Binary Directional Marker Placement for Mobile Robot Localization

by

River Allen
B.Sc., University of Victoria, 2011

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Binary Directional Marker Placement for Mobile Robot Localization

by

River Allen
B.Sc., University of Victoria, 2011

Supervisory Committee

Dr. Sue Whitesides, Co-Supervisor
(Department of Computer Science)

Dr. Mantis Cheng, Co-Supervisor
(Department of Computer Science)

Supervisory Committee

ABSTRACT

This thesis looks at the problem of optimally placing binary directional proximity markers to assist a robot as it navigates waypoints through an environment. A simple planar fiducial marker is developed to serve as the binary directional proximity marker. A scoring function is proposed for marker placement as well as a method for random generation of hallway maps. Several common metaheuristic algorithms are run to find optimal marker placements with respect to the scoring function for a number of randomly generated hallway maps. From these results, placements are then evaluated by physical experimentation on an iRobot Create equipped with relatively inexpensive webcams.

ACKNOWLEDGEMENTS

I would like to begin by thanking my supervisor Dr. Sue Whitesides. Sue has mentored me throughout the thesis process, imparting valuable life knowledge and wisdom along the way. She has been the best supervisor a student could ask for. It has been an pleasure and an honour to be Sue's student and I do not think I will ever be capable of putting into words just how thankful I am for the opportunity.

I would also like to thank my co-supervisor, Dr. Mantis Cheng who sparked passion and interest in this subject. I have learned an unquantifiable amount from Dr. Cheng during my time at UVic. Without his incredible ability to teach and inspire, I would not be here.

I especially want to thank Dr. Dimitri Marinakis for his immense knowledge and for his encouragement during my graduate studies. Additionally, I thank Dr. Junaed Sattar for his valuable commentary in his role as external examiner.

My family has always been vital to my success. During this thesis they have continued to give me the constant love, support and motivation that I will always be grateful for. The recent introduction of my niece and nephew into our family have been of great significance during this time.

Last but not least, I would like to thank my CGAR lab mates and personal friends over the years. I give special thanks to Nishat who has been an incredible friend and collaborator throughout this period.

Without the generous funding from NSERC sources, Howard E. Petch research scholarship and various UVic scholarships this thesis would not have been possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As computers systems have become more and more ubiquitous in automating everyday tasks, automation is rapidly accelerating into the physical world in the form of robotics. In 2012, NASA successfully autonomously landed the *Curiosity* rover on the surface of Mars. The mission was a huge feat because of the difficulty involved with decelerating and landing in the Martian environment the 899 kg rover payload which was equipped with sensitive scientific equipment. In the same spirit, Google is conducting the *Google Lunar XPrize* where several commercial teams are competing to land rovers on the Moon's surface by 2016 for $30 million dollars in prizes. The vast improvements of and access to robotics are not limited to space, with new robotic systems constantly being built and improved for ocean floor mapping, deep sea exploration and observing, search and rescue, self-driving cars and automated warehouses to name a few applications. Consumer robotic products have also arisen, robot vacuum cleaners, toys and the vast growth of quadcopters due to the large hobbyist DIY (do it yourself) drone movement.

This growing push of interest in robotics applications requires constant innovation on all fronts in the field of robotics. One aspect of robotics is the subfield of mobile robotics, which is concerned with the problem of effectively moving a robot around an environment. This is especially of importance when dealing with robotic systems that are designed for autonomous or semi-autonomous control. A requirement of autonomous and semi-autonomous control is that a robot be able to know its current position with some level of accuracy. The problem of a robot determining its current position and orientation (*pose*) is known as *localization.* There have been great strides taken in localization, but with all these advancements there continues to be a need for developing methods that allow a robot to be able to localize using simpler and lower cost solutions.

The ideas of this thesis are inspired by earlier work to place directional ultrasonic beacons [6] to assist a robot in localization. This thesis sets aside directional ultrasonic beacons to look instead at the notion of optimally placing what we call *binary directional proximity markers* in an environment to provide localization capabilities to a robot as it travels along waypoints.

A binary directional proximity marker can be detected by a robot when the robot is within what we call the *visibility region* or *field of visibility* of the marker. In this thesis, our markers are implemented as a crude planar fiducial marker system (signs printed on paper that a camera can detect). Hence, the visibility region for this type of marker is a function of the robot's distance from the marker and the robot's direction vector to the marker with respect to the outward normal of the marker.

Throughout this thesis, we will refer to related research in the problem of sensor placement. While a marker and sensor are obviously different, they share certain similarities. A sensor has a field of view described by its sensing range. With a binary directional proximity *sensor* its field of view or range determines whether a target is detected. Similarly, a *marker* has a field of visibility or visibility region such that, when a robot is within the visibility region, the robot can detect the marker. Whether the binary directional proximity sensor detects a robot that has entered its field of view or the robot detects the binary directional proximity marker in the marker's visibility region, the localization information gained is determined by the region of the field of view or visibility region, respectively. We recognize that markers and sensors are not exactly the same, but we believe previous sensor placement research is relevant. As such, this thesis will focus on the placement of markers, but some of the ideas presented may be possible to translate over to binary directional proximity sensor research.

One benefit given by the use of markers is that they do not consume power unlike most active sensors. As a result, this thesis does not require dealing with the issue of minimizing power consumption, in contrast to other sensor placement research.

To measure the effectiveness of a given placement of markers an intuitive scoring function is proposed. Using this scoring function, we run several common metaheuristic algorithms to find "good" placements for a specified number of markers on randomly generated hallway maps. Finally, the "good" metaheuristic placements using the proposed scoring function are tested on a physical robot in an actual office hallway. The results of this experimentation are then discussed.

1.1 Problem Overview

Now we give a high-level description of the overall problem. The problem is defined formally as:

Given an environment map, a desired robot path described by a sequence of waypoints, and a discrete collection of possible directional marker poses, what is a *good* configuration of these marker poses? Here, we assume that there are only k markers and that we want to minimize the deviation of the robot from each waypoint as it travels along the sequence of waypoints.

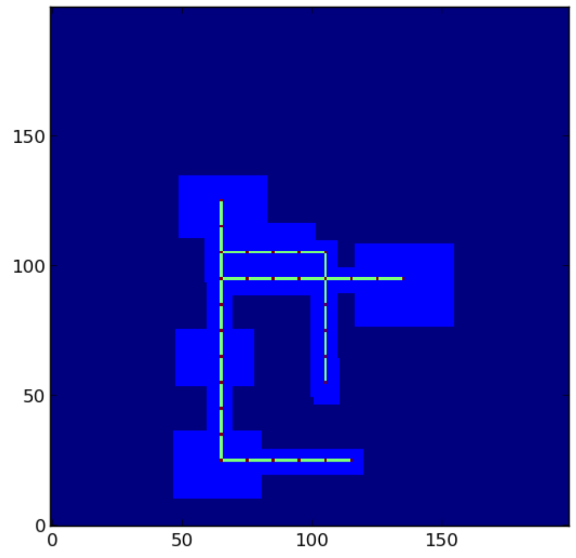


Figure 1.1: An example of an office environment a robot could traverse through. Each pixel is a decimeter^2. The dark blue pixels represent area that is non-navigable. Red pixels represent the path the robot is expected travel. Possible marker poses for this map are provided and are shown as orange pixels; in this case they correspond to corners.

In this thesis, an environment map, G, is a binary, 2D, $m \times n$ occupancy grid where each 0 grid square represents non-navigable area (usually a wall) and each 1 grid square represents free-space or navigable area. For simplicity, we represent a path as sequence of discrete 2D waypoints that a robot will travel along. The waypoints lie in navigable area in the environment map.

Localization is the process of determining a robot's true position and orientation (or *pose*). The localization problem comes in three general flavours: robot tracking or local localization, global localization, and the "kidnapped robot" problem. The *robot* *tracking problem* is the well researched topic of tracking a robot's position as it moves around an environment where some belief of its initial position is known. How the robot moves through the environment is usually known *a priori* and modeled with some error, where this error is mitigated using markers to aid in the control of the robot. The *global localization problem* is the problem of localizing a robot within an environment without any prior belief of its position – that is, all initial positions in the environment are equally likely. The *kidnapped robot problem* is a combination of robot tracking and global localization: a robot travels around an environment, but can be "teleported" to another location. The problem then becomes one of determining the robot's position given its sudden relocation to new surroundings. A related area of research is the problem of moving a robot through an environment with little or no *a priori* knowledge of the environment. Thus, a robot must map the environment as it moves through it, and its current localization state will inform mapping. This then feeds back into the map which will inform localization, if the robot returns to mapped areas. This well known problem is called *Simultaneous Localization and Mapping (SLAM)*. For more on the subject, the reader is directed to Thrun et al. [7].

This thesis will focus on robot tracking. Any possible extensions of this work to global localization and/or the kidnapped robot problem are left for future work.

Just as humans may use their eyes, ears, sense of balance and touch to navigate their surroundings, so too must a robot use sensor information to localize itself. There are many sensor systems that exist to accomplish localization (or just position estimation), one of the most popular being the Global Positioning System (GPS). Although GPS is practical in many applications, it has the detriment of not being able to function in several key environments: underwater, underground, indoor and areas on an extraterrestrial surface not equipped with the necessary and complex satellite infrastructure. GPS also has a position error bound of several meters when used in civilian applications on Earth, which may be more error than certain mobile robotics problems can tolerate.

The sensor focused on in this thesis is the camera which will be used to detect *binary directional proximity markers*. This means a marker is associated with a binary output (0 or 1) that is determined by whether the robot is within a visibility region of the marker (1) or is outside the visibility region (0). In order for our system to work, the robot must have an omni-directional field-of-view. An additional assumption is that each marker is uniquely identifiable. In this thesis, the markers are *planar fiducial markers* fixed to flat upright surfaces. Fiducial markers are artificial visual landmarks placed in an environment to assist an agent in some task. For our purposes, the agent is a robot, and the task is mobile robot localization. Our fiducial markers are planar for reasons discussed below.

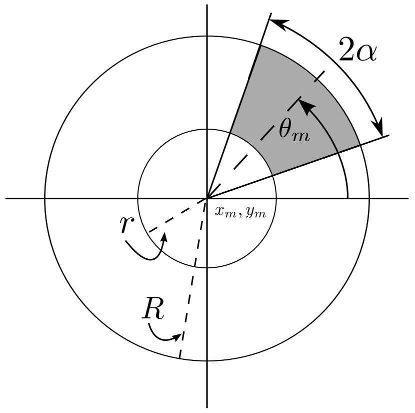


Figure 1.2: A 2D annulus. The grey shaded area represents an annulus sector.

The visibility region for the fiducial marker is represented by a 2D annulus sector (see Figure 1.2) that is described by a 6-parameter vector: $< x_m, y_m, \theta_m, \alpha, r, R >$. The first three parameters $< x_m, y_m, \theta_m >$ describe the marker's pose while the last three parameters $< \alpha, r, R >$ describe the sector angle α, the minimum detection range r and the maximum detection range R. The θ_m of the marker corresponds to the yaw of the marker, where the the marker's pitch is fixed upright and roll is fixed. For our problem, α is generally assumed to satisfy $0 < \alpha \leq \pi$ as any visual detector behind the plane of the planar marker will be assumed incapable of seeing the front side of the marker. Testing if a point p is within the annulus sector can be done relatively efficiently by first converting the test point $p =< x_p, y_p >$ to polar coordinates origined at the marker's position $< x_m, y_m >$:

$$r_p = \sqrt{(x_p - x_m)^2 + (y_p - y_m)^2}$$

$$\theta_p = \arctan(y_p - y_m, x_p - x_m).$$

The transformation to polar coordination yields the new vector $< r_p, \theta_p >$. The binary function that indicates whether a point is in the annulus sector can then be determined using the following function:

$$f(r_p, \theta_p) = \begin{cases} 1 & \text{If } r \leq r_p \leq R, |\theta_p - \theta_m| \leq \alpha \\ 0 & \text{Otherwise.} \end{cases}$$

The main focus of this thesis is the problem of placing these planar fiducial markers in a "*good*" way so as to minimize robot error as the robot traverses a path through an environment. The use of "*good*" is intentionally vague to encompass the fact that an appropriate formal definition of the type of error to be minimized is task dependent. There can be many trade-off issues, and the solution found will likely be non-ideal, but acceptable. This optimization process is explored through the simulation of various metaheuristic optimization techniques. These simulation results are then used to inform physical localization experimentation with a robot in an actual office environment which will in turn validate the findings of the simulation.

1.2 Outline

This section briefly reviews the intent and content of the remaining chapters.

In Chapter 2, we discuss related work, motivation and applications of this work to provide context for the reader.

In Chapter 3, we delve into more details about the problem and discuss some of the practical challenges in passing through the threshold from theoretical models to physical reality.

In Chapter 4, we look into the simulation aspects of the thesis. Different metaheuristic algorithms are described and evaluated to demonstrate their effectiveness at marker placement. The metaheuristic algorithms are evaluated on random maps of hallway environments; the process for generating these random maps is also detailed.

In Chapter 5, we explore evaluating marker placements in an actual physical office environment and detail the software and hardware involved to accomplish these experiments.

In Chapter 6, we present a summary of the research and results presented in this thesis as well as reflect on possible future work.

1.3 Contributions

This thesis explores a number of various aspects-of-interest, but its main contributions can be summarized by the following:

1. Evaluation in simulation and experimentation of various metaheuristics under different parameters for the directional sensor placement problem.
2. An original computational method, together with source code, for generating random 2D office maps.
3. A Python library for encoding and decoding a simple planar fiducial marker: *redcomet*.

Chapter 2

Related Work

In this chapter we will provide background on related work to the problem as well as describe possible applications.

2.1 Directional Sensor Placement

Sensor networks, especially wireless sensor networks, encompass a major area of research because of their numerous applications. Wireless sensor networks can involve a variety of sensors including infrared, sonar, radio and light sensors. These sensors tend to be *active*, that is, they require a power source, which can enable them to have greater range and intelligence. Sensors or markers that are not powered are *passive* and include passive *Radio Frequency Identification* (RFID) [8], light reflectors [9] and fiducial markers, for example. The sensors that form sensor networks are then used to accomplish a variety of useful tasks such as localization, environment sensing, and surveillance to name a few. One aspect of sensor networks is choosing the location and orientation of the sensors. In some research, sensors are distributed randomly or uniformly about an environment, irrespective of cost and effective coverage. Other research, and the focus of this thesis, deals with the placement of sensors to best facilitate the effectiveness of the sensors for some set of tasks. There is a large body of literature on sensor placement with omni-directional sensor coverage, wherein sensor coverage areas are usually modeled as circles or spheres. The focus of this thesis does not consider these problems, but instead focuses on *directional sensors*.

When the task of the system that uses the sensors is unknown, the notion of an effective sensor placement tends to be defined by a sensor placement that covers an environment or target points in an environment. When dealing with only directional sensors, this is called the *directional sensor coverage problem.* A relatively recent survey on the *directional sensor coverage problem* is that of Guvensan and Yavuz [10] and serves as a good introduction to the field. One of the original papers on directional sensor placement with randomly deployed sensors is that of Ai and Abouzeid [11]. In the paper, the authors define the *Maximum Coverage with Minimum Sensors (MCMS)* problem, whereby sensors are randomly placed in an environment and the goal is to orient them to cover target points. For this problem, sensors have five parameters: position (x, y), orientation, sector angle and radius. These parameters define a sector (see Figure 2.1). The MCMS problem is defined as follows (taken from [11]):

"***Given:*** A set of targets $\mathcal{S} = \{s_1, s_2, ..., s_m\}$ to be covered; a set of n homogenous directional sensors, each of which has p possible orientations, randomly deployed in a two-dimensional plane. Hence, a collection of subsets $\mathcal{F} = \{\Phi_{ij}, 1 \leq i \leq n, 1 \leq j \leq p\}$ can be generated based on the TIS test[1], where each element Φ_{ij} is a subset of $\mathcal{S}$."

"***Problem:*** Find a subcollection $\mathcal{Z}$ of $\mathcal{F}$, with the constraint that at most one Φ_{ij} can be chosen for the same i, to maximize the cardinality of the union of chosen $\cup_{(i,j)}\Phi_{ij}$ (i.e., the number of covered targets) while minimizing the cardinality of $\mathcal{Z} = \{\Phi_{ij}, (i, j) \text{ is chosen}\}$ (i.e., the number of activated directional sensors)."

Ai and Abouzeid prove the MCMS problem to be $\mathcal{NP}$-Complete by converting MCMS to the $\mathcal{NP}$-Complete *MAX_COVER* problem [12] in polynomial time. In addition, they provide an Integer Linear Programming (ILP) formulation for the problem that leads to an optimal exponential time solution, which does not scale well. For large target coverage problems, Ai and Abouzeid also describe two greedy algorithms: a centralized greedy algorithm and a distributed greedy algorithm. The distributed greedy algorithm is scalable as it allows sensors to configure themselves amongst local neighbors without the need for a central hub. Their simulations show that the distributed greedy algorithm gives comparable coverage results to the centralized greedy approach, but uses more sensors.

Fusco and Gupta build on the work of Ai and Abouzeid in [13], describing the *k-coverage maximization problem.* In k-coverage maximization, the problem is the same as MCMS except that it is not enough to cover a target once – it must be covered k times, where $k \geq 1$. They, too, describe a greedy approach for finding the

[1]"TIS" means "Target In Sector", and the TIS test evaluates whether a given point is in a given sector.

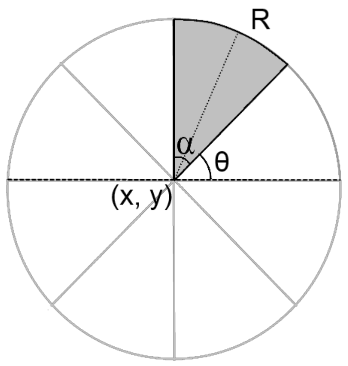


Figure 2.1: The typical model for a sector-shaped directional sensor defined by the five parameters: $(x, y, \theta, \alpha, R)$. This differs slightly from our own visibility region model where α would be $\frac{\alpha}{2}$ in their model.

minimum number of sensors needed and prove that the greedy algorithm will k-cover at least half the targets.

Wu et al. [14] also look at the k-coverage directional sensor placement problem, but from the point-of-view of a *probabilistic sensor model.* There are two types of sensor models: binary and probabilistic. In a binary sensor model, if a target is within the coverage region of the sensor it is regarded as detected; otherwise, it is not detected. (Note that, this thesis assumes a binary sensor model, but discusses a probabilistic model in Appendix A). In a probabilistic sensor model, the sensor's coverage is probabilistic, where different points in the environment will give different probability densities according to the sensor model of whether the target will be detected by the sensor. Given the probabilistic sensor model constraints, Wu et al. [14] give an ILP-based approximation algorithm and distributed approach called the *Coverage Benefit Detection Algorithm.*

In recent work by Akbarzadeh et al. [15], the authors detail a far more sophisticated and realistic 3D probabilistic directional sensor model than that used in most previous coverage research. They test several coverage placement algorithms with real terrain maps to demonstrate real world issues that arise as a result of points in the map having a blocked line-of-sight to sensors in the real-world terrain. They prove that an existing geometric approach to coverage is far less effective at coverage than the optimization techniques they test.

Although these discussed approaches focus on coverage, they do not consider or do not assume the knowledge of the possible path a robot will take and as a result, the "best" coverage is not always efficient or effective when the main focus is ensuring accurate mobile robot navigation.

Some initial work on sensor placement for localizing a robot with the inclusion of error was done by Zhang [16]. Sensors were represented by their covariance matrices[2], assumed to be Gaussian and ellipses defined by the eigenvectors[2] and values of the covariance matrix. Zhang then framed the problem as determining the orientations around a common origin that would minimize the determinant of the joint covariance matrix of all the sensors. Zhang proved that if a non-trivial local optimum configuration (angles that are not 0^o and 90^o) of orientations exists then it is the global optimum and its minimum covariance ellipse is a circle. This approach made several simplifying assumptions that are not practical in more complicated real world problems. For example, the scope of the work does not include consideration of a robot moving around the environment.

Jourdan and Roy [18] detail optimal sensor placement with respect to their Position Error Bound (PEB) function, a lower bound for range-based localization sensors which is similar to the geometric dilution of precision (a metric for measuring the quality of a configuration of GPS satellites), but which accounts for sensor bias and sensor variance dependent on the distance of the reading. A certain number of k sensors are placed on boundaries of polygons to minimize the overall average PEB at specific points using a *coordinate descent* algorithm that they prove to converge with a specific error bound. This approach does not include the robot, but focuses on minimizing error at specific points in an environment.

Vitus and Tomlin [19] take the robot's uncertainty along a planned trajectory into account when placing omnidirectional sensors. Using a linear Gaussian approach, the robot's error is represented by a collection of Gaussian covariance matrices along the planned path. The covariance matrix at a point along the path is determined through the covariance of the previous state and the covariance of sensors that are visible at that current state. To determine whether a sensor is visible given that a sensor has a maximum sensing range, the positional covariance at a point along the path is represented by an ellipse of the position covariance with a given Chi square confidence δ (typically $\delta \geq 85\%$). If the sensing range of the sensor completely encompasses the robot's covariance ellipse, then the robot can likely be perceived. Using M sensors, the optimal placement of sensors is determined based on minimizing the sum of the traces of the covariance matrices. The authors present an incremental algorithm

[2]For more information on covariance matrices as they relate to multivariate normal density function and eigenvectors the reader is directed to [17].

where a new sensor is appended to the existing ones and its pose is optimized with gradient descent. They demonstrate through simulation that their approach provides better placements more consistently than does simulated annealing.

Beinhofer et al. [20] evaluated placement of artificial landmarks using a so-called unscented Kalman Filter approximation of the robot's traversal of a trajectory. They proved that a particular subset of the problem of artificial landmark placement, where the decision of movement of the robot is not directly determined by the previous state, is a *submodular set function.* The intuitive consequence of a submodular function is that the positive effects of adding landmarks will diminish as more and more landmarks are added. Nemhauser et al. [21] proved that a greedy approximation algorithm for submodular set functions gives a guaranteed result of at least $(1-\frac{1}{e}) \approx 63\%$ of the optimal solution. In the case where a robot's movement is determined by the previous state (*autonomous control*) and submodularity no longer holds, their simulations and experiments showed that a greedy approximation algorithm still performs well in this case.

Recently, Beinhofer et al. [22] improves on similar work by Vitus and Tomlin, in that points along a given path are represented by Gaussian distributions and computed through linear Gaussians. The problem is altered so that instead of k landmarks being placed to minimize the overall error of the robot's trajectory, landmarks are instead incrementally added to satisfy the property that the robot does not deviate from the desired path by more than a distance threshold d_{max} with a probability p_{min}. Although the number of landmarks the algorithm uses may not be the minimum number of landmarks for the global optimum, it gives a heuristic minimum number of landmarks to satisfy deviation constraints. Their approach is thoroughly validated through extensive experimentation with $d_{max} = 0.5$ meters and $p_{min} = 99\%$. These results are extended to be more robust in [23], wherein optimal landmark placement would take into consideration the possibility of missing or unobserved landmarks.

Analogous research to that presented in this thesis was reported in the Master's thesis of Meger [24]. That thesis dealt with localizing a mobile robot fitted with fiducial markers that moved within a camera sensor network (in contrast, this thesis has cameras on the robot and fiducial markers in the environment). The camera network approach has many benefits as a significant amount of information to estimate the robot's pose can be gained using a camera network. It is also a logical approach in an environment where a camera network may already exist, as is typically the case with security surveillance systems. However, in remote or underwater environments, such a camera network may not be as feasible or practical. Cameras within the network require power and the cameras require calibration. In underwater environments cameras have a reduced field of view – a problem that also exists for any cameras on a robot. That said, it may be more cost effective and practical to flood an environment with many crude markers and outfit a robot with cameras than to power a large network of camera sensors. We recognize that both approaches have different advantages in different circumstances.

2.2 Fiducial Tags

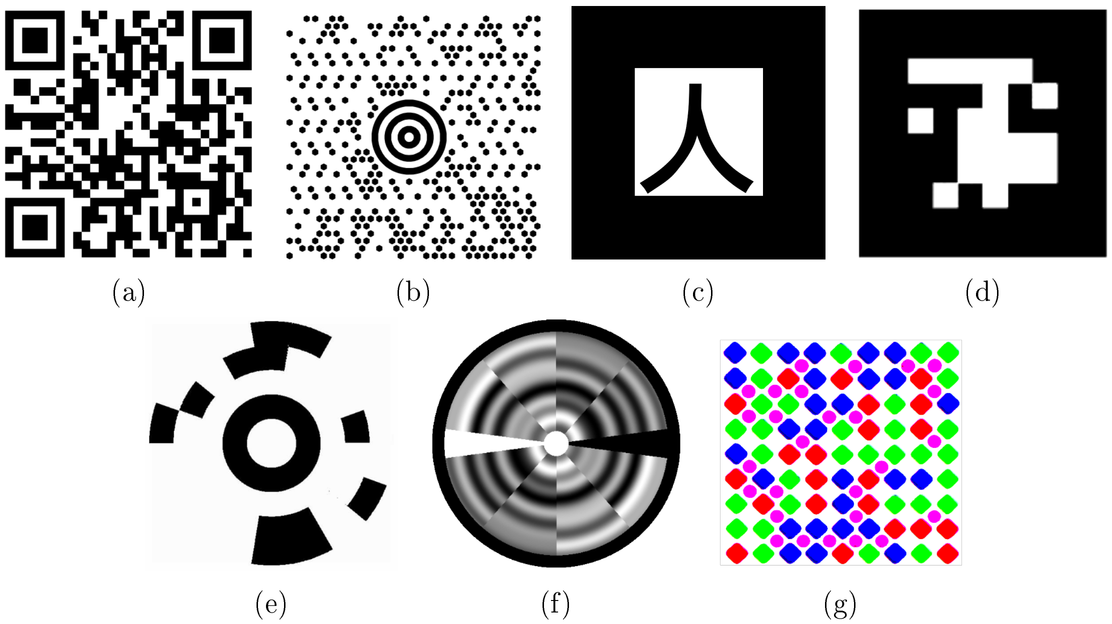


Figure 2.2: a) QR code (image from wikipedia.com). b) MaxiCode (image from wikipedia.com). c) ARToolkit (image from ARToolkit software package). d) ARTag (image from [1]). e) Cantag, circle inner version (image from [2]) f) Fourier tag (image from [3]). g) robust pseudo-random array fiducial marker (image from [4]).

Planar fiducial markers encode data into a planar pattern that can be decoded by computer vision techniques. While this thesis will use fiducial markers in the experimental component, the topic of fiducial markers is not a focus of this thesis, so we will not go into depth on the subject. For a more complete overview of the topic the reader is recommended to read [4], which gives a relatively recent and in-depth review of existing marker systems. Some of the most common visual encodings are the DensoWave *Quick Response (QR) Code* (Figure 2.2a) and UPS' *MaxiCode*

(Figure 2.2b). Both these systems are designed for use as barcodes and therefore are used to encode relatively large amounts of data. Decoding then requires a *good* capture of the barcode. In applications where the barcode must be read from a range of perspectives and at different resolutions, decoding is not very robust, and so the use of barcodes does not serve as an optimal choice for robot localization.

Two bitonal (i.e. only two tones are used) fiducial markers of note were developed in the early 2000's for use in augmented reality, but have extended to far different applications as well (i.e. [25]). A major contribution has been the *Augmented Reality Toolkit* (ARToolkit), a fiducial marker whereby the marker pattern consists of a unique symbol surrounded by a dark border (Figure 2.2c). Recognition of the marker is accomplished by calculating the correlation of detected sample pixels to all marker symbols being used in the particular application. The dark border surrounding the symbol also gives the added benefit that the four corners that define the marker square can be used as four known points in the environment which can then be used to determine the 3D position and orientation of the viewer. This approach has several issues, such as the correlation process mistaking one marker for another (*interconfusion*) as well as scalability concerns as the number of markers used increases. An alternative approach is the *Augmented Reality Tag (ARTag)* created by Fiala [1], which incorporates a linear error correcting codes approach. A binary data ID from roughly the range $[0, 1024]$ is encoded with a Reed-Solomon error correcting code together with a checksum into a 36 bit packet. This packet is then arranged into a 6×6 grid that defines the markers (see Figure 2.2d). There are a variety of different markers that are very similar to ARTag (Cantag, ARToolkit Plus, BinAryID, etc.). Olson [26] implemented and detailed a C and Java open source tag called *AprilTag* that is a more robust ARTag. We became aware of AprilTag after implementing our own fiducial marker and so we decided to continue with our simple Python implementation (see Chapter 3.3) instead of using AprilTag. Use of AprilTag may be done in future work.

Stathakis et al. [4, 27] introduce the *Robust Psuedo-Random Array Marker*, a colour-based fiducial marker that can better handle occlusion and that affords 3D pose estimation. Rather than relying on edge detection methods to recognize the border of a marker, this marker uses *pseudo random arrays* as unique features that can be used to detect the presence of a marker. Although a seemingly logical choice for this thesis, it was not chosen for reasons discussed in Chapter 3.

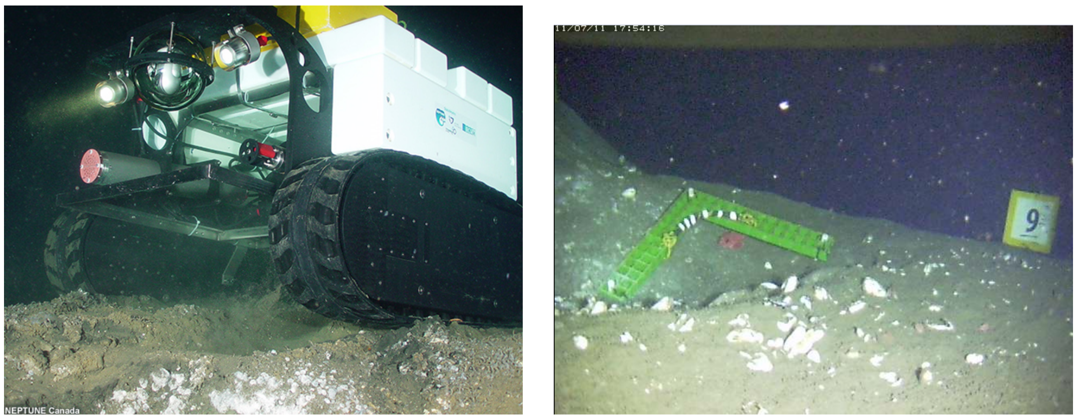
A common feature with the previous mentioned approaches is the assumption that the data of markers is either completely recognized in full or not at all. This leads to distance thresholds, where after some distance the marker is assumed to be no longer recognizable and ceases to provide any information. One fiducial tag that allows for smooth degradation of information as distance from the marker increases is the *Fourier tag* [28, 3], which encodes marker data by transforming the binary digits of the marker into relatively high amplitude peaks in the frequency domain. The Fourier tag was not selected for this thesis because the need for smooth degradation is not required and introduces more complexity.

Harle and Hopper [29] describe a system that increases the robustness and data of markers by creating a cluster of multiple Cantag fiducial markers [2] (see Figure 2.2e for one example of a Cantag marker). This gives a much greater data size, as data can be stored amongst a collection of markers. Any failure to recognize an individual marker is mitigated by the recognition of the other markers in the cluster. This is an interesting approach, but introduces far more complexity than necessary for our goals.

2.3 Applications

Investigation into different approaches to localization will continue to be relevant because there will always exist applications where GPS is not available: under the surface of the ocean [30], underground and in indoor environments. Other sensors can make up where GPS fails, but they do have problems – one of the major issues being cost. Planar fiducial markers are relatively cheap as the majority of cost is in the printer ink and can be measured in cents. Because of their low price, flooding an environment with a large number of markers is not an issue, assuming that placement of the markers can be readily done. Most sensors in a sensor network require power to operate as well as transmit readings over a radio. The use of power usually requires the use of batteries, which limits sensor lifetime or requires sensor maintenance. Because the fiducial markers are made of paper their lifetime is not of critical concern. A binary sensor is the most primitive sensor that can exist and although modern sensors provide a variety of useful information for localization, binary sensing can still provide some information. For example, the modern fiducial markers discussed above can be used to estimate the pose of the viewer to the marker, but as far as we are aware this requires the camera to be calibrated. Setting aside the benefits of price and maintenance that markers give, it is also possible to imagine scenarios where the camera of the robot becomes uncalibrated or damaged during operation, but is still capable of detecting markers without the capability of accurately using the pose estimation information from the marker. By being aware of the the crudest form of observation, the detection or lack of detection of certain markers, a robot can still operate with some effectiveness in its environment knowing that a marker is only visible under certain conditions (i.e. within an annulus sector origined at the marker).

The problem of a robot requiring accurate traversal along routine paths in a known environment is very common in the automation of factory and warehouse enterprises. Because of its location here in Victoria, we now elaborate on an interesting example that is provided by the NEPTUNE Canada project, under Ocean Networks Canada. At the time of this writing, NEPTUNE Canada is the largest underwater observatory in the world. The overall goal of NEPTUNE Canada is to provide a network of long-term *in situ* observatories to collect data on underwater environments for the study of a wide range of topics [31]. These include fauna populations, methane deposits, bacterial mats, and shifts in ocean properties over time in relation to climate change, to name a few.



(a) (b)

Figure 2.3: a) The remotely operated benthic crawler: Wally. b) An example of a markers (yellow ‘9’ sign) used by the teleoperaters of Wally to help them localize. The images are from Ocean Networks Canada.

One area being monitored as part of Ocean Networks Canada is Barkley Canyon because of its extensive hydrate mounds. To allow for mobile observation of hydrates, Barkley Canyon is equipped with a benthic (sea-floor) remotely operated crawler known as “Wally” (see Figure 2.3a). Its purpose is to monitor “methane flux variations and gas hydrate dynamics at the Barkley Canyon hydrate outcrops” [32]. Wally is powered by a 70 meter cable connected to the Barkley Hydrates instrument platform operating at a depth of 870 meters. The instrument platform also provides high bandwidth communication to and from Wally. Wally was designed and is teleoperated by a team at Jacob's University in Bremen, Germany. There are two crawlers, Wally I and Wally II, but only one robot is active at the underwater site at any time. It operates for approximately one year and is swapped for the other robot during the summer. The swapped out robot is then taken to the surface for repairs and upgrades. A long-term goal is to eventually have Wally operate autonomously. The choice of sensors and their placement can become an issue as the the cost associated with sensor placement underwater can be expensive. In addition, it could be of some value to localize Wally's pose using the existing old data, which include buoyant markers that are used by Wally's teleoperaters (see Figure 2.3b).

The sea-floor conditions of Barkley Canyon are quite dynamic. Strong currents will cause markers to slightly move, change orientation and be buried. The sea-floor is very uniform in texture, but can change drastically over time making the finding of landmark features difficult. The environmental noise of marine snow and sediment clouds can also make the accurate detection of landmarks difficult. Under these conditions, the simple act of a marker being detected where it would only be visible from a certain range and/or angle may be the only useful and recognizable source of data for localization purposes.

Chapter 3

Approach, Methodology and Parameter Estimation

In this chapter, we discuss some of the theoretical background and reasoning for the approach of this thesis. Because the research of this thesis is ultimately intended for application in the physical world, discussion will also include some of the nuances and challenges associated with moving from theory to practice.

3.1 Localization

In the real world it is difficult to truly know the exact pose of a robot to infinite precision. As a result, it is common in robotics to best view the pose of a robot through the perspective of "belief". Belief is described mathematically through probability and random variables. A random variable Y is a function with a discrete or continuous domain (or *sample space*) and a range such that for every value y in the range of Y $p(Y = y) \geq 0$ and $\int p(Y = y)\,\mathrm{d}y = 1$.

In the next sections, we discuss the notion of a robot's "belief" about its pose, as it makes a movement or action given inputs and resolves error through marker observation. We specifically use a *particle filter* in this thesis (which we will describe), so we provide some of the high level mathematical background to justify why it works.

3.1.1 Motion Model

We assume our robot will be operating in 2 dimensions and will work in discrete steps of time. The interval between time steps may not necessarily be constant, but

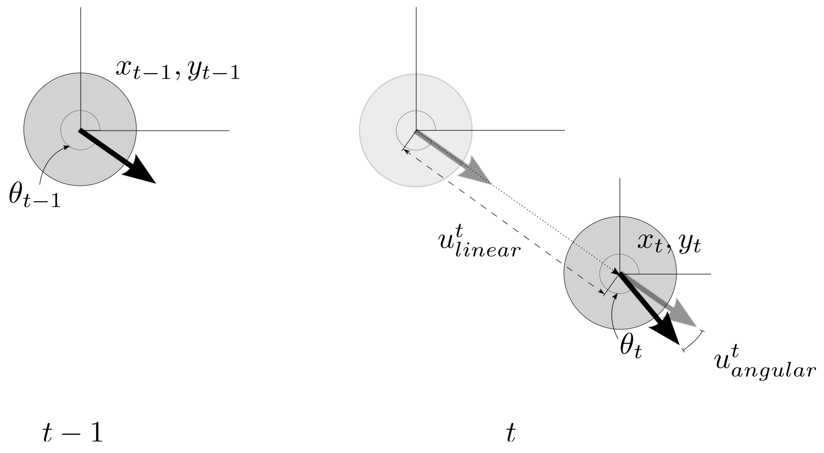


Figure 3.1: An illustration of the robot (gray circle) at time $t-1$ moving to a new position at time t using the control inputs $u^t_{linear}, u^t_{angular}$.

the robot's state at time t is represented by a random variable X_t that describes the robot's pose in a global coordinate frame (without error):

$$X_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$$

Here, (x_t, y_t) represents the robot's position with respect to a global origin and θ_t is the robot's orientation with respect to the global origin axes. We focus on modeling a mobile robot that traverses an environment. When the robot moves it alters its *state* between each discrete time step. The type of robot we work with is a differential drive robot, which can drive straight as well as turn in place. (The actual robot used is described in more detail later in Chapter 5). In our work, to control the robot to move from one point to another, we use a sequence of in-place turns and straight-line movements, or *angular* and *linear movements*. These are given by linear and angular inputs: u^t_{linear} and $u^t_{angular}$ at time t, respectively (see Figure 3.1). The linear and angular control inputs describe the linear distance the robot has traveled on its current heading and the change in its heading. The change in distance and heading is ultimately described by the linear and angular velocity, for example $u^t_{linear} = v^t_{linear}(t-(t-1))$ where v^t_{linear} is a velocity given to the robot. Some models will view control inputs as the velocity inputs at different times. In our case, we work with a robot's onboard odometry. The robot we use is equipped with a tachometer on each wheel which gives us the distance the robot has traveled and turned when polled.

Assuming the robot moves perfectly, the robot's new pose after one time step $t-1$ to t can be approximated by the linear transformation:

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & u^t_{linear} \times \cos\theta_{t-1} \\ 0 & 1 & 0 & u^t_{linear} \times \sin\theta_{t-1} \\ 0 & 0 & 1 & u^t_{angular} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \\ 1 \end{bmatrix}$$

Because our robot operates in the physical world, errors will inevitably occur as the robot travels. These errors can be caused by several factors, such as an imperfect actuation of the robot and wheel(s) slippage, to name a few. Such errors must be considered when working with a robot operating in the real world. To account for error, we include possible errors from linear and angular motions: e^t_{linear} and $e^t_{angular}$ respectively for time t. We assume the error is described by zero mean normal distributions:

$$\mathcal{N}(x, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$e^t_{linear} = \mathcal{N}(0, {\sigma^t_{linear}}^2)$$

$$e^t_{angular} = \mathcal{N}(0, {\sigma^t_{angular}}^2)$$

where σ^t_{linear} and $\sigma^t_{angular}$ are standard deviations based on observing the deviation of the robot as it moves and the particular movement at time t. Intuitively, a larger linear distance traveled at time t should have $\sigma^t_{linear} \geq \sigma^{t-1}_{linear}$ if the robot at $t-1$ traveled a smaller linear distance. The same logic applies to $\sigma^t_{angular}$.

To represent the belief about a robot's pose while considering error, we represent the belief about a robot's pose as a random variable. The belief about the robot's initial position at time $t = 0$ is encapsulated in the notation: $bel(X_0)$. For the convenience of the reader, we mention now that the random variable we use to represent X_0 is still not defined, but the following is meant to demonstrate the requirements and implications for any random variable. We will discuss more about the random variable used shortly.

Given we know $bel(X_t)$, we can predict the robot's pose after a given motion command by incorporating the odometry readings after a motion and the belief about a robot's pose at the previous state by using a *motion model* or *action model*:

$$\overline{bel}(X_t) = \int p(X_t|u_t, X_{t-1})\, bel(X_{t-1})\, \mathrm{d}X \tag{3.1}$$

Equation 3.1 is the motion model of the classic *Bayes Filter*. The $p(X_t|u_t, X_{t-1})$, from a high-level perspective, describes through the language of probability how the belief of the robot will change given the motion input u_t and previous belief X_{t-1}. This component will introduce what the expected error from such a motion will be (for a particle filter visualization of this, see Figure 3.3). The reason we use the line over *bel* (i.e. $\overline{bel}(X_t)$) is that the motion model serves as a intermediary step for when we correct for error using observations (described below in Section 3.1.2). If there were no correction for error included, we could simply have $bel(X_t) = \overline{bel}(X_t)$, but the state of the robot would accumulate errors. One of the key implications of Equation 3.1 is that the only information taken into account is the previous state X_{t-1} and the current motion u_t; all previous states $X_{t-2}, X_{t-3}, ..., X_0$ and motions $u_{t-1}, u_{t-2}, ..., u_1$ are assumed to be encompassed in the previous state. This assumption (the *Markov assumption*[1]) makes the motion model recursive and this assumption works well enough in practice.

As mentioned above, Equation 3.1 describes any random variable. With that in mind, the choice for the random variable to represent the belief about the robot's pose will ultimately affect how Equation 3.1 is actually implemented. One of the logical choices for a random variable is one of the most common probability density functions: the *multivariate Gaussian function*. The multivariate Gaussian function is the Gaussian function extended into > 1 dimensions. It is parameterized by a d-vector mean μ and $d \times d$ square covariance matrix Σ, where d is the number of dimensions. For our purposes where we are working in 2D and consider the robot's orientation (x, y, θ), so $d = 3$. In cases where a robot operates in 3D, a system must consider the 3D position of the robot (x, y, z) as well as its three orientations: yaw, pitch and roll. In such a 3D system (not used in this thesis), the multivariate Gaussian function would been be defined by $d = 6$. In order for the multivariate Gaussian function to exist, we assume Σ is comprised of real numbers and must be *nonsingular* (i.e. $|\Sigma| \neq 0$) and *positive definite* (i.e. given any non-zero column vector b, $b^T\,\Sigma\, b > 0$). The probability density for a given d-vector x can be found by:

[1]For more information about the *Markov assumption* refer to [17].

$$pdf(x, \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right) \tag{3.2}$$

Using a multivariate Gaussian function for the random variable X_t leads to a commonly used filter: the *Kalman filter* [33]. Although this thesis does not use the Kalman filter, we believe its existence warrants mention. The formulation of the Kalman filter requires a linear Gaussian motion and observation model (the observation model will be discussed below). The Kalman filter's requirement for everything to be Gaussian and linear is not always realistic or possible (i.e. some sensors will not return readings that are in a (x, y, θ) Gaussian form). Solutions to address these requirements have arisen. One example is the Extended Kalman filter, which works by approximating an action and/or observation model that is a nonlinear differentiable function by a partial derivative Jacobian matrix through a first order Taylor series expansion. This essentially approximates an action or observation model into a linear Gaussian form [7]. This linear approximation can then be used within the Kalman filter. The first order Taylor series approximation can work quite well when the approximated model(s) have small error, but if the error is large then the filter can suffer as the models are only an approximation [7]. An additional issue that can arise from working with the Kalman filter is that a Gaussian function is *unimodal*, that is it only has one peak. In the course of localization, especially global localization, it is possible that the robot could likely be in any one of multiple general locations. The Kalman filter is quite efficient and entails certain requirements that can usually be satisfied, but not so in our case.

A popular alternative to Kalman filters, and the one used in this thesis, is the *Monte Carlo Localization* (MCL) approach [34] first introduced as the *bootstrap filter* by Gordon, Salmond and Smith [35]. The basis of MCL is that a probability density function that represents the belief of our robot's pose, which is continuous, can be approximately represented by a "large" discrete collection of *samples* or *particles*[2]. This approach is commonly known as a *particle filter*. The use of 'large' is left vague, but the number of particles N that are necessary to effectively represent a probability density function is dependent on the properties and complexity of the function. As a reminder, the domain of the probability density function for this thesis is the set of possible poses (x, y, θ) of the robot in its environment. A density function with

[2]This thesis will use the words 'particles' and 'samples' interchangeably. At times we will also refer to the process of taking samples – which involves generating samples or particles from a probability density function using pseudo random number generation.

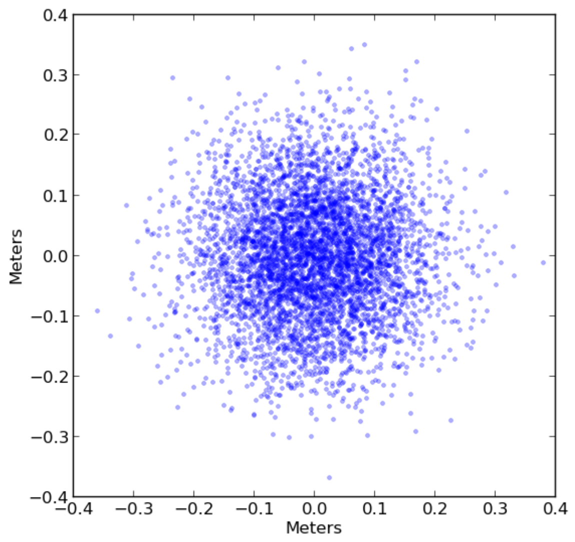


Figure 3.2: Example of 5000 samples taken from a 3D (x, y, θ) zero mean multivariate Gaussian plotted in 2D (x, y). Each of these particles can be seen to represent a possible initial pose of the robot. The orientation of each sample is not shown.

many dimensions and possible modes (multimodal) will likely require more particles. By discretizing the probability density function into N samples, each sample can be viewed as a possible pose of the robot. The size of N can be fixed or dynamic. The choice of N is typically in the thousands with current processors. In this thesis, we arbitrarily set $N = 5000$ which proved overly sufficient. We define a sample i at time t as:

$$s_t^i = \begin{bmatrix} x_t^i \\ y_t^i \\ \theta_t^i \end{bmatrix}$$

Additionally, each sample has a corresponding weight w_t^i that satisfies $\sum_{i=1}^{N} w_t^i = 1$. Initially, when $t = 0$, particles are sampled from a multivariate Gaussian function where the mean μ_0 is the starting pose of the robot and the covariance matrix Σ_0 has relatively low variance relative to how well known the starting pose is. In the case of global localization, samples can be taken from a uniform density function, sampled within the navigable area in an environment. An example of particles for the particle filter generated at time $t = 0$ can be seen in Figure 3.2. The weight for every particles i at $t = 0$ is $w_0^i = \frac{1}{N}$.

With the particle filter, updating the robot's pose after a movement can be viewed

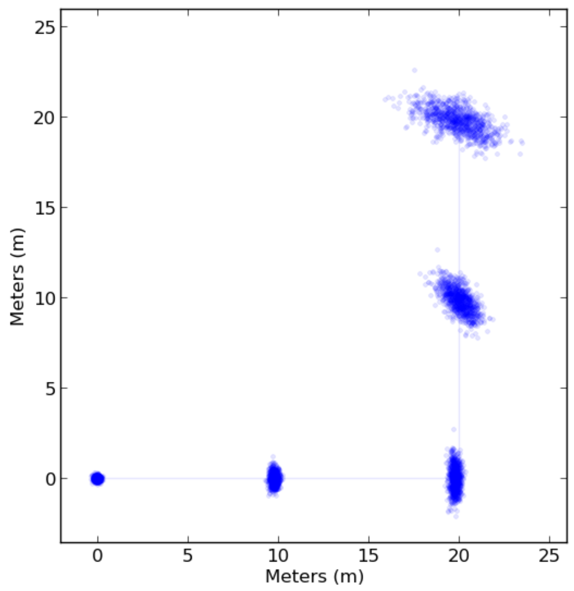


Figure 3.3: The spread of particles at different intervals as the robot travels 40 meters from the bottom left to the top right. The particles have been projected into 2D (x, y).

as moving each particle using samples from the motion model. In our case, we draw N samples from $\phi_t^i = e_{linear}^t$ and $\psi_t^i = e_{angular}^t$. Recall, e_{linear}^t and $e_{angular}^t$ are 1D normal functions with zero means and standard deviations described by σ_{linear}^t and $\sigma_{angular}^t$, respectively. They describe the error of the linear and angular motions. After a movement of u_{linear}^t and $u_{angular}^t$ at time t, then a particle s_{t-1}^i is updated by:

$$s_t^i = \begin{bmatrix} x_t^i \\ y_t^i \\ \theta_t^i \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & (u_{linear}^t + \phi_t^i) \times \cos\theta_{t-1}^i \\ 0 & 1 & 0 & (u_{linear}^t + \phi_t^i) \times \sin\theta_{t-1}^i \\ 0 & 0 & 1 & u_{angular}^t + \psi_t^i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1}^i \\ y_{t-1}^i \\ \theta_{t-1}^i \\ 1 \end{bmatrix} \tag{3.3}$$

Equation 3.3 is applied to all N particles and comprises the robot's belief after moving $\overline{bel}(X_t)$. Figure 3.3 demonstrates this process where particles in the particle filter are taken at intervals after applying input commands over 40 meters of travel.

3.1.2 Observation Model

As seen in the Figure 3.3, as a robot continues to travel the motion errors will accumulate, such that as time goes to infinity, so too will the error. Ultimately, a robotic system must bound this error in order to accomplish any meaningful navigation. To bound the error, the robot must collect sensor observations, either through an external positioning system or by observing its environment for landmarks, that is, features in an environment that help the robot determine where it is. In our work, the robot observes artificial landmarks in the environment in the form of planar fiducial markers that are fixed at known locations which can be specified.

After the robot has updated its previous belief ($bel(X_{t-1})$) following a movement that completes by time t to get a preliminary belief about its new pose ($\overline{bel}(X_t)$), the robot observes its environment and detects or does not detect every marker in the environment. Given M total markers in the environment, these observations are represented by the collection of observations $z_t = \{z_t^1, z_t^1, ..., z_t^M\}$, where z_t^j is a random variable associated with marker j at time t. In this thesis, we treat markers as binary directional proximity markers, so we define the observation random variable z_t^j as capable of taking on only two events: if the marker is detected, $z_t^j = 1$; if the the marker is not detected $z_t^j = 0$.

After the robot observes the environment resulting in a collection of marker readings z_t, the robot's preliminary belief ($\overline{bel}(X_t)$) after moving needs to be revised using the marker readings to get the new belief ($bel(X_t)$). This is accomplished mathematically by the *observation model* or *sensor model* of the Bayes Filter [7]:

$$bel(X_t) = \eta \int p(z_t | \overline{bel}(X_t)) \, \overline{bel}(X_t) \, \mathrm{d}X \tag{3.4}$$

Here η is a *normalization factor*, that is, a constant that represents a component of Bayes' theorem that ensures all values sum to 1. The exact value of the normalization factor is not important for now. Throughout the various equations below, the normalization factor changes, but we keep the same notation η.

We assume that the detections of markers are independent of one another, so this allows us to re-express Equation 3.4 as:

$$bel(X_t) = \eta \int \left(\prod_{j=1}^{M} p(z_t^j \mid \overline{bel}(X_t)) \right) \overline{bel}(X_t) \, \mathrm{d}X \tag{3.5}$$

In the particle filter, the belief of the robot's pose is described by the weights associated with the particles. The collection of the particles s_t and their weights w_t comprise $bel(X_t)$. Thus, for the particle filter, Equation 3.5 is simplified by determining the weight for each particle given the observations:

$$\overline{w}_t^i = \left(\prod_{j=1}^{M} p(z_t^j \mid s_t^i) \right) \, w_{t-1}^i$$

where we let $\overline{w}_t^i$ denote the preliminary weight of the particle s_t^i before normalization to ensure that all the weights should sum to 1.

To simplify the model, we introduce a 0,1-valued random variable $v_t^{i,j}$ that is associated with the visibility of a marker j at time t by the particle s_t^i. The random variable indicates whether the particle s_t^i is within the visibility region of marker j, namely, $v_t^{i,j} = 1$ when the particle is in the visibility region of marker j and $v_t^{i,j} = 0$ otherwise. Our approach borrows some of the ideas from Djurić et al. in [36]. This new random variable gives rise to the following expression for the conditional probability $p(z_t^j \mid s_t^i)$:

$$p(z_t^j \mid s_t^i) = \eta \int_v p(z_t^j \mid v_t^{i,j},\, s_t^i)\, p(v_t^{i,j} \mid s_t^i)\, \mathrm{d}v \tag{3.6}$$

By assuming conditional independence, this can then be simplified to:

$$p(z_t^j \mid s_t^i) = \eta \int_v p(z_t^j \mid v_t^{i,j})\, p(v_t^{i,j} \mid s_t^i)\, \mathrm{d}v \tag{3.7}$$

Because $v_t^{i,j}$ is discrete, the integral of Equation 3.7 becomes a discrete sum:

$$p(z_t^j \mid s_t^i) = \eta \sum_v p(z_t^j \mid v_t^{i,j})\, p(v_t^{i,j} \mid s_t^i) \tag{3.8}$$

$$= \eta \, \big(p(z_t^j \mid v_t^{i,j} = 1)\, p(v_t^{i,j} = 1 \mid s_t^i) + p(z_t^j \mid v_t^{i,j} = 0)\, p(v_t^{i,j} = 0 \mid s_t^i) \big) \tag{3.9}$$

Because z_t^j and $v_t^{i,j}$ are discrete and only take the values $\{0, 1\}$, the conditional probability distribution for z_t^j and $v_t^{i,j}$ is simple to represent in table form:

$p(z_t^j = 1 \mid v_t^{i,j} = 1) = a$
$p(z_t^j = 1 \mid v_t^{i,j} = 0) = b$
$p(z_t^j = 0 \mid v_t^{i,j} = 1) = c$
$p(z_t^j = 0 \mid v_t^{i,j} = 0) = d$

In the table, $a + b = 1$, $c + d = 1$, $a > b$ and $d > c$.

If the detection of markers were perfect within the annulus sector and never detected outside the annulus sector then $a = d = 1$ and $b = c = 0$. In reality, there are times where the robot will miss the detection of a marker when the robot is within the visibility region and successfully detect the marker when the robot is outside the visibility region. The values for a, b, c, d will ultimately affect the impact of a detected marker or lack of detected marker on the belief of the robot's pose. If a is close to 1 and the robot does not detect the marker when the robot is within the visibility region of the marker, the particle filter will confidently make a huge a mistake. Similar logic applies for d. The values chosen can be found through experimentation, but a and d should chosen to prevent the particle filter from becoming too confident. We chose the following values after observing the robot and for the sake of simplicity:

$p(z_t^j = 1 \mid v_t^{i,j} = 1)$	0.85
$p(z_t^j = 1 \mid v_t^{i,j} = 0)$	0.15
$p(z_t^j = 0 \mid v_t^{i,j} = 1)$	0.15
$p(z_t^j = 0 \mid v_t^{i,j} = 0)$	0.85

With $p(z_t^j \mid v_t^{i,j})$ defined, we need to define $p(v_t^{i,j} \mid s_t^i)$. Because s_t^i is continuous, $p(v_t^{i,j} = 1 \mid s_t^i)$ and $p(v_t^{i,j} = 0 \mid s_t^i)$ *must* be continuous distributions. In the next sections we look at the uniform continuous probability distribution to describe the *visibility model*. An alternative probability distribution for a visibility model is described in Appendix A, but it is not used in this thesis.

3.1.3 Uniform Visibility Model

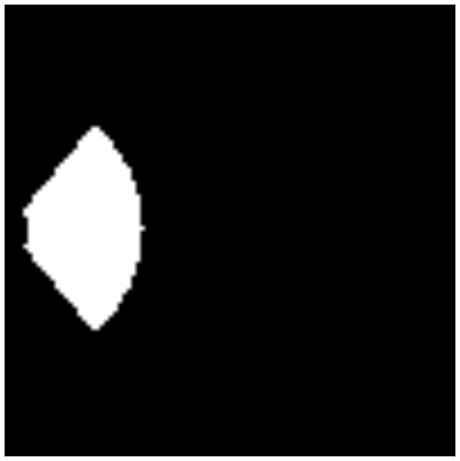


Figure 3.4: The uniform visibility marker model.

One of the simplest notions for $p(v_t^{i,j} \mid s_t^i)$ is to make the assumption that any point that is visible, no matter if it is in the middle of the annulus sector or at the very edge, is equally likely. A continuous distribution that embodies this is the *continuous uniform distribution.* The continuous uniform distribution for an interval $[a, b]$ is defined as:

$$\mathcal{U}_{[a,b]}(x) = \begin{cases} \frac{1}{|b-a|} & \text{If } a \leq x \leq b \\ 0 & \text{Otherwise} \end{cases} \tag{3.10}$$

This continuous uniform distribution only works for an interval, but the idea can easily be extended for the annulus sector. We define a continuous distribution for marker j as:

$$\mathcal{U}_t^j(s_t^i) = \begin{cases} \frac{1}{\nu_t^j} & \text{If } r^j \leq r_t^i \leq R^j,\ |\omega_t^i - \theta^j| \leq \alpha^j \\ 0 & \text{Otherwise} \end{cases} \tag{3.11}$$

where ν_t^j is the area of the visibility annulus sector, r_t^j, R_t^j are the minimum and maximum range of marker j respectively, θ_t^j is the orientation of the marker j and α_t^j is the sector angle of marker j. Additionally, r_t^i and ω_t^i are the distance and angle from the sample s_t^i to marker j respectively defined by:

$$r_t^i = \sqrt{(x_t^i - x_t^j)^2 + (y_t^i - y_t^j)^2}$$

$$\omega_t^i = \arctan(y_t^i - y_t^j, x_t^i - x_t^j)$$

If we assume that every visible point is equally likely then we define $p(v_t^{i,j} = 1 \mid s_t^i)$ and $p(v_t^{i,j} = 0 \mid s_t^i)$ as:

$$p(v_t^{i,j} = 1 \mid s_t^i) = \mathcal{U}_t^j(s_t^i) \tag{3.12}$$

$$p(v_t^{i,j} = 0 \mid s_t^i) = 1 - \mathcal{U}_t^j(s_t^i) \tag{3.13}$$

An illustration of Equation 3.12 given several (x, y) points can be seen in Figure 3.4.

It should be noted that the probability density $\frac{1}{\nu_t^j}$ comes from the area of the visibility annulus sector ν_t^j.

3.1.4 Resampling

After sensing observations and computing the new weights for each particle, each sample is then normalized to remove η and ensure all the weights sum to 1:

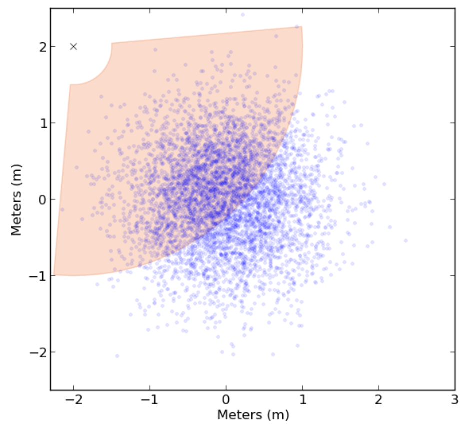
$$w_t^i = \frac{\overline{w}_t^i}{\sum_{k=1}^{N} \overline{w}_t^k}$$

When the observations have been considered, particles that clearly do not represent the robot's state should likely be removed. If the underlying distribution is known, new particles can be generated from it. If it is unknown, a practical solution can be to get new particles by reusing existing particles in such a way that the new particles better fit the belief of the robot's state after observation. In this process of selecting new particles from existing ones, particles with a higher weight should have a better chance of being selected during the resampling process. In this thesis, we use an $O(N)$ running time algorithm (see Algorithm 1) that is discussed in [7]. The algorithm gives good importance replacement without sacrificing too many low weight particles.

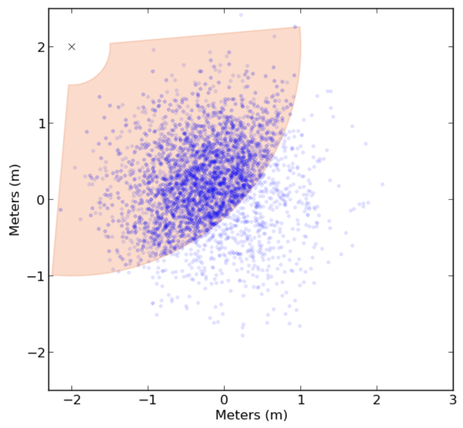
Algorithm 1: Low_variance_sampler(S_t, W_t) (taken from [7])

```
1  $\overline{S}_t \leftarrow \emptyset$
2  $r \leftarrow \mathrm{rand}(0; N^{-1})$
3  $c \leftarrow w_t^1$
4  $i \leftarrow 1$
5  for $m \leftarrow 1 \ldots N$ do
6      $u \leftarrow r + N^{-1}(m-1)$
7      while $u > c$ do
8          $i \leftarrow i + 1$
9          $c \leftarrow c + w_t^i$
10     add $s_t^i$ to $\overline{S}_t$
11 return $\overline{S}_t$
```

The resampled particles become S_t and each of the weights associated with the particles is set to $w_t^j = \frac{1}{N}$ for fairness. Using the observation model of Equation 3.9 given the uniform visibility models of Equation 3.12, the figures in Figure 3.5 demonstrate particles after resampling. When resampling is complete, the collection of particles and weights now encompass the robot's belief at time t. The full particle filter algorithm for one time iteration is shown in Algorithm 2.



(a) Uniform visibility model (before)



(b) Uniform visibility model (after)

Figure 3.5: Particles in the particle filter before and after observation when the marker is detected.

Algorithm 2: Particle_filter$(S_{t-1}, W_{t-1}, U_t, Z_t)$

```
1 for $i \leftarrow 1 \dots N$ do
2     $s_t^i \leftarrow p(s_t^i | s_{t-1}^i, U_t)$
3 for $i \leftarrow 1 \dots N$ do
4     for $j \leftarrow 1 \dots M$ do
5         $\overline{w}_t^i \leftarrow \big(p(z_t^j \mid v_t^i = 1)p(v_t^{i,j} = 1 \mid s_t^i) + p(z_t^j \mid v_t^i = 0)p(v_t^{i,j} = 0 \mid s_t^i)\big)\ w_{t-1}^i$
6 $W_t = \frac{\overline{w}_t^i}{\sum_{k=1}^{N} \overline{w}_t^k} \forall w_t^i \in W_t$
7 $S_t =$ Low_variance_sampler$(S_t, W_t)$
8 $W_t = \frac{1}{N} \forall w_t^i \in W_t$
9 return $S_t, W_t$
```

3.2 Localization Patches and the Scoring Function

When the annulus sector visibility regions of markers are placed in an environment, they along with the environment form a collection of possibly overlapping planar regions. In this thesis, we will refer to these planar regions as *patches* in keeping with terminology from the literature [37]. More precisely, we define that a patch is a maximal connected planar region such that all its points are seen by the same set of markers. Thus, the patches partition the environment. Each patch has a signature, that is the set of markers that see its points. Adjacent patches must have different signatures. However, depending on the environment and placement of markers it is possible two nonadjacent patches can have the same signature. In a binary sensor network, a target's localization belief is tied to the area of the patch within which it is sensed [37]. Given k sensors (or markers) the total number of possible signatures that can occur is 2^k. Although 2^k implies an exponential function, this requires that all visibility regions overlap with all other visibility regions. In practice, this is quite rare because ensuring effective coverage of an area or targets will likely result in clusters of markers at different areas and having k be large enough to create this concern for a single area is likely the result of a highly ineffective placement of markers. Because the area of a patch relates to the localization belief of a target (in this case the target is our robot), the area of patches can be used as a metric to evaluate the effectiveness of a marker placement.

Let $M = \{m_1, m_2, ..., m_k\}$ denote a placement of k markers, where m_i gives the position and orientation of the ith marker. The robot traverses through the environment through a sequence of m waypoints $W = \{w_1, w_2, ..., w_m\}$. The distance between waypoints should be small enough that the robot should be able to move to the next waypoint with small error by only *dead reckoning*. Because the robot must traverse through m waypoints W, the metric we use is the area of the patch covering a waypoint $w_i \in W$. Specifically, we define the overall effectiveness $g(E, M, W)$ of markers M in an environment E as:

$$g(E, M, W) = \sum_{i=1}^{m} f(E, M, w_i) \tag{3.14}$$

where $f(E, M, w_i)$ gives the area of the patch that contains the waypoint w_i. The function $g(E, M, W)$ that evaluates the effectiveness of a placement of markers will be our *scoring function*. Given the waypoints W and environment E the goal then becomes to find a marker placement M of k markers that makes $g(E, M, W)$, our scoring function, as small as possible.

To simplify the problem: markers and waypoints are discretized to work within an occupancy grid. A discretized approximation of a marker's visibility region is a *discretized annulus sector* (see Figure 3.6b). The environment may block the visibility of the discretized annulus sector through obstructions, such as walls (Figure 3.6a). We call the grid that takes into account the line-of-sight of a discrete annulus sector a *visibility grid* (Figure 3.6c). A naive line-of-sight test can be quickly implemented that tests whether all the points in the visibility grid are visible from the marker's (x_m, y_m) position, checking if any point along a line from the marker position to the

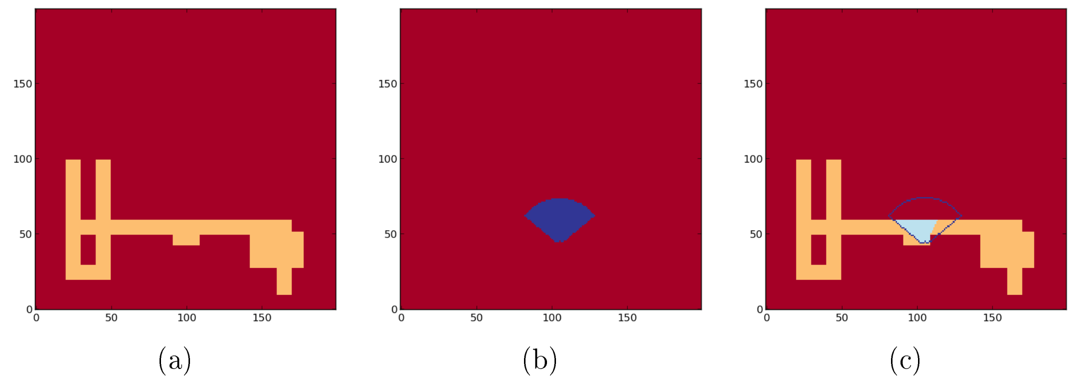


Figure 3.6: a) The environment as an occupancy grid (orange is navigable area). b) The discrete annulus sector (blue). c) The visibility grid (teal) overlayed in the environment. The visibility grid considers line of sight. The visibility grid is outlined by the discrete annulus sector (blue).

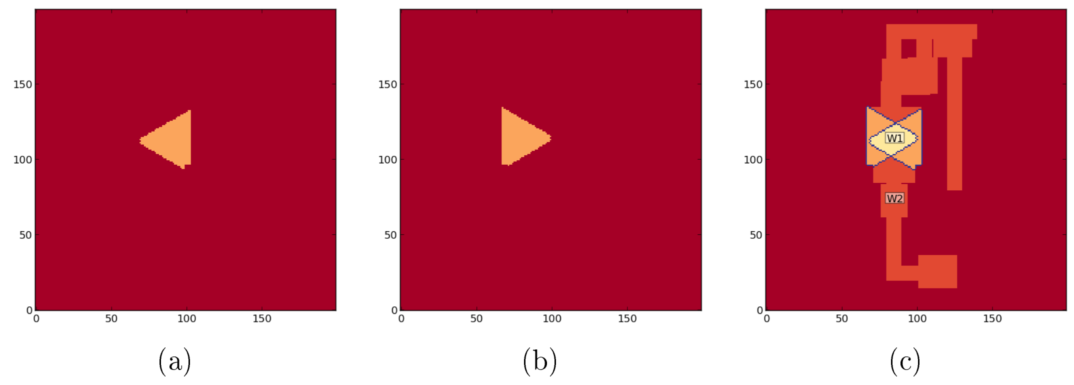


Figure 3.7: a) Visibility grid 1; b) Visibility grid 2; and c) The overlay grid comprised of the two visibility grids and environment occupancy grid (the blue lines are added to help distinguish the shape of the visibility grids). Two example waypoints W1 and W2 are included. Using this marker placement, W1 will have a score of the area of the yellow region where the two visibility grids overlap. This marker placement will also give W2 a score of the area of the large, orange, southern region of the occupancy grid where no visibility grids are present.

point in the discretized annulus sector intersects with an obstruction.

Because we are working in the discrete realm, this allows us to work with a finite number of possible poses for the markers. We are using planar fiducial markers, so we make the design decision that each marker must placed against a wall. By placing the markers against walls, the orientation of markers becomes fixed and all possible positions for marker placements are the grid points along the boundary of the environment occupancy grid. All possible poses can then be represented by a set S of visibility grids corresponding to all possible marker poses. For our discrete purposes, we define a marker placement M' as $M' \subset S$. To be clear, a marker placement M' is a collection of k visibility grids representing a discrete placement of k corresponding marker poses.

The visibility grids that form a marker placement M' can be overlayed on top of one another within the environment to form an *overlay grid* (see Figure 3.7). Different patches in a placement M will have different values based on the total number of the overlayed visibility grids at a given grid point in the overlay grid. These different *discrete patches* (the continuous set of grid points contained in a patch) can be naively labeled by iterating over the unique number of overlaps present using a common *black and white labeling algorithm* on each unique overlap. Such a black and white labeling algorithm can be found in many textbooks (i.e. [38]). The total number of labels after this process will then be the total number of patches. Finding an approximation of the area of the patch that contains a particular waypoint w_i can be done by first finding the label L_i that corresponds to waypoint w_i and then finding the total number of occurrences of L_i. This results in an approximate area of the visibility grid $f'(E, M', w_i)$, which is the discrete version of $f(E, M, w_i)$ (see Figure 3.7). With $f'(E, M', w_i)$, we redefine our scoring function $g'(E, M', W)$ for the discrete case:

$$g'(E, M', W) = \sum_{i=1}^{m} f'(E, M', w_i) \tag{3.15}$$

We will be using Equation 3.15 from this point on in our simulations (Chapter 4) and experimentation (Chapter 5).

3.2.1 A Theoretical Maximum Distance Deviation

Although we are using an approach to minimize the overall area of the patches of the waypoints, an alternative approach could be taken similar to that by Beinhofer et al. [22] that attempts to provide guarantees of *maximum deviation* – the maximum distance of the robot's true location from a waypoint. Although this topic is left for future work, we thought it useful to mention a simple geometric heuristic for a trivial open space environment with homogeneous markers that can be placed anywhere and oriented in any direction and where possible line-of-sight blockage is not considered. For each waypoint, place a circle of radius r_w centered at the waypoint. Place three

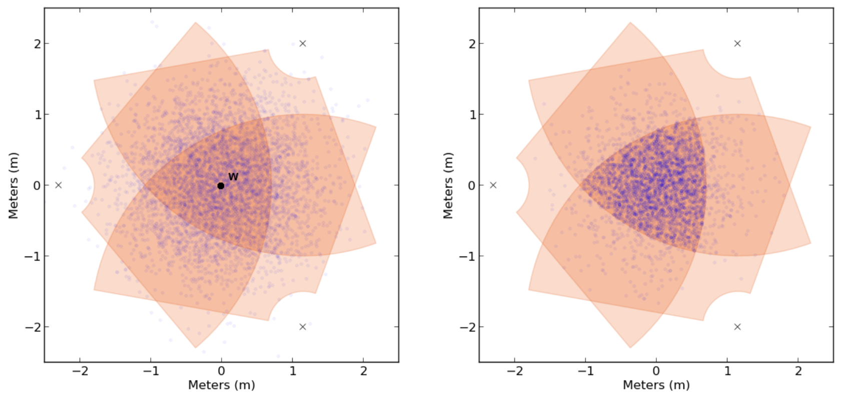


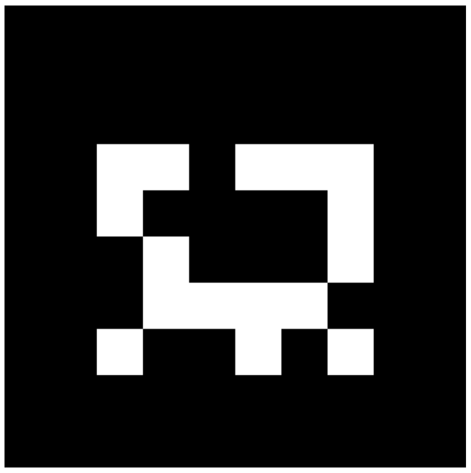
Figure 3.8: The patch formed around the waypoint w by the markers at positions marked "x" is a Reuleaux triangle. Using the uniform sensing model seen in Figure 3.5a. Here are the particles: a) before observing and b) after observing. After observation, the particles are more densely inside the Reuleaux triangle patch.

markers uniformly at regular angular intervals around the circle with each facing the waypoint. Suppose that all of the markers have the same maximum sensing distance R, minimum sensing distance r, and maximum sensing angle 2α. If the circle's radius r_w satisfies $r \leq r_w < R$ the patch in which all of the markers overlap will form a Reuleaux triangle (see Figure 3.8). The area of this Reuleaux triangle is $\frac{1}{2}(\pi - \sqrt{3})((1 - \frac{1}{\sqrt{3}})(R - r_w))^2$ and the maximum deviation of a point in this triangle from the waypoint would be $(\frac{1}{\sqrt{3}-1})(R - r_w)$ (see Appendix B for more details). If more than n markers are added uniformly around the circle, where $n \geq 3$ and n is odd, then they will form an n-Reuleaux polygon. A Reuleaux triangle was used because it only uses three markers, provides a tight shape with constant width and by the Blaschke-Lebesgue theorem: "*...amongst all convex domains of constant width w, the Reuleaux triangle has the smallest area...*" [39].

3.3 RedCometTag – Fiducial Planar Marker

This thesis uses a planar fiducial marker called RedCometTag (RCTag) that is heavily influenced by the ARTag created by Fiala [1]. It differs in a few minor aspects which will be described in detail below. Before delving into the details of the fiducial marker, it is necessary to justify some of the reasons for this approach. Because we are only requiring the detection of a marker, we opted to forego unnecessary complexity that is present in some of the other marker systems reviewed in Chapter 2.2. For our purposes, this required markers that were black and white, could be printed and detected on letter size paper, and that could effectively handle 25 or more markers. Although the RCTag may not have been the best choice for a fiducial marker in general, it is nevertheless sufficient for our purposes.

3.3.1 Encoding


(a) Marker 55

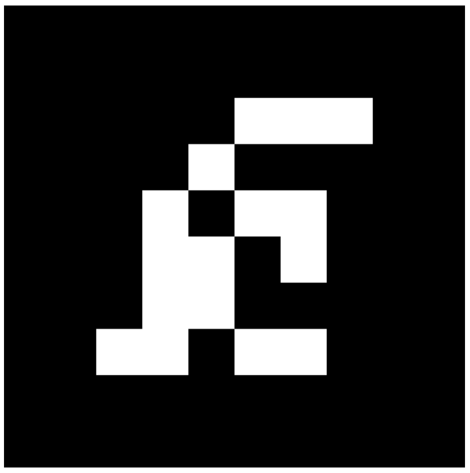

(b) Marker 456

Figure 3.9: Examples of the RedCometTag.

ARTag is a bitonal, commonly black and white, fiducial marker that *encodes* approximately 1024[3] unique marker IDs that can be quickly and robustly decoded using a camera. The fact that there are essentially 1024 possible unique marker IDs comes from the fact that ARTag uses 10 bits to encode the marker ID ($2^{10} = 1024$). To increase robustness, ARTag uses error checking and error correcting techniques from coding theory to handle errors. For error checking capabilities, 10 bits are used for a cyclic redundancy check (CRC). To potentially recover from errors, an additional 16 bits are used for Reed-Solomon forward error correcting. The 10 data bits, 10 CRC bits and 16 Reed-Solomon bits sum to a total of 36 bits for each string that represents a marker. ARTag splits the 36 bit encoded string into a 6×6 grid of bits represented as unit squares. A black unit square represents a 0 and white unit square represents

[3]Not all 1024 markers are valid or "safe". For example, the '0' marker (without the use of a mask) is a black square. In addition, the number of valid markers can be doubled with a trick that involves inverting the tone of each grid square, including the border. The exact number of valid markers for ARTag is 1001 [40].

a 1. The 6×6 grid is bordered by a two unit thick border of black squares. The two bit padded border has a few benefits. The thick border makes a marker's overall rectangle shape more prevalent in the detection phase, discussed later in Section 3.3.2. Additionally, the border can help reduce false positives in the decoding process, as a detected marker should have a border that is uniformly 0.

As its name suggests, ARTag was designed with a focus for use in augmented reality. In augmented reality, it is key to know the 3D pose of an object that one intends to highlight or alter. In a pinhole model camera, at least four known points are required to create a perspective matrix that allows one to potentially map 2D points/pixels in an image to 3D points in the real world. ARTag is comprised of a square, and thus when recognized, its four corners can serve to determine the perspective matrix, if its position is known. In reality, cameras have intrinsic errors that can affect the mapping of 2D points/pixels in an image to 3D real world points using the perspective matrix. This can be corrected by calibrating a camera using well known techniques [41, 42]. The process for calibrating a camera can be quite involved[4]. This thesis does not adopt this feature of the marker as the focus is on a binary observation model, so its implementation would be unnecessary.

RCTag is very similar to ARTag, but differs with a few cruder, less effective simplifications (see Figure 3.9). Like ARTag, RCTag uses 36 bits, but instead of using 10 data bits like ARTag, RCTag uses 12 data bits. The reason for this is that RCTag bypasses the 10 bit CRC and uses a 24 bit Reed-Solomon code, specifically part of a 255 byte Reed-Solomon encoder. This simplifies the implementation, but can lead to issues with the minimum hamming distance between markers. One way to significantly reduce *false positives* and *interconfusion* (discussed in Section 3.3.5) is to ensure that the minimum hamming distance between all marker encodings is *large*. The minimum hamming distance for a given marker can be determined by finding the minimum hamming distance between all markers and their 3 rotations. A histogram of the overall hamming distance between all markers can be seen in Figure 3.10. There are 220 "bad" codes that have a minimum hamming distance of 0 or 4, making them particularly vulnerable for misreading, so they are not used. Other than these bad codes, all other codes have a minimum hamming distance of 6 against any other code and rotation. A minimum hamming distance this low increases the chances of false positives and interconfusion. Another issue with this Reed-Solomon simplification is that it uses an odd number of bytes (24 bits = 3 bytes). Because of the nature of

[4]There are techniques being developed to make camera calibration less effortful [43, 44].

Reed-Solomon codes, there are no byte error correcting gains with 3 bytes over using only 2 bytes. The main reason for these changes was that it made implementation simple and the markers functioned effectively enough for experimentation purposes.

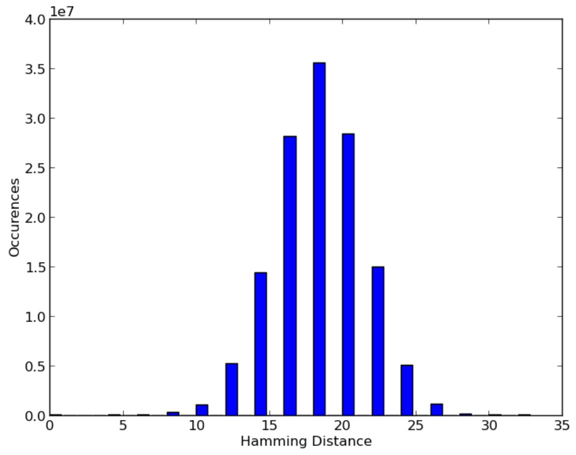


Figure 3.10: A histogram of the hamming distance between all *valid* codes and their three rotations.

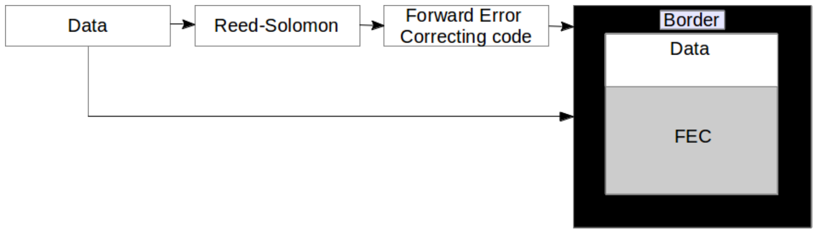


Figure 3.11: A visualization of the encoding process. The marker data is input through Reed-Solomon to produce a forward error correcting code. The data and forward error correcting code are concatenated and split into a 6×6 matrix of bits to produce the RedCometTag.

3.3.2 Decoding

The decoding process of RCTag resembles the decoding process of ARTag. First, colours are not necessary, so the image is converted to grayscale. To reduce noise in the image, the image is smoothed by convolving it with a Gaussian kernel (see Figure 3.12b). A major component of the decoding process is determining possible markers or more simply: detecting *valid* rectangles. To detect rectangles, the edges of the image must be determined, see Figure 3.12d. Edges are found using the classic

Canny Edge Detector algorithm [5]. The detected edges are used to find contours, specifically the corner points that define contours, see Figure 3.12e. To make detection more robust to large amounts of noise, this process is repeated on two down-sampled versions of the image, where the images are down-sampled in size by half, then a quarter. Any contours they detect are scaled up by the inverse of their down sampling scale (i.e. if the down sampled image scale is half, its detected contour points need only be scaled by 2). Valid rectangles can be easily identified from the set of all detected contours by checking if the contour has 4 corners and that the area formed by the contour is above some minimum area. Through testing, this minimum area was found to be 500 pixels^2. If a contour meets these criteria it becomes a candidate marker.

Candidate markers need to be evaluated to recognize whether they are valid RC-Tags. The recognition process begins by using the four corners of the rectangle as four necessary input points to solve a *perspective matrix*. A perspective matrix is essentially a matrix describing the translation, rotation and skew values needed to map points from one plane to another. To ensure a suitable perspective matrix, the corners used to build the perspective matrix should be relatively accurately known. This requires that the detected corners we used be determined down to a sub pixel accuracy, which is accomplished by using a sub pixel window algorithm.

With an accurate perspective matrix we can map sample bit points to their corresponding pixels in the detected rectangle, see Figure 3.12g. If the pixel intensity at a sample point is dark it should be a 0, and if it is bright it should be a 1. Due to variable lighting conditions, we using the Local Mean C Adaptive Thresholding technique [45] to find a valid threshold to make bit sampling more robust to lighting variations, see Figure 3.12c. The pixel values of these sample points are then used to form a 36 bit string that can be quickly decoded using a 255 byte Reed-Solomon decoder. The orientation of the viewed marker is unknown. As a result, all 4 orientations of the 36 bits must be evaluated to check if the RCTag in the image has been rotated. In the 6×6 square matrix form, the 36 bit string can be rotated clockwise quickly by first multiplying on the left by a matrix for reflecting about the line $y = x$ and then taking the transpose of the result:

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$(BA)^T = C$$

Here A is the 6×6 square matrix of the 36 bit string, B is the reflection matrix for reflection about the line $y = x$ and C, the result, is A rotated clockwise by $\frac{\pi}{2}$. If a valid bit string is successfully decoded using Reed-Solomon then an RCTag has been recognized (Figure 3.12h).

3.3.3 Parameter Estimation

To determine the visibility of markers for cameras on the robot in practice, images of a marker at various distances and angles were collected. The RCTags generated for these experiments were 600 pixels $\times$ 600 pixels that when printed on standard letter sized paper were approximately 15.5 cm in width and height (see Figure 3.14). For each distance and angle point of collection, fifty images were taken in artificial lighting and fifty images were taken in natural lighting. There were two types of cameras used by the robot: the *Microsoft LifeCam 3000* (MLC3000) and the *Logitech C310* (LC310). Parameters for both cameras had to be estimated. This resulted in a collection of over 17,000 images (see Figures 3.14 and 3.15). Using this collection of images, the effectiveness of parameters for each camera could be identified by the overall number of times the marker was detected in these images. There were three main parameters that were evaluated: the amount of Gaussian blur (σ), the threshold window size as a percentage of the total area size (t_w) of images, and the C threshold parameter (t_c). The parameters can be dependent on one another, but for simplicity the parameters were maximized sequentially. First, σ was evaluated with $t_w = 0.3$ (0.3 is for a 30% window size) and $t_c = 0.0$ (see Figure 3.13a). The peaks for σ occur at about 0.5 and 0.3 for the MLC3000 and LC310 cameras respectively. Next, the window size percentage was evaluated using the peak σ parameters just described and with $t_c = 0.0$ (see Figure 3.13b). The peaks for t_w occur at about 0.9 and 0.1 for the MLC3000 and LC310 cameras respectively. With the peak values of σ and t_w for the cameras estimated, we then estimate t_c using these values (see Figure 3.13c). The peak of t_c occurs at about 21 and 7 for the MLC3000 and LC310 cameras respectively. With the three parameters estimated for both cameras, we can now evaluate the distance and angle visibility resolution for the markers. This will ultimately guide the parameters that describe the visibility annulus sector of our planar fiducial marker used in simulation (Chapter 4) and experimentation (Chapter 5).

3.3.4 Range and Angle

To facilitate optimal marker placement, we need to know the visibility annulus sector for the RCTags. Experiments were conducted to determine the range over which the RCTag could be consistently recognized. Our experimentation involves traversing a hallway that may not be well lit in some areas, so the images captured also included natural lit images of the marker (see Figure 3.14). The experimental results show that, as might be expected intuitively, consistent detection reduces as distance increases (see Figure 3.16a). RCTag recognition has significant drop-offs before 0.7 and after 4.5 meters.

To form the visibility annulus sector it is also necessary to know the angular range over which RCTag has consistent recognition. Experiments were conducted rotating the marker at different angles to the camera from different distances, see Figure 3.15. The results (Figures 3.16b and 3.16c) demonstrate that recognition begins to fail at around 30^o when rotated left or right from facing the camera at certain distances.

The results show that the cameras detect the marker with different consistency under different circumstances. To encapsulate the visibility of both cameras into a single visibility annulus sector, we define the shape parameters of the visibility annulus sector to be $\alpha = 30^o$, $r = 0.7$ meters and $R = 4.5$ meters.

3.3.5 False Positives, True Negatives, Interconfusion and Occlusion

RCTag is effective enough for our localization experiments, but it still requires mention of some issues, specifically, the detection of *false positives* (detecting the presence of a marker when none is present), *true negatives* (not detecting a marker when it is present) and *interconfusion* [1] (detecting a marker, but recognizing the incorrect code).

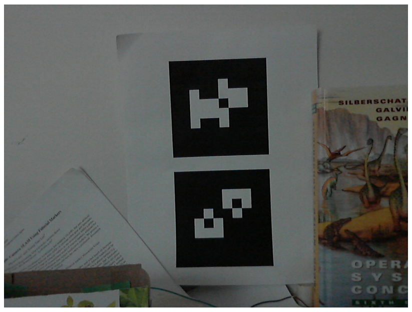
False positives can be an issue for RCTag when it is in the presence of a highly variable, rectangle cluttered area (e.g. boxes, bulletin boards containing documents). Although it does not consistently succumb to false positives, there are cases where it will sporadically detect a marker in the clutter. This can best be demonstrated in Figures 3.17c and 3.17d where an obscure box within a poster is detected as marker 4. Interconfusion can occur in a similar fashion to false positives, where a marker is incorrectly read and the hamming distance is small enough to read it as another marker ID.

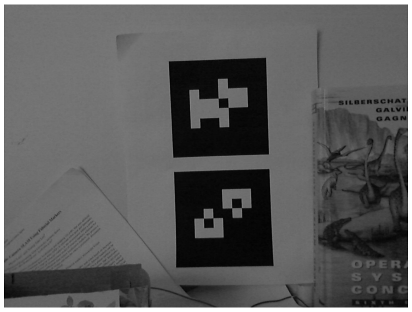
True negatives tend to occur most in the presence of *extreme* lighting conditions and when the marker is obstructed. Marker recognition will usually fail when the image is too dark as in the case of Figure 3.17a or when the image is too bright or subject to spectral reflection as in the case of Figure 3.17b. Although the autothresholding discussed in Section 3.3.2 can correct for some variance in indoor lighting conditions, it does not perform as well under these circumstances.

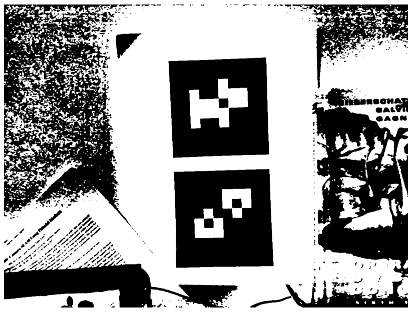
Other fiducial markers can handle some level obstruction relatively well, but RCTag fails when occluded, even partially.

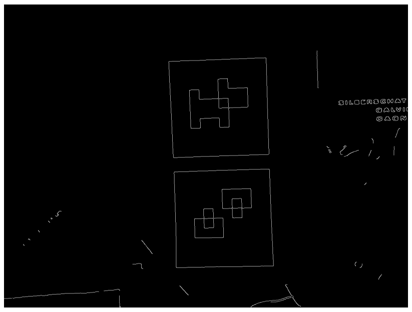
3.3.6 Implementation

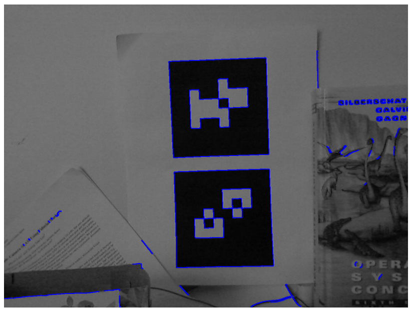
The implementation for RCTag, both encoding and decoding, was written in Python, and used the OpenCV, reedsolo [46](a Reed Solomon python package) and Numpy/Scipy libraries. The code was also influenced by the talk given by Mike Thompson [47]. The source code for RCTag can be found at *https://bitbucket.org/raw/redcomet.*

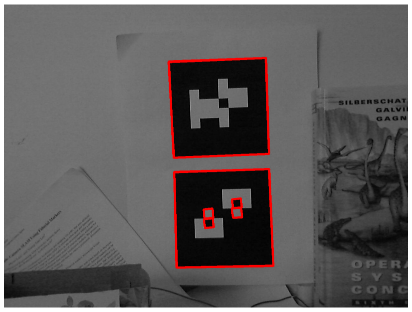

(a)

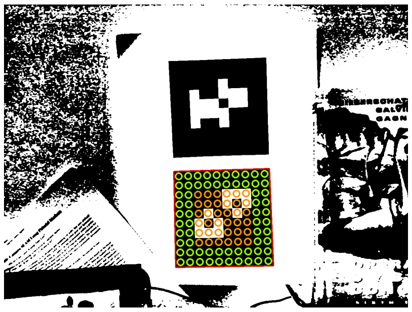

(b)

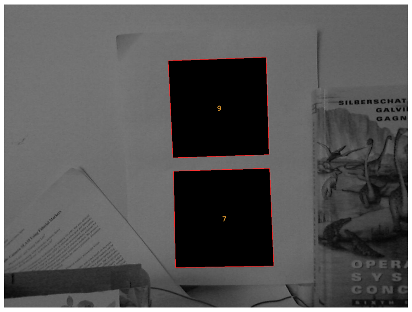

(c)


(d)


(e)

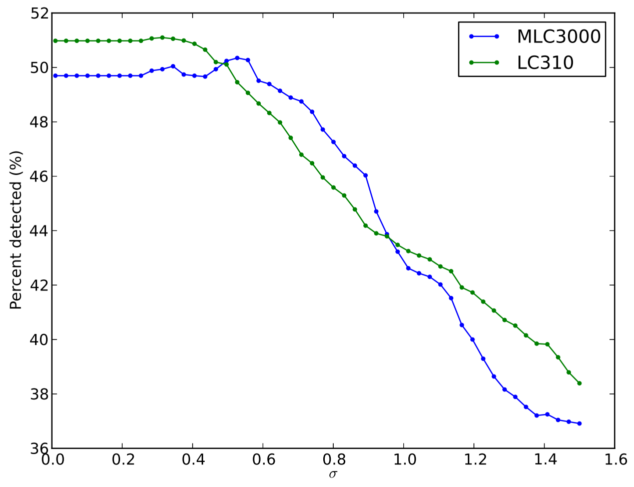

(f)


(g)

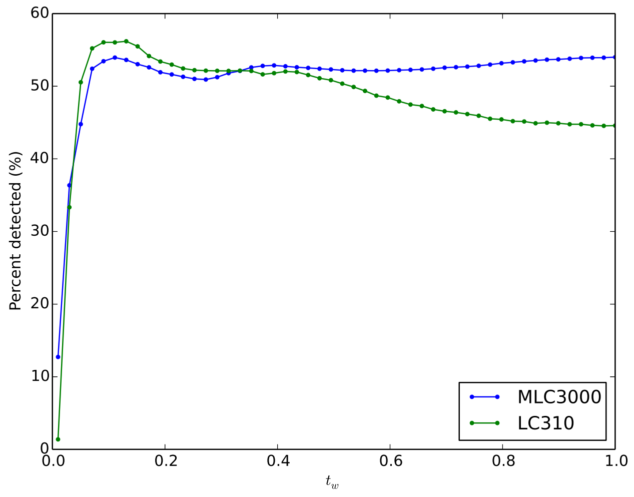


(h)

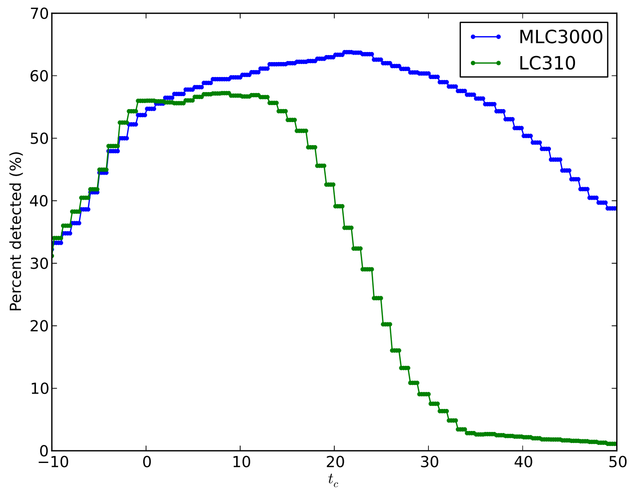
Figure 3.12: Process of decoding a RCTag a) the original image, b) the image converted to grayscale and smoothed using a Gaussian kernel, c) the image adaptively thresholded using a 401 size window, d) Canny edge detection [5] on the smoothed image b), e) the contours found, f) the remaining four point contours, g) the data sample points for the perspective matrix formed by the rectangle (orange circles are data points, green circles are border points) and h) the final resolved markers.



(a)

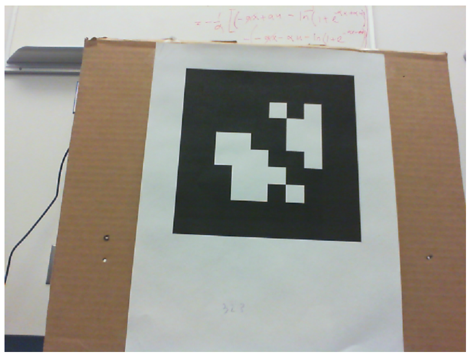


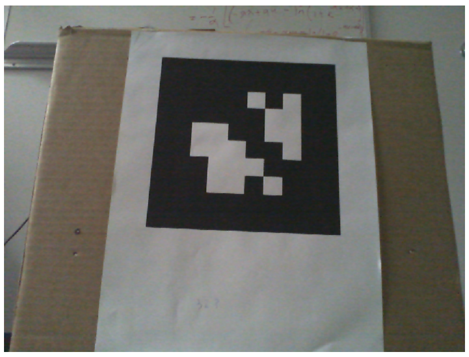
(b)

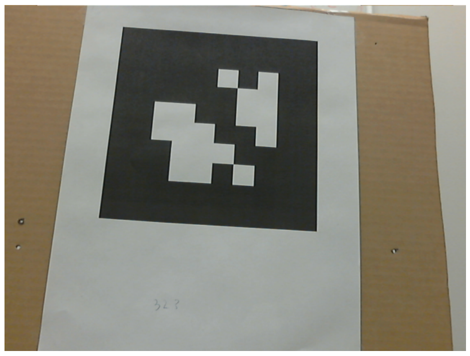


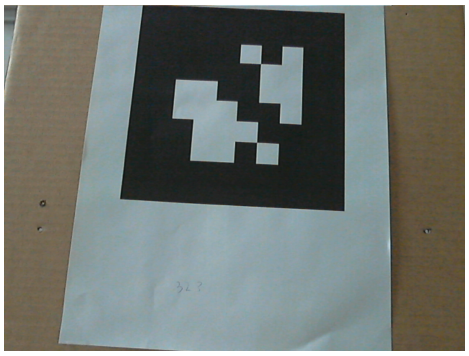
(c)

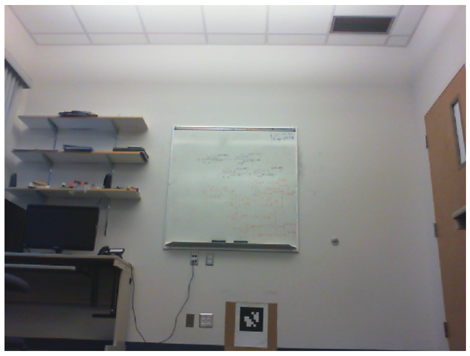
Figure 3.13: The three parameters evaluated against the successful detection of a marker taken at various distances and angles: a) the Gaussian blur σ parameter; b) the threshold window size percent parameter t_w; and c) the C threshold parameter t_c.


(a) MLC3000: 0.5 m

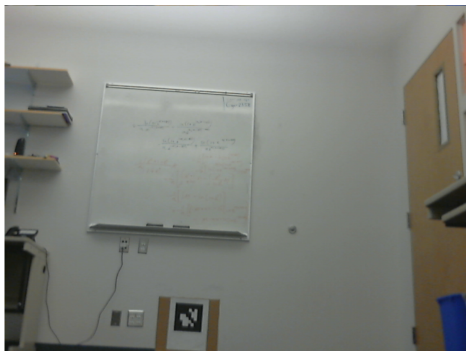

(b) MLC3000: 0.5 m


(c) LC310: 0.5 m


(d) LC310: 0.5 m


(e) MLC3000: 3.5 m

(f) MLC3000: 3.5 m


(g) LC310: 3.5 m

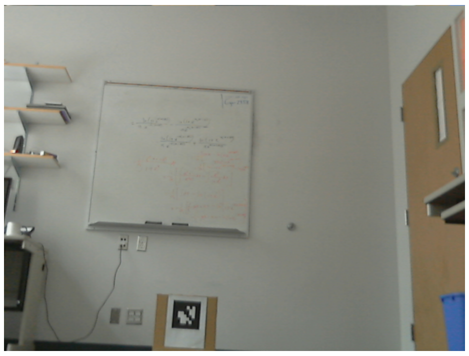
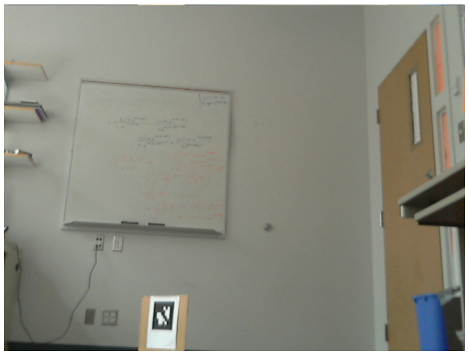

(h) LC310: 3.5 m

Figure 3.14: Example images of measurements at different distances and lighting.

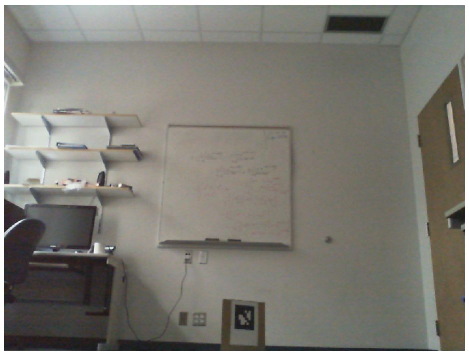
(a) MLC3000: 3.5 m, -40^o

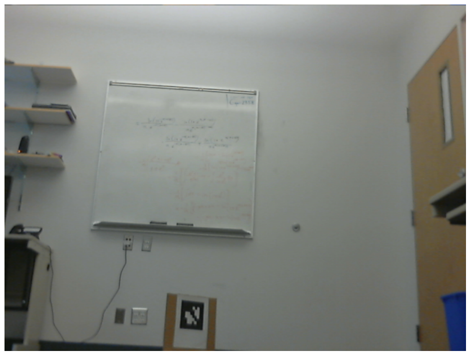
(b) MLC3000: 3.5 m, -40^o

(c) LC310: 3.5 m, -40^o


(d) LC310: 3.5 m, -40^o

(e) MLC3000: 3.5 m, 40^o


(f) MLC3000: 3.5 m, 40^o


(g) LC310: 3.5 m, 40^o

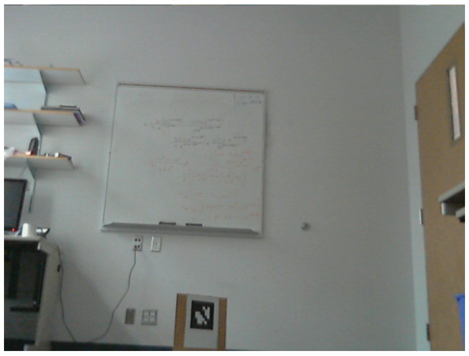
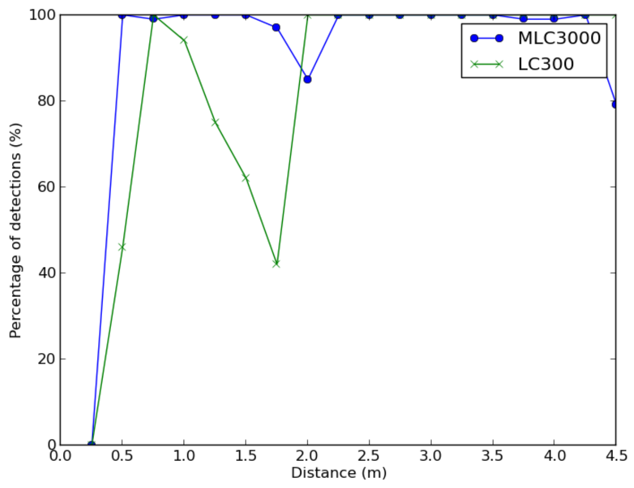
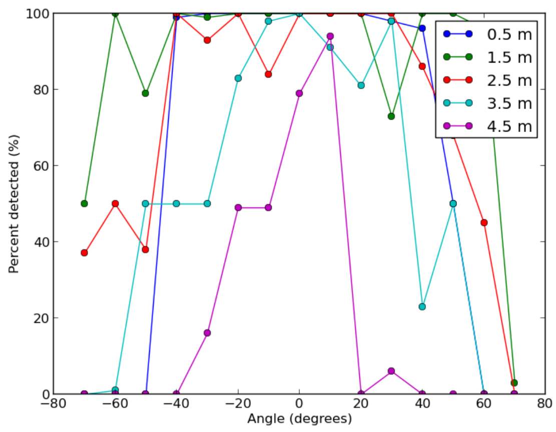

(h) LC310: 3.5 m, 40^o

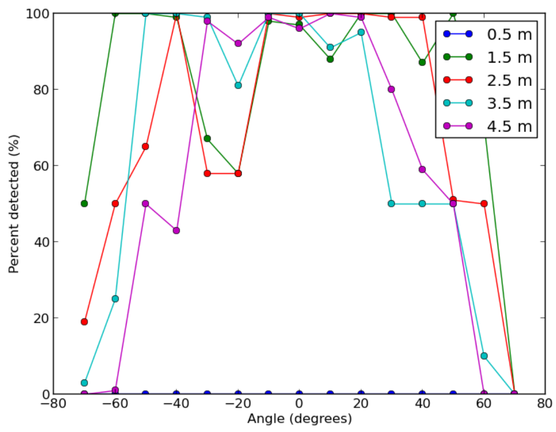
Figure 3.15: Example images of measurements with different angles and lighting.



(a)

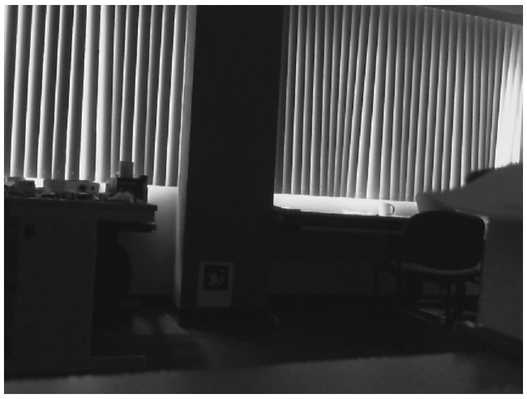


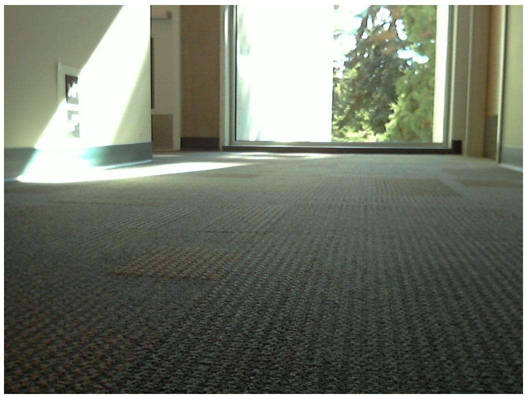
(b)

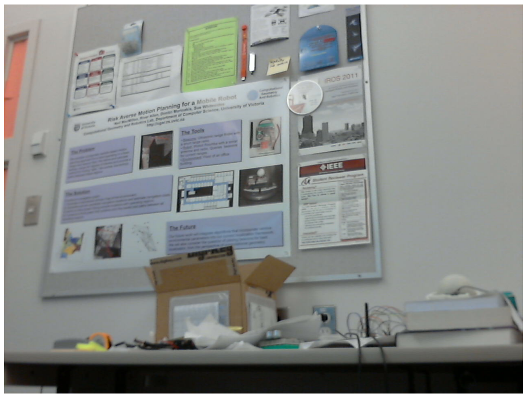


(c)

Figure 3.16: Plots describing the successful detection of a marker with different cameras: a) The marker is at a 0 angle with multiple distances; b) MLC3000 angle readings at multiple distances; and c) LC310 angle readings at multiple distances.


(a)


(b)


(c)

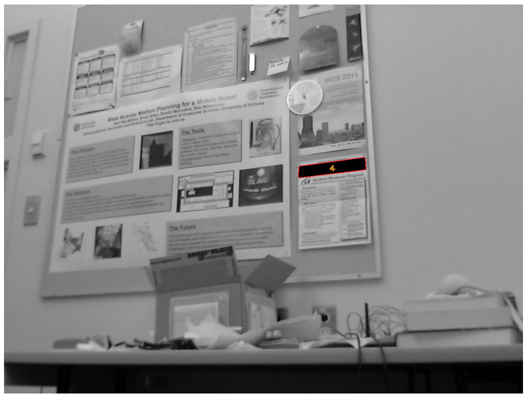

(d)

Figure 3.17: Examples of failures with RCTag: a) images that are too dark will lead to failed detection (marker at bottom center); b) images that are too bright or if the marker is subject to spectral reflection will usually fail (the markers are in the top left); c) the original source image of a false positive; and d) the red box within a poster is detected as marker **4**.

Chapter 4

Simulation

To determine an algorithm for finding good placements of markers using the proposed scoring function $g'(E, M', W)$ (Chapter 3.2), we turn to simulation to evaluate several common metaheuristic techniques. These metaheuristic techniques will score various placements of markers using the scoring function to arrive at a "good placement."

All of the simulation source code created for this thesis was written in Python and is available upon request. Python was chosen because of its cross-platform capabilities, ease of scientific computing with the Numpy and Scipy libraries, relatively fast speed, basic parallel processing functionality using the built-in *multiprocessing* library, and trivial integration with the robot code used in experimentation (Chapter 5).

4.1 Metaheuristic Optimization

Recall that in Section 3.2, we score a marker placement based on the sum of areas of the patches covering waypoints. One configuration of markers is regarded as superiour to another if it has a smaller score. The scoring process was simplified by discretizing the environment and visibility regions into a collection of grid points. This scoring function is clearly non-linear, as can be demonstrated with the following simple example. Consider an environment E and two markers A and B that both overlap a single waypoint w such that $f'(E, \{A, B\}, w) < f'(E, \{A\}, w)$ and $f'(E, \{A, B\}, w) < f'(E, \{B\}, w)$. This then means that

$$f'(E, \{A, B\}, w) < f'(E, \{A\}, w) + f'(E, \{B\}, w)$$

which implies

$$f'(E, \{A, B\}, w) \neq f'(E, \{A\}, w) + f'(E, \{B\}, w)$$

This violates *additivity* (i.e. $h(x, y) = h(x)+h(y)$), a requirement for a linear function.

Because we are working with a discrete, non-linear function we turn to common metaheuristic algorithms to find a good, though likely suboptimal, marker placement that may have many local minima and maxima. There are many metaheuristic algorithms, so we evaluate three popular choices as well as a greedy approach. We believe the scoring function we are using is likely a *submodular set function* or can be converted into one. This would imply a strict lower bound for the greedy algorithm [21].

4.1.1 Hill Climbing

In order to find the best placement of k markers with N possible marker poses where $k < N$, a brute force approach would have to evaluate all possible positions and would require $\binom{N}{k} = \frac{N!}{k!(N-k)!}$ evaluations. It is not atypical for a given environment to have $N = 180$ and $k = 30$, resulting in a total of 1.3×10^{34} combinations. To put that number into context, an estimate for the total number of stars in the universe is 3×10^{23}. That many possibilities is unfeasible to search over. Because of this, we turn to metaheuristic algorithms, and one simple algorithm that works relatively well is *hill climbing* [17].

Hill climbing works by picking an initial random placement of markers $C_c = \{M_1, M_2, \dots, M_k\}$ and evaluating it to get a score of u_c. Hill climbing then typically searches through all of its *neighbours* (defined shortly) and finds the neighbour C_n with the lowest score u_n. This neighbour then becomes the new C_c. This process is then repeated over and over, with hill climbing moving towards an optimum. For our problem, we define a neighbour placement C_n of a marker placement C_c to satisfy $||C_c \bigcap C_n|| = k - 1$, that is, neighbours only differ by one marker placement. Thus, any marker placement has $k(N - k)$ neighbours.

Evaluating the effectiveness of a given marker placement takes time, and the total number of neighbours for a marker placement can be quite large. As a result, searching over all neighbours may not be feasible. When the number of neighbours is large an adaption to the algorithm is to look at random neighbours, and if a neighbour has a lower score u_n than the current best score u_c, then that neighbour placement is chosen as the new placement C_c. This is called *first-choice* and is considered to work well enough in cases where there are a large number of neighbours [17].

Hill climbing can eventually get trapped in a local optimum, where there are no neighbours with a lower score. To get out of the local optimum, one solution is to restart the hill climbing process again with a new initial marker placement. Determining the number of iterations before hill climbing can assume it is in a local optimum requires a design decision. The number of iterations should be large enough that hill climbing can assume it has reached a local optimum, but small enough so that it is not wasting iterations looking for a better neighbour. We chose to set the number of iterations before restarting to be a percentage of the total number of iterations. The percentage chosen was 10% because it seemed to give relatively good results over other percentages.

The pseudo code implementation of our random restart first-choice hill climbing algorithm can be seen in Algorithm 3.

4.1.2 Simulated Annealing

Simulated annealing is a common iterative metaheuristic algorithm proposed by Kirkpatrick et al. [48] and is still commonly used today [17, 15]. Simulated annealing is inspired by statistical mechanics, a process of looking at how a system with a large number of particles (10^{20+}) will behave overall instead of computing the behaviour of each individual particle. An interesting property is that particle systems will, under the right conditions, configure themselves into a lowest overall energy state (or ground state) as a low temperature is imposed on the system. A system can have multiple ground states, and a process known as annealing can be used to expose these ground states. In annealing, a system is brought to a high melting temperature and then the temperature is slowly lowered, eventually leading to an alternative ground state.

This same concept is applied in optimization, but is simulated (i.e. simulated annealing). The algorithm is giving an initial placement and operates similarly to hill climbing: if a neighbouring placement has a better score, then the neighbouring placement will be chosen at the next stage. Simulated annealing differs from hill climbing in that in simulated annealing, the algorithm will randomly move to a lower scoring neighbour placement. This movement allows it to potentially jump out of a local optimum. The randomness of movement is determined by a "temperature" value. A random number is sampled and if the random number is lower than the temperature value, then a lower scoring placement is accepted. As a result, the temperature is usually defined by a temperature function that decreases as a function of the number of iterations processed. Thus, during the early iterations of optimization, simulated annealing will jump frequently, but as the temperature is lowered simulated annealing

Algorithm 3: HillClimbing(g', E, W, S, k)

```
Input: $g'$: Scoring function
Input: $E$: Environment occupancy grid
Input: $W$: Waypoints
Input: $S$: All visibility grids, $S = \{S^1, S^2, \dots S^N\}$
Input: $k$: Number of visibility grids making up a marker placement, $k < N$
1  $N \leftarrow ||S||$
2  $C_c \leftarrow$ Pick $k$ visibility grids from $S$ at random
3  $C_g \leftarrow C_c$
4  $m \leftarrow k(N - k)$
5  $u_c \leftarrow g'(E, C_c, W)$
6  $u_g \leftarrow u_c$
7  $no\_change \leftarrow 0$
8  for $i \leftarrow 1 \dots m$ do
9      if $no\_change > m * 0.1$ then
10         $no\_change = 0$
11         $C_c \leftarrow$ Pick $N$ random poses from $S$
12         $u_c \leftarrow g'(E, C_c, W)$
13         continue
14     $r \leftarrow$ uniform random integer from $1 \dots k$
15     $y \leftarrow$ uniform random integer from $1 \dots N$ and $S^y \notin C_c$
16     $C_i \leftarrow C_c$
17     $C_i^r \leftarrow S^y$
18     $u_i \leftarrow g'(E, C_i, W)$
19     if $u_i > u_c$ then
20         $no\_change \leftarrow 0$
21         $u_c \leftarrow u_i$
22         $C_c \leftarrow C_i$
23         if $u_c > u_g$ then
24             $u_g \leftarrow u_c$
25             $C_g \leftarrow C_c$
26     else
27         $no\_change \leftarrow no\_change + 1$
28 return $C_g$, $u_g$
```

will approach a local optimium. The temperature function used in our simulations is taken from Akbarzadeh et al. [15]: letting m denote the total number of iterations and i denote the current iteration step, the temperature t_i at iteration i is defined as

$$t_i = \frac{1}{2} exp\left(-\frac{2\ln 2i}{m}\right)$$

The pseudo code for simulated annealing can be seen in Algorithm 4.

Algorithm 4: Simulated Annealing(g', E, W, S, k)

```
Input: $g'$: Scoring function
Input: $E$: Environment occupancy grid
Input: $W$: Waypoints
Input: $S$: All visibility grids, $S = \{S^1, S^2, \dots S^N\}$
Input: $k$: Number of visibility grids making up a marker placement, $k < N$
1  $N \leftarrow ||S||$
2  $C_c \leftarrow$ Pick $k$ visibility grids from $S$ at random
3  $C_g \leftarrow C_c$
4  $m \leftarrow k(N - k)$
5  $score_{current} \leftarrow g'(E, C_c, W)$
6  $score_{global} \leftarrow score_{current}$
7  for $i \leftarrow 1 \dots m$ do
8      $r \leftarrow$ uniform random integer from $1 \dots k$
9      $y \leftarrow$ uniform random integer from $1 \dots N$ and $S^y \notin C_c$
10     $C_i \leftarrow C_c$
11     $C_i^r \leftarrow S^y$
12     $score_i \leftarrow g'(E, C_i, W)$
13     if $score_i > score_{current}$ then
14         $score_{current} \leftarrow u_i$
15         $C_c \leftarrow C_i$
16         if $score_{current} > score_{global}$ then
17             $score_{global} \leftarrow score_{current}$
18             $C_g \leftarrow C_c$
19         $T \leftarrow T - t$
20     else
21         $x \leftarrow$ uniform random real from $0 \dots 1$
22         if $x < \frac{1}{2} exp\left(-\frac{2\ln 2i}{m}\right)$ then
23             $score_{current} \leftarrow u_i$
24             $C_c \leftarrow C_i$
25 return $C_g$, $score_{global}$
```

4.1.3 Coordinate Descent

Coordinate descent [49] is another metaheuristic algorithm that was implemented for this thesis work to find "good" marker placements using the proposed scoring function. Like hill climbing and simulated annealing, coordinate descent picks an initial marker placement. With this initial marker placement, coordinate descent picks a dimension (1 to k) of the marker placement and evaluates each available marker pose to find the minimum score. It then does this for the remaining $k - 1$ dimensions, finding the marker pose in each dimension that gives the minimum score while keeping all other marker poses fixed. With k markers to be placed and N possible poses for each marker, coordinate descent must perform $N - k$ evaluations for a single dimension; thus, $k(N - K)$ evaluations in total. For fairness, this is why hill climbing and simulated annealing are run for $k(N - k)$ iterations. Coordinate descent is like simulated annealing and hill climbing in that its result depends on the initial marker placement. The pseudo code for the coordinate descent algorithm is detailed in Algorithm 5.

4.1.4 Greedy Heuristic

One of the easiest and fastest running metaheuristic algorithms for optimizing a function is the greedy heuristic. The main purpose for implementing the greedy heuristic was to determine if the other metaheuristic algorithms could demonstrate any significant improvement. The greedy heuristic implemented in this thesis resembles those seen in [11] and [13]. It finds a "good" marker placement by choosing visibility grids one-by-one.

The greedy heuristic begins with an empty set $C_c = \emptyset$ of visibility grids. Each marker in $S = \{S^1, S^2, \ldots, S^N\}$ is then evaluated to find

$$S^i = \underset{\forall S^i \in S}{\arg\min}\, g'(E, \{S^i\}, W)$$

with S_i, $C_c = \{S^i\}$. This process is then done for a reduced set S
C_c of visibility grids:

$$S^j = \underset{\forall S^j \in S,\, S^j \notin C_c}{\arg\min}\, g'(E, C_c \bigcup \{S^j\}, W)$$

Again, the current marker placement is then updated to include this visibility

Algorithm 5: Coordinate Descent(g', E, W, S, k)

```
Input: $g'$: Scoring function
Input: $E$: Environment occupancy grid
Input: $W$: Waypoints
Input: $S$: All visibility grids, $S = \{S^1, S^2, \dots S^N\}$
Input: $k$: Number of visibility grids making up a marker placement, $k < N$
1  $N \leftarrow ||S||$
2  $C_c \leftarrow$ Pick $k$ visibility grids from $S$ at random
3  $C_g \leftarrow C_c$
4  $u_c \leftarrow g'(E, C_c, W)$
5  $u_g \leftarrow u_c$
6  for each $i \in k$ do
7      for $j \leftarrow 1 \dots N$ do
8          if $S^j \in C_c$ then
9              continue
10         $C_c^i \leftarrow S^j$
11         $u_i \leftarrow g'(E, C_c, W)$
12         if $u_i > u_c$ then
13             $u_c \leftarrow u_i$
14             $C_c \leftarrow C_i$
15             if $u_c > u_g$ then
16                 $u_g \leftarrow u_c$
17                 $C_g \leftarrow C_c$
18 return $C_g$, $u_g$
```

grid that produced the best score, $C_c = C_c \bigcup \{S^j\}$. This process is repeated until $||C_c|| = k$.

The algorithm is not unlike coordinate descent, but runs in less time because the evaluation of a smaller marker placement (i.e. $||C_c|| < k$) tends to be faster than evaluating a full marker placement (i.e. $||C_c|| = k$). The pseudo code for the greedy heuristic can be found in Algorithm 6.

4.1.5 Uniform Placement

The final placement strategy we evaluate is uniformly placing a given k visibility grids around the environment. The uniform placement process is deterministic, and involves evenly placing the visibility grids amongst the possible visibility grids S. Because this even spacing has gaps, the uniform placements are modulo shifted through

Algorithm 6: Greedy Heuristic(g', E, W, S, k)

```
Input: $g'$: Scoring function
Input: $E$: Environment occupancy grid
Input: $W$: Waypoints
Input: $S$: All visibility grids, $S = \{S^1, S^2, \ldots S^N\}$
Input: $k$: Number of visibility grids making up a marker placement, $k < N$
1  $N \leftarrow ||S||$
2  $C_c \leftarrow \emptyset$
3  $C_g \leftarrow C_c$
4  $u_c \leftarrow g'(E, C_c, W)$
5  $u_g \leftarrow u_c$
6  for $i \leftarrow 1 \ldots k$ do
7      $C_c \leftarrow C_g$
8      for $j \leftarrow 1 \ldots N$ do
9          if $S^j \in C_c$ then
10             continue
11         $C_c^i \leftarrow S^j$
12         $u_c \leftarrow g'(E, C_c, W)$
13         if $u_c > u_g$ then
14             $u_g \leftarrow u_c$
15             $C_g \leftarrow C_c$
16 return $C_g$, $u_g$
```

all intermediate spaces. The modulo shift with the best score is then the returned uniform marker placement.

4.2 Randomly Generated Hallway Environments for Marker Placement

To thoroughly test the metaheuristic algorithms they need to be evaluated against a variety of different indoor office-like hallway environments. To accomplish this testing, an algorithm for generating simple office environments was developed as part of this thesis work. Examples of randomly generated hallway maps can be seen in Figure 4.1. This algorithm is based on procedural generation methods like those in [50].

Before going into the details of the algorithm, it is best to begin with its motivation. To understand the underlying idea of the algorithm we ask: *what is the purpose of a typical office building hallway*? For the most part, a hallway's sole purpose is to

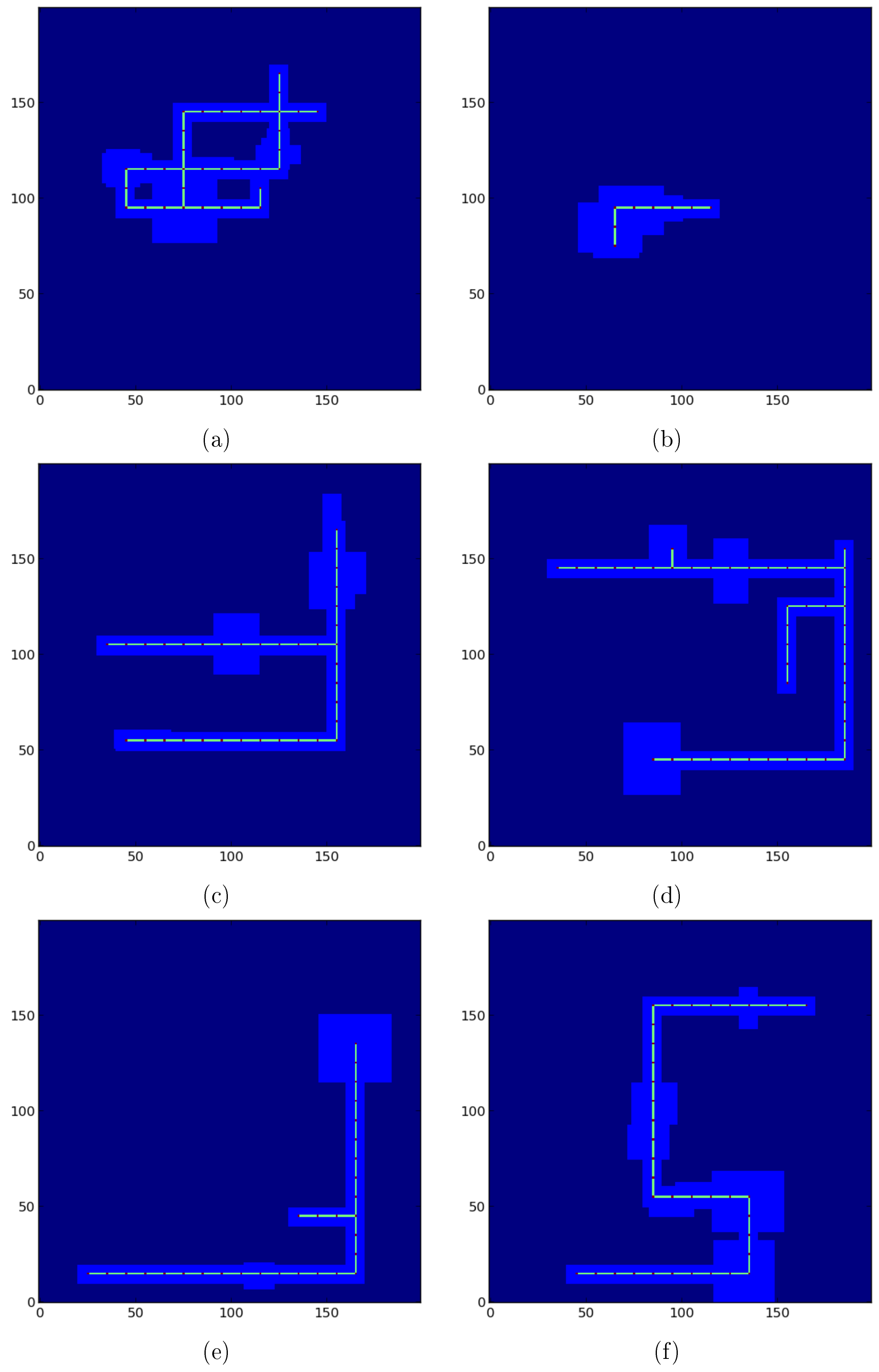


Figure 4.1: Example randomly generated hallway maps. The unit length along x and y axes is decimeters. The red dots represent waypoints with the teal lines connecting them to demonstrate potential paths.

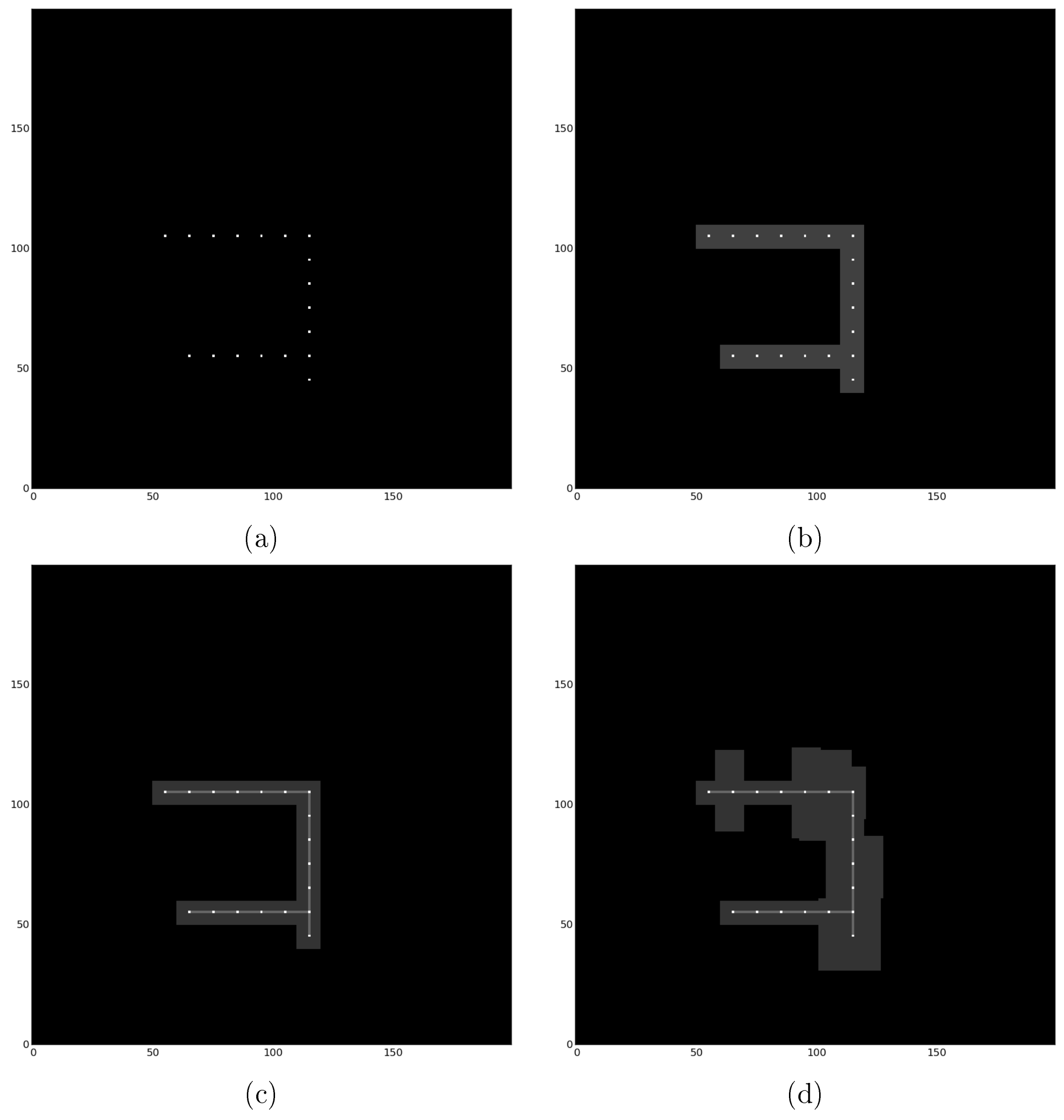


Figure 4.2: Steps for generating a random hallway map: a) generate the 'L' line segments (shown by the white pixels); b) *inflate* the 'L' line segment points to form hallways; c) generate a collection of waypoints to form a path using the initial 'L' line segments; and d) add random "rooms" and nooks to generate the final randomly generated hallway map.

serve as a "tunnel" through a building through which people can travel from point A to point B efficiently. To simplify design, construction and travel, many office hallways are orthogonal, or comprised of 90^o turns. This results in hallways that consist of a collection of 'L' line segments; see Figure 5.1a for an example of an actual office hallway. The hallway random map generation algorithm is built around this basic observation.

Initially, two points p_1, p_2 within an occupancy grid or "map" are randomly chosen, where $p_1 \neq p_2$. These two points form the first 'L' segment, where the 'L' segment is a straight horizontal line followed by a straight vertical line or vice-versa from p_1 to p_2. The total number of 'L' segments is randomly chosen in a range of positive integers. The succeeding 'L' segments are created in a similar manner to the first except that p_1 is randomly chosen from any point along the *existing* 'L' segments. The result of all points along the 'L' line segments can be seen in Figure 4.2a. These 'L' line segment points are then *inflated* to form the hallway, as seen in Figure 4.2b. A benefit of this approach is that the intervals along the line segments that form the hallways can be used to form waypoints that a robot might travel. Waypoints and the visualization of a possible path are shown in Figure 4.2c.

To make the maps more heterogeneous, "rooms" are overlayed on top of the hallway. "Rooms", in this case, are simply rectangles of randomly chosen width and height that are added at random points along the hallway. They result in more open areas that resemble nooks that might be observed in a typical hallway. These nooks give maps a less uniform and more believable look, as seen in Figure 4.2d.

4.2.1 Marker Placement Poses

Given an orthogonal hallway map, the next step is to determine possible positions for markers to be placed. In the physical world, planar markers will be placed against the hallway wall, which will fix their orientation with outward normals being normal to the wall. Given the map, this involves finding the *inner edges* of the hallway, or the discrete inner navigable boundaries of the occupancy grid (Figure 4.3c). Finding the inner edges can be achieved using well known edge and corner finding techniques [51, 5], but because our map or occupancy grid is binary and orthogonal there is a simpler approach.

A simpler approach involves using the binary morphological operation of *erosion* to find the inner edge of the hallway. Binary morphological operations are primitive

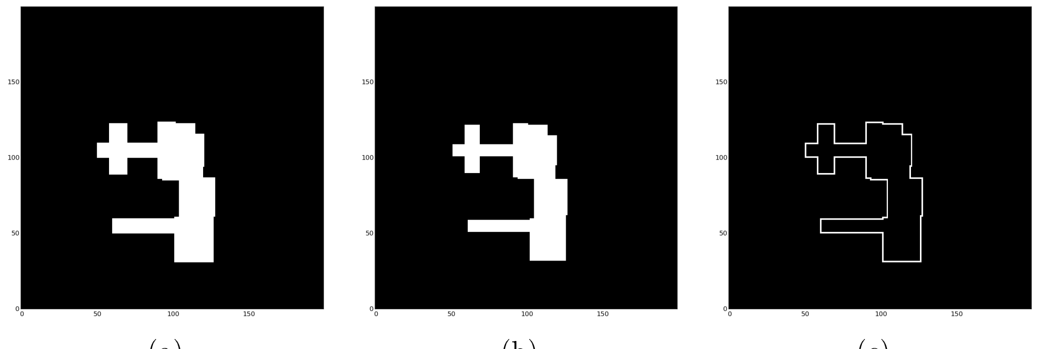


Figure 4.3: Illustrations for steps for finding the inner edges of the hallway map: a) the original hallway map; b) the map from a) eroded using a 3-by-3 square structuring element; and c) the inner edges found by the difference of a) and b).

binary operations that are framed around applying a structuring element or mask B against every grid point in a binary grid A. The erosion operation $A \ominus B = C$ overlays the center of B on every grid point p of A such that B is *contained* in A when placed at p. If every grid point in B with value '1' overlaps with a grid point of value 1 in A centered at p, then p is set to 1; otherwise, p is set to 0. Using a square, 3-by-3 structure element B given by:

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

the occupancy grid is eroded by a one grid point thick border (Figure 4.3b). The removed border comprises the inner edges of the hallway G along which the markers will be placed (Figure 4.3c). The inner edges G can be found by subtracting the original map E (Figure 4.3a) by the eroded map $(E \ominus B)$:

$$G = E - (E \ominus B)$$

4.2.2 Marker Placement Sampling

The planar markers are placed on the walls of the office environment at consistent, fixed heights. This is modeled on the 2D map of the environment as placements on the inner edges of the occupancy grid G. In the previous section, the method for finding the inner edges was described. The number of possible grids points for

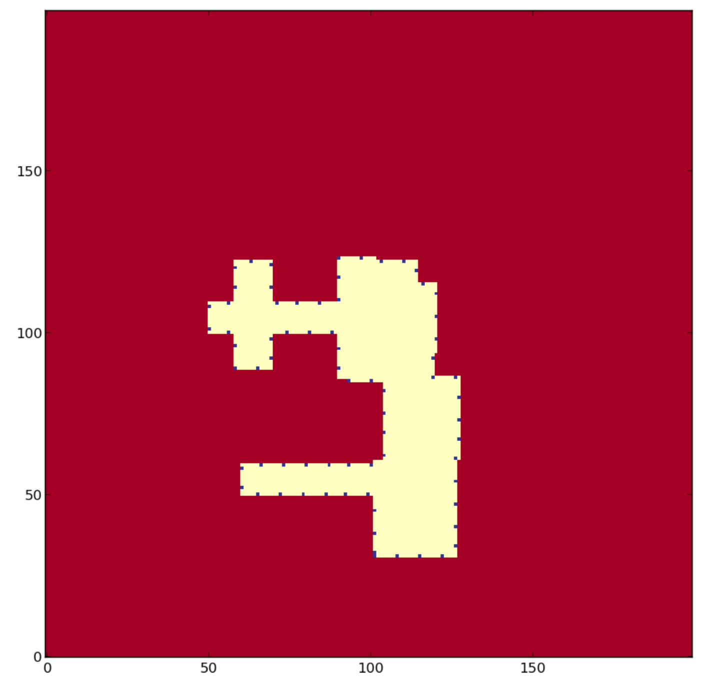


Figure 4.4: Uniformly spaced selection of 15% of all the marker poses of Figure 4.3c.

marker poses that the metaheuristic algorithms have to consider can be larger than necessary and results in much larger computational effort. To reduce the number of a possible marker positions S to be tested, the inner edges of G are sampled. For simplicity, sampling is done uniformly. An example of uniform sampling can be seen in Figure 4.4.

4.3 Results

To evaluate the performance of the metaheuristic algorithms, each metaheuristic algorithm was tasked with effectively (relative to the scoring function) placing markers in 30 randomly generated hallway maps (the maps can be seen in Appendix D). The number of markers to be placed was dependent on the size of the map. Hill climbing, simulated annealing and coordinate descent are *stochastic*, that is they give results that are non-deterministic. Because these metaheuristic algorithms are stochastic each was given 30 iterations for *fair* assessment. The greedy heuristic and uniform placement are deterministic, so they were only run once.

The results of the metaheuristic placements can be seen in Table 4.1. Additionally, the computing time results for placements are given in Table 4.2. Because the maps differ in complexity and waypoints, the number of markers used were different for each map. The number of each markers used in each map are also included.

The results indicate that a clear advantage of a certain metaheuristic algorithm is difficult to determine. The greedy heuristic (GH) produced the best scores in general, but not significantly better than hill climbing (HC) or coordinate descent (CD). The main benefit of the greedy heuristic, though, is its speed in finding a placement solution. Uniform placement (UP) gave poor results for the scoring function, but finished far quicker than any of the other algorithms. Simulated annealing (SA) consistently performed very poorly. It is possible that the temperature function, large neighbour space and/or limited iterations hindered its ability to find good placements.

Map ID	# of Markers	UP	HC	SA	CD	GH
000	42	1.42E+04	4.16E+03 ±2.27E+02	7.61E+03 ±3.05E+02	4.24E+03 ±2.62E+02	4.01E+03
001	40	1.17E+04	3.11E+03 ±2.02E+02	6.05E+03 ±2.89E+02	3.26E+03 ±2.20E+02	2.95E+03
002	23	7.54E+03	2.36E+03 ±1.69E+02	4.03E+03 ±2.78E+02	2.39E+03 ±1.58E+02	2.70E+03
003	9	4.07E+03	1.13E+03 ±9.26E+01	1.50E+03 ±1.43E+02	1.14E+03 ±1.82E+02	1.05E+03
004	9	5.48E+03	1.47E+03 ±1.18E+02	2.08E+03 ±1.43E+02	1.47E+03 ±1.72E+02	1.49E+03
005	25	4.34E+03	2.15E+03 ±1.38E+02	3.43E+03 ±1.96E+02	2.17E+03 ±1.76E+02	2.17E+03
006	14	3.39E+03	1.95E+03 ±1.51E+02	2.69E+03 ±2.01E+02	1.88E+03 ±1.42E+02	1.77E+03
007	34	1.67E+04	3.32E+03 ±1.42E+02	7.13E+03 ±6.28E+02	3.31E+03 ±1.91E+02	3.05E+03
008	13	3.22E+03	1.87E+03 ±1.15E+02	2.48E+03 ±1.50E+02	1.87E+03 ±1.42E+02	1.60E+03
009	12	4.53E+03	1.72E+03 ±1.49E+02	2.44E+03 ±2.00E+02	1.82E+03 ±2.02E+02	1.74E+03
010	20	6.65E+03	2.37E+03 ±1.42E+02	3.46E+03 ±2.26E+02	2.32E+03 ±1.28E+02	2.35E+03
011	6	5.40E+03	1.10E+03 ±1.09E+02	1.25E+03 ±1.36E+02	1.00E+03 ±1.22E+02	1.15E+03
012	20	5.54E+03	1.79E+03 ±1.18E+02	2.89E+03 ±2.30E+02	1.79E+03 ±1.50E+02	1.54E+03
013	30	1.13E+04	3.74E+03 ±2.31E+02	6.91E+03 ±3.30E+02	3.80E+03 ±2.56E+02	3.51E+03
014	10	2.44E+03	1.34E+03 ±1.00E+02	1.78E+03 ±2.14E+02	1.27E+03 ±1.22E+02	1.30E+03
015	15	3.04E+03	1.94E+03 ±1.73E+02	2.88E+03 ±2.28E+02	1.95E+03 ±1.69E+02	1.85E+03
016	12	3.93E+03	1.55E+03 ±1.41E+02	2.26E+03 ±1.59E+02	1.53E+03 ±1.21E+02	1.57E+03
017	28	1.06E+04	2.93E+03 ±1.49E+02	5.11E+03 ±2.14E+02	2.94E+03 ±1.73E+02	2.94E+03
018	33	1.40E+04	3.25E+03 ±1.68E+02	6.10E+03 ±4.05E+02	3.46E+03 ±1.69E+02	3.44E+03
019	6	3.51E+03	1.16E+03 ±1.40E+02	1.44E+03 ±1.84E+02	1.06E+03 ±1.45E+02	9.95E+02
020	21	4.53E+03	1.78E+03 ±1.05E+02	2.87E+03 ±1.53E+02	1.72E+03 ±1.27E+02	1.72E+03
021	24	5.69E+03	2.56E+03 ±1.08E+02	3.95E+03 ±2.39E+02	2.57E+03 ±1.42E+02	2.64E+03
022	3	1.53E+03	7.55E+02 ±7.46E+01	7.81E+02 ±1.10E+02	7.00E+02 ±9.11E+01	6.63E+02
023	25	1.48E+04	2.68E+03 ±1.92E+02	4.42E+03 ±2.40E+02	2.85E+03 ±2.45E+02	2.65E+03
024	21	4.57E+03	1.36E+03 ±7.27E+01	2.25E+03 ±1.49E+02	1.43E+03 ±1.12E+02	1.53E+03
025	15	5.70E+03	2.11E+03 ±1.40E+02	3.24E+03 ±1.72E+02	2.12E+03 ±2.49E+02	1.84E+03
026	26	8.52E+03	2.73E+03 ±1.53E+02	4.67E+03 ±2.80E+02	2.71E+03 ±1.42E+02	2.72E+03
027	14	3.32E+03	1.66E+03 ±1.07E+02	2.37E+03 ±1.72E+02	1.63E+03 ±9.38E+01	1.59E+03
028	20	6.17E+03	2.69E+03 ±1.82E+02	4.03E+03 ±2.23E+02	2.72E+03 ±2.29E+02	2.74E+03
029	7	7.11E+03	1.67E+03 ±1.29E+02	2.05E+03 ±1.99E+02	1.61E+03 ±1.49E+02	1.41E+03
ALL	19.2	6.79E+03 ±4.03E+03	2.15E+03 ±8.19E+02	3.47E+03 ±1.79E+03	2.16E+03 ±8.63E+02	2.09E+03 ±8.22E+02

Table 4.1: Score results with standard deviations on all maps for the four metaheuristic algorithms. The lower the score the "better" the placement.

Map ID	# of Markers	UH (sec)	HC (sec)	SA (sec)	CD (sec)	GH (sec)
000	42	6.01	268.13 ±9.36	230.75 ±4.60	261.04 ±12.24	162.12
001	40	6.19	223.19 ±7.93	182.60 ±2.27	215.34 ±14.19	139.41
002	23	2.52	61.26 ±3.22	53.47 ±1.02	60.43 ±4.77	40.34
003	9	0.52	4.67 ±0.18	4.24 ±0.15	4.58 ±0.35	2.83
004	9	0.60	5.74 ±0.23	4.98 ±0.15	5.51 ±0.52	3.80
005	25	2.88	70.08 ±3.55	58.45 ±0.96	65.14 ±4.56	38.94
006	14	1.00	11.39 ±0.45	10.71 ±0.28	11.40 ±0.84	8.50
007	34	3.29	157.82 ±6.09	99.34 ±1.84	131.68 ±9.80	93.33
008	13	1.02	12.50 ±0.65	11.40 ±0.31	12.08 ±0.82	8.29
009	12	0.72	8.43 ±0.40	8.06 ±0.16	8.41 ±0.56	5.47
010	20	1.50	27.87 ±0.92	25.84 ±0.51	27.93 ±1.80	16.79
011	6	0.28	1.86 ±0.11	1.74 ±0.06	1.89 ±0.13	1.34
012	20	1.80	35.85 ±1.77	30.88 ±0.67	34.98 ±2.66	23.57
013	30	4.27	127.92 ±4.70	116.74 ±1.34	129.38 ±8.93	79.10
014	10	0.74	6.09 ±0.29	5.46 ±0.19	5.98 ±0.41	3.84
015	15	1.37	16.95 ±0.70	15.43 ±0.30	16.98 ±1.21	11.25
016	12	0.82	11.46 ±0.46	10.67 ±0.24	11.84 ±0.87	7.20
017	28	3.76	111.39 ±4.10	95.98 ±1.07	107.71 ±5.33	75.24
018	33	3.99	138.50 ±6.20	100.53 ±1.26	123.85 ±7.40	82.30
019	6	0.28	1.90 ±0.07	1.76 ±0.05	1.89 ±0.15	1.39
020	21	1.86	42.17 ±1.44	36.36 ±0.78	41.77 ±2.91	29.11
021	24	2.50	63.47 ±3.59	55.85 ±1.47	62.15 ±5.33	42.12
022	3	0.10	0.28 ±0.01	0.28 ±0.01	0.29 ±0.03	0.21
023	25	2.42	77.35 ±3.48	64.68 ±1.14	74.46 ±5.32	54.07
024	21	1.79	39.33 ±2.00	32.30 ±0.73	37.89 ±2.80	24.90
025	15	1.66	26.51 ±0.90	24.09 ±0.37	26.23 ±1.99	15.79
026	26	3.34	89.93 ±3.25	71.87 ±1.10	83.93 ±5.68	59.64
027	14	0.90	14.86 ±0.78	13.04 ±0.28	14.57 ±0.78	9.72
028	20	1.82	37.72 ±1.62	35.09 ±0.67	38.52 ±2.44	24.96
029	7	0.48	3.44 ±0.09	3.29 ±0.07	3.43 ±0.23	2.20
ALL	19.2	2.01E+00 ±1.59E+00	5.66E+01 ±6.68E+01	4.69E+01 ±5.45E+01	5.40E+01 ±6.34E+01	3.56E+01 ±4.09E+01

Table 4.2: Computing time results with standard deviation of simulations on all maps for the metaheuristic algorithms. The greedy heuristic and uniform placement were only run once, so they have a standard deviation of 0 (not shown). Repeated trials were deemed unnecessary as they are clearly faster than the others.

Chapter 5

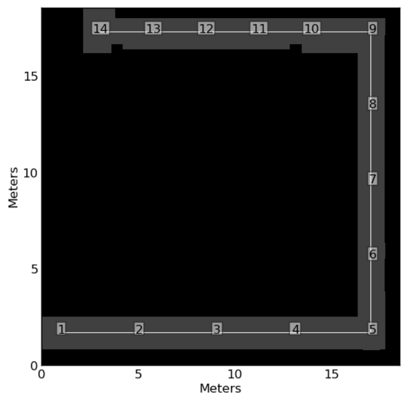
Experimentation

To test the effectiveness of proposed solutions to the marker placement problem in the real world, experiments were conducted in a real indoor office environment using printed fiducial markers and a robot equipped with cameras. The effectiveness of placements was evaluated and scored in the same manner as described in Chapter 4 with the various metaheuristic algorithms.

5.1 Environment

The environment used for experimentation is the southern half of the fourth floor of the Engineering and Computer Science Building (ECS) on the campus of the University of Victoria (Figure 5.1a). This area is a carpeted indoor office hallway (Figure 5.1b). The robot is given a set of waypoints along the center of the hallway. There are three connected hallway segments the robot traverses, for an approximate total distance of 45 meters. The hallway's width is approximately 1.6 meters except for a few areas where it can deviate by approximately ± 0.2 meters. The lighting in the hallway segments can be quite different, with some areas being artificially lit and others being lit naturally by windows. This can lead to stark contrasts in lighting, as seen in Figure 5.1c.

The building is in active use, so the hallways frequently experience bursts of human traffic. People can obscure the robot's field of view and generally make experimental runs less consistent. Although humans add excellent variability in many experiments, in this case their randomness could cause an unfair disadvantage to one of the placements. For this reason, the experiments were paused when humans approached and



(a)

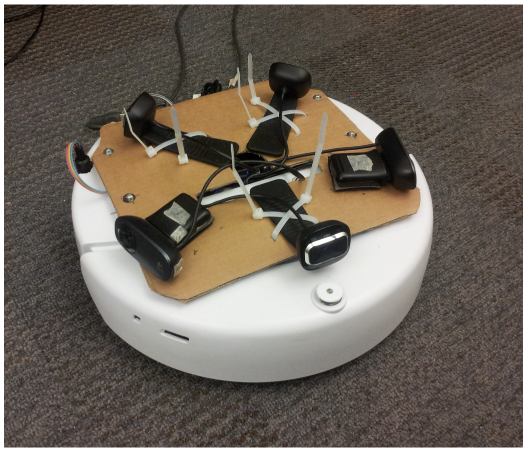
(b) (c)

Figure 5.1: a) The southern half of the fourth floor of the ECS Building and the waypoints the robot will traverse. b) One of the hallway segments the robot traverses. c) The same hallway as b), but from the opposite perspective. The light from the window can make marker detection more difficult.

were resumed when they became out-of-range.

5.2 Hardware

For our experimentation we used the iRobot Create because of its relatively low cost (Figure 5.2a). Although the Create is quite proficient at moving along the hallway carpeted environment, its onboard yaw odometry is quite inaccurate. As a result, the robot was equipped with a ITG-3200 gyro to measure the robot's change in yaw. The gyro is not a perfect sensor, so any errors the gyro incurs accumulate over time – affecting the robot. Nonetheless, the gyro makes the robot's navigation more accurate than without. To read the gyro requires a device that can read I^2C and transmit via USB. The robot was also equipped with an *Arduino Nano* which serves only to transmit the gyro's data (Figure 5.2b).


(a)

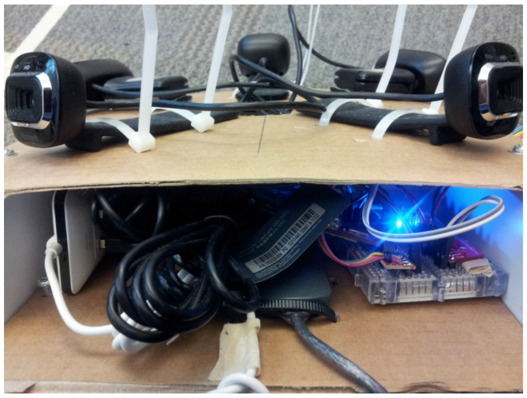

(b)

Figure 5.2: a) The iRobot Create fitted with five webcams to give the robot a near omnidirectional field of view. b) The back of the robot, showing the microcontroller and wiring.

To detect the presence of markers, the robot is equipped with five webcams distributed at equal angular intervals at the top of the robot. Three of the five webcams were *Microsoft LifeCam 3000*s (MLC3000) and the remaining two were *Logitech C310*s (LC310). Both had a cost of roughly $40 CAD at the time they were purchased. Although it was assumed in previous chapters that the robot has an omnidirectional field of view, this was not true in practice. The field of view for each of the webcams was limited to 60^o or less. This resulted in major blind spots for the robot. To lessen these blind spots, when the robot made observations, it performed an additional 36^o counterclockwise rotation (Figure 5.3) to collect more images to aid in marker recognition. After taking sufficient images, it returned with a 36^o clockwise rotation.

5.3 Software

The software to drive and localize the robot was built upon the Robot Operating System (ROS) software package [52]. ROS is a highly modular message passing system that passes messages between processes (*nodes*) locally or over a network. It closely resembles a client-server architecture, where nodes subscribe and publish to *topics* where messages are posted and read by other nodes asynchronously. ROS was selected because it is open source and has evolved a relatively good community and

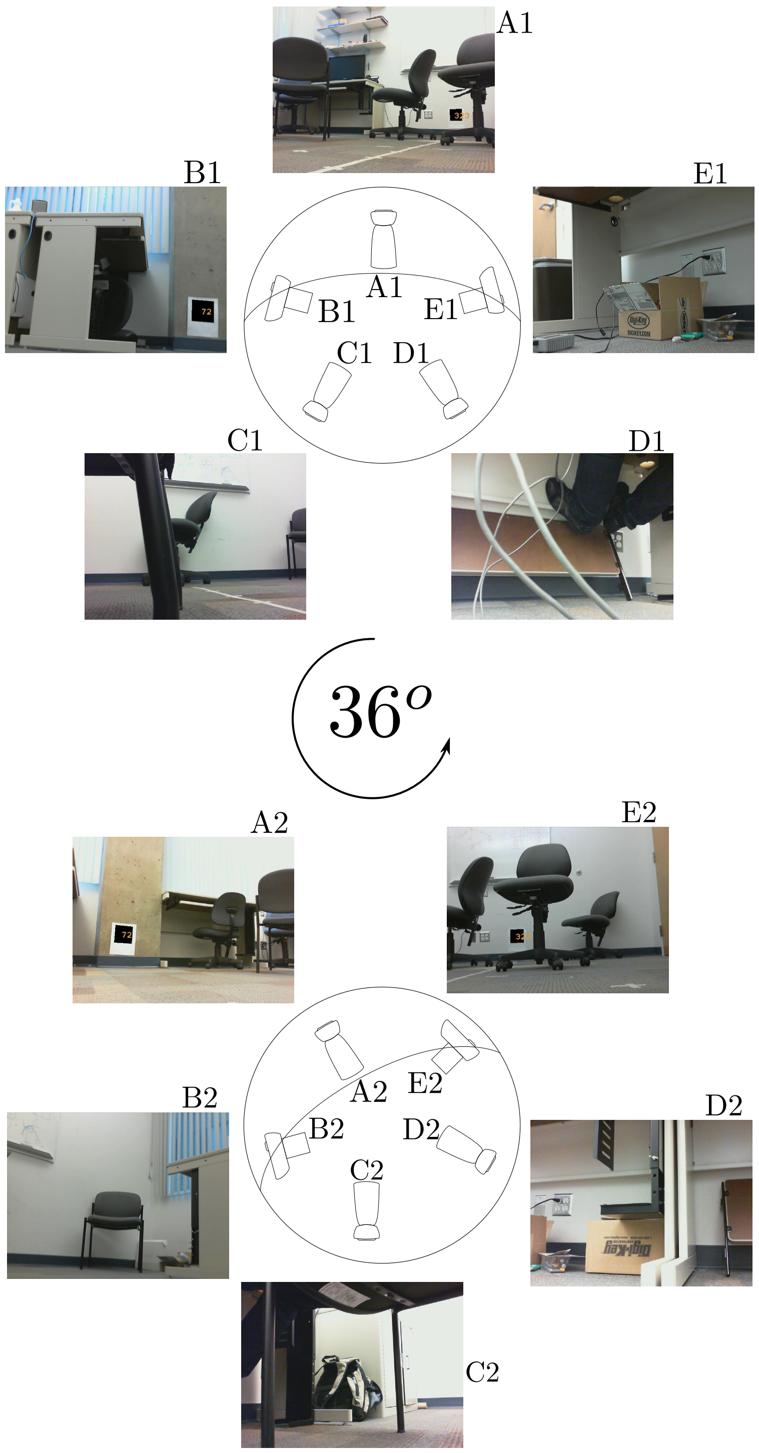
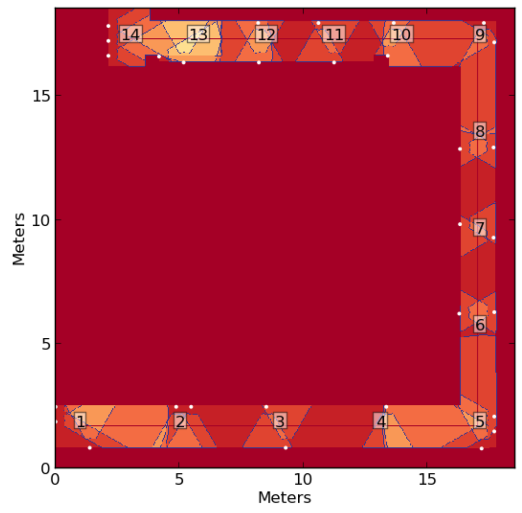


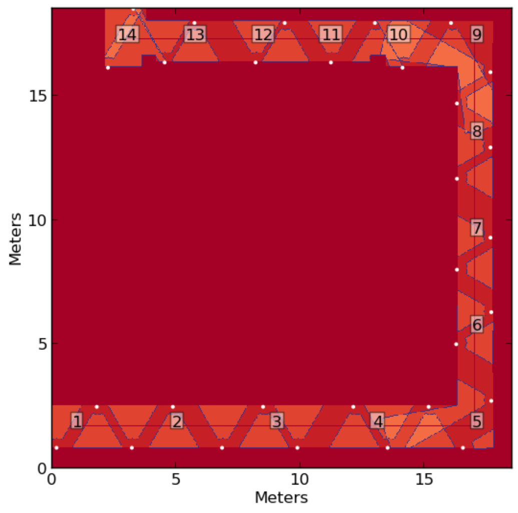
Figure 5.3: An example of the field of view of the robot using the five cameras. In the upper part of the image, the five cameras have large blind spots. By turning 36^o left and including these new images, the blind spots are fewer and significantly smaller.

documentation for embedded and computer vision applications. ROS also runs in C++ and/or Python, which allowed for easy integration of the simulation code into the experimentation code.

5.4 Results



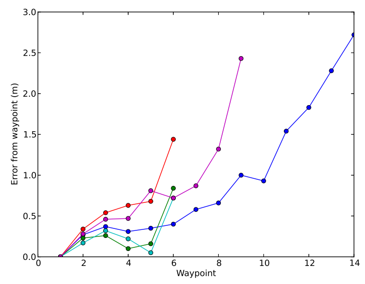
(a) CD Placement



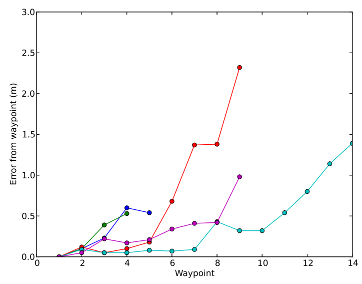
(b) Uniform Placement

Figure 5.4: The placements produced by the coordinate descent and uniform placement algorithms for the ECS map. The white circles represent marker locations. The visibility grids of the markers are overlayed on one another and each visibility grid is outlined in blue.

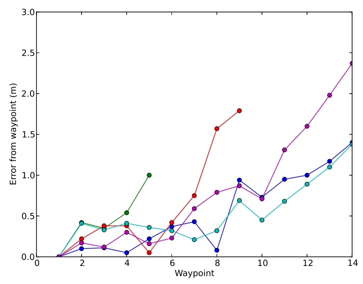
To test the effectiveness of the proposed scoring function, we ran the fourth floor ECS map through the metaheuristic algorithms detailed in Chapter 4. To decrease the number of all possible poses for the markers, we uniformly selected 5% of the inner edge points (discussed in Chapter 4.2.2). A higher sampling percent would have greatly increased the running time of finding optimal local placements, and given the resolution of the ECS map the choice of 5% did not seem unreasonable. We evaluated the highest scoring placement among all the proposals from heuristics (HC, SA, CD, GH), a uniform placement, and the robot traveling the waypoints without the use of markers. The latter choice of no markers was introduced to serve as a control benchmark and ultimately proved necessary. The highest scoring placement came from a coordinate descent placement that clumped markers around the waypoints (Figure 5.4a), as might be predicted, given the scoring function chosen. The



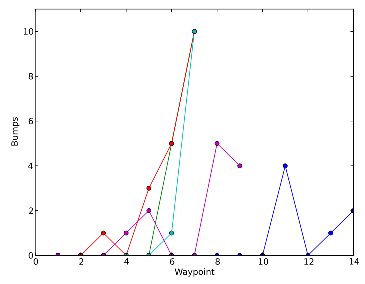
(a) No Markers



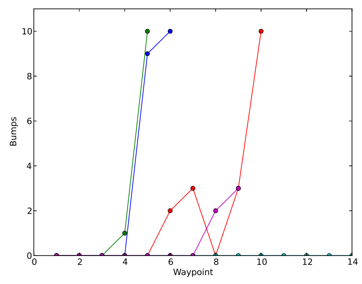
(b) Best Score Placement



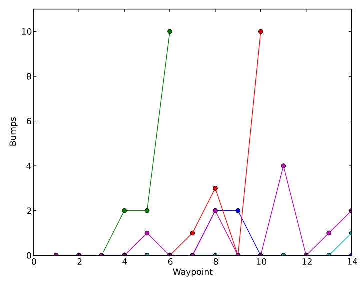
(c) Uniform Placement



(d) No Markers



(e) Best Score Placement



(f) Uniform Placement

Figure 5.5: Each colour line represents an individual run. (Top row) The measured error between each waypoint for the 5 runs of each placement of markers. (Bottom row) The number of times the robot bumped into the wall between waypoints for the 5 runs of each placement. A run is stopped as a failure if either 10 bumps occurred between a waypoint or the distance from the robot to its next waypoint was greater than four meters.

coordinate descent placement had a score of 5456 as compared to the uniform placement (Figure 5.4b) score of 71676. Experiments were then conducted to evaluate the effectiveness of these placements in practice.

Experimentation consisted of performing five runs of the robot evaluating the coordinate descent placement, uniform placement and with no markers placed. The robot traveled through the ECS hallway following the waypoints and would periodically pause to observe markers after one meter of travel or if it bumped into a wall. When the robot reached a waypoint, the robot paused and a human would measure the distance of the robot from its desired waypoint. Due to limitations with the robot, the robot was permitted to bump into the wall. The information gained from bumping into the wall was used by the particle filter. If the robot bumped into the wall ten times between traveling from one waypoint to the next, the run would be stopped and considered a failure. This policy was enacted based on observations that after about 8 or 9 consecutive bumps the robot's true pose would no longer be

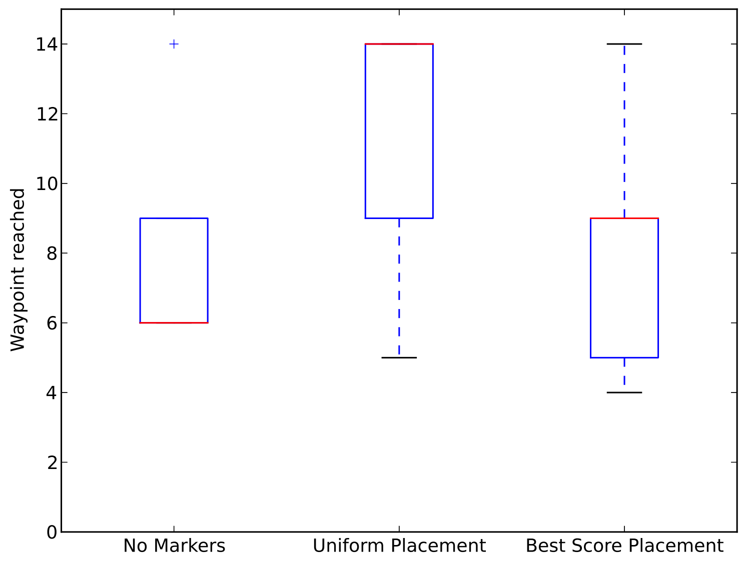


Figure 5.6: Box-and-whisker plot of the waypoints reached for each placement. See Appendix C for an explanation of the box-and-whisker plot.

accurately represented by the particle filter (discussed in Chapter 3.1). A robot run would also stop and be considered a failure if the robot's perceived position was in error exceeding four meters to the next waypoint. This was usually the case when the robot's true pose would no longer be represented by the particle filter and by chance, the robot would be traveling away from its next waypoint. Each run took approximately 25 to 35 minutes to complete, depending on the distance the robot traveled. The results of the experimentation can be seen in Figures 5.5 and 5.6. The recorded measurements for each run can be seen in Appendix E.

The results indicate that the uniform placement of markers performed better than the runs without markers and the coordinate descent placement, which was the placement with the best score out of all evaluated placements by the various heuristics. This was surprising because of the score the uniform placement had relative to the coordinate descent score using the proposed scoring function. It was also surprising how poorly coordinate descent performed – providing a marker placement that appeared no better than using no markers at all when physical experiments were done.

5.4.1 Discussion of Results

The experiments lend evidence that the proposed scoring function is not able to accurately capture the effectiveness of a marker placement.

The uniform placement tended to perform better than the best scoring placement and no markers. One possible explanation may be that the distance between waypoints (2.3 to 4 meters) was too far. Thus, the accumulated errors in the robot's motion between waypoints may be been too great. If the waypoints were more frequent, for example every 1 to 1.5 meters, the metaheuristics may have found a placement similar to the uniform placement. More waypoints would have led to patches which together would have provided more coverage of the environment.

During the runs, there were cases where the robot would fail to detect certain markers which would lead to inappropriate movements. This was especially noticeable when the robot moved around corners (waypoints 5 and 9). The proposed scoring function currently does not take into account when the robot does not observe markers that should be visible. Doing so might lead to better and more robust placements.

Additionally, having the robot look for markers after 1 meter of travel was imposed as a practical matter, to keep the experimental runs from taking overly long. More frequent observations may have greatly assisted the robot's localization capability, at the expense of time needed to travel along the waypoints.

The experimentation also exposed some flaws in the intuition behind the proposed scoring function. Although the patch area ultimately determines the localization bound for the robot, it does not appear to be sufficient for a "good" placement in practice. A patch at a waypoint that allows effective localization for the robot is only useful if the robot can reach that patch. This leads to an observation that the patch at a waypoint is not necessarily the most important factor. The patches in between waypoints and around waypoints, that collectively partition the environment, can have an impact on robot localization. The knowledge of these patches can also be used to assist motion planning. If the robot, from its current position, follows a linear path through large patches, it could be more effective to plot an alternative path to or through smaller area patches if possible. This is similar to some of our previous work [53] that looked at navigating a robot between low risk regions.

The proposed scoring function's scoring of placements did not take into account the robot's motion and error model as it traversed the environment with a given marker placement. Thus, how the robot would behave when navigating and observing an environment with a particular placement was unknown.

Possible alternative scoring functions for future work are discussed in Chapter 6.2.

Chapter 6

Conclusions

As robots increase in use and availability, mobile robotics will continue to be a field of interest and value. In this thesis, we explored an aspect of mobile robotics, namely, the placement of binary directional sensors along waypoints to assist a robot in localization in known environments. We modeled a binary directional marker's visibility region as an annulus sector – which takes into account the minimum range most sensors possess. We turned away from binary directional proximity sensors that use batteries and developed a crude, planar fiducial marker we named RedComet based on ARTag [1]. We proposed a scoring function for evaluating the effectiveness of a placement of markers that involved minimizing the patch area around waypoints. The nature of the problem required a metaheuristic approach to find an optimal placement with respect to the scoring function, so several common metaheuristic algorithms were evaluated. In addition, an algorithm was developed for generating random hallways maps. These maps were then used to evaluate the metaheuristic algorithms. Overall, the greedy heuristic appeared to give relatively good results with a better time complexity than the other cases.

To validate the proposed scoring function, physical experiments with a robot in a real hallway environment were conducted on a placement with the best score found using the proposed scoring function, a uniform placement, and finally, the same hallway environment with no markers placed. The results from experimentation implied that best score placement did not perform as well as the uniform placement.

While research projects often limit themselves in scope entirely to simulation results, we found that the inclusion of physical experiments exposed weaknesses in the scoring function. Our experimentation demonstrated the strong need for physical experiments in order for a hypothesis about a good marker placement to be honestly evaluated. Improvement of the scoring function by an iterative process, alternating between physical experiments and simulation, would be valuable albeit time consuming process for future research. Clearly, future refinements require experimental results to ensure that a scoring function reflects the reality of the robot. Although the physical results were not consistent with the "optimal" results found by simulation based on our scoring function, the pursuit of better inroads into binary sensing and placement should continue because of the relative simplicity and low cost this method promises. This thesis may serve as guidance to the challenges involved in this pursuit.

6.1 Contributions

While this thesis has explored many facets of directional binary marker placement the major contributions can be summarized as follows,

1. The evaluation of several metaheuristics were evaluated in simulation and experimentation with different parameters on the directional binary placement problem.

2. An algorithm with Python source code for generating random 2D office maps.

3. *Redcomet*: a Python library for encoding and decoding a simple planar fiducial marker similar to ARTag.

The thesis may provide an introduction and guidance to future researchers who continue research on the directional binary marker and sensor placement problem.

6.2 Alternative Scoring Functions

The physical experiments of this thesis exposed issues with the proposed scoring function, as discussed in Chapter 5.4.1. One of the main issues of the proposed scoring function is the underlying assumption that minimizing patches at waypoints is all that is necessary. If the robot misses the waypoint patch or improperly detects markers there, then there is no localization improvement. To deal with this, possible future scoring functions could score marker placements based on other patches as well as those at waypoints. For example, a possible scoring function could include the patch area of the waypoint as well as the patch areas for all patches adjacent to the waypoint patch. This could lead to marker placements that will improve localization when the robot gets near a waypoint – as the scoring function will also try to minimize all the patches around a waypoint.

Another approach to the scoring function is to focus less on the waypoints and more on coverage. That is, focus on a scoring function that leads to useful localization patches throughout the environment. This could be done in a way similar to Beinhofer et al. [22] approach where the scoring function rewards a relatively small maximum patch area. A crude approximation of this could be to have the scoring function be the area of the patch with the maximum area. Minimizing this score would lead to a placement that partitions the entire environment into relatively small patches. This could be useful for coverage and allowing a robot to use patches in path planning (as mentioned in Chapter 5.4.1).

These approaches all are still heuristics that approximate a "good" marker placement for mobile robot localization. Although they may prove effective, none of them take into account the robot actually navigating through the environment or give a bound on how well the robot will navigate under a given marker placement. One possible avenue of further study is following a similar strategy to marker placement as Vitus and Tomlin [19] and Beinhofer et al [22]. This approach would involve having the scoring function be directly related to how well a simulation of the robot would travel the waypoints given a marker placement. In their work, a sensor is only capable of giving readings to the robot when the robot's entire region of probable positions lies within the sensor's field of view. These readings give information about the robot's pose that can then be used to reduce the robot's pose error. With binary directional proximity markers, the localization value of detecting a marker is that the robot is likely in the visibility region of the marker. So, *if the robot's region of most probable positions lies within the visibility region of a marker, there is no localization benefit.* This means such a scoring function following this approach would need to be supplemented by alternative means.

One possibility is to simulate the robot navigating the environment using a particle filter. The scoring function would then be how well the robot navigates the environment given a marker placement. A specific example could be the maximum deviation of the robot's position from the path through the waypoints. A major issue with this approach is that the particle filter is non-deterministic, so for a given marker placement to be evaluated would require a number of runs to ensure statistically meaningful evaluation. This could lead to heavy computational run times just to test a single marker placement. Finding a minimum score marker placement could involve thousands of such tests.

Ultimately, any continued effort into developing a scoring function must be evaluated through physical experiments. That said, future work into finding an effective scoring function could prove very useful.

6.3 Future Work

Throughout the process of exploring this problem, there were several avenues of further study that were not pursued due to time constraints. In this section, some of these possible future paths of study are discussed.

The only localization problem explored in this thesis was that of robot tracking. It could be of interest to experiment with the effectiveness of marker placement in regards to global localization and the kidnapped robot problem. Exploring this could help demonstrate real world robustness.

Lastly, it became clear during experimentation that placement of fiducial markers in an environment shared with people could be visually unpleasant to the eye. This could become an issue of using this approach in certain environments. It would be interesting to reevaluate this approach from a human-robot interaction point-of-view with a focus on placement localization that takes into account aesthetics. This could coincide well with work on more aesthetically pleasing and/or subtle fiducial markers ([54, 55]) and may be interesting to design and carry out experiments from this perspective.

Bibliography

[1] M. Fiala, "ARTag revision 1, a fiducial marker system using digital techniques," Tech. Rep. 7419/ERB-1117, National Research Council Publication, Nov. 2004.

[2] A. Rice, A. Beresford, and R. Harle, "Cantag: an open source software toolkit for designing and deploying marker-based vision systems," in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 10–21, Mar. 2006.

[3] A. Xu and G. Dudek, "Fourier tag: A smoothly degradable fiducial marker system with configurable payload capacity," in *Proceedings of the Canadian Conference on Computer and Robot Vision (CRV)*, pp. 40–47, May 2011.

[4] A. Stathakis, "Vision-based localization using reliable fiducial markers," Master's thesis, University of Ottawa, 2012.

[5] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the Alvey Vision Conference (AVC)*, pp. 147–151, Sept. 1988.

[6] R. Allen, N. MacMillan, D. Marinakis, R. I. Nishat, R. Rahman, and S. Whitesides, "The range beacon placement problem for robot navigation," in *Proceedings of the Canadian Conference on Computer and Robot Vision (CRV)*, pp. 151–158, May 2014.

[7] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[8] S. Han, H. Lim, and J. Lee, "An efficient localization scheme for a differential-driving mobile robot based on RFID system," *IEEE Transactions on Industrial Electronics*, vol. 54, pp. 3362–3369, Dec. 2007.

[9] Y. Mesaki and I. Masuda, “A new mobile robot guidance system using optical reflectors,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, pp. 628–635, July 1992.

[10] M. Amac Guvensan and A. Gokhan Yavuz, “On coverage issues in directional sensor networks: A survey,” *Ad Hoc Networks*, vol. 9, pp. 1238–1255, Sept. 2011.

[11] J. Ai and A. A. Abouzeid, “Coverage by directional sensors in randomly deployed wireless sensor networks,” *Journal of Combinatorial Optimization*, vol. 11, pp. 21–41, Feb. 2006.

[12] D. S. Hochbaum, “Approximation algorithms for NP-hard problems,” p. 94–143, Boston, MA, USA: PWS Publishing Co., 1997.

[13] G. Fusco and H. Gupta, “Selection and orientation of directional sensors for coverage maximization,” in *Proceedings of the IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pp. 1–9, June 2009.

[14] Y. Wu, J. Yin, M. Li, Z. En, and Z. Xie, “Efficient algorithms for probabilistic k-coverage in directional sensor networks,” in *Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pp. 587–592, Apr. 2008.

[15] V. Akbarzadeh, C. Gagne, M. Parizeau, M. Argany, and M. Mostafavi, “Probabilistic sensing model for sensor placement optimization based on line-of-sight coverage,” *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 2, pp. 293–303, 2013.

[16] H. Zhang, “Two-dimensional optimal sensor placement,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 25, pp. 781–792, May 1995.

[17] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence, Upper Saddle River: Prentice Hall, 3rd ed., 2010.

[18] D. B. Jourdan and N. Roy, “Optimal sensor placement for agent localization,” *ACM Transactions on Sensor Networks*, vol. 4, pp. 1–40, June 2008.

[19] M. P. Vitus and C. J. Tomlin, "Sensor placement for improved robotic navigation," in *the Online Proceedings of Robotics: Science and Systems*, June 2010.

[20] M. Beinhofer, J. Müller, and W. Burgard, "Near-optimal landmark selection for mobile robot navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2011.

[21] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Mathematical Programming*, vol. 14, pp. 265–294, Dec. 1978.

[22] M. Beinhofer, J. Müller, and W. Burgard, "Effective landmark placement for accurate and reliable mobile robot navigation," *Robotics and Autonomous Systems*, vol. 61, pp. 1060–1069, Oct. 2013.

[23] M. Beinhofer, J. Muller, A. Krause, and W. Burgard, "Robust landmark selection for mobile robot navigation," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3637–2643, Nov. 2013.

[24] D. P. Meger, "Planning, localization, and mapping for a mobile robot in a camera network," Master's thesis, McGill University, 2007.

[25] G. Dudek, J. Sattar, and A. Xu, "A visual language for robot control and programming: A human-interface study," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2507–2513, Apr. 2007.

[26] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3400–3407, May 2011.

[27] A. Stathakis and E. Petriu, "Robust pseudo-random fiducial marker for indoor localization," in *Proceedings of the IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pp. 19–24, Sept. 2011.

[28] J. Sattar, E. Bourque, P. Giguere, and G. Dudek, "Fourier tags: Smoothly degradable fiducial markers for use in human-robot interaction," in *Proceedings of the Canadian Conference on Computer and Robot Vision (CRV)*, pp. 165–174, May 2007.

[29] R. Harle and A. Hopper, "Cluster tagging: Robust fiducial tracking for smart environments," in *Proceedings of the International Conference on Location- and Context-Awareness (LoCA)*, no. 3987 in Lecture Notes in Computer Science, pp. 14–29, Springer Berlin Heidelberg, Jan. 2006.

[30] P. Corke, C. Detweiler, M. Dunbabin, M. Hamilton, D. Rus, and I. Vasilescu, "Experiments with underwater robot localization and tracking," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4556–4561, Apr. 2007.

[31] C. Bonnet, A. Spicer, M. Hoeberechts, D. Owens, D. Abeysirigunawardena, M. Heesemann, K. Juniper, M. Matabos, S. Mihaly, K. Moran, A. Ravindran, and M. Scherwath, *NEPTUNE Canada: an invitation to science.* Victoria, B.C: University of Victoria, 2012.

[32] L. Thomsen, C. Barnes, M. Best, R. Chapman, B. Pirenne, R. Thomson, and J. Vogt, "Ocean circulation promotes methane release from gas hydrate outcrops at the NEPTUNE canada barkley canyon node," *Geophysical Research Letters*, vol. 39, Aug. 2012.

[33] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME - Journal of basic Engineering*, vol. 82, pp. 35–45, 1960.

[34] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 1322–1328, May 1999.

[35] N. Gordon, D. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *IEEE Proceedings F, Radar and Signal Processing*, vol. 140, no. 2, pp. 107–113, 1993.

[36] P. Djuric, M. Vemula, and M. Bugallo, "Target tracking by particle filtering in binary sensor networks," *IEEE Transactions on Signal Processing*, vol. 56, no. 6, pp. 2229–2238, 2008.

[37] N. Shrivastava, R. M. U. Madhow, and S. Suri, "Target tracking with binary proximity sensors: Fundamental limits, minimal descriptions, and algorithms,"

in *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, p. 251–264, Nov. 2006.

[38] M. Sonka, *Image processing, analysis, and machine vision.* Pacific Grove, CA: PWS Pub, 2nd ed., 1999.

[39] M. Bezdek, "On a generalization of the blaschke-lebesgue theorem for disk-polygons," *Contributions to Discrete Mathematics*, vol. 6, no. 1, pp. 77–85, 2011.

[40] M. Fiala, "Designing highly reliable fiducial markers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, pp. 1317–1324, July 2010.

[41] R. Tsai, "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," *IEEE Journal of Robotics and Automation*, vol. 3, no. 4, pp. 323–344, 1987.

[42] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[43] M. Fiala and C. Shu, "Self-identifying patterns for plane-based camera calibration," *Machine Vision and Applications*, vol. 19, pp. 209–216, July 2008.

[44] M. Große, M. Schaffer, B. Harendt, and R. Kowarschik, "Camera calibration using time-coded planar patterns," *Optical Engineering*, vol. 51, pp. 1–9, Aug. 2012.

[45] G. Bradski, "Opencv library," *Dr. Dobb's Journal of Software Tools*, 2000.

[46] T. Filiba, "reedsolo," 2012 (Accessed June 2014). `https://pypi.python.org/pypi/reedsolo`.

[47] M. Thompson, "Fiducial markers for robotic vision, navigation and object recognition," Nov. 2009 (Accessed June 2014). `http://archive.org/details/HBRobotics_Fiducial-Markers`.

[48] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[49] D. P. Bertsekas, *Nonlinear programming.* Belmont, MA: Athena Scientific, 1999.

[50] D. Ashlock, T. Manikas, and K. Ashenayi, “Evolving a diverse collection of robot path planning problems,” in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pp. 1837–1844, July 2006.

[51] J. Canny, “A computational approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[52] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: an open-source robot operating system,” *IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Software*, vol. 3, no. 3.2, 2009.

[53] N. MacMillan, R. Allen, D. Marinakis, and S. Whitesides, “Range-based navigation system for a mobile robot,” in *Proceedings of the Canadian Conference on Computer and Robot Vision (CRV)*, pp. 16–23, May 2011.

[54] Z. Farkas, P. Korondi, D. Illy, and L. Fodor, “Aesthetic marker design for home robot localization,” in *Proceedings of the Annual Conference of IEEE Industrial Electronics Society (IECON)*, pp. 5510–5515, Oct. 2012.

[55] S. Saito, A. Hiyama, T. Tanikawa, and M. Hirose, “Indoor marker-based localization using coded seamless pattern for interior decoration,” in *Proceedings of the IEEE Virtual Reality Conference (VR)*, pp. 67–74, Mar. 2007.

[56] J. W. Tukey, *Exploratory data analysis.* Addison-Wesley series in behavioral science, Reading, MA: Addison-Wesley Pub. Co, 1977.

Glossary

ARTag Augmented Reality Tag. 14, 34

ECS The Engineering and Computer Science Building at the University of Victoria. 63

localization The problem of determining the pose of a robot. 1

overlay grid Visibility grids added on top of each other. 33

pose Cartesian position and orientation (i.e. (x, y, θ) in 2D). 1

RCTag RedCometTag. 34

ROS Robot Operating System. 65

SLAM Simultaneous Localization and Mapping. 4

visibility grid The discretized annulus sector of a marker's visibility region after considering its line-of-sight given obstructions in the environment. 31

Appendix A

Gradient Visibility Model

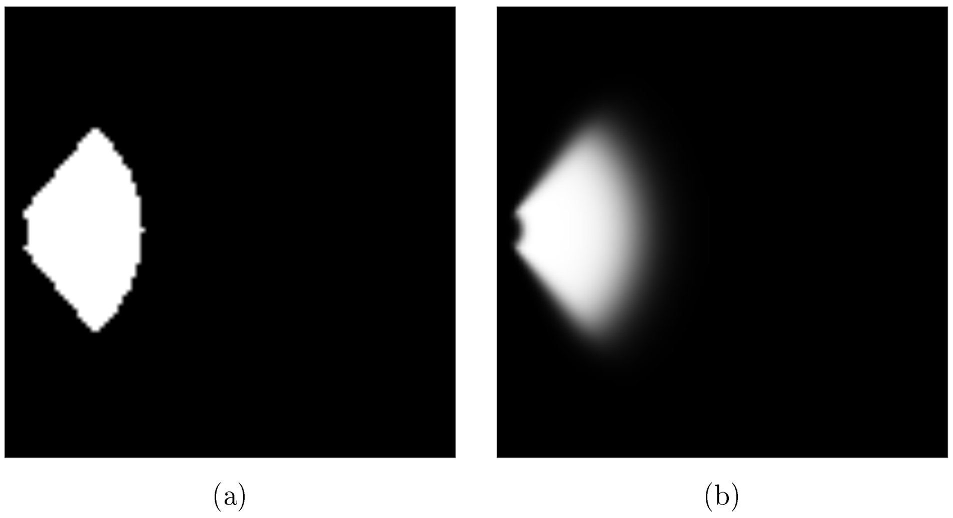
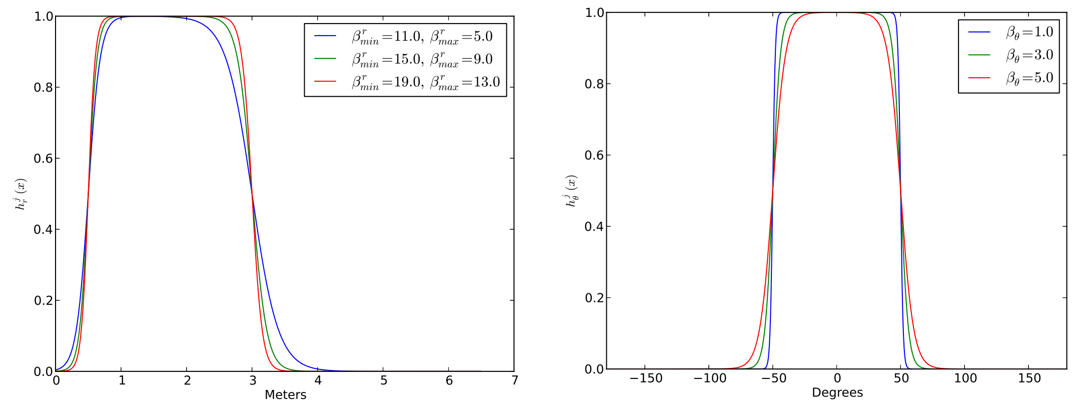


Figure A.1: (a) The uniform visibility marker model. (b) The gradient visibility marker model.

An alternative to the uniform visibility model discussed in Section 3.1.3 is to use a probabilistic visibility model that allows for a more gradual change in the probability density function values as opposed to the hard thresholds of the uniform continuous distribution. This leads us to the following gradient visibility model. First, using the probabilistic model from Akbarzadeh et al. [15], we define a function that is the difference of two sigmoid functions:

$$h(x) = \frac{1}{1 + exp(-\beta_1\,(x\ -\ \alpha_1))} - \frac{1}{1 + exp(-\beta_2\,(x\ -\ \alpha_2))} \tag{A.1}$$

where β_1 and β_2 are constants that define the shape of the curves, and α_1 and α_2



(a) (b)

Figure A.2: (a) Equation A.2 of the gradient visibility model with $r_{min} = 0.5$ and $r_{max} = 3$. (b) Equation A.3 of the gradient visibility model with $\alpha^j_t = 50^o$.

determine the lower and higher midpoints of each curve. Next, this function is then used to define a function for the gradient of the distance from the sample point to the marker:

$$h^j_r(r^i_t) = \frac{1}{1 + exp(-\beta_{r_{min}}(r^i_t - r_{min}))} - \frac{1}{1 + exp(-\beta_{r_{max}}(r^i_t - r_{max}))} \tag{A.2}$$

and a function for the gradient of the angle from the sample point to the marker:

$$h^j_\theta(\omega^i_t) = \frac{1}{1 + exp(-\beta_\theta(\omega^i_t + \alpha^j_t))} - \frac{1}{1 + exp(-\beta_\theta(\omega^i_t - \alpha^j_t))} \tag{A.3}$$

The constants $-\beta_{r_{min}}$, $-\beta_{r_{min}}$ and $-\beta_\theta$ describe the steepness of the dropoff (see Figure A.2).

Finally, to use Equations A.2 and A.3 as definitions for continuous probability distributions, the expressions on the right of A.2 and A.3 must be normalized. For equation A.2 the possible domain of distances is from $[0, \infty)$, so the integration of $h^j_r(r^i_t)$ is from 0 to ∞:

$$\nu^j_r = \int_0^\infty h^j_r(r^i_t)\, \mathrm{d}r^i_t$$

$$\nu^j_r = \frac{1}{\beta_{r_{max}}} \ln|1 + exp(\beta_{r_{max}} r_{max})| - \frac{1}{\beta_{r_{min}}} \ln|1 + exp(\beta_{r_{min}} r_{min})| \tag{A.4}$$

and similarly for $h^j_\theta(\omega^i_t)$:

$$\nu_\theta^j = \int_{-180^o}^{180^o} h_\theta^j(\omega_t^i)\,\mathrm{d}\omega_t^i$$

$$\nu_\theta^j = \frac{1}{\beta_\theta}\left[\ln|1 + exp(-\beta_\theta\omega_t^i + \beta_\theta\alpha_t^j)| - \ln|1 + exp(-\beta_\theta\omega_t^i - \beta_\theta\alpha_t^j)|\right]_{-180^o}^{180^o} \tag{A.5}$$

With the normalizing constants ν_r^j and ν_θ^j we can then define our gradient visibility distribution as:

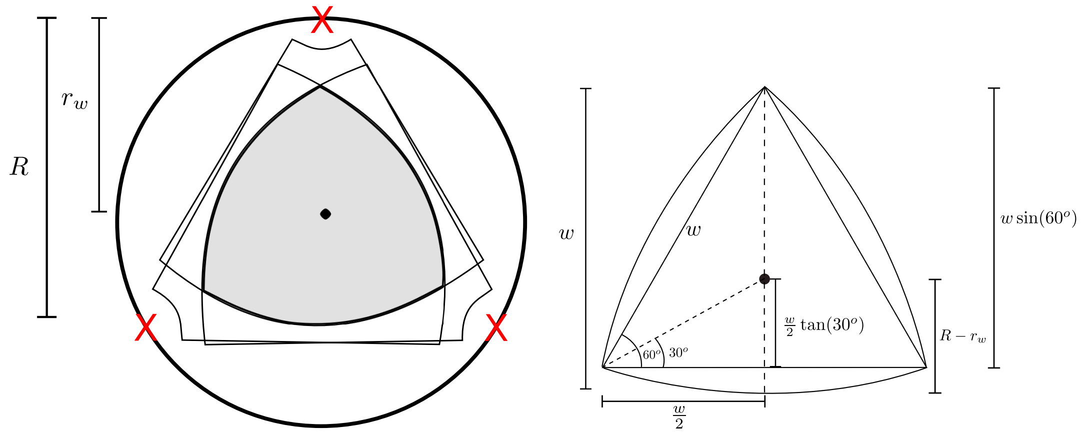
$$p(v_t^{i,j} = 1 \,|\, s_t^i) = \frac{h_r^j(r_t^i)}{\nu_r^j} \times \frac{h_\theta^j(\omega_t^i)}{\nu_\theta^j} \tag{A.6}$$

$$p(v_t^{i,j} = 0 \,|\, s_t^i) = 1 - \left(\frac{h_r^j(r_t^i)}{\nu_r^j} \times \frac{h_\theta^j(\omega_t^i)}{\nu_\theta^j}\right) \tag{A.7}$$

The results for sample points using the gradient visibility distribution for equation A.6 can be seen in Figure A.1b.

Appendix B

Reuleaux Triangle



(a) (b)

Figure B.1: a) The formation of the Reuleaux triangle by 3 uniformly placed markers (marked by ‘x’) around a waypoint at the center. b) The properties of the Reuleaux triangle given we only know $R - r_w$.

In this appendix, we elaborate on the details of the Reuleaux triangle discussed in Chapter 3.2.1. Based on the uniform placement of three markers r_w distance around the waypoint center, we know the distance from the waypoint center to the marker edge is $R - r_w$ (see Figure B.1a). Because the markers are uniformly placed, the overlap of all their annulus sector visibility regions form a Reuleaux triangle. By definition, a Reuleaux triangle has a constant width w and an area of $\frac{1}{2}(\pi - \sqrt{3})w^2$ [39]. Within a Reuleaux triangle is an equilateral triangle, where each side is length w and the angles at each corner are 60^o. From this we can observe certain properties as seen in Figure B.1b.

Although we do not know what w is and want to find it, it can be seen that:

$$w = w\ \sin(60^o) + ((R - r_w) - \frac{w}{2}\tan(30^o))$$

After simplifying (with $\sin(60^o) = \frac{\sqrt{3}}{2}$ and $\tan(30^o) = \sqrt{3}$):

$$w = \frac{R - r_w}{1 - \sqrt{3}}$$

Substituting w into the original area formula arising from the definition of a Reuleaux triangle, the area of this Reuleaux triangle is $\frac{1}{2}(\pi - \sqrt{3})((1 - \frac{1}{\sqrt{3}})(R - r_w))^2$. The maximum deviation d within this triangle is the distance from the waypoint to a point on the boundary of the triangle. This maximum deviation distance is $d = w - (R - r_w)$. By again substituting for w, the maximum deviation distance has an upper bound of $d = (\frac{1}{\sqrt{3}-1})(R - r_w)$. If the angle 2α of the annulus sectors is smaller than that needed to fill the Reuleaux triangle, given a particular value for r_w, then the maximum distance from the waypoint to this other shape will be less than $(\frac{1}{\sqrt{3}-1})(R - r_w)$. This is why d is given as an upper bound.

Appendix C

Box-and-whisker Plot

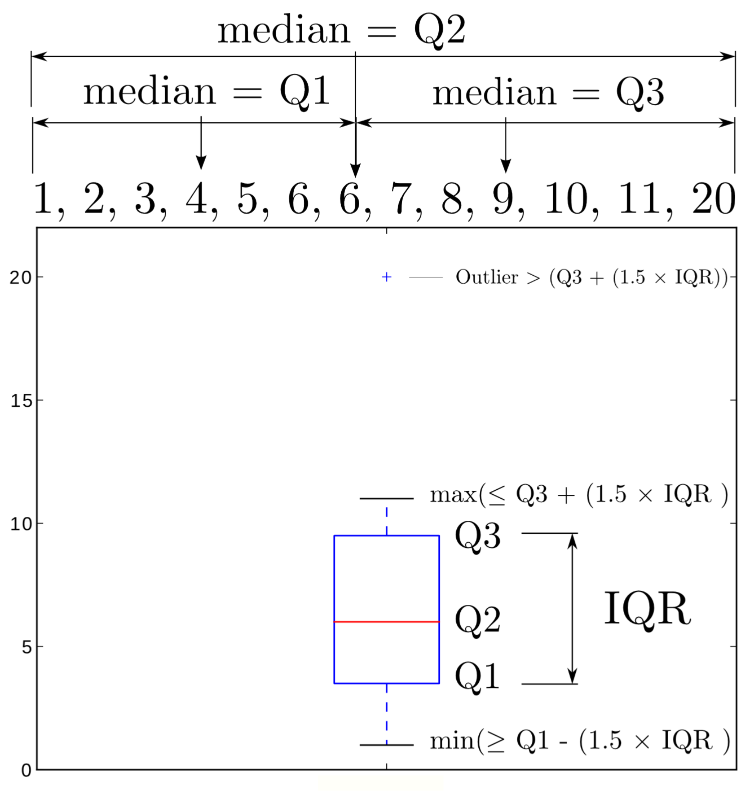


Figure C.1: An illustration of the box-and-whisker plot with some example input data.

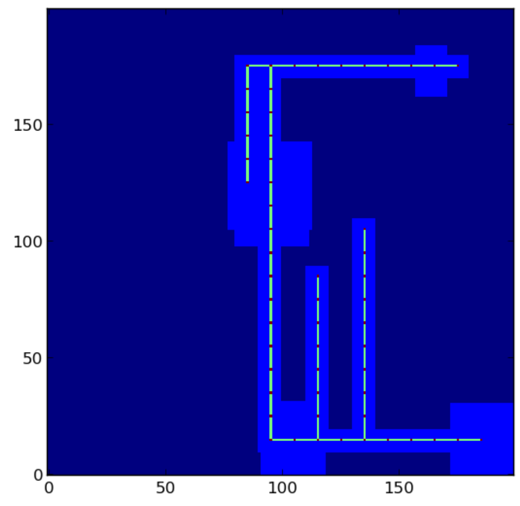
In this appendix we describe the box-and-whisker plot used in Chapter 5.4. The box-and-whisker plot is a type of plot created by John Tukey [56] that can help illustrate the distribution of data points – in particular, their skew and variance (Figure C.1). The box component of the box-and-whisker plot is defined by three *quartiles*. The second quartile (Q2) is the median of all the data points. The first quartile (Q1) is the median of all data points lower than and including the second quartile, while the third quartile (Q3) is the median of all data points higher than and including the second quartile. The size of the box InterQuartile Range (IQR) is defined as $Q3 - Q1$.

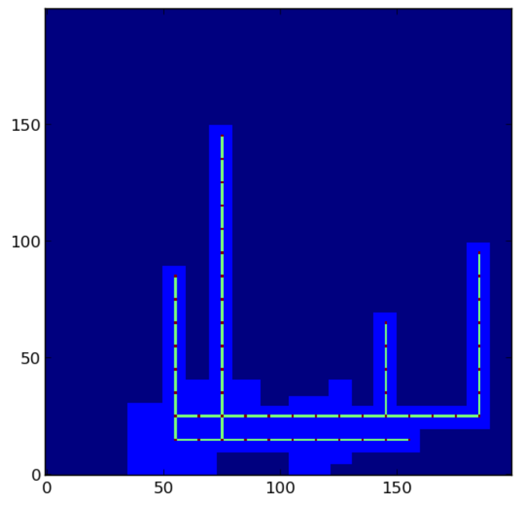
In this thesis, the top whisker of the box-and-whisker plot is defined as the maximum data point that is less than or equal to $Q3 + (1.5 \times IQR)$. Similarly, the bottom whisker is defined as the minimum data point that is greater than or equal to $Q1 - (1.5 \times IQR)$. Any data points above the top whisker or below the bottom whisker are considered outliers and plotted as points.

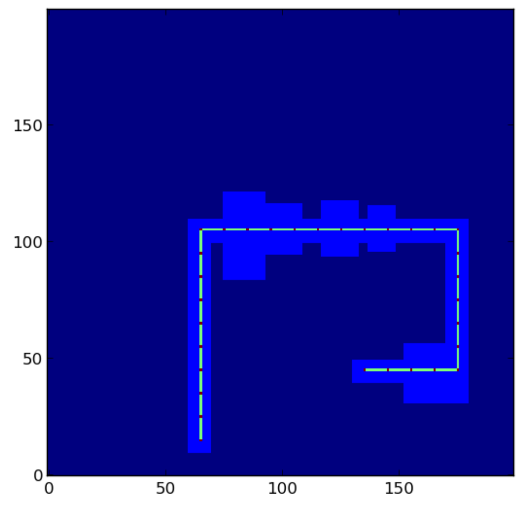
Appendix D

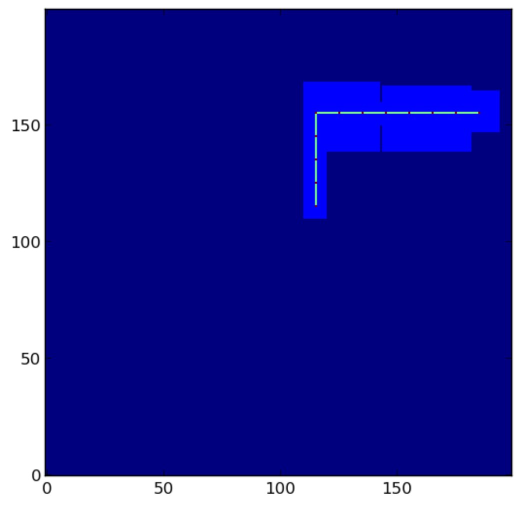
Random Hallway Maps

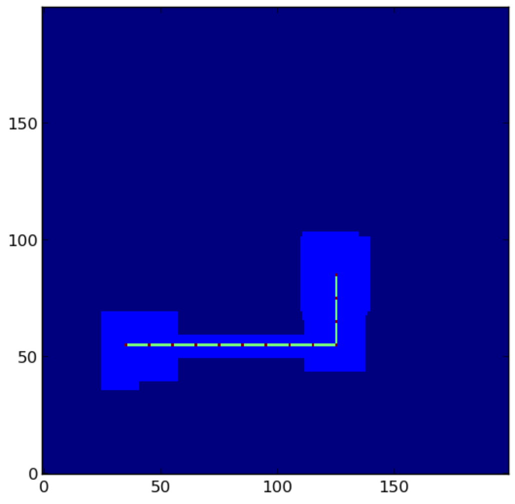
The following figures are the random hallway maps (occupancy grids and waypoints) generated using our method described in Chapter 4.2 for simulation in Chapter 4.3. The seed for each random hallway map is the map ID number (i.e. the seed for 'Map 0' is 0).


Map 0

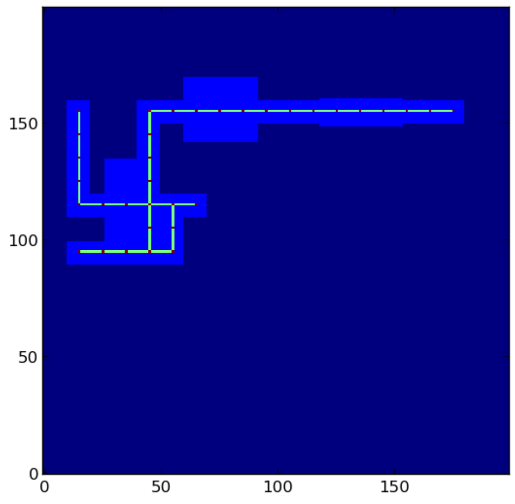

Map 1


Map 2

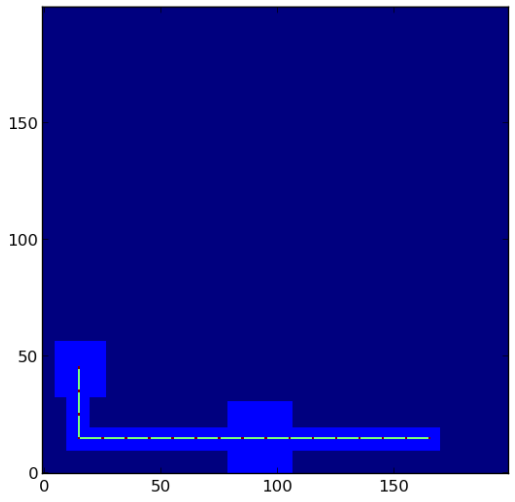

Map 3



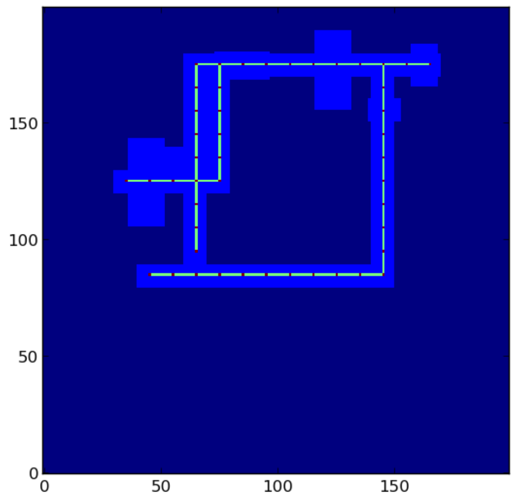
Map 4



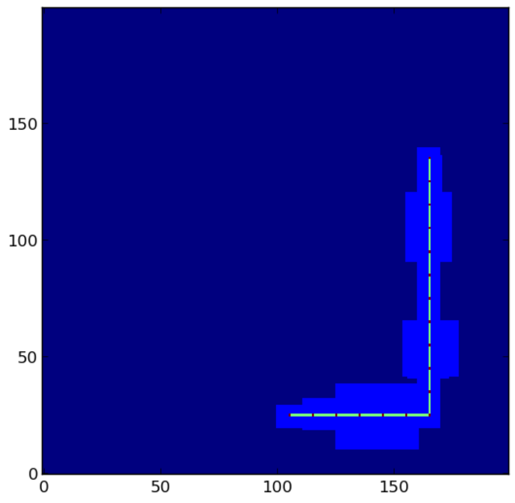
Map 5



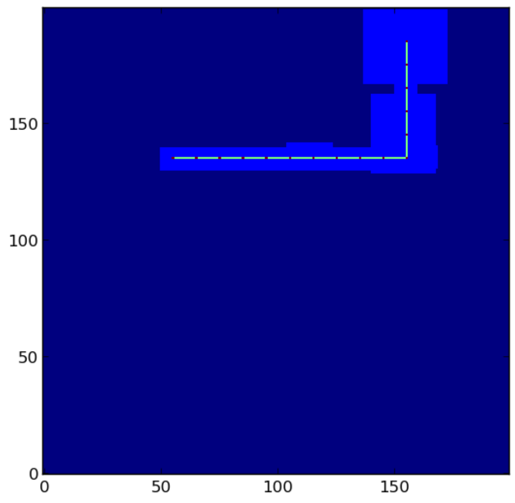
Map 6



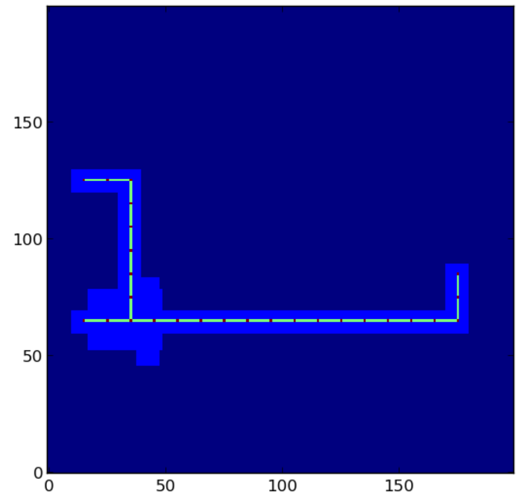
Map 7



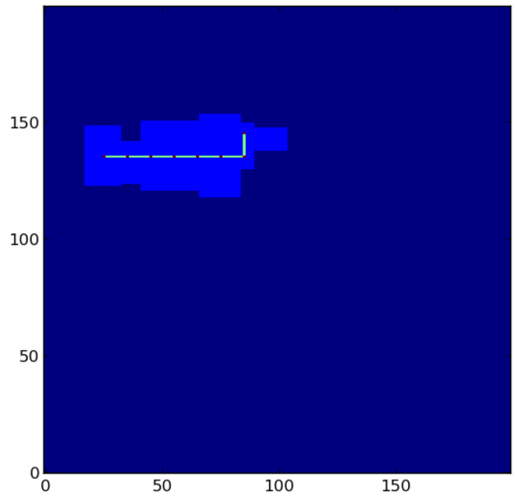
Map 8



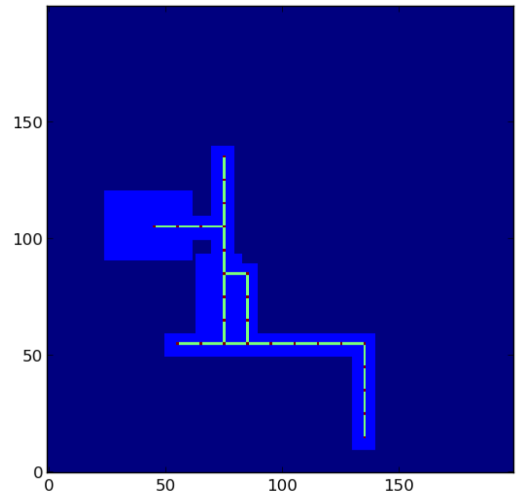
Map 9



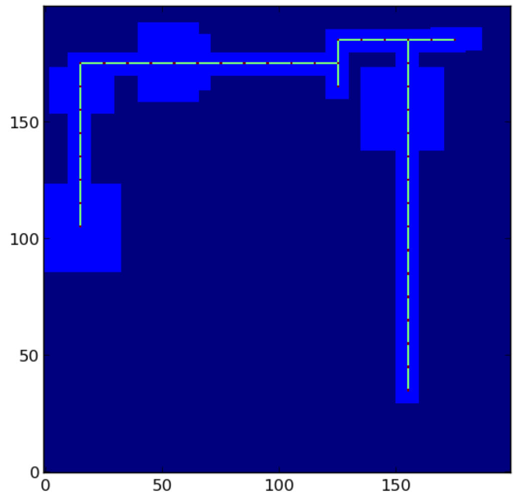
Map 10



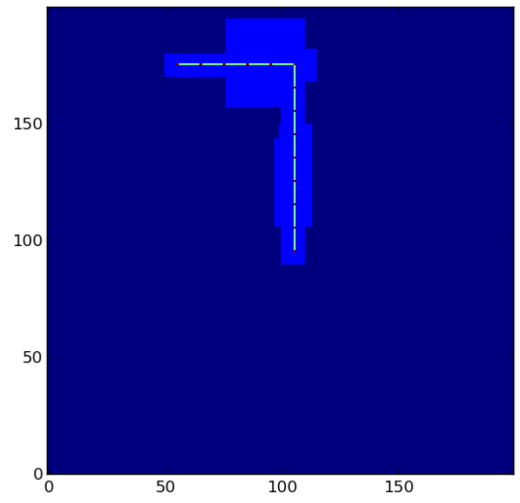
Map 11



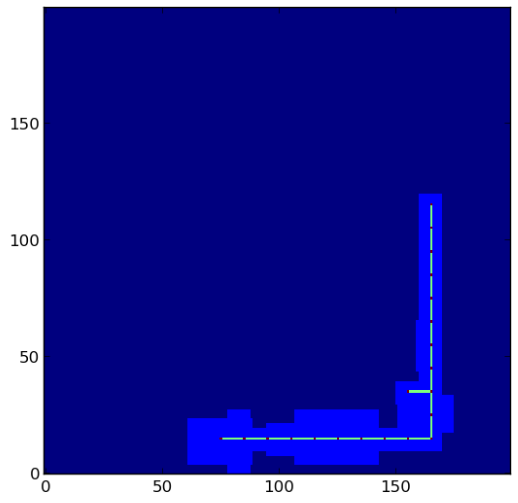
Map 12



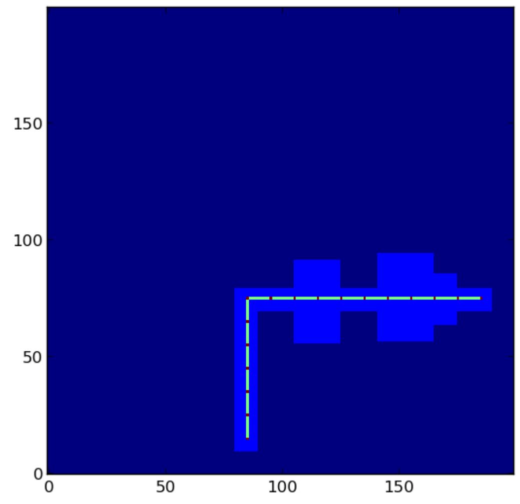
Map 13



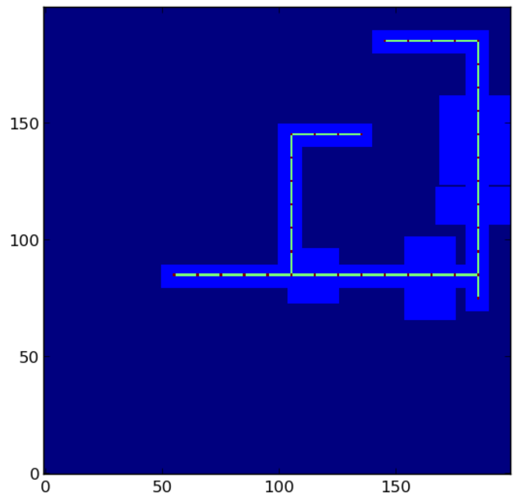
Map 14



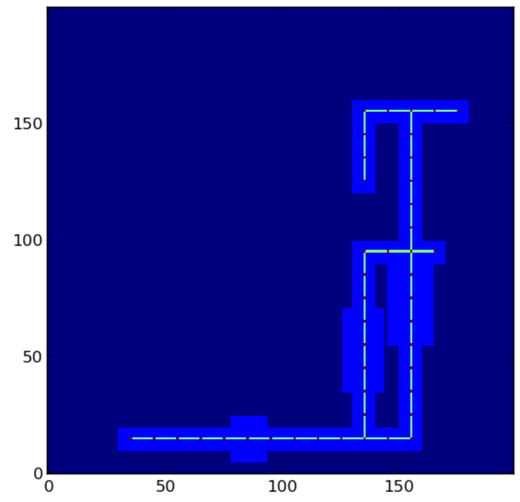
Map 15



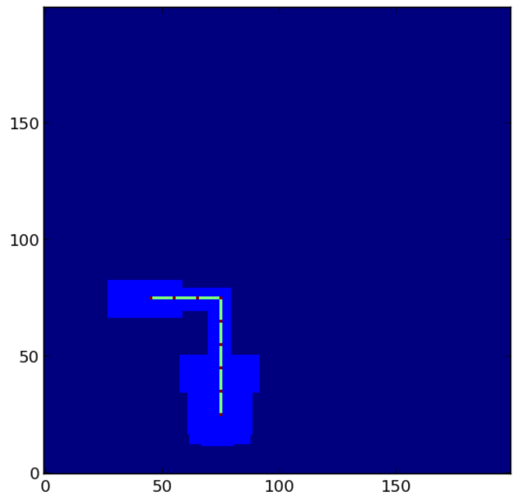
Map 16



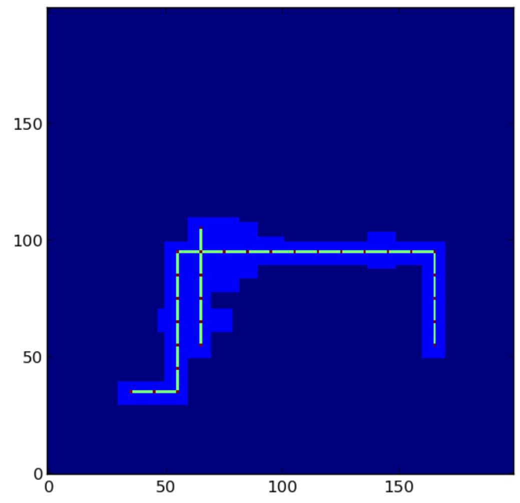
Map 17



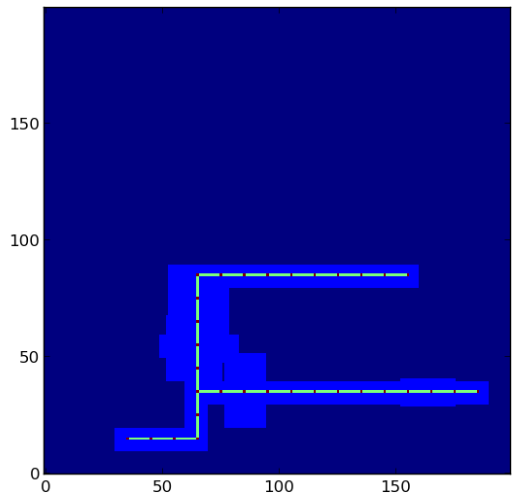
Map 18



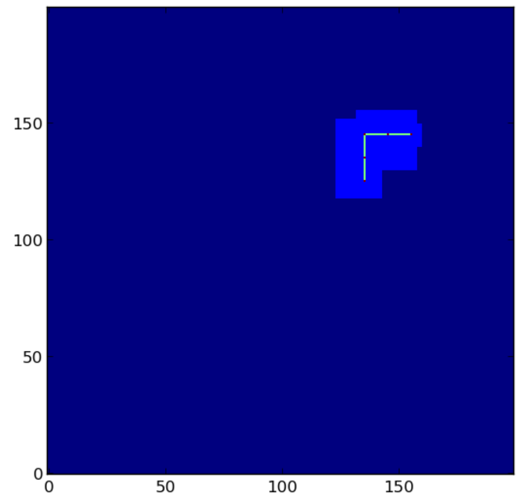
Map 19



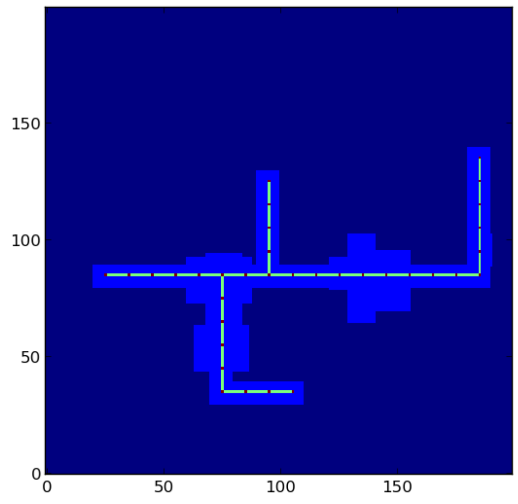
Map 20



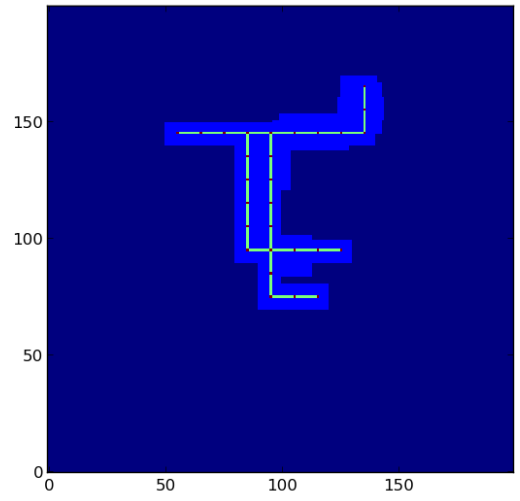
Map 21



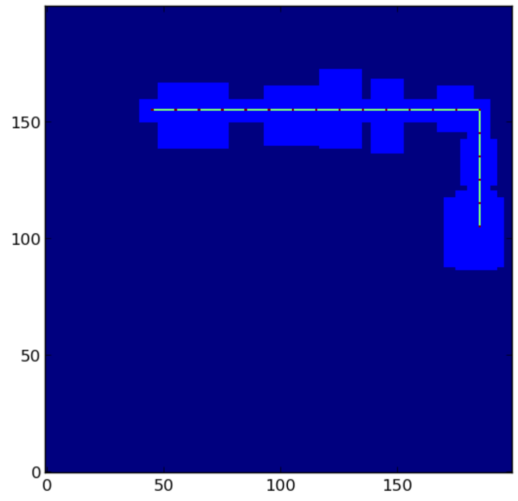
Map 22



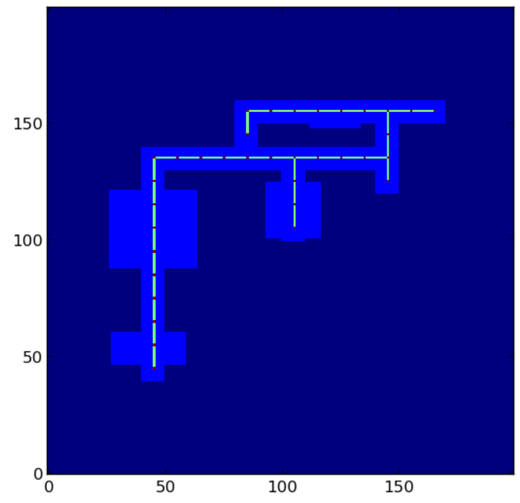
Map 23



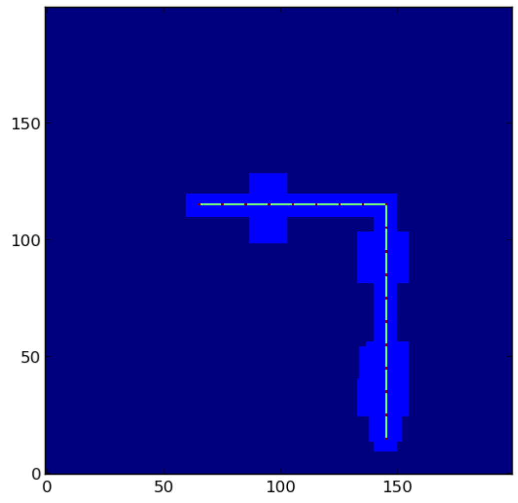
Map 24



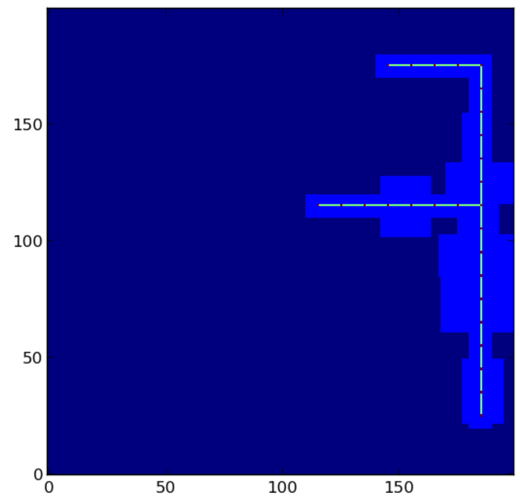
Map 25



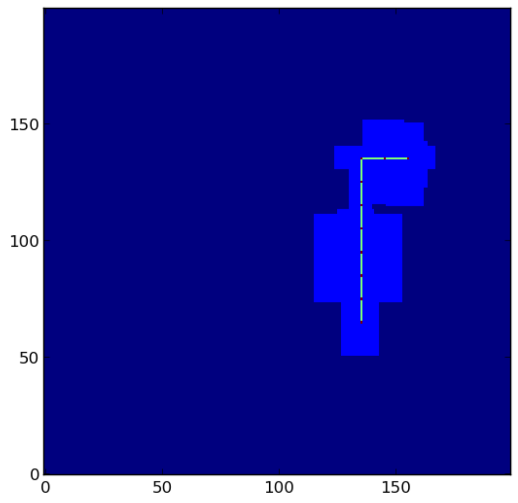
Map 26



Map 27



Map 28



Map 29

Appendix E

Experiment Runs Data

This section includes the data of all the physical experimental runs of the robot in the hallway as discussed in Chapter 5.4. To recap, the total length of the run was approximately 45 meters, with the width of the hallway being approximately 1.6 ±0.2 meters. The distance between waypoints varied from approximately 2.3 to 4 meters. Most runs that failed were the result of 10 bumps, but some also failed because the robot had an error distance from the expected waypoint exceeding 4 meters. This was usually the result of the robot moving in the wrong direction. These failures are indicated by 'Err'.

Waypoint	Run 1	Run 2	Run 3	Run 4	Run 5
1	0.0	0.0	0.0	0.0	0.0
2	0.27	0.23	0.34	0.17	0.28
3	0.37	0.26	0.54	0.32	0.46
4	0.31	0.1	0.63	0.22	0.47
5	0.35	0.16	0.68	0.05	0.81
6	0.4	0.84	1.44	0.72	0.72
7	0.58	-	-	-	0.87
8	0.66	-	-	-	1.32
9	1.0	-	-	-	2.43
10	0.93	-	-	-	-
11	1.54	-	-	-	-
12	1.83	-	-	-	-
13	2.28	-	-	-	-
14	2.72	-	-	-	-

Table E.1: No Markers - Measured error from waypoints (meters).

Waypoint	Run 1	Run 2	Run 3	Run 4	Run 5
1	0.0	0.0	0.0	0.0	0.0
2	0.1	0.1	0.12	0.09	0.05
3	0.23	0.39	0.05	0.05	0.22
4	0.6	0.53	0.1	0.05	0.17
5	0.54	-	0.18	0.08	0.21
6	-	-	0.68	0.07	0.34
7	-	-	1.37	0.09	0.41
8	-	-	1.38	0.43	0.42
9	-	-	2.32	0.32	0.98
10	-	-	-	0.32	-
11	-	-	-	0.54	-
12	-	-	-	0.8	-
13	-	-	-	1.14	-
14	-	-	-	1.39	-

Table E.2: Best Score Placement - Measured error from waypoints (meters).

Waypoint	Run 1	Run 2	Run 3	Run 4	Run 5
1	0.0	0.0	0.0	0.0	0.0
2	0.1	0.42	0.22	0.41	0.17
3	0.11	0.35	0.38	0.33	0.12
4	0.05	0.54	0.38	0.41	0.3
5	0.22	1.0	0.05	0.36	0.16
6	0.37	-	0.42	0.32	0.23
7	0.43	-	0.75	0.21	0.59
8	0.08	-	1.57	0.32	0.79
9	0.94	-	1.79	0.69	0.87
10	0.73	-	-	0.45	0.71
11	0.95	-	-	0.68	1.31
12	1.0	-	-	0.89	1.6
13	1.17	-	-	1.1	1.98
14	1.4	-	-	1.38	2.37

Table E.3: Uniform Placement - Measured error from waypoints (meters).

Waypoint	Run 1	Run 2	Run 3	Run 4	Run 5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	1	0	0
4	0	0	0	0	1
5	0	0	3	0	2
6	0	5	5	1	0
7	0	10	10	10	0
8	0	-	-	-	5
9	0	-	-	-	4
10	0	-	-	-	Err
11	4	-	-	-	-
12	0	-	-	-	-
13	1	-	-	-	-
14	2	-	-	-	-

Table E.4: No Markers - number of Bumps between waypoints.

Waypoint	Run 1	Run 2	Run 3	Run 4	Run 5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	1	0	0	0
5	9	10	0	0	0
6	10	-	2	0	0
7	-	-	3	0	0
8	-	-	0	0	2
9	-	-	3	0	3
10	-	-	10	0	Err
11	-	-	-	0	-
12	-	-	-	0	-
13	-	-	-	0	-
14	-	-	-	0	-

Table E.5: Best Score Placement - number of Bumps between waypoints.

Waypoint	Run 1	Run 2	Run 3	Run 4	Run 5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	2	0	0	0
5	0	2	0	0	1
6	0	10	0	0	0
7	0	-	1	0	0
8	2	-	3	0	2
9	2	-	0	0	0
10	0	-	10	0	0
11	0	-	-	0	4
12	0	-	-	0	0
13	0	-	-	0	1
14	0	-	-	1	2

Table E.6: Uniform Placement - number of Bumps between waypoints.