View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Infoscience - École polytechnique fédérale de Lausanne

MOTION PLANNING AND OBSTACLE AVOIDANCE FOR MOBILE ROBOTS IN HIGHLY CLUTTERED DYNAMIC ENVIRONMENTS

THÈSE N^{O} 3146 (2004)

PRÉSENTÉE À LA FACULTÉ SCIENCES ET TECHNIQUES DE L'INGÉNIEUR

Institut d'ingénierie des systèmes

SECTION DE MICROTECHNIQUE

ÉCOLE POLYTECHNIQUE FÉDERALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES TECHNIQUES

PAR

Roland PHILIPPSEN

ingénieur en microtechnique diplômé EPF
et de nationalité allemande

acceptée sur proposition du jury:

Prof. R. Siegwart, directeur de thèse
Prof. O. Khatib, rapporteur
Dr C. Laugier, rapporteur
Prof. J.-P. Thiran, rapporteur

Lausanne, EPFL
2004

Acknowledgment

I would like to express my deepest gratitude to the following people for helping me with the work presented in this dissertation, be it through moral support during periods of stress and doubt, through supplying alternative and usually simplifying views during scientific and technical discussions, or by providing the possibility of embarking on the extraordinary experience of participating in the adventures of mobile robotics.

First of all, I am deeply indebted to Roland Siegwart, my principal adviser. He provided me with the free and open minded environment that allowed me to develop professionally as well as personally. And thank you very much for accepting me back onto the project after I had temporarily abandoned!

I present many thanks to the jury that kindly accepted to judge on the fulfillment of the requirements for a PhD in mobile robotics. You turned the final exam into a fruitful discussion on remaining issues with this dissertation. In this final version I have striven to take your comments fully into account. In particular, I would like to thank Oussama Khatib for his reassuring influence the day before the exam and his calm but insisting and detailed remarks, Christian Laugier for his very valuable suggestions concerning the readability of chapter 4, and Jean-Philippe Thiran for helping me improve the cohesion between the various parts of the thesis.

The contact with Kurt Konolige provided a valuable impulse for the E^* algorithm, he pointed me to the Level Set Method during discussions about his Gradient Method. Without this information, some of the key realization behind chapter 4 might never have seen the day. Illah Nourbakhsh provided fresh and often cheerful support during the early stages of this project.

Thanks go out to my friends and colleagues, Björn Jensen, Nicola Tomatis, Daniel Burnier, Pierre Lamon, Francesco Mondada, Frédéric Pont, my friend and flat-mate Ivo Stotz, and all the others at the Autonomous Systems Lab. I am afraid that naming you all would exceed the reasonable length for this preamble. You know I tend to make things more complicated than necessary, and you often brought my feet back on the ground. Special thanks to Marie-Jose Pellaud for forgiving me my lack of administrative talent.

Last but not least, my family has always remained a reliable anchor providing support, love, and understanding. Thank you Ursula, Peter, Karolin, and Ansgar, for being available. And also for providing very valuable feedback on my developing writing skills, which often lacked the necessary rigor.

Abstract

After a quarter century of mobile robot research, applications of this fascinating technology appear in real-world settings. Some require operation in environments that are densely cluttered with moving obstacles. Public mass exhibitions or conventions are examples of such challenging environments. This dissertation addresses the navigational challenges that arise in settings where mobile robots move among people and possibly need to directly interact with humans who are not used to dealing with technical details. Two important aspects are solved: Reliable reactive obstacle avoidance to guarantee safe operation, and smooth path planning that allows to dynamically adapt environment information to the motion of surrounding persons and objects.

Given the existing body of research results in the field of obstacle avoidance and path planning, which is reviewed in this context, particular attention is paid to integration aspects for leveraging advantages while compensating drawbacks of various methods. In particular, grid-based wavefront propagation (NF1 and fast marching level set methods), dynamic path representation (bubble band concept), and high-fidelity execution (dynamic window approach) are combined in novel ways. Experiments demonstrate the robustness of the obstacle avoidance and path planning systems.

Zusammenfassung

Nach einem Vierteljahrhundert der Forschung erscheinen nun immer konkreter werdende Anwendungen der mobilen Robotik. Einige davon bedeuten Einsätze in Umgebungen, die dicht besiedelt sind mit bewegten Objekten. Die vorliegende Arbeit behandelt die Herausforderungen, die an das Navigationssystem des Roboters gestellt werden, wenn dieser in Menschenmengen zurechtkommen muss. Als Beispiele können Massenausstellungen oder Konferenzen herangezogen werden. Möglicherweise sind die Menschen, mit denen der Roboter interagiert, nicht bewandt im Umgang mit detailliertem technischen Wissen, und diesem Umstand wird Rechnung getragen. Für zwei ausschlaggebende Aspekte werden Lösungen präsentiert: Zuverlässiges und sicheres Ausweichen von Hindernissen; sowie eine flüssige Wegplanung, die sich der verfügbaren Information über die teilweise bewegten Hindernisse dynamisch anpasst.

Contents

1 Introduction **1**
1.1 Motion Skills and Artificial Intelligence 1
1.2 Overview and Contribution 4

2 Scope and Prior Art **5**
2.1 Project Chronology 5
2.1.1 Incremental Development Approach 6
2.1.2 Collision Avoidance Development 7
2.1.3 Adding Planning and Path Adaptation 8
2.1.4 Planning with Environment Dynamics 8
2.1.5 A Dynamic Approach to Weighted Region Planning 9
2.2 Publication Catalog 10
2.2.1 Motion Generation on Robox 10
2.2.2 Dynamic Planning and Interpolation 11
2.2.3 Background 12

3 Motion in Dynamic Cluttered Environments **15**
3.1 Robox at the Swiss National Exhibition 15
3.2 Objectives of Robotics@Expo.02 17
3.3 Approach 19
3.4 Dynamic Window 20
3.4.1 Velocity Space 20
3.4.2 Obstacle Model 20
3.4.3 Collision Prediction 23
3.4.4 Objective Functions 25
3.4.5 Switching Movement Behavior 27
3.4.6 Grid Effects 28
3.5 Elastic Band 29
3.5.1 Bubbles 30
3.5.2 Obstacle Masking 30
3.5.3 Amount of Bubbles 32
3.5.4 Artificial Forces 34
3.6 NF1 35

3.6.1 Grid Based Environment Representation 36
3.6.2 Path Properties 37
3.7 System Integration 37
3.7.1 Perception 37
3.7.2 Multitasking and Time Constraints 38
3.7.3 Memory Constraints 40
3.7.4 Program Structure 43
3.7.5 Replanning Behavior 47
3.8 Results 48
3.9 Future Work 50
3.10 Conclusion 51
3.11 Summary of Parameters 51

4 E*: Dynamic Interpolated Planning 53
4.1 The Need for Smooth Dynamic Planning 54
4.2 Navigation Functions as Distance Maps 55
4.3 Dynamic Planning and Wavefront Propagation 56
4.4 Summary of the Level Set Formulation 57
4.4.1 The Lagrangian and Eulerian Formulations 58
4.4.2 The Eikonal Case: Fast Marching 59
4.5 E* Framework 62
4.5.1 Interpolation Kernels 63
4.5.2 Algorithm Structure 63
4.6 Interpolation 68
4.6.1 Graph Distance 68
4.6.2 Huygens' Principle 69
4.6.3 Gradient Approximation 72
4.7 Evaluation and Performance Measurements 75
4.8 Global Planning with E* 87
4.8.1 `GridPlanner` Usage 87
4.8.2 Illustrated Operation of `GridPlanner` 93
4.9 Conclusion and Outlook 93

5 Conclusion 99

Bibliography 101

A `MotionPlanner` Implementation Details 107
A.1 A Numerically Stable Quadratic Equation Solver 107
A.2 An Implementation of NF1 108
A.3 A Variant of the Lloyd-Max Quantizer 110

B E* Implementation Details 115
B.1 Event Propagation 115
B.2 Interpolation Kernels 116

List of Figures

1.1 Children playing with Robox . . . 2
1.2 Kasparov vs. Deep Blue . . . 3
1.3 Gestaltist effects . . . 4

3.1 Robox tour-guide robot . . . 16
3.2 Environment clutter at EXPO.02 . . . 17
3.3 Flowdiagram of motion generation . . . 21
3.4 Velocity space definition for DWA . . . 22
3.5 Collision prediction . . . 23
3.6 Calculating the heading objective . . . 26
3.7 Calculating the speed objective . . . 26
3.8 Calculating the clearance objective . . . 27
3.9 Avoiding grid effects of w_{head} and w_{speed} . . . 29
3.10 Elastic band definition . . . 31
3.11 Circle intersection . . . 33
3.12 NF1 example . . . 35
3.13 NF1 grid placement . . . 36
3.14 Subsystems on Robox . . . 39
3.15 Look-up operations in the dynamic window . . . 41
3.16 Lloyd-Max quantizer . . . 42
3.17 Motion generation class diagram . . . 44
3.18 State machine of `MotionPlanner` . . . 45
3.19 Real data of a common situation during EXPO.02 . . . 48
3.20 Typical replan sequence . . . 49

4.1 Continuous domain wavefront . . . 57
4.2 Continuous wavefront formulation . . . 58
4.3 Lagrangian formulation of wavefront . . . 59
4.4 Eulerian formulation of wavefront . . . 60
4.5 One-dimensional Eulerian wavefront . . . 60
4.6 Eikonal case of wavefront propagation . . . 61
4.7 Overview of the entities in E^* . . . 64
4.8 Example of event-based propagation . . . 66

4.9 Interpolation by Huygens' Principle 70
4.10 Cell neighborhood 73
4.11 Geometric interpretation of LSM interpolation 74
4.12 Simulation setup for evaluating E* 76
4.13 Summary of tables 4.1, 4.2, and 4.3 80
4.14 Summary of table 4.4 81
4.15 Summary of table 4.5 85
4.16 Simulated robot movement with E* 88
4.17 States of `GridPlanner` 89
4.18 Maze exploration with LSM 94
4.19 Maze exploration with NF1 95
4.20 Obstacle removal with LSM 96
4.21 Obstacle removal with NF1 97

A.1 NF1 example plan 108

List of Tables

3.1 Technical requirements 19
3.2 Motion generation parameters 52

4.1 Relative error of E* without interpolation 78
4.2 Relative error of E* with HPR interpolation 78
4.3 Relative error of E* with LSM interpolation 79
4.4 Relative error of E* in zig-zag environment 79
4.5 E* performance with and without interpolation 84
4.6 E* computational complexity with and without interpolation 86
4.7 E* operation cost with and without interpolation 86

Code Listings

A.1 NF1 C++ code 109
A.2 Creating the histogram for Lloyd-Max quantizing. 111
A.3 C++ implementation of the Lloyd-Max quantizer 112
A.4 Replacing *double* values by quantizer indices. 113
B.1 `LowerEvent` propagation algorithm. 115
B.2 `RaiseEvent` propagation algorithm. 116
B.3 `NF1Interpolation` implementation 116
B.4 `HPRInterpolation` implementation 117
B.5 Computations for the LSM update equation (4.20). 118

Chapter 1

Introduction

> *We may hope that machines will eventually compete with men in all purely intellectual fields.*
>
> Alan M. Turing [57], 1950

This dissertation is a contribution to the field of mobile robot path planning and obstacle avoidance. A large part of the work presented herein was developed in the context of a public mass exhibition confronting individuals of various ages, background, and interests with interactive, mobile machines: The *Robotics* pavilion at Swiss national exhibition EXPO.02 included an area where people met eleven *Robox* tour guide robots. Their task was to give tours of exhibits, present themselves and the researchers who created the robots, and interact with the visitors. This was an enriching experience for the majority of individuals involved, including those behind the scenes who had spent months developing the robotic systems and were now, for the first time, confronted with the reactions of adults and children (see figure 1.1) exposed to Robox — and also with how their brainchild reacted to the environment it had been built for. The five months of EXPO.02 would have provided an excellent field of study for a new breed of anthropologists.

Humans are very good at navigating crowded places, be it in train stations or bazaars, during carnivals or international conferences. There is little or no conscious effort involved. So it cannot possibly be very difficult, or can it?

1.1 Motion Skills and Artificial Intelligence

This thesis is about motion generation for wheeled mobile robots. This includes path planning and obstacle avoidance, which have been research topics since the beginning of robotics, and in particular mobile robotics since the late sixties to early seventies. Why is it still interesting to work on making these machines move? Because real world applications lead to objectives that are not necessarily met by existing approaches.

Mobile robot applications are (slowly) growing in number and complexity, at least in technologically advanced countries. Some observers are predicting that a “killer app” for mobile robots is inevitable in the near or mid term future, which could trigger an evolution

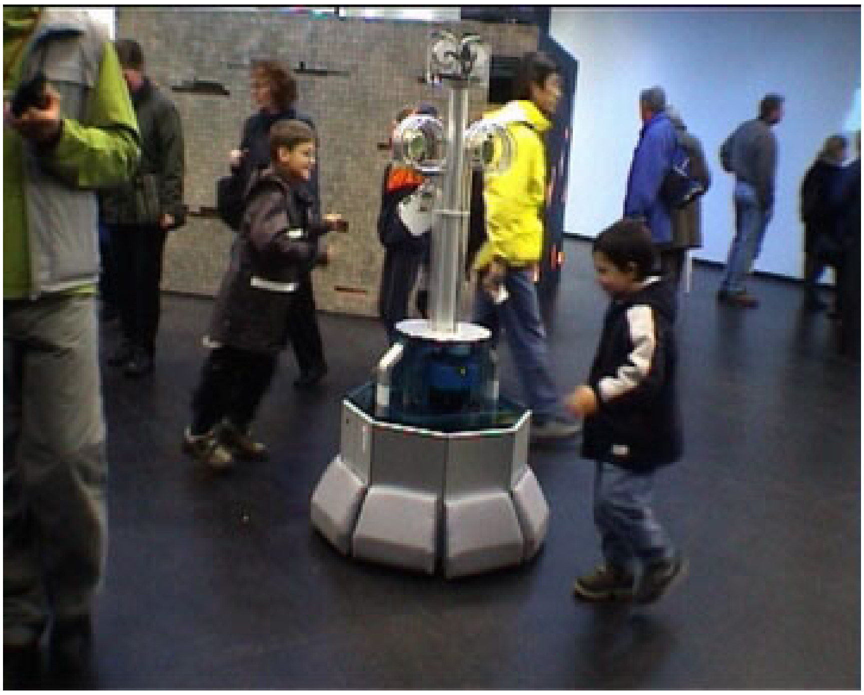


Figure 1.1: Children playing with Robox during Robotics@Expo.02. Many visitors enjoyed trying to trap the robot, which was difficult for a single person.

of autonomous systems that can be likened to the PC revolution. Whether these predictions are realistic is not in the scope of this thesis, but it is clear that more and more mobile robot installations take place in public settings in close contact with people who are not trained for interacting with such machines. Examples are exhibitions and fairs, where robots can effectively capture the visitor's attention. Technologies such as mechatronics, locomotion concepts, control software, or system architectures that were developed in research labs are making their way into real world settings. And it turns out that robustness and safety become of primary importance, along with other shifts in objectives. In particular, motion being a distinctive property of mobile robots, path planning and obstacle avoidance have a decisive impact on how people perceive and react to an autonomous system. The aim is to produce motion behavior that is convincing for the general public: The robot's motion intent should be clear so that people can move out of its path, yet its motion must not be threatening.

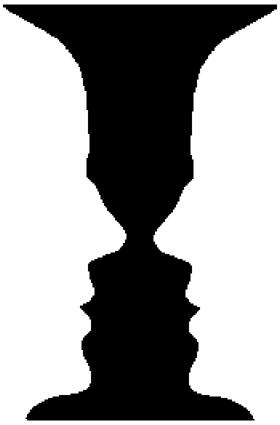
But what constitutes convincing movement, and how can a mobile robot exhibit such behavior? In the context of public events with participating robots, it is argued that *communicating the intent of the motion* is of primary importance. Under the assumption that humans can effectively navigate in crowds because we are able to sense such intent in others, it follows that a mobile robot should have a certain presence and indicate where it is headed. This is expected to facilitate reaching the goal, even if the robot has to nudge its way through a crowd. Goal directedness does not necessarily mean using the geometrically

Figure 1.2: In 1997, chess world champion Garry Kasparov was defeated by the Deep Blue computer system. And yet, the considerable effort that went into developing a program that could outperform a specific human in this intellectual game with clearly defined rules, raises the question of whether it was really the computer or rather the team of scientists and engineers who won.

shortest path. It might be an advantage to circumvent regions with a lot of environment dynamics, or even better to go there on purpose if the flow of people goes into the right direction.

This line of thought can be summarized as human-inspired motion behavior. If the robot should convince humans with its movements, then these movements should reflect properties of human motion to some extent. This is reminiscent of another field of endeavor which aims at making machines behave like humans. Artificial intelligence (AI) focuses on reproducing (and surpassing) human *intellectual* skills such as logical reasoning or theorem proving. This immaterial concept of AI is reflected in the formulation of the Turing test [57], which is intended to be independent from non-intellectual properties of the investigated system. It is also apparent in the amount of effort invested to create a computer system that could beat a human at chess (figure 1.2).

In the field of artificial intelligence, it is common to distinguish “hard” from “easy” problems. And it is also common to note the curious phenomenon that the difficulty of a given task is often the opposite of what researchers first expect! A possible explanation of this frequent mis-evaluation lies in the fact that subconscious processes seem easy to humans, because we are not aware of the effort involved. Hard AI problems that were initially thought to be easy have a tendency to include subconscious processes. For example, computer vision seemed easy at first – in his plenary speech during the International Conference on Robotics and Automation 2003, Takeo Kanade mentioned that at the beginning of his career, he once expected a student to solve object recognition in one semester – but today the computer vision field is far from exhausted and many hard problems remain to be solved, robust object recognition being just one of them. In hindsight this is less surprising: Already at the beginning of the 20^{th} century, Gestaltists described several



OXXXXXXXXXX
XOXXXXXXXXX
XXOXXXXXXXX
XXXOXXXXXXX
XXXXOXXXXXX
XXXXXOXXXXX
XXXXXXOXXXX
XXXXXXXOXXX
XXXXXXXXOXX
XXXXXXXXXOX
XXXXXXXXXXO

Figure 1.3: The Gestaltist effects of figure-ground separation on the left (do you see two faces, or a vase?), and grouping by similarity on the right (the letters “O” are immediately perceived as a diagonal line). Phenomena such as these two are innate in the human perceptive system and have been studied since the turn of the last century. However, these seemingly straightforward effects continue to elude the kind of understanding needed to engineer systems of similar performance.

vision phenomena that still lack satisfactory explanation. Classical examples include regrouping by similarity and distinguishing figure from ground, illustrated in figure 1.3. Such phenomena do not require conscious effort, they seem easy because the human perceptive apparatus, naively put, “just does” the processing. Tasks involving intellectual skills seem harder, but the fact that humans are teaching those skills to others means that the problem and its solution are already formalized, which simplifies programming a machine to perform it. Skills acquired through evolution or infant learning on the other hand are less understood. Robot motion that is convincing for humans can be likened to hard AI problems — most of the time, little or no conscious effort is required to cope with situations that remain tough problems for scientists and engineers.

1.2 Overview and Contribution

After the general introduction in this chapter and the description of related work necessary for situating this thesis (chapter 2), the new contributions to path planning and obstacle avoidance for mobile robots in highly cluttered dynamic environments are divided into two parts. The first part is the motion generation system used on Robox during Expo.02, which is presented in chapter 3 with particular focus on practical aspects of a real-world tour-guiding application. The second, more theoretical, contribution is an interpolated navigation function with dynamic replanning capabilities, presented in chapter 4. Chapter 5 concludes this dissertation by discussing the contributions in an encompassing context.

Chapter 2

Scope and Prior Art

Robot motion planning and obstacle avoidance has been a research topic for around three decades [31]. Manipulator research provides a basis for large parts of the more recent mobile robotics. The amount of related work is relatively important. This chapter presents an overview of literature relevant for this thesis on applied path planning and obstacle avoidance for autonomous mobile robots in dynamic and cluttered environments. It is not intended to be an exhaustive overview of publications on path planning and obstacle avoidance, its scope is limited to the one of this thesis. Consult a good textbook such as [50] for more introductory details[1].

This chapter is organized as follows: Section 2.1 presents the development of this thesis in more or less chronological order, explaining how it fits into activities at the Autonomous Systems Lab (ASL), as well as introducing the most important references in context. Section 2.2 presents very succinct summaries of prior art in form of a catalog which is divided into three subsections according to subject. The first two correspond to the main contributions of this thesis (2.2.1 on motion generation for applications in crowds of people and section 2.2.2 on interpolation and dynamic replanning for navigation functions). The third subsection of the catalog (2.2.3) presents a selection of work, both general and applied, that is relevant when a mobile robot needs to cope with dynamic environments. The works summarized under section 2.2.3 influenced this thesis, but the algorithms have not been implemented on Robox, although some were evaluated in simulation.

2.1 Project Chronology

When technical development for the Robotics@EXPO.02 project started in spring of 2001 at the ASL, it was obvious that its prior path planning and obstacle avoidance system was inappropriate for the envisioned tour-guiding task. It had been the result of a successful four-months diploma work by an exchange student [38] and employed a simplified (rectangular differential drive robots) and reduced (heuristic slicing of velocity space) Dynamic Window Approach (DWA) [17] to follow a path generated using the NF1 planner [31].

---

[1] In particular, chapter 6 “Planning and Navigation”.

NF1 was invoked on sub-goals determined by depth-first search of the a-priori graph-based map developed for localization research at ASL [2]. Robot speeds were heuristically predetermined for each grid cell traversed by the NF1 path to reduce problems with the piecewise linear trajectory. This performed reasonably well in relatively static environments, but lacked the flexibility and smoothness required for operating in dynamic and cluttered environments.

2.1.1 Incremental Development Approach

The duration available for developing a new path planning and obstacle avoidance system was approximately one year, after which a very safe and robust system had to be in place for multi-robot tour-guiding in a crowded exhibition[2]. The bibliography and some of the evaluations taken from the above mentioned diploma thesis [38] proved valuable. However, it was chosen to completely start over for several reasons: The existing system was expected to perform poorly in dynamic environments, it would have been inappropriate to limit the search for smooth and flexible planning, and the system lacked the modularity required for incremental and concurrent engineering[3].

During the evaluation phase of the present PhD thesis, the choice between developing a completely novel approach or taking inspiration from prior art was straightforward: Given the important number of publications, the existing approaches were expected to cover a very broad range of possible technologies, it was thus chosen to concentrate on finding a good combination of existing approaches, supplemented by original work especially for providing a compatible formulations and rigorously addressing practical aspects. The stringent safety and robustness requirements of the Robotics@Expo.02 project quickly lead to discarding technologies that were not based on relatively detailed models of the robot and the environment, because a formal description is necessary to ascertain for instance that the robot can come to a complete stop instead of colliding with a given object.

Once these fundamental technological choices had been made, it was necessary to define a development method that ensured an incremental and modular system: The core requirement being collision avoidance, followed by adaptation to changing environments, motion appropriate for operation in tour-guiding, and smooth trajectories, it was decided to first develop pure obstacle avoidance (which would not address issues with local minima) and subsequently decide on planning and flexible path representation. Additionally, the development of hardware and software for the project were done in parallel, and it was clear that the prototype of Robox was not going to be available anytime soon, in addition it was clear from the outset that the prototype would have to be shared among the dozen developers involved in the project[4]. Avoiding the resource bottleneck of testing on the

---

[2] More details on the requirements can be found in the introduction of chapter 3.

[3] The fact that the new system described in chapter 3 also uses the DWA, but in a modified and complete formulation, is due to a fully independent evaluation of alternative technologies.

[4] For two other PhD students at ASL, the Robotics@Expo.02 project was an important part of their thesis: Björn Jensen worked on Human Robot Interaction [23] and Kai-Oliver Arras was in the final stages of his work on Localization [2]. Other engineers addressed mechanics, electronics, and the large amount of

physical robot is one of the reasons why a simple yet complete two-dimensional simulator was developed in the context of this thesis.

2.1.2 Collision Avoidance Development

Scientific and technical work started by evaluating existing algorithms using published results and simulation of the most promising candidates (in Matlab, C, and C++). This included the Curvature Velocity and Lane-Curvature methods [51, 28], Dynamic Window Approach [17], Nearness Diagram [34], Vector Field Histogram (VFH [6], VFH+ [58], VFH* [59]), Potential Field Approach and variations [26, 25, 14]; as well as work by Schlegel [43], Strobel [54], and Konolige [29] (the latter more for its Local Perceptual Space than the Gradient Method, which subsequently become important for the second part of this thesis).

At an intermediate stage of the Robotics@Expo.02 project, it became apparent that the computational resources available for the real-time controllers on Robox would be largely (over 40%) eaten up by the driver for the SICK laser scanners. Even though few other tasks required hard real-time performance, many others (most importantly localization, monitoring, and transmission of scanner data to the second on-board computer) that were to run as non-real-time threads needed to have a sufficient number of processor cycles at their disposal. It was therefore important that obstacle avoidance and path planning should use computational resources as efficiently as possible. By the time this additional constraint appeared, the DWA was the favorite candidate for the core collision avoidance system by virtue of its model fidelity[5] and the possibility of implementing behavior-switches consistently with collision avoidance[6]. Given the high computational burden of the calculation required for the DWA, in light of its good qualities, it was chosen to discretize the robot's immediate surroundings into a local obstacle grid such that the complex collision predictions could be precalculated and stored in a lookup table for very quick retrieval during operation. The additional grid effects were addressed by using conservative approximations. This was inspired by Schlegel's use of lookup tables in [43].

Once the precalculated tables had been implemented and tested in simulation, the first runs on the physical robot revealed another unanticipated constraint: The XO/2 operating system limits memory allocations to chunks of 256 kilobytes, which could only be respected by choosing overly coarse resolutions for the local obstacle grid and discrete actuator space, even when using single precision (64 bit) floating point values. Also, the XO/2 filesystem is based on TFTP[7] and RAM[8], which lead to the adoption of a compression scheme based on a

---

work for developing a high-performance tour-guiding robot complete with application-specific peripherals and subsystems. For example, multi-robot human interaction and visitor flow management is presented in [21], the design and implementation is published in [48], and the complete project is presented in [49].

[5] Geometric, kinematic, and dynamic constraints are taken into account when predicting collisions.

[6] By acting on the form and relative weights of the objective functions that serve to optimize the actuator commands.

[7] This stands for "trivial file transfer protocol", a simple file serving layer.

[8] 256 megabytes on Robox, no hard disk

modified Lloyd-Max quantizer[9] such that memory would not be wasted for high-resolution lookup tables.

2.1.3 Adding Planning and Path Adaptation

By now the core obstacle avoidance was operational for any non-holonomic robot with a convex polygonal outline. Preliminary evaluations had shown that the elastic band concept [40] was the most promising path representation, due to its clear formalism, flexibility in the face of environmental changes, and the possibility of simplifying it to make it extremely lightweight. Other candidates had been the Gradient Method [29] (abandoned for its slightly inferior smoothness and more intensive computations), and various bug-like algorithms [31] for their very direct sensor-based approach (but they lacked predictability in the highly dynamic context of a mass exhibition). It was also tried to heuristically add sensitivity to environment dynamics as a custom objective function of the DWA, using grid-based motion detection similar to optical flow calculations; this approach was abandoned mainly because its overall performance could not be formally investigated and it was very sensitive to parameter tuning.

It was necessary to decide on a geometric planner because the pre-existing graph-based approach was adapted to localization and could not provide the flexibility required for programming scenographical motion elements[10]. Given the quick convergence of the simplified elastic band, as opposed to the slow convergence mentioned in the original formulation [40], it was chosen to use the NF1 for lightweight planning and let the elastic smooth the resulting path.

At this stage the motion generation system on the physical Robox (prototype) performed well in the lab, but there still had been no test in a setting cluttered with many moving objects. Simulations with a large number of virtual visitors indicated that the replanning frequency[11] would be higher than necessary: When visitors move into a section of the elastic that is far from the robot, it is often a waste of time to adapt the elastic because by the time the robot reaches that region, the situation is bound to have changed. An application-specific heuristic (the masking distance introduced in section 3.5.1) was invented to create an artificially empty corridor around distant sections of the path.

2.1.4 Planning with Environment Dynamics

Consider the following issue that can arise at the transition between NF1 and elastic band: The NF1 grid is initialized with the current scanner readings[12], the algorithm is run, the

---

[9] A lossy compression scheme used in image processing, based on optimizing the sum of squared errors between a color and it's compressed representation. It has the advantage of very fast decompression.

[10] It required the robot to move along edges and stop on nodes of an a-priori map, which eliminated the possibility of specifying a large class of useful movements, such as "move forward 0.6 meters and turn right 30 degrees."

[11] I.e. the number of times the NF1 has to be invoked for the same goal due to visitor movements that invalidate the existing plan.

[12] Each laser point is blown up by the robot radius to implement $\mathcal{C}$-projection as described in [31].

sequence of grid cells from robot to goal is translated into an initial elastic band, and finally a single update iteration of the elastic is performed[13] for initial smoothing. The smoothing uses a more recent scan than the NF1 initialization, which could cause the initial band to be invalid because of visitor movement.

The risk of wasting a planning cycle in this way grows with the number and speed of visitors as well as with the time required to run the NF1. The former is application dependent and can not be influenced, whereas the latter is influenced by the load of the on-board computer. The system load is hard to influence during operation[14]. Various NF1 delays were tested in simulation under the hypothesis that visitors and robots moved at walking speed, with the informal result that the delay between initialization and smoothing should not exceed 0.5 seconds, at which point more than a quarter of the planning cycles were wasted[15]. During Expo.02 this was ensured by assigning highest priority to the NF1 and elastic band threads. However, a proper solution to this problem requires treating moving objects differently from static ones. It was decided to defer this until after the Robotics event to avoid jeopardizing its safety and robustness.

Extracting reliable information about environment dynamics is one of the results of the thesis on human-robot interaction by Björn Jensen [23]. An approach for using this information for path planning was presented during a workshop at the IJCAI conference 2003 [22]. It employs the Gradient Method by Kurt Konolige [29] to solve a weighted region path planning problem using grid-based wavefront propagation. These results were not included in this thesis for reasons presented below.

2.1.5 A Dynamic Approach to Weighted Region Planning

The implementation of grid-based wavefront propagation for planning with regions that are weighted according to environment dynamics as presented in [22] was not apt to cope with practical aspects due to the large amount of time required for calculations. It also became apparent that the Gradient Method produces paths that are not sufficiently smooth when compared with the results of the elastic band.

The ad-hoc solution to the lack of smoothness produced by NF1 was deemed sufficient for operation during the Robotics event. A better solution with a sound theoretical foundation was developed after Expo.02 and the IJCAI 2003 workshop. An email exchange with Kurt Konolige, the author of the Gradient Method, lead to the inclusion of the Level Set Method formalism by J. A. Sethian [47]. Inspiration was also taken from the D* algorithm by Anthony Stentz [52] in order to make the planning approach flexible for dynamically changing environments. Chapter 4 presents this method, which has been named E*.

---

[13] Adjusting the amount of bubbles (section 3.5.3) and moving bubbles according to artificial forces (section 3.5.4).

[14] For example, it can peak during relocalization cycles that generate a large number of hypotheses, or when the network is under heavy load.

[15] This also depends on the tuning of NF1 and elastic band parameters. It is unfortunate that the pressure shortly before the opening of Expo.02 lead to neglecting the documentation of these tests.

2.2 Publication Catalog

2.2.1 Motion Generation on Robox

Chapter 3 presents the local path planning and obstacle avoidance system that was used on Robox during Expo.02. In order to understand the choices that were made during design and implementation of that system, it helps to be familiar with the publications that were most influential during its development.

The **NF1 navigation function algorithm** is described in textbooks such as (Latombe 1991 [31]). It is a simple grid based path planner.

The Curvature-Velocity Method for Local Obstacle Avoidance (Simmons 1996 [51]): CVM treats obstacle avoidance as a constrained optimization in velocity space. Constraints formalize vehicle dynamics and obstacle information. The optimum is defined in terms of speed, safety, and goal-directedness.

The Dynamic Window Approach to Collision Avoidance (Fox et.al. 1997 [17]): The DWA is very similar to the CVM in the sense that it uses constrained search in velocity space to determine actuator commands. It also trades off speed, safety, and goal-directedness. However, the grid-based representation makes it more straightforward to compute velocity space obstacles, at the cost of increased memory requirements.

The system presented in chapter 3 uses the DWA[16]. The dynamic window is the only hard real-time task of obstacle avoidance on the Expo.02 system, and it is the part that uses the fewest approximations in the geometrical, kinematic, and dynamic models of the robot.

High-Speed Navigation Using the Global Dynamic Window Approach (Brock and Khatib 1999 [9]): Combines the NF1 navigation function and the DWA for goal-directed reactive motion in unknown dynamic environments. In other words, obstacle avoidance is provided by DWA following a globally planned path (which presents the drawbacks of NF1).

Global Nearness Diagram Navigation (GND) (Minguez et.al. 2001 [35]): Partly inspired by the way the Global Dynamic Window Approach extends the DWA, this method adds global reasoning to the *Nearness Diagram* (presented in section 2.2.3). It consists of *Mapping ND* which integrates information in a model of the environment, and *Mapping-Planning ND* which exploits connectivity information of free space using NF1.

These two publications are examples of using a relatively simple global planner in conjunction with high-performance reactive obstacle avoidance in order to avoid local minima. The main drawback of using NF1 is the type of paths it produces, which are not smooth and graze obstacles. This is the reason why the Expo.02 system adds an additional layer, the elastic band.

Elastic Bands: Connecting Path Planning and Control (Quinlan and Khatib 1993 [41]): Aimed at bridging the gap between planning and execution, elastic bands are

---

[16] CVM and DWA being conceptually close (if not identical) and the fact that CVM uses a more compact velocity representation could have led to the adoption of CVM, had the memory problems on Robox appeared earlier in the development phase

a flexible plan representation. Connected *bubbles* of free space [40] are subject to artificial forces: Repulsive forces from obstacles, attractive forces from neighboring bubbles. The elastic band iteratively smoothes the plan and adapts to moving or previously unknown obstacles.

Fast Local Obstacle Avoidance under Kinematic and Dynamic Constraints for a Mobile Robot (Schlegel 1998 [43]): This work treats obstacle avoidance as a constrained optimization problem in the vehicle's actuator velocity space. It trades off speed, goal-directedness, and remaining distance until collision. In order to achieve efficient real-time performance given arbitrary robot shapes, pre-calculated lookup tables are used.

One of the objectives of the system used for tour-guiding with Robox was the extremely short processor time available for the real-time part, coupled with relatively tight memory constraints. Look-up tables were used to address the timing issues. A modified Lloyd-Max quantizer was used to compress the tables.

2.2.2 Dynamic Planning and Interpolation

After the praxis-oriented description of Robox as a tour-guide in chapter 3, chapter 4 is more theoretical. It describes an algorithm that combines the advantages of two approaches (one from mobile robotics, the other from supercomputing) in order to produce smooth navigation functions that provide dynamic replanning capabilities.

A Gradient Method for Realtime Robot Control (Konolige 2000 [29]): A grid-based navigation function with interpolation is used to calculate the optimal path from each cell to the goal using wavefront propagation. NF1 is a special case of the proposed algorithm. At each timestep of the controller, the navigation function is recalculated to take into account environment dynamics and information about previously unexplored regions.

This work provides a relatively smooth navigation function that can be calculated quickly enough to do without dynamic replanning. It became the starting point of the developments that ultimately led to the algorithm described in chapter 4.

Optimal and Efficient Path Planning for Partially-Known Environments (Stentz 1994 [52]): This paper presents the D* algorithm, a graph based approach with efficient replanning for changes in environment representation. Used on grids, D* produces the same paths as NF1 but can "repair" existing paths when new information arrives.

The Focussed D* Algorithm for Real-Time Replanning (Stentz 1995 [53]): An extension of the D* algorithm that changes the order of replanning propagation to make it more efficient. Repairing the path cost information is concentrated on regions that are close to the current position of the robot, making it more likely that the calculations will be useful for its progress towards the goal.

D* is *the* algorithm for dynamic replanning. Formulated on graphs it is applicable to many situations in mobile robotics. However, it lacks a method for interpolating between the nodes and edges, which would be important for smooth motion.

A Fast Marching Level Set Method for Monotonically Advancing Fronts (Sethian 1996 [45]): Level set methods [47] are numerical techniques for computing the position of propagating fronts. A very fast implementation is possible in the case of monotonically advancing fronts whose speed depends on position only (e.g. one that represents possible configurations during robot motion planning).

Computing Geodesic Paths on Manifolds (Kimmel and Sethian 1998 [27]): An extension of the Fast Marching Method from rectangular orthogonal meshes to triangulated domains. It also contains an intuitive summary of the Fast Marching Method.

Note that Fast Marching Methods share a property with D*: Values are calculated in an *upwind* manner. While D* provides a framework for keeping track of upwind direction after propagation, such a mechanism is absent from the other (the direction is implicit in the order of evaluation, but is not stored permanently).

2.2.3 Background

This section presents a small selection of publications that are relevant for the kind of application presented in this thesis.

A Note on Two Problems in Connexion with Graphs (Dijkstra 1959 [13]): A classical paper for planning on graphs with edges of known length. *Dijkstra's Algorithm* allows (*i*) constructing the tree of minimum total length between the nodes and (*ii*) finding the path of minimum total length between two given nodes. A* [31] is a refinement of Dijkstra's algorithm that allows focussed search.

Spatial Planning: A Configuration Space Approach (Lozano-Perez 1983 [33]): A classical paper presenting algorithms for computing constraints on the position of an object due to the presence of other objects, for polygonal or polyhedral geometries. Popularized the notion of *configuration space* $\mathcal{C}$ in motion planning, which contains one dimension per degree of freedom to allow a point-representation of the robot. Once obstacles have been mapped from $\mathcal{W}$ to $\mathcal{C}$, this representation simplifies various aspects of path planning and obstacle avoidance.

The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots (Borenstein and Koren 1991 [6]): VFH uses a two-stage reduction of a local *histogram grid* to calculate control commands that steer the robot towards a *valley* in a *polar obstacle density histogram.* The chosen valley usually is the one closest to the goal direction. When the robot drives around an obstacle, the choice is further influenced by the the direction with which the obstacle is circumvented.

VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots (Ulrich and Borenstein 1998 [58]): Several improvements of the VFH method: (*i*) Obstacles are enlarged by the robot radius and a security distance, (*ii*) a hysteresis is applied on the polar histogram to reduce oscillations between valleys, (*iii*) valleys that require control inputs exceeding actuator limits are blocked, and (*iv*) goal-directedness, path smoothness, and continuity of motor commands are traded off.

VFH*: Local Obstacle Avoidance with Look-Ahead Verification (Ulrich and Borenstein 2000 [59]): Based on VFH+ but additionally predicts robot movements before choosing a motor command. A projected position tree is built and searched using A*.

Real-Time Obstacle Avoidance for Manipulators and Mobile Robots (Khatib 1986 [26]): This paper describes the *Potential Field Approach* (PFA), of which numerous extensions and variations exist (see [31] for a good presentation). The robot is treated like a point that performs gradient descent on an artificial potential field which is constructed such that obstacles generate repulsive forces and the goal exercises an attractive force.

An Extended Potential Field Approach for Mobile Robot Sensor-Based Motions (Khatib and Chatila 1995 [25]): A variation of PFA. By filtering out certain obstacles based on robot shape, orientation, and direction with respect to the goal, some drawbacks are alleviated (e.g. wall following).

Real-Time Path Planning Using Harmonic Potentials in Dynamic Environments (Feder and Slotine 1997 [14]): Closed-form solutions to artificial *harmonic potential* are used to construct collision-free paths given a known model of a dynamic environment. Expressing the environment in terms of harmonic potentials is not straightforward.

Nonholonomic Deformation of a Potential Field for Motion Planning (Sekhavat and Chyba 1999 [44]): Nonholonomic motion planning using approximation methods (i.e. planning a holonomic path, then adapting it to the constraints) can be improved by making the initial plan more appropriate. This paper presents such a method. It derives potential fields for holonomic gradient descent to be subsequently adjusted to nonholonomic constraints.

The Lane-Curvature Method for Local Obstacle Avoidance (Ko and Simmons 1998 [28]): Combines the CVM with the *Lane Method*, which chooses among lanes that divide the environment into zones of direction. CVM is used to follow and switch lanes, given physical limitations and environmental constraints.

Nearness-Diagram Navigation (ND): A new Real Time Collision Avoidance Approach (Minguez and Montano 2000 [34]): ND uses a sectored (polar) environment representation that is used to express distances to obstacles and allows selecting an optimal valley. As navigation strategy, five laws of motion are used, selected on the basis of an interpretation step.

Reactive Collision Avoidance for Navigation with Dynamic Constraints (Minguez et.al. 2002 [36]): Dynamic constraints are used to re-map the spatial representation underlying many obstacle avoidance schemes, resulting in an *Ego-Dynamic Space* and a *Spatial Window*. Reactive schemes can then be generically extended to take vehicle dynamics into account.

Navigation in Partially Unknown, Narrow, Cluttered Space (Strobel 1999 [54]): Directed graph-based path planning with a user-defined optimality criterion. Robot shapes and obstacles are modeled as multi-layered polygons. Graph expansion is based on *standardized path elements* that take into account kinematic constraints. A *branch and bound* method provides speedup of Dijkstra's algorithm.

Motion Planning in Dynamic Environments Using the Relative Velocity Paradigm (Fiorini and Shiller 1993 [15]): Extends $\mathcal{C}$ to allow representing *velocity space* obstacles for circular robots and objects moving with constant velocities. It is a state space representation in which a collision occurs if and only if the robot's configuration is inside a $\mathcal{C}$ obstacle or the tip of its velocity vector is inside a *collision cone*.

Trajectory Planning in a Dynamic Workspace: A 'State-Time Space' Approach (Fraichard 1999 [18]): *State-time space* $\mathcal{ST}$ is inspired by $\mathcal{C}$ and unifies moving obstacles and dynamic constraints of vehicle motion. Searching a solution over a restricted set of canonical trajectories is used to plan a path on a graph embedded in $\mathcal{ST}$.

Collision Prediction and Avoidance Amidst Moving Objects for Trajectory Planning Applications (Bernabeu et.al. 2001 [4]): A planning method that takes into account translational object movements, based on (*i*) an algorithm for calculating distances between *spherically extended polytopes*, (*ii*) recursive subdivision of time intervals, and (*iii*) the hypothesis of constant velocities during planning intervals.

Towards Real-Time Global Motion Planning in a Dynamic Environment Using the NLVO Concept (Large et.al. 2002 [30]): A set of approximations allows extending *velocity space obstacles* of circular objects with constant velocities to non-circular objects on non-linear trajectories. Adding a notion of risk (imminent collisions are more dangerous than distant ones), this allows obstacle avoidance using a cost function on the robot's velocity.

On the Influence of Sensor Capacities and Environment Dynamics onto Collision-Free Motion Plans (Alami et.al. 2002 [1]): In order to guarantee a robot's immobility before collision at *planning time*, a (maximum) velocity profile is calculated. This profile takes into account sensor capacities (e.g. field of view of laser scanner) and the maximum velocity of moving objects. The resulting plans pass doors and corners defensively by preferring greater distances to unseen regions.

Randomized Kinodynamic Planning (LaValle and Kuffner 2001 [32]): Trajectory planning that takes into account kinematic and dynamic constraints is approached using *Rapidly-exploring Random Trees* (RRT), which allows for continuous-domain representation and probabilistically complete planning at the cost of non-optimality. RRTs are a randomized planning technique specially designed for nonholonomic constraints and high dimensional $\mathcal{C}$.

Real-Time Randomized Path Planning for Robot Navigation (Bruce and Veloso 2002 [10]): An example of extending RRTs to interleave planning and execution. This paper presents a method that allows continuous-domain planning for *RoboCup* multi-robot control.

Randomized Kinodynamic Motion Planning with Moving Obstacles (Hsu et.al. 2002 [20]): Planning under kinematic and dynamic constraints with moving obstacles (piecewise constant velocity) by randomly sampling control inputs and integrating the movement equations. The resulting roadmap consists of milestones in $\mathcal{ST}$ and is constructed from scratch for each query.

Other papers of interest are on roadmap planners based on the Hierarchical Generalized Voronoi Graph [11, 12], and the Weighted Region Path-Planning Problem [37, 42].

Chapter 3

Autonomous Motion in Highly Dynamic Cluttered Environments

This chapter presents the path planning and obstacle avoidance approach for mobile robots, developed for the Robotics@EXPO.02 mass exhibition [49][1]. This work has been published in a compact way as [39].

It describes the motion generation method (local path planning and obstacle avoidance) used on the autonomous tour-guide robot *Robox* shown in figure 3.1. The system has proven its value during a 5 month operation of eleven such robots in a real-world application, a very crowded exhibition. Three existing approaches (DWA, elastic band, NF1) have been integrated into a system that performs smooth motion efficiently, in the sense of computational effort as well as goal-directedness. This chapter gives detailed description of how these algorithms have been adapted to the tour-guiding task, and includes aspects of software engineering.

3.1 Robox at the Swiss National Exhibition

The Swiss National Exhibition takes place approximately every 30 to 40 years. For its 2002 edition, the Autonomous Systems Lab at the Swiss Federal Institute of Technology in Lausanne was involved in the Robotics@EXPO.02 project aimed at bringing the world of robotics closer to the visitors. This was achieved through an exhibition about robotics in a wide sense, with a special scenographical twist: Mobile robots served as tour guides, creating a sort of auto-reference in the frame of the exhibition. For this purpose, eleven interactive robots named Robox were built and programmed. Among the numerous capabilities required for such interactive tour guide robots, some could be adopted with little changes from existing systems of the Autonomous Systems Lab. Among the skills that needed to be written from scratch, local motion planning and obstacle avoidance had to be tackled and resulted in an important part of this thesis. In the following, the specific requirements of this navigation subsystem are presented.

---

[1] `http://robotics.epfl.ch/`

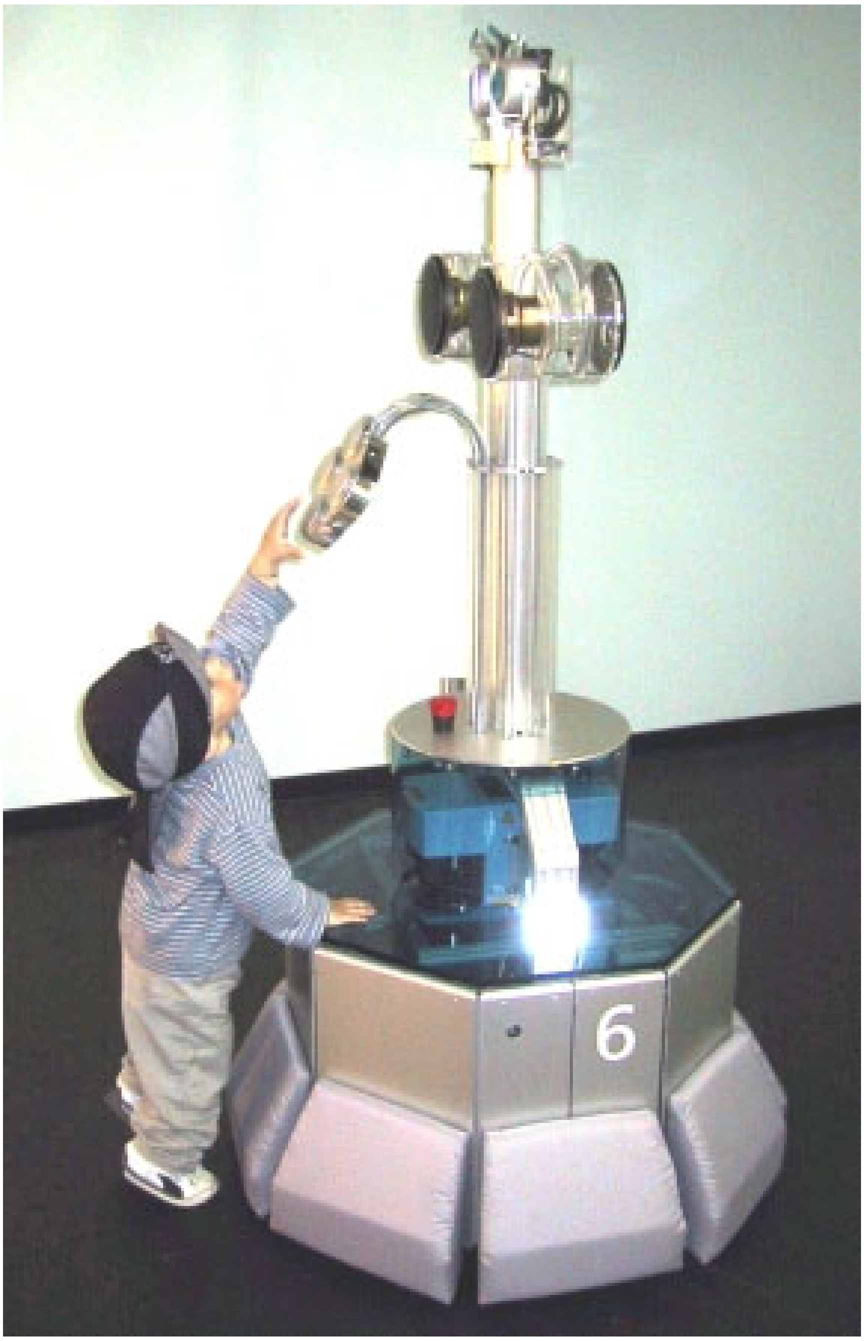


Figure 3.1: Robox was developed at the Autonomous Systems Lab for tour-guiding during the Robotics@Expo.02 event.

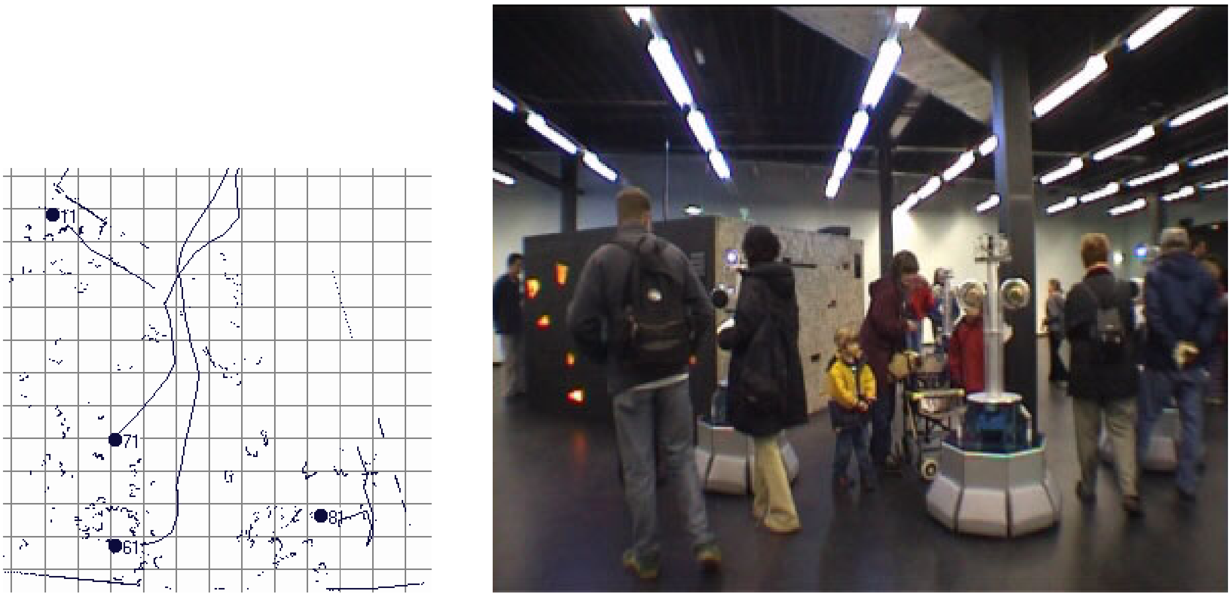


Figure 3.2: Environment clutter at Expo.02: On the left is a laser scanner snapshot, with scan points (dots), robot locations (large numbered dots), and their trajectories (lines). It shows raw data from all robots transformed to their respective poses, the grid resolution is 1m. On the right is a video still showing a typical scene.

A tour-guide robot has to be able to move autonomously, acquire the attention of the visitors and interact with them efficiently. Usually, the environment is known and accessible, but the visitors make it highly cluttered and dynamic. Robox's navigation subsystem comprises an embedded Power PC G3 at 380 MHz running the XO/2 real-time operating system [8], two SICK laser scanners, 8 contact sensors with soft bumpers and a differential drive architecture.

3.2 Objectives of Robotics@Expo.02

Imagine an exhibition area of approximately $300m^2$ filled with around 100 persons of all ages, with an expected flow of up to 500 visitors per hour. Figure 3.2 shows a video still from Expo.02 and a snapshot of laser scans. In order to provide visitors with a novel and enriching experience, the goal was to have 11 interactive mobile robots giving individual tours to ad-hoc groups in a free-flow exhibition. This scenario has several implications for planning and control of the mobile robot:

- Safety for visitors, personnel, and robots is of paramount importance in a public exhibition, for ethical reasons as well as for administrative requirements and insurance policies.

- Human movements are hard to predict and the high visitor density results in ever changing freespace configurations.

- Even though the set of possible goal locations is limited to the places of interest in the exposition, it is not possible to pre-program the routes. Visitors get to chose which exhibit they want to visit next.

- Expo.02 policy required a certain visitor flow (400 to 500 per hour), the robots were supposed to ensure this movement by using appropriate interaction modalities and ego-motion.

- Robox being one of the attractions and the only interactive part of the Robotics exhibition during Expo.02, it was expected that people would play with the machine out of curiosity. This expectation turned out to be correct, many visitors could not resist the urge to try and trap the robot.

- Having multiple robots with identical capabilities means that deadlock situations (two robot trying to reach the same goal location) have to be avoided.

- The laser sensors of all Robox are mounted at the same height, leading to a difficulty of perceiving other robots in ways suitable for simple free-space calculations. Reflective bands were used to circumvent this problem, but lead to new issues: The detection of the reflectors depends on the relative orientation of the robots and on occlusions by visitors.

- A similar aspect stems from the group of visitors following a robot, it can be regarded as an obstacle influenced by that robot's movements, albeit in ways that are hard to formalize.

- Several processes need to run on the robot, resulting in relatively tight computational constraints for motion planning and execution.

- Reliable collision avoidance taking into account geometric, kinematic, and dynamic properties of the robot.

- The delay between a visitor's goal request and the start of the robot's movement should be unperceivable.

- Robot movements should be dynamic but not threatening.

This list gives an impression of the challenges encountered. However, not all these aspects were addressed on the level of motion planning and execution. For instance, safety considerations and regulations led to the use of a redundant controller circuit that would short-circuit the phases of the robot's drive motors e.g. in case of bumper contact (see [55] for more details on safety features of Robox). This somewhat alleviated the responsibility of the obstacle avoidance software, but also created new challenges for it: The robot could not nudge its way through visitors in an overly insistent manner, because each contact with too much force would trigger the emergency brake.

Translational speed	$\leq 0.6\text{m/s}$
Translational acceleration	$\leq 0.6\text{m/s}^2$
Rotational speed	$\leq 2.5\text{rad/s}$
Rotational acceleration	$\leq 2.5\text{rad/s}^2$
Frequency of the obstacle avoidance process	$\geq$ 10Hz
Peak processor load for motion generation	$<$ 30%
Goal request response time	$<$ 0.25s

Table 3.1: Technical requirements of motion generation (path planning and obstacle avoidance) for Robotics@Expo.02

In short, the collision risk must be low and the eventual effects of a collision be harmless. Smooth motion is important, as visitors anticipate movement when they follow the guide. The obstacle avoidance control loop should be fast in order to not only run in real-time, but also leave enough processing resources to other modules such as localization, sensor acquisition, web server and motor control. Table 3.1 summarizes the requirements in more technical terms.

3.3 Approach

Path planning and obstacle avoidance have been treated in previous works. Similar to [9, 43, 3], existing algorithms were evaluated and the chosen ones combined such that their drawbacks cancel out as much as possible and their advantages cover the requirements. The main contribution here is a consistent fusion of the interfaces between the sub-tasks without compromising any component's functionality. Also, the used methods were modified to either improve a component or their interaction. For instance, real-time performance and memory usage was optimized without effect on other components by using transparently compressed look-up tables.

The task of the motion planner was divided into two layers, one to supply a path plan, the other to follow it while taking into account the exact geometry of the robot, its kinematics, and the dynamics of its actuators.

From the candidates for the reactive level [6, 14, 17, 44, 51], the dynamic window (DWA) [17] was chosen because of its physically meaningful representations (actuator speeds and accelerations, robot geometry) and one-step calculation of rotational and translational speed.

Among the works [9, 24, 29, 35, 41] studied for the planning layer, or for their ability to solve reactive and planning problems at the same time, the elastic band [41] approach was chosen for Robox. Its path representation is compact and physically meaningful, smooth, and designed to accommodate environment dynamics. The NF1 [31] is used to generate the initial plan, the elastic band helps to compensate for its drawbacks (grazing of obstacles, unsmooth paths). Elastic band and NF1 taken together are used for local path planning on Robox, whereas global path planning employs a graph-based a-priori map[2].

Figure 3.3 shows the flow of information and control in the motion planner. The DWA is a real-time (RT) control loop at $10Hz$, the elastic band is a non-RT loop that typically runs at $5Hz$ and the NF1 is calculated upon request and can take up to $0.5s$. Non-RT execution times depend on the path length, the clutter in the environment and the overall system load.

3.4 Dynamic Window

The DWA generates actuator commands such that the robot does not collide with obstacles, the commands do not violate the dynamic capabilities of the actuators, and the robot follows the elastic band.

In our implementation, the robot shape (polygon) is defined at startup (instead of being hard-coded). Additionally, a significant speed-up and a predictable maximum cycle time have been achieved by calculating a look-up table for the collision prediction, also during startup. This vital part of the DWA would otherwise constantly require expensive computations of varying numbers of intersections between circles and lines, up to 90'000 on Robox. A similar idea can be found in [43].

3.4.1 Velocity Space

Robox is a differential drive robot. The kinematic model and its inverse are given in (3.1) and (3.2).

$$\boldsymbol{v}(\dot{\boldsymbol{q}}) = \begin{bmatrix} \dot{s}(\dot{q}_l, \dot{q}_r) \\ \dot{\theta}(\dot{q}_l, \dot{q}_r) \end{bmatrix} = \begin{bmatrix} \frac{R_{\text{wheel}}}{2}(\dot{q}_l + \dot{q}_r) \\ \frac{R_{\text{wheel}}}{D_{\text{base}}}(\dot{q}_r - \dot{q}_l) \end{bmatrix} \tag{3.1}$$

$$\dot{\boldsymbol{q}}(\boldsymbol{v}) = \begin{bmatrix} \dot{q}_l(\dot{s}, \dot{\theta}) \\ \dot{q}_r(\dot{s}, \dot{\theta}) \end{bmatrix} = \begin{bmatrix} (\dot{s} + \frac{D_{\text{base}}}{2}\dot{\theta})/R_{\text{wheel}} \\ (\dot{s} - \frac{D_{\text{base}}}{2}\dot{\theta})/R_{\text{wheel}} \end{bmatrix} \tag{3.2}$$

where R_{wheel} is the radius of the drive wheels, D_{base} is the wheel base, $(\dot{q}_l, \dot{q}_r)$ are the rotational speeds of the left and right wheel and $(\dot{s}, \dot{\theta})$ are the translational and rotational speeds of the robot. On Robox, $R_{\text{wheel}} = 0.09m$ and $D_{\text{base}} = 0.521m$

Using the actuator space $\dot{\boldsymbol{q}} = (\dot{q}_l, \dot{q}_r)$ properly models the acceleration and speed limits of the actuators (figure 3.4), as opposed to the usual $\boldsymbol{v} = (\dot{s}, \dot{\theta})$.

3.4.2 Obstacle Model

Obstacles are represented as points $\boldsymbol{p}_j$ in a set P (3.3). This representation is appropriate for the two SICK laser scanners that serve as main input to MotionPlanner on Robox. They have high resolution (1 mm, 0.5°) and low noise (less than 1 cm).

---

[2] The a-priori map was designed for localization, it contains topological and geometric information (feature visibility, position, distance between nodes). Depth-first search is used for planning on this map.

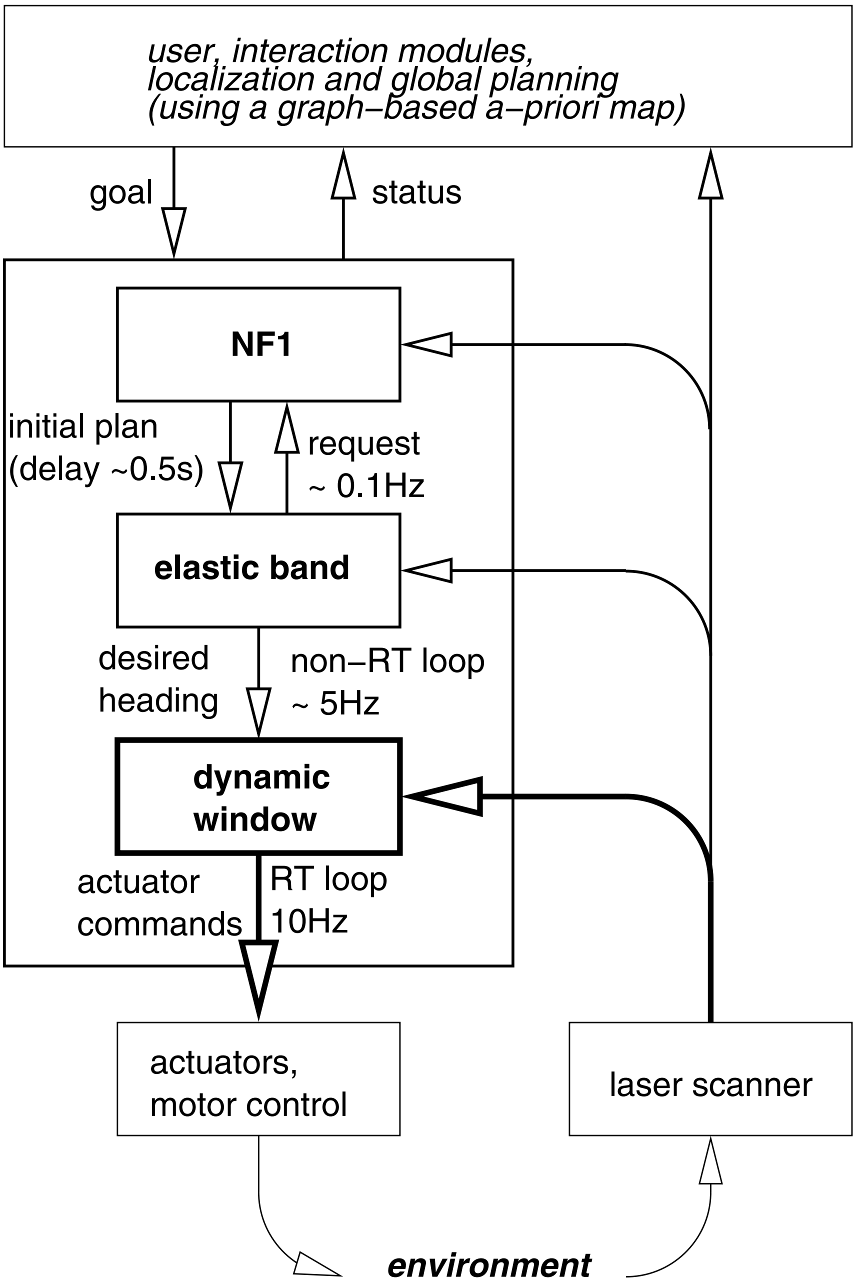


Figure 3.3: Diagram of motion planning and control loops and how they are integrated into the overall navigation architecture. Deliberative levels specify a global goal position to the motion planner and can query its status. The output is in the form of actuator commands sent to the motor control level.

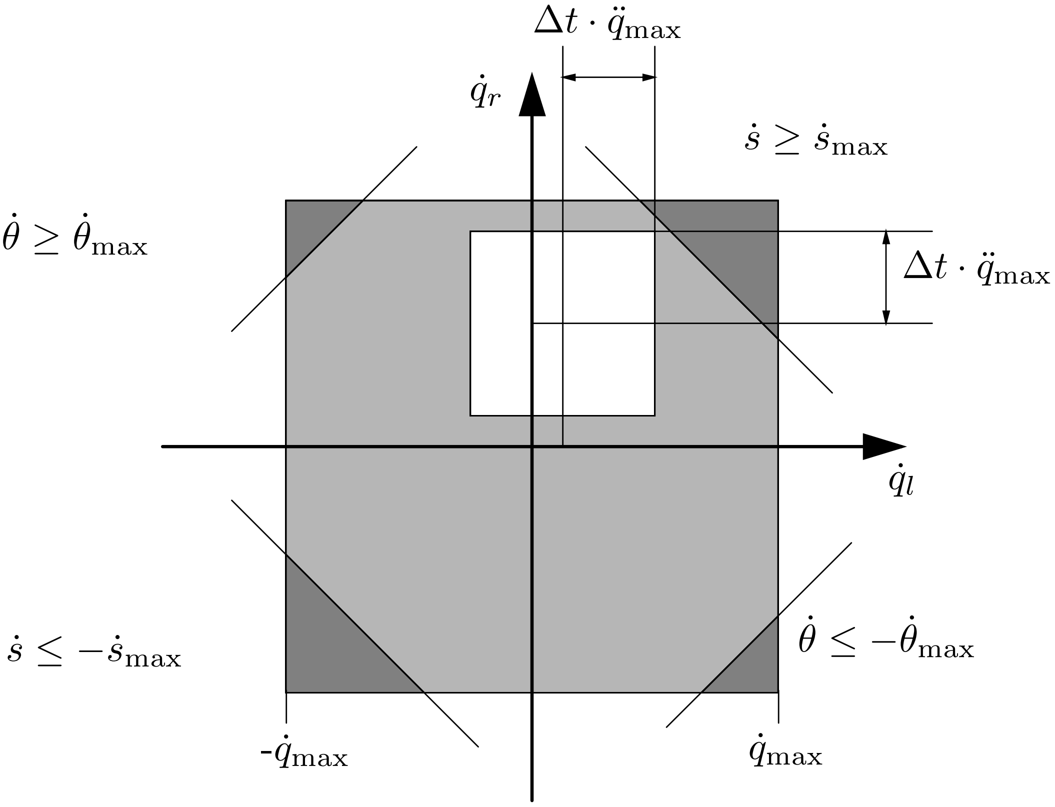

$\Delta t \cdot \ddot{q}_{\max}$
$\dot{q}_r$
$\dot{s} \geq \dot{s}_{\max}$
$\dot{\theta} \geq \dot{\theta}_{\max}$
$\Delta t \cdot \ddot{q}_{\max}$
$\dot{q}_l$
$\dot{s} \leq -\dot{s}_{\max}$
$\dot{\theta} \leq -\dot{\theta}_{\max}$
$-\dot{q}_{\max}$
$\dot{q}_{\max}$

Figure 3.4: Definition of the dynamic window: $(\dot{q}_l, \dot{q}_r)$ are the actuator speeds, $\dot{q}_{\max}$ is the maximum actuator speed, $\ddot{q}_{\max}$ the maximum actuator acceleration, Δt the time-step of the control loop, $(\dot{s}, \dot{\theta})$ the robot speed in Euclidean space, $\dot{s}_{\max}$ and $\dot{\theta}_{\max}$ are global speed limits. Dark gray regions are forbidden by $\dot{s}_{\max}$ or $\dot{\theta}_{\max}$, the light gray region are the available speeds and the white square is an example of reachable actuator speeds at a given moment. Admissible speeds are those inside the white square which would not lead to a collision. On Robox, $\dot{q}_{\max} = 6.5rad/s$, $\ddot{q}_{\max} = 6.5rad/s^2$, $\Delta t = 0.1s$, $\dot{s}_{\max} = 0.6m/s$ and $\dot{\theta}_{\max} = 2.5rad/s$

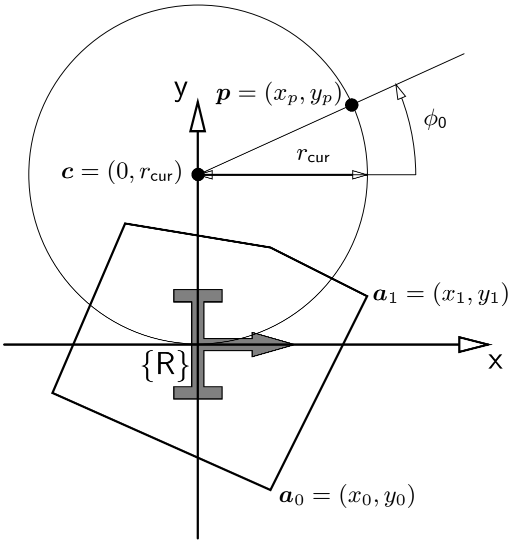

y
$\boldsymbol{p} = (x_p, y_p)$
ϕ_0
r_{cur}
$\boldsymbol{c} = (0, r_{\text{cur}})$
$\boldsymbol{a}_1 = (x_1, y_1)$
{R}
x
$\boldsymbol{a}_0 = (x_0, y_0)$

Figure 3.5: Collision prediction – definition of required terms, based on the hypotheses of static obstacles and a trajectory of constant curvature until collision.

$$P = \{\boldsymbol{p}_j\},\ \boldsymbol{p}_j = \begin{pmatrix} x_{pj} \\ y_{pj} \end{pmatrix} \tag{3.3}$$

3.4.3 Collision Prediction

The collision prediction in [17] calculates the distance to travel before hitting an obstacle. This is not applicable to pure rotations because any collision would seem instantaneous. This problem was solved by using the time until collision, which does not present such a singularity. As a side effect, the same geometric distance appears closer at high speeds, effectively adding a buffer distance proportional to speed.

General Case

Figure 3.5 shows the required items for predicting collisions. Under the assumption of immobile obstacles and robot motion at constant curvature, the trajectory of the obstacle in the mobile robot frame is circular with center on the y-axis and is characterized by these parameters:

- $\boldsymbol{c} = (0, r_{\text{cur}})^T$ is the center of the obstacle trajectory, with $r_{\text{cur}} = \dot{s}/\dot{\theta}$ the radius of curvature of the robot movement.
- $r_{\text{gir}} = \sqrt{x_p^2 + (y_p - r_{\text{cur}})^2}$ the giration radius of the obstacle $(x_p, y_p)^T$ around $\boldsymbol{c}$.
- $A = \{a_i\},\ a_i = \{x_{0i}, y_{0i}, x_{1i}, y_{1i}\}$ is the robot outline given as line segments from $(x_{0i}, y_{0i})^T$ to (x_{1i}, y_{1i}^T). The notation $\boldsymbol{a_0} = (x_0, y_0)^T$ and $\boldsymbol{a_1} = (x_1, y_1)^T$ is used in the

development for calculating the intersection between the circle and one line segment.

- $\phi_0 = \arctan2(y_p - r_{\text{cur}}, x_p)$ gives the starting angle of the obstacles movement along the circle. Depending on the sign of $\dot{\theta}$, the obstacle moves clockwise or counter-clockwise.

In order to predict the collision, the intersection between a line segment (robot outline) and a circle (obstacle trajectory w.r.t. the robot) is calculated (3.4). The intersection is converted to an angle ϕ that measures how far the obstacle moves along the circle until hitting the outline (3.5). The direction of the movement needs to be taken into account when determining ϕ.

$$\begin{aligned} &\|\boldsymbol{a_0} + \lambda(\boldsymbol{a_1} - \boldsymbol{a_0}) - \boldsymbol{c}\|^2 = \|\lambda\boldsymbol{\Delta a} - \boldsymbol{\Delta c}\|^2 = r_{\text{gir}}^2 \\ &\qquad\Leftrightarrow\ \lambda^2\boldsymbol{\Delta a}^2 - 2\lambda\boldsymbol{\Delta a}\boldsymbol{\Delta c} + \boldsymbol{\Delta c}^2 - r_{\text{gir}}^2 = 0 \end{aligned} \tag{3.4}$$

where $\lambda_{1,2} \in [0,1]$ is a parametric form of the intersections between the line and circle, $\boldsymbol{\Delta a} = \boldsymbol{a_1} - \boldsymbol{a_0}$, and $\boldsymbol{\Delta c} = \boldsymbol{c} - \boldsymbol{a_0}$. Note that there can be 0, 1, or 2 solutions, and that a numerically stable implementation of the quadratic equation is used (described in appendix A.1). In case of two solutions, the one leading to the *smaller* collision time t_{coll} is used in (3.5).

$$\begin{aligned} \phi &= \begin{cases} \phi_0 - \arctan2(y_0 + \lambda\Delta y - r_{\text{cur}}, x_0 + \lambda\Delta x) & \Leftarrow\ \dot{\theta} > 0 \\ \arctan2(y_0 + \lambda\Delta y - r_{\text{cur}}, x_0 + \lambda\Delta x) - \phi_0 & \text{otherwise} \end{cases} \\ t_{\text{coll}} &= \frac{\phi \bmod 2\pi}{\dot{\theta}} \end{aligned} \tag{3.5}$$

where $\Delta y = y_1 - y_0$ and $\Delta x = x_1 - x_0$. t_{coll} is the time until collision for this particular line segment. In order to determine the collision time for the whole robot outline, it is necessary to loop over all segments and keep the smallest t_{coll}.

Pure Translational Case

Pure rotations are no problem for the presented method. Straight line movement however would result in an infinite radius of curvature. This causes the calculations to break down, which is why a numerical guard is used in the implementation: If $|\epsilon\dot{s}| \geq |\dot{\theta}/\epsilon|$ (which would mean $|r_{\text{cur}}| \geq 1/\epsilon^2$), a straight line approximation is used instead, replacing the circle by a horizontal half-infinite line anchored at the obstacle point, and the collision prediction now involves an intersection between two lines (3.6). The numerical threshold on Robox during EXPO.02 was $\epsilon = 10^{-9}$ and is used in several places.

$$\boldsymbol{p} + \mu \boldsymbol{e_\mu} = \boldsymbol{a_0} + \lambda \boldsymbol{\Delta a}$$

$$\mu \geq 0,\ \lambda \in [0,1],\ \boldsymbol{e_\mu} = \begin{cases} (1,0)^T & \Leftarrow\ \dot{s} < 0 \\ (-1,0)^T & \text{otherwise} \end{cases}$$

$$t_{\text{coll}} = \mu / \dot{s} \quad (3.6)$$

3.4.4 Objective Functions

The DWA chooses among admissible commands by maximizing an objective function on the sampled $(\dot{q}_l, \dot{q}_r)$ space. The usual sub-objectives are used: Heading, speed and clearance.

Heading Objective

The heading objective w_{head} makes the robot follow the elastic band or, once the goal radius has been reached, orient itself along a specified direction. Figure 3.6 illustrates the calculations involved. It is important to include the braking time in the pose prediction in order to avoid oscillations. The pose prediction is precalculated for all valid motion commands and stored in a lookup table for quick retrieval during real-time execution.

Transforming the elastic band to a direction is delicate because the real-time DWA task and the elastic band update thread are not synchronized due to the computational constraints mentioned earlier. If one simply took the direction from the robot's position to the first bubble[3], the heading error $d\theta$ would not be reliable: It would jump at each update of either the DWA (robot position) or the elastic band (first bubble). It was thus chosen to use as intermediate goal the position of the first bubble that lies outside the robot radius[4].

Speed Objective

The speed objective is a linear function $w_{\text{speed}} = w_{\text{speed}}(\dot{s})$ shown in figure 3.7. Like the heading objective, the speed objective w_{speed} can be switched at run-time (more details in section 3.4.5). These three behaviors are stored in lookup tables to allow changing the behavior during runtime using a single pointer assignment.

Clearance Objective

The clearance objective w_{clear} tends to maximize the space between robot and obstacles. It measures by how much the collision prediction exceeds the braking time for $\dot{\boldsymbol{q}}$, see (3.7) and (3.8). This function is illustrated in figure 3.8.

---

[3] Bubbles encode the elastic band, they are introduced in section 3.5.1. Each bubble represents a point along the planned path, and associated freespace information.

[4] The R_{robot} parameter is the largest distance from the wheel axle center to the robot's outline. It can be interpreted as a $\mathcal{C}$-projection and is used to simplify the NF1 and elastic band calculations.

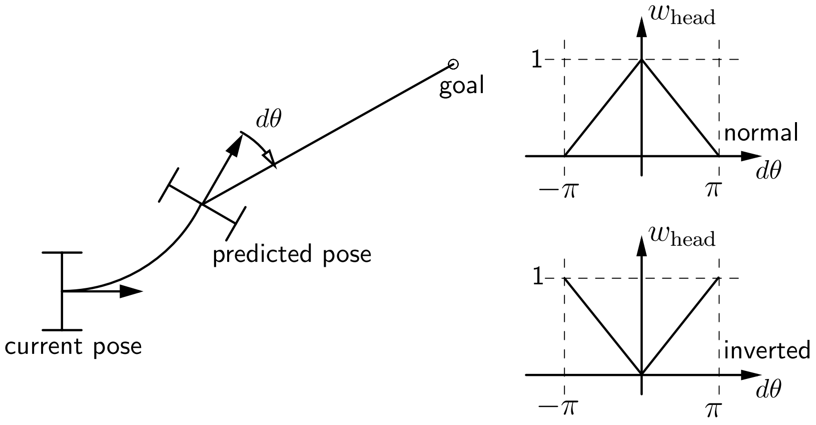


Figure 3.6: Calculating the heading objective w_{head}. For each motion command, the robot's movement is predicted until standstill. The predicted pose is then used to calculate the heading error $d\theta$ (shown on the left), which in turn is normalized into an objective value (shown on the right). The inverted heading objective is used for moving backwards (see section 3.4.5). The pose prediction is based on one time-step at the given motion command, followed by a deceleration at constant radius of curvature until full stop. If the pose was not predicted until standstill, an oscillating behavior would result.

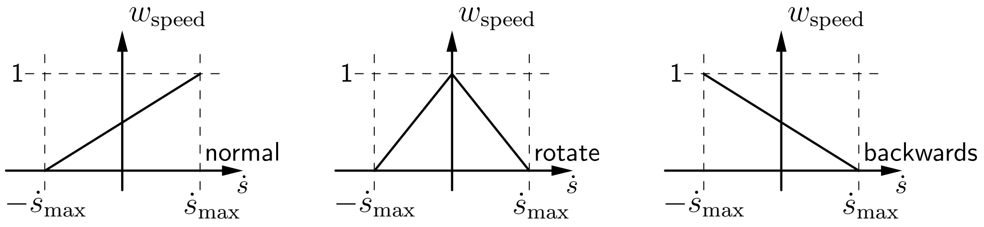


Figure 3.7: Calculating the speed objective depends on the wanted behavior. Usually, it is $w_{\text{speed}} = (\dot{s} + \dot{s}_{\max})/2\dot{s}_{\max}$ to make the robot move forwards. The two alternatives are $w_{\text{speed}} = |\dot{s} - \dot{s}_{\max}|/\dot{s}_{\max}$ for turning on the spot and $w_{\text{speed}} = (\dot{s}_{\max} - \dot{s})/2\dot{s}_{\max}$ for moving backwards.

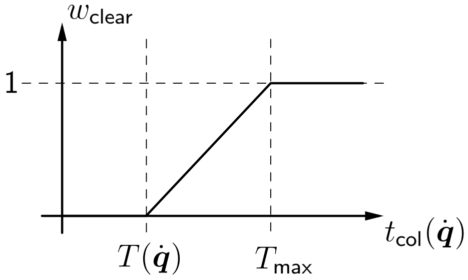


Figure 3.8: Calculating the clearance objective is behavior independent. Collision times lower than the current brake time are mapped to zero as they would surely lead to collisions. This makes it straightforward to detect non-admissible motion commands.

$$w_{\text{clear}}(\dot{\boldsymbol{q}}) = \begin{cases} 0 & \Leftarrow t_{\text{col}} \leq T(\dot{\boldsymbol{q}}) \\ \frac{t_{\text{col}} - T(\dot{\boldsymbol{q}})}{T_{\text{max}} - T(\dot{\boldsymbol{q}})} & \Leftarrow T(\dot{\boldsymbol{q}}) < t_{\text{col}} < T_{\text{max}} \\ 1 & \text{otherwise} \end{cases} \tag{3.7}$$

$$T(\dot{\boldsymbol{q}}) = max(\dot{q}_l, \dot{q}_r)/\ddot{q}_{\text{max}} \tag{3.8}$$

where t_{col} is the collision prediction for $\dot{\boldsymbol{q}}$ given the current sensor readings, $T(\dot{\boldsymbol{q}})$ is the braking time when traveling at $\dot{\boldsymbol{q}}$ and $T_{\text{max}} = \dot{q}_{\text{max}}/\ddot{q}_{\text{max}}$ is the braking time at maximum speed ($T_{\text{max}} = 1s$ during EXPO.02). Speeds where w_{clear} is zero are constraints: They are flagged as non-admissible because they would surely lead to collision.

Overall Objective

The overall objective w^* (3.9) is a weighted sum of the sub-objectives. The next motion command (3.10) is chosen to maximize w^*.

$$w^* = \alpha_{\text{clear}} w_{\text{clear}} + \alpha_{\text{speed}} w_{\text{speed}} + \alpha_{\text{head}} w_{\text{head}} \tag{3.9}$$

$$\dot{\boldsymbol{q}}^* = \arg(\max_{\dot{\boldsymbol{q}}}(w^*)) \tag{3.10}$$

where α_{clear}, α_{speed} and α_{head} define the relative weights of the sub-objectives. During EXPO.02, values were set to $\alpha_{\text{clear}} = 0.5$, $\alpha_{\text{speed}} = 0.1$ and $\alpha_{\text{head}} = 0.1$.

3.4.5 Switching Movement Behavior

Three kinds of behavior should be displayed by the motion generation subsystem of Robox: Moving forwards or backwards along circular arcs or straight lines, and turning on the spot. In order to ensure collision avoidance for all these behaviors, they are implemented by switching between variants of the heading and speed objective.

Turning on the spot is feasible most of the time due to the robot's octagonal shape and outer ring of soft bumpers, which can be considered a circle in most circumstances[5]. It is used to orient the robot along the wanted heading at a goal position, or to align it with the elastic band if the heading error becomes too large. To use this behavior, w_{speed} is switched to preferring low translational speeds over high ones. The influence of w_{head} is then sufficient to choose between turning left or right. However, w_{clear} can cause the optimal motion command to contain a translational component in some cases. If this ever becomes problematic, this issue can be addressed by decreasing α_{clear} or defining a stricter w_{speed}.

Moving forwards is the usual case during movement. In some cases it is necessary to make the robot move backwards, which can be achieved by inverting both w_{speed} (such that it prefers negative translational speeds) and w_{head} (such that it prefers a heading error of $\pm\pi$). If w_{head} is not inverted, then Robox backs away from the goal, whereas not inverting w_{speed} would makes it turn around and move away from the goal.

3.4.6 Grid Effects

Care has to be taken in order to circumvent unwanted effects of discretized motion commands: If the highest objective values are not achieved for pure translations (for forward and backward movements) or pure rotations (for turning on the spot), then the robot will rapidly switch back and forth from going slightly left to right during translation, or moving slightly forward and backward during rotation. These effects stem from an interaction between w_{head} and w_{speed}, the latter pushes the optimum towards high speeds regardless of (small) heading errors, whereas the former influences the optimum only once sufficient error has accumulated. By choosing the velocity space discretization with care, it is possible to avoid this unwanted behavior. Figure 3.9 compares the wanted with the unwanted situation. The implemented way of avoiding the unwanted effects is to use soft speed limits $\dot{s}^*_{\max}$ (3.11) and $\dot{\theta}^*_{\max}$ (3.12) that are adjusted towards smaller values if the specified ones are inappropriate.

$$\begin{aligned}
\dot{q}_l = \dot{q}_r = \dot{q} = \frac{2\dot{q}_{\max}}{N-1} i,\ i \in \{\dots, -1, 0, 1, \dots\} & \\
\dot{s} = \frac{2R_{\text{wheel}}\dot{q}_{\max}}{N-1} i \leq \dot{s}_{\max} \ \Rightarrow\ i \leq i^* = \left\lfloor \frac{N-1}{2R_{\text{wheel}}\dot{q}_{\max}} \dot{s}_{\max} \right\rfloor & \\
\dot{s}^*_{\max} = \begin{cases} \dot{s}_{\max} & \Leftarrow \frac{2R_{\text{wheel}}\dot{q}_{\max}}{N-1}(i^* + \frac{1}{2}) \leq \dot{s}_{\max} \\ \frac{2R_{\text{wheel}}\dot{q}_{\max}}{N-1}(i^* + \frac{1}{4}) & \text{otherwise} \end{cases} & \qquad (3.11)
\end{aligned}$$

---

[5] This is the main reason for which it is possible to plan and represent the path as if the robot was holonomic.

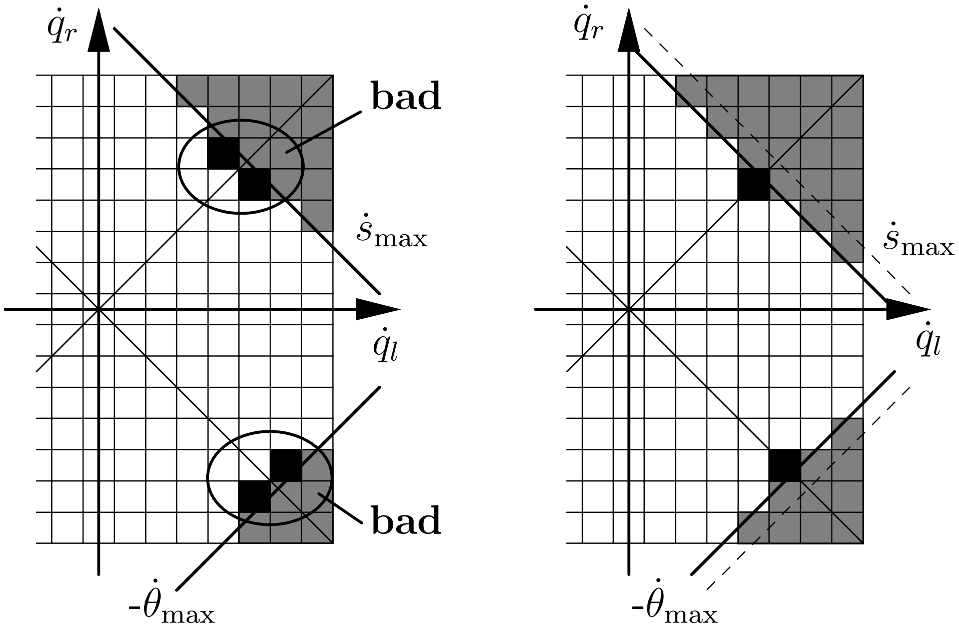


Figure 3.9: Avoiding grid effects of w_{head} and w_{speed} by decreasing $\dot{s}_{\text{max}}$ and $\dot{\theta}_{\text{max}}$. On the left, the soft speed limits lead to a border which does not include cells from the diagonals that correspond to pure translations or rotations. The sketch on the right shows the situation after adjustment.

where N is the dimension of the velocity space grid. Note that this should always be an odd number in order to guarantee that stillstand $\dot{\boldsymbol{q}} = \mathbf{0}$ is part of the possible motion commands. The adjustment for $\dot{\theta}_{\text{max}}$ is very similar.

$$
\begin{aligned}
-\dot{q}_l = \dot{q}_r = \dot{q} &= \frac{2\dot{q}_{\text{max}}}{N-1} i,\ i \in \{\dots,-1,0,1,\dots\} \\
\dot{\theta} &= \frac{4R_{\text{wheel}}\dot{q}_{\text{max}}}{D_{\text{base}}(N-1)} i \le \dot{\theta}_{\text{max}} \;\Rightarrow\; i \le i^* = \left\lfloor \frac{D_{\text{base}}(N-1)}{4R_{\text{wheel}}\dot{q}_{\text{max}}} \dot{\theta}_{\text{max}} \right\rfloor \\
\dot{\theta}^*_{\text{max}} &= \begin{cases} \dot{\theta}_{\text{max}} & \Leftarrow \frac{4R_{\text{wheel}}\dot{q}_{\text{max}}}{D_{\text{base}}(N-1)}(i^* + \frac{1}{2}) \le \dot{\theta}_{\text{max}} \\ \frac{4R_{\text{wheel}}\dot{q}_{\text{max}}}{D_{\text{base}}(N-1)}(i^* + \frac{1}{4}) & \text{otherwise} \end{cases}
\end{aligned}
\tag{3.12}
$$

3.5 Elastic Band

The elastic band (illustrated in figure 3.10) is responsible for path representation and adapting the plan to the robot's movement and changes in the environment. During Robotics@EXPO.02, path lengths rarely exceeded 20 meters. In contrast to the DWA, the elastic band is not a hard real-time task, but a high priority thread. Whereas the former relies on lookup tables to ensure a low and deterministic maximum time for the computations, the latter presents more variability in its time and space requirements.

Each bubble (see section 3.5.2) of the band can be influenced by up to 722 laser points. The number of bubbles varies as a function of the distance to the goal, the clutter and movement in the environment, and even the movement of the robot (as that influences which previously unseen objects appear). Nevertheless, the elastic band should be updated with a frequency on the order of the one used for the DWA[6].

3.5.1 Bubbles

The elastic band B is a directed list of waypoints with associated free space information stored in bubbles b_i (3.13).

$$B = \{b_i\},\ b_i = \{\boldsymbol{c}_i, r_i, m_i\}\ ,\ i = 0 \ldots M-1 \tag{3.13}$$

where $\boldsymbol{c}_i$ is the center of a circle of radius r_i, m_i is a masking distance, and M is the number of bubbles in the band. The masking distance can be used to make the elastic band more stable in environments where most obstacles are dynamic, as was the case during EXPO.02 (details in section 3.5.2). The need for m_i has been presented in section 2.1.3: Environment dynamics far down the planned path can invalidate the elastic, even though by the time the robot arrives at that location the situation would probably have cleared up. m_i has a linear relationship (3.18) to path length L_i (3.17) and it can be tuned to specific environments. The effect of the masking distance is that certain obstacles $\boldsymbol{p}_j$ are ignored when calculating the radius of a bubble.

3.5.2 Obstacle Masking

Each b_i has an associated set of masked obstacles $\{\boldsymbol{p}_{m,ij}\}$ defined in (3.14). The obstacle $\boldsymbol{p}_i^*$ closest to b_i is found (3.15) and determines r_i (3.16).

$$\{\boldsymbol{p}_{m,ij}\} = \{\boldsymbol{p}_j : \parallel \boldsymbol{c}_i - \boldsymbol{p} \parallel > m_i\} \tag{3.14}$$

$$\boldsymbol{p}_i^* = \arg\left(\min_{\boldsymbol{p} \in \{\boldsymbol{p}_{m,ij}\}} \parallel \boldsymbol{c}_i - \boldsymbol{p} \parallel\right) \tag{3.15}$$

$$r_i = \min_{\boldsymbol{p} \in \{\boldsymbol{p}_{m,ij}\}} \parallel \boldsymbol{c}_i - \boldsymbol{p} \parallel \tag{3.16}$$

The masking distance m_i has a linear relationship to the bubble's position L_i along the path, see (3.17) and (3.18).

$$L_i = \sum_{j=1}^{i} \parallel \boldsymbol{c}_{bj-1} - \boldsymbol{c}_{bj} \parallel \tag{3.17}$$

---

[6] Simulations and experience during EXPO.02 indicate that elastic band performance decreases noticeably if its update frequency falls below approximately 3Hz in environments with humans moving at walking speed.

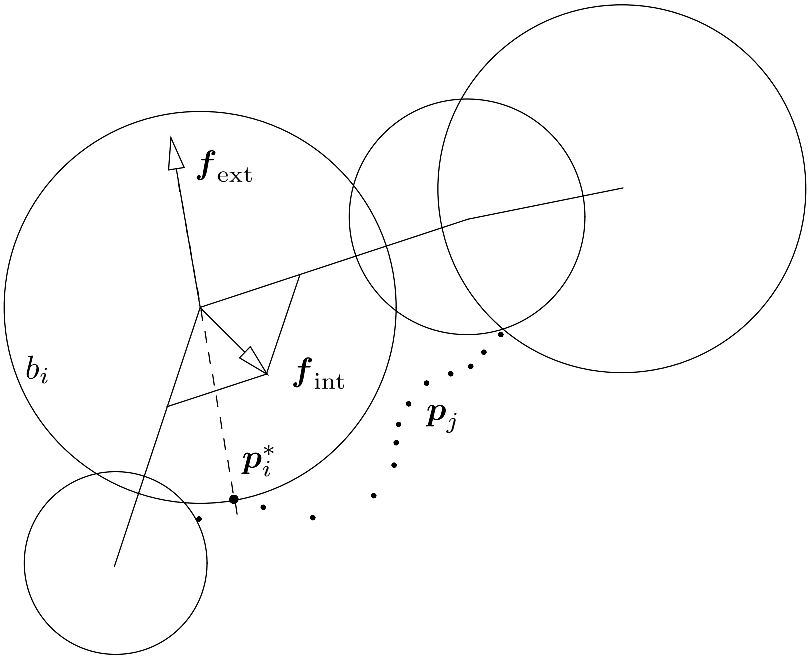


Figure 3.10: An elastic band is a directed list of bubbles b_i which represent subregions of free space. Bubbles evolve under the influence of artificial forces $\boldsymbol{f}_{\text{int}}$ (internal forces to smooth the band) and $\boldsymbol{f}_{\text{ext}}$ (external forces to increase clearance). Obstacles are a point cloud $\{\boldsymbol{p}_j\}$ and each bubble b_i has an associated closest obstacle $\boldsymbol{p}_i^*$ which determines the radius r_i and the direction of $\boldsymbol{f}_{\text{ext}}$.

$$m_i = m_{\max} \cdot \begin{cases} 0 & \text{if } L_i \leq L_{\min} \\ 1 & \text{if } L_i \geq L_{\max} \\ \frac{(L_i - L_{\min})}{L_{\max} - L_{\min}} & \text{otherwise} \end{cases} \tag{3.18}$$

where $L_{\min}$ and $L_{\max}$ define the cumulative path lengths over which m_i stretches and $m_{\max}$ is the maximum distance at which readings can be ignored. During Expo.02, values were set to $L_{\min} = 2.0m$, $L_{\max} = 8.0m$ and $m_{\max} = 8.5m$.

3.5.3 Amount of Bubbles

The elastic band is a dynamic representation. Under the effect of virtual forces (section 3.5.4) it iteratively adapts to novel sensory information, be it due to moving obstacles or changes in occlusion resulting from the robot's motion. This implies that M is not constant — bubbles need to be inserted when the overlap between adjacent bubbles is not sufficient, and superfluous bubbles should be removed from the band in order to keep computational effort small. Determining the overlap is based on calculating circle intersections.

Suppose that the robot travels along a straight line between the centers of two adjacent bubbles. In order to ensure that this can be done without collision, the overlapping region between the bubbles needs to be large enough to let the robot pass. Figure 3.11 shows the cases that can arise, with parameter definitions for the non-degenerate case. In order to simplify the following developments, the chosen frame of reference has its origin at $\boldsymbol{c}_i$ and its x-axis contains $\boldsymbol{c}_j$.

$$\begin{aligned} &\hat{x}^2 + \hat{y}^2 = r_i^2 \;\cap\; (\hat{x} - d)^2 + \hat{y}^2 = r_j^2 \\ &\qquad\Rightarrow \begin{cases} (i) & \text{no overlap} & \Leftarrow \; d > r_i + r_j \\ (ii) & \text{inclusion} & \Leftarrow \; d > |r_i - r_j| \\ (iii) & \text{concentric} & \Leftarrow \; d < \epsilon \\ (iv) & \hat{x}^2 - 2\hat{x}d + d^2 + r_i^2 - \hat{x}^2 = r_j^2 & \text{otherwise} \end{cases} \\ &\qquad\qquad\qquad\qquad (iv) \;\Rightarrow \begin{cases} \hat{x} = \frac{r_i^2 - r_j^2 + d^2}{2d} \\ \hat{y} = \sqrt{r_i^2 - \hat{x}^2} \end{cases} \end{aligned} \tag{3.19}$$

where $(\hat{x}, \hat{y})$ is the intersection point and $d = \|\boldsymbol{c}_i - \boldsymbol{c}_j\|$ is the Euclidean distance between the two centers.

Removing Bubbles

Superfluous bubbles are the ones that are not strictly necessary to guarantee sufficient free space along the elastic band. They are removed such that space and time requirements for updating the band are reduced. This is done by looping over all bubbles and calculating the intersection between a given bubble b_i and it's second next neighbor b_{i+2}:

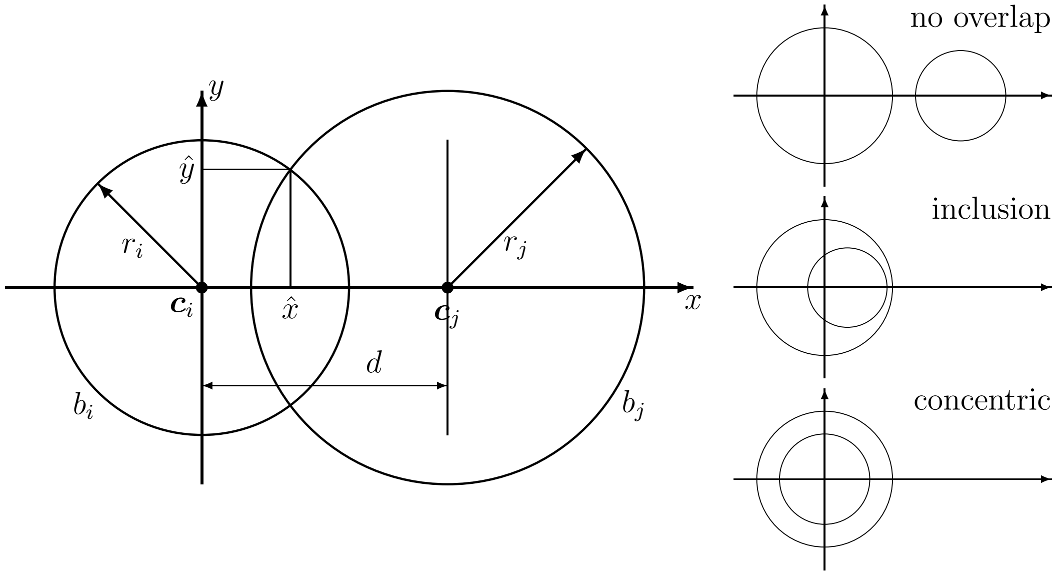


Figure 3.11: Circle intersection parameter definitions and special cases. The non-degenerate case requires calculating $(\hat{x}, \hat{y})$ to see whether the robot passes. In the case of no overlap the band might be fixed by inserting a bubble (which can also be attempted in the non-degenerate case). The inclusion and concentric cases need not necessarily be distinguished from each other, both imply that the robot has enough clearance (provided that it fits into both bubbles).

(i)	no overlap:	leave b_i in place;
(ii)	inclusion:	remove b_i;
(iii)	concentric:	remove b_i;
(iv)	otherwise:	remove b_i if $2\hat{y} < W_{\text{rem}}$, else leave it in place.

The parameter W_{rem} is used to tune the removal frequency of superfluous bubbles. It is bigger than the robot width W_{robot} in order to keep some spare coverage in the band[7]. On Robox, $W_{\text{robot}} = 1.8m$ and $W_{\text{rem}} = 1.8W_{\text{robot}} = 3.24m$.

Adding Bubbles

After removing extra bubbles, the remaining bubbles have to be checked for sufficient coverage. This time the calculation is performed on b_i and its direct neighbor b_{i+1}:

(i)	no overlap:	insert a bubble with $\lambda = \frac{1}{2} + \frac{r_i - r_{i+1}}{2d}$;
(ii)	inclusion:	do nothing;
(iii)	concentric:	do nothing;
(iv)	otherwise:	if $2\hat{y} > W_{\text{add}}$ then insert a bubble with $\lambda = \frac{\hat{x}}{d}$.

The new bubble is inserted between b_i and b_{i+1}, with center at $c_b = \lambda c_{bi} + (1 - \lambda) c_{bi+1}$. The parameter W_{add} has a role very similar to W_{rem} and allows tuning when bubbles are added. On Robox, $W_{\text{add}} = 1.2W_{\text{robot}} = 2.16m$.

3.5.4 Artificial Forces

To reduce the computational load, Euclidean distances and forces whose magnitude is proportional to the distance between c_{bi} and $\boldsymbol{p}_i^*$ are used. A non-linear magnitude would produce smoother evolution of the elastic band, at the cost of more calculations. Linear forces are acceptable because the DWA ensures the dynamic, kinematic and geometric constraints. More importantly, certain obstacles are masked out to make the algorithm more stable in highly dynamic environments, where movement far from the robot can "snap" the elastic band unnecessarily.

This simplification has negative implications because linear forces lead to oscillating behavior of the elastic band. In applications with more computational resources available for motion generation, a smoother model should be used [40][8].

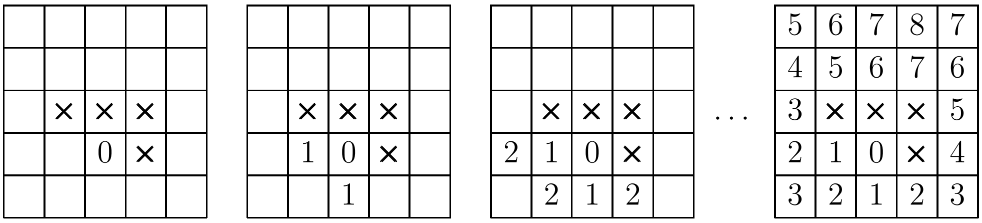
The equations for the internal (3.20) and external (3.21) forces determine the iterative movement $\boldsymbol{c}_{i,t+1} = \boldsymbol{c}_{i,t} + \Delta\boldsymbol{c}_i$ of the bubbles (3.22). The first bubble follows the robot's position and the last bubble is immobile at the goal. Only the two direct neighbors of a bubble are used to calculate the internal force acting on it: $j = i \pm 1$ in equation 3.20.

$$\boldsymbol{f}_{\text{int},ij} = \alpha_{\text{int}} \cdot \begin{cases} 0 & \Leftarrow \|\boldsymbol{c}_i - \boldsymbol{c}_j\| \leq \epsilon \\ \frac{\boldsymbol{c}_j - \boldsymbol{c}_i}{\|\boldsymbol{c}_i - \boldsymbol{c}_j\|} & \text{otherwise} \end{cases} \tag{3.20}$$

---

[7] Otherwise the band would easily snap after a removal.

[8] An earlier work by the same author [41] also uses linear expressions.



	×	×	×	
		0	×	

initialized

	×	×	×	
	1	0	×	
		1		

step 1

	×	×	×	
2	1	0	×	
	2	1	2	

step 2

. . .

5	6	7	8	7
4	5	6	7	6
3	×	×	×	5
2	1	0	×	4
3	2	1	2	3

finished

Figure 3.12: NF1 example, the goal is initialized to 0 and obstacles are denoted with "×". On the left is the initialized grid; the grid after completing the algorithm is shown on the right. By filling the grid with consecutive values of connected neighbors, a discrete navigation function is constructed.

$$\boldsymbol{f}_{\text{ext},i} = \alpha_{\text{ext}} \cdot \begin{cases} 0 & \Leftarrow \; r_i \leq \epsilon \;\cup\; r_i \geq r_{\text{lim}} \\ \frac{r_{\text{lim}} - r_i}{r_i}(\boldsymbol{c}_i - \boldsymbol{p}^*{}_i) & \text{otherwise} \end{cases} \tag{3.21}$$

$$\Delta \boldsymbol{c}_i = \alpha_{\text{tot},i} \cdot \left(\boldsymbol{f}_{\text{int},i,i-1} + \boldsymbol{f}_{\text{int},i,i+1} + \boldsymbol{f}_{\text{ext},i}\right) \tag{3.22}$$

$$\alpha_{\text{tot},i} = \begin{cases} 1 & \Leftarrow r_i > r_{\text{lim}} \\ \frac{r_i}{r_{\text{lim}}} & \text{otherwise} \end{cases} \tag{3.23}$$

where r_{lim} is a parameter that defines the distance at which the elastic band starts to react to obstacles, α_{int} and α_{ext} are parameters that define how strong these forces are, $\alpha_{\text{tot},i}$ makes smaller bubbles move smaller amounts and ϵ is used to avoid divisions by zero. During EXPO.02, values were set to $r_{\text{lim}} = 1.6m$, $\alpha_{\text{int}} = 0.1$, $\alpha_{\text{ext}} = 0.1$.

3.6 NF1

Creating an initial plan from the robot's current position to the goal is the task of the NF1. It is a grid-based method that constructs a navigation function (illustrated in figure 3.12), which can be considered an approximate potential field with a globally unique minimum at the goal. Once this navigation function is calculated, the robot can reach the goal by descending along the gradient of this function. The idea underlying the NF1 is rather simple: Divide the environment into equally sized grid cells, mark all cells that lie within one robot radius of an obstacle (laser readings), then construct a monotonically increasing potential starting at the cells that are in the goal region. It can be implemented as a cell-labeling method described in appendix A.2. If no path is found, replanning is tried again after doubling (then tripling etc.) the width of the grid (keeping the resolution constant)[9].

As described in section 2.1.4, only the most current scan is used to initialize the NF1 grid with obstacle information. Dynamic objects are treated the same as dynamic objects,

---

[9] The width of the grid is reset to the base value after a path has been found.

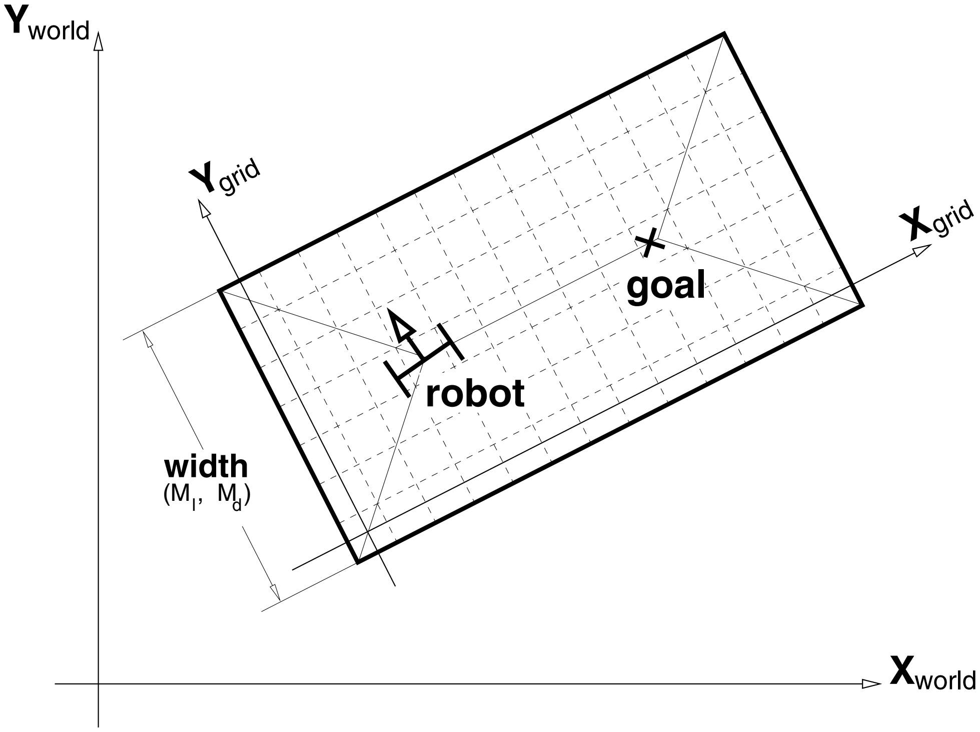


Figure 3.13: NF1 grid placement: After transforming a point $\boldsymbol{p} = (x, y)^T$ from world to grid frame, a linear relation of the form $i = \mathrm{round}((x + d_{\mathrm{off}})/\Delta)$ is used to determine its grid indices (i, j). The width parameters M_{l} (distance) and M_{d} (dimension) are tunable, the length adapts to the situation.

leading to the issues with potentially wasted planning cycles and other problems presented in that section, such as the incomplete view of the environment's topology. Chapter 4 presents an approach that can be used to solve this problem, however it was developed after Expo.02 and could thus not be tested as extensively as the system described in the present chapter.

3.6.1 Grid Based Environment Representation

The robot's $\mathcal{C}$ space is approximated by a bounded two-dimensional grid containing binary obstacle information. It is made up of square cells which are either free or occupied, and the robot is considered a point on this grid. It is further assumed that the robot configuration is constrained to the center points of the cells, which is not strictly necessary if the cell size is taken into account during the $\mathcal{C}$ transform. Using $\mathcal{C}$ projection to ignore the robot's orientation (reducing the dimension from 3 to 2) requires approximating its shape by circumscription.

Figure 3.13 shows how global coordinates are mapped to grid indices. The geometric width M_{l} and number of cells M_{d} along the width are user-defined (the cell size is thus $M_{\mathrm{l}}/(M_{\mathrm{d}} - 1)$). The grid length reflects the distance from robot to goal at the time of grid creation. The alignment of grid the axes with the direction from robot to goal is an application-dependent optimization which reduces grid-artifacts in presence of an unstructured obstacle distribution[10]. Values used during EXPO.02 are $M_{\mathrm{l}} = 5m$ for the width and $M_{\mathrm{d}} = 41$ for the grid dimension along the width. The cell size is calculated accordingly ($12.5cm$ in this case).

3.6.2 Path Properties

The resulting path presents certain drawbacks and advantages when compared to other methods, particularly sensor-based potential field approaches. Most prominent of these is the fact that methods based purely on (local) potential fields can get trapped in local minima, but NF1 makes the robot graze obstacles and move along straight segments joined by angles that are multiplies of $\pi/4$. More details can be found in appendix A.2.

3.7 System Integration

3.7.1 Perception

The main sensor input for the motion planner comes from the laser scanners. Bumpers are also used, but only to stop the robot if it touches something. The robot also stops if a laser reading indicates that an object is inside the robot's outline. The outline is modelled as a convex polygon (Robox has octogonal shape). It would be straightforward to extend the existing system to use a set of convex polygons for robots with concave shapes.

All calculations are based on the latest sensor data only, without any memory effects. This is acceptable in the Robotics pavilion because the layers above the motion planner provide subgoals which lie close to each other along a topologically feasible path and most obstacles are visitors and tend to move out of the robot's way sooner or later.

Having more than one robot in the same space adds the problem of how robots can detect each other. Each Robox has the laser scanners at the same height. This is problematic because even if another robot is detected, it's not seen as the actual outline. This was addressed by attaching reflector bands in the blind zone between the laser scanners. The intensity of the laser data is used to detect the reflectors on other robots, which are then injected as virtual "ghost" points into the laser data to represent the approximate shape of surrounding robots[11].

Ghost points are not only useful for avoiding other robots, they were also used to keep Robox from leaving the exhibition space. By placing ghost walls at the entry and exit

---

[10] In other kinds of environments, another alignment should be chosen. For instance, typical indoor environments with straight rectangular walls would benefit from aligning the grid with the wall direction.

[11] The current implementation simply creates 16 points on a circle of radius $R_{\mathrm{robot}} = 0.9m$ centered at the center of gravity of a cluster of consecutive bright scan points. The reflector results in brightness which is an order of magnitude higher than that of usual objects, so detection is not a major issue. An example can be seen in figure 3.19

(stored in the global map), the DWA refuses to move the robot outside the allowed area even if visitors try to make it go there.

However, this second application of ghost points relies on localization, which can be an issue in settings where the robot might loose track of its position. For example, at ASL we use reflector bands to mark a descending stairway: The robot should never attempt to go there – but if localization fails, the goal location given by the global planner does not correspond to the wanted physical location and might make the robot attempt to move down the stairs. In this example, ghost walls would obviously be useless because the situation only arises when the robot is lost.

3.7.2 Multitasking and Time Constraints

In order to ensure obstacle avoidance, at least a part of the program has to run in real-time. If there are no guarantees about the maximum execution time (or minimum frequency of controller loops), even the most sophisticated models can lead to collisions. This is obvious if you consider the worst case: During a given iteration in the absence of obstacles the system will choose a high speed, and if an obstacle appears at a later time the obstacle avoidance task has to iterate again.

However, it is neither necessary nor advisable to constrain the whole system to follow such timing constraints. For instance, running the path planning thread is only necessary if a new plan is required, otherwise it would only cost computational resources without improving the avoidance performance. The importance of separating the motion generation system into hard real-time tasks and less constrained threads (typically using priority driven scheduling) thus becomes apparent.

This is especially true if the computational resources need to be shared among the large number of subsystems on sophisticated robots. For instance, the PowerPC on Robox runs drivers for sensors and actuators, maintains network connections for remote control and data access, contains several threads related to localization, global and local path planning, and obstacle avoidance – to name just a few. Figure 3.14 is an overview of the most important subsystems on Robox. This diagram reflects a redesign of the system architecture that had already been developed on predecessors of Robox at the Autonomous Systems Lab, but not all components were ported to the new layout.

Obstacle avoidance is one of the safety critical tasks [55]. It is responsible for avoiding injuries and damage to surrounding living beings and objects, as well as damage to the robotic system itself. However, if all computational resources were spent on avoiding collisions, the robot would not be able to perform any other task. This would make the whole endeavor rather pointless, so the time available to other tasks should be maximized.

Hard real-time tasks are the ones that provide the least flexibility for the scheduler. Other threads could be suspended in favor of more important ones, for instance preferring localization over serving a web page, but sacrificing any real-time thread could lead to violations of safety constraints. This discussion should make it clear that obstacle avoidance itself should be subdivided such that a minimum amount of instructions require real-time scheduling. Less critical parts can be high or even medium priority threads, but the core

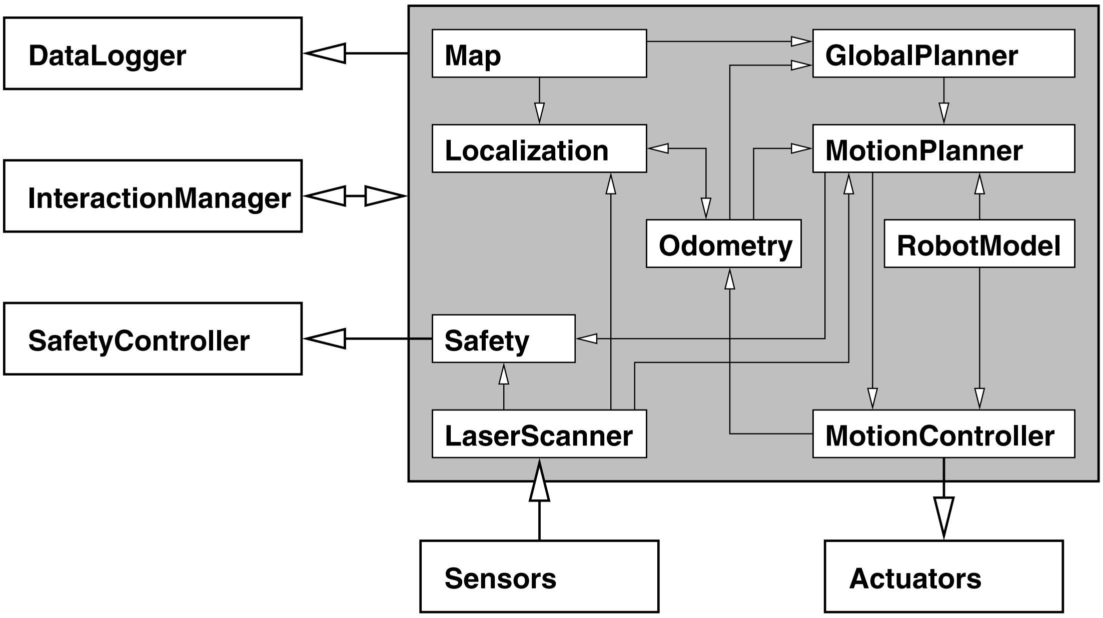


Figure 3.14: Subsystems on Robox, arrows indicate direction of information flow. DataLogger and InteractionManager are on separate computers connected via TCP/IP to the navigation system running on a PowerPC under XO/2. SafetyController is running on a PIC for redundant control of safety critical tasks. Other boxes denote a selection of components.

set of calculations that determines the speed commands must be run in real-time context. This raises two questions:

1. What constitutes this core avoidance subsystem?
2. How can it be integrated into the overall scheme, which includes non real-time threads?

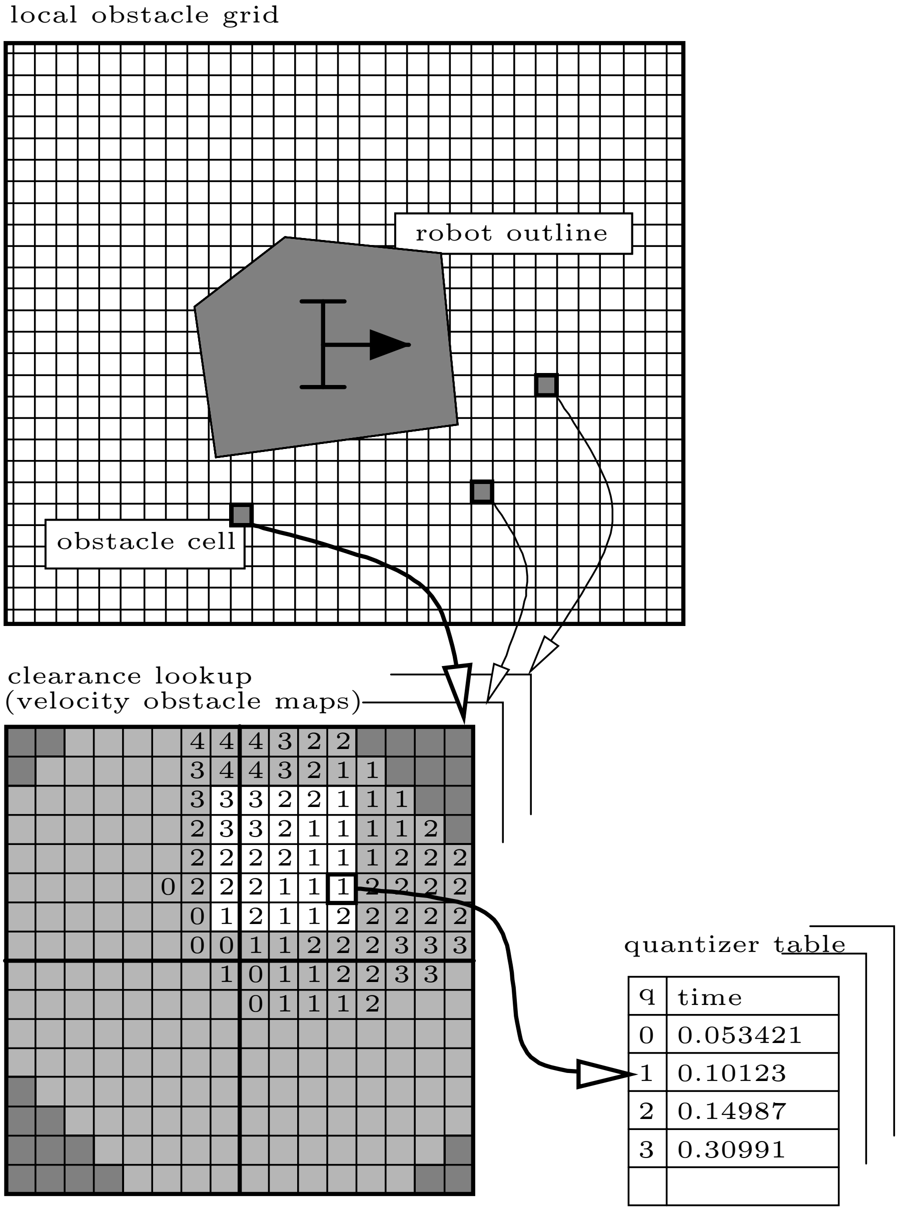
The first question is addressed in detail in section 3.4. The second question calls for techniques of inter-process communication and synchronization. Specific solutions have to be found for problems such as modifying the goal or adapting a plan to unexpected obstacles during motion, and the overhead required for this synchronization should not place too much of a burden on the real-time task, nor make it violate its timing requirements due to dependency on non real-time events or worse yet race conditions. A related issue is making the core algorithm as efficient and deterministic as possible, in order to leave resources to other subsystems and provide guarantees about collision avoidance.

3.7.3 Memory Constraints

Apart from the timing aspects, memory can be a tight constraint for real-time tasks. Especially allocating memory inside such tasks must be considered forbidden, even though certain operating systems might allow it in principle. Allocation is simply too unpredictable, particularly on garbage collected systems. Another issue with memory allocation is the maximum chunk size, which can rule out otherwise straightforward space-time tradeoffs for accelerating the computations. A concrete example is described below, it was developed after it turned out that the required four-dimensional lookup table could not be allocated in a single chunk as that would have lead to an unacceptably coarse resolution. In addition, the available amount of RAM was largely used up to allow diskless operation and garbage collection, so memory was a very scarce resource. To circumvent that particular issue, a method for transparent compression and fragmentation had to be developed.

Two grids are involved in our implementation of DWA, illustrated in figure 3.15. The local obstacle grid samples the workspace around the robot (projected to two dimensions). Each of its cells contains a pointer to another two-dimensional structure, the lookup table which maps motion commands to collision times for obstacles contained in that cell. In other words, the overall structure is four-dimensional. It maps workspace points to velocity space ($\mathcal{V}$) obstacles. The $\mathcal{V}$ transform involves looping over the local obstacle grid and finding the minimum collision time for each motion command.

During initialization, the center points of the cells are used to calculate the time until collision, which can lead to over-estimations if the actual point lies closer to the outline. As a remedy, the robot outline is grown by half the cell diagonal to counter this. Remaining collision prediction errors are always under-estimations of the time until collision: In the worst case, the robot stops too early, but never too late.



						4	4	4	3	2	2				
						3	4	4	3	2	1	1			
						3	3	3	2	2	1	1	1		
						2	3	3	2	1	1	1	1	2	
						2	2	2	2	1	1	1	2	2	2
					0	2	2	2	1	1	1	2	2	2	2
						0	1	2	1	1	2	2	2	2	2
						0	0	1	1	2	2	2	3	3	3
							1	0	1	1	2	2	3	3	
								0	1	1	1	2			

q	time
0	0.053421
1	0.10123
2	0.14987
3	0.30991

Figure 3.15: Look-up operations in the dynamic window. Obstacles are stored in a two-dimensional grid. Each cell (ij) stores associated velocity obstacle information in a collision table $t_{\mathrm{coll},ij} = t_{\mathrm{coll},ij}(\dot{q}_l, \dot{q}_r)$. Transforming obstacles from $\mathcal{W}$ to actuator space (fig. 3.4) involves finding $\min(t_{\mathrm{coll},ij})$ over all (ij) that are occupied. In other words, for each obstacle cell, we iterate over its clearance map and keep the smallest value at each motion command $(\dot{q}_l, \dot{q}_r)$. This is denoted by the three arrows in the upper part.
Compression is quantizer based: The clearance lookup contains 8-bit indices into an associated table of actual collision times. This is denoted with the arrow pointing from the velocity obstacle map to an entry in the quantizer table. The latter is generated using a variant of the Lloyd-Max algorithm.

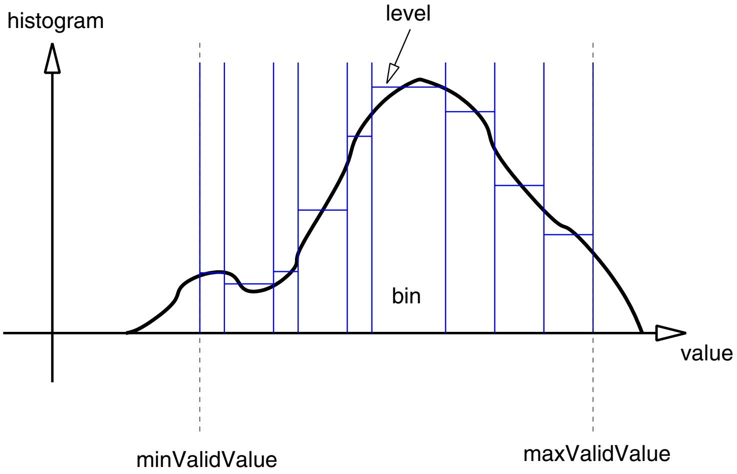


Figure 3.16: The Lloyd-Max quantizer replaces values (represented here by their histogram) by a reduced number of bins. It minimizes the total squared error between the level of a bin and the values contained therein.

Compressed Lookup Tables

Two layers of look-up operations are involved (figure 3.15), compression applies to the second indirection, where collision times are retrieved for a given motion command (dimensions three and four of the overall structure). These compressed tables are instantiated only for those cells in the obstacle grid which can actually lead to collision predictions that are inferior to $T_{\max}$. Other cells, as well as those contained within the robot outline, contain null pointers and don't use up RAM.

The compression algorithm is a variant of the Lloyd-Max quantizer used for image processing[12]. It operates on the histogram of the collision times stored in a buffer initialized with the results of the calculations described in section 3.4.3. Our variant uses a special bin for all negative values, and -1 is used to signify "no collision within $T_{\max}$". The process is illustrated in figure 3.16 and described in appendix A.3. Lloyd-Max calculates the boundaries and levels of a given number of bins by minimizing the sum of squared errors made when replacing all values by their bin's level. Our implementation starts with equally spaced bins[13] across the range of time values inside the buffer (clamped at $T_{\max}$).

Once the optimal bins have been determined, the buffer is quantized and stored in a dedicated data structure along with the quantizer values. Attention is paid to the fact that it is worse to underestimate a collision time than to overestimate it: The quantizer does

---

[12] Other possibly simpler compression approach would have been possible on Robox due to symmetries in shape: Collapsing collision times along trajectory curvatures, which would have made the implementation of the DWA closely resemble the Curvature Velocity Method.

[13] This means that some bins will be empty and could be used to improve the resolution, especially for cells with very uneven histograms. The current implementation can only remove empty bins, not split existing ones. Should the resolution or error of the compressed values ever become a problem, this is where it can be addressed.

not use the levels provided by Lloyd-Max but the smallest value inside each bin.

Instead of $(N^2 \times 10)$ bytes when using 80-bit floating point values, compression requires $(N^2 \times 1 + 256 \times 10)$ bytes in the case of 8-bit bins in the worst case, which leads to a break-even point at $N = 17$. Experience shows that the quantizer table rarely has more than 100 entries (breaking even at $N = 11$). During EXPO.02, N was 41 and compression decreased the lookup size from 16kB to 2kB for each non-null cell of the local obstacle grid.

BubbleFactory

The elastic band is a dynamic path representation with a varying number of bubbles, this makes it hard to predict runtime memory requirements. Allocating (and destroying) bubbles would have cost too much runtime overhead, but the straightforward alternative of pre-allocating a fixed number of bubbles would have lead to a waste of resource with little flexibility if the fixed number is chosen too small.

The adopted solution is a separate allocation thread with a scheduling priority that depends on the level of a pool of `Bubble` objects. It was inspired by *Kanban* just-in-time production management: Three levels of criticality are defined (`RED`, `YELLOW`, and `GREEN`) with associated pool thresholds and scheduling priorities. Instead of using the operating system's allocation routines, objects and methods requiring bubbles use a `BubbleFactory` for creating and destroying bubbles. More details can be found in section 3.7.4. During EXPO.02, thresholds were `RED` = 15, `YELLOW` = 30, and `GREEN` = 100.

3.7.4 Program Structure

An overview of the classes involved in motion generation is given in figure 3.17. Some of the components are readily recognized from earlier descriptions, whereas others will be described in the following.

`MotionPlanner`

The top-level class which provides an interface for other subsystems to communicate with the motion generation subsystem (e.g. configuration, setting the goal, querying the motion state) is called `MotionPlanner`. It orchestrates the components to achieve the required overall behavior by querying the elastic band about the best heading and feeding this information to the DWA. If no band exists, it falls back to a purely local algorithm using the direct line from robot to goal. It is also the motion planner's task to switch between pure rotation and forward movement. Replanning on the other hand is initiated independently by the elastic band.

The real-time obstacle avoidance task is anchored in `MotionPlanner`, descending into `DynamicWindow` and its associated objects for performing the actual obstacle avoidance. The reason for placing the RT task here is the finite state machine which requires global information (figure 3.18), whereas the `DynamicWindow` was kept purely local to keep it as independent as possible from the rest of the system.

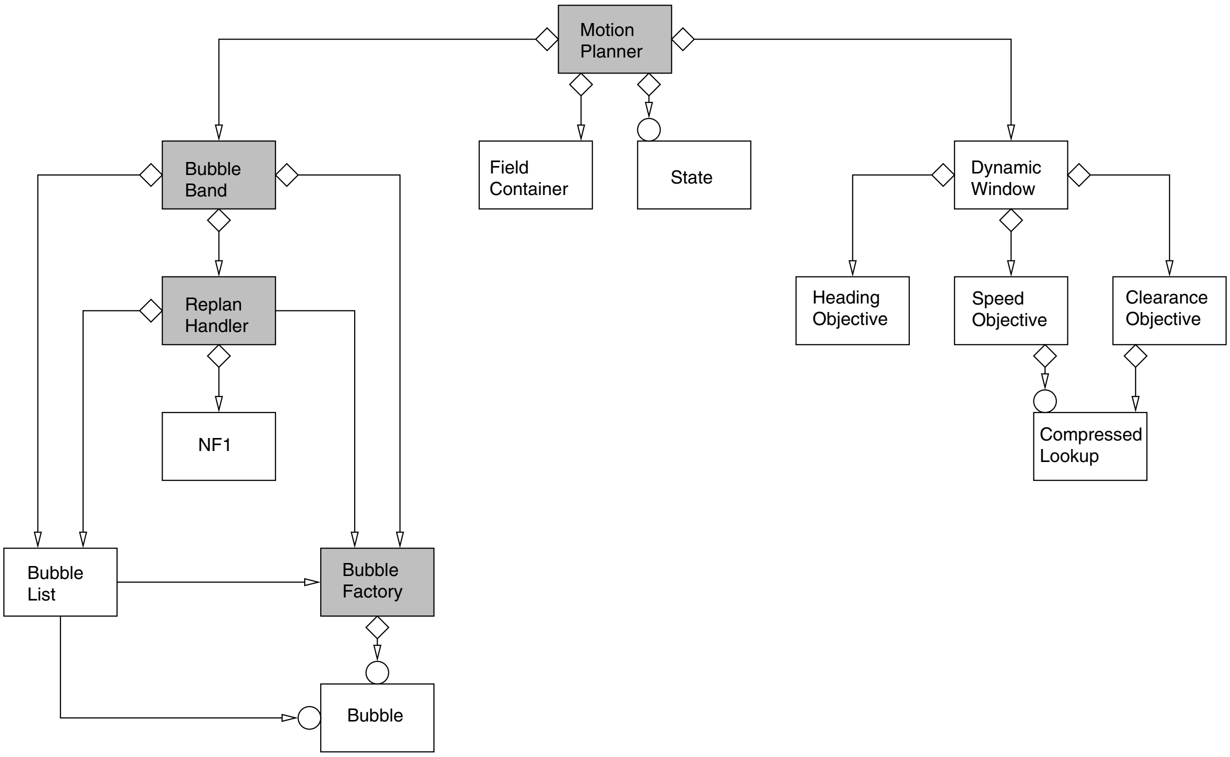


Figure 3.17: Motion generation class diagram: `MotionPlanner` serves as façcade to other subsystems on Robox and orchestrates the objects which implement motion generation. The components `NF1`, `BubbleBand`, and `DynamicWindow` encapsulate path planning, path representation, and obstacle avoidance. Grey boxes denote classes that have an associated thread. `MotionPlanner` manages the hard real-time obstacle avoidance task.

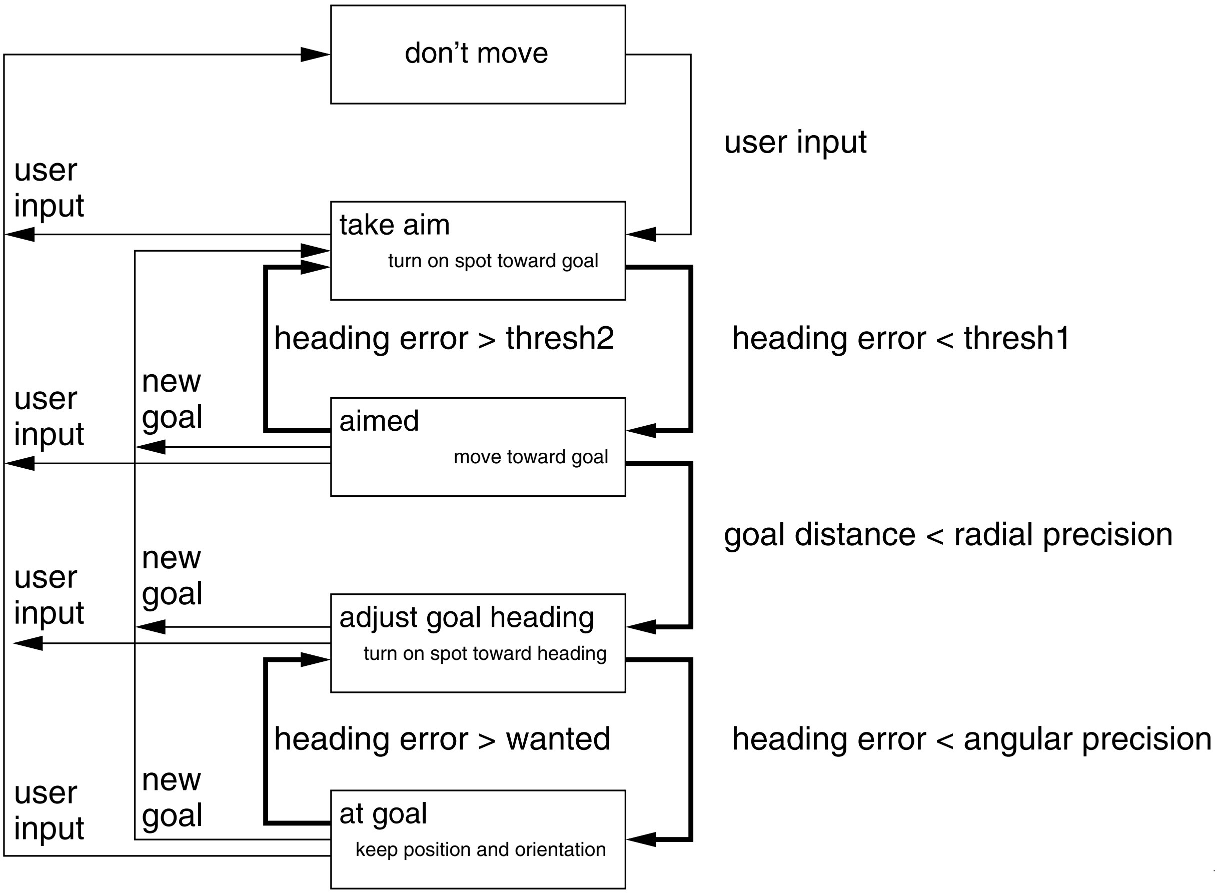


Figure 3.18: The state machine of `MotionPlanner`. Arrows indicate state changes and under which circumstances they are invoked. The task of each state is briefly described in each box.

`FieldContainer` and `State` are closely associated with `MotionPlanner`. They allow implementing a *state pattern* [19] such that the finite state machine is easier to adapt and maintain.

`DynamicWindow`

This is where the DWA is implemented. `MotionPlanner` takes care of defining the local goal and `DynamicWindow` behavior (e.g. turning on the spot or forward movement). Several computations are delegated to instances of the abstract `Objective` base class (not shown in figure 3.17).

`HeadingObjective` allows setting an offset angle and calculates w_{head}. `SpeedObjective` behavior can be switched using the methods `GoFast()`, `GoSlow()`, `GoForward()`, and `GoBackward()` in order to calculate w_{speed}. `ClearanceObjective`[14] is the most complex of all `Objective` subclasses: It contains a local obstacle grid composed of `CompressedLookup` instances constructed at startup, used for processing the current laser readings to determine which motion commands are admissible (those that do not lead to collisions) and assign w_{clear} in function of the sensed obstacles and the stored or detected ghost points.

`CompressedLookup`

The data structure for compressed lookup tables described in section 3.7.3 is implemented in `CompressedLookup`[15]. A static buffer is filled with floating point values using the `LoadBuffer()` method, then transformed into quantizer bins and values using the Lloyd-Max algorithm upon calling `SaveBuffer`. The bin and quantized values are stored in fields (i.e. non-static data) of `CompressedLookup` instances, and the `Get()` method performs the translation from quantized to floating point value.

`BubbleBand`

This class is the high-level interface for path representation and planning[16]. It is the entry point for the thread which updates the elastic band and contains a `ReplanHandler`. This object is at the core of keeping replan events from interfering with the smoothness of robot motion, by using a dedicated thread for replanning and preparing a new elastic band in the background. The robot continues to use the old plan and switches to the new band once it is ready.

`BubbleBand` plays a role which is similar to `MotionPlanner`, i.e. it maintains a state and orchestrates sub-objects, most calculations are delegated. The actual path representation is stored in a `BubbleList` instance containing `Bubble` objects. The separate list encapsulation

---

[14] Due to historical reasons, this class is actually called `DistanceObjective` in most existing versions of the software.

[15] This class is simply called `Lookup` in most versions.

[16] In the design used during Robotics@EXPO.02, `NF1` is "buried" underneath `BubbleBand`. This simplifies `MotionPlanner` but limits the flexibility for changing the planning algorithm. Separating planning from representation at a higher level would be a useful refactoring [16].

makes it easier to switch from a snapped to a freshly planned elastic band by swapping two pointers. It also performs list operations such as inserting or removing bubbles. `Bubble` objects use laser and ghost points to determine their parameters (such as radius and artificial forces) and provide methods for calculating the overlap between bubbles.

`BubbleFactory`

The dedicated memory manager for `Bubble` objects is called `BubbleFactory`. It's *raison d'être* is explained in section 3.7.3.

There are two "allocation" methods, `New()` retrieves a bubble from the pool and initializes it with values passed as arguments, whereas `Clone()` uses an existing bubble instance to initialize one from the pool. Both return a null pointer in case of an empty pool, which can happen when building a new elastic band under heavy system load as the production thread can then be starved by real-time threads[17]. It is thus important that clients of `BubbleFactory` do some low-level checking themselves to guarantee consistent behavior[18].

Another important aspect is deleting `Bubble` instance when they are not used any longer. In order to not waste the time invested during allocation, the instance is given back to the factory for recycling (filling the pool for later retrieval). This could lead to pool sizes largely exceeding the `GREEN` level, and a low-priority bubble deallocation thread could be used as remedy[19]. However, deallocation was completely removed from `BubbleFactory` to solve a problem with elastic band creation in very cluttered situations when the robot had been active for only a short period of time.

`NF1`

This class implements the NF1 algorithm and the grid representation. It also provides methods `ResetTrace()` and `GlobalTrace()` for determining successive cell center points from the robot position to the goal. These methods are used by `ReplanHandler` to generate a new elastic band by placing a bubble at each cell center[20].

3.7.5 Replanning Behavior

When the elastic band cannot guarantee sufficient clearance along the path, it snaps and a new plan has to be generated. We want Robox to do so without halting its movement. Here, environment properties can be taken advantage of: The band usually snaps because

---

[17] This phenomenon can be eliminated almost completely by filling the pool up to the `GREEN` level during initialization of `BubbleFactory` and disabling pool shrinking.

[18] During EXPO.02, allocation problems were handled by replacing the elastic band by a straight line to the goal and continuously replanning until a complete new band could be constructed.

[19] In order to not starve other threads requiring memory.

[20] subsequent elastic band updates quickly remove bubbles from this excessive translation, but it means that during construction of a new `BubbleList` the demand on `BubbleFactory` displays a sharp peak. However, performing more efficient translation from NF1 to elastic band would require more computations and thus increase the delay between planning and execution.

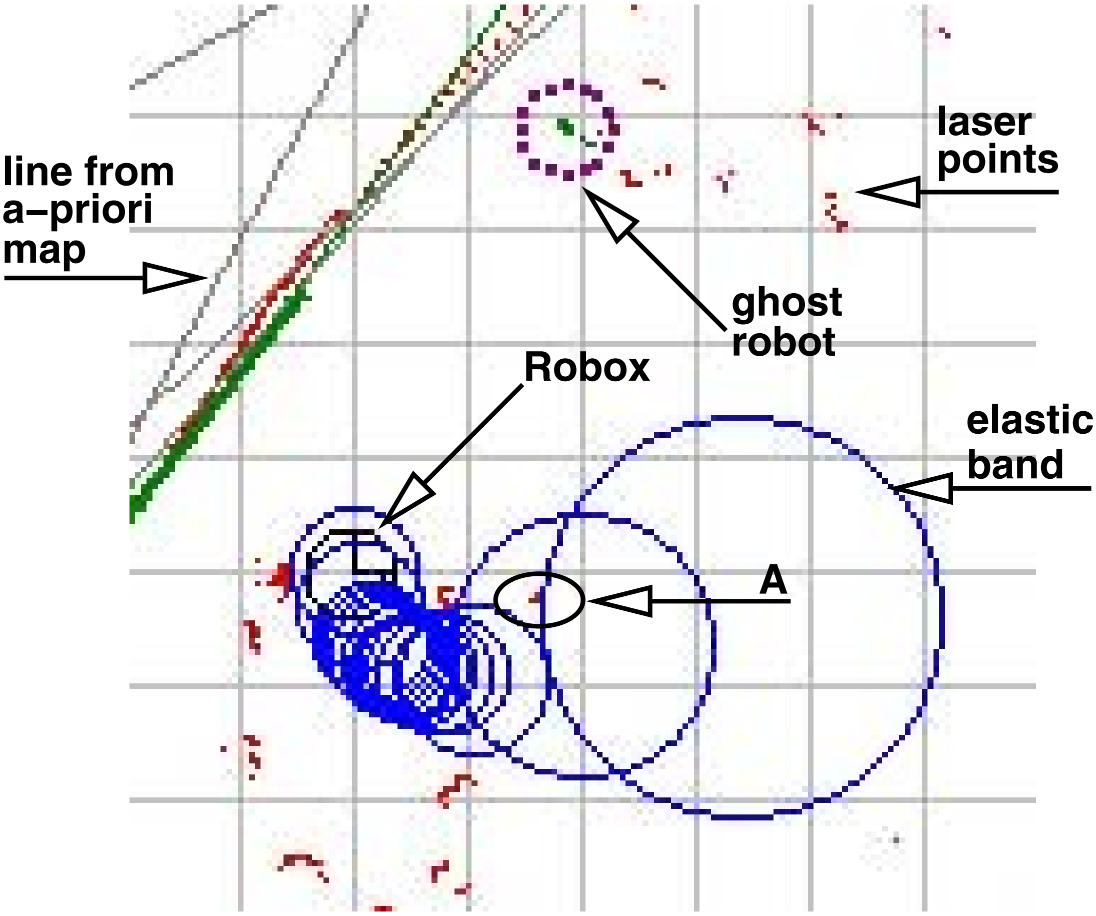


Figure 3.19: An image generated on the robot shows laser scanner data, a "ghost robot" constructed around a reflector of another Robox, the elastic band and lines from the map. The points in region A are masked, otherwise the second to last bubble would be smaller.

visitors move into it from both sides, and a couple of updates later the band can become valid again if they move on. So, Robox fires a background replanning thread but continues to use a snapped band, confident that the DWA keeps it from colliding. As soon as the new plan is available, it is used instead of the broken band[21].

3.8 Results

Robox moves smoothly through densely crowded exhibitions. Not only are the technical requirements of table 3.1 met, but its motion is very convincing. Numerous visitors have been seen stepping out of the way of approaching robots with an attitude of respectful interest, not because they were scared of being overrun. Virtually all children fearlessly

---

[21] This process is illustrated in figure 3.20.

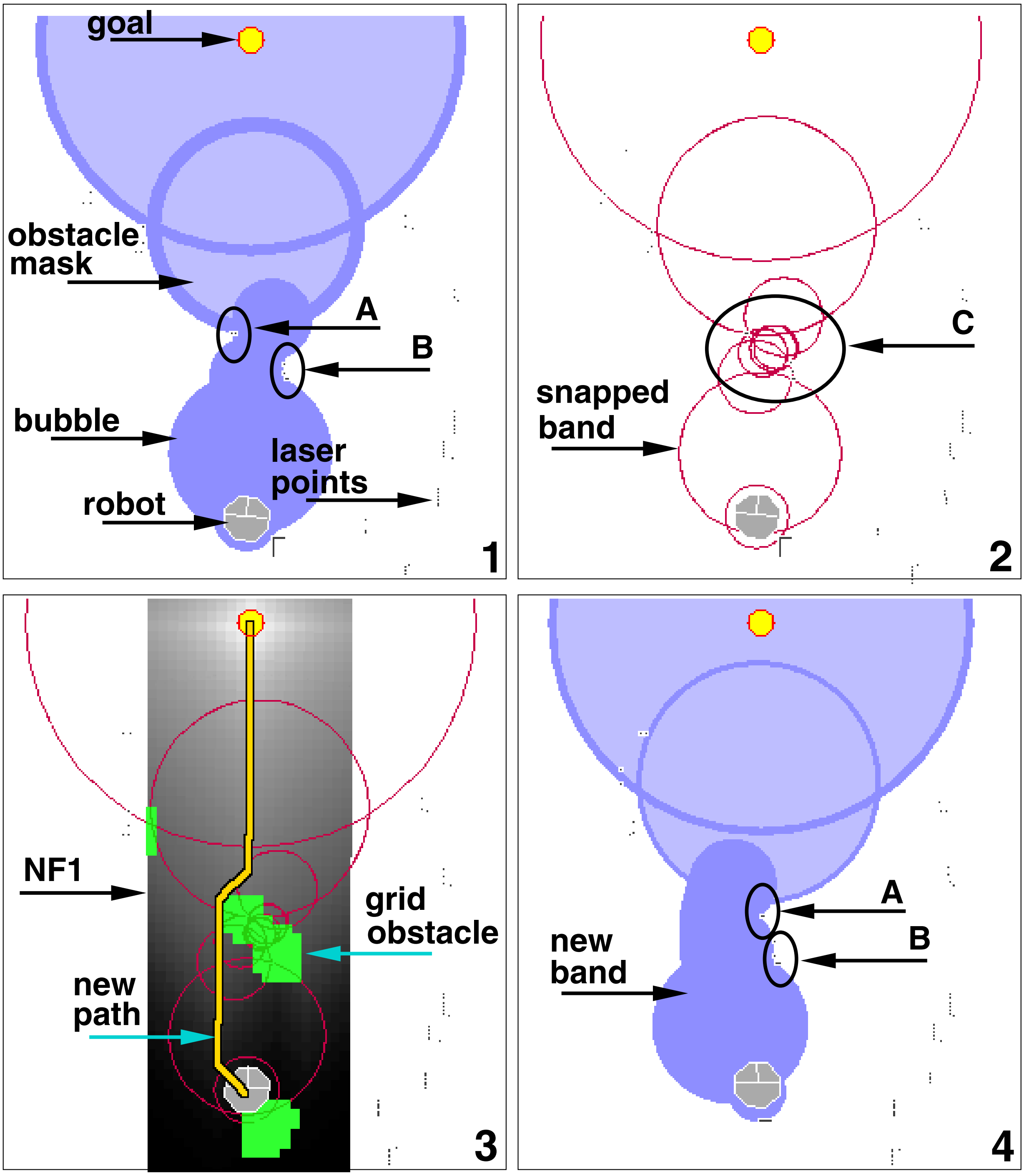


Figure 3.20: Typical replan sequence (simulated, took up to 0.5s during EXPO.02). 1: An intact band is squeezed by two visitors A and B approaching each other. 2: The band snaps in C, a replanning thread is started. The robot continues to update and use the snapped band to keep its movements smooth. 3: The unsmooth NF1 path is an intermediate step of the replanning thread. 4: The new path has been translated into a new elastic band.

approached moving robots in order to see how the machine would react – if they could trap it alone or with others (often resulting in children dancing around robots trying to rotate their way out of the trap). The overall visitor flow was maintained at a high level (approximately 500 per hour) which shows that even in the presence of curious kids, the robot succeeded in giving tours through the crowd.

Depending on the number of obstacles and the current speed, calculating one iteration of the DWA on Robox can take up to $22ms$, with a mean value of $7.5ms$.

During Expo.02, the eleven Robox accumulated an operation time of $13313h$, of which $9415h$ were spent in movement for a total distance of $3315km$ and met approximately 686000 visitors. The maximum speed of the robot was set to a relatively low $60cm/s$ such that visitors would not feel threatened. No harm was done to any visitor, exhibit or robot.

3.9 Future Work

No collisions with immobile objects that are detectable by the laser scanner have been reported, yet some collisions happen. They are mostly due to objects that are out of the laser's field of view (especially feet and protruding parts of buggies and wheelchairs), in which case the robot simply stops as soon as contact is detected. Two main failure modes remain. One can appear when moving Robox from the main exhibition hall into a very narrow corridor, the other stems from the simplistic detection and modeling of other robots:

1. Moving into a corridor, Robox sometimes gets stuck at a corner when turning into the corridor leading to its charging station. The entry to this corridor barely leaves 20cm of leeway, it should thus be traversed as perpendicularly as possible. However, the approximation of holonomic movement in the path representation, can lead Robox towards the opening such that one of its corners disappears in the blind zone before the robot has engaged the door. Then, further along the path, the corner reappears inside the modeled robot outline and triggers a stop. This problem could be alleviated using a variety of techniques such as building a local map, using a nonholonomic planner and path representation, or adding sensors that cover the blind zone.

2. Robots can also get stuck when two of them move towards each other while not detecting the reflectors, which can be hidden behind visitors. When the reflectors are later detected, the resulting ghost robots appear instantly, often inside the outline and make Robox stop. A seemingly obvious solution to this problem would be to wirelessly broadcast robot positions, but this would introduce a plethora of multi-robot issues such as synchronization of time and reference frames, not to mention the loss of autonomy.

A third issue is oscillating replanning that has appeared in simulation but was not observed during Expo.02. Integrating perceptive memory along the lines of the local perceptual space mentioned in [29] could alleviate this if it ever becomes a problem.

These failure modes were the only observed cases where the robots got stuck due to deficiencies of the motion planner. Other blocked situations arose from localization failures which made the robot seek goals inside walls, and rare sensor failures at the beginning of Expo.02 which were subsequently detected automatically to make the robot perform an emergency stop and notify the supervision personnel of this hardware problem.

3.10 Conclusion

This chapter presented a novel combination of known algorithms for mobile robot path planning and control. It was shown that our combination performs well enough to be deployed in a challenging long-term real-world application.

Using a time-based clearance measure solves a singularity present in the original DWA and bases the dynamic window on a physically meaningful representations. Using the speed objective as a means to switch between overall robot behaviors ensures that they all appropriately avoid obstacles.

Relying on the DWA, the elastic band could be simplified to a point where it becomes computationally efficient. The main speedup comes from using Euclidean distances instead of a kinematically correct measure. Heuristically masking some obstacles reduces the frequency of replan requests to improve the overall performance of the motion planner in the context of highly cluttered and dynamic environments.

3.11 Summary of Parameters

Numerous parameters influence the operation of the motion generation system presented in this chapter. Table 3.2 summarizes their names, how they influence the behavior, and their numerical values during Expo.02.

Parameter (EXPO.02 value)	Description	Remarks
Kinematics		
$R_{\text{wheel}} = 0.09m$ $D_{\text{base}} = 0.521m$	Wheel radius Wheel base	better calibration [5] ⇒ better pose prediction
DWA		
$\Delta t = 0.1s$	Task cycle time	Real-time task frequency
$\dot{q}_{\text{max}} = 6.5\frac{rad}{s}$ $\ddot{q}_{\text{max}} = 6.5\frac{rad}{s^2}$ $\dot{s}_{\text{max}} = 0.6m/s$ $\dot{\theta}_{\text{max}} = 2.5\frac{rad}{s}$ $T_{\text{max}} = \frac{\dot{q}_{\text{max}}}{\ddot{q}_{\text{max}}} = 1s$	Max wheel speed Max wheel acceleration Max translational speed Max rotational speed Max braking time	Hardware and application dependent (i.e. limit to walking speed during EXPO.02)
$\alpha_{\text{clear}} = 0.5$ $\alpha_{\text{speed}} = 0.1$ $\alpha_{\text{head}} = 0.1$	Weight of clearance objective Weight of speed objective Weight of heading objective	$\alpha_{\text{clear}} \gg \alpha_{\text{speed}}, \alpha_{\text{head}}$ ⇒ react early to obstacles (but can deviate from elastic band); higher α_{speed} ⇒ stricter on-spot turns
$L_x = 2.2m$ $L_y = 1.5m$ $\Delta_{\text{grid}} = 0.03m$	Obstacle grid depth (x-axis) Obstacle grid width (y-axis) Grid resolution (cell size)	(L_x, L_y) such that braking displacement $(\propto T_{\text{max}}^2 + R_{\text{robot}})$ contained in grid
$N = 41$ $\Delta\dot{q} = \frac{2\dot{q}_{\text{max}}}{N-1} = 0.325\frac{rad}{s}$	Velocity space grid dimension Velocity space grid resolution	N odd number (ensured by software); $\Delta\dot{q} < \Delta t \cdot \ddot{q}_{\text{max}}$ (otherwise no mvt.)
Elastic band		
$L_{\text{min}} = 2.0m$ $L_{\text{max}} = 8.0m$ $m_{\text{max}} = 8.5m$	Start of obstacle masking End of obstacle masking Max obstacle masking	many small moving obstacles ⇒ low $\{L_{\text{min}}, L_{\text{max}}\}$ and high m_{max} to decrease replanning frequency
$r_{\text{lim}} = 1.6m$ $\alpha_{\text{int}} = 0.1$ $\alpha_{\text{ext}} = 0.1$	$\boldsymbol{f}_{\text{ext}}$ threshold Weight of internal force Weight of external force	$r_{\text{lim}} > R_{\text{robot}}$; high clutter ⇒ low r_{lim}; higher update frequency ⇒ lower $\{\alpha_{\text{int}}, \alpha_{\text{ext}}\}$; dynamic environment ⇒ higher α_{ext} (also higher r_{lim} if little clutter)
$W_{\text{robot}} = 2R_{\text{robot}} = 1.8m$ $W_{\text{rem}} = 3.24m$ $W_{\text{add}} = 2.16m$	Robot width Bubble removal threshold Bubble insertion threshold	$W_{\text{rem}} > W_{\text{add}}$ (higher ΔW ⇒ more stable band); higher W ⇒ more stable band
Bubble factory		
`RED` = 15 `YELLOW` = 30 `GREEN` = 100	High priority threshold Normal priority threshold Idle priority threshold	smaller values ⇒ less RAM but depletion risk; similar (small) thresholds ⇒ high processor load
NF1		
$M_{\text{l}} = 5m$ $M_{\text{d}} = 41$ $M_{\text{l}}/(M_{\text{d}} - 1) = 0.125m$	Width of grid N° cells along width Cell size	time and space $O(M_{\text{d}}^2)$; smaller $\{M_{\text{l}}, M_{\text{d}}\}$ ⇒ might not find existing path
Other		
$\epsilon = 10^{-9}$	Considered ≈ 0	

Table 3.2: Motion generation parameters

Chapter 4

E*: Generic Dynamic Interpolated Navigation Functions for Planning

This chapter presents the E* algorithm, a generic path planning method that combines dynamic replanning capabilities with path cost interpolation. Even though the algorithm is grid-based, it produces navigation functions that are practically free from discretization effects that commonly flaw grid-based calculations. The need for a high-performance algorithm to solve this problem arose during work on [22], an approach to incorporating knowledge of object movement during path planning. E* has been inspired by the dynamic replanning capabilities of D* [52], adding generic interpolation which leads to high fidelity navigation functions.

Dynamic replanning means that it is possible to incrementally adapt an existing navigation function to changes in the environment model, instead of recalculating the function on the whole domain. Path cost interpolation means that the distance from a given node to the goal is not a sum of edge distances, but is measured in the continuous domain "between" edges. The algorithm is generic in the sense that it uses an approach to dynamic replanning that is designed to work with a generic formulation of interpolation kernels. Any method that fulfills the requirements of this formulation, which is encapsulated as a *class* in the object-oriented implementation, can be plugged into E*.

The advantage of an interpolated navigation function is its smoothness, which considerably improves the path quality during gradient descent. However, interpolation requires more computational resources, a drawback which is alleviated by dynamic replanning capabilities. As a generic algorithm independent of the interpolation method, E* makes it straightforward to trade off computational effort versus interpolation quality.

Chapter Overview

Section 4.1 of the present chapter explains why the combination of NF1 and elastic band was deemed insufficient and how the idea for a novel approach developed. Then comes a more formal introduction to path planners in section 4.2, which also explains how so-called navigation functions can be viewed as distance maps in order to better understand this chapter. Related work on dynamic planning and wavefront propagation is presented in section 4.3, followed by a summary of the Level Set Approach in section 4.4. The Level Set Method is an important prior work for this chapter, but it has been rarely used in the field of robotics and must thus be introduced in sufficient detail.

After these introductory sections, the E* framework is presented in section 4.5. It is a generic algorithm that, among other features, allows to interchange interpolation methods. Three such methods are then presented in section 4.6, with the so-called LSM kernel being the most important of the three. The performance of E* is then evaluated in section 4.7, which concentrates on the core algorithm and how it compares to D*.

At this stage, E* will have been presented in a detailed manner, and section 4.8 proceeds to address the question of how to include it into a robotic system as a global planner. It gives a high-level view and presents a finite-state machine that takes care of the various cycles required for E* operation in conjunction with a changing environment model. Finally, section 4.9 presents the conclusions of this chapter and gives an outlook to future work, particularly in view of extending E* to a more general graph-based formulation.

4.1 The Need for Smooth Dynamic Planning

The path planning approach used during the Robotics@Expo.02 event had some drawbacks that could not be addressed in an ad-hoc way. Apart from the lack of smoothness inherent in the NF1 approach[1], there is no distinction between static and dynamic objects (e.g. visitors). On a related note, the NF1 was initialized using only the most recent scan data, which can be problematic because the environment topology is rarely fully reflected in such a snapshot.

There exists a fair amount of published research that incorporates environment dynamics during path planning, for example[2] [15, 18, 4, 30, 1, 20]. However, these approaches are not appropriate for an application in highly cluttered dynamic environments such as mass exhibitions: They either rely on relatively elaborate modeling techniques that require information and computational resources that are not readily available for such unpredictable settings (i.e. extending $\mathcal{C}$ to a full-fledged state-time representation), or are limited to constant velocity models for the objects in the robot's vicinity. Alami's work [1] is an exception, as it treats environment dynamics using worst-case scenarios that take into account the sensor capacities, but during planning it treats all known obstacle information as static.

Suppose that the robot has a sensor-based method of distinguishing static from dynamic objects. It is not necessarily practical nor required that the motion information contain a predictive model, even though it would be an advantage if such information exist. For example, one of the results of [23] is a polar array that contains distance and motion information in a format similar to what a laser scanner provides[3]. The idea is to formulate

---

[1] This had been worked around using the elastic band, see section 3.3.

[2] See also chapter 2, in particular section 2.2.3.

[3] It also tracks a visitor's movement and contains information that can be used to predict motions.

the path planning problem in a way that treats static objects as topological information, but moving objects in a more flexible way by assigning collision risks based on the amount and direction of movement in a given zone. This would yield plans that are not necessarily collision free, relying on lower levels to guarantee obstacle avoidance, but that trade off the risk inherent in traversing a dynamic region against the additional expected path length for circumventing that area. Obstacles that are known to be static should of course still be treated as non-traversable in order to maintain the topological correctness of the path. In other words, planning in highly cluttered dynamic environments is viewed as a weighted region path planning problem [37, 42], but in this context the regions can not be pre-determined because they depend on movement in the environment. A more flexible environment representation such as a grid-based risk map is more appropriate.

Another issue with the approach presented in chapter 3 (using the elastic band to smooth an initial plan) is that the elastic does not take into account movement information in its present formulation. Applying it to a path planned with motion information would counter the effort that went into adding this information into the planner. Making the elastic band react to motion could be interesting research, but if the planner provides smooth plans then this requirement disappears.

The aim of this chapter can be summarized as follows: Develop a path planner that is capable of taking into account a continuous risk measure, defined on regions that are neither static nor known beforehand. The planner must produce topologically correct and smooth paths that trade off collision risk against expected path length. Additionally, changes to the environment model are expected to be frequent due to environment dynamics, and the planner should be able to efficiently adapt existing plans to such changes. In particular, it should be avoided to replan if the change is of no concern to the robot's current action (i.e. it is farther away from the goal than the robot).

4.2 Navigation Functions as Distance Maps

Mobile robot path planning approaches can be divided into five classes [31]. Roadmap methods extract a network representation of the environment and then apply graph search to find a path. Exact cell decomposition methods construct non-overlapping regions that cover free space and encode cell connectivity in a graph. Approximate cell decomposition is similar, but cells are of predefined shape (e.g. rectangles) and do not exactly cover free space. Potential field methods differ from the other four in that they don't lead to a graph representation, but treat the robot as a point evolving under the influence of forces that attract it to the goal while pushing it from obstacles.

In the cited work, navigation functions are treated as a special case of potential fields. They have the advantage of being free of local minima, but introduce drawbacks due to the fact that they are calculated on grids. In this work, a different stance is taken: Navigation functions share certain properties of graph-based planners, and can also be considered samplings of a distance function which takes into account environment topology. This dual interpretation is the key to formulating the E^* algorithm.

- A grid can be considered an approximate cell decomposition. All cells are of identical shape, they only approximate free space.
- Calculating the navigation function is equivalent to graph search. It starts at the goal location(s) and (monotonically) propagates through the grid until a path is found.
- The distinguishing property of navigation functions is their unique minimum at the goal. By gradient descent, the robot monotonically reduces the "height" of its location on the grid just as it monotonically decreases the distance to the goal. This equivalence leads to interpreting the navigation function as the sampling of an underlying distance function.

Graph-based calculations have inherent drawbacks due to their discontinuous representation of workspace $\mathcal{W}$ or configuration space $\mathcal{C}$, e.g. NF1 [31] produces paths that graze obstacles and that are constituted of straight line segments joined by angles that are integer multiples of $\pi/4$. Potential field methods produce smoother paths than NF1 and can often be expressed directly in terms of sensor readings, however they are flawed by the existence of local minima.

4.3 Dynamic Planning and Wavefront Propagation

Graph-based planners rely on mechanisms that propagate path cost information among neighboring locations and require replanning if the underlying environment model changes. For example, A* uses its Open list to propagate path costs in an upwind manner, but in case the environment changes the whole algorithm has to be run again. This drawback is addressed by the D* algorithm [52, 53], which minimizes the computational cost of replanning by recalculating the navigation function only where necessary.

During path cost propagation in D*, the set of nodes that enter and leave the Open list can be thought of as a discrete wavefront sweeping through the graph, being confined to the edges and nodes. Now, in order to extend this concept, imagine a continuous contour to sweep outward from the goal throughout the environment (see figure 4.1), and record when this hypothetical wavefront crosses each cell. It is important to notice that the crossing time divided by the propagation speed yields a distance. The similarity with the progress of D* is clear, however the main insight comes from turning the problem around: Consider the crossing times at the nodes as samples of an underlying continuous navigation function, instead of extending a discretely defined distance function into the continuous domain. This idea is not limited to the two-dimensional case.

In order to use this approach for path planning, the propagation speed (normal to the curve) is chosen to be dependent on position only (e.g. low speed in risky regions). By adding a time axis, the surface traced by the evolving curve becomes a navigation function. The gradient method [29], as well as [7] and [56] take similar stances[4].

---

[4] Like D*, the gradient method encodes the risk of a location in its *intrinsic cost* rather than a speed.

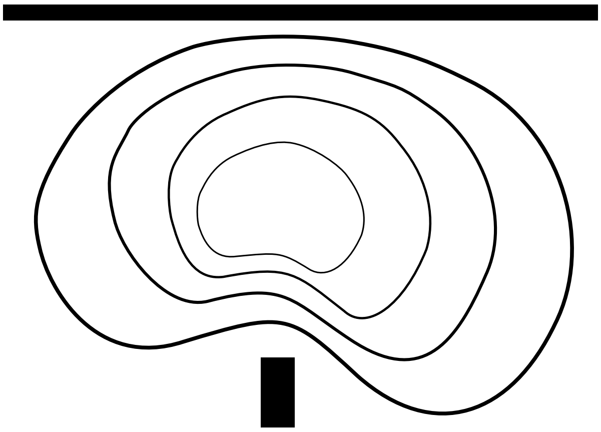


Figure 4.1: Continuous domain wavefront formulation. A contour sweeps outward from the goal throughout the environment, taking into account obstacle information. It is important to realize that, by making the propagation speed dependent on position to reflect environmental properties, the resulting crossing-time map can be used to reach the goal by gradient descent.

Section 4.6 presents possible approximations of this continuous formulation. This kind of problem has been treated in fields such as fluid mechanics or computer vision. The large body of work on Level Set Methods [46] provides a theoretical foundation for robustly interpolating the crossing time. Fast Marching Methods [45] are a special case formulation that take advantage of *monotonically* advancing fronts such as the one considered here, leading to a considerable reduction in computation.

An important aspect of Level Set Methods is the *upwind* property. It ensures that the wavefront propagation does not violate *Huygens' principle* – shocks and rarefactions need to be treated correctly (see [47] for details). This is particularly apparent in Fast Marching Methods and their one-pass calculation that sweeps out from the initial wavefront. In D* the upwind property holds as well, and it is traced through the use of *backpointers*. This is instrumental for dynamic replanning: Backpointers trace on which neighbor a cell's path cost depends, such that all descendants of a location can be visited if that location's environmental information changes. One of the contributions of E* is the extension of the backpointer concept to an ordered set, such that the upwind property holds in the presence of interpolation during dynamic replanning.

4.4 Summary of the Level Set Formulation

The Level Set Method (LSM) developed by J. A. Sethian was an important inspiration for developing the E* algorithm. It is a powerful approach to calculating evolving interfaces in various fields, such as fluid mechanics and computer vision. As it is not commonly known in the field of mobile robotics, this section gives a quick introduction to the LSM concepts necessary for understanding E*. For a more detailed presentation, the book by

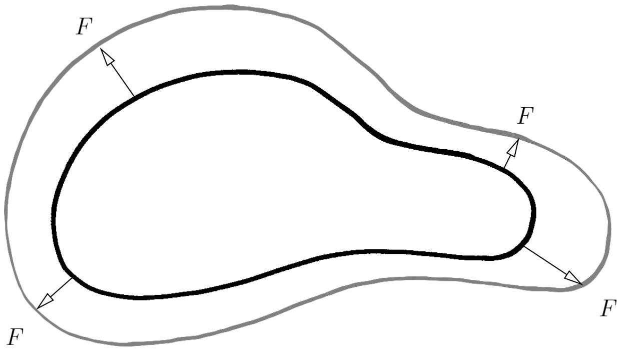


Figure 4.2: The wavefront propagates with speed F along its normal vector $\boldsymbol{n}$. F can depend on several kinds of factors (see text for details). This illustration is inspired by figure 1.1 of [47].

J. A. Sethian is an excellent source [47]. The material in this section is mainly based on chapters 1, 2, and 9 of that book.

Even though E* can be used without it, tests have shown that the LSM provides the most robust and precise distance information of all the tested methods (see section 4.7). It is also useful to already introduce the concepts of *upwind property* and the *Fast Marching Method* because they make the remainder of this chapter easier to understand.

4.4.1 The Lagrangian and Eulerian Formulations

Figure 4.1 illustrates the idea of calculating a distance map by sweeping a continuous-domain wavefront from the goal through the environment, such that it avoids obstacles and slows down in regions of higher collision risk. This idea is formulated more precisely in figure 4.2, which introduces the notion of *propagation speed* $F = F(L, G, I)$ that can depend on local properties L (e.g. curvature), global properties G (e.g. integrals along the front), and independent properties I (e.g. underlying flow). The wavefront propagates along its normal vector $\boldsymbol{n}$ with a magnitude given by F.

One approach to formulating the wavefront propagation would be the so-called *Lagrangian* perspective: Parameterize the curve such that its normal vector can be calculated, discretize time and space, and repeatedly move each discrete point on the curve along $\Delta t F \boldsymbol{n}$. Figure 4.3 and equation (4.1) illustrate this.

$$\begin{cases} \gamma : & \boldsymbol{x}(s,t),\ 0 \le s \le S,\ \boldsymbol{x}(0,t) = \boldsymbol{x}(S,t) \\ \boldsymbol{n} : & \boldsymbol{n}(s,t) \perp \boldsymbol{x}(s,t) \\ \Rightarrow & \boldsymbol{n} \cdot \frac{\partial \boldsymbol{x}}{\partial t} = F(L,G,I) \end{cases} \tag{4.1}$$

where γ denotes the front's parameterized form (note the periodicity of the parameter s), $\boldsymbol{n}(s,t)$ is the normal vector to the front at $\boldsymbol{x}(s,t)$, and the last line is the differential equation that has to be solved (numerically).

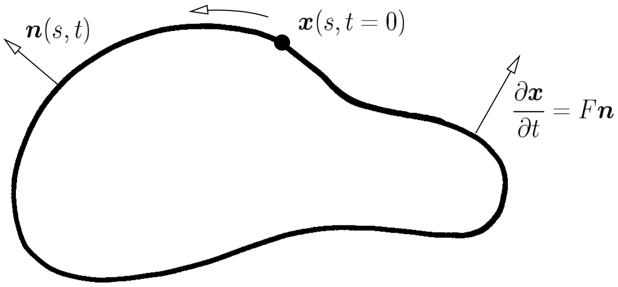


Figure 4.3: The Lagrangian formulation of wavefront propagation: Time and space are explicitly discretized (see text for details). This illustration is inspired by figure 1.2 of [47].

The Lagrangian formulation presents several serious drawbacks, which [47] explains in detail. For instance, involved heuristics are needed to treat shocks correctly, oscillations tend to get amplified, and topological changes in the wavefront (merging or disappearing sections) are very difficult to detect. The solution proposed by J. A. Sethian lies in using the *Eulerian* formulation, which adds a dimension to the problem and then treats the wavefront as the intersection between a graph and the zero-level of the additional dimension. This is illustrated in figure 4.4 and equation (4.2).

$$\begin{cases} \Gamma(t): & \text{closed } (N-1)\text{D surface} \\ \Phi(\boldsymbol{x},t): & \mathbb{R}^N \to \mathbb{R} \\ t_0: & \Phi(\boldsymbol{x},t=0) = \pm d(\boldsymbol{x},\Gamma(t=0)) \\ \Rightarrow & \Gamma(t) = \{\boldsymbol{x} \mid \Phi(\boldsymbol{x},t)=0\} \end{cases} \tag{4.2}$$

where Γ denotes the wavefront, N is the supporting dimension (i.e. $N=2$ in figure 4.4), Φ is the graph that is intersected with the zero level to yield Γ, the line labelled t_0 : indicates that Φ is initialized to the signed distance from the initial wavefront, and the last line formalizes the way in which Γ and Φ are related.

The advantage of adding the extra dimension stems from the fact that topology changes can now occur without special treatment, and that numerically more stable methods are available for solving the differential equation that describes the front's evolution. This is illustrated in figure 4.5 and equation (4.3).

$$\frac{\partial \Phi}{\partial t} + F|\nabla\Phi| = 0 \tag{4.3}$$

4.4.2 The Eikonal Case: Fast Marching

The Level Set Method can now be fully described: Convert the initial wavefront $\Gamma(t=0)$ into a graph $\Phi(\boldsymbol{x},t=0)$ by taking the signed distance from $\boldsymbol{x}$ to the initial front ($\boldsymbol{x}$ being sampled from a grid); repeatedly solve equation (4.3) using a fixed timestep; determine the front's evolution $\Gamma(t)$ by intersecting $\Phi(t)$ with the zero level. This requires a discrete

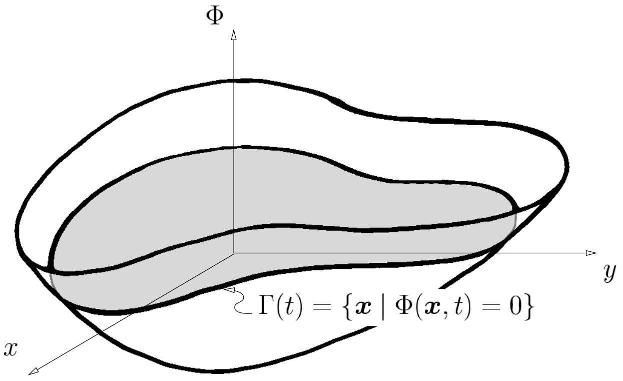

Φ
y
$\Gamma(t) = \{\boldsymbol{x} \mid \Phi(\boldsymbol{x},t) = 0\}$
x

Figure 4.4: In the Eulerian perspective, the front is interpreted as the intersection between a graph and the zero-level of an additional dimension Φ. This formulation is more indirect than the Lagrangian perspective and leads to a more robust algorithm. The evolution of the Φ-graph is calculated, and at each instant the wavefront $\Gamma(t)$ can be retrieved by intersecting with $\Phi = 0$. This illustration is inspired by figure 2.1 of [47].

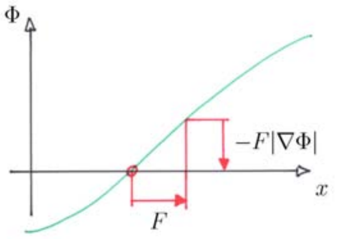


Figure 4.5: The one-dimensional case of the Eulerian formulation claryfies how equation (4.3) describes the wavefront's evolution: In order to make the intersection move towards the right with speed F, the whole curve Φ has to move downwards with speed $F|\nabla\Phi|$.

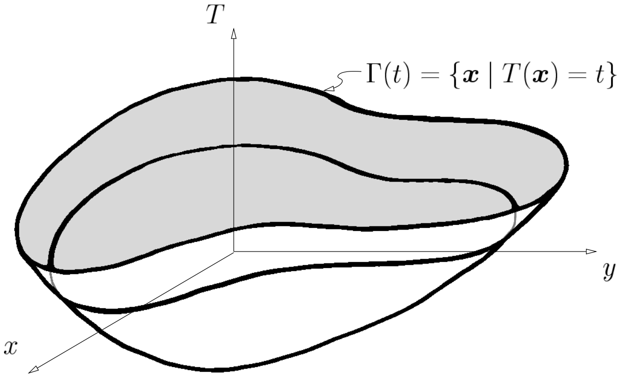


Figure 4.6: The Eikonal case leads to a simplified formulation of the Level Set Method that can be solved very efficiently. This case occurs if the propagation speed is always positive (or negative) and depends on position only. Note the notational change from Φ to T. This illustration is inspired by figure 2.3 of [47].

approximation of the gradient operator ∇, as well as some other heuristics to make it efficient and stable. Details are presented in [47].

An important special formulation that makes these calculations very efficient is the *Eikonal* case, which is applicable when the propagation speed is always positive (or negative) and depends on position only. This special case is thus applicable to the path planning problem described in this chapter: Robot speed depends on position only (the traversal risk does not depend e.g. on the front's curvature), and it is always positive. The *Fast Marching Method* can then be applied: It treats Φ as a crossing-time map. In other words, $\Gamma(t)$ is no longer defined by the zero level of Φ, but by intersecting Φ with the level of height t. In order to stress this change of interpretation, a notational change is introduced: Φ becomes T in order to indicate that it is interpreted as a crossing *time.* Figure 4.6 and equation (4.4) illustrate the Eikonal case.

$$\begin{cases} F = F(\boldsymbol{x}) > 0 \\ \Gamma(t) = \{\boldsymbol{x} \mid T(\boldsymbol{x}) = t\} \\ |\nabla T| F = 1 \end{cases} \tag{4.4}$$

where F denotes the propagation speed (which depends only on position and is always positive), the wavefront Γ is now defined as the intersection between the crossing-time map T and a given instant t, and the simplified differential equation that has to be solved is given in the last line.

The advantage of this formulation lies in the fact that the crossing time T can be built outward starting at $T = 0$. This is due to the *upwind property*: A given location is traversed only once by the wavefront, which can thus be likened to a grass fire spreading through a prairie (once a patch has been burned, it stays burnt). Only locations that are downwind from a given point can be influenced by the value of T at that point. In other words, in order to calculate a location's T, only its *upwind* neighbors must be considered. Given that the LSM is calculated on a grid, it is useful to now introduce a discrete notation (limited here to the two-dimensional case of two grid indices i and j). Equation (4.5) gives the result when using a first-order gradient approximation that satisfies the so-called *entropy condition* which is a formulation of the upwind property (for more details, consult [47]).

$$\max(D_{ij}^{-x} T_{ij}^{n+1}, 0)^2 + \ldots + \min(D_{ij}^{+y} T_{ij}^{n+1}, 0)^2 = F_{ij}^2 \tag{4.5}$$

where D_{ij}^{-x} denotes the finite difference along negative x at grid point (i, j), T_{ij}^{n+1} is the crossing time at (i, j) for the next step (step n starts at $n = 0$ and is incremented by one at each iteration), and F_{ij} is the propagation speed at (i, j). Note that equation (4.5) leads to a quadratic equation, an efficient way of solving it has been developed for this thesis and is presented in section 4.6.3.

The Fast Marching Method is best summarized by quoting the introductory paragraph from section 9.2 *The Eikonal equation and the fast marching method* of [47]. The equation (9.4) mentioned in the quotation corresponds to equation (4.5) in this thesis:

> *The key to constructing a fast marching algorithm is the observation that the upwind difference structure of equation (9.4) means that information propagates "one way", that is, from smaller values of T to larger values. Hence, we can "solve" equation (9.4) by building the solution outwards from the smallest T value. The algorithm is made fast by confining the "building zone" to a narrow band around the front. The idea is to sweep the front ahead in an upwind fashion by considering a set of points in a narrow band around the existing front, and to march this narrow band forwards, freezing the values of existing points and bringing new ones into the narrow band structure. The key is the selection of* which *grid point in the narrow band to update.* [J. A. Sethian, 1996]

Tracing Fast Marching: Backpointers

One of the key realizations that lead to the development of the E* algorithm is the similarity between the "marching" of the narrow band mentioned in the above quotation, and the order of evaluation used in the D* algorithm. In D*, this evaluation order is traced using *backpointers*, which makes it possible to subsequently repair the plan to changes in the environment model by re-initializing a wavefront starting at the changed cells and following the backpointers to repair path costs in accordance with the upwind property. There is also a mechanism for changing the stored upwind directions, which can be considered equivalent to changing the environment topology. The upwind property applies to D*, but the main difference lies in the fact that the Fast Marching Method employs a gradient approximation that takes into account more than one neighbor of a given cell.

4.5 E* Framework

The E* formulation uses abstractions of the concepts *interpolation* and *wavefront propagation* to make it generic. A *grid* of *cells* is used to represent the environment and the navigations function. E* relies on other processes to keep the environment model up to date, the wavefront propagation treats it as constant (see section 4.8). Changes to the environment information are signaled to E* through a high-level interface, which takes care of initializing dynamic replanning. Wavefront propagation relies on an independently configurable interpolation method to determine the exact form of the navigation function.

4.5.1 Interpolation Kernels

Locations are cells c_i that belong to a given domain $C = \{c_i\}$. Each cell has a set of neighbors N_i and represents its environmental properties as meta information F_i (4.6). The crossing-time at each location is stored in the cell value T_i. Wavefront propagation runs for a number of iterations which is not necessarily known beforehand. During an iteration, a cell can change its T_i value. This is an operation based on the propagator set[5] $P_i \subseteq N_i$ and the cell's meta information F_i (4.7). Each cell has a set of backpointers $B_i \subseteq P_i$.

$$c_i \in C,\ N_i \subset C,\ c_i \notin N_i,\ c_1 \in N_2 \Leftrightarrow c_2 \in N_1 \tag{4.6}$$

$$F_i \geq 0,\ T_i \geq 0,\ \{T_i, B_i\} = k(F_i, P \subseteq N_i) \tag{4.7}$$

The remainder of this chapter treats interpolating the wavefront direction as "coming from" between *two* cells. As a consequence, a special case of the above formulation is used: The "best" and "second best" neighbors $P = \{c_1, c_2\}$ of a cell c_0 are used to construct the distance to the wavefront using the interpolation kernel $k(F_i, c_1, c_2)$.

At first sight, the propagators are the same as the backpointers. However, the kernel might fail to provide valid interpolation for certain combinations, and the use of *fallback* or degenerate solutions is required in these cases, and this information has to be passed to E* such that the backpointers reflect the actual dependency between cells. This is why the kernel provides not only the updated value, but also backpointer information needed to trace propagation direction during dynamic replanning.

Section 4.6 presents three examples of interpolation kernels which will be used in the evaluation of E* in section 4.7. One of these kernels is not interpolating but designed to mimic the behavior of NF1, which makes E* equivalent to D*. This makes it possible to determine the effects of adding interpolation.

4.5.2 Algorithm Structure

Figure 4.7 shows the structure of E*. The terms used in this section reflect the object-oriented programming paradigm employed to implement the algorithm. The algorithms is not strictly required to be implemented this way, however this is highly recommended. `GridPlanner` is a high-level façade [19] that orchestrates the `Wavefront` and the `Grid`.

---

[5] The propagator set depends on the values of N_i.

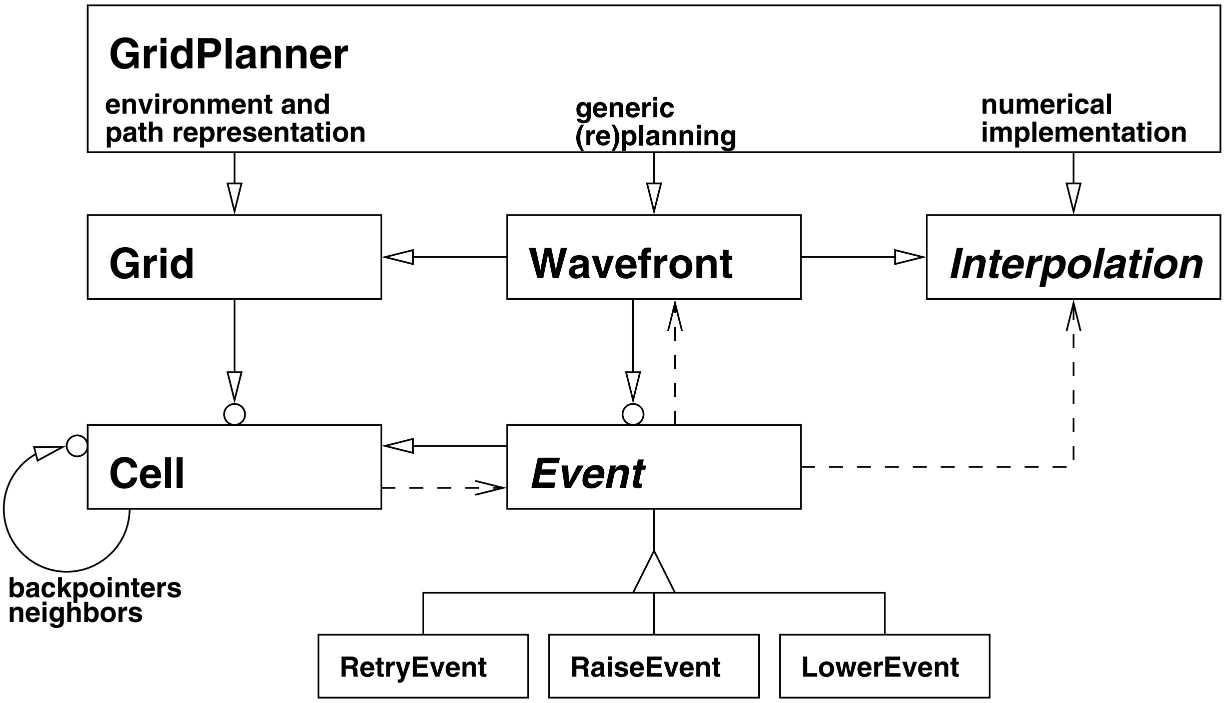


Figure 4.7: Overview of the entities in E*. Names in italics denote abstract classes, arrows denote references among objects. Arrows that terminate in a circle indicate one-to-many relationships, whereas dotted lines denote temporary relationships. Inheritance is shown using a triangle.

Communication with other software components goes through this façade as well. `Interpolation` is a strategy object [19] to keep the framework independent from the choice of interpolation method.

The environment model and navigation function values (path costs) are stored in the `Cell` objects that constitute the `Grid`. The environmental representation is referred to as *meta* information, stored in a cell's F_i. How exactly this meta information is used depends on the `Interpolation` object in place, but conceptually it encodes the cost or risk of traversing a given region. Other components are kept independent from this implementation detail by requiring changes to the world model to pass through `GridPlanner`, which uses the concept of normalized *risk* $0 \leq r_i \leq 1$. More details can be found in section 4.8.

The `Wavefront` plays a role similar to the OPEN list in A* and D*: It contains a queue of propagation events that is sorted such that the navigation function is built in an upwind manner. `Event` objects are an explicit representation of propagation steps, comparable to the implicit RAISE and LOWER states in D*.

Event Objects

An `Event` encapsulates the elemental propagation step. It contains information and functionality that is required to propagate local information to more distant cells. This includes pointers to the destination of the update. Figure 4.8 illustrates how events (the circles) *encapsulate* propagation functionality.

- Events are stored in a queue that ensures upwind propagation order. A *queue key* is assigned to each event, and the queue is sorted by ascending key.

- *Priority* information is dependent on the type of event, it is used to ensure that a cell never has more than one pending update request by resolving the conflict that arises when an event is to be created on a cell that is already queued for update.

- Events make it possible to write generic event handling code while keeping implementation details open for change. They provide a location for implementing propagation logic that would otherwise be intertwined with other code.

Three subclasses of the abstract `Event` are used to implement generic wavefront propagation: `LowerEvent` for path cost decreases, `RaiseEvent` for path cost increases, and `RetryEvent` for attempting to decrease a path cost after a wake of `RaiseEvents` has swept an area.

The queue key corresponds to the upper bound on the optimal path cost at which the event must be triggered in order to respect the upwind property. The upper bound on optimal path costs is a value maintained by the wavefront, all cells with values at or below this bound are valid (they can not be influenced by further event propagations). This property of the queue key is important for correct propagation results, especially during dynamic replanning.

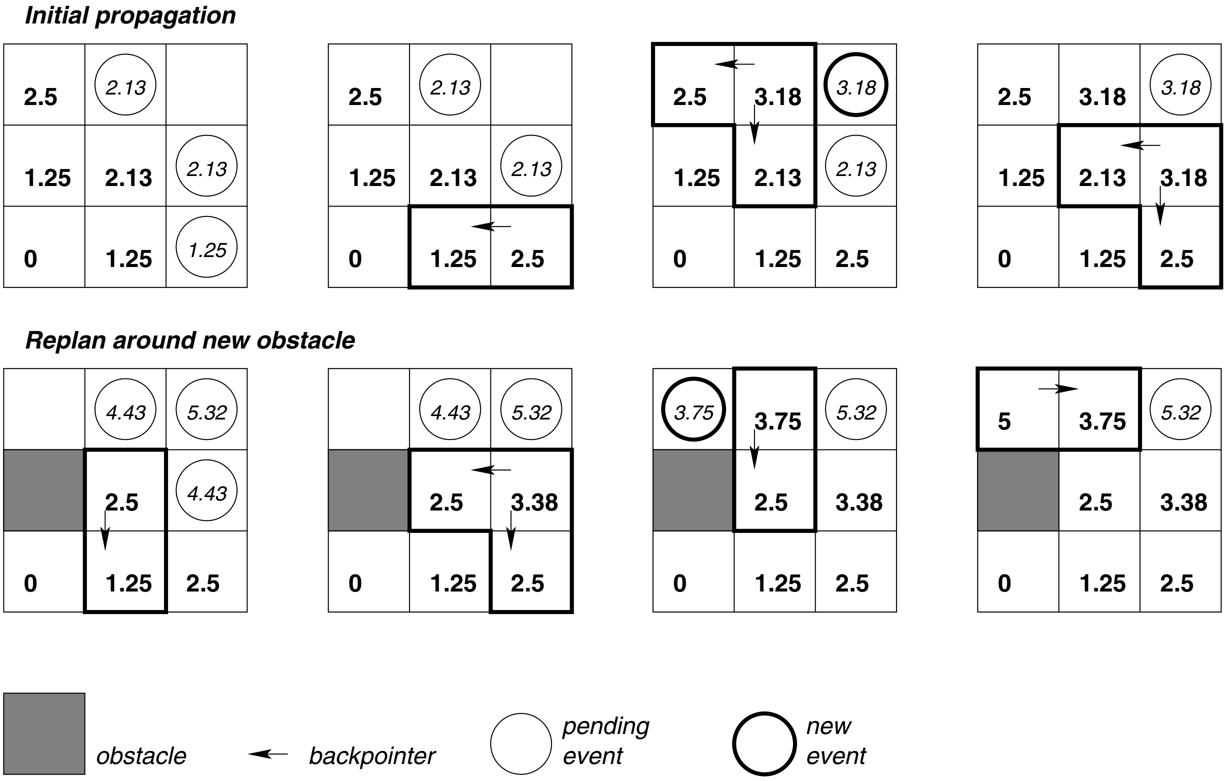


Figure 4.8: Example of event-based propagation: These two sequences illustrate the use of event objects for wavefront propagation on a grid with resolution $h = 1.25$. Bold numbers denote cell values T_i, empty cells are at $T_i = \infty$. Oblique numbers denote queue keys that are associated with each event, which are denoted by circles. Arrows are backpointers (shown only right after their creation to alleviate the illustration, but the information remains in the grid). The grey square is an obstacle cell which was added *after* completion of the initial propagation. Changes are highlighted using a thick border around the involved cells. The mechanism will be explained in more detail in subsequent sections, here the aim is to explain the role of event objects with the help of a concrete example.

`LowerEvent`s are sent to all neighbors of a cell whose value decreases, e.g. when a previously blocked passage is discovered to be open. The value *after* the decrease is used as queue key: The lower event was triggered because the highest known optimal path cost reached its target cell, lowering that cell's value implies the same decrease for the path cost bound. Propagating a lower event means calculating the best possible path cost estimate and updating the backpointers accordingly (unless the value was not decreased). Then, new lower events are sent to all neighbors whose value lies above the current one. Listing B.1 in appendix B shows the C++ implementation for propagating lower events.

`RaiseEvent`s are created when a cell's value increases, e.g. due to a previously undetected obstacle across the planned path. It is sent to all neighbors that have a backpointer to the cell, using the value *before* increase as queue key: It indicates that path costs higher than the cell's old value are now non-optimal, which is required such that the neighbors with backpointers to the just-updated cell get propagated next. When a raise event is propagated, the destination's value is set to infinity[6] and its backpointers set to null. Then, raise events are sent to all concerned neighbors in order to propagate the information upward. Finally, a `RetryEvent` is triggered on the cell that has just been increased, with a queue key determined to allow subsequent path cost decreases as soon as possible (the updated value plus the spacing h between cells). Listing B.2 shows the implementation of raise event propagation in C++.

A `RetryEvent` is the same as a `LowerEvent`, except for its higher priority. It acts as a sort of rear guard trailing behind wakes of `RaiseEvent`s to prevent backpointer loops and a "back wash" phenomenon (a duplicate raise wake going in the opposite direction of the original one).

Raise Events and Backpointer Consistency

There is a fundamental difference between `RaiseEvent` propagation in E* and RAISE state calculations in D*. The latter does not set the node's path cost to infinity, but calculates the usual path cost propagation. This is consistent with the single backpointers used in D*, but interpolation and the multiple backpointers needed to trace cell dependencies raise the following problem in E*.

Suppose a cell receives a raise event from one of its neighbors. If we now recalculated the interpolation, this would be likely to result in a change of backpointers because the previously good propagator is now at a higher path cost. However, the backpointers are needed to propagate a path cost increase to all *descendants* of a location: A descendant is a neighbor that has a backpointer to the cell. Changing a backpointer while this chain has not been completely traced back violates the upwind property and leads to inconsistent results of dynamic replanning.

An alternative would be to force the use of the same backpointers for propagating raise events. However, interpolation kernels typically fall back to their degenerate solutions in this case because the values of the two backpointers will tend to be far apart after one of

---

[6] The implementation of ∞ is a finite value much higher than the maximum meaningful accumulated path cost.

the backpointers has been raised. The fallback solution requires fewer backpointers than the non-degenerate case, which either violates the upwind property (especially since the non-raised backpointer will in all likelihood become the new single backpointer because it lies below the other one), or the backpointers do not correspond to the value of the cell. The first alternative creates the same problem described above, and the second alternative will create this problem as soon as the next wake of raise events reaches the cell with inconsistent backpointers.

These issues are addressed by setting a cell's value to infinity when a raise event is propagated, and passing on the raise wake to the unmodified backpointers. Now that the cell is at infinity, the backpointers become useless (the cell cannot be raised further in any case), which is why they are set to null to avoid creating spurious raise events that would result in no change of the navigation function. In any case, after a raise event, a cell will be subject to a retry event which applies the interpolation and sets new backpointers. Constructing sensible calculations for propagating raise events would thus be a waste of processing (and development) resources.

Wavefront

The `Wavefront` acts as event creator, queue, and sink. The separation from the grid makes the representation of the cells completely independent of the propagation logic.

During insertion of events into the queue, attention has to be paid to not overwrite existing ones. Events win by priority, or by their queue key in a tie. Priorities reflect the importance of events: `RetryEvent` > `RaiseEvent` > `LowerEvent`. Raise events are considered more important than lower events, because missing an existing shortcut is less critical than trying to go through a region that is known to be of high risk. Also note that raise events trigger retry events after they have been propagated: Retry events have the same effect as lower events, in other words a lower event is only delayed by the higher priorities of raise and retry events. Overriding raise by retry events is consistent because the latter are only triggered after a raise event has been propagated (the cell in question has an infinite value anyways, it cannot be raised further).

4.6 Interpolation

The algorithm presented above is independent of the interpolation method, but in order to actually use E* for planning, an implementation has to be provided. In this section, three possible approaches are presented. Their advantages and drawbacks will be investigated in section 4.7.

4.6.1 Graph Distance

The simplest "interpolation" kernel is one that uses only one propagator, in other words it does not interpolate at all. This is actually useful because it allows to quantify the effects of interpolation. In this case, E* becomes equivalent to D* on grids[7]. Update equation (4.8) and environment representation (4.9) for this non-interpolating kernel are fairly straightforward.

$$\begin{array}{l} T_0 = \min_{c \in N_0} (T_c + h + F_0) \\ B_0 = \arg\min_{c \in N_0} (T_c + h + F_0) \end{array} \tag{4.8}$$

$$F_i = \begin{cases} 0 & \Leftarrow \boldsymbol{c}_i \in \text{free space} \\ \infty & \Leftarrow \boldsymbol{c}_i \in \text{C-space obstacle} \end{cases} \tag{4.9}$$

where T_0 is the value after propagation, h is the grid resolution, F_0 is the meta information of the cell which is being updated, and N_0 is the set of its neighbors. Here, meta information is treated as the additional cost of going from a cell to the one that is being updated. This differs from the definition used in D*, which assigns such cost to the edges between nodes, but it is equivalent if all incoming edges of a node are set to T_c and the outgoing edges set to the T values of the other node.

In the implementation, this kernel is called `NF1Interpolation`. It is given in appendix B.2.

4.6.2 Huygens' Principle

The evolution of physical waves can be explained by Huygens' Principle, which interprets an arbitrarily shaped wavefront as an infinity of elementary spherical waves emanating from each point on the front. The envelope of the elementary waves at a later instant gives the wavefront at that instant.

Figure 4.9 shows how this principle can be used to develop an interpolation method for E*. In order to calculate the crossing time T_0 of a cell $\boldsymbol{c}_0$, we find two neighbors $\boldsymbol{c}_{1,2}$ of known crossing times $T_{1,2}$. Using the propagation speed F_0 at $\boldsymbol{c}_0$, the origin $\boldsymbol{s}$ of the elementary wave that lead to $T_{1,2}$ can be determined. Given $\boldsymbol{s}$, $\boldsymbol{c}_0$, and F_0, it is possible to calculate T_0 as a ratio of distance over speed.

This approach is valid provided that the elementary wave from $\boldsymbol{s}$ first crosses $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ before hitting $\boldsymbol{c}_0$. It is also necessary that, seen from $\boldsymbol{c}_0$, the wave come from between $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, otherwise the kernel would *extra*polate the wave. These conditions mean that $\boldsymbol{s}$ lies on the opposite side of x^P than $\boldsymbol{c}_0$ and inside the region spanned by $(\boldsymbol{c}_1 - \boldsymbol{c}_0)$ and $(\boldsymbol{c}_2 - \boldsymbol{c}_0)$. Finally, in order to preserve the upwind property of the algorithm, $T_0 \geq T_2 \geq T_1$ has to hold.

The following notations are used (see also figure 4.9). Positions are expressed as vectors, which can be either in the global (G) or propagation (P) frame of reference, denoted using a right superscript[8]. The unit vectors along x^P and y^P are expressed in the global frame but written $\boldsymbol{e}_{x,y}$ to simplify the notation. Equation (4.10) shows the definitions of given terms.

---

[7] The possible extension of E* to graphs is discussed in section 4.9

[8] In some expressions independent of the reference frame, the superscript is omitted.

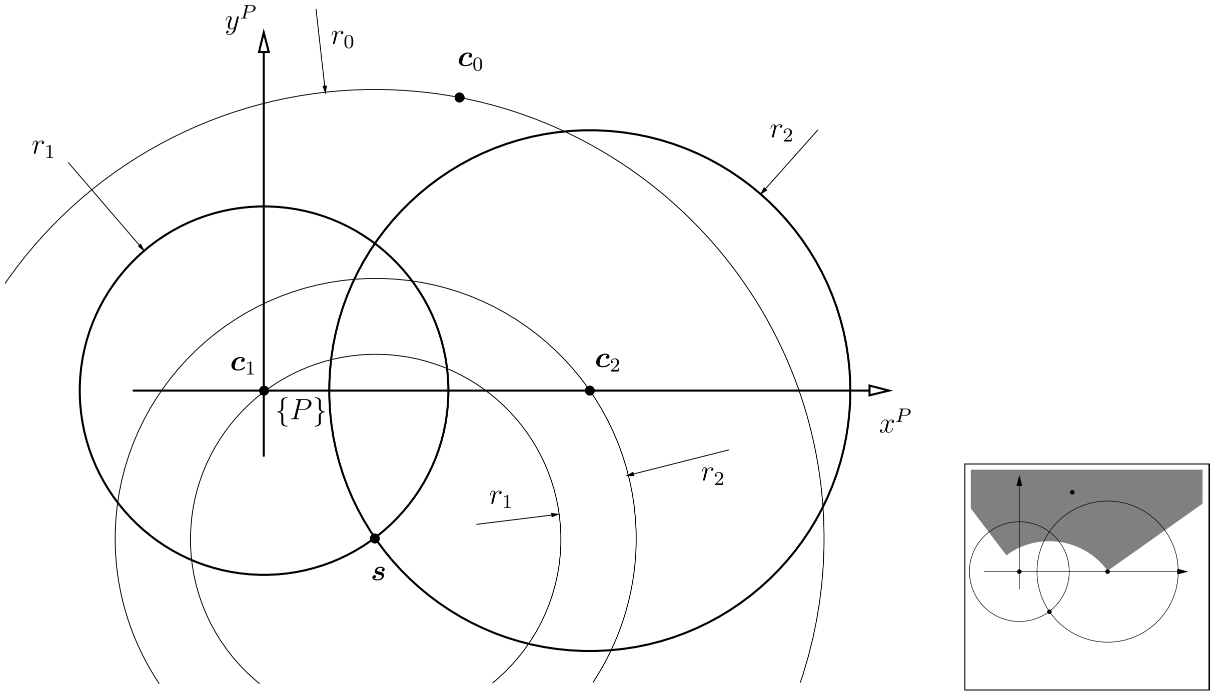


Figure 4.9: Interpolation by Huygens' Principle – reconstruction of the origin of the elementary wave that led to the crossing times at two known location $\boldsymbol{c}_{1,2}$ makes it possible to determine the crossing time at the cell of interest $\boldsymbol{c}_0$. The thick circles illustrate the calculations for finding the wave origin $\boldsymbol{s}$ and the thin circles show the wave from $\boldsymbol{s}$ to $\boldsymbol{c}_{0,1,2}$. The coordinate frame P is chosen to simplify the intersection calculations, its axes are x^P and y^P. The global frame G is not shown. The inset shows the conditions under which this method leads to valid upwind interpolation: c_0 has to be inside the shaded region.

$$\begin{array}{ll} \text{position} & \boldsymbol{c}^G_{0,1,2},\ \boldsymbol{c}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \\ \text{crossing times} & T_{1,2} \\ \text{propagation speed} & F_0 \end{array} \tag{4.10}$$

The first step is to calculate the wave origin $\boldsymbol{s}$ by intersecting two circles around $\boldsymbol{c}_{1,2}$. The calculations are performed in frame P, whose unit vectors are calculated according to (4.11). The wave source s^P is given in (4.12) which is a rewritten form of (3.19).

$$\begin{aligned} \boldsymbol{e}_x &= \frac{\boldsymbol{c}_2 - \boldsymbol{c}_1}{d} \\ \boldsymbol{e}_y &= \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \boldsymbol{e}_x \end{aligned} \tag{4.11}$$

$$\boldsymbol{s}^P = \begin{bmatrix} s^P_x \\ s^P_y \end{bmatrix} \begin{cases} s^P_x = \frac{d^2 + r_1^2 - r_2^2}{2d} \\ s^P_y = \pm\sqrt{r_1^2 - s^{P2}_x} \end{cases} \tag{4.12}$$

where $d = \|\boldsymbol{c}_2 - \boldsymbol{c}_1\|$ and the radii $r_{1,2} = \frac{T_{1,2}}{F_0}$ are calculated by observing that the propagation speed of the elementary wave that is to be reconstructed is given by the cell $\boldsymbol{c}_0$ for which the crossing time is to be determined. The sign of s^P_y is chosen to be the opposite of the sign of x^P_0. The expressions for $\boldsymbol{c}^P_{0,1,2}$ are given in (4.13).

$$\begin{aligned} \boldsymbol{c}^P_0 &= \begin{bmatrix} x^P_0 \\ y^P_0 \end{bmatrix} = [\boldsymbol{e}_x \boldsymbol{e}_y]^{\mathrm{T}} \left(\boldsymbol{c}^G_0 - \boldsymbol{c}^G_1 \right) \\ \boldsymbol{c}^P_1 &= \begin{bmatrix} x^P_1 \\ y^P_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \boldsymbol{c}^P_2 &= \begin{bmatrix} x^P_2 \\ y^P_2 \end{bmatrix} = \begin{bmatrix} d \\ 0 \end{bmatrix} \end{aligned} \tag{4.13}$$

Note that two cases can require a fallback solution (presented below): If $d < \epsilon$, then $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are considered to be the same point, and $\boldsymbol{e}_x$ would require a division by zero; and if $s^P_y \notin \mathbb{R}$, there is no valid intersection.

The final check before calculating T_0 is the verification if $\boldsymbol{c}_0$ lies in the region spanned by the half infinite lines $\boldsymbol{s} \to \boldsymbol{c}_1$ and $\boldsymbol{s} \to \boldsymbol{c}_2$. This is done using the sign of outer products (4.14). If $\boldsymbol{s}$ does not pass this test (which is illustrated in the inset of figure 4.9), the fallback solution is used. Otherwise, the crossing time at $\boldsymbol{c}_0$ is calculated according to (4.15). Finally, if $T_0 < T_2$ then the fallback solution is used instead, in order to respect the upwind property.

$$\text{valid } s \Leftarrow \begin{cases} x_0^P > 0 \quad \Rightarrow \begin{cases} \left(\boldsymbol{c}_0^P - \boldsymbol{s}^P\right) \wedge \left(\boldsymbol{c}_1^P - \boldsymbol{s}^P\right) > 0 \\ \cap \\ \left(\boldsymbol{c}_2^P - \boldsymbol{s}^P\right) \wedge \left(\boldsymbol{c}_0^P - \boldsymbol{s}^P\right) > 0 \end{cases} \\ x_0^P < 0 \quad \Rightarrow \begin{cases} \left(\boldsymbol{c}_0^P - \boldsymbol{s}^P\right) \wedge \left(\boldsymbol{c}_1^P - \boldsymbol{s}^P\right) < 0 \\ \cap \\ \left(\boldsymbol{c}_2^P - \boldsymbol{s}^P\right) \wedge \left(\boldsymbol{c}_0^P - \boldsymbol{s}^P\right) < 0 \end{cases} \end{cases} \tag{4.14}$$

$$T_0 = F_0 r_0 = F_0 \|\boldsymbol{c}_0 - \boldsymbol{s}\| \tag{4.15}$$

the validity check of $\boldsymbol{s}$ can also be written $(\boldsymbol{\Delta}_{s0} \wedge \boldsymbol{\Delta}_{s1}) \cdot (\boldsymbol{\Delta}_{s2} \wedge \boldsymbol{\Delta}_{s0}) > 0$ where $\boldsymbol{\Delta}_{ij} = \boldsymbol{c}_j - \boldsymbol{c}_i$. If any of the above validity checks fail, then the fallback solution (4.16) is used. However, if the fallback is used then $\boldsymbol{c}_2$ is not a backpointer of $\boldsymbol{c}_0$ because only $\boldsymbol{c}_1$ influences the equation.

$$T_0 = T_1 + F_0 \|\boldsymbol{c}_0 - \boldsymbol{c}_1\| \tag{4.16}$$

In order to determine the best interpolation for a given cell, a brute force approach is used: All combinations of neighbors that lie on different axes are tried, and the lowest solution is retained, preferring non-degenerate over fallback solutions. The implementation (named `HPRInterpolation`) is given in appendix B.2.

Reconstructing the source of an elementary wave implies encoding environmental information as propagation speeds. Obstacles can be represented by setting $F = 0$, because then the wavefront will never reach the obstacle (this requires special handling in the implementation to avoid divisions by zero). The maximum speed is achieved in cells for which the risk of collision is zero, and in order to keep the interpretation of goal distance, this maximum should be set to $F = 1$. The continuous range between zero and one can be used to finely model environment properties. For example, if a region should be circumvented in most cases, but can be traversed if the alternatives are long detours, a value of $F \approx 0.5$ could be used.

4.6.3 Gradient Approximation

Gradient approximation refers to an implementation of the first-order upwind interpolation scheme for Fast Marching Methods presented in [27]. It is constructed to satisfy the conditions of Level Set Methods, resulting in (4.17).

$$\max(D_{ij}^{-x}T, -D_{ij}^{+x}T, 0)^2 + \max(D_{ij}^{-y}T, -D_{ij}^{+y}T, 0)^2 = 1/F_{ij}^2 \tag{4.17}$$

where D_{ij}^{-x} is the finite difference operator along negative x at the grid point (i, j), and F_{ij} is the (known) propagation speed at (i, j). T corresponds to the navigation function that is to be calculated.

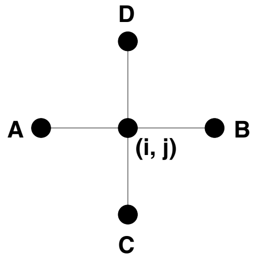


Figure 4.10: Cell neighborhood – the cell in the center is being updated. Interpolation implies using up to two neighbors, which need to lie on different axes.

Developing $D_{ij}^{\pm\{x,y\}}T$ leads to a quadratic equation with coefficients that take values based on a switch on the sign and magnitude of the finite difference operators. However, a geometrical interpretation is presented below. It helps visualizing the process. In particular, it will be used to determine the propagator set that yields the optimal solution prior to interpolating.

Geometrical Interpretation

Figure 4.10 shows the situation when updating a cell (i, j) (following the development in [27]). The cell in the center is being updated. Interpolation implies using up to two neighbors, which need to lie on different axes (one along x, the other along y). Without loss of generality, it can be assumed that the two neighbors leading to the best interpolation are **A** and **C**, and that $T_{\mathrm{A}} \leq T_{\mathrm{C}}$. The update equation becomes (4.18).

$$(T - T_{\mathrm{A}})^2 + (T - T_{\mathrm{C}})^2 = h^2/F^2 \;\Leftrightarrow\; \begin{cases} T = t_{\mathrm{A}} = t_{\mathrm{C}} \\ (t_{\mathrm{A}} - T_{\mathrm{A}})^2 + (t_{\mathrm{C}} - T_{\mathrm{C}})^2 = h^2/F^2 \end{cases} \tag{4.18}$$

where T is the value to be determined, T_{A} and T_{C} are the values of the best neighbors, h is the grid scale (the distance between two consecutive cells), and F is the propagation speed at cell (i, j).

The novel geometrical interpretation is based on introducing two parameters t_{C} and t_{A} that are interpreted as the axes of a Cartesian coordinate frame. The solutions for (4.18) are found at the intersections between the diagonal $t_{\mathrm{A}} = t_{\mathrm{C}}$ and a circle of radius h/F centered at $(T_{\mathrm{C}}, T_{\mathrm{A}})$.

The switch expressions surrounding $D_{ij}^{\pm\{x,y\}}T$ in (4.17) lead to constraints that need to be added to (4.18): Either it has a real solution T with $T > T_{\mathrm{C}}$, or a real solution to the degenerate form (4.19) with $T_{\mathrm{A}} < T \leq T_{\mathrm{C}}$. The degenerate (fallback) solution is equivalent to finding the intersection between a horizontal line $t_{\mathrm{A}} = T_{\mathrm{A}} + h/F$ and the diagonal $t_{\mathrm{A}} = t_{\mathrm{C}}$.

$$(T - T_{\mathrm{A}})^2 = \frac{h^2}{F^2} \;\Leftrightarrow\; \begin{cases} T = t_{\mathrm{A}} = t_{\mathrm{C}} \\ t_{\mathrm{A}} = T_{\mathrm{A}} + h/F \end{cases} \tag{4.19}$$

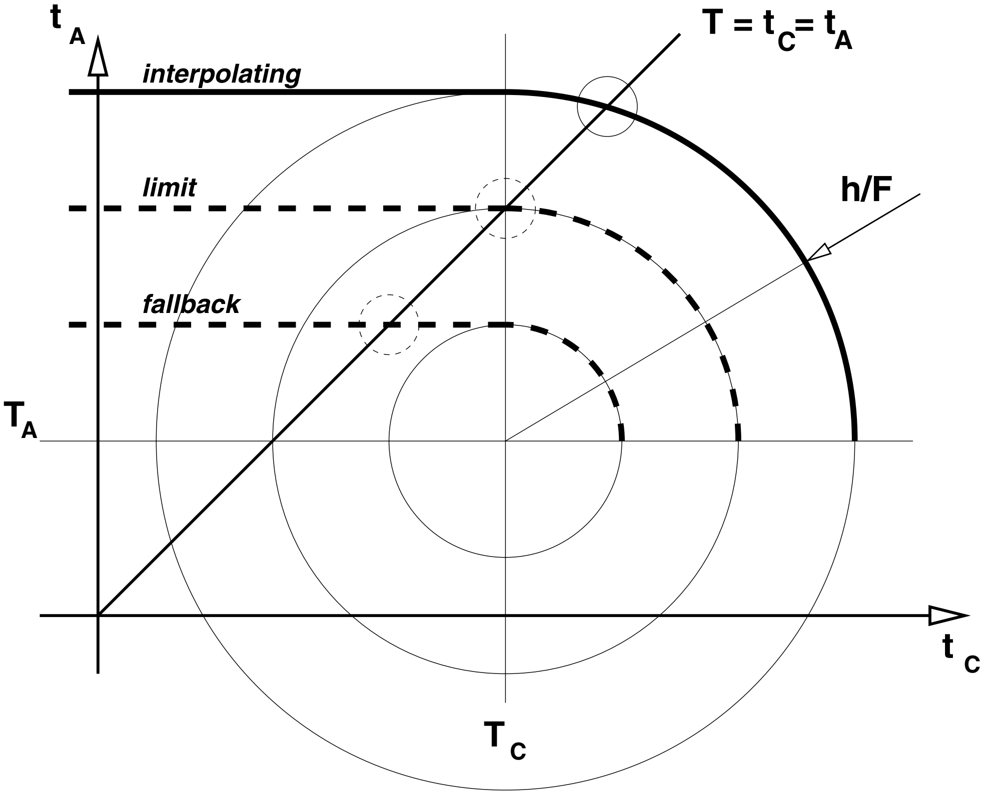


Figure 4.11: Geometric interpretation of LSM interpolation – equations (4.18) and (4.19) can be read as finding the intersection between the line $t_{\mathrm{C}} = t_{\mathrm{A}}$ and the curve labelled *interpolating* (thick solid line). Two dashed curves illustrate how the interpolation behaves when h/F becomes smaller (*limit* and *fallback* curves). The small solid circle indicates the intersection that serves as solution for the interpolating case, and the small dashed circles show the same for the limit and fallback cases.

Figure 4.11 shows the overall geometrical interpretation for a given $(T_{\mathrm{C}}, T_{\mathrm{A}})$. It can be seen that the quadratic equation (4.18) has to be solved only if the point $(T_{\mathrm{C}}, T_{\mathrm{A}} + h/F)$ lies above $t_{\mathrm{C}} = t_{\mathrm{A}}$ (i.e. the *interpolating* curve in figure 4.11), and that only the higher of the two intersections has to be found. The final equation is (4.20).

$$
\begin{aligned}
T &= \begin{cases} T_{\mathrm{A}} + h/F & \Leftarrow\ T_{\mathrm{C}} - T_{\mathrm{A}} \geq h/F \\ \frac{1}{2}\left(-\beta + \sqrt{\beta^2 - 4\gamma}\right) & \text{otherwise} \end{cases} \\
\text{where} & \begin{cases} \beta = -\left(T_{\mathrm{A}} + T_{\mathrm{C}}\right) \\ \gamma = \frac{1}{2}\left(T_{\mathrm{A}}^2 + T_{\mathrm{C}}^2 - h^2/F^2\right) \end{cases}
\end{aligned}
\tag{4.20}
$$

recall that $T_{\mathrm{A}} \leq T_{\mathrm{C}}$ and note that $F \rightarrow 0 \ \Rightarrow\ T \rightarrow \infty$, which is treated specially in the implementation for reasons of numerical stability. Also note that cells on the border of the grid might not have neighbors of type **A** and **C**, in which case the fallback solution is used. Listing B.5 shows the implementation in C++.

4.7 Evaluation and Performance Measurements

In order to verify that E* yields useful results, it is necessary to test the consistency of the framework and its ability to incorporate various interpolation kernels. It is also interesting to compare the performance of the interpolation methods presented in section 4.6. Performance measurements are presented this section, followed by a more qualitative experiments that show how E* performs as a global planner (section 4.8).

By comparing the navigation function with hand crafted "ground truth" distances in completely known environments it is possible to verify that it yields an approximation of the true distance to the goal, and also to evaluate the precision of different kernels. As the interpolated distance to the wavefront is measured along an approximated direction, it overestimates the increment in crossing time because the true direction yields the minimum distance. The error between ground truth and the navigation function is thus expected to be positive throughout the grid, except for small negative errors that can be attributed to floating point processing.

It is also important to evaluate the consistency of dynamic replanning: Does it produce the same results as reinitializing and replanning the whole grid? This is especially important in the case of environment modifications that occur while dynamic replanning is already in progress, and because it is intended to use dynamic replanning only up to the robot position which leaves un-propagated events in the queue. These events have keys that lie above the path cost at the robot position and thus cannot influence its movement, however they can not be simply erased because the encode pending changes that are likely to be necessary for correctly propagating future environmental changes.

Dynamic replanning is supposed to make it more efficient to change the environment model after a navigation function has been calculated. How much work can actually be saved? And does this depend on the kernel? By counting the number of event propagations with and without re-initialization, the performance increase can be measured.

1. The first suite of tests is run in an empty environment (lower left of figure 4.12). The comparison with ground truth (Euclidean distance to the goal) is straightforward in this case. All three interpolation kernels are applied and their precision compared.

2. After verifying the results in an empty environment, the same tests are run in a simple environment with obstacles (the "zig-zag" in figure 4.12). Ground truth is calculated by defining polygonal regions and associated way-points, a location's true distance is the sum of the Euclidean distance to the corresponding way-point and the way-point's distance to the goal.

3. Consistency and performance of dynamic replanning can be measured by simulating a robot's movement through an environment with unknown obstacles that are discovered when they enter the sensor range of the vehicle (the "zig-zag" and "maze" maps in figure 4.12).

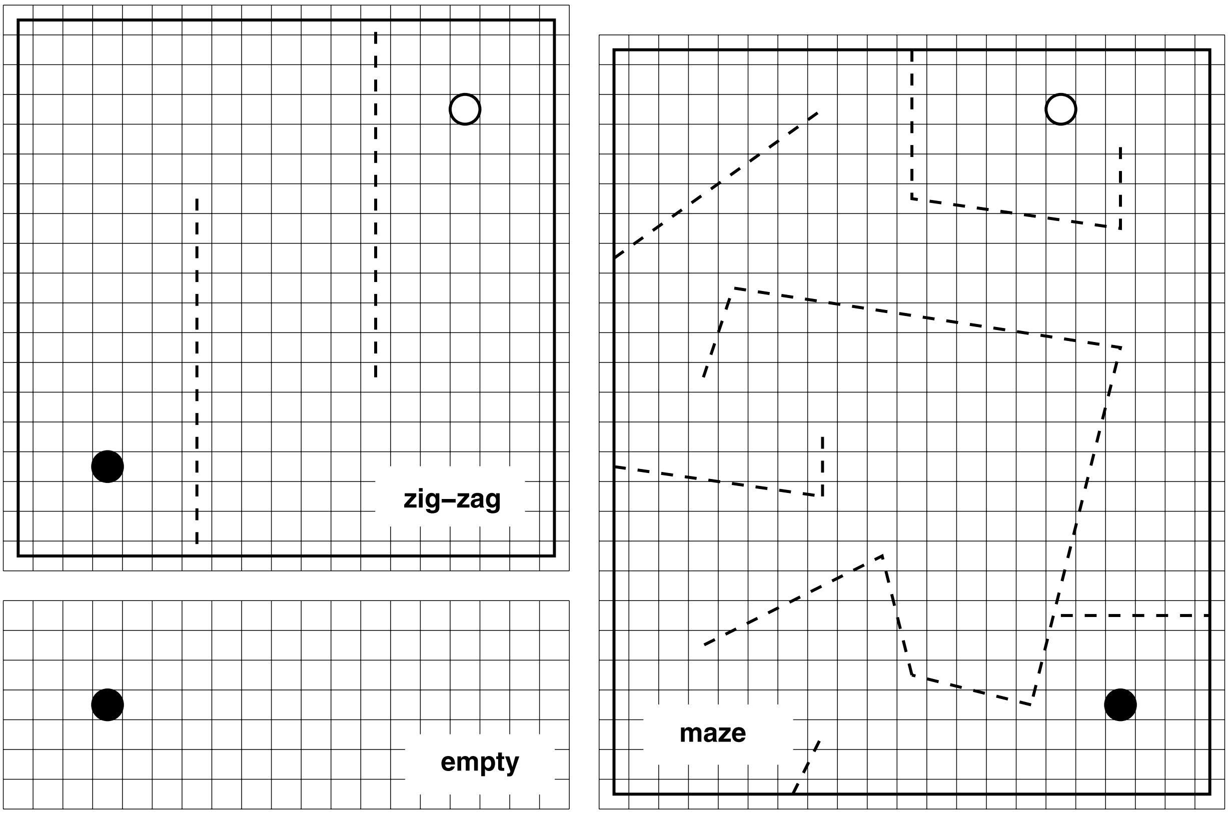


Figure 4.12: Simulation setup for evaluating E*– solid lines denote a-priori known obstacles, dotted lines indicate obstacles that need to be discovered by the robot during its movement. The empty circle is the start, and the solid circle the goal. The empty environment is used for determining in which way the initial goal radius influences the convergence towards the true Euclidean distance. The zig-zag and maze environments are used to determine the gain from dynamic replanning and to compare the path smoothness produced by different interpolation methods.
Actual cell sizes vary in the experiments. The grid resolution shown here is very coarse in order to illustrate how the grid is placed onto the environment.

The goal region is a circle around the goal point, all cells within this region are initialized to the Euclidean distance from their centers to the goal point. All cells outside the goal are initialized to infinity. To initialize the wavefront, the cells just outside the border of the goal are assigned lower events. Using larger goal radii increases the number of initial cells and gives the interpolation a better starting condition because on the goal border it can rely on true distance. In the extreme case, only one cell is in the goal region, and the kernels are forced to start out with fallback solutions (there are no cells available for secondary backpointer).

Note that for all these experiments, obstacle information is binary: A cell is either in free space (risk=0) or occupied (risk=1). Also, obstacles are not grown to the robot radius (the robot is considered a point). These simplifications are acceptable for evaluating the general characteristics of E*. A more elaborate planning approach that uses continuous risks, realistic robot sizes, and buffer zones around obstacles is presented in section 4.8.

Deviation from Ground Truth

The relative error of the propagation result is measured to investigate how the kernel, the resolution, and the goal radius influence the precision. Results are given in tables 4.1 (no interpolation), 4.2 (Huygens' Principle reconstruction, abbreviated HPR), and 4.3 (Level Set Method). Values inside the goal radius are not taken into account in the relative error calculation (4.21). Figure 4.13 summarizes the three mentioned tables.

$$e_{\mathrm{c}} = \frac{T_{\mathrm{c}} - d_{\mathrm{c}}}{d_{\mathrm{c}}} \tag{4.21}$$

where T_{c} is the value of the navigation function at cell $\boldsymbol{c}$ and d_{c} is the true distance from $\boldsymbol{c}$ to the goal.

None of the kernels underestimates the distance to the goal (not to within an error of 10^{-14} which is considered to be due to numerical effects). All improve their maximum relative error when increasing the ratio of goal radius over cell size, as expected. Note that the first run in each series of a given cell size is initialized using a single goal cell and thus indicates the effects of fallback solutions. This shows the non-interpolating nature of the NF1 kernel ($(2-\sqrt{2})/\sqrt{2} = 41.4\%$ for not being able to interpolate along the diagonal between two neighbors of the goal cell).

Running these ground truth comparisons in the zig-zag environment indicates that the wavefront is capable of "going around corners". The results of the NF1 and LSM kernels are given in table 4.4 and figure 4.14. The HPR kernel exhibits numerical instabilities for small cell sizes, resulting in incomplete propagation. Note the very high fidelity of the LSM kernel when the goal radius is sufficiently large (starting at four times the cell size, it produces maximum errors that lie below 3.5%).

Setup			NF1		
cell size	N$^\mathrm{o}$ cells	goal radius	min e_c	mean e_c	max e_c
1	341	1	0	20.7%	41.4%
		2	0	14.0%	33.3%
		4	0	8.06%	23.3%
0.5	1281	0.5	0	20.7%	41.4%
		1	0	16.8%	37.3%
		2	0	12.5%	31.9%
		4	0	6.58%	21.0%
0.1	30'401	0.1	$-5.27\cdot 10^{-15}$	20.6%	41.4%
		0.5	$-2.72\cdot 10^{-15}$	17.7%	39.0%
		1	$-2.60\cdot 10^{-15}$	15.0%	35.7%
		2	$-2.72\cdot 10^{-15}$	11.0%	30.3%
		4	$-2.83\cdot 10^{-15}$	5.96%	20.5%

Table 4.1: Relative error between propagation result and true distance in an empty environment of 10×30. The kernel based on graph distances presented in section 4.6.1 is used, which is not interpolating but yields results equivalent to NF1.

Setup			HPR		
cell size	N$^\mathrm{o}$ cells	goal radius	min e_c	mean e_c	max e_c
1	341	1	$-2.81\cdot 10^{-16}$	11.5%	29.3%
		2	0	11.8%	29.3%
		4	$-1.99\cdot 10^{-16}$	5.04%	14.9%
0.5	1281	0.5	$-2.81\cdot 10^{-16}$	14.9%	34.9%
		1	0	15.0%	34.9%
		2	$-1.99\cdot 10^{-16}$	8.80%	24.6%
		4	$-1.99\cdot 10^{-16}$	4.81%	14.9%
0.1	30'401	0.1	$-2.61\cdot 10^{-15}$	18.7%	41.4%
		0.5	$-1.97\cdot 10^{-15}$	16.4%	37.4%
		1	$-1.65\cdot 10^{-15}$	13.0%	32.5%
		2	$-1.63\cdot 10^{-15}$	8.92%	25.5%
		4	$-8.12\cdot 10^{-16}$	4.87%	15.9%

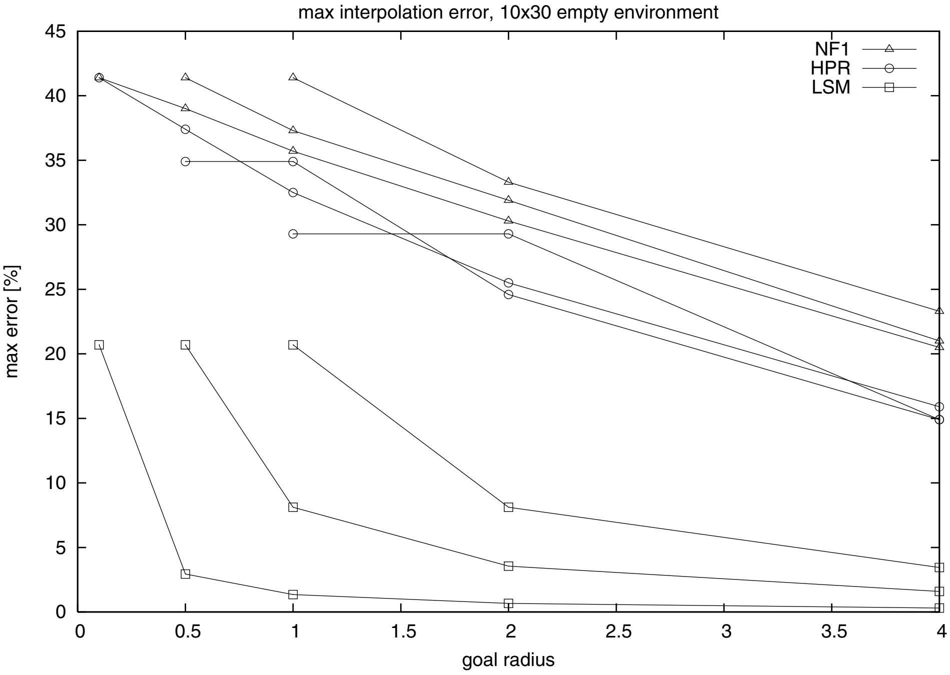
Table 4.2: Relative error between propagation result and true distance in an empty environment of 10×30. The kernel based on reconstructing Huygens' Principle was used (section 4.6.2).

Setup			LSM		
cell size	N° cells	goal radius	min e_c	mean e_c	max e_c
1	341	1	0	4.38%	20.7%
		2	0	2.88%	8.11%
		4	0	1.24%	3.45%
0.5	1281	0.5	0	2.96%	20.7%
		1	0	2.23%	8.11%
		2	0	1.27%	3.55%
		4	0	0.545%	1.59%
0.1	30'401	0.1	$-5.27\cdot10^{-15}$	1.00%	20.7%
		0.5	$-2.72\cdot10^{-15}$	0.609%	2.94%
		1	$-2.60\cdot10^{-15}$	0.410%	1.35%
		2	$-2.72\cdot10^{-15}$	0.232%	0.661%
		4	$-2.83\cdot10^{-15}$	0.0985%	0.302%

Table 4.3: Relative error between propagation result and true distance in an empty environment of 10 × 30. LSM refers to the interpolation based on Level Set Methods, presented in section 4.6.3.

cell size	N° cells	goal radius	mean e_c		max e_c	
			NF1	LSM	NF1	LSM
0.92	1'249	1	30.2%	9.04%	44.5%	20.7%
		2	27.2%	8.52%	43.0%	18.0%
		4	23.5%	8.05%	40.3%	17.6%
0.5	4'429	0.5	28.5%	5.32%	41.4%	20.7%
		1	26.7%	5.04%	39.6%	10.5%
		2	24.7%	4.66%	38.7%	10.2%
		4	21.6%	4.47%	36.9%	10.0%
0.1	109'661	0.1	27.0%	1.32%	41.4%	20.7%
		0.5	25.8%	1.17%	39.1%	2.94%
		1	24.5%	1.08%	36.2%	2.29%
		2	22.5%	1.01%	35.2%	2.25%
		4	19.7%	0.973%	33.6%	2.22%

Table 4.4: Relative error of E^* in a 33×33 zig-zag environment. The cell size of 1 has been adapted to 0.92 such that the grid better fits the environment.



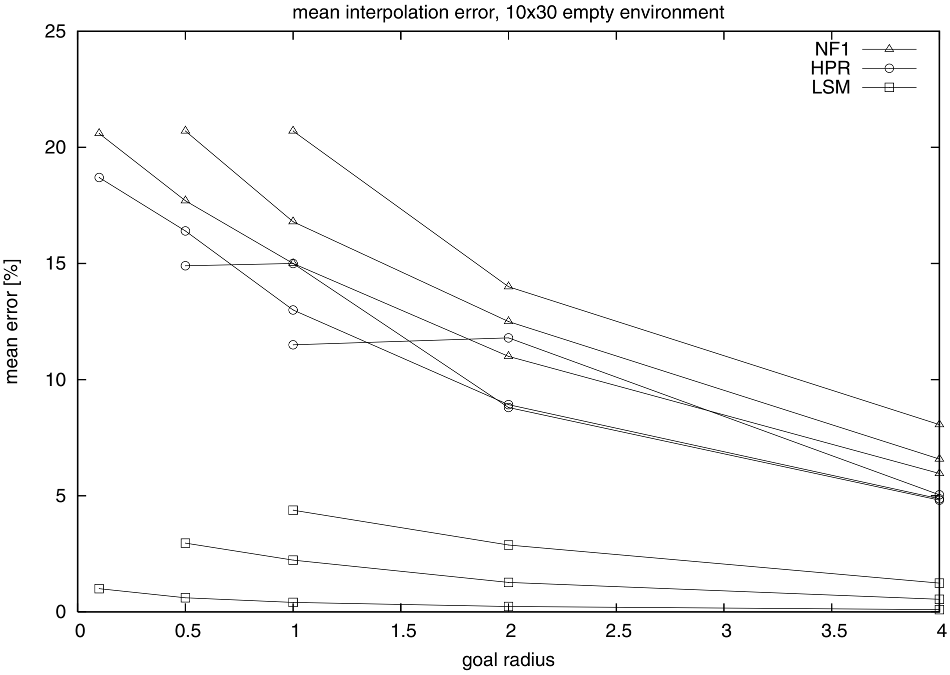
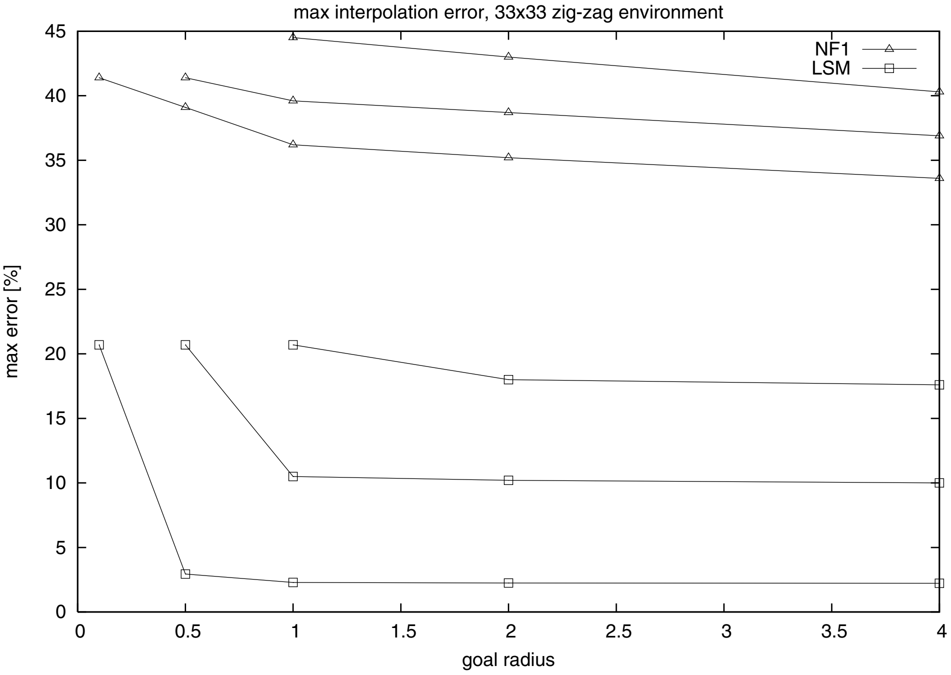


Figure 4.13: These two plots summarize tables 4.1, 4.2, and 4.3. The top graph shows the maximum error e_c in function of the interpolation method and the goal radius. The bottom graph shows the mean error. It can be seen that LSM performs better than HPR, which performs slightly better than NF1. The three lines per interpolation method correspond to the three cell sizes: 1 at the top, 0.5 in the middle, and 0.1 at the bottom. Note the case where the cell size equals the goal radius, it illustrates the maximum error because the interpolation can not take advantage of a smoothly initialized goal region, because the goal consists of a single cell.



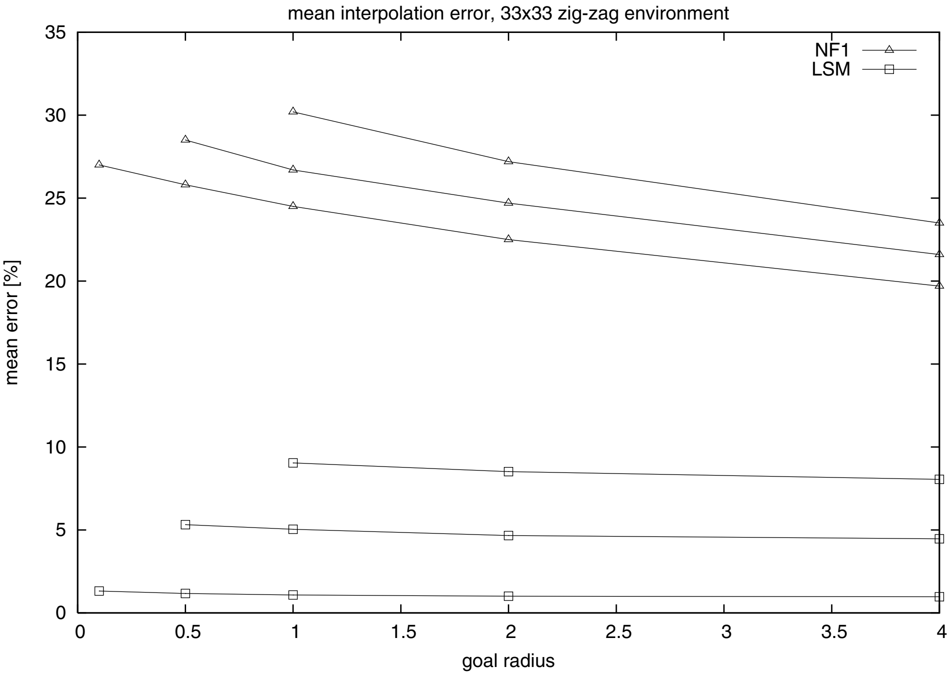


Figure 4.14: These two plots summarize table 4.4. As in figure 4.13, the top graph shows the maximum error and the bottom graph shows the mean error. As mentioned in the text, HPR has been left out of this experiment due to numerical instabilities. The overall heightened error is due to the difference between ground truth calculation (which can use the exact endings of the interior walls) and the propagation (which is forced to go through the first non-occupied cell near the end of each wall).

Performance and Consistency

Dynamic replanning is interesting because it eliminates the need to recalculate the propagation throughout the whole environment, but how much work can actually be saved? This depends on the environment and how many changes to the meta information occur before the robot reaches the goal. These experiments were performed on the zig-zag and maze environments. A simple robot (point size, mass evolving under a force equal to the negated gradient[9], bounded acceleration and speed) equipped with a sensor of limited range moves through the environment. When a previously unknown obstacle is detected[10], propagation is performed until the highest known optimal path cost lies above the value at the robot's position. During replanning, the robot is stopped. The computational complexity is measured by counting the number of events that are propagated until the robot resumes movement. The number of events required for full replanning (until the highest optimal path cost lies above the robot) is also counted.

In order to determine the effects of interpolation, runs are performed with and without interpolation. In the latter case, the method is equivalent to D*. It requires fewer events because there is only up to one backpointer in the non-interpolated case. The LSM kernel requires up to two backpointers, so raise events are propagated to up to twice as many descendants than for the NF1 kernel. The additional computational burden needed to obtain smoother paths is reflected in the higher number of events for LSM interpolation.

The parameters that were kept fixed during all those runs are the following:

- Size of the environment: 20 × 20m for zig-zag, 20 × 25m for maze.
- Robot parameters: Maximum speed 1m/s, maximum acceleration 2m/s^2, sensor range 2m, simulation timestep 0.1s.
- Goal radius 0.5m.

Table 4.5 and figure 4.15 compare the performance with and without interpolation. The gain is calculated without taking into account the initial planning. The path length indicates that interpolation produces better navigation functions. The meanings of the line labels in these tables are the following:

- **cell size** is the grid spacing h e.g. in (4.18). Sizes have been chosen to roughly equal $\{0.8; 0.4; 0.2\}$ but adapted such that the bounds of the environment exactly fit on cell centers.
- **N^o cells** is the total number of cells in the grid. This number is shown to give an idea of the relative amount of work required for replanning.

---

[9] The mean along x and y of $D_{ij}^{\pm\{x,y\}}$ is used as gradient

[10] Obstacles are discovered line-wise to simplify the simulation. Long lines are subdivided into segments to keep the information gain local.

- **with init. / without init.** indicates event counts with or without the very first propagation wake. This initial planning is the same regardless of dynamic or complete replanning.
- **N$^{\mathrm{o}}$ dyn. propagations** is the event propagation count over the whole run when using dynamic replanning (the sum of event counts for each replanning triggered by the discovery of a new obstacle).
- **N$^{\mathrm{o}}$ replan prop.** is the event count over the whole run when using complete replanning. Comparing this number to the previous one quantifies the gain of dynamic replanning.
- **gain** is the relative reduction in the number of events when using dynamic replanning. It is calculated as $(n_{\text{complete}} - n_{\text{dynamic}})/n_{\text{complete}}$.
- **path length** is the length of the robot trajectory. It gives a rough idea of path smoothness. Shorter paths are usually smoother.

There is a performance increase when using interpolation. The cost of this improvement is split into two aspects: An increase of computational complexity (number of operations) due to the higher number of backpointers, and an increase in the duration of each operation due to the more elaborate calculations required for interpolation. Table 4.6 compares the propagation counts to measure the complexity increase, and table 4.7 presents operation timing measurements. The former depends on the environment due to the backpointers' dependency on events which in turn depend on when the robot discovers which parts of the environment, whereas the latter is concerned with the operation of the kernels and depends on the platforms and optimizations used to execute the program. The complexity increases by up to 62% in the studied settings. The operation cost is up to 37% higher. Note that the operation cost measurements are for a single calculation of the interpolation kernel, which requires only a few floating point operations and some binary branches. Overall run times are also influenced by the complexity of insertion and removal operations on the event queue, which is $O(n \log n)$ if balanced binary trees are used in the implementation.

Figure 4.16 shows the improved path quality produced by the interpolated method. On the left, the preference for displacements along coordinate axes and the two diagonals illustrates the grid-based distance measure of NF1, whereas the paths produced using interpolation are very close to the line-of-sight towards the edge of a known obstacle (or the goal if no known obstacle is in the way).

These measurements show the performance gain from using dynamic replanning, but is this consistent with the results that would be obtained with complete replanning? Consistency of dynamic replanning means that the result of repairing the navigation function is exactly the same as what would be achieved by re-initializing the grid with the obstacle and goal information and replanning from the goal. More importantly for dynamic replanning, namely the propagation of raise events along backpointers, consistency means that the backpointers of each cell are identical for the two cases. This has been verified using

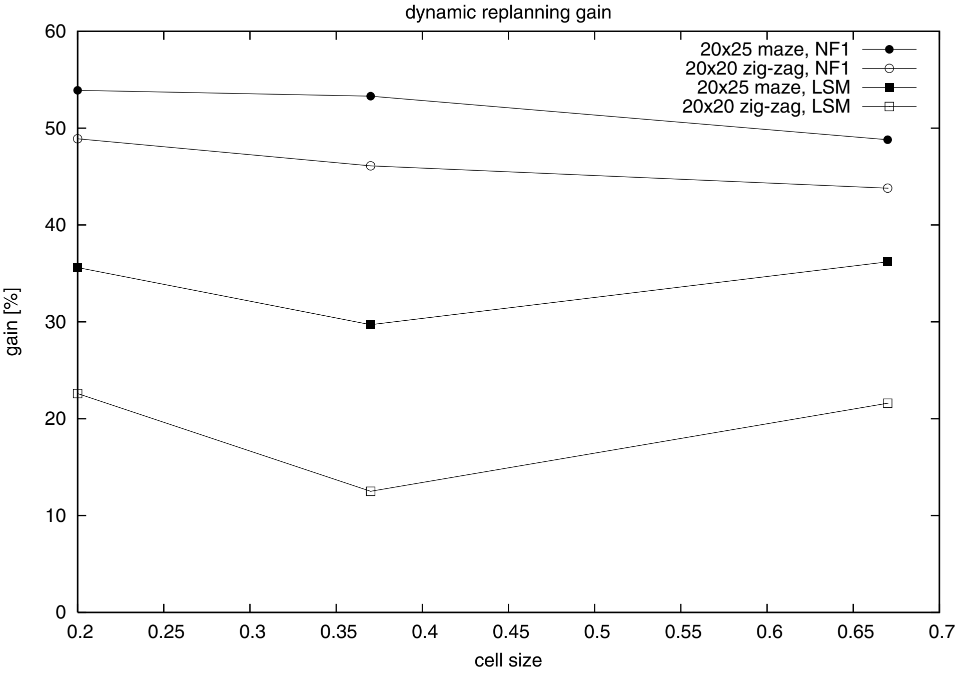
NF1 kernel in zig-zag map				
	cell size	0.67	0.37	0.20
	N° cells	961	3'025	10'609
with init.	N° dyn. propagations	2'048	6'596	23'289
	N° replan prop.	3'015	9'949	35'888
without init.	N° dyn. propagations	1'242	3'926	13'627
	N° replan prop.	2'209	7'279	26'226
	gain	**43.8** %	**46.1** %	**48.9** %
	path length	**44.1**	**42.8**	**42.1**

LSM kernel in zig-zag map				
	cell size	0.67	0.37	0.20
	N° cells	961	3'025	10'609
with init.	N° dyn. propagations	2'565	9'082	29'889
	N° replan prop.	3'049	10'002	35'821
without init.	N° dyn. propagations	1'759	6'417	20'260
	N° replan prop.	2'243	7'337	26'192
	gain	**21.6** %	**12.5** %	**22.6** %
	path length	**41.2**	**39.4**	**38.0**

NF1 kernel in maze map				
	cell size	0.71	0.38	0.20
	N° cells	1'044	3'498	12'726
with init.	N° dyn. propagations	4'184	14'233	52'524
	N° replan prop.	7'544	27'806	103'763
without init.	N° dyn. propagations	3'521	11'890	43'836
	N° replan prop.	6'881	25'463	95'075
	gain	**48.8** %	**53.3** %	**53.9** %
	path length	**107**	**97.8**	**91.3**

LSM kernel in maze map				
	cell size	0.71	0.38	0.20
	N° cells	1'044	3'498	12'726
with init.	N° dyn. propagations	6'245	21'653	85'139
	N° replan prop.	9'346	29'646	126'530
without init.	N° dyn. propagations	5'464	18'887	74'880
	N° replan prop.	8'565	26'880	116'271
	gain	**36.2** %	**29.7** %	**35.6** %
	path length	**102**	**90.7**	**88.9**

Table 4.5: E* performance with and without interpolation on a 20×20 zig-zag and a 20×25 maze map. The gain from dynamic over complete replanning is slightly worse when interpolating. The path length is an approximate indicator of smoothness.



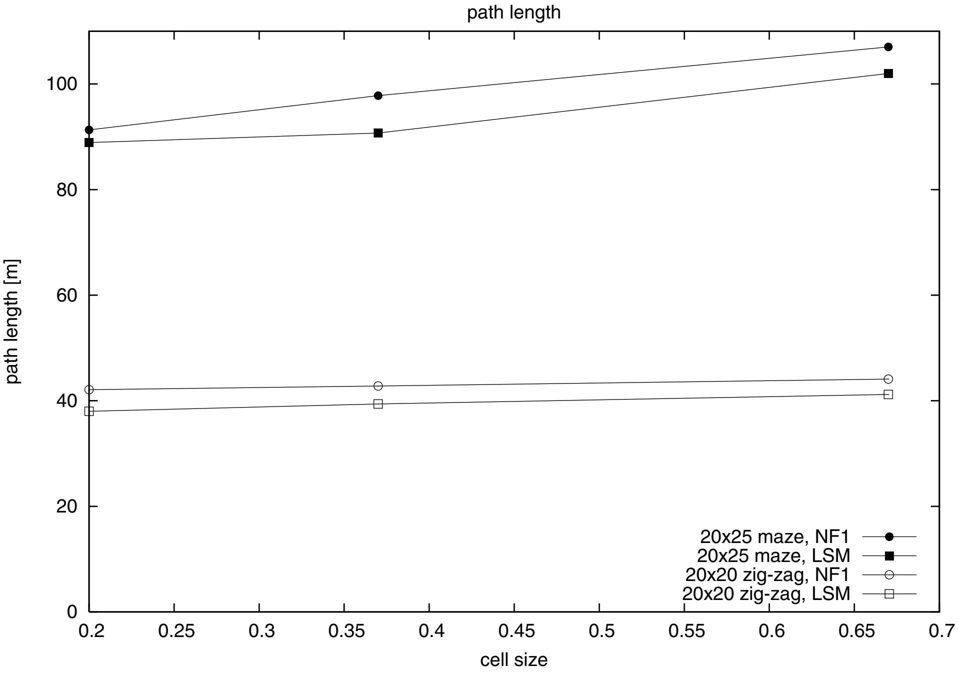


Figure 4.15: These plots illustrate the data in table 4.5. The top graph shows the gain from using dynamic over full replanning, it can be seen that using interpolation causes the gain to drop. This is due to the larger "spread" of the replanning wakes, caused by the higher number of backpointers. The bottom shows the overall path length, an indication that the LSM produces smoother (hence shorter) paths than the non-interpolated approach. The dependency on cell size is more or less random, except for the heightened border effects (the relative number of border cells grows when the size diminishes).

Relative computational complexity / zig-zag			
cell size	0.71	0.38	0.20
relative burden ρ(NF1)	0.562	0.539	0.520
relative burden ρ(LSM)	0.784	0.875	0.774
ρ(LSM)/ρ(NF1)	1.39	1.62	1.49

Relative computational complexity / maze			
cell size	0.71	0.38	0.20
relative burden ρ(NF1)	0.512	0.467	0.461
relative burden ρ(LSM)	0.638	0.703	0.644
ρ(LSM)/ρ(NF1)	1.25	1.50	1.40

Table 4.6: E* computational complexity with and without interpolation. These numbers are propagation counts that indicate the relative complexity of interpolation. The theoretical increase is twofold (twice as many backpointers), however the values observed in the zig-zag and maze environments lie between a 25% and 65% percent increase. $\rho = n_{\text{dynamic}}/n_{\text{complete}}$ is a normalized measure of the number of dynamic replanning events. ρ(LSM)/ρ(NF1) indicates how much wider raise events spread when using interpolation.

Relative and absolute operation cost					
		System A		System B	
		debug	optimized	debug	optimized
	overhead [ns]	33.0	31.3	11.3	10.9
	N° calls	513’939	513’971	513’939	513’971
NF1	kernel [ns]	33.7	31.6	12.8	11.0
	optimum [ns]	225	181	77.2	67.0
LSM	kernel [ns]	46.3	34.6	15.4	12.1
	optimum [ns]	273	185	89.8	67.1
LSM / NF1	**kernel**	**1.37**	**1.09**	**1.21**	**1.10**
	optimum	**1.21**	**1.03**	**1.16**	**1.00**

Table 4.7: E* operation cost with and without interpolation in maze environment (20×25, cell size 0.2). The operations of the LSM kernel are up to 37% more expensive than the non-interpolating ones. The times for single kernel calculations and finding the optimum (lowest) interpolation for a cell were measured under desktop system load. System A is a 466MHz Intel Celeron running linux-2.6.7, system B is a 1.8GHz Intel Pentium 4 running linux-2.4.22. The measurements were performed with debug and optimized executables produced with the `-g` and `-O3` flags of GCC-3.3.

the same setups as for the performance measurements presented above. No discrepancies have been detected.

4.8 Global Planning with E*

Previous sections of this chapter presented the core of E*. It produces smooth navigation functions that combine the advantages of interpolation and dynamic replanning, provided the meta information corresponds to the environment and the choice of interpolation method. However, using the core of E* still requires a relatively deep understanding of the involved entities. This section describes the next higher level in the robot control structure: Providing a convenient way for using E* in global path planning. Section 4.8.1 presents `GridPlanner`, a façade [19] that provides a simple interface for setting obstacle information and retrieving the gradient of the navigation function. The results that can be obtained with `GridPlanner` are presented in section 4.8.2.

4.8.1 `GridPlanner` Usage

The `GridPlanner` façade makes it easy to use the E* algorithm for global grid based path planning. In particular, it takes care of creating the `Grid`, `Wavefront`, and `Interpolation` instances based on parameters provided by the user. In addition, it manages the environment model such that it is appropriate for the E* algorithm. Thus, obstacle additions and removals (as well as direct manipulations of *relative risk*) are handled through this interface.

States

`GridPlanner` defines states that correspond to various planning phases, from modification of the goal or environment information to propagation of path costs. Figure 4.17 illustrates the associated finite state machine. The following list provides more details about the states.

`IDLE` No pending modifications of obstacle information and no pending events for the path cost propagation. Calling `PropagateAll()` will normally leave the planner in `IDLE` state (unless an error occurs). If the goal of obstacle information is modified, the planner will respectively go into states `UPDATE_OBSTDIST` or `UPDATE_PATHCOST`.

`RESET` Transitional state that allows correct initialization of the planner's state after setting a goal, or after a call to `Reset()`.

`ERROR` This state can only be left by calling `Reset()`. It signals an internal error. Occurrence of this state indicates a bug in the implementation.

`UPDATE_OBSTDIST` The planner is in the process of updating obstacle distance information. This is done by propagating a specialized wavefront that starts from the obstacles

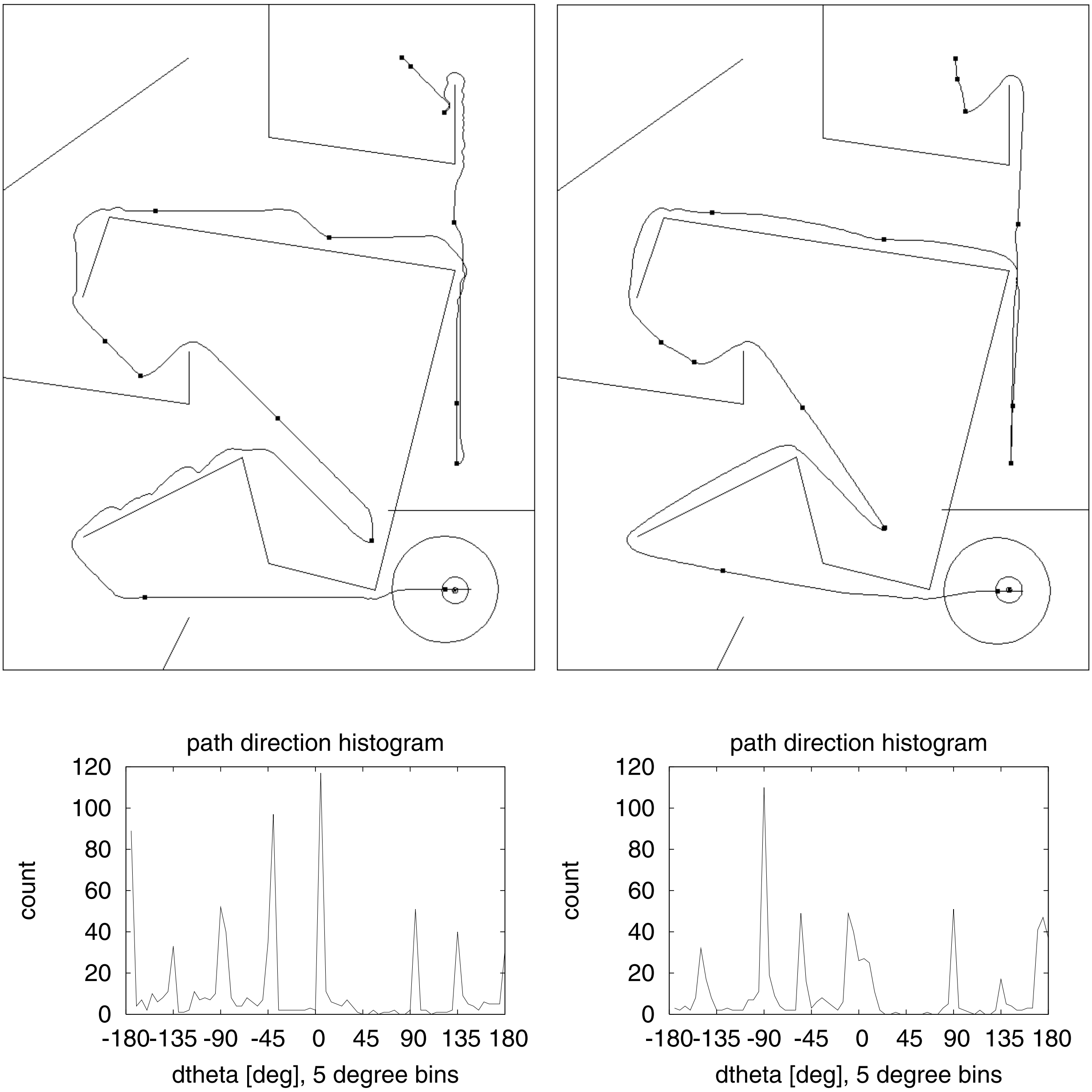


Figure 4.16: Example paths from the simulations.
The top row shows the final paths: On the left, no interpolation was used and the path shows grid effects. On the right, interpolation has been used to achieve a better distance map. The thick dots on the trajectory indicate where replanning occurred. These paths correspond to the experiments at cell size 0.2 in the maze environment (table 4.5). Path lengths are 91.3 on the left and 88.9 on the right.
The bottom row shows histograms of the path directions: On the left, you can see that the NF1 prefers directions along multiples of $\pi/4$, whereas the plot on the right indicates there is less preference for these angles in LSM. Note that the environment presents sections where the best path direction lies very close to a multiple of $\pi/4$, hence the remaining peaks in the LSM histogram.

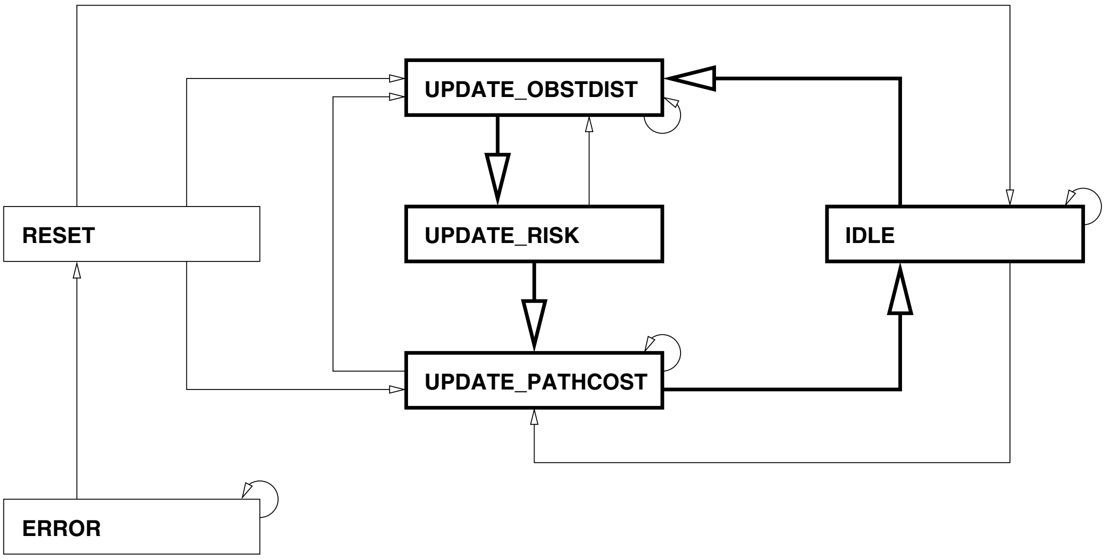


Figure 4.17: States of `GridPlanner` – thick lines highlight the main states and the normal transitions between them. `IDLE` is usually followed by `UPDATE_OBSTDIST` after obstacle information has been modified. Then, the planner performs an update of meta information when in state `UPDATE_RISK`, a transitional state that is followed by `UPDATE_PATHCOST`. The latter is the state in which actual path costs are propagated through the grid. The planner goes back to `IDLE` after this task has been completed. States `RESET` and `ERROR` are not directly related to stages of the E* algorithm, they are used for resetting the planner and signalling internal errors. Table **??** provides more details.

and sweeps into free space, calculating the distance to the closest obstacle for each location. As soon as the lower bound on the known obstacle distance reaches the sum of robot radius and width of the transitional zone, the planner will switch to `UPDATE_RISK`.

`UPDATE_RISK` Just after completing the calculation of the obstacle distance map, the distances are mapped through the `RiskMap` and `Interpolation` instances to calculate the meta information of each cell. Changes to this meta information trigger lower and raise events, so the next state normally is `UPDATE_PATHCOST`.

`UPDATE_PATHCOST` After completing the update of the meta information, any changes to the environment model are propagated. During this phase, the meta information does not change. Calling `PropagateTo()` usually leaves the planner in this state.

Allocation

In order to facilitate creation and initialization of `GridPlanner` instances, a factory method called `Create()` is provided. It takes a special `Parameters` object which regroups all parameters that are important for creating the planner object. The most important parameters are described in the following.

double `robot_radius` The radius of the robot. This value is used to "blow up" obstacles for the $\mathcal{C}$ projection that allows to plan without taking into account the robot's orientation.

double `transition_width` The width of transitional buffer around the robot. Within this distance in addition to the robot radius, the risk decreases from 1 to 0. How exactly this monotonically decreasing mapping is done depends on the chosen `RiskMap`. The transition width acts as a sort of buffer distance, which can be violated if the passage is narrow but which usually keeps the robot from grazing walls.

int `xdim` The width (x-dimension) of the grid, measured in the number of columns. Together with the grid height and scale (cell size h) this defines the workspace region covered by the planner.

int `ydim` The height (y-dimension) of the grid, defines the number of lines in the grid. Together with the grid width and scale (cell size h) this defines the workspace region covered by the planner.

double `scale` The cell size h or *scale* of the grid defines the resolution in meters. Together with the grid width and height this defines the workspace region covered by the planner.

string `interpolation_name` The interpolation method is chosen by name as well. The purely graph based method of section 4.6.1 is called "nf1" due to it's similarity with the NF1 function, the reconstruction of Huygens' Principle (section 4.6.2) is chosen by specifying "hpr", and the Level Set Method's gradient approximation of section 4.6.3 is selected using "lsm".

string `risk_map_name` This name selects which risk map should be used for translating obstacle distance to relative risks (0 signifying free space, 1 indicates a locations of guaranteed collisions). `RiskMap` instances take the robot radius and transition zone width as parameters, the `risk_map_name` defines the *type* of mapping, possible values are "linear" or "quad", the latter being a quadratic function tangent to the x-axis at the transition to zero risk.

Accessors

Accessors allow querying the planner about information that is required for integrating `GridPlanner` into the rest of the robot software. The most important accessor methods are listed below.

state_t `GetState`() Returns the current state of the planner.

bool `IsReachable`(*position_t* pos) Determines whether a given location is reachable under the currently known obstacle information. The returned value is only accurate if there are no pending obstacle modifications.

bool `IsGoal`(*position_t* pos) Returns `true` if the given position lies inside a cell of the goal region. Useful for determining when the robot has reached the goal.

bool `IsObstacle`(*position_t* pos) Returns `true` if the given position is an obstacle (added with one of the `AddObstacle`() methods). Non-obstacle cells can still have a risk of 1 because of the risk map, which grows obstacles by one robot radius.

double `GetPathcost`(*position_t* pos) Returns the path cost at a given location, at the current state in the propagation. The returned value is only optimal if there are no pending obstacle changes and the pathcost wavefront has been propagated far enough (i.e. if GetState() == IDLE or just after a call to the PropagateTo() method). If the provided location is not in the grid, -1 is returned.

position_t `CalculateGradient`(*position_t* pos) Calculate the gradient at a given location. The robot can be controlled by making it follow the *negated* gradient. The returned gradient is only optimal if there are no pending obstacle changes and the pathcost wavefront has been propagated far enough (i.e. if GetState() == IDLE or just after a call to the PropagateTo() method). If the location is not in the grid, $(0, 0)^T$ is returned.

Navigation Methods

The navigation interface of `GridPlanner` is the part that is designed to provide high-level control of E* when it is in charge of translating obstacle information into meta values and propagating the path cost from goal to robot. A selection of its most important methods is given here.

`Reset`() Resets the planner. All components (path costs, obstacles, goal, wavefront) are reset to initial values (i.e. infinite path costs, empty environment, no goal specified).

`SetGoal`(*Goal* goal) Resets path costs to infinity outside goal and the provided initial values (encapsulated in a `Goal` object) inside the goal. Initializes the wavefront to the new goal, but leaves obstacle information untouched.

`PropagateAll`() Calculates the path cost throughout the environment by propagating first the obstacle distances, then mapping the risks, and finally propagating the cell values until all events have been processed. It is usually not necessary to use up all events, especially if the robot is already close to the goal. Using the `PropagateTo`() method is thus preferable.

`PropagateTo`(*position_t* pos) Calculates the path costs until the given position lies inside the region of known optimal path costs. If an invalid position is specified (one that lies outside the grid) this method returns without taking any other action. Method `IsReachable`() is provided to test if the wavefront has been propagated to the wanted position.

Environment Model

Part of the `GridPlanner` interface is dedicated to managing environment information at a high level, allowing to add and remove obstacles using global coordinates and even to directly specify risk values (overriding the risk that would otherwise be obtained using the obstacle distance and risk map). These methods are listed below. Note that the $\mathcal{C}$ transform is simplified by projecting along θ, that is to say obstacles are grown by the robot radius. In addition, a buffer zone can be inserted to implement custom transitions from risk 1 ($\mathcal{C}$ obstacle) to risk 0 (free space). This is done through a `RiskMap` strategy object.

`AddObstacle`(*position_t* pos) Adds a location (specified either in global coordinates or with grid indices) to the set of obstacles. This does *not* immediately change the risk or meta information associated with that location, first the obstacle distance has to be propagated.

`RemoveObstacle`(*position_t* pos) Removes a location (specified either in global coordinates or with grid indices) from the set of obstacles. As for `AddObstacle`(), the obstacle distance has to be propagated before this change will start influencing the path cost map (navigation function).

`SetRisk`(*position_t* pos, *double* risk) Override a cell's relative risk (the one calculated from obstacle distance). This is useful for defining high-risk zones that should be avoided, or lower the risk in special places. As opposed to the other two ways of influencing the environment model, changing the relative risk of a location directly triggers dynamic replanning events.

The risk map translates risks $\in [0, 1]$ to meta information F_i. `GridPlanner` provides the distance to the closest obstacle for each cell, which is provided to the `RiskMap` to yield a value between zero and 1, and then translated to a meta value using the `Interpolation` object. This two-stage process provides flexibility with respect to (i) the form of the buffer zone and (ii) the interpolation kernel.

Note that just like the path cost information (i.e. the crossing time map), the obstacle distance is maintained using the E* mechanism. This is achieved by using obstacle locations as "goals" in an otherwise empty environment, the resulting crossing times are equivalent to the distance to the closest obstacle cell.

4.8.2 Illustrated Operation of `GridPlanner`

The four figures shown in this section are simulated sequences that were constructed to demonstrate the performance of E* when used as a full-fledged global planner. The setups are both run with two different kernels: LSM interpolation has proven the most robust and precise method, and the NF1 kernel provides a comparison with the performance that can be expected from D*. In the first setup (figures 4.18 and 4.19), the robot has to navigate through an unknown maze. The second setup (figures 4.20 and 4.21) demonstrates the effect of removing an existing obstacle, which can lead to topological changes in the optimal path direction.

4.9 Conclusion and Outlook

The E* algorithm, so named because it is based on events and can be made to mimic D*, allows to calculate and update smooth navigation functions that approximate true distance much better than other grid or graph based methods – by an order of magnitude or more given the right initial conditions. The additional computational complexity required for this achievement is a factor of two in the theoretical worst case (for first-order kernels), but experiments suggest a factor of approximately 1.2 to 1.6 in practice. Each of the propagation steps becomes more elaborate as well, because a robust interpolation kernel requires more calculations than the non-interpolating case. This amounts to a factor below 1.4 in the case of the robust LSM interpolation when compared with the NF1 kernel that mimics D*.

In addition to providing an interpolated navigation functions with dynamic replanning, E* is independent of interpolation details and can thus be used to evaluate different kernels in terms of their quality and computational costs.

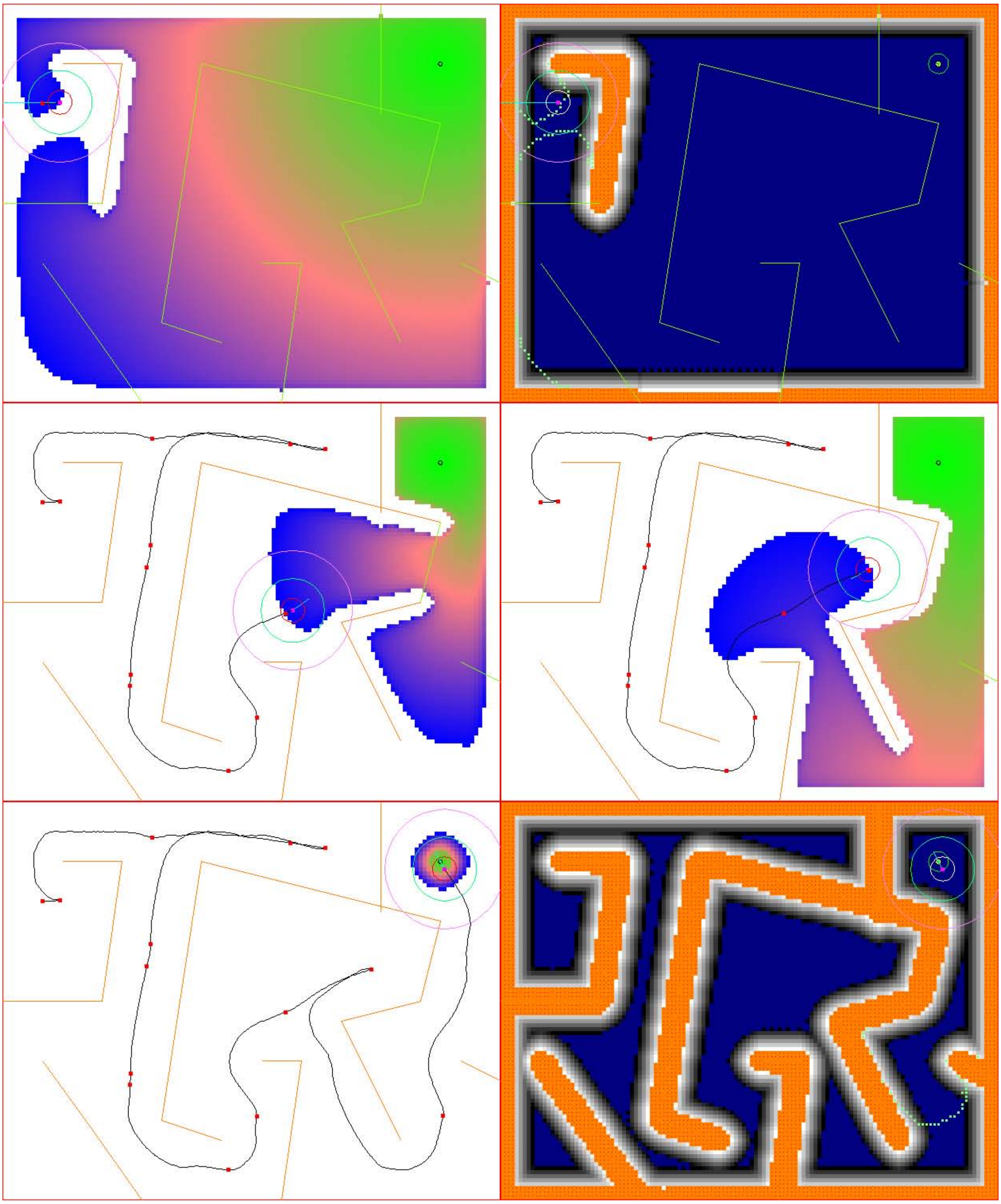


Figure 4.18: Maze exploration with LSM. Shortly after the beginning (top row), the navigation function (left) and risk map (right) reflect the lack of information about the majority of obstacles (orange cells are $\mathcal{C}$ obstacles, dark blue ones are free). The second row illustrates a topology change due to the discovery of a wall across a passage previously expected to be free. The bottom row shows the complete path from start to goal (left) and the final environmental model (right).

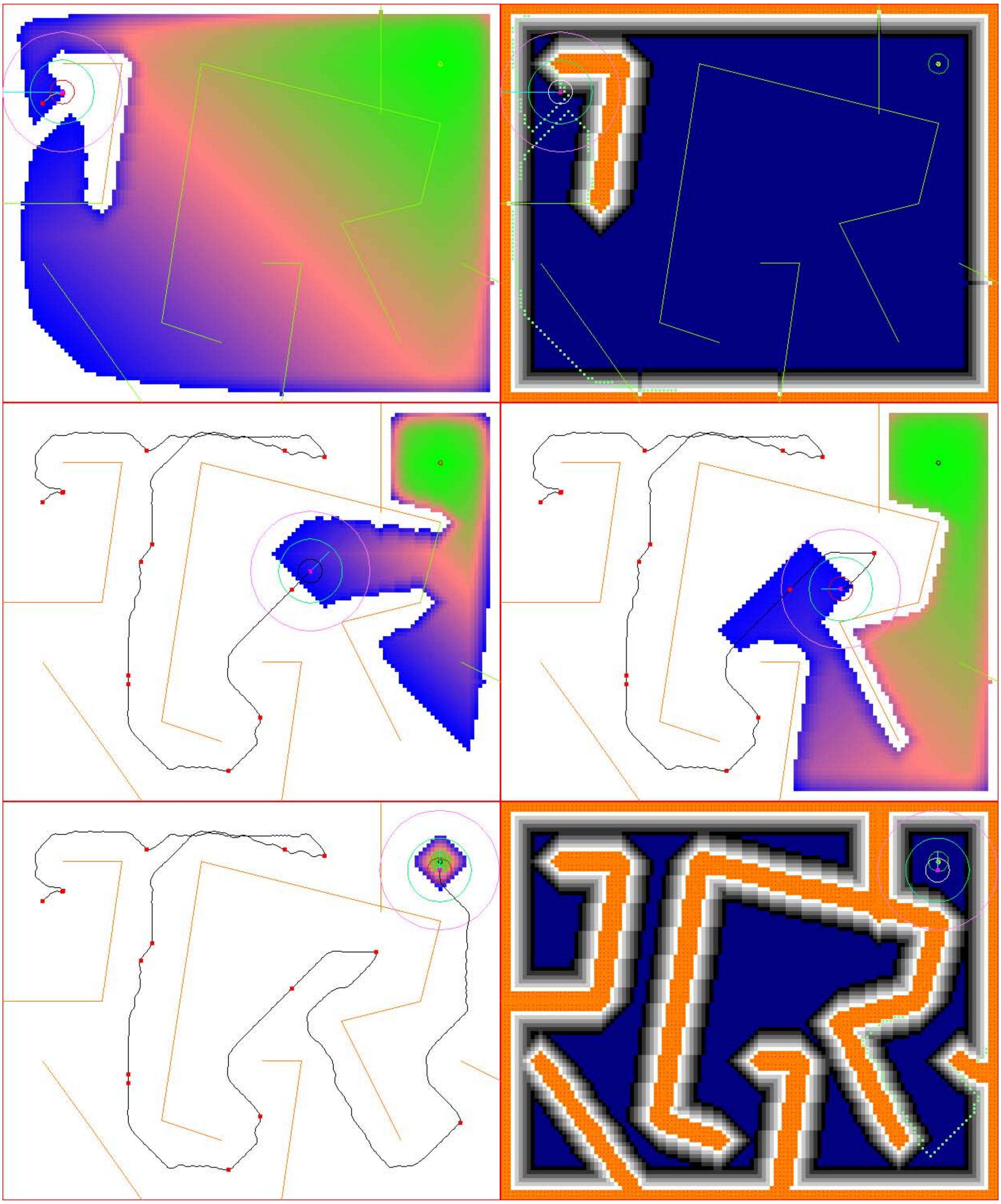


Figure 4.19: Maze exploration with NF1, the sequence of images is the same as in figure 4.18. Note the grid discretization effects that are clearly visible in the navigation function and risk map. This leads to an unsmooth and longer path.

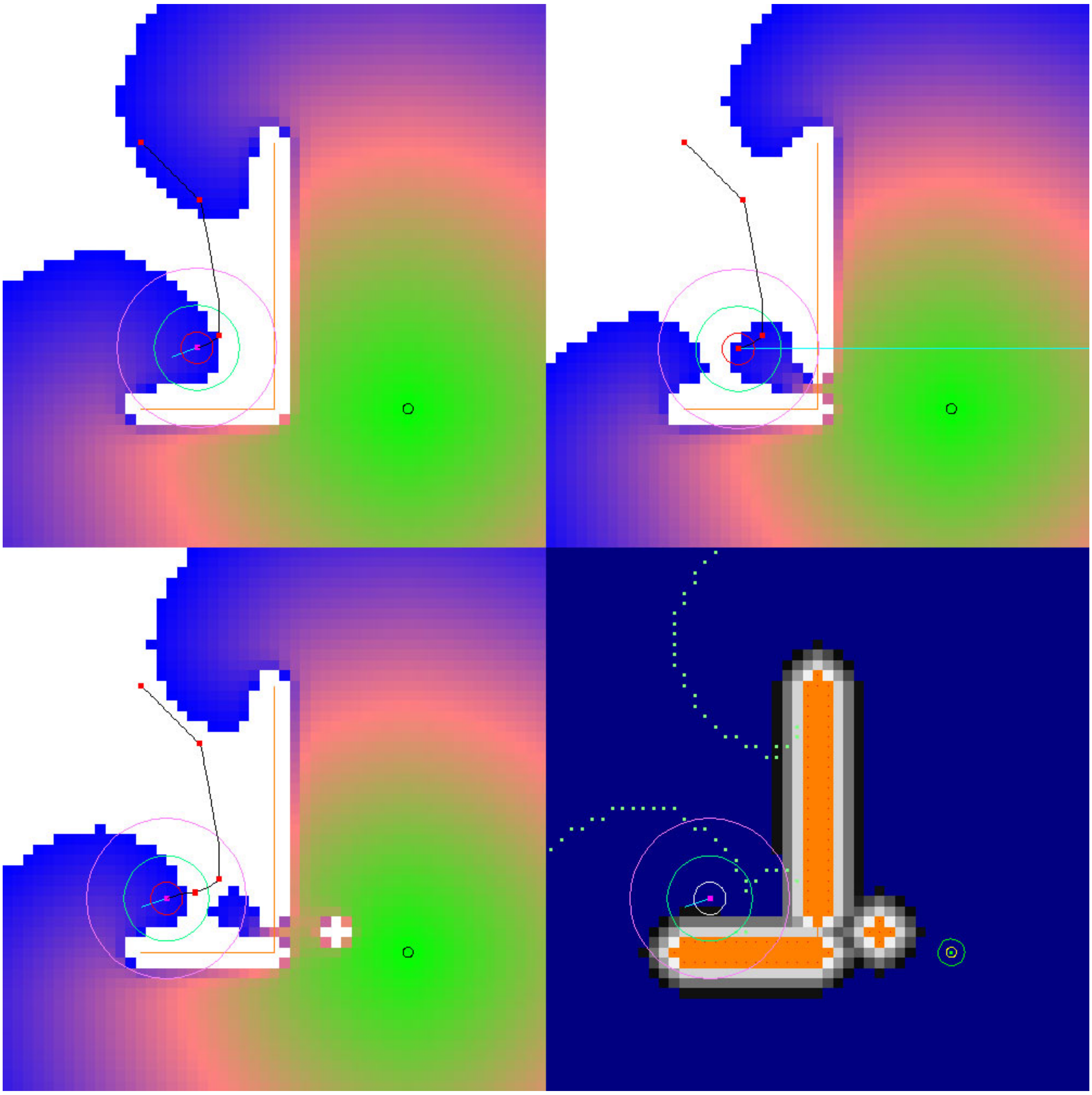


Figure 4.20: Obstacle removal with LSM: After the robot has discovered the L-shaped obstacle, it plans a path around it (top left). The top right shows the navigation function after freeing a passage in the obstacle, the optimal path switches topology by going through this opening. Then, an obstacle was added behind the passage, but such that the robot could pass (bottom right shows the risk map). However, the optimal path switches back to the old topology, because the new obstacles causes a higher accumulated risk: The robot takes a detour to avoid the dangerous zone (bottom left).

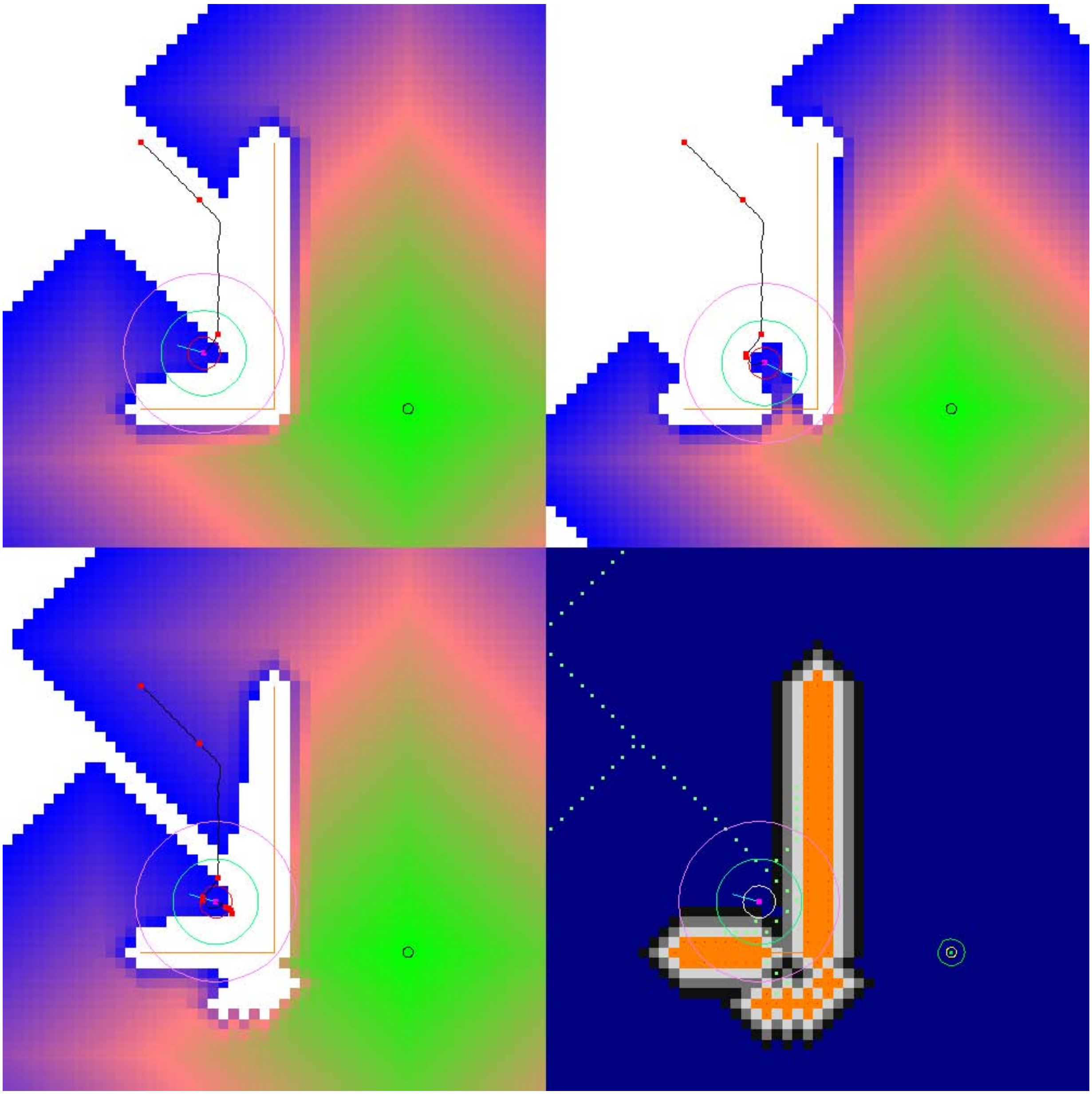


Figure 4.21: Obstacle removal with NF1, the sequence of images is the same as in figure 4.20. Again, note the grid effects, and also that more obstacles were needed to make the robot switch back to the original topology (bottom row). This indicates that the `RiskMap` and `Interpolation` *combined* determine the amount of risk required for accepting a detour.

Generalizing E* to higher dimensions and interpolation orders is relatively straightforward: Event propagation relies on neighborhood information which can be defined independent of the dimensionality. Extending to non-grid representation is feasible as well: The only part of this chapter which really depends on the grid nature is the LSM interpolation, and this has already been applied to triangulated domains[11] in [27]. With E* extended to graphs, it will be more readily applicable to the weighted regions path planning problem [37, 42].

---

[11] Triangulated domains are used as mesh representation of surfaces embedded in higher dimensional space. A graph representation of a robot's environment can be described as a triangulated domain if the regions created by its edges are all triangles.

Chapter 5

Conclusion

> *If I have been able to see further, it was only because I stood on the shoulders of giants.*
>
> Isaac Newton in a letter to Robert Hooke, 1675

This thesis contains two new contributions to the field of mobile robot path planning and obstacle avoidance: An application-oriented robust motion generation system (described in chapter 3) and a novel framework for interpolated navigation functions with dynamic replanning (presented in chapter 4). All software developed for this thesis is open source and released under the GNU General Public License[1]. This facilitates exchange for research, deployment for applications, and ensures that the system remains available.

During the five months of the Robotics@Expo.02 event it has been shown that path planning and obstacle avoidance in dynamic cluttered environments can be done in an effective way. Furthermore, the resulting robot movement is convincing for individuals without technical background, a requirement that has been likened to hard AI problems in the introduction, due to the largely subconscious nature of human movement. The stringent safety requirements for such a public event have been met and the system presented in chapter 3 is mature. Its most important tunable parameters are summarized and explained in table 3.2 in order to ease its adaption to future applications.

The E* framework presented in chapter 4 addresses the need for smooth navigation functions in a flexible way. It builds on the stable foundations of the D* algorithm and (Fast Marching) Level Set Methods and defines a generic approach to dynamic replanning. It produces a very good approximation to the topologically correct Euclidean distance from any point in the environment to the goal. It is flexible with respect to the interpolation kernel and incorporates a continuous notion of risk which is important for practical motion planning because it allows defining buffer zones around obstacles or, more generally, regions of varying traversability. In its current implementation, E* applies to regular grids. However, most of the underlying theory is based on the more general graph structure, and an extension to graphs exists for the Fast Marching Method. It can thus be anticipated that

---

[1] `http://www.gnu.org/`

E^* will be endowed with exact cell-decomposition, the missing prerequisite for complete planning with this framework.

The contributions of this work share an important trait: Formulating novel combinations of existing methods in order to fulfill objectives that are partially addressed by the composing methods, making the whole become more than the sum of its parts. The success that has been met with this approach illustrates the growing maturity of mobile robot path planning and obstacle avoidance, a technology that is poised to greatly influence the lifestyle in our societies.

Bibliography

[1] R. Alami, T. Siméon, and K. Madhava Krishna. On the influence of sensor capacities and environment dynamics onto collision-free motion plans. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[2] Kai Oliver Arras. *Feature-Based Robot Navigation in Known and Unknown Environments*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2003.

[3] K.O. Arras, J. Persson, N. Tomatis, and R. Siegwart. Real-time obstacle avoidance for polygonal robots with a reduced dynamic window. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[4] Enrique J. Bernabeu, Josep Tornero, and Masayoshi Tomizuka. Collision prediction and avoidance amidst moving objects for trajectory planning applications. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[5] J. Borenstein and L. Feng. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(5):869–80, 1996.

[6] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–88, June 1991.

[7] Michael S. Branicky and Ravi Hebbar. Fast Marching for Hybrid Control. In *Proceedings of the IEEE International Symposium on Computer Aided Control System Design*, 1999.

[8] R. Brega, N. Tomatis, and K.O. Arras. The need for autonomy and real-time in mobile robotics: A case study of XO/2 and pygmalion. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2000.

[9] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[10] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[11] Howie Choset and Joel Burdick. Sensor-based exploration: The hierarchical generalized voronoi graph. *International Journal of Robotics Research*, 19(2):96–125, 2000.

[12] Howie Choset, Sean Walker, Kunnayut Eiamsa-Ard, and Joel Burdick. Sensor-based exploration: Construction of the the hierarchical generalized voronoi graph. *International Journal of Robotics Research*, 19(2):126–48, 2000.

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

[14] Hans Jacob S. Feder and Jean-Jacques E. Slotine. Real-time path planning using harmonic potentials in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1997.

[15] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using the relative velocity paradigm. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1993.

[16] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 2000.

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

[18] Thierry Fraichard. Trajectory planning in a dynamic workspace: a 'state-time space' approach. *Advanced Robotics*, 13(1):75–94, 1999.

[19] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[20] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, 21(3):233–255, 2002.

[21] B. Jensen, G. Froidevaux, X. Greppin, A. Lorotte, L. Mayor, M. Meisser, G. Ramel, and R. Siegwart. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[22] B. Jensen, R. Philippsen, and R. Siegwart. Motion detection and path planning in dynamic environments. In *Workshop Proceedings Reasoning with Uncertainty in Robotics, International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[23] Björn Jensen. *Motion Tracking for Human-Robot Interaction.* PhD thesis, École Polytechnique Fédérale de Lausanne, 2004.

[24] M. Khatib, H. Jaouni, R. Chatila, and JP. Laumond. Dynamic path modification for car-like nonholonomic mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1997.

[25] Maher Khatib and Raja Chatila. An extended potential field approach for mobile robot sensor-based motions. In *Proceedings of the International Conference on Intelligent Autonomous Systems (IAS)*, 1995.

[26] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1), 1986.

[27] R. Kimmel and J.A. Sethian. Computing geodesic paths on manifolds. *Proc. Natl. Acad. Sci. USA*, 95(15):8431–8435, July 1998.

[28] Nak Yong Ko and Reid G. Simmons. The lane-curvature method for local obstacle avoidance. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1998.

[29] Kurt Konolige. A gradient method for realtime robot control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2000.

[30] Frederic Large, Sepanta Sekhavat, Zvi Shiller, and Christian Laugier. Towards real-time global motion planning in a dynamic environment using the NLVO concept. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[31] J.-C. Latombe. *Robot motion planning.* Kluwer Academic Publishers, Dordrecht, Netherlands, 1991.

[32] Steven M. LaValle and James J. Kuffner Jr. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, May 2001.

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

[34] J. Minguez and L. Montano. Nearness diagram navigation (ND): A new real time collision avoidance approach. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2000.

[35] J. Minguez, L. Montano, T. Simeon, and R. Alami. Global nearness diagramm navigation (GND). In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[36] Javier Minguez, Luis Montano, and Oussama Khatib. Reactive collision avoidance for navigation with dynamic constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[37] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: Finding shortest paths trhough a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, 1991.

[38] Jan Persson. Obstacle avoidance for mobile robotics. In *Diploma thesis, Dept. of Electrical Eng., Linköpings University, Sweden*, 2000.

[39] R. Philippsen and R. Siegwart. Smooth and efficient obstacle avoidance for a tour guide robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[40] Sean Quinlan. *Real-Time Modification of Collision-Free Paths.* PhD thesis, Computer Science Department, Stanford University, 1994.

[41] Sean Quinlan and Oussama Khatib. Elastic bands: connecting path planning and control. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1993.

[42] N. C. Rowe and R. S. Alexander. Finding optimal-path maps for path planning across weighted regions. *International Journal of Robotics Research*, 19(2):83–95, 2000.

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

[44] S. Sekhavat and M. Chyba. Nonholonomic deformation of a potential field for motion planing. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[45] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Applied Mathematics, Proc. Natl. Acad. Sci. USA*, (93):1591–1595, February 1996.

[46] J. A. Sethian. Evolution, implementation, and application of level set and fast marching methods for advancing fronts. *Journal of Computational Physics*, (169):503–555, 2001.

[47] J.A. Sethian. *Level Set Methods – Evolving interfaces in geometry, fluid mechanics, computer vision, and materials science.* Cambridge University Press, 1996.

[48] R. Siegwart, K.O. Arras, B. Jensen, R. Philippsen, and N. Tomatis. Design, implementation and exploitation of a new fully autonomous tour guide robot. In *Proceedings of the 1st International Workshop on Advances in Service Robotics, ASER 2003*, 2003.

[49] Roland Siegwart, Kai O. Arras, Samir Bouabdallah, Daniel Burnier, Gilles Froidevaux, Xavier Greppin, Björn Jensen, Antoine Lorotte, Laetitia Mayor, Mathieu Meisser, Roland Philippsen, Ralph Piguet, Guy Ramel, Gregoire Terrien, and Nicola Tomatis. Robox at Expo.02: A large-scale installation of personal robots. *Robotics and Autonomous Systems*, 42:203–222, 2003.

[50] Roland Siegwart and Illah Nourbakhsh. *Introduction to Autonomous Mobile Robots*. A Bradford Book, The MIT Press, Cambridge, Massachusetts and London, England, 2004.

[51] Reid Simmons. The curvature-velocity method for local obstacle avoidance. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1996.

[52] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1994.

[53] Anthony Stentz. The focussed D^* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[54] Matthias Strobel. Navigation in partially unknown, narrow, cluttered space. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[55] N. Tomatis, G. Terrien, R. Piguet, D. Burnier, S. Bouabdallah, K.O. Arras, and R. Siegwart. Designing a secure and robust mobile interacting robot for the long term. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[56] John N. Tsitsiklis. Efficient Algorithms for Globally Optimal Trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, September 1995.

[57] A. M. Turing. Computing machinery and intelligence. *Mind*, (59), 1950.

[58] Iwan Ulrich and Johann Borenstein. VFH+: Reliable obstacle avoidance for fast mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1998.

[59] Iwan Ulrich and Johann Borenstein. VFH*: Local obstacle avoidance with look-ahead verification. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

Appendix A

`MotionPlanner` Implementation Details

A.1 A Numerically Stable Quadratic Equation Solver

A numerically stable method for solving the quadratic equation $\alpha T^2 + \beta T + \gamma = 0$ is developed below. The general solution is (A.1). Special cases arise for $\alpha, \beta, \gamma \approx 0$ and combinations thereof, by comparing their absolute values with $\epsilon \ll 1$ considered equivalent to zero. During Robotics@EXPO.02, we used $\epsilon = 10^{-9}$.

$$T_{1,2} = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha} \tag{A.1}$$

$$(|\alpha| < \epsilon) \rightsquigarrow \beta T + \gamma = 0$$

$$\rightsquigarrow \begin{cases} T = -\gamma/\beta & \Leftarrow |\beta| \geq \epsilon \\ T \in \mathbb{R} & \Leftarrow (|\beta| < \epsilon) \cap (|\gamma| < \epsilon) \\ \text{no solution} & \text{otherwise} \end{cases} \tag{A.2}$$

$$(|\alpha| \geq \epsilon) \cap (|\beta| < \epsilon) \rightsquigarrow \alpha T^2 + \gamma = 0 \Leftrightarrow T^2 = -\gamma/\alpha$$

$$\rightsquigarrow \begin{cases} T = 0 & \Leftarrow |\gamma/\alpha| < \epsilon^2 \\ T_{1,2} = \pm\sqrt{-\gamma/\alpha} & \Leftarrow (|\gamma/\alpha| \geq \epsilon^2) \cap (-\gamma/\alpha > 0) \\ \text{no solution} & \text{otherwise} \end{cases} \tag{A.3}$$

$$(|\alpha| \geq \epsilon) \cap (|\beta| \geq \epsilon) \cap (|\gamma| < \epsilon) \rightsquigarrow \alpha T^2 + \beta T = 0$$

$$\Rightarrow \begin{cases} T_1 & = 0 \\ T_2 & = -\beta/\alpha \end{cases} \tag{A.4}$$

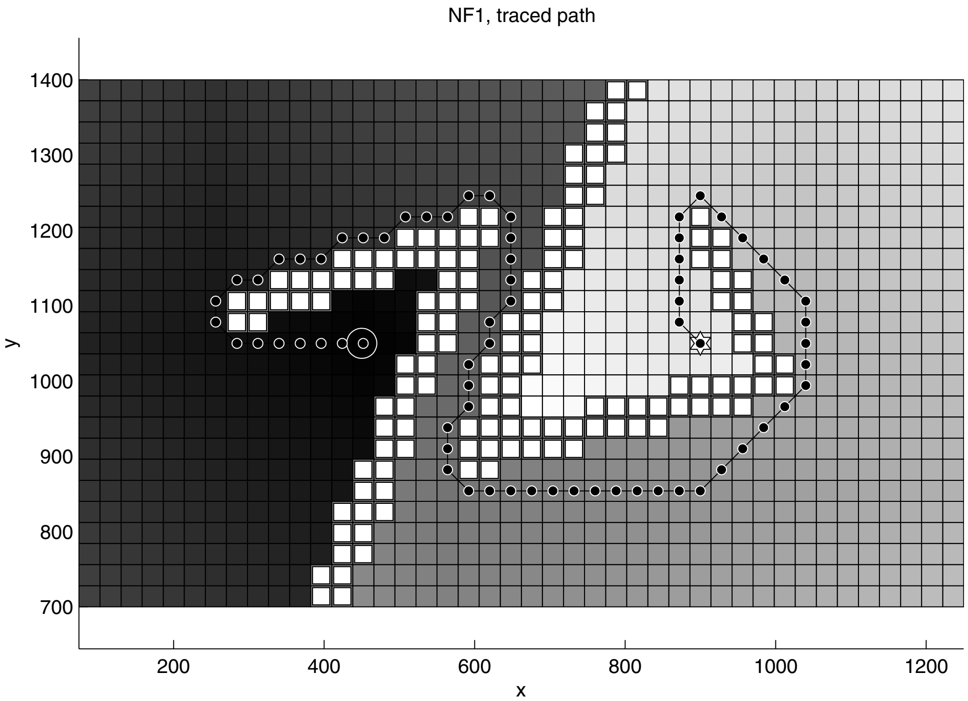


Figure A.1: Example of path plan generated using NF1. Grid values are represented as shades of gray (0 = black), the white squares inside cells are obstacles. The robot starts at the position indicated by the star. Note the two typical properties of paths planned using NF1: It grazes obstacles and consists of straight segments joined by angles that are multiplies of $\pi/4$.

$$(|\alpha| \geq \epsilon) \cap (|\beta| \geq \epsilon) \cap (|\gamma| \geq \epsilon)$$
$$\rightsquigarrow \begin{cases} \text{no solution} & \Leftarrow \beta^2 - 4\alpha\gamma < 0 \\ T = \frac{-\beta}{2\alpha} & \Leftarrow 0 \geq \beta^2 - 4\alpha\gamma < \epsilon^2 \\ T_{1,2} = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha} & \text{otherwise} \end{cases} \quad (A.5)$$

A.2 An Implementation of NF1

The NF1 navigation function is a relatively simple algorithm that constructs a discrete potential with a unique minimum at the goal location. Any location with a path to the goal gets assigned a natural number, and following the steepest negative gradient (i.e. choosing the neighbor of lowest height as intermediate goal) is guaranteed to lead the robot to the goal. Locations from which the goal is not reachable are easily recognized as well, as the special value that signifies unvisited cells is never changed.

Listing A.1 NF1 C++ code

```
void NF1::Calculate(){
  grid.Fill(-2);                              // -2 denotes unlabeled cells
  for(int i(0); i < n_scan_points; ++i)
    grid.SetDisk(scan_point[i], radius, -1);  // -1 denotes occupied cells
  for(int i(0); i < n_goal_points; ++i)
    grid.SetValue(goal_point[i], 0);          // 0 denotes goal cells

  bool finished(false);                       // denotes end of iterations
  int current_label(0);                       // wavefront label
  int next_label(1);                          // wavefront for next iteration

  while( ! finished){
    finished = true;
    for(int ix(0); ix < dimx; ++ix)
      for(int iy(0); iy < dimy; ++iy){
        // skip cells that are not on the wavefront
        if(grid.Value(ix, iy) != current_label)
          continue;

        // check for grid boundaries and update unlabeled neighbors
        if((ix > 0) && (grid.Value(ix-1, iy) == -2)){
          grid.SetValue(ix-1, iy, next_label);
          finished = false;
        }
        if((ix < dimx-1) && (grid.Value(ix+1, iy) == -2)){
          grid.SetValue(ix+1, iy, next_label);
          finished = false;
        }
        if((iy > 0) && (grid.Value(ix, iy-1) == -2)){
          grid.SetValue(ix, iy-1, next_label);
          finished = false;
        }
        if((iy < dimy-1) && (grid.Value(ix, iy+1) == -2)){
          grid.SetValue(ix, iy+1, next_label);
          finished = false;
         }
      }
    ++current_label;
    ++next_label;
  }
}
```

Listing A.1 is an implementation in C++. Figure A.1 shows an example run. When comparing NF1 with potential field methods, the following aspects are important:

Model or sensor based: When equipped with sufficiently dense and accurate sensors, it is possible to implement the potential field method without model: Directly convert range readings into potentials, according to the idea that "the world is it's own best model". This is not possible with the NF1, because all known obstacles have to be added to the grid (the environment model) prior to running the propagation loop.

Global or local: NF1 is a method to find a globally valid path, taking into account the collected obstacle information and usually traversing every cell in the grid, even the ones far from the actual robot position. Potential field methods are local in the sense that each obstacle's influence is calculated only at the robot's position. This difference enables NF1 to avoid local minima, but at the same time requires some kind of model to base that global calculation on.

Smoothness: Taken in it's raw form, a path planned using the NF1 is not smooth (figure A.1). Potential field methods on the other hand tend to produce smooth trajectories (if their parameters are correctly tuned to the environment, the robot's sensors, and its actuators).

A.3 A Variant of the Lloyd-Max Quantizer

This implementation of the Lloyd-Max quantizer assumes all values respect $0 \leq v \leq T_{\max}$, and it uses the minimum value of each bin for representing all values within that bin (instead of the mean value used in the original formulation). Once the buffer has been filled with the values (i.e. collision predictions of DWA), the methods `CreateHistogram()` (listing A.2), `LloydMax()` (listing A.3), and `Quantize()` (listing A.4) are called in that order to produce the compressed collision lookup for a particular cell of the local obstacle grid.

Listing A.2 Creating the histogram for Lloyd-Max quantizing.

```
void Lookup::
CreateHistogram()
{
  nBins = maxNbins;

  // load buffer into histogram
  int count(0);
  for(int i = 0; i < dimension; ++i)
    for(int j = 0; j < dimension; ++j){
      histogram[count] = buffer[i][j];
      ++count;
    }

  // sort histogram (insert sort, simple but slow)
  for(int j = 1; j < dimension * dimension; ++j){
    double key(histogram[j]);
    int i(j - 1);
    while((i >= 0) && (histogram[i] > key)){
      histogram[i + 1] = histogram[i];
      --i;
    }
    histogram[i + 1] = key;
  }
}
```

Listing A.3 C++ implementation of the Lloyd-Max quantizer

```
void Lookup::
LloydMax()
{
  // initialize bins
  double vmax;
  bool found(false);
  for(int i = 0; i < dimension * dimension; ++i)
    if(histogram[i] <= maxValid){ found = true; vmax = histogram[i]; }
    else break;
  if( ! found){ noValid = true; return; }
  noValid = false;

  double vmin;
  for(int i = 0; i < dimension * dimension; ++i)
    if(histogram[i] >= minValid){ vmin = histogram[i]; break; }
  double scale((vmax - vmin) / maxNbins);
  for(int i = 0; i < maxNbins; ++i){
    bin[i].t = vmin + i * scale;
    bin[i].r = -1;
  }
  bin[maxNbins].t = vmax;

  // Main loop
  nBins = maxNbins;
  bool finished(false);
  while( ! finished){
    finished = true;

    double r2;
    for(int i = 0; i < nBins - 1; ++i){
      r2 = PartialMean(bin[i].t, bin[i + 1].t);
      if(absval(bin[i].r - r2) > 1e-9) finished = false;
      bin[i].r = r2;
    }
    r2 = PartialMeanInclusive(bin[nBins - 1].t, bin[nBins].t);
    if(absval(bin[nBins - 1].r - r2) > 1e-9) finished = false;
    bin[nBins - 1].r = r2;

    // remove empty bins
    for(int i = 0; i < nBins; ++i)
      if(bin[i].r < 0){
        --nBins;
        for(int j = i; j < nBins; ++j){
          bin[j].t = bin[j + 1].t;
          bin[j].r = bin[j + 1].r;
        }
        bin[nBins].t = bin[nBins + 1].t; // one more boundary than levels
        --i;
      }

    // calculate boundaries
    for(int i = 1; i < nBins; ++i)
      bin[i].t = 0.5 * (bin[i - 1].r + bin[i].r);
  } //   while(!finished)
}
```

Listing A.4 Replacing *double* values by quantizer indices.

```
void Lookup::
Quantize()
{
  if(noValid)
    return;

  if(quantizer != 0)
    delete[] quantizer;
  quantizer = new double[nBins + 1]; // one special bin for "no intersection"

  // init quantizer values
  for(int i = 0; i < nBins - 1; ++i)
    quantizer[i] = PartialMin(bin[i].t, bin[i + 1].t);
  quantizer[nBins - 1] = PartialMinInclusive(bin[nBins - 1].t, bin[nBins].t);
  quantizer[nBins] = -1;

  // quantize buffer and store it in value[][]
  for(int i = 0; i < dimension; ++i){
    for(int j = 0; j < dimension; ++j){
      double val(buffer[i][j]);
      int k;
      for(k = 0; k < nBins; ++k)
        if((val >= bin[k].t) && (val < bin[k + 1].t)){
          value[i][j] = k;
          break;
        }
      if(k == nBins){
        if(val == bin[nBins].t){ // last bin is "right-inclusive"
          value[i][j] = nBins - 1;
        }
        else{
          value[i][j] = nBins; // no intersection (or too far)
        }
      }
    }
  }
}
```

Appendix B

E* Implementation Details

B.1 Event Propagation

Listing B.1 shows the propagation of lower events implemented in C++. The target cell is stored in `_to`, and the `interpolation` is used to calculate the best (i.e. lowest) value from the cell's neighbors[1]. If it is an improvement, the cell and its backpointers are updated accordingly. Then, the `front` is used to lower all neighbors with values above the one just updated.

Listing B.2 shows the implementation of raise event propagation in C++. The `front` is used to enqueue `RaiseEvent`s to all neighbors with a backpointer to the current cell, and to schedule a future `RetryEvent` which will attempt to lower this cell once the wake of raise events has moved on. The cell's value is set to infinity and its backpointers are removed to ensure that no neighbor will use this cell for path cost calculations.

Listing B.1 `LowerEvent` propagation algorithm.

```
void LowerEvent::
Propagate(Wavefront & front, Interpolation & interpolation)
{
  Interpolation::lowest_t lowest(interpolation.CalculateBest(_to));
  if(_to->Value() < lowest.value)
    return;

  _to->SetValue(lowest.value);
  _to->SetPrimary(lowest.primary);
  _to->SetSecondary(lowest.secondary);

  front.InitLower(lowest.value, _to);
}
```

---

[1] `lower_t` is a type that contains the value and backpointers that indicate which cells have been used to calculate the value.

Listing B.2 `RaiseEvent` propagation algorithm.

```
void RaiseEvent::
Propagate(Wavefront & front, Interpolation & interpolation)
{
  front.InitRaise(_to->Value(), _to);
  front.Retry(_to->Value() + front.Scale(), _to);
  _to->SetValue(front.Infinity());
  _to->SetPrimary(0);
  _to->SetSecondary(0);
}
```

Listing B.3 C++ implementation of the NF1 kernel presented in section 4.6.1

```
Interpolation::lowest_t NF1Interpolation::
Calculate(Cell * primary, Cell * secondary, Cell * to)
{
  if(to->Meta() == ObstacleMeta())
    return lowest_t(_infinity, 0, 0);

  double primval(primary->Value());
  if(primval >= _infinity)
    return lowest_t(_infinity, 0, 0);

  return lowest_t(primval + _scale + to->Meta(),
                  primary, 0);
}
```

B.2 Interpolation Kernels

Listing B.3 shows the implementation of graph-distance based "interpolation". It checks for for propagations to or from obstacles before performing the actual calculations of equation (4.8).

The kernel based on Huygens' Principle is given in listing B.4. `vec2d` is a utility class with two-dimensional vector semantics. At the beginning, attempts to propagate from or to obstacles (and un-visited cells) are caught. Then cell positions are expressed in the global and propagating coordinate frames. The check for `dist < epsilon` is to avoid using coincident backpointers. Then comes the calculation of the wave source $\boldsymbol{s}$ and the tests for the inclusion of $\boldsymbol{c}_0$ in the region of valid interpolation.

Listing B.5 shows the implementation in C++ of the update equation (4.20). Parameter `primary` refers to node **A** of figure 4.10, `secondary` is node **C**, and `to` corresponds to the center node. Obstacles are handled specially, because the solution breaks down for zero propagation speeds. The second check is to ensure that obstacles and unvisited cells are not used for updates. The third check is for the fallback solution, and finally the calculation of the non-degenerate solution.

Note that `root` is never negative because previous checks ensure that the solution lies

Listing B.4 C++ implementation of the HPR kernel presented in section 4.6.2

```
Interpolation::lowest_t HPRInterpolation::
Calculate(Cell * primary, Cell * secondary, Cell * to)
{
  const double F0(to->Meta());
  if(F0 < epsilon)
    return lowest_t(_infinity, 0, 0);
  const double T1(primary->Value());
  if(T1 >= _infinity)
    return lowest_t(_infinity, 0, 0);
  const double T2(secondary->Value());
  if(T2 >= _infinity)
    return CalculateFallback(primary, to);

  const vec2d c1G(_grid.LocalPoint(primary->Ix(),   primary->Iy()));
  const vec2d c2G(_grid.LocalPoint(secondary->Ix(), secondary->Iy()));
  const vec2d delta21G(c2G - c1G);
  const double dist(delta21G.norm());
  if(dist < epsilon)
    return CalculateFallback(primary, to);

  const vec2d echi(delta21G / dist);
  const vec2d egamma(echi.x, - echi.y);
  const vec2d c0G(_grid.LocalPoint(to->Ix(), to->Iy()));
  const vec2d delta01G(c0G - c1G);
  const vec2d c0P(vec2d::inner(delta01G, echi),
                  vec2d::inner(delta01G, egamma));

  const double r1(T1 / F0);
  const double r2(T2 / F0);
  if(r1 + r2 < dist)
    return CalculateFallback(primary, to);

  vec2d sP((square(dist) + square(r1) - square(r2)) / 2 / dist, 0);
  sP.y = sqrt( square(r1) - square(sP.x) );
  if(c0P.y > 0)
    sP.y = - sP.y;

  const vec2d ds0(c0P - sP);
  const vec2d ds1(    - sP);
  const vec2d ds2(dist - sP.x, - sP.y);
  const double signcheck(vec2d::outer(ds0, ds1) * vec2d::outer(ds2, ds0));
  if(signcheck < 0)
    return CalculateFallback(primary, to);

  const double solution(ds0.norm() * F0);
  if(solution < T2)
    return CalculateFallback(primary, to);

  return lowest_t(solution, primary, secondary);
}
```

Listing B.5 Computations for the LSM update equation (4.20).

```
Interpolation::lowest_t LSMInterpolation::
Calculate(Cell * primary, Cell * secondary, Cell * to)
{
  if(to->Meta() == ObstacleMeta())
    return lowest_t(_infinity, 0, 0);

  const double primval(primary->Value());
  if(primval >= _infinity)
    return lowest_t(_infinity, 0, 0);

  const double secval(secondary->Value());
  const double rad(_grid.Scale() / to->Meta());
  if(rad <= secval - primval)
    return lowest_t(primary->Value() + rad,
                    primary, 0);

  const double b(primval + secval);
  const double c((  square(primval)
                  + square(secval)
                  - square(rad)    ) / 2);
  const double root(square(b) - 4 * c);

  return lowest_t((b + sqrt(root)) / 2,
                  primary, secondary);
}
```

on the circular section of the curve shown in figure 4.11, so there is no risk of floating point exception when calling `sqrt(root)`.

Curriculum Vitae

Roland Philippsen was born to German parents on May 6th 1976 in Stanford, USA. He has an older brother and a younger sister. Starting in 1983, he went to school in Switzerland, Germany, and the USA, until 1995 when he obtained a Scientific High School Diploma (Maturität Typus C) at the Gymnasium Bäumlihof in Basel, Switzerland. In 2000, he received a M. Sc. Microengineering EPFL (Ing. Microtechnique Dipl. EPFL) at Ecole Polytechnique Fédérale de Lausanne, Switzerland (EPFL – Swiss Federal Institute of Technology). The master thesis "Investigation of the Automated DNA Sequencing System at the Stanford Genome Center " analyzes complex material and information flow through a partially automated genomics installation shared by several research groups. He started work at the Autonomous Systems Lab (ASL) at EPFL shortly afterwards and has been involved in several of its activities, for example setting up the system and drivers for the educational robot Smartease, giving lectures, supervising student projects, and participating in the launch of a conference web site. In 2001, he started work on his doctoral thesis "Motion Planning and Obstacle Avoidance for Mobile Robots in Highly Cluttered Dynamic Environments" at the ASL, with the chance of beginning in the context of a challenging real-world application: The Robotics@Expo.02 event. He passed his PhD exam on November 26, 2004.