HIGH-SPEED NAVIGATION
WITH APPROXIMATE MAPS

RICHARD FROOM
MAY 1995

AI95-232
HIGH-SPEED NAVIGATION WITH APPROXIMATE MAPS

Richard Allan Froom
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712
froom@cs.utexas.edu

May 1, 1995
Abstract

A global map of a mobile robot's environment is essential for high-performance navigation in large-scale space. When portions of the environment are not visible, a map is needed for route planning and enables high performance by allowing the robot to anticipate regions that are occluded or beyond sensor range. Yet, autonomously acquired global map information is inevitably uncertain due to the low positioning accuracy of mobile robots and the possibility of changes to the environment.

Previous work in high-speed navigation falls into two categories. Global optimization approaches assume that an accurate model of environment geometry and robot dynamics are available, and address the problem of efficiently approximating the minimum-time control between a start and goal state. Reactive navigation methods use only immediately sensed environment geometry to avoid obstacles while moving to a specified goal position. The global optimization approach has the theoretical advantage of high performance, but it does not address the significant uncertainty typical of mobile robots. The reactive navigation approach can respond to unanticipated geometry, but its overall performance is limited.

This dissertation describes a method for high-speed map-guided navigation in realistic conditions of uncertainty. A previously-developed method is used to acquire a topologically correct, metrically approximate map of the environment despite positioning errors. Information in the approximate map guides the operation of a novel, high-performance reactive navigator. Performance does not critically depend on the availability of expensive, accurate metric information. Nonetheless, the map may be elaborated with more detailed information, and, as its level of detail and accuracy is improved, performance smoothly improves.
Contents

1 Introduction
   1.1 Exploration and Mapping .......................................................... 6
   1.2 Going Fast: Overview ............................................................... 7
   1.3 Going Fast in Open Space ......................................................... 8
   1.4 Going Fast While Avoiding Obstacles ......................................... 11
   1.5 Using Stored Knowledge of Environment Geometry ....................... 12
   1.6 Improving Performance with Experience ..................................... 14
   1.7 Overview ............................................................................... 16

2 A High-Speed Robot Model
   2.1 Dynamics ............................................................................. 18
   2.2 Sensors ................................................................................. 19
   2.3 Modeling Control and Estimation Error ................................. 20
      2.3.1 Error Magnitude ............................................................. 22
   2.4 Representing Approximate Position Information ...................... 23
      2.4.1 Compounding ................................................................. 24
      2.4.2 Merging ................................................................. 24
      2.4.3 Equiprobability Contour ............................................... 25

3 Going Fast in Open Space
   3.1 A Constraint Allocation Approximation ..................................... 27
   3.2 Travel Time Discontinuities ....................................................... 30
   3.3 Solving for the Acceleration Allocation ..................................... 34
   3.4 Comparison to Optimal Performance ......................................... 35
   3.5 Adding a Velocity Constraint .................................................... 35
   3.6 Uncertainty ............................................................................ 37
      3.6.1 Handling Undershoot in One Dimension .......................... 39
      3.6.2 Handling Undershoot in Two Dimensions ....................... 40
   3.7 Summary .............................................................................. 41
4 Going Fast While Avoiding Obstacles
   4.1 Introduction ................................................. 42
   4.2 Previous Work .................................................. 42
   4.3 High-Speed Reactive Navigation ................................. 46
   4.4 How to Dodge Obstacles ......................................... 46
       4.4.1 Collision Checking ......................................... 47
       4.4.2 Choosing an Edge Point ..................................... 49
       4.4.3 Choosing a Temporary Target: The Approach ............... 51
       4.4.4 What Is a Reasonable Trajectory? .......................... 53
       4.4.5 Is There a Reasonable Trajectory to This Temp Target? .. 54
       4.4.6 Picking a Fast Target Velocity .............................. 56
       4.4.7 When Reasonableness Restrictions Are Not Met .......... 57
       4.4.8 Sensory Horizon Effects .................................... 60
       4.4.9 Rechecking for Collision ................................... 61
       4.4.10 Uncertainty in Control Outcomes ......................... 62
   4.5 Conclusion ...................................................... 62

5 Map-Guided Obstacle Avoidance Control .............................. 65
   5.1 Qualitative Exploration and Mapping ........................... 66
   5.2 Outline of Map-Level Control .................................. 69
   5.3 Summary .......................................................... 70

6 Representing Knowledge of Environment Shape ........................... 73
   6.1 Introduction .................................................... 73
   6.2 Storing Knowledge of Environment Geometry .................... 73
   6.3 Using Stored Knowledge of Environment Geometry for Obstacle Avoidance Control .......................... 76
       6.3.1 Generating the Plan Ogrid ................................ 76
       6.3.2 Using the Plan Ogrid ....................................... 80

7 Improving Global Performance ......................................... 84
   7.1 Introduction .................................................... 84
   7.2 Approach ......................................................... 85
   7.3 Frontier Initialization ......................................... 86
   7.4 Subgoal Improvement Step ...................................... 87
   7.5 Traveling Route with Frontier Update .......................... 89
   7.6 Summary .......................................................... 91

8 Performance Comparisons ............................................. 93
   8.1 Feedforward Control vs. Map-guided Reactive Navigation ....... 94
   8.2 Reactive Navigation vs. Map-Guided Dodging .................... 94
   8.3 Performance with Varied Levels of Map Knowledge ............... 96
9 Conclusion

9.1 Summary .................................................. 105
9.2 Contributions ............................................. 106
9.3 Future Work .............................................. 108

A Appendix A: Compounding and Merging Operations 109
Chapter 1

Introduction

Mobile robots become much more useful in many applications when they are able to navigate at high speeds. When only part of the robot’s environment is immediately visible from a given location, a global environment map is essential for high-performance navigation. Yet, map information that is acquired autonomously is inevitably uncertain due to the low positioning accuracy of mobile robots and the possibility of changes to the environment. Nonetheless, previous work has not addressed the application of uncertain map information to the high-speed navigation task. This dissertation describes a method for acquiring and representing approximate map information, and using it for high-performance navigation.

In order to travel as rapidly as possible, a mobile robot needs a “map,” or representation of its environment. In general, only the nearby portion of the robot’s environment can be immediately sensed (the environment is large-scale) and some knowledge of non-local environment geometry is necessary for efficient navigation, for several reasons. First, the robot needs a map in order to choose a direct route to the goal; otherwise, even if the goal’s relative position were known, it might go down a “blind alley” or take an unnecessarily circuitous route to the goal. Second, a robot traveling at high speeds cannot stop quickly, so in order to avoid collision it must anticipate the environment geometry it will encounter. The knowledge of environment geometry beyond sensor range represented in a map allows the robot to travel faster while guaranteeing non-collision.

However, map acquisition is constrained by the limited positioning accuracy of mobile robots. Mapping a large-scale environment requires combining multiple sensor readings separated by significant distances. But the robot’s position error accumulates as it moves, so the estimated relative positions of points separated by long traversals will be subject to considerable error [Chatila and Laumond, 1985; Brooks, 1985].

Previous work in robot navigation can be usefully classified according to two characteristics. The first is the modeling of robot dynamics. Either dynamics are ignored (appropriate if the robot only moves slowly) and the objective is to find a purely geometric path, or the robot’s dynamics are explicitly modeled and the objective is to find a time sequence of states, or a “trajectory,” that obeys the robot’s dynamic constraints. The second characteristic is
the control style. Control is either generated off-line (using accurate stored map information) or on-line (using current sensor information).

The earliest work in mobile robot path planning treated only the geometric aspect of the problem, and assumed a complete prior map of the environment so that a feedforward solution could be generated [Lozano-Perez and Wesley, 1979; Thorpe, 1984; Yap, 1987]. More-recent work in geometric path generation has addressed the task of a feedback approach to navigation that uses only locally sensed information about environment geometry [Lumelsky and Lewis, 1988; Slack, 1993]. These methods are inappropriate for high-speed navigation since they disregard the robot’s dynamic constraints.

Research addressing the trajectory generation task explicitly addresses the robot’s dynamic constraints. This work can similarly be categorized by application of stored or on-line environment information and control style. At one end of this spectrum are methods that assume completely accurate prior knowledge of large-scale space. Using this knowledge and an accurate model of robot dynamics, an approximately time-optimal feedforward control between two locations can be found using a variety of techniques [Gilbert and Johnson, 1985; Shiller and Chen, 1990; Zaharakis and Guez, 1990; Donald and Xavier, 1989]. These methods are not designed to respond to sensor information during navigation, and hence do not address map errors, control errors, and environment changes. At the other end of this spectrum, methods such as those described by Feng and Krogh [1990] and Borenstein and Koren [1989; 1991] focus on local navigation, and do not make use of a stored map of the environment at all. Instead, the robot navigates to a goal position (expressed in global coordinates) using only currently sensed range information to avoid collision. This eliminates the need for an accurate global map, but since the robot is unable to anticipate environment geometry beyond its sensor range, its performance is restricted. Furthermore, this method is not appropriate for standalone application in a large-scale environment, since without the ability to reason about the environment globally the robot may choose an inefficient route to the goal or become trapped.

To achieve the highest performance possible while dealing with the uncertainty arising from low positioning accuracy and environment changes, a navigation system should be able to use the approximate map information that is available, as well as respond to immediate sensor information. The map resource allows the robot to travel in a direct route to the goal and anticipate what lies ahead to achieve a high level of performance. At the same time the robot can respond to current sensor information, allowing it to adapt to control, estimation, and map errors as well as changes in the environment. This research explores this approach.

The high-speed navigation problem is formulated as follows. A mobile robot is placed in a novel environment. The robot has local range sensing with the ability to detect nearby obstacles and proximity to nearby “distinctive places” in the environment. The robot has access to internally generated odometric information which is subject to error. The robot is controlled by specifying a time-sequence of accelerations, and control outcomes are subject to position error. Acceleration and velocity are each constrained. The robot’s task is to map the environment and travel through it at high speed. The following sections provide an
overview of the solution to this navigation problem.

1.1 Exploration and Mapping

When placed in a new environment, the robot's first task is to explore and map the environment. The low positioning accuracy of mobile robots considerably complicates the mapping of large-scale space. A metrically oriented approach of simply fitting local measurements into a global coordinate frame is vulnerable to positioning error [Brooks, 1985; Chatila and Laumond, 1985; Levitt, 1987]. The problem goes beyond a simple distortion of the map: when the robot returns to a previously mapped region, its position estimate may easily be sufficiently displaced that it appears to be a similarly shaped, but different region.

This type of problem is avoided by using the qualitative mapping method developed by Kuipers and Byun [1988; 1991; Byun, 1990]. Rather than attempting to directly integrate local measurements into a global metric map, this approach begins by characterizing the qualitative layout of the environment. First, the environment is described in terms of concepts arising from the sensorimotor interaction between the robot and its environment: distinctive places and distinctive paths. A distinctive place is defined as the local extremum of a distinctiveness measure defined on a subset of the robot's sensory features, such as "extent of distance differences to near objects;" once in the neighborhood of a distinctive place, the robot can move to it by executing a control strategy which performs hill-climbing in the distinctiveness measure. Travel between distinctive places along distinctive paths is performed by executing local control strategies such as following an obstacle's boundary at some constant distance.

Using an exploration procedure that establishes or disproves the identity of places with similar appearance, places and paths defined by control strategies can be linked into a network that provides a correct topological description of the environment. This topological map enables route-planning and navigation at low speeds with very low dependence on the accuracy of the robot's odometry data.

Once this topologically correct framework is in place, it can easily be augmented with approximate metrical information. In this exploration phase, the approximate relative positions of distinctive places are added to the map. This information will be useful for high-speed navigation as well as for choosing a shortest path to a goal, and is readily available from odometry data. The representation of this approximate position information has two parts: the best estimate of the displacement, and the level of uncertainty in the estimate. The uncertainty level is represented by a covariance matrix, as in the representations developed by Smith and Cheeseman [1986] and Crowley and Ramparany [1987].

Figure 1.1 shows the result of exploring and mapping a simple environment. This is output from the NX simulator described by Kuipers and Byun [1988] and reimplemented by Wan-Yik Lee. Distinctive places are found at each corner of the environment, with distinctive paths connecting them. The topological properties of the environment are described by a
Figure 1.1: The result of a partial exploration of a simple environment. The robot has discovered six distinctive places, labeled P1 through P6. Illustrations in the remainder of this chapter illustrate high-speed navigation of the robot from place P6 back to place P1.

graph whose nodes and edges correspond to distinctive places and paths, respectively. Edges are annotated with approximate measures of their relative positions. The resulting map is a topologically correct and metrically approximate summary description of the environment.

Despite the limited accuracy of the map, it is well-suited to the high-speed navigation task. A particular navigation task is specified by the robot’s initial position and a goal place. Using the graph representation of the environment plus the metrical edge information, a sequence of distinctive places along a short path to the goal place can easily be generated. The high-speed navigator will then direct the robot to each distinctive place in the sequence, again making use of the metrical edge information.

The relative positions of adjacent distinctive places are known more accurately than those of places that are further removed. The knowledge of the position of the next place that the robot is heading toward is most important for high-speed control. Though the robot’s knowledge of the relative positions of distant places may be quite inaccurate, the topological integrity of the overall map is maintained at the topological level, thus supporting the path planning function.

1.2 Going Fast: Overview

Once at least part of the environment has been mapped, the robot can use the map for high-speed travel. The conflict between the demand for high performance and limited infor-
mation creates a challenge in the creation of a high-speed controller. Due to control errors, uncertain knowledge of obstacle layout, and the potential for changes in the environment, the controller must respond to new information that becomes available as it moves through the environment. But generating the minimum-time control for a robot subject to both dynamic and obstacle constraints is a difficult problem. While approximation algorithms for this problem exist, they are quite computationally intensive, and hence inappropriate for frequent recomputation as the robot moves through the environment.

The high-speed control method developed here is based on the observation that minimum-time control in the presence of obstacles can be well approximated by a sequence of easier-to-compute open space minimum-time control segments. For instance, when paths between distinctive places are unobstructed, navigating to the goal can be divided into a sequence of navigations between subgoals placed in the vicinity of distinctive places. Each subgoal is a target state for the robot, consisting of a position and a velocity.

In order to meet the demands of this application, this basic control capability is expanded in several ways. Flexibility is added to the open-space control algorithm by adding an obstacle-avoidance capability that allows the robot to respond to obstacles within sensor range and guarantee non-collision. The robot can travel at higher speeds while maintaining a non-collision guarantee by using stored knowledge of environment geometry. Finally, global performance is improved with experience by modifying the subgoals in the vicinity of each distinctive place so that segments of open-space control are fitted into an approximation to the globally optimal solution.

The following sections describe these parts of the control algorithm:

- Computing an approximation to open-space minimum-time control.
- Obstacle-avoidance control.
- Using stored knowledge of environment geometry beyond sensor range.
- Modifying subgoals.

1.3 Going Fast in Open Space

As mentioned in the previous section, the high-speed control method developed here makes use of the observation that minimum-time control in the presence of obstacles can be approximated by a sequence of open-space minimum-time control segments. For example, suppose that positioning error is low and the sight lines between adjacent distinctive places are clear. By choosing zero-velocity subgoals positioned at distinctive places, the open-space trajectories between adjacent subgoals will follow straight lines, and hence be unobstructed. As shown in Figure 1.2, the open-space minimum-time control algorithm can be applied to travel between adjacent subgoals along the known route from Place 6 at the bottom left of the figure to Place 1 at the bottom right.
Figure 1.2: The robot uses the map of the environment created with low-speed exploration to travel along the route from place P6 (lower left) to place P1 (lower right) using the high-speed control method. The subgoals have zero velocity and are positioned at distinctive places. The robot’s sensor range here is 6.4 meters, its maximum acceleration is 2.5 m/s, and the robot’s position is shown at the beginning of each 0.5 s time step. Total travel time = 38 s.
The open-space control task can be described as follows: given an initial state, a goal state, and a model of the robot's dynamics, generate the control sequence that transfers the robot from the initial state to the goal state in minimum time. The robot's dynamics are modeled as a particle subject to a controllable acceleration:

\[
\begin{bmatrix}
\dot{x} \\
\dot{v}
\end{bmatrix} = \begin{bmatrix}
0 & I \\
0 & 0
\end{bmatrix} \begin{bmatrix}
x \\
v
\end{bmatrix} + \begin{bmatrix}
0 \\
u
\end{bmatrix}
\] (1.1)

where \( I \) is the 2D identity matrix, \( x, v \), and \( u \) are two-dimensional vectors describing the robot's position, velocity, and applied acceleration, respectively. The robot's velocity and control are subject to constraints: \( \| u \| < a_{\text{max}}, | v_1 | < v_{\text{max}}, | v_2 | < v_{\text{max}} \).

The magnitude constraint on applied acceleration means that the control must lie in a circle of radius \( a_{\text{max}} \); hence the acceleration constraint in one dimension is dependent on the level of control in the other dimension. If the control was constrained independently in each dimension, the problem could be decoupled into two one-dimensional problems, one for each dimension. Each of these one-dimensional problems could be solved with the simple, closed-form bang-bang solution [Athans and Falb, 1966]. The magnitude constraint is chosen because it models an omnidirectional robot's dynamic constraints more accurately than independent acceleration constraints. However, finding the exact solution to the problem with an acceleration magnitude constraint requires a computationally expensive procedure.

The simplest way to get an efficient solution would be to artificially constrain the available acceleration to a square inscribed within the circular magnitude constraint as shown in Figure 1.3a. However, this significantly reduces the potential performance of the controller since it does not use all available acceleration. An alternative is to dynamically allocate independent constraints, as illustrated in Figure 1.3b, using the method described in Chapter 4. This results in near-optimal performance while still being efficiently computable.

While this algorithm generates a feedforward solution, in the present application this algorithm will not be directly applied in an open-loop fashion. There is uncertainty in the robot's estimate of the relative position of adjacent distinctive places, and since there is uncertainty associated with each movement, as the robot moves from one distinctive place toward the other, the error will accumulate until the robot gets close enough to "see" the distinctive place. The algorithm should be applied in a closed-loop fashion to make use of new information as it becomes available. This can be done by treating the robot's mean state estimate at the start of a time step as the initial state, computing the open-space control to the current subgoal, applying this control over the time step, and repeating the process over subsequent time steps. Using the mean value of the state may allow the robot to overshoot the target, but this does not present serious problems when incorporated into the overall navigation strategy described in the following sections. Several other small modifications to deal with uncertainty are described in Chapter 4.
Figure 1.3: The acceleration magnitude constraint restricts allowable acceleration to lie within the circular region. By restricting acceleration further so that the acceleration constraints in each dimension are independent, the simple solution to the 1D min-time problem can be applied in each dimension for a solution to the 2D problem. Choosing equal acceleration constraints in each dimension for every problem as shown in (a) is simple, but significantly reduces performance. Choosing an acceleration allocation to suit the demands of each problem, as shown in (b), provides a fair approximation to the optimal control.

1.4 Going Fast While Avoiding Obstacles

The open-space control algorithm, by itself, is not an adequate control procedure for navigation. Due to control and estimation errors, the robot’s trajectory will not exactly follow the direct path to the next distinctive place, or may overshoot the target position. When the robot does not stop at each place, the trajectories between places will become curved, possibly moving the robot in the vicinity of obstacles. Moreover, the environment itself may change. For all of these reasons, in addition to being guided by the estimated position of the next subgoal, the robot must be able to respond to the currently sensed geometry in order to ensure that it will avoid collision. An obstacle-avoidance controller is needed to direct the robot toward the current subgoal while using current range data to avoid obstacles.

Chapter 4 describes a high-speed obstacle-avoidance controller based on the idea that the robot should be able to reach its current subgoal without collision by traveling toward the subgoal as long as there are no obstacles in its path, and, when an obstruction is found, traveling toward a “temporary target,” chosen to direct the robot around the obstruction. This “dodging” controller is capable of high performance, but does not require the intensive computation of global trajectory optimization in the presence of obstacles because control generation is broken into two easy-to-compute procedures: selection of a temporary target, and generation of open-space minimum-time control to the target.

The dodging algorithm checks the current range image at the start of a control interval, assigns a temporary target, and then follows the open-space control to it over the interval. Temporary target selection is performed by checking whether the robot’s open-space trajec-
Figure 1.4: Collision with an obstacle may be avoided by directing the robot toward a point, e, positioned near the edge of the occluding surface nearest the projected hit point, h. The robot’s current position and goal position are labeled r and g, respectively.

...tory to the current subgoal would result in a collision. If a potential collision is detected, the temporary target is positioned near an edge of the obstacle, as illustrated in Figure 1.4; otherwise a zero-velocity temporary target is positioned where the open-space trajectory crosses out of sensor range. This control procedure provides a non-collision guarantee because it always chooses a control so that the robot can safely come to a stop within currently visible free space. The non-collision guarantee is maintained in the presence of control errors by growing visible obstacles by the amount of position error the robot might accumulate at that position.

With the dodging algorithm to guide it, it is safe to let the robot travel past distinctive places without stopping by pursuing the next subgoal when it gets within some specified range of the current subgoal. This could not be done safely without a collision-avoidance capability because the resulting open-space trajectory could intersect obstacles. Figure 1.5 illustrates the trajectory that results when the robot begins to pursue the next subgoal when it gets within 3 meters of the current subgoal.

Of course, the dodging algorithm also allows the robot to avoid collisions with unexpected obstacles. The robot’s trajectory with the addition of several obstacles is illustrated in Figure 1.6.

1.5 Using Stored Knowledge of Environment Geometry

One of the advantages of obstacle-avoidance control is that it allows the robot to respond to unanticipated environment features that arise due to control and estimation errors and changes in the environment. However, disadvantages of purely local navigation methods were noted earlier: the robot may choose an inefficient route to the goal, and performance is restricted because the robot can only anticipate environment geometry that is within sensor range. The former disadvantage is avoided by following the sequence of subgoals along a route established during the exploration phase. However, the dodging algorithm guarantees non-collision by ensuring that the robot can always come to a stop within the currently
Figure 1.5: With the obstacle-avoidance control mechanism in place, the robot can safely begin pursuing the next subgoal before the current subgoal is reached. Here, the robot switches to the next subgoal when it gets within 3 meters of the current subgoal. Total travel time: 33.5 s.

Figure 1.6: The obstacle-avoidance control mechanism allows the robot to avoid unexpected obstacles. Total travel time: 33.5 s.
Figure 1.7: As the robot travels through the environment, it stores local range information in data structures called “panes.” Also stored in each pane is a description of the robot’s position; this range information is recorded in terms of the approximate relative positions of adjacent distinctive places.

visible free space, so the robot’s velocity is limited by sensor range.

With access to knowledge of previously encountered environment geometry, it is unnecessary to restrict temporary targets to lie within the visible region; they can be positioned further ahead, reducing the restriction on robot velocity posed by sensor range. However, the representation of “open space” beyond sensor range is generated from stored data; if this is no longer correct due to environment changes then non-collision cannot be guaranteed.

A simple and general method for representing environment geometry is to store a sequence of range images. Uncertainty regarding the relative positions of these images is represented by annotating each stored range image with approximate transforms to adjacent distinctive places.

This information can be easily applied to navigation by selecting appropriate stored range images and combining them to generate an occupancy grid which covers a region larger than a single range image, as illustrated in Figures 1.7 and 1.8. The dodging algorithm can use this occupancy grid in place of the range image.

1.6 Improving Performance with Experience

Using stored knowledge of environment geometry to anticipate obstacles that are beyond sensor range or occluded allows the robot to travel at higher speeds while maintaining a non-collision guarantee in a static environment. However, there is room for still more im-
Figure 1.8: The three panes shown in Figure 1.7 plus a current range reading are combined to create an approximate representation of the layout of nearby obstacles that are beyond sensor range. The robot's position is at the center of this square. The dark grey regions correspond to the mean estimated positions of obstacles. The light grey regions are "grown" around the mean positions so that the actual position of the obstacles lie within these grey regions with 98% likelihood.

Figure 1.9: Using stored knowledge of environment geometry, the robot anticipates environment geometry beyond sensor range and travels more rapidly. In this example, a plan occupancy grid is created with a radius of 12.8 meters, double the sensor range. Total travel time: 28 s.
provement in performance for two reasons.

First, whereas the high-speed algorithm’s goal is to navigate to the current subgoal, the overall objective is to get to the goal state in minimum time. Directing the robot to the subgoal may not ideally serve this objective since the precise location of the subgoal and its velocity are chosen somewhat arbitrarily. Performance could be improved if the subgoals were modified to reflect the overall objective.

Second, although the dodging algorithm is designed for high performance, it has some limitations. The most basic is that the method is inherently local: temporary target selection does not consider features beyond the nearest obstacle, nor is control influenced by the presence of an obstacle until the obstacle appears within range of the occupancy grid. Also, to keep things reasonably simple, there are restrictions on the selection of the temporary target (e.g., the temporary target velocity is always aligned with the robot’s line of sight to the temporary target position) that may prevent the optimal temporary target state from being chosen.

These shortcomings can be avoided by introducing new subgoals and updating subgoals. When the robot travels along a route at high speed, a new subgoal can be added wherever a dodge must be performed to avoid collision. Introducing such a subgoal allows the control that steers around this obstacle to reflect global factors, rather than being performed on-line on the basis of local information. Then, sometime before the route is traversed again, each of the subgoal states can be updated in a direction that reduces the overall travel time.

Subgoals are updated incrementally due to uncertainty in knowledge of environment geometry and robot control outcomes. Then when the route is traveled again, subgoal positions can be updated so that they remain at the trajectory’s point of closest approach to obstacles. This ensures that the open-space control between subgoals is (at least nearly) collision-free without the use of a global metrical model of environment geometry. By applying this procedure several times, the robot arrives at a control that truly reflects the global layout of the environment. Performance of the robot after its subgoals have been updated over the course of two prior traversals of the route is illustrated in Figure 1.10.

1.7 Overview

High speed navigation is difficult in part because of a conflict between performance and uncertainty. Generating the best possible performance requires an accurate model of the environment and of the robot’s dynamics, and is a computation-intensive task. However, due to control, estimation, and map errors, the robot cannot simply follow a feedforward plan, but must respond to the information that becomes available as it travels through the environment. Previous work has tended to either focus on performance, assuming that accurate environment and robot models are available, or to focus on the responsiveness that can help the robot cope with uncertainty, neglecting the opportunity for higher performance that is afforded by a global model of the environment. The objective of this research is
Figure 1.10: Performance is improved by modifying the position and velocity of subgoals. Stored knowledge of environment geometry is used again in this run. Total travel time: 26 s.

the development of a navigation method that uses map knowledge to attain a high level of performance under realistic conditions of limited positioning accuracy. A basic feature of this navigation method is the combination of guidance from an approximate map with responsiveness to current sensory input.

Chapter 2 describes some details of the robot model for which the navigation system has been developed. Chapter 3 describes the open-space control algorithm that is the foundation of the high-performance obstacle avoidance controller described in Chapter 4. Chapter 5 describes a method for acquiring a topologically accurate and metrically approximate map of an environment despite positioning errors. Using stored knowledge to anticipate environment geometry beyond sensor range is described in Chapter 6. With experience, control can be improved so that it reflects the global layout of the environment using the method described in Chapter 7.
Chapter 2

A High-Speed Robot Model

This chapter describes the robot model for which the high-speed navigation system has been developed. In addition to describing the model of the robot’s dynamics, sensors, and control and estimation errors, the method used to represent approximate position information is described.

2.1 Dynamics

The following simple model is chosen for the robot’s dynamics:

\[
\begin{bmatrix}
\dot{x} \\
\dot{v}
\end{bmatrix} = \begin{bmatrix}
0 & I \\
0 & 0
\end{bmatrix} \begin{bmatrix}
x \\
v
\end{bmatrix} + \begin{bmatrix}
0 \\
u
\end{bmatrix}
\]  (2.1)

where \(I\) is the 2D identity matrix, \(x\), \(v\), and \(u\) are two-dimensional vectors describing the robot’s position, velocity, and applied acceleration, respectively. The robot’s applied acceleration is subject to magnitude constraint: \(\|u\| < a_{\text{max}}\). Velocity is constrained independently in each dimension: \(|v_1| < v_{\text{max}}, |v_2| < v_{\text{max}}\).

Equation 2.1 simply describes the behavior of a particle in two dimensions subject to a constrained applied acceleration \(u\) and with constrained velocity. This is an appropriate model for omnidirectional mobile robots, which, unlike conventionally-steered vehicles, are not subject to steering curvature constraints. Examples of omnidirectional mobile robots include the Llaniator Cart [Daniel et al., 1985], Uranus [Moravec, 1986], and the Unimation robot [Carlisle, 1983].

Certainly this is not a complete model of the dynamics of any actual mobile robot. However, a high-speed control solution to this system can be used to control a physical omnidirectional robot as follows. The solution is expressed as a sequence of control (acceleration) values. These control values, applied to the equation of motion for this model, generate a sequence of states, or a reference trajectory. It has been demonstrated that by choosing acceleration and velocity constraints to match the characteristics of a particular robot, the
reference trajectory can be quite closely followed by the robot under control of a properly
designed servomechanism [Feng et al., 1989]. The servoed robot will not follow the reference
trajectory exactly, of course, so the actual outcome of applying the servo control will be a
trajectory that only approximately behaves as predicted by the model of equation 2.1. The
modeling of control errors is described in Section 2.3.

2.2 Sensors

The robot is assumed to have three sources of sensory information: wheel encoders, a range
sensor, and a distinctive place detector.

**Wheel encoders.** At the end of each time step, the robot has access to an uncertain
estimate of the robot's movement and change in orientation over the time step and its current
velocity. Modeling of error in this data is discussed in section 2.3.

**Range sensor.** Range data is supplied to the robot in the form of a robot-centered
occupancy array. Cells marked as clear correspond to locations that are free of obstacles;
those marked as filled either contain an obstacle surface or are occluded by an obstacle
surface from the robot's current position. In the current implementation, the occupancy
array is square, with 32 elements per side. The center of the array corresponds to the
robot's position.

Distance sensing error is assumed to be proportional to the distance to the sensed ob-
ject. Support for this assumption is provided by Miller and Wagner's work with an optical
range finder [1987]. Obstacle surfaces are described by line segments, and errors are intro-
duced into the range sensor data by incrementing the apparent position of the endpoints
of segments within sensor range by a the value of a two-dimensional random variable. The
random variable is drawn from a symmetric, two-dimensional Gaussian distribution with the
magnitude of nonzero covariance entries proportional to the square of the distance from the
robot to the segment endpoint.

This local range data is presently used solely for obstacle avoidance. The robot also
should be able to recognize previously seen distinctive features in the environment. The
implementation of a matching capability in the presence of noise is a nontrivial problem,
and the details depend on the particular characteristic of the sensing system. Since there
appears to be little value in implementing matching capability for this particular simulated
system, an abstraction of this capability is used in the simulation: a "distinctive place
detector".

**Distinctive place detector.** When the robot is within some threshold distance of a
distinctive place, its approximate position relative to the robot is indicated by the distinctive
place detector; otherwise it indicates that there is no distinctive place within range. To model
error in this measurement, the relative coordinates returned by this sensor are randomly
drawn from a radially symmetric two-dimensional Gaussian distribution with mean value
equal to the actual relative position of the subgoal. The magnitude of nonzero entries in the
distribution’s covariance matrix are proportional to the squared distance to the subgoal.

### 2.3 Modeling Control and Estimation Error

Since mobile robots have limited positioning accuracy, position uncertainty in control outcomes should be modeled. Moreover, the robot’s source of information about its movements is wheel encoder data, so its movement estimates will be subject to similar errors and these should be modeled as well. Experimental work with mobile robots under servo control indicates that the maximum displacement error from the intended target position due to a movement is proportional to the distance of the movement [Feng et al., 1989; Nelson and Cox, 1988]. Position error along the direction of the movement is generally due to wheel slip. There will also be position error orthogonal to the movement, predominantly due to errors in the robot’s estimated orientation [Crowley, 1989] ¹.

In the simulation, control and estimation errors are modeled as follows. When the control for the next time step is chosen, this control and the robot’s current state are plugged into the equation of motion for the system described by Equation 2.1 to obtain a nominal next state. The position of the actual next state is drawn from a distribution with mean value corresponding to the position of the nominal next state. The estimate of the movement is drawn from a distribution with mean value corresponding to the displacement from the current state to the actual next state.

Because the robot’s movement estimates are based on the movement of its drive mechanism, the sources of error affecting movement will also affect the estimation of movement, and the same distribution is used for both control and estimation errors.

The distribution of control and estimation errors for a given displacement is modeled by a two-dimensional Gaussian distribution, with covariance chosen so that maximum displacement error at a given level of likelihood is proportional to displacement magnitude. The variables used to define the distribution are illustrated in Figure 2.1. The mean displacement, corresponding to the displacement from the current state to either the nominal next state (in the case of control error generation) or to the actual next state (in the case of estimation error generation) is described by the magnitude of the displacement, $d$, and its direction, $\theta$, with respect to the robot’s local coordinate frame. The estimate of movement along the direction of the displacement is modeled by a Gaussian random variable $l$ with mean $d$ and variance $\sigma_l^2 = \kappa_l d$. The error in the movement estimate orthogonal to the direction of the robot’s movement is modeled by a Gaussian random variable $m$ with mean zero and variance $\sigma_m^2 = \kappa_m d$. The values of $\kappa_l$ and $\kappa_m$ can be chosen to reflect the error characteristics of a particular system. Then the random vector, $X$, describing the movement estimate in the

¹Because the simulated robot in this study is omnidirectional, it is unnecessary to model the robot’s orientation (its orientation doesn’t affect the control actions that can be performed from that state). The effect of orientation error is implicitly modeled by having a significant component of error perpendicular to the movement.
Figure 2.1: Variables used in the definition of the form of the covariance matrix for the control and estimation error distribution. $d$ and $\theta$ describe the magnitude and orientation of the mean displacement, respectively. $l$ and $m$ are random variables with mean values $d$ and zero, respectively. $(x_1, x_2)$ are the coordinates of the position corresponding to a particular value of the random variables $l$ and $m$.

The robot's coordinate frame is given by:

$$
\begin{bmatrix}
  x_1 \\
  x_2 
\end{bmatrix} =
\begin{bmatrix}
  \sigma_l \cos(\theta) & \sigma_m \sin(\theta) \\
  \sigma_l \sin(\theta) & \sigma_m \cos(\theta)
\end{bmatrix}
\begin{bmatrix}
  l \\
  m
\end{bmatrix}
$$

(2.2)

Since $l$ and $m$ have variance $\sigma_l$ and $\sigma_m$, respectively, $X$ can be rewritten as a function of unit variance random variables $l' = \frac{l}{\sigma_l}$, $m' = \frac{m}{\sigma_m}$ as follows:

$$
\begin{bmatrix}
  x_1 \\
  x_2 
\end{bmatrix} = c
\begin{bmatrix}
  l' \\
  m'
\end{bmatrix}
$$

(2.3)

Where

$$
c =
\begin{bmatrix}
  \sigma_l \cos(\theta) & \sigma_m \sin(\theta) \\
  \sigma_l \sin(\theta) & -\sigma_m \cos(\theta)
\end{bmatrix}
$$

(2.4)

If $l$ and $m$ are independent, the covariance matrix of $X$ is given by $cc'$:

$$
\begin{bmatrix}
  \sigma_l^2 \cos^2(\theta) + \sigma_m^2 \sin^2(\theta) & \sigma_l^2 \sin(\theta) \cos(\theta) - \sigma_m^2 \sin(\theta) \cos(\theta) \\
  \sigma_l^2 \sin(\theta) \cos(\theta) - \sigma_m^2 \sin(\theta) \cos(\theta) & \sigma_l^2 \sin^2(\theta) + \sigma_m^2 \cos^2(\theta)
\end{bmatrix}
$$

(2.5)

The Gaussian distribution with this covariance matrix can be described as follows. The equiprobability contour of a two-dimensional Gaussian distribution is elliptical. The axes of an equiprobability contour ellipse for the Gaussian distribution with this covariance are parallel and antiparallel to the direction of displacement; the length of the axes are proportional to $\sqrt{\sigma_l}$ and $\sqrt{\sigma_m}$, respectively.
2.3.1 Error Magnitude

Since the level of error in the simulations should reflect the error level encountered with physical mobile robots, this section discusses how to choose parameters for the error distribution described in the previous section so that the level of simulated errors will be comparable to error levels measured in physical mobile robots.

As mentioned earlier, experiments indicate that a mobile robot's maximum displacement error, \( e_{\text{displ}} \) resulting from a movement of a distance \( s \) can be modeled as a linear function of movement distance:

\[
e_{\text{displ}} = \zeta s.
\]

The level of the maximum error scaling factor \( \zeta \) varies with such factors as the degree of precision of the robot's drive mechanism and the smoothness and cleanliness of the floor. Observed values for \( \zeta \) have ranged between .02 and 0.10 [Feng et al., 1989; Nelson and Cox, 1988].

This worst-case characterization of error can be thought of as describing a two-dimensional uniform probability distribution with radius \( \zeta s \). As described in the previous section, the simulations in this research use a Gaussian distribution for positioning error, and the smoothness of this distribution seems a bit more plausible. The parameters of the Gaussian distribution in the present simulations can be chosen with a level of error comparable to the empirical results stated in worst-case terms by choosing the variance to be equal to the variance of the uniform distribution with radius \( \zeta s \):

\[
\sigma^2 = \frac{(\zeta s)^2}{3} \quad (2.6)
\]

One of the uses of a model of the robot's error is to ensure noncollision despite positioning error by growing obstacle representations to compensate for the uncertainty in the robot's position knowledge and control outcomes. While it is possible to grow obstacles by an amount corresponding to the worst-case error when error is modeled using a uniform distribution, with the Gaussian error distribution, obstacles must be grown by an amount corresponding to the maximum error at some likelihood level, as discussed in Section 4.4.10. An important benefit of this formulation is that whereas the magnitude of the worst-case error in a measurement will grow proportionally to the measured distance, the magnitude of error at some confidence level will grow proportionally to the square root of the measured distance. For example, suppose that the robot makes a sequence of \( N \) movements, the distance of each movement is described by random variables \( s_1, \ldots, s_N \), and for all \( i \), \( \mathbb{E}[s_i] = \mu_0 \), \( \text{Var}[s_i] = \sigma_0^2 \). The variance of the distribution of the sum of these distances, \( S \), is given by the sum of the variance for each individual distance:

\[
\text{Var}[S] = \sum_{i=1}^{N} \text{Var}[s_i]
\]

Thus the variance in the distribution of the sum of the movements scales linearly with the number of time steps \( N \) as well as with the mean total distance traveled, \( N\mu_0 \). The width
of the error bound at some confidence level is proportional to the standard deviation of the
distribution of the error, so the width of the error bound is proportional to the square root
of the variance and to the square root of the distance traveled.

Notice that, while the variance of this distribution over multiple time steps scales with the
total distance, the variance according to Equation 2.6 scales with the square of the distance.
To make things a bit cleaner and have the same dependence on distance for both single-step
and multi-step error distributions, the variance in position over a single time step can be
chosen to vary linearly with the distance of the move, \(s\):

\[
\sigma^2 = \frac{\xi^2 s_{\text{max}}^2}{3} s
\]

where \(s_{\text{max}} = v_{\text{max}} \delta t\) is the maximum move over a time step, \(\delta t\). This slightly overstates
the error over a time step compared to Equation 2.6. Then the method for modeling error
described in the previous section will generate this level of error by choosing \(\kappa_l = \kappa_m = \frac{\xi^2 s_{\text{max}}}{3}\).
Reported simulation results will indicate the level of error in the simulation by indicating
the value of \(\xi\).

2.4 Representing Approximate Position Information

As noted above, the robot’s measurements of its movements are subject to error, so the
robot must be able to represent uncertain position knowledge. In the current application,
this uncertainty level must be represented for several reasons. First, when independent
estimates of the same value are combined, the level of uncertainty in each estimate must be
known in order to properly weight each estimate in generating a new estimate. Second, the
level of error in the estimated position of obstacles must be known in order to guarantee
noncollision (at a given likelihood level).

A practical way to represent the level of uncertainty in a position estimate is to use a
covariance matrix, as proposed by several researchers [Smith and Cheeseman, 1986; Crowley
and Ramparany, 1987]. Here, approximate position estimates are represented using Smith
and Cheeseman’s approximate transforms (AT’s). An AT consists of two parts: a vector
describing the estimated transform from one coordinate frame to another, and a covariance
matrix indicating the level of uncertainty in this estimate.

There are several operations that need to be performed on AT’s. First, individual esti-
mates of a sequence of movements must be combined with the compounding operation to
obtain an estimate of the movement over the entire sequence. Second, several independent
estimates of the same value must be combined, or merged, to obtain a lower-error estimate
of the value. Finally, it will be useful to determine an equiprobability contour for an AT to
some point: the region that, with some probability, the point will lie within.
2.4.1 Compounding

Suppose that there are three points A, B, and C, and there are ATs describing the estimated displacement from A to B and from B to C. In a variety of circumstances, it is necessary to "compound" these individual estimates to find an estimate of the displacement from A to C. For example, upon beginning to travel to a distant subgoal, the robot may have an AT describing the distance to the subgoal. After moving toward the subgoal for one time step, it will have an estimate of how far it has moved, also represented as an AT. Combining these two ATs with the compounding operation provides an estimate of the robot’s new distance to the subgoal. Since there is uncertainty in each AT, the accuracy of the new estimate is lower than that of either initial AT.

The mean vector resulting from compounding two ATs is given by the sum of their mean vectors. The covariance matrix resulting from the compounding operation should be the covariance of the distribution of the sum of the multivariate Gaussian random variables corresponding to the two initial ATs. The general form of the compounding operation is described in Appendix A, but the idea can be illustrated using a one-dimensional example. Suppose A, B, and C are collinear, and the estimated distance from A to B and from B to C are described by one-dimensional ATs \((d_1, \sigma_1^2)\) and \((d_2, \sigma_2^2)\), respectively; each AT can be interpreted as the distribution of a one-dimensional Gaussian random variable. Then an estimate of the distance from A to C is given by the mean and variance of the distribution of the sum of these random variables, or \((d_1 + d_2, \sigma_1^2 + \sigma_2^2)\).

2.4.2 Merging

The need to combine several independent estimates of a value may arise as follows. During travel between subgoals, the robot updates its estimate of the position to the current subgoal using the compounding operation as described in the previous section. When it gets within the vicinity of the subgoal, local sensory information provides an independent estimate of the subgoal position. The merging operation optimally combines these two estimates using the Kalman update equations. The result is a new position estimate with a lower level of uncertainty than either of the initial estimates. The procedure for performing merging in the general case is described in Appendix A. In the one-dimensional case, approximate transforms \((d_1, \sigma_1^2)\) and \((d_2, \sigma_2^2)\) are combined by compounding to produce an approximate transform \((d_3, \sigma_3^2)\) as follows:

\[
x_3 = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2} x_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} x_2
\]

(2.7)

\[
\sigma_3^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 + \sigma_2^2}
\]

(2.8)
2.4.3 Equiprobability Contour

When uncertainty in the estimated displacement to a point is described by a covariance matrix, it is useful to translate the covariance matrix into a range of positions within which the point likely falls. An approximate transform can be viewed as a two-dimensional Gaussian distribution that approximates the estimated value’s underlying distribution. Therefore, an equal probability contour of the Gaussian distribution approximates the range of positions that the actual position will fall within with a given level of likelihood. The contour of equal probability at a specified confidence level for a Gaussian distribution is an ellipse described by its half major axis length, half minor axis length, and the orientation of the major axis with respect to the robot’s local coordinate frame, respectively denoted \(a\), \(b\), and \(\theta\). These parameters depend on the covariance matrix \(V\) of the distribution and the confidence level, \(Pr\), as follows:

\[
\begin{align*}
\theta &= \frac{1}{2} \arctan \left( \frac{2B}{A - C} \right) \\
a &= \sqrt{-4 \log(1 - Pr)} \quad \frac{A + C - T}{A + C + T} \\
b &= \sqrt{-4 \log(1 - Pr)} \quad \frac{A + C - T}{A + C + T}
\end{align*}
\]

where \(A = \frac{\|v\|}{\|\nu\|}\), \(B = \frac{-\|v\|}{\|\nu\|}\), and \(C = \frac{\|\nu\|}{\|\nu\|}\).
Chapter 3

Going Fast in Open Space

The foundation for the approach to high-speed navigation developed in this research is an approximate solution to the open-space minimum-time control problem. This problem can be stated as follows. Given an initial position and velocity, a final position and velocity, and a description of the robot’s dynamics, find a time sequence of control, \( u(t) \), that minimizes the robot’s travel time from the initial state to the final state. As discussed in Section 2.1, the following model is chosen for the robot’s dynamics:

\[
\begin{bmatrix}
\dot{x} \\
\dot{v}
\end{bmatrix} = \begin{bmatrix}
0 & I \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
x \\
v
\end{bmatrix} + \begin{bmatrix}
0 \\
u
\end{bmatrix}
\] (3.1)

where \( I \) is the 2D identity matrix, \( x, v \), and \( u \) are two-dimensional vectors describing the robot’s position, velocity, and applied acceleration, respectively, with \( ||u|| < a_{max}, |v_1| < v_{max}, |v_2| < v_{max} \).

This chapter first describes a solution to the problem when acceleration alone is constrained; Section 3.5 discusses how a velocity constraint can easily be added to the solution.

The magnitude constraint on the acceleration applied to the robot means that allowable accelerations lie within a circle of radius \( a_{max} \), as shown in Figure 3.1a. Were allowable accelerations subject to independent constraints in each dimension, allowable accelerations would be constrained to a square with half-width \( a_{max} \), as shown in Figure 3.1b. While independent acceleration constraints are suitable for modeling, say, a Cartesian manipulator, the magnitude constraint more accurately describes an omnidirectional mobile robot’s dynamic constraints.

This is unfortunate since the minimum-time problem with independent acceleration constraints has a simple exact solution. The decoupling of the constraints in each dimension permits the decomposition of the problem into two one-dimensional problems: finding a control to move from the x-component of the initial state to the x-component of the final state, and similarly for y. The well-known bang-bang solution to the problem in one dimension [Athans and Falb, 1966] can be applied to each dimension independently.\(^1\)

\(^1\)The optimal travel time in each dimension will usually be different. An optimal solution to the 2D
Figure 3.1: In (a), the shaded region shows allowable accelerations when acceleration is subject to a magnitude constraint $a_m$. In (b), the shaded region indicates the allowable accelerations when acceleration is subject to an independent constraint in each dimension

one-dimensional bang-bang control is defined in Table 3.1 for reference.

Finding the optimal solution when acceleration is subject to a magnitude constraint is more difficult. An exact solution to this problem has been described by Feng and Krogh [1986]. Finding the time sequence of control entails solving a set of nonlinear equations; the authors found that standard equation-solving packages often failed to converge, so they solved the problem using a continuation method in which the solution to successively closer approximations to the target problem are used as input to the next approximation.

For application to the mobile robot navigation task, a more computationally efficient solution is needed since, due to uncertainty, the robot’s minimum-time control to its current target will be recomputed as it moves through the environment and updated information becomes available. A reasonably close approximation to the true optimal solution is adequate, so we can hope to find a faster approximate algorithm.

### 3.1 A Constraint Allocation Approximation

An obvious approximate solution to the minimum-time problem with acceleration magnitude constraint $a_{max}$ is to use the solution to the problem with independent constraints $\sqrt{2}a_{max}$ in each dimension. This solution would restrict the control to lie within a square inscribed in the radius $a_{max}$ circle corresponding to the robot’s acceleration constraint, as shown in Figure 3.2a.

This is attractive because it lets us apply the simple, fast solution to the independent constraint problem to the magnitude constraint problem. The drawback, of course, is that problem is given by using the bang-bang solution in the dimension that takes longer, and using any control that gets to the final state in the other dimension in the same time.
### 1D Minimum-Time ("Bang-Bang") Control

<table>
<thead>
<tr>
<th>Control Action</th>
<th>Time Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1 = sa$</td>
<td>$t_1 = (v_{sw} - v_i)/(sa)$</td>
</tr>
<tr>
<td>$u_2 = -sa$</td>
<td>$t_2 = (v_f - v_{sw})/(-sa)$</td>
</tr>
</tbody>
</table>

$s = \begin{cases} 
-1 & \text{if } (v_i \leq v_f \text{ and } x_i > \Psi_2(v_f)) \text{ or } (v_i > v_f \text{ and } x_i \geq \Psi_1(v_f)) \\
1 & \text{otherwise} 
\end{cases}

where \( \Psi_1(v) = \frac{v^2 - v_f^2}{-2a} + x_f \) \quad \Psi_2(v) = \frac{v^2 - v_f^2}{2a} + x_f

$v_{sw} = s\sqrt{as(\alpha_f - \alpha_i)}$

where $\alpha_i = x_i - v_i^2/2sa \quad \alpha_f = x_f + v_f^2/2sa$

Table 3.1: Definition of minimum-time control strategy, without velocity constraint. The initial state is \((x_i, v_i)\), the target state is \((x_f, v_f)\) and the allowed acceleration is denoted \(a\).

---

**Figure 3.2:** There is a relatively simple solution to the min time with a magnitude constraint problem if we use a subset of the available acceleration with independent constraints in each dimension. Either a fixed constraint scheme (a) or a more efficient constraint allocation scheme (b) may be used.
Figure 3.3: The value of theta for which the x and y travel-time curves cross indicates the correct allocation of acceleration constraints for a given problem.

the robot does not make full use of the available acceleration, significantly diminishing performance. The performance of this "fixed constraints" strategy is compared to the optimal performance in Section 3.4 (see Table 3.3). For the cases in this table (all of which have zero velocity targets), the travel time using this approximation takes 24% longer than optimal, on average. In cases with nonzero target velocity, an incremental reduction in available acceleration may cause a discontinuous change in the travel time, resulting in an even larger difference in performance.

It is possible to take advantage of the simplicity and efficiency of the independent constraint solution without a severe performance penalty by changing the allocation of acceleration available in each dimension to suit the demands of a particular problem. The idea is illustrated in Figure 3.2. Rather than always allocating equal acceleration in each dimension as in Figure 3.2a, more or less acceleration can be allocated in each dimension for different problem instances. This approach will be referred to as constraint allocation. The allocation chosen for a problem instance may be denoted by the angle \( \theta = \arctan\left(\frac{a_{max_y}}{a_{max_x}}\right) \).

How should the allocation be chosen for a specific problem? For a given one-dimensional problem, the travel time will decrease as the available acceleration increases, somewhat as shown in Figure 3.3a. For the two-dimensional problem, the travel times in each dimension can be drawn in a single plot versus \( \theta \) as shown in Figure 3.3b. A given value of \( \theta \) implies a level of maximum acceleration in each dimension; the values of the x- and y-plots indicate the corresponding travel times. For a given acceleration allocation \( \theta \), the two-dimensional travel time cannot be less than the maximum of the two curves at that value. The travel time is minimized at the value of \( \theta \) where the two curves cross, assuming that the y travel-time curve decreases monotonically in \( \theta \) and the x travel-time curve increases monotonically.
3.2 Travel Time Discontinuities

The situation is slightly more complicated than indicated by Figure 3.3, however, because there can be a discontinuity in the travel-time curve in some circumstances. The source of this discontinuity can be understood by looking at the diagram representing the one-dimensional minimum-time control in Figure 3.4a. The horizontal and vertical dimensions indicate the robot’s position and velocity, respectively. When the robot state is in the unshaded region, the robot should apply the maximum acceleration in the positive direction; otherwise it should apply the maximum acceleration in the negative direction. The bang-bang control operates by applying the appropriate acceleration until the parabolic “switch curve” boundary between the regions is reached. Thereafter, the opposite control is applied, and the robot state follows along the switch curve to the goal state. The steepness of the switch curve is determined by the level of maximum acceleration.

A discontinuity in the travel-time curve can arise when the initial state is in the “jump region” illustrated in Figure 3.4b. In this region, when the robot is on one side of the switch curve it will go directly to the goal, but when it is on the other it must go past the goal position and then loop back in order to arrive at the target position with the target velocity. Hence there is a discontinuity in the travel time curve at the “critical value” of maximum acceleration for which the switch curve passes through the goal state. When the initial state is in the “kink region”, there is a discontinuity in the first derivative of the travel-time vs. acceleration curve at the critical value of a.

In order to find this critical value, note that all switch curves passing through the goal state are described by

\[ x = \frac{v^2 - v_f^2}{a} + x_f \]

where \( x_f, v_f \) are the position and velocity of the goal state and \( a \) is the level of acceleration. For a given state \( (x, v) \) within the jump or kink region, the critical acceleration, referred to as \( a_{crit1} \), is given by solving this equation for \( a \):

\[ a_{crit1} = \frac{v^2 - v_f^2}{2 \left( x - x_f \right)} \]  \hspace{1cm} (3.2)

The jump region discontinuity presents a problem for our algorithm since it is possible that the travel-time curves in x- and y- may not cross at all, as shown in Figure 3.5. In this example, choosing an acceleration allocation corresponding to a theta value of 1.0 would result in a fairly low travel time in each dimension, but the robot would get to its target in x before reaching its target in y. This is unacceptable, since to reach the final state each component of the final state must be reached simultaneously (except in the special case that the final velocity in one or both dimensions is zero).

It is possible to specify a control that reaches the goal in any specified time, filling in the gap in the travel time curve, by adding two control strategies that apply when the initial state is in the jump region: “delay control” and “backtrack control”. Like the minimum-time
Jump Region:
\[
\{(x, v) \mid v_f \neq 0 \land \text{sign}(v) = \text{sign}(v_f) \land \text{sign}(x_f - x) = \text{sign}(v_f)\}
\]

Kink Region:
\[
\{(x, v) \mid v_f \neq 0 \land \text{sign}(v) \neq \text{sign}(v_f) \land \text{sign}((x_f - x)(v + v_f)) = \text{sign}(v_f)\}
\]

Figure 3.4: The control action for one-dimensional minimum-time control to goal state G from any state for a particular value of maximum acceleration is illustrated in panel (a). Negative acceleration is applied when the robot’s state lies in the shaded region. Discontinuities in the travel-time vs. maximum acceleration curve occur only when the robot is in the “jump” or “kink” regions of state space shown in panel (b). In these regions, the sign of acceleration applied depends on the magnitude of maximum acceleration.
Figure 3.5: **Discontinuity in the travel time curve.** Travel time in each dimension versus theta. Due to the discontinuity in the travel time curve for x, the two curves do not cross and there is no allocation of acceleration such that the minimum time control in each dimension reaches the target simultaneously. These travel times are for traveling from position (0 0) velocity (5 5) to position (35 50) velocity (18 0) with acceleration magnitude constraint 8.

control, each of these strategies applies a constant acceleration for some time period, followed by a period of equal magnitude acceleration of the opposite sign; the initial acceleration always has sign opposite the sign of the goal velocity. Constant acceleration curves for each of these phases are shown in Figure 3.6a. When the acceleration is less than a critical level $a_{crit1}$, the curves cross at two points. To perform delay control, the robot applies the initial acceleration, slowing down until it reaches the first switch point, and then applies the acceleration with opposite sign to reach the goal (Figure 3.6b). In backtrack travel, the robot applies the initial control until it is actually moving away from the goal and reaches the second switch point, whereupon it applies the opposite control to reach the goal (Figure 3.6c).

Note that these two control strategies combine to provide a control for every travel time in the “gap” in the travel time curve. As the allocated acceleration approaches $a_{crit1}$, delay control becomes identical to the minimum-time control with acceleration $a_{crit1} + \epsilon$, while the backtrack control becomes identical to the minimum-time control with acceleration $a_{crit1} - \epsilon$. As acceleration is increased from the value $a_{crit1}$, the constant acceleration curves of Figure 3.6 become steeper until value $a_{crit2}$ is reached and they meet at one point at the origin. This acceleration is given by:

$$a_{crit2} = \frac{v_f^2 + v_i^2}{2(x_1 - x_f)}$$

When the value of allocated acceleration is $a_{crit2}$, delay travel and backtrack travel are identical. Thus, the travel time curves for delay control and backtrack control together form a closed curve that fills the gap in the travel time curve as shown in Figure 3.7.
Figure 3.6: Two ways to get to a target more slowly than the minimum time control: "delay control" (b) and "backtrack control" (c).

Figure 3.7: Adding travel time curves for delay and backtrack control fills in the gap in the minimum-time control travel-time curve so that it is always possible to find controls for each dimension that are perfectly synchronized.
Delay Travel and Backtrack Travel Control

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>apply $u_1 = -\text{sign}(v_1)\ a$ for time $t_1 = (v_2 - v_1)/u_1$</td>
<td></td>
</tr>
<tr>
<td>apply $u_2 = +\text{sign}(v_1)\ a$ for time $t_2 = (v_f - v_2)/u_2$</td>
<td></td>
</tr>
</tbody>
</table>

\[ v_2 = s\sqrt{\frac{v_1^2 + v_f^2}{2}} + u_1(x_f - x_1) \quad \text{where} \quad s = \begin{cases} \text{sign}(v_1) & \text{for delay control} \\ -\text{sign}(v_1) & \text{for backtrack} \end{cases} \]

Table 3.2: Definition of delay and backtrack travel control strategies. The initial state is $(x_i, v_i)$, the target state is $(x_f, v_f)$ and the maximum acceleration magnitude is denoted $a$.

The delay and backtrack control strategies are defined in Table 3.2. The velocity $v_2$ corresponds to the velocity of one of the intersection points of the two constant acceleration curves of Figure 3.6a.

### 3.3 Solving for the Acceleration Allocation

With the gap in the travel time curve filled in, the best acceleration allocation is given by the value of $\theta$ at the intersection point between the x- and y-travel-time curves with the lowest travel time. The constraint allocation algorithm simply finds this intersection point, and generates the corresponding control for each dimension.

The travel-time curves in each dimension consist of as many as four continuous segments each:

1. mintime\textbf{1}: minimum-time control with acceleration greater than $a_{\text{crit1}}$.
2. mintime\textbf{2}: minimum-time control with acceleration less than $a_{\text{crit1}}$.
3. delay: delay control (applies for acceleration values between $a_{\text{crit1}}$ and $a_{\text{crit2}}$).
4. backtrack: backtrack control (applies for acceleration values between $a_{\text{crit1}}$ and $a_{\text{crit2}}$).

Each of these segments of the travel-time curves are monotonic, so it is easy to tell if there is an intersection between a pair of segments from each dimension. Choose the largest interval over which the segments are both defined. There is an intersection if the difference between the x- and y-travel-times at the endpoints of the interval have opposite signs. When an interval containing an intersection is found, the control type (minimum-time, delay, or backtrack) corresponding to the segment in each dimension should be recorded. The value of theta at the intersection can be found by applying a root-finding algorithm such as the van-Wijngaarden-Dekker-Brent method [Press and others, 1992]. This value of theta indicates how much acceleration should be allocated in each dimension.
Given the control type and the allocated acceleration, the control for each dimension can be readily generated as described in Tables 3.1 and 3.2. When there are multiple intersections between segments, the intersection corresponding to the control with the shortest travel time should be selected. Applying these controls simultaneously in each dimension yields the two-dimensional control.

3.4 Comparison to Optimal Performance

To evaluate the performance of the constraint allocation algorithm, the travel times resulting from its application are compared to the optimal travel times for a set of problems from Feng and Krogh’s paper [1986] on the exact minimum-time control with an acceleration magnitude constraint. The results are shown in Table 3.3. On these examples, the constraint allocation algorithm took 3.31% longer than optimal, on average, with the time increments ranging from 0.01% to 7.69%.

This represents a considerable improvement over the fixed constraints approximation which, as noted earlier, took 24.10% longer than optimal, on average, with time increments ranging from 10.96% to 32.29%. Thus, it seems that the constraint allocation approach provides a worthwhile performance benefit.

The results can also be compared to the “steering decision algorithm” developed by Feng [1990]. This is another approximate solution to this minimum-time control problem which, however, restricts goals to have zero velocity. With the average performance taking 10.52% longer than optimal, its performance on these zero-velocity goal problems falls between that of the constraint allocation and fixed constraint approaches.

3.5 Adding a Velocity Constraint

Only very modest modifications are necessary to expand the algorithm to allow for a velocity constraint. In one dimension, the minimum-time control is nearly identical: apply acceleration $-a_{\text{max}}$ when the state is in the shaded region of Figure 3.4, and $a_{\text{max}}$ in the unshaded region, except apply zero acceleration in either region once the maximum velocity has been attained. The minimum-time control algorithm with velocity constraints is defined in Table 3.5.

This of course affects travel time, and hence must be reflected in a change to the travel-time curve. The travel time for the minimum-time control with velocity constraint is given by the sum of times $t_1$, $t_2$, and $t_3$ defined in Table 3.5.

Delay travel control and backtrack travel control are unchanged, as are the critical values of acceleration, $a_{\text{crit1}}$ and $a_{\text{crit2}}$.

The allocation of acceleration in each dimension is performed as before, excepting that the travel-time curve is based on the velocity-constrained travel time.
<table>
<thead>
<tr>
<th>$p(t_0)$</th>
<th>$v(t_0)$</th>
<th>$T_{opt}$</th>
<th>Fixed</th>
<th>Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(15.000 10.000)</td>
<td>(-1.000 4.000)</td>
<td>12.541</td>
<td>16.637</td>
<td>31.61</td>
</tr>
<tr>
<td>(15.562 9.101)</td>
<td>(-1.520 3.833)</td>
<td>12.033</td>
<td>15.921</td>
<td>32.31</td>
</tr>
<tr>
<td>(16.913 6.229)</td>
<td>(-2.886 2.945)</td>
<td>9.625</td>
<td>12.527</td>
<td>30.15</td>
</tr>
<tr>
<td>(17.254 5.224)</td>
<td>(-3.250 2.537)</td>
<td>8.631</td>
<td>11.024</td>
<td>27.72</td>
</tr>
<tr>
<td>(17.531 4.201)</td>
<td>(-3.557 2.086)</td>
<td>7.622</td>
<td>9.366</td>
<td>22.89</td>
</tr>
<tr>
<td>(17.748 3.164)</td>
<td>(-3.801 1.597)</td>
<td>6.813</td>
<td>7.559</td>
<td>10.96</td>
</tr>
<tr>
<td>(17.903 2.115)</td>
<td>(-3.979 1.081)</td>
<td>6.380</td>
<td>7.203</td>
<td>12.89</td>
</tr>
<tr>
<td>(17.997 1.059)</td>
<td>(-4.087 0.545)</td>
<td>6.215</td>
<td>7.206</td>
<td>15.94</td>
</tr>
<tr>
<td>(18.028 0.000)</td>
<td>(-4.123 0.000)</td>
<td>6.177</td>
<td>7.207</td>
<td>16.67</td>
</tr>
</tbody>
</table>

Average 24.10 3.31

Table 3.3: Travel times for constraint allocation and fixed constraint open-space control methods compared with optimal performance. Available acceleration is subject to a magnitude constraint of 1.0 m/s$^2$. Table entries display performance in moving from initial position $p_0$ and velocity $v_0$ to a final state at the origin with zero velocity. Optimal travel times are in the column headed $T_{opt}$. Travel times for the constraint allocation and fixed constraint methods are under the headings labeled $T$. The % incr column displays the percentage increase in these travel times over optimal.
<table>
<thead>
<tr>
<th>$p(t_o)$</th>
<th>$v(t_o)$</th>
<th>$T_{opt}$</th>
<th>$T$</th>
<th>%incr</th>
</tr>
</thead>
<tbody>
<tr>
<td>(15.000 10.000)</td>
<td>(-1.000 4.000)</td>
<td>12.641</td>
<td>13.0</td>
<td>2.84</td>
</tr>
<tr>
<td>(15.562 9.101)</td>
<td>(-1.520 3.833)</td>
<td>12.033</td>
<td>12.75</td>
<td>5.96</td>
</tr>
<tr>
<td>(16.070 8.171)</td>
<td>(-2.013 3.598)</td>
<td>11.330</td>
<td>12.5</td>
<td>10.30</td>
</tr>
<tr>
<td>(16.522 7.213)</td>
<td>(-2.471 3.300)</td>
<td>10.528</td>
<td>11.25</td>
<td>6.86</td>
</tr>
<tr>
<td>(16.913 6.229)</td>
<td>(-2.886 2.945)</td>
<td>9.625</td>
<td>10.5</td>
<td>9.09</td>
</tr>
<tr>
<td>(17.254 5.224)</td>
<td>(-3.250 2.537)</td>
<td>8.631</td>
<td>10.0</td>
<td>15.86</td>
</tr>
<tr>
<td>(17.531 4.201)</td>
<td>(-3.557 2.086)</td>
<td>7.622</td>
<td>8.75</td>
<td>14.80</td>
</tr>
<tr>
<td>(17.748 3.164)</td>
<td>(-3.801 1.597)</td>
<td>6.813</td>
<td>7.25</td>
<td>6.41</td>
</tr>
<tr>
<td>(17.903 2.115)</td>
<td>(-3.979 1.081)</td>
<td>6.380</td>
<td>7.25</td>
<td>13.64</td>
</tr>
<tr>
<td>(17.997 1.059)</td>
<td>(-4.087 0.545)</td>
<td>6.215</td>
<td>7.25</td>
<td>16.66</td>
</tr>
<tr>
<td>(18.028 0.000)</td>
<td>(-4.123 0.000)</td>
<td>6.177</td>
<td>7.0</td>
<td>13.32</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td>10.52</td>
</tr>
</tbody>
</table>

Table 3.4: Comparison of Feng's Steering Decision Algorithm (SDA) with the optimal control. Available acceleration is subject to a magnitude constraint of 1.0 m/s$^2$. Table entries display performance in moving from initial position $p_0$ and velocity $v_0$ to a final state at the origin with zero velocity. Optimal travel times are in the column headed $T_{opt}$. The final column displays the percentage increase in travel time for the SDA over optimal.

### 3.6 Uncertainty

The discussion of open-space control has thus far assumed that perfect state information is available, but in the intended application the robot's position relative to the target will only be known approximately. Moreover, in this application, the objective is to transfer the robot to the vicinity of the goal state, rather than reaching the goal state precisely. A greater concern than reaching a particular state is avoiding collision, and the separate obstacle-avoidance mechanism discussed in the next chapter guarantees non-collision.

A simple approach is used to deal with uncertainty in the robot's state estimate. At regular intervals, the control to the target is recomputed based on the mean value of the current state estimate, and applied over the next interval. (This method of dealing with uncertainty is sometimes referred to as a "naive feedback" or "certainty equivalent" controller.)

But there is a problem with directly applying the naive feedback approach to the minimum-time open-space control task. Consider the simpler one-dimensional case: an arbitrarily small change in state estimate that moves the robot across the switch curve boundary changes the control from the maximum acceleration in one direction to the maximum in the other. And the minimum-time control generally spends a lot of time at the switch curve boundary since the final phase of control drives the state along it to the goal state.

Whether or not this creates a problem depends on which part of the switch curve the robot state is close to. The switch curve can be divided into three sections (Figure 3.8) based
Minimum-Time Control with Velocity Constraint

If \( \| v_{sw} \| \leq v_{max} \) then
use regular minimum-time control

Else if \( \| v_1 \| < v_{max} \) or \( s \neq \text{sign}(v_1) \) then
apply \( u_1 = sa \) for time \( t_1 = (sv_{max} - v_1)/sa \)
apply \( u_2 = 0 \) for time \( t_2 = \frac{\alpha_f - \alpha_i - v_{f}^2 - v_{max}^2}{sa} \)
apply \( u_3 = -sa \) for time \( t_3 = (v_f - sv_{max})/ - sa \)

Else
apply \( u_2 = 0 \) for time \( t_2 = (x_f - x_i)/v_i \)
apply \( u_3 = -sa \) for time \( t_3 = (v_f - v_i)/ - sa \)

where \( x_2 = \frac{v_i^2}{2sa} + \alpha_f \)

Table 3.5: Definition of minimum-time control strategy with velocity constraint. The first set of control rules applies as long as the velocity has not overshot the velocity constraint. The initial state is \( (x_i, v_i) \), the target state is \( (x_f, v_f) \), the allowed acceleration is \( a \), and the maximum velocity is \( v_{max} \). \( v_{sw} \), \( s \), \( \alpha_i \), and \( \alpha_f \) are as defined in the minimum-time control definition.
on the velocity $v$ of the phase plane region it occupies:

1. $\{v \mid \text{sign}(v - v_f) = \text{sign}(v_f) \text{ or } v_f = 0\}$

2. $\{v \mid \text{sign}(v_f - v) = \text{sign}(v)\}$

3. $\{v \mid \text{sign}(v) \neq \text{sign}(v_f) \text{ or } v_f = 0\}$

Failure of the mean state estimate to stay right on the switch curve doesn’t create much of a problem if the robot is in the vicinity of switch curve section 1 or 3. In either case, if the estimated state moved back to the side of the switch curve it was on initially, then the control would simply move the state back toward the switch curve again and on toward the goal. When the estimated state is beyond the switch curve, the control applied is the same as when the robot is on the switch curve. If the robot’s state is beyond the switch curve in section 1, then this indicates that the robot will overshoot the goal state. This behavior is acceptable, however, because our concern is not that the robot not overshoot the state, but that it not collide with obstacles. Guaranteeing non-collision, again, is the province of the obstacle-avoidance mechanism.

However, when the robot is moving along the switch curve in section 2, the minimum-time control will accelerate the robot away from the goal state if the state estimate falls into the shaded region shown in Figure 3.8. I will refer to this as an “undershoot” condition since this action is taken when applying all available acceleration in the direction of the target state would still result in the robot attaining a velocity at the goal position with lower magnitude than the target velocity. The minimum-time control response to this state, accelerating the robot away from the target state to make another run at it, is the correct response in the perfect information case, since it is the fastest way to get precisely to the goal state. But the response is not appropriate in conditions of uncertainty, where the objective is simply to get into the vicinity of the goal state. When the undershoot condition arises, the robot should continue accelerating in the direction of the goal, even though the robot may reach the goal position with velocity magnitude less than that of the goal.

### 3.6.1 Handling Undershoot in One Dimension

In order to respond appropriately, the undershoot condition must be detected. A necessary condition for undershoot is that the robot’s state estimate is within the shaded region of Figure 3.8, or:

$$\text{sign}(v_f - v) = \text{sign}(v) \land \text{sign}(u_1) \neq \text{sign}(v)$$

where $v$ is the robot’s current velocity estimate, $v_f$ is the target velocity, and $u_1$ is the initial control to be taken in this state according to the minimum-time control. However, this is not a sufficient condition for undershoot; in Figure 3.8a, for example, being in the shaded region does not constitute undershoot on the part of the trajectory before the robot crosses the switch curve. To distinguish an undershoot condition, a state variable is added. When
Figure 3.8: The undershoot condition arises when the robot is traveling along the switch curve in region 2 (where $\text{sign}(v_f - v) = \text{sign}(v)$) and it inadvertently crosses the switch curve due to error. Panels (a) and (b) display examples of how the robot can get into the undershoot state.

The robot begins traveling toward its goal state, the variable $\text{ctrl-opposite-vf-sign*}$ is initialized to the boolean value $\text{sign}(u_1) \neq \text{sign}(v_f)$. Then at each time step the variable is updated as follows:

\[
\text{(if (and \text{ctrl-opposite-vf-sign*} (= (sign u_1) (sign vf)))}
\text{(setq \text{ctrl-opposite-vf-sign*} \text{nil}))}
\]

A non-nil value of the variable indicates that the initial leg of the control applies acceleration with sign opposite the sign of $v_f$, so that the robot’s presence in the shaded region does not indicate an undershoot condition. But once the switch curve is reached and an acceleration with the same sign as $v_f$ is applied, subsequent presence in the shaded region indicates the undershoot condition.

When the undershoot condition is detected, the appropriate response in this one-dimensional case is to apply maximum acceleration with the same sign as the target velocity.

3.6.2 Handling Undershoot in Two Dimensions

This solution to the undershoot condition can be adapted to two dimensions. A version of the $\text{ctrl-opposite-vf-sign*}$ variable is initialized and maintained for each dimension as previously described.

When an overshoot condition is detected in either dimension, the solution must be somewhat different than the one-dimensional solution since only a share of the available acceleration can be allocated to each dimension. In order to allocate the acceleration, a procedure similar to the usual constraint allocation algorithm can be applied. Supposing that the undershoot condition is in the x-dimension, replace the x-travel-time vs. theta curve with a function describing the time for the robot to get to the target position with available
acceleration in the x-dimension \( a_{\text{max}} \cos(\theta) \). From the relation

\[
x_f = x + vt + \frac{1}{2}a_{\text{max}} \cos(\theta)t^2
\]

the travel time to the goal position, \( x_f \) with available acceleration \( a_{\text{max}} \cos(\theta) \) can be derived:

\[
t = \frac{-v + \sqrt{v^2 + 2a_{\text{max}} \cos(\theta)(x_f - x)}}{a_{\text{max}} \cos(\theta)}
\]  \hspace{1cm} (3.3)

An appropriate acceleration allocation can be found using a root-finding procedure similar to that in the usual constraint allocation algorithm, but using equation 3.3 in place of the x-travel-time curve. This allocation is such that, in the absence of uncertainty, it would transfer the robot to the target position and velocity in y, and simultaneously to the target position in x.

Simultaneity in the two dimensions is not guaranteed, however, because the time to the target position described by Equation 3.3 is bounded above by \((x_f - x)/v\) for all values of \( \theta \). Therefore, it is possible that the travel time to the target state in y is greater than the travel time to the target position in x for all values of \( a \), so that the y-travel-time curve and the plot of equation 3.3 do not intersect. In this case, all acceleration is allocated to the y-dimension, and the robot travels at constant velocity in the x-dimension toward the target position. Since travel times in each dimension are not equal, the target position will not be reached exactly. This condition seldom arises in practice, and when it does, the difference in travel-times is small. This is as expected, since the control prior to this condition was designed to reach the target in each dimension simultaneously; the difference arises only due to control and estimation error. As noted earlier, missing the exact subgoal position is acceptable; it is much more important to avoid collisions.

### 3.7 Summary

This chapter has described a high-performance approximation to the open-space minimum-time control problem for a double integrator with an acceleration magnitude constraint. The extensions of adding a velocity constraint and dealing with uncertainty in the robot’s state estimate were described. This solution to the open-space control problem provides the foundation of the high-performance reactive navigation algorithm described in the next chapter.
Chapter 4
Going Fast While Avoiding Obstacles

4.1 Introduction

The open-space controller of the previous chapter is not a complete solution to navigation between subgoals. While it is desirable to take advantage of its relative efficiency and high performance when possible, a direct application of this controller cannot easily ensure non-collision. Control between subgoals must respond to sensed local obstacles because any plan based on a stored map will need to be modified to avoid obstacles due to map, control, and estimation errors as well as environment changes.

The necessary responsiveness to sensed local obstacles can be provided by a solution to the "reactive navigation" task. The objective of this task is to transfer the robot to a goal state while avoiding obstacles when the following sources of information are available:

- a model of the robot’s kinematics and dynamics
- the robot’s current position relative to the goal and current velocity
- the geometry of the environment visible from the robot’s current position only

The reactive navigation task has been addressed by a number of researchers. In this chapter I describe a new approach to reactive navigation that is capable of high-speed travel and avoids shortcomings of many previous approaches. The next section describes previous work in mobile robot guidance and especially reactive navigation; succeeding sections describe the new algorithm.

4.2 Previous Work

The initial work in reactive navigation was a reaction to earlier work in mobile robot guidance which formulated the task as a planning problem. In this planning formulation, the input to the algorithm is a description of the geometry of the environment, the robot’s kinematic
and/or dynamic constraints, and the start and goal state. The objective is to generate a detailed plan which, when precisely followed, will guide the robot to the goal without collision. Some of the earliest work simply had the objective of finding a geometric path to the goal that avoids obstacles, possibly subject to a minimum-distance criterion [Lozano-Perez and Wesley, 1979; Reif, 1979; Brooks, 1983; Thorpe, 1984; Takahashi and Schilling, 1989]. But at high speeds, a mobile robot cannot follow an arbitrary geometric path due to its dynamic constraints. For high speed travel, a more appropriate formulation of the problem is the “trajectory planning problem” which has the objective of finding a time-sequence of states, generally subject to a minimum-time criterion. Donald and Xavier [1989] described a combinatorial algorithm for approximating the minimum-time control among obstacles for a particle with acceleration and velocity subject to independent constraints in each dimension. The algorithm is guaranteed to find a solution within a user-specified error bound of the optimal solution; the running time of the algorithm is polynomial in the reciprocal of the error bound, but the algorithm is impractically slow in practice. Zafarakis and Guez [1990] described a more efficient algorithm for the independent constraints problem which assumes that obstacles are described as rectangles aligned with coordinate axes. The solution capitalizes on the nonuniqueness of the two-dimensional minimum-time control with independent acceleration constraints and the availability of a direct solution for the one-dimensional case. Computation time was on the order of tens of seconds on contemporary microcomputers. Shieller and Chen described a method for planning an approximate time-optimal path in a three-dimensional terrain [Shieller and Chen, 1990]. Paths which would allow the robot to slip or tip over are avoided. The method, based on Bobrow's method for finding a minimum time trajectory along a specified geometric path [Bobrow et al., 1985], employs an unconstrained optimization of the parameters describing the geometric path of the robot.

Whether dynamics are taken into account or not, these planning approaches proceed by generating a completely detailed description of the actions that the robot will take based on stored knowledge of the world. This “classical planning” approach assumes that the robot can acquire an accurate model of the world, from which it generates a detailed action plan. In fact, models are inevitably inaccurate due to errors and changing environments. Due to these sources of uncertainty, a practical system must respond to information that becomes available during the execution of the plan. Within the framework of these planning approaches, this would entail a time-consuming procedure of updating the model, replanning, and finally resuming execution.

This type of problem has suggested an alternative to planning approaches which eschews creation of detailed world models or action plans, instead simply reacting to current sensor input. The “reactive navigation” task is formulated as follows: given a goal position, a model of robot dynamics, and current sensor input, direct the robot to the goal while avoiding collision.

The basic problem in designing a solution to the reactive navigation task is how to determine what action to take based on the goal position and knowledge of nearby obstacles.
A simple and elegant solution is to associate an imaginary attractive force with the goal position, and an imaginary repulsive force with each obstacle. The sum of these forces at some position indicates the control action to take at this point. Khatib [1986] proposed applying this "potential field" approach to dynamic obstacle avoidance for mobile robots and manipulators. Brooks [1986] and Arkin [1989] applied a potential field approach to low-speed obstacle avoidance for physical mobile robots. Borenstein and Koren [1989] combined the potential field idea with an occupancy grid representation of local obstacles to implement a high-speed reactive navigator on a physical mobile robot.

Despite their simplicity and appeal, several problems have been noted with the potential field approach. Parameters control the relative influence of obstacle and goal fields, and a single setting may not be appropriate for all conditions. A relatively high weighting of the goal field which allows the robot to move directly to the goal may draw the robot unnecessarily close to obstacles or even allow collision if the robot is traveling at high speeds. But a relatively high weighting of the obstacle field may cause excessive conservatism in some regions, and a local minima may result from the superposition of obstacle fields from adjacent obstacles, preventing the robot from reaching the goal. Obstacles that are not in the way of the robot's path to the goal may nonetheless adversely influence the robot's trajectory [Krogh, 1984; Slack, 1993]. In some circumstances, problems arise with unstable oscillatory behavior in the vicinity of obstacles [Koren and Borenstein, 1991].

Fuzzy logic techniques were proposed as an alternative to potential field methods for reactive navigation by Reignier [1994]. The application of these techniques to the quasistatic reactive navigation task revealed oscillation and local minima problems similar to those encountered with potential fields. Dubrawski and Crowley [1994] described a reactive navigation system for a quasistatic robot whose model is partially known. The initial control is provided by a fuzzy logic controller and the control method is adapted using neural network methods. However, oscillation and local minima effects were sometimes still encountered after learning.

In response to some of these problems, Borenstein and Koren developed a method for reactive navigation on a physical mobile robot that replaces potential field control with a navigation method that steers the robot in the direction of open space. The robot is essentially driven at a constant velocity of approximately 0.7 m/s and its dynamics are not directly modeled.

Slack [1993] described an alternative to potential field local navigation methods for quasistatic mobile robot navigation called "navigation templates", or "NaTs". Similarly to a potential field approach, the desired movement direction at a position is determined by the combined influence of a substrate NaT which moves the robot toward its goal, and modifier NaTs which move the robot around obstacles. But a more complex procedure than superposition of fields is used to generate control: nearby obstacles are searched to choose an objective direction that both makes progress in the direction specified by the substrate NaT and is directed around a nearby obstacle; the final direction of travel is modified from this objective direction to maintain a reasonable distance from obstacles. This method avoids
several of the problems of potential field methods. There are no weighting parameters, and the associated problems are eliminated: the robot is able to pass between nearby obstacles, yet obstacles will be given wider berth when there is room available. Obstacles that are not in the robot's way do not influence the trajectory. In addition, Slack introduces the idea that it is useful to be able to specify a preferred direction of rotation around an obstacle. Then, an approximate navigation plan may be conveyed by the combination of a substrate NaT and rotation directions attached to certain obstacles. With potential field methods, the side of the obstacle that the robot traverses around will depend on details of how it happens to approach the obstacle. Choosing the "wrong" direction may lead the robot far astray, so it is desirable to be able to specify rotation direction in approximate plans. Despite these advantages, the navigation templates method just indicates a preferred direction of travel for the robot, and is not appropriate for directing high-speed control.

Newman and Hogan [1987] describe a method for high-speed control and obstacle avoidance. The method is restricted to the case of independent acceleration constraints and assumes that obstacles are described by rectangles aligned with coordinate axes. Minimum-Time open-space control is performed in each dimension until deceleration to a stop is needed in some dimension to avoid collision. A benefit of this method is that it gracefully accommodates a goal moving at a constant velocity.

Work by Feng and Krogh [1990] addressed high-speed navigation with a method that also avoids shortcomings of potential field methods. Their method allows navigation among nonintersecting convex polygonal obstacles by a robot with acceleration subject to a magnitude constraint. The method consists of two parts: a subgoal selection algorithm, and the steering decision algorithm (SDA) for directing the robot to the current subgoal. The objective of this work was to choose a satisfactory trajectory that obeyed the robot's dynamic constraints, rather than emphasizing high performance. The performance of the SDA is lower than that of the dynamic allocation method described in the previous chapter, and the design of the algorithm requires that subgoals have zero velocity, reducing performance. The SDA employs a simplified description of free space as a triangular region which requires the creation of a subgoal at a nearby obstacle although it is not actually in the robot's way. This compounds the decrement in performance due to zero velocity targets.

None of the previous work is suited to traveling as rapidly as possible when only local obstacle information is available and control decisions must be made quickly on the basis of new information as the robot moves through the environment. The controller described in the following sections has been designed with the objective of the highest performance possible given the locality of the robot's knowledge of obstacles and the need to respond rapidly. It avoids the problems of many reactive navigation approaches, including oscillatory behavior, local minima created by nearby obstacles, and adverse influence from obstacles that are not in the robot's way. The controller can be directed to go around obstacles in a preferred direction.
4.3 High-Speed Reactive Navigation

The rest of this chapter describes a new solution to the high-speed reactive navigation task. As with other formulations of the reactive navigation task, the robot’s goal is specified as a goal state – a position and a velocity in a global frame – but otherwise the robot has no prior knowledge of the geometry of the environment. As the robot travels to the goal state, it has access to distance readings to obstacles within some range and accurate state estimates. The robot’s dynamics are modeled as those of a particle with constrained velocity and subject to a controllable, constrained acceleration as described by Equation 2.1. The objective is to get to the goal state as rapidly as possible while guaranteeing collision avoidance on the basis of visible obstacle geometry.

The initial development of the algorithm assumes that obstacles are convex and nonintersecting. Any local navigation method may mistakenly go down the blind alley of a deep concavity in an obstacle. The robot can be prevented from being trapped in such concavities either by using a trap detection and recovery algorithm, or, more efficiently, avoiding the blind alley in the first place by using global map knowledge as described in Chapter 5. When this reactive algorithm is used in combination with map information, the convexity restriction can be lifted.

Similarly, the assumption of accurate state estimates and control outcomes will be relaxed later, and a method for handling uncertainty described.

The objective of the high-speed reactive navigation task is similar to that of the computation-intensive task of finding the global minimum-time control. But since immediate sensor information provides the only available information about environment geometry, the control must be recomputed frequently as the robot moves through the environment and new sensor information is acquired. In consequence, a solution to the high-speed reactive navigation task requires a less computationally intensive method of computing control.

The new algorithm’s approach to this problem is to detect when the open-space control to the goal state would collide with an obstacle, and “dodge” it by choosing a temporary target to direct the robot around the obstacle. Hence, I’ve dubbed this algorithm the dodging controller. The dodging controller is capable of high performance, but it does not entail the intensive computation of a global optimization procedure since control selection is divided into two easy-to-compute procedures: selection of a temporary target and generation of the open space minimum-time control to this target.

4.4 How to Dodge Obstacles

Since obstacle avoidance is based entirely on visible environment geometry, new information becomes available as the robot travels toward its goal, and the control must be updated periodically to reflect the new information. At the start of each control interval (0.5 s in current simulations), the controller receives updated information: an estimate of the robot’s
current state, and a range reading from the robot's current location. Using this information, a temporary target consisting of a position and a velocity in the robot's local frame is chosen, and the open space control to this target is applied over the control interval. The temporary target must be selected so that the following objectives are met:

1. The robot travels at high speed.

2. The robot avoids collision.

The temporary target selection procedure consists of the following three operations.

- **Collision Checking** The open-space trajectory to the goal state is checked to see if it passes through or within some proximity to an obstacle within sensor range.

- **Choosing an Edge Point** If there is a projected collision or near-collision, then in preparation for choosing a temporary target (which consists of both a position and a velocity) an edge point (which is just a position) is chosen. The edge point is chosen near an edge of the visible portion of the obstacle and indicates a position that the robot should pass through on its way around the obstacle.

- **Choosing a Temporary Target** If there was a projected collision, the temporary target is chosen so that the open-space trajectory to it avoids obstacles and the subsequent trajectory to the edge point is collision-free. Otherwise, a zero-velocity temporary target is chosen near the intersection of the open-space trajectory to the goal with the sensor range boundary.

These operations are described in the following sections.

### 4.4.1 Collision Checking

To check whether an evasive maneuver must be taken to avoid a nearby obstacle, the algorithm compares the open-space trajectory to the goal with occupancy data derived from current sensor input to see if there is a hit point: a point where the open-space trajectory to the goal state crosses out of free space. Since we are assuming that the sole source of information about obstacle geometry is current sensor input, the robot cannot tell whether beyond-range or occluded regions are occupied by obstacles. Thus, only the visible obstacle-free regions count as free space. This occupancy data is represented in a pixel format, and contiguous pixels with a single pixel value are grouped together by representing the pixel array with a quadtree to facilitate collision checking. An example of this representation is shown in Figure 4.1.

To see if the trajectory to the goal state leaves free space, the open-space control to the goal is computed, and all quadtree leaves that the open-space trajectory would pass through are checked. This can be done by sequentially computing the trajectory states at each
leaf boundary the trajectory crosses. Since the control generated by the open-space control algorithm is a sequence of constant acceleration values, the trajectory for each period of constant control $u$ has a simple analytic description:

$$x(t) = x(0) + v(0)t + \frac{1}{2}ut^2$$

$$v(t) = v(0) + ut$$

The time that the robot leaves the current cell can be found by solving for the time for the trajectory to reach the two cell boundaries in each dimension; the time to leave the cell is given by the minimum of the positive values. This time is substituted into the equations above to find the state when a leaf is exited. The time for collision checking is therefore proportional to the number of quadtree leaves traversed. Hence the virtue of organizing the occupancy grid into a quadtree: this trajectory computation does not have to be performed for every pixel that the trajectory crosses.

Quadtree leaves are checked until either:

- the edge of the sensor array is reached (no collision within sensor range) or
- an occupied quadtree leaf is reached or
- the quadtree leaf containing the goal is reached (no collision)

If the edge of the sensor array or an occupied quadtree leaf is reached, then the point where the trajectory leaves free space becomes the hit point.

If the leaf containing the goal is reached, then the robot should pursue the open-space control to the goal state. However, if the goal state has nonzero velocity, moving to the goal state could put the robot in danger of subsequent collision if the goal state has not been properly chosen. To maintain the non-collision guarantee, the amount of free space ahead of the goal in the direction of the goal velocity, $d$, is measured. If necessary, the
velocity magnitude of the goal state is truncated so that it is no greater than \( \sqrt{2a_{\text{max}}d} \). Although this simple method of maintaining the non-collision guarantee is conservative in some circumstances, it does not seriously degrade performance because, as will be discussed in Chapter 5, the robot will not pursue a nonzero velocity goal state all the way to the goal position, but, once it gets in the vicinity of the goal, will begin pursuing a more distant goal.

The collision-checking routine can be expanded to detect when the trajectory would pass very close to obstacles by “growing” the obstacles — uniformly expanding their boundary — before performing collision checking. This can be accomplished by adding one or more layers of pixels adjacent to existing boundary pixels. Given the special nature of the obstacles on the sensor grid (every obstacle is connected to the boundary due to shadowing) this can be easily accomplished as follows. Starting from the robot position, move toward an edge of the sensor array, checking each cell until an obstacle or the sensor edge is reached. Call this the initial edge point. Proceed clockwise along the obstacle surface (or sensor edge), marking pixels that are adjacent to an obstacle boundary until the initial edge point is reached again. Marking the adjacent pixels with a value that is distinct from the occupancy value makes it possible to discriminate between near scrapes and outright collisions.

### 4.4.2 Choosing an Edge Point

Once a hit point is found, the next step is to choose an edge point: the position that the robot should pass through on its way to the goal while dodging the obstacle that is in the way.

When the hit point is on the edge of the sensor array (the projected trajectory to the goal produced no collision within sensor range), then a fine choice for the edge point is simply the hit point itself. Since there’s no obstacle in the way, it is appropriate to simply direct the robot along the open space trajectory. In this case, a zero-velocity target positioned at the edge point will be created in order to roughly direct the robot along this trajectory while ensuring that the robot will be able to stop within free space.

When the hit point is on an obstacle, it is fairly straightforward to choose an edge point. To travel efficiently around the obstacle, the robot should pass somewhere quite near to either the rightmost or leftmost edge of the occluding surface. Since in practice it is probably desirable to provide clearance between the robot’s trajectory and an obstacle, the edge point can be chosen some specified radius away from the edge of the occluding surface, as shown in Figure 4.2. The location of either edge point can be found simply by starting from the hit point and searching the sensor array along the occluding surface in either direction.

But which edge of the occluding surface should be chosen as the edge point? When the robot has no \textit{a priori} knowledge of the environment, it follows the heuristic of choosing the occluding edge that is nearest to the hit point. When the edge points are a similar distance from the hit point, the robot could oscillate between choosing the left edge and then the right edge over multiple time steps. To avoid this behavior, when collision is first detected with an obstacle, the heuristic is used to choose a rotation direction around the obstacle. When
Figure 4.2: The edge point, \( e \), is chosen in the vicinity of the edge of the occluding surface nearest the hit point, \( h \). The position of the robot and of the goal are denoted \( r \) and \( g \), respectively.

Figure 4.3: The rotation direction heuristic chooses the obstacle edge point \( e \) closest to the hit point \( h \). This may result in a poor choice due to a limited sensory horizon (a) or occlusion from another obstacle (b).

potential collisions are subsequently detected with the same obstacle, this same rotation direction is used.

Of course, this heuristic may not make the best choice. Figure 4.3a illustrates a case in which the "wrong" rotation direction appears more attractive due to limited sensor range. Figure 4.3b illustrates a case in which the more desirable edge of an obstacle is not visible due to occlusion.

An alternative to such an heuristic is to make a decision to pass on one side of an obstacle or another on the basis of prior knowledge of the environment, regardless of the local appearance of the obstacle. If a global plan has associated a "spin" with an obstacle [Slack, 1993] indicating a desired side to pass on, then this directive can be easily be followed simply by choosing the corresponding edge point.

Once an edge point is found, a ray in the direction from the robot to the edge point and originating from the edge point is checked for "collision" with occupied cells or the edge of
the sensor array. The distance to this collision will be helpful in determining a safe velocity for the robot at the edge point as described in Section 4.4.3.

When the robot’s sensor has a limited range, it will frequently come to pass that only part of the occluding surface of an obstacle will be visible. In this case, the robot simply chooses the edge point near a visible edge of an occluding surface as shown in Figure 4.4. While it would be desirable for the robot to choose the edge point at the occluding edge of the obstacle itself, this is the best choice possible when only immediate sensor information is available. As will be discussed below, while having a limited sensor horizon degrades performance, it is still possible to guarantee collision avoidance.

4.4.3 Choosing a Temporary Target: The Approach

As noted in the previous section, when the hit point is at the edge of the sensor array, a zero-velocity temporary target can be chosen at the hit point. When the hit point is at an obstacle, the chosen edge point indicates where the robot should go to avoid the obstacle. However, it is not sufficient to simply assign a zero-velocity temporary target to this position and follow the open space control to it. Collision avoidance is not guaranteed even though the edge point is chosen away from the obstacle and space ahead is free, because the open-space trajectory to the edge point may collide with the obstacle, as illustrated in Figure 4.5. In addition, it is desirable to choose a nonzero velocity for the temporary target when possible to increase the robot’s overall velocity.

In short, we wish to choose a temporary target such that:

- The trajectories to the temporary target, between the temporary target and the edge point, and beyond the edge point are all collision-free.

- The travel time to the temporary target is minimized.

Of course, the true optimal choice for the temporary target can’t be found because the complete layout of obstacles is not known. And in any event, this minimization would likely
be prohibitively expensive. To deal with limited knowledge of environment geometry and computational constraints, the problem is further simplified as follows.

First, the non-collision constraint is simplified. Rather than basing collision avoidance on the detailed shape of the obstacle, the algorithm defines an *obstacle boundary line* which bounds the obstacle surface between the hit point and the edge point as illustrated in Figure 4.6. The obstacle boundary line is parallel to the line between the hit point and the edge point, and positioned so that the obstacle surface between the hit point and the edge point lies entirely on one side of the line. Then collision will be avoided as long as the robot does not cross the obstacle boundary line.

To create the obstacle boundary line, the following inputs are used: the edge point and a list of occupied cells encountered during the search of the obstacle surface between the hit point and the edge point, inclusive. Choose a local coordinate frame whose origin lies on the line between the hit point cell and the edge point and whose x-axis is normal to the line and pointing into the half-plane occupied by the robot. Transform the locations of the occupancy cells on the list into this local frame. The cell whose location has the largest x-value is the most prominent cell; call the most prominent point of this cell the *extreme obstacle point*. The obstacle boundary line passes through the extreme obstacle point and is normal to the local x-axis. A nonzero lower bound on the distance between the robot’s trajectory and the obstacle can be established by moving the obstacle boundary line a chosen distance from the extreme obstacle point.

The requirement that the trajectory between the temporary target and the edge point and beyond is collision-free can be simplified as follows. The temporary target’s position is chosen at the intersection of the obstacle boundary line and the robot-to-edge-point line. The velocity of the temporary target is directed along the robot-to-edge-point line, for a couple of reasons. First, this ensures that there is a collision-free trajectory between the temporary target state and the edge point since the velocity vector points along the visibility line between the robot and the edge point. Second, it approximately aligns the velocity with the sensor shadow edge so that the velocity is likely to point into free space and a large velocity will be safe. A state is considered “safe” when the robot can come to a stop from this state within visible free space. The safety of the temporary target state is guaranteed by measuring the free space ahead along the robot-to-edge-point direction and requiring that
Figure 4.6: The temporary target is chosen so that the trajectory to it and which does not cross the obstacle boundary line will avoid collision with the obstacle. The temporary target is denoted \( t \).

The target velocity is no greater than a target velocity constraint:

\[
v_c = \sqrt{2a_{max}d}
\]

where \( d \) is the distance from the temporary target state to the first encountered obstacle. This \( v_c \) is chosen so that applying acceleration \( a_{max} \) in the opposite direction will bring the robot to a stop in distance \( d \).

With the temporary target selection problem thus structured, all that remains is to choose a velocity magnitude for the temporary target that minimizes travel time to the temporary target subject to these constraints:

1. The velocity magnitude is less than \( v_c \).
2. The trajectory to the temporary target with this velocity does not cross the obstacle boundary line.

In fact it will not always be possible to find a velocity magnitude for the temporary target so that the open space trajectory to the temporary target position will be “reasonable”. For example, this trajectory may cross over the obstacle boundary line regardless of the velocity magnitude chosen. In this case a new target position should be chosen; how to do so will be described in section 4.4.7. But first, the reasonableness of a trajectory is defined.

### 4.4.4 What Is a Reasonable Trajectory?

It is fairly straightforward to define a reasonable trajectory in terms of the type of one-dimensional control that the open-space control generates in each dimension. Recall that the open space control algorithm operates by allocating acceleration to each dimension so that applying a one-dimensional control pattern in each dimension will drive the robot to the goal in each dimension simultaneously. The control patterns are backtrack, delay, and minimum-time, which may be further categorized as mintime1 if initial control is toward the
goal and mintime2 if initial control is away from the goal. Restrictions on which of these control patterns are allowed in each dimension, can ensure that the trajectory is reasonable.

The requirement that the trajectory to the temporary target not cross the obstacle boundary line can easily be translated into a restriction on the control patterns if we first transform the problem into a coordinate frame whose x-axis is perpendicular to the obstacle boundary line. Then the obstacle boundary line will not be crossed as long as the x-coordinate of the temporary target position is not crossed.

If, as will usually be the case, the x-component of the robot’s current velocity is zero or in the direction of the x-component of the temporary target position (\(\text{sign}(v_{0x}) = \text{sign}(p_{fx} - p_{ox})\)), then the x-component of the temporary target position will not be crossed before the temporary target state is reached if only mintime1 control is allowed in x. This control accelerates the robot toward the goal and then decelerates the robot to the goal state so that the robot’s movement is unidirectional in this dimension. (Delay travel is also unidirectional, but only mintime1 control will be allowed for the sake of simplicity.)

If the x-component of the robot’s current velocity points away from the temporary target position, the requirement that the trajectory not cross the obstacle boundary line will always be met: neither mintime1 nor mintime2 control (the only control patterns usable in this circumstance) from this state crosses the x-component of the temporary target position. However, the robot does not travel in one direction in x: it moves away from the temporary target before reversing velocity and traveling toward it. If the control in y also resulted in a reversing trajectory, then the result could be a looping trajectory which crosses over itself, another undesirable behavior. We can exclude this behavior by requiring that the control in y be mintime1 control so that movement in y is unidirectional.

The requirements imposed on control in each dimension to ensure reasonable behavior can be summarized as follows:

- If the x-component of the robot’s initial velocity is toward the temporary target position, then the control in x must be mintime1.
- If the x-component of initial velocity is away from the temporary target, then the control in y must be mintime1.

### 4.4.5 Is There a Reasonable Trajectory to This Temp Target?

Given a problem consisting of an initial state, temporary target position, and obstacle boundary orientation, the algorithm must establish whether the temporary target position is ok — some assignment of velocity will result in a reasonable trajectory — before attempting to choose a specific value of velocity to minimize travel time. Recall that in the operation of the open-space control algorithm, the category of control selected for each dimension is determined by where the x and y travel-time curves intersect. So the algorithm can determine whether the temporary target position is ok by checking whether the travel time curves intersect in the appropriate regions for any value of temporary target velocity. As discussed
in the previous section, the appropriate region will be either the mintime1 portion of the x travel-time curve or the mintimel portion of the y travel-time curve.

For a particular value of temporary target velocity, it is easy to establish whether the travel-time curves intersect at the mintimel portion of the x-mintime curve. To describe the intersection conditions, I will refer to the values of theta that correspond to values of \( a_{crit1} \) and \( a_{crit2} \) in x as \( cp1_x \) and \( cp2_x \) and similarly for y. The correspondence is as follows:

\[
cp1_x = \tan^{-1} \left( \frac{\sqrt{a_{max}^2 - a_{crit1x}^2}}{a_{crit1x}^2} \right) \quad cp1_y = \tan^{-1} \left( \frac{a_{crit1y}}{\sqrt{a_{max}^2 - a_{crit1y}^2}} \right)
\]

Critical points \( cp2_x \) and \( cp2_y \) are defined similarly by substituting \( a_{crit2x} \) and \( a_{crit1x} \) for \( a_{crit1x} \) and \( a_{crit1y} \), respectively. Consider the maximum travel time of the mintimel portion of the x-mintime curve (its value at \( cp1_x \)). As illustrated in Figure 4.7(a), if \( cp1_x \) does not lie between \( cp1_y \) and \( cp2_y \) (or there is no \( cp2_y \)), then the x- and y-mintime curves will intersect at the mintimel portion of the x travel-time curve if:

\[
\text{mintime1}_x(cp1_x) > \text{mintime1}_y(cp1_x)
\]

If \( cp1_y < cp1_x < cp2_y \), then the curves intersect at the mintimel portion of the x travel-time curve if:

\[
\left( \text{mintime1}_x(cp1_x) > \text{mintime1}_y(cp1_x) \right) \land \\
\left( \text{mintime1}_x(cp1_x) > \text{backtrack}_y(cp1_x) \right) \lor \\
\left( \text{mintime1}_x(cp1_y) < \text{mintime1}_y(cp1_y) \right) \lor \\
\left( \text{mintime1}_x(cp1_x) < \text{delay}_y(cp1_x) \right)
\]

Versions of the x travel-time curve corresponding to each of the disjuncts are labeled 1, 2, 3, respectively in Figure 4.7(b).

The most straightforward method to find whether the \( \text{mintime1}_x \) curve is intersected for any final velocity magnitude in the range \([0, v_c]\) would be to search through temporary target velocity values. However, a much simpler test almost always indicates this condition. First, check whether the intersection criterion is met for target velocity \( v_f \) chosen so that its x-component is equal to the x-component of the initial velocity, or if that value would exceed \( v_c \), then for a final velocity corresponding to \( v_c \). If not, then check if the intersection criterion is met for \( v_f = 0 \). If the intersection criterion is not met for any of these cases, then assume that there is no final velocity magnitude so that the open-space trajectory to the temporary target state is reasonable.

The justification for using this check is that when \( v_f \) is chosen so that its x-component is equal to the x-component of the initial velocity, critical acceleration \( a_{crit1} \) in x is zero. Hence all acceleration is allocated to the y-dimension and y-travel-time is minimized for this \( v_f \), increasing the likelihood that the travel time in y is less than the corresponding
travel time in x (so the curves intersect at the mintimel portion of the x travel-time curve). Choosing \(v_f\) magnitude in this fashion may fail to detect a potential intersection with the x mintimel curve if it positions \(cp_{1x}\) between \(cp_{1y}\) and \(cp_{2y}\). The additional check with \(v_f = 0\) is added to handle this case. With this value, there is no gap between \(cp_{1y}\) and \(cp_{2y}\). These two checks are not guaranteed to detect all cases where some value of \(v_f\) magnitude causes an intersection at the x-mintimel curve. However, experience indicates that it almost always works in practice. It is not critical to detect all such cases because, as discussed in section 4.4.7, there are graceful ways to deal with robot states that do not allow a reasonable open-space trajectory to the temporary target.

The same procedure can be used to check if the y mintimel curve is intersected by switching the roles of the x and y dimensions.

4.4.6 Picking a Fast Target Velocity

Having found one value for target velocity \(v_f\) that results in a reasonable trajectory, we wish, finally, to find a value that minimizes travel time to the temporary target position subject to the velocity constraint, \(v_c\). Recall that as the value of \(v_f\) changes, the shape of both travel time curves change, consequently changing the intersection point and the travel time of the corresponding control. The travel time is a function of \(v_f\), and the problem of minimizing travel time could be treated using standard minimization techniques. However, each evaluation of this function involves the iterative procedure of finding the intersection point of the travel-time curves, so a standard minimization approach would be quite expensive. It’s really not worth spending a lot of resources on this optimization problem for several reasons. First, \(\frac{\partial t}{\partial v_f}\) typically becomes small in the neighborhood of the minimum, so there is little added benefit to finding the true minimum over finding a value of \(v_f\) in the rough
neighborhood of the minimum. Second, a new temporary target may be generated at each time step, so the control will seldom be followed all the way to the temporary target. For example, by the time the robot gets in the neighborhood of the temporary target there will no longer be a potential collision with the obstacle and the temporary target will be moved far ahead, reducing the influence of target velocity on performance.

Given the value of \( v_f \) that has been found to work, a simple search procedure can be used to improve it. The partial derivative of travel time with respect to final velocity is negative for all sections of the travel time curves but mintime2 and backtrack control. If the intersection involves either of these two types of control, then increasing \( v_f \) may either increase or decrease the travel time. In this case, leave the value of \( v_f \) as it is. If the intersection does not involve either one of these types of control, both curves move downward with increasing \( v_f \), and the travel time at the intersection point will also decrease. Thus, there is an opportunity to reduce the travel time further by finding the maximal \( v_f \) subject to these two constraints:

1. The curves still intersect in control regions other than mintime2 and backtrack.

2. The reasonableness conditions are met.

The algorithm performs a coarse bisection search for a value of \( v_f \) that minimizes travel time while obeying the reasonableness restriction. The possible values for \( v_f \) range from the value of \( v_f \) that has been found to work up to \( v_c \). If \( v_c \) is ok (the two constraints above are met), then stop and return that value. Otherwise, the \( v_f \) that works is an initial lower bound and \( v_c \) is an upper bound. The lower bound is always known to be ok while the upper bound is not. Check the point midway between the lower bound and upper bound. If it is ok, it becomes the lower bound, else it becomes the upper bound. Quit when the difference between the upper and lower bounds are less than some threshold distance. Since it is not important to get an optimal value, experience indicates that it is adequate to set the threshold at 20% of the difference between the initial lower bound and \( v_c \), thereby restricting the bisection search to no more than three iterations.

4.4.7 When Reasonableness Restrictions Are Not Met

The reasonableness restrictions that constrain control in one dimension or another to be mintime1 can fail in one of two ways. First, mintime1 travel in the constrained dimension may require more acceleration than \( a_{max} \). Second, mintime1 travel in the constrained dimension may be too fast relative to the other dimension (the mintime1 curve does not intersect the travel-time curve of the other dimension). These failures are handled differently depending on whether the x-component of the robot’s initial velocity is toward the temporary target position and the x-dimension control is required to be mintime1, or the x-component of initial velocity is away from the temporary target and control in the y-dimension is constrained.
Figure 4.8: Cases where there is no reasonable control to the temporary target. The top row illustrates two failure cases where the x-component of the initial velocity is away from the goal. Panel (a) shows a situation where the critical acceleration in y is greater than \( a_{\text{max}} \), as illustrated in panel (b). Panel (c) shows a situation where using mintimel control in y always gets the robot to the y-component of the temporary target state before reaching the x-component of the temporary target state regardless of the choice of \( v_y \) magnitude. Panel (d) illustrates that the mintimel curve in y lies below the x travel-time curve. The second row similarly illustrates these two failure cases when the x-component of initial velocity is toward the goal. \( r, h, e, t \) label the robot, hit point, edge point, and target positions, respectively.

**Case I** First, consider the less common case that the x-component of initial velocity is directed away from the temporary target. In this case, control in y is required to be mintimel.

A situation where mintimel control in y requires more acceleration than \( a_{\text{max}} \) is illustrated in Figure 4.8(a) & (b). Ordinarily this type of condition would not arise since the temporary target at the prior time step was safe. However, it could conceivably arise when the current temporary target is positioned significantly closer to the robot than at the previous time step. This could happen, for instance, if what appears to be a gap between obstacles in the earlier time step proves to be a concavity in one obstacle. While the temporary target was previously positioned within the concavity, once the end of the concavity is sensed the temporary target will be placed back outside the concavity, closer to the robot. Though the robot cannot safely reach this target, the trajectory to the previous temporary target is guaranteed to be safe, and if the previous temporary target had nonzero velocity, then there is adequate space ahead for the robot to decelerate to a stop. Thus, the robot can pursue the default strategy of traveling to the previous temporary target and then decelerating to a stop if the temp target velocity is nonzero. In the example in Figure 4.8(a), the robot would come to a stop within the concavity.
A situation where mintimel travel in $y$ is too fast relative to the travel time in $x$ is illustrated in Figure 4.8(c)&(d). This situation is unusual, but can arise when the robot has had to move away from the goal to avoid a prior obstacle, or the robot has just begun pursuing a new goal. Using the slowest mintimel control in $y$ would nonetheless drive the $y$-component of the robot’s state to the target before the $x$-component of the target is reached. Using a slower control in $y$ could result in a looping trajectory. This looping behavior is unnecessary because there is no need for the robot to pass right through this edge point – it can just as well pass further from the robot, so a different temporary target position should be chosen. The algorithm does so by choosing control in each dimension corresponding to $cp_{1y}$ for a target velocity of zero. Applying this control would result in the robot reaching the $y$-component of the target state before the $x$-component is reached. If the $x$-component of the robot’s velocity at this point is in the direction of the edge point, then choose the robot’s state at this point as the new trajectory target. Otherwise, pursue the default strategy.

**Case II** More frequently the $x$-component of initial velocity is toward the target. In this case, it is again possible to have a condition for which mintimel control in $x$ requires more acceleration than $a_{max}$, as illustrated in Figure 4.8(e)&(f). When this condition holds, even if all acceleration is allocated to the $x$-dimension it will not be enough to bring the robot to a stop in $x$ before crossing the obstacle boundary. This condition might arise for reasons similar to those in the $y$ case, and the algorithm again deals with it by pursuing the default strategy.

A situation where mintimel travel in $x$ is too fast relative to the travel time in $y$ is illustrated in Figure 4.8(g)&(h). This condition means that using the slowest possible non-backtracking control in $x$ (the minimum-time control that results from using the critical acceleration in the $x$-dimension), the robot still gets to the $x$-component of the temporary target faster than it can reach its $y$-component. Using a slower (non-mintimel) control in $x$ could cause the robot to cross the obstacle boundary line. This condition will typically arise due to sensory horizon effects, as described in the next section. This situation can be treated by using $a_{critx}$ in $x$ to travel to the $x$-component of the temporary target position. This will take the robot to a position on the surface of the obstacle boundary with zero velocity component normal to the boundary surface. Simultaneously, use this time to travel as far as possible to the temporary target in $y$. Choose the resulting state as the new temporary target state.

The procedure can be summarized as follows:

1. Find the critical acceleration to the initial temporary target in $x$ ($a_{cx}$) and the corresponding travel time ($t_{cx}$).

2. Find the minimum-time control to the initial temporary target in $y$, assuming a maximum $a_{y}$ acceleration corresponding to $a_{cx}$ ($a_{y} = \sqrt{a_{m}^2 - a_{cx}^2}$).
3. Apply $a_y$ for time $t_{cx}$. The resulting state is the $y$-component of the new temporary target.

4. The $x$-component of the new temporary target is $x = 0$, $v_x = 0$.

Thus, the temporary target state selected will be located on the obstacle boundary, with velocity directed along the surface of the boundary.

This procedure will allow us to find a control that doesn’t cross the obstacle boundary as long as $a_{cx}$ is less than $a_{max}$. Another problem could arise if $a_{cx}$, although less than $a_{max}$, is close enough to it that there is very little acceleration available for allocation to the $y$-dimension. If the initial $y$-velocity is directed away from the initial temporary target, then after reaching $x = 0$ the $y$-component of velocity could still be directed away from the temporary target. While the robot would not have crossed the obstacle boundary, it may yet collide with an obstacle. For it may land on a portion of the obstacle boundary that is not between the hit point and the initial temporary target, the region over which the obstacle boundary provides an outer bound for the obstacle. (For a convex obstacle, the obstacle must lie entirely on the other side of the boundary.) Neither of these cases should arise, however. At every time step, the robot is guaranteed to be in a safe state (it can come to a stop within known free space) so the robot has adequate acceleration to avoid colliding with an obstacle. However, stopping at the obstacle boundary provides a slightly more stringent requirement (the robot could cross the obstacle boundary and still avoid collision). It would be more complex to provide a guarantee that the robot must always have adequate acceleration to cross an obstacle boundary, so we can maintain the safety guarantee by adding the following rule: if the $y$-position upon reaching $x = 0$ is not between the hit point and the initial temporary target selected at this time step, or the $y$-component of velocity upon reaching $x = 0$ is directed away from the initial temporary target selected at this time step, then continue to travel to the (safe) temporary target from the prior time step.

4.4.8 Sensory Horizon Effects

The failure to find a reasonable open-space trajectory to a temporary target because $\text{min}t_{\text{time}1}$ control is too fast will typically arise when the view of an obstacle at a new time step reveals an obstacle or portion of an obstacle of considerable extent that was previously unseen. An example of this situation is illustrated in Figure 4.9. At time $n$, no obstacle is within sensor range, and edge point $e_n$ has been chosen at the edge of the sensor array (and a zero-velocity temporary target assigned at this position). At time $n + 1$, the obstacle is within view, and edge point $e_{n+1}$ is chosen in the vicinity of one of the obstacle edges. The minimum-time travel to the edge point in $x$ is quite short because of the large component of velocity in this dimension, while travel time in $y$ is significantly longer, as illustrated in the travel-time diagram of Figure 4.9(b). Hence edge point $e_{n+1}$ is unreachable, and a temporary target position will be assigned somewhere between the hit point and $e_{n+1}$. 

60
In the extreme case that at time $n$ the obstacle is just beyond sensor range, then $e_n$ will be positioned at the surface of the obstacle. When the obstacle is revealed, all available acceleration may be needed simply to bring the robot to a stop at $e_n$. When the robot comes to a stop or near-stop in this fashion, its performance will suffer. This condition can be ameliorated by choosing temporary targets some distance away from the sensory horizon. Then even in the worst case that an obstacle is just beyond sensor range, this separation of the target from the sensor edge ensures that the robot will never have to come to a dead stop at the temporary target.

There is, of course, a tradeoff involved in the choice of this distance. Choosing a large distance will reduce performance by restricting the robot’s velocity when there are no obstacles ahead, while choosing a small value may mean that the robot will need to come to a near-stop when it encounters obstacles. Experiments indicate that a distance in the neighborhood of 10% of the sensor radius markedly improves the smoothness of the robot’s dodging response to a new obstacle while reducing the maximum velocity by approximately 5%.

4.4.9 Rechecking for Collision

The temporary target selection process was carefully designed to ensure that the trajectory to it avoids collision with the obstacle with which collision was originally detected. However, collision avoidance is not guaranteed under several circumstances. When there is no obstacle detected, and a zero-velocity temporary target is selected at the intersection of the open-space trajectory to the goal and the edge of the sensor array, the shape of the open-space trajectory to the zero-velocity temporary target will differ somewhat from the original open-space trajectory to the goal. Thus, the non-collision of the original trajectory doesn’t guarantee the non-collision of the trajectory to the temporary target. It is also possible that the change in the trajectory to avoid the obstacle with which collision was originally detected may move the trajectory in the path of another obstacle.

To ensure that the trajectory to the temporary target is safe, the collision-checking procedure can be re-employed. If a collision is found, the dodging procedure can be re-
applied recursively. At any point the procedure can be timed out and the default strategy of pursuing the prior, guaranteed safe trajectory target can be employed.

### 4.4.10 Uncertainty in Control Outcomes

The dodging controller ensures non-collision by establishing that the projected trajectory to the temporary target will be collision free. However, as noted earlier, mobile robots are susceptible to positioning error, so the robot may not follow the projected trajectory exactly. As noted in Section 2.3, the robot’s maximum positioning error over some movement can be modeled as being proportional to the distance of the movement. Therefore, the non-collision guarantee can be provided despite positioning error by growing obstacles by an amount corresponding to the maximum error that will accumulate by that position. Then, when the projected trajectory among the grown obstacles is collision-free, the actual trajectory will also be collision-free, even if the the maximum level of positioning error occurs. The “maximum level of positioning error” refers to the maximum level of displacement error at a chosen likelihood level.

Obstacles are grown to account for positioning error as follows. In the current implementation, environment surfaces are represented using line segments. The displacements from the robot to each environment segment endpoint within sensor range are computed, and the movement error distributions corresponding to each displacement are computed, as described in Section 2.3. An equiprobability contour ellipse is derived for each distribution at the chosen certainty level. For each segment, the sensor array can be filled to cover the area that the grown segment lies within and its corresponding occlusion region, as shown in Figure 4.10.

### 4.5 Conclusion

This chapter has presented the dodging controller for reactive navigation. The method uses the approximation to the open-space minimum-time control developed in the preceding chapter in combination with an algorithm for choosing temporary targets to direct the robot around locally sensed obstacles. An example showing the performance of the dodging controller in a cluttered environment is shown in Figure 4.11. The nearest competitor to the dodging controller is the reactive navigation system described by Feng and Krogh [1990]. To support the claim that the dodging controller is capable of higher performance than Feng and Krogh’s system, the dodging controller was applied to the same example problem that was presented in Feng and Krogh’s paper. The parameters for this example are as follows:

- Time step: 1.0 s
- Maximum velocity: 8.0 units/s
- Maximum acceleration: 1.0 units/s
Figure 4.10: The non-collision guarantee is maintained despite positioning errors by growing obstacles. The distribution in position outcome is calculated for a movement to each endpoint of environment segments within sensor range. An equiprobability contour ellipse is derived for each distribution at the chosen certainty level. A grown version of the environment segment and its occlusion region can then be drawn as shown, where \( r \) labels the robot position.

- Zero positioning error

Using the dodging controller, the travel time in this task was 84 s, a 10% reduction compared to Feng and Krogh's reported travel time of 93 s. The dodging algorithm's trajectory is illustrated in Figure 4.12.

In the next chapter, the discussion turns to guiding the dodging controller using an approximate map of the environment.

Figure 4.11: The trajectory of the simulated robot under control of the dodging algorithm. The robot is traveling from the initial position at the left of the figure to a goal at the right. The simulated robot had maximum acceleration of 3.0 m/s\(^2\), maximum velocity of 5.5 m/s and a sensor range of 8 m.
Figure 4.12: Comparing performance of the dodging controller with the performance of Feng's reactive navigation system. Travel time = 84 s. (Feng's reported travel time: 93 s.)
Chapter 5

Map-Guided Obstacle Avoidance Control

While the previous two chapters have described a method for high-speed reactive navigation using only local sensor information, as noted in the introduction a complete navigation system must make use of a map of the environment. When the robot is operating in large-scale space, some kind of map is necessary if for no other reason than for the robot to know about locations that it might want to go to beyond sensor range. Even supposing that the robot is supplied with the coordinates of a goal location in the deus ex machina fashion presumed in the problem statement of reactive navigation, a reactive navigation system alone is not an adequate solution to the high-speed navigation task. Without a map, the reactive navigation system may lead the robot down dead ends or otherwise along an indirect route to the goal. Due to positioning errors, it is not adequate to express a goal simply as a coordinate in some frame; if the robot is to travel over a considerable distance, by the time the robot gets in the vicinity of the goal, its estimate of the goal position may be separated from the intended goal by a wall. Moreover, without a map, the robot cannot take advantage of the opportunity for higher-speed performance afforded by global knowledge of the environment.

While having a map is essential to navigation in large-scale space, map creation is complicated by mobile robots' poor positioning accuracy. Since position error accumulates, knowledge of the relative position of distant points is uncertain, so local obstacle measurements cannot simply be inserted into a global coordinate frame.

Due to inevitable map inaccuracies, control and estimation errors, and possible changes to the environment, the robot cannot rely on map information alone for guidance. It must also be able to respond to current sensor information to ensure obstacle avoidance.

This chapter discusses two topics. The first is a method for environment exploration and map acquisition developed by Kuipers and Byun [1988; 1991; Byun, 1990]. The method develops a topologically correct, metrically approximate map of the environment in a way that is robust in the face of positioning errors. The basic map generated by this method
consists of a topological description of the environment and approximate measurements of the relative displacement of "distinctive places" in the environment. The second topic is how to use this basic map in combination with the dodging controller reactive navigation method for high-speed navigation. The robot makes use of the approximate information in the map to travel a direct route to the goal, while responding to current sensor information to compensate for control, estimation, and map errors and guarantee non-collision. Subsequent chapters will discuss enhancements of the basic map which enable the robot to take advantage of global map information to improve performance.

5.1 Qualitative Exploration and Mapping

Early work in exploration and mapping chose the approach of analyzing sensor input to create a metrically accurate map of the environment and subsequently deriving a topological description of the environment from it. Here, I adopt a qualitative exploration and mapping method which instead discovers qualitative features of the environment to directly create a topological description of the environment with little reliance on error-prone metrical information. This method, developed by Kuipers and Byun [1988; 1991; Byun, 1990], describes the environment in terms of qualitative concepts arising from the robot's sensor-motor interaction with the environment: distinctive places and distinctive paths. Metrical information can be added to this topological framework. This is a robust method for deriving a topologically accurate, metrically approximate map of the environment.

The central element of the qualitative map is the topological network description. The nodes of this network correspond to distinctive places, and links between nodes correspond to distinctive paths.

Distinctive places are locations in the environment in which the pattern of sensory information is distinctive in the sense that it is qualitatively distinguishable from the pattern of sensory information in the neighborhood of the location. A distinctive places can be defined as the location of a local maximum of one of a set of appropriately defined distinctiveness measures. In the implementation of the qualitative mapping strategy in the NX system described by Kuipers and Byun [1988; 1991; Byun, 1990], the distinctiveness measures defined on the robot's local range readings include the following:

- Extent of differences in distances to near objects.
- Temporal discontinuity in one or more sensors, experienced over a small step.
- Extent and quality of symmetry across the center of the robot or a line.

In the exploration shown in Figure 1.1, for example, distinctive places were found in each corridor corner at a location that minimized the distance differences to nearby obstacles.

During the mapping, as the robot moves through the environment it constantly monitors its sensory information. When the robot recognizes that it is in the neighborhood of a
distinctive place, it applies a hill-climbing control strategy to reach the distinctive place at
the local maximum of the distinctiveness measure. If this is a previously unseen distinctive
place, a node in the topological network is created, and connecting paths from the distinctive
place are recorded, along with the places that they connect to, if known.

Distinctive paths, corresponding to the links in the topological network, are each defined
by the trajectory that the robot takes when it applies a local control strategy. Examples of
the local control strategies used in the NX system include:

- Follow midline.
- Follow along an object to the right.
- Follow along an object to the left.

For example, in the exploration trace shown in Figure 1.1, the follow midline control strategy
is used to travel along corridors. During exploration, when the robot is at a distinctive place,
an appropriate local control strategy is chosen to move the robot in an open direction. During
travel, the robot monitors its sensor information for evidence of features indicating that it
is in the neighborhood of a distinctive place. When the execution of the control strategy
is interrupted by the discovery of a distinctive place, an edge in the topological network is
created, linking this distinctive place with the distinctive place at the start of the path, along
with a description of the local control strategy used.

The flow of control of the exploration strategy is described by the diagram in Figure 5.1.
The typical sequence of operation initiates with the moving-toward-open-space behavior, and
proceeds through the states in the state diagram in a clockwise fashion. The robot analyzes
its sensor input to detect regions of open space, and on the basis of this analysis chooses
a direction to move in and a local control strategy to employ. The local control strategy
is used to follow along the path until the robot detects that it is in the neighborhood of a
distinctive place whereupon it performs hill-climbing in the distinctiveness measure to reach
the distinctive place. The robot chooses a direction to explore from this distinctive place and
the sequence repeats. The other transitions in the diagram deal with exceptional conditions
such as regions with multiple overlapping distinctiveness measures, and initial choices of local
control strategies or distinctiveness measures which fail to pan out because the conditions
for them were incorrectly recognized.

The direction to be chosen once the robot reaches a distinctive place is determined by
the contents of the exploration agenda. When the robot reaches a new distinctive place,
one or more (Place, Direction) pairs are pushed onto the agenda, corresponding to paths
in directions from this place that have not yet been explored. The area to explore next is
generally chosen from the top of the agenda. Agenda entry (Place1, Direction1) is deleted
from the agenda when the robot either visits Place1 and leaves in direction Direction1 or
returns to Place1 from the direction opposite to Direction1.

To ensure that the robot arrives at a correct topological description of the environment,
it must be able to recognize a previously visited place, and recognize a previously unseen
Figure 5.1: The flow of control of the exploration strategy is described by this state-event diagram. Typical operation of the exploration strategy cycles through the states in a clockwise sequence. Exceptional conditions are handled by the other transitions in the diagram.

place as new. Places that have similar geometry cannot be distinguished on the basis of local information alone. If the local range information at a distinctive place is a sufficiently close match to the stored range image at a previously seen place, then the robot should test the hypothesis that the current place is the same as the previously recorded place. It can do this by comparing adjacent regions of the environment with the description of the environment adjacent to the previously stored place. This capability is implemented by attempting to travel to a place in the map adjacent to the previously seen place, and then returning. If this route is performed as predicted, then the robot concludes that the current place is the same as seen before. Otherwise, the robot concludes that it is at a new place. The robot may be fooled about the identity of a place using this particular procedure if there are extended regions of the environment that are sufficiently similar. The likelihood of being fooled can be reduced by increasing the depth of exploration of adjacent places, or false positive matches can be eliminated altogether if there is a globally unique reference place using the methods of Dudek, et al. [1988].

The basic outcome of the exploration of the environment is the topological network description of the environment. The nodes of this network correspond to the distinctive places discovered during exploration, and the edges correspond to the distinctive places between them. This basic topological description can be augmented with available metrical information. For the initial high-speed control, the only additional information that is assumed to be in the map is the approximate displacement between adjacent distinctive places, expressed as an approximate transform. This information is readily available from the range encoder data generated during the robot’s traversal of a path.

This “basic map” can be acquired at low cost in the sense that it does not require the
complex and expensive processing necessary to create a metrically accurate description of the environment. Yet, it is adequate for providing high-level direction to the high-speed reactive navigator. This high-level direction allows the robot to follow a direct route to the goal and avoid circuitous routes. Subsequent chapters will discuss methods for augmenting the information in the map for higher performance.

5.2 Outline of Map-Level Control

The approximate description of the environment in the basic map can be used in combination with reactive navigation to efficiently guide the robot to its goal. This guidance is performed by a map-level controller that uses map information plus a description of a route, or path to the robot’s goal, to guide the dodging controller by specifying a goal for the dodging controller at each time step.

A route consists of a sequence of subgoals: target states each consisting of a position relative to a distinctive place and a velocity. With the robot initially at some distinctive place, and given a goal place, the topological network can easily be searched to find a sequence of distinctive places that lead to the goal. A route is obtained by assigning a subgoal to each distinctive place, each initially positioned at the distinctive place and having zero velocity. Of course, there may be several sequences of distinctive places that lead to the goal. In this case, it would be desirable to choose the sequence that could be traveled most quickly. In general, choosing the best sequence would require computation of the minimum-time control using a complete model of the environment and robot dynamics. This is undesirable both because of the considerable computation involved and because a completely accurate model is not available. Using a simple heuristic selection procedure, such as choosing the sequence with the shortest estimated path length, will frequently find the best path. If there is ample opportunity for experimentation, several alternate routes can be tried to find the one that is fastest.

The map-level controller specifies a target for the dodging controller by performing two functions: selecting a subgoal on the route for the robot to follow, and keeping track of the position of this subgoal relative to the robot.

Supposing that the robot is initially at the first distinctive place on the route, the subgoal at the second distinctive place becomes the current subgoal, or the subgoal to be pursued by the dodging controller. An estimate of the relative position of this subgoal can be computed from the map, and updated as the robot moves toward it. The map-level controller monitors this estimate, and when the robot gets within a threshold subgoal switch distance of the current subgoal, the next subgoal on the route is reassigned to be the current subgoal. This process of reassigning the current subgoal is repeated until the robot reaches the goal.

The estimated position of the current subgoal is represented using the approximate transform (AT) representation discussed in Section 2.4. With the robot at the first distinctive place on the route, and the current subgoal near the second distinctive place, the initial
estimate of the relative position of the current subgoal is obtained by compounding the map entry describing the estimated position of the second distinctive place relative to the first distinctive place with the estimated position of the current subgoal relative to its nearby distinctive place. After one time step, the robot has an estimate of its motion over the time step from wheel encoder data, expressed as an AT. As shown in Figure 5.2, this AT can be combined with the subgoal position estimate using the compounding operation to arrive at a new, lower-accuracy estimate of the current subgoal’s relative position. As the robot proceeds toward the current subgoal, its estimated relative position is repeatedly updated in this fashion.

When the distinctive place at the current subgoal is within range of the robot’s distinctive place detector, the detector reading, compounded with the distinctive place to subgoal displacement, provides an independent estimate of the current subgoal’s relative position. As shown in Figure 5.2, this information can be merged with the prior estimate to obtain a new, higher-accuracy subgoal position estimate.

5.3 Summary

The qualitative exploration and mapping procedure described in this chapter can reliably develop a topologically correct description of the environment despite positioning errors. Using the approximate, limited information in the basic map, a map-level controller can provide global direction to the dodging controller, allowing the robot to effectively and directly navigate to a goal state.

An example of the performance that results from combining map-level control with the dodging controller is shown in Figure 5.3.

The robot is capable of higher performance if the information in the basic map is augmented, as discussed in the two following chapters. Chapter 6 describes storing approximate knowledge of environment geometry on the map to enable the robot to anticipate environment geometry that is beyond sensor range or occluded, and hence travel faster while maintaining a non-collision guarantee. Chapter 7 describes how the map may be annotated with control information that allows the robot’s control to more accurately reflect the global layout of the environment, improving overall performance.
Figure 5.2: The position estimate to the current subgoal is updated as the robot moves using the *compounding* and *merging* operations. In a), the robot is at Place 1, and its approximate knowledge of the relative position of its current target at Place 2 is represented as an approximate transform (AT), here displayed graphically as an arrow (indicating the nominal value of the estimate) and an ellipse (a constant probability contour of the error distribution of the estimate). In b), after moving part way down the corridor, an estimate of the robots' position relative to Place 1 is represented by another AT, and in c) these two AT's are *compounded* to obtain a (lower accuracy) estimate of the current position of Place 2 relative to the robot. As the robot continues down the corridor, this operation is repeated and the uncertainty in the robot's position estimate increases. But when the robot gets close enough to Place 2, as in d), its distinctive place detector provides an independent estimate of the position of Place 2. These two estimates are *merged* to get a higher accuracy estimate.
Figure 5.3: Combining the information in the basic map with the dodging controller allows the robot to travel directly and efficiently along the intended route. The subgoals have zero velocity and are positioned at distinctive places. The subgoal switch distance is 3 meters. The robot's sensor range is 6.4 meters, and its maximum acceleration is 2.5 m/s^2. The robot operates below its maximum velocity of 8.0 m/s due to the limited sensor range. The robot's position is shown at the beginning of each 0.5 s time step. Total travel time: 33.5 s.
Chapter 6

Representing Knowledge of Environment Shape

6.1 Introduction

The robot is capable of creditable performance with the basic map of the environment consisting only of a graph describing the connectivity of distinctive places and paths, and the approximate relative displacement of adjacent distinctive places. However, performance is restricted somewhat by basing collision avoidance solely on immediate range sensor input because open space that is occluded or beyond sensor range must be treated as occupied to guarantee non-collision. If, on the other hand, the robot had knowledge of the layout of obstacles in these unseen regions, it would not have to restrict temporary targets to the visible region, and could choose temporary targets further ahead and hence be able to travel more rapidly. Because the robot’s estimates of its movements are subject to error, its knowledge of environment geometry will be uncertain, and a representation of environment geometry must be capable of expressing this uncertainty.

This chapter describes a simple and general method for representing and applying the robot’s knowledge of environment geometry. Information about environment geometry along a path is stored as a sequence of range readings, each annotated with approximate transforms to adjacent distinctive places. To use this information to guide the robot, stored range readings from the region of interest can be combined to generate an approximate description of occupied regions.

The following sections describe how geometric information is stored and used.

6.2 Storing Knowledge of Environment Geometry

As the robot travels between a pair of distinctive places, it performs range sensing at each time step, providing information about the geometry of the environment along the path. This
Figure 6.1: As the robot travels through the environment, it stores local range information in data structures called “panes”. Also stored on each pane is a description of the robot’s position where this range information is recorded in terms of the approximate relative positions of adjacent distinctive places.

Geometric information is represented simply by storing a subset of these range images along with information indicating the estimated position of the range image along the path. The process is illustrated in Figure 6.1 which shows a trajectory of the robot and a sequence of the panes recorded during part of the trajectory. Each range image and associated position information is stored on a data structure called a “pane”, and the panes are stored on a doubly linked list in the same order that they are recorded.

As the robot travels along a path, the current range image is stored when at the next time step the robot will have traveled a distance exceeding some threshold value since the last range image was stored. Therefore, this threshold value provides an upper bound on the separation of two adjacent panes. The choice of an appropriate threshold value is influenced by two considerations: a smaller value will use more storage, while a larger value may fail to record some environment features.

There are two factors to take into account in evaluating the coverage provided by a range image spacing. The first is the minimum width of the region along the robot’s path entirely covered by at least one range reading, denoted $W$ in Figure 6.2(a). The second is the amount of the covered region that is covered by range readings from at least two positions. A range image of an obstacle from a single viewpoint may have a significant occlusion region, as shown in Figure 6.2(b). A view of the obstacle from two vantage points can often significantly
Figure 6.2: There are two measures of the coverage provided by a given spacing of range images. The first is the minimum width of the region along the robot’s path entirely covered by at least one range reading, denoted $W$ in panel (a). Second, it is desirable to have considerable overlap in the range images, because combining views of an obstacle from several positions can considerably reduce the occlusion region, as shown in panels (b) and (c).

reduce the size of the occlusion region as shown in Figure 6.2(c), providing a more accurate representation of environment geometry. As the figure illustrates, a maximum range image spacing value of half the sensor range provides a reasonable level of coverage.

When a range image is to be stored on a new pane, in addition to recording the range image, some approximate position information is added to the pane:

1. Approximate transforms to adjacent subgoals.

2. Approximate transform to the previous pane (prior pane AT).

It is useful to store approximate transforms (ATs) to both the current subgoal and the previous subgoal, because the error level in each estimate varies as the robot moves between the two subgoals. When the robot recalls a stored pane, the AT from the robot to the pane origin will be given by compounding the AT from the robot to a subgoal with the AT from that subgoal to the pane. Storing the AT’s from a pane to both current and previous subgoals allows the option of computing the robot-pane AT via the subgoal that results in the lower error.

The prior pane AT is stored on the current pane because it is useful for reducing the error level in the target position estimates for some of the panes. A typical example of the sequence of the error level in a robot’s estimated position to some distinctive place is illustrated in Figure 5.2. Starting at place one, the robot’s estimate of the relative position of place two is based on information from previous traversals, and has a moderate level of error. As the robot travels toward place two, estimates of the robot’s movements are compounded with the initial estimate, and the error in the estimated position of place two accumulates, becoming relatively high. Then, when the robot gets close enough to sense place two, the error level in the estimate drops dramatically. Panes recorded before the robot got within sensor range of place two are recorded with a low-accuracy estimate of the relative position of place two. The accuracy of these estimates can be improved by propagating the accurate measurement to place two back to previous panes using the AT to the previous pane stored on each pane.
Once the robot gets within sensor range of place two so that it has a low-error estimate of the position of place two, the error level of ATs to place two stored on previously recorded panes can be reduced by using the low-error estimate to compute a new AT between a previously recorded pane — call it the pane of interest — and place two, and merging the result with the old value for the AT to place two. The new AT to place two is computed as follows. An AT from the last pane recorded to the pane of interest is computed by compounding all prior pane ATs between the intervening panes; reversing the sign of components of the mean vector results in an AT from the pane of interest to the last pane recorded. Compounding this AT with the AT from the last pane recorded to the current position and the AT from the current position to place two yields the new AT from the pane of interest to place two. This AT can be merged with the AT to place two stored on the pane because the two AT’s are based on independent measurements.

6.3 Using Stored Knowledge of Environment Geometry for Obstacle Avoidance Control

When the robot is traveling through the environment, it can make use of the information stored on the panes to anticipate geometry that is beyond sensor range or occluded. This is done by combining current range sensor input with the stored information to create an occupancy grid that covers the region of interest. The region of interest will generally be the region that the robot is heading toward, out to some distance from the robot. Since the dodging algorithm always chooses a control that allows it to stop within the free space of the occupancy grid, the radius chosen for this region will determine the robot’s maximum safe velocity.

The locations of the panes relative to this grid are known only approximately, of course, so an obstacle recorded in a pane is “grown” so that the obstacle will fall within the occupied region with a chosen level of likelihood. The occupancy grid is then organized into a quadtree that can be used for detecting potential collisions in the same manner as the occupancy quadtree created from local range input. In this application, this occupancy information is used for planning the robot’s control, so I’ll refer to it as the “plan ogrid”.

6.3.1 Generating the Plan Ogrid

To create this occupancy grid, the first step is to find the stored panes that each significantly overlap with the region of interest. The estimated position of a pane relative to the robot is calculated as follows. Compound the AT stored on the pane describing its position relative to the previous subgoal with the AT describing the robot’s current position relative to the previous subgoal to get an estimate of the position of the pane relative to the robot. Repeat this for the AT from the pane to the current target to get a second estimate of the relative
position of the pane and robot. Merge these two estimates to get the best possible estimate of the position of the pane relative to the robot.

Once the applicable panes are found, they should be combined with the current sensor information to create a plan ogrid. The procedure to create the plan ogrid must address two issues:

1. How to deal with the error in the pane’s estimated positions.

2. How to combine overlapping panes to create a meaningful plan ogrid.

Since the pane’s estimated position relative to the plan ogrid is described by an AT, the pane’s position error can be handled by growing the occupied regions of a pane so that when it is drawn on the plan ogrid, the actual obstacle position will fall within the grown occupied region with a chosen level of likelihood (to the extent that the AT reflects the actual distribution of the relative location of the pane).

The rules for combining information from panes where they overlap should take into account several features of the panes’ occupancy information. The range reading from a single position may include a shadow region (a region occluded by an obstacle surface) that is not occupied. But a view of the region from another position may reveal that the region is not occupied. Moreover, one pane (or the current range input) may provide a lower-error view of some obstacle than another pane. For both of these reasons, where panes or the current range sensor image overlap, an area of the occupancy grid should be marked as occupied only if it is marked as filled in both panes.

The procedure for creating the plan ogrid from current sensor and pane data can be outlined as follows. The plan ogrid is cleared, and the current sensor information written onto it. For each applicable pane, the occupancy data is grown to account for the error in its estimated position relative to the ogrid, and the resulting occupancy data is written onto an intermediate grid. The intermediate grid data is combined with the occupancy data already on the plan ogrid.

To speed up the generation of the grown occlusion region for a pane, a pane’s range information is not stored in pixel format, but is simply stored as a collection of line segments describing the environment surfaces within sensor range. The endpoints of the line segments are described by approximate transforms to indicate the level of range sensing error. The AT from the pane to the plan ogrid can be compounded with the endpoint AT’s, to obtain an AT describing the robot’s uncertain knowledge of the endpoint position. An equiprobability contour ellipse is derived for each endpoint at the chosen certainty level. For each segment, the intermediate grid can be filled to cover the area that the segment might lie within and its corresponding occlusion region as shown in Figure 6.3.

Once the grown occupancy data for a given pane has been drawn on the intermediate grid, it can be combined with the occupancy data already present on the plan ogrid. Overlapping regions of occupancy data can be handled properly by *anding* the values in the intermediate grid with those in the plan ogrid. Occupied regions are indicated by a value of 1; free space
Figure 6.3: Obstacles are grown to compensate for uncertainty in the position of panes relative to the plan ogrid. The endpoint of each environment segment stored on a pane is described by an AT which reflects the error level in the position estimate. An equal probability contour ellipse can be derived from the AT so that the actual endpoint position will fall within the ellipse with a chosen likelihood level. A grown version of the environment segment and its occlusion region can then be drawn as shown in the figure, where r indicates the robot position relative to the environment segment at the time the snapshot was taken.

by a value of 0. At the start of creating the plan ogrid, it is initialized to the value 1. Then anding the occupancy data will have the intended effect: a region in the plan ogrid will only be marked as occupied if all the overlapping panes in the region are occupied.

When the environment has narrow passages and the robot’s level of position uncertainty is high, the structure of the environment may be obscured – where in fact there is a passage, it may not be discernable in the planning ogrid because adjacent walls have been grown to the point that they merge together. To ensure that the structure of the environment within range of the planning ogrid can be discerned at all times, a separate bit plane is drawn on the plan ogrid whose occupancy values reflect the mean estimated position of environment segments in each pane. This can be done very easily by replacing the AT from the pane to the plan ogrid with a zero covariance AT with the same mean displacement.

The result of combining the panes shown in Figure 6.1 and organizing the information into a quadtree is illustrated in Figure 6.4. The darker shade of grey in the figure reflects the mean estimated position of environment segments; the lighter grey indicates the area occupied by the grown occlusion regions.

It is instructive to demonstrate the importance of fuzzifying the boundary of the range data as well as the obstacles themselves. An example of the problem that may arise if obstacle boundaries are not also fuzzified is illustrated in Figure 6.5. As shown in panel (a), a rectangular obstacle was just beyond sensor range when the range reading was recorded on a pane. On a subsequent traversal of the route, there is substantial error in the estimated
Figure 6.4: The three panes shown in Figure 6.1 plus a current range reading are combined to create an approximate representation of the layout of nearby obstacles beyond sensor range. The robot's position is at the center of this square. The black regions correspond to the mean estimated positions of obstacles. The grey regions are "grown" around the mean positions so that the actual position of the obstacles lie within these grey regions with 95% likelihood.
position of this pane, and its mean estimated position now encompasses the actual position of the rectangular obstacle, incorrectly indicating that the space occupied by the rectangular obstacle is free. Fuzzifying the obstacles in the range image and anding the resulting image to an array filled with occupied values results in the image shown by the shaded region in Panel (b). The actual positions of the obstacles are superimposed on the image, illustrating that the rectangular obstacle lies within the region marked as unoccupied. The correct approach is to mark as clear only the area that is clear for all positions of the pane within the error ellipse of the position estimate – the area within the dotted lines in panel (c). This can be done by marking the area of the pane outside this clear region as occupied when the pane is drawn on the intermediate grid. The resulting image is shown in panel (d).

6.3.2 Using the Plan Ogrid

Once the plan ogrid has been created, it can be used in place of the local range data quadtree for obstacle avoidance control. Since the plan ogrid represents the approximate layout of obstacles some distance beyond sensor range, the dodging algorithm can choose temporary targets beyond sensor range, allowing the robot to travel faster than if subgoals were restricted to lie within sensor range.

As long as the environment has not changed since the panes were recorded, the robot is able to travel at this higher velocity while still providing a probabilistic non-collision guarantee. The actual positions of obstacles within range of the plan ogrid will fall within the plan ogrid’s occlusion regions with the chosen likelihood. Since the dodge algorithm directs the robot so that the occlusion regions are avoided, the robot will avoid collision with at least the chosen likelihood. If, however, the environment has changed and the plan ogrid indicates that there is free space where in fact there is an obstacle, then non-collision is no longer guaranteed, although the robot may be able to dodge the obstacle once it appears within sensor range.

Although the layout of obstacles beyond sensor range is only known approximately, adding this knowledge allows temporary targets to be chosen beyond sensor range in many circumstances, significantly improving the robot’s velocity. When the error level is moderate and passageways between obstacles are not too small, the only effect of the position uncertainty of the stored geometry is that temporary targets are somewhat displaced, with a slight effect on the robot’s trajectory.

When the error level becomes fairly large or passageways are small, then the grown occupancy data should be treated slightly differently by the dodging algorithm because of several potential problems. The first is that growing occlusion regions to account for pane position uncertainty may obscure relatively narrow passageways. The second is that occlusion regions may be grown to envelop a subgoal position.

Since the plan ogrid includes information about the mean estimated positions of obstacles, the underlying structure of the environment is represented even when it is obscured by the grown occupancy data. So the dodging algorithm can detect two hit points: the point
Figure 6.5: To ensure that all obstacles are encompassed by occupied regions, both obstacle regions and pane boundaries must be fuzzified. Panels (a) and (b) illustrate the problem that can arise if pane boundaries are not fuzzified: in (b) the rectangular obstacle lies within the region that is marked as clear. Panels (c) and (d) illustrate the effect of fuzzifying the pane boundaries as well as the obstacles themselves: in (d), the rectangular obstacle now lies within a region marked as occupied.
where the open-space trajectory to the subgoal crosses into the grown occlusion region, and the point where it crosses into the mean occlusion region. If no collision is detected with the mean occlusion region, but a hit point is found at the grown occlusion region, then a zero-velocity trajectory target is placed at the grown occlusion region hit point. If two hit points are found, then edge points are found for the mean occlusion region and for the grown occlusion region. A threshold distance is chosen with a value equal to 2.0 times the half-width of the major axis of the error ellipse that determines the growth of obstacles in this region. If the grown occlusion region edge point is further than this threshold distance from the mean occlusion edge point, then it is assumed that a passageway is being obscured. A dodge point is found using the mean occlusion data, and the trajectory to it is checked for a hit point with the grown occlusion region. A zero-velocity temporary target is placed at this hit point.

When a subgoal position is buried within the grown occlusion region, this problem can be dealt with as follows. Once again, a threshold distance is chosen with a value equal to 2.0 times the half width of the major axis of the error ellipse that determines the growth of obstacles in this region. If the hit point is within the threshold distance of the subgoal, then a zero velocity temporary target is chosen at the hit point. Otherwise, a temporary target is chosen as usual.

In summary, adding stored knowledge of environment geometry can considerably improve the robot’s performance by allowing the dodging algorithm to place temporary targets beyond sensor range while still providing a non-collision guarantee. For example, using current range readings alone for obstacle avoidance, a robot with a sensor range of 6.4 meters took 33.5 seconds to travel along the route shown in Figure 5.3. Using stored knowledge of environment geometry to create a planning ogrid with radius 12.8 meters, the robot’s travel time improves to 28 seconds, as shown in Figure 6.6.

Performance is significantly improved despite position uncertainty in stored information because the open space ahead of the robot is uncluttered, allowing the robot to safely place a temporary target well beyond sensor range. When position uncertainty becomes very large or passageways are small, the performance advantage may be diminished. The dodging algorithm can operate opportunistically, taking advantage of the stored knowledge when it is useful, and, in the worst case, reverting to performance based on current sensor information alone.
Figure 6.6: Using stored knowledge of environment geometry, the robot anticipates environment geometry beyond sensor range and travels more rapidly. In this example, a plan occupancy grid is created with a radius of 12.8 meters, double the sensor range. Maximum acceleration: 2.5 m/s². Maximum velocity: 8.0 m/s. Subgoal switch distance: 3.0 m. The robot’s position is shown at the beginning of each 0.5 s time step. Total travel time: 28 s.
Chapter 7

Improving Global Performance

7.1 Introduction

To achieve the highest level of performance possible, there should be a mechanism for improving the choice of subgoal states, for two reasons. First, subgoals are initially positioned at distinctive places with zero velocity without consideration of the demands of high speed navigation, and actually passing through intermediate distinctive places on the route unnecessarily slows the robot. Second, although the dodging controller is a high-performance reactive navigator, its performance is inherently limited by the locality of its operation. Modifying subgoal states provides a way to incorporate global information into the control.

Of course, as demonstrated in Chapter 1, the robot can avoid going all the way to the current subgoal simply by pursuing the next subgoal once it gets within some range of the current subgoal. One might imagine that performance can be optimized simply by choosing a subgoal switch range that is sufficiently large. However, the subgoal state influences the robot’s trajectory over the entire trajectory segment during which the robot is pursuing it, not just in the vicinity of the subgoal. For instance, in the example illustrated in Figure 7.1, the robot begins making an unnecessary movement in the positive y direction when it is midway between subgoal 1 and subgoal 2. Performance can be improved considerably by choosing a large subgoal switch range, but of course this relies on the dodging capability to direct the robot around obstacles. This poses a significant problem since there is nothing to prevent the dodging controller from going down a blind alley, as illustrated by the bold trajectory in Figure 7.1. Without the requirement that the robot reaches the proximity of the current subgoal before switching, there can be no assurance that the robot will follow the intended route.

Another drawback to relying on the dodging controller for obstacle avoidance is that the locality of the controller places certain restrictions on the selection of the temporary target that may reduce performance. The dodging controller cannot begin to direct the robot around an obstacle until the obstacle is within sensor range, whereas an optimal control strategy may respond sooner. Temporary target choice is restricted by the requirement that
Figure 7.1: With a moderate distance, $s_1$, chosen for the robot to switch from pursuing subgoal 2 to subgoal 3, the robot trajectory (indicated by dashed line) travels most of the way to subgoal 2, making an unnecessary move in the positive y direction. Choosing a very large subgoal switch distance $s_2$ prevents this movement, but also allows the robot to choose an incorrect path (bold dashed line).

its velocity must be aligned with the robot’s sightline to the edge of the obstacle, and in general, the dodging control cannot select the best control when it should be influenced by obstacles that are occluded.

Though the robot faces inherent performance limitations when it makes on-line control decisions on the basis of local information, it seems reasonable to expect that with experience the robot would be able to improve its control to reflect global factors. A simple way to do this is to modify subgoals. The robot’s control between subgoals is a fairly close approximation to the optimal open-space control. Travel along the robot’s entire route can similarly approximate optimality when the subgoals are optimized. The subgoals can relatively easily be optimized by incrementally updating them in a direction that reduces overall travel time. Where the robot needs to dodge obstacles, new subgoals can be introduced and subsequently modified so that obstacle avoidance reflects global factors instead of being based on an on-line, possibly suboptimal procedure.

7.2 Approach

The goal is to choose subgoals so that they minimize overall travel time while the open-space control between the subgoals remains collision-free. In order to represent obstacle constraints in the absence of an accurate, global model of environment geometry, it is adequate to represent local, critical regions of the environment and their approximate relative positions. In fact, it will be sufficient to represent the geometry of a set of one-dimensional slices of
the environment, each of which passes through a subgoal and is oriented perpendicular to
the direction of the robot’s trajectory at this point. I call the set of states consisting of all
velocities at positions along a slice of free space a frontier. In order for the robot to reach
the goal, it must pass through some state on each frontier. The idea is to find the state on
each frontier (subgoal) that minimizes the total travel time to the goal using a minimization
procedure.

Restricting the subgoal state to lie on the frontier prevents the minimization procedure
from moving the subgoal past an obstacle edge. When frontiers are positioned at the traject-
ory’s closest approach to an obstacle, restricting each subgoal to a position on its frontier
ensures that the open space trajectory between each subgoal will be collision-free – or nearly
so, which is adequate when the robot is directed between subgoals using the dodging con-
troller. The frontier construct allows the use of a simple one-dimensional penalty function
to keep the trajectory away from obstacles and avoids problems of local minima that can
arise with two-dimensional representations of geometry.

The procedure for improving subgoals is as follows.

1. Frontiers are initially established at each subgoal.

2. Using stored information about frontiers, their relative positions, and a model of the
robot’s open-space travel time, the subgoals are modified in a direction that improves
overall travel time.

3. The robot travels the route using the improved subgoals. If during the traversal the
robot needs to dodge obstacles, or passes within some threshold distance of an obstacle,
then proximity data is recorded.

4. Frontiers are updated so that they are still positioned at the trajectory’s closest ap-
proach to obstacles. If a dodge is needed in a region where there was no frontier
previously, then a new frontier is introduced.

Steps 2 through 4 may be repeated until there is no further improvement in overall travel
time. Due to model and control error, the optimization does not yield an exact “best” subgoal
configuration. But modifying subgoals in accord with the mean estimated model of environ-
ment geometry and robot dynamics considerably improves the robot’s overall performance.
These four steps are described in more detail in the following sections.

7.3 Frontier Initialization

A frontier is a three-dimensional surface in the four-dimensional phase space of the robot
consisting of the positions along a line segment and all possible velocities. Each frontier is
described by a data structure with the following elements:
Frontier
Origin Prev:
Origin Next:
Subgoal Displacement:
Subgoal Velocity:
Orientation:
Segment Length:

One of the endpoints of each frontier's line segment is distinguished as the origin of the frontier. The origin prev entry specifies the mean estimated position of the frontier's origin relative to the previous subgoal; the origin next entry is its mean estimated position relative to the next subgoal. The subgoal displacement is the subgoal’s distance along the line segment from the origin and the subgoal velocity is the velocity of the subgoal. Together, the subgoal displacement and velocity describe the subgoal state. The orientation entry specifies the orientation of the line segment. The segment length entry indicates the length of the line segment.

Once the basic map has been created and a zero velocity subgoal created for each intermediate distinctive place along the chosen route, a frontier is established for each subgoal. Each frontier’s line segment includes the subgoal’s position, and its orientation is chosen to be midway between the orientation of two rays: the first from this subgoal to the predecessor subgoal; the second from this subgoal to the successor subgoal. The endpoints of the segments are defined where a line with this orientation intersects visible obstacle surfaces.

7.4 Subgoal Improvement Step

Once the initial frontiers have been established at each distinctive place, the algorithm steps each subgoal in a direction that reduces route travel time. A single improvement step is not intended to arrive at the best subgoal states, but simply move each subgoal to a somewhat better state on the corresponding current frontier. Repeated applications of the subgoal improvement step, route traveling, and frontier updating sequence are performed to arrive at the best subgoal assignment.

A limited subgoal improvement step is performed for a given set of frontiers because frontiers may not initially capture all relevant environment features; making large changes to the subgoals could mean that open-space trajectories between the subgoals will not necessarily be collision-free. Rather than attempting to internally simulate the consequences of a change in subgoals using models of environment geometry and robot dynamics, the environment is used as its own model: after making moderate changes to the subgoals, the route is traveled again, the proximity to obstacles will be measured, and frontier positions updated as necessary.

This subgoal improvement step is implemented by performing a limited number of steps of gradient descent in an objective function. The first component of the objective function
is the total travel time along the route, which can be expressed as the sum of the times for travel between adjacent subgoals:

\[ T = \sum_{i=1}^{N-1} time(i, i+1) \]

Here, \( N \) is the total number of frontiers, and \( time(i, j) \) refers to the open-space travel time between frontier \( i \) and frontier \( j \), a function of the velocity and displacement coordinates along both frontiers: \( v_{i1}, v_{i2}, d_i, v_{j1}, v_{j2}, \) and \( d_j \). Thus the derivatives of the total time, \( T \), are given by

\[ \nabla_k T = \nabla_k time(k-1, k) + \nabla_k time(k, k+1) \]

where \( \nabla_k \) is the vector of partial derivatives with respect to the coordinates of frontier \( k \).

To prevent the gradient descent procedure from moving the subgoal position past the edge of the frontier, a penalty term for each frontier is added to the objective function to prevent moving the position of the subgoal past the edge of the frontier (and hence colliding with an obstacle). For frontier \( i \), the penalty term is given by

\[ penalty(i) = \begin{cases} 
A e^{-\kappa \cdot \text{dist}(d_i)} & \text{if } \text{dist}(d_i) < R \\
0 & \text{otherwise}
\end{cases} \]

where \( A, \kappa, \) and \( R \) are positive constants, and

\[ \text{dist}(d) = \min(d, \text{segment length} - d) \]

indicates the distance to the edge of the frontier nearest the displacement \( d \).

While the derivative of the penalty term of the objective function can be easily derived, finding the derivative of the travel time term is slightly more complicated. This travel time term, \( time(i, i+1) \), should be the time for the robot to travel between frontiers \( i \) and \( i+1 \) using the open space control algorithm described in Chapter 3. The derivative of the travel time for this algorithm cannot be derived analytically. In the present implementation, an approximation to this travel time that allows simple derivative computation is used for the \( time(i, j) \) term: the travel time with independent acceleration constraint \( \sqrt{2}a_{\text{max}} \) in each dimension. As discussed in Chapter 3, a solution to an instance of this problem can serve as a (suboptimal) solution to the problem with a magnitude constraint, since the accelerations it allows are completely overlapped by the accelerations allowed with acceleration constraint \( a_{\text{max}} \). Yet the travel time for the independent constraints problem can be easily calculated as the maximum of the one-dimensional travel times for the \( x \)-component and \( y \)-component of the problem. The one-dimensional travel times are given in Table 3.1. The tradeoff with this approximation is that the subgoals that minimize the objective based on this travel time will be less aggressive than if the true travel time were used.

Derivatives of the objective function are used by a gradient descent method to improve subgoals. A single gradient descent step evaluates derivatives of travel time with respect to
frontier coordinates at all subgoal states and increments subgoal coordinates by the product of the derivatives and a step size constant. A complete subgoal improvement step consists of a fixed number of gradient descent steps; modification of any given subgoal is halted when the next gradient descent step would exceed the chosen bound on a change to subgoal position or velocity in a single subgoal improvement step. Since derivatives of travel time with respect to velocity are generally much larger than derivatives with respect to position, a succession of steps along the gradient rapidly reach a near-optimal subgoal velocity, while the subgoal's position moves little; reaching the optimal position takes many more iterations. For improved performance, the stepsize for the position derivatives is increased by a factor of 15. Since there may be discontinuities in travel time as a function of initial and final states, some special treatment is needed to prevent moving to a state that increases travel time.

At the end of each subgoal improvement step, the frontier orientation is adjusted to be perpendicular to the subgoal velocity, unless the frontier was previously positioned to intersect a near obstacle point (as described in the next section).

7.5 Traveling Route with Frontier Update

After the subgoals have been modified offline, when the route is traveled again the frontiers will be updated on the basis of the robot's proximity to obstacles during the route traversal. Two types of modification may be performed: a new frontier may be inserted, or an existing frontier may be moved.

Insertion of new frontiers may be justified because the modification of the subgoals causes the trajectory along the route to take a different shape. So even if the open-space trajectory between a pair of subgoals was previously collision free, it may now cross obstacle boundaries. For example, when the robot is moving in an environment with corridors, travel between zero velocity subgoals at corridor intersections may initially be collision-free, but as the subgoal states are modified, the open-space trajectory between them may curve outward, intersecting with a corridor wall. (Compare Figure 1.2 and Figure 1.10.) This doesn't place the robot in danger of colliding with the obstacle, of course, since the robot is employing the dodging controller. However, it is desirable to choose the subgoals so that the open space control between obstacles are collision free (or would be in the absence of error). For then obstacle avoidance is implemented by following subgoals selected by the global optimization process, instead of being implemented by a local, on-line procedure that may be suboptimal. For this reason, when a potential collision is detected and a dodge is performed, a new frontier is inserted in the region of the dodge.

Similarly, changes in the trajectory may change the desired location of an existing frontier. Each frontier should be positioned so that the trajectory crosses it at its point of closest approach to an obstacle, but this point of closest approach will change as the trajectory shape changes.

In the implementation, a proximity sensor is simulated. This sensor, an abstraction of
the robot's range sensing capability, returns the approximate distance and orientation of the nearest obstacle point. This capability is well-matched to the characteristics of sonar range sensors which return the distance to the nearest obstacle within the sonar cone, and don't suffer from specular reflection when an obstacle surface is normal to a direction between it and the sensor (as would be the case at the nearest point on an obstacle for smooth obstacle shapes).

Modification and insertion of frontiers makes use of a minimum tracking procedure to identify a trajectory state at a local minimum of the robot's proximity to obstacles. Once initiated, the minimum tracking procedure monitors the reading from the proximity sensor, and stores the state with the nearest obstacle proximity found so far. When the obstacle proximity starts to increase, the procedure returns the velocity and estimated position of the state with the closest obstacle proximity. To avoid signaling spurious minima due to control error, sensor error, and small-scale environment features, the procedure is parameterized by an error threshold value. A minimum proximity state is signaled only when a proximity reading exceeds the sum of the error threshold value and the minimum proximity reading found so far. If the procedure is signaled to stop before a minimum is found, it returns the state with the closest obstacle proximity encountered so far.

Existing frontiers are modified as follows. When the robot's estimate of the distance to the current subgoal goes below a specified minimum tracking range (greater than the subgoal switch distance – see Section 5.2), the minimum tracking procedure is initiated. A minimum proximity state will be found either when the procedure itself returns, or when it is signaled to stop when the robot's distance to the subgoal exceeds the minimum tracking range. The frontier's line segment is modified to be collinear with the position of the minimum proximity state and the nearest obstacle point. The subgoal is chosen to be the minimum proximity state found by the minimum tracking procedure.

A new frontier may be inserted when a potential collision is detected and the robot performs a dodge. At a time step when a potential collision is detected, if the minimum tracking procedure is not in progress, then it is initiated. When it signals a minimum proximity state, a new frontier is established at this state as described above. If the distance between the subgoal on this new frontier and either adjacent subgoal is below a threshold value, then the adjacent frontier is deleted. When the obstacle being dodged is convex, this procedure will have the effect of placing a new subgoal at the nearest edge of the obstacle that the robot passes by. When the obstacle is nonconvex, it is possible that an unnecessary frontier will be inserted. As shown in Figure 7.2 the robot may encounter a proximity minimum before reaching the vicinity of the temporary target. This doesn't create a real problem, since the subgoal states on these frontiers will be optimized and hence won't slow down the robot.
Figure 7.2: When the obstacle being dodged is convex, as in panel (a), a single subgoal, s, and associated frontier will be inserted at the trajectory's closest approach to the obstacle. When the obstacle is nonconvex, as in panel (b), a subgoal may be inserted at a point such as s1, though the trajectory target needed to dodge the obstacle is in the vicinity of subgoal s2. Although subgoal s1 is unnecessary, its presence will not diminish performance.

7.6 Summary

The effect of applying the subgoal improvement algorithm is shown in Figure 7.3. A run without subgoal improvement but using stored knowledge of environment geometry took 28 s. After two subgoal improvement cycles the run took 26 s, with performance illustrated in the figure.

The subgoal improvement procedure described in this chapter provides a final stage in the development of an approximate map of the environment that is appropriate for guiding high-speed control. Although the map is not metrically accurate, this subgoal update procedure allows the map to incorporate information into the map that reflects the global layout of the environment.
Figure 7.3: Performance is improved by modifying the position and velocity of subgoals. A run using stored knowledge of environment geometry but without subgoal improvement took 28 s. Shown here is the trajectory after two subgoal improvement cycles. Total travel time: 26 s.
Chapter 8

Performance Comparisons

This dissertation has developed a set of methods for high-speed navigation that addresses two facts of life for mobile robots. The first is that a map is important for navigation since it both enables the robot to navigate directly to the goal and permits enhanced performance by allowing the robot to anticipate non-local environment features. The second is low positioning accuracy which is reflected in map, control, and estimation errors.

Previous work has not adequately addressed both of these constraints simultaneously. While reactive navigation approaches do not take advantage of a map, global optimization methods generate a feedforward control, assuming the availability of an accurate map and model of the robot’s performance. This chapter presents some empirical results to support the claim that the methods developed here for combining a local control method with approximate global information improves performance compared to independent application of the reactive navigation and global optimization methods.

In particular, the results presented in this chapter support two claims:

1. The methods developed here successfully deal with a level of positioning error that causes a feedforward approach to fail.

2. The methods developed here for guiding reactive navigation with map information improve performance compared to a pure reactive navigation approach.

The first claim will be supported by comparing:
- The dodging controller guided by the basic map + stored obstacle geometry + optimized subgoals
- Feedforward open-space control between the optimized subgoals.

A single example is sufficient to make the point that the feedforward approach cannot deal with the level of error handled by the map-guided dodging controller.

To support the second claim, the performance of the following navigation techniques will be compared:
• The dodging controller alone.

• The dodging controller guided by the basic map.

• The dodging controller guided by the basic map + stored obstacle geometry

• The dodging controller guided by the basic map + stored obstacle geometry + optimized subgoals

8.1 Feedforward Control vs. Map-guided Reactive Navigation

A feedforward approach to mobile robot control is not appropriate in the presence of significant positioning error. There is error in both the robot’s knowledge of environment geometry and the robot’s tracking of the planned trajectory. As a result, above some level of error, following a feedforward plan will direct the robot through obstacle boundaries. To demonstrate that this effect is present with a level of error easily handled by map-guided reactive navigation, feedforward performance is demonstrated subject to positioning error generated as described in Section 2.3. The value of the maximum error scaling factor ζ is assigned value 0.1, corresponding to a maximum error of 10% of distance moved. The feedforward control is generated using the subgoals generated by the subgoal optimization process as follows. Open-space control between adjacent subgoals (using their mean estimated relative positions) is generated using the method described in Chapter 3. Positioning error in the application of the feedforward control is due both to errors in the estimated relative positions of subgoals and errors in the control outcome as the feedforward control is applied to the robot.

An example demonstrating the trajectory generated by applying this feedforward control is shown in the top panel of Figure 8.1. Following this control would result in collision. On the other hand, this level of positioning error can be easily handled by the map-guided dodging controller as illustrated in the bottom panel of this figure. The map in this run includes stored obstacle geometry and optimized subgoals.

8.2 Reactive Navigation vs. Map-Guided Dodging

As discussed earlier, there are several problems with applying a purely reactive control strategy to the high-speed navigation task. The first is that the robot needs a representation of its environment in order to ensure that it will follow a direct route to its goal rather than taking a circuitous route or backtracking. The second is that it forgoes the higher performance that is possible when stored information allows the robot to anticipate environment
Figure 8.1: The top panel illustrates the trajectory followed by an open loop control strategy when the control outcomes and the robot’s estimates of subgoal positions are subject to positioning error corresponding to a maximum error level of 10% of distance moved. The bottom panel illustrates the performance of the map-guided dodging controller with the same level of error. The map includes stored obstacle geometry and optimized subgoals.
geometry beyond sensor range and hence travel more rapidly than when it uses local input alone.

This section presents an example of how a pure reactive navigation strategy can fail to follow a desirable route. The next section presents results demonstrating the improvement in performance due to adding stored knowledge of obstacle geometry and optimizing subgoals.

Since reactive navigation techniques rely on local sensor input, these control strategies (when applied independently) can make a navigation decision that locally makes progress to the goal, but ultimately is very inefficient since it neglects the shape of the environment further ahead where, for instance, the path to the goal position may be blocked.

A demonstration of how a purely reactive strategy can run into problems in a perfectly reasonable environment is illustrated in the top panel of Figure 8.2. This figure shows the result of applying the dodging algorithm alone. The robot’s only source of global information is the position of the goal state near the upper right corner of the environment. In this example, the dodging controller follows a reasonable path most of the way to the goal. However, when the robot encounters the last obstacle that separates it from the goal, the dodging controller happens to choose a counterclockwise path around the obstacle. Although there is no indication of it in the local information available to the robot, there is no clear path to the goal in the counterclockwise direction, so the robot eventually must backtrack in order to reach the goal. This problem is avoided when the robot is guided by the basic map, as illustrated in the bottom panel of Figure 8.2.

8.3 Performance with Varied Levels of Map Knowledge

In this section, empirical results are presented that demonstrate the effect on performance of adding information to the map. The results demonstrate that, as expected, adding map information that allows the robot to anticipate the environment beyond sensor range improves the robot’s performance.

These experiments are performed in a set of environments that are representative of the environments mapped by the qualitative exploration and mapping system described by Kuipers and Byun [1988; 1991; Byun, 1990]. Two levels of error are used corresponding to maximum positioning error 2\% and 10\% of distance traveled. Because travel times may differ somewhat due to randomness in control outcomes, 10 trials (differing only in the initialization of the random number generator) are performed in each case.

In all of the figures, the positions of subgoals are indicated by a ‘+’. Robot positions are shown at each time step, and the time step duration in these experiments is 0.5 seconds. Other parameter values are as follows:

\[1\] The robot continues some distance along this incorrect direction because of the rule that the robot should continue to choose the same rotation direction around an obstacle as long as the hit point at the present time step is within some threshold distance of the hit point at the previous time step.
Figure 8.2: The top panel demonstrates an application of dodging control without the map (only the goal position is known). Travel time: 41.0 s. The bottom panel demonstrates the robot's trajectory to the same goal position when it is guided by a basic map. Travel time: 29.0 s. In both examples, maximum error level = 10%.
<table>
<thead>
<tr>
<th>Trial No.</th>
<th>Basic Map</th>
<th>+ Stored Geometry</th>
<th>+ Optimized Subgoals</th>
<th>10% Maximum Error</th>
<th>2% Maximum Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29.5</td>
<td>26.5</td>
<td>24.5</td>
<td>29.5</td>
<td>26.5</td>
</tr>
<tr>
<td>2</td>
<td>30.0</td>
<td>26.5</td>
<td>24.0</td>
<td>29.0</td>
<td>27.0</td>
</tr>
<tr>
<td>3</td>
<td>28.5</td>
<td>27.0</td>
<td>25.0</td>
<td>29.0</td>
<td>26.5</td>
</tr>
<tr>
<td>4</td>
<td>29.0</td>
<td>26.0</td>
<td>24.5</td>
<td>29.0</td>
<td>27.0</td>
</tr>
<tr>
<td>5</td>
<td>29.0</td>
<td>26.5</td>
<td>24.5</td>
<td>29.0</td>
<td>26.5</td>
</tr>
<tr>
<td>6</td>
<td>29.5</td>
<td>26.5</td>
<td>25.0</td>
<td>29.0</td>
<td>26.5</td>
</tr>
<tr>
<td>7</td>
<td>29.0</td>
<td>27.0</td>
<td>25.0</td>
<td>28.5</td>
<td>27.0</td>
</tr>
<tr>
<td>8</td>
<td>29.0</td>
<td>26.5</td>
<td>25.0</td>
<td>29.0</td>
<td>27.0</td>
</tr>
<tr>
<td>9</td>
<td>28.5</td>
<td>27.0</td>
<td>24.5</td>
<td>28.5</td>
<td>27.0</td>
</tr>
<tr>
<td>10</td>
<td>29.0</td>
<td>27.0</td>
<td>24.5</td>
<td>29.5</td>
<td>27.0</td>
</tr>
<tr>
<td>avg</td>
<td>29.1</td>
<td>26.7</td>
<td>24.6</td>
<td>29.0</td>
<td>26.8</td>
</tr>
</tbody>
</table>

Table 8.1: Performance data for the “cheap apartment” environment.

- Maximum velocity: 8 m/s
- Maximum acceleration: 3 m/s²
- Subgoal switch distance: 2.5 m

The system’s performance with varying levels of map information in three different environments are presented in Tables 8.1 through 8.3. The three levels of map information are:

- The basic map.
- The basic map + stored obstacle geometry.
- The basic map + stored obstacle geometry + optimized subgoals.

The robot’s performance in the different environments is illustrated in Figures 8.3 through 8.11. Figures 8.3 through 8.5 illustrate the robot's performance in the “cheap apartment” environment with maximum positioning error 10% of distance traveled. Figures 8.6 through 8.8 illustrate the robot’s performance in the “two islands” environment with maximum positioning error 2% of distance moved. Figures 8.9 through 8.11 illustrate the robot’s performance in the “two-cross” environment with maximum positioning error 10% of distance traveled. As expected, as more information is added to the map, the robot’s performance improves. Performance improves in all environments and at both error levels.
<table>
<thead>
<tr>
<th>Trial No.</th>
<th>10% Maximum Error</th>
<th>2% Maximum Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Basic + Stored + Optimized</td>
<td>Basic + Stored + Optimized</td>
</tr>
<tr>
<td></td>
<td>Map Geometry Subgoals</td>
<td>Map Geometry Subgoals</td>
</tr>
<tr>
<td>1</td>
<td>23.0 23.8 19.5</td>
<td>22.5 21.0 19.0</td>
</tr>
<tr>
<td>2</td>
<td>24.0 23.5 21.5</td>
<td>22.0 21.0 18.5</td>
</tr>
<tr>
<td>3</td>
<td>23.5 22.5 21.0</td>
<td>22.5 20.5 19.0</td>
</tr>
<tr>
<td>4</td>
<td>23.5 23.0 19.5</td>
<td>23.0 20.5 19.5</td>
</tr>
<tr>
<td>5</td>
<td>23.5 23.0 22.0</td>
<td>22.5 20.5 19.0</td>
</tr>
<tr>
<td>6</td>
<td>24.0 23.5 20.5</td>
<td>22.5 20.5 18.5</td>
</tr>
<tr>
<td>7</td>
<td>24.5 22.5 21.5</td>
<td>23.0 21.0 19.5</td>
</tr>
<tr>
<td>8</td>
<td>24.0 23.0 21.0</td>
<td>22.5 21.0 19.0</td>
</tr>
<tr>
<td>9</td>
<td>24.5 23.0 22.0</td>
<td>22.5 20.5 19.5</td>
</tr>
<tr>
<td>10</td>
<td>23.5 22.5 20.0</td>
<td>22.0 20.5 19.0</td>
</tr>
<tr>
<td>avg</td>
<td>23.8 23.0 20.9</td>
<td>22.5 20.7 19.1</td>
</tr>
</tbody>
</table>

Table 8.2: Performance data for the “two islands” environment.

<table>
<thead>
<tr>
<th>Trial No.</th>
<th>10% Maximum Error</th>
<th>2% Maximum Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Basic + Stored + Optimized</td>
<td>Basic + Stored + Optimized</td>
</tr>
<tr>
<td></td>
<td>Map Geometry Subgoals</td>
<td>Map Geometry Subgoals</td>
</tr>
<tr>
<td>1</td>
<td>32.0 27.0 25.0</td>
<td>29.5 26.5 24.5</td>
</tr>
<tr>
<td>2</td>
<td>29.5 27.0 25.0</td>
<td>29.5 26.5 24.0</td>
</tr>
<tr>
<td>3</td>
<td>29.5 26.5 25.5</td>
<td>29.5 26.5 23.5</td>
</tr>
<tr>
<td>4</td>
<td>31.5 27.5 26.5</td>
<td>29.5 26.0 24.0</td>
</tr>
<tr>
<td>5</td>
<td>30.0 28.0 26.5</td>
<td>29.5 26.0 24.0</td>
</tr>
<tr>
<td>6</td>
<td>30.5 27.0 26.0</td>
<td>29.5 26.0 24.0</td>
</tr>
<tr>
<td>7</td>
<td>30.5 26.5 25.0</td>
<td>29.0 25.5 23.5</td>
</tr>
<tr>
<td>8</td>
<td>30.0 27.0 25.5</td>
<td>29.5 26.0 24.5</td>
</tr>
<tr>
<td>9</td>
<td>30.0 27.5 25.0</td>
<td>29.5 25.5 24.0</td>
</tr>
<tr>
<td>10</td>
<td>29.5 27.5 26.5</td>
<td>29.5 26.5 24.0</td>
</tr>
<tr>
<td>avg</td>
<td>30.3 27.2 25.7</td>
<td>29.5 25.9 24.0</td>
</tr>
</tbody>
</table>

Table 8.3: Performance data for the “two cross” environment.
Figure 8.3: Control using the basic map in the “cheap apartment” environment. The trajectory starts at the upper left. Travel time: 29.0 s. Maximum error level = 10%.

Figure 8.4: Control using the basic map plus stored knowledge of environment geometry in the “cheap apartment” environment. The trajectory starts at the upper left. Travel time = 26.5 s. Maximum error level = 10%.
Figure 8.5: Control using improved subgoals with stored knowledge of environment geometry in the “cheap apartment” environment. The trajectory starts at the upper left. Travel time = 24.5 s. Maximum error level = 10%.

Figure 8.6: Control using the basic map in the “two islands” environment. The starting and ending point of the circuit is at the leftmost point of the trajectory; travel is counterclockwise. Travel time: 22.5 s. Maximum error level = 2%
Figure 8.7: Control using the basic map plus stored knowledge of environment geometry in the "two islands" environment. The starting and ending point of the circuit is at the leftmost point of the trajectory; travel is counterclockwise. Travel time = 21.0 s. Maximum error level = 2%.

Figure 8.8: Control using improved subgoals with stored knowledge of environment geometry in the "two islands" environment. The starting and ending point of the circuit is at the leftmost point of the trajectory; travel is counterclockwise. Travel time = 19.0 s. Maximum error level = 2%.
Figure 8.9: Control using the basic map in the "two cross" environment. The trajectory starts and ends at the right top corridor intersection; travel is counterclockwise. Travel time: 29.5 s. Maximum error level = 10%.

Figure 8.10: Control using the basic map plus stored knowledge of environment geometry in the "two cross" environment. The trajectory starts and ends at the right top corridor intersection; travel is counterclockwise. Travel time = 28.5 s. Maximum error level = 10%.
Figure 8.11: Control using improved subgoals and stored knowledge of environment geometry in the "two cross" environment. The trajectory starts and ends at the right top corridor intersection; travel is counterclockwise. Travel time = 25.0 s. Maximum error level = 10%.
Chapter 9

Conclusion

9.1 Summary

The purpose of this dissertation has been to address high-speed navigation in a way that acknowledges both that a map is essential for navigation, and that an autonomously acquired map will be approximate. A map is needed to allow the robot to pursue a direct route to the goal, and it also enhances performance by enabling the robot to anticipate non-local environment features. At the same time, due to map, control and estimation errors, the robot must be able to respond to the updated information that becomes available as it travels through the environment. This research has produced a method for high-performance navigation under realistic conditions of positioning error. The method is based on the combination of reactive navigation with the capability to acquire and apply approximate map knowledge.

High-performance reactive navigation is performed using a novel approach dubbed “dodging.” The open-space minimum-time control to the target is generated, and the corresponding trajectory is checked for collisions with obstacles within sensor range. If a potential collision is detected, a temporary target is chosen for the robot to travel toward so that the robot is directed around the obstacle.

For the chosen model of robot dynamics – a two-dimensional double integrator with acceleration magnitude constraint – generation of the exact open-space minimum-time control is computationally intensive. A faster, approximate solution has been developed which makes use of the well-known “bang-bang” solution to the one-dimensional problem.

If a potential collision is detected, an edge point that the robot should pass through on its way around the obstacle is chosen near an edge of the visible portion of the obstacle. The obstacle surface between the potential collision point and the edge point is represented by a line segment. A simple optimization procedure is employed to select a temporary target velocity that reduces travel time to the temporary target while ensuring that the trajectory to the temporary target will not cross the line segment. The dodging procedure is designed so that, as long as obstacles are properly represented in its occupancy array, non-collision is guaranteed. A probabilistic non-collision guarantee may be provided when there is position
error in control by growing the obstacles in the occupancy array by the maximum likely position error at a given distance from the robot.

A map of the environment is autonomously acquired using Kuipers' and Byun's qualitative mapping approach [1988; 1991; Byun, 1990]. The central element of the map is a topological network that describes the environment in terms of qualitative features: distinctive places and distinctive paths. Development of a topologically correct map is robust in the face of positioning errors. Relative metrical information can be added to the topological network without casting the metrical information in global coordinates. A "basic map" consisting of the topological network and the approximate displacement between adjacent distinctive places can be acquired at low cost.

A map-level controller can guide the dodging controller by using information in the basic map. Given a goal, expressed relative to some nearby distinctive place, the basic map can easily be searched to find a sequence of distinctive places that lead to the goal. The map-level controller can effectively guide the dodging controller to follow a direct route to the goal by choosing a target at a distinctive place, and then, when the robot nears this distinctive place, choosing a target at the next distinctive place in the sequence. Although information in the map is approximate, relatively high performance is attainable in combination with the reactive navigation capability of the dodging controller.

Geometric information can be added to the map by storing range readings, annotated with their approximate positions relative to adjacent distinctive places. As the robot travels through the environment, this information is used to construct an approximate description of environment geometry that is presently occluded or beyond sensor range. Although the stored information is approximate, it can nonetheless improve performance considerably, particularly if the robot has limited sensor range, by allowing the dodging algorithm to choose temporary targets further ahead.

Performance can be improved further by updating the subgoal states. In the limit of perfect information, and to the extent that the travel time of the robot's open-space control approximates the optimal time, travel along the robot's entire route will similarly approximate the global optimal solution when subgoals are optimized. Subgoals are improved by repeatedly modifying them in a direction that reduces overall travel time. A sequence of small modifications to the subgoals are made, each followed by a traversal of the route. This conservative approach allows subgoals to be modified to improve overall travel time while ensuring that the open-space trajectories between subgoals remain approximately collision-free without having a global, accurate model of the environment by using the environment as its own representation.

9.2 Contributions

This research has produced the first method for high-performance, map-guided navigation under realistic conditions of positioning error. Despite map uncertainty, the available, ap-
proximate information can be used to effectively guide high-speed navigation. As might be expected from a traditional metrically accurate map, the map improves performance over a purely reactive navigation system by providing guidance based on large-scale features of the environment. However, the performance of this set of techniques does not critically depend on the availability of expensive, accurate metrical information. Nonetheless, the map may be elaborated with more detailed information, and as the level of detail and accuracy is improved, performance can smoothly improve. Map information can always be safely followed despite its uncertainty because the reactive navigation component provides a non-collision guarantee.

The dodging controller stands on its own as an improvement over previous reactive navigation methods. Unlike much work on reactive navigation, the dodging controller operates the robot at the limits of its dynamic constraints for high-speed performance. The dodging controller has several other advantages over reactive navigation methods that use “potential field”-style approaches including:

- It does not need adjustable parameters to balance between conflicting demands of obstacle avoidance and travel through narrow passages.
- It does not modify the path for obstacles that are not in the way.
- It is not subject to the same sources of oscillatory behavior as potential field approaches, and simulations of the dodging controller have only shown smooth, non-oscillatory behavior.
- It can be directed to pass an obstacle in a chosen rotation direction.

Compared to the most similar reactive navigation method, the “satisficing feedback” (SF) method developed by Feng and Krogh [1990], the dodging controller is more flexible and capable of higher performance. Specifically:

- SF targets must have zero velocity, causing the robot to slow down as a target is approached, reducing performance. Also, critical to application in the navigation system developed here, the SF method could not be used in connection with the subgoal updating algorithm which generates nonzero velocity subgoals.
- The “steering decision algorithm” controller used by the SF method to direct the robot to a target results in a slower travel time than the open-space controller used by the dodging controller, again reducing performance.
- The SF’s method of representing free-space with three linear boundaries can cause inefficient use of free space, for instance choosing a target closer to the robot than necessary. This compounds the performance issue posed by the restriction to zero velocity targets. The dodging controller, in contrast, bases obstacle avoidance decisions on a grid representation of free space.
9.3 Future Work

The success of the high-speed navigation method developed here suggests several areas of future work.

It would be valuable to apply the methods developed here to a physical mobile robot. This would entail development of a platform-specific servo to control the robot to follow the reference trajectory provided by the dodging controller, and a method for sensing local environment shape.

If local sensing for the dodging controller were performed with a vision system or laser range-finding system, it would be interesting to explore the possibility of task-directed perception to reduce the computational demands of the sensing task. Since the dodging controller operates by checking for collision along the open-space route to the target, sensing can be focused in this region. When a potential collision is found, a “visual” search could be performed along the surface of the obstacle to find an edge point.

A version of the dodging algorithm could be developed for a conventionally steered mobile robot with steering curvature constraints. It seems likely that a heuristic minimum-time open-space controller could be developed for this style of system, and a dodging solution developed analogous to that for the omnidirectional case.
Appendix A

Appendix A: Compounding and Merging Operations

Consider two approximate transforms $A_1$ and $A_2$ with nominal estimates $n_1$ and $n_2$ (expressed as vectors), and covariance matrices $C_1$ and $C_2$. To merge these approximate transforms, the Kalman gain factor should first be computed:

$$K = C_1 \ast [C_1 + C_2]^{-1}. \quad \text{(A.1)}$$

Then the result of merging these approximate transforms is an approximate transform with nominal transform given by

$$x_3 = x_1 + K \ast C_1. \quad \text{(A.2)}$$

and with covariance matrix

$$C_3 = C_1 - K \ast C_1. \quad \text{(A.3)}$$

Since for our purposes, the rotation in the nominal transform will always be zero, we can write $n_1$ and $n_2$ in terms of their components as $n_1 = [x_1 \ y_1 \ 0]^T$ and $n_2 = [x_2 \ y_2 \ 0]^T$. Then the approximate transform that results from compounding $A_1$ and $A_2$ has nominal transform

$$n_4 = [x_1 + x_2 \ y_1 + y_2 \ 0]^T \quad \text{(A.4)}$$

and covariance matrix

$$C_4 = HC_1H^T + C_2 \quad \text{(A.5)}$$

where

$$H = \begin{bmatrix}
1 & 0 & -y_2 \\
0 & 1 & x_2 \\
0 & 0 & 1
\end{bmatrix}.$$
Bibliography


