Topological Mapping for Limited Sensing Mobile Robots Using the Probabilistic Gap Navigation Tree

by

Valerie Gordeski

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2008

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2008

Certified by .....
Nicholas Roy
Assistant Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Topological Mapping for Limited Sensing Mobile Robots Using the Probabilistic Gap Navigation Tree

by

Valerie Gordeski

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2008, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis proposes a novel structure for robotic navigation with minimal sensing abilities called the Probabilistic Gap Navigation Tree (PGNT). In this navigation approach, we create a topological map of the environment based on a previously created Gap Navigation Tree (GNT) [40]. The "gap" in the gap navigation algorithm represents a discontinuity in the robotic field of vision. The robot is able to use the gaps to represent its world as a tree structure (GNT), in which each vertex corresponds to a gap. Ideally, the robot navigates in the world by following the tree branches to its desired goal. However, due to the sensor uncertainty, the robot may detect discontinuities when there are none present, and vice versa. The Probabilistic Gap Navigation Tree compensates for the measurement noise by sampling from a distribution of the gap navigation trees to obtain the most likely tree given the sensor measurements, similar to the particle filtering algorithm used in Monte Carlo localization. Therefore, the PGNT allows navigation in an unknown environment using a realistic rangefinder, as opposed to the ideal sensor model assumed previously. We demonstrate the ability to build a PGNT in a simulated environment.

Thesis Supervisor: Nicholas Roy
Title: Assistant Professor of Aeronautics and Astronautics

Acknowledgments

There are many people that I would like to thank for making it possible for me to make it this far.

First, I would like to thank my thesis advisor, Nick Roy, for his patience and support during the past two years. Coming to MIT would not be possible without my manager, Kenneth Roberts, who encouraging me to apply for the Raytheons Advanced Studies program that provided financial support for my studies here.

I would like to thank all of the members of the Robust Robotics Group for their support and guidance. I thank RJ He, Tomas Kollar, Sam Prentice and Abe Bachrach for their help with my research problems and software issues. I especially thank Matt Walter for providing me with advice and tremendous help in finishing my thesis.

I owe a big thanks to the EECS undergraduate office: Anne Hunter, Vera Sayzew and Linda Sullivan for their incredible support and understanding during the completion of my thesis.

Lastly, I thank my family and friends for sticking with me through sleepless nights. My mom and my grandmother sacrificed everything to give me the opportunity to study in the United States, Georg Reichstein was my constant cheerleader who believed in me when I did not believe in myself, and many others offered their encouragement and support. I am eternally in their debt.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Autonomous robots have many useful applications, including search and rescue missions, surveillance operations, and deep-sea or extraterrestrial exploration, among others. In order to perform these tasks, whether to collect data from the surface of Mars or to locate a hostage inside a remote building, the robot needs to have an accurate representation of its surrounding world and an efficient navigation scheme within this environment. This thesis proposes a mapping approach for a robot exploring an indoor environment with limited onboard sensing. We create a minimal representation of an indoor environment that the robot constructs in real-time using probabilistic techniques commonly used for autonomous navigation. Section 1.1 of this chapter introduces the problem of autonomous mapping with limited sensing and motivates the utility of our representation. Section 1.3 describes the key innovation to mapping presented in this thesis, and section 1.4 outlines the roadmap for the rest of this document.

1.1 Problems in Autonomous Mapping

One of the fundamental problems within autonomous navigation is the inherent uncertainty in the robot's position, which increases as it moves through the environment. In the absence of external global position devices (such as GPS), the robot may have to rely upon dead-reckoning and internal sensors, a process which results in accumu-

lation of error over time. A map of the environment that is available to the robot *a priori* can be used with environmental sensing to provide absolute position fixes and reduce the error from dead-reckoning. In most applications, a map of the environment is not readily available, so the robot has to construct one using its onboard sensors and computational resources, either during exploration, or later in post-processing. The kinds of maps created by a robot can vary from a topological map, which is a crude representation that defines connectivity between different places in an environment [2, 6, 19, 31], to a highly dense (metric) representation that can model walls as polyhedrons, as finely discretized segments (feature based representation), or as occupancy grids (gridmap representation) ([7, 27, 37] and many others).

Metric approaches create detailed environmental representations that allow the robot to disambiguate its position within a global coordinate frame. For example, figure 1-1 shows a detailed three dimensional map of a corridor based on scan data obtained by a robot using two SICK laser rangefinders [37]. The metric approaches can often capture the detailed structure of the environment, and can create high fidelity models which allow to disambiguate structure and aid in closing the loop in large cyclic environments. The density of the representation can capture various landmarks, enabling the robot to easily recognize a place that it has visited previously. The fine resolution of the metric maps can also be their weakness, creating enormous complexity. Some metric-based approaches require a very large number of sensor readings to construct a map and to reliably update robot's position [35].

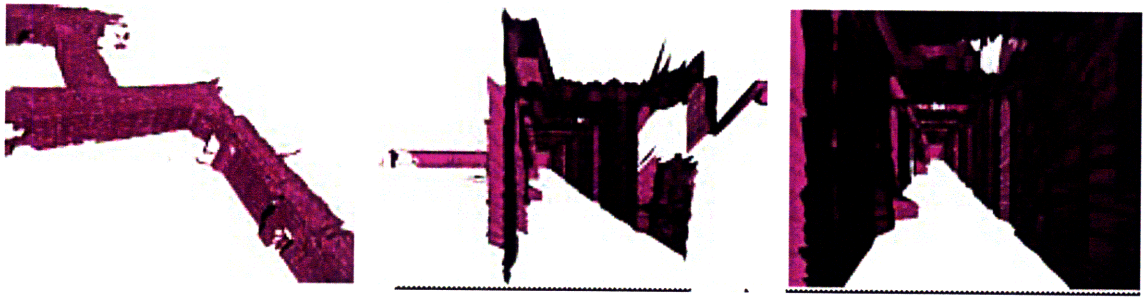


Figure 1-1: A three dimensional map of a corridor, which was originally obtained using scan matching dense data from two SICK laser rangefinders. Obtained from Thrun et al [37].

The desire for high resolution maps has led to the development of mapping techniques that rely on everything from expensive, high precision laser rangefinders [11], sonar arrays [22], GPS, monocular and stereovision cameras [5] and so on. However, these sensors are subject to errors, often referred to as measurement noise. Global positioning using GPS is not always available for error correction, as the signal can be lost due to interference in densely populated urban environments or disappear all together. Dead-reckoning causes the accumulated error to occur in the map. [36].

In contrast to the metric maps, many topological representations are naturally compact. Topological maps model the coarse structure of the world using connected graphs, which can be traced back to work by Kuipers [18]. The nodes of the graph denote locations of interest, while the edges between the nodes signify a direct path between them. Figure 1-2 shows step by step how an environment can be decomposed into a topological graph. The compactness of the graph representation offers many advantages: easier robotic path planning, more efficient storage, and greater robustness to errors in internal control since there is usually no need for a complete knowledge of the robot's position [35].

For applications that do not require a detailed world representation, topological maps are a natural choice. For example, it is not necessary to have the full 3D map of figure 1-1 for a robot to safely move from one room in the corridor to another. Autonomous robots have various applications that need a robust real-time path planning capability paired with a cheap implementation platform that would use as few and as cheap sensors as possible while consuming minimal power and computation time. For this reason, this thesis focuses on robotic navigation with limited sensing and computational requirements - using just a range sensor and a simple topological map. The limited sensing problem is an active area of research as witnessed by several techniques from Rao, Dudek and Whitesides [30], Beevers [3], Tovar et al [40, 41], Landa et al [20, 21] and others. Chapter 3 discusses in detail the previous works associated with limited sensing, placing a particular emphasis on the work of Tovar, Guilamo, Murrieta-Cid and LaValle, on which this thesis is primarily based.

Tovar et al [40, 41] previously described environment representation in terms of a

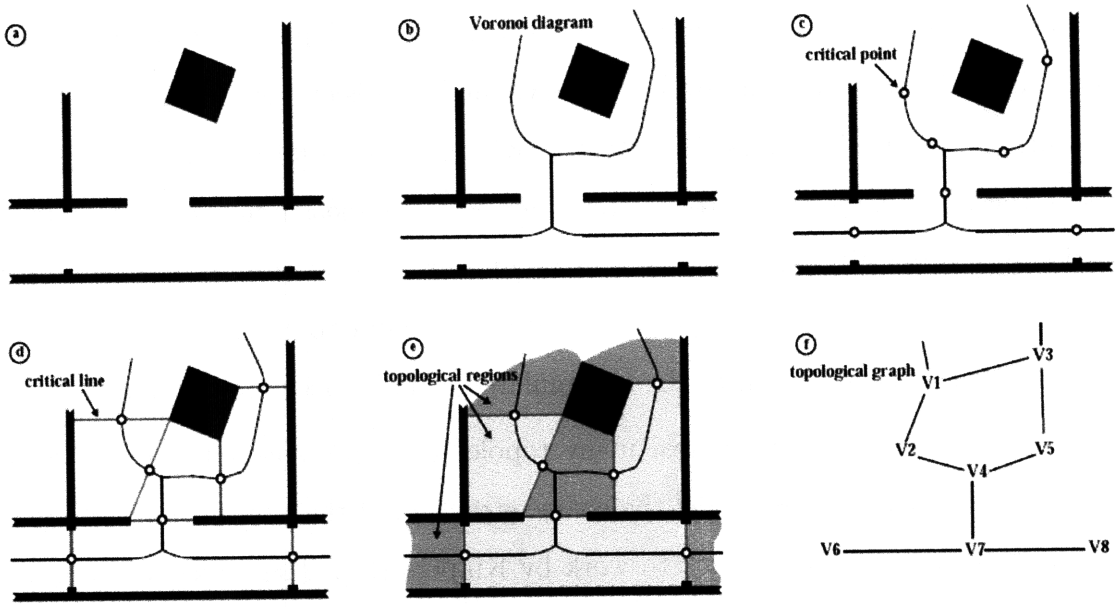


Figure 1-2: These images represent how a metric map can be decomposed into topological representations. (a) Metric map, (b) Voronoi diagram, (c) critical points (locations of interest), (d) critical lines (connecting locations of interest), (e) topological regions, and (f) the topological graph. Adapted from Thrun [35]

rooted tree, called the Gap Navigation Tree (GNT). They assume that a robot is a point object in a simply connected planar environment and that it is equipped with an infinite range edge detector that is able to extract discontinuities in its field of view. They named these discontinuities "gaps". As the robot moves around, it is able to track its position by keeping track of changes in its perception of the gaps, called critical events. By storing these gaps in a systematic way as leaves in a tree structure during robotic exploration, the robot is able to encode a globally optimal (in a Euclidean sense) path through this tree to any point of interest (see Chapter 3 for further details). Figure 1-3 shows a simple environment and a GNT that represents the environment at two different robot locations.

Outside of an ideal world, a robot has a fixed radius and its sensors are subject to noise and range limitations. Additionally, the GNT algorithm assumes that the range sensor is infinite, which makes its implementation on a real testbed very difficult. The GNT algorithm does not address what happens when there are no discontinuities to detect, or when there is an error in the sensor data that suggests discontinuities

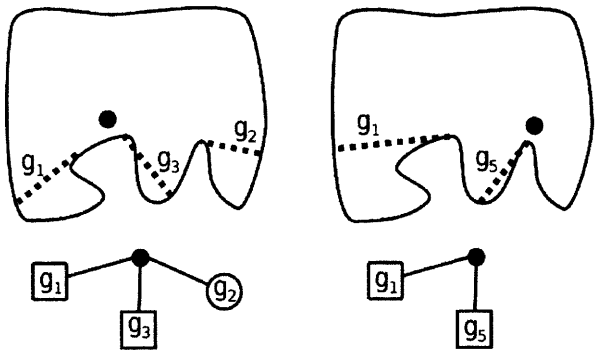


Figure 1-3: This image shows a simple map and two topological representations using the Gap Navigation Tree. The black dot in the middle represents the robot position. The dashed lines indicate borders between visible and hidden regions from the robot's perspective. Adapted from [41], courtesy of Benjamin Tovar.

where none are present. It also assumes that the robot moves slowly enough that only one critical event happens with each edge detection, which is not always a valid assumption.

1.2 Thesis Statement

Many mapping algorithms rely on probabilistic methods to cope with sensor uncertainty (for example, [17, 34]). The main problem of the Gap Navigation algorithm lies in its automatic encoding of all (even false) visibility events into its environment representation, because it views all observations in a deterministic fashion. Therefore, we propose a new algorithm, called the Probabilistic Gap Navigation Tree (PGNT) algorithm, that does not treat sensor measurements as ground truth, but creates a probabilistic model of the environment representation that is robust to the measurement noise. The PGNT algorithm, which uses a probability distribution over the maps represented by a set the Gap Navigation Trees, enables navigation in an unknown environment with a realistic sensor.

1.3 Thesis Contributions

The new probabilistic navigation structure accounts for the uncertainty about the environment by sampling from the space of possible gap navigation trees based upon the sensor observations. For each time step, it determines the probability for each tree sample, and resamples the distribution according to these probabilities. This approach is based on the particle filter algorithm [38]. Chapter 4 provides a detailed explanation of the chosen implementation strategy and the optimizations that allow the PGNT to be a simple, yet robust, topological representation of the indoor environments. This algorithm is able to run real time in unknown indoor environment, as verified by running it CARMEN robotic simulation toolkit [24].

The main contributions of this work are as follows:

- We create a special structure called the Probabilistic Gap Navigation Tree (PGNT). The Probabilistic Gap Navigation Tree creates a posterior distribution over trees by sampling the set of possible Gap Navigation Trees, and navigates according to the most likely tree at each time step. The most likely tree is determined based on the learned model of how the tree can change at each time step in combination with current observations of its environment.

- We show that including a limited subset of possible samples of the PGNT, instead of sampling over the entire state space, reduces the dimensionality of the problem without significant degradation in performance.

- We apply a solution to the cyclic order preserving cost assignment problem (COPAP) [10] to predict correspondence between the predicted GNTs and the GNT constructed using robot's laser range finder measurements. We use the correspondence to calculate part of the probability weights for the trees.

- We expand the original GNT model to allow for multiple critical events, making the PGNT more applicable to real environments.

- We demonstrate the functionality of the PGNT by building the navigation struc-

ture using maps from real environments in the CARMEN simulation environment [24].

1.4 Thesis Outline

The following chapters elaborate on the state of the art in topological mapping techniques, and describe the GNT representation and, in full detail, our approach to navigation in non cyclic indoor environments.

Chapter 2 gives an overview of the key concepts in probabilistic navigation. It then introduces the Bayes filter algorithm and one specific implementation the particle filter algorithm, and explains how they can be used in estimating the robot's state.

Chapter 3 discusses navigation algorithms that have been used for limited sensing, focusing on the Gap Navigation Tree (GNT) algorithm proposed by Tovar et al [40, 41]. It details the special environmental representation used in this algorithm, how it changes with the exploration of the environment, and how the robot uses the model to construct a dynamic data structure for navigation in a simply connected, planar environment.

Chapter 4 describes the Probabilistic Gap Navigation Tree (PGNT) algorithm and implementation details. It applies the material presented in Chapters 2 and 3 to develop the probabilistic framework for the PGNT to address some of the shortcomings of the original GNT approach. It further discusses how a solution to the cyclic order preserving cost assignment problem [10] is used to match consecutive sequences of gaps to each other and to derive probability distributions over the environment.

Chapter 5 discusses the implementation details of the PGNT and subsequent testing of the algorithm using three different environments with the CARMEN navigation toolkit using a scout robot simulator with two SICK laser scanners mounted in the front and in the rear. It compares the performance of the GNT and PGNT on the same data sets.

Chapter 6 presents a summary of our work, and expands on the results from Chapter 5 to discuss possible directions for future research in this area.

Chapter 2

Robotic Navigation

One of the most challenging problem that the robots face is the inherent uncertainty in the world around them. Every sensor measurement is imperfect, from the rate of the wheel rotation, to the robot heading direction, to the distances from the robot to the perceived obstacles. In the absence of localization devices such as GPS, one approach to describe the trajectory of the robot is by using a probabilistic framework that assigns a certain likelihood to each state of the robot, and is able to estimate its current state by incorporating its observations about the world. This chapter introduces concepts central to robotic navigation using a probabilistic framework. Section 2.1 introduces the probabilistic models of robot's state and measurements utilized in the remainder of the thesis. Section 2.2 introduces the Bayes filter, which is a general algorithm for predicting a robot's state from surrounding observations. Section 2.3 describes a special type of the Bayes filter called the particle filter, which forms the basis for the navigation approach proposed in this thesis.

2.1 States, Probabilities and Beliefs

First, it is important to define key terms used throughout this work. The state of the robot at a time t will be denoted as x_t. The state of the robot is defined as the collection of all aspects of the robot that can impact the future. For a rigid robot confined to a two dimensional plane, its state can be described through three state

variables - (x,y) coordinates to describe its position in the Cartesian plane and an angle θ for its heading direction.

The robot interacts with its environment in several ways. It can take measurements of the environment with a sensor, for example a range scan measurement with a laser range finder or a sonar. In this discussion, such measurements at a time t will be labeled z_t. The robot can also utilize its control data, such as the robot's velocity, to move in its environment. This control data will be labeled u_t. Odometers, which measure the revolution of the robot's wheels, carry information about the change of state, and thus odometry information can sometimes be also considered as control data [38].

In the probabilistic framework, we may characterize the probability or the likelihood of the robot being in state x_t as $p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t})$. Since the robot relies on the measurements and the odometry information to infer its state, this probability distribution assumes that the current state x_t is a function of all the previous measurements, of all the previous states and of all the previous and current controls. Now, we can simplify the distribution by making an independence assumption, which contends that each future state of the robot is independent of all the previous states, controls, and measurements, given the current state. Thus, we do not need to explicitly keep track of $x_{1:t-2}$, $z_{1:t-1}$ and $u_{1:t-1}$, because they are independent of state x_t given the current state x_{t-1}. The condensed distribution can therefore be written as follows:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{2.1}$$

The simplified expression on the right of equation 2.1 is known as *the state transition probability,* because it describes the evolution of state x_t from the previous state [38].

Using a similar independence assumption, we can construct a likelihood model for the measurements z_t that depends only on the current state, so that all the measurements are independent given the knowledge of the state sequence:

$$p(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \tag{2.2}$$

In equation 2.2, the probability distribution on the right is aptly called the *measurement probability*, because it specifies how the measurements z_t are dependent on the state x_t [38].

There are two other important probability distributions that are often used to describe a robot's perception of its own state. The first is the distribution over the state that incorporates all the measurement and control information:

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{2.3}$$

The distribution shown in equation 2.3 is known as the *posterior distribution*, or the *belief*. The $bel(x_t)$ is a sufficient static of the state x_t, which means that each belief distribution sufficiently represents all of the current and previous measurements and controls. This distribution assumes that the latest measurement z_t has been incorporated into that belief [38].

We can also define the *prediction probability distribution*, which is the belief before incorporating the last measurement:

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \tag{2.4}$$

Having an accurate belief of the robotic state is crucial to robotic navigation, since the state contains all of the sensor and world information. The next two sections will describe how the belief state is calculated. For a more detailed treatment of the subject matter, please see Thrun et al. [38].

2.2 Bayes Filter

Armed with the definitions presented in Section 2.1, we can construct a general algorithm for calculating the robot's belief of its own state (equation 2.3), called the Bayes filter algorithm. It is presented in Algorithm 1.

To help understand the Bayes' filter, we first describe two laws from probability theory. The first is called the *Bayes' rule*. It states that an unknown distribution of

the form $p(a|b)$ can be calculated if one knows conditional distribution of the form $p(b|a)$, and a prior $p(a)$:

$$p(a|b) = \frac{p(b|a)p(a)}{p(b)} \tag{2.5}$$

Since the denominator of equation 2.5 does not depend on the variable of interest a, it can be defined as a multiplicative constant

$$\eta = p(b)^{-1}. \tag{2.6}$$

Bayes filter also uses another mathematical law called the *Law of Total Probability*, which is used to derive the prior $p(a)$:

$$p(a) = \int_b p(a|b)p(b)db \tag{2.7}$$

In this equation the dependence on variable b is eliminated by integrating the variable out, or marginalizing over it.

The above two rules play an essential role in the derivation of $bel(x_t)$ in Algorithm 1. The Bayes filter takes the $bel(x_{t-1})$, control u_t and measurement z_t as inputs. Line 2 uses the Law of Total Probability to calculate the prior $\overline{bel}(x_t)$ by integrating over the posterior of the previous state $bel(x_{t-1})$. To compare with equation 2.7, variable x_t takes place of a, and variable x_{t-1} replaces b. This calculation is called the *control update*, or the *prediction* step of the Bayes' filter.

Line 3 calculates the $bel(x_t)$ by applying the Bayes rule to separate the belief into two measurable quantities: the measurement probability of equation 2.2 and the *prediction probability* of line 2, with the normalization constant η shown in equation 2.6. In this case, the state x_t is analogous to the variable a and the measurement z_t is analogous to the variable b from the equation 2.5. This step is called the *measurement update.*

The shortcoming of the Bayes algorithm lies in the integration of line 2, which requires either a closed form for the state transition probability, or a finite state

Algorithm 1 Bayes Filter Algorithm

Require: Posterior $bel(x_{t-1})$, control u_t and measurement z_t

1: **for** all x_t **do**
2: $\quad \overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})\, bel(x_{t-1})\, dx_{t-1}$
3: $\quad bel(x_t) = \eta\, p(z_t|x_t)\, \overline{bel}(x_t)$
4: **end for**
5: **return** $bel(x_t)$

model, so that the integral of line 2 can be approximated as a sum. In practice, that can only be achieved for a very small set of problems, and therefore many variants of the Bayes filter were created to make that calculation tractable.

2.3 Particle Filter

The particle filter is an example of a nonparametric Bayes filter. In general, nonparametric Bayes filters avoid the closed form requirement for the integration in the prediction step by approximating the beliefs by a finite number of values. In particular, the particle filter approximates the state space by a finite number of values called *particles*, drawn from the posterior distribution $bel(x_t)$. The particles are most densely distributed where the likelihood of the posterior is large, and sparsely distributed where the posterior is small. Particle filtering is also known as Monte Carlo localization [29, 39].

Each belief $bel(x_t)$ is described by a set of sampled states

$$X_t := \{x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]}\} \tag{2.8}$$

where each x_t is a state at time t, also known as a particle, and M is the total number of particles. The full particle filter algorithm is presented in Algorithm 2.

The input to the algorithm is the set of particles from the previous time step, X_{t-1}, control u_t and most recent measurement z_t. In line 3, the state transition distribution is sampled to generate m-th particle of the state $\bar{X}_t$. In the general case, a uniform prior is assumed for the initial distribution the particles in the absence of any information about the environment or the controls. The full sampled set of M

Algorithm 2 Particle Filter Algorithm

Require: Set of particles $\bar{X}_t$, control u_t and measurement z_t

1: $\bar{X}_t = X_t = \emptyset$
2: **for** $m = 1$ to M **do**
3: *sample* $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
4: $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
5: $\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6: **end for**
7: **for** $m = 1$ to M **do**
8: draw i with probability $\propto w_t^{[i]}$
9: add $x_t^{[i]}$ to X_t
10: **end for**
11: **return** X_t

particles x_t represent the prediction $\overline{bel}(x_t)$ of the Bayes' filter.

After each sample has been obtained, line 4 assigns each $x_t^{[m]}$ a probability, or a weight, based on the measurement likelihood at state $x_t^{[m]}$. The weight w_t is also referred to as the *importance factor*, and its role is described below. The new weighted particle is incorporated into the set of $\bar{X}_t$ in line 5. This new set of weighted particles now represents the posterior $bel(x_t)$ of the Bayes filter.

The next *for* loop implements the *importance sampling*, or *resampling*, part of the particle filter algorithm. The particles are drawn with replacement from the set $\bar{X}_t$ using their weight to determine how likely each particle is to be in the final set X_t. The new set of M particles will contain multiple copies of the most likely particles thereby giving higher density to the most likely regions in the belief distribution. In this process, some particles may completely disappear from the regions of very low probability. Therefore, through resampling, the distribution $\overline{bel}(x_t)$ is transformed into the posterior $bel(x_t) = \eta\, p(z_t|x_t^{[m]})\, \overline{bel}(x_t)$, which is the end goal of the Bayes filters. The constant η is no longer explicitly calculated, but is absorbed into the algorithm through the sampling process and the calculation of the weights.

Through their sample-based representation, particle filters are able to represent distributions of arbitrary shape, making them very useful for representing complex beliefs. However, particle filters require a certain number of samples to represent each state. This means that the number of particles necessary to represent a distribution

scales exponentially with the number of its estimated states. Therefore, they are most useful for problems that have a small state space, and must be modified and/or replaced with an alternative solution for high dimensional problems.

Chapter 3

Limited Sensing

The ability to use non-ideal sensors for robotic navigation with incomplete state information is relevant in building real-life robots. A reliable robot that uses cheap sensors, limited power and computational resources would be a convenient and versatile platform for teaching, research, and commercial applications. For this reason, researchers have long studied theoretical limits of sensing information that a robot needs to successfully navigate in its environment. The robots that use only bump sensors and a map, the ones that use finite range lasers without a map, the robots that use low resolution IR or sonar data for localization are all considered to be a part of the limited sensing framework, as describes in this chapter. Section 3.1 provides a general survey of research directions in limited sensing. Section 3.2 is dedicated to discussing a particular limited sensing navigation approach called the Gap Navigation Tree algorithm, which lays the foundation for our research presented in this thesis.

3.1 Limited Sensing Algorithms

Limited sensing approaches have taken various forms in literature. For example, limited sensing for grasping has been studied first by Erdmann and Mason [8]. For a robotic arm to pick up an object, the object either has to be in the same configuration each time, or the arm can use sensory feedback to learn where the object is. They choose the former approach that does not rely on sensors. They devise a

specific tray tilting strategy which is able to position an initially randomly object in a completely determined finite configuration after a certain number of time steps. Although this particular work focuses on grasping, it has been quoted as a basis for several navigation algorithms with limited sensing [9, 40].

Acar et al. [1] approach the problem of sensor-based coverage using a finite range detector. Sensor-based coverage is an approach to create a path that passes a detector over all points of an unknown space. It is useful in fields such as mine detection. Acer et al. use visibility regions to create an algorithm that navigates the world in two different visibility regions, ones where the walls are within a range detector reach and the ones where the sensor range cannot detect all the obstacles. Our thesis is not concerned with an efficient coverage of the visibility area, but rather with an efficient navigation approach that creates a path from a start point to a goal.

Rao et al. [30] tackle a global localization problem for a robot that uses only a finite range sensor and that does not have an initial estimate of its position. To systematically eliminate uncertainty in the pose of the robot, the robot may to move several times to gain information about its environment. Rao et al. present an algorithm that creates the shortest-distance path that localizes the robot in a minimum number of time steps. One limitation of their approach is that they do not consider sensor noise when constructing their algorithm.

O'Kane and LaValle [26] consider localization in an *a priori* mapped environment for robots equipped with a compass and a bump sensor. They prove that it is possible to construct a path for localization in a simply connected, closed polygonal environment, and present algorithms for effective localization. They furthermore show that if the robot does not use a compass, but instead relies on its odometry information to derive the heading direction, localization becomes impossible.

One of the early approaches for robotic navigation with limited sensing without a map was introduced by Lumelsky and Stepanov [23]. They proposed the first formulation of the *bug algorithm*. The algorithm assumes that the robot is a point, obstacles can have an arbitrary shape and a finite size, and the environment can be infinitely large. The information available to the robot comes from its odometry and

a bump sensor. This information is proven to be sufficient for reaching a known target position, or concluding in finite time that the target cannot be reached. Kamon et al. [16, 15] have extended the bug algorithm to incorporate range measurements and exhibit wall following properties, in addition to moving straight towards the goal.

Rajko and LaValle [28] assume the same limited sensing capabilities without a map as Lumelsky and Kamon, and formulate an algorithm that searches for an unpredictable moving target in a closed polygonal environment. They prove that the target will always be detected in finite time. In their work, they demonstrate that the robot can solve the target finding problem just as efficiently and accurately as a robot having full knowledge of its environment and perfect sensors.

Beevers [3] published a PhD thesis and several papers on topological mapping with limited sensing. In Huang and Beevers [14], they propose an algorithm for creating a topological map using very limited infrared range sensors (80 cm maximum range) in a closed environment. Their topological map represents the world as a graph, where the corners of the wall are the vertices of the graph, and the walls are the undirected edges. In their algorithm, by performing wall following, the robot eventually completes a loop and returns to the original starting point. By continuing to follow the walls after the initial loop closing, the robot can disambiguate its location by comparing new sensor measurements to the previously acquired world representation. In their approach, the distance travelled to a particular corner may not be optimal, which they attribute to the limitation of the sensor model used in their work.

In his PhD thesis [3], Beevers developed a model of mapping sensors that he then combined with a simple mapping algorithm to characterize a set of sensors in terms of their suitability for mapping. His work focused on simultaneous localization and mapping (SLAM) with low-resolution sensor measurements. He created a novel update technique for limited sensing as well as new sampling strategies to improve particle filtering SLAM in the limiting sensing context.

The Gap Navigation Tree from Tovar et al. [40, 41] is the main subject of the next section. It creates a topological representation of an environment using a tree, that allows the robot to navigate with only one sensor that tracks direction and depth of

discontinuities in the robotic field of view. It is the basis for our probabilistic mapping framework, and it has inspired other related research. For example, Landa et al.'s [20, 21] work creates a path planning algorithm, in which a group of autonomous vehicles explores and constructs a map of an unknown environment. Their mapping approach borrows the concept of the navigation tree, but uses a different method for creating it based on the sensor measurements. First, they sample from opaque objects (obstacles) in the environment to create point clouds. They project the point clouds onto a sphere centered at the observed locations, and perform essentially non-oscillatory (ENO) interpolation [12] to the projected data. The ENO interpolation allows the surface curvature to be calculated through approximation of higher order derivatives. They then use the surface curvatures to determine the boundaries of the visibility regions, and thus extract edges for the gap navigation tree. Although applicable to robotic navigation, their focus is defining visibility regions for graphics and surveillance applications.

Murphy and Newman [25] have used the point cloud sampling of Landa et al to generalize the Gap Navigation Tree to guide a Simultaneous Localization and Mapping process. Their navigation approach assumes a noisy range sensor with limited range. They use probabilistic inference on the point clouds to predict the location of each gap at every time step. They eliminate appearances of false gaps due to sensor noise by waiting a few time steps to make sure that the new gap persists over several measurements. Although our work also utilizes a probabilistic model, it is different from Murphy's in that we predict all the possible gap navigation trees based on the navigation model, instead of predicting just the gap locations.

3.2 Gap Navigation Trees

The Gap Navigation Tree structure was introduced by Tovar et al. [40, 41]. In their navigation approach, the authors try to determine the feasibility of navigating with minimal sensor information available to the robot. They created the GNT structure that allows the robot to navigate in a simply connected, closed polygonal environment.

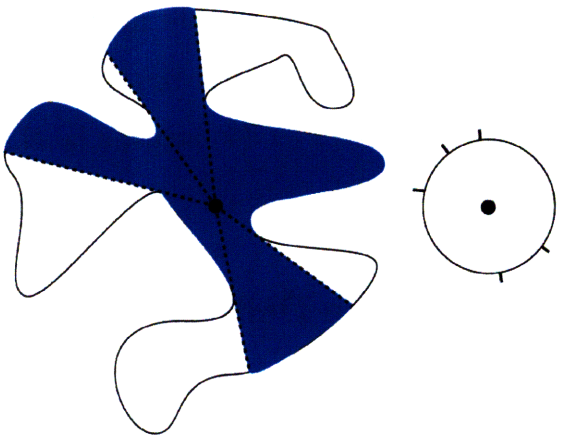


Figure 3-1: Left: A robot in the environment uses a range sensor to sees areas in blue and doesn't see areas in white. Right: gaps corresponding to the shown environment. Image courtesy of Benjamin Tovar [41]

The authors show that without storing any range or heading information for the robot, they can successfully build a topological map and use it to navigate in an unknown environment.

3.2.1 Gap Definition

The key idea behind the Gap Navigation Tree is the novel way it represents the environment. It divides the environment into regions of visibility, distinguishing between surroundings the robot can and cannot see. For example, in Figure 3-1, a robot equipped with a 360° laser range finder would see the area shaded in blue, but will not be able to see the white areas because those are occluded. These discontinuities in the robot's field of vision are called *gaps*, and the region that is not visible to the robot is labeled R. The robot is assumed to have a gap sensor that reliably detects the discontinuities and tracks how these gaps change as the robot moves (not necessarily a laser rangefinder). The robot does not have any other sensors such as a compass, odometers, or other range sensors and is not capable of building an exact map of the environment.

The sequence of gaps that appears in the gap sensor when the robot is at position

$x \in R$ is defined as $G(x) = [g_1, ..., g_n]$. If x is on the interior of R, we can assume that $G(x)$ will be cyclically ordered, which means the sequence $[g_1, ..., g_n]$ is equivalent to $[g_n, g_1, ..., g_{n-1}]$. The subscript of the $g_i \in G(x)$ is just a label and it does not contain any angle or depth information.

3.2.2 Gap Navigation Tree Definition

Any environment can be decomposed into regions of *similar visibility*, within which navigation always produces the same sequence of gaps $G(x)$. However, due to the geometry of the environment, as soon as the robot crosses the boundary into another visibility region, the observed gap sequence $G(x)$ changes. These changes may cause the gaps g_i to appear in the field of view, disappear from it, two gaps may merge to leave only one gap in their place, or one gap may split to create two gaps. The changes in the environment can be tracked in a systematic way to create an environmental representation called the Gap Navigation Tree (GNT). [41].

To motivate the utility of the tree like structure for navigation, we first attempt to track changes by solely using gap sequence $G(x)$. Assume that the robot moves along a path $\tau : [0, 1] \rightarrow R$. Then, for every time step s we can define a gap sequence $G(\tau(s))$ that is observed by the gap sensor. At time $s = 0$, suppose that $G(\tau(0)) = [g_1, g_2, g_3]$. At some later time step s_l, the gap g_2 splits into two gaps $[g_4, g_5]$. To store that new information, we remove g_2 from the gap sequence, and insert the new gaps in its place to preserve cyclic ordering, transforming our gap sequence $G(\tau(s_l)) = [g_1, g_4, g_5, g_3]$. At an even later time step s_{l2}, g_4 and g_5 merge, causing yet another update to G. The update to the sequence can either be labeled as a new gap g_6, or it can be labeled as a combination $[g_4, g_5]$ to keep track of the underlying structure of the merges in the sequence $G(\tau(s_{l2})) = [g_1, [g_4, g_5], g_3]$. These additions, continuing over the course of exploration, result in complicated nested expressions in the gap sequence $G(\tau(s))$.

A more elegant approach to keeping track of the gaps is to transform the gap sequence G into a tree structure which will store the information about the changes in its environment in its nodes. This structure is called the *Gap Navigation Tree* (GNT), and has the following properties:

- Every non-root vertex of the GNT represents a gap in $G(\tau(s))$ at some time $s \in [0, 1]$.

- Every child of the root vertex corresponds to a gap in the most current sequence $G(\tau(1))$, and the children are cyclically ordered in the same way as they appear in G.

- All vertices that are not children of the root represent gaps that appeared in $G(\tau(s))$ for some $s < 1$, but have disappeared from the set $G(\tau(1))$ due to merging.

- Children of every non-root vertex v are the gaps that merged to form that vertex, and appear in the same cyclic ordering as they appeared in G.

The details of the Gap Navigation Tree construction is the subject of the next two sections.

3.2.3 Gap Navigation Tree Construction Using Critical Events

The Gap Navigation Tree is constructed incrementally as the robot moves through its environment by storing the critical events from the gap sensor. At the beginning, the GNT contains the children of the root corresponding to $G(\tau(0))$. As the robot moves along a path τ, the gaps around it can change in the four following ways, forcing an update to the Gap Navigation Tree:

1. Gap Disappearance: When a hidden section of the environment becomes exposed, a gap defining the hidden section disappears. The leaf corresponding to that gap is then removed (illustrated in Figure 3-2 (c).

2. Gap Appearance: As the robot moves along, a section that has previously been visible becomes hidden again due to the environment geometry. In this case, a gap appears and is added to the GNT as a child of the root, preserving the cyclic ordering of the original gaps as in Figure 3-2 (a).

3. Gap Merge: Two gaps previously associated with two separate occlusions merge into one due to the environment geometry and the robot's vantage point. A new gap is added in their place as a child of the root, while the two previous gaps are preserved as children of the new gap as shown in Figure 3-2 (b).

4. Gap Split: The opposite of gap merge happens when one occlusion separates into two as the robot continues to explore the environment. If the original gap was a childless leaf, it is removed and two gaps are inserted in its place. If the original gap had children, the children are inserted as children of the root while the original gap is removed. The latter case is illustrated in Figure 3-2 (d).

Because the measurements as assumed are taken infinitely often, gap merging and splitting are not discontinuous processes: there exists a measurement such that $(g_i - g_j) < \epsilon \forall \epsilon$. This allows us to distinguish between a split and an appearance, and between a merge and a disappearance events.

By moving and storing critical events, the robot is building a topological representation that will allow it to navigate in an unknown environment. The robot follows a specific plan for optimizing the distance traveled, which is the subject of Section 3.2.4.

3.2.4 World Exploration: Building a GNT

The robot explores the environment by choosing one of the gaps, and moving towards it until it disappears. This motion primitive is called a *chase action*, and is labeled *chase(g)*. If a chase motion is performed and the gap being chased disappears, the entire region behind the gap is now visible. Therefore, the robot finished exploring that part of the environment and must choose another gap toward which to move. If the gap chased by the robot splits, the robot chooses yet another gap to chase. The robot continues chasing gaps until the Gap Navigation Tree is complete.

How does the robot decide when the Gap Navigation Tree is complete? That happens when all the hidden corners of the environment have been revealed to the robot at some point during its exploration. To keep track of the explored portions of the

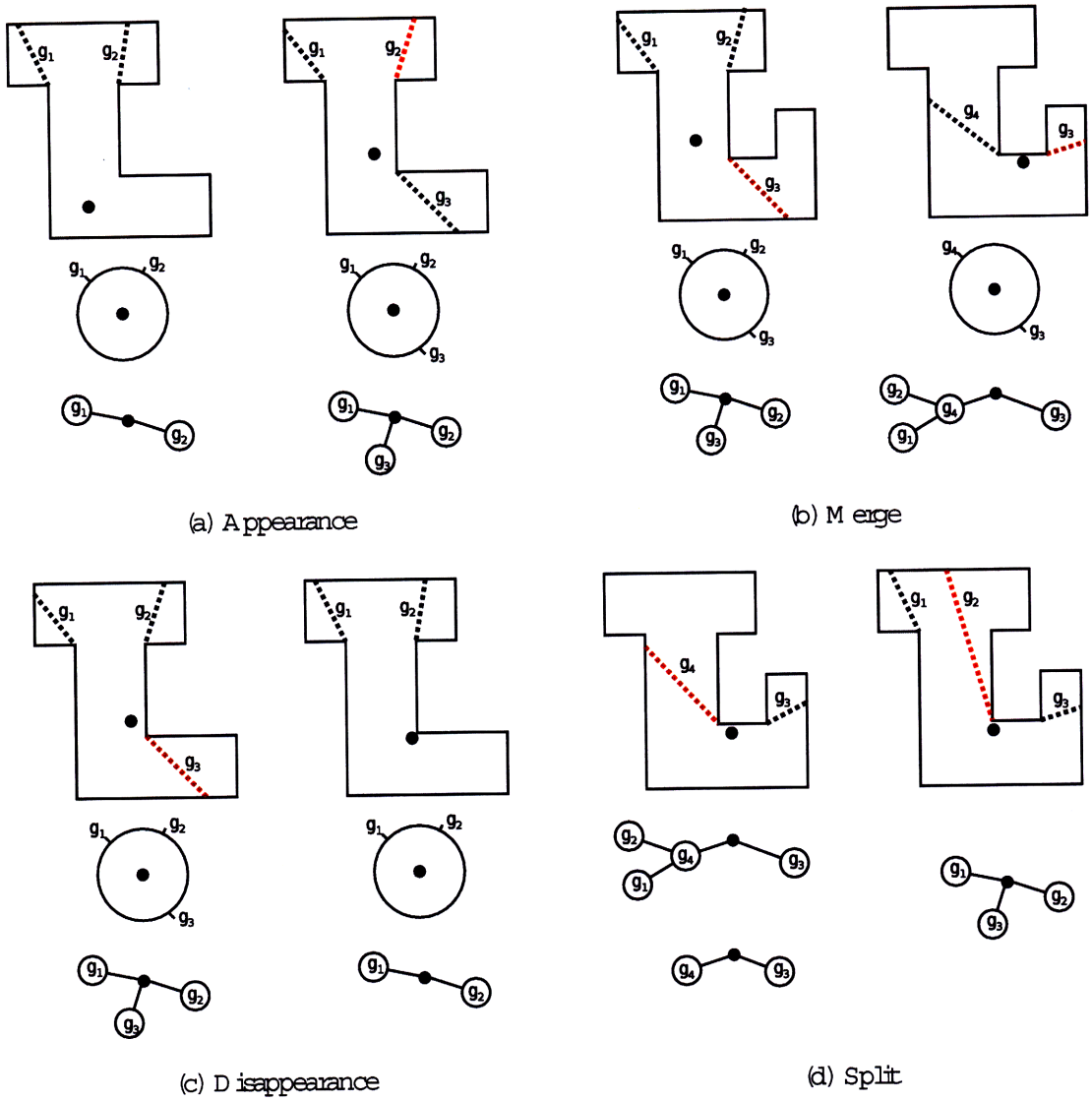


Figure 3-2: Illustration of the gap critical events and how they influence the tree structure for (a) Appearance, (b) Merge, (c) Disappearance and (d) Split. Image courtesy of Benjamin Tovar [41].

environment, each gap is assigned a label: *primitive* or *nonprimitive*. Nonprimitive gaps are all those that split when the robot chases them, thus further expanding the navigation tree. Primitive gaps are the ones that disappear when chased, forcing the robot to find another gap to chase. Therefore, the gaps that appear during exploration are labeled as primitive, while all others are labeled as nonprimitive. (The gaps that appear cannot split any further, because if the robot decide to chase the gap it must split again). The tree is complete when all the leaves have been labeled

Algorithm 3 Gap Navigation Tree

Require: Set of gaps $G(x)$

1: initialize GNT from $G(x)$
2: **while** $\exists$ nonprimitive $g_i \in GNT$ **do**
3: choose any nonprimitive $g_k \in GNT$
4: **for** each time step s **do**
5: $chase(g_k)$ until *critical_event*
6: update GNT according to *critical_event*
7: **end for**
8: **end while**
9: **return** GNT

as primitive as the result of exploration.

The algorithm presented in this chapter is formally summarized in Algorithm 3. The input to the algorithm is the sequence of gaps $G(x)$. The GNT is initialized by inserting the children at the root of the tree corresponding to each gap $g_i \in G(x)$. Lines 2 through 8 show that to navigate in the environment, the robot first chooses a non primitive gap, then chases it until a critical event happens, and updates the GNT. If there are more non primitive gaps left, the robot continues exploration; otherwise it returns the GNT, which, by the above definition is complete.

3.2.4.1 Landmark Encoding

A complete navigation tree contains all the information about the changes in the visibility regions. To navigate using the tree, the robot first searches the tree for the appropriate leaf (or vertex) that represents its desire goal, and then follows the shortest path as dictated by the path through the tree from the root to the vertex.

As an extension to the basic navigation, the robot can be tasked to find certain landmarks in the environment (for illustration, see Figure 3-3). It can store the object of interest as a child of the gap that first obscured the object from view. This way, the gap marks the object location. The authors demonstrate that the shortest path in a Euclidean sense from one landmark to another can be achieved by traversing the Gap Navigation Tree.

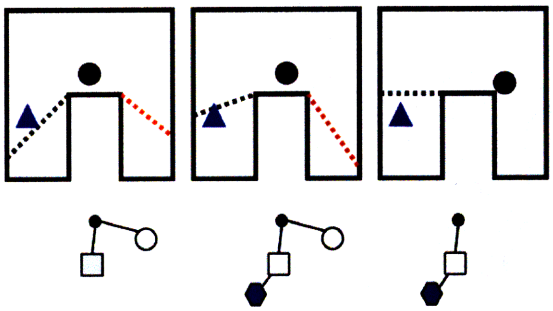


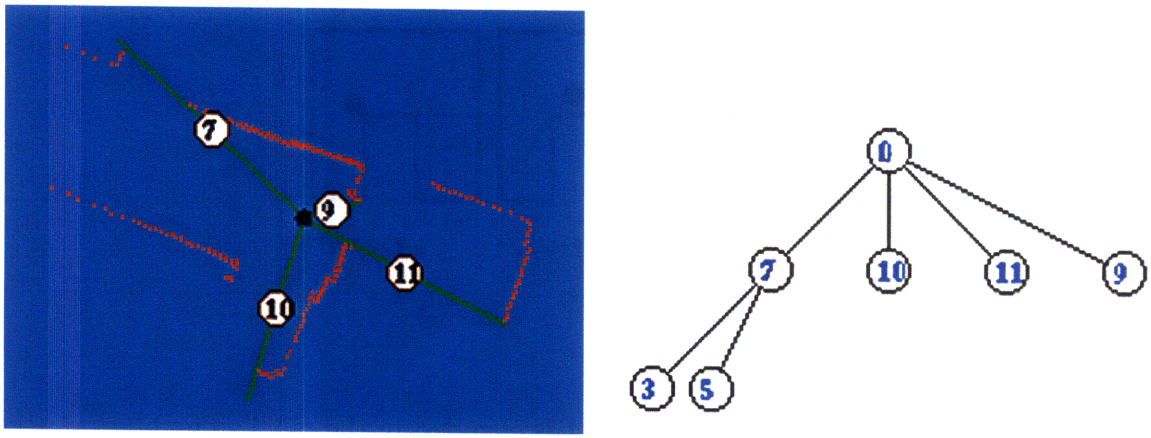
Figure 3-3: This figure shows how an object is encoded in the GNT. The black dot is the robot, the blue triangle is the object. In the figure on the left, the object is visible. As the robot moves, the object becomes hidden by a new gap (middle). The object is encoded into the tree by being inserted as a leaf of that gap. During navigation, the landmark will appear as soon as the gap hiding the object disappears during a *chase* action. Image courtesy of Benjamin Tovar [41].

3.2.5 Limitations of the GNT

As a foundation of our thesis work, we implement the Gap Navigation Tree using the CARMEN robotic navigation toolkit [24]. The simulated navigation platform is the Nomad Scout robot equipped with one front and one rear SICK laser rangefinder, allowing a 360° view of the world. In this particular simulation, the laser range is limited to 5 meters.

A graphical user interface displaying the gaps and their corresponding Gap Navigation Tree as the robot moves around is shown in Figure 3-4. The navigation environment is a floor of a building on a University of Freiburg campus obtained from the Radish dataset [13]. The red outline represents the walls that the laser range finder detects. The green lines extending from the robot center indicate the positions of the detected gaps. In the image of the navigation tree on the right, one of the nodes has two children. These children are a result of a merge in the current time step, and they are added to the tree as was discussed in Section 3.2.3 and demonstrated in Figure 3-2.

Our attempt to model a simulated world with the GNT proved unsuccessful. The implementation of the GNT as described by Tovar et al. assumes that we have an ideal, infinite range gap sensor. Given perfect laser scan data at each time step, the



(a) (b)

Figure 3-4: This is a screen shot of two graphic user interfaces. Figure (a) displays the environment as seen by the robot (black dot). Red dots represent obstacles, in this case walls of the corridor with view into some rooms. The gaps are shown as green lines, and indicate the borders of the visibility region. Each gap has a circle and a number on top of it, that indicates the node of the navigation tree in (b) to which that particular gap corresponds.

edge detector could successfully extract the gaps. However, all crude edge detectors, and even sophisticated laser rangefinders, are susceptible to measurement noise, and the modeled SICK lasers were no exception. Presence of sensor noise causes the rangefinder to detect gaps where they do not exist, and the tree to unreliably track the critical events in the environment.

Additionally, the algorithm suffers from the assumption that the robots are point features when, in reality they cannot be treated as such. While performing a chase action, the point assumption may cause the robot to bump into the wall as it is moving straight for the gap. The robot has to be equipped with a robust wall following technique to reliably execute its motion.

Another failure mode of the GNT is when the robot measures gaps from a position close to a wall. The distance between consecutive data points in one laser range scan further along the wall is large enough to create false gaps with the current GNT approach. The GNT does not have a reliable mechanism for discarding those gaps. Another unreal assumption is that, at each gap detection step, there can be only one critical event, which is not the case with a realistic environment such as the University

of Freiburg map that we use here.

All these limitations suggested a new sensor model can be applied to create a more robust navigation approach, which is the motivation for our work. We expand the GNT to include sensor noise in its gap tracker to reliably detect gaps and the changes among them. We also include the ability to track more than one critical event per time step. The details of the implementation are the subject of Chapter 4.

Chapter 4

Probabilistic Gap Navigation Tree

The purpose of our work is to establish a navigation approach for autonomous robots in unknown environment with sensing subject to measurement noise. We utilize an environment representation called the GNT based on the boundaries of the visibility regions as described in detail in Chapter 3. Our contribution is centered around the creation of the Probabilistic Gap Navigation Tree (PGNT) algorithm that would build an optimal gap navigation tree subject to error in the range sensors. Section 4.1 introduces the probabilistic framework for the PGNT and discusses some other probabilistic navigation techniques from literature. Section 4.2 establishes the probabilistic framework specific to the PGNT. Section 4.3 discusses the application of the Cyclic Order Preserving Assignment Problem (COPAP) to correspondences for tracking changes in the environment as well as assigning probability weight to each environment representation stored by the algorithm. Section 4.4 gives a detailed description of the algorithm implementation. Section 4.5 explores an alternative sampling strategy for increasing the efficiency of the PGNT algorithm, while section 4.6 describes a modification of the GNT to allow multiple critical events in the navigation tree.

4.1 Algorithm Introduction

The PGNT algorithm creates a distribution of the gap navigation trees, in which each sample is weighted according to the most likely tree based the observed sequence of gaps at time t. This approach adopts the state update based on the particle filter algorithm introduced in Chapter 2 (with a more detailed treatment in Thrun [38]). The posterior distribution of robot states is represented as a set of gap navigation trees, created before the latest measurement has been incorporated. Each gap navigation tree is assigned a weight based on the most recent laser range scan measurement of the environment. The posterior is then resampled according to the weights to create a hypothetical current state, or the *belief*.

The idea of using probabilistic navigation is far from new. There are numerous probabilistic approaches for topological maps, for example the work of Ranganathan and Dellaert [29] that specifically utilizes particle filtering for topological mapping. In particular the particle filter is used to construct a posterior distribution of all possible environment topologies given a measurement. Their work even proposes a method to finding a global metric map from the topological map in post processing, thus attempting to solve the cyclical mapping in large environments problem that normally is impossibly using topological mapping approaches.

Bulata and Devy [4] propose an algorithm for robotic exploration in which the robot must build successive snapshot models from the laser rangefinder, and fuse them in a global model so that it can localize itself with respect to some global reference frame. Instead of extracting boundaries of the visibility regions they extract useful landmarks from local feature groupings, like wall corners, doors, corridor crossings etc. to create a topological map. They use a form of the Bayes filter called the extended Kalman filter for their belief updates.

These papers represent just two of voluminous research on this subject, and a more comprehensive overview of the earlier probabilistic mapping navigation techniques can be found in [36, 35].

4.2 Defining Probabilities for the PGNT

To generalize the Gap Navigation Tree algorithm to a probabilistic framework, we revisit the definitions presented in Chapter 2 and apply them to our mapping approach. Recall that the Bayes' filter calculates the belief $bel(x_t)$ of the robot's state x_t, where the robot's state is its coordinates and heading direction, or the robot's pose. In the GNT framework, we do not want to estimate the robot's pose at each time step, but we want to predict the Gap Navigation Tree. Therefore, the state x_t is replaced by the GNT at a time step s (to keep the notation from Chapter 3), which gives

$$bel(x_t) \rightarrow bel(GNT_s).$$

Furthermore, the Gap Navigation Tree is independent of the robot's odometry, so we can simplify the belief distribution as follows:

$$bel(GNT_s) = p(GNT_s \mid z_{0:s}) \tag{4.1}$$

Moreover, the current belief of the tree is a sufficient statistic of all previous trees and measurements, because the automatically encodes the measurements into its structure. Therefore, we can simplify equation 4.1 to

$$bel(GNT_s) = p(GNT_s \mid z_{0:s}) = p(GNT_s \mid z_s) \tag{4.2}$$

As described in the previous chapter, the measurements that cause the update to the GNT are the critical events. Critical events, in turn, are calculated from comparing the GNT from the previous time step to the new gap sequence G_s. Therefore, we formally define the measurement probability of equation 2.2 as follows:

$$p(z_t \mid x_t) \rightarrow p(G_s \mid GNT_s) \tag{4.3}$$

We represent the $bel(GNT_s)$ as a collection of tree instantiations t at time step s:

$$T_s := \{t_s^{[1]}, t_s^{[2]}, ..., t_s^{[M]}\} \tag{4.4}$$

By representing the belief as in equation 4.4, we can use the particle filter approach to estimate the belief at each time step. We rewrite the necessary probability distributions from Algorithm 2 as follows:

$$p(x_t \mid u_t, x_{t-1}^{[m]}) \rightarrow p(t_s \mid t_{s-1}^{[m]}) \tag{4.5}$$

$$p(z_t \mid x_t^{[m]}) \rightarrow p(G_s \mid t_s^{[m]}) \tag{4.6}$$

Equation 4.5 describes the distribution that is sampled to obtain new trees. Since we do not incorporate control data, the variable u disappears from the equation. The probability distribution on the right is the state transition probability for the tree, which indicates the likelihood of a tree at time step s given m-th particle at the previous time step $s-1$. The way to calculate this probability is describe in Section 4.4.

Equation 4.6 represents the measurement probability of the mth tree, and it echoes equation 4.3 above. This is the probability used to calculate the weight of each tree, and we perform the calculation by utilizing Bayes rule:

$$p(G_s \mid t_s^{[m]}) = p(t_s^{[m]} \mid G_s)\, p(G_s) \tag{4.7}$$

The probability distribution $p(G_s)$ was determined experimentally by driving the robot around in a simulated environment and counting the number of time steps the sequence G_s stayed the same versus undergoing a change due to a critical event.

The distribution $p(t_s^{[m]} \mid G_s)$ is calculated by utilizing a metric that measures correspondence between the mth tree sample and the most recent gap sequence G_s. For the correspondences, we not only store the latest gaps in the sequence G_s, but also the angular measurements that corresponds to those gaps. The full description of this method is the subject of Section 4.3.

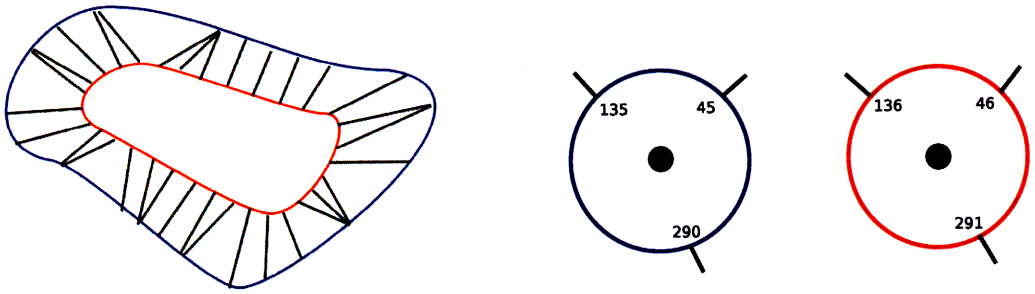
4.3 Cyclic Order Preserving Assignment Problem for Correspondences

Cyclic Order Preserving Assignment Problem, or COPAP for short, is a problem of matching two shape countours to one another while preserving the cyclical ordering, where the countours are represented by a set of points that do not necessarily have one to one correspondence. A good mathematical description of the shape correspondence problem is presented in [10], and a recent optimal solution presented in [10, 33].

This problem is very similar to the problem of corresponding two gap sequences: we can think of a robotic field of view as a polygon where the gaps are samples on the viewing boundary. Comparing two sequences of gaps at times $s-1$ and s is the same as calculating the correspondence between two very sparsly sampled polygons. Figure 4-1 illustrates this concept. In Figure 4-1 (a), two shapes are sampled and then matched as illustrated by black lines. Notice that one sample on the blue contour may correspond to more than one sample on the red contour. Figure 4-1 (b), shows two consecutive gap representations of the environment. They are meant to be matched or *corresponded* as described in line 8 of algorithm 4. The problem becomes to correspond a cyclic set of gaps $G^b = [45, 135, 290]$ from the blue circle to the set $G^r = [46, 136, 291]$ from the red one. Because this is a *cyclic* assignment problem, in a set $[g_1^b, ..., g_n^b]$, g_1^b may be matched to g_n^r.

4.3.1 Extracting the Gap Critical Event from the Angular Gap Positions

The example from Figure 4-1 (b) is going to serve as a reference for explaining how the solution to the COPAP problem will determine the gap critical events in the tree. After visually inspecting the image in (b), it is easy to tell how the two sets of gaps correspond to one another, because in each pair of matching gaps they are very close together. They are separated by only 1° from each other, while being removed from neighboring gaps by as much as 156°.



(a) (b)

Figure 4-1: This is an analogy between shape matching and gap matching during gap navigation tree building. Figure a) shows matching the blue shape to the red shape via black lines. Each line originates and terminates at a sample on the shape contour. From [10]. Figure (b) displays the environment as seen by the robot (black dot) analogous to representation in figure 3-1. We want to match the blue circle to the red one. In this case, the black marks represents the gaps, and the numbers next to the marks represent angular orientation of the gaps.

The distance between each pair of gaps in the sequences G^b and G^r becomes a cost metric for whether or not that pair should be formally defined as corresponding or not. Figure 4-2 illustrates the distance matrix D created for the example problem, in which every $D(i,j)$ node represents the distance between gaps g_i^b and g_j^r. The cost of corresponding all the gaps is the sum of individual correspondence costs. If one or more of the gaps remain unassigned, a certain penalty is added to the sum. The goal is to match gaps in such as way as to minimize the total correspondence cost, even if that means that some gaps remain unassigned.

As the robot moves through the environment, the angular positions of the gaps will change, and new gaps will be added or removed from the environment. If a new gap suddenly appeared between $g_1^b = 45°$ and $g_2^b = 135°$, to give the consecutive scan that looks like $g^r = [46°, 100°, 136°, 291°]$, it is clear that this critical event is a gap appearance, because 100° is far enough away from both 46° and 136°. How far is "far enough"? A parameter λ is chosen to decide whether or not the distance between the gaps is close enough for the correspondence matching. So if the new gap is closer than λ to one of its neighbors then action is classified as a split, and vice versa for the gap merge. For most of our environments, λ was chosen to be 16°.

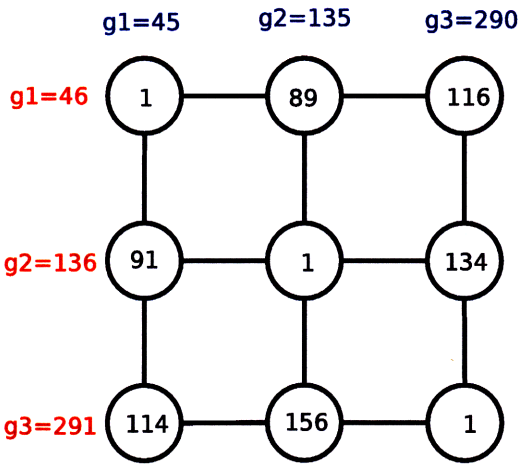


Figure 4-2: This is the distance matrix $D(i, j)$ built using the distances between each gaps. Each (i, j) pair represents the distance between gaps g_i^b and g_j^r

Formally, a standard way to solve the COPAP is to first consider the linear order preserving assignment problem (LOPAP), and then to solve the linear version multiple times for different starting points. LOPAP looks the same as COPAP except that it doesn't allow wrap around, so it is not cyclical. There is no wrap around in the sample problem, so the LOPAP solution will be the optimal one in this specific case which we present here. The way to solve this problem is to look for the lowest cost path originating at (0,0) in the distance matrix D.

The solution to finding the lowest cost path can be found in dynamic programming approaches. Using the distance matrix, a special cost graph $C(i, j)$ is formed in which the distances from $D(i, j)$ become transition, or correspondence costs of assigning index i to corresponding index j. If there is no match between the gaps, λ is added to the cost path as a penalty.

Figure 4-3 (a) shows a cost matrix with a filled out bottom row. It is of size (m+1, n+1), with the bottom and rightmost rows filled out with a special pattern of λs. The arrows are color coded to show every way in which cost may be computed as traveling through the matrix: the penalties of λ for non matching are shown in black, while the costs equal to the distances from the D matrix are shown in orange. When computing the cost of a node C(i,j), if the distance D(i,j) is lower than λ then it is accepted as a cost, and if the distance is above λ then the correspondence for that

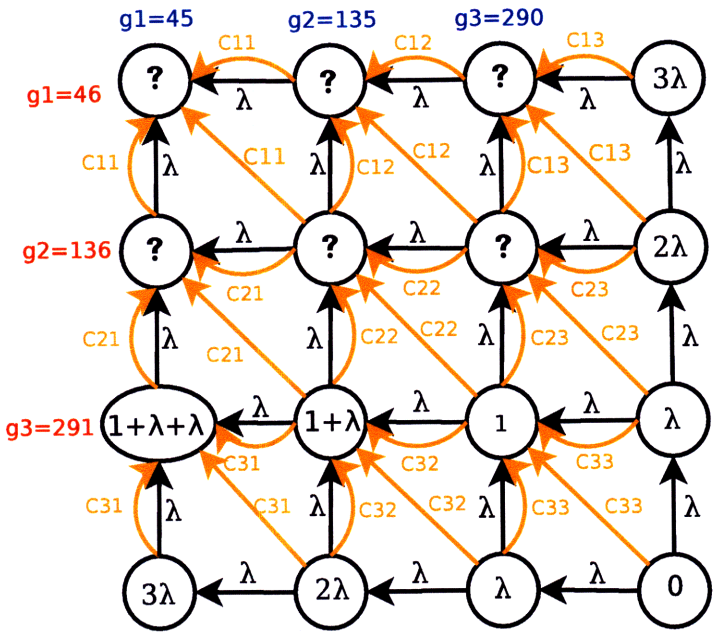
node pair is stored as "does not exist". Figure 4-3 (b) shows the filled out matrix using $\lambda = 16$ and values from the sample problem. The minimum cost is highlighted in green, and shows what is easily seen to a human, which is that the cost of going through the matrix is minimized when corresponding all the nodes where $i = j$.

What would happen if our sample problem involved a "wrap around"? This can happen when a gap moves from 359° to 1° at the next time step. In this case, the LOPAP problem would not give the minimum matching because the minimal cost path to the top node (matching g_1^r to g_1^b) does not give the optimal solution. Figure 4-4 shows a cost matrix with a path to the top node, which shows the total cost of getting to the top a lot higher than in the previous example, although the distance metric between the gaps has not changed.

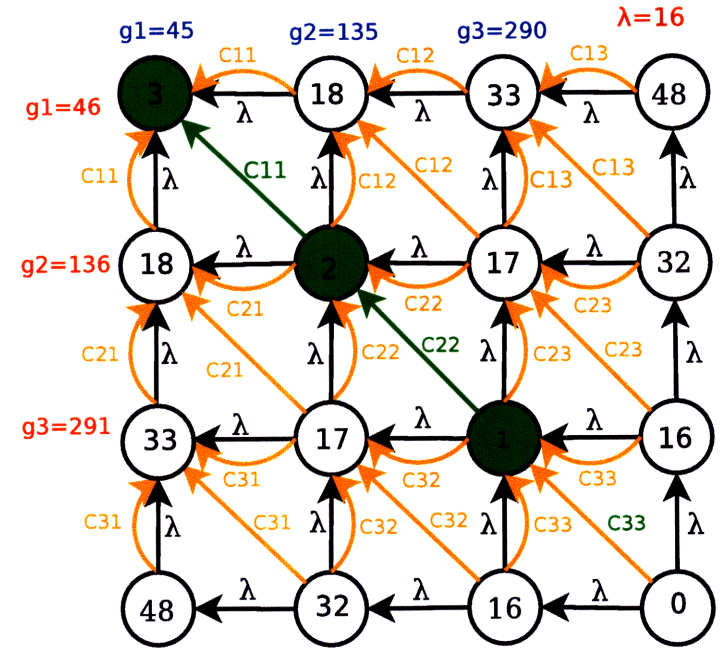
For the cyclical problems, it is therefore necessary to consider more than one arrangement of the cost matrix. Upon visual inspection of Figure 4-4 (b) we can see that the optimal matching can be obtained if the top row of the matrix is moved to the bottom, because then the matrix D from Figure 4-4 (b) would look exactly like the matrix D from Figure 4-2, which computes the optimal minimal cost path to the top node. Therefore, the full COPAP problem is solved when the cost matrix is computed for each cyclical permutation of the rows of the cost matrix. The permutation that produces the minimum cost path to the top node is the one that is accepted as the correct solution.

4.3.2 Calculating Measurement Probability

In the particle filtering algorithm, assigning the correct weights to each particle is crucial to the success of the implementation, because the weight determines how well the posterior is approximated. The analogous probability function in the PGNT framework is defined in equation 4.6. The correspondence function helps to determine how closely related the current gap sequence G_s is to the proposed m-th navigation tree $t_s^{[m]}$, which defines probability $p(t_s|G_s)$. Using COPAP is possible due to a modification to the GNT that allows each child in the top node of the GNT to store an angle associated with that gap.

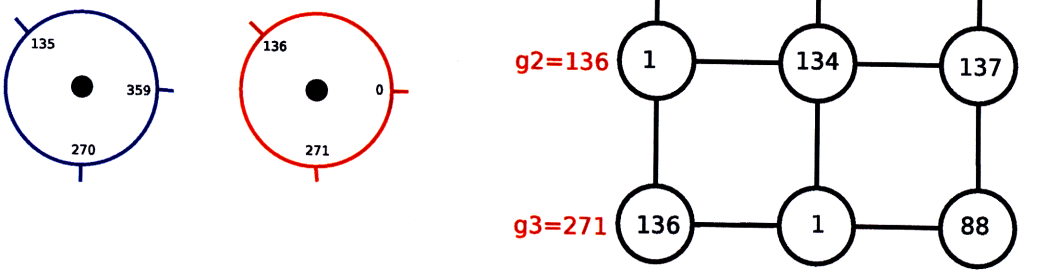


(a)

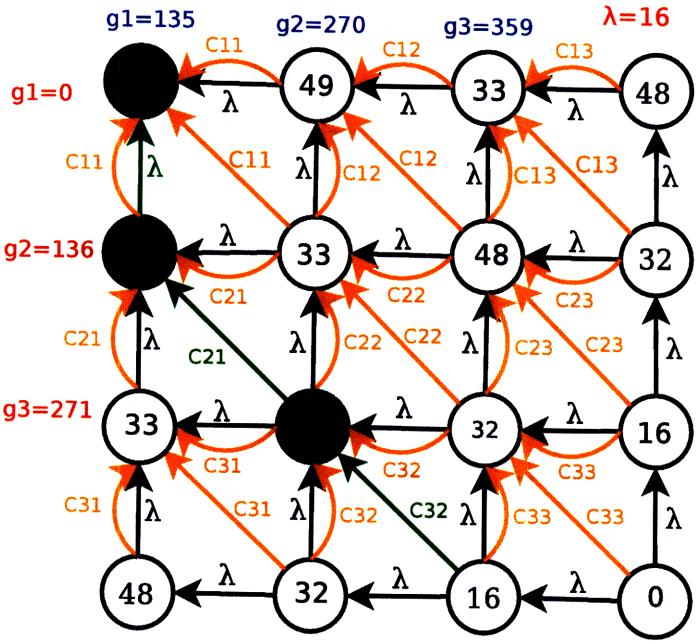


(b)

Figure 4-3: The two images illustrate the state of the cost matrix in (a) after the first row is constructed and (b) after substituting for $\lambda = 16$ and computing the whole matrix. The path of the optimal match is highlighted in green.



(a) (b)



(c)

Figure 4-4: The three images illustrate the cyclical wrap around problem: (a) shows the robot gap view with gaps, (b) demonstrate the distance matrix D, while (c) constructs the full cost matrix with the optimal matching is highlighted in green. In this case, the optimal matching is not the right solution to the problem.

Several steps are executed to correspond each tree to the sequence G and to assign the appropriate probability. First, the angles of the children of the root node, call them G_{tree}, are extracted from the tree, because they reflect the state of the environment according to $\overline{T}_s$. Then, they are matched to the sequence of gap angular measurements G_{angles} extracted from the latest scan of the laser rangefinder. If there is an *unmatched* gap in G_{tree}, it is treated as a wildcard, and the correspondence algorithm matches it in an optimal way to preserve cyclic ordering of the gaps and producing an optimal cost for the matching.

If after the end of correspondence there are still unmatched gaps, $p((t_s^{[m]}|G_s))$ is multiplied by a *penalty factor* for each one, which is a penalty showing that uncorresponded trees are not common. Vice versa, if there are more gaps in the environment that is found in the vector g_{tree}, the probability is again penalized for each missing gap to account for the mismatch. The penalty factor was determined experimentally by running several log files many times.

4.4 PGNT Implementation

The PGNT algorithm implementation is presented in Algorithm 4. The inputs to the algorithm are the previous set of trees T_{s-1}, the previous measurement G_{s-1}, which is a list of angles corresponding to a sequence of gaps extracted from the previous laser range scan, and the new measurement G_s, which is a sequence of gap angles from the current range scan.

The gap extraction is a simple procedure in which we compare the distance between each consecutive pair of points in the laser rangefinder measurement, and declare a gap if the distance between the points is above a certain threshold. The best threshold depends on the size of the features that need to be resolved. For example, a low threshold for the gap may enable an extraction of gaps in places where there is a shallow cavity for the door in a hallway, while a larger threshold may not detect it due to a large difference required between consecutive measurements to label a gap.

The if statement in line 2 of the PGNT algorithm makes sure that we already

Algorithm 4 Probabilistic Gap Navigation Tree

Require: Previous tree array T_{s-1} of length M, previous gaps G_{s-1} and new gaps G_s

1: **for** $i = 1$ to N **do**
2: **if** $(T_{s-1} = \emptyset)$ **then**
3: sample $t_s^{[n]} \sim p(t_{s-1}^{[m]}|G_s)$
4: $w_s^{[n]} = 1$
5: $\overline{T_s} = \overline{T_s} + \langle t_s^{[n]}, w_t^{[n]} \rangle$
6: **else**
7: sample $t_s^{[m]} \sim p(t_{s-1}^{[m]}|G_{s-1})$
8: $w_s^{[m]} = p(t_s^{[m]}|G_s))\, p(G_s)$
9: $\overline{T}_s = \overline{T}_s + \langle t_s^{[m]}, w_s^{[m]} \rangle$
10: **end if**
11: **end for**
12: **for** $i = 1$ to M **do**
13: draw i with probability $\propto w_s^{[i]}$
14: add $t_s^{[i]}$ to T_s
15: **end for**
16: **return** $\mathcal{T}_s$

have a previous set of trees to work from: in other words, the algorithm has been already running for at least one time step. If the set T_{s-1} does not exist, we proceed to create the first set of trees deterministically: we extract the gaps from the laser measurements, and create a list of N identical trees corresponding to those gaps. Each tree is assigned importance weight of 1.

If the previous array of trees T_{s-1} is present, we proceed with the particle filter approach in lines 7 - 9, and sample this posterior to account for all possible changes in the environment according to the standard gap navigation rules presented in Chapter 3. The discussed gap navigation rules are defined by the four gap critical events - gap appearance, gap disappearance, gap split and gap merge that can happen at each time step. In our sampling strategy, we assume that each leaf can either undergo each one of the gap critical events, or it can remain unchanged. Furthermore, for now we assume that only one critical event can happen in a tree for a laser scan measurement, which is the same hypothesis that was declared in the original GNT[1]. Therefore, each tree in the original array produces N different trees, where

[1]Later, in section 4.6 we present a modification to this assumption where we allow multiple gap critical events to happen at one time step.

*N = (number of children in top node) * (5 possible events)*

in a new array of hypothetical sampled trees. As the number of children in top node increases, the number of samples in posterior $\overline{T_s}$ increases proportionally.

A challenge in building an accurate PGNT is the ability to formulaicly determine which critical event happened according to the extracted angular gap positions. Due to the cyclical nature of gap lists, this problem can be compared to the Cyclic Order Preserving Assignment Problem for shape matching, and solved in the fashion presented earlier in section 4.3. The algorithm allows each angular measurement from the previous set of gaps $[g_1...g_n]$ to be matched to each angular measurement from the new set $[g_1^{new}...g_n^{new}]$, returning a set of correspondences between the two sets, including information about multiple matches and unmatched gaps.

The same algorithm is used to determine how likely is each tree sample, as seen by the *correspond* statement in line 8. The results of the correspondence algorithm are used to compare the children of the root node of each hypothetical tree to the new sequence of gap angles and assigns a weight based on how closely the tree matches the observation. The comparison therefore helps to define the probability $p(t_s^{[m]} \mid G_s)$ for that sample by assigning penalty values for cases when the tree does not match the observation. Note that it is multiplied by the probability of the measurement $p(G_s)$, which by the Bayes rule from equation 2.5, which is a distribution from the update step of the particle filter algorithm, assuming the constant η is absorbed.

After the weights for each sample have been determined, the tree array is resampled in lines 12 through 15. The trees with higher weights now appear more frequently in the new distribution of trees T_s. After the resampling step, all the weights are reset to 1 and the algorithm repeats.

The computation described above happens fast enough to be performed while the robot is driving around in real-time when the number of gaps is small and there is only one critical event per time step. Section 4.5 shows another sampling strategy that doesn't sacrifice algorithm performance and greatly prunes the number of samples generated per tree.

The critical innovation in the above approach lies in the probabilities are assigned

to each tree. Therefore, the most likely tree may not have the same number of gaps as the current measurement, depending on the heuristic chosen for determining the weights.

4.5 An Alternative Sampling Strategy

In section 4.4, we have outlined a sampling strategy that gives a complete representation of the tree distribution $\bar{T}_s$. However, when running exhaustive trials several observations about the system were made:

1. As the robot drives around, sensor noise mostly manifests itself in creating false critical events. For example, a gap may split into two for just one or two time steps, and then merge again without a change in the visibility region.

2. The majority of time robot spends there are no critical events and the gaps stay the same for several time steps before and after each true critical event occurs.

3. There was never a time observed when the measurement predicts one critical event while the ground truth dictates another critical event. For example, if the measurements indicate that a merge has happened, ground truth observation never indicated a split or an appearance or a disappearance. This observation is the same for all critical events.

Given these observations and over 200 trials performed with 10 different log files, a new approach to sampling is devised. Since most of the time the robot's visibility region doesn't change, the probability of no change is very high, over 90%. If the correspondence of two gap arrays indicates that the environment has changed through split, merge, appearance or disappearance, the likelihood of those events happening is lower than 20%. So during the sample step of algorithm 4 line 7, if a tree sample is created according to a merge rule at the the first child of its root node while the measurement says there is a split, the likelihood of that tree is virtually 0.

Therefore, to reduce the dimensionality of the representation, the samples with really low probability can be discarded in advance, or simply never created. The new

sampling strategy dictates that to create an optimal representation of the environment assuming only one critical event per time step, it is enough to have two samples in $\overline{T}_s$ for each one in T_{s-1}: one no-change sample that is transferred unchanged from T_{s-1}, and one sample that follows the critical event extracted from the sequence G_s through correspondence. The performance of the algorithm under the new sampling strategy did not show a visible decrease in the number of trees it built correctly.

4.6 Allowing Multiple Critical Events

In the Gap Navigation Tree approach it is assumed that the robot moves slowly enough that only one critical event can potentially happen with each new measurement. However, in our simulations, very often there were two, three and even four and higher critical events with each measurement. Some of these events are a byproduct of the measurement noise, but some of are critical events that correspond to ground truth. The problem of incorporating multiple events into the tree representation requires first, to create a posterior distribution of trees that captures all of the possible critical events that are happening to the tree considering every combination of events; and second, to assign the right probability to each tree sample in order to best approximate the shape of the *belief* in the resampling step of the PGNT algorithm.

Using the correspondence algorithm, section 4.3 describes how to extract potential critical events from the measurement data. In fact, the correspondence vector captures ALL of the critical events that the measurement data suggests. So instead of extracting just one critical event and its angular position, we can store a vector of all the critical events and their corresponding places in the tree.

When it comes to creating a hypothetical tree, the alternative sampling strategy explained in section 4.5 is used. Now that there is more than one potential critical event, one has to consider all of the possible combinations of critical events that can be created from the space of critical events suggested by the measurement. For example, if data suggests that we have a [split, merge, disappearance], then we create a sample for each of these permutations:

- [split]
- [merge]
- [disappearance]
- [split, merge]
- [split, disappearance]
- [merge, disappearance]
- [split, merge, disappearance]

So in this case, one sample of T_{s-1} produces 8 hypothetical trees: 7 with modifications listed above plus 1 unchanged tree. This process is repeated for every tree in T_{s-1} for each iteration of the PGNT algorithm.

Assigning the right probability is a lot harder. For the initial try, the probabilities of the tree with multiple critical events is just a product of the probability for each event. Thus, for a tree with [split, merge] action,

$$p(critical\_event = [split, merge]) = p(critical\_event = split) \cdot p(critical\_event = merge) \tag{4.8}$$

However, because the multiplication involves two decimals much smaller than one, the probability of the tree undergoing multiple events was reduced to the point where the tree did not appear in the final distribution $\mathcal{X}_t$. Therefore, this approach was replaced by the one which assigned a smaller penalty to the trees with multiple events. In the future, there may be a need for a more sophisticated approach to calculating these probabilities.

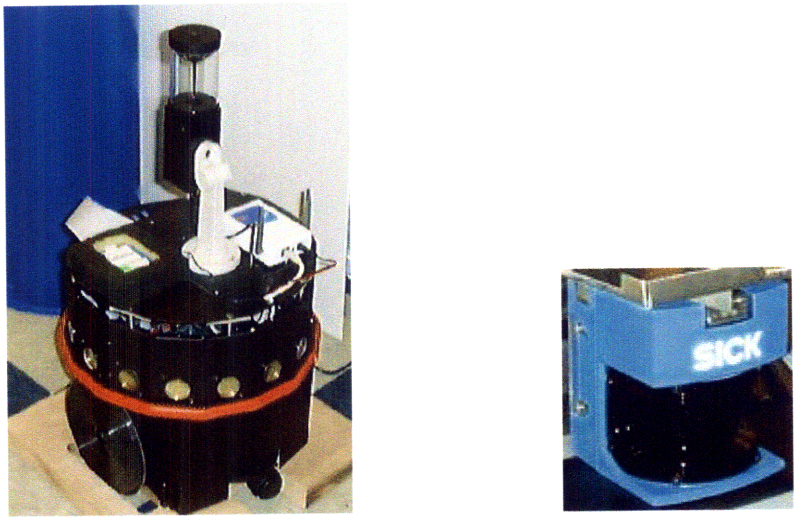
Chapter 5

PGNT Implementation

The purpose of this chapter is to introduce experimental results that were collected to support the theoretical framework for the PGNT introduced in Chapter 4. Section 5.1 describes the experimental setup and the simulations performed to validate the PGNT approach. Section 5.2 compares the results of our experiment to the original GNT and discusses the performance of our algorithm.

5.1 Simulation Experiments

To verify the performance of our algorithm, we have implemented it on a simulated robot using CARMEN, Carnegie Mellon Robotics Navigation Toolkit [24]. CARMEN is an open-source collection of software for mobile robot control. It provides basic navigation abilities including sensor control, obstacle avoidance, localization, path planning, and mapping. It is able to convert an image of a floor plan, or previously collected laser scans, into a map that can be navigated by a simulated robot. We use the toolkit for collecting laser rangefinder sensor data in a specified map, and storing it into a logfile, which is later processed to create a PGNT. The logfile was used to allow for superior debugging capabilities. Another version of the software was created to build the PGNT in real-time without the use of logfile for proof-of-concept. The maps were obtained from the RADISH data set [13], and modified using the CARMEN map editor tool box to remove objects from the middle of the rooms



(a) (b)

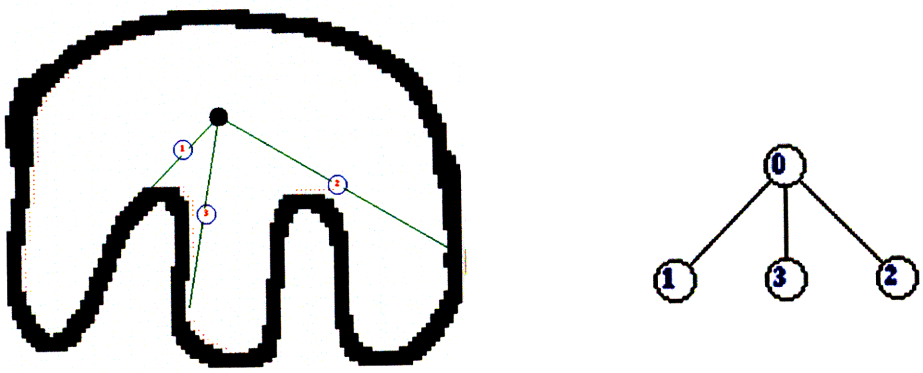
Figure 5-1: (a) Nomad Scout robot. Image courtesy of University of Amsterdam. (b) SICK laser rangefinder. Image courtesy of Czech Technical University of Prague

that would violate the assumption of a simply connected environment.

Inside CARMEN, we utilized a model for the robot "scout" carrying two back-to-back SICK laser rangefinders, enabling a virtual 360° view of the surrounding simulated environment as assumed in the GNT and PGNT formulation. The robotic toolkit creates very realistic simulated models, taking into account actual size, wheel location, laser location and maximum speed of the robot as it moves. The key simulation parameters are listed in Appendix 5.A. The simulated robot has a non-zero radius, so to avoid collisions with walls and ensure continuous data collection, the robot were driven around manually in the environment to mimic gap following.

Figure 5-1 shows a real robot Nomad Scout as well as a SICK laser rangefinder; implementing our software on the hardware platforms and performing experiments is an interesting extension of the current work.

Figure 5-2 shows the implemented graphic user interface that displays a map, and draws the robot position within a map, the red dots representing the laser range measurement positions, and the green lines representing gaps with their appropriate gap names. The GUI on the right in (b) shows the most likely tree that corresponds to the robot's current position in the environment, and the children of the top 0 node



(a) (b)

Figure 5-2: This is a screen shot of two graphic user interfaces. Figure (a) displays the group truth map, with the robot inside(black dot). The red dots represent obstacles that the robot actually sees, and green lines are the gaps. Each gap has a circle and a number on top of it, that indicates the node of the navigation tree in (b) to which that particular gap corresponds.

correspond to the gaps seen on the left.

The bulk of experiments was performed using the map shown in Figure 5-2. Because of its simplicity, it was very easy to deduce the ground truth for what the GNT should be, and to compare it with output of the algorithm. Due to the nature of the environment, even when the robot moved at a high speed there was only one critical event per time step, which simplified the analysis further. The idea for this map was obtained from an example in Tovar et al [41], in which they demonstrate a very similar environment and show exactly how the navigation tree is built. For comparison, that example is reproduced in Figure 5-3.

Several other environments served as our testing platform. One is a floor plan of a building at the University of Freiburg, Germany, obtained from the RADISH dataset [13]. The snapshot of one of the tests is shown in Figure 5-5. The image in 5-5 (a) shows a birds eye view of the floor plan. The image in 5-5 (b) zooms in on the robot building a PGNT inside the environment, along with the tree in (c) that corresponds to the time step captured in (b). Another one is the 5300 corridor of Wean Hall at Carnegie Mellon University in Pittsburgh, which is pictured in Figure 5-4. This is a real, simply connected indoor environment, and it proved to be the main challenge of the probabilistic navigation tree. The associated challenged are described in the

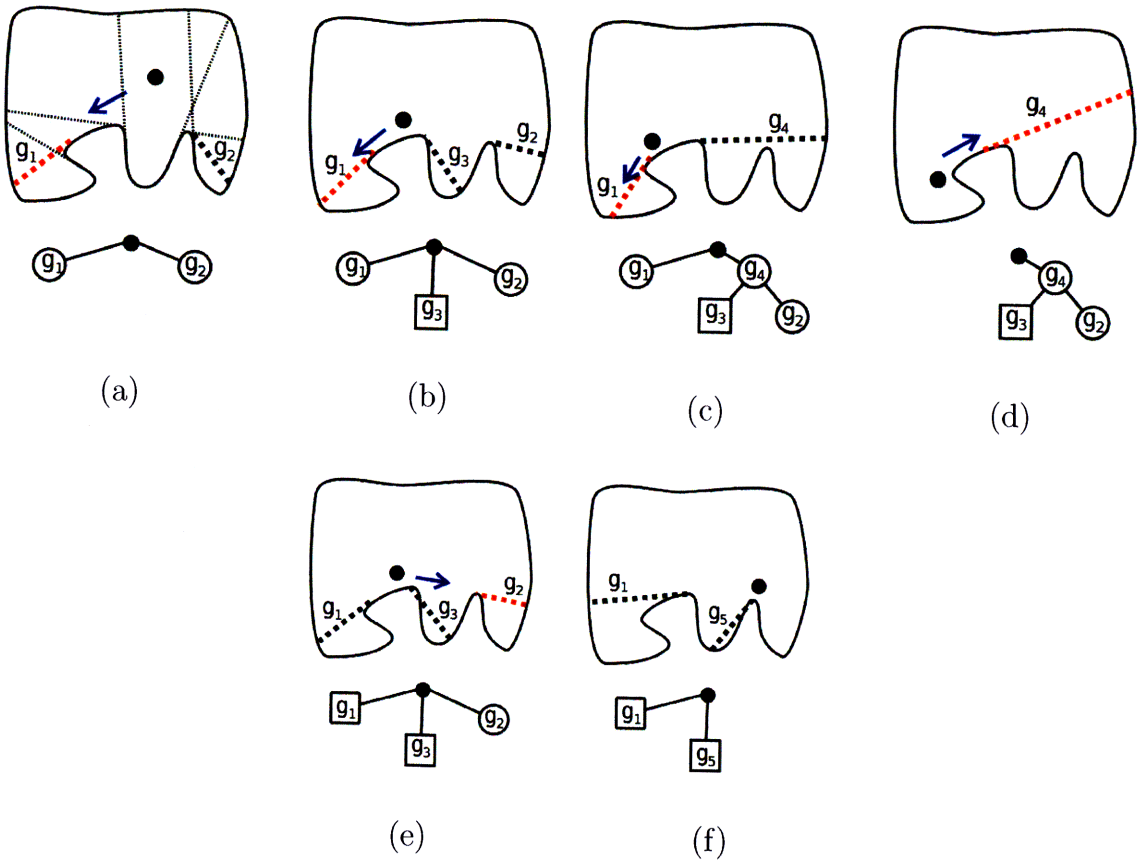


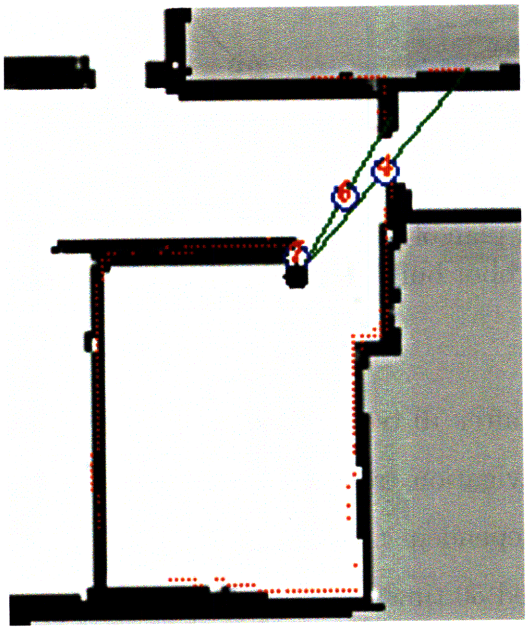
Figure 5-3: The six images illustrate how the navigation tree is build using an idealized environment. Dashed lines geometrically indicate borders of visibility regions, upon crossing which a critical event occurs in the GNT. Adapted from Tovar et al [41]

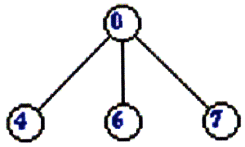
next section.

5.2 Results and Performance

The results show that navigating with the deterministic GNT will always fail to produce the right tree because it doesn't account for the sensor measurement uncertainty, nor for the possibility of multiple critical events within one time step. The PGNT approach proved to be quite successful on the small environment shown in Figure 5-3. We ran simulations for 15 different log files (log files are easier to analyze), which contain laser measurement data for a path of a robot inside the environment that mimics the movements of the robot going through GNT generation. Only 2 of these

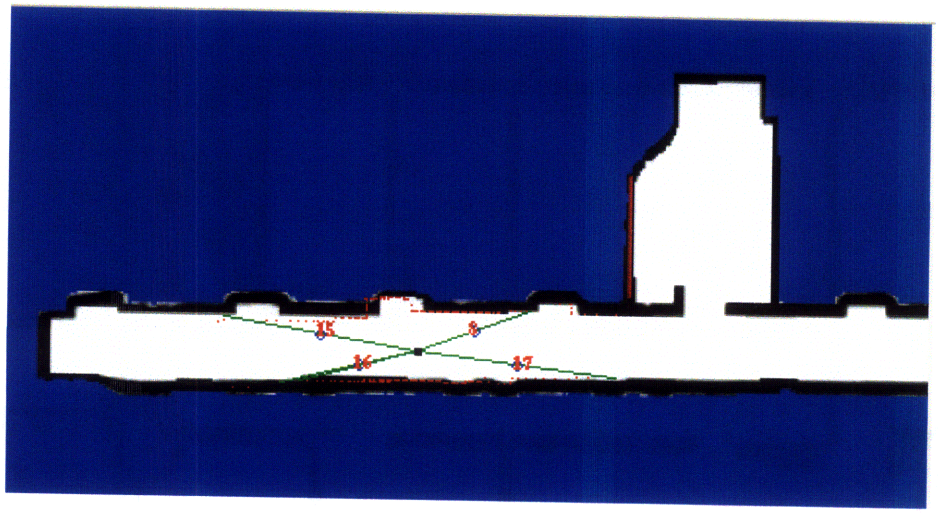
(a)



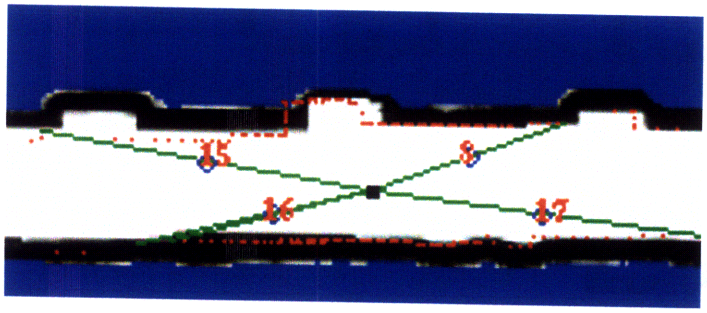


(b) (c)

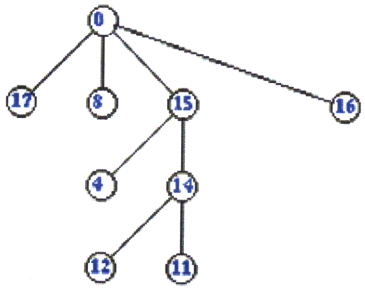
Figure 5-4: These images show (a) the floor plan of the University of Freiburg, (b) a detail insert showing the robot building a PGNT and (c) the PGNT corresponding to the image in (b). Thanks go to Cyrill Stachniss for providing this data



(a)



(b)



(c)

Figure 5-5: These images show (a) the floor plan of the Wean Hall at Carnegie Mellon University, (b) a detail insert showing the robot building a PGNT and (c) the PGNT corresponding to the image in (b)

logfiles failed to generate the correct tree after 10 trials. Two of the logfiles gave a 90% success rate of building a correct navigation tree, while the rest have built a tree with a 100% success rate after 10 independent trials. The toughest logfile that contained over 450 time steps was simulated 50 times with an 80% success rate.

While attempting to build a full tree in the University of Freiburg environment, we have encountered several challenges. The number of gaps appearing at each time step has increased, which lead us to implement a more efficient sampling approach for the navigation tree to still allow the algorithm to run in real time. As mentioned earlier, due to the very sharp angles present in the environment, often there were multiple

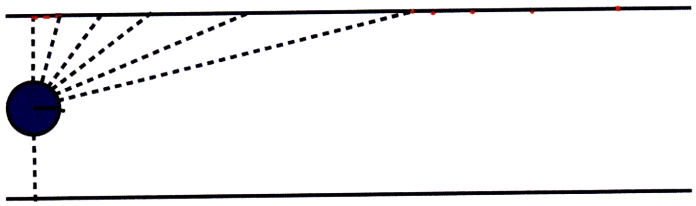


Figure 5-6: Angular resolution as a function of range

critical events present. How we approached the problem of handling multiple events is described in Section 4.6. However, the solution we found is not sufficient because it often leads to incorrect environment representation when the multiple events occur.

Another significant problem encountered in our experiments is the limited resolution of the laser rangefinder when resolving walls far away, and when close to the wall. This concept is illustrated in Figure 5-6. The robot (in blue) is positioned at the end of a long hallway, and the signals from the rangefinder are shown as dashed lines. Close to the robot (right above it), there are a lot more measurements per angle subtended than are far away on the right. At far enough distances, the difference between consecutive range measurements can become large enough to indicate a presence of a gap when there is done. Therefore, when using a large enough detection range , a lot of false gaps are detected that are currently not being processed by the PGNT in the right fashion. This shortcoming is illustrated in Figure 5-7, where the robot is located at an end of a hallway, and the SICK range is set to be 15m.

One solution to this problem is to limit the range that we consider for gap navigation, thus discarding all distance information above a certain range, lets say above 5 m. Now, the only detected gaps are the ones in the near proximity to the robot and therefore are a lot more likely to be real, or at least follow the probabilistic navigation rules derived in Chapter 4. However, an additional problem arises due to the robot not able to see all of the gaps it should see. As the robot moves, more gaps come into view and they are counted as appearance events. The gaps that are beyond the limited range of the sensor are counted as disappearance event, and by using the critical event metric they are deleted from the tree, whereas normally they would eventually merge into other gaps. Recall that the power of the navigation tree comes

from following branches to the goal, and if even one branch disappears the environment representation will be incomplete. There is no good heuristic at this moment to distinguish between a real gap disappearance, which happens when a hidden part of environment becomes fully visible, or a gap disappearance due to the gap located outside of laser rangefinder field of view. Therefore, using the range limited laser rangefinder, although the algorithm is capable of building well defined local trees, it is unable to fuse them into a bigger picture due to the disappearance from the tree.

In spite of these shortcomings, the PGNT still demonstrated an improvement of the GNT by mitigating sensor noise. Figure 5-8 shows three different time steps, in which sensor noise removed and then inserted the same gap into the tree. Under the deterministic approach, the original gap label would be lost, and replaced by a new one. While labeling is not a problem, removing a gap from a tree also removes all of its children that carry important information about traversing the tree.

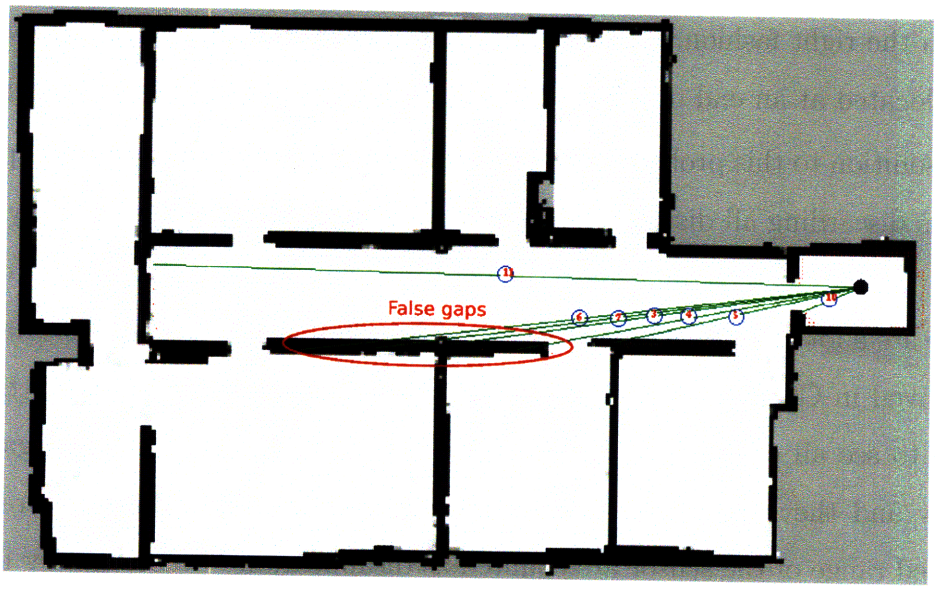
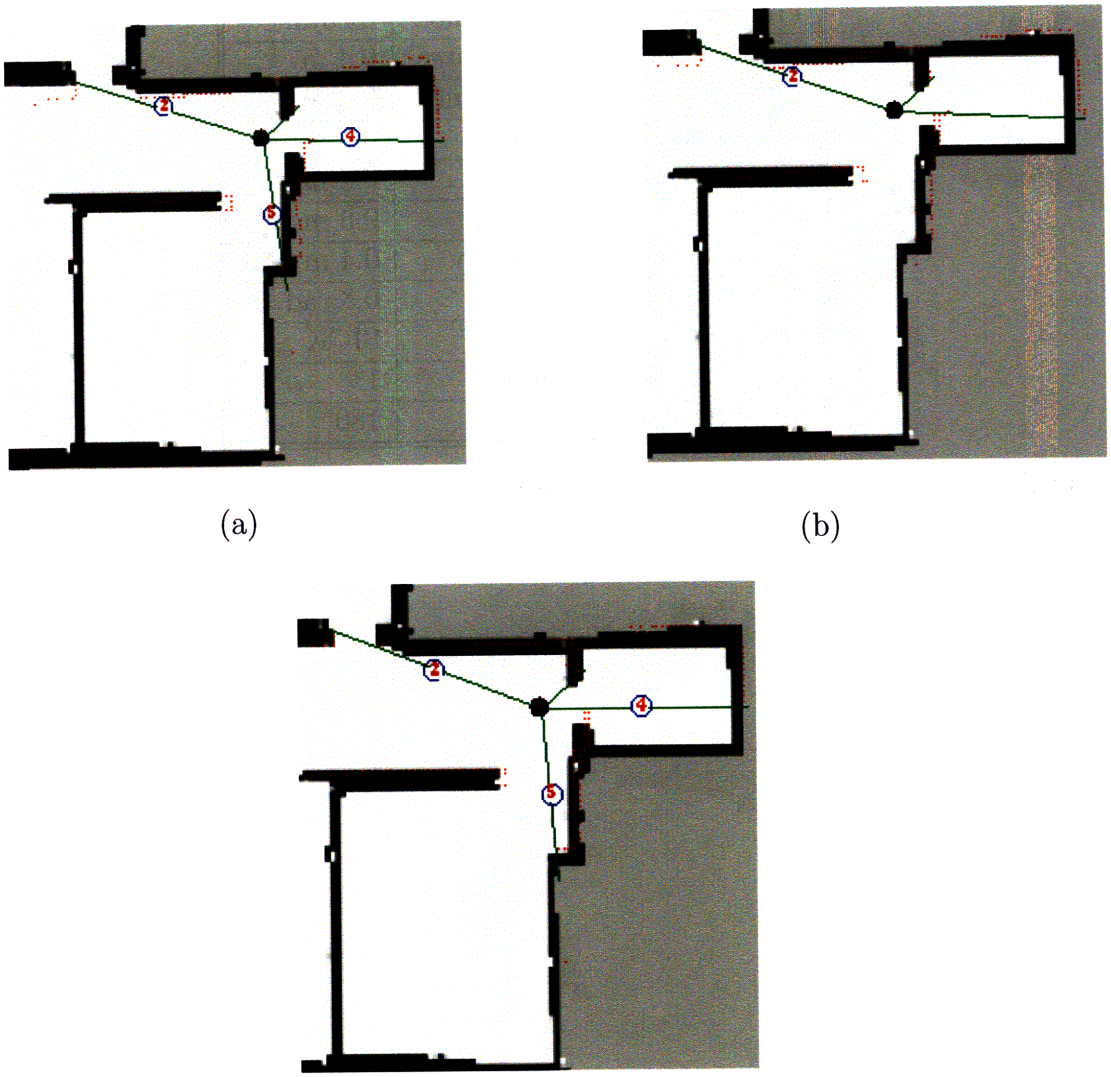


Figure 5-7: False gaps in the University of Freiburg environment



(a) (b)

(c)

Figure 5-8: These images show how the tree moves through the environment at three different time steps, (a) t_1, (b) t_2 (c) t_3. Gap 5 visible in t_1 disappears, and later reappears in the same place without causing a critical event in the tree.

5.A Parameters for Scout and SICK laser

Robot Parameters	
Length	0.6 m
Width	0.46 m
Front laser offset from center	0.25 m
Rear laser offset from center	0.4 m
Minimum approach distance	0.3 m
Minimum side distance	0.1 m
Acceleration	0.2 m/s^2
Deceleration	0.5 m/s^2
Max translational velocity	0.4 m/s^2
Max rotational velocity	0.8 rad/s^2
Laser Type	SICK
Laser angular resolution	1°
Laser field of view	180°

Table 5.1: This is a list of essential robot and laser parameters as they are modeled in CARMEN

Chapter 6

Conclusions

The purpose of this chapter is to summarize the key contributions of this thesis and to describe the areas where additional research will address some of the shortcomings of the current approach and expand the areas of interest. Section 6.1 provides an overview of our thesis, while section 6.2 suggests future directions for research.

6.1 Summary

In this thesis, we tried to address an issue of creating a feasible environmental representation for a robot with non ideal sensors. As a basis of environment presentation we borrowed the Gap Navigation Tree, an approach developed by Tovar et al. which creates a tree structure based on the discontinuities in the robotic field of vision and which is thoroughly discussed in Chapter 3. At the heart of the GNT is its assumption that the robot has a perfect detector for measuring discontinuities, which is not true in the real world.

Our work demonstrates the utility of applying probabilistic principles when dealing with noisy measurements in critical sensors available for navigation. We created a probabilistic representation of the navigation environment based on the particle filter that is described in Chapter 4. Furthermore, we utilize a solution to a cyclic order preserving assignment problem to define the evolution of the navigation tree particles, as well as to calculate the weight for each tree. Chapter 5 describes experiments that

were performed to validate our approach and discusses how the results can be used in future work.

6.2 Directions for Future Research

Chapter 5 discusses difficulties encountered during implementation that need to be resolved for the PGNT to be a viable approach for navigation in realistic indoor environments. Firstly, a reliable approach to navigation when there are multiple critical events happening at each time step is needed. Right now, it is not clearly understood how the probability distribution of trees changes when multiple events are present, and what probabilities should be assigned to the trees produced from subset of multiple actions.

Secondly, the success of the PGNT implementation relies on the parameters for gaps to be finetuned for each environment to produce good navigation results. Each environment has corridors with unique widths and door sizes, the gap threshold has be set such that all the openings and doors in the environment can be recognized as gaps to enable the robot to navigate towards the rooms and enter them. An interesting problem is to create a way for the robot to learn these critical parameters as it is building the navigation tree, or perhaps to figure out a formula by which those parameters can be adjusted automatically for any environment.

Thirdly, as discussed in Chapter 5, as with all non ideal systems, we are also limited by the resolution of the laser rangefinder. There needs to be a reliable way to distinguish true gaps from the gaps created due to the rangefinder being far away, and therefore creating false gaps. The already suggested approach of deciding if a gap is "real" or not by looking at its neighbors has a potential of discarding true gaps that happen to be in the region of poor angular resolution, and thus should be evaluated further.

A robust wall following approach should be implemented to make this a truly autonomous system, and enabling future experiments not only with a variety of indoor environments, but also on a hardware platform such as the one pictured in Figure

5-1. Tovar et al. have performed experiments on a hardware platform, but they constructed a fake environment as opposed to navigating in a natural indoor one. This work also does not address navigation with a more limited laser rangefinder than an expensive SICK laser that has up to 80 m range. For example, the Hokuyo laser rangefinder has a much more limited range (reliable only to about 3 m), but it has a high angular resolution. A natural extension of the work would be to prove if the gap navigation is possible with a limited range laser, perhaps relying on approaches such as coastal navigation [32].

In their most recent work, Tovar et al [41] have extended their navigation tree approach to multiply connected environments. There may be a way to generalize the PGNT to the multiply connected environments as well, which would extend its utility even further.

The probabilistic gap navigation tree approach presents a navigation challenge. Since the most likely tree may not always represent what the robot sees with its laser range finder, the previous heuristic for the robot navigation needs to be modified. Recall that in the previous approach, the robot follows the gap until it detects a gap critical event. With the probabilistic gap navigation tree, a gap critical event, even if true, doesn't become incorporated into the most likely tree until several time steps, or laser scans, later. A heuristic for navigation that only relies on the most likely tree may not be the most efficient one, and therefore it could be an interesting problem to solve in the future.

Although there are many current proposed approaches to solve the navigation, mapping and localization problem in two dimensions, few have attempted to deal with robotics in a 3-D space. An important future research direction could be in the development an autonomous helicopter that is capable of solving localization and navigation problems in a multi-dimensional environment using only an onboard 2D-laser rangefinder as a guide, without relying on GPS. Autonomous indoor air vehicles are a good platform for minimal sensing navigation, since they have an inherent limitations in payload, which translates few sensors and limited on board data processing abilities.

Bibliography

[1] Ercan Acar and Howie Choset. Complete sensor-based coverage with extended-range detectors: A hierarchical decomposition in terms of critical points and voronoi diagrams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1305 – 1311, October 2001.

[2] Aronov, Guibas, Teichmann, and Zhang. Visibility queries in simple polygons and applications. In *ISAAC: 9th International Symposium on Algorithms and Computation*, 1998.

[3] Kristopher R. Beevers. *Mapping With Limited Sensing.* PhD thesis, Rensselaer Polytechnic Institute, June 2007.

[4] H Bulata and M Devy. Incremental construction of a landmark-based and topological model of indoor environments by a mobile robot. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1054–1060, April 1996.

[5] Wen-Chung Chang and Shu-An Lee. Real-time feature-based 3d map reconstruction for stereo visual guidance and control of mobile robots in indoor environments. In *IEEE International Conference on Systems, Man and Cybernetics*, 2004.

[6] Howie Choset, Ilhan Konukseven, and Alfred Rizzi. Sensor based planning: A control law for generating the generalized voronoi graph. In *IEEE Int. Advanced Robotics*, Washington, DC, 1996.

[7] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22:46–57, June 1989.

[8] Michael Erdmann and Matthew T. Mason. An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4(4):369–379, 1988.

[9] Lawrence Erickson, Joseph Knuth, Jason M. O'Kane, and Steven M. LaValle. Probabilistic localization with a blind robot. In *IEEE International Conference on Robotics and Automation*, May, 2008 2008.

[10] Jared Glover, Christian Uldall Pedersen, and Erik Taarnhoj. Solving the cyclic order preserving assigment problem. Advanced Algorithms (MIT Course 6.854) Final Project, December 2006.

[11] Dirk Hähnel, Wolfram Burgard, and Sebastian Thrun. Learning compact 3d models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems*, 44:15–27, 2003.

[12] A. Harten, S. Osher, B. Engquist, and S. R. Charkavarthy. Some results on uniformly high-order accurate essentially nonoscillatory schemes. *Applied Numerical Mathematics*, 2:347 – 378, 1986.

[13] Andrew Howard and Nicholas Roy. The robotics data set repository (Radish), http://radish.sourceforge.net, 2003.

[14] Wesley H. Huang and Kristopher R. Beevers. Topological mapping with sensing-limited robots. *Workshop on Algorithmic Foundations of Robotics*, 2004.

[15] Ishay Kamon, Elon Rimon, and Ehud Rivlin. A new range-sensor based globally convergent navigation algorithm for mobile robots. In *IEEE Conference on Robotics and Automation*, 1996.

[16] Ishay Kamon and Ehud Rivlin. Sensory based motion planning with global proofs. In *IEEE Transactions on Robotics and Automation*, volume 13, pages 814–822, December 1997.

[17] Sven Koenig and Reid G. Simmons. Unsupervised learning of probabilistic models for robot navigation. In *IEEE Conference on Robotics and Automation*, pages 2301–2308, Minneapolis, Minnesota, April 1996.

[18] Benjamin Kuipers. Modeling spatial knowledge. In *Cognative Science*, pages 129–153, 1978.

[19] Benjamin Kuipers and Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:47–63, 1991.

[20] Yanina Landa, David Galkowski, Yuan R. Huang, Abhijeet Joshi, Christine Lee, Kevin K. Leung, Gitendra Malla, Jennifer Treanor, Vlad Voroninski, Andrea L. Bertozzi, and Yen-Hsi R. Tsai. Robotic path planning and visibility with limited sensor data. *American Control Conference*, pages 5425–5430, July 2007.

[21] Yanina Landa, Richard Tsai, and Li-Tien Cheng. Visibility of point clouds and mapping of unknown environments. *Advanced Concepts for Intelligent Vision Systems*, 4179, October 2006.

[22] John J. Leonard and Hugh F. Durrant-Whyte. Simultaneous map building and localization of an autonomous mobile robot. *IEEE/RSJ International Workshop on Intelligent Robots and Systems*, pages 1442–1467, November 1991.

[23] V.J. Lumelsky and A.A. Stepanov. Path planning strategies for a point mobile automation moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[24] M. Montemerlo, N. Roy, S. Thrun, D. Haehnel, C. Stachniss, and J. Glover. Carnegie mellon robot navigation toolkit (CARMEN), http://carmen.sourceforge.net.

[25] Elizabeth Murphy and Paul Newman. Using incomplete online metric maps for topological exploration with the gap navigation tree. In *IEEE Conference on Robotics and Automation*, Pasedena, CA, May 2008.

[26] Jason O'Kane and Steven M. LaValle. Almost sensorless localization. In *IEEE International Conference on Robotics and Automation*, pages 3764–3769, Barcelona, Spain, April 2005.

[27] Yaron Rachlin, John M. Dolan, and Pradeep Khosla. Efficient mapping through exploitation of spatial dependencies. In *International Conference on Intelligent Robots and Systems*, 2005.

[28] Stjepan Rajko and Steven LaValle. A pursuit-evasion bug algorithm. In *International Conference on Robotics and Automation*, Seaul, Korea, May 2001.

[29] A. Ranganathan and F. Dellaert. A rao-blackwellized particle filter for topological mapping. In *IEEE Conference on Robotics and Automation*, May 2006.

[30] Malvika Rao, Gregory Dudek, and Sue Whitesides. Minimum distance localization for a robot with limited visibility. In *IEEE International Conference on Robotics and Automation*, pages 2438–2445, Barcelona, Spain, April 2005.

[31] Emilio Remolina and Benjamin Kuipers. Towards a general theory of topological maps. *Artificial Intelligence*, 152:47–104, 2004.

[32] Nick Roy, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. Coastal navigation - mobile robot navigation with uncertainty in dynamic environments. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 35–40, Detroit, MI, May 1999.

[33] Clayton Scott and Robert Nowak. Robust contour matching via the order-preserving assigment problem. *IEEE Transactions on Image Processing*, 15(7), July 2006.

[34] Hagit Shatkay and Leslie P. Kaebling. Learning geometrically-constrained hidden markov models for robot navigation: Bridging the topological-geometrical gap. *Journal of Artifical Intelligence Research*, 16:167–207, March 2002.

[35] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[36] Sebastian Thrun. Robotic mapping: A survey. Technical report, Carnegie Mellon University, February 2002.

[37] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. In *IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 2001.

[38] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[39] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence Journal*, 2001.

[40] Benjamin Tovar, L. Guilamo, and Steven M. Lavalle. Gap navigation trees: A minimal representation for visibility based tasks. In *Workshop on Algorithmic Foundations of Robotics*, page 1124, 2004.

[41] Benjamin Tovar, Rafael Murrieta-Cid, and Steven M. LaValle. Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Transactions on Robotics*, 23:506–518, June 2007.