of Table 7.2.

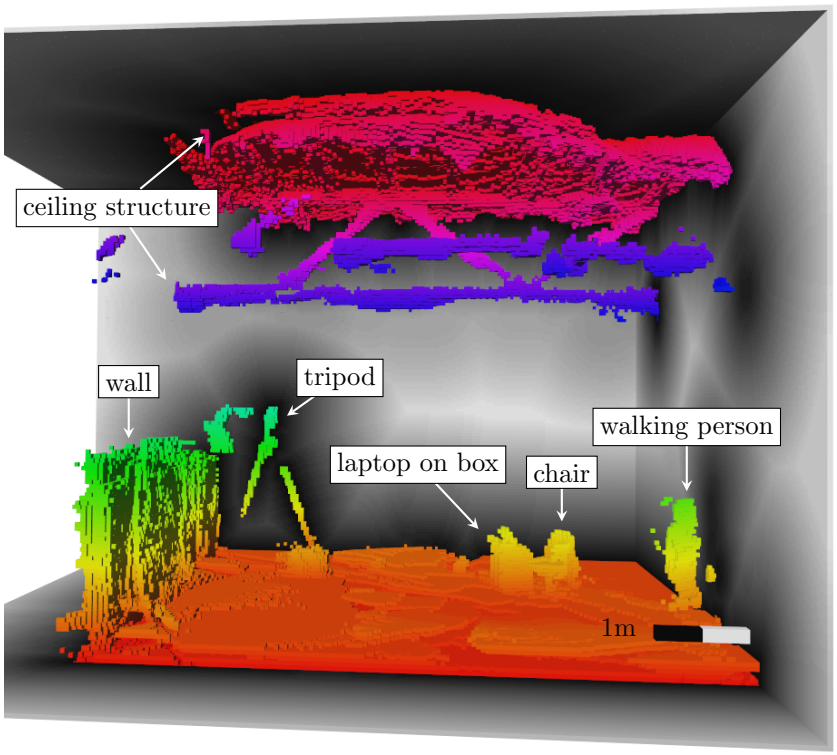


Figure 7.5: Excerpt of the 3D map of the hall used for our experiments, showing roughly 1/4 of the space. The color encodes the height of each cell over ground. The gray planes are slices from the 3D distance map of that space. Here, brighter cells denote increasing Euclidean distance from obstacles.

Table 7.2: Performance of incremental 3D distance map updates

Data set	$d_{\max}$	Avg. cell visits	Avg. update time
Hall, moving: 693 frames	0.5 m	152,613	0.0734 s
size: 14.25×14.2×7.45 m^3	1.0 m	267,097	0.1493 s
total cells 12,226,500	1.5 m	354,061	0.2174 s
Avg. flipped 347.0	2.0 m	396,937	0.2532 s
Hall, static: 1,443 frames	0.5 m	93,460	0.0482 s
size: 14.25×14.2×7.45 m^3	1.0 m	180,178	0.1034 s
total cells 12,226,500	1.5 m	245,062	0.1479 s
Avg. flipped 214.8	2.0 m	285,867	0.1782 s
Lab, static: 1,618 frames	0.5 m	41,760	0.0231 s
size: 6.0×5.15×3.15 m^3	1.0 m	48,231	0.0269 s
total cells 798,720	1.5 m	48,231	0.0275 s
Avg. flipped 164.7	2.0 m	48,231	0.0275 s

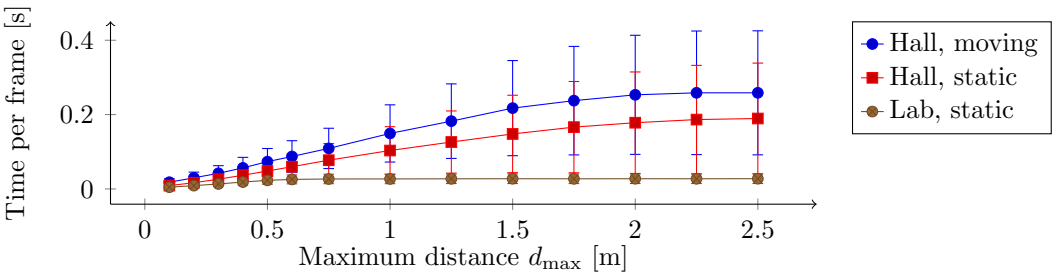


Figure 7.6: Performance of incremental 3D distance map updates. The plot shows the average computation time per frame along with the standard deviations for different sequences, depending on the maximum propagated distance $d_{\max}$.

After constructing the maps, we recorded three 3D sequences of two walking people people and continuously updated the pre-recorded maps using the same techniques. When integrating a point cloud into the 3D map, the OctoMap library can return a list of the cells that switched from free to occupied or vice versa. In each frame, we register any flipped cell s using $\mathbf{setObstacle}(s)$ or $\mathbf{removeObstacle}(s)$, and then update the 3D distance map as described in Chapter 4.

The number of frames per sequence, the average number of flipped cells per frame, and the resulting average computation time per frame are given in the right column of Table 7.2. In one sequence, the camera was manually moved to follow a person, in the others it was static. An excerpt of the 3D map of the hall and 2D slices of the corresponding 3D distance maps are shown for an example frame in Figure 7.5.

The average computation time per frame, depending on the maximum propagated distance $d_{\max}$ is shown in Table 7.2 and the plot in Figure 7.6. The error bars show the standard deviation obtained by averaging over the frames. These values are high, since the computational requirements depend on the amount of changes in the environment, which varies over time.

The average computation time per frame grows with an increasing distance limit $d_{\max}$, up to the point where the update radius is naturally limited by the maximum distances found in the environment.

For applications like path planning, collision avoidance or localization, a distance limit $d_{\max}$ between 0.5 m and 1.0 m is mostly sufficient. The maximum computation times per frame under these conditions were achieved in the large hall and corresponded to frame rates between 2 and 5 fps, with average frame rates between 4 and 13 fps. For the smaller lab environment, the update rates do not fall below 10 fps. Note that these numbers consider the computation time required by our algorithms, but not the occupancy mapping by the OctoMap.

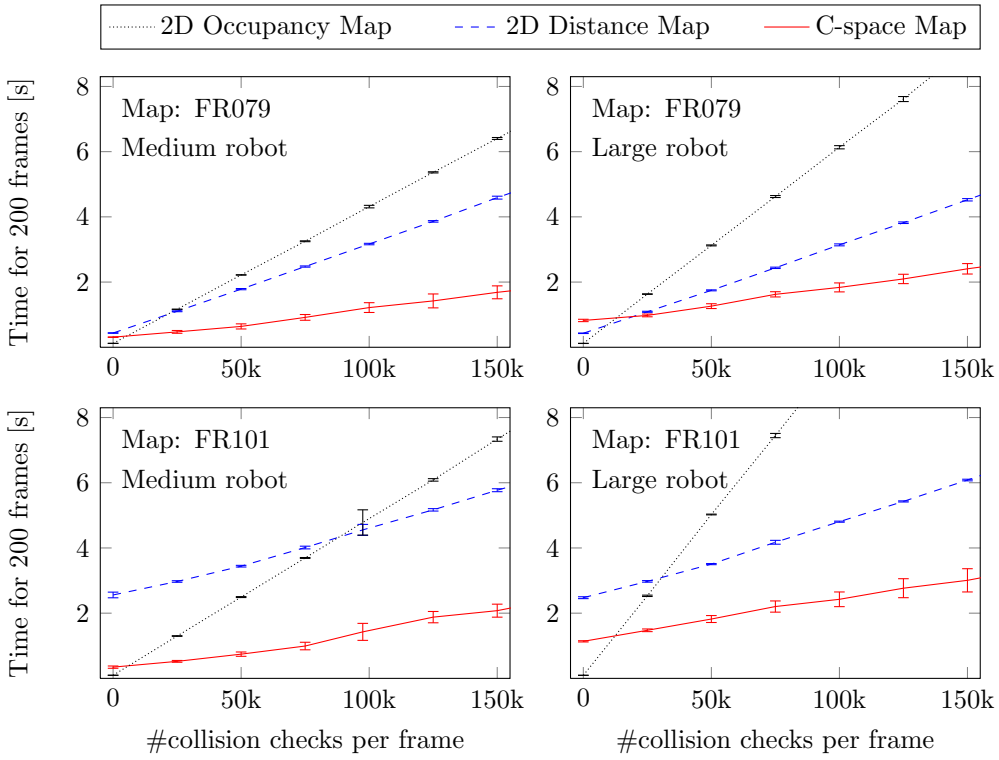


Figure 7.7: Computation time for different collision checking routines for two sequences and two robot models. The update required in every frame for the c-space collision map pays off starting from 10,000 collision checks per frame. The plots show mean and standard deviations averaged over 10 runs.

7.4 C-Space Obstacle Maps and Collision Checks

This section benchmarks our incrementally updatable c-space representations on the 2D laser data sequences described in Section 7.1. For the updates, the maximum range of the laser scanner was limited to 5 m. To simulate 2.5D and 3D obstacles, we augmented the laser data with random height values between 0 m and the robot height.

In 2D, we assumed a medium sized rectangular robot with a footprint of 0.85 m x 0.45 m, and a large one with 1.75 m x 0.85 m. In 2.5D, we modeled a wheelchair with a low front and a high rear part, as in Figure 2.1b. In 3D, the robot was modeled like a Willow Garage PR2, with a frontal extension for the base and the fixed arms (see Figure 2.1c). To speed up our algorithms, we used OpenMP for parallelization with up to 6 threads.

The c-space collision map presented in Chapter 6 requires an incremental update in every time step, but then, each collision check for the whole robot only needs a single map lookup. In the 2D model, we exploit the symmetry of the rectangular robot as described in Section 6.4.

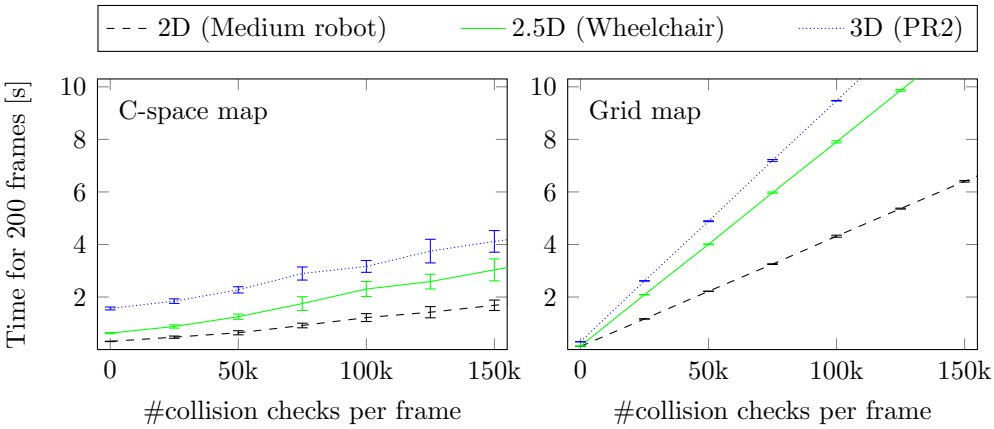


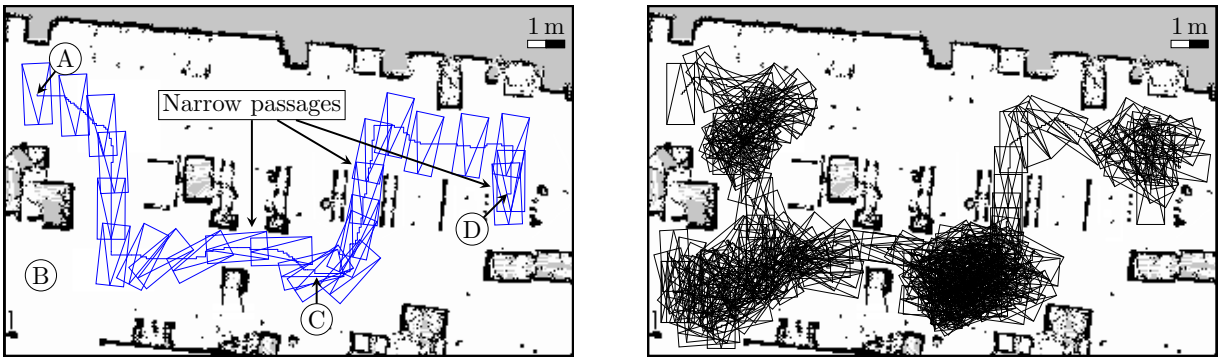
Figure 7.8: Collision check performance for different robot and obstacle models, using our updatable c-space collision map (left) vs. the straightforward occupancy grid map approach (right). The costs for updating the c-space map are remedied by the faster collision checks for 10,000 checks or more per frame. The plots show mean and standard deviations averaged over 20 runs.

We compare our method to a previous collision checking approach for rectangular robots that uses recursive distance queries on incrementally updatable 2D distance maps [Sprunk *et al.*, 2011]. As a baseline, we also evaluate a straightforward approach that checks every cell of the robot's footprint for collision using an up-to-date 2D occupancy map.

The results of this benchmark are shown in Figure 7.7. The time required for updating the distance and c-space maps is shown by the first data point of each plot (zero collision checks). The slopes of the curves depend on the cost per collision check. In contrast to the distance map approach, the update time for the c-space map grows with the size of the robot (right vs. left column), but does not suffer from the open area in FR101 (bottom vs. top row). The update for the c-space collision map pays off for 10,000 or more collision checks, which can easily be required during path planning or trajectory optimization. In comparison, the break-even point for a single disc-shaped object was at 22,400 for the disc-decomposition method by Ziegler and Stiller, and $5 \cdot 10^6$ for the full c-space [2010].

We repeat the experiment, but with 2.5D and 3D obstacles and robots this time. Compared to the 2D rectangular robot (dashed), the costs for the c-space update with 2.5D and 3D are higher, since the robots are not symmetric anymore and consist of two and three parts, respectively, see Figure 7.8 (left). However, the costs per collision check (slope of the plots) are the same, as opposed to the curves for the straightforward occupancy map approach (right).

In all cases, the update of the c-space map takes less than 15 ms per frame. Performing 150,000 collision checks per frame additionally requires at most another 15 ms. This corresponds to



(a) C-space Voronoi (b) KPiece

Figure 7.9: Map of a factory floor (9.5x15.4 m) with start location (A) and three goals (B), (C), and (D). Example paths from (A) to (D) are shown for two different planners. The sampling-based planner (right) is challenged by narrow passages, while the performance of the Voronoi planner (left) is unaffected.

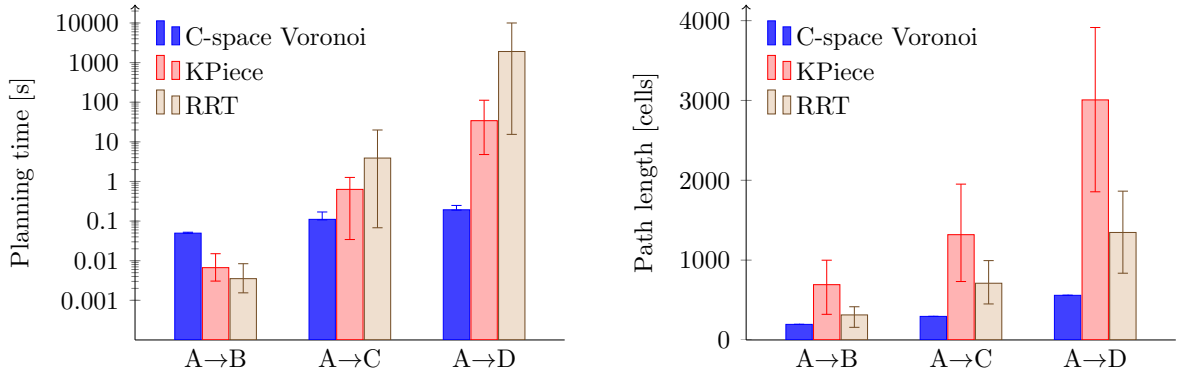


Figure 7.10: Planning time and path length for three planners and the three planning tasks in Figure 7.9. The plot shows mean and min/max for 20 runs. In contrast to the Voronoi planner, the sampling-based planners require several orders of magnitude more planning time for each narrow passage in the path.

$10 \cdot 10^6$ collision checks per second for arbitrary robot shapes, which clearly outperforms even modern GPU-based approaches with $0.5 \cdot 10^6$ collision checks per second for simple polygons [Pan and Manocha, 2011].

7.5 Path Planning using C-Space Voronoi Maps

The c-space Voronoi maps presented in this thesis provide means for complete grid map planning for non-circular omnidirectional robots using standard graph search algorithms like A* or

D* Lite [Koenig and Likhachev, 2002]. With our algorithms for incremental updates they are applicable in dynamic environments. This experiment uses the Voronoi bubble technique proposed in Section 5.3.

We use A* to plan paths for the large robot model (see above) on the grid map of the factory floor shown in Figure 7.9. The start pose is given by (A), and three possible goal poses by (B), (C), and (D). Each of the consecutive goals requires traversing another narrow passage. For comparison, we test our method against the KPiece and RRT implementations available in the Open Motion Planning Library [Şucan *et al.*, 2010]. All planners use our c-space map for collision checking, thus the performance differences are due to the tested planner.

The average resulting planning times and path lengths for 20 runs per start-goal combination are shown in Figure 7.10. Each additional narrow passage requires several orders of magnitude more planning time for the sampling-based planners, while the time taken by the Voronoi planner grows roughly linearly with the path length. Using per-cell collision checking rather than the c-space collision maps for the sampling based planners increases the computation times by a factor of 3.

As another application example, we plan the path of a PR2 robot using a c-space Voronoi map generated from real 3D point cloud data (see Figure 7.11). After a precomputation phase of 0.5 s, planning a path on the incrementally updatable c-space Voronoi map takes less than 2.5 ms.

Clearly, Voronoi planning is of advantage in narrow areas as long as the grid resolution is fine enough. Our incrementally updatable c-space Voronoi representation allows to apply this idea to non-circular robots in dynamic environments, and could also be used in Voronoi sampling routines of other path planners [Zhang and Manocha, 2008].

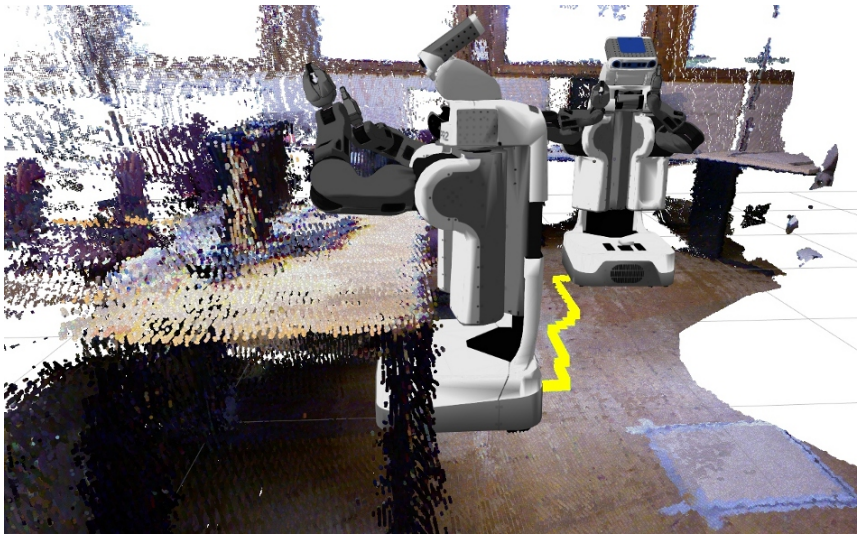


Figure 7.11: Table docking with a PR2 robot in a 3D map using Voronoi planning. The yellow line shows the planned path, the rendered robots denote the start and goal pose. Note that the goal pose requires 3D collision checking, since the table overlaps with the robot's footprint.

8 Conclusion

We presented incremental algorithms to update distance maps, Voronoi diagrams, and configuration-space maps. These representations can be initialized with a grid map or point cloud. For efficient online operation, our methods only update cells that are affected by changes in the environment. Thus, they can be used in real-world scenarios with unexpected or moving obstacles for applications like SLAM, path planning, collision avoidance, or localization.

Compared to previous approaches, our methods for updating two-dimensional Euclidean distance maps and Voronoi diagrams require about 60-70% less cell updates and computation time. At the same time they provide equal or more accurate results without any drawbacks. Our Voronoi approach is easy to implement and better handles non-convex obstacle compounds like indoor areas. With modifications that limit the maximum propagated distances and reduce the neighborhood size during propagation, we can also update three-dimensional distances maps at a speed that is suitable for online applications.

For the three-dimensional configuration space of non-circular robots we also presented methods to incrementally update collision maps, distance maps, and Voronoi diagrams. They can consider different obstacle representations, namely a robot moving on a plane with overhanging obstacles, or vice versa, obstacles elevating from the ground, and a robot with overhanging parts. The approaches are also applicable to 2D and full 3D obstacle representations and can exploit symmetries in the robot shape.

For Voronoi-based path planning in dynamic environments, we proposed a method to avoid abrupt changes in the planned paths for small variations of the start location. Using the configuration space Voronoi diagrams, this method is also applicable to non-circular holonomic robots. Especially in narrow passages, our method clearly outperforms the compared RRT and KPiece approaches in terms of computation time and length of generated paths.

Our algorithms have been implemented and tested on real-world data sets. The achieved minimum frame rates for updates, collision checks, and path planning range between 5 and 20 fps in 2D environments, and between 2 and 5 fps in 3D environments. If required, the frame rates can be further increased by limiting the propagated distances. The source code of all our algorithms is available online [Lau *et al.*, 2012]. The 3D distance map is also available as part of the OctoMap software package [Wurm and Hornung, 2012].

Part II

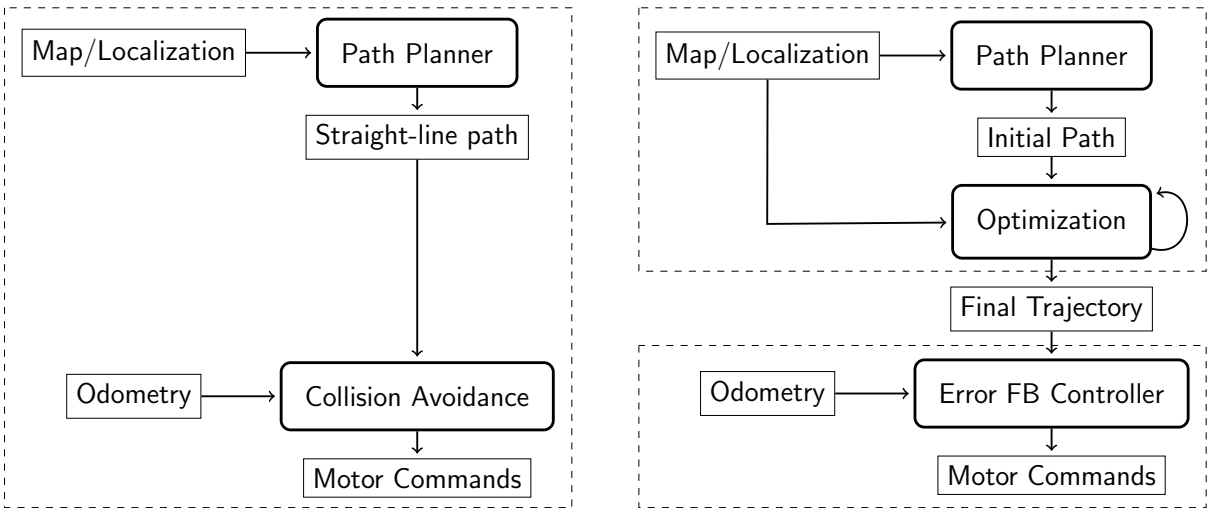
Kinodynamic Motion Planning

9 Introduction

Motion planning is a fundamental task for wheeled mobile robots. It consists of planning a collision-free path from the position of the robot to a given goal location using a representation of the environment, and computing motion commands that make the robot platform follow this path. To determine such a path, existing motion planning systems often use a global path planner like A* or its descendants, e.g., [Thrun and Bücken, 1998; Koenig and Likhachev, 2002; Likhachev *et al.*, 2005], which find the shortest path on a 2D grid or graph that represents the traversable space. These paths can be optimal in the sense that they represent the shortest path from start to goal in the given discrete representation. However, they typically contain sharp corners and can only be accurately followed by stopping and turning on the spot. This is not the desired behavior, since it significantly increases the time of travel, which is an important cost criterion as well.

In practice, the generation of actual motor commands is therefore often carried out by reactive systems that run in the same control loop as the global path planner (see Figure 9.1a). Typically, these systems use collision avoidance approaches like potential fields [Khatib, 1986], vector field histograms [Borenstein and Koren, 1991] or nearness diagrams [Minguez and Montano, 2004]. They consider the vector to the next one or two waypoints in the planned path and the distance to obstacles perceived by the sensors of the robot. The Dynamic Window Approach by Fox *et al.* [1997] additionally considers the platform's kinodynamic constraints to constrain and smoothen velocity changes. All of these approaches have in common that they smooth the planned path, which reduces the time required to reach the goal.

The downside of this solution is, that (a) optimality properties of the straight line path do not apply to the resulting continuous trajectory and no time of travel optimality is achieved, (b) the shape of the path, e.g., how much corners are cut, depends on global parameter settings rather than optimization or search considering the environment, (c) velocities and accelerations are not planned in advance but are subject to reactive behavior, which prevents accurate motion prediction and makes the satisfaction of hard constraints difficult, and (d) no guarantees can be made for the control stability or convergence behavior of the system. Robots like autonomous cars, wheelchairs, autonomous transport vehicles, or other service platforms can be required to carry heavy or sensitive payload, execute precise motion, or show predictive behavior, and therefore demand solutions to the above-mentioned problems.



(a) Path planner with collision avoidance

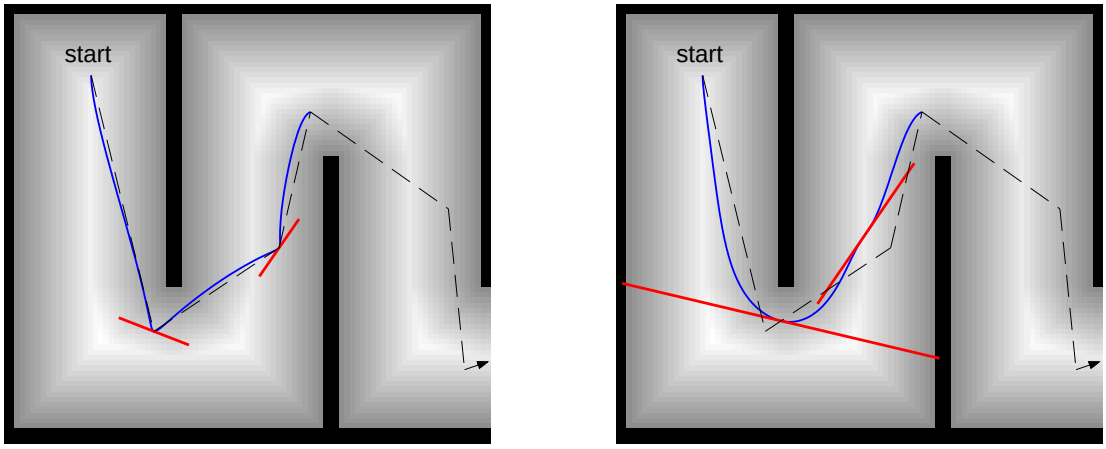
(b) Proposed optimization-based approach

Figure 9.1: System architecture of traditional motion planning systems (a) and the proposed approach (b). The dashed rectangles indicate independent control loops.

If the desired trajectory of a robot is specified as a function of time, an error-feedback controller can steer the robot along this trajectory, as long as the trajectory obeys the hardware constraints of the mobile platform and the controller is executed at a sufficient rate [Siciliano *et al.*, 2009]. The key challenge is to find a suitable trajectory representation and a method to generate a feasible trajectory for a given start and goal location and a representation of obstacles in the environment.

This part of the thesis discusses different trajectory representations for mobile robots in dynamic environments and proposes a novel path model. This model can specify smooth and extendable trajectories with a smaller number of parameters compared to standard approaches. Based on this path model, it further presents a method to generate and optimize robot trajectories that minimize the required time of travel. The proposed system is designed for robots with differential or synchro drive, but could be adapted to other drive types as well.

As shown in Figure 9.1b, our system generates an initial path from sparse waypoints obtained from a global path planner. This path is augmented with a velocity profile to specify the pose of the robot over time. Executing this trajectory requires the robot to perform turns on the spot as shown in Figure 9.2a, but the trajectory already complies with the kinodynamic constraints of the robot hardware and is free of collisions with the environment. This curve is iteratively optimized to yield a smooth and time-optimal trajectory as shown in Figure 9.2b. An error-feedback controller is used to steer the robot along the trajectory. In this way, our approach



(a) Initial trajectory (b) Optimized trajectory

Figure 9.2: Initial and optimized spline trajectories created for the first four waypoints of a piecewise linear path (dashed). The optimization adjusts the location of waypoints and length of their tangents (red lines). Trajectories are shown in blue on the distance map, where darker values are closer to obstacles.

addresses all the problems mentioned above.

The remainder of this part is organized as follows. After a discussion of related work, Chapter 11 discusses the suitability of different types of parametric curves for motion planning, and Chapter 12 introduces a novel path model based on quintic Bézier splines. Chapter 13 proposes a method to compute the maximum velocities for each point of a given robot path, which implicitly defines the pose of the robot for every point in time. Chapter 14 describes the means for generation, optimization and execution of such trajectories in a real-world motion planning system. In Chapter 15 we evaluate our system in experiments on real robots, while Chapter 16 presents extensions and applications of our approach by other authors.

10 Related Work

The problem of *time-optimal kinodynamic motion planning* was first posed by Donald *et al.* as to determine the motor commands that guide a robot from start to goal on a collision-free path in minimal time while respecting kinodynamic constraints [1993]. Stachniss and Burgard included discretized translational and rotational velocities into a five-dimensional A* search, which returns a series of velocity commands to be issued in discrete time intervals [Stachniss and Burgard, 2002]. Deviations from the trajectory are handled by frequently executing the global path planner, which generate s a new trajectory for the current position.

Planned trajectories can be precisely tracked using an error-feedback controller, which runs in a control loop that does not alter the trajectory. As input, a trajectory representation is desired that continuously defines position, velocities, and accelerations over time. In the past, several types of parametric curves have been used for this purpose. Some authors have used cubic Bézier splines to this end [Mandel and Frese, 2007; Sahraei *et al.*, 2007]. Howard and Kelly proposed spline interpolation of velocity commands to generate such trajectories [2007]. Likhachev and Ferguson perform a search on a discrete set of action primitives to connect discrete nodes of a state lattice and thus obtain a smooth trajectory for an autonomous car [Likhachev and Ferguson, 2008]. So far, all of these approaches have in common that the generated curves represent continuous position changes, but are not continuous in curvature, which induces abrupt rotational accelerations. Likhachev and Ferguson acknowledged this problem and alleviate it by limiting the car's velocity to 2 m/s at high curvatures.

Gulati and Kuipers emphasize the need for smooth and comfortable motion when transporting people, and propose the use of cubic B-Splines [2008], as also done by Shiller and Gwo [1991]. While cubic B-splines can indeed be curvature continuous from start to end, it is not possible to directly influence the curvature at the control points. Therefore it is not practically feasible to attach or replace a segment of an existing trajectory with curvature continuity at an arbitrary point of the spline. This is required if the trajectory has to be extended or changed during execution, for example in the presence of unmapped obstacles which are not accounted for in these approaches [Gulati and Kuipers, 2008; Shiller and Gwo, 1991]. The same applies for the clothoid-based approaches that require zero curvature at start and end for the construction of the clothoid spline [Girbés *et al.*, 2011; Wilde, 2009; Henrie and Wilde, 2012]. To overcome this

problem, we propose to use *quintic Bézier splines*, which are curvature continuous and additionally allow for attaching segments without curvature discontinuities. In this way, trajectories can be extended or changed on the fly, e.g., in order to react to sudden changes.

In contrast to approaches that perform a search on a discretized action set, e.g., [Likhachev and Ferguson, 2008; Macek *et al.*, 2008; Ziegler *et al.*, 2008], Connors and Elkaim propose to perform optimization of parameterized trajectories [2007]. Their system is initialized with straight-line segments that can pass through obstacles. Aiming for a smooth and collision-free trajectory, the system iteratively moves control points without a guarantee to find a solution if it exists. Our approach also employs optimization to obtain fast and smooth motion, but starts from an initial straight-line path that is collision-free. Thus, it has any-time characteristics, i.e., it can be interrupted at any point in time to retrieve a collision-free solution.

Most of the existing approaches determine just the shape of the planned trajectory by search or optimization, but rely on local heuristics to set the translational velocities [Sahraei *et al.*, 2007; Likhachev and Ferguson, 2008; Gulati and Kuipers, 2008; Ziegler *et al.*, 2008; Connors and Elkaim, 2007]. However, in order to determine truly time-optimal trajectories, the admissible velocities governed by constraints on velocities and accelerations have to be considered. In the work by Macek *et al.* [2008] and Stachniss and Burgard [2002], the velocity is part of the search, but the systems are limited to a discretized action space. This also applies to more recent work in sampling-based motion planning. Jaillet *et al.* for example present fast planning times with their environment-guided RRT planner, but restrict their action set to only five different accelerations and five different steering angles [Jaillet *et al.*, 2011]. Many sampling-based motion planners generate paths that would require additional post-processing for practical use, e.g., due to their suboptimal path length [Plaku *et al.*, 2010]. Karaman *et al.* proposed an anytime motion planning algorithm based on RRT* that refines generated paths while moving [2011]. However, it remains unclear if their system can gracefully evade unexpected obstacles, and how their action space is discretized.

Shiller and Gwo [1991] consider a curve that defines the shape of the trajectory, and generate a corresponding velocity profile that considers constraints on the translational velocity and acceleration for an Ackermann drive. Our system is similar to this approach, but adds the capability of curvature continuous replanning to update the trajectory while moving. During optimization, it considers constraints on translational and rotational velocities and accelerations. Furthermore, it regards the maximum allowed centripetal acceleration and a speed limit in the vicinity of obstacles. In this way, it generates space-time trajectories that are time-optimal and traversable.

11 Basic Path Representations

An essential characteristic of any motion planning system is the representation used to model robot paths. They can either be defined implicitly through the actions a robot takes, e.g., with a list of motor commands, or directly as desired poses in world or map coordinates. To specify robot poses as a function of time, parametric curves are appealing.

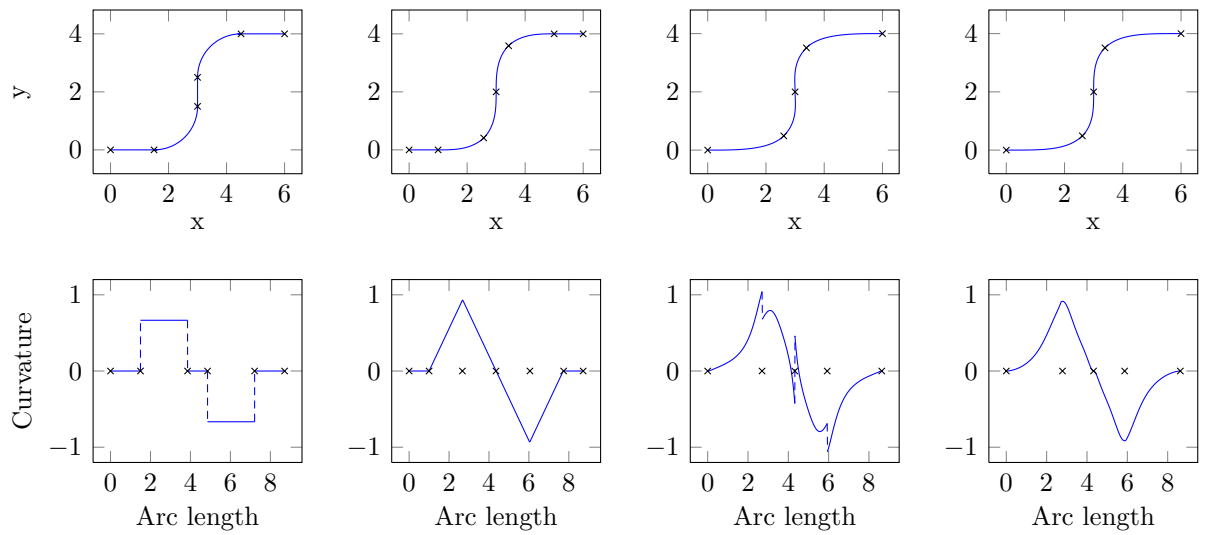
This chapter presents several parametric path representations based on different mathematical functions. Considering properties like smoothness, feasibility for given kinodynamic constraints, the number of model parameters, and the effort required to position them in space, we discuss their applicability to motion planning.

In the context of this work, a robot *path* specifies a series of two-dimensional robot positions in map or odometry coordinates without a relation to time. The robot orientation is implicitly defined and given by the direction of the path tangent at any point on the path. We use the term *trajectory* to refer to the robot pose as a function of time, which implicitly specifies the corresponding velocities. Paths can be represented in different mathematical ways. The most common path models use concatenations of linear path segments, circular arcs, clothoids, or polynomial curves, which are discussed in the following.

11.1 Linear Path Segments and Circular Arcs

Piecewise linear paths are probably the simplest kind of path representation, since they only consist of linear segments that connect a set of consecutive waypoints. The allowed angles between the segments can be restricted, e.g., to multiples of 45 or 90 degrees when resembling movements on a classic A* grid search. Arbitrary polygonal paths are also common, e.g., when pruning paths with restricted angles using sight-lines for given obstacles, or when using interpolating path planners like the Field D* algorithm by Ferguson and Stentz [2007].

While piecewise linear paths can efficiently be planned using grid search approaches, their direct application in real-world systems is problematic: due to inertia and finite accelerations, a robot would have to stop at each waypoint and turn on the spot to precisely follow the discontinuous direction of the path. As discussed in Section 9, these paths are therefore often smoothed using



(a) Circular arcs (b) Clothoids (c) Cubic Bézier (d) Quintic Bézier

Figure 11.1: Manually constructed paths using segments of different curve types. The top row shows the path shapes in Cartesian space, and the bottom row the curvature of each path as a function of arc length. Although all paths look smooth, their curvature can be discontinuous (dashed). The crosses mark segment borders.

reactive collision avoidance systems like the Dynamic Window approach by Fox *et al.* [1997], which deliberately deviate from the planned path.

Most wheeled mobile robots move on circular arcs as long as their motor commands remain constant. This is not only the case for car-like vehicles or tricycles with one or more steered wheels, but also for robots with differential or synchro drive controlled by translational and rotational velocity commands. Given that motor commands are typically updated in discrete time intervals, a straightforward path representation is the concatenation of circular arcs. This also includes straight lines, which can be seen as circular arcs with infinite curve radius. The resulting paths are continuous and have a piecewise constant curvature. At the junction points of circular arcs with different radii, however, a curvature discontinuity occurs as shown in Figure 11.1a. Unless the vehicle stops at such points, these paths can also not be tracked precisely due to finite accelerations and steering rates.

11.2 Clothoid Paths

A set of concatenated clothoids, also called “clothoid spline”, are similar to concatenated circular arcs, but with linear curvature changes instead of piecewise constant curvature.

Mathematically, a clothoid path segment is a section of the two-dimensional parametric curve known as “Cornu spiral” or “Euler’s spiral”. This curve is defined as a function of an internal parameter $u \in \mathbb{R}$, and given by

$$x(u) \;=\; \alpha\sqrt{\pi}\, S\Big(\tfrac{u}{\sqrt{\pi}}\Big) \;=\; \alpha\sqrt{\pi} \int_0^u \sin \tfrac{1}{2} v^2 \,\mathrm{d}v \tag{11.1}$$

$$y(u) \;=\; \alpha\sqrt{\pi}\, C\Big(\tfrac{u}{\sqrt{\pi}}\Big) \;=\; \alpha\sqrt{\pi} \int_0^u \cos \tfrac{1}{2} v^2 \,\mathrm{d}v \tag{11.2}$$

where $\alpha \in \mathbb{R}$ scales the spiral, and thus also affects the arc length $s(u) = u{\cdot}|\alpha|$ and the curvature of the curve, $c(u) = u/\alpha$. The Fresnel integrals $S(u) \in \mathbb{R}$ and $C(u) \in \mathbb{R}$ cannot be evaluated in closed form, but approximated with high accuracy [Heald, 1985; 1986].

Given the linearity of arc length and curvature as functions of the internal parameter u, the curvature expressed as a function of arc length is also linear (see Figure 11.1b) and given by $c(s) = s/(\alpha \cdot |\alpha|)$. Thus, the curvature also changes linear with time if a vehicle drives with constant translational velocity, e.g., due to a speed limit. This motivates the use of clothoids for constructing highways [Galin *et al.*, 2010] and for motion planning on car-like robots [Henrie and Wilde, 2012; Walton and Meek, 2005].

This assumption, however, does typically not hold for service robots moving in buildings or populated environments. For example, they often have to reduce their translational velocity in curves due to limited rotational velocities or an upper bound on centrifugal accelerations to ensure stability. Assume a robot that brakes and accelerates from an initial velocity v_0 with constant acceleration a on a clothoid, respectively. The curvature as a function of time t is then given by $c(t) = \left(\frac{1}{2}at^2 + v_0 t\right) / (\alpha \cdot |\alpha|)$. This implies quadratic rotational accelerations or steering rates, which makes clothoids less appealing for these applications.

Due to the linear dependencies between the internal parameter, arc length, and the curvature of clothoid paths, it is rather straightforward to compute the motor commands required to steer a robot along such paths. However, the positioning of clothoid paths in Cartesian space, e.g., to connect a sequence of waypoints, is rather complicated. Walton and Meek presented an approach to create a path of clothoid segments for given control points $\mathbf{P}_i$, $i \in 0, \ldots, N$ that implicitly specify the waypoints interpolated by the spline [Walton and Meek, 2005].

Such a path consists of $N-1$ parts that each connect a pair of waypoints as in Figure 11.2. The waypoint $\mathbf{W}_0 = \mathbf{P}_0$ at the start and $\mathbf{W}_{N-1} = \mathbf{P}_N$ at the end are given by the corresponding control

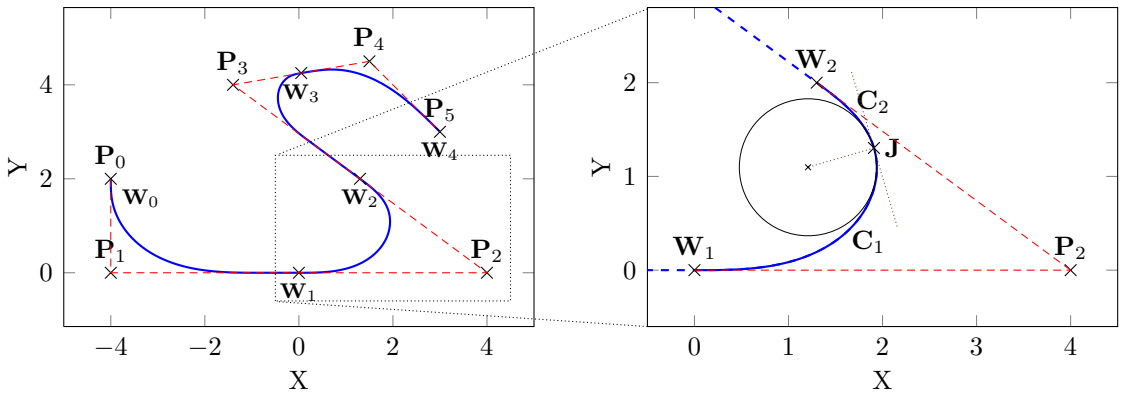


Figure 11.2: Clothoid spline for a control polygon $\mathbf{P}_0 \dots \mathbf{P}_5$ (left). The intermediate waypoints $\mathbf{W}_1 \dots \mathbf{W}_3$ are obtained by bisecting the polygon segments. Each spline part is constructed as in the magnified example (right): a pair of waypoints $\mathbf{W}_1$, $\mathbf{W}_2$ is connected by two clothoids that start with zero curvature at $\mathbf{W}_1$ and $\mathbf{W}_2$, pointing at the enclosed control point $\mathbf{P}_2$. They join at $\mathbf{J}$ with equal curvature, as indicated by the circle.

points. The inner waypoints $\mathbf{W}_i$ with $i = 1, \dots, N-2$ are not specified directly, but obtained by bisecting the straight line segments connecting the control points, $\mathbf{W}_i = \frac{1}{2}(\mathbf{P}_i + \mathbf{P}_{i+1})$.

Each part of the spline is specified by a control point $\mathbf{P}_i$ and the adjacent waypoints $\mathbf{W}_i$,$\mathbf{W}_{i+1}$ as shown in Figure 11.2 (right). A clothoid $\mathbf{C}_1$ is inserted at $\mathbf{W}_1$ with its tangent pointing towards $\mathbf{P}_2$. It starts with zero curvature at the waypoint and bends towards the other waypoint $\mathbf{W}_2$. Similarly, a second clothoid is inserted at $\mathbf{W}_2$ bending towards $\mathbf{W}_1$. The individual scaling parameters α of both clothoids are determined such that the curves join at a point $\mathbf{J}$ with equal curvature (see figure). In addition to this construction method, Walton and Meek propose to extend one of the clothoids with a straight line if the length of the straight line segments between the waypoints and $\mathbf{P}_2$ is substantially different. Additionally, they describe how to insert a circular arc between the joining clothoids to reduce the maximum curvature in such a part.

To apply this approach to motion planning, one can for example generate a curvature continuous clothoid path to smoothen a piecewise linear path obtained by A* planning. In practice, several problems occur with this approach:

1. Since the curve cuts the corners of the initial plan, it can collide with obstacles in the environment. Moving the control points can potentially restore the collision-freeness, but a solution cannot be guaranteed.

2. This path model implies zero curvature at the implicitly defined waypoints between each pair of control points. This prevents the connection of two waypoints with a single curve,

if their tangents are not pointing towards a common curve point.

3. The curvature at the very start of the path has to be zero as well. Thus, this path model does not support planning of a new path that joins an old one in a curve, e.g., to react to an unexpected obstacle while the robot moves.

To our knowledge, no approach has been developed that is able to connect arbitrary waypoints with clothoid segments that overcomes these limitations. Practical applications still make use of this curve type, but always in scenarios that do not require generic placement of clothoids, for example to model lane changes with autonomous cars between zero-curvature poses [Wilde, 2009], or as motion primitives in pre-computed action sets [Likhachev and Ferguson, 2008].

11.3 Polynomial Splines

Polynomial splines are parametric curves consisting of piecewise polynomial functions, usually of the same degree $d \in \mathbb{N}$. Cubic splines are widely used in computer graphics, e.g., to model smooth shapes and surfaces. As discussed in Chapter 10, they have been applied to motion planning as well. Each segment of these splines is a two-dimensional polynomial of degree $d = 3$ defined over an internal parameter u,

$$\begin{pmatrix} x \\ y \end{pmatrix} = \mathbf{a} + \mathbf{b}u + \mathbf{c}u^2 + \mathbf{d}u^3,\ u \in [0, 1]\,. \tag{11.3}$$

The two-dimensional coefficients $\mathbf{a}$,$\mathbf{b}$,$\mathbf{c}$,$\mathbf{d} \in \mathbb{R}^2$ control the shape of the curve and are usually not directly specified. Instead, one uses a set of control points that correspond for example to waypoints and tangents, and thus have a more intuitive semantic meaning. Several types of spline models exist that differ in how their control points are mapped to the actual coefficients of the polynomials, and thus construct splines with different properties.

B-splines are linear combinations of a set of N control points $\mathbf{P}_i,\ i{=}0 \ldots N{-}1$ with $N \geq d{+}1$. The curve results form the variation of the weights in the linear combination, which are given by basis functions that depend on the internal parameter u. B-splines are $d{-}1$-times continuously differentiable, and thus have a continuous curvature for cubic or higher-order degrees, which is appealing for use as a robot path model. However, a B-spline does not pass through its control points, unless a control point is repeated at least d times at start and end.

Bézier splines and Hermite splines are also polynomial splines that consist of one or more polynomial curve segments. Both kinds are mathematically equivalent, since they only differ in the representation of tangents, which can be converted from one to the other. For the mathematical derivation of these curves, see Chapter 12.

In contrast to B-splines, Bézier curves pass through their waypoints, which is desirable for motion planning applications, since it simplifies the deformation of paths with respect to obstacles. When using the common cubic form, however, the curvature at the junction points of the individual segments is discontinuous as shown in Figure 11.1c, which is not the case for B-splines. Higher-order splines have enough degrees of freedom to support curvature continuity (see Figure 11.1d), but require additional constraints to achieve it. Our path model presented in the following chapter is based on quintic Bézier splines, and uses heuristics to control a part of the parameters.

Compared to clothoid splines, polynomial splines are much easier to handle in the spatial domain, since they can directly be defined by waypoints and tangents. On the other hand, the derivation of velocity commands for mobile robots following polynomial curves is mathematically more involved, since the relation between arc length s and internal parameter u is nonlinear. A common approach to this problem is arc length re-parametrization of the spline, which requires numerical integration in discrete intervals as described in Section 12.4.

12 Path Model with Heuristics

This chapter presents a novel path model based on quintic Bézier splines. The curves generated using our model are curvature continuous like cubic B-splines, and at the same time pass through their waypoints like cubic Bézier curves. Additionally, the curvature at the start and end point can be chosen freely, which facilitates the attachment of newly planned paths, e.g., to replan a trajectory after the robot perceives an unexpected obstacle during motion. By using heuristics to control a subset of the degrees of freedom, this representation models smooth paths with a small number of parameters and is therefore especially suitable for kinodynamic motion planning. It has first been published in [Lau *et al.*, 2009], and has been used and extended in several applications.

12.1 Quintic Bézier Curves

We denote a two-dimensional robot path by $\mathbf{Q}(u)$, which specifies the position $(x, y)^T$ of the robot on the ground in world coordinates as a function of an internal parameter u. If $\mathbf{Q}(u)$ is represented by a single quintic Bézier curve segment, it is parameterized by $u \in [0, 1]$ from start to end.

To define $\mathbf{Q}(u)$, we use the quintic Hermite form. Thus, the segment is directly specified by the desired value of the polynomial and its first and second derivative at the start ($u = 0$) and end ($u = 1$) of the segment. Using $\mathbf{q}_u^k$ to denote the desired value of the k-th derivative of $\mathbf{Q}(u)$ at u, the segment is then given by the linear combination

$$\mathbf{Q}(u) = \sum\nolimits_{k=0}^{2} h_0^k(u) \cdot \mathbf{q}_0^k + h_1^k(u) \cdot \mathbf{q}_1^k \,, \tag{12.1}$$

where $h_0^k(u)$ and $h_1^k(u)$ are polynomials called Hermite basis functions. They are obtained by solving Equation (12.1) for general quintic polynomials $\mathbf{Q}$, h_0^k, h_1^k and the conditions $\mathbf{Q}^{(k)}(0) = \mathbf{q}_0^k$, $\mathbf{Q}^{(k)}(1) = \mathbf{q}_1^k$ in a system of linear equations for $k \in \{0, 1, 2\}$. Factoring out the $\mathbf{q}_u^k$, the

basis functions for the quintic case compute as

$$\begin{aligned} h_0^0 &= -6u^5 + 15u^4 - 10u^3 + 1 \\ h_0^1 &= -3u^5 + 8u^4 - 6u^3 + u \\ h_0^2 &= -\tfrac{1}{2}u^5 + \tfrac{3}{2}u^4 - \tfrac{3}{2}u^3 + \tfrac{1}{2}u^2 \\ h_1^0 &= 6u^5 - 15u^4 + 10u^3 \\ h_1^1 &= -3u^5 + 7u^4 - 4u^3 \\ h_1^2 &= \tfrac{1}{2}u^5 - u^4 + \tfrac{1}{2}u^3 \end{aligned} \tag{12.2}$$

The basis functions and coefficients for cubic and septic polynomials can be determined in a similar way, a more general derivation is given in [Sprunk *et al.*, 2012]. Since $\mathbf{q}_0^k$ and $\mathbf{q}_1^k$ specify the desired position of the curve segment $\mathbf{Q}(u)$, we denote them with the waypoint symbols $\mathbf{W}_0$ and $\mathbf{W}_1$, respectively. The first derivative at $\mathbf{W}_0$, $\mathbf{W}_1$ which is specified by $\mathbf{q}_0^1$, $\mathbf{q}_1^1$ represents the tangents at the waypoints, and we denote them by $\mathbf{T}_0$ and $\mathbf{T}_1$, respectively. Similarly, the desired second derivatives $\mathbf{q}_0^2$, $\mathbf{q}_1^2$ are denoted by $\mathbf{A}_0$ and $\mathbf{A}_1$ (accelerations).

12.2 Bézier Splines with Continuous Curvature

Similar to the construction of basic cubic splines, we concatenate multiple quintic Bézier spline segments to a longer spline. Let $\mathbf{Q}(u)$ denote a robot path consisting of N_Q concatenated segments $\mathbf{Q}_i(u_i)$, with $i = 0, \ldots, N_Q - 1$. While a $u_i \in [0, 1]$ parameterizes the i-th segment from its start to end, the variable $u \in [0, N_Q]$ is the internal parameter of the whole curve and results from concatenating the u_i. Thus, the trajectory can be denoted by $\mathbf{Q}(u) = \mathbf{Q}_{\lfloor u \rfloor}(u - \lfloor u \rfloor)$ for $u < N_Q$, where $\lfloor u \rfloor$ is the truncation of u to the next smaller integer number. $\mathbf{Q}(N_Q)$ is the end point of the path, and given by the end of the last segment, $\mathbf{Q}(N_Q) = \mathbf{Q}_{N_Q - 1}(1)$.

To achieve a smooth concatenation of the segments, the waypoint $\mathbf{W}$, tangent $\mathbf{T}$ and acceleration $\mathbf{A}$ specifying the end of each segment with index i are set to be equal with the respective parameters at the start of the succeeding segment $i+1$. Thus, we use the index i to denote $\mathbf{W}_i$, $\mathbf{T}_i$, $\mathbf{A}_i$ as the parameters for the start of the i-th segment, which at the same time specify the end of the segment $i-1$. Finally, $\mathbf{W}_{N_Q}$, $\mathbf{T}_{N_Q}$, $\mathbf{A}_{N_Q}$ are the parameters for the end of the path $\mathbf{Q}(u)$. Since $\mathbf{W}_i$, $\mathbf{T}_i$, and $\mathbf{A}_i$ specify the location as well as the first and second derivative of the adjacent segments at the junction point, the path $\mathbf{Q}(u)$ and its derivatives are continuous at the

junction points. Since the segments are continuous as well due to their polynomial nature, the whole path $\mathbf{Q}(u)$ is continuous up to the second derivative.

When following the path $\mathbf{Q}(u)$ with a robot, the curvature $c(u)$ of $\mathbf{Q}(u)$ is also required to be continuous to achieve smooth and precise robot motion. The curvature is the local radius of the curve at u and can be formulated using the partial derivatives of $\mathbf{Q}(u)$ [Stewart, 2002],

$$c(u) = \frac{\mathbf{Q}'_x(u) \cdot \mathbf{Q}''_y(u) - \mathbf{Q}'_y(u) \cdot \mathbf{Q}''_x(u)}{\|\mathbf{Q}'(u)\|^3} \,. \tag{12.3}$$

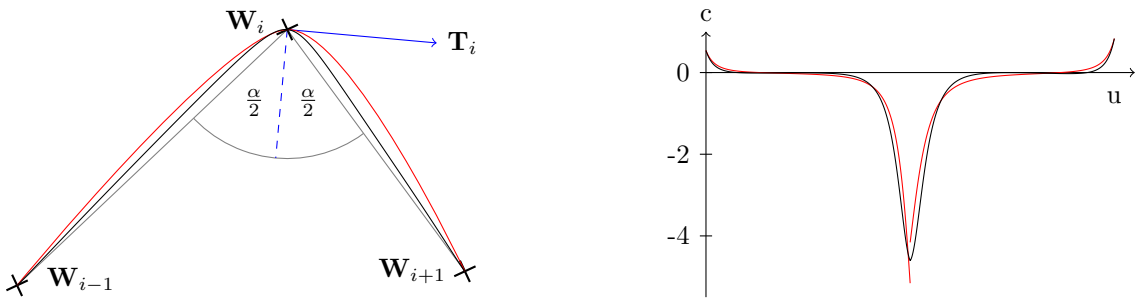
Since this function and all its terms, i.e., $\mathbf{Q}$ and its derivatives, are continuous, the curvature $c(u)$ of the path $\mathbf{Q}(u)$ is continuous as well.

The quintic Bézier splines presented up to this point are the extension of the commonly used cubic Bézier splines to higher-degree polynomials. Unlike with the cubic variant, we can design the quintic splines to have a continuous second derivative even at the junction points of the segments, and thus have a continuous curvature. Additionally, the second derivative at the start and end points can be freely specified. This allows us to plan paths with arbitrary start curvatures, which is required for a robot to adapt or replan paths while driving in curves, e.g., to react to unexpected obstacles.

There is also a downside to the additional degrees of freedom of the higher-order polynomials: when searching the space of possible paths or optimizing the free parameters, each additional parameter increases the dimensionality of the search space or optimization manifold. Additionally, we do not require the full expressivity of the quintic polynomials which can wiggle substantially more than cubic ones. The remainder of this chapter introduces heuristics that we use to control the wiggle, and at the same time, reduce the number of free parameters without losing the ability to control the orientation and curvature at start and end of the spline.

12.3 Heuristics for First and Second Derivatives

The path model presented in this section aims to smoothly connect a set of given waypoints $\mathbf{W}_i$. Besides the waypoints, the first and second derivative of the curve specified by the tangent $\mathbf{T}_i$ and Cartesian acceleration $\mathbf{A}_i$ at each waypoint also have a strong impact on the overall shape of the curve.



(a) Cubic and quintic Bézier splines

(b) Curvature of the splines

Figure 12.1: Heuristics for quintic Bézier splines. Two quintic spline segments (black) are joined at $\mathbf{W}_i$. A heuristic sets the tangent $\mathbf{T}_i$ (blue) orthogonal to the bisector of angle α (dashed blue). The second derivative at $\mathbf{W}_i$ averages the second derivative of joining cubic splines. The resulting curve looks similar to a cubic Bézier spline (red), but is curvature continuous at the junction point as shown on the right.

We adapt a tangent heuristic that is commonly used for cubic Bézier curves [Loustau and Dillon, 1992]. The angle of the tangent at each inner waypoint $\mathbf{W}_i$ is set to be perpendicular to the angular bisector of the line segments defined by $\mathbf{W}_i$ and its adjacent waypoints $\mathbf{W}_{i-1}$, $\mathbf{W}_{i+1}$ as shown in Figure 12.1a. The magnitude $\|\mathbf{T}_i\|$ of the tangent is set to half the Euclidean distance from $\mathbf{W}_i$ to the closest adjacent waypoint. Thus, $\mathbf{T}_i$ is given by

$$\mathbf{T}_i = e_i \cdot \frac{1}{2} \min\left\{\|\mathbf{L}_{i-1}\|,\ \|\mathbf{L}_i\|\right\} \cdot \frac{1}{2}\left(\frac{\mathbf{L}_{i-1}}{\|\mathbf{L}_{i-1}\|} + \frac{\mathbf{L}_i}{\|\mathbf{L}_i\|}\right), \tag{12.4}$$

where e_i is a scalar elongation factor that we use to adjust the sharpness of curves during path optimization, and $\mathbf{L}_i = \mathbf{W}_{i+1} - \mathbf{W}_i$ the vector corresponding to the i-th segment in the linear path specified by the waypoints.

In contrast to the quintic Bézier curves used by our model, the second derivative of cubic curves is not a free parameter. However, cubic splines minimize the integral over the absolute value of the second derivative of the curve [Plato, 2003]. In general, this corresponds to minimal changes in curvature, which is appealing for our path model. This motivates the heuristic that we propose for specifying the second derivatives $\mathbf{A}_i$ at the inner waypoints, which mimics the behavior of cubic splines while providing curvature continuity.

Given the waypoints $\mathbf{W}_{i-1}$, $\mathbf{W}_{i+1}$ adjacent to $\mathbf{W}_i$ and the corresponding tangents, cubic Bézier splines are completely defined for both adjacent segments. They join at $\mathbf{W}_i$ with discontinuous curvature, and we use a weighted mean of their second derivatives at $\mathbf{W}_i$ as the desired second derivative $\mathbf{A}_i$ of the quintic Bézier spline. The weights are proportional to the normalized length

of the respective other segment, which prevents long segments from deforming short segments. Thus, $\mathbf{A}_i$ is given by

$$\mathbf{A}_i = \mathbf{Q}''_{i-1}(1) = \mathbf{Q}''_i(0) = \frac{\|\mathbf{L}_i\|}{\|\mathbf{L}_{i-1}\| + \|\mathbf{L}_i\|}\,\mathbf{C}''_{i-1}(1) + \frac{\|\mathbf{L}_{i-1}\|}{\|\mathbf{L}_{i-1}\| + \|\mathbf{L}_i\|}\,\mathbf{C}''_i(0)\,, \tag{12.5}$$

where $\mathbf{C}_i(u_i)$ is the cubic equivalent of the quintic spline $\mathbf{Q}_i(u_i)$. Using this heuristic, the shape of the joined quintic Bézier spline is similar to the joined cubic Bézier spline, but is curvature continuous at the junction point $\mathbf{W}_i$ as shown in Figure 12.1.

For the sake of clarity, we formulated the heuristics presented in this section for inner waypoints only. However, their application at the start or end of a path is straight forward, since both are based on weighted averages of two adjacent control points where one can simply be left out. In the context of motion planning, however, the tangent orientation and the acceleration at the start point can be predetermined from the current pose of the robot and the curvature of its current path. Thus, the optimization of our motion planning system adjusts the position of the inner waypoints $\mathbf{W}_1, \ldots, \mathbf{W}_{N_Q-1}$ and the elongation factors e_i for all waypoints $\mathbf{W}_i$, $i = 0, \ldots, N_Q$, and uses the heuristics to determine the tangent orientations and the second derivatives for all but the start point.

12.4 Arc Length Parametrization

For application in motion planning systems, it can be desirable to express a path $\mathbf{Q}(u)$ as a function of its own arc length s, for example, to specify a velocity profile based on s. This requires a mapping from s to the internal spline parameter u, which is a non-linear relation when using polynomial splines.

In general, the metric distance s along a path $\mathbf{Q}(u)$ from the start up to u is given by integrating over the norm of the first derivative of $\mathbf{Q}$,

$$s(u) = \int_0^u \|\mathbf{Q}'(U)\|\, dU\,. \tag{12.6}$$

If this integral cannot be solved in closed form for a given curve type, one can use numerical integration to compute the mapping. A straight forward solution is to evaluate the curve in small

intervals of u and to sum up the Euclidean distance between these points. Given a set of u_k that follows u in small steps, the arc length can be approximated by

$$s(u) \approx \sum_{0<u_k \leq u} \|\mathbf{Q}(u_k) - \mathbf{Q}(u_{k-1})\| \,. \tag{12.7}$$

If higher accuracy is required, one can use more sophisticated approximations like Gaussian quadrature as proposed by Guenter and Parent [1990].

When iterating over the u_k, one can store the corresponding $s(u_k)$ in an array to allow for quick lookups. By choosing the u_k such that the resulting s_k are equidistant, we can also lookup the inverse mapping $u(s)$. Thus, we can efficiently evaluate $\mathbf{Q}(s) = \mathbf{Q}(u(s))$.

13 Trajectories and Velocity Profiles

As discussed before, we use the term *trajectory* to refer to a two-dimensional function $\mathbf{Q}(t)$ that defines the robot position $(x, y)^T$ for every point in time t. Similar to previous approaches like the work by Howard and Kelly [2007] and Shiller and Gwo [1991], we represent the shape and the planned translational velocity of the robot independently. Thus, we consider a *path* $\mathbf{Q}(u)$ that specifies the shape of a trajectory as a function of an internal parameter u as described in Chapter 12. A corresponding velocity profile $v(s)$ encodes the desired translational velocity of the robot for every point on the path, expressed as a function of the arc length s of the path. Integrating the velocity profile creates a mapping from s to time,

$$t(s) = \int_0^s \frac{1}{v(S)} dS \,. \tag{13.1}$$

Like the arc length parametrization $s(u)$ described in Section 12.4, the mapping $t(s)$ can be evaluated numerically and inverted to obtain $s(t)$. Chaining it with $u(s)$ finally yields the trajectory $\mathbf{Q}(t) = \mathbf{Q}(u(s(t)))$.

In order to guarantee that $\mathbf{Q}(t)$ is a feasible trajectory that the robot can traverse without high errors, it should obey the kinematic and dynamic limitations of the robotic hardware. Additionally, the presence of obstacles or sensitive payload might pose further constraints on the allowed velocities and accelerations. While potential kinematic constraints can be addressed by the path $\mathbf{Q}(u)$ alone, the velocity profile has to specify velocities and accelerations that implement the dynamic constraints for a given path.

This chapter presents an approach to compute a feasible velocity profile $v(s)$ for a given path. The velocity profile minimizes the travel time and at the same time satisfies a given set of dynamic constraints. We model $v(s)$ as a piecewise linear function $v(s_k)$ with K closely spaced support points s_k, $k = 0, \ldots, K-1$. This implies piecewise constant translational accelerations between the s_k. Each $v_k = v(s_k)$ is set to the maximum feasible velocity that obeys the constraints and the iterative method described in the following.

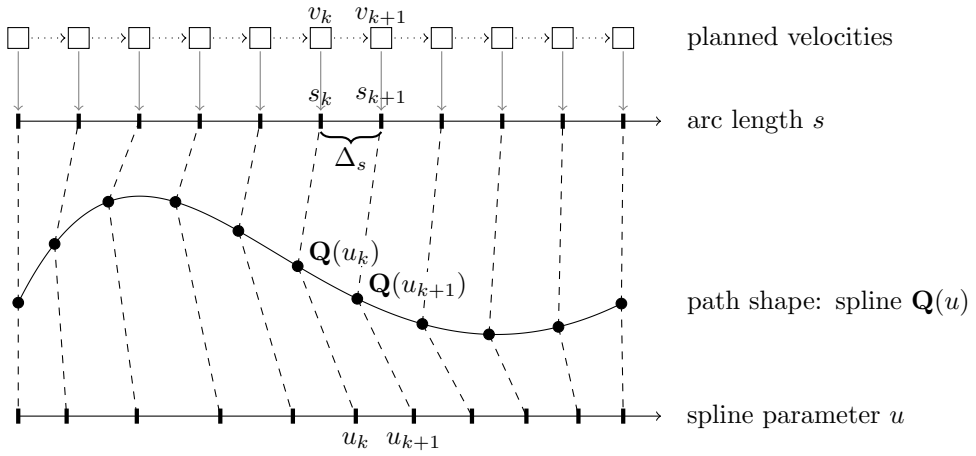


Figure 13.1: Velocity profile $v(s)$ for a path $\mathbf{Q}(u)$. Maximum admissible velocities v_k are computed for support points s_k placed in equidistant arc length intervals Δ_s along the curve. The arc length parametrization of $\mathbf{Q}(u)$ computes values for u_k that correspond to the s_k. In practice, the interval Δ_s is very small with respect to the path shape.

13.1 Direct Velocity Constraints

This section introduces a set of constraints that directly limit the admissible velocities. This means that the velocities v_k can be computed for each support s_k, independently from the other $v_j,\ j \neq k$. Without loss of generality, we assume the robot to perform forward motion, i.e., $v_k \geq 0$, which allows for a more compact notation.

Let $v_{\max}$ be the constant maximum translational velocity of the robot. Obviously, this is a first constraint $v_{k|v}$ on all v_k with $k = 0, \ldots, K-1$, and given by

$$v_k \leq v_{k|v} = v_{\max} \tag{13.2}$$

Robots equipped with synchro, differential, or holonomic drives can perform very sharp turns or even turn on the spot. In most cases, the hardware or the payload imposes an upper bound on the rotational velocity ω. Since ω and the translational velocity v are proportional for a given curvature c, a bound on ω implies a constraint $v_{k|\omega}$ on all v_k according to

$$v_k \leq v_{k|\omega} = \frac{\omega_{\max}}{|c_k|}\,, \tag{13.3}$$

with c_k being the curvature of the path at s_k.

To ensure safe motion in the presence of obstacles, the sum of the braking distance s_b and the traveled distance s_r during the reaction time t_r of the robot has to be smaller than the distance d to the closest obstacle. Given the maximum deceleration a_b during braking, we can derive the following inequality:

$$\begin{aligned} & s_r + s_b \leq d \\ \Leftrightarrow\ & v\, t_r + \frac{v^2}{2a_b} \leq d \\ \Leftrightarrow\ & v^2 + 2a_b t_r v \leq 2a_b d \end{aligned}$$

Solving this inequality for the velocity v yields the corresponding constraint $v_{k|\text{obst}}$ on v, which is given by

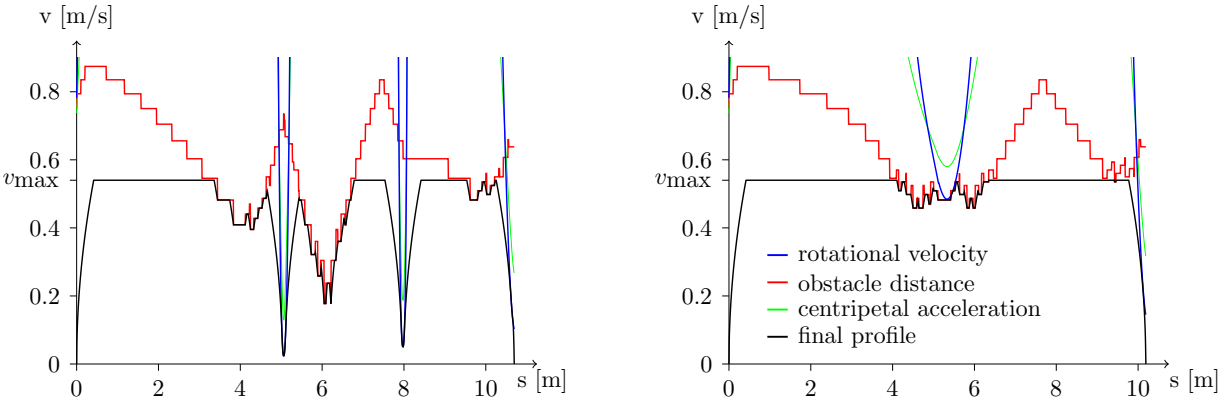
$$v_k \leq v_{k|\text{obst}} = -a_b \cdot t_r + \sqrt{a_b^2 \cdot t_r^2 + 2a_b d}\,, \tag{13.4}$$

which only depends on the given parameters for the maximum deceleration a_b, the reaction time t_r and the minimum obstacle distance d according to the map or the sensors of the robot. Note that this is a conservative constraint – a more complex formulation could account for the robot's direction of motion with respect to the obstacle.

Finally, we limit the centripetal acceleration a_{cent} to prevent skidding of the vehicle in curves and to protect sensitive payload. Since $a_{\text{cent}} = v^2 \cdot c$ depends on the velocity and the curvature, we can also express this as a constraint $v_{k|\text{cent}}$ on the v_k, which is given by

$$v_k \leq v_{k|\text{cent}} = \sqrt{\frac{a_{\text{cent}}}{|c_k|}} \tag{13.5}$$

An example for the impact of the constraints on the velocity profiles is shown in Figure 13.2. The colored curves plot the values of $v_{k|\omega}$, $v_{k|\text{obst}}$, and $v_{k|\text{cent}}$, while the constant speed limit $v_{k|v} = v_{\max}$ is marked on the vertical axis. Our optimization alters the path shape to alleviate the impact of the constraints and allow faster movement, which reduces the time of travel. The slopes of the final velocity profile, as visible at the start and end of the path as well as around sharp corners, are governed by the accelerational constraints described in the following section.



(a) Velocity profile of the initial path

(b) Velocity profile of the optimized path

Figure 13.2: Velocity profiles of the trajectories in Figure 9.2. The profiles obey the direct velocity constraints (colored curves), which limit the maximum admissible velocity especially in sharp curves. Furthermore, they comply with accelerational constraints that cause the smooth velocity changes at start and end, as well as before or after sharp curves.

13.2 Accelerational Constraints

In addition to the isolated constraints described above, v_k is also limited by the constraints on v_{k-1} and v_{k+1} in connection with the maximum translational acceleration a_{trans} and the maximum rotational acceleration a_{rot} of the robot.

In many cases, the start velocity v_0 is fixed, e.g., with $v_0 = 0$ if the trajectory is planned for the initially standing robot, or with a given velocity if the trajectory is replanned during motion. Let Δ_t be the travel time between two supports s_{k-1} and s_k. Assuming constant acceleration between s_{k-1} and s_k, the translational velocity v_k is constrained by v_{k-1} according to

$$v_k \geq v_{k|\text{amin,k-1}} = v_{k-1} - a_{\text{trans}} \cdot \Delta_t \tag{13.6}$$

$$v_k \leq v_{k|\text{amax,k-1}} = v_{k-1} + a_{\text{trans}} \cdot \Delta_t \tag{13.7}$$

$$v_k \cdot c_k \geq \omega_{k|\text{amin,k-1}} = \omega_{k-1} - a_{\text{rot}} \cdot \Delta_t = v_{k-1} \cdot c_{k-1} - a_{\text{rot}} \cdot \Delta_t \tag{13.8}$$

$$v_k \cdot c_k \leq \omega_{k|\text{amax,k-1}} = \omega_{k-1} + a_{\text{rot}} \cdot \Delta_t = v_{k-1} \cdot c_{k-1} + a_{\text{rot}} \cdot \Delta_t \tag{13.9}$$

For an implementation, a formulation of the constraints depending on Δ_s rather than Δ_t is desirable. See the work by Sprunk for the mathematical derivations [2008].

If the acceleration and deceleration constraints are not the same, different values have to be used for a_{trans} in these two phases. Only for synchro drive robots, a_{trans} and a_{rot} are totally

independent. However, since the mass m of platform and payload governs these limits via translational inertia and rotational moment of inertia, it is reasonable to assume a_{trans} and a_{rot} to be independent for other platforms as well.

The constraints presented above affect velocities in a "forward" direction, i.e., each velocity v_k depends on the previous one given by v_{k-1}. In practice, these constraints have an effect when accelerating after moving slow or after a curve. Similarly, there are "backwards" constraints, for example, braking before a curve or when stopping at the goal. Thus, each velocity v_k also depends on the succeeding one given by v_{k+1}, which yields the limits $v_{k|\text{amin,k+1}}$, $v_{k|\text{amax,k+1}}$, $\omega_{k|\text{amin,k+1}}$, and $\omega_{k|\text{amin,k+1}}$.

For a given v_k, the translational and rotational acceleration impose an upper and a lower bound on the feasible velocities of v_{k+1} (forward), and on v_{k-1} (backwards) as well. Depending on the curvature of the path and the velocity assumed for v_k, the ranges specified by these bounds can be disjunct. In practice, this could be a vehicle that approaches a sharp curve with an excessive velocity v_k. If no valid velocity v_{k+1} can be found, the trajectory is infeasible, i.e., the vehicle cannot execute it without substantial deviations, even if no skidding occurs.

However, as shown by Sprunk [2008], we can compute an upper bound $v_{k|\text{overlap}}$ as an additional constraint on each v_k that guarantees the accelerational constraints to overlap. Thus, if $v_{k-1} \leq v_{k-1|\text{overlap}}$ and $v_{k+1} \leq v_{k+1|\text{overlap}}$, there exists at least one admissible value for v_k. This constraint can be computed in closed form, and is a function of the curvature c_k and the maximum accelerations of the platform [Sprunk, 2008].

13.3 Computing Compliant Profiles

Given a planned path $\mathbf{Q}(s)$, we seek the velocity profile $v(s)$ that minimizes the time required to traverse the resulting trajectory $\mathbf{Q}(t)$, and at the same time obeys all constraints described in Section 13.1 and 13.2. As written before, we consider piecewise constant accelerations, which causes $v(s)$ to be a piecewise linear function. The corresponding values v_k can be computed efficiently in three iterations as described in the following.

In the first iteration, we compute the maximum value for each v_k that satisfies the direct velocity constraints and the overlap constraints according to

$$v_k \leftarrow \min\left\{v_{k|v},\, v_{k|\omega},\, v_{k|\text{obst}},\, v_{k|\text{cent}},\, v_{k|\text{overlap}}\right\} \text{ for } k = 0, 1, \ldots, K-1 \tag{13.10}$$

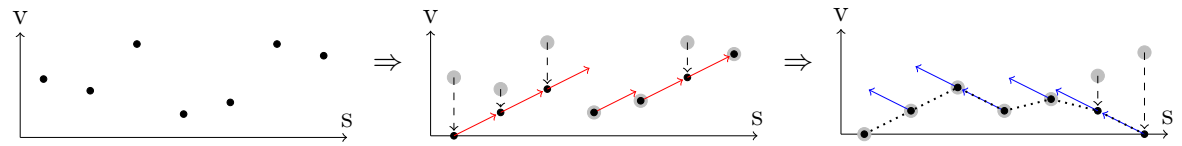


Figure 13.3: Velocity profile generation in three phases. Black points denote the admissible translational velocities v_k, gray discs mark values from the previous phase. At first, the v_k are independently limited by direct velocity constraints (left). Secondly, the profile is made consistent for increasing s_k, obeying a given start velocity and acceleration constraints (red arrows). Finally, we establish consistency for decreasing s obeying the end velocity and deceleration constraints (blue arrows), which yields the final profile (dashed).

As shown in Figure 13.3 (left), each v_k is independent from the values of other supports.

The second iteration establishes forward consistency, i.e., every v_k is feasible regarding the accelerational constraints with respect to v_{k-1}. Therefore, each v_k is updated according to

$$v_k \leftarrow \min \left\{ v_k,\ v_{k|\text{amax,k-1}},\ \tfrac{1}{c_k} \cdot \omega_{k|\text{amax,k-1}} \right\} \text{ for } k = 1, 2, \ldots, K-1 \tag{13.11}$$

Note that the order of the update is important, since the bounds on v_k depend on the current value of the previously computed v_{k-1}. Figure 13.3 shows the example profile after the second iteration for a given start velocity $v_0 = 0$.

Finally, the third iteration additionally establishes backward consistency by updating the v_k with the remaining constraints in reversed order,

$$v_k \leftarrow \min \left\{ v_k,\ v_{k|\text{amax,k+1}},\ \tfrac{1}{c_k} \cdot \omega_{k|\text{amax,k+1}} \right\} \text{ for } k = K-2, \ldots, 1, 0 \tag{13.12}$$

Since (a) all iterations can only reduce the translational velocity for a given support, (b) all direct velocity constraints are formulated as upper bounds, and (c) the accelerational constraints are satisfiable for velocities below $v_{k|\text{overlap}}$, no iteration can invalidate the constraints satisfied in previous iterations. Thus, after completing the third phase, the velocity profile realizes piecewise constant accelerations and meets all described velocity and acceleration constraints as visualized in Figure 13.3 (right). Thus, it is therefore traversable by the robot with high accuracy. With monotonous curvature changes between the closely spaced support points, the constraints hold not only at the support points, but in between as well. Note that if the velocity profile was defined over time instead of distances, changing velocities would imply changes of the position of support points on the path, which complicates the algorithm.

14 Trajectory Generation and Execution

This chapter presents techniques to create, optimize, and execute kinodynamic trajectories using the trajectory model described in the previous chapters. We use a global path planner to plan waypoints forming a piecewise linear path. Given these waypoints, we create an initial spline-based trajectory that closely approximates the piecewise linear path, and thus represents a feasible yet inefficient trajectory. By optimizing the trajectory with respect to a user-defined cost function, the initial plan is modified to reduce for example the time of travel or the energy consumption.

After a final trajectory has been determined, it can be executed by a robot. We therefore present the employed error-feedback controller which determines the control signals for the robotic hardware. Furthermore, we discuss the importance of the ability to update planned paths while moving and describe how this is done by our system.

14.1 Global Planning and Creation of Initial Trajectories

A number of global path planning algorithms like A* exist that plan on grids or graphs. They generate for example piecewise linear paths given a start and goal location and a representation of the traversable space around the robot. Our trajectory generation approach requires such a planner to obtain waypoints for parametric trajectories, but it is not specific to a certain method.

For our experiments, we employ the value iteration planner by Thrun and Bücken [1998], which operates on a high-resolution 2D grid map. To account for unmapped obstacles, the map is regularly updated using the onboard sensors of the robot. The planner returns the shortest traversable path from the position of the robot to a given goal location as a piecewise linear path, where each segment connects two neighboring grid cells in the grid map. This path is collision-free, but contains sharp discontinuous corners. It is reduced to sparse waypoints by iteratively replacing two adjacent line segments with a direct straight line, if the resulting path is still free of collisions and the resulting segment is shorter than a maximum length.

Because of the sparsity of the waypoints after pruning, only the first four (including the start) are usually in the robot's field of view, cf. Figure 9.2. In order not to spend computational time on motion planning for areas where unmapped obstacles cannot be perceived and correctly accounted for, we only use the first four waypoints for trajectory generation.

Between each pair of consecutive waypoints, we create a quintic Bézier spline using the path model described in Chapter 12. To approximate the piecewise linear path given by the waypoints, a small tangent elongation factor $e = 0.5$ is used, which causes sharp turns at the waypoints. The resulting spline path approximates the given piecewise linear path as shown in Figure 9.2a. If the deviations of the spline are large enough to cause collisions with the environment, we insert additional waypoints that bisect the affected line segments. According to Equation (12.4), this shortens the tangents at the waypoints. Due to the convex hull property of Bézier splines, this reduces the deviation of the spline from the piecewise linear path as well.

The result is an initial trajectory consisting of three or more continuously concatenated spline segments. It can usually only be traversed with very low speed due to the sharp turns, but it is collision-free and respects the kinodynamic constraints of the hardware platform. This trajectory is refined during the optimization process to minimize the estimated time of travel, which is typically achieved by performing wider turns as shown in Figure 9.2b.

14.2 Optimization

The optimization system takes an initial trajectory (generated as described above) and a distance-transformed obstacle map as input. This map contains obstacles that are known a-priori, as well as previously unmapped obstacles that are detected by the sensors of the robot. The optimization changes the shape of the trajectory to reduce a given cost measure by adjusting a set $\mathcal{P}$ of tunable parameters. In our experiments, we use the overall time of travel as cost measure, but other costs like the estimated energy consumption could be considered as well. The parameter set consists of the tangent elongation factor e for the first and each of the inner waypoints, as well as the position of the inner waypoints relative to their closest obstacle. The location of the first and last waypoint are usually fixed, since they correspond to the start and goal location.

As shown in Figure 13.2, the optimization alters the shape of the trajectory in a way that alleviates the influence of the constraints on the admissible velocity: the optimized trajectory is faster due to less sharp curves and larger distances from the wall (elongated tangents), and shorter as well (moved waypoints).

Algorithm 14.1 Iterative trajectory optimization

$\hat{\mathbf{Q}}_{\text{best}} \leftarrow$ initial trajectory
$\mathcal{P} \leftarrow$ parameters of initial trajectory
repeat
  $\Delta_{\text{term}} \leftarrow 0$
  for all $p \in \mathcal{P}$ **do**
    $\hat{\mathbf{Q}}_{\text{curr}} \leftarrow \hat{\mathbf{Q}}_{\text{best}}$
    $d_p \leftarrow d_p^0$
    repeat
      $\hat{\mathbf{Q}}_{\text{mod}} \leftarrow \text{MOD}(\hat{\mathbf{Q}}_{\text{curr}}, p \leftarrow p + d_p)$
      if $\text{cost}(\hat{\mathbf{Q}}_{\text{mod}}) < \text{cost}(\hat{\mathbf{Q}}_{\text{best}})$ **then**
        $\Delta_{\text{term}} \leftarrow \max(\Delta_{\text{term}}, \text{cost}(\hat{\mathbf{Q}}_{\text{best}}) - \text{cost}(\hat{\mathbf{Q}}_{\text{mod}}))$
        $\hat{\mathbf{Q}}_{\text{best}} \leftarrow \hat{\mathbf{Q}}_{\text{mod}}$
        break
      $d_p \leftarrow \begin{cases} 1.2 d_p & \text{cost}(\hat{\mathbf{Q}}_{\text{mod}}) < \text{cost}(\hat{\mathbf{Q}}_{\text{curr}}), \\ -0.5 d_p & \text{else.} \end{cases}$
      $\Delta_{\text{cost}} \leftarrow |\text{cost}(\hat{\mathbf{Q}}_{\text{mod}}) - \text{cost}(\hat{\mathbf{Q}}_{\text{curr}})|$
      $\hat{\mathbf{Q}}_{\text{curr}} \leftarrow \hat{\mathbf{Q}}_{\text{mod}}$
    until $\Delta_{\text{cost}} < \epsilon_p$ OR time is up
until $\Delta_{\text{term}} < \epsilon_{\text{term}}$ OR time is up
return $\hat{\mathbf{Q}}_{\text{best}}$

The cost function, i.e., the total time of travel, depends on the trajectory shape together with the velocity profile, and thus on several constraints and the obstacle distance map. Since this function is not differentiable, we use an optimization scheme inspired by the derivative-free RPROP algorithm by Riedmiller and Braun [1993].

The optimization algorithm, shown in Algorithm 14.1, refines a trajectory as follows: in every iteration, as long as planning time is left and the optimization of any of the parameters in the last iteration has brought an improvement bigger than a threshold ϵ_{term}, the parameters are optimized successively: for a chosen parameter $p \in \mathcal{P}$ a new spline is computed by adding an initial offset $d_p := d_p^0$ to that parameter.

Now a velocity profile is generated for the modified spline, as described in Chapter 13. The spline and the velocity profile together form the modified trajectory $\hat{\mathbf{Q}}_{\text{mod}}$. From this, the cost, i.e., the time needed to traverse the trajectory is determined. If the trajectory is not collision-free according to the obstacle map, its cost is set to infinity.

If this cost is lower than the cost of the current best trajectory $\hat{\mathbf{Q}}_{\text{best}}$, the changed parameter $p \leftarrow p + d_p$ is kept, and the next parameter is optimized. Otherwise, the parameter p is further optimized with adaptive step sizes as in RPROP: the offset d_p is increased to $1.2 d_p$ if the cost

is reduced, and inverted/reduced to $-0.5d_p$ if the cost is increased. The optimization for that parameter terminates if a modified trajectory is better than the best one, if the resulting changes in cost are smaller than a threshold ϵ_p, or if planning time is up.

The trajectories generated by our approach are limited to the area around the waypoints in the piecewise linear path provided by the global path planner. To also consider trajectories that follow a different route with respect to obstacles, the algorithm can be run in parallel with different piecewise linear path inputs, e.g., the k-shortest paths that are topologically different with respect to obstacles.

14.3 Error-feedback Controller

Given a planned trajectory $\hat{\mathbf{Q}}(t)$ that defines the desired robot position for each time step t, we use the *dynamic feedback linearization* controller developed by Oriolo *et al.* [2002] to steer the robot along the trajectory.

The second derivative $\hat{\mathbf{Q}}''(t)$ of the trajectory is the desired vector of accelerations in Cartesian coordinates, which the controller uses as feed-forward control signal. The deviation of the robot's actual position $\mathbf{x}(t)$ and Cartesian velocity $\mathbf{x}'(t)$ from the targeted position $\hat{\mathbf{Q}}(t)$ and velocity $\hat{\mathbf{Q}}'(t)$ are used as error-feedback signals. In every time step t, the signal u aggregates the feed-forward and the error signals according to

$$\mathbf{u}(t) = \hat{\mathbf{Q}}''(t) + \kappa_p \left(\hat{\mathbf{Q}}(t) - \mathbf{x}(t) \right) + \kappa_d \left(\hat{\mathbf{Q}}'(t) - \mathbf{x}'(t) \right) , \tag{14.1}$$

where we choose $\kappa_p = 1.0$ and $\kappa_d = 0.7$ as control gains. Considering the orientation of the robot $\theta(t)$, the controller computes the state change of a dynamic compensator [Oriolo *et al.*, 2002], which is given by

$$\xi'(t) = \mathbf{u}_x(t) \cdot \cos\theta(t) + \mathbf{u}_y(t) \cdot \sin\theta(t) . \tag{14.2}$$

By integrating ξ over time, the dynamic compensator computes the appropriate control signals for the robotic hardware, i.e., the translational and rotational velocities $v(t)$ and $\omega(t)$ according to

$$v(t) = \xi(t) , \qquad \omega = \frac{\mathbf{u}_y(t) \cdot \cos\theta(t) - \mathbf{u}_x(t) \cdot \sin\theta(t)}{\xi(t)} . \tag{14.3}$$

The dynamic compensator given by Equation (14.3) has a singularity for $\xi = 0$, which in practice can result in erroneous rotations when the robot softly starts moving or stops at the goal. In our experiments we found it sufficient to choose $\xi = 0.01$ when computing ω for smaller values of ξ to overcome this problem.

To compensate for a control delay in the robotic hardware, one can use an equivalent look-ahead in time by adding an offset to t when evaluating $\hat{\mathbf{Q}}''(t)$ in Equation (14.1).

The controller requires the position $\mathbf{x}$ of the robot to compute the error-feedback signal. Without external sensors, the position estimate can for example be obtained from a laser-based localization, or from the odometry of the robot. In many applications, the pose estimate of the localization system might not be updated at a rate that is sufficient as motor control frequency. Additionally, a jump in the pose estimate would cause a jump in the control signals. Therefore, we found it more practical to use integrated odometry measurements to estimate the position. To prevent the system from drifting due to the accumulation of odometry errors, we use the localization system to update the estimated displacement between pose estimates from odometry and localization, and employ the replanning procedure described in the following to update the planned trajectories accordingly.

14.4 Replanning Procedure

When a motion planning system is used on a real robot, several issues can occur that require the robot to frequently replan its trajectory:

- If the odometry is used to obtain position estimates for the error-feedback controller, errors in the odometry measurements accumulate.
- The pose estimate of the robot's localization system can jump, especially when using multi-modal distributions to estimate the position.
- Unmapped obstacles such as moving people can occur that were not expected when planning the trajectory.
- When planning only partial trajectories for a certain distance ahead of the robot, new pieces have to be added frequently to complete the trajectory.

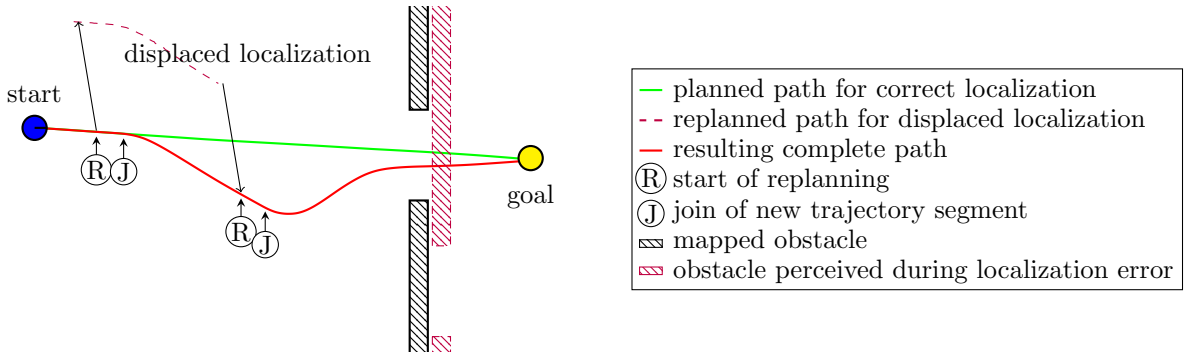


Figure 14.1: Replanning and resulting trajectory (red) for a temporarily erroneous pose estimate from the localization system. While the original planned path (green) considers the mapped obstacle (black), the erroneous pose estimate requires replanning to consider the unexpected obstacle (purple), and another replanning after the localization system has recovered.

In all cases, it is desirable to update the planned trajectory while moving in order to avoid delays. Furthermore, the modified or new trajectory pieces should join the current one smoothly and without curvature discontinuities. Assume the system starts replanning a trajectory $\hat{\mathbf{Q}}(t)$ at time step t_R. When allowing a planning time t_{plan}, the robot will move along $\hat{\mathbf{Q}}$ until time step $t_J = t_R + t_{plan}$. Thus, the updated trajectory should join the current one at $\hat{\mathbf{Q}}(t_J)$. Figure 14.1 illustrates this procedure for an erroneous temporary displacement of the pose estimate of the robot. When the erroneous pose estimate causes the planned trajectory (green path) to be invalid due to the anticipated location of the obstacle (purple), the system starts the replanning procedure at the first position marked with (R). It allocates a certain time for planning and chooses the position (J) for the join of the new trajectory. This considers the erroneous pose estimate and steers the robot towards the anticipated opening (red path). As soon as the pose estimate jumps back, the system starts another replanning procedure (R) and attaches a new trajectory piece (J) that steers the robot through the actual opening and to towards goal. All trajectory joins (J) are smooth and with continuous curvature.

This replanning procedure takes also care of odometry drift, allows the robot to evade moving obstacles like walking people, and enables the robot to extend a planned trajectory while moving.

15 Experiments

To test specific properties of our motion planning system, we conducted several experiments in simulation. On a real robot, we evaluated the precision and predictability of generated motion trajectories and their execution. Additionally, we evaluated the applicability of our system to real-world tasks in different populated environments.

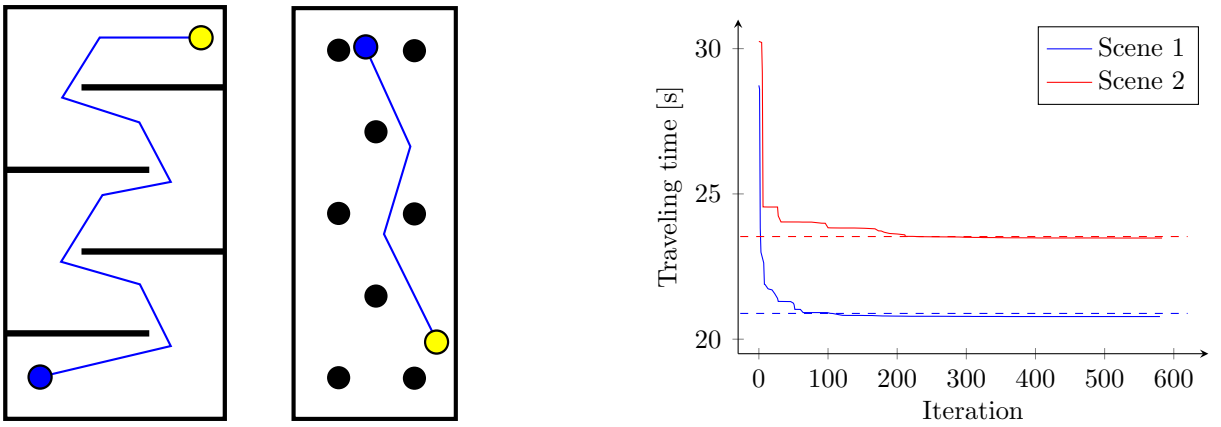
15.1 Evaluation in Simulation

To test the appropriateness of the proposed trajectory representation and the choice of optimization parameters for online motion planning, we created seven artificially maps, two of which are shown in Figure 15.1. Using the simulator of the CARMEN robot navigation system [Montemerlo *et al.*, 2003], we let our system generate and optimize trajectories for given start and goal locations these environments.

Just as RPROP, our optimization algorithm is able to deal with local minima to some extent. However, an ill-natured optimization manifold could cause suboptimal results. To assess this problem, we compared the trajectories optimized by our system with the best trajectories found by exhaustive search in a finely discretized version of the parameter space. In all cases, the parameters of the trajectory generated by our optimization approach fell into the discretization bin of the optimal parameters determined by the exhaustive search.

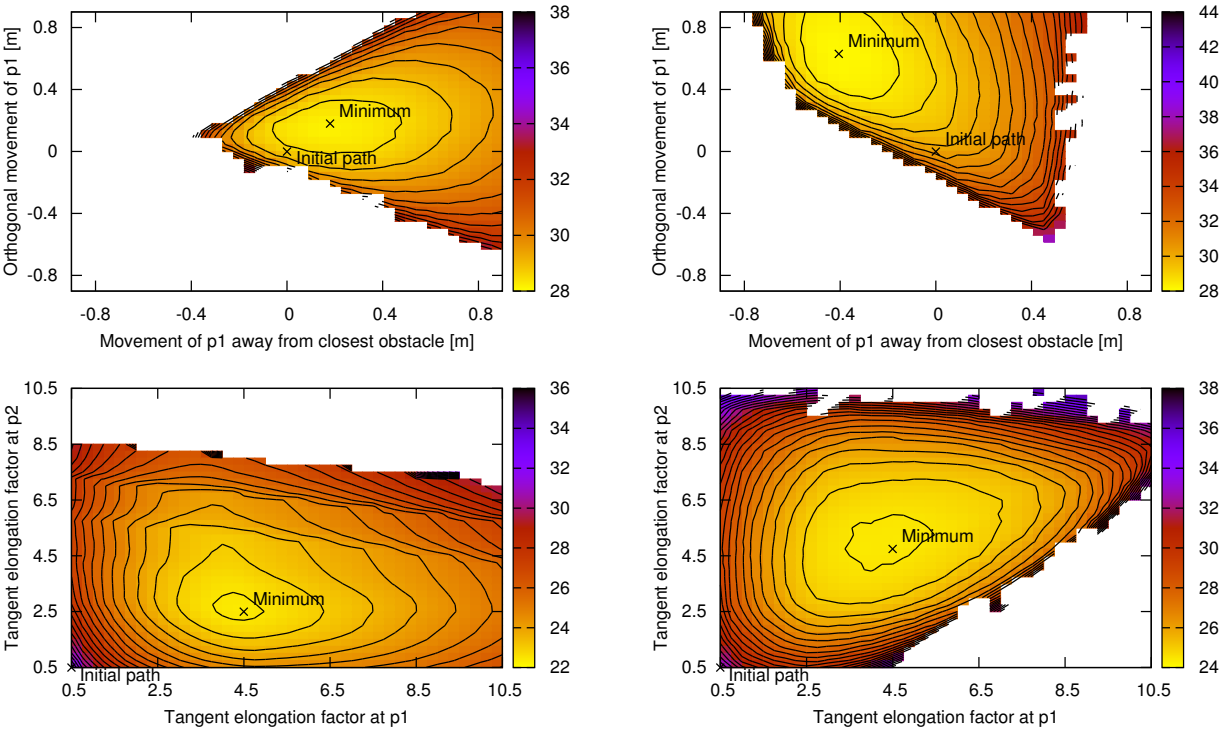
Figure 15.2 visualizes optimization manifolds for two generated trajectories. The plots show an exemplary subspace of the parameter set, namely the translation and tangent elongation of the first inner waypoint. The parameter values of the initial path as well as the ones of the achieved minimum when optimizing only these two parameters are marked in the figure. In the analysis, all parameters showed smooth and convex optimization manifolds without local minima.

Figure 15.1c shows the achieved reduction of traveling times for two trajectories as a function of executed optimization steps, which are exemplary for all tested scenarios. On average, the optimization reduces the estimated traveling time by 31% compared to the initial trajectory. In most cases, the optimization achieves the major improvements in the first 50 iterations, and 300 iterations were sufficient to reach the optimum. We therefore choose the maximum planning time



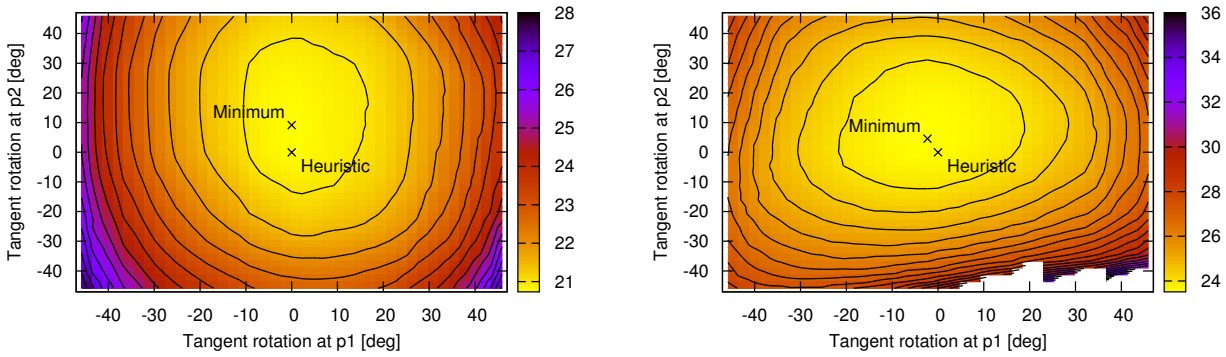
(a) Scene 1 (b) Scene 2 (c) Optimization of travel times

Figure 15.1: Trajectory optimization on artificial scenes. The blue lines in (a) and (b) denote the pruned piecewise linear paths connecting start (blue circle) and goal (yellow circle). The optimization alters the path shape to reduce the travel time. The dashed lines in (c) indicate the minimum found using exhaustive search in the discretized parameter space.



(a) Scene 1 (b) Scene 2

Figure 15.2: Optimization manifolds for the parameters of the first inner waypoint. The color and contour lines represent the traveling time as a function of the parameters, with a contour interval of 0.5 s. White areas denote configurations that lead to colliding trajectories.



(a) Scene 1 (b) Scene 2

Figure 15.3: Traveling times achieved using our heuristic compared to optimized values for the tangent orientations at the first inner waypoints. See the caption of Figure 15.2.

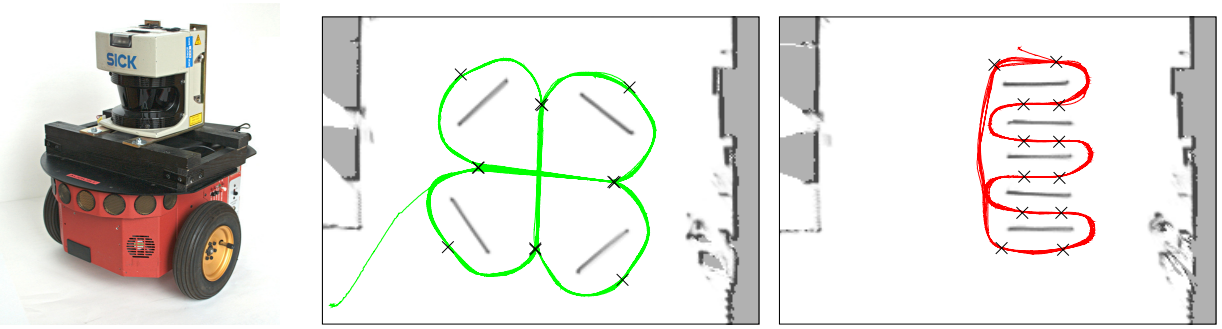
$t_{\text{plan}} = 0.4\,s$ for our experiments, which roughly corresponds to 400 iterations on the employed Intel Core 2 Duo at 1.6 GHz.

By performing an exhaustive search in the discretized parameter spaces of the optimization problems, we obtained reference trajectories with optimal traveling times, up to the discretization. The traveling times of the optimized trajectories were $0-1\%$ shorter than the respective reference times, which can be attributed to the advantage of continuous optimization over search even in a finely discretized space.

As discussed in Section 12.3, we use a heuristic to determine the tangent orientations at inner waypoints. This way, our path model based on quintic Bézier splines has even fewer free parameters than basic cubic splines. Optimizing the tangent orientations could further improve the traveling times at the expense of additional computation time. By using exhaustive search in the full parameter space including tangent orientations, we determined the resulting traveling times for our simulated test scenarios. As shown in Figure 15.3, the reduction of traveling time by optimizing tangent rotations is rather small, since the values chosen by our heuristic are already close to the optimum.

15.2 Motion Planning in Obstacle Courses

This experiment tests the precision and predictiveness of our motion planning system. We use a Pioneer robot equipped with a subnotebook (Intel Core 2 Duo, 1.6 GHz) for all computations and a laser range finder for localization and obstacle detection. The robot drives with a velocity limit



(a) Pioneer robot (b) Obstacle course “clover” (c) Obstacle course “zigzag”

Figure 15.4: Trajectory of a Pioneer robot overlaid on a grid map for two obstacle courses. The crosses mark intermediate goal locations. The robot drove 10 rounds in the clover course, and 5 rounds in the zigzag course. The frames cover an area of 14m x 10m.

of 0.5 m/s. The error-feedback controller, the laser scanner and the odometry are all running at 35 Hz. We have set up two obstacle courses and marked a sequence of goals for the global path planner in the map. The path planner switches to the next goal whenever the robot is closer than 1 m.

The trajectories of the robot are shown in Figure 15.4. While the goal locations for the global path planner are fixed, replanning is executed periodically and the locations used for trajectory planning are not spatially aligned. Thus, the positions of pruned waypoints, which are input to the trajectory planner, vary between the rounds. Nevertheless, the trajectories from the different rounds are very similar, which indicates that a) the parameter set for the optimization, which contains tangent elongation and movement of waypoints, is appropriate to find a global optimum, and b) the optimization generates reproducible trajectories that are not critically dependent on the input waypoints.

The average positional tracking error, i.e., how far the robot deviates from the planned path according to odometry, is below 2 cm for both obstacle courses, and the average deviation in velocity is below 2 cm/s, as shown in Figure 15.5 (left). Both errors are a little higher in the zigzag course which has tighter curves than the clover course.

The planned trajectories can be used to predict the position of the robot over time in global coordinates. The corresponding error is shown in Figure 15.5 (right) as a function of a lookahead time added to t_{plan}. Thus, a lookahead of 0 s corresponds to predictions over t_{plan}, which are needed for replanning.

Without replanning, the prediction error would reduce to the tracking error plus the accumulated error in the odometry readings. With replanning, as in the experiments, the predicting trajectory

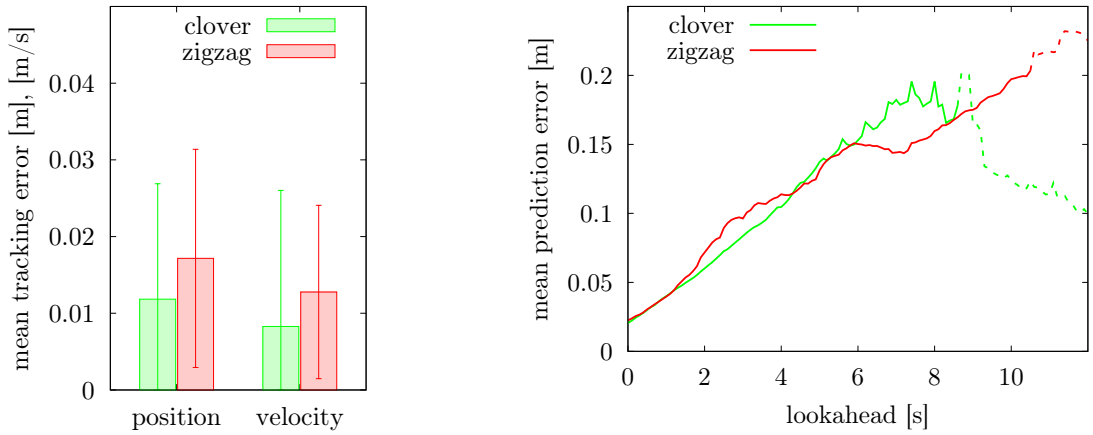


Figure 15.5: Experimental results for the obstacle courses shown in Figure 15.4. *Left:* Average Euclidean distance between planned and actual values of position and translational velocity. The error bars show the standard deviation. *Right:* Average Euclidean distance between predicted and actual position of the robot, as a function of the temporal lookahead added to the planning delay t_{plan}. In the dashed parts, less than 20 values contributed to the mean.

is seamlessly replaced after one second at most. Still, the mean prediction error is small, e.g., only around 15 cm for a lookahead of 6 s. This shows that replanning does not cause abrupt changes in the planned trajectory. Human observers can actually not recognize when switching of replanned trajectories takes place. Naturally, changes of the global goal location falsify these values and are filtered out, which causes a low number of values contributing to the averages for large lookahead times.

15.3 Navigation in Populated Environments

To evaluate the performance of our motion planning system in an application with dynamic obstacles, we tested the system on our tour-guide robot "Albert" during a three-day trade show in Freiburg. Motion planning was executed on the same subnotebook, with the odometry and the error-feedback controller running at 10 Hz. An example trajectory is shown in Figure 15.6. One person was purposely blocking the path of the robot for a longer period (1). Normally, the motion planning system plans smooth trajectories around unmapped obstacles that are seen in advance (2), but can also execute sharp turns if required, i.e., if an obstacle suddenly occurs in front of the robot (3). The performance in presence of a person who deliberately blocks the path of the robot could be improved by adding an additional strategic planner to avoid oscillation of plans generated by the global path planner when the person moves from one side to the other.

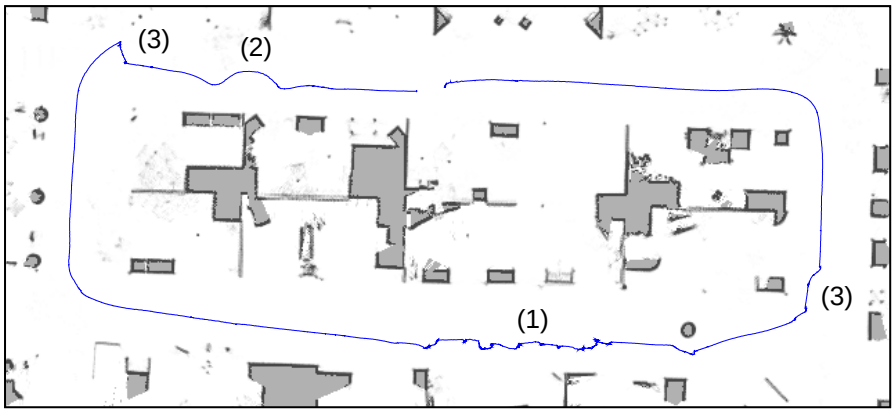
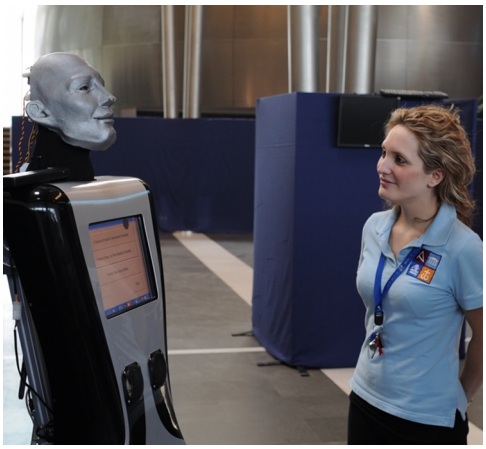
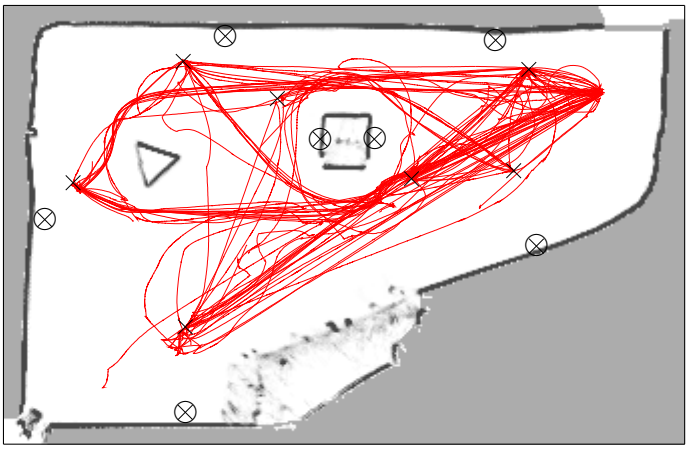


Figure 15.6: Sample trajectory of our tour-guide robot "Albert" at a trade show. The frame covers an area of 33m x 15m.



(a) Tour guide robot "INDIGO"



(b) Overlay of all trajectories driven in the Tholos foyer

Figure 15.7: Field trials of our motion planning system in the Hellenic Cosmos museum, Athens

Our system was also used on the INDIGO robot shown in Figure 15.7a, a personality and dialog enabled robot that was designed to work as a museum tour guide. The field trials were run in three different spaces in the Hellenic Cosmos museum in Athens. As shown in Figure 15.7b, the robot navigated between predefined goal locations (crosses) to show the visitors a number of exhibits (encircled crosses). In the 28 days of operation, the INDIGO robot travelled a total distance of 2859 m in 25 hours, see Table 15.1. Figure 15.7b shows an overlay of all trajectories driven in the Tholos foyer. As long as the path of the robot was not abruptly blocked by a person, it executed smooth trajectories among the moving people.

Table 15.1: Field trials of the tour guide robot "INDIGO" at the Hellenic Cosmos, Athens

Dates	Environment	Days	Travelled distance [m]	Travelled time [h]
2009 July 1-10	Tholos foyer	7	947.8	5.0
2009 Sept 21-25	Ismene hall	5	887.0	5.0
2009 Dec 1-22	Niobe room	16	1024.1	15.4
	Total	28	2858.9	25.4

16 Extensions and Applications

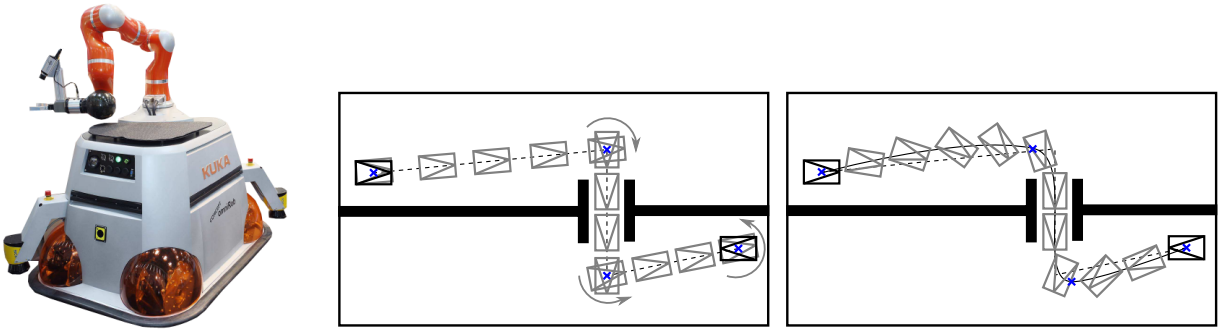
Besides from the presented experiments, the proposed motion planning system or parts of it have been used in different application scenarios. Lee *et al.* for example proposed an online complete coverage planning system for applications like lawn mowing, floor cleaning, demining or harvesting [Lee *et al.*, 2011]. It uses our path representation based on quintic Bézier splines (see Chapter 12), the constraint formulations for direct and accelerational constraints (Chapter 13), as well as the trajectory tracking mechanism (Section 14.3).

This chapter presents more applications of our motion planning system system and its path model proposed by Sprunk *et al.*, namely an extension for omni-directional platforms [2011] and a method to teach trajectories to a robot by demonstration [2012].

16.1 Trajectory Generation for Omni-Directional Robots

Robotic vehicles with an omni-directional drive like the KUKA omniRob shown in Figure 16.1a are appealing for industrial applications like mobile manipulators, fork lifts, or transportation vehicles [Sprunk *et al.*, 2011]. While differential or synchro drives allow for moving in any given direction by performing an appropriate turn on the spot, omni-directional platforms can rotate and translate independently and simultaneously. This additional degree of freedom greatly enlarges the space of possible trajectories: the path of a differential drive vehicle can be modeled by a two-dimensional curve that specifies its position over time. In this case, the orientation is implicitly defined by the tangent of the curve. In contrast, paths of an omni-directional platform require three-dimensional splines.

However, a straightforward extension of our path model to 3D would not be sufficient. A non-circular robot shape can for example require straight motion without rotation in narrow spaces as shown in Figure 16.1b, which can not directly be modeled with interpolating splines that continuously rotate the robot. Therefore, Sprunk *et al.* proposed a novel extension to our path model that introduces additional rotation control points [2011]. This enables the system to confine the rotation to the waypoints for narrow passages or initial paths as shown in Figure 16.1b, but also to gradually distribute the rotation on the trajectory segments (Figure 16.1c). Technically, this is



(a) KUKA omniRob (b) Initial path (slow) (c) Optimized for time of travel (fast)

Figure 16.1: Omni-directional robots like the omniRob can translate and rotate independently and simultaneously. The initial path only executes turns on the spot while the optimized one exploits this capability to reduce the time of travel.

achieved by subdividing the splines at the rotation control points and inserting special segments with constant orientation.

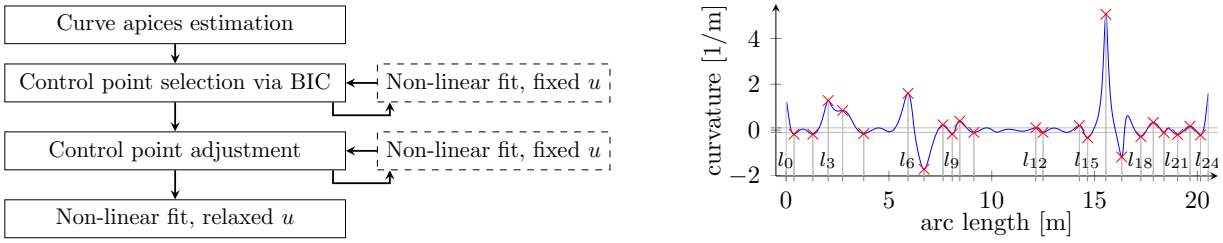
The energy consumption of omni-directional robots can depend on the direction of travel, e.g., forward motion can be more efficient due to the wheel construction. By exchanging the cost function, the system can optimize the trajectories for different criteria, e.g., travel time or energy efficiency. This greatly affects if the resulting trajectories let the platform rotate while moving or prefer forward motion.

The holonomic motion planning system was successfully evaluated in several industrial scenarios [Sprunk *et al.*, 2011] and demonstrated at the AUTOMATICA 2010 and IROS 2011 conferences.

16.2 Teaching Paths to Mobile Robots

In industrial applications, it is often desired to predefine robot trajectories instead of relying on an autonomous navigation system, e.g., to maximize predictability in work spaces shared with humans. To make such a system programmable by non-experts without changing the environment as required with physical path markers, programming by demonstration is an appealing approach. At the same time, the robot should still be able to react online to unmapped obstacles and optimize the learned trajectory.

Sprunk *et al.* proposed a system for trajectory teaching by demonstration with a joystick based on non-linear least-squares fitting. They discuss a trade-off concerning the number of parameters



(a) System overview

(b) Estimation of curve apices

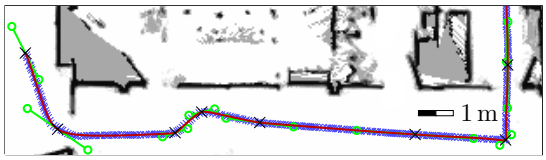
Figure 16.2: Multi-stage path fitting and optimization method by Sprunk *et al.* using our path model. It places control points in curve apices identified by the curvature of a fitted septic spline. In multiple steps, the control points are optimized to reduce the residual fitting error.

in the model of the trajectory shape: a high number allows for high accuracies when fitting the trajectory to the teaching data but at the same time strongly increases the required computational resources during path optimization [Sprunk *et al.*, 2012].

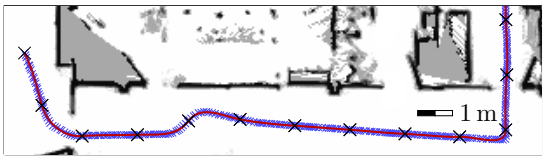
To achieve high accuracies with a smaller number of parameters than standard approaches, Sprunk *et al.* use our path model presented in Chapter 12 in combination with a novel multi-stage fitting and optimization method shown in Figure 16.2a. In a first step, the system estimates curve apices using a linear least-squares fit of generic septic splines as shown in Figure 16.2b. By evaluating the Bayesian Information Criterion (BIC) in combination with a non-linear least-squares fit of our path model, a subset of these points is selected. The control point adjustment optimizes the locations of control points on the fitted trajectory, i.e., where they are anchored to the data points. In a final step, the trajectory is fitted to the data with relaxed correspondences of these anchors.

The fitting procedure called in all steps optimizes the shape of the curve by adjusting the position and tangent elongation of each control point. Thus, it uses the same parameters as the motion planning approach presented in this thesis. The optimization is a non-linear least-squares optimization that can be provided with a good initial guess from a linear least-squares fit of a basic quintic Bézier spline.

Figure 16.2b shows fitting results for an example trajectory using our path model and basic cubic splines in comparison. The cubic splines require about twice the number of parameters to achieve similar residual fitting errors. For a more thorough analysis on many trajectories, please refer to [Sprunk *et al.*, 2012].



(a) Non-linear spline fit using our path model
8 segments, 23 parameters, $e = 0.007\,m$



(b) Basic cubic spline fitting
13 segments, 50 parameters, $e = 0.008\,m$

Figure 16.3: Fitting result using our path model (a) compared to basic cubic spline fitting with twice as many parameters (b). Despite the substantially reduced number of parameters, the residual error e achieved by Sprunk *et al.* is slightly lower compared to the standard approach. Blue crosses denote the teaching data, black crosses and green circles the control points, and the red line the fitted trajectory.

17 Conclusion

This part of the thesis presented an approach to kinodynamic motion planning for wheeled mobile robots. It is suitable for platforms that can turn on the spot, e.g., service robots with differential, synchro, or omnidirectional drives.

Our approach is based on a novel path model that uses quintic Bézier splines to represent the desired path of the robot in a parametric fashion. In contrast to cubic spline methods, our model generates and optimizes paths that are curvature continuous from start to end, while the start curvature is a free parameter. This facilitates the modification of existing trajectories by smoothly attaching new segments somewhere on the path, for example to avoid unexpected obstacles or extend the trajectory currently followed by the robot. This replanning aspect also allows for graceful recovery in the case of localization discontinuities, and accounts for the accumulation of odometry errors. Using our model, the shape of a path is controlled by the position of waypoints and tangent lengths, i.e., the magnitude of the first derivative at the waypoints. The other degrees of freedom, namely the tangent orientations and the second derivatives, are controlled by heuristics that mimic the behavior of cubic splines. Thus, our path model combines the potential of higher-order splines with the advantage of a small number of parameters during optimization.

To represent the position and velocity of the robot as a function of time, our path representation is augmented with velocity profiles that specify the velocity of the robot as a function of arc length. We proposed a three-step method to efficiently compute velocity profiles that minimize the traveling time for a given path while satisfying interdependent platform constraints on velocities and accelerations. The resulting trajectories can be executed with high accuracy using an error-feedback controller in a small control loop.

The input to our algorithm is a set of waypoints that represent a valid piecewise linear path. These waypoints can for example be generated using a map of the environment and a global path planner like A* or Rapidly-exploring Random Trees. From these waypoints, we instantly create an initial parametric trajectory that closely approximates the piecewise linear path. By altering the path shape in an optimization process, our method minimizes the required time to reach the goal, mainly by widening curves and increasing the robot's distance to obstacles.

Since the unoptimized initial trajectory is already traversable, our optimization method is able to provide a current best solution at any time. Additionally, this maintains the completeness property of the employed waypoint planner. Together with the possibility of replanning existing trajectories while maintaining curvature continuity for the whole path, these features make up the novelty of our method with respect to previous approaches.

We implemented and tested our approach on real robots in complex and populated environments. Our experiments show that our system plans motion trajectories that can be executed precisely, i.e., with very low deviations from planned positions and velocities. Additionally, it can effectively replan if the robot encounters unexpected obstacles, for example in populated environments. Finally, as the movements are planned ahead in time, the robot can predict its own motion with high precision. Extensions and applications of our approach by other authors comprise smooth coverage planning, trajectory generation for omni-directional platforms, and teaching trajectories to a robot by demonstration.

Part III

Laser-Based Tracking of People in Groups

18 Introduction and Related Work

The ability of robots to keep track of people in their surrounding using on-board sensors is fundamental for a wide range of robotic applications, such as personal and service robots, intelligent cars, crowd control, or surveillance. Most traditional approaches to people tracking start with the automatic detection of people in sensor data, for example, by classifying clusters of laser range readings as belonging to a person or not. The actual tracking routine then establishes a correspondence between observations of the same person from different time steps. The observations are usually grouped and stored as individual person tracks, together with a track state that estimates the current position and velocity of each person.

Especially in crowded areas, determining the correct data associations is often a key problem in people tracking: people are social beings and tend to form groups, interact with each other, merge to larger groups, or separate from groups. Tracking individual people in these formation processes can be hard due to the high chance of occlusion and the large extent of data association ambiguity. This causes the space of possible associations to become huge and the number of possible assignment histories quickly become intractable. For many applications, however, knowledge about groups can be sufficient as the task does not require to know the state of each individual. In such situations, tracking groups that consist of multiple people is more efficient. Additionally, it reveals semantic information about activities and social relations of people.

This part of the thesis focuses on group tracking in populated environments with the goal to track a large number of people in real-time. The approach attempts to maintain the state of groups of people over time, considering possible splits and merges as shown in Figure 18.1. For our experiments we use a mobile robot equipped with a laser range finder, but our method should be applicable to data from other sensors as well.

We propose a tracking system for groups of people using an extended Multi-Hypothesis Tracking (MHT) approach to hypothesize over both, the group formation process (models) and the association of observations to tracks (assignments). Each model, defined to be a particular partitioning of tracks into groups, creates a new tree branch with its own assignment problem. As a further contribution we propose a group representation that includes the shape of the group, and we show how this representation is updated in each step of the tracking cycle. This extends previous approaches to group tracking where groups are assumed to have Gaussian shapes only [Gennari and Hager, 2004; Mucientes and Burgard, 2006]. The group tracker proposed

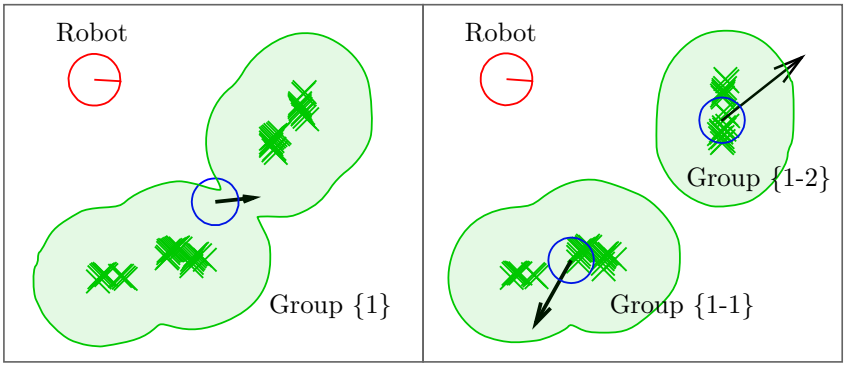


Figure 18.1: Tracking groups of people with a mobile robot. Groups are shown by their position (blue), velocity (black), the associated laser points (green), and a contour for visualization. In the two frames, a group of four people splits up into two groups with two people each.

here also estimates the number of people in groups and employs a labeling system to represent the history of group interactions.

Finally, we use the psychologically motivated *proxemics* theory introduced by Hall [1974] for the definition of a group. The theory relates social relation and body spacing during social interaction and proposes thresholds that separate the intimate, personal, social, and public space around people.

In most of the related work on laser-based people tracking, tracks correspond to individual people [Kluge *et al.*, 2001; Fod *et al.*, 2002; Schulz *et al.*, 2003; Cui *et al.*, 2005; Zajdel *et al.*, 2005]. Taylor and Kleeman [2004] and Arras *et al.* [2008] represent tracks by the state of legs which are fused to people tracks in a later stage. Khan *et al.* [2006] proposed an MCMC-based tracker that is able to deal with non-unique assignments, i.e., measurements that originate from multiple tracks, and multiple measurements that originate from the same track. Actual tracking of groups using laser range data was, to our knowledge, first addressed by Mucientes and Burgard [2006]. Most research in group tracking was carried out in the vision community [McKenna *et al.*, 2000; Gennari and Hager, 2004; Bose *et al.*, 2007]. Gennari and Hager as well as Bose *et al.* both address the problem of target fragmentation (splits) and grouping (merges), but do not integrate data association decisions over time. This, however, is a key property of the Multi-Hypothesis Tracking (MHT) approach, which was initially presented by Reid [1979] and later extended by Cox and Hingorani [1996]. The MHT framework belongs to the most general data association techniques as it produces joint compatible assignments, integrates them over time, and is able to deal with track creation, matching, occlusion, and deletion.

The previous approaches closest to this work are by Mucientes and Burgard [2006] and Joo

and Chellappa [2007]. Both address the problem of group tracking using an MHT approach. Mucientes and Burgard employ two separate MHTs, one for the regular association problem between observations and tracks, and a second stage MHT that hypothesizes over group merges. In their approach, however, people tracks are not replaced by group tracks, hence there is no gain in efficiency. Thus, the main benefit is the additional semantic information about the formation of groups.

Joo and Chellappa [2007] present a vision-based group tracker using a single MHT to create hypotheses of group splits and merges and observation-to-track assignments. They develop a variant of Murty's algorithm [Murty, 1968] that generates the k-best *non-unique* assignments which enables them to make multiple assignments between observations and tracks, thereby describing target splits and merges. However, the method only produces an approximation of the optimal k-best solutions since the posterior hypothesis probabilities depend on the number of splits, which, at the time when the k-best assignments are being generated, is unknown. In our approach, the split, merge and continuation events are given by the model *before* computing the assignment probabilities, and therefore, our k-best solutions are optimal.

This thesis part is structured as follows: after a review of related work, Chapter 19 introduces our definition of groups, describes the detection of groups in laser range data, and introduces group tracks. Chapter 20 describes the cycle of our Kalman filter-based group tracker, the generation of group formation models, and how their probabilities are computed. Chapter 21 describes the experimental results, before Chapter 22 concludes this part.

19 Group Detection and Group Tracks

This chapter defines the concept of a group, describes the detection of groups in range data, specifies the representation and initialization of group tracks, and derives the probabilities of group-to-observation assignments and group-to-group assignments.

What makes a collection of people a group is a highly complex question in general, which involves social relations among subjects that are difficult to measure. A concept related to this question is the proxemics theory introduced by Hall [1974]. He found from a series of psychological experiments that social relations among people are reliably correlated with physical distance during interaction. This finding allows us to infer group affiliations by means of body spacing information available in the range data. Concretely, the clustering threshold d_P used in the following corresponds to a threshold for inter-person distances in the context of group detection.

19.1 Group Detection in Range Data

Detecting people in range data has been approached with motion and shape features [Kluge *et al.*, 2001; Fod *et al.*, 2002; Schulz *et al.*, 2003; Cui *et al.*, 2005; Zajdel *et al.*, 2005; Mucientes and Burgard, 2006] as well as with a learned classifier using boosted features [Arras *et al.*, 2007]. However, these systems were designed (or trained) to extract single people. In the case of densely populated environments, groups of people typically produce large blobs in which individuals are hard to recognize. We therefore pursue the approach of background subtraction and clustering. Given a previously learned model such as a map of the environment, the background is subtracted from the scans and the remaining points are passed to the clustering algorithm. This approach is also able to detect standing people as opposed to the work of Mucientes and Burgard [2006] which relies on motion features. Note that the detection method is not critical to the system and could also be replaced by map-free approaches that employ appearance information, motion features, or other filtering techniques.

Concretely, a laser scanner generates measurements consisting of bearing and range values. The measurements are transformed into Cartesian coordinates $\mathbf{z}_l = (x_l, y_l)^T$ and grouped using *single linkage clustering* with a distance threshold d_P [Hartigan, 1975]. The outcome is a set of

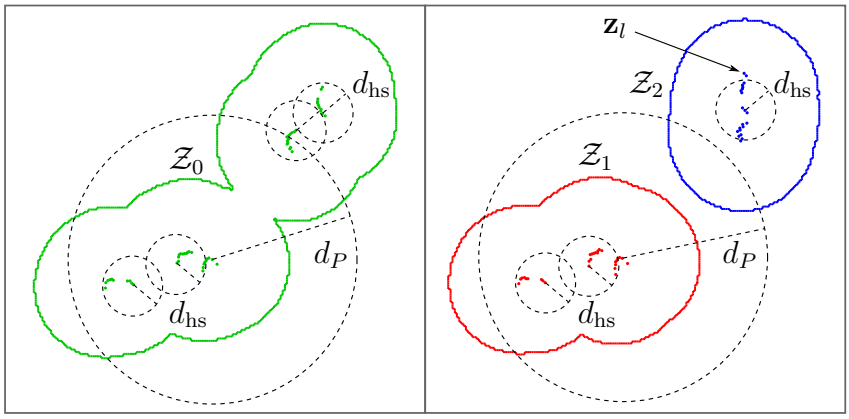


Figure 19.1: Illustration of the detection step. *Left:* One group is detected since all shortest links between the measured points $\mathbf{z}_l$ are smaller than the single-linkage clustering threshold d_P. *Right:* Two groups are found since the shortest link between their points exceeds d_P. To estimate the group sizes, we determine the number of human-sized blobs in a group by applying the same clustering procedure with threshold d_{hs}.

clusters $\mathcal{Z}_i$ making up the current observation set $\mathcal{Z}(k) = \{\mathcal{Z}_i \,|\, i = 1, \ldots, N_{\mathcal{Z}}\}$. Each cluster $\mathcal{Z}_i$ is a complete set of measurements $\mathbf{z}_l$ that fulfills the cluster condition, i.e., two clusters are joined if the distance between their closest points is smaller than d_P. This grouping is equivalent to the formulation by Gennari and Hager [2004], which considers the connected components of a graph that contains all measurements $\mathbf{z}_l$ as vertices and an edge between each pair of measurements whose distance is smaller than d_P. Depending on the cluster distance d_P, the clustering can attempt to group range readings that correspond to single legs, individual people, or groups of people.

Even though tracking of individuals in groups is often not feasible due to frequent occlusions, the number of detected individuals in a group correlates with the true number of people in a group. As an observation of the group size, we therefore take the number of human-sized clusters $n_{\text{hs}}(\mathcal{Z}_i)$ found in an observation cluster $\mathcal{Z}_i$. We determine this by counting the clusters after reapplying single linkage clustering to the points in $\mathcal{Z}_i$ with an appropriate distance threshold d_{hs}, with $d_{\text{hs}} < d_P$.

An example for the clustering is given in Figure 19.1. On the left, all links are shorter than d_P so that the measurements are grouped into one cluster $\mathcal{Z}_0$ that contains four human-sized clusters. On the right, the shortest distance between the two groups exceeds d_P so that they are kept as two clusters, $\mathcal{Z}_1$ and $\mathcal{Z}_2$. The two people in $\mathcal{Z}_2$ are counted as only one human-sized cluster.

19.2 Representation and Initialization of Group Tracks

We represent a group track as a tuple $G = \langle \mathbf{x}, C, \mathcal{P}, \mathcal{L} \rangle$ with the track state $\mathbf{x}$, the state covariance matrix C, the set of contour points $\mathcal{P}$ belonging to G, and the set of identification labels $\mathcal{L}$. The track state vector $\mathbf{x} = (x,\, y,\, \dot{x},\, \dot{y}, n)^T$ is composed of the position $(x,\, y)^T$ and Cartesian velocity $(\dot{x},\, \dot{y})^T$ of the group, and n, the number of people in the group.

The points $\mathbf{x}_{\mathcal{P}_l} \in \mathcal{P}$ are an approximation of the current shape or spatial extension of the group. Shape information will be used for data association under the assumption of *instantaneous rigidity*. That is, a group is assumed to be a rigid object over the duration of a time step Δt, and consequently, all points in $\mathcal{P}$ move coherently with the estimated group state $\mathbf{x}$. The points $\mathbf{x}_{\mathcal{P}_l}$ are represented in Cartesian coordinates relative to the state $\mathbf{x}$.

The label set $\mathcal{L}$ contains identification labels that are associated with the group. These labels explicitly represent the history of track interactions, which can be of high interest for certain applications, e.g., to determine which people belong together.

If the tracker creates a new group track G_j from an observation cluster $\mathcal{Z}_i$ in time step k, the positional components $(x_j, y_j)^T$ of track state $\mathbf{x}_j(k|k)$ are initialized with the centroid position of the measurement cluster. The contour points $\mathcal{P}_j$ are the points in $\mathcal{Z}_i$ represented relative to the centroid (omitting the time index $(k|k)$ for readability):

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} := \bar{\mathbf{z}}_i = \frac{1}{|\mathcal{Z}_i|} \sum_{\mathbf{z}_l \in \mathcal{Z}_i} \mathbf{z}_l \;, \quad \mathcal{P}_j := \bigcup_{\mathbf{z}_l \in \mathcal{Z}_i} \mathbf{z}_l - \bar{\mathbf{z}}_i \;. \tag{19.1}$$

The unobserved velocity components $(\dot{x}_j, \dot{y}_j)^T$ of $\mathbf{x}$ are set to zero, the size estimate is set to the number of human-sized blobs in the measurement cluster, $n_j := n_{\text{hs}}(\mathcal{Z}_i)$, and the label set is assigned a unique number as its only element, e.g., $\mathcal{L}_j := \{0\}$ for the first group after starting up the tracker. The initial state covariance is given by $C_j = C_0$, where C_0 is a diagonal matrix with $({\sigma_x}^2, {\sigma_y}^2, {\sigma_{\dot{x}}}^2, {\sigma_{\dot{y}}}^2, {\sigma_n}^2)$ being the elements on the main diagonal. To account for the unknown components in the initial state vector, high uncertainty values are used for the corresponding entries in the initial state covariance matrix.

19.3 Motion Model for Group Tracks

To track groups over time, the state $\mathbf{x}(k|k)$ and state covariance $C(k|k)$ of each group track in time step k are predicted into the next time step using a motion model. The predictions are denoted as $\mathbf{x}(k{+}1|k)$ and $C(k{+}1|k)$, respectively. For tracks that are *continued*, i.e., no splits or merges take place from one frame to the next, we assume constant velocity for the centroid of the group, and a constant number of people in the group. Using a linear Kalman filter, we get

$$\mathbf{x}(k+1|k) = A\,\mathbf{x}(k|k) \tag{19.2}$$

$$C(k+1|k) = A\,C(k|k)\,A^T + Q \tag{19.3}$$

for the state prediction. The state transition matrix A and the process noise covariance matrix Q are given by

$$A = \begin{pmatrix} 1 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \; Q = \begin{pmatrix} {\epsilon_x}^2 & 0 & 0 & 0 & 0 \\ 0 & {\epsilon_y}^2 & 0 & 0 & 0 \\ 0 & 0 & {\epsilon_{\dot{x}}}^2 & 0 & 0 \\ 0 & 0 & 0 & {\epsilon_{\dot{y}}}^2 & 0 \\ 0 & 0 & 0 & 0 & {\epsilon_n}^2 \end{pmatrix}. \tag{19.4}$$

The entries of Q reflect the acceleration capabilities of a typical human. The noise for the number of people in the group, controlled by ϵ_n, accounts for people joining or leaving the group without being noticed. The actual noise values used in our experiments are given in Section 21.

As mentioned before, we assume instantaneous rigidity for the shape of a group. Since the points in $\mathcal{P}$ are relative coordinates with respect to the moving centroid, the point set remains unchanged, and thus $\mathcal{P}(k+1|k) = \mathcal{P}(k|k)$.

If two observations can be associated with a group track G, i.e., they both fall into the validation gate of G, the tracker can consider to *split* the track into two new tracks according to an interaction model (see Section 20.1). Since the actual partitioning in the split is unknown at this stage, two new predicted group tracks G_1 and G_2 are created by duplicating the predicted state and covariance of G. The same applies for the point set $\mathcal{P}$ and the label set $\mathcal{L}$. To make the label sets unique, we attach different indices to the label, e.g., a group with label set $\{0\}$ would split up into two groups with label sets $\{0-0\}$ and $\{0-1\}$. Again, the component of the state that represents the number of people in the group, n, is treated differently: the sum of people in the

resulting groups must be equal to the original number of people. However, the actual partitioning is not known in the prediction step. Therefore, we use $n_1 = n_2 = n/2$, and reinitialize the state covariances of the new split tracks with C_0.

If the tracker considers to *merge* two group tracks G_i and G_j according to a track interaction model, the track prediction has to be computed accordingly. The predicted set of contour points of the merged group is the union of the two former point sets, $\mathcal{P}_{ij} = \mathcal{P}_i \cup \mathcal{P}_j$. The track states $\mathbf{x}_i$ and $\mathbf{x}_j$ of the merging group track represent the position and velocity of the centroids of the groups. Thus, the state of the merged track, $\mathbf{x}_{ij}$, is computed as the weighted mean of the original track states, using weights proportional to the number of points in the merging sets $\mathcal{P}_i$ and $\mathcal{P}_j$. The tracks before the merge are assumed to be independent. According to the summation and scaling laws for covariances, the covariance matrix of the merging track is the weighted mean of the original covariances with squared weights,

$$\mathbf{x}_{ij} = w_i \cdot \mathbf{x}_i + w_j \cdot \mathbf{x}_j \tag{19.5}$$

$$C_{ij} = {w_i}^2 \cdot C_i + {w_j}^2 \cdot C_j \,, \tag{19.6}$$

where $w_i = |\mathcal{P}_i|/|\mathcal{P}_{ij}|$ and $w_j = |\mathcal{P}_j|/|\mathcal{P}_{ij}|$. Note that this applies only for the first four components of $\mathbf{x}_{ij}$ and the upper-left $4{\times}4$ block of C_{ij}. The fifth component, namely the group size n_{ij}, is excluded, since the number of people in the merging groups naturally add up to $n_{ij} := n_i + n_j$. Consequently, the corresponding uncertainty values are summed up as well. Finally, the label set of the new group is the union of the label sets of the original tracks, $\mathcal{L}_{ij} = \mathcal{L}_i \cup \mathcal{L}_j$. To remove redundant labels, an optional pruning can be done in this step: whenever all tracks that resulted from a split have merged again, the additional indices added in the split step can be removed, e.g., when the groups with labels $\{0-0\}$ and $\{0-1\}$ merge, they can be labeled $\{0\}$ again. Although this can remove split and merge events from the history represented by the labeling, it keeps the semantic information consistent.

19.4 Group-to-Observation Assignment Probability

For data association we need to calculate the probability that an observed cluster $\mathcal{Z}_i$ belongs to a predicted group $G_j = \langle\, \mathbf{x}_j(k+1|k),\, C_j(k+1|k),\, \mathcal{P}_j,\, \mathcal{L}_j \,\rangle$. Therefore, we are looking for a distance function $d(\mathcal{Z}_i, G_j)$ that, unlike the Mahalanobis distance used by Mucientes and Burgard [2006], accounts for the shape of the observation cluster $\mathcal{Z}_i$ and the contour $\mathcal{P}_j$ of the group, rather than just for their centroids. To this end, we use a variant of the Hausdorff distance.

As the regular Hausdorff distance is the *longest* distance between points on two contours, it tends to be too sensitive to large variations in depth that can occur in range data. This motivates the use of the minimum average Hausdorff distance as defined by Dubuisson and Jain [1994]. It computes the minimum of the contour distance from $\mathcal{Z}_i$ to $\mathcal{P}_j$ and from $\mathcal{P}_j$ to $\mathcal{Z}_i$,

$$d_{\text{HD}}(\mathcal{Z}_i, G_j) = \min\left\{d(\mathcal{Z}_i, \mathcal{P}_j), d(\mathcal{P}_j, \mathcal{Z}_i)\right\}, \tag{19.7}$$

where $d(\mathcal{Z}_i, \mathcal{P}_j)$ is the directed average Hausdorff distance from $\mathcal{Z}_i$ to $\mathcal{P}_j$,

$$d(\mathcal{Z}_i, \mathcal{P}_j) = \frac{1}{|\mathcal{Z}_i|} \sum_{\mathbf{z}_l \in \mathcal{Z}_i} \min_{\mathbf{x}_{\mathcal{P}_j} \in \mathcal{P}_j} \left\{D(\nu_{lj},\, S_{lj})\right\}. \tag{19.8}$$

Since we deal with uncertain entities, we calculate the distance $d(\mathcal{Z}_i, \mathcal{P}_j)$ using the Mahalanobis distance

$$D(\nu_{lj}, S_{lj}) = \sqrt{{\nu_{lj}}^T\, S_{lj}^{-1}\, \nu_{lj}}, \tag{19.9}$$

with ν_{lj} being the innovation and S_{lj} being the innovation covariance between a point $\mathbf{z}_l \in \mathcal{Z}_i$ and contour point $\mathbf{x}_{\mathcal{P}_j}$ of the predicted set $\mathcal{P}_j$ transformed into the sensor frame. More precisely, these two terms are given as

$$\nu_{lj} = \mathbf{z}_l - (H\mathbf{x}_j(k+1|k) + \mathbf{x}_{\mathcal{P}_j}) \tag{19.10}$$

$$S_{lj} = H\, C_j(k+1|k)\, H^T + R_l, \tag{19.11}$$

where $H = \left(\begin{smallmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{smallmatrix}\right)$ is the measurement Jacobian and R_l the 2×2 observation covariance whose entries reflect the noise in the measurement process of the range finder.

The probability that cluster $\mathcal{Z}_i$ originates from group track G_j is finally given by a zero-centered Gaussian,

$$\mathcal{N}_i = \frac{1}{2\pi\sqrt{\det(S_{lj})}} \exp\left(-\tfrac{1}{2} d^2_{\text{HD}}(\mathcal{Z}_i,\, G_j)\right). \tag{19.12}$$

19.5 Group-to-Group Assignment Probability

To determine the probability that two groups G_i and G_j merge, we compute the distance between their closest contour points in a Mahalanobis sense. In doing so, we have to account for the clustering distance d_P, since we consider G_i and G_j to be one group as soon as their contours come closer than d_P. Let $\Delta\mathbf{x}_{\mathcal{P}_{ij}} = \mathbf{x}_{\mathcal{P}_i} - \mathbf{x}_{\mathcal{P}_j}$ be the vector difference of two contour points of G_i and G_j, respectively. We then subtract d_P from $\Delta\mathbf{x}_{\mathcal{P}_{ij}}$ unless $\Delta\mathbf{x}_{\mathcal{P}_{ij}} \leq d_P$ for which

$\Delta\mathbf{x}_{\mathcal{P}_{ij}} = \mathbf{0}$. Concretely, the modified difference becomes $\Delta\mathbf{x}'_{\mathcal{P}_{ij}} = \max(\mathbf{0},\ \Delta\mathbf{x}_{\mathcal{P}_{ij}} - d_P\,\mathbf{u}_{\mathcal{P}_{ij}})$ where $\mathbf{u}_{\mathcal{P}_{ij}} = \Delta\mathbf{x}_{\mathcal{P}_{ij}}/|\Delta\mathbf{x}_{\mathcal{P}_{ij}}|$.

To obtain a similarity measure that accounts for nearness of group contours *and* similar velocity, we augment $\Delta\mathbf{x}'_{\mathcal{P}_{ij}}$ by the difference in the velocity components,

$$\Delta\mathbf{x}^*_{\mathcal{P}_{ij}} = (\Delta\mathbf{x}'^T_{\mathcal{P}_{ij}},\ \dot{x}_i - \dot{x}_j\,,\ \dot{y}_i - \dot{y}_j)^T. \tag{19.13}$$

We now determine the statistical compatibility of two groups G_i and G_j according to the four-dimensional minimum Mahalanobis distance

$$d^2_{\text{min}}(G_i,\, G_j) = \min_{\substack{\mathbf{x}_{\mathcal{P}_i}\in\mathcal{P}_i \\ \mathbf{x}_{\mathcal{P}_j}\in\mathcal{P}_j}} \left\{ D^2(\Delta\mathbf{x}^*_{\mathcal{P}_{ij}},\, C_i{+}C_j) \right\}. \tag{19.14}$$

The probability that two groups actually belong together, is finally given by

$$\mathcal{N}_{ij} = \frac{1}{(2\pi)^2\sqrt{\det(C_i{+}C_j)}} \exp\left(-\tfrac{1}{2}d^2_{\text{min}}(G_i,\, G_j)\right). \tag{19.15}$$

In this formulation, only the upper-left 4×4 blocks of C_i and C_j are used, which excludes the group size estimate and the corresponding uncertainties from the group-to-group assignment probabilities.

20 Multi-Model MHT

In this chapter we describe our multi-model multi-hypothesis tracking approach to group tracking. It is based on the original MHT by Reid [1979], which hypothesizes about data association of measurements to tracks. There, a hypothesis Ω^k in time step k specifies a number of tracks being present at that time, and includes an assignment of current observations to these tracks. The tracks are usually realized as individual Kalman filters. They are updated using the assigned observations and predicted into the next time step. In most cases, the data association is ambiguous, which gives rise to multiple hypotheses branching out from every parent hypothesis in the previous time step. Each hypothesis is associated with a probability, that depends on the likelihood of the observations given the assignment and the probability of the parent hypotheses. For details, please refer to Reid [1979] and Cox and Hingorani [1996].

In addition to the data association hypotheses, our tracker also hypothesizes about group formation models described below. Therefore, the multi-model MHT introduces an intermediate tree level for each time step, on which models for track continuation, merging, or splitting spring off from parent hypotheses.

20.1 Model Generation and Model Probability

A model is defined as a partitioning of tracks into groups, and assumes a particular state of the group formation process. New models, whose generation is described in this section, hypothesize about the evolution of that state. As this happens recursively, that is, based on a model of the previous time step, the problem can be seen as a recursive clustering problem.

The space of possible model transitions is large since each group track can split into an unknown number of new tracks, or merge with an unknown number of other tracks. We therefore impose the following gating conditions for observations and tracks, thus implementing a data-driven aspect into the model generation step.

- Multiple group tracks G_i can merge into one track only if there is an observation which is statistically compatible with all G_i according to χ^2 tests.

- A group track can only split into multiple tracks that are all matched with observations in that very time step. Splits into occluded or obsolete tracks are not allowed.

Gating and statistical compatibility are both determined on a significance level α, using the minimum average Hausdorff distance where applicable.

We further limit the possible number of model transitions as we assume that merge and split are binary operators. More precisely, we assume:

- At most two group tracks G_i, G_j can merge into one track at the same time.
- A track G_i can split at most into two tracks in one frame.
- A group track can not be involved in a split and a merge action at the same time.

These limitations are justified by the assumption that we observe the world much faster than the rate with which it evolves. This fact alleviates the impact of violations of the above assumptions: even if, for instance, a group splits into three subgroups at once, the tracker requires only two cycles to reflect this change.

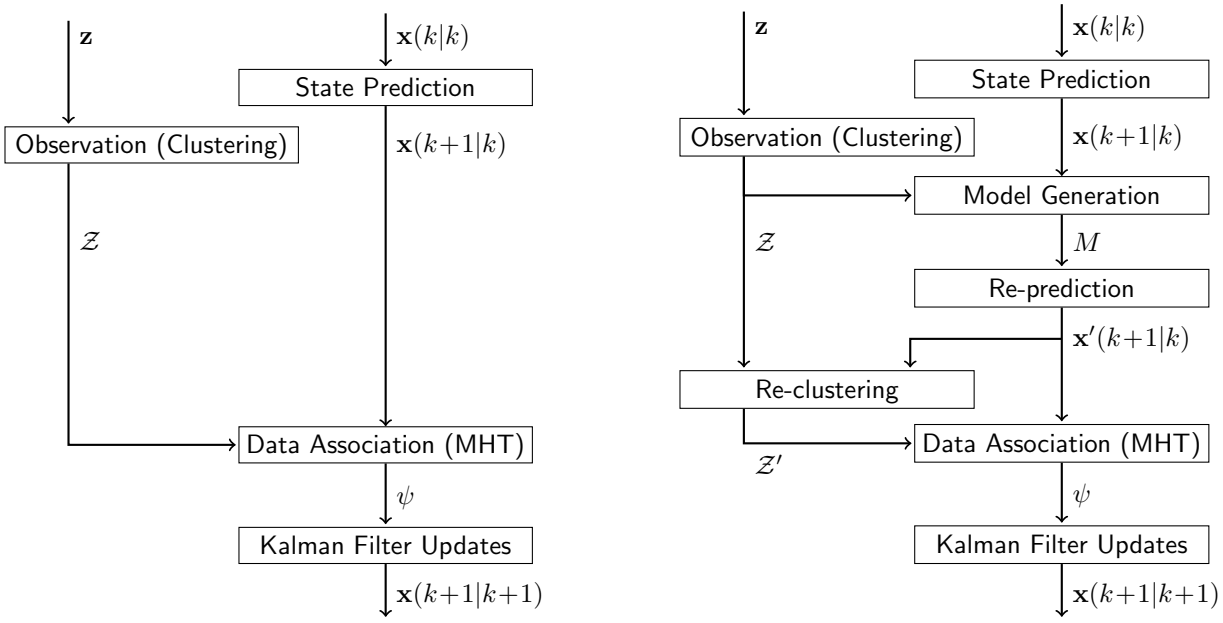
A new model defines for each group track if it is continued, split, or merged with another group track. The probability of a model is calculated using the constant prior probabilities for continuations and splits, p_C and p_S respectively, and the probability for a merge between two tracks G_i and G_j as $p_G \cdot \mathcal{N}_{ij}$. The latter term consists of a constant prior probability p_G and the group-to-group assignment probability $\mathcal{N}_{ij}$ defined in Section 19.5. Let N_C and N_S be the number of continued tracks and the number of split tracks in model M respectively, then the probability of M conditioned on the parent hypothesis Ω^{k-1} is

$$P(M|\Omega^{k-1}) = p_C^{N_C} \cdot p_S^{N_S} \prod_{G_i, G_j \in \Omega^{k-1}} (\, p_G \cdot \mathcal{N}_{ij})^{\delta_{ij}} \tag{20.1}$$

with δ_{ij} being 1 if G_i, G_j merge and 0 otherwise.

20.2 Tracking Cycle

This section describes the steps in the cycle of our Kalman filter-based group tracker, which are also visualized by the flow diagram in Figure 20.1. The structure differs from a regular tracker



(a) Regular Multi-Hypothesis Tracker (MHT)

(b) Multi-Model Multi-Hypothesis Tracker

Figure 20.1: Tracking cycle of the proposed Multi-Model Multi-Hypothesis Tracker in comparison to a regular MHT system. The additional steps *model generation*, *re-prediction*, and *re-clustering* were introduced to track the group formation process.

in the additional steps *model generation*, *track re-prediction*, and *re-clustering*.

- *State prediction:* In this step, the states of all existing group tracks are predicted under the assumption that they continue without interacting with other tracks, i.e., without splits or merges. See Section 19.3 for details.

- *Observation:* As described in Section 19.1, this step involves grouping the laser range data into clusters $\mathcal{Z}_i$.

- *Model Generation:* Group formation models are generated based on the predicted group tracks and the clusters $\mathcal{Z}_i$, see Section 20.1.

- *Re-prediction:* Based on the model hypotheses that postulate a split, merge, or continuation event for each track, groups are re-predicted using these hypotheses so as to reflect the respective model, as explained in Section 19.3.

- *Re-clustering:* Re-clustering an observed cluster $\mathcal{Z}_i$ is necessary when it might have been produced by more than one group track, that is, it is in the gate of more than one track. If

the model hypothesis postulates a merge for the involved tracks, nothing needs to be done. Otherwise, $\mathcal{Z}_i$ needs to be re-clustered to prevent under-segmentation, for example if two different groups are passing closely. The re-clustering is done using a nearest-neighbor rule: those points $\mathbf{z}_l \in \mathcal{Z}_i$ that share the same nearest neighbor track are combined to a new cluster. This step follows from the uniqueness assumption, which is common in target tracking and according to which an observation can only be produced by a single target.

- *Data Association (MHT):* This step involves the generation, probability calculation, and pruning of data association hypotheses that assign re-predicted group tracks to re-clustered observations. See Sections 20.3 to 20.5.

- *Kalman Filter Updates:* Each group track G_j that has been assigned to a cluster $\mathcal{Z}_i$ is updated with a linear Kalman filter. We use an observation vector $\tilde{\mathbf{z}}_i = (\bar{\mathbf{z}}_i, n_{\text{hs}}(\mathcal{Z}_i))^T$, that contains both the centroid position $\bar{\mathbf{z}}_i$ of $\mathcal{Z}_i$ and the number of human-sized blobs $n_{\text{hs}}(\mathcal{Z}_i)$ in the cluster. The update is then given by

$$\mathbf{x}(k+1|k+1) = \mathbf{x}(k+1|k) + K\Big(\tilde{\mathbf{z}}_i - \tilde{H}\mathbf{x}(k+1|k)\Big) \tag{20.2}$$

$$C(k+1|k+1) = C(k+1|k) - K\tilde{H}\,C(k+1|k) \tag{20.3}$$

with K being the Kalman gain matrix and $\tilde{H}$ the corresponding measurement Jacobian,

$$K = C(k+1|k)\cdot\ \tilde{H}^T\Big(\tilde{H}C(k+1|k)\tilde{H}^T + R_l\Big)^{-1} \tag{20.4}$$

$$\tilde{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{20.5}$$

The contour points in $\mathcal{P}_j$ are replaced by the points in $\mathcal{Z}_i$ after being transformed into the reference frame of the posterior state $\mathbf{x}(k+1|k+1)$, as described in Section 19.2. Thereby, $\mathcal{P}_j$ always contains the most recent approximation of the group.

20.3 Data Association

Let Ω^{k-1} be a hypothesis from the previous time step $k-1$ that postulates a number of existing group tracks. After predicting their states for time step k, a number of group formation models are generated that represent differing hypotheses about which tracks continue, split up, or merge.

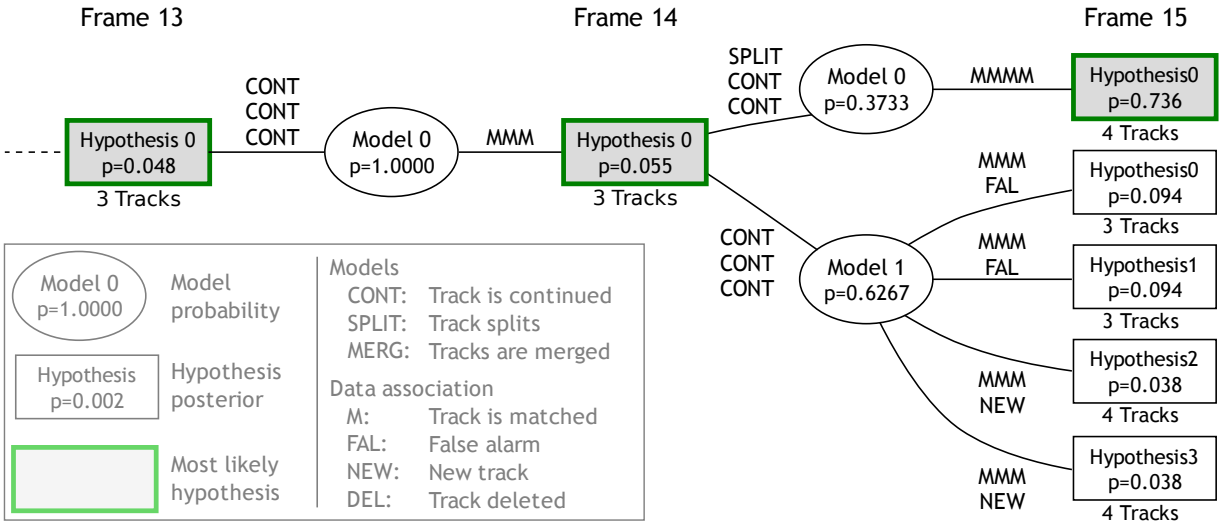


Figure 20.2: The proposed multi-model MHT. For each parent hypothesis, model hypotheses (ellipses) branch out and create their own assignment problems. In our application, models define which tracks of the parent hypothesis are continued, split, or merged. The tree shows frames 13 to 15 of Figure 21.2. The split of group 1 between frames 14 and 15 is the most probable hypothesis after data association following model branch 0, although the continuation following model branch 1 is more probable (see the legend for details).

In each model branch, the tracks of the parent hypothesis are first re-predicted to implement that particular model and then associated with observations.

To perform data association between observations and the re-predicted tracks, we follow the approach by Arras *et al.* [2008]. Possible assignments for observations are existing tracks that *match* with the observation, or labels that classify observations as *false alarms* or *new tracks*. At the same time, tracks are interpreted as *matched* when being associated with an observation, but can also be labeled as *deleted* or *occluded*. Thus, the MHT handles the entire life-cycle of tracks from creation over repeated confirmation (by matching) or occlusion (which is non-detection and non-deletion) to deletion.

The generation of assignments is done as described for example by Cox and Miller [1995]: by computing the probability for each possible association, we obtain a cost table that allows us to pose the data association as a linear assignment problem. Using the Hungarian method or the Jonker-Volgenant algorithm [Jonker and Volgenant, 1987], the best solution to such a problem can be found in polynomial time. Murty's algorithm is used to generate alternative solutions as new hypotheses: after computing the best solution, this method enforces or prevents certain associations in an iterative procedure, and thus provides the K-best solutions to the assignment problem at hand [Murty, 1968].

Given a number of alternative assignments ψ_i^k in time step k, we create corresponding child hypotheses Ω_i^k, and compute their probabilities as described in the following section.

20.4 Probability of Assignment Sets and Hypotheses

The probability of a hypothesis in the multi-model MHT is calculated as follows. We compute the probability of a child hypothesis Ω_i^k given the observations from all time steps up to k, denoted by Z^k. According to the Markov assumption, it is the joint probability of the assignment set $\psi_i(k)$, the model M, and the parent hypothesis $\Omega_{p(i)}^{k-1}$, conditioned on the current observation $Z(k)$. Using Bayes rule, this can be expressed as the product of the data likelihood with the joint probability of assignment set, model and parent hypothesis

$$\begin{aligned} & P(\Omega_i^k|Z^k) \\ &= P(\psi, M, \Omega_{p(i)}^{k-1}|Z(k)) && (20.6) \\ &= \eta \cdot P(Z(k)|\psi, M, \Omega_{p(i)}^{k-1}) \cdot P(\psi, M, \Omega_{p(i)}^{k-1}). && (20.7) \end{aligned}$$

By using conditional probabilities, the third term on the right hand side can be factorized into the probabilities of the assignment set, the model, and the parent hypothesis

$$\begin{aligned} & P(\psi, M, \Omega_{p(i)}^{k-1}) \\ &= P(\psi|M, \Omega_{p(i)}^{k-1}) \cdot P(M|\Omega_{p(i)}^{k-1}) \cdot P(\Omega_{p(i)}^{k-1}). && (20.8) \end{aligned}$$

The third factor in this product is known from the previous iteration, whereas the second factor represents the model probability derived in Section 20.1.

It remains to specify the first factor which is the probability of the assignment set ψ. The set ψ contains the assignments of observed clusters $\mathcal{Z}_i$ and group tracks G_j either to each other or to one of their possible labels listed above. Assuming independence between observations and tracks, the probability of the assignment set is the product of the individual assignment probabilities. Namely, they are p_M for matched tracks, p_F for false alarms, p_N for new tracks, p_O for tracks found to be occluded, and p_T for obsolete tracks scheduled for termination. If the numbers of new tracks and false alarms follow a Poisson distribution (as assumed by Reid [1979]), the probabilities p_F and p_N have a sound physical interpretation as $p_F = \lambda_F V$ and $p_N = \lambda_N V$, where λ_F and λ_N are the average rates of events per volume multiplied by the observation volume V (the field of view of the sensor). The probability for an assignment ψ given a model M

and a parent hypothesis Ω^{k-1} is then computed as

$$\begin{aligned} & P(\psi|M,\Omega^{k-1}) \\ & = \; p_M^{N_M}\; p_O^{N_O}\; p_T^{N_T}\; \lambda_F^{N_F}\lambda_N^{N_N}\; V^{N_F+N_N}, \end{aligned} \tag{20.9}$$

where the Ns are the number of assignments to the respective labels in ψ.

Thanks to the independence assumption, also the data likelihood $P(Z(k)|\,\psi, M, \Omega^{k-1}_{p(i)})$ is computed by the product of the individual likelihoods of each observation cluster $\mathcal{Z}_i$ in $Z(k)$. If ψ assigns an observation $\mathcal{Z}_i$ to an existing track, we assume the likelihood of $\mathcal{Z}_i$ to follow a normal distribution, given by Eq. 19.12. Observations that are interpreted as false alarms and new tracks are assumed to be uniformly distributed over the observation volume V, yielding a likelihood of $1/V$. The data likelihood then becomes

$$P(Z(k)|\psi, M, \Omega^{k-1}) = \left(\tfrac{1}{V}\right)^{N_N+N_F} \prod_{i=1}^{N_\mathcal{Z}} \mathcal{N}_i^{\delta_i}, \tag{20.10}$$

where δ_i is 1 if $\mathcal{Z}_i$ has been assigned to an existing track, and 0 otherwise.

Substitution of Equations (20.1), (20.9), and (20.10) into Eq. (20.6) leads, like in the original MHT approach, to a compact expression, independent on the observation volume V.

Finally, normalization is performed yielding a true probability distribution over the child hypotheses of the current time step. This distribution is used to determine the current best hypothesis and to guide the pruning strategies.

20.5 Hypothesis Pruning

Pruning is essential in implementations of the MHT algorithm, as otherwise the number of hypotheses grows boundless. We employ the following strategies:

K-best branching: instead of creating all children of a parent hypothesis, the algorithm proposed by Murty [1968] generates only the K most probable hypotheses as mentioned before. We use the multi-parent variant of Murty's algorithm, mentioned by Cox and Miller [1995], that generates the global K best hypotheses for all parents.

Ratio pruning: a lower limit on the ratio of the current and the best hypothesis is defined. Unlikely hypotheses with respect to the best one, being below this threshold, are deleted. Ratio pruning overrides K-best branching in the sense that if the lower limit is reached earlier, less than K hypotheses are generated.

N-scan back: the N-scan-back algorithm considers an ancestor hypothesis at time $k-N$ and looks ahead in time onto all children at the current time k (the leaf nodes). It only keeps the subtree at $k-N$ with the highest sum of leaf node probabilities. All other branches at $k-N$ are discarded.

More details on these pruning strategies can be found in the work of Cox and Hingorani [1996].

21 Experiments

To analyze the performance of our system, we collected two data sets in the entrance hall of a university building, shown in Figure 21.1. We used a Pioneer II robot equipped with a SICK laser scanner mounted at 30 cm above the floor, scanning at 10 fps. In two unscripted experiments, dataset 1 with a stationary robot and dataset 2 with a moving robot, up to 20 people are in the field of view of the sensor. They form a large variety of groups during social interaction, move around, stand together and jointly enter and leave the hall, as shown in Figure 21.2.

To obtain ground truth information, we labeled each single range reading according to the following scheme: beams that belong to a person receive a person-specific label, other beams are labeled as non-person. These labels are kept consistent over the entire duration of the data sets. People that socially interact with each other (derived by observation) are said to belong to a group with a group-specific label. Summed over all frames, the ground truth contains 5,629 labeled groups and 12,524 labeled people.[1] For further details, see Table 21.1.

The ground truth data is used for performance evaluation and to learn the parameters of our tracker. The values, determined by counting the related events in the ground truth and dividing by the total number of these events, are $p_M = 0.79$, $p_O = 0.19$, $p_T = 0.02$, $p_F = 0.06$, $p_N = 0.02$ for the data association probabilities, and $p_C = 0.63$, $p_S = 0.16$, $p_G = 0.21$ for the group formation probabilities. When evaluating the performance of the tracker, we separated the data into a training set and a validation set to avoid overfitting.

The state uncertainty for new tracks was manually set to $\sigma_x = \sigma_y = 0.1,\ \sigma_{\dot{x}} = \sigma_{\dot{y}} = 0.5,\ \sigma_n = 0.2$. The noise parameters for the motion model were set to $\epsilon_x = \epsilon_y = 0.2,\ \epsilon_{\dot{x}} = \epsilon_{\dot{y}} = 0.3,\ \epsilon_n = 0.1$.

Six frames of the current best hypothesis from the second dataset are shown in Figure 21.2. The corresponding hypothesis tree for frame 15 is shown in Figure 20.2. The sequence exemplifies movement and formation of several groups.

[1] The data sets, hand-labeled ground truth information, and videos of our experiments are available online at `http://www.informatik.uni-freiburg.de/~lau/grouptracking`

Figure 21.1: The entrance hall where we recorded the datasets for our experiments.

Table 21.1: Information about the two datasets used in the experiments.

	Dataset 1	Dataset 2
Number of frames	578	991
Avg. / max people	6.25 / 13	8.99 / 20
Avg. / max groups	2.60 / 4	4.16 / 8
Number of splits / merges	5 / 10	48 / 44
Number of new tracks / deletions	19 / 15	34 / 39

21.1 Clustering Error

This section analyzes how well the presented group tracker can recover the true group formation processes, i.e., which people actually belong together according to their social interaction as encoded in the ground truth.

We compute the clustering error of the tracker using the ground truth information on a per-beam basis. This is done by counting how often a set of points $\mathcal{P}$ of a track contains too many or wrong points (under-segmentation) and how often $\mathcal{P}$ is missing points (over-segmentation). Two examples for over-segmentation errors can be seen in frame 15 of Figure 21.2, where group 0 and group 1-0 are temporarily over-segmented, compared to the ground truth which is visualized with a rectangle. However, from the history of group splits and merges stored in the group labels, the correct group relations can be determined in such cases.

For the first dataset, the clustering error rates for under-segmentation, over-segmentation, and the sum of both are shown in Figure 21.3 (left), plotted against the clustering distance d_P.

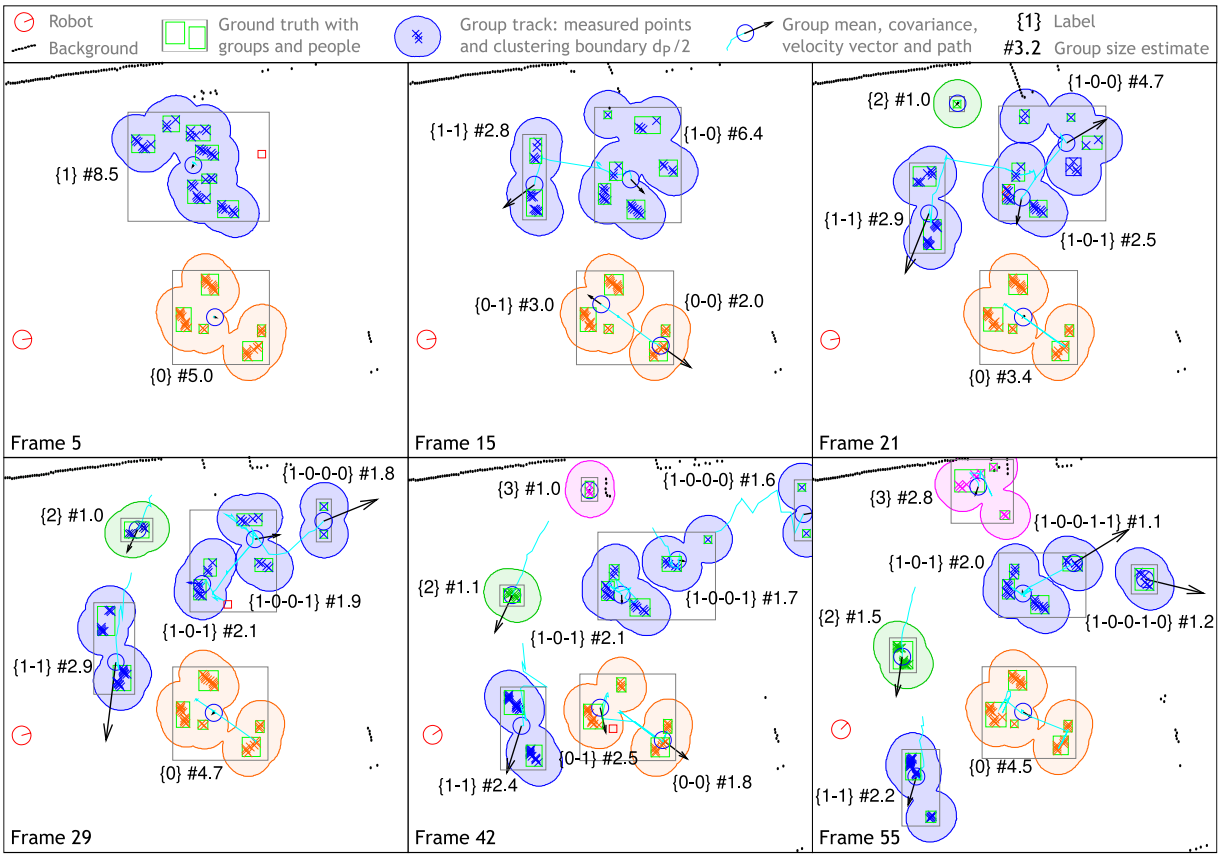


Figure 21.2: Tracking results for Dataset 2. In frame 5, two groups are present. In frame 15, the tracker has correctly split group 1 into 1-0 and 1-1 (see Figure 20.2). Between frames 15 and 29, group 1-0 has split up into groups 1-0-0 and 1-0-1 and split up again. New groups, labeled 2 and 3, enter the view in frames 21 and 42, respectively.

We compare the clustering performance of our group tracker with a memory-less group clustering approach, which performs single-linkage clustering of the range data as described in Section 19.1 without using a tracking framework. The result is shown in Figure 21.3 (middle).

The minimum clustering error of 3.1% is achieved by the tracker at $d_P = 1.3\,\text{m}$. The minimum error for the memory-less clustering is 7.0%, more than twice as high. In the second dataset, the error rates are higher due to the larger number of occlusions and the increased complexity in group interactions. Here, the minimum clustering error of the tracker is 9.6% while the error of the memory-less clustering reaches 20.2%, again more than twice as high.

To further investigate situations where tracking results differ from memory-less clustering, we recorded laser data of groups of people walking and passing each other in a corridor. An example is shown in Figure 21.4, where one person passes between a group of two people. The memory-less approach would merge them immediately while the tracking approach, accounting for the

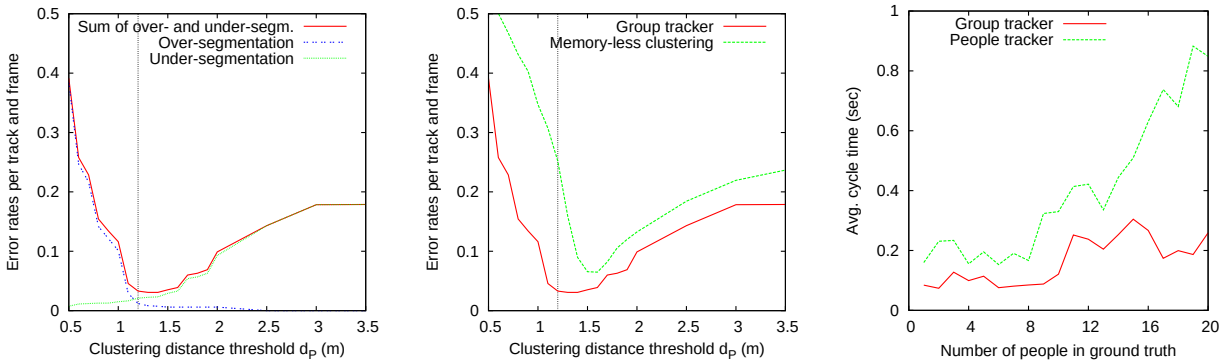


Figure 21.3: *Left:* Clustering error of the group tracker as the sum of over-segmentation and under-segmentation error. The smallest error is achieved for a cluster distance of 1.3 m which is very close to the border of personal and social space according to the proxemics theory, marked at 1.2 m by the vertical line. *Middle:* Clustering error of the group tracker compared to memory-less single linkage clustering (without tracking). *Right:* Average cycle time for the group tracker versus a tracker for individual people plotted against the ground truth number of people.

velocity information, correctly keeps the groups apart by using re-clustering. This result shows that the group tracking problem is a *recursive* clustering problem that requires integration of information over time.

In the light of the proxemics theory, the result of a minimal clustering error at 1.3 m is noteworthy. The theory predicts that when people interact with friends, they maintain a range of distances between 0.45 to 1.2 m called personal space. When engaged in interaction with strangers, this distance is larger. As our data contains students who tend to know each other well, the result appears consistent with the findings of Hall.

21.2 Tracking Efficiency

When tracking groups of people rather than individuals, the assignment problems in the data association stage are of course smaller in size. At the same time, the introduction of an additional tree level, on which different models hypothesize over different group formation processes, comes with additional computational costs. We therefore compare our system with a person-only tracker realized by inhibiting all split and merge operations and reducing the cluster distance d_P to the value that yields the lowest error for clustering single people given the ground truth. For the second dataset, the resulting average cycle times depending on the number of

people being present according to ground truth are shown in Figure 21.3 (right). The plots are averaged over different K from the range of 2 to 200 at a scan-back depth of $N = 30$.

With an increasing number of people, the cycle time for the people tracker grows much faster than the cycle time of the group tracker. Interestingly, even for small numbers of people the group tracker is faster than the people tracker. This is due to occasional over-segmentation of people into individual legs tracks. Also, as mutual occlusion of people in densely populated environments occurs frequently, the people tracker has to maintain many more occluded tracks than the group tracker, as occlusion of entire groups is rare. Also, the additional complexity of multiple models in the group tracker virtually disappears when the tracks are isolated due to the data-driven model generation.

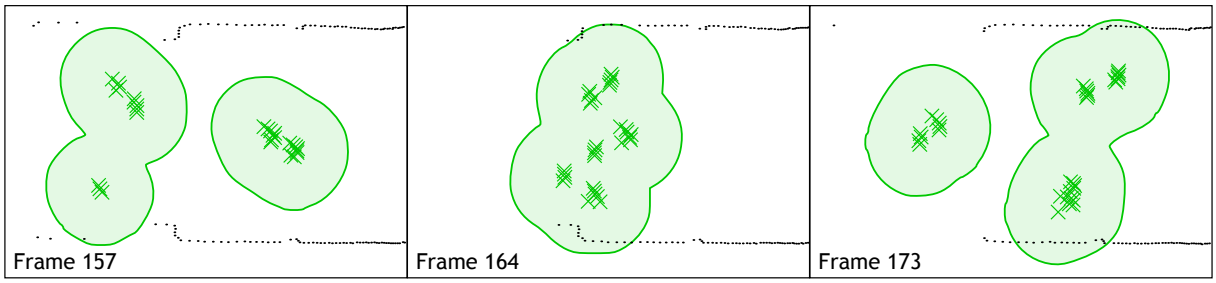
This result clearly shows that our group tracking approach is more efficient. With an average cycle time of around 100 ms for up to 10 people on a Pentium IV at 3.2 GHz, the algorithm can run online even with a non-optimized implementation.

21.3 Group Size Estimation

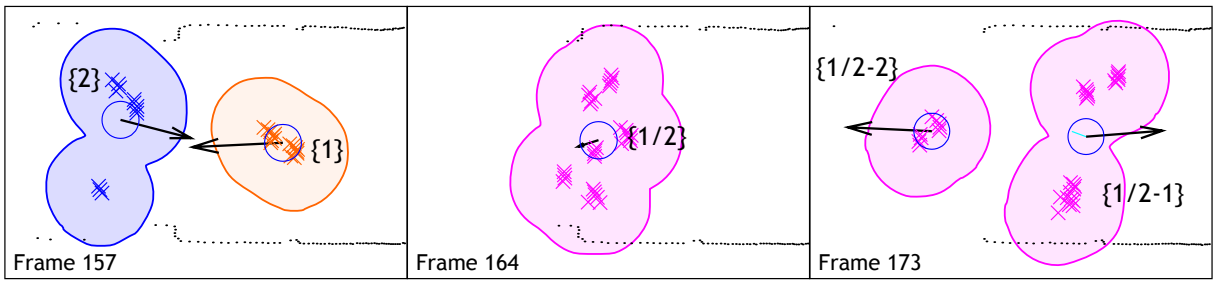
To evaluate the accuracy of our group size estimation approach, we define the corresponding error as the absolute difference between the estimated number of people in a group and the true value according to the labeled ground truth. For counting the number of human-sized clusters in a group as described in Section 19.1, a clustering threshold $d_{\text{hs}} = 0.3\,\text{m}$ is used.

For the first dataset, we find that the average error in group size estimation is 0.23 people with a standard deviation of 0.30. In the more complex dataset 2, the average error is 0.33 people with a standard deviation of 0.49. If the estimated group sizes are rounded to integers, the tracker determined the correct value in 88.9% of all cases in dataset 1 and in 84.3% for dataset 2.

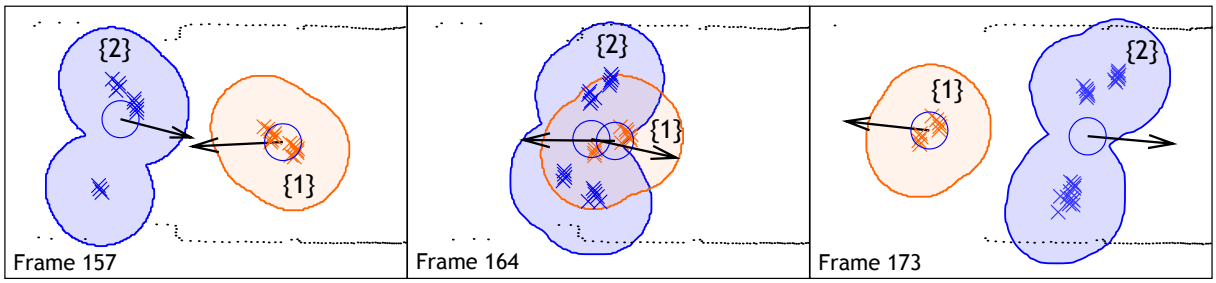
If only deviations of more than one person are considered an error, the system was correct in 99.5% of all cases in dataset 1 and 97.5% in dataset 2.



(a) Memory-less clustering



(b) Group tracking: tracks can merge and split



(c) Group tracking: tracks can continue separately

Figure 21.4: One person crosses a group of two people. Since the groups interweave, memory-less clustering (top) unifies the two groups. Our group tracker can also create a model that postulates a merge of the groups, followed later by a split (middle). However, the model hypothesis leading to the most probable hypothesis in this situation continues both tracks and triggers re-clustering (see Section 20.2). This way, the crossing groups are tracked correctly (bottom). For a legend, see Figure 21.2.

22 Conclusion

We presented a multi-model multi-hypothesis tracking approach to track groups of people. We extended the original MHT approach to incorporate model hypotheses that describe track interaction events that go beyond what data association can express. In our application, models encode the formation of groups during split, merge, and continuation events. We further introduced a group representation that tracks the group shape, and employed the minimum average Hausdorff distance to account for the group shape when calculating association probabilities.

The proposed tracker has been implemented and tested using a mobile robot equipped with a laser range finder. The experiments with up to 20 people forming groups of different sizes demonstrated that the system is able to robustly track groups of people as they undergo complex formation processes. Given ground truth data that reflects true interactions of people with over 12,000 labeled occurrences of people and groups, the experiments showed that the tracker could reproduce such processes with a low clustering error and estimate the number of people in groups with high accuracy. They also showed that in comparison with a memory-less single-frame clustering scheme, our system performs substantially better in determining which people form a group.

The experiments demonstrated the ability of the approach to recover the actual social grouping of interacting people when compared to the ground truth. It was further found that the clustering threshold for detection that produces the best tracking results appears consistent with the proxemics theory. Finally, we showed that tracking groups of people is clearly more efficient than tracking individual people.

Part IV

Discussion and Outlook

23 Discussion

This thesis presented novel techniques for mobile robot navigation in dynamic environments. The three main parts focused on the topics of grid-based spatial representations, kinodynamic motion planning, and tracking groups of people. The fundamental research questions in these areas are not new, and a lot of previous work has been published to address them. In the context of many real-world applications, however, new challenges arise due to the dynamic nature of the environment. We therefore addressed, for example, how spatial representations can be updated efficiently, how planned trajectories can be modified to avoid unexpected obstacles, or how to track a large number of people in online applications.

This chapter summarizes the results and discusses a number of research insights that highlight the contribution of this thesis with respect to previous approaches.

23.1 Efficient Grid-Based Spatial Representations

When using grid-based spatial representations like distance maps, Voronoi diagrams, or configuration space maps in dynamic environments, a main issue is the computation time required to perform an update when the environment changes. In the case of collision avoidance, for example, updates are required usually several times per second. We presented novel algorithms that are able to perform efficient incremental updates of these representations by visiting only the cells that are affected by changes.

The algorithm we proposed to update distance maps is a dynamic version of the brushfire algorithm. An interesting observation is the comparison to the D* lite algorithm which has previously been used to update distance maps [Kalra *et al.*, 2009]. D* lite can actually compute and update the distances from each cell to a goal location, which is more than required for the problem at hand. Our dynamic brushfire algorithm in contrast only considers the distances to the closest obstacle of each cell, which facilitates the substantially shorter computation times.

When comparing different algorithms to compute a Voronoi diagram (GVD) for a given grid map, it is common that the outputs are not the same. Most previous approaches generate 8-connected Voronoi lines that use diagonal connections, which might be more visual appealing

than 4-connected ones. We discovered that 4-connected GVDs better represent the underlying Voronoi graph, since they do not cause erroneous interconnections between neighboring lines. Another apparent difference is the number of generated Voronoi lines, which corresponds to an implicitly assumed obstacle model. When treating each occupied map cell as an obstacle, the GVD is overly full. On the other hand, when treating connected groups of cells as the same obstacle, the GVD misses Voronoi lines inside of non-convex obstacles like the interior of buildings. To our knowledge, our approach is the first method to incrementally update Voronoi diagrams that correctly works in indoor environments.

The proposed combination of distance maps and Voronoi diagrams with our updatable configuration space maps provides efficient means for collision checking and path planning of non-circular robots: with a single map lookup, one can determine the robot-to-obstacle distance for a given rotation, and plan a path on the Voronoi subspace of the configuration space. Furthermore, our bubble-technique for Voronoi path planning ensures similar results for consecutive path queries from a moving robot. Since all involved representations are incrementally updatable, these methods can be applied online in dynamic environments.

23.2 Kinodynamic Motion Planning

This thesis also presented an approach to kinodynamic motion planning for mobile robots. An important building block of such a system is the representation used to model trajectories. We proposed a novel path model that uses quintic Bézier splines to generate smooth parametric trajectories. The additional degrees of freedom in comparison to cubic splines allow us to make the trajectories curvature continuous and to choose their start curvature. This facilitates replanning a trajectory while a robot is moving: after choosing a point on the trajectory and determining the local curvature of the path, we can compute a new trajectory that starts at this point with the given curvature, and thus alter the planned trajectory without interruptions or discontinuities. To reduce the number of free parameters in the model, we use heuristics that specify the orientation of tangents and the parameters that control the second derivative of the spline. As a result, our path model uses a smaller number of parameters than cubic splines while being more suitable for motion planning.

We also discussed the properties of clothoid splines. At first sight, they seem to be an appealing curve type for modeling robot paths due to their linear increase or decrease of curvature for constant translational velocities. However, service robots often reduce their velocity in curves,

which mitigates this advantage. Additionally, the placement of clothoids to connect given waypoints is rather complicated and not necessarily possible in closed form with start curvatures being not equal to zero.

In our experiments we used an error-feedback controller to steer a robot along a planned path, and achieved average deviations below 2 cm with a Pioneer platform. A pre-requisite for this accuracy was to raise the frequency of odometry and control signals from the standard 10 Hz to 35 Hz. This supports the idea of running control loops at a higher frequency than the computational more demanding path planning and localization routines.

To our knowledge, our motion planning method is the first any-time path deformation approach which generates curvature continuous parametric paths that can be modified during execution.

23.3 Laser-Based Tracking of People in Groups

The laser-based people tracking approach proposed in this thesis detects and tracks groups of people rather than individuals. This is different from previous work on group tracking where the group representations are added on top of a tracking system for individuals. As a result, the computational complexity of our approach scales with the number of groups rather than the number of people being tracked.

By introducing an additional group model layer, our tracker can hypothesize about the group formation process. Specifically, it can pursue the hypotheses that a group splits up, merges with another group, or remains unchanged. To keep the number of generated models tractable, we use a data-driven approach similar to the gating of data associations between tracks and observations. The split and merge probabilities depend on the differences in position and velocity of the separate group tracks, i.e., before a merge or after a potential split. Additionally, we assume that splits and merges are binary operators, i.e., each split or merge only involves two groups. Since the tracker can model several splits or merges per second, this does not compromise the tracking result.

In most laser-based tracking systems, the actual observations are pre-clustered sensor readings. This clustering is a critical step, since it is usually not done in a multi-hypothesis way. Consequently, a clustering error causes erroneous observations, and can degrade the tracking performance substantially. In our experimental data we observed that the most common clustering

error is the unification of two close groups, e.g., people passing in a corridor in opposite directions. We therefore proposed a re-clustering mechanism that is triggered whenever a single cluster fits to two different group tracks that are not merging. In this case, we redo the clustering by assigning individual measurements to the compatible group tracks.

As shown in an experiment, the re-clustering mechanism together with the velocity-vector-based split and merge probabilities allow our tracker to maintain passing groups as separate tracks while groups standing together can be merged into one. Thus, our tracker provides some information about the social relation of the people being tracked.

24 Outlook

We believe that this thesis provides novel methods that are suitable for direct use in mobile robot navigation systems. Additionally, the proposed techniques can be useful for other approaches.

The Voronoi-based path planning method proposed in Part I is a very efficient approach to holonomic path planning in the configuration space of non-circular robots. In future work, the configuration space Voronoi maps could be used to implement an efficient Voronoi sampling routine for planners based on Rapidly-Exploring Random Trees, and thus also contribute to the problem of non-holonomic path planning.

An interesting extension of our spline-based motion planning system would be the adaptation to autonomous cars. The curvature continuity of generated trajectories achieved with only three parameters per control point is appealing, especially for vehicles driving at high velocities. Another strength of our approach is the guaranteed traversability of the initial path. However, the adaptation of this property to vehicles that cannot turn on the spot is an open problem.

The tracking research community keeps inventing and evolving probabilistic tracking frameworks and data association methods. Markov chain Monte Carlo (MCMC) techniques for example provide a more general approach to data association than the methods used in standard multi-hypothesis trackers, since they are not restricted to cost matrices of linear assignment problems. Combining MCMC with our group formation models could be an interesting line of research. Also, the advent of affordable 3D cameras brings new possibilities and challenges to people tracking research and could facilitate hybrid approaches for group tracking as well.

List of Figures

List of Algorithms

List of Tables

Bibliography

[Arras *et al.*, 2007] K. O. Arras, Óscar Martínez Mozos, and W. Burgard. Using boosted features for the detection of people in 2d range data. In *IEEE International Conference on Robotics and Automation (ICRA)*, Rome, Italy, April 2007.

[Arras *et al.*, 2008] K. O. Arras, S. Grzonka, M. Luber, and W. Burgard. Efficient people tracking in laser range data using a multi-hypothesis leg-tracker with adaptive occlusion probabilities. In *IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, CA, USA, May 2008.

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

[Beeson, 2006] P. Beeson. EVG-Thin: A thinning approximation to the extended Voronoi graph. `http://www.cs.utexas.edu/users/qr/software/evg-thin.html`, 2006.

[Behar and Lien, 2010] E. Behar and J.-M. Lien. A new method for mapping the configuration space obstacles of polygons. Technical Report GMU-CS-TR-2011-11, Department of Computer Science, George Mason University, 2010.

[Blanco *et al.*, 2005] F. J. Blanco, V. Moreno, B. Curto, and R. Therón. C-space evaluation for mobile robots at large workspaces. In *IEEE International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain, April 2005.

[Borenstein and Koren, 1991] J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[Borgefors, 1986] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344–371, 1986.

[Bose *et al.*, 2007] B. Bose, X. Wang, and E. Grimson. Multi-class object tracking algorithm that handles fragmentation and grouping. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, Minneapolis, MN, USA, June 2007.

[Canny, 1985] J. Canny. A Voronoi method for the piano-movers problem. In *IEEE International Conference on Robotics and Automation (ICRA)*, St Louis, MO, USA, March 1985.

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

[Choset *et al.*, 2000] H. Choset, S. Walker, K. Eiamsa-Ard, and J. Burdick. Sensor-based exploration: Incremental construction of the hierarchical generalized Voronoi graph. *The International Journal of Robotics Research*, 19:126–148, 2000.

[Connors and Elkaim, 2007] J. Connors and G. Elkaim. Manipulating B-Spline based paths for obstacle avoidance in autonomous ground vehicles. In *Proc. of the ION National Technical Meeting*, San Diego, CA, USA, January 2007.

[Cox and Hingorani, 1996] I. Cox and S. Hingorani. An efficient implementation of Reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(2):138–150, 1996.

[Cox and Miller, 1995] I. Cox and M. Miller. On finding ranked assignments with application to multi-target tracking and motion correspondence. *IEEE Transactions on Aerospace and Electronic Systems*, 31(1):486–489, 1995.

[Şucan *et al.*, 2010] I. A. Şucan, M. Moll, and L. E. Kavraki. The open motion planning library (OMPL). `http://ompl.kavrakilab.org`, 2010.

[Cui *et al.*, 2005] J. Cui, H. Zha, H. Zhao, and R. Shibasaki. Tracking multiple people using laser and vision. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Alberta, Canada, August 2005.

[Cuisenaire and Macq, 1999] O. Cuisenaire and B. Macq. Fast Euclidean distance transformation by propagation using multiple neighborhoods. *Computer Vision and Image Understanding*, 76:163–172, 1999.

[Danielsson, 1980] P.-E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.

[Donald *et al.*, 1993] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the Association for Computing Machinery*, 40(5):1048–1066, 1993.

[Dubuisson and Jain, 1994] M. P. Dubuisson and A. K. Jain. A modified Hausdorff distance for object matching. In *Intl. Conference on Pattern Recognition*, volume 1, pages A:566–568, Jerusalem, Israel, 1994.

[Fabbri *et al.*, 2008] R. Fabbri, L. da Fontoura Costa, J. C. Torelli, and O. M. Bruno. 2D Euclidean distance transform algorithms: A comparative survey. *ACM Computing Surveys*, 40(1), 2008.

[Ferguson and Stentz, 2007] D. Ferguson and A. Stentz. Field D*: An interpolation-based path planner and replanner. In S. Thrun, R. Brooks, and H. Durrant-Whyte, editors, *Robotics Research*, volume 28 of *Springer Tracts in Advanced Robotics (STAR)*, pages 239–253. Springer Berlin / Heidelberg, 2007.

[Fod *et al.*, 2002] A. Fod, A. Howard, and M. J. Mataric. Laser-based people tracking. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3024–3029, Washington D.C., USA, May 2002.

[Foskey *et al.*, 2001] M. Foskey, M. Garber, M. C. Lin, and D. Manocha. A Voronoi-based hybrid motion planner. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Maui, HI, USA, October 2001.

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

[Galin *et al.*, 2010] E. Galin, A. Peytavie, N. Maréchal, and E. Guérin. Procedural generation of roads. *Computer Graphics Forum*, 29(2):429–438, May 2010.

[Gennari and Hager, 2004] G. Gennari and G. D. Hager. Probabilistic data association medhods in visual tracking of groups. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[Geraerts and Overmars, 2004] R. Geraerts and M. Overmars. A comparative study of probabilistic roadmap planners. In *Algorithmic Foundations of Robotics V*, volume 7 of *Springer Tracts in Advanced Robotics (STAR)*, pages 43–58. Springer Berlin / Heidelberg, 2004.

[Girbés *et al.*, 2011] V. Girbés, L. Armesto, and J. Tornero. Continuous-curvature control of mobile robots with constrained kinematics. In *IFAC World Congress*, volume 18, Milano, Italy, 2011.

[Gold *et al.*, 1997] C. M. Gold, P. R. Remmele, and T. Roos. Voronoi methods in GIS. In *Algorithmic Foundations of Geographic Information Systems*, volume 1340. Springer Berlin / Heidelberg, 1997.

[Grisetti *et al.*, 2007] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[Guenter and Parent, 1990] B. Guenter and R. Parent. Computing the arc length of parametric curves. *Computer Graphics and Applications*, 10(3):72–78, May 1990.

[Guibas *et al.*, 1992] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[Gulati and Kuipers, 2008] S. Gulati and B. Kuipers. High performance control for graceful motion of an intelligent wheelchair. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2008.

[Hall, 1974] E. Hall. *Handbook of Proxemics Research*. Society for the Anthropology of Visual Communications, 1974.

[Hartigan, 1975] J. Hartigan. *Clustering Algorithms*. John Wiley & Sons, 1975.

[Heald, 1985] M. A. Heald. Rational approximations for the fresnel integrals. *Mathematics of Computation*, 44(170):459–461, 1985.

[Heald, 1986] M. A. Heald. Corrigenda. *Mathematics of Computation*, 46(174):771, 1986.

[Henrie and Wilde, 2012] J. Henrie and D. Wilde. *Experience from the DARPA Urban Challenge*, chapter Planning continous curvature paths using constructive polylines. Springer-Verlag London, 2012.

[Howard and Kelly, 2007] T. M. Howard and A. Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *International Journal of Robotics Research (IJRR)*, 26:141–166, 2007.

[Jaillet *et al.*, 2011] L. Jaillet, J. Hoffman, J. van den Berg, P. Abbeel, J. Porta, and K. Goldberg. EG-RRT: Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2646–2652, San Francisco, CA, USA, September 2011.

[Jonker and Volgenant, 1987] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, December 1987.

[Joo and Chellappa, 2007] S.-W. Joo and R. Chellappa. A multiple-hypothesis approach for multiobject visual tracking. *IEEE Transactions on Image Processing*, 16(11):2849–2854, November 2007.

[Kalra *et al.*, 2009] N. Kalra, D. Ferguson, and A. Stentz. Incremental reconstruction of generalized Voronoi diagrams on grids. *Robotics and Autonomous Systems (RAS)*, 57:123–128, 2009.

[Karaman *et al.*, 2011] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime motion planning using the RRT*. In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[Kavraki, 1995] L. E. Kavraki. Computation of configuration-space obstacles using the fast Fourier transform. *IEEE Transactions on Robotics and Automation*, 11(3):408–413, 1995.

[Khan *et al.*, 2006] Z. Khan, T. Balch, and F. Dellaert. MCMC data association and sparse factorization updating for real time multitarget tracking with merged and multiple measurements. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12), 2006.

[Khatib, 1986] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Intl. Journal of Robotics Research*, 5:90–98, 1986.

[Kluge *et al.*, 2001] B. Kluge, C. Köhler, and E. Prassler. Fast and robust tracking of multiple moving objects with a laser range finder. In *IEEE International Conference on Robotics and Automation (ICRA)*, Seoul, Korea, May 2001.

[Koenig and Likhachev, 2002] S. Koenig and M. Likhachev. D* lite. In *Eighteenth National Conference on Artificial Intelligence (AAAI)*, pages 476–483, 2002.

[Lau *et al.*, 2009] B. Lau, C. Sprunk, and W. Burgard. Kinodynamic motion planning for mobile robots using splines. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA, 2009.

[Lau *et al.*, 2010] B. Lau, C. Sprunk, and W. Burgard. Improved updating of euclidean distance maps and Voronoi diagrams. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, 2010.

[Lau *et al.*, 2011] B. Lau, C. Sprunk, and W. Burgard. Incremental updates of configuration space representations for non-circular mobile robots with 2D, 2.5D, or 3D obstacle models. In

Proc. of the European Conference on Mobile Robots (ECMR), pages 49–54, Örebro, Sweden, 2011.

[Lau *et al.*, 2012] B. Lau, C. Sprunk, and W. Burgard. Open source implementation of dynamically updatable distance maps, Voronoi diagrams, and configuration space representations. `http://www.informatik.uni-freiburg.de/~lau/dynamicvoronoi`, 2012.

[Lee and Gahegan, 2002] I. Lee and M. Gahegan. Interactive analysis using Voronoi diagrams: Algorithms to support dynamic update from a generic triangle-based data structure. *Transactions in GIS*, 6(2):89–114, 2002.

[Lee *et al.*, 2011] T.-K. Lee, S.-H. Baek, Y.-H. Choi, and S.-Y. Oh. Smooth coverage path planning and control of mobile robots based on high-resolution grid map representation. *Robotics and Autonomous Systems (RAS)*, 59(10):801–812, October 2011.

[Likhachev and Ferguson, 2008] M. Likhachev and D. Ferguson. Planning long dynamically-feasible maneuvers for autonomous vehicles. In *Robotics: Science and Systems Conference*, Zurich, Switzerland, 2008.

[Likhachev *et al.*, 2005] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In *Proc. of the Intl. Conf. on Automated Planning and Scheduling (ICAPS)*, Monterey, CA, USA, June 2005.

[Linan and Zhenmin, 2005] J. Linan and T. Zhenmin. Building configuration space for multiple UGVs. In *IEEE Intl. Conf. on Vehicular Electronics and Safety*, Xi'an, Shaan'xi, China, October 2005.

[Lindemann and LaValle, 2004] S. Lindemann and S. LaValle. Incrementally reducing dispersion by increasing Voronoi bias in RRTs. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3251–3257, New Orleans, LA, USA, April 2004.

[Loustau and Dillon, 1992] J. Loustau and M. Dillon. *Linear Geometry with Computer Graphics*. CRC Press, 1992.

[Lozano-Perez, 1983] T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, February 1983.

[Macek *et al.*, 2008] K. Macek, D. Vasquez, T. Fraichard, and R. Siegwart. Safe vehicle navigation in dynamic urban scenarios. In *Proc. of the 11th IEEE Intl. Conf. on Intelligent Transportation Systems (ITSC)*, pages 482–489, Beijing, China, October 2008.

[Mandel and Frese, 2007] 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 (ECMR)*, Freiburg, Germany, September 2007.

[Maurer *et al.*, 2003] C. R. Maurer, Jr., R. Qi, and V. Raghavan. A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):265–270, 2003.

[McKenna *et al.*, 2000] S. McKenna, S. Jabri, Z. Duric, A. Rosenfeld, and H. Wechsler. Tracking groups of people. *Computer Vision and Image Understanding*, 80(1):42–56, October 2000.

[Minguez and Montano, 2004] J. Minguez and L. Montano. Nearness diagram navigation (ND): Collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, 20(1):45–59, 2004.

[Montemerlo *et al.*, 2003] M. Montemerlo, N. Roy, and S. Thrun. Perspectives on standardization in mobile robot programming: the Carnegie Mellon Navigation (CARMEN) Toolkit. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2436–2441, 27–31 Oct. 2003.

[Mucientes and Burgard, 2006] M. Mucientes and W. Burgard. Multiple hypothesis tracking of clusters of people. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 692–697, October 2006.

[Murty, 1968] K. Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, 16:682–687, 1968.

[Oriolo *et al.*, 2002] 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, November 2002.

[Pan and Manocha, 2011] J. Pan and D. Manocha. GPU-based parallel collision detection for real-time motion planning. In *Algorithmic Foundations of Robotics IX*, volume 68 of *Springer Tracts in Advanced Robotics (STAR)*. Springer Berlin / Heidelberg, 2011.

[Plaku *et al.*, 2010] E. Plaku, E. Kavraki, and M. Vardi. Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Transactions on Robotics*, 26(3):469–482, 2010.

[Plato, 2003] R. Plato. *Concise Numerical Mathematics*, volume 57 of *Graduate studies in mathematics*. American Mathematical Society, 2003.

[Rao *et al.*, 1991] N. Rao, N. Stoltzfus, and S. Iyengar. A 'retraction' method for learned navigation in unknown terrains for a circular robot. *IEEE Transactions on Robotics and Automation*, 7(5):699–707, October 1991.

[Reid, 1979] D. B. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, AC-24(6):843–854, 1979.

[Riedmiller and Braun, 1993] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE Intl. Conf. on Neural Networks*, pages 586–591, San Francisco, CA, 1993.

[Sahraei *et al.*, 2007] A. Sahraei, M. T. Manzuri, M. R. Razvan, M. Tajfard, and S. Khoshbakht. *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*, chapter Real-time trajectory generation for mobile robots, pages 459–470. Springer, 2007.

[Scherer *et al.*, 2009] S. Scherer, D. Ferguson, and S. Singh. Efficient C-Space and cost function updates in 3D for unmanned aerial vehicles. In *IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, 2009.

[Scherer *et al.*, 2012] S. Scherer, J. Rehder, S. Achar, H. Cover, A. Chambers, S. Nuske, and S. Singh. River mapping from a flying robot: state estimation, river detection, and obstacle mapping. *Autonomous Robots*, 33:189–214, 2012.

[Schlegel, 1998] C. Schlegel. Fast local obstacle avoidance under kinematic and dynamic constraints for a mobile robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Victoria, Canada, 1998.

[Schulz *et al.*, 2003] D. Schulz, W. Burgard, D. Fox, and A. Cremers. People tracking with a mobile robot using sample-based joint probabilistic data association filters. *International Journal of Robotics Research (IJRR)*, 22(2), 2003.

[Shiller and Gwo, 1991] Z. Shiller and Y. Gwo. Dynamic motion planning of autonomous vehicles. *IEEE Transactions on Robotics and Automation*, 7:241–249, 1991.

[Siciliano *et al.*, 2009] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics, Planning and Control*, chapter 11. Springer-Verlag London, 2009.

[Sprunk *et al.*, 2011] C. Sprunk, B. Lau, P. Pfaff, and W. Burgard. Online generation of kinodynamic trajectories for non-circular omnidirectional robots. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, Shanghai, China, 2011.

[Sprunk *et al.*, 2012] C. Sprunk, B. Lau, and W. Burgard. Improved non-linear spline fitting for teaching trajectories to mobile robots. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, St. Paul, MN, USA, May 2012.

[Sprunk, 2008] C. Sprunk. Planning motion trajectories for mobile robots using splines. University of Freiburg, `http://www.informatik.uni-freiburg.de/~lau/students/Sprunk2008.pdf`, 2008.

[Stachniss and Burgard, 2002] C. Stachniss and W. Burgard. An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System*, volume 1, pages 508–513, 30 Sept.–5 Oct. 2002.

[Stachniss *et al.*, 2012] C. Stachniss, U. Frese, and G. Grisetti. OpenSLAM website. Online: `http://openslam.org`, 2012.

[Stentz, 2004] A. Stentz. Optimal and efficient path planning for partially-known environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3310–3317, San Diego, CA, USA, May 2004.

[Stewart, 2002] J. Stewart. *Calculus: Early Transcendentals*. Brooks Cole, Pacific Grove, 5th edition, 2002.

[Tang *et al.*, 2011] M. Tang, Y. J. Kim, and D. Manocha. CCQ: Efficient local planning using connection collision query. In *Algorithmic Foundations of Robotics IX*, volume 68 of *Springer Tracts in Advanced Robotics (STAR)*. Springer Berlin / Heidelberg, 2011.

[Tao *et al.*, 2011] T. Tao, S. Tully, G. Kantor, and H. Choset. Incremental construction of the saturated-GVG for multi-hypothesis topological SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3072–3077, May 2011.

[Taylor and Kleeman, 2004] G. Taylor and L. Kleeman. A multiple hypothesis walking person tracker with switched dynamic model. In *Proc. of the Australasian Conference on Robotics and Automation*, Canberra, Australia, 2004.

[Therón *et al.*, 2003] R. Therón, V. Moreno, B. Curto, and F. J. Blanco. Configuration space of 3D mobile robots: Parallel processing. In *11th Intl. Conf. on Advanced Robotics*, volume 1–3, Coimbra, Portugal, June 2003.

[Thrun and Bücken, 1998] S. Thrun and A. Bücken. Learning maps for indoor mobile robot navigation. *Artificial Intelligence*, 99:21–71, 1998.

[Thrun, 2001] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 20(5):335–363, 2001.

[Verwer *et al.*, 1989] B. J. H. Verwer, P. W. Verbeek, and S. T. Dekker. An efficient uniform cost algorithm applied to distance transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(4), April 1989.

[Walton and Meek, 2005] D. Walton and D. Meek. A controlled clothoid spline. *Computers & Graphics*, 29(3):353–363, June 2005.

[Wilde, 2009] D. K. Wilde. Computing clothoid segments for trajectory generation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA, October 2009.

[Wise and Bowyer, 2000] K. D. Wise and A. Bowyer. A survey of global configuration-space mapping techniques for a single robot in a static environment. *International Journal of Robotics Research (IJRR)*, 19(8):762–779, 2000.

[Wu *et al.*, 2006] X. J. Wu, J. Tang, and K. H. Heng. On the construction of discretized configuration space of manipulators. *Robotica*, 25:1–11, 2006.

[Wurm and Hornung, 2012] K. M. Wurm and A. Hornung. OctoMap, an efficient probabilistic 3D mapping framework based on octrees. `http://octomap.github.com`, 2012.

[Wurm *et al.*, 2010] K. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *ICRA Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, Anchorage, AK, USA, May 2010.

[Zajdel *et al.*, 2005] W. Zajdel, Z. Zivkovic, and B. Kröse. Keeping track of humans: Have I seen this person before? In *IEEE International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain, April 2005.

[Zhang and Manocha, 2008] L. Zhang and D. Manocha. An efficient retraction-based RRT planner. In *IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, CA, USA, May 2008.

[Ziegler and Stiller, 2010] J. Ziegler and C. Stiller. Fast collision checking for intelligent vehicle motion planning. In *IEEE Intelligent Vehicles Symposium*, San Diego, CA, USA, June 2010.

[Ziegler *et al.*, 2008] J. Ziegler, M. Werling, and J. Schröder. Navigating car-like robots in unstructured environments using an obstacle sensitive cost function. In *IEEE Intelligent Vehicles Symposium (IV 08)*, pages 787–791, Eindhoven, Netherlands, June 2008.