Tools and Algorithms for Mobile Robot Navigation with Uncertain Localization

Thesis by

Kristopher L. Kriechbaum

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

2006
(Defended May 23, 2006)

Acknowledgements

I wish to thank my adviser, Joel Burdick, for his guidance, support, and constant patience. I would like to thank Stergios Roumeliotis for teaching me all I know about Kalman filters. My thanks to the robotics group for their help and feedback. I would like to thank all the members of the Caltech DARPA Grand Challenge team for providing me with such a unique experience.

My good friends and officemates Lars Cremean and Jim Radford provided many worthwhile discussions, as well as much needed distractions. Many thanks to Michael Wolf, Nicolas Hudson, Alexandros Taflanidis, Mohamed El-Naggar, Joseph Klamo, and Timothy Chung for the battles they provided. I also want to thank the Aero and SOPS soccer teams for giving me an excuse not to work on the weekends, and the Caltech Club Soccer team for giving me an excuse not to work evenings.

I would like to thank my family for their love and support, and Ophelia for the unconditional love that only a dog can give. Most of all, I would like to thank my wife, Maria, for her continuous encouragement throughout my graduate career.

Abstract

The ability for a mobile robot to localize itself is a basic requirement for reliable long range autonomous navigation. This thesis introduces new tools and algorithms to aid in robot localization and navigation. I introduce a new range scan matching method that incorporates realistic sensor noise models. This method can be thought of as an improved form of odometry. Results show an order of magnitude of improvement over typical mobile robot odometry. In addition, I have created a new sensor-based planning algorithm where the robot follows the locally optimal path to the goal without exception, regardless of whether or not the path moves towards or temporarily away from the goal. The cost of a path is defined as the path length. This new algorithm, which I call "Optim-Bug," is complete and correct. Finally, I developed a new on-line motion planning procedure that determines a path to a goal that optimally allows the robot to localize itself at the goal. This algorithm is called "Uncertain Bug." In particular, the covariance of the robot's pose estimate at the goal is minimized. This characteristic increases the likelihood that the robot will actually be able to reach the desired goal, even when uncertainty corrupts its localization during movement along the path. The robot's path is chosen so that it can use known features in the environment to improve its localization. This thesis is a first step towards bringing the tools of mobile robot localization and mapping together with ideas from sensor-based motion planning.

Contents

Acknowledgements **iv**

Abstract **v**

1 Introduction **1**

1.1 Motivation 1

1.2 Review of Prior Work 2

1.3 Contributions of this Thesis 5

1.4 Thesis Outline 6

2 Background **8**

2.1 Tangent Bug 8

2.2 Kalman Filtering 11

2.2.1 The Basics 12

2.2.2 Kalman Filters and Robotic Localization 15

2.2.2.1 Localization of a Point Robot Using Landmarks 15

3 Weighted Scan Matching **21**

3.1 Introduction 21

3.2 The Weighted Range Sensor Matching Problem 25

3.2.1 The Measurement Model 25

3.2.2 A General Covariance Model 27

3.2.3 Displacement Estimation via Maximum Likelihood 28

3.2.4 The Algorithm and Its Initial Conditions 30

3.2.5 Covariance of the Displacement Estimation Error 31

3.3 Scan Matching Error/Noise Models 33

3.3.1 Measurement Process Noise 33

3.3.2 Correspondence Error 34

3.3.3 Measurement Bias Effects 37

3.4 Selection of Point Correspondences 38

3.5 Estimating the Incidence Angle 39

3.6 Experiments 40

3.6.1 Robustness and Accuracy Comparisons 40

3.6.2 Multi-Step Runs 47

3.6.3 Comparison of Computational Demands 50

3.6.4 Experiments with Bias Compensation 51

3.7 Conclusions 52

4 Optim-Bug **54**

4.1 Setup and Definitions 54

4.2 Optim-Bug Overview 56

4.3 Shortest Path Properties 59

4.4 Proof of Completeness 60

4.5 Conclusion 71

5 Path Optimization **72**

5.1 The Localization Framework 72

5.2 Cost Function 75

5.3 Practical Optimization Approach 76

5.4 Initial Condition Generation 79

5.5 Results 80

5.5.1 Robot Drives Past the Goal 82

5.5.2 Robot Follows the Landmarks 83

5.5.3 Examples with Obstacles 84

5.6 Summary and Discussion 85

6 Uncertain Bug **88**

6.1 Setup and Definitions 89

6.2 Motivation and Background 91

6.3 The Uncertain Bug Algorithm 92

6.3.1 Effects of Uncertainty 94

6.4 Uncertain Bug Algorithm Properties 96

6.5 Simulation Results 100

6.6 Discussion 102

7 Conclusion **105**

7.1 Summary of Contributions 105

7.2 Future Directions 106

A Weighted Scan Matching Derivations **107**

A.1 Weighted Translation Solution 107

A.2 Weighted Rotation Solution 107

A.3 Covariance Estimation 110

B Implementation Details **114**

B.1 Obstacle Constraints 114

B.1.1 The Constraint Equations 116

C Shortest Path Properties **119**

D Bounded Uncertainty Freespace **124**

D.1 Goal Only 124

D.2 Goal and One Landmark 126

D.3 Goal and Multiple Landmarks 128

Bibliography **129**

List of Figures

2.1 The local tangent graph, with the optional goal node, T_g, added. 9
2.2 A situation where the robot would switch into boundary following mode. . . 10
2.3 An example Tangent Bug sequence where the robot thinks it is at the goal, but the error is large. 12
2.4 Setup of a Kalman filter for mobile robot localization. 15
2.5 A three-step propagation example. This shows how the estimate error continues to grow. 19
2.6 A single-step propagation example showing multiple trials. 19
2.7 Example data illustrating using a landmark for localization. 20
3.1 Geometry of the range sensing process. The robot acquires dense range scans in poses i and j. The circles represent robot position, while the x-y axes denote the robot's body fixed reference frames. 22
3.2 Representation of the uncertainty of selected range scan points. 24
3.3 A) Experiments with initial displacement perturbations between scans taken at a single pose. B) Closeup of robot pose with results. 42
3.4 A) Experiments with initial displacement perturbations between scans taken at different poses. B) Closeup of pose 2 with results. 44
3.5 A) Experiments with initial displacement perturbations in a non-static environment. B) Closeup of pose 2 with results. 45
3.6 A) Experiments with initial displacement perturbations in a hallway environment. B) Closeup of pose 2 with results. 46
3.7 A) A 109-pose 32.8-meter loop path. B) Closeup of final path poses, shown the covariance estimates of the weighted and unweighted algorithms. 48
3.8 A) 83-pose 24.2-meter loop. B) Closeup of final loop poses. 49

3.9 The bias fit function. 51

4.1 The *current visibility sets*, $v(x_k)$, shown at each location of a four-step path. At x_2, obstacle $\mathcal{O}_2$ is a *blocking obstacle*. 55

4.2 The *total visibility set*, V_{tot}, for the same four-step hypothetical example of Figure 4.1. 56

4.3 A hypothetical example showing obstacle endpoints, tangent points, and their associated goal-tangent lines. 60

4.4 Illustration for Lemma 4.1 showing that the total visibility set, V_{tot} grows at each step. 62

4.5 At every step while navigating around a blocking obstacle, the robot always sees a new segment of the obstacle boundary that is contiguous with a previously known segment. It may see additional boundary segments that are not contiguous. 65

4.6 A hypothetical sequence of steps showing how Optim-Bug can oscillate back and forth along a blocking obstacle. This also illustrates how a continuous piece of the boundary segment is mapped out. 67

4.7 A hypothetical example where the robot is blocked by the same obstacle twice. Portions of the obstacle in the robot's map are denoted by a darker color. . . 68

5.1 The environment setup assumed for the path planning process. The robot starts at location $\mathbf{x}_R$ and is instructed to travel to $\mathbf{x}_g$. L_1, L_2, L_3, and L_4 are landmarks that the robot has knowledge of. The circles are obstacles that the robot must avoid. 73

5.2 Sample measurement variance profile as a function of range. $\beta = .1$, $\gamma = 10$, and $r_{max} = 3$. Note that the y-axis scale is logarithmic. 81

5.3 Example where the robot initially drives past the goal in order to use a landmark. For simplicity, the sensing range is plotted about the landmark - if the robot moves inside of that circle, it is close enough to see the landmark. . . . 83

5.4 A multi-step path where the robot significantly deviates from the shortest path in order to use the landmarks. 84

5.5 An simple example with obstacles. 85

5.6 The same landmark configuration as that of Figure 5.4. Obstacles were placed to block the original optimal path. 86

6.1 Setup assumed for the Uncertain Bug algorithm. The robot does not know the position of obstacles in the environment. The robot's sensor has a finite sensing range, r. 90

6.2 Hypothetical example where the robot chooses a path that causes it to step into an uncertainty obstacle. 95

6.3 Figure illustrating an extra termination condition for Uncertain Bug. The next robot path is shown by the dashed line. The robot will follow the path to the point labeled x_2. 98

6.4 A simulation of Uncertain Bug. The Tangent Bug path is shown as the pink dashed line. Final cost at the goal is 39% of the cost of the Tangent Bug path. 101

6.5 The planned path for steps 9 and 10. Note that when the robot sees the obstacle, it re-plans the path, taking into account the new information about the world. 102

6.6 A simulation of Uncertain Bug. The Tangent Bug path is shown as the pink dashed line. Final cost at the goal is 32% of the cost of the Tangent Bug path. 103

B.1 A portion of the path where the robot passes by an obstacle. 115

B.2 Setup of a single line and circle. 115

C.1 A fixed endpoint variation. 120

D.1 Maximum size of covariance ellipse to fit inside circle with radius of ϵ. 125

D.2 Set of points from which the goal can be reached with $\sigma_{max} < \epsilon$. The set is the darkened box. 126

D.3 A bound on the set of points from which the goal could be reached with $\sigma_{max} < \epsilon$. 127

D.4 Union of the multiple sets. 128

List of Tables

3.1 Position and orientation error values for trials with no initial perturbations. . 41
3.2 Robustness and accuracy comparison statistics for trials with initial perturbations. 41

Chapter 1

Introduction

1.1 Motivation

The ability for a mobile robot to localize itself is a basic requirement for reliable long range autonomous navigation. This thesis introduces new tools and algorithms to aid in robot localization and navigation. I introduce a new range scan matching method that incorporates realistic sensor noise models. This method can be thought of as an improved form of odometry. Results show an order of magnitude of improvement over typical mobile robot odometry. In addition, I have created a new sensor-based planning algorithm where the robot follows the locally optimal path to the goal without exception, regardless of whether or not the path moves towards or temporarily away from the goal. The cost of a path is defined as the path length. This new algorithm, which I call "Optim-Bug," is complete and correct. Some of the ideas and issues from Optim-Bug are used to assist in the discussion of the case where the robot does not have perfect positional knowledge.

Finally, I developed a new on-line motion planning procedure that determines a path to a goal that optimally allows the robot to localize itself at the goal. This algorithm is called "Uncertain Bug." In particular, the covariance of the robot's pose estimate at the goal is minimized. This characteristic increases the likelihood that the robot will actually be able to reach the desired goal, even when uncertainty corrupts its localization during movement along the path. I assume that the robot has a number of (possibly uncertain) landmarks available to aid in its localization process. The robot's path is chosen to exploit these landmarks and use them for better localization. I also assume that the robot has noisy odometry. Hence, the algorithm effectively finds a balance between longer paths that pass close to landmarks (thereby increasingly localization ability) and shorter paths that introduce less odometry error. This method can also include any other localization aids based on either proprioceptive (e.g., inertial navigation unit) or exteroceptive (e.g., laser scan matching algorithms [42]) whose operation can be modeled in a Kalman filter framework. In the simulations presented in this thesis, I focus strictly on the combination of landmarks and on-board odometry.

Two highly desirable features of any planner are completeness and correctness. A path is *correct* if it lies wholly within the freespace (i.e., the path does not intersect any obstacles). Completeness means that the planner will generate a path to the goal if one exists, and will terminate in a finite amount of time. Essentially all of the complete and correct sensor-based motion planning algorithms assume that the robot has perfect knowledge of its location at all times (e.g., [11, 20, 38]). Over very short distances, this assumption may not be so bad, and the robot will likely get close enough to the goal to be considered successful. Nonetheless, even in moderately sized environments, the localization error may grow to the point that the robot is lost for all intents and purposes. The field of simultaneous localization and mapping (SLAM) provides many useful tools to help keep the robot's error low. However, SLAM algorithms do not fully address the issue of navigation. It is the disconnect between the navigation problem and the localization and mapping problem that I hope to bridge with this work.

My path planning method takes into account the rich body of literature and accumulated experience with using the Kalman filter for localization. However, prior work in localization and mapping has not fully addressed the problem of how to plan a correct path to a goal while taking into account possible localization uncertainty. Instead, the primary focus has been on incorporating newly discovered landmarks into a map, and localizing the robot by using measurements of currently known landmarks [31, 50, 55].

1.2 Review of Prior Work

Some of the earliest complete and correct sensor-based motion planning algorithms are the Bug algorithms by Lumelsky and Stepanov [38]. The Bug algorithms assume nothing more than a point robot with a sensor that can detect whether or not the robot is touching an obstacle. They prove that the robot will reach the goal after a finite amount of time if the goal is reachable. These algorithms are the origin of later "Bug"-like planners.

Kamon, Rimon, and Rivlin extended the Bug algorithms to the case of a sensor with finite range and a 360° field of view in Tangent Bug [20]. Tangent Bug produces *locally optimal* paths. The paths are locally optimal in that they are the optimal paths given the robot's limited knowledge of the environment. Laubach modified the algorithm further to the case of a sensor with a more limited field of view in Wedge Bug [26]. All of the Bug-style algorithms are efficient with respect to memory requirements. However, they all require that the robot have perfect position knowledge at all times.

Roadmap methods, such as Choset's HGVG [11], take a different approach to the problem. The roadmap captures interesting topological features of the environment, such as the connectivity of two adjacent rooms. The roadmap can be more efficient than other approaches, such as building up a map in Cartesian coordinates. In all of these methods the proof of convergence of the robot to the goal requires perfect dead reckoning for the robot.

There are numerous methods available to keep the robot's position estimate accurate. In some cases, they make the perfect dead-reckoning assumption tractable. Matching 2-D range scans from separate robot positions has been proposed by Gonzalez and Gutierrez [17] and Lu and Milios [36, 37]. Their scan matching algorithms all assume that the physical sensor returns perfect range measurements. They also assume that the range scans at different robot positions sample the environment boundary at exactly the same point.

Any method of improved odometry will still suffer from the problem of a growing position estimate error over time. To reduce the estimate error, methods using measurements of external features must be used. Early works using Kalman filters [19] for robot localization were proposed by Moutarlier and Chatila [40], Cox [14], and Smith et al. [49, 50]. One assumes that the robot is given the location of a number of landmarks or features in the environment that the robot can recognize. Extensions to this include methods for the robot to self-select landmarks and incorporate them into its map [31]. Other methods such as particle filters [16, 54] allow the tracking of non-Gaussian distributions. Using tools such as a Kalman filter for localization has become a standard method in mobile robotics.

Recent works have started taking the robot's localization capability and landmarks into account as part of the path planning process. Lorussi, Marigo, and Bicchi [35] provided an elegant and detailed solution to the problem of choosing the path of an exploring point robot so as to optimize its ability to localize in the presence of two precisely known landmarks. This method assumes that the landmarks are perfectly known, and does not consider uncertain robot motion. It also does not incorporate movement towards a specific goal.

Briggs et al. [8, 9] use landmarks in the presence of significant sensor uncertainty. They formulate the expected shortest path problem as a Markov decision process. Their method requires that the visibility graph of the landmarks be constructed beforehand. Each edge of the visibility graph is augmented with information about the probability that the edge is passable or not, i.e., the target landmark can be detected from the current landmark. Blei and Kaelbling [7] take a similar approach, but they assume that the state of an edge is only known with a certain probability. Neither approach considers localization capability along the path.

Lazanas and Latombe [29, 30] have also developed algorithms for motion planning with landmarks. They assume that landmarks are areas of perfect sensing and perfect motion execution. When the robot is not within range of a landmark, it is assumed to have bounded motion uncertainty. They use the idea of chaining together sequences of landmarks that the robot can successfully reach. While their method allows uncertainty in the robot's motion, their solution ignores any issues of localization or sensing uncertainty effects.

Other works have proposed using some function of a covariance matrix as a cost function. Trawny and Barfoot [56] considered the best formations for a team of robots to maintain localization ability when inter-robot communication is allowed. They use a cost function equal to the determinant of the covariance matrix. Their method does not incorporate external landmarks, but the robots use each other to perform localization.

Logothetis et al. [33, 34] propose using the trace of the target state error covariance as the cost function in a target tracking problem. The aim of their approach is to find the best paths for observers that can measure only the bearing to the target. They solve the optimization problem using both dynamic programming and brute force enumeration over all possible observer paths. Because it is a target tracking problem, there is no notion of a final goal location.

Rezaei et al. [43] introduced a graph-based algorithm to plan the motion of a vehicle so as to increase the overall information content in a discrete localization map. In particular, they use a cost function that is equal to the trace of the covariance matrix. Odometry and incorporation of multiple sensing modalities was not included in their work.

Lavalle et al. [27, 28] have developed some general methods for motion planning with uncertainty—though their methods do not specifically take localization uncertainty into account. They specifically address the problem of a changing environment, such as doors that open and close according to some statistical process, but do not address sources of error such as sensing and odometry [28]. Both of these methods use a dynamic programming algorithm to compute robot plans.

Lambert and Fraichard [23] address the problem of navigating a car-like robot with motion and sensing uncertainty. Their cost function is a combination of path length and a measure of localization capability along the path. Their approach requires a prior map of the environment. They find the features that are best for localization in different regions of the map. The regions of the map that share the same features are connected using a roadmap. The shortest path is computed using the Dijkstra algorithm[13].

One of the most closely related prior works is the "coastal navigation" algorithm of Thrun and coworkers [46, 47]. This work similarly formulated a cost function–based algorithm to find paths that allow a robot to take advantage of a previously constructed grid-based map of an environment. Their formulation resulted in a costly dynamic programming (or policy iteration) solution. At first glance, the problem presented in this thesis could be solved using dynamic programming. However, the structure my formulation permits a simplification of the solution to a more efficient optimal control solution. In turn, it can be practically solved using a simple collocation and gradient descent method.

1.3 Contributions of this Thesis

This thesis introduces a new algorithm to estimate a robot's planar displacement by weighted matching of dense two-dimensional range scans. Based on models of expected sensor uncertainty, the algorithm weights the contribution of each scan point to the overall matching error according to its uncertainty. A maximum likelihood formulation is used to estimate the optimal displacement between two consecutive poses. Uncertainty models that account for effects such as measurement noise, sensor incidence angle, and correspondence error are developed. This also gives a more realistic covariance of the displacement estimate than is found in prior work. This work was done jointly with Samuel Pfister and Stergios Roumeliotis.

With some modifications, the Tangent Bug algorithm can be recast within an optimization framework. The original Tangent Bug does not always follow the shortest path to the goal. A new algorithm that always follows the shortest path to the goal, given limited information about the world, is developed. This new algorithm is called *Optim-Bug.* I prove that this new algorithm is complete and correct. Results and general ideas from Optim-Bug are used to motivate discussion and development of an algorithm to handle the case of noisy odometry and imperfect sensing.

An off-line optimization method assuming complete knowledge of the environment that computes the best path to minimize robot pose uncertainty at the goal is presented. By minimizing the robot's expected position error covariance at the goal, I am maximizing the possibility that the robot will be able to recognize the goal when its position estimate says that it is at the goal. This optimization method is a first step towards bridging the gap between sensor-based planning algorithms and localization and mapping techniques.

A main contribution of this work is a sensor-based planning algorithm called *Uncertain Bug*, in which the robot is not assumed to have perfect dead-reckoning and prior knowledge of the environment's geometry. Uncertain Bug also takes into account noisy sensor measurements and uncertain landmark locations. Issues such as the choice of landmarks for navigation, or algorithms to learn important landmarks [53], are not addressed in this thesis.

The Uncertain Bug algorithm finds the path to the goal that minimizes the expected robot position uncertainty at the goal. It is assumed that if the robot can get "close enough" to the goal, it will be able to recognize it and declare success. On the other hand, if no paths exist that reach the goal within this threshold, the algorithm declares failure. An interesting consequence of this threshold is that the robot may fail to reach the goal even when an acceptable path existed from the start. In other words, the fact that the algorithm chose one acceptable path over another means that it can no longer guarantee that it will get close enough to the goal to recognize it.

1.4 Thesis Outline

Chapter 2 provides background material on Kalman filters and the Tangent Bug algorithm. It is assumed that the reader has some knowledge of Kalman filtering, with main results presented for the case of a mobile robot. Chapter 3 presents theory and results from a range scan matching algorithm. The main improvement over previous techniques is that the new algorithm takes into account the uncertainty of sensor measurements. Chapter 4 introduces Optim-Bug, an algorithm that always follows the shortest path to the goal in an unknown environment. It is shown that this new algorithm is complete and correct. Chapter 5 introduces the details of an optimization method used to find the path with the smallest robot pose estimate covariance at the goal. Both results and details of the optimization approach are presented. Finally Chapter 6 presents the Uncertain Bug algorithm. Uncertain Bug plans the path with the least robot pose estimate uncertainty at the goal.

Chapter 2

Background

2.1 Tangent Bug

This section provides a short description of the Tangent Bug algorithm developed by Ishay Kamon, Elon Rimon, and Ehud Rivlin in 1995 [20]. Much of the inspiration for Optim-Bug and Uncertain Bug comes from the Tangent Bug algorithm. The aim of Tangent Bug is to find a path through previously unknown terrain from the robot's current location to some given goal location. The path must not intersect any obstacles (correctness). Moreover, the algorithm must either reach the goal or determine that the goal is unreachable in a finite amount of time (completeness).

The robot starts at a location x_R, and is commanded to move to a position of x_g. The robot is assumed to be equipped with an omnidirectional sensor with a maximum sensing range of R. Tangent Bug makes use of the *local tangent graph*, or LTG. The LTG consists of nodes at the robot's location and the endpoints of sensed obstacles, and edges between the robot and the sensed obstacle endpoints. An optional node, T_g, is added at the intersection between the line segment $\overline{x_R x_g}$ and the circle at x_R with a radius of R. T_g is added if and only if the line segment does not intersect any obstacles and $d(x_R, x_g) > R$. T_g is the projection of the goal onto the visible set, and is added only if the straight-line path to the goal is free. Figure 2.1 shows an example of what the LTG may look like. In this example, the path to the goal within the robot's visible region is unobstructed, so T_g is added.

Tangent Bug operates in two modes: *motion to goal* (MtG) mode and *boundary following* (BF) mode. During the motion to goal mode, the robot monotonically decreases its distance to the goal. Boundary following is used to escape local minima in the distance to the goal function. The robot is constantly sensing the environment and computing the LTG. The

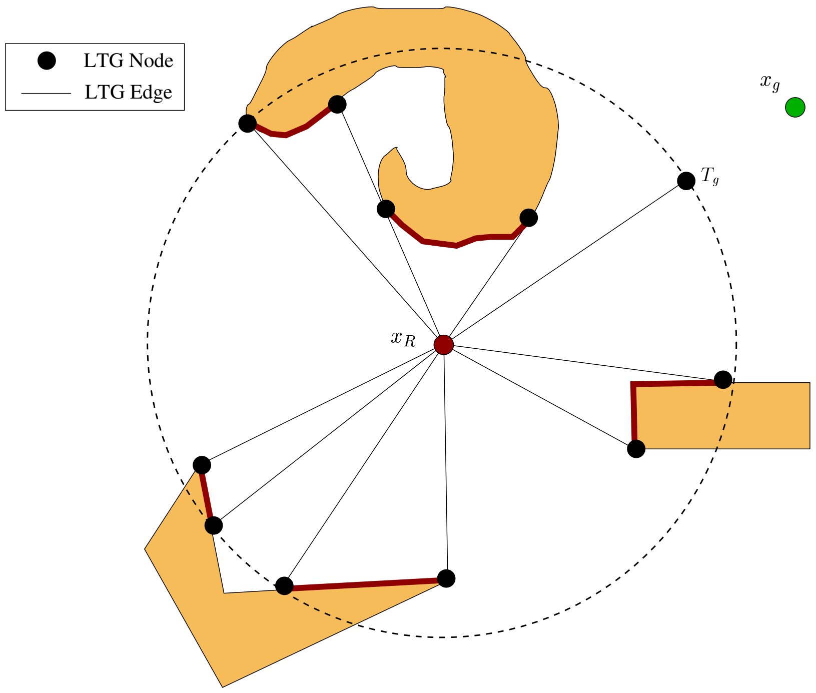

Figure 2.1: The local tangent graph, with the optional goal node, T_g, added.

LTG is then used to determine the next motion. Together with the switching conditions, Tangent Bug guarantees that the robot will reach the goal, if it is reachable.

In MtG mode, the robot always moves closer to the goal. At each step, the robot constructs the LTG. Next the LTG is searched to find the *locally optimal* direction along which the robot should move. It is locally optimal in that it is the shortest path to the goal, taking into account only the obstacle information that is available in the finite sensing range. MtG mode continues until the robot reaches the goal or the robot detects that it is trapped within a local minimum of the distance to goal function. This situation is caused by an obstacle blocking the robot's path. When this happens, the algorithm switches to boundary following.

An alternative interpretation of the MtG mode is that of an optimal control problem. Consider an optimal control problem where the system state (robot) starts at location x_R, with the goal of moving to state x_g. The problem is to find the lowest cost path that moves from x_R to x_g, where the cost of a path is defined as the total length of the path. Motion to goal mode is equivalent to this optimal control problem when only the obstacle information (constraints) in the current visibility set (denoted $v(q)$) is taken into account, and the robot only executes the portion of the path inside $v(q)$. The realization that Tangent Bug can be

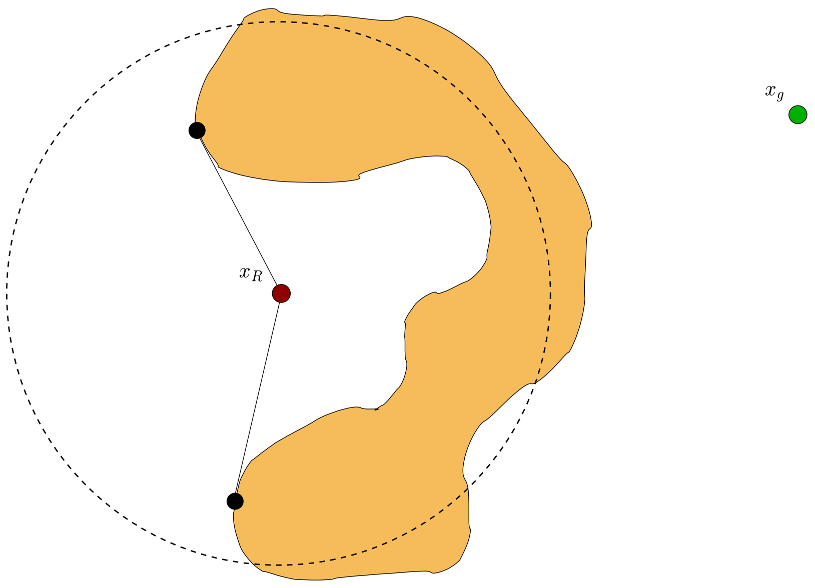

Figure 2.2: A situation where the robot would switch into boundary following mode.

formulated as an optimal control problem motivates the algorithms in Chapters 4 and 6.

Boundary following mode is used to navigate around the blocking obstacle that prevents the robot from making further progress towards the goal. The robot switches to BF mode when there are no nodes of the LTG that are closer to the goal than its current position—it is trapped in a local minima of the distance to goal function. An example of when BF mode must be used is presented in Figure 2.2

When it switches to BF mode, the robot remembers d_{min}, the smallest distance to x_g from any point in the currently visible freespace at the start of the current boundary following sequence. It also chooses a direction to follow the obstacle boundary. While following the obstacle boundary, the LTG is continually updated and checked for the *leaving condition*, a node that will move the robot closer to the goal than d_{min}. When such a node is found, the robot moves to that node and switches back to motion to goal mode. Of course, if the robot happens upon the goal while circumnavigating the obstacle, the algorithm terminates. On the other hand, if the robot completes a loop around the obstacle without satisfying the leaving condition, the goal is deemed unreachable and the algorithm terminates.

Tangent Bug in summary:

1. Motion to goal: Choose the locally optimal path towards the goal, until

   (a) The goal is reached. Stop.
   (b) A local minimum of the distance to goal function $d(\cdot, x_g)$ is detected. Go to boundary following mode.

2. Boundary following: Pick a direction to move around the obstacle. Move around the obstacle boundary while updating d_{min}, the closest encountered distance to the goal so far, until

   (a) the goal is reached. Stop;
   (b) the leaving condition is met. Go to motion to goal mode;
   (c) the robot detects that the goal is unreachable. Stop.

Similar to nearly all other complete and correct planners, Tangent Bug assumes that the robot has perfect dead reckoning. That is, the robot knows exactly where it is in the global reference frame at all times with no error. Any realistic system is bound to have noisy odometry from low-quality hardware, noisy sensors, wheel slippage, and other factors. No matter how small these errors are, they will grow without bound (as demonstrated in Figure 2.5). Figure 2.3 depicts a sample run of the Tangent Bug algorithm. In this example, the robot's odometry estimate is corrupted with Gaussian noise at each step. Although the goal is relatively near, the final error is approximately 15% of the total distance traveled.

2.2 Kalman Filtering

This section discusses the Kalman filter and its use in robotics. A Kalman filter is an estimator for a linear system that is corrupted by white Gaussian noise. It is assumed that the reader has some familiarity with Kalman filters, and only a short introduction to Kalman filtering is given in order to establish notation and key formulas. After the brief introduction, results for the case of a mobile robot taking measurements of landmarks are derived. These results are used extensively in later sections.

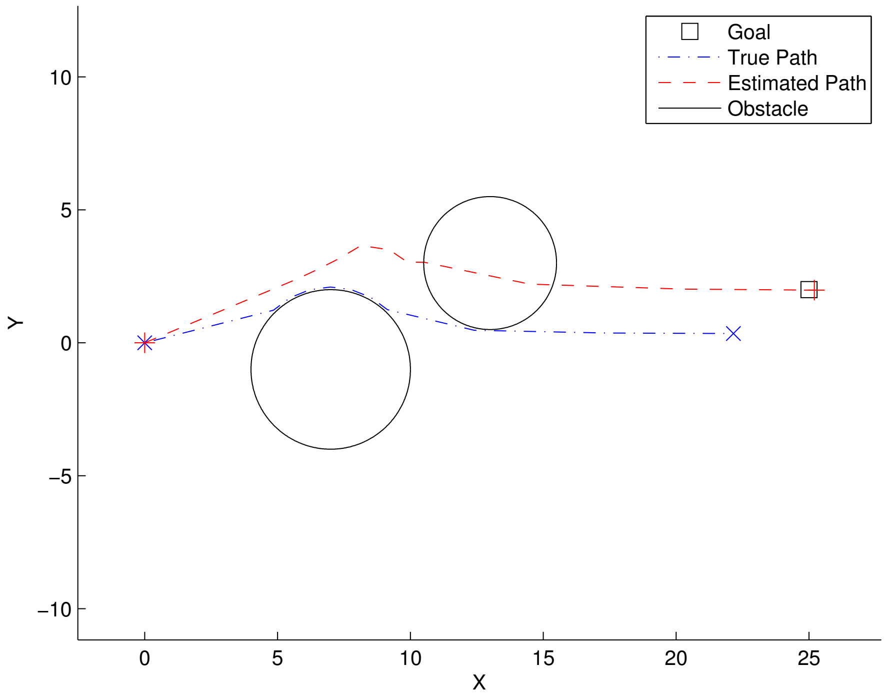

Figure 2.3: An example Tangent Bug sequence where the robot thinks it is at the goal, but the error is large.

2.2.1 The Basics

A discrete-time Kalman filter estimates the state $x \in \mathbb{R}^n$ of a linear system whose evolution is described by the discrete-time linear equation

$$x(k+1) = F_k x(k) + B_k u(k) + G_k w_k \tag{2.1}$$

from measurements $z \in \mathbb{R}^m$ that are assumed to be related to the state by the equation

$$z(k+1) = H_{k+1} x(k+1) + v_{k+1}. \tag{2.2}$$

The random variables $w_k \in \mathbb{R}^p$ and $v_{k+1} \in \mathbb{R}^m$ are the process noise and the measurement noise, respectively. They are assumed to be independent, white, and Gaussian noise processes with the following properties:

$$E\left[w_k\right] = 0, \tag{2.3}$$

$$E\left[w_k w_k^T\right] = Q_k, \tag{2.4}$$

$$E\left[v_{k+1}\right] = 0, \tag{2.5}$$

$$E\left[v_{k+1} v_{k+1}^T\right] = R_{k+1}, \tag{2.6}$$

where $E[\,\cdot\,]$ denotes the expectation operation. Note that the system and measurement matrices, $F \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times l}, G \in \mathbb{R}^{n\times p}$, and $H \in \mathbb{R}^{n\times m}$ need not be constant. It is also assumed that $u(k)$ is known at each time step.

The Kalman filter proceeds in two main steps, the *propagation* step and the *update* step. The propagation step uses the system dynamics to predict a new state estimate at the next time step. Since the system is noisy, this prediction will have some error associated with it. The update step uses the measurements z to help correct the estimate. Since measurements are also corrupted by noise, this correction will not make the estimate perfect. Let $\hat{x}(k/j)$ be the estimate of the state at time k, using measurements up to time j. The same convention will also be used to denote the dependence of the state covariance on the time and measurement indices. The state $\hat{x}(k+1/k)$ can be considered to be the prior (before the measurement information is incorporated), while $\hat{x}(k+1/k+1)$ is analogous to the posterior (after the measurement information has been used). The best estimate of the state at the next time step assumes that w_k takes its zero mean,

$$\hat{x}(k+1/k) = F_k \hat{x}(k/k) + B_k u_k. \tag{2.7}$$

Because the real system is corrupted by the unknown noise, the estimate will always have some error. Let the prior estimate error at time $k+1$ be denoted by $\tilde{x}(k+1/k)$,

$$\tilde{x}(k+1/k) = x(k+1/k) - \hat{x}(k+1/k). \tag{2.8}$$

The covariance of the estimate error after the propagation step, denoted $P(k+1/k)$, is given by

$$\begin{aligned} P(k+1/k) &= E\left[\tilde{x}(k+1/k)\tilde{x}^T(k+1/k)\right] && (2.9) \\ &= F_k P(k+1/k) F_k^T + G_k Q_k G_k. && (2.10) \end{aligned}$$

At every time step of system evolution, the estimate and the estimate error covariance are propagated. If no measurements are incorporated to reduce the estimate error, the estimate error covariance $P(k+1/k)$ will grow without bound. The update step of the Kalman filter reduces the estimate error covariance.

In deriving the update equations, the goal is to find an equation that gives a posterior estimate, $\hat{x}(k+1/k+1)$, as a combination of the prior, $\hat{x}(k+1/k)$, and a weighted difference between a true measurement, $z(k+1)$, and a predicted measurement, $H_{k+1}\hat{x}(k+1/k)$:

$$\hat{x}(k+1/k+1) = \hat{x}(k+1/k) + K_{k+1}\left(z(k+1) - H_{k+1}\hat{x}(k+1/k)\right). \qquad (2.11)$$

The quantity $(z(k+1) - H_{k+1}\hat{x}(k+1/k))$ is termed the *residual.* The matrix K is the *Kalman gain*, and is found by minimizing the posterior estimate error covariance. The resulting K can be written in many forms. One such form is

$$K_{k+1} = P(k+1/k) H_{k+1}^T S_{k+1}^{-1}, \qquad (2.12)$$

where the matrix $S_{k+1} \in \mathbb{R}^{m\times m}$ is the covariance of the residual $(z(k+1) - H_{k+1}\hat{x}(k+1/k))$:

$$\begin{aligned} S_{k+1} &= E\left[\left(z(k+1) - H_{k+1}\hat{x}(k+1/k)\right)\left(z(k+1) - H_{k+1}\hat{x}(k+1/k)\right)^T\right] && (2.13) \\ &= H_{k+1} P(k+1/k) H_{k+1}^T + R_{k+1}. && (2.14) \end{aligned}$$

The equation describing the posterior error covariance, $P(k+1/k+1)$, can also take many forms. One form is given by

$$P(k+1/k+1) = P(k+1/k) - K_{k+1} H_{k+1} P(k+1/k). \qquad (2.15)$$

There are many possible approaches to estimating the state if the system dynamics are non-linear. In the most straightforward approach, the non-linear system is linearized about the current mean and covariance. The matrices from the linearization are then used as above. This is referred to as an *Extended Kalman filter*, or EKF. For a thorough review of Kalman filters, the reader is referred to [39].

2.2.2 Kalman Filters and Robotic Localization

The following sections review the use of the Kalman filter for mobile robot navigation. Figure 2.4 depicts the basic setup. The robot is located at position x_R, and it can measure the range d and/or the bearing ϕ to each landmark, denoted x_{Li}. The range and bearing measurements allow the robot to calculate the relative position between itself and the landmark.

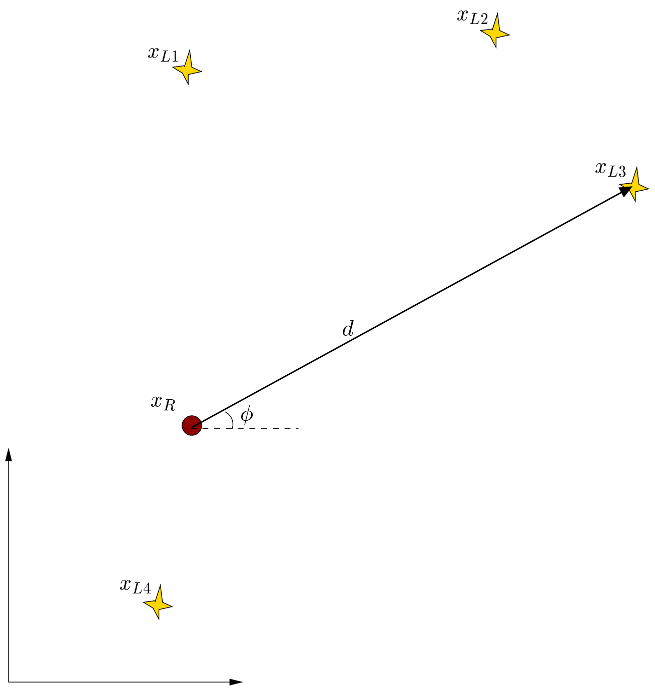

Figure 2.4: Setup of a Kalman filter for mobile robot localization.

2.2.2.1 Localization of a Point Robot Using Landmarks

In the case of a point robot operating in a Cartesian workspace, the motion model assumes the robot has omnidirectional motion capabilities. The robot is given knowledge of the location of N landmarks, as well as the covariances of these landmark locations. It is assumed that the robot can solve the *data association* problem [22], i.e., the robot can distinguish which landmark it is looking at. If this is not the case and the robot cannot tell landmark A from landmark B, then multiple hypothesis methods must be used [6]. The state vector $\mathbf{x}$ contains the positions of both the robot and all of the landmarks, i.e.,

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_R & \mathbf{x}_{L1} & \cdots & \mathbf{x}_{LN} \end{bmatrix}^T, \tag{2.16}$$

where $\mathbf{x}_R$ is the Cartesian position of the robot and $\mathbf{x}_{Li}$ is the Cartesian position of the i^{th} landmark,

$$\mathbf{x}_R = \begin{bmatrix} x_r \\ y_r \end{bmatrix} \text{ and } \mathbf{x}_{Li} = \begin{bmatrix} x_{Li} \\ y_{Li} \end{bmatrix}. \tag{2.17}$$

The discrete time kinematic equation for the robot's movement model is

$$\mathbf{x}_R(k+1) = \mathbf{x}_R(k) + V(k)\Delta t, \tag{2.18}$$

where Δt is the time step between discrete motions, and $V(k)$ represents the robot's velocity at the k^{th} time interval:

$$V(k) = \begin{bmatrix} v_x(k) \\ v_y(k) \end{bmatrix}, \tag{2.19}$$

and $v_x(k)$ and $v_y(k)$ represent the translational velocities at time step k in the x and y directions, respectively. This model assumes that velocity is constant in between samples. Using measurements via the robot's internal odometry, inertial navigation unit, GPS, scan-matching [37, 42], or other means[1], one can propagate the estimate of the robot's state with the following equation:

$$\hat{\mathbf{x}}_R(k+1/k) = \hat{\mathbf{x}}_R(k/k) + V_m(k)\Delta t, \tag{2.20}$$

where

$$V_m(k) = V(k) + w_V(k) = \begin{bmatrix} v_x(k) \\ v_y(k) \end{bmatrix} + \begin{bmatrix} w_{v_x}(k) \\ w_{v_y}(k) \end{bmatrix} \tag{2.21}$$

[1]Some of the methods that provide additional input to the state estimation process may require that additional states be added to the system state defined in Equation (2.16).

are the measurements of the robot's translational velocities. These are corrupted by independent zero-mean white Gaussian noise $w_V(k)$ with covariance

$$Q(k) = E\left[w_V(k)w_V^T(k)\right] = \begin{bmatrix} \sigma_{v_x}^2(k) & 0 \\ 0 & \sigma_{v_y}^2(k) \end{bmatrix}. \tag{2.22}$$

The N landmarks are assumed to be in a fixed, but possibly uncertain, position, i.e.,

$$\hat{\mathbf{x}}_{Li}(k+1) = \hat{\mathbf{x}}_{Li}(k), \qquad i = 1, ..., N, \tag{2.23}$$

where $\hat{\mathbf{x}}_{Li}$ is the estimate of the i^{th} landmark's position. By inspection, the F and G matrices are

$$\begin{aligned} F &= I_{(2N+2)}, \\ G &= \begin{bmatrix} G_R \\ 0_{2N\times 2} \end{bmatrix}, \\ G_R &= \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix}. \end{aligned} \tag{2.24}$$

The zeros in the lower part of the G matrix reflect the assumption that the landmarks are fixed. The estimate error covariance matrix will assume a block structure:

$$P = \begin{bmatrix} P_{RR} & P_{RL} \\ P_{LR} & P_{LL} \end{bmatrix}, \tag{2.25}$$

where P_{RR} is the 2×2 covariance matrix of the robot's position error, P_{LL} is the $2N \times 2N$ matrix of landmark position error covariances, and $P_{RL} = P_{LR}^T$ are the cross-coupling error covariances. It is not assumed that the landmarks are known perfectly, so P_{LL} need not be all zeros. Successive use of Equations (2.20) and (2.10) will propagate the estimate of the robot and landmark states forward in time.

The robot can measure the relative position between itself and a landmark i at any time:

$$\begin{aligned} z(k+1) &= \mathbf{x}_{Li}(k+1) - \mathbf{x}_R(k+1) + n(k+1) && (2.26) \\ &= \begin{bmatrix} x_{Li} - x_r \\ y_{Li} - y_r \end{bmatrix} (k+1) + \begin{bmatrix} n_x \\ n_y \end{bmatrix} (k+1), && (2.27) \end{aligned}$$

where $n(k+1)$ is a zero-mean white Gaussian noise process with covariance

$$R(k+1) = E\left[n(k+1)n(k+1)^T\right]. \tag{2.28}$$

The H matrix is then

$$H(k+1) = \begin{bmatrix} -I_2 & 0_{2\times 2(i-1)} & I_2 & 0_{2\times 2(N-i)} \end{bmatrix}. \tag{2.29}$$

With the H matrix, the Kalman gain (Equation (2.12)) can be computed. The Kalman gain can then be used to update the state estimate and the estimate error covariance.

Figures 2.5 and 2.6 show representative data for the propagation of a Kalman filter. Figure 2.5 depicts a single movement sequence where the robot takes three steps. In this specific run, the robot's position estimate from the first step is particularly bad. This example was chosen to illustrate how large the errors can be. It also shows how the estimate covariance grows after each step.

Figure 2.6 presents an example where the robot takes a single step 500 times from the same starting position. Although the starting position and the true final positions are the same for each trial, the noise values that corrupt the robot's velocity (Equation 2.21) are different for every run. Thus, the final estimates are all slightly different. Figure 2.6 also shows a zoomed-in view of all of the final estimates for the same 500 trials. If the number of trials were increased, the final estimate average would move closer and closer to the true final position. The 99.7 % covariance ellipse plotted in the figure is the mean of the covariances for all trials. As expected, most of the final estimates lie within this ellipse.

Figure 2.7 shows representative results from the use of the update step of a Kalman filter to localize the robot. In this example, the robot takes two steps. The sequence includes both sensing noise and landmark positional uncertainty. After the first step, the landmark is not within the robot's sensing range. Thus, the update step does not correct the robot's

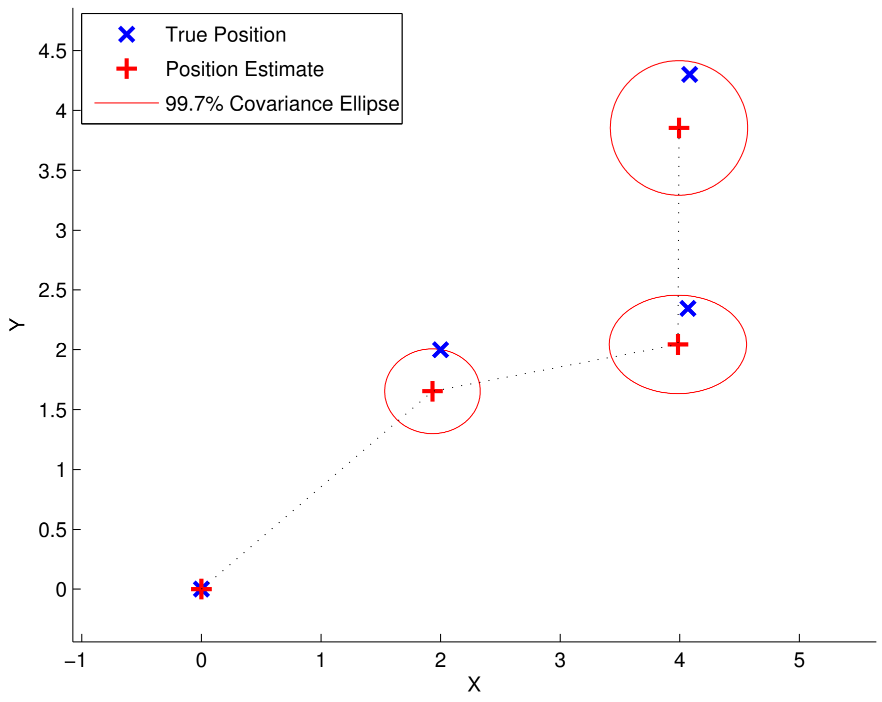

Figure 2.5: A three-step propagation example. This shows how the estimate error continues to grow.

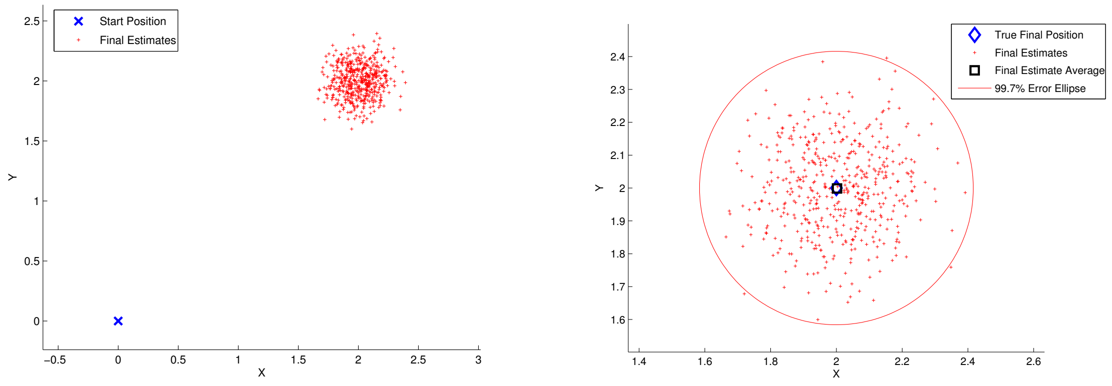

Figure 2.6: A single-step propagation example showing multiple trials.

position estimate. One can see that both the pre-update and post-update estimates are the same, and the 3σ covariance ellipses are also the same. After the second step, however, the robot is close enough to the landmark to use it for localization. Because of odometric errors, the robot's pre-update estimate error is well over 0.5 units. After the update, the absolute error in the estimate is greatly reduced, as is the estimate error covariance. The update step also reduced the error covariance in the landmark position estimate (not shown in the figure). When using a Kalman filter to localize a mobile robot, there is nothing special about the robot relative to the landmarks. The robot can be thought of as a moving landmark for which the dynamics are known.

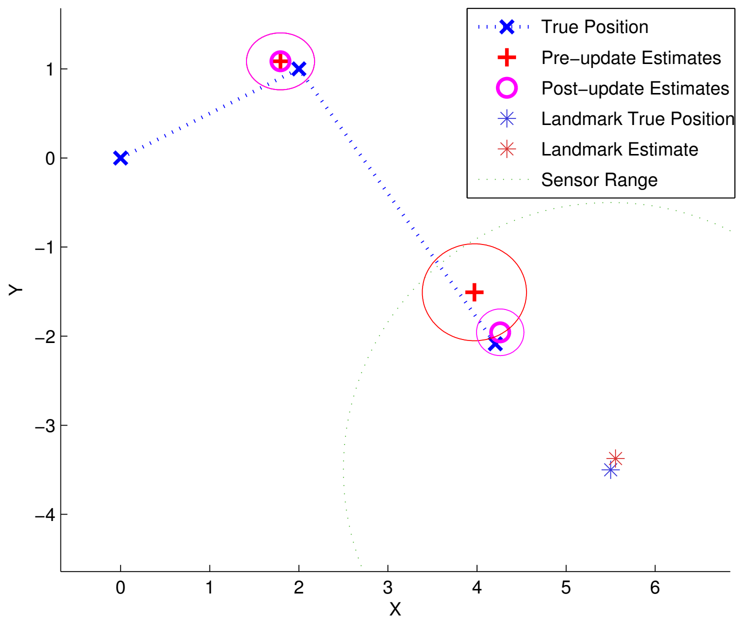

Figure 2.7: Example data illustrating using a landmark for localization.

Chapter 3

Weighted Scan Matching

3.1 Introduction

This chapter introduces a weighted range sensor data matching algorithm to estimate a robot's displacement between the configurations where dense two-dimensional range scans are obtained. This novel algorithm takes into account several important physical phenomena that affect range sensing accuracy that have been neglected in prior work. The experiments in Section 3.6 show that this algorithm is not only efficient, but appreciably more accurate than non-weighted matching methods, such as that of Ref. [37]. Moreover, by computing a more realistic covariance of the displacement estimates, the weighted matching algorithm provides a better basis for fusion of these estimates with odometric and/or inertial measurements [45]. The fused estimates can subsequently be used to support localization and mapping tasks. This work was performed jointly with Samuel Pfister and Stergios Roumeliotis.

To understand the content of this chapter and its contributions best, the basic problem, how the solution differs from previous ones, and the generality of this approach are described. The focus is on mobile robots operating in planar environments. It is assumed that the robot is equipped with a dense planar range sensor (e.g., a laser range scanner). As discussed in Section 3.2.4, on-board odometry is useful, but not essential.

The robot starts at an initial configuration, g_1, and moves through a sequence of configurations, g_i, $i = 2, \ldots, m$. Here $g_i \in SE(2)$ denotes the robot's position and orientation relative to a fixed reference frame, g_0. It is assumed that at each pose, the robot measures the range to the boundary of its nearby environment along rays that are separated by a

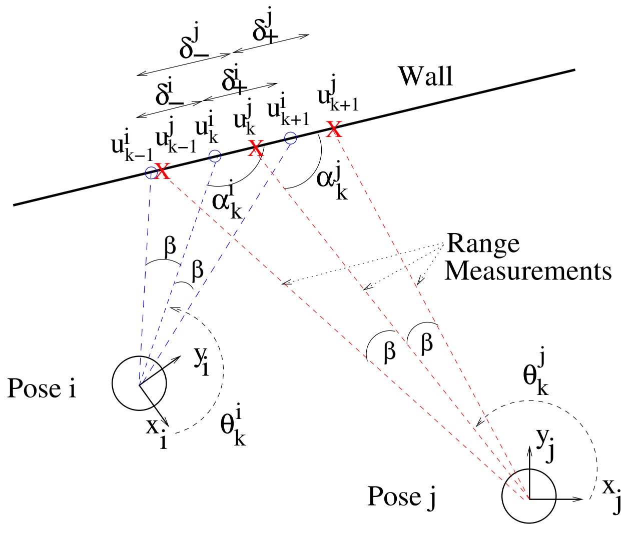

Figure 3.1: Geometry of the range sensing process. The robot acquires dense range scans in poses i and j. The circles represent robot position, while the x-y axes denote the robot's body fixed reference frames.

uniform[1] angle, β (see Figure 3.1). As described below, various uncertainties in this range measurement are accounted for.

Let the set of Cartesian coordinates of the n_i scan points taken in the i^{th} robot pose be denoted by $\{\vec{u}_k^i\}$, $k = 1, \ldots, n_i$. The scan point coordinates are described in the robot's body fixed reference frame. Typically, the Cartesian coordinate of the scan point is derived from range data according to the expression

$$\vec{u}_k^i = \begin{bmatrix} x_k^i \\ y_k^i \end{bmatrix} = l_k^i \begin{bmatrix} \cos\theta_k^i \\ \sin\theta_k^i \end{bmatrix}, \tag{3.1}$$

where l_k^i is the measured distance to the environment's boundary along the k^{th} measuring ray. The measuring ray is oriented in the direction denoted by θ_k^i, where θ_k^i is the angle made by the k^{th} measuring ray with respect to the x-axis of the body fixed reference frame (see Figure 3.1).

The main goal is to accurately estimate the robot's displacement between poses by matching range data obtained in sequential poses. This displacement estimate can be used as the basis for a form of odometry, or fused with conventional odometry and/or inertial measurements to obtain better relative robot pose estimates. In turn, these estimates can

[1]The extension to non-uniform angle β is straightforward.

support localization and mapping procedures. First, assume that the range scans at poses i and j have a sufficient number of corresponding points to be successfully matched (see Section 3.4). Let $\{\vec{u}_k^i, \vec{u}_k^j\}$ for $k = 1, \ldots, n_{ij}$ be the set of corresponding matched scan point pairs, where n_{ij} is the number of corresponding pairs. From these pairs, the relative displacement between poses i and j: $g_{ij} = g_i^{-1} g_j = (R_{ij}, p_{ij})$, will be estimated where

$$R_{ij} = \begin{bmatrix} \cos\phi_{ij} & -\sin\phi_{ij} \\ \sin\phi_{ij} & \cos\phi_{ij} \end{bmatrix}, \qquad \vec{p}_{ij} = \begin{bmatrix} x_{ij} \\ y_{ij} \end{bmatrix}, \tag{3.2}$$

i.e., the displacement between poses i and j is described by a translation, (x_{ij}, y_{ij}), and a rotation, ϕ_{ij}.

Next, the covariance, P^{ij}, of the displacement estimate is calculated. This covariance has two main uses. First, it reflects the quality of the displacement estimates. Large diagonal elements of the covariance matrix indicate increased uncertainty. Any localization process should be aware of the level of confidence in its computed pose estimates. Second, the covariance is needed when combining displacement estimates with measurements provided by other sensors. More accurate and realistic estimates of the contributing covariances lead to more accurate overall estimates in a sensor fusion algorithm, such as a Kalman filter.

This approach differs from prior work in that the contribution of each scan point to the final displacement estimate is individually weighted according to that point's specific uncertainty. The scan point uncertainties are estimated using sensor measurement noise models, as well as models of specific geometric issues within the matching process itself. Figures 3.1 and 3.2 illustrate these issues. Figure 3.1 depicts a situation where a range sensor (e.g., a laser range finder) samples points on a nearby wall. The boundary points sampled in pose i are indicated by circles, and labeled by $\vec{u}_{k-1}^i$, $\vec{u}_k^i$, and $\vec{u}_{k+1}^i$. The nearby boundary points sampled in pose j are indicated by X's and are labeled by $\vec{u}_{k-1}^j$, $\vec{u}_k^j$, and $\vec{u}_{k+1}^j$. Prior range matching methods (e.g., [14, 17, 57]) have made the simplifying assumption that the range scans of different poses sample the environment's boundary at *exactly* the same points—i.e., point $\vec{u}_k^i$ is assumed to be exactly the same point as $\vec{u}_k^j$, etc. *This assumption is generally not true.* Here, this *correspondence error* is modelled and its effect incorporated into the matching algorithm.

As described in Sections 3.3.1 and 3.3.3, the range measurements are corrupted by

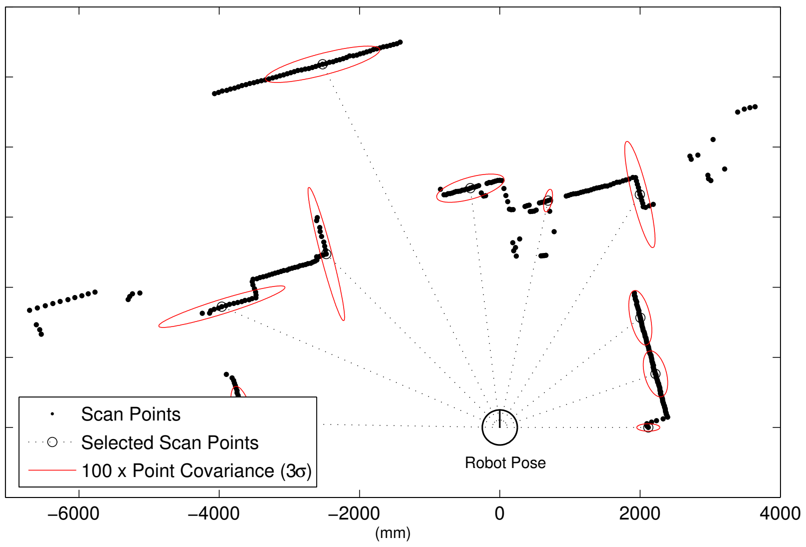

Figure 3.2: Representation of the uncertainty of selected range scan points.

noise and possibly a bias term that is a function of the range sensing direction, θ_k^i, and the sensor beam's incidence angle, α_k^i (Figure 3.1). Figure 3.2 shows the 95% confidence level ellipses associated with the covariance estimates (calculated using the methods that are introduced later) of selected data points from an actual laser range scan. The wide variation in uncertainties seen in Figure 3.2 strongly suggests that not all range data points are of equal precision. Hence, the potentially large variability should be taken into account in the estimation process. While the existence of these uncertainty sources has previously been suggested [1, 2, 3, 5, 14], this algorithm is the first to explicitly model and account for their effects within the estimation process. Some prior works have no explicit noise modeling (e.g., [17]), or apply a uniform uncertainty to all contributing points. The most complete existing methods, [5] and [36], employ statistical methods to calculate displacement estimate uncertainty. These methods do not take sensor uncertainty models into account in the displacement estimation process, and use an unweighted assumption for the contributing points. Also, [5] and [36] do not use any specific sensor noise characteristics as a basis for calculating uncertainty. Instead, they use a numerical sample of perturbations to extract an estimate of covariance. Significant improvements over previous unweighted methods are demonstrated by developing physically based uncertainty models for each individual point and incorporating these models in both the displacement estimation process and the covariance calculation.

The basic principle behind this new approach generally applies to any case of dense range data, such as sonars, infrareds, cameras, radars, etc. The basic weighted matching formulation and its solution given in Section 3.2 are independent of any sensor specifics. To use the general results, specific models of sensor uncertainty are needed. These detailed sensor models are developed in Section 3.3. Since some of the assumptions underlying these sensor models are best suited to laser range scanners, the application of the detailed sensor model formulas is best suited to the use of laser scanners in indoor environments (though they can be extended to structured outdoor environments). However, the general approach of Section 3.2 should work for other range sensors and other operating environments with reasonable modifications to the sensor models.

This chapter is structured as follows: Section 3.2 describes a general weighted point feature matching problem and its solution. Section 3.3 develops correspondence and range measurement error models. Sections 3.4 and 3.5 summarize the point pairing selection and sensor incidence angle estimation procedures. Experiments in Section 3.6 demonstrate the algorithm's accuracy, robustness, and convergence range. Direct comparisons with previous methods (e.g., [36, 37]) validate the effectiveness of this approach.

3.2 The Weighted Range Sensor Matching Problem

This section describes a general point feature matching problem and its basic solution.

3.2.1 The Measurement Model

Let the sets of Cartesian range scan data points acquired in poses i and j be denoted by $\{\vec{u}_k^i\}$ and $\{\vec{u}_k^j\}$, respectively. These measurements will be imperfect. Let $\{\vec{r}_k^i\}$ and $\{\vec{r}_k^j\}$ be the "true" Cartesian scan point locations. The measurements can generally be decomposed into the following terms:

$$\begin{aligned} \vec{u}_k^i &= \vec{r}_k^i + \delta\vec{u}_k^i + \vec{b}_k^i \\ \vec{u}_k^j &= \vec{r}_k^j + \delta\vec{u}_k^j + \vec{b}_k^j, \end{aligned} \tag{3.3}$$

where $\delta\vec{u}_k^i$ and $\delta\vec{u}_k^j$ represent noise or uncertainty in the range measurement process, and $\vec{b}_k^i$ and $\vec{b}_k^j$ denote the possible range measurement "bias." These noise and bias terms are discussed in more detail in Sections 3.3.1 and 3.3.3. The term $\delta\vec{u}_k$ is typically well modelled by a zero-mean Gaussian noise process. The bias $\vec{b}_k$ is an unknown offset that can be approximated by a term, [2] $\vec{o}_k$ corrupted by a zero-mean additive Gaussian noise, $\delta\vec{b}_k$ [2]. The covariance of this noise component reflects the level of confidence in the value $\vec{o}_k$. Contingent on this approximation, $\vec{b}_k^i$ and $\vec{b}_k^j$ take the form

$$\vec{b}_k^i = \vec{o}_k^i + \delta\vec{b}_k^i; \qquad \vec{b}_k^j = \vec{o}_k^j + \delta\vec{b}_k^j. \tag{3.4}$$

Let $(\vec{u}_k^i, \vec{u}_k^j)$ be points that correspond in the range scans at poses i and j. As shown in Figure 3.1, these points are not necessarily the same physical point, but the closest corresponding points. Accounting for the fact that scan data is measured in a robot-fixed frame, the error between the two corresponding points is

$$\varepsilon_k^{ij} = \vec{u}_k^i - R_{ij}\vec{u}_k^j - p_{ij} \tag{3.5}$$

for a given displacement (R_{ij}, p_{ij}) between poses. Substituting Equation 3.3 into Equation 3.5 results in

$$\varepsilon_k^{ij} = \underbrace{(\vec{r}_k^i - R_{ij}\vec{r}_k^j - p_{ij})}_{(i)} + \underbrace{(\delta\vec{u}_k^i - R_{ij}\delta\vec{u}_k^j)}_{(ii)} + \underbrace{(\vec{b}_k^i - R_{ij}\vec{b}_k^j)}_{(iii)}. \tag{3.6}$$

A relative pose estimation algorithm aims to estimate the displacement $g_{ij} = (R_{ij}, p_{ij})$ that suitably minimizes Equation 3.6 over the set of all correspondences. If the dense range scans do sample the exact same boundary points, then $\vec{r}_k^i - R_{ij}\vec{r}_k^j - p_{ij} = 0$ when R_{ij} and p_{ij} assume their proper values. However, $\vec{r}_k^i$ and $\vec{r}_k^j$ generally do not correspond to the same boundary point. Therefore, term (i) in Equation 3.6 is the *correspondence error*, denoted by c_k^{ij}:

$$c_k^{ij} = \vec{r}_k^i - R_{ij}\vec{r}_k^j - p_{ij}. \tag{3.7}$$

The matching error ε_k^{ij} for the k^{th} corresponding point is also a function of: (ii) the error due to the measurement process noise, and (iii) the measurement bias error.

[2]The value of $\vec{o}_k$ can be determined by statistical analysis of measurement data.

For the sake of simplicity, the bias offsets are ignored for now (i.e., assume that $\vec{b}_k^i = \vec{b}_k^j{=}0$), but their effect will be considered again in Section 3.3.3.

3.2.2 A General Covariance Model

For subsequent developments, a generalized expression for the covariance of the measurement errors is needed:

$$\begin{aligned} P_k^{ij} &\triangleq E\left[\varepsilon_k^{ij}(\varepsilon_k^{ij})^T\right] \qquad (3.8)\\ &= E\left[(c_k^{ij} + \delta\vec{u}_k^i - R_{ij}\delta\vec{u}_k^j)(c_k^{ij} + \delta\vec{u}_k^i - R_{ij}\delta\vec{u}_k^j)^T\right], \end{aligned}$$

where $E[\cdot]$ is the expectation operator. Recall that bias effects are ignored for now. P_k^{ij} captures the uncertainty in the error between corresponding range point pairs. Because the range measurement noise is assumed to be zero mean, Gaussian, and independent across measurements, $E[\delta\vec{u}_k^i(\delta\vec{u}_k^j)^T] = E[\delta\vec{u}_k^j(\delta\vec{u}_k^i)^T] = 0$. Practically speaking, one would expect that the range measurement noise of the k^{th} scan point in pose i to be uncorrelated with the measurement noise of the k^{th} corresponding range point in pose j. Therefore, this is a fine assumption in practice.

The correspondence error, c_k^{ij}, is a deterministic variable that is a function of the geometry of the robot's surroundings. However, since the geometry of the environment is not assumed to be known ahead of time, a reasonable *probabilistic approximation* is made to this term, which accounts for the fact that the geometry of the surroundings is unknown a priori. In this probabilistic approximating model, the correspondence error and sensor measurement error terms are independent. Therefore, $E[c_k^{ij}(\delta\vec{u}_k^i)^T] = E[c_k^{ij}(\delta\vec{u}_k^j)^T] = E[\delta\vec{u}_k^i(c_k^{ij})^T] = E[\delta\vec{u}_k^j(c_k^{ij})^T] = 0$. See Section 3.3.2 for a more detailed discussion.

Under these assumptions, the covariance of the matching error at the k^{th} point correspondence of poses i and j becomes:

$$\begin{aligned} P_k^{ij} &\triangleq E\left[\varepsilon_k^{ij}(\varepsilon_k^{ij})^T\right] = E\left[c_k^{ij}(c_k^{ij})^T\right] + E\left[\delta\vec{u}_k^i(\delta\vec{u}_k^i)^T\right]\\ &+ R_{ij}E\left[\delta\vec{u}_k^j(\delta\vec{u}_k^j)^T\right]R_{ij}^T\\ &= {}^{C}P_k^{ij} + {}^{N}P_k^i + R_{ij}{}^{N}P_k^jR_{ij}^T \qquad (3.9)\\ &= Q_k^{ij} + R_{ij}S_k^{ij}R_{ij}^T \qquad (3.10) \end{aligned}$$

where

$$\begin{aligned}
{}^{C}P_k^{ij} &= \text{covariance associated with the approximating correspondence error model,} \\
{}^{N}P_k^{i} &= \text{measurement noise covariance of the } k^{th} \text{ scan point in the } i^{th} \text{ pose,} \\
{}^{N}P_k^{j} &= \text{measurement noise covariance of the } k^{th} \text{ scan point in the } j^{th} \text{ pose,} \\
Q_k^{ij} &\triangleq {}^{C}P_k^{ij} + {}^{N}P_k^{i}, \\
S_k^{ij} &\triangleq {}^{N}P_k^{j}.
\end{aligned}$$

The matrices Q_k^{ij} and S_k^{ij} represent the configuration-independent and configuration-dependent terms of P_k^{ij}. As shown below, the correspondence errors depend on the sensor beam's incidence angle. The noise covariances will also be a function of the variables θ_k^i, θ_k^j, l_k^i, and l_k^j. Thus, the covariance matrix P_k^{ij} is expected to vary for each scan point pair (see Figure 3.2 for an illustration). It is not suitable to assume that P_k^{ij} is a constant matrix for all scan point pairs, as has been done in prior work (e.g., [36, 37]).

3.2.3 Displacement Estimation via Maximum Likelihood

A maximum likelihood (ML) framework is used to formulate a general strategy for estimating the robot's displacement from a set of non-uniformly weighted point correspondences. Let $\mathcal{L}(\{\varepsilon_k^{ij}\}|g_{ij})$ denote the *likelihood function* that captures the likelihood of obtaining the set of matching errors $\{\varepsilon_k^{ij}\}$ given a displacement, g_{ij}. Under the assumptions above, the $k = 1, \ldots, n_{ij}$ range pair measurements are independent[3] and the likelihood can be written as a product:

$$\mathcal{L}(\{\varepsilon_k^{ij}\}|g_{ij}) = \mathcal{L}(\varepsilon_1^{ij}|g_{ij})\mathcal{L}(\varepsilon_2^{ij}|g_{ij}) \cdots \mathcal{L}(\varepsilon_{n_{ij}}^{ij}|g_{ij}). \tag{3.11}$$

Recall that the measurement noise is considered to be a zero-mean Gaussian process. Finally, as shown in Section 3.3.2, the correspondence noise can be approximated by a zero-mean Gaussian process. Neglecting the bias offset for the moment (see Section 3.3.3), the

[3]Possible dependencies of these measurements will be briefly considered in Section 3.3.2. Generally, the only effect that will lead to dependence is possible couplings in the correspondence error that arise if the geometry of the environment is a priori known.

above assumptions imply that $\mathcal{L}(\{\varepsilon_k^{ij}\}|g_{ij})$ takes the form

$$\mathcal{L}(\{\varepsilon_k^{ij}\}|g_{ij}) = \prod_{k=1}^{n_{ij}} \frac{e^{-\frac{1}{2}(\varepsilon_k^{ij})^T(P_k^{ij})^{-1}\varepsilon_k^{ij}}}{2\pi\sqrt{\det P_k^{ij}}} = \frac{e^{-M^{ij}}}{D^{ij}}, \tag{3.12}$$

where

$$M^{ij} = \frac{1}{2}\sum_{k=1}^{n_{ij}}(\varepsilon_k^{ij})^T(P_k^{ij})^{-1}\varepsilon_k^{ij}, \tag{3.13}$$

$$D^{ij} = \prod_{k=1}^{n_{ij}} 2\pi\sqrt{\det P_k^{ij}}. \tag{3.14}$$

The optimal displacement estimate is the one that maximizes the value of $\mathcal{L}(\{\varepsilon_k^{ij}\}|g_{ij})$ with respect to displacement. One can use any numerical optimization scheme to obtain this displacement estimate. Note, however, that maximizing Equation 3.12 is equivalent to maximizing the log-likelihood function:

$$ln[\mathcal{L}(\{\varepsilon_k^{ij}\}|g_{ij})] = -M^{ij} - ln(D^{ij}). \tag{3.15}$$

From a numerical point of view, it is often preferable to work with the log-likelihood function.

Before discussing the solution to this estimation problem, this formulation is compared with prior work. Most prior algorithms that take an "unweighted" approach to the displacement estimation problem assume that all of the covariance matrices P_k^{ij} are uniformly the 2×2 identity matrix. Consequently, the maximization of the log-likelihood function reduces to a standard least-squares problem. However, as Figure 3.2 and experiments in Section 3.6 show, such a simplistic covariance approximation for all data points is typically not a theoretically sound one. Although [57] allowed for a scalar weighting term, no guidance was provided on how to select the value of the scalar.

The weighted estimation problem has some inherent structure that leads to efficiency in the maximization procedure. Appendix A.1 proves that the optimal estimate of the robot's translation can be computed using the following closed form expression:

Proposition 3.1. *The weighted scan match translational displacement estimate, $\hat{p}_{ij}$, is*

$$\hat{p}_{ij} = P_{pp} \sum_{k=1}^{n_{ij}} \left((P_k^{ij})^{-1} (\vec{u}_k^i - \hat{R}_{ij} \vec{u}_k^j) \right), \tag{3.16}$$

where $\hat{R}_{ij} = \hat{R}_{ij}(\hat{\phi}_{ij}^-)$ is the estimated rotational matrix calculated with the current estimate of the orientation displacement $\hat{\phi}_{ij}$, and P_{pp} is given by the formula

$$P_{pp} = \left(\sum_{k=1}^{n_{ij}} (P_k^{ij})^{-1} \right)^{-1}. \tag{3.17}$$

An exact closed form expression for estimating the rotational displacement ϕ_{ij} does not exist. Nonetheless, there are two efficient approaches to computing this estimate. In the first approach, the translational estimate of Equation 3.16 is substituted into Equation 3.12 (or equivalently, into Equation 3.15). Since the resulting expression is a function of the single variable, ϕ_{ij}, the estimation procedure reduces to numerical maximization over a single scalar variable, ϕ_{ij}, for which there are many efficient algorithms.

Alternatively, one can develop the following second order iterative solution to the non-linear estimation problem (Appendix A.2):

Proposition 3.2. *The weighted scan match rotational displacement estimate is updated as $\hat{\phi}_{ij}^+ = \hat{\phi}_{ij}^- + \delta\hat{\phi}_{ij}$, where*

$$\delta\hat{\phi}_{ij} \simeq -\frac{\sum_{i=1}^{n_{ij}} p_k^T (P_k^{ij})^{-1} J q_k}{\sum_{k=1}^{n_{ij}} q_k^T J (P_k^{ij})^{-1} J q_k}, \tag{3.18}$$

where

$$J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \qquad \begin{array}{rcl} q_k & = & \hat{R}_{ij} \vec{u}_k^j \\ p_k & = & \vec{u}_k^i - \hat{p}_{ij} - \hat{R}_{ij} \vec{u}_k^j \end{array}. \tag{3.19}$$

Using various experimental data, this approximation has been found to agree with the exact numerical solution up to five significant digits. Furthermore, it is computationally more efficient to implement.

3.2.4 The Algorithm and Its Initial Conditions

Propositions 3.1 and 3.2 suggest an iterative algorithm for estimating displacement. An initial guess $\hat{\phi}_{ij}^-$ for ϕ_{ij} is chosen. A translation estimate $\hat{p}_{ij}$ is computed using Proposition 3.1. This estimate can be used with an exact numerical optimization procedure or with

Proposition 3.2 to update the current rotational estimate $\hat{\phi}_{ij}^{-}$. The improved $\hat{\phi}_{ij}^{+}$ is the basis for the next iteration. The iterations stop when a convergence criterion is reached.

The initial guess, $\hat{\phi}_{ij}^{-}$, will usually be derived from an odometry estimate. However, odometry is not necessary for the method to work. An open loop estimate of the robot's displacement based on the known control inputs that generate the displacement will often provide sufficient accuracy for an initial guess. It is shown in Section 3.6.1 that the algorithm's performance is not hampered by large errors in the initial value of the displacement used as a seed for the algorithm. Note that if odometry does provide the initial guess, there will be no correlation between the estimate arising from the scan matching algorithm and the odometry estimate since the accuracy of the latter is not considered in the estimation process. This simplifies subsequent fusion of these estimates, which may be desired for some applications.

An iterative algorithm is preferred for two reasons. First, non-linear ML problems are suited to iterative computation. Second, the correct correspondence between point pairs cannot be guaranteed in the point correspondence problem (see Section 3.4). This is especially true in the first few algorithm iterations, where some inaccurate initial pairings are unavoidable. The iterative approach allows for continual readjustment of the point correspondences as the iterations proceed.

3.2.5 Covariance of the Displacement Estimation Error

Letting $\tilde{p}_{ij} = p_{ij} - \hat{p}_{ij}$, $\tilde{\phi}_{ij} = \phi_{ij} - \hat{\phi}_{ij}$ (i.e., $\tilde{p}_{ij}$, $\tilde{\phi}_{ij}$ are the translational and the rotational displacement error estimates), a direct calculation yields the following:

Proposition 3.3. *The covariance of the displacement estimate is*

$$P^{ij} = \begin{bmatrix} P_{pp} & P_{p\phi} \\ P_{\phi p} & P_{\phi\phi} \end{bmatrix} = \begin{bmatrix} E\{\tilde{p}_{ij}\tilde{p}_{ij}^T\} & E\{\tilde{p}_{ij}\tilde{\phi}_{ij}^T\} \\ E\{\tilde{\phi}_{ij}\tilde{p}_{ij}^T\} & E\{\tilde{\phi}_{ij}\tilde{\phi}_{ij}^T\} \end{bmatrix}$$

with

$$
\begin{aligned}
P_{p\phi} &= \frac{1}{r_T}\left(\sum_{k=1}^{n_{ij}}(P_k^{ij})^{-1}\right)^{-1}\sum_{k=1}^{n_{ij}}\left((P_k^{ij})^{-1}Jq_k\right), && (3.20)\\
P_{\phi p} &= P_{p\phi}^T, && (3.21)\\
P_{\phi\phi} &= \frac{1}{r_T}, && (3.22)\\
r_T &= -\sum_{k=1}^{n_{ij}} q_k^T J (P_k^{ij})^{-1} J q_k, && (3.23)
\end{aligned}
$$

and P_{pp} is given by Equation 3.17.

The proofs for Proposition 3.3 are given in Appendix A.3. For a given sensor, one must derive appropriate uncertainty models, which are then substituted into the above procedure.

Note 1: The matrix $-J\ (P_k^{ij})^{-1}\ J = \frac{1}{\det(P_k^{ij})}\ P_k^{ij}$ in Equation 3.23 is a positive definite matrix. Therefore $P_{\phi\phi}$ is a positive number.

Note 2: From Equations 3.22 and (3.23), for bounded covariance ($\|(P_k^{ij})^{-1}\| < K\ ,\ \ 0 < K < \infty$):

$$\lim_{\|\vec{u}_k^j\|\to\infty} P_{\phi\phi} = \lim_{\|q_k\|\to\infty} P_{\phi\phi} = 0.$$

This result leads to the following corollary:

Corollary 3.4. *Matching of distant features (in the limit features at infinite distance from the current location) minimizes the expected error in the orientation displacement estimate. In the limit, the relative orientation error is zero.*

Note 3: Since all matrices $P_k^{ij},\ k = 1, \ldots, n_{ij}$, in Equation 3.17 are positive definite, the covariance of the translational estimate, P_{pp}, can be written as

$$
\begin{aligned}
(P_{pp})^{-1} &= \sum_{k=1}^{n_{ij}}(P_k^{ij})^{-1} > (P_k^{ij})^{-1} \Leftrightarrow \\
P_{pp} &< P_k^{ij}\ ,\ \ k = 1, \ldots, n_{ij}. && (3.24)
\end{aligned}
$$

Here the notation $X > Y$ indicates that the difference $X - Y$ is a positive definite matrix. Equation 3.24 leads to the following corollary:

Corollary 3.5. *Let $U^{ij} = \min_{k=1,\ldots,n_{ij}} P_k^{ij}$ denote the minimum covariance over all corresponding point pairs. The translational covariance estimate, P_{pp}, given by Equation 3.17 is bounded above by U^{ij}: $P_{pp} < U^{ij}$.*

This corollary states that the covariance of the translational estimate will always be less than the best single covariance associated with any corresponding point pair.

3.3 Scan Matching Error/Noise Models

In order to derive explicit expressions for the covariances of Equation 3.10, this section develops models for the errors inherent in the range scan matching process. Most of the models are quite general, though a few assumptions are made at some points that are most appropriate for laser range scanners.

3.3.1 Measurement Process Noise

Many range sensing methods are based on the time of flight (e.g., ultrasound and some laser scanners) or modulation of emitted radiation [2, 3]. The circuits governing these measurement methods are subject to noise. These effects can often be well modelled in a simple way, enabling the computation of the covariance contributions, ${}^{N}P_k^i$, and ${}^{N}P_k^j$. The computation of ${}^{N}P_k^i$ is focused on, as the one for ${}^{N}P_k^j$ is completely analogous.

Recall the polar representation of scan data, Equation 3.1. Let the range measurement, l_k^i, be comprised of the "true" range, L_k^i, and an additive noise term, ε_l: $l_k^i = L_k^i + \varepsilon_l$. The noise, ε_l, is assumed to be a zero-mean Gaussian random variable with variance σ_l^2 (see e.g., Ref. [2] for justification of this assumption). Also assume that error or uncertainty exists in the measurement θ_k^i. That is, the actual scan angle differs from the reported or assumed angle of the scan snapshot. Thus, $\theta_k^i = \Theta_k^i + \varepsilon_\theta$, where Θ_k^i is the "true" angle of the k^{th} scan direction, and ε_θ is again a zero-mean Gaussian random variable with variance σ_θ^2. Hence:

$$\vec{r}_k^{\,i} = L_k^i \begin{bmatrix} \cos\Theta_k^i \\ \sin\Theta_k^i \end{bmatrix} = (l_k^i - \varepsilon_l) \begin{bmatrix} \cos(\theta_k^i - \varepsilon_\theta) \\ \sin(\theta_k^i - \varepsilon_\theta) \end{bmatrix}. \qquad (3.25)$$

For small ε_θ, ε_l (which is a good approximation for most laser scanners), expanding Equation

3.25 and using the relationship $\delta\vec{u}^i_k = \vec{u}^i_k - \vec{r}^i_k$ yields

$$\delta\vec{u}^i_k = (l^i_k)\varepsilon_\theta \begin{bmatrix} -\sin\theta^i_k \\ \cos\theta^i_k \end{bmatrix} + \varepsilon_l \begin{bmatrix} \cos\theta^i_k \\ \sin\theta^i_k \end{bmatrix}. \tag{3.26}$$

Assuming that ε_θ and ε_l are independent, then

$$\begin{aligned} {}^N P^i_k &= E[\delta\vec{u}^i_k(\delta\vec{u}^i_k)^T] = \frac{(l^i_k)^2\sigma^2_\theta}{2} \begin{bmatrix} 2\sin^2\theta^i_k & -\sin 2\theta^i_k \\ -\sin 2\theta^i_k & 2\cos^2\theta^i_k \end{bmatrix} \\ &+ \frac{\sigma^2_l}{2} \begin{bmatrix} 2\cos^2\theta^i_k & \sin 2\theta^i_k \\ \sin 2\theta^i_k & 2\sin^2\theta^i_k \end{bmatrix}. \end{aligned} \tag{3.27}$$

The quantities θ^i_k and l^i_k are the ones measured by the laser scanner.

3.3.2 Correspondence Error

Here, the correspondence error described in Section 3.2.1 is analyzed, and a probabilistic approximation to this error is derived. The derivation assumes that the sensor beam strikes an environmental boundary that is locally a straight- line segment (Figure 3.1). However, this derivation can be extended to other boundary geometries, or it can serve as an excellent tangent approximation for moderately curved boundaries.

We first develop a formula for the maximum possible correspondence error that can occur due to the fact that the exact same boundary points are not sampled in two successive range scans. Consider how nearby scan points will be matched in the vicinity of points $\vec{u}^i_k$ and $\vec{u}^j_k$ in Figure 3.1. Let

$$\delta^i_+ = ||\vec{u}^i_{k+1} - \vec{u}^i_k||, \qquad \delta^i_- = ||\vec{u}^i_k - \vec{u}^i_{k-1}|| \tag{3.28}$$

denote the distance to the adjacent scan points (from pose i's scan) near the candidate matching point $\vec{u}^i_k$ (see Figure 3.1). Similarly, let $\delta^j_+ = ||\vec{u}^j_{k+1} - \vec{u}^j_k||$ and $\delta^j_- = ||\vec{u}^j_k - \vec{u}^j_{k-1}||$ denote the distances to the adjacent scan points (from pose j's scan) near the candidate matching point $\vec{u}^j_k$. The maximum distance (or error) between any pair of points that are chosen to be in correspondence will be half of the minimum distance between adjacent scan points. If the error is greater than this value, the point will be matched to another point, or it will not be matched at all. On average, this error will be the minimum of $(\delta^i_+ + \delta^i_-)/4$

or $(\delta^j_+ + \delta^j_-)/4$. Simple geometric analysis of Figure 3.1 shows that

$$\begin{aligned}\frac{\delta^i_+ + \delta^i_-}{4} &= \frac{l^i_k \sin\beta}{4}\left[\frac{1}{\sin(\alpha^i_k+\beta)} + \frac{1}{\sin(\alpha^i_k-\beta)}\right] \\ &= \frac{l^i_k \sin\beta}{2}\left[\frac{\sin\alpha^i_k \cos\beta}{\sin^2\alpha^i_k - \sin^2\beta}\right]. \end{aligned} \tag{3.29}$$

Substituting j for i yields the analogous formula for $(\delta^j_+ + \delta^j_-)/4$.

We now propose a probabilistic model for the correspondence errors, and develop explicit formulas for its first two moments. For simplicity, and without loss of generality, let the robot be situated so that $\delta^i_+ + \delta^i_- < \delta^j_+ + \delta^j_-$ (i.e., the correspondence error is defined by pose i). Recall the correspondence error formula of Equation 3.6: $c^{ij}_k = \vec{r}^i_k - R_{ij}\vec{r}^j_k - p_{ij}$. Letting x be the position along the boundary relative to $\vec{u}^i_k$, the correspondence error is locally a function of x. With no correspondence error, $x = 0$. Since the correspondence error is locally collinear with the boundary's tangent, let $\mu^{ij}_k = c^{ij}_k \cdot t_k$ be the projection of c^{ij}_k onto the unit boundary tangent vector, t_k, at $\vec{u}^i_k$. The vector t_k is positive pointing from $\vec{u}^i_k$ to $\vec{u}^i_{k+1}$. Hence, μ^{ij}_k is a signed quantity, and $c^{ij}_k = \mu^{ij}_k t_k$. The expected value (mean) of the error in the interval $x \in [-\delta^i_-, \delta^i_+]$ is

$$E[\mu^{ij}_k] = \int_{-\delta^i_-}^{\delta^i_+} \mu^{ij}_k(x)\mathcal{P}(x)dx, \tag{3.30}$$

where $\mathcal{P}(x)$ is the probability that the k^{th} scan point from pose j will be located at x.

It is assumed that the geometry of the robot's surroundings is not previously known. Therefore, it is not possible to know a priori the probabilistic distribution of the correspondence errors, $\mathcal{P}(x)$. The reasonable assumption that $\mathcal{P}(x)$ has an a priori uniform probability is made. That is, the scan point $\vec{u}^j_k$ that is matched to $\vec{u}^i_k$ could lie anywhere in the interval $[-\delta^i_-, \delta^i_+]$ with no preferred location. Hence $\mathcal{P}(x) = 1/(\delta^i_+ + \delta^i_-)$. Realizing that $\mu^{ij}_k(x) = x$ in the interval $[-\delta^i_-, \delta^i_+]$, evaluation of Equation 3.30 yields

$$\begin{aligned} E[\mu^{ij}_k] &= \frac{(\delta^i_+)^2 - (\delta^i_-)^2}{\delta^i_+ + \delta^i_-} = \delta^i_+ - \delta^i_- \\ &= -2\frac{l^i_k \sin^2\beta \cos\alpha^i_k}{\sin^2\alpha^i_k - \sin^2\beta}. \end{aligned} \tag{3.31}$$

Note that when the incidence angle is not normal ($\alpha^i_k \neq 90^o$), the mean is non-zero. However, since the mean is proportional to $\sin^2 \beta$, this term is negligible when the magnitude of β is small. Hence, the correspondence error can be considered to be a zero-mean quantity when β is small (this holds for the experiments described in Section 3.6). To compute the variance of the correspondence error (using the zero-mean assumption),

$$E[(\mu_k^{ij})^2] = \int_{-\delta_-^i}^{\delta_+^i} \frac{x^2}{\delta_+^i + \delta_-^i} dx = \frac{(\delta_+^i)^3 + (\delta_-^i)^3}{3(\delta_+^i + \delta_-^i)}. \tag{3.32}$$

Letting $\eta_k^i = \alpha_k^i + \theta_k^i$, and keeping the above results in mind, the covariance of the correspondence error, ${}^C P_k^i$ of Equation 3.10, can be found as

$$\begin{aligned} {}^C P_k^i &= E[c_k^{ij} (c_k^{ij})^T] = E[(\mu_k^{ij})^2] t_k t_k^T \\ &= \frac{(\delta_+^i)^3 + (\delta_-^i)^3}{3(\delta_+^i + \delta_-^i)} \begin{bmatrix} \cos^2 \eta_k^i & \cos \eta_k^i \sin \eta_k^i \\ \cos \eta_k^i \sin \eta_k^i & \sin^2 \eta_k^i \end{bmatrix}. \end{aligned} \tag{3.33}$$

Note that this expression is a function of the sensor beam's incidence angle, α_k^i. Section 3.5 discusses how to estimate this quantity from the range scan data.

Because we do not want to assume prior knowledge of the environment's geometry, the correspondence errors are considered to be independent. This assumption is conservative in that no structure in the environment beyond the immediate geometry of the local point pairs is assumed. It would be possible to predict subsequent correspondence errors along a wall (or other regular geometric structure) given the knowledge that the subsequent corresponding point pairs did indeed come from the same exactly straight wall. With a proper line fitting method (e.g., see [41]), the correlations between correspondence errors could be estimated from the line fitting method's uncertainty model.[4]

In general, knowing that adjacent corresponding pairs lie along a common wall will significantly reduce the magnitude of Equation 3.32, which in turn will lead to lower variances for most of the points along the wall. In this case, the correspondence error variance becomes dominated by the uncertainty in the wall's geometry, which in turn is a function of the line fitting method. These effects can fit easily within this framework if desired, leading to even better displacement estimates and tighter estimate covariances. However, a conservative approach is taken where we do not assume that the robot's surrounding geometry is a

[4]In the case of correspondence error correlations, the likelihood model of Equation 3.11 will no longer take a product form. The form of the likelihood model in this case will depend upon the line fitting method.

priori known. Moreover, since the reduction in uncertainty will only occur for points along one line (or other geometric feature), in even modestly complex environments, the amount of precision to be gained by using this approach is unlikely to be worth the complexity of implementing these more advanced methods.

3.3.3 Measurement Bias Effects

Range measurement bias is an artifact of some range sensing methods (e.g., see [2]). Since bias models will strongly depend upon the given range sensing method, it is not possible to give a complete summary of bias models for common sensing methods. Instead, a general approach is considered for calculating the effect of bias on the displacement estimate.

To analyze the bias effect, let $\tilde{\varepsilon}_k^{ij} \triangleq \varepsilon_k^{ij} + \tilde{o}_k^{ij}$, where $\tilde{o}_k^{ij} = \vec{o}_k^{i} - R_{ij}\vec{o}_k^{j}$ is the total constant bias offset effect at the k^{th} correspondence, and ε_k^{ij} is the previously defined matching error (that ignored the constant bias term). Incorporating the bias offsets, the likelihood function takes the form

$$\mathcal{L}(\{\tilde{\varepsilon}_k^{ij}\}|g_{ij}) \quad = \quad \prod_{k=1}^{n_{ij}} \frac{e^{-\frac{1}{2}(\tilde{\varepsilon}_k^{ij}-\tilde{o}_k^{ij})^T(\tilde{P}_k^{ij})^{-1}(\tilde{\varepsilon}_k^{ij}-\tilde{o}_k^{ij})}}{2\pi\sqrt{\det \tilde{P}_k^{ij}}} \tag{3.34}$$

where $\tilde{P}_k^{ij}$ is the covariance matrix with bias uncertainty taken into account:

$$\tilde{P}_k^{ij} = \tilde{Q}_k^{ij} + R_{ij}\tilde{S}_k^{ij}R_{ij}^T, \tag{3.35}$$

where $\tilde{Q}_k^{ij} = Q_k^{ij} + {}^{B}P_k^{i}$ and $\tilde{S}_k^{ij} = S_k^{ij} + {}^{B}P_k^{j}$, with ${}^{B}P_k^{i} = E[\delta\vec{b}_k^{i}(\delta\vec{b}_k^{i})^T]$ and ${}^{B}P_k^{j} = E[\delta\vec{b}_k^{j}(\delta\vec{b}_k^{j})^T]$. That is, the covariance formula is updated to include uncertainty in the bias term. To obtain these results, it is again assumed that the bias noise is uncorrelated with the range measurement noise and the correspondence error (since variance in bias is typically a function of the variability of the surface properties, rather than measurement noise).

Following the derivations that lead to Proposition 3.1, one can show that the translation estimate in this case is

$$\hat{p}_{ij} = \tilde{P}_{pp} \sum_{k=1}^{n_{ij}} \left((\tilde{P}_k^{ij})^{-1}(\vec{u}_k^{i} - \hat{R}_{ij}\vec{u}_k^{j} + \tilde{o}_k^{ij}) \right). \tag{3.36}$$

Formulas analogous to Equation 3.18 can be derived for the orientation estimate as well. The previous covariance formulas take the same structure, with Q_k^{ij} and S_k^{ij} modified to $\tilde{Q}_k^{ij}$ and $\tilde{S}_k^{ij}$ (i.e., to include possible bias uncertainty terms). Clearly, Equation 3.36 shows that bias effects can influence the displacement estimate. However, bias models can be used to compensate for bias effects in the estimate.

3.4 Selection of Point Correspondences

The focus of this work is to improve displacement estimation via more accurate considerations of the noise and uncertainty inherent in the estimation process. However, the displacement estimation process depends upon the ability to successfully match corresponding points from range scans taken in adjacent poses. In order to isolate the benefits of the weighted estimation method, we use a very simple "closest-point" rule similar to the one in [37].

Given two scan sets $\{\vec{u}_k^i\}$ and $\{\vec{u}_k^j\}$, the *outliers* are removed in the first step. These are the points visible in one scan, but not in the other (see [37] for details). After removing the outliers, correspondences between scan point pairs in the two poses are found. For every point in pose i, we search for a corresponding scan point in pose j that satisfies a *range criterion:* The corresponding point must lie within a given distance: $||\vec{u}_k^i - \vec{u}_k^j|| < d$. If no points in pose j satisfy this criterion, then the point is marked as having no correspondence. The parameter d is initially set at a value defined by the error in the initial translation estimate (e.g., the estimated odometry error). Thereafter, to speed convergence, d is monotonically reduced to a value whose order is the maximum point error predicted by the noise model.

It is also possible to establish point correspondences based on a chi-squared analysis of point pairs using the detailed sensor noise models already computed in this method. Though this approach shows promise, in experimental tests we chose to isolate the estimation benefits of this work. Because unweighted scan-matching methods lack the uncertainty models to perform a chi-squared based point correspondence determination process, presented results use the "closest-point" method for all tests, as this leads to the fairest comparison procedure.

3.5 Estimating the Incidence Angle

The correspondence error model of Section 3.3.2 assumes knowledge of each scan point's incidence angle. While any method of incidence angle estimation can be used, we have chosen a method that estimates the local geometry of the scan points using a Hough transform. The Hough transform [15] is a general pattern detection technique that is used to determine an estimate of the supporting line segment about a point. The incidence angle can then be estimated from the configuration of the line segment. In the general Hough transform line finding technique, each scan point $\{x_k, y_k\}$ is transformed into a discretized curve in the Hough space. The transformation is based on the parametrization of a line in polar coordinates with a normal distance from the line to the origin, d_L, and a normal angle, ϕ_L:

$$d_L = x_k \sin(\phi_L) + y_k \cos(\phi_L). \tag{3.37}$$

Values of ϕ_L and d_L are discretized with $\phi_L \in \{0, \pi\}$ and $d_L \in \{-D, D\}$ where D is the maximum sensor distance reading. The Hough space is comprised of a two-dimensional hash table of discrete bins, where each bin corresponds to a single line in the scan point space. For each scan point, the bins in Hough space that correspond to lines passing through that point are incremented. Peaks in the Hough space correspond to lines in the scan data set. As the bins in the Hough space are incremented, we maintain a history of the contributing scan point coordinates in the bin, so that when a peak is determined to represent a line, the contributing set of points can be recovered. The incidence angles can then be estimated for every point in the line.

The algorithm is only precise up to the level of discretization chosen for the line parameters. Both computational complexity and the memory needed for the hash table grow with finer discretization so it is important to establish a reasonable balance between precision and computing resources. For implementation, a line angle measurement precise to the nearest degree is assumed to be adequate for incidence angle estimation. Discretization in distance was set to 10 mm, though this choice of this value is less significant as only the orientation of the fit lines is used.

The Hough transform is not limited only to straight line detection. It can also be used to detect and fit simple curves such as circles and ellipses and even arbitrary shapes [4]. The tangent vectors to these curves (and subsequently the incidence angle) can then easily be estimated from the transform. For most indoor environments the line fitting method is sufficient to determine incidence angles. More accurate line fitting methods (e.g., [41] and references therein) can be used to get more accurate estimates of incidence angle, but the extra computation is typically not balanced by sufficiently better estimation accuracy.

For points that are not found to be clustered into a line, no incidence angle estimate is calculated. These points are weighted only according to the computed measurement noises such that the covariance of the matching error at the k^{th} point correspondence of poses i and j from Equation 3.9 becomes

$$P_k^{ij} \triangleq {}^N P_k^i + R_{ij} {}^N P_k^j R_{ij}^T, \tag{3.38}$$

where the correspondence covariance estimate ${}^C P_k^{ij}$ has been dropped.

3.6 Experiments

This method was implemented on a Nomadics 200 mobile robot equipped with a Sick LMS-200 laser range scanner. This sensor measures the range to points in a plane at every half degree over a 180-degree arc, as seen in Figure 3.2. For the purpose of comparison, an unweighted least-squares scan matching algorithm analogous to that of Lu and Milios [37] was implemented also, and hereafter called the “UWLS.” Both the weighted and unweighted estimation algorithms used the same point correspondence algorithm so that the comparison could fairly focus on the relative merits of both estimation schemes. Section 3.6.1 compares the robustness and accuracy of the algorithms in four different environment geometries. Section 3.6.2 compares results from two longer runs. Section 3.6.3 presents the estimated computational costs of the algorithms, while Section 3.6.4 experimentally explores bias compensation. All experiments used the values $\beta = 0.5^o$, $\sigma_l = 5$ mm, and $\sigma_\theta = 10^{-4}$ radians obtained from the Sick LMS-200 laser specifications.

3.6.1 Robustness and Accuracy Comparisons

The experiments reported in this section focus on two aspects of estimation performance: the robustness with respect to errors in the initial displacement estimate that seeds the iterations of the algorithm; and the accuracy of the displacement estimates. A more robust

	Unperturbed Trial: Final Error in Position (mm)		Unperturbed Trial: Final Error in Orientation (mrad)	
Test	Weighted	UWLS	Weighted	UWLS
Fig 3.3	0.19	1.33	0.23	8.8
Fig 3.4	1.5	3.6	0.43	1.4
Fig 3.5	2.5	9.8	0.57	16.0
Fig 3.6	1.8	4.1	0.0334	0.31

Table 3.1: Position and orientation error values for trials with no initial perturbations.

	Percentage of 1525 Perturbed Trials Converged		Converged Trials: Average Error in Position (mm)		Converged Trials: Average Error in Orientation (mrad)	
Test	Weighted	UWLS	Weighted	UWLS	Weighted	UWLS
Fig 3.3	91.0%	64.9%	0.63	1.8	0.79	8.6
Fig 3.4	82.0%	56.9%	1.8	6.0	0.67	2.6
Fig 3.5	95.5%	31.2%	2.5	11.1	0.57	16.0
Fig 3.6	75.1%	3.0%	3.1	14.5	0.0392	0.47

Table 3.2: Robustness and accuracy comparison statistics for trials with initial perturbations.

algorithm can successfully recover from a wider range of errors in the initial displacement guess. In practice, such errors in the initial displacement estimate come from large odometry errors, or might arise in the absence of odometry when the initial guess is provided by an open loop estimate of the robot's motion response.

To test for robustness, each algorithm was run through multiple trials with the same pair of scans, each time only perturbing the initial displacement guess. Some initial guesses were sufficiently poor that the algorithm converged to an erroneous solution. An estimate was deemed successful when the true measured displacement lay within the 3σ deviation range as defined by the algorithm's calculated covariance (the UWLS covariance was calculated using the formula given in [36]). The initial displacements ranged from 0 to 600 mm at 8 radial directions (every $\pi/4$ radians) at increments of 200 mm in position, and ranged from -0.6 to 0.6 radians in orientation, at increments of 0.02 radians. For each of the 25 discrete initially perturbed positions, 61 initially perturbed orientations were used to generate 1525 unique initial condition perturbations. These perturbations were added to the true displacement to create initial conditions for the 1525 trials for each algorithm and each environmental condition described below.

We also compare the overall accuracy of each algorithm's displacement measurement.

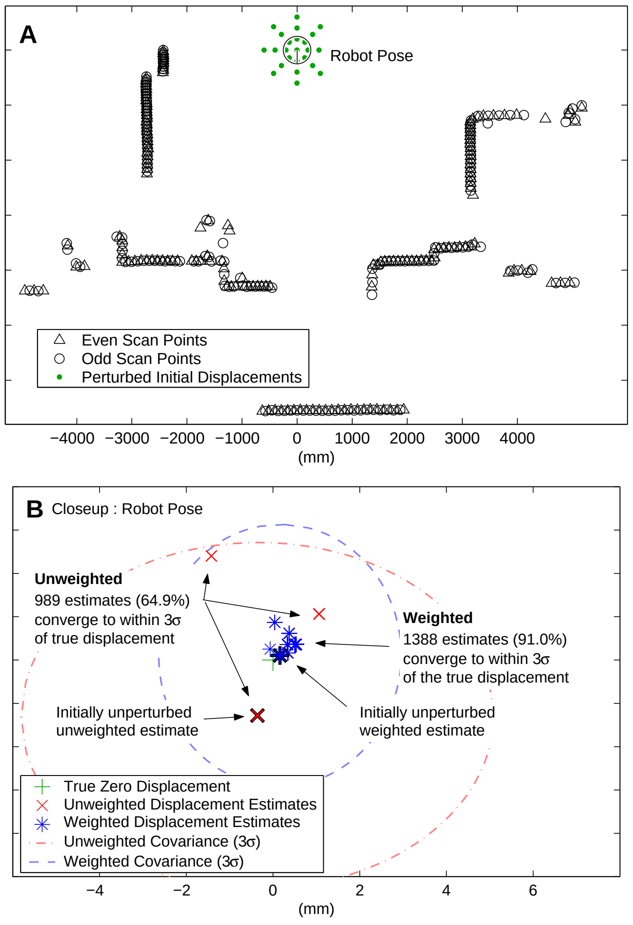

Figure 3.3: A) Experiments with initial displacement perturbations between scans taken at a single pose. B) Closeup of robot pose with results.

The true displacements are measured by hand with an uncertainty of less than 2 mm in displacement and 0.002 radians in orientation. We ran this robustness and accuracy test over four different scan pairs.

Single-Pose Test

The first experiment shown in Figure 3.3 tests for robustness and accuracy while isolating the effects of the modeling of the point correspondence error (Section 3.3.2). In this test, two scans were taken from the exact same robot pose (i.e., the robot was not moved between scans), with one scan comprised only of the even scan points and the second scan comprised only of the odd scan points. In this way, correspondence errors are artificially introduced into the two scans.

The two scans and the initially perturbed positions are shown in Figure 3.3A. The displacement estimates of the successfully converged estimates are shown in Figure 3.3B. The results of the two runs with unperturbed initial guesses are shown with boldfaced markers, along with the 3σ uncertainty boundary of these estimates (shown as dashed ellipses). Of the 1525 runs with initial displacement perturbations thie weighted algorithm converged successfully in 91.0% of the cases while the UWLS algorithm was successful in 64.9% of the cases. The average error for successful weighted estimates was 0.63 mm and 0.00079 radians while the average error for successful UWLS algorithm estimates was 1.8 mm and 0.0086 radians. The error for the case when the initial displacement guess is unperturbed is 0.19 mm and 0.00023 radians for the weighted algorithm and 1.33 mm and 0.0088 for the UWLS algorithm. Though the true displacement between the poses is exactly zero (since the scans were taken at the same robot pose), due to the even/odd nature of the scans no two corresponding scan points sample the exact boundary points of the environment. The effect of this correspondence error on the UWLS algorithm can be visualized in the presence of three distinct local minima in Figure 3.3B. This multi-modal result surrounding the value is often seen in UWLS algorithm robustness test results.

Two-Pose Test

Figure 3.4 shows results from initial condition robustness testing on two scans taken in our lab with true position and orientation displacements of 683 mm and 0.467 radians. Figure 3.4A shows the robot poses and scans under consideration, as well as the initial perturbed displacement guesses. Figure 3.4B shows the results obtained by starting the algorithms from the 1525 different initial displacement perturbations. The weighted algorithm successfully converged in 82.0% of the cases while the UWLS algorithm was successful in 56.9% of the cases. The average error for successful weighted estimates was 1.8 mm and 0.00067 radians while the average error for successful UWLS algorithm estimates was 6.0 mm and 0.0026 radians. The error for the case when the initial displacement guess is unperturbed is 1.5 mm and 0.00043 radians for the weighted algorithm and 3.6 mm and 0.0014 for the UWLS algorithm.

Two-Pose Test With IntraScan Changes in the Environment

Figure 3.5 shows the results of the same type of testing performed on a pair of scans in which the environment changed between scans. Note that the horizontal double wall on

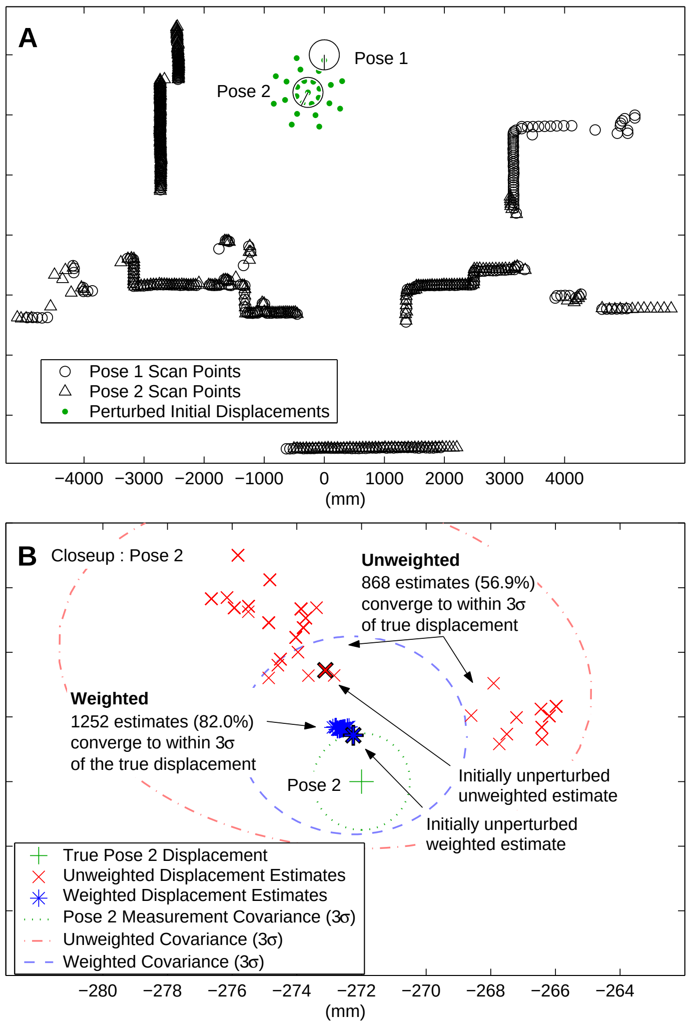

Figure 3.4: A) Experiments with initial displacement perturbations between scans taken at different poses. B) Closeup of pose 2 with results.

the lower left side of the figure is actually a table at almost exactly laser height. The first scan sampled the wall behind the table while the second scan sampled the front edge of the table due to small changes in floor geometry. The additional nearby obstruction to the left of the robot was caused by a person who moved between the two scans. The range points associated with these non-repeating objects represent 29.2% of the total number of scan points.

For the 1525 trials with different initial displacement perturbations, the weighted algorithm was successful in 95.5% of the cases, while the UWLS algorithm was successful in 31.2% of the cases. The average error for successful weighted estimates was 2.5 mm and 0.00057 radians while the average error for successful UWLS algorithm estimates was 11.1

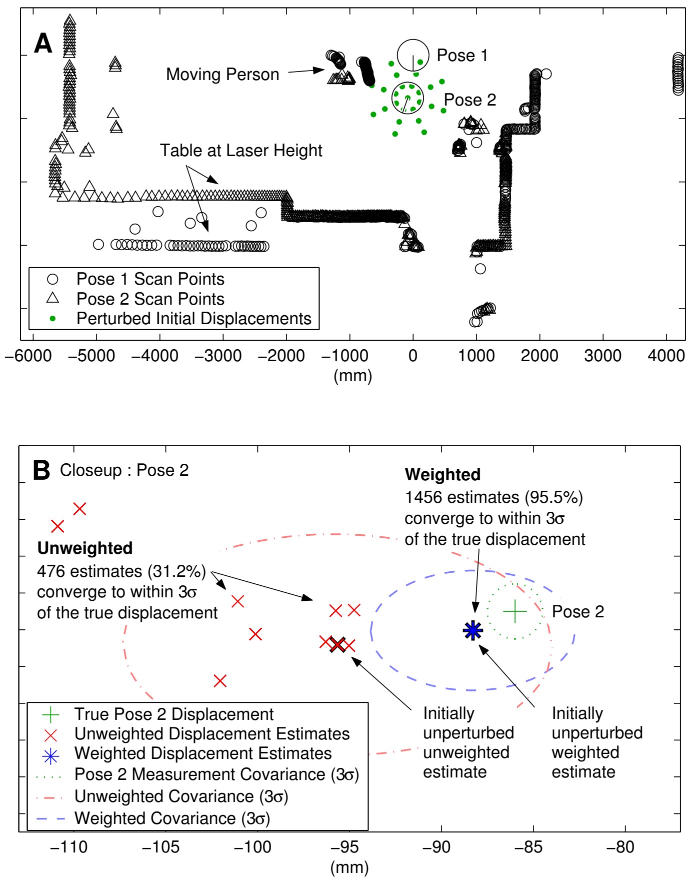

Figure 3.5: A) Experiments with initial displacement perturbations in a non-static environment. B) Closeup of pose 2 with results.

mm and 0.016 radians. The error for the case when the initial displacement guess is unperturbed is 2.5 mm and 0.00057 radians for the weighted algorithm and 9.8 mm and 0.016 for the UWLS algorithm. These results show that this method's emphasis on weighting each scan point results in superior robustness to the presence of a significant number of non-corresponding range points.

Two-Pose Test in a Hallway

Figure 3.6 shows the results of analogous testing done in a nearly symmetrical hallway. In a perfectly symmetrical hallway with no discernible details along the walls, no scan-based algorithm can effectively correct initial displacement errors in the direction along the hallway's main axis. In this test, a single door is open at a slight angle on the left side of the hallway. The presence of this feature allows for possible scan matching convergence. For the

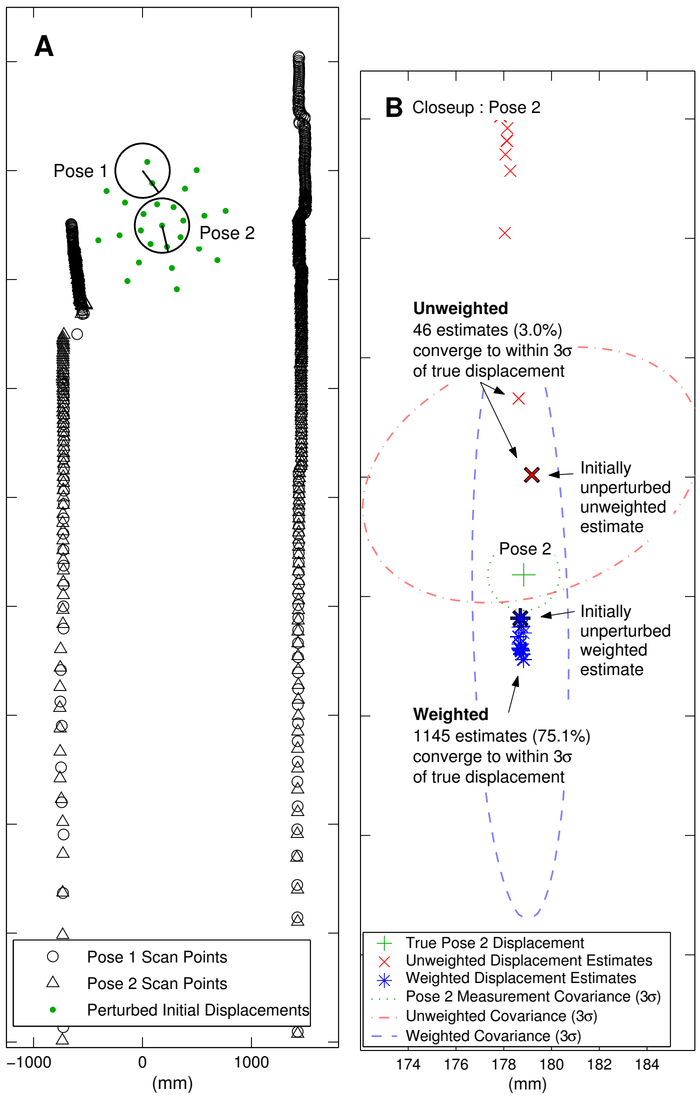

Figure 3.6: A) Experiments with initial displacement perturbations in a hallway environment. B) Closeup of pose 2 with results.

set of 1525 initial displacement perturbations, the weighted algorithm successfully converged in 75.1% of the cases while the UWLS algorithm was successful in only 3.0% of the cases. The average displacement estimate error for the successful weighted estimates was 3.1 mm and 3.92×10^{-5} radians while the average error for successful UWLS algorithm estimates was 14.5 mm and 0.00047 radians. The error for the case when the initial displacement guess is unperturbed is 1.8 mm and 3.34×10^{-5} radians for the weighted algorithm and 4.1 mm and 0.00031 radians for the UWLS algorithm. In effect, the weighted algorithm better uses the hallway's small non-symmetries to correct the position estimation along the hallway axis. This significantly better performance is primarily due to the approach of modeling the correspondence errors, which discounts the contributions along the hallway's axis (since there is very low certainty in that direction). Instead, the small asymmetries are effectively accentuated. Conversely, in the UWLS algorithm the contributions of the non-symmetries are effectively lost, resulting in very poor correction of position errors along the hallway. The plots of the uncertainty ellipses in Figure 3.6B also show how only the weighted algorithm's calculated covariance reflects a greater uncertainty in the direction parallel to the hallway, as would be expected.

3.6.2 Multi-Step Runs

The above results showed not only the improvement in robustness of the weighted algorithm over the UWLS algorithm, but also a significant improvement in the overall accuracy of the successful final displacement estimates. This improvement in accuracy is best seen in longer runs with multiple displacement estimates added end to end.

Long Run with Accurate Odometry

Figure 3.7 shows a 32.8-meter loop path consisting of 109 poses with the final pose the same as the starting pose. Because of the difficulty of hand measuring each pose, only the initial and final positions are compared. For each step the current and previous scans are processed by each algorithm with odometry supplying the initial guess, and updated displacement and covariance estimates are calculated. In order to maintain statistical independence in the estimates, two scans were taken at each pose, with scan 1 used to match with the pose behind and scan 2 used to match with the pose ahead. In practical applications, such a dual scan procedure would not be necessary, as a Kalman filter could incorporate the scans while accounting for the correlation between successive displacement estimates. However, that approach is not used here so that we can focus directly on the properties of the displacement estimate, and not worry about the impact of the filter on the results.

In order to close the loop, the second scan taken at the last pose is matched with the first scan taken at the initial pose. Therefore a perfect series of displacement estimates added tip to tail would result in exactly a zero overall displacement estimate. For the run shown in Figure 3.7, the final odometry error is 1.817 meters and 0.06 radians. The final UWLS algorithm error is 0.271 meters and 0.021 radians while the final weighted algorithm error is 0.043 meters and 0.0029 radians. The ratio of the final translation error

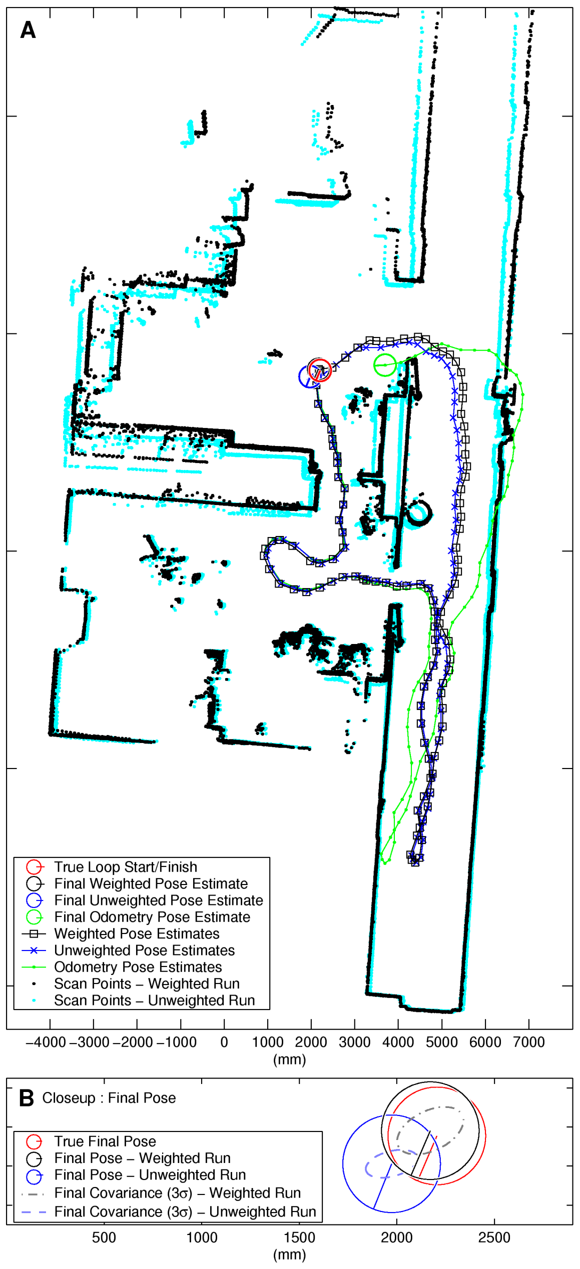

Figure 3.7: A) A 109-pose 32.8-meter loop path. B) Closeup of final path poses, shown the covariance estimates of the weighted and unweighted algorithms.

to total path length is 5.54% for odometry, 0.82% for the UWLS algorithm, and 0.131% for the weighted algorithm. Perhaps more importantly, as shown in Figure 3.7B, the final covariance calculation for the weighted algorithm clearly encompasses the true final pose within the 3σ bounds, while the covariance calculation of the UWLS algorithm does not.

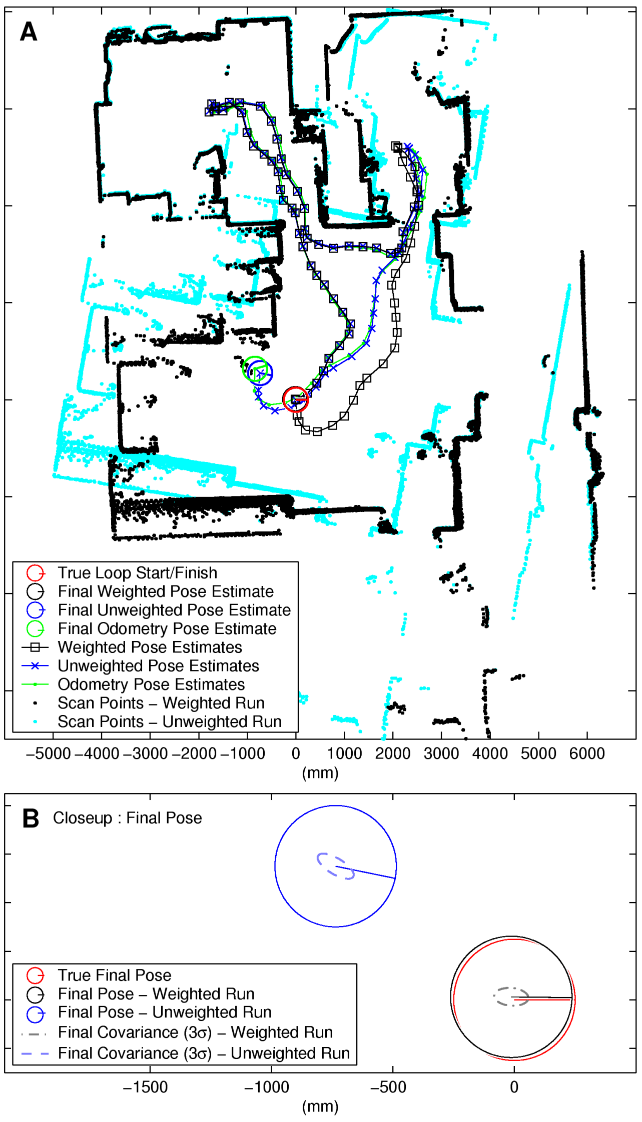

Figure 3.8: A) 83-pose 24.2-meter loop. B) Closeup of final loop poses.

Long Run with Inaccurate Odometry

The improvement of the weighted algorithm over the UWLS algorithm is even more pronounced in the presence of poor odometry estimates. Figure 3.8 shows an actual run where one of the odometry readings was substantially corrupted as the robot rolled over a door jamb when heading into the room in the upper right hand corner of the plot. This path is a 24.2-meter loop consisting of 83 poses with the scans taken and loops closed as in the previous path. For the path shown in Figure 3.8 the final odometry error is 1.040 meters and 0.354 radians. The final UWLS algorithm error is 0.919 meters and 0.200 radians while the final weighted algorithm error is 0.018 meters and 0.013 radians. The ratio of the final translation error to total path length is 4.30% for odometry, 3.80% for the UWLS algorithm, and 0.074% for the weighted algorithm.

3.6.3 Comparison of Computational Demands

Both algorithms were implemented in Matlab, and their computational demands were analyzed using the Matlab Profiler on a desktop computer with a Pentium 4, 1.80GHz CPU with 512M RAM. Within each iteration, computation is divided between the point correspondence phase (which usually consumes the bulk of the computation) and the estimation phase. The number of iterations required to reach convergence also affects the overall cost of computation.

In the 109 steps of run 1 shown in Figure 3.7, the correspondence method used on both algorithms comprises 81.0% of the total UWLS algorithm computation time of 0.112 seconds/iteration and 44.3% of the weighted algorithm computation time of 0.205 seconds/iteration. For the relatively low initial odometry errors in run 1, the UWLS algorithm converges in an average of 12.78 iterations for an average computation time of 1.43 seconds per displacement while the weighted algorithm converges in an average of 10.36 iterations with a total average computation time of 2.12 seconds per pose displacement. For larger initial odometry errors, especially in orientation, the difference in iterations to convergence increases to the point where the weighted algorithm is actually faster than the UWLS algorithm. For the data shown in Figure 3.4, when the orientation error is greater than 0.2 radians the UWLS algorithm converges in an average of 42.98 iterations for an average computation time of 4.81 seconds per displacement while the weighted algorithm converges in an average of 22.60 iterations for an average computation time of 4.63 seconds per displacement.

In summary, experiments show that in real world indoor environments, the weighted method provides significantly greater estimation accuracy and robustness as compared to an unweighted approach without a significant increase in computational cost. Clearly, the computational demands in the estimation phase are larger for the weighted algorithm (as compared to an unweighted algorithm). However, since the computations required by the estimation part of the algorithm account for only a small portion of each iteration, and the algorithm often converges in fewer iterations compared to the UWLS algorithm, the total

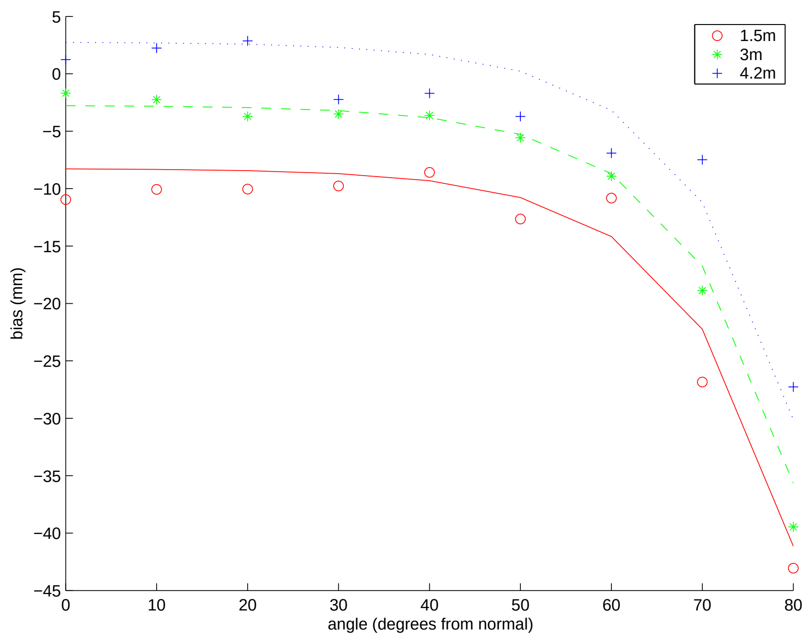

Figure 3.9: The bias fit function.

run time is reduced.

3.6.4 Experiments with Bias Compensation

For completeness, we also implemented the bias compensation scheme of Section 3.3.3. In order to implement this scheme, we experimentally determined the laser's range bias in a controlled laboratory setting, and fit a functional relationship to the experimental data. For experiments, a white paper target was placed at a known distance from the sensor. The center beam of the laser was aligned so as to be normal to the axis of rotation of the target. A total of 100 range measurements were recorded for every 10 degrees of rotation up to 80 degrees from the normal. This process was repeated for nominal ranges of approximately 1.5 m, 3 m, and 4.2 m. Ref. [58] provides a more detailed characterization of this specific laser. The data provided there could be used to build a more detailed model as compared to the one given below.

This experiment showed that the bias for this particular laser sensor is a function of both distance and incidence angle. A function was fit to these data, which was then employed to determine the bias, b_r (in mm), in the measurement given the reported distance, r (in mm), from the laser sensor and the angle from normal, α (in radians), from the Hough transform. Figure 3.9 shows both the data collected and the fitting function. The bias function is given by

$$b_r(r, \alpha) = -14 + 0.004r - 0.035e^{4.9\alpha}. \tag{3.39}$$

When this bias model was incorporated in the WLSM estimation process, the resulting position estimates were almost unchanged. Over the 21.8-meter, eight-step path described in [42], the incorporation of the bias term resulted in an improvement of only 1.8 mm or 0.0082% in the final position estimate. There are two reasons for such a small contribution from the bias term. First, as can be seen in Figure 3.9, this laser's bias is quite small and relatively constant (~1 cm) for angles up to 60 degrees from normal. This excellent behavior is certainly due in part to pre-processing that occurs inside the sensor itself. Most of the corresponding points processed by the WLSM algorithm are recorded at angles within the 60° range. At larger incidence angles, range points are usually sparsely distributed on surfaces far from the sensor and are usually rejected by the matching algorithm since they cannot be paired with the required level of confidence. Even if these points are included, their associated matching covariance is large enough to make their effective contribution negligible. Moreover, symmetries in the environment result in mutual cancellation of the bias effect introduced by points found in opposite directions. Nevertheless a similar process for estimating the bias can be used and can provide improved accuracy in the case of lower-quality distance measuring sensors that experience significant bias.

3.7 Conclusions

This chapter introduced a new method for estimating robot displacement based on dense range measurements. In particular, the effects of different error and noise sources on the convergence and accuracy properties of these motion from structure algorithms were investigated. Experiments showed that careful attention to the details of error modeling can significantly enhance overall displacement and covariance estimation accuracy.

The first part of the chapter gave a general formulation of the displacement estimation problem using weighted point pair correspondences. A general solution to the estimation problem and formulas for the covariance of the displacement estimate were then derived. The application of these results then depends upon explicit error models, and general models for range measurement noise, bias error, and correspondence error were presented. Although parts of this analysis were mainly aimed at planar laser range sensors, the methods can likely be extended to algorithms for non-planar laser scanners [32, 18], where detailed uncertainty modeling has not been considered, and other range sensors such as stereo cameras, radar, ultrasound, etc. These techniques should also be useful for methods that use both planar laser range finders and cameras to estimate three-dimensional motion parameters [52, 48]. The specifics of this analysis must be modified to incorporate the appropriate error/noise models for each particular sensor.

The accurate displacement estimates afforded by this method can be fused with odometry estimates [45] to provide better robot localization capability. Similarly, the improved displacement estimation afforded by this method should in the future lead to more accurate map making and localization procedures.

Chapter 4

Optim-Bug

This Chapter gives a description and convergence results for the Optim-Bug algorithm. Like Tangent Bug, Optim-Bug is a complete and correct planner that assumes perfect dead reckoning. Detailed simulations are not presented, because ideas from Optim-Bug are primarily intended to help introduce the Uncertain Bug algorithm in Chapter 6. Section 4.1 defines the terminology used in the examples and in the proof of completeness. Section 4.2 summarizes the Optim-Bug algorithm, and Sections 4.3 and 4.4 provide more details and the proof of convergence and completeness.

The Tangent Bug algorithm chooses the *locally optimal* path while it is in motion to goal (Mtg) mode. The path is locally optimal (instead of globally optimal) because the robot does not have complete information about the environment. Recall that the second mode of operation for Tangent Bug is boundary following (BF) mode. If the cost of a path is defined as the path length, then the paths generated while in MtG mode are the exact optimal solutions given present knowledge about the world and ignoring knowledge of previous obstacles. However, the paths chosen while in BF mode are not optimal solutions.

Now consider an algorithm where the robot operates in a single mode: it always follows the locally optimal path to the goal, regardless of whether or not that path moves towards or temporarily away from the goal. The cost of a path is defined as the path length. This algorithm is called "Optim-Bug."

4.1 Setup and Definitions

This section introduces some of the concepts and terminology that will be used later in Optim-Bug's proof of convergence. Let $\mathcal{C}$ denote the *configuration space* of the robot—the set of all possible configurations the robot can take [25]. It is assumed that the environment is populated with a finite number of compact obstacles whose boundaries are smooth. The *freespace* is defined as the complement of the obstacles' points:

$$\mathcal{F} = \mathcal{C} \setminus \cup_i \mathcal{O}_i, \tag{4.1}$$

where the $\mathcal{O}_i$ are the physical obstacles. $\overline{\mathcal{F}}$ is the closure of freespace, which is the freespace plus the obstacle boundaries. Let x_k denote the robot's configuration at time k:

$$x_k = x(t_k). \tag{4.2}$$

It is also assumed the robot is equipped with an omnidirectional range sensor with a maximum sensing range of R. Let $v(x_k)$ be the *current visibility set* at configuration x_k:

$$v(x_k) = \left\{ q \mid q \in \mathcal{F};\ \|q - x_k\| < R;\ q(1-t) + x_k t \in \overline{\mathcal{F}}\ \forall t \in [0,1] \right\}. \tag{4.3}$$

This is the region contained in freespace centered at x_k, and bounded by the robot's sensing range R. The boundary of the current visibility set consists of circular arcs with radius R and the sensed obstacle boundaries from robot location x_k. Figure 4.1 shows a sequence of hypothetical steps of the Optim-Bug algorithm. At each robot location, the visibility set is labeled.

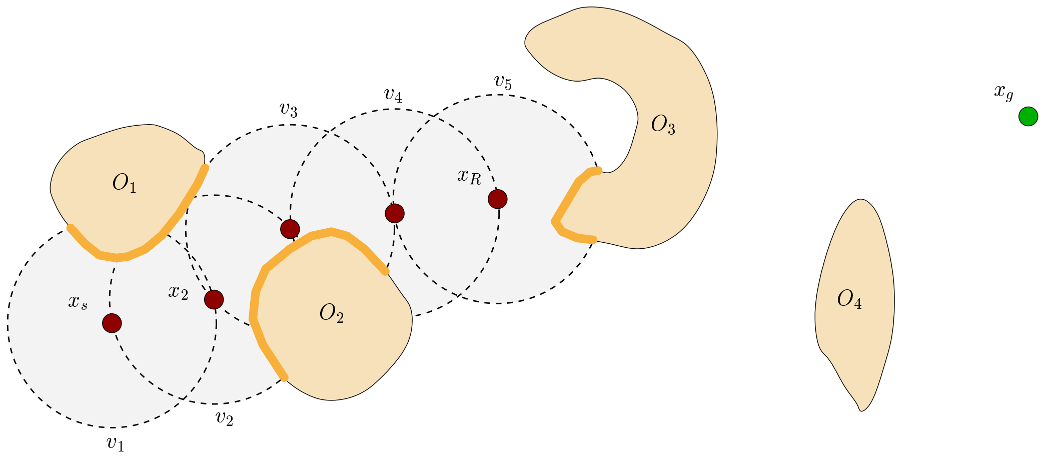

Figure 4.1: The *current visibility sets*, $v(x_k)$, shown at each location of a four-step path. At x_2, obstacle $\mathcal{O}_2$ is a *blocking obstacle*.

A *blocking obstacle* is defined as an obstacle that the robot encounters that blocks the straight-line path to the goal. That is, if the condition

$$\mathcal{O}_i \bigcap v(x_k) \neq \emptyset,\ x_g(1-t) + x_k t \in \mathcal{O}_i \tag{4.4}$$

is true for some $t \in [0,1]$ at step k, then obstacle $\mathcal{O}_i$ is a blocking obstacle. In Figure 4.1, obstacle $\mathcal{O}_2$ is a blocking obstacle from robot location x_2 because it lies between the robot and the goal.

The *total visibility set*, $V_{tot}(x_k)$, is the union of all the $v(x_k)$ for all robot configurations up to time k:

$$V_{tot}(x_k) = \bigcup_{i=1}^{k} v(x_i). \tag{4.5}$$

Figure 4.2 shows the total visibility set for the example of Figure 4.1 (which consists of five robot positions). The total visibility set, $V_{tot}(x_k)$, contains all of the environment that the robot has seen up to time k, and therefore has a memory of.

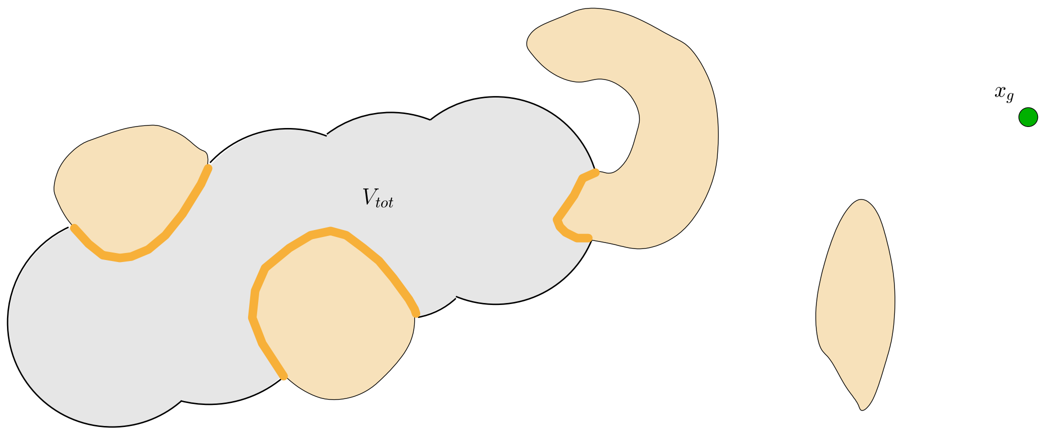

Figure 4.2: The *total visibility set*, V_{tot}, for the same four-step hypothetical example of Figure 4.1.

4.2 Optim-Bug Overview

There are two main differences between Tangent Bug and Optim-Bug. The first difference is that Optim-Bug has only one mode—that of finding the shortest path to the goal given its current knowledge of the world. In the version presented here, Optim-Bug requires more memory than Tangent Bug does, because it must remember all obstacles seen up to the current configuration.

To allow this single mode of operation, it is necessary for Optim-Bug to build up a map of the environment as it moves along, remembering the location of any portions of obstacles that it has seen. This memory is needed for two reasons. First, it removes the necessity for a boundary following mode. Tangent Bug has a boundary following mode in order to eliminate the need for a memory of obstacles it has seen. Second, the memory allows the robot to detect when the goal is completely enclosed inside an obstacle, and thus declare failure.

Figure 4.1 depicts a short hypothetical sequence of executed steps of Optim-Bug, illustrating the required memory of obstacle locations and geometries. The robot starts at location x_s, while the goal is at x_g. At the point shown in the figure, it has taken four steps to location x_R. During these consecutive steps, it has seen and remembered the darkened portion of the obstacles. The unknown obstacle parts are shown in a lighter color. It is assumed that the obstacles block sensing of other obstacles, so the known obstacles are modeled as thin walls.

At the beginning of each new step, the known portions of the obstacles are used as constraints in the optimization process. As long as the goal is believed to be reachable, the shortest length path is computed at each step. Once the shortest path is found, the robot follows this path until it reaches the edge of its total visibility set. In the example, this would be the boundary of the overlapping gray discs. Once it has stepped to the boundary of the total visibility set, the robot takes its next view of the environment, thereby adding the newly viewed area to its map. This process continues until the robot arrives at the goal, or the goal is determined to be unreachable. In summary, one step of the Optim-Bug algorithm proceeds as follows, assuming the robot starts at x_k and $V_{tot}(x_k)$ has been computed:

- While the goal is still reachable:

    1. Compute the shortest length path to the goal based on $V_{tot}(x_k)$ (i.e., taking into account currently known and previously seen obstacles).
    2. Follow the path to the boundary of the total visibility set (position x_{k+1}, or to the goal if it is within the current visibility set. If the robot has reached the goal, terminate with success; Otherwise, compute the new visibility set, $v(x_{k+1})$.
    3. Add the newly viewed region, $v(x_{k+1})$, to the map: $V_{tot}(x_{k+1}) = V_{tot}(x_k) \bigcup v(x_{k+1})$.

4. If there is no path that reaches the goal without intersecting an obstacle, terminate with failure. This means that the goal is inside an obstacle.

With perfect information, i.e., no odometry and sensing errors, Tangent Bug is likely a more practical algorithm than Optim-Bug. It requires very little memory, and in many cases the paths generated by Tangent Bug and Optim-Bug will be quite similar. As shown below, in the case of non-convex obstacles, Optim-Bug can temporarily oscillate back and forth on an obstacle boundary, while Tangent Bug chooses a more uniform path. Optim-Bug's memory requirements are greater, as the robot must remember the location of every obstacle it has seen. However, in complex environments, Optim-Bug may have some advantages. It is also likely that the extension of Optim-Bug to 3-D is relatively straightforward, whereas the 3-D version of Tangent Bug is quite cumbersome.

Recall the assumption of an omnidirectional range sensor. In most realistic situations, such a sensor would sense only at discrete angles around the robot. If the angular discretization is small enough and all obstacles are large relative to this angular separation, the sensor could be considered to be continuous. However, it may be desirable to implement a local obstacle avoidance scheme if the angular discretization is large, or if obstacles are small relative to the angular separation. In these situations the sensor could miss detecting an obstacle. Because Optim-Bug always follows the path to the boundary of V_{tot}, a local obstacle avoidance scheme would mitigate these issues.

This discussion of Optim-Bug will be used to help motivate the structure of Uncertain Bug in Chapter 6. Uncertain Bug uses an optimization framework that is different from Tangent Bug. The cost function used in Uncertain Bug is different than that of Optim-Bug, but some of the general issues are the same. The robot always seeks feasible paths that are optimal in the context of known information. As long as a feasible path exists, the robot will continue to attempt to reach the goal. If and when the robot finds that no feasible paths exist, it can conclude that the goal is unreachable from its current state and declare failure.

At a high level, the proof for Tangent Bug uses a relatively simple Lyapunov-based method. During the motion to goal mode, the robot always decreases the distance from itself to the goal. In boundary following (BF) mode, the robot is allowed to increase this distance temporarily, but it cannot switch out of BF mode until it has reached a point nearer to the goal than the closest it could have been when it started BF mode. Thus, the robot is always making progress towards the goal (in MtG mode) with the exception of a finite number of switches into BF mode. Since the robot only leaves BF mode when it is again closer to the goal, the trend is that the robot is always moving closer to the goal. Consequently, the distance from the robot to the goal converges to zero, since there are only a finite number of obstacles. Of course, if the robot stays in BF mode long enough to completely circumnavigate an obstacle, then the goal is unreachable and the algorithm terminates.

The proof methodology for Optim-Bug is slightly different. In the following section, it is shown that the shortest paths follow lines that are either straight lines or are tangent to the currently known obstacles. Then it is shown that there are a finite number of such paths. Finally, the algorithm is structured so that in a finite number of steps, the robot must reach a straight-line path to the goal, or conclude that the goal is unreachable.

4.3 Shortest Path Properties

First, let us consider the properties of optimal paths, because these properties are needed for the proof of convergence and completeness. Let $\alpha(t)$ be a smooth path in $\overline{\mathcal{F}}$. Below, the necessary conditions for the shortest path in a freespace bounded by smooth curves are stated. Shortest paths consist of segments that are either:

1. a straight line from the robot to the goal in freespace, or

2. tangent to the convex hull of obstacles.

In general the formula for the length of a smooth path $\alpha(t) : [a, b] \to \mathbb{R}^k$ (using the Euclidean norm) is given by

$$l(\alpha) = \int_a^b \|\dot{\alpha}(t)\| dt. \tag{4.6}$$

Appendix C gives details on the proof of these conditions. In addition, such features are well established in the visibility graph literature.

4.4 Proof of Completeness

Let a *goal-tangent line* be a straight-line segment that passes through the goal and is tangent to an obstacle boundary. A *tangent point* is a point on an obstacle where the tangent line is tangent to the obstacle. A *true goal-tangent* is defined to be a goal-tangent that lies entirely in the closure of freespace. If a goal-tangent is not a true goal-tangent, it is defined to be a *false goal-tangent*. An *obstacle endpoint* is a point where an obstacle boundary intersects the boundary of the total visibility set. Figure 4.3 shows an example obstacle configuration and its associated goal-tangent lines, tangent points, and obstacle endpoints.

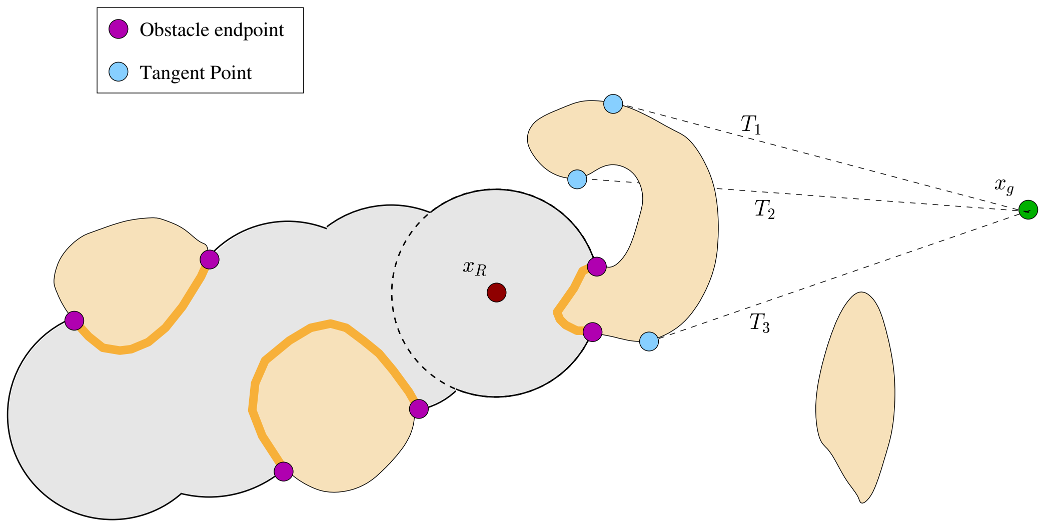

Figure 4.3: A hypothetical example showing obstacle endpoints, tangent points, and their associated goal-tangent lines.

As described in the algorithm overview, the robot is commanded to follow the optimal path to the boundary of the total visibility set, V_{tot}, at each step. If the straight-line path to the goal is clear, the robot will follow it to the boundary of V_{tot}. When the robot encounters a blocking obstacle, the optimal path will be tangent to the *known* obstacle boundaries, i.e., the portion of the obstacles that lie in $V_{tot}(x_k)$. The path planned from x_k will always pass through an obstacle endpoint of the current blocking obstacle because of the properties of the shortest paths. Moreover, the robot will always move to, and end its current iteration at, one of the obstacle endpoints in the presence of a blocking obstacle. In practice, it is assumed that the robot will move ϵ away from the obstacle boundary at the obstacle endpoint. It is also assumed that the robot's sensing range is greater than this distance, i.e., $R > \epsilon$.

Axiom 4.1. At an obstacle endpoint, the obstacle boundary is transverse to the boundary of V_{tot}.

If an obstacle boundary were not transverse to the boundary of V_{tot} at an obstacle endpoint, it would imply that the obstacle boundary point was still within the boundary of V_{tot}, with its location further along ∂V_{tot}. If that were the case, then the robot would see that part of the boundary, and the obstacle endpoint would be in a different location. A non-generic case can occur if the obstacle boundary has an inflection point that lies on the boundary of V_{tot}. Any small perturbation of the robot position or the obstacle location will eliminate this case, so it is assumed that it will not occur.

Lemma 4.1. *The total visibility set, V_{tot}, grows with each robot step.*

Proof. Assume the robot has moved to x_{k+1} from x_k. Let $\mathcal{I}_{k+1}$ be the intersection of the current visibility set at x_{k+1} and the previous total visibility set:

$$\mathcal{I}_{k+1} = V_{tot}(x_k) \bigcap v(x_{k+1}). \tag{4.7}$$

In other words, $\mathcal{I}_{k+1}$ is the part of the robot's environment known at step k that can still be seen at step $k+1$. Let $\mathcal{D}_{k+1}$ be defined as

$$\mathcal{D}_{k+1} = v(x_{k+1}) \ \setminus \mathcal{I}_{k+1}. \tag{4.8}$$

$\mathcal{D}_{k+1}$ is the "new" part of the world seen by the robot at step $k+1$. $\mathcal{D}_{k+1}$ must be non-empty for this Lemma to be true.

By definition of the Optim-Bug algorithm, the robot always follows the shortest path to the edge of the total visibility set at each step. The properties of the shortest path dictate that the next robot configuration, x_{k+1}, will lie either in freespace at the boundary of V_{tot}, or at an endpoint of a sensed obstacle boundary. Recall the assumption that in the case of a blocking obstacle, the robot maintains some small distance, ϵ, away from the obstacle boundary. It is also assumed that the robot sensing range, R, is greater than ϵ.

The boundary of V_{tot} consists of circular arcs of radius R. If x_{k+1} lies in $\mathcal{F}$, then the new visibility set, $v(x_{k+1})$ must contain some newly seen area. Thus, the robot sees a previously unseen part of the environment, and $\mathcal{D}_{k+1} \neq \emptyset$.

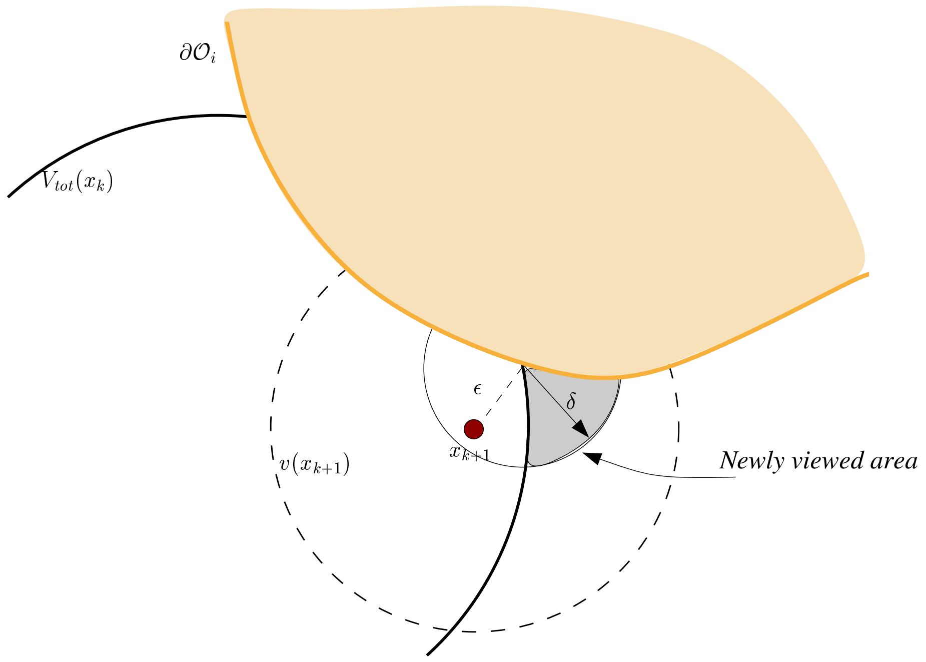

Figure 4.4: Illustration for Lemma 4.1 showing that the total visibility set, V_{tot} grows at each step.

If x_{k+1} lies ϵ away from an obstacle endpoint, the robot will still see some new region of the environment, and $\mathcal{D}_{k+1} \neq \emptyset$. Since $R > \epsilon$, $\exists$ some $\delta > 0$ such that a ball of radius δ centered at the obstacle endpoint will lie inside $v(x_{k+1})$. Because the obstacle boundary is transverse to the boundary of $V_{tot}(x_k)$, the ball must contain some newly seen area. Figure 4.4 depicts this graphically.

Thus, the total visibility set grows with each robot step. □

When the robot encounters a blocking obstacle, the properties of the optimal path dictate that the path will be tangent to the convex hull of obstacles. The robot is “navigating” around a blocking obstacle until it reaches a point where the optimal path is no longer tangent to that particular obstacle boundary. This will happen if the robot reaches the goal or reaches a goal-tangent.

Lemma 4.2. *While navigating around a blocking obstacle, the robot's next step will always take it to an endpoint of a known obstacle boundary segment.*

Proof. The properties of the optimal path (See Appendix C) imply that the optimal path will be tangent to the convex hull of the known obstacle boundaries in $V_{tot}(x_k)$, where x_k is the current robot configuration.

The Optim-Bug algorithm dictates that the robot's next step will follow the optimal path to the edge of $V_{tot}(x_k)$. Either the path will be a straight line towards the goal, in which case there is no blocking obstacle, or the path must be tangent to the blocking obstacle. Known obstacle boundary endpoints lie on the edge of $V_{tot}(x_k)$.

Therefore, when navigating around a blocking obstacle, the robot's next step will take it to an endpoint of a known obstacle boundary segment. □

Once the robot has stepped to one of the obstacle endpoints, the shortest path will continue to follow the obstacle boundary or its convex hull until it reaches a tangent point. At a tangent point, the optimal path departs the obstacle boundary and follows the associated goal-tangent line. At that point, the robot is no longer navigating around that particular blocking obstacle, although it may re-encounter it later. The goal-tangent line is a straight-line path to the goal by definition, so the cycle starts all over again. The robot will follow the goal-tangent line until it either:

- reaches the goal,
- encounters another blocking obstacle,
- re-encounters a locally non-convex portion of the same obstacle.

Let $\partial\mathcal{O}_i$ denote the boundary of obstacle i. Let $\partial\mathcal{O}$ denote the boundary of all obstacles, i.e.,

$$\partial\mathcal{O} = \bigcup_i \partial\mathcal{O}_i. \tag{4.9}$$

Let the boundary of $\mathcal{O}_i$ be parameterized by $s \in \mathbb{R}$. The parameter, s, is similar to an arc-length parameter that defines the distance along the obstacle boundary from some starting point where $s = 0$. However, s does not need to be an arc-length parameter. The parameter, s, increases in a clockwise direction around the obstacle. The actual location of $s = 0$ on the boundary is of no consequence. Let $x_i(s)$ denote the boundary point on $\partial\mathcal{O}_i$ at a distance s.

Let $S^i(a,b)$ denote a continuous boundary segment of obstacle $\mathcal{O}_i$ from $s = a$ to $s = b$, i.e.,

$$S^i(a,b) = \{s|x(s) \in \partial\mathcal{O}_i, s_a \leq s \leq s_b\}. \tag{4.10}$$

Let $\mathcal{B}_k^i$ denote the set of continuous intervals of the boundary of obstacle $\mathcal{O}_i$ seen up to step k, i.e.,

$$\mathcal{B}_k^i = \left\{ S_1^i\left(a,b\right), S_2^i\left(c,d\right), \ldots S_j^i\left(\phi,\upsilon\right)\right\}, \tag{4.11}$$

where j is the number of segments of $\mathcal{O}_i$ seen up to step k.

Lemma 4.3. *Assume the robot has encountered $\mathcal{O}_i$ as a blocking obstacle at step p. While navigating around $\mathcal{O}_i$, the robot will always see at least one new portion of the blocking obstacle boundary that is contiguous with a previously known segment.*

Proof. When the robot first encounters $\mathcal{O}_i$ at step p, it will see at least one boundary segment, though it may see more. If more than one segment is seen, without loss of generality, let $S_1^i\left(\alpha,\beta\right)$ be the segment whose endpoint is contained in the optimal path computed at x_p.

The properties of the optimal path dictate that the next robot step will pass through either $x_i(\alpha)$ or $x_i(\beta)$. The properties of the Optim-Bug algorithm dictate that while navigating around a blocking obstacle, the next robot step will end at one of these points (see Lemma 4.2). Without loss of generality, assume that the robot moves to $x_i(\alpha)$ at step $p+1$. See Figure 4.5 for a graphical interpretation.

Because it is a known obstacle endpoint, the point $x_i(\alpha)$ is necessarily on the boundary of the total visibility set. Even though the robot will move to a point ϵ away from $x_i(\alpha)$, the robot will see a new segment of the blocking obstacle boundary because $R > \epsilon$ and because the obstacle boundary is transverse to ∂V_{tot}. Without loss of generality, let this new segment be $S_2^i\left(\delta,\gamma\right)$. Because the robot is at $x_i(\alpha)$, the point $s = \alpha$ on $\partial\mathcal{O}_i$ must be within the current visibility set and lie on the new segment, i.e.,

$$\delta < \alpha < \gamma. \tag{4.12}$$

Thus, the new segment $S_2^i\left(\delta,\gamma\right)$ must overlap the old segment, $S_1^i\left(\alpha,\beta\right)$ (see Figure 4.5).

At step $p+2$, the robot could move to either $x_i(\beta)$ or $x_i(\delta)$ (See Figure 4.5). Regardless of which of these two points the optimal path passes through, the same analysis applies as the robot continues to navigate around the blocking obstacle.

Therefore, the robot will always see at least one new blocking obstacle boundary segment that is contiguous with a previously known segment. In particular, the new segment will be contiguous with the current blocking segment. Because the $s = 0$ point on the obstacle boundary can be arbitrarily assigned, this point can always be chosen such that the above inequalities hold. $\square$

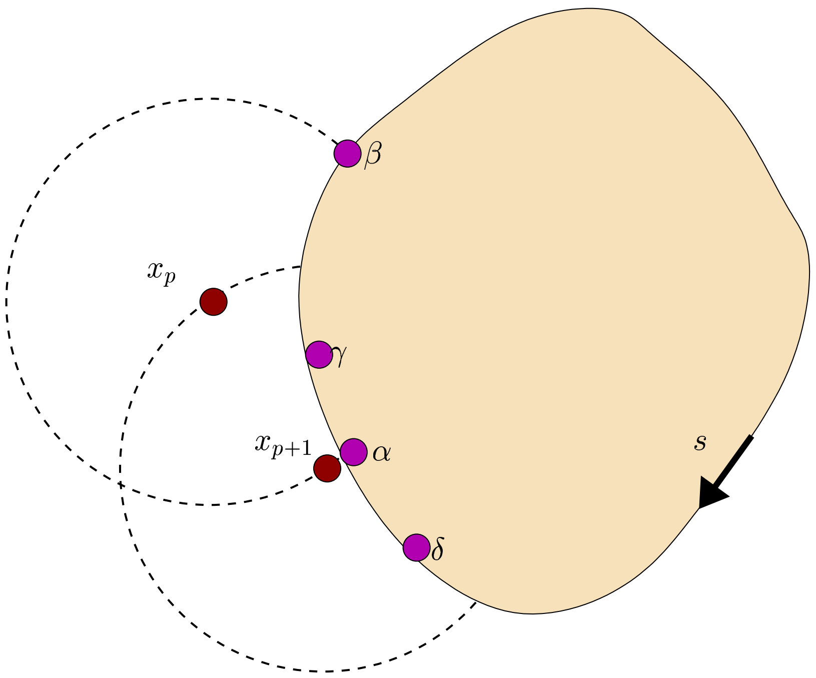

Figure 4.5: At every step while navigating around a blocking obstacle, the robot always sees a new segment of the obstacle boundary that is contiguous with a previously known segment. It may see additional boundary segments that are not contiguous.

Proposition 4.4. *From the time that the robot first encounters a blocking obstacle, it will map a contiguous segment of the boundary until a goal-tangent point is contained in the blocking segment, assuming there exists a goal-tangent point on the blocking obstacle. The goal-tangent will be reached in a finite number of steps.*

Proof. The optimal paths will be tangent to the known obstacle boundaries and the goal. Therefore, the optimal path will depart from the blocking obstacle's boundary at a goal-tangent point. Once a goal-tangent point lies in a known boundary segment, the robot will no longer keep moving to the endpoints of known obstacle boundaries. Instead, it will follow the goal-tangent path.

By Lemma 4.3, the robot will always see a new segment of the blocking obstacle boundary at each step that is contiguous with the previously explored boundary segment.

Thus, if it exists, a goal-tangent point on the blocking obstacle boundary will eventually be contained in the newly mapped boundary segment. The robot may still continue to see a continuous portion of the blocking obstacle as it follows the goal-tangent, but it is not required for the algorithm. As shown below, a goal-tangent will not exist only when the goal is not reachable.

Because the new boundary segment the robot sees is of finite size, and because the obstacle boundaries have a finite size, the robot will reach the goal-tangent in a finite number of steps. □

If the obstacle is larger than the robot's sensing range, the robot may oscillate back and forth along the obstacle boundary as it continually finds and follows the shortest path. Figure 4.6 shows a hypothetical sequence of steps where the robot oscillates back and forth along an obstacle boundary until it reaches a tangent point. The process of moving along the obstacle until a tangent point is reached will take a finite amount of time. At each step, as the robot moves to an obstacle endpoint, it learns and maps a new portion of the obstacle boundary.

Recall that as the robot follows a goal-tangent path that emanates from the boundary of the i^{th} blocking obstacle it will encounter one of the following three conditions:

1. The robot reaches the goal.
2. The robot encounters a new blocking obstacle.
3. The robot re-encounters the same blocking obstacle at a new unexplored boundary segment. This will occur if the i^{th} blocking obstacle is non-convex.

In some ways situations 2 and 3 are treated similarly. It may not initially be possible for the robot to know whether the newly encountered blocking obstacle is indeed a new obstacle or a previously encountered one of which it has partial knowledge. The newly encountered blocking obstacle could be a portion of a non-convex obstacle that the robot has already seen, as shown in Figure 4.7. The robot is first blocked by the obstacle at location x_s. The shortest path takes the robot along the boundary until it reaches goal-tangent T_2. The robot continues along T_2 until it is blocked again. At this point the robot will search back and forth along the obstacle boundary until it reaches T_1.

Proposition 4.5. *Goal-tangents are not revisited.*

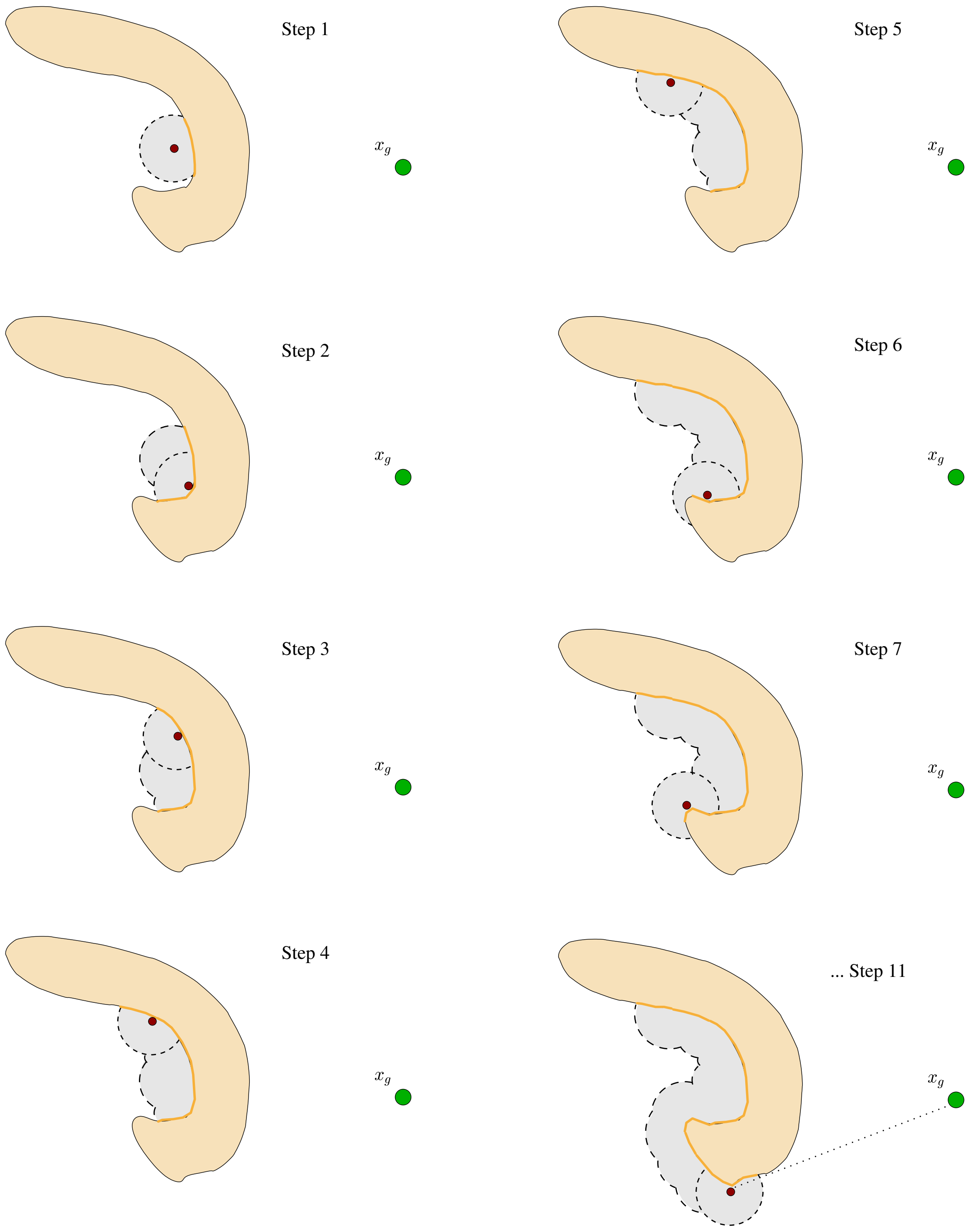

Figure 4.6: A hypothetical sequence of steps showing how Optim-Bug can oscillate back and forth along a blocking obstacle. This also illustrates how a continuous piece of the boundary segment is mapped out.

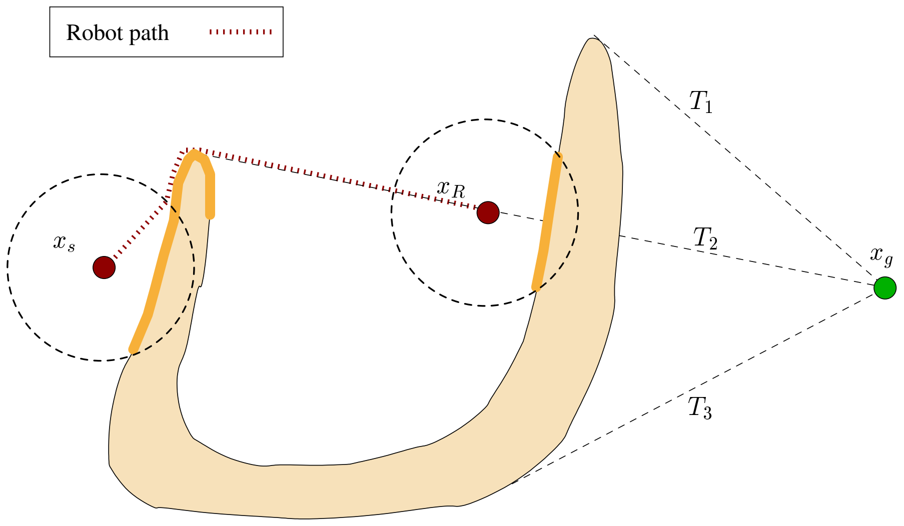

Figure 4.7: A hypothetical example where the robot is blocked by the same obstacle twice. Portions of the obstacle in the robot's map are denoted by a darker color.

Proof. By definition, a goal-tangent will pass from a point tangent to an obstacle boundary to the goal. If the goal-tangent the robot is following is not blocked, the robot will reach the goal along that goal-tangent and terminate with success.

If the goal-tangent that the robot is following *is* blocked, the robot will encounter a blocking obstacle along that goal-tangent line. Once the robot sees the segment of the blocking obstacle, $S^i\,(a, b)$ and places it in its map, no path to the goal can lie along that goal-tangent line, since it would intersect the obstacle segment. Thus, the robot will never follow that particular goal-tangent again. $\square$

Lemma 4.6. $\exists$ *only a finite number of goal-tangent lines.*

Proof. Let $\theta_i(s)$ be a map from the i^{th} obstacle boundary length parameter s to the angle subtended by the line from x_g to $x_i(s)$, with respect to the x-axis of a fixed global reference frame. Because $\partial\mathcal{O}_i$ is assumed to be smooth, $\theta_i(s)$ is smooth.

Goal-tangent points for $\mathcal{O}_i$ will be local extrema of the function $\theta_i(s)$. Because $\theta_i(s)$ is smooth, and s lies in a bounded interval, $\theta_i(s)$ will have a finite number of local extrema, and each obstacle will have a finite number of goal-tangents. Because it is assumed that there are a finite number of obstacles in the environment, there are a finite number of goal-tangent lines. $\square$

Proposition 4.7. *If the goal is not located inside an obstacle, i.e., the goal is reachable, the algorithm will find a true goal-tangent after a finite amount of time.*

Proof. Let $\mathcal{G}$ be the set of goal-tangents, i.e.,

$$\mathcal{G} = \{g_i\}, \, i = 1, \ldots, N, \tag{4.13}$$

where g_i is the i^{th} goal-tangent, and N is the number of goal-tangent lines. $\mathcal{G}$ can be divided into two exclusive sets—the set of true goal-tangents, $\mathcal{G}^{\text{true}}$, and the set of false goal-tangents, $\mathcal{G}^{\text{false}}$:

$$\mathcal{G}^{\text{true}} = \left\{g_k^{\text{true}}\right\}, \, k = 1, \ldots, N_{\text{true}}, \tag{4.14}$$

$$\mathcal{G}^{\text{false}} = \left\{g_k^{\text{false}}\right\}, \, k = 1, \ldots, N_{\text{false}}. \tag{4.15}$$

By definition of the true and false goal-tangents,

$$\mathcal{G}^{\text{true}} \bigcap \mathcal{G}^{\text{false}} = \emptyset. \tag{4.16}$$

Lemma 4.6 implies that $\mathcal{G}$ has a finite number of elements, so $\mathcal{G}^{\text{true}}$ and $\mathcal{G}^{\text{false}}$ must also have a finite number of elements.

When the robot starts out, it may or may not be on a goal-tangent. If it is, and it is on $g \in \mathcal{G}^{\text{true}}$, it will reach the goal on that particular goal-tangent. If it starts out on $g \in \mathcal{G}^{\text{false}}$, then by Proposition 4.4 it will reach a new goal-tangent after a finite number of steps. This new goal-tangent could be in $\mathcal{G}^{\text{true}}$ or $\mathcal{G}^{\text{false}}$, and the process will continue. If the robot does not start out on a goal-tangent, Proposition 4.4 implies that it will find one after a finite number of steps.

$\mathcal{G}^{\text{false}}$ is a finite set. The algorithm will continually find new elements in $\mathcal{G}$. By Proposition 4.5, goal-tangents are not revisited. Either the robot will find a $g \in \mathcal{G}^{\text{true}}$, or it must exhaust all elements in $\mathcal{G}^{\text{false}}$, thereby eventually reaching a $g \in \mathcal{G}^{\text{true}}$ after a finite number of steps. □

Once a goal-tangent line has been explored and found to be blocked, the robot will never follow that goal-tangent again. If the goal is reachable, at least one of the goal-tangent lines must be wholly contained within the freespace. Because there are a finite number of goal-tangents and the robot spends a finite amount of time to transition from a blocking obstacle to a goal-tangent, Optim-Bug will reach the goal in a finite amount of time if the goal is reachable. In the worst case, the robot will explore all of the goal-tangent lines.

If the goal is not reachable, the robot will detect that the goal is enclosed in an obstacle. Recall that Optim-Bug assumes perfect odometry and sensing, so the robot's map that is built during its exploration will also be perfect. In the failure case, it is not necessarily true that the robot will explore all of the goal-tangent lines before declaring failure.

Proposition 4.8. *If the goal is inside an obstacle $\mathcal{O}_i$, i.e., the goal is* not *reachable, then the robot will circumnavigate $\mathcal{O}_i$ in a finite number of steps, indicating failure. In other words, no boundary segments of $\mathcal{O}_i$ will be unseen.*

Proof. Assume that the goal is inside obstacle $\mathcal{O}_i$. There may or may not exist goal-tangents depending on the shape of the obstacle.

If $\mathcal{O}_i$ does not have any goal-tangent lines (e.g., it is convex), interpretation of Proposition 4.4 means that the robot will completely circumnavigate the obstacle without finding any goal tangent lines. Furthermore, this means that it will map out the entire boundary of the obstacle as one continuous segment.

If $\mathcal{O}_i$ does have goal-tangent lines, by Proposition 4.4 the robot will map out a continuous segment until it reaches one of the goal-tangents (e.g., if the goal were inside the obstacle of Figure 4.7, there could still be goal-tangent lines). Because the goal is inside the obstacle, any goal-tangent (even those that are tangent to other obstacles) will necessarily be blocked. In this case, the boundary of $\mathcal{O}_i$ will not be mapped out in one contiguous segment.

Since the algorithm always finds a new contiguous piece of obstacle boundary at each step (Lemma 4.3), it will determine that the disconnected segments are all part of the same obstacle in a finite number of steps. □

Theorem 4.9. *Optim-Bug is complete.*

Proof. According to Proposition 4.8, if the goal is not reachable the robot will determine this and declare failure in a finite number of steps.

If the goal is reachable, the robot will find a true goal-tangent in a finite number of steps by Proposition 4.7. According to Lemma 4.6, there are only a finite number of goal-tangent lines, and once a goal-tangent line is found to be blocked it is never revisited (Proposition 4.5).

Thus, Optim-Bug will determine if the goal is reachable or not and terminate in a finite amount of time. □

In some sense, there is a list of goal-tangents that the robot can check off one by one as it finds them to be blocked. In the worst case, it will end up checking all of the blocked goal-tangents before it finds one of the goal-tangents that reaches the goal.

4.5 Conclusion

This chapter presented the Optim-Bug algorithm, a complete and correct planner. Optim-Bug is based on finding and following the shortest path in the environment. Optim-Bug assumes a point robot with a sensor that has a finite range. Like Tangent Bug, Optim-Bug requires perfect dead-reckoning. Optim-Bug requires the robot to keep a memory of obstacles it has seen. Optim-Bug may prove to extend well to higher dimensions. There is a 3-D version of Tangent Bug [21], but the proof of completeness is non-intuitive.

One possibility for improvement is in eliminating the requirement that Optim-Bug remember the location of every obstacle it has seen. If one placed a bound on the size of obstacles in the robot's environment, it seems that an obstacle could be removed from the robot's memory once the robot had moved a certain distance away from it. If the robot could "be done" with an obstacle and guarantee that it would never see that obstacle again, it could reduce the overall memory requirements of the algorithm.

Chapter 6 presents the Uncertain Bug algorithm, which does not require the robot to have perfect dead-reckoning. It uses an optimization framework that is similar to Optim-Bug's, but uses a different cost function. Some of the analysis performed in this chapter will help motivate similar analyses of the Uncertain Bug algorithm.

Chapter 5

Path Optimization

This chapter describes an optimization procedure used to find the path that a robot can follow in order to minimize its positional uncertainty at the goal. The *optimal path* is defined as the path that has the least robot pose uncertainty at the goal. This procedure will find the entire path from the robot's current location to the goal while avoiding obstacles. In Chapter 6, this optimization method is modified to fit into a sensor-based motion planning algorithm. Although I will talk about the *optimal* path, it is possible that the path generated by the algorithm is only a local optimal, as opposed to the globally optimal, solution. The presence of obstacles in the environment, and the non-linearity of the cost function make it practically difficult to find the globally optimal solution.

5.1 The Localization Framework

This section describes the general localization framework that is assumed for this planning method. I use a standard Kalman filtering approach as described in Section 2.2.2 to track the covariance of the robot state and to perform localization. The covariance of the filter will play a central role in the path planning method. As in Section 2.2.2, assume that the robot is given the location of N landmarks, as well as the covariances of these landmark location estimates. The environment is populated with a finite number of obstacles. The robot's task is to plan a collision-free path from its current position to the goal location, $\mathbf{x}_g$, a point specified in the global reference frame. This setup is shown in Figure 5.1.

A brief summary of the setup from Section 2.2.2: The state vector $\mathbf{x}$ contains the

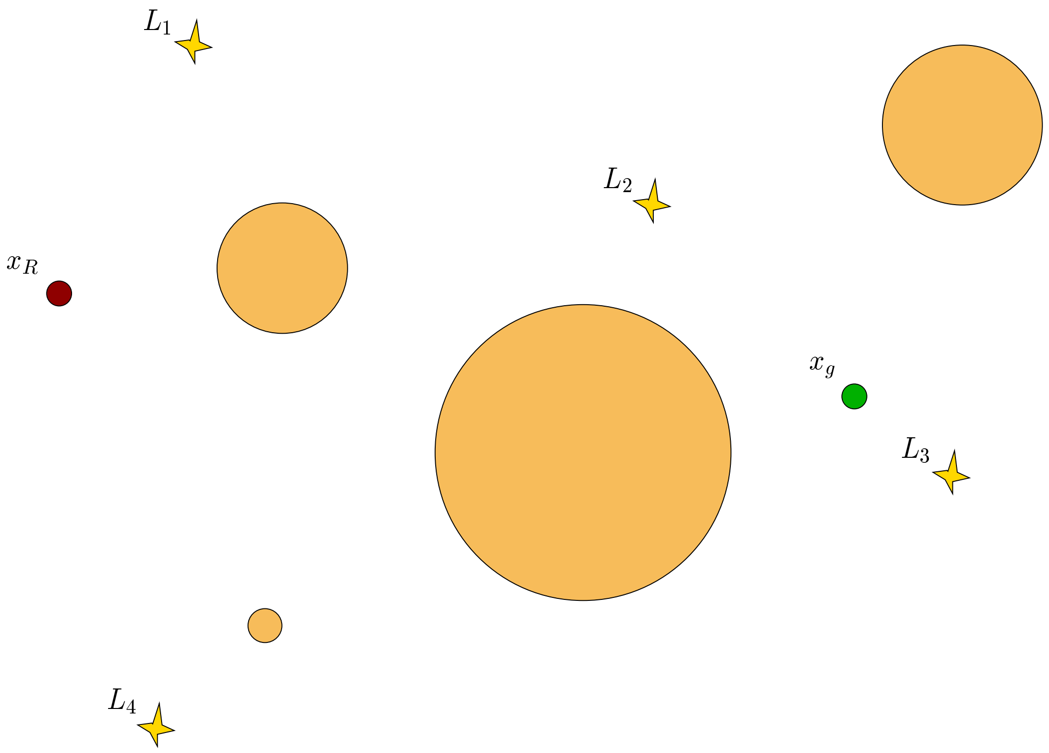

Figure 5.1: The environment setup assumed for the path planning process. The robot starts at location $\mathbf{x}_R$ and is instructed to travel to $\mathbf{x}_g$. L_1, L_2, L_3, and L_4 are landmarks that the robot has knowledge of. The circles are obstacles that the robot must avoid.

positions of both the robot and all of the landmarks, i.e.,

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_R & \mathbf{x}_{L1} & \cdots & \mathbf{x}_{LN} \end{bmatrix}^T, \tag{5.1}$$

where $\mathbf{x}_R$ is the Cartesian position of the robot and $\mathbf{x}_{Li}$ is the Cartesian position of the i^{th} landmark, i.e.,

$$\mathbf{x}_R = \begin{bmatrix} x_r \\ y_r \end{bmatrix} \text{ and } \mathbf{x}_{Li} = \begin{bmatrix} x_{Li} \\ y_{Li} \end{bmatrix}. \tag{5.2}$$

For the range and bearing observation model presented previously (Equation (2.27)), the measurement of landmark i is

$$z_i(k+1) \quad = \quad \mathbf{x}_{Li}(k+1) - \mathbf{x}_R(k+1) + n_i(k+1), \tag{5.3}$$

where $n_i(k+1)$ is a zero-mean white Gaussian noise process with covariance

$$R_i(k+1) = E[n_i(k+1)n_i(k+1)^T]. \tag{5.4}$$

The measurement noise need not be the same for each landmark. Range-dependent variations are allowed in the noise model in order to accommodate many realistic sensing situations (see Equation (5.16) below). Although all examples presented use the range and bearing measurements, this formulation allows for any type of measurements. At each measurement update, the robot takes a measurement of every landmark, so the entire measurement vector and measurement covariance are given by

$$\begin{aligned} z(k+1) &= \begin{bmatrix} z_1^T & \cdots & z_N^T \end{bmatrix}^T, & (5.5) \\ R(k+1) &= \operatorname{diag}\left(R_1, R_2, ..., R_N\right). & (5.6) \end{aligned}$$

Note that in reality, not all landmarks may be visible from a given robot pose due to limitations in sensing range. These effects can be incorporated into the sensor uncertainty model (see Fig. 5.2 for an example).

The measurement matrix, H, for the measurement of all landmarks is

$$H(k+1) = \begin{bmatrix} H_1^T & H_2^T & \cdots & H_N^T \end{bmatrix}^T, \tag{5.7}$$

where the H_i (the individual measurement Jacobians) are given by Equation (2.29).

As described in later sections, the primary concern is to minimize the robot's estimated pose covariance at the goal. This makes it unnecessary to track explicitly the robot's position estimate for a given path. Therefore the state update will not be used in finding the optimal path, but the covariance update (equation (2.15) will be used. In order to avoid needless computation of the Kalman gain matrix K, the following form of the covariance update is used:

$$P(k+1/k+1) = P(k+1/k) - P(k+1/k)H^T S^{-1} H P(k+1/k), \tag{5.8}$$

where the S matrix is described in Equation (2.14).

5.2 Cost Function

A primary concern is having the robot reliably reach a given goal position. For example, the robot may be able to recognize it has reached the goal only when it is within a certain range, ϵ, of the goal. This would imply that minimizing the uncertainty of the robot's position at the goal should be a primary objective. I choose a norm on the robot pose covariance at the goal as the cost function of the optimization problem, i.e.,

$$J = ||P_{RR}(M/M-1)||, \tag{5.9}$$

where M is the number of steps the robot has taken to reach the goal from a given starting position. Any norm can be chosen, each having its own interpretation. I use the Frobenius norm, which for a matrix A is defined to be:

$$\begin{aligned} ||A||_F &= \sqrt{\text{trace}(AA^T)} && (5.10) \\ &= \sqrt{\sum_i \sigma_i^2}, && (5.11) \end{aligned}$$

where the σ_i are singular values of the matrix A. The Frobenius norm yields good numerical behavior because it does not involve any non-smooth max or min operations. It also has an intuitive interpretation. In this case the matrix $A = P_{RR}$ is symmetric and positive definite, so its singular values are the same as its eigenvalues. Consequently, this norm provides the least-squares minimization of the principle values of the robot's positional uncertainty ellipse.

With the cost function chosen, the optimization problem is defined below. Let $\gamma(t)$ be a robot path from the start to the goal, i.e., $\gamma(0) = \mathbf{x}_R(0)$ and $\gamma(t_m) = \mathbf{x}_g$. Let $\overline{\mathcal{F}}$ denote the closure of *freespace*, the area of the environment not occupied by obstacles. Define the feasible path set, $\mathcal{P}$, as the set of paths that lie entirely in $\overline{\mathcal{F}}$:

$$\mathcal{P} = \left\{\gamma \mid \gamma(t) \in \overline{\mathcal{F}} \ \forall t \in [0, t_m]\right\}. \tag{5.12}$$

The problem is to find the lowest-cost feasible path:

$$\min_{\gamma(t)\in\mathcal{P}} \quad J(\gamma(t_M)) \tag{5.13}$$

$$\text{subject to} \quad \dot{\gamma}(t) = u. \tag{5.14}$$

The following sections describe the method to solve this optimization problem.

5.3 Practical Optimization Approach

Conceptually, given the Kalman filter framework described in Section 5.1, one could parametrize the robot's path in terms of the velocity control inputs. Given a sequence of control inputs, the covariance of the state estimate can be found by the application of the Kalman filtering equations given in Section 2.2. The robot's covariance can be propagated along the proposed path by use of Equation (2.10). Wherever the robot stops to perform a measurement of the landmarks, Equation (2.15) would be used to calculate the the robot's new position estimate error covariance. Using the landmarks will reduce the robot's position estimate error covariance. Because the robot has a finite sensing range, it must be near enough to the landmarks to be able to use them for localization. The robot could perform this landmark measurement at every time step, or at a smaller number of positions along the path. The controls can then be varied throughout their feasible space to find the lowest-cost path, using the cost function given by Equation (5.9). As described below, I choose a slightly different and more practical parametrization that yields the same effect.

Because of the position-dependent measurement model, it is more convenient to parametrize the optimization problem in terms of robot positions, or waypoints, instead of the open loop command velocities V. A collocation approach is chosen where the robot path is discretized into a small number of waypoints. An M step robot path will have $M-1$ waypoints, which are denoted by

$$U = \begin{bmatrix} U_1 & U_2 & \cdots & U_{M-1} \end{bmatrix}, \tag{5.15}$$

where U_i is a single waypoint on the robot path, specified as the Cartesian location (x_i, y_i).

There are a few reasons why the path is discretized into a small number of waypoints as opposed to finding the continuous path. The first is a computational concern. Because I use a gradient descent method, the time required to minimize an initial guess grows larger and larger as the number of degrees of freedom is increased. The second reason is a more practical one. The waypoints are the locations where the robot will stop, take a measurement of landmarks, and perform an update to re-localize itself. A realistic robot would likely use sensors such as lasers or cameras to find the landmarks. The image or scan processing required to pick out the landmarks and solve the correspondence problem is much greater than the processing required to propagate a step of odometry. Assuming that the robot can continually process image or scan information at high rates is unrealistic.

It would be natural to parametrize the cost function in terms of the control inputs V. However, such a parametrization would require parametrizing the measurement covariance matrix R in terms of the control inputs as well, or assuming that the matrix is constant. Assuming that the matrix R is constant everywhere is an unreasonable assumption. While the measurement noise may be relatively constant in a region around a particular landmark, any real sensor is bound to have a limit to its range. One way to alleviate this problem is to make the measurement noise a function of the landmark being measured and the position of the robot, e.g.,

$$R_i(k+1) = f\left(\mathbf{x}_R(k+1), \mathbf{x}_{Li}(k+1)\right). \tag{5.16}$$

This functional form allows for many different assumptions for the measurement noise. Practically useful forms for R_i would include quadratic or exponential dependence on the sensing range to L_i. An exponential dependence provides a particularly convenient way to include limitations in sensing range while keeping the cost function smooth. Many common sensors, such as a laser scanner, have a finite sensing range. Within the sensing range, the process noise is relatively constant [1]. However, modeling the noise as a rapidly rising exponential outside of that range effectively incorporates a limited sensing range.

Making an assumption such as (5.16) allows the Kalman filter update step (Equation (2.15)) to be parametrized in terms of the robot's path. To keep a consistent set of variables, the propagation step (Equation (2.10)) is modified to use the same optimization variables. Making the assumption that the variance in the robot's velocity is proportional to its velocity, i.e.,

$$\sigma_v = \alpha V, \tag{5.17}$$

where α is a proportionality constant, changes the variables of the propagation step to be the path waypoints. By substituting Equations (2.24) into Equation (2.10), the covariance propagation equation takes a somewhat simple form:

$$P(k+1/k) = P(k/k) + \Delta t^2 \begin{bmatrix} Q(k) & 0_{2\times 2N} \\ 0_{2N\times 2} & 0_{2N\times 2N} \end{bmatrix}, \tag{5.18}$$

where $Q(k)$ is as in Equation (2.22). With (5.17) substituted in, the robot's portion of the covariance matrix propagation step is

$$P_{RR}(k+1/k) = P_{RR}(k/k) + \Delta t^2 \begin{bmatrix} \alpha^2 v_x^2(k) & 0 \\ 0 & \alpha^2 v_y^2(k) \end{bmatrix}. \tag{5.19}$$

Using the fact that

$$\begin{aligned} v_x(k)\Delta t &= x_R(k+1) - x_R(k), \\ v_y(k)\Delta t &= y_R(k+1) - y_R(k), \end{aligned}$$

Equation (5.19) simplifies to

$$P_{RR}(k+1/k) = P_{RR}(k/k) + \alpha^2 \Delta P(k+1), \tag{5.20}$$

where the covariance propagation increment $\Delta P(k+1)$ is defined as

$$\Delta P(k+1) = \begin{bmatrix} (x_R(k+1) - x_R(k))^2 & 0 \\ 0 & (y_R(k+1) - y_R(k))^2 \end{bmatrix}. \tag{5.21}$$

The robot is implicitly constrained to seek the goal during the final step. Hence, the final propagation increment is

$$\Delta P(M) = \begin{bmatrix} (x_g - x_R(M-1))^2 & 0 \\ 0 & (y_g - y_R(M-1))^2 \end{bmatrix}.$$

The first propagation increment $\Delta P(0)$ is similar, but is a function of the robot's starting location, $\mathbf{x}_0$.

The goal of Chapter 6 will be to show how to incorporate these ideas into a true sensor-based motion planning algorithm. To do this, the algorithm must incorporate sensing of obstacles. It is assumed that obstacles block sensing of other obstacles, but not sensing of landmarks. One way to handle obstacles in the optimization process is to use them as constraints. Tangent Bug can be thought of in this way, where the cost function is simply the path length, and the viewable obstacles are state constraints. Another possibility would be to augment the cost via a penalty-function [51] for violating the constraints. I choose the former, i.e., the path cannot intersect any obstacles. For all simulations, the obstacles are simply modeled as circles. For illustration purposed, specific formulas for incorporating disk obstacles as constraints are included in Appendix B.

The optimization problem can now be defined in more detail as follows: Given a start location, a goal location, and the number of path steps M, find the set of waypoints $[U_1, U_2, \ldots U_{M-1}]$ that minimize the cost function $J([U])$, as defined in (5.9). There are multiple methods one could use to minimize the cost function. One possibility is minimization via enumeration [24]. If the waypoints were discretized into a grid with C_U cells, the cost of every path of M steps could be calculated. This would be guaranteed to yield the minimum cost path, but it has computational cost of $O(C_U^{M-1})$ [24]. Even for a 20×20 grid, which is quite small, there are 64 million paths with three waypoints. To keep computation times to a minimum, a gradient descent method is used. Of course, using gradient descent will only find a local minimum of the cost, not necessarily the global minimum. A separate step is required to identify all of the locally minimal paths.

5.4 Initial Condition Generation

Using a gradient descent method to solve the optimization problem implies that the solution will only be a local optimum, and not necessarily the globally optimal solution. Which locally optimal solution is converged upon is highly dependent on the initial conditions. To improve the chance of finding the global optimal, the algorithm first generates a number of initial conditions that take "different" paths through the environment. Each of these initial conditions is optimized in turn, and in the end the one with the lowest cost is chosen.

I propose and use a simple heuristic to generate a variety of initial conditions. First, the local tangent graph (LTG) at the robot's start location is constructed. This construction ensures that there are paths that leave on both sides of physical obstacles in the robot's vicinity. To generate one set of initial paths, a path is constructed that goes from the robot's start position to an LTG node, then to a landmark, and finally to the goal. These paths can be thought of as ones that detour to take advantage of the landmarks. This is repeated for all combinations of LTG nodes and landmarks. If there are n nodes in the LTG and m landmarks, this first set will contain $n \times m$ initial paths.

Another set of paths is created that goes directly from the robot's current position to a landmark, to a second landmark, then directly to the goal. This set of paths is similar to the first set, but it is not constrained to pass near an obstacle.

A third set of paths is created that goes from the robot start, to an LTG node, then directly to the goal. These paths can be thought of as taking the direct route around an obstacle to the goal. This second set will contain n initial paths. In many situations, multiple initial conditions will converge to the same solution (within numerical accuracy).

There are many other options that one could use to create more initial conditions. Instead of initial paths that visit only two landmarks, paths that visit m landmarks could be generated. Because the robot has complete knowledge of the obstacles, creating the LTG with a larger sensing radius than the robot's sensor has could be used to create initial paths that "weave" around different sides of obstacles. A completely different construct, such as the generalized Voronoi graph [10] could be used also to create initial paths that pass on differing sides of obstacles. Of course, the more initial conditions that have to be optimized, the slower the overall algorithm will become.

5.5 Results

To illustrate the performance of this method, this section presents a sequence of simulation results that involve different numbers of waypoints, landmarks, and obstacles. These simulations model a point robot with an on-board landmark sensor whose uncertainty model has an exponential dependence on the range to the landmark. Specifically, the range-dependent measurement variance follows the form:

$$\sigma_i^2(k+1) = \beta + \exp\left(\gamma(r_i(k+1) - r_{max})\right), \tag{5.22}$$

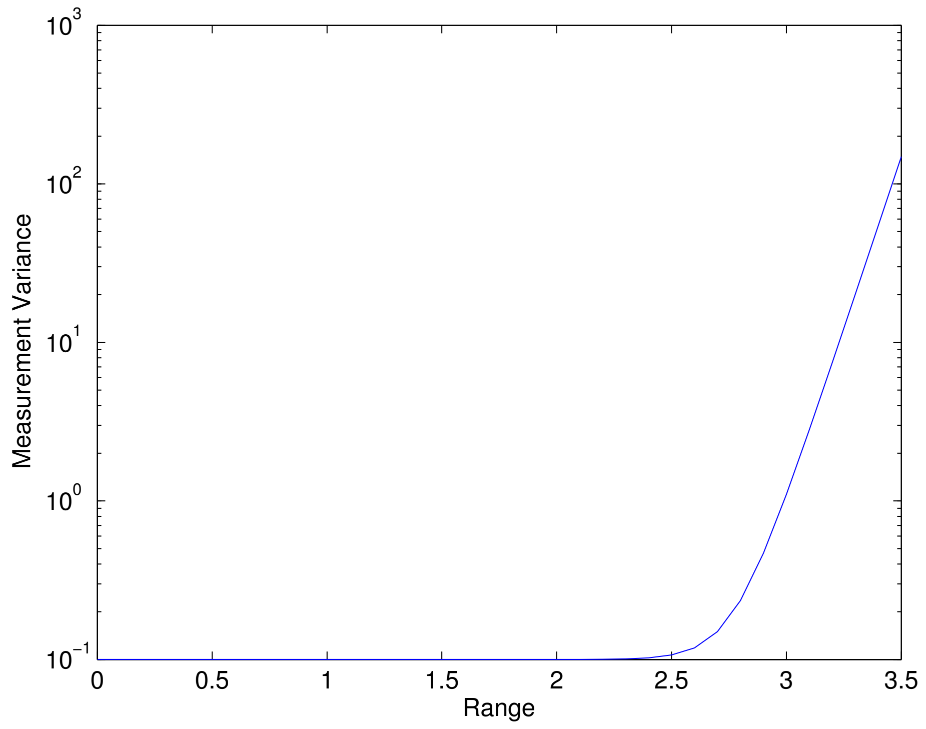

Figure 5.2: Sample measurement variance profile as a function of range. $\beta = .1$, $\gamma = 10$, and $r_{max} = 3$. Note that the y-axis scale is logarithmic.

where β and γ are chosen to give a realistic profile, r_{max} is the sensor's maximum useful range, and r_i is the range to the landmark L_i, i.e.,

$$r_i^2(k+1) = \left[\mathbf{x}_R - \mathbf{x}_{Li}\right]^T \left[\mathbf{x}_R - \mathbf{x}_{Li}\right]. \tag{5.23}$$

The parameter β can be thought of as the nominal measurement variance, while γ affects how quickly the exponential increases. Then the matrix $R_i(k+1)$ is simply

$$R_i(k+1) = \begin{bmatrix} \sigma_i^2(k+1) & 0 \\ 0 & \sigma_i^2(k+1) \end{bmatrix}. \tag{5.24}$$

An example of this profile is shown in Figure 5.2. It can be seen that as the measurement range increases past r_{max}, the noise variance increases dramatically. From the perspective of using the measurement in a Kalman filter, the landmark becomes invisible. This model allows the sensor's finite range to be taken into account, while maintaining continuity of the variance.

The following results consider different scenarios where the robot goes out of its way to use the landmarks. For all examples, the initial covariance of the robot is zero, and all landmarks have initial covariances of

$$P_{LiLi} = \begin{bmatrix} .1 & 0 \\ 0 & .1 \end{bmatrix}. \tag{5.25}$$

Additionally, the value of the base measurement variance for Equation (5.22) $\beta = .1$, the sensor range $r_{max} = 3$, and the multiplier $\gamma = 10$. Finally, the odometry noise proportionality constant $\alpha = 0.15$. These are all reasonable numbers for a real mobile robot.

All simulations were implemented in Matlab using the `fmincon` function to perform the minimization of the cost function. The `fmincon` function finds the minimum of a constrained non-linear multi-variable function. The computer used for all simulations is a 2.8GHz Pentium 4 processor with 1.5 GB of RAM.

5.5.1 Robot Drives Past the Goal

The first example consists of only a single landmark with no obstacles. The landmark is located so that it is out of the sensor's effective range for all points along the straight-line path to the goal. The simulation result (Figure 5.3) illustrates the utility of using the landmark in the path planning process. If the robot were to take the straight line path to the goal, it would not get close enough to the landmark to use it for localization. However the path that minimizes the cost takes a detour to use the landmark. The final cost for the detouring path is 0.075. For comparison purposes, the final cost for the straight line path (in this case the shortest path) is 0.09. While this is not a large decrease in the cost, it illustrates the value of using this method in even simple situations.

For this simple example, there were three initial conditions generated. The time required to optimize all initial conditions was 0.73 seconds. It is interesting to note that the path only goes close enough to the landmark to enter the region where the sensing noise is relatively constant. There is no need to drive all the way to the landmark, as doing so would only increase the cost due to the extra odometric error.

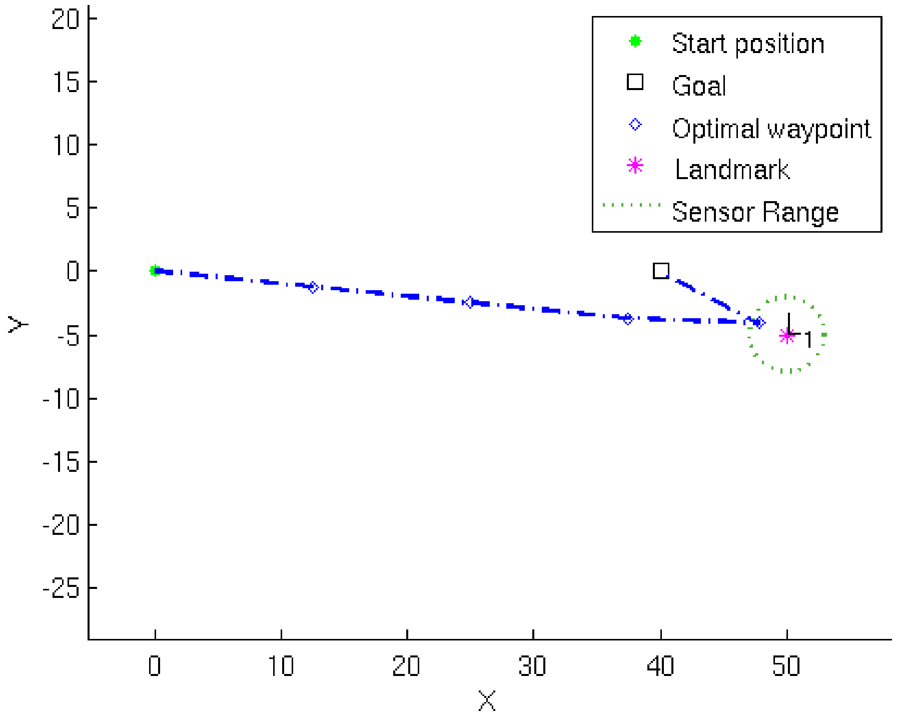

Figure 5.3: Example where the robot initially drives past the goal in order to use a landmark. For simplicity, the sensing range is plotted about the landmark - if the robot moves inside of that circle, it is close enough to see the landmark.

5.5.2 Robot Follows the Landmarks

The second example shown in Figure 5.4 consists of a scenario with multiple landmarks and no obstacles. There are four waypoints for this example. There were 31 initial conditions generated for this example, and a total run time of 39.5 seconds. The cost for the optimized path is 0.049, and the cost for the straight-line path is 0.09. Thus, there is a substantial reduction in final uncertainty when landmarks are taken into account.

Except for the two landmarks nearest the goal, the other landmarks in this example are far enough apart that no two landmarks can be seen at once by the robot. Because of this, the optimal path takes short steps from one landmark to the next. Another interesting result is the location of the third waypoint. It lies in an area where multiple landmarks are within the sensing range. Incorporating measurements from multiple landmarks during an update allows for a greater reduction in the estimate error than using just one landmark. It makes sense that these multi-view regions are optimal locations for an update.

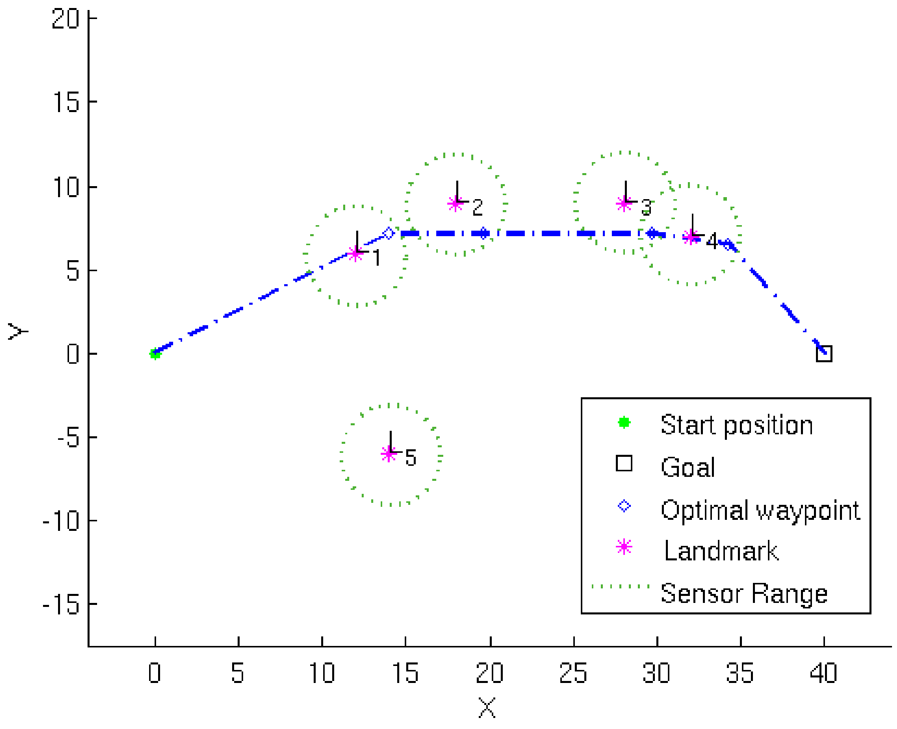

Figure 5.4: A multi-step path where the robot significantly deviates from the shortest path in order to use the landmarks.

5.5.3 Examples with Obstacles

Obstacles are used as constraints during the optimization process. The results from Appendix B were implemented to incorporate disk obstacles into the simulation. Figure 5.5 shows an example with some obstacles in the environment. It is similar to Figure 5.3 in that there is a landmark nearby, but past the goal. The cost for the optimized path using the landmarks for localization is 0.039, while the cost of the shortest path is 0.091. This example had 31 initial conditions, and the total run time was 299.6 seconds. As in the first example, the robot does not go directly to the goal, but first uses the landmark L_5 near the goal. This time, the optimal path also passes by the landmark L_1 to further reduce the expected uncertainty at the goal. This example demonstrates that even in the presence of obstacle constraints, the optimization procedure still finds a low-cost path through the environment.

Figure 5.6 shows results for the same landmark configuration as that of Figure 5.4, but with obstacles in the environment. The cost of the optimized path using the landmarks

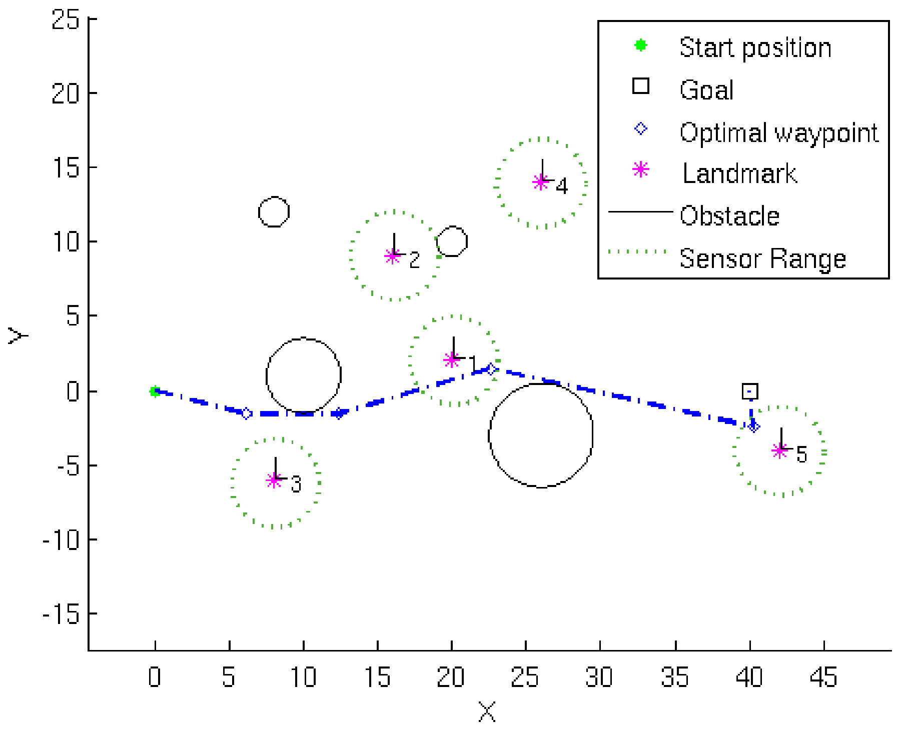

Figure 5.5: An simple example with obstacles.

is 0.056, while the cost of the shortest path is 0.093. There were 31 initial conditions for this example, with a total runtime of 121 seconds. The obstacles were placed such that they blocked the original optimal path. As the figure shows, the optimal path still follows the trail of landmarks, but it also closely skirts the obstacle boundaries. The optimization procedure uses the landmarks for localization, while still attempting to keep the path as short as possible.

5.6 Summary and Discussion

This chapter presented an optimization method that minimizes the robot's expected pose uncertainty at the goal. This off-line method assumes full knowledge of all obstacles in the environment. Simulation results clearly show the utility of explicitly using landmarks as part of the path planning process. This allows the robot to "go out of its way" to use landmarks to aid in its localization. Path length is traded off with localization capability.

One of the weaknesses of this method is its reliance on good initial guesses. Because

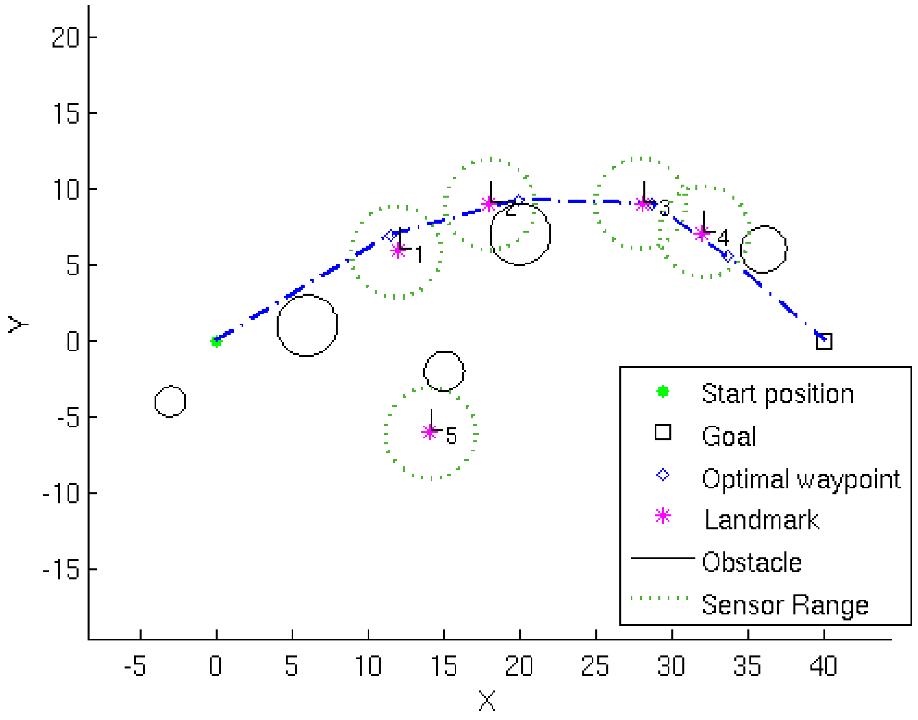

Figure 5.6: The same landmark configuration as that of Figure 5.4. Obstacles were placed to block the original optimal path.

the method is based on a gradient descent approach, the algorithm may end up missing a lower-cost path because none of the initial guesses lie within the basin of attraction for that particular local minimum. The heuristic described in Section 5.4 performs reasonably well most of the time, but of course it is not perfect.

An interesting possibility for the future would be to understand better the properties of the cost function. There are an infinite number of paths that go from the robot's current position to the goal and pass through a particular point. The robot is only interested in the "best" path that passes through a particular point, though. The surface defined by the cost of the best path through a particular point may have properties that could be exploited to more efficiently solve the problem.

However, this is just the first step towards bridging the gap between the fields of sensor-based motion planning and SLAM. In order to make this method a fully sensor-based algorithm, the full knowledge assumption must be removed. The robot will have to use its on-board sensors to incrementally build up its knowledge of the world, and appropriately replan its path. The following chapter presents one algorithm to do this, called Uncertain Bug.

Chapter 6

Uncertain Bug

This chapter presents the Uncertain Bug algorithm. Uncertain Bug is a sensor-based motion planner that incorporates knowledge of landmarks and takes into account the robot's localization ability and uncertainty along a path. Uncertain Bug makes heavy use of the optimization procedure presented in Chapter 5. The main difference between the setup for Uncertain Bug and the setup for the optimization procedure presented in Chapter 5 is that the robot does not have prior knowledge of the location of any obstacles in its environment.

The localization uncertainty that the cost function captures comes about from numerous sources. One source is the robot's imperfect odometry. The robot may be integrating accelerations and angular rates from a gyro to estimate its position. Or it may be counting the number of ticks on an encoder. Regardless of how the robot performs its odometry, it will be an imperfect estimate whose uncertainty grows with distance traveled. Another source of localization error is the landmarks. When the robot uses the landmarks to localize itself, the landmark's positions will be known imprecisely. Even if the landmarks were known perfectly, any realistic sensing process will return an imperfect measurement of the landmark positions.

The basic problem setup is discussed in Section 6.1. A description of the Uncertain Bug algorithm is given in Section 6.3, with particular attention given to the effects of uncertainty on the motion planning process. Properties of the algorithm are given in Section 6.4, and simulation results are presented in Section 6.5.

6.1 Setup and Definitions

Before explaining the Uncertain Bug algorithm the basic setup of the problem is described, which parallels that of the optimization method presented in Chapter 5. Pertinent details and important differences are presented in this section, but other similar aspects are left out for brevity. Please see Chapter 5 for the complete setup.

The position of the robot with respect to the global reference frame will be denoted by $\mathbf{x}_R$. The robot is modeled as a point moving in a plane, so

$$\mathbf{x}_R = \begin{bmatrix} x_r \\ y_r \end{bmatrix}. \tag{6.1}$$

The robot is instructed to move from its current position to some goal configuration, which is specified in the global coordinate system. The goal configuration will generally be specified as an (x, y) pair in the plane, i.e.,

$$\mathbf{x}_g = \begin{bmatrix} x_g \\ y_g \end{bmatrix}. \tag{6.2}$$

Before starting its task, the robot is given the position of N landmarks. The Cartesian position of the i^{th} landmark in the global reference frame is denoted by $\mathbf{x}_{Li}$:

$$\mathbf{x}_{Li} = \begin{bmatrix} x_{Li} \\ y_{Li} \end{bmatrix}. \tag{6.3}$$

The location of the landmarks may be uncertain, and the robot is equipped with a sensor capable of making noisy measurements of the landmark positions and detecting obstacles. This sensor has a finite sensing range, R, unless otherwise noted. For particulars on the landmark measurements, see Sections 5.1 and 2.2.2.

The state of the entire system is simply the state of the robot and all landmarks:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_R & \mathbf{x}_{L1} & \cdots & \mathbf{x}_{LN} \end{bmatrix}^T. \tag{6.4}$$

The estimate of the state is not assumed to be perfect. It is assumed that the robot is equipped with an estimator such as a Kalman filter (See Section 2.2) that can provide estimates of the state as well as the state estimate error covariance. The estimate error

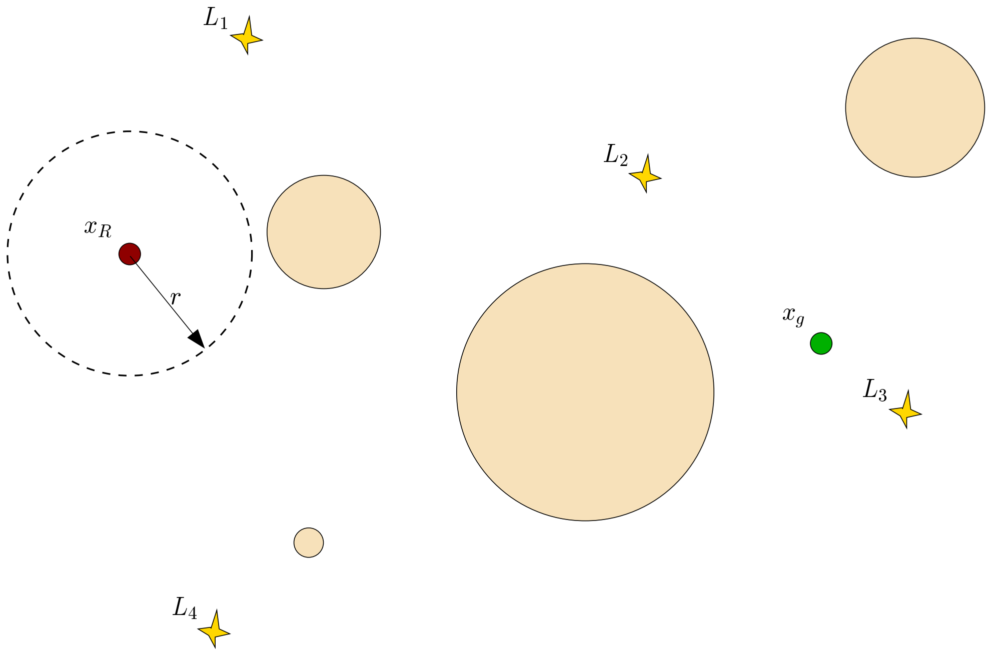

Figure 6.1: Setup assumed for the Uncertain Bug algorithm. The robot does not know the position of obstacles in the environment. The robot's sensor has a finite sensing range, r.

covariance will have a block structure:

$$P = \begin{bmatrix} P_{RR} & P_{RL} \\ P_{LR} & P_{LL} \end{bmatrix}, \tag{6.5}$$

where P_{RR} is the 2×2 covariance matrix of the robot's position error, P_{LL} is the $2N \times 2N$ matrix of landmark position error covariances, and $P_{RL} = P_{LR}^T$ are the cross-coupling error covariances. It is not assumed that the landmarks are known perfectly, so P_{LL} need not be all zeros. The robot is assumed to have initial covariances for all landmark locations.

While the cost function used for Tangent Bug and Optim-Bug is the robot's total path length, Uncertain Bug uses a cost function that aims to minimize the expected positional uncertainty at the goal:

$$J = ||P_{RR}(M/M-1)||, \tag{6.6}$$

where M is the number of steps the robot has taken to reach the goal, and $P_{RR}(M/M-1)$ is the covariance of the robot position estimate error at step M, given all measurements up to step $M-1$. Also recall that the norm is the Frobenius norm (See Equation 5.11).

It is assumed that if the robot is closer than a distance of ϵ to the goal, it will be able to identify the goal and the algorithm will terminate with success. Presumably, when the robot is within ϵ of the goal, it could use a local feedback algorithm such as visual servoing to guide its final adjustments. This assumption places a limit on the (expected) maximum magnitude of the robot's position estimate error covariance at the goal. In order to decide whether or not a given path meets this criteria, some level set of the robot's covariance matrix is chosen, and the assumption made that the robot will lie in that set. For example, one could pick the 3σ level, which translates to a 99.7% chance that the robot will be in that region. Let U_g denote the maximum dimension of the robot's covariance ellipse for a particular level set, i.e.,

$$U_g = c \max_i \sqrt{\lambda_i \left(P_{RR}(M/M-1) \right)}, \tag{6.7}$$

where c is a parameter that describes which particular level set is chosen, and $\lambda_i\left(P_{RR}\right)$ is the i^{th} eigenvalue of the matrix P_{RR}. I generally choose $c = 3$, although any other level could be chosen. If a particular path reaches the goal with $U_g < \epsilon$, the path is said to be feasible, or has *acceptable uncertainty*.

6.2 Motivation and Background

Algorithms such as Tangent Bug and Optim-Bug are useful because of their completeness property. But their assumption of perfect dead-reckoning makes them difficult to implement in realistic situations and renders any proof of convergence unworkable in practice.

Thrun's coastal navigation approach is one step towards combining the path planning problem with localization methods [46, 47]. They model the information content of all configurations in the environment, including the possibility that measurements could be corrupted (by a previously unknown moving obstacle, not just the statistical process noise). However, the coastal navigation approach requires a prior map of the environment in order to create this "information map."

One would like to be able to show that if there is a path from the start location to the goal, then the robot will find this path and reach the goal even in the presence of uncertain robot positions and noisy sensor measurements. Unfortunately it is not necessarily the case that the robot can reach the goal (even if the goal is reachable) with uncertain position measurements. The robot must learn about the location of obstacles in the environment. When encountering new information, the robot must often make choices between multiple alternative paths. It may make a choice that causes it to fail to be able reach the goal with some given level of positional uncertainty. For example, if the robot had chosen to go "left" around an obstacle instead of "right," it would have reached the goal successfully. This issue is discussed in more detail in Section 6.3.1.

6.3 The Uncertain Bug Algorithm

The goal of the Uncertain Bug algorithm is to find a path through an a priori unknown environment so that the robot reaches the goal with acceptable uncertainty, even in the presence of sensing and localization errors. This section presents a short overview of the Uncertain Bug algorithm, followed by more detailed discussion. Uncertain Bug has a single mode of operation—that of computing the optimal path to the goal and following it. Uncertain Bug uses the path optimization procedure from Chapter 5, but does so in an incremental manner. Because the robot does not know a priori the location of any obstacles in the environment, it must re-calculate its path when it encounters a new obstacle and gains more information about the world. In summary, the main step of the Uncertain Bug algorithm operates in the following manner:

1. Take a view of the world, i.e., compute the current visibility set $v(x_k)$ and then update the total visibility set V_{tot} with the newly gathered information.
2. Calculate the optimal path to the goal (using the optimization procedure presented in Chapter 5), taking into account knowledge of the environment obtained so far. Practically, this optimization step proceeds through these steps:
    (a) Generate a number of initial guesses.
    (b) Minimize the cost of each initial guess.
    (c) Check the cost of all of the optimized initial guesses. If none of them reaches the goal with $U_g < \epsilon$, the goal cannot be reached from the current position with acceptable uncertainty. Terminate with failure. Additional termination criteria are discussed below.

3. Follow the optimal path to the edge of the total visibility set, or to the goal if the path is contained within the current visibility set. If the robot is within a distance of ϵ the goal, terminate with success. If not, continue.

Uncertain Bug is similar to Optim-Bug in that it always computes the optimal path to the goal at each step given currently available information. But in Uncertain Bug, the aim is to minimize the robot's expected position estimate error covariance at the goal. Uncertain Bug relies heavily on the optimization approach of Chapter 5. Recall that this optimization process required knowledge of all obstacles in the environment. The obstacles are used as constraints—the path cannot intersect any obstacle. In Uncertain Bug, the robot only takes into account the obstacles that it has seen from its starting position to its current position.

As in the Optim-Bug algorithm, the robot must remember the locations of the obstacles it has encountered in the environment. This is simply a binary obstacle map, where a position is either impassable (an obstacle) or passable (freespace). This memory requirement is due to the nature of the optimization process—the obstacles are used as constraints in the path optimization process. The obstacle information stored in the map will also be subject to uncertainties. Issues that arise from map errors are discussed in Sections 6.4 and 6.6. It is assumed that when the robot is moving within V_{tot}, it uses a local obstacle avoidance scheme to miss any collisions that could arise due to the obstacle location and navigation uncertainty. As the robot sees more and more of the world, it can more accurately calculate the optimal path to the goal, or determine that the goal is unreachable.

Similar to Tangent Bug and Optim-Bug, the robot will never take a step beyond the region of the world that it currently knows about. It will only follow the current optimal path to the extent of its current knowledge. After that step, the robot will learn more information about the world that will allow it to make a more accurate calculation of the optimal path to the goal. In some cases, the optimal path will not change at all. In other cases the robot will see an obstacle that blocks the optimal path it calculated in the last time step. This new information will be incorporated into the robot's map, which is then used to find a new feasible path to the goal, if one exists.

6.3.1 Effects of Uncertainty

Recall that the main idea is for the robot to reach the goal reliably. This is the motivation for the choice of the cost function. One way to visualize the effects of localization uncertainty on the motion planning process is through level sets of U_g. Consider a surface in $\mathbb{R}^3$ where the height of the surface is equal to the value of U_g for the optimal path from that point to the goal. Recall that ϵ is the acceptable uncertainty at the goal. Choosing a particular value of ϵ can be thought of as slicing through the uncertainty surface with a plane at a height of ϵ. For any point where $U_g \geq \epsilon$ is a configuration from which the robot could not reach the goal with acceptable uncertainty.

If the robot knew the entire world geometry ahead of time, it could calculate the exact shape of this surface and immediately report whether or not it could reach the goal with acceptable uncertainty from its start location. But because it does not know the obstacle configuration, it can only make a guess at what the surface looks like, given its current knowledge. As it moves through the environment and learns more information about the obstacle configuration, the shape of the surface that is based on available knowledge will change as some paths become inaccessible because they are blocked by physical obstacles.

As previously noted, any point on this surface where $U_g \geq \epsilon$ is a point where the robot cannot reliably reach the goal within the given distance threshold. Connected regions where $U_g \geq \epsilon$ can be thought of as *uncertainty obstacles*, because they are regions of the configuration space that the robot must avoid. If the robot ever steps into an uncertainty obstacle, it can no longer guarantee that it will reach the goal with acceptable uncertainty, and must report failure. It may seem that the algorithm should just make sure that it never ventures into one of these uncertainty obstacles. Unfortunately, the geometry of the uncertainty obstacles is a function of the geometry of physical obstacles that may lie far ahead of the robot and outside of its current knowledge. The difficulty is that because the robot doesn't know the uncertainty obstacle configuration, it may not know it has stepped into one until it is too late.

Consider a scenario where there is one obstacle in the environment, but outside of the robot's initial sensing range. Also assume that the configuration of the landmarks is such that there are only two locally optimal paths that reach the goal with $U_g < \epsilon$, but one of these paths passes through the obstacle. This setup is shown in Figure 6.2. Because the

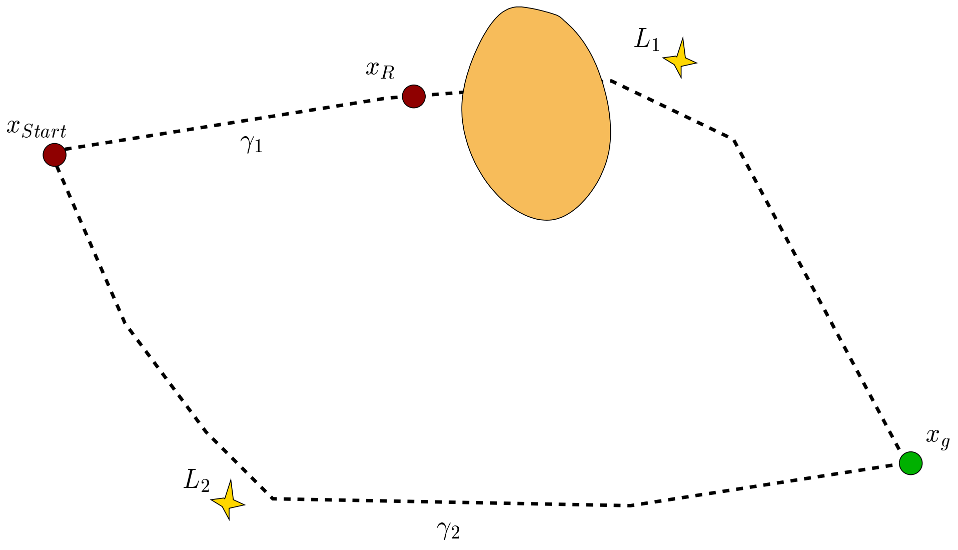

Figure 6.2: Hypothetical example where the robot chooses a path that causes it to step into an uncertainty obstacle.

robot does not know about the obstacle from its start position, it picks the path with the smallest cost. Assume that it chooses the path that intersects the obstacle, which is γ_1 in the figure. A few steps later, the obstacle enters the robot's sensing range. At that point, the robot realizes that the path it was following no longer reaches the goal with acceptable uncertainty because of the added path length needed to circumnavigate the obstacle. In addition, it has accrued enough uncertainty that it can no longer re-route to the other path and still reach the goal with acceptable uncertainty. So the robot will not know that it is stepping into an uncertainty obstacle until it has already done so. Thus, it is not generally possible to guarantee completeness of the algorithm. The strongest statement that can be made about convergence is that the algorithm will halt in a finite amount of time.

An interesting property of these uncertainty obstacles is that they will only grow in size as the robot learns more about the environment. They will never become smaller. Let $\mathcal{F}_\epsilon$ denote the region of freespace from which the robot can reach the goal with $U_g < \epsilon$, i.e,

$$\mathcal{F}_\epsilon = \{q \mid q \in \gamma(t), \gamma(t) \in \mathcal{P}_\epsilon\}, \tag{6.8}$$

where $\mathcal{P}_\epsilon$ is the set of all paths that reach the goal with acceptable uncertainty, $\gamma(t)$ is a particular path, and $q \in \mathbb{R}^2$ is a robot configuration. Let $\mathcal{UO}$ denote the set of uncertainty obstacles. The uncertainty obstacles are the parts of freespace not in $\mathcal{F}_\epsilon$, i.e.,

$$\mathcal{UO} = \mathcal{F} \setminus \mathcal{F}_\epsilon. \tag{6.9}$$

With no obstacles in the world, $\mathcal{F}_\epsilon$ could be calculated exactly from the start. It would be a function of landmark locations, sensing uncertainty, and odometric uncertainty. But as the robot learns about the location of more obstacles, the set of available configurations q in $\mathcal{F}_\epsilon$ will be reduced. Going back to the surface analogy, it is as if more and more configurations end up above the level set of $U_g = \epsilon$.

6.4 Uncertain Bug Algorithm Properties

The following proofs of the properties of the Uncertain Bug algorithm will assume that the goal is not placed inside an obstacle, i.e., the goal is physically reachable. Discussion of what happens when the goal is inside an obstacle follows in Section 6.6.

To be able to say anything concrete about the algorithm, one must make certain assumptions about the performance of the underlying optimization process. The optimization process must return a feasible path if one exists. Practically speaking, this means that the optimization process is implemented perfectly. Of course in reality this may be difficult. Because the optimization process uses a gradient descent method, if there is not an initial condition generated that lies in the basin of attraction of the global minimum, that path will be overlooked.

Axiom 6.1. Given $V_{tot}(x_k)$, the optimization algorithm of Chapter 5 will return a locally optimal, feasible path if one exists, or return with failure.

Success or failure of Uncertain Bug depends on the size of the robot's covariance at the goal. If the robot's method for tracking this covariance is faulty, then the robot may think that a path is feasible when in fact it is infeasible. So it must be assumed that this covariance reflects the true estimate error covariance. This means that all of the parameters that affect the estimate such as the odometry noise, the sensing noise, and the landmark position estimate covariances are correct.

Axiom 6.2. The robot's covariance estimate is correct.

If the covariance estimate is conservative, i.e., it always bounds the true estimate, then this could cause the robot to prematurely terminate with failure even if there still exists a path to the goal with $U_g < \epsilon$. However, if the covariance estimate is overconfident, then the robot may think that the goal is still reachable with acceptable uncertainty when it is not.

One can assume that freespace is bounded, as could be the case in an office-like environment or the surface of Mars (which is large, but still bounded). On the other hand, a few intuitive arguments can be used to place a tighter bound on $\mathcal{F}_\epsilon$.

Lemma 6.1. *Let there be N landmarks, each at a location (L_{ix}, L_{iy}), in a bounded set. The landmark positions are imperfect. The robot is modeled as a point operating in $\mathbb{R}^2$ with a position of (x_R, y_R), and is equipped with a sensor with a finite range, R. The robot's odometry is assumed to be such that the variance in its velocity is proportional to the velocity, i.e., $\sigma_v = \alpha V$. The goal is at a position of (g_x, g_y).*

Under these conditions, the region of freespace from which the robot can reach the goal with $U_g < \epsilon$ is bounded.

Proof. Given the setup described above and further described in Sections 5.1 and 6.1, $\mathcal{F}_\epsilon$ is bounded by the union of the squares given by the following inequalities:

$$\max\left(|L_{ix} - x_R|, |L_{iy} - y_R|\right) < \frac{R}{3\alpha} \tag{6.10}$$

$$\max\left(|g_x - x_R|, |g_y - y_R|\right) < \frac{\epsilon}{3\alpha} \tag{6.11}$$

That is, if any of the inequalities are violated, then the robot is guaranteed to not be in $\mathcal{F}_\epsilon$. Derivations of these conditions are given in Appendix D. □

Lemma 6.2. *At each step, the robot will see a new part of the world, i.e., V_{tot} will grow, or the algorithm will terminate with failure due to localization errors.*

Proof. The proof follows that of Lemma 4.1 for Optim-Bug, with an additional termination condition caused by localization errors. The robot is assumed to have moved to x_{k+1} from x_k.

By definition of the algorithm, the robot will always follow the current optimal path to the edge of V_{tot}. Although the robot is assumed to move a distance of ϵ (not necessarily the same ϵ within which the robot must reach the goal), because the sensing range, R, is greater than ϵ, there will be a finite-sized region of $\mathcal{F}$ that the robot will view for the first time.

As long as the termination criteria discussed below is not violated, the robot is guaranteed to see a new part of the environment, and as such V_{tot} will grow at each step.

If the additional uncertainty accrued by moving from x_k to x_{k+1} is greater than $R - \epsilon$, then it cannot be guaranteed that the robot will see a new part of it's environment at step $k + 1$. When this occurs, the algorithm must terminate with failure.

Otherwise, by analysis of Lemma 4.1, the robot will see a new portion of the environment.

□

This additional termination condition can be illustrated as follows: Consider an example where the robot has mapped a relatively large V_{tot}. Recall that the algorithm dictates that the robot will always follow the optimal path to the edge of V_{tot}. If at the next step the robot drives a long distance across V_{tot}, the localization errors accrued from that step alone could become large enough such that the algorithm cannot guarantee that the robot will see a new part of $\mathcal{F}$.

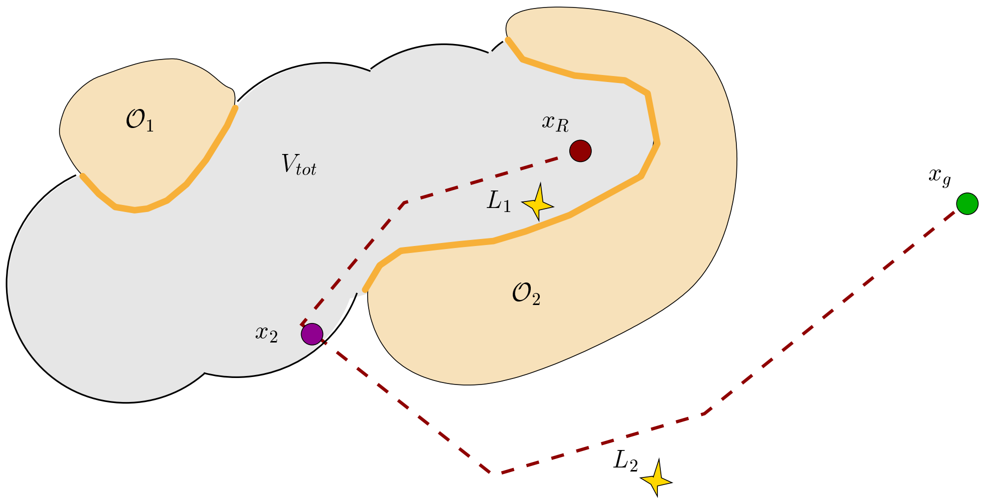

Figure 6.3: Figure illustrating an extra termination condition for Uncertain Bug. The next robot path is shown by the dashed line. The robot will follow the path to the point labeled x_2.

A hypothetical scenario where this condition could arise is illustrated in Figure 6.3. The next robot path is shown by the dashed line. By definition of the algorithm, the robot will step to location x_2. If any dimension of the covariance ellipse grows by a value of more than $R - \epsilon$, then that means it would be possible for the robot's true position to lie at the center of one of the past visibility sets, v_k, in which case the robot will not see a new part of the environment.

Proposition 6.3. *Uncertain Bug will not terminate with failure unless no feasible path to the goal exists from the robot's current position, x_k, or localization errors are too large to continue.*

Proof. By Axiom 6.1, the optimization algorithm will return a feasible path to the goal if one exists, or return with failure. This guarantees that the robot will always get a new path if the goal is still reachable with $U_g < \epsilon$. Alternatively, the algorithm will return with failure if a path does not exist.

Therefore, Uncertain Bug will not terminate with failure unless no feasible path exists, or the localization errors are too large to continue. □

Proposition 6.4. *Uncertain Bug will terminate in a finite amount of time.*

Proof. If the robot knew the configuration of all obstacles prior to starting, it could immediately calculate whether or not there exist any feasible paths to the goal. But Uncertain Bug does not assume prior knowledge of obstacles—the robot must map the obstacles as it attempts to reach the goal.

Because of Lemma 6.1, the total area that the robot *could* map out is finite. Because of Lemma 6.2, the robot will see a new portion of this bounded area at every step.

In the worst case, the robot will explore all of $\mathcal{F}_\epsilon$ before determining that the goal is reachable or not. This will occur in a finite amount of time, because the robot sees a new, finite-sized portion of $\mathcal{F}_\epsilon$ at every step. The robot will reach the goal before mapping all of $\mathcal{F}_\epsilon$, or terminate after a finite number of steps because all of $\mathcal{F}_\epsilon$ has been seen and no path to the goal exists with acceptable uncertainty.

Therefore Uncertain Bug will terminate after a finite amount of time. □

Two different termination scenarios can occur as a result of errors introduced into the map. These errors are directly caused by the robot's positional error. However, the map errors only affect cases where the optimal path is very near the threshold of $U_g > \epsilon$. If the optimal path has $U_g \ll \epsilon$, small errors in the map will not affect it greatly enough.

In one case, the robot may think the goal is not reachable with acceptable uncertainty because an obstacle placed (incorrectly) in the map blocks any path with $U_g < \epsilon$. If the obstacle were placed correctly in the map, the robot would determine that the goal is still reachable with acceptable uncertainty. In this case, Uncertain Bug will still terminate in a finite amount of time. Although the goal is reachable with $U_g < \epsilon$, the robot has no way of knowing this. The map it builds is the best information it has available.

A second situation can occur where the robot thinks the goal is still reachable with acceptable uncertainty when it is not. In this case, the errors in the map lead the robot to be overconfident. The robot continues following the optimal path to the goal (which has $U_g > \epsilon$), but will eventually discover new information that leads it to conclude the goal is not reachable with acceptable uncertainty. For example, the robot's local obstacle avoidance scheme could cause it to drive farther than expected. Thus, the robot will eventually determine that the goal is not reachable with acceptable uncertainty. Although errors in the map can cause undesirable termination, Uncertain Bug will terminate in a finite amount of time.

6.5 Simulation Results

Simulations have been performed using the optimization algorithm as presented in Section 5. Recall that this method uses gradient descent to optimize a number of initial conditions. The initial guesses are created using the heuristic described in Section 5.4. Representative results are presented to illustrate performance of the algorithm. For all simulations, the value of α, the odometry noise parameter, was set at 0.15. Recall that an exponential model for the sensing noise is used in order to help keep the cost function smooth. See Sections 5.3 and 5.5 for details. For these sensing noise parameters, $\beta = 0.1$, and $\gamma = 10$. All initial landmark covariances were set to $\texttt{diag}(0.1, 0.1)$. The robot's sensing range is 3 units. Note that in all plots, landmarks have a circle of a radius equal to the robot's sensing range plotted around them. So if the robot lies within the circle, then it can see the landmark. Recall that it is assumed that obstacles block sensing of other obstacles, but that obstacles do not block sensing of landmarks.

Figure 6.4 shows a complete simulation run of Uncertain Bug with two obstacles and three landmarks in the environment. The robot does not know about the obstacles when it begins. For this example, the robot requires 17 steps to reach the goal. The final cost at the goal is only 39% of the cost of the Tangent Bug path, i.e., the robot reaches the goal with

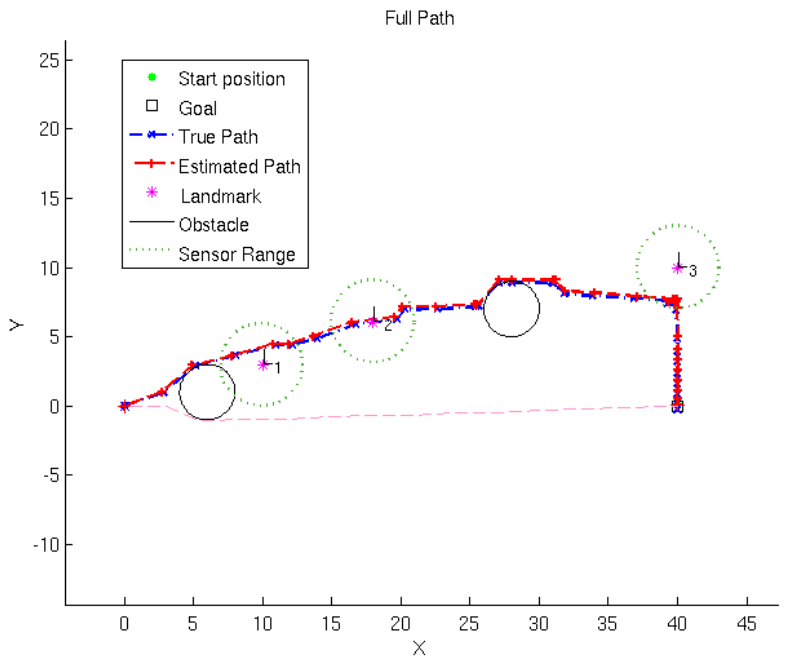

Figure 6.4: A simulation of Uncertain Bug. The Tangent Bug path is shown as the pink dashed line. Final cost at the goal is 39% of the cost of the Tangent Bug path.

much less positional uncertainty than Tangent Bug. Because the path is chosen to exploit the landmarks for better localization, the robot has a much reduced position estimate error at the goal.

Figure 6.5 shows detailed results from the ninth and tenth steps along the robot's path to the goal. When the robot plans step number 9, it does not know about the obstacle. Because of this, the optimal path from the robot's start position for step 9 passes through the obstacle. Upon executing step 9, the obstacle is within the robot's sensing range. This obstacle is incorporated into the robot's map, and the new plan takes this obstacle into account. Step 10 appropriately avoids the obstacle, while still attempting to use the landmarks for an improved position estimate at the goal.

Figure 6.6 shows results from a simulation run of Uncertain Bug with nine obstacles in the environment. The final cost of the Uncertain Bug path is only 32% of the cost of the Tangent Bug path. As in previous examples from Chapter 5, the robot makes use of the landmark near the goal (L_5). It may seem that the robot should follow the locally shortest path (the Tangent Bug path) to landmark L_1, as opposed to the path it chooses. In this

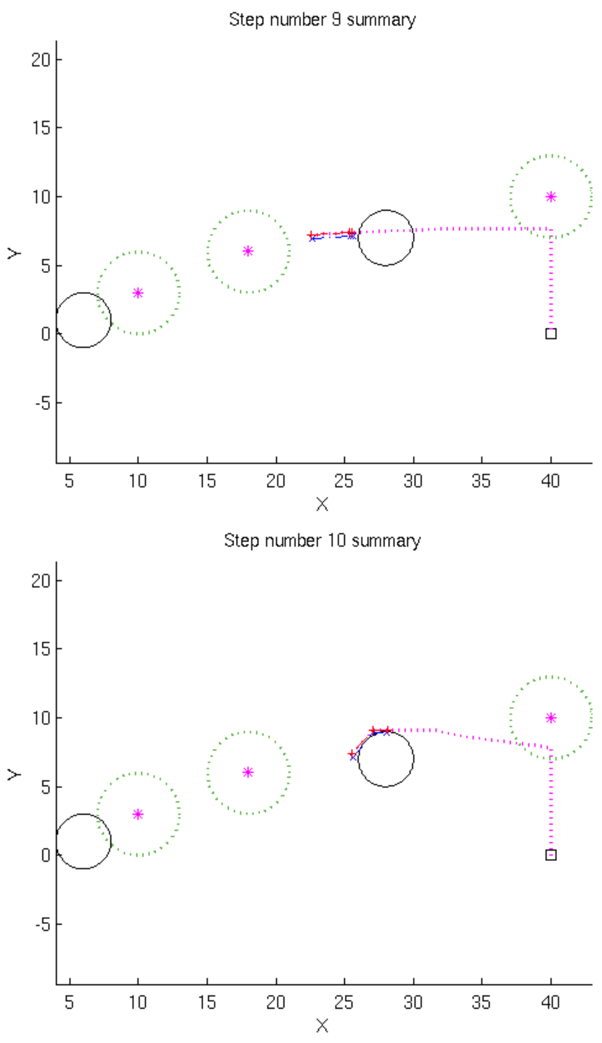

Figure 6.5: The planned path for steps 9 and 10. Note that when the robot sees the obstacle, it re-plans the path, taking into account the new information about the world.

example, there was no initial condition generated that passed between the obstacles like the locally shortest path does.

6.6 Discussion

This chapter presented a new sensor-based motion planning algorithm called Uncertain Bug. Uncertain Bug aims to minimize the expected robot position estimate error covariance at the goal, in order to improve the chances that the robot will be able to move close enough to the goal to recognize it. Given the assumptions presented, Uncertain Bug is guaranteed to terminate in a finite amount of time. Unfortunately, it is not possible to guarantee that

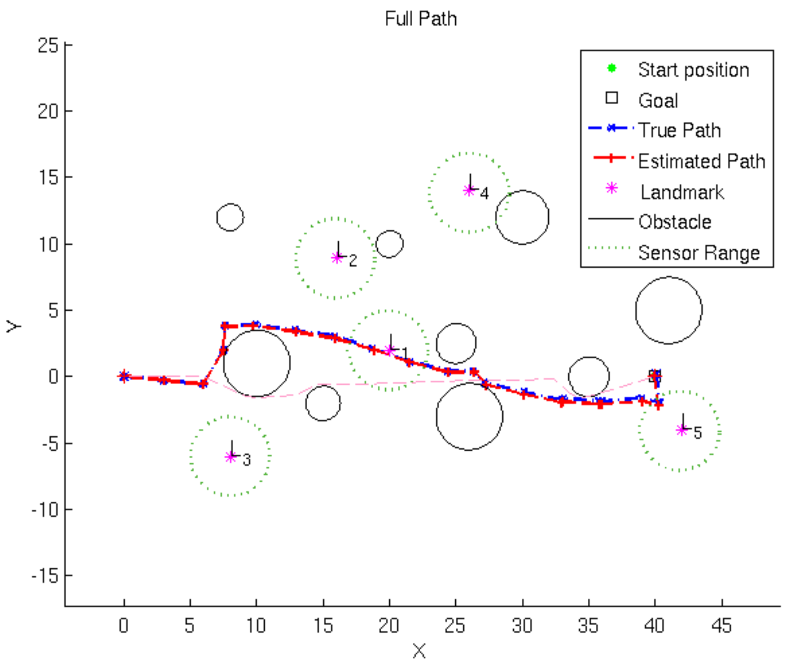

Figure 6.6: A simulation of Uncertain Bug. The Tangent Bug path is shown as the pink dashed line. Final cost at the goal is 32% of the cost of the Tangent Bug path.

the robot will reach the goal using Uncertain Bug, even if a path to the goal exists.

If the goal is not physically reachable, i.e., it lies inside of an obstacle, Uncertain Bug will still terminate in a finite amount of time. The fact that that map itself is uncertain makes this situation much more difficult to recognize than in the case of perfect dead-reckoning (such as Tangent Bug or Optim-Bug). To guarantee that the goal is enclosed in an obstacle, the robot would need to drive an extra distance around the obstacle. This extra distance would be on the order of the robot's positional uncertainty. In effect, the robot would be making sure that it has seen enough "overlap" in the obstacle boundary to be certain that the obstacle is closed around the goal. If the robot had the capability to recognize features on obstacles, then it could verify that it had circumnavigated an obstacle by identifying the same feature.

This type of algorithm that takes the robot's localization capability into account as part of the planning process would be useful in wide-ranging situations. For robotic planetary exploration, a rover may have a small number of landmarks from either an overhead satellite or from a rover that has navigated the same area in the past. Another scenario is that of underwater navigation. Underwater environments typically have a sparse number of features that can be used for localization. Both of these examples could benifit from an algorithm such as this.

While I have outlined the different termination conditions, the robot could still continue seeking the goal once some of the termination criteria are met. The termination conditions that deal with the size of the robot's uncertainty do not guarantee that there is not a path that reaches the goal, just that there is no path that reaches the goal with acceptable uncertainty. It is entirely possible that the robot could continue planning paths to the goal, "get lucky," and find the goal. Of course, the robot could also continue for an indeterminate amount of time while trying to do this.

Further insights could be gained into the properties of the algorithm if one considered the limit as uncertainty goes to zero. Of course, if there is no uncertainty, the cost function will be zero everywhere. The current cost function implicitly encodes path length, as longer paths will accrue more odometric uncertainty. This analysis could require adding an independent path-length parameter to the cost function to account for this.

Another possible extension of this work would be to determine if there are cases where completeness could be guaranteed. Is there some single (or a small number) parameter that describes the different values of the noise (odometry, sensing, landmark) that can be used to prove completeness? For example, if one can place a worst-case bound on the maximum level of positional uncertainty in the environment, what other conditions need to be met to be able to guarantee that the robot will reach the goal if the goal is in fact reachable?

Chapter 7

Conclusion

7.1 Summary of Contributions

I have developed and presented new tools and algorithms for mobile robot navigation. The weighted scan-matching method of Chapter 3 can be used as a much improved form of odometry. Optim-Bug, presented in Chapter 4, is a complete and correct sensor-based motion planning algorithm. Chapters 5 and 6 describe a method to find the path that minimizes the robot's position error at the goal, and an associated sensor-based algorithm, Uncertain Bug, that uses this optimization method.

The weighted scan-matching algorithm is a new method for estimating robot displacement based on dense range measurements. In particular, I investigated the effects of different error and noise sources on the convergence and accuracy properties of these motion from structure algorithms. Experiments showed that careful attention to the details of error modeling can significantly enhance overall displacement and covariance estimation accuracy.

Optim-Bug is a complete and correct sensor-based planner. The algorithm is based on finding and following the shortest path in the environment. Optim-Bug assumes a point robot with a sensor that has a finite range, and requires that the robot have perfect dead-reckoning. It also needs the robot to keep a memory of obstacles it has seen. Optim-Bug is guaranteed to reach the goal in a finite amount of time if it is reachable, or terminate in a finite amount of time if the goal is not reachable.

My path optimization method minimizes the robot's expected pose uncertainty at the goal. This off-line method assumes full knowledge of all obstacles in the environment. Simulation results show the utility of explicitly using landmarks as part of the path planning process. This allows the robot to "go out of its way" to use landmarks to aid in it's localization. Path length is traded off with localization capability.

Uncertain Bug is a new sensor-based motion planning algorithm that takes the robot's localization capability into account when planning the path. The algorithm aims to minimize the expected robot position estimate error covariance at the goal, in order to improve the chances that the robot will be able to move close enough to the goal to recognize it. Uncertain Bug is guaranteed to terminate in a finite amount of time. Unfortunately, it is not possible to guarantee that the robot will reach the goal using Uncertain Bug, even if a path to the goal exists.

7.2 Future Directions

This thesis suggests continued research into many areas related to mobile robot navigation. As presented, Optim-Bug assumes the robot is operating in a planar environment. Optim-Bug's approach of always following the shortest path given current information may extend well to higher dimensions. In higher dimensions, the goal-tangent lines would become surfaces that are tangent to obstacles and the goal. Instead of building up an obstacle one segment at a time, the robot would map out the obstacle one "patch" at a time.

The path optimization approach of Chapter 5 is heavily dependent on the initial conditions. Better understanding of the cost function, as well as the "path space" that the optimal solution lies in could provide insight into choosing better initial conditions, or using something different from gradient-descent.

One possible approach to this would be to find if there is an uncertain analog to the goal-tangent lines. In Optim-Bug, it is known that the optimal paths follow goal-tangent lines. In the uncertain case, the optimal paths are generally *not* tangent to obstacle boundaries. If there were a construct similar to the goal-tangents, they could be exploited to improve the algorithm performance, and perhaps even prove completeness in some situations.

It is unfortunate that Uncertain Bug is not guaranteed to reach the goal. Perhaps there is a single parameter or small family of parameters that, under certain situations, can be used to guarantee that the robot will reach the goal, even in the presence of odometric, sensing, and landmark uncertainties. For example, if one places a bound on the maximum robot positional uncertainty in the environment, would that allow completeness to be proven?

Appendix A

Weighted Scan Matching Derivations

A.1 Weighted Translation Solution

Recall the log-likelihood formula of Eq. (3.15). Since D^{ij} is independent of x_{ij} and y_{ij}, the necessary condition for an extremal in the log-likelihood function with respect to the variable $p_{ij} = [\ x_{ij}\ y_{ij}\ ]^T$ is

$$\begin{aligned}
\nabla_{p_{ij}}(M^{ij}) = 0 \quad &\Leftrightarrow \\
\sum_{k=1}^{n_{ij}} \nabla_{p_{ij}} \left((\varepsilon_k^{ij})^T (P_k^{ij})^{-1} \varepsilon_k^{ij} \right) = 0 \quad &\Leftrightarrow \\
2 \sum_{k=1}^{n_{ij}} \left[\left(\nabla_{p_{ij}} (\varepsilon_k^{ij})^T \right) (P_k^{ij})^{-1} \varepsilon_k^{ij} \right] = 0 \quad &\Leftrightarrow \\
-2 \sum_{k=1}^{n_{ij}} \left[I \ (P_k^{ij})^{-1} \varepsilon_k^{ij} \right] = 0 \quad &\Leftrightarrow \\
\sum_{k=1}^{n_{ij}} \left[(P_k^{ij})^{-1} (\vec{u}_k^i - R_{ij} \vec{u}_k^j - p_{ij}) \right] = 0. &
\end{aligned}$$

Rearranging this formula results in Eq. (3.16).

A.2 Weighted Rotation Solution

Given an initial estimate of the translational displacement $\hat{p}_{ij}$, the rotational displacement can be derived by maximizing the likelihood function in Eq. (3.12), or equivalently, the log-likelihood function in Eq. (3.15) with respect to $\phi_{ij} = \phi$, i.e.,

$$\frac{\partial M^{ij}(\phi)}{\partial \phi} = 0. \tag{A.1}$$

Instead of directly computing the gradient of M^{ij} with respect to ϕ, we calculate it as follows:

$$\frac{\partial M^{ij}(\phi)}{\partial \phi} = \frac{\partial M^{ij}(\hat{\phi} + \delta\phi)}{\partial(\delta\phi)} \frac{\partial(\delta\phi)}{\partial \phi} = \frac{\partial M^{ij}(\delta\phi)}{\partial(\delta\phi)}, \tag{A.2}$$

where we used the relation

$$\phi = \widehat{\phi} + \delta\phi \Rightarrow \frac{\partial \phi}{\partial(\delta\phi)} = 1. \tag{A.3}$$

Here we derive an exact expression for the quantity M^{ij} as a function of $\delta\phi$. From the Taylor series expansion for the functions sin and cos we have

$$\begin{aligned} \cos\phi &= \cos\widehat{\phi} - \frac{1}{1!}\sin\widehat{\phi}\ \delta\phi - \frac{1}{2!}\cos\widehat{\phi}\ \delta\phi^2 + \cdots, \\ \sin\phi &= \sin\widehat{\phi} + \frac{1}{1!}\cos\widehat{\phi}\ \delta\phi - \frac{1}{2!}\sin\widehat{\phi}\ \delta\phi^2 - \cdots. \end{aligned}$$

Substituting in Eq. (3.2), the rotational matrix R_{ij} can be written as

$$R_{ij}(\phi) = \left(I + \frac{1}{1!}J\delta\phi - \frac{1}{2!}I\delta\phi^2 - \frac{1}{3!}J\delta\phi^3 + \ldots \right) \hat{R}_{ij}(\widehat{\phi}),$$

where J is defined in Eq. (3.19). The error ε_k^{ij} between two corresponding laser points, defined in Eq. (3.5), can be described as a function of the orientation error $\delta\phi$:

$$\begin{aligned} \varepsilon_k^{ij} &= \vec{u}_k^i - p_{ij} - R_{ij}\vec{u}_k^j \\ &= \vec{u}_k^i - p_{ij} - \hat{R}_{ij}\vec{u}_k^j - \frac{1}{1!}J\hat{R}_{ij}\vec{u}_k^j\delta\phi \\ &\quad + \frac{1}{2!}\hat{R}_{ij}\vec{u}_k^j\delta\phi^2 + \cdots. \end{aligned} \tag{A.4}$$

The covariance matrix for the matching error at the k^{th} point correspondence of poses i and j in Eq. (3.10) can also be described as a function of $\delta\phi$:

$$\begin{aligned} P_k^{ij}(\delta\phi) &= Q_k^{ij} + \widetilde{S}_k^{ij} + (J\widetilde{S}_k^{ij} - \widetilde{S}_k^{ij}J)\delta\phi \\ &- (\widetilde{S}_k^{ij} + J\widetilde{S}_k^{ij}J)\delta\phi^2 - \frac{2}{3}(J\widetilde{S}_k^{ij} - \widetilde{S}_k^{ij}J)\delta\phi^3 \\ &+ \frac{1}{3}(\widetilde{S}_k^{ij} + J\widetilde{S}_k^{ij}J)\delta\phi^4 + \cdots, \end{aligned} \tag{A.5}$$

where

$$\widetilde{S}_k^{ij} = \hat{R}_{ij}(\hat{\phi})S_k^{ij}\hat{R}_{ij}^T(\hat{\phi}).$$

The inverse $I_k^{ij}(\delta\phi) = (P_k^{ij}(\delta\phi))^{-1}$ of the covariance matrix can be computed using Taylor series expansion as

$$I_k^{ij}(\delta\phi) = I_k^{ij(0)}(0) + I_k^{ij(1)}(0)\delta\phi + \frac{1}{2!}I_k^{ij(2)}(0)\delta\phi^2 + \ldots \tag{A.6}$$

with

$$I_k^{ij(n)}(0) = \left.\frac{\partial^n(I^{ij}(\delta\phi))}{\partial(\delta\phi)^n}\right|_{\delta\phi=0},$$

where

$$\begin{aligned} I_k^{ij(0)}(0) &= (P_k^{ij}(0))^{-1} = (P_k^{ij})^{-1} = (Q_k^{ij} + \widetilde{S}_k^{ij})^{-1}, \\ I_k^{ij(1)}(0) &= -(Q_k^{ij} + \widetilde{S}_k^{ij})^{-1}(J\widetilde{S}_k^{ij} - \widetilde{S}_k^{ij}J)(Q_k^{ij} + \widetilde{S}_k^{ij})^{-1}, \\ I_k^{ij(2)}(0) &= 2I_k^{ij(1)}(0)P_k^{ij}(0)I_k^{ij(1)}(0) + 2(\widetilde{S}_k^{ij} + J\widetilde{S}_k^{ij}J). \end{aligned}$$

By substituting from Eq.s (A.4), (A.6) to Eq. (3.13) we have:

$$
\begin{aligned}
M^{ij} = & \frac{1}{2}\sum_{k=1}^{n_{ij}}\{p_k^T I_k^{ij}(0)p_k \\
& + \left[-2p_k^T I_k^{ij}(0)Jq_k + p_k^T I_k^{ij(1)}(0)p_k\right]\delta\phi \\
& + \Big[p_k^T I_k^{ij}(0)q_k - q_k^T J I_k^{ij}(0)Jq_k \\
& \quad - 2p_k^T I_k^{ij(1)}(0)Jq_k + \frac{1}{2}p_k^T I_k^{ij(2)}(0)p_k\Big]\delta\phi^2 \\
& + \ldots \},
\end{aligned}
\tag{A.7}
$$

where

$$
\begin{aligned}
p_k &= \vec{u}_k^i - p_{ij} - \hat{R}_{ij}\vec{u}_k^j, && \text{(A.8)} \\
q_k &= \hat{R}_{ij}\vec{u}_k^j, && \text{(A.9)} \\
\|p_k\| &<< \|q_k\|. && \text{(A.10)}
\end{aligned}
$$

Note that there has been *no* approximation made up to this point. Eq. (A.7) is a complete expression of the cost function M_{ij}, expressed as an infinite series of terms polynomial in the orientation estimation error $\delta\phi$. In order to minimize this function, we approximate it after considering a limited number of terms. For small errors in the initial orientation estimate ($\delta\phi < \pi/6$), a second-order approximation is sufficient when a large number of point correspondences are available. Higher-order approximations are necessary as the number of point correspondences decreases.

By substituting Eq. (A.7) in Eq. (A.2) and employing Eq. (A.10)[1] we derive the expression for the orientation displacement error of Eq. (3.18).

A.3 Covariance Estimation

Here we consider the estimation problem where n_{ij} measurements $Z = [Z_1^T \ \ldots \ Z_{n_{ij}}^T]^T$ (with $Z_k = [(\vec{u}_k^i)^T \ (\vec{u}_k^j)^T]^T$) are processed to derive an estimate of a vector λ of the motion

[1]Eq. (A.10) expresses the fact that the point correspondence errors are very small compared to the distances to these points.

parameters

$$\hat{\lambda} = \begin{bmatrix} \hat{p}_{ij} \\ \hat{\phi}_{ij} \end{bmatrix} = \begin{bmatrix} h_p(Z) \\ h_\phi(Z) \end{bmatrix} = h(Z) \tag{A.11}$$

with the expressions for functions h_p and h_ϕ given by Equations (3.16) and (3.18). A first-order approximation of the error in the estimate of the parameter vector $\hat{\lambda}$ is given by

$$\varepsilon_{\widehat{\lambda}} = \nabla_Z^T h(Z)\ \varepsilon_Z = \sum_{k=1}^{n_{ij}} \nabla_{Z_k}^T h(Z_k)\ \varepsilon_{Z_k} \tag{A.12}$$

with

$$\nabla_Z^T h(Z) = \begin{bmatrix} \nabla_{Z_1}^T h(Z) & \dots & \nabla_{Z_{n_{ij}}}^T h(Z) \end{bmatrix} \tag{A.13}$$

and

$$\nabla_{Z_k}^T h(Z) = \begin{bmatrix} \nabla_{Z_k}^T h_p(Z) \\ \nabla_{Z_k}^T h_\phi(Z) \end{bmatrix}. \tag{A.14}$$

Note that

$$E\{\varepsilon_{\hat{\lambda}}\} = E\{\nabla_Z^T h(Z)\ \varepsilon_Z\} = \nabla_Z^T h(Z)\ E\{\varepsilon_Z\} = \vec{0}_{3\times 1}.$$

The covariance of the estimate $\widehat{\lambda}$ is

$$P^{ij} = P_{\widehat{\lambda}} = E\{\varepsilon_{\widehat{\lambda}}\varepsilon_{\widehat{\lambda}}^T\} = \nabla_Z^T h(Z)\ P_Z\ \nabla_Z h^T(Z), \tag{A.15}$$

where

$$P_Z = E\{\varepsilon_Z \varepsilon_Z^T\} = \begin{bmatrix} P_{Z_1} & . & 0 \\ . & & . \\ 0 & . & P_{Z_{n_{ij}}} \end{bmatrix} \tag{A.16}$$

and

$$\begin{aligned} P_{Z_k} &= E\{\varepsilon_{Z_k}\varepsilon_{Z_k}^T\} = E\{\begin{bmatrix} \delta\vec{u}_k^i \\ \delta\vec{u}_k^j \end{bmatrix}\begin{bmatrix} (\delta\vec{u}_k^i)^T & (\delta\vec{u}_k^j)^T \end{bmatrix}\} \\ &= \begin{bmatrix} Q_k^{ij} & 0 \\ 0 & S_k^{ij} \end{bmatrix}. \end{aligned} \tag{A.17}$$

Substituting from Equations (A.13) and (A.16) in Equation (A.15) yields

$$\begin{aligned} P_{\widehat{\lambda}} &= \sum_{k=1}^{n_{ij}} \nabla_{Z_k}^T h(Z)\ P_{Z_k}\ \nabla_{Z_k} h^T(Z) \\ &= \sum_{k=1}^{n_{ij}} \begin{bmatrix} \nabla_{Z_k}^T h_p(Z) \\ \nabla_{Z_k}^T h_\phi(Z) \end{bmatrix} P_{Z_k} \left[\nabla_{Z_k} h_p^T(Z)\ \ \nabla_{Z_k} h_\phi^T(Z)\right] \\ &= \begin{bmatrix} P_{pp} & P_{p\phi} \\ P_{\phi p} & P_{\phi\phi} \end{bmatrix}. \end{aligned} \tag{A.18}$$

For $\xi, \zeta \in \{p, \phi\}$ each of the previous sub-matrices can be written as

$$\begin{aligned} P_{\xi\zeta} &= \sum_{k=1}^{n_{ij}} \nabla_{Z_k}^T h_\xi(Z) P_{Z_k} \nabla_{Z_k} h_\zeta^T(Z) \\ &= \sum_{k=1}^{n_{ij}} \Big((\nabla_{\vec{u}_k^i}^T h_\xi)\ Q_k^{ij}\ (\nabla_{\vec{u}_k^i} h_\zeta^T) \\ &\qquad + (\nabla_{\vec{u}_k^j}^T h_\xi)\ S_k^{ij}\ (\nabla_{\vec{u}_k^j} h_\zeta^T) \Big), \end{aligned} \tag{A.19}$$

where we substituted from Eq. (A.17) and the relation

$$\nabla_{Z_k}^T h_\xi(Z) = \begin{bmatrix} \nabla_{\vec{u}_k^i}^T h_\xi(Z) & \nabla_{\vec{u}_k^j}^T h_\xi(Z) \end{bmatrix}.$$

In order to derive the expressions for the covariance sub-matrices we compute the following quantities from Equations (3.16) and (3.18):

$$\nabla^T_{\vec{u}^i_k} h_p = \left(\sum_{m=1}^{n_{ij}} (P^{ij}_m)^{-1} \right)^{-1} (P^{ij}_k)^{-1}, \tag{A.20}$$

$$\nabla_{\vec{u}^j_k} h_p = - \left(\sum_{m=1}^{n_{ij}} (P^{ij}_m)^{-1} \right)^{-1} (P^{ij}_k)^{-1} \hat{R}_{ij}, \tag{A.21}$$

$$\nabla^T_{\vec{u}^i_k} h_\phi \simeq -\frac{1}{r_T} q_k J (P^{ij}_k)^{-1}, \tag{A.22}$$

$$\nabla^T_{\vec{u}^i_k} h_\phi \simeq -\frac{1}{r_T} q_k J (P^{ij}_k)^{-1} \hat{R}_{ij}, \tag{A.23}$$

with

$$\begin{aligned} P^{ij}_k &= Q^{ij}_k + \hat{R}_{ij} S^{ij}_k \hat{R}^T_{ij}, \\ q_k &= \hat{R}_{ij} \vec{u}^j_k, \\ r_T &= -\sum_{k=1}^{n_{ij}} q^T_k J (P^{ij}_k)^{-1} J q_k \ . \end{aligned}$$

In Equations (A.22) and (A.23) we employed the approximation made in Eq. (A.10). The interested reader is referred to [44] for the details of these derivations.

By substituting Equations (A.20) through (A.23) in Equation (A.19) the sub-matrices of the covariance matrix for the estimated motion vector $\hat{\lambda}^T = [\ \hat{p}^T_{ij} \ \ \hat{\phi}_{ij}\ ]$ in Eq. (A.18) can now be computed. The final expressions are given by Equations (3.17)–(3.22).

Appendix B

Implementation Details

B.1 Obstacle Constraints

A requirement for any useful motion planning algorithm is that the paths it generates are *correct*, i.e., the paths are contained wholly within the freespace. In other words, the algorithm should not generate paths that intersect obstacles. To maintain correctness, the optimization method must have some knowledge of all the obstacles that the robot knows about. We choose to incorporate the obstacles as constraints in the optimization process.

For simulation purposes, we assume that all obstacles are circles. This was chosen as a tradeoff between simplicity of implementation and accuracy. However, this assumption does not preclude complex situations. A union of circles can well approximate virtually any obstacle.

Since the path is specified as a number of waypoints, one constraint is that none of these waypoints lies inside any obstacle. Additionally, if we assume that the robot moves in straight lines between consecutive waypoints, there must be an additional constraint that prevents these line segments from intersecting the obstacles. Figure B.1 shows a simple example with one obstacle. The center of the obstacle is at location x_c with radius R. The robot steps from waypoint x_{i-1} to x_i, then on to x_{i+1}. The constraint is as follows: Each path segment cannot intersect the circle. Any numerical optimization scheme will require that the constraint be written as $c(x) < 0$. $c(x)$ is a column vector, where each row of c describes a single constraint equation as a function of x, the variables being optimized.

First consider a single line and a single circle. The circle is defined as above, and the

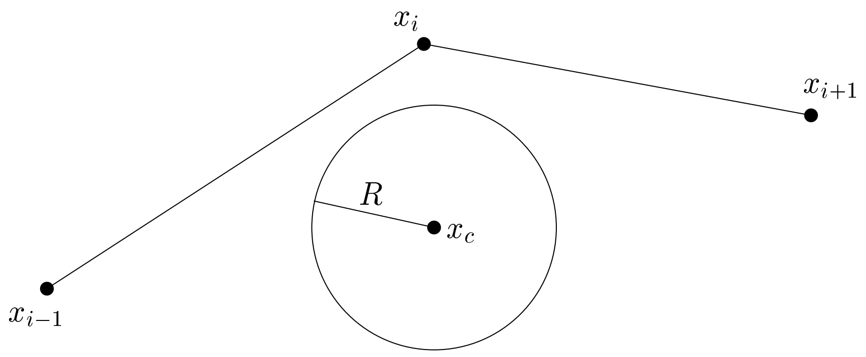

Figure B.1: A portion of the path where the robot passes by an obstacle.

line is defined by the points x_1 and x_2. The parametric equation for a line is

$$x = x_1 + t(x_1 - x_2), \tag{B.1}$$

where t is the parameter, and x is any point on the line. The point on the line that is closest to the circle is also closest to the center of the circle. Call this point closest to the circle x_d, and call the distance from the center of the circle to this point d_c. See Figure B.1 for a graphical explanation. The point x_d will lie on a line that passes through x_c and is

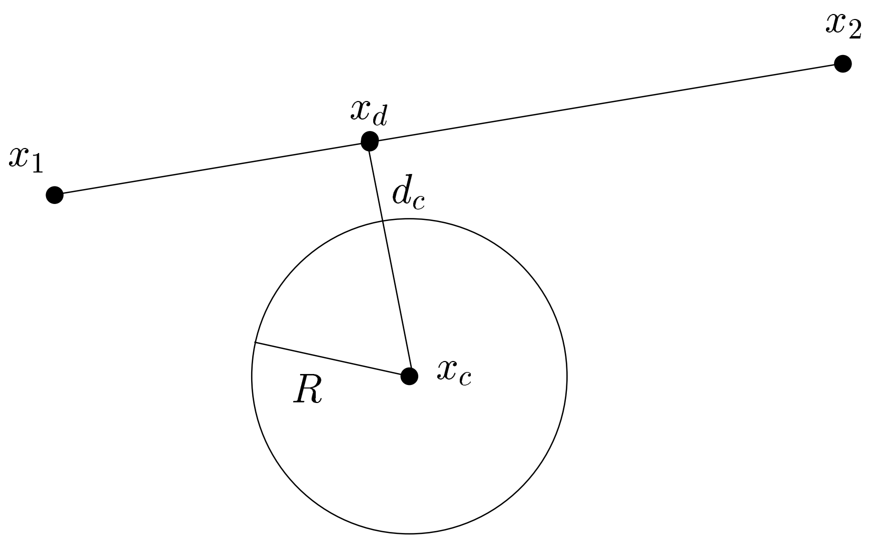

Figure B.2: Setup of a single line and circle.

perpendicular to the line defined by x_1 and x_2. We can write this as the dot product of these lines being equal to zero. To constrain x_d to the line defined by x_1 and x_2, we can use (B.1). This gives us three equations and three unknowns (the two components of the point x_d and the parameter t_d). These equations are

$$
\begin{aligned}
(x_2 - x_1) \cdot (x_c - x_d) &= 0, && \text{(B.2)}\\
x_1 + t_d(x_2 - x_1) &= x_d. && \text{(B.3)}
\end{aligned}
$$

Equation (B.3) is actually two separate equations, one for each component. It is not too difficult to solve these three equations for x_d and t_d. Note that we will end up with equations for x_d and t_d as functions of x_1, x_2, and x_c. This is exactly what we want, as x_1 and x_2 are the variables of the optimization problem we are trying to solve.

The equation for t_d is

$$
t_d = \frac{x_1(1)(x_1(1) - x_2(1)) + x_1(2)(x_1(2) - x_2(2)) + (-x_1(1) + x_2(1))x_c(1) + (-x_1(2) + x_2(2))x_c(2)}{(x_1(1) - x_2(1))^2 + (x_1(2) - x_2(2))^2}. \tag{B.4}
$$

Inspection of this equation leads to a slightly more compact form:

$$
t_d = \frac{x_1 \cdot (x_1 - x_2) + (x_2 - x_1) \cdot x_c}{\|x_1 - x_2\|^2}. \tag{B.5}
$$

Once t_d is known, it is trival to substitute it back into (B.3) and find x_d.

B.1.1 The Constraint Equations

Now that we have functions for the variables of interest, we can write down what the constraints themselves are. If $d_c > R$, then the line segment (in fact the entire line) does not intersect the circle. If on the other hand $d_c < R$, the line does intersect the circle, but it is still possible that the segment that we care about does not. Adding in the constraint that the segment endpoints must lie outside the circle, i.e.c

$$
\begin{aligned}
d(x_1, x_c) &> R, && \text{(B.6)}\\
d(x_2, x_c) &> R, && \text{(B.7)}
\end{aligned}
$$

where $d(\cdot, \cdot)$ is the Euclidean distance between arguments further reduces the cases where the segment intersects the circle, but not yet completely. If the parameter t_d lies between zero and one, then the closest point on the line is also on the segment of interest. So restricting t_d to be greater than one or less than zero will fully constrain the segment to be outside the circle.

In summary there are three separate cases, one of which must be satisfied for the line segment to lie outside the obstacle. The first is that the entire line lies outside the circle:

$$d_c > R. \tag{B.8}$$

The second and third cases are that the line intersects the circle, but both of the segment endpoints lie outside the circle and the closest point on the line to the circle center does not lie between the segment endpoints. There are two separate cases here, the first being

$$\begin{aligned} d(x_1, x_c) &> R, \\ d(x_2, x_c) &> R, \\ td &< 0. \end{aligned} \tag{B.9}$$

and the second case is

$$\begin{aligned} d(x_1, x_c) &> R, \\ d(x_2, x_c) &> R, \\ td &> 1. \end{aligned} \tag{B.10}$$

These sets of constraint equations will constrain the segment to lie outside the circle. But we only need one of the three sets to be true at any time. If (B.8) is true, then we do not need to check (B.9) and (B.10). In a logical sense, we want to OR the constraints together, and check that the outcome is true. So it will be necessary to write this group of seven equations as a single constraint.

Condensing this group of AND's and OR's into a single equation can be done with max and min operations. For example if we want to require that

$$a > 0 \text{ AND } b > 0, \tag{B.11}$$

then to write this as one constraint $c(x) > 0$, it becomes

$$\min(a, b) > 0. \tag{B.12}$$

If the constraint must be specified as $c(x) < 0$,

$$\max(a, b) < 0. \tag{B.13}$$

On the other hand if the constraint is

$$a > 0 \text{ OR } b > 0, \tag{B.14}$$

we can write it as

$$\max(a, b) > 0 \tag{B.15}$$

to express the constraint as $c(x) > 0$. To write our specific constraints as one equation, we start with

$$\begin{array}{rl} R - d_c < 0 & \text{OR} \\ max(R - d(x_1, x_c), R - d(x_2, x_c), t_d) < 0 & \text{OR} \\ max(R - d(x_1, x_c), R - d(x_2, x_c), 1 - t_d) < 0 & , \end{array} \tag{B.16}$$

Where the equations have been reorganized to be of the form $c(x) < 0$. Matlab's `fmincon` requires the non-linear inequality constraints to be written in this form. To finally combine them all together, we have

$$min(c_1, c_2, c_3) < 0, \tag{B.17}$$

where

$$c_1 = R - d_c, \tag{B.18}$$

$$c_2 = max(R - d(x_1, x_c), R - d(x_2, x_c), t_d), \tag{B.19}$$

$$c_3 = max(R - d(x_1, x_c), R - d(x_2, x_c), 1 - t_d). \tag{B.20}$$

Appendix C

Shortest Path Properties

Let $\alpha(t)$ be a smooth path in the free configuration space $\mathcal{F}$. Our goal in this section is to establish the following necessary condition for the shortest path in a freespace containing objects with smooth boundaries:

Necessary Condition: *Let $\mathcal{F} \subset \mathbb{R}^k$ be bounded by polygonal c-obstacles. Let $\alpha(t)$ be the shortest path in $\mathcal{F}$ connecting q_{init} to q_{goal}. Let the path be parametrized such that $\|\dot{\alpha}(t)\| = 1$ for all t.*

1. *If $\alpha(t)$ passes at $t = 0$ through an interior point of the freespace, $\alpha(t)$ must be a* **straight line** *in a neighborhood about $t = 0$.*

2. *If $\alpha(t)$ passes at $t = 0$ through a vertex d_0 of a c-obstacle $\mathcal{CB}_i$, $\alpha(0) = d_0$, then the path's acceleration, $\ddot{\alpha}(0)$, must be* **antipodal** *to the generalized gradient of the distance function $dst(d, \mathcal{CB}_i)$ at d_0,*

$$\ddot{\alpha}(0) = -\lambda\xi \quad \textit{for some } \xi \in \partial dst(d_0, \mathcal{CB}_i),$$

where $\lambda \geq 0$ is a non-negative scalar, $\partial f(\cdot)$ is the generalized gradient of $f(\cdot)$, and $dst(x, S)$ is the distance between the point x and the set S, defined as

$$d(x, S) = \min_{q \in S} \|x - q\|. \tag{C.1}$$

Remark: It can be shown that $\text{dst}(d, \mathcal{S})$ is continuous and Lipschitz. Consequently it is differentiable almost everywhere. In the interior of $\mathcal{S}$ its gradient vector is always zero. At points d outside $\mathcal{S}$ that have a unique closest point d^* in $\mathcal{S}$ its gradient is $\nabla \text{dst}(d, \mathcal{S}) = (d - d^*)/\|d - d^*\|$. At points on the boundary of $\mathcal{S}$ its generalized gradient is the convex combination of zero and the outward normals to the smooth patches comprising the boundary in a small neighborhood about this point [12].

The basic tool used by calculus of variations is the following variation of a path:

Definition 1. Let $\alpha(t) : [a, b] \to \mathcal{F} \subset \mathbb{R}^k$ be a smooth path. A **variation** of $\alpha(t)$ is a smooth map $\psi(t, s) : [a, b] \times (-\epsilon, \epsilon) \to \mathcal{F}$ such that $\psi(t, 0) = \alpha(t)$ ($\epsilon > 0$ is some fixed positive constant). It is **fixed endpoint variation** if $\psi(a, s) = \alpha(a)$ and $\psi(b, s) = \alpha(b)$ for all $s \in (-\epsilon, \epsilon)$.

Figure C.1 shows the image of a typical variation. It is a smooth (not necessarily homeomorphic!) image of the rectangle $[a, b] \times (-\epsilon, \epsilon) \subset \mathbb{R}^2$, whose backbone curve is $\alpha(t)$. For fixed s, the path parametrized by t:

$$\alpha_s(t) \equiv \psi(t, s) \quad (s \text{ held constant}) \tag{C.2}$$

is a smooth path that runs "parallel" to $\alpha(t)$. As s is varied, a collection of curves $\alpha_s(t)$ is generated. This family can be interpreted as a "curve of curves" in a suitably defined (∞-dimensional) manifold whose "points" are curves connecting $\alpha(a)$ to $\alpha(b)$. In this manifold $\beta(s) \equiv \alpha_s(t)$ is a smooth path passing through the point corresponding to the backbone curve $\alpha(t)$ at $s = 0$.

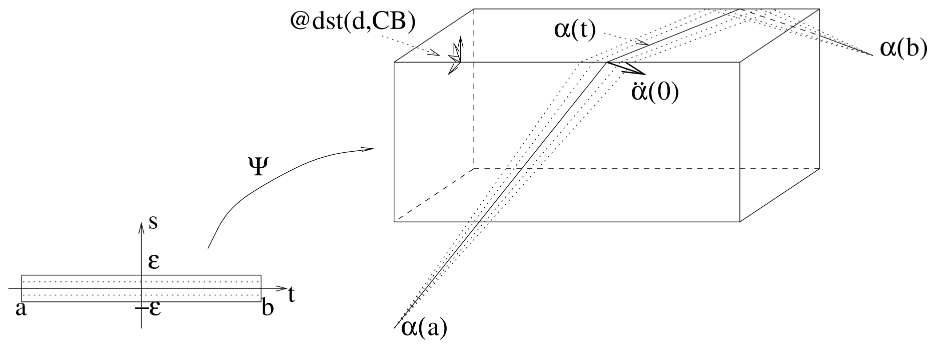

Figure C.1: A fixed endpoint variation.

The Jacobian matrix of $\psi(t, s)$ is the $k \times 2$ matrix $D\psi(t, s)$. We will need the following two *coordinate vector fields* along $\psi(t,s)$, which are exactly the columns of $D\psi(t,s)$:

$$E_1(t,s) \equiv D\psi(t,s)\begin{bmatrix}1\\0\end{bmatrix} = \frac{\partial\psi(t,s)}{\partial t}, \tag{C.3}$$

$$E_2(t,s) \equiv D\psi(t,s)\begin{bmatrix}0\\1\end{bmatrix} = \frac{\partial\psi(t,s)}{\partial s}. \tag{C.4}$$

The vector field $E_1(t,s)$ is tangent to the curve $\alpha_s(t)$ (s constant). In particular, at $s=0$,

$$E_1(t,0) = \dot\alpha(t) \tag{C.5}$$

is the tangent along $\alpha(t)$. The other vector field, $E_2(t,s)$, is tangent to curves resulting from tracing $\psi(t,s)$ for fixed t. They run in a direction transversal to the backbone curve $\alpha(t)$. In particular, at $s=0$, $E_2(t,s)$ gets a special name:

$$X(t) = E_2(t,0), \tag{C.6}$$

and is called the *variation vector field.* Note that for a fixed endpoint variation $X(a) = X(b) = 0$.

In general the formula for the length of a smooth path $\alpha_s(t) : [a,b] \to \mathbb{R}^k$ (using the Euclidean norm) is given by

$$l(\alpha_s) = \int_a^b \|\dot\alpha_s(t)\| dt \tag{C.7}$$

$$= \int_a^b \|E_1(t,s)\| dt. \tag{C.8}$$

When evaluted on a family of curves, smoothly parametrized by s, l becomes a smooth real-valued function of s, assigning to every s in $(-\epsilon,\epsilon)$ the length of $\alpha_s(t)$.

The basic idea of the calculus of variation is as follows: If $\alpha(t)$ is the shortest path in the family $\alpha_s(t)$, $s=0$ must be a stationary point of $l(\alpha_s)$. That is, the derivative $\frac{d}{ds}\big|_{s=0} l(\alpha_s)$ must be zero. Let us now compute the derivative $\frac{d}{ds}l(\alpha_s)$ at $s=0$. First we have

$$\left.\frac{d}{ds}\right|_{s=0} l(\alpha_s) = \int_a^b \left.\frac{d}{ds}\right|_{s=0} \|E_1(t,s)\| dt. \tag{C.9}$$

Using the chain rule,

$$\left.\frac{d}{ds}\right|_{s=0} \|E_1(t,s)\| = \left.\frac{1}{\|E_1(t,s)\|}\right|_{s=0} E_1(t,s) \cdot \left.\frac{d}{ds}\right|_{s=0} E_1(t,s). \tag{C.10}$$

The coefficent is unity since $\|E_1(t,0)\| = 1$. Now observe that

$$\begin{aligned}\left.\frac{d}{ds}\right|_{s=0} E_1(t,s) &= \left.\frac{d}{ds}\right|_{s=0} \frac{\partial}{\partial t}\psi(t,s) && \text{(C.11)}\\ &= \frac{\partial}{\partial t} \left.\frac{\partial}{\partial s}\right|_{s=0} \psi(t,s) && \text{(C.12)}\\ &= \frac{d}{dt} X(t). && \text{(C.13)}\end{aligned}$$

Substituting $\dot{X}(t)$ for $\left.\frac{d}{ds}\right|_{s=0} E_1(t,s)$ and $\dot{\alpha}(t)$ for $E_1(t,0)$ in the integral gives

$$\begin{aligned}\left.\frac{d}{ds}\right|_{s=0} l(\alpha_s) &= \int_a^b \dot{X}(t) \cdot \dot{\alpha}(t) dt && \text{(C.14)}\\ &= \int_a^b \left(\frac{d}{dt} (X \cdot \dot{\alpha}) - X \cdot \ddot{\alpha} \right) dt && \text{(C.15)}\\ &= \left.(X \cdot \dot{\alpha})\right|_{t=b} - \left.(X \cdot \dot{\alpha})\right|_{t=a} - \int_a^b (X \cdot \ddot{\alpha})\, dt. && \text{(C.16)}\end{aligned}$$

The first two terms vanish for fixed endpoint variation. The resulting formula,

$$\left.\frac{d}{ds}\right|_{s=0} l(\alpha_s) = -\int_a^b (X \cdot \ddot{\alpha})\, dt, \tag{C.17}$$

is called the *first variation of the path-length integral.*

The necessary condition

$$\int_a^b (X \cdot \ddot{\alpha})\, dt = 0, \tag{C.18}$$

it can be shown, is satisfied by the shortest path $\alpha(t)$ only if the integrand $X(t) \cdot \ddot{\alpha}(t)$ is zero for all t. Let us now sketch how this implies the necessary condition stated at the beginning.

Let $\alpha(t)$ be the shortest path. First let us see why $\ddot{\alpha}(0) = 0$ (i.e., a straight line) about every interior point. Let $\alpha(0)$ be an interior point of $\mathcal{F}$. If $\ddot{\alpha}(0) \neq 0$, it is possible to construct a variation ψ of the form

$$\psi(t,s) = \alpha(t) - s\sigma(t)\ddot{\alpha}(0), \tag{C.19}$$

where $\sigma(t)$ is a smooth real-valued function that is exactly unity at $t = 0$, and decays smoothly to zero away from $t = 0$ (a *bump function*). Note that at $t = 0$ we have that

$$X(0) \cdot \ddot{\alpha}(0) = -\|\ddot{\alpha}(0)\|^2 < 0. \tag{C.20}$$

This means that $\alpha(t)$ has a neighboring curve, $\alpha_s(t)$ for some small fixed $s > 0$, whose length is smaller than the length of $\alpha(t)$. The same idea applies when $\alpha(0) = d_0$ is a boundary point. If $\ddot{\alpha}(0)$ is *not* antipodal to the generalized gradient $\partial\mathsf{dst}(d_0, \mathcal{S})$, it is possible to construct a variation $\psi(t, s)$ of $\alpha(t)$, whose variation vector field $X(t)$ points into the halfspace of directions pointing away from $\ddot{\alpha}(0)$, so that

$$X(0) \cdot \ddot{\alpha}(0) < 0. \tag{C.21}$$

This implies the existence of a neighboring curve in $\mathcal{F}$ of shorter length.

Appendix D

Bounded Uncertainty Freespace

What is the set of points from which the robot can reach the goal from with $U_g < \epsilon$? The following sections show that this set is bounded. The exact shape of the boundary is not determined, but a bound is placed on the extremes of this region.

D.1 Goal Only

First consider the case where there are no landmarks. The robot has to rely on its odometry alone for localization. In this case the robot's uncertainty will continue to increase as it moves.

For the simple point robot model that is assumed here, and under the assumption that the variance of the robot velocity is proportional to the velocity, i.e., $\sigma_v = \alpha V$, it is known that

$$P_{RR}(k+1/k) = P_{RR}(k/k) + \alpha^2 \begin{bmatrix} (x_R(k+1) - x_R(k))^2 & 0 \\ 0 & (y_R(k+1) - y_R(k))^2 \end{bmatrix}. \tag{D.1}$$

In the best case the robot will start out with a perfect estimate of its pose, i.e., $P_{RR}(0) = 0_{2\times 2}$. If the initial covariance is non-zero, then the bounds will be "smaller." An explicit limit can be placed on how large the robot's covariance can grow in this case before it terminates with failure. Requiring the robot to get to within ϵ of the goal translates to a limit on the maximum size of the covariance matrix. Further assuming that the robot's true position will always be within the 3σ ellipse of the position estimate error covariance,

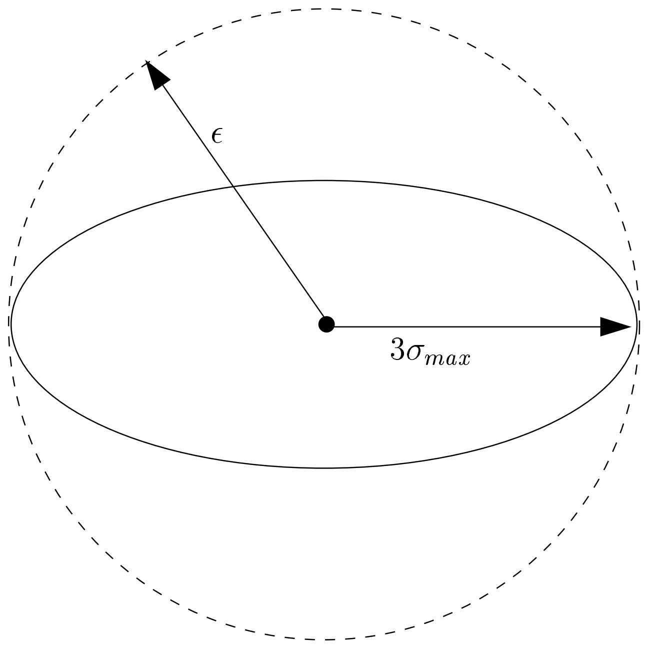

Figure D.1: Maximum size of covariance ellipse to fit inside circle with radius of ϵ.

then an upper limit on the maximum variance exists:

$$3\sigma_{max} < \epsilon. \tag{D.2}$$

Graphically, the inequality looks like Figure D.1. The 3σ covariance ellipse has to fit inside the circle of radius ϵ.

If the robot takes one step from its starting location to the goal, its covariance after that step will be

$$P_{RR}(1/0) = \alpha^2 \begin{bmatrix} (g_x - x_R(0))^2 & 0 \\ 0 & (g_y - y_R(0))^2 \end{bmatrix}, \tag{D.3}$$

where $g = \begin{bmatrix} g_x & g_y \end{bmatrix}^T$ is the goal position. The length of the axes of the covariance ellipse are proportional to the eigenvalues of the covariance matrix, namely,

$$\sigma_i = \sqrt{\lambda_i}, \tag{D.4}$$

where σ_i is the variance along the i^{th} direction, and λ_i is the corresponding eigenvalue. In this case the covariance matrix is diagonal, so the eigenvalues are just the diagonal elements

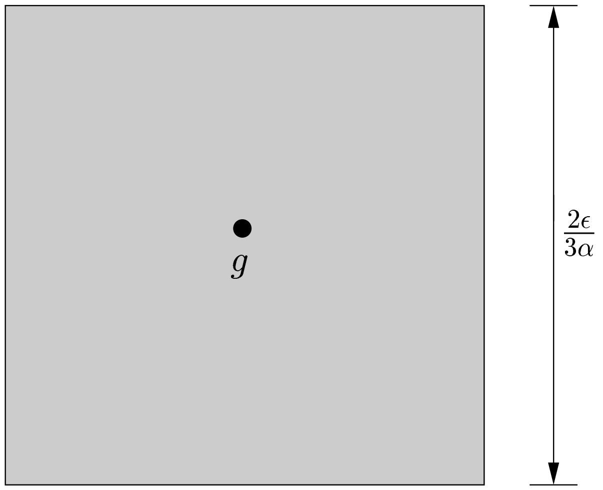

Figure D.2: Set of points from which the goal can be reached with $\sigma_{max} < \epsilon$. The set is the darkened box.

themselves. Therefore,

$$\sigma_{max} = \alpha \max\left(|g_x - x_R(0)|, |g_y - y_R(0)|\right). \tag{D.5}$$

Equation (D.2) can be re-written as

$$3\alpha \max\left(|g_x - x_R(0)|, |g_y - y_R(0)|\right) < \epsilon, \tag{D.6}$$

$$\max\left(|g_x - x_R(0)|, |g_y - y_R(0)|\right) < \frac{\epsilon}{3\alpha}. \tag{D.7}$$

These constraint equations can be interpreted as requiring $\begin{bmatrix} x_R(0) & y_R(0) \end{bmatrix}$ to lie inside a square centered at g with side length $\frac{2\epsilon}{3\sigma}$. Figure D.2 shows this graphically.

D.2 Goal and One Landmark

Now consider the case where there is one landmark in the environment. Extension of these results to the case of multiple landmarks can be found in a later section. Again, this is only placing bounds on the set of points the robot could reach the goal from, not explicitly finding the set.

This problem can be broken down into finding subsets of the entire set. First, one can find the set of points from which the robot can reach the goal by relying on odometry alone. This set was found in Section D.1. The set of points from which the robot can start

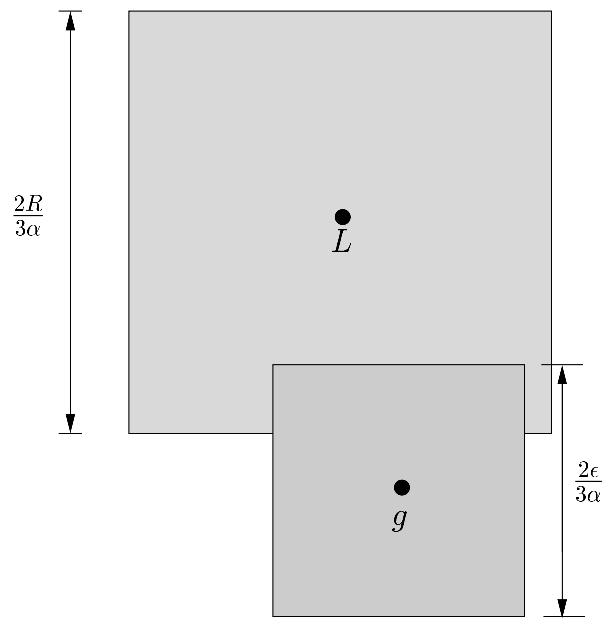

Figure D.3: A bound on the set of points from which the goal could be reached with $\sigma_{max} < \epsilon$.

and reach the landmark reliably is also needed. If the robot starts too far away from the landmark (or the goal), it will become "lost" before it even reaches the landmark and has a chance to localize itself. The landmark can be thought of as an intermediate goal. This is analagous to the case of reaching the goal, but instead of requiring that the robot get within ϵ, the requirement is that the robot get within R of the landmark. R is the robot's sensor range, and it must get at least that close to the landmark to be able to use it for localization. The union of these two sets contains the set of all possible points that the robot could start and reach the goal from. If the robot starts anywhere outside this union, it will not be able to reach either the goal or the landmark before its uncertainty exceeds a level where it can no longer guarantee that it will find the goal or the landmark.

To find the set of points from which the robot can reach the landmark with uncertainty $< R$, ϵ is replaced with R and g is replaced with L (the landmark position) in Equation (D.7). This analysis yields the result

$$\max\left(|L_x - x_R(0)|, |L_y - y_R(0)|\right) < \frac{R}{3\alpha}. \tag{D.8}$$

Figure D.3 shows the union of these two sets.

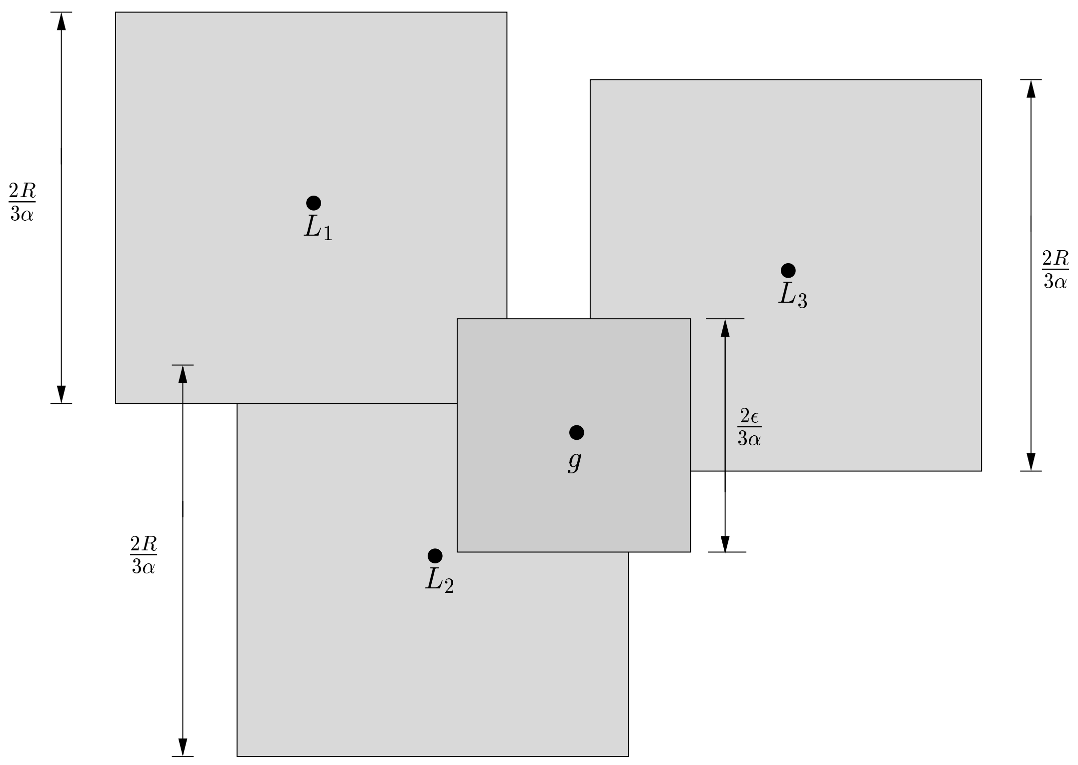

Figure D.4: Union of the multiple sets.

D.3 Goal and Multiple Landmarks

The case of multiple landmarks is a straightforward extension of the single-landmark case. The union of all of the sets defined by each landmark and the set defined by the goal is taken. If the robot starts anywhere outside this union, then it cannot reach the goal or a landmark without exceeding an uncertainty threshold that guarantees that it will be able to see the landmark (goal) when it thinks it has arrived there. Figure D.4 shows an example of this set with three landmarks. Note that the subsets defined by an individual landmark need not be the same size. If for some reason the robot can sense one landmark at a different range than another, then the boxes could be different sizes. Again, this set is *not* the set of points that the goal can be reached from with $U_g < \epsilon$, but it does contain it.

Bibliography

[1] M.D. Adams. Lidar design, use, and calibration concepts for correct environmental detection. *IEEE Trans. Robotics and Automation*, 16(6):753–761, Dec. 2000. 3.1, 5.3

[2] M.D. Adams and P.J. Probert. The interpretation of phase and intensity data from AMCW light detection sensor for reliable ranging. *Int. J. of Robotics Research*, 15(5):441–458, Oct. 1996. 3.1, 3.2.1, 3.3.1, 3.3.3

[3] M.C. Amann, T. Bosch, M. Lescure, R. Myllylä, and M. Rioux. Laser ranging: a critical review of usual techniques for distance measurement. *Opt. Eng.*, 40(1):10–19, Jan. 2001. 3.1, 3.3.1

[4] D. Ballard. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981. 3.5

[5] O. Bengtsson and A.J. Baerveldt. Localization in changing environments— estimation of covariance matrix for th IDC algorithm. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1931–7, Maui, Hawaii, Oct. 2001. 3.1

[6] S.S. Blackman. Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine*, 19(Issue 1, Part 2):5–18, Jan. 2004. 2.2.2.1

[7] D.M. Blei and L.P. Kaelbling. Shortest paths in a dynamic uncertain domain. In *Proc. of the IJCAI Workshop on Adaptive Spatial Representations of Dynamic Environments*, 1999. 1.2

[8] A. Briggs, C. Detweiler, D. Scharstein, and A. Vandenberg-Rodes. Expected shortest paths for landmark-based robot navigation. In *Proc. of the Fifth Int. Workshop on the Algorithmic Foundations of Robotics*, December 2002. 1.2

[9] A. Briggs, C. Detweiler, D. Scharstein, and A. Vandenberg-Rodes. Expected shortest paths for landmark-based robot navigation. *Intl J. of Robotics Research*, 23(7-8):717–728, July-August 2004. 1.2

[10] H. Choset and J. Burdick. Sensor based planning, part I: The generalized Voronoi graph. In *Proc. IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 1649 – 1655, May 1995. 5.4

[11] H. Choset and J.W. Burdick. Sensor based exploration: the hierarchical generalized Voronoi graph. *Int. J. of Robotics Research*, 19(2):96–125, Feb 2000. 1.1, 1.2

[12] F. H. Clarke. *Optimization and nonsmooth analysis.* Wiley, 1983. C

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* MIT Press, 2001. 1.2

[14] I.J. Cox. Blanche—an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Trans. on Robotics and Automation*, 2:193–204, 1991. 1.2, 3.1, 3.1

[15] R.O. Duda and P.E. Hart. Use of hough transform to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972. 3.5

[16] D. Fox, W. Burgard, and S. Thrun. Active markov localization for mobile robots. *Robotics and Autonomous Systems*, 25:195–207, 1998. 1.2

[17] J. Gonzalez and R. Gutierrez. Mobile robot motion estimation from a range scan sequence. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1034–9, New York, NY, Apr. 20-25 1997. 1.2, 3.1, 3.1

[18] A.E. Johnson and A. Miguel San Martin. Motion estimation from laser ranging for autonomous comet landing. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1788–1795, San Francisco, CA, Apr. 24–28 2000. 3.7

[19] R.E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960. 1.2

[20] I. Kamon, E. Rimon, and E. Rivlin. Tangentbug: A range sensor based navigation algorithm. *Int. J. of Robotics Research*, 17(9):934–953, Sept. 1998. 1.1, 1.2, 2.1

[21] I. Kamon, E. Rimon, and E. Rivlin. Range-sensor-based navigation in three-dimensional polyhedral environments. *Int.l J.l of Robotics Research*, 20(1):6–25, January 2001. 4.5

[22] T. Kirubarajan and Y. Bar-Shalom. Probabilistic data association techniques for target tracking in clutter. *Proc. of the IEEE*, 92(3):536–557, Mar 2004. 2.2.2.1

[23] A. Lambert and T. Fraichard. Landmark-based safe path planning for car-like robots. In *Proc. IEEE International Conference on Robotics and Automation*, 2000. 1.2

[24] R.E. Larson and J.L. Casti. *Principles of Dynamic Programming.* 1978. 5.3

[25] J.C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991. 4.1

[26] S.L. Laubach. *Theory and Experiments in Autonomous Sensor-Based Motion Planning with Applications for Flight Planetary Microrovers.* PhD thesis, California Institute of Technology, 1999. 1.2

[27] S.M. LaValle and S.A. Hutchinson. An objective-based framework for motion planning under sensing and control uncertainties. *Int. J. Robotics Research*, 17(1):19–42, Jan. 1998. 1.2

[28] S.M. LaValle and R. Sharma. On motion planning in changing, partial predictable environments. *Int. J. Robotics Research*, 16(6):775–805, December 1997. 1.2

[29] A. Lazanas and J.C. Latombe. Landmark based robot navigation. *Algorithmica*, 13(5):472–501, May 1995. 1.2

[30] A. Lazanas and J.C. Latombe. Motion planning with uncertainty, a landmark approach. *Artificial Intelligence*, 76(1-2):287–317, July 1995. 1.2

[31] J. Leonard and H. Durrant-Whyte. Mobile robot localization by tracking geometry beacons. *IEEE Trans. on Robotics and Automation*, 7(3):376–382, June 1991. 1.1, 1.2

[32] Y. Liu and M.A. Rodriques. Accurate registration of structured data using two overlapping range images. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 2519–24, Washington D.C., May 11–15 2002. 3.7

[33] A. Logothetis. *EM Algorithms for State and Parameter Estimation of Stochastic Dynamical Systems.* PhD thesis, University of Melbourne, Victoria, Australia, August 1997. 1.2

[34] A. Logothetis, A. Isaksson, and R. J. Evans. An information theoretic approach to observer path design for bearings-only tracking. In *Proc. of the 36th Conference on Decision and Control*, pages 3132–3137, San Diego, CA, USA, December 1997. CDC. 1.2

[35] F. Lorussi, A. Marigo, and A. Bicchi. Optimal exploratory paths for a mobile rover. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2001. 1.2

[36] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997. 1.2, 3.1, 3.2.2, 3.6.1

[37] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2D range scans. *J. of Intelligent and Robotic Systems*, 20:249–275, 1997. 1.2, 2.2.2.1, 3.1, 3.1, 3.2.2, 3.4, 3.6

[38] V.J. Lumelsky and A.A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987. 1.1, 1.2

[39] P. S. Maybeck. *Stochastic Models, Estimation, and Control.* Academic Press, Inc., 1979. 2.2.1

[40] P. Moutarlier and R. Chatila. Stochastic multisensory data fusion for mobile robot location and environment modelling. In *Proc. International Symposium on Robotics Research*, pages 85–94, Tokyo, August 1989. 1.2

[41] S. Pfister and J. W. Burdick. Weighted line fitting algorithms for mobile robot map building and efficient data representation. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Taipei, Taiwan, Sept. 2003. 3.3.2, 3.5

[42] S.T. Pfister, K.L. Kriechbaum, S.I. Roumeliotis, and J.W. Burdick. Weighted range sensor matching algorithms for mobile robot displacement estimation. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Washington, D.C., May 2002. 1.1, 2.2.2.1, 3.6.4

[43] S. Rezaei, J. Guivant, J. Nieto, and E. M. Nebot. Simultaneous information and global motion analysis ("SIGMA") for car-like robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2004. 1.2

[44] S.I. Roumeliotis. Dense range feature matching: weighted rotational displacement estimation. Technical report, C.I.T., 2001. http://robotics.caltech.edu/~stergios/tech_reports/tr_wlsm_orientation.pdf. A.3

[45] S.I. Roumeliotis and J. Burdick. Stochastic cloning: A generalized framework for processing relative state measurements. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1788–1795, Washington D.C., May 11-15 2002. 3.1, 3.7

[46] N. Roy, W. Burgard, D. Fox, and S. Thrun. Coastal navigation—mobile robot navigation with uncertainty in dynamic environments. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 1999. 1.2, 6.2

[47] N. Roy and S. Thrun. Coastal navigation with mobile robots. In *Proc. of Conference on Neural Information Processing Systems*, 1999. 1.2, 6.2

[48] P.W. Smith, N. Nandhakumar, and C.H. Chien. Object motion and structure recovery for robotic vision using scanning laser range sensors. *IEEE Trans. on Robotics and Automation*, 13(1):74–80, Feb. 1997. 3.7

[49] R.C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *Int. J. of Robotics Research*, 5(4):56–68, 1986. 1.2

[50] R.C. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. *Autonomous Robot Vehicles*, pages 167–193, 1990. 1.1, 1.2

[51] R.F. Stengel. *Optimal Control and Estimation.* Dover, 1994. 5.3

[52] S. Takahashi and B.K. Ghosh. Motion and shape identification with vision and range. *IEEE Trans. on Robotics and Automation*, 47(8):1392–6, Aug. 2002. 3.7

[53] S. Thrun. Bayesian landmark learning for mobile robot localization. *Machine Learning*, 3(1):41–76, Oct. 1998. 1.3

[54] S. Thrun. Particle filters in robotics. In *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, 2002. 1.2

[55] S. Thrun, D. Fox, and W. Burgard. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, 31:29–53, 1998. 1.1

[56] N. Trawny and T. Barfoot. Optimized motion strategies for cooperative localization of mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2004. 1.2

[57] A.C. Victorino, P. Rives, and J. Borrelly. A relative motion estimation by combining laser measurement and sensor based control. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 3924–9, Washington D.C., May 11–15 2002. 3.1, 3.2.3

[58] C. Ye and J. Borenstein. Characterization of a 2-d laser scanner for mobile robot obstacle negotiation. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 2512–2518, Washington D.C., May 11–15 2002. 3.6.4