Building Local Maps for Robust Visual Navigation

by

Jeremy S. Gerstle

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

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2001

© Jeremy S. Gerstle, MMI. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.
Building Local Maps for Robust Visual Navigation

by

Jeremy S. Gerstle

Submitted to the Department of Electrical Engineering and Computer Science on February 8, 2001, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis describes a robust low-level visual navigation system used to drive Erik, an RWI B21r, robot around cluttered labrooms as well as spacious hallway corridors and playrooms of the AI lab. The system achieves robust navigation of office-like environments in the presence of partially unmodelled noise by combining simple and efficient visual processing, fast mapping and a near-optimal anytime planning algorithm. It is intended to demonstrate that sophisticated near-optimal navigation strategies can be found using fast integration of highly dense approximate visual and odometric data into small, temporary room-sized maps. The navigation algorithm follows from recent research on embedded learning agents and the theory of stochastic optimal control, and can be used as a framework for integrating multiple sources of information into coherent room-level navigation policies. The system is also fairly robust to modest but constrained changes in a number of its internal parameters. In this thesis, I will give an overview of the system’s construction, describing the theory behind the inner workings of each subsystem, and analyzing the validity of simplifying assumptions in the context of the robot’s operating environment. I will then conclude with the results of several test runs illustrating overall system functionality, and discuss possible improvements that can be made in the future.

Thesis Supervisor: Leslie P. Kaelbling
Title: Professor of Computer Science and Electrical Engineering
Acknowledgments

A number of people have lent me their expertise, wisdom, and support throughout my days at MIT and especially while working on this thesis. First, I would like to thank the entire Learning in Intelligent Systems lab, especially my advisor Professor Leslie Kaelbling for her consistent and helpful insight, extraordinary patience, and for being a true mentor; showing me how to be methodical, critical, unceasing in my pursuit for results, and unwilling to throw in the towel and claim defeat. A special thanks to Bill Smart who taught me the difference between good and bad C/C++ programming technique, for writing SeRF without which I would never have gotten off the ground, and for providing nearly round the clock technical support. Also, a special thanks to Terran Lane for spending significant time helping me understand and improve my algorithm. I would also like to thank Selim Temizer for his helpful insight and hours of code debugging help, along with Kurt Steinkraus for lending me his tcp/ip networking classes without which I might have spent unnecessary time struggling to write my own. Outside of LIS I would also like to thank PhD Sandy Wells for his help with understanding the math of camera calibration, and Polina Golland along those same lines for giving me the idea for camera tilt parameterization and hours of help developing the math to support it. While there are a number of other people in the AI lab that have helped me with this M.Eng thesis, I would like to give a special acknowledgement and thank you to PhD student Michael Wessler who first introduced me to world of reinforcement learning, taught me how to analyze and understand its mathematics, and without whom I might not have had an M.Eng thesis. Additionally, I would like to thank my friend Dan Zelazo for his support and constructive critiques throughout the research process. Romy Nakagawa and Rania Khalaf, and the rest of my friend at MIT for making me take breaks when I thought I was going to lose my head. And finally, an all important thank you to my family. I am grateful for your unceasing support and love. Especially to my brother Ari for taking on the undesirable task of helping me proof read and correct this paper in its final hours.
Contents

1 Introduction 9
   1.1 Architecture: System Overview ................................... 12

2 Obstacle Detector 15
   2.1 Theoretical Overview .............................................. 15
      2.1.1 The Ground Plane Constraint .................................. 16
      2.1.2 The Background Texture Constraint .......................... 16
   2.2 Design and Construction ......................................... 19
      2.2.1 Edge Detection .................................................. 19
      2.2.2 Using the Obstacle Detector for Reactive Control ............ 20
   2.3 Subsystem Analysis ............................................... 22

3 Depth Recovery 24
   3.1 Perspective Projection Camera Model ............................ 24
      3.1.1 Perspective Transform: Affine Equations .................. 25
      3.1.2 Projective Geometry ......................................... 27
   3.2 Camera Calibration .............................................. 28
   3.3 Recovering Floor Coordinates ................................... 30

4 Occupancy Grid Map 35
   4.1 Constructing Occupancy Grids ................................... 35
      4.1.1 Background on Use and Application to Vision .............. 36
      4.1.2 Integrating Occupancy Evidence ............................. 37
5 Navigation: The Planning Module

5.1 The Agent-Environment Feedback Model .................................. 43
  5.1.1 Overview of Markov Decision Processes ............................... 45
  5.1.2 Sidebar: On the Applicability of the MDP Framework ............ 47
  5.1.3 Solving finite MDPs .................................................. 52

5.2 Value Iteration and Planning Algorithm ................................. 54
  5.2.1 Stopping Criteria: Planning Time versus Control Time .......... 54
  5.2.2 Incorporating Proprioceptive Knowledge ............................ 57
  5.2.3 Optimal Behavior: Exploitation versus Exploration ............. 59

6 Actuator Control  ..................................................................... 61
  6.1 Base Actuation ................................................................. 61
  6.2 Pan-Tilt Actuation ............................................................. 62

7 Results .................................................................................. 64
  7.1 System Analysis ............................................................... 64
    7.1.1 The Problem with Camera Tilt ...................................... 65
    7.1.2 Map Size Parameters .................................................. 67
    7.1.3 Probability and Certainty Parameters .............................. 69
  7.2 Conclusion and Future Work ................................................. 70

A Calibration Code ................................................................. 72
List of Figures

1-1 The Navigation Task: Local Signposts give global information . . . . . . 9
1-2 Block Diagram of System Interactions. . . . . . . . . . . . . . . . . . . . 12

2-1 The Ground Plane Constraint . . . . . . . . . . . . . . . . . . . . . . . . 17
2-2 Processing an Image into a Radial Depth Map. . . . . . . . . . . . . . 18
2-3 Comparison of Edge Detectors: detector used by Erik picks up the
   legs of the table but also more shadows. . . . . . . . . . . . . . . . . . . . 20
2-4 Partitions of the Radial Depth Map: l(t), r(t), c(t) marked. . . . . . . 21

3-1 Perspective Projection Camera Model . . . . . . . . . . . . . . . . . . . 25
3-2 Calibration Pattern: Black tape reference points, and red circles show
   their calculated reprojection. . . . . . . . . . . . . . . . . . . . . . . . . . . 29
3-3 Recovery of Floor Coordinates: World coordinate system rigidly at-
   tached to the floor. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
3-4 Accuracy of Floor Coordinate Recovery . . . . . . . . . . . . . . . . . . 33

4-1 RDM gives grid cell occupancy information. . . . . . . . . . . . . . . . 38
4-2 Evolution of an Occupancy Grid . . . . . . . . . . . . . . . . . . . . . . . 42

5-1 The Agent-Environment Feedback Model . . . . . . . . . . . . . . . . . 43
5-2 Planning Algorithm Employed on Erik . . . . . . . . . . . . . . . . . . . 48
5-3 Evolving belief state: cooler colors indicate lower map occupancy, and
   lower total expected cost to goal in the corresponding plan . . . . . . . . 50
5-4 The value function and policy mature with time and exploration. The maps on the left are occupancy grids with stable wall obstacles clearly defining the boundary of the explored space. The associated value function which it generates is to its right. 58

5-5 The Modified Transition Function: Erik's diameter is three grid cells in length. 59

7-1 Obstacle Detector fails leading to imminent collision. 66

7-2 Different occupancy grids: images scaled to show relevant detail. 68
List of Tables

3.1 Calibration Data and Estimation Statistics ............... 31

5.1 The value-iteration algorithm for finite state-space MDPs. ........ 54
Chapter 1

Introduction

Suppose you blind-folded a person, dragged him or her to room 916 of the AI lab, took off their blindfold and told them to go to the conference room at the end of the hallway (presuming they’ve never even been in the AI lab before). Assuming they understand what a hallway is, and can read the label on the door of the conference room, the task is trivial: leave room 916, walk straight down the hallway, and near the end look for the conference room door. However, suppose the hallway is blocked by hallway trash that regularly accumulates on the 9th floor of the AI lab. The
person would probably still be able to find his way by finding a door opening to the hallway on the other side of the building which also connects to the conference room. Finding the door and piecing together how rooms connect is the process of forming a topological map, a graph of landmarks (nodes) connected by rooms and corridors (edges). Navigating within rooms and corridors to discover passable ways of getting to known landmarks requires avoiding obstacles and methods for getting to the each “signpost”. A mobile robot that can navigate rooms and corridors well without getting stuck in local minima will stand a better chance of discovering new landmarks and eventually knowledge of the underlying graph structure. The construction of a framework for learning robust visual navigation policies for robots in an office building environment therefore begins with a low-level navigator that can recognize obstacles and avoid getting stuck in local minima. This is the impetus for the system built in this thesis.

The inspiration for building the navigation system described in this thesis came from Ian Horswill's work on Polly, a vision based mobile robot that could give tours of the 7th floor of the AI lab. Polly used simple visual machinery and purely reactive control processes in a subsumption like architecture[2] to achieve its navigation tasks. While its simple machinery required very little computational power, the overall efficiency of the controller was due to the specialization of each of its subsystems to its task and environment. The ability of each subsystem to perform its given task was therefore highly dependent on the degree to which idealized assumptions made by the specialized subsystem matched a given environment. Polly could easily recover from errors in its high level navigation modules due to their interdependent interaction with the environment. If the robot got confused or lost, it could wander around relying on its obstacle avoidance routine to keep it from bumping into things until its place recognition system triggered on something that would re-localize it. The robot's recovery was therefore dependent on its embedding in the environment, it had no method for replanning: changing its path intelligently once its current plan failed.
By keeping track of where obstacles are in the world a robot can account for errors in perception and can recover quickly from failure in its high-level navigation subsystems by replanning. This is the formula for the navigation system described in this thesis. Gathering local information into a global occupancy grid data structure, Erik our RWI B21r robot, captures a cumulative history of what it has seen, and continuously replans given its current state of information.
1.1 Architecture: System Overview

The navigation system consists of five subsystems:

- **Obstacle Detector** – produces immediate information describing where obstacles are in the robot's visual field relative to its base.

- **Depth Recovery** – converts local image pixel data into global map coordinates.

- **Occupancy Grid Map** – integrates obstacle detector data over time to construct a map of the probability of occupancy at discretized locations in the global or world coordinate frame.

- **Path Planner** – combines information from the occupancy grid and robot odometry instructing the robot which immediate control actions to take.

- **Actuator Control** – applies the appropriate control to execute the path planner's instructions, and checks overall system sanity, possibly overriding the planner's instructions when needed.
Each of these subsystems is built to work in a modular fashion such that any one subsystem can be swapped out and replaced by another bearing the same input-output relationship and satisfying a basic set of constraints. Figure 1.2 illustrates the overall system processing.

The *Obstacle Detector* used is borrowed from Ian Horswill's work on low-level visual collision avoidance for mobile robot navigation. The detector takes as input a video stream from the robot and produces a radial depth map, a description of relative distance along the floor from the robot’s center to obstacles in its field of view that project up significantly from the ground plane. Details of the detector’s functionality are described in chapter 2. It is worth mentioning here that this particular choice of detector has led to certain overall system design considerations which might be dispensed with given another choice of obstacle detector. The detector makes a reasonable assumption about the robot’s operating environment, namely that the floor in an office environment is is textureless at a large scale. This assumption imposes certain constraints which may impact the reliability of the occupancy grid and path planning subsystems if violated. Other detectors, discussed at the end of this thesis, may allow for a more flexible system.

The *Occupancy Grid Map* module is adapted from the technique pioneered by Moravec and Elfes [8], and has been used by numerous mobile robot researchers including Kaelbling, et al. [9]. The occupancy grid map receives the result of passing the radial depth map found by the *Obstacle Detector* through the *Depth Recovery* module, and combines it with odometric information taken from the robots shaft encoders to produce a global description of where the robot perceives these obstacles. Using a well constructed update rule, it then integrates this new information given a prior belief of occupancy for each grid cell. The resulting set of occupancy values between 0 and 1 for each grid location, are likened to probabilities of occupancy and constitute a belief state for the robot agent acting in a partially observable Markov environment[7]. By making a *static world assumption*, i.e., that the robot is the only one moving, the problem of acting optimally in such environments becomes deterministic at each time step, and well known dynamic programming solutions can
be employed. This issue is discussed at further length in chapter 4.

The Path Planner module computes a near-optimal navigation policy conditioned on the robot's belief state at each time step. The policy allows the robot to determine the best action given its current location in the occupancy grid in constant time. Chapter 5 takes an in-depth look at the governing theory behind the continuous replanning approach for navigation and details the conditions under which this approach is optimal.

Finally, the last stage is the Actuator Control subsystem which applies the necessary translation and rotation velocities to the base to execute the action computed from the Path Planner's policy given the robot's current grid cell state. This system also controls the camera head's pan and tilt which can greatly influence the robot's perception of the environment and can aid in reducing overall system error.

Together these subsystems form a single competent behavior for our robot: to navigate carefully within rooms and hallways without running into obstacles and without getting stuck in dead ends. It is a basic building block with which large-scale landmark based topological mapping can be achieved. Although system does a good job of planning given the information it has seen, and under most circumstances only has to make minor adjustments to the current policy given new information, there are a number of problems with the current implementation. Most of the failures of this system stem from the robot's inability to servo control well enough to follow plans accurately for sufficient lengths of time. This can sometimes lead to large errors in perception from which the system cannot recover. To the degree to which this system therefore exhibits competent behavior, although theoretically grounded, requires significantly more debugging and testing to prove its effectiveness.
Chapter 2

Obstacle Detector

2.1 Theoretical Overview

Horswill broke the problem of collision avoidance down into the two difficult subproblems of obstacle recognition and depth recovery. In his investigations Horswill was able to derive a set of constraints and idealizations of the robot agent's environment which, if satisfied, would allow the agent to employ simple mechanisms for the solution of each subproblem. His visual collision-avoidance routine which was implemented on both Polly and Frankie, two (now retired) vision-based robots, as well on robots by researchers elsewhere, empirically validated these assumptions. More importantly, the ubiquitous employment of his algorithm on robots outside this lab is testimony that simple machinery well tailored to the environment or a specific task can reduce the complexity of the agent's computation.

Every visual navigation algorithm makes certain assumptions about the appearance of objects and obstacles in the environment. Different algorithms can be seen as embodying different assumptions about the appearance of obstacles in the environment [5]. The Obstacle Detector exploits the fact that it is often easier to recognize the background than to recognize an obstacle. Following closely Horswill's arguments, we note several characteristics which seem generally true for a mobile robot operating in office like environments:
• Obstacles generally project up significantly from the floor, i.e., walls, file cabinets, chairs, etc.

• The carpeted floor surface in most office environments is generally textureless at a moderate scale.

• The environment is uniformly illuminated by overhead fluorescents so no shadows are cast.

These assumptions about the operating environment generate two constraints, the *Ground Plane Constraint* (GPC), and the *Background Texture Constraint* (BTC), which simplify the *Obstacle Detector* by allowing us to tailor it to the given environment.

### 2.1.1 The Ground Plane Constraint

The GPC embodies the assumption that obstacles that project up from the ground plane form a boundary with the floor whose distance is directly proportional to the height of the corresponding boundary edge in the robot’s camera image. A 1-d perspective projection camera model tells us this relationship exactly: distance to the obstacle base $D$ is related via similar triangles to pixel height $h$ of the base edge in the robot’s camera image. Figure 2.1 illustrates this model. It is assumed that overall distance from the robot base to the obstacle can be recovered fully from $D$, and the distance to the intersection of the optical axis with the floor calculated from simple geometry. Consequently, any operator which separates ground plane pixels in the image can be used to detect obstacles and reconcile relative depth information.

### 2.1.2 The Background Texture Constraint

The BTC embodies the assumption that the floor surface in the office environment is a nearly textureless one. The carpeted surface in many office environments consists of a uniform color with some gritty texture: a DC component plus high frequency noise. This noise can usually be removed while preserving the DC component by
applying a smoothing operator; low-pass filtering the image with a cutoff at $\omega_{image}$, the lowest spatial frequency of the carpet’s high frequency component as it appears in the image. If $\omega$ is the lowest spatial frequency of the carpet’s high frequency noise spectrum, it will appear in the image with frequency

$$\omega_{image} = \frac{\omega d}{f}$$  \hspace{1cm} (2.1)

for image plane parallel to the ground and looking down from a distance $d$ through a lens with focal length $f$. Changing the camera’s orientation effectively changes $d$ and therefore can only increase this frequency, so the low-pass filter will be invariant with respect to changes in the robot camera’s view orientation. Since an edge detector effectively functions as a band-pass filter, by tuning it so that its high cutoff is less than $\omega_{image}$, we preserve only the boundaries between areas of different DC offsets. Let $u, v$ be row and column indices of an image pixel in a bottom-left to top-right
coordinate system. By applying the bottom projection operator

\[ b(J) = \min_u : J(u, v) = 1 \]  \hspace{1cm} (2.2)

to \( E(I(t)) \) the edge detected image found by applying the edge detection operator \( E(\cdot) \) to \( I(t) \), the robot’s camera image at time \( t \), we obtain the radial depth map \( RDM_t(u) \)

\[ RDM_t(u) = b(E(I(t))), \]  \hspace{1cm} (2.3)

a mapping from direction to the amount of freespace in that direction. Figure 2.2 illustrates the transformation from image to radial depth map; the bottom frame shows the radial depth map displayed as a polar graph with image pixel height measuring radius in the robot’s radial field of view. Noise in the RDM resulting from the edge detection process, although difficult to see in the color shaded image, is typically ignored in subsequent data processing.

Figure 2-2: Processing an Image into a Radial Depth Map.
### 2.2 Design and Construction

#### 2.2.1 Edge Detection

The choice of $E(\cdot)$ will, in general, trade off robustness to environment variability and speed of computation. The detection algorithm used on Erik requires significantly more computation than the gradient threshold edge detection employed by Horswill for processing Polly’s greyscale image stream. Specifically, the edge detection process works by first separating the image into RED, GREEN, and BLUE color plane images $(r, g, b)$. A histogram profile is calculated for each, and the maximum intensity for each histogram is found as well as the maximum over all color planes. We then apply smoothing to each image before convolving with a Sobel edge detection filter, a standard edge detection filter often used in robot vision because of its high tolerance to variability in illumination conditions. The separate edge strength images are then averaged together. However, we set to zero the edge strength of any pixel that comes from the edge image of maximum color plane intensity, for which the intensity value is reasonably close to the maximum value. This reduces the detector’s dependence on uniform lighting by helping to eliminate edges which come from the reflectance of overhead lights on the floor even in the presence of slowly changing floor color perception. The resulting averaged edge strength image is then thresholded to give a binary edge detected image. Compared with thresholding the image resulting from applying a first-order gradient differencing operator, the visual processing described here requires approximately $O(n)$ pixels more computation. The detector used has been found to be much more robust to changes in room lighting and to shadow-casting objects which violate the GPC, like tables and chairs. In Figure 2.3, which shows the comparison between the two detectors for a set of styrofoam blocks and for a wood grained table, the color enhanced detector used by Erik picks up the table legs and the table shadows and chairs in the background, but also incorporates shadows cast by the styrofoam blocks. So the tradeoff in extra computation for more robust detection in the face of constraint violations is somewhat offset by the perception of somewhat fuzzier or enlarged obstacles. This is not necessarily a bad thing when
considering the problem of obstacle avoidance, and can even be viewed as providing an additional boundary cushion or error margin.

2.2.2 Using the Obstacle Detector for Reactive Control

For a holonomic robot with a relatively low center of gravity, a simple reactive obstacle avoidance control system can readily be constructed from $RDM_t(u)$ whose qualitative behavior is invariant with respect to any strictly increasing function $f(\cdot)$ applied to $RDM_t(u)$. This is important for a reactive controller because it obviates the need to find the transformation mapping pixel distance to true metric distance as a function of radial direction. Horswill’s collision avoidance routine, described here for completeness, bears similar relation to the scheme employed by the Actuator Control module for both head and base actuation. The controller partitions $RDM_t(u)$ into
three freespace functions, left, right, and center:

\[
\begin{align*}
  l(t) &= \min_{u < u_o} RDM_t(u) \\
  r(t) &= \min_{u > u_o} RDM_t(u) \\
  c(t) &= \min_{|u-u_o| < w} RDM_t(u)
\end{align*}
\]  

(2.4)

where \( u_o \) is the \( u \) coordinate of the center column of the image, and \( w \) is an implementation-dependent width parameter. Figure 2.4 shows the values for the radial depth map from a single image and pose of the styrofoam blocks (\( w \) is taken to be approximately twenty percent of the field of view). Although translation and rotation are coupled for a holonomic base, treating them as approximately separable allows us to model each degree of freedom by a first-order, zero delay subsystem. This allows for simple proportional control of the form:

\[
\begin{align*}
  \dot{\theta}(t) &= c_\theta (l(t) - r(t)) \\
  v(t) &= c_v (c(t) - d_{\text{min}})
\end{align*}
\]  

(2.5)
where $\dot{\theta}(t)$ is the rotational velocity of the robot, $v(t)$ is its translational velocity, $d_{\text{min}}$ is the closest the robot should ever come to an obstacle, and $c_\theta$, $c_v$ are user defined controller gains. This controller will cause the robot to turn in the direction of greater freespace at a rate proportional to the difference in freespace in the left and right hemispheres of its field of view; and drive forward or backward proportional to its distance from the nearest obstacle in view. This is a reasonable choice for reactive control. When the robot is in open space the robot can move quickly and turns only a little in each control cycle. In more cluttered spaces, the projection of obstacles in the image has more drastic influences, and the robot will turn more and slow down to avoid impending collision. If the robot can actively tilt its head, the robot can keep the nearest object in its field of view at near constant height in the image by applying the head tilt control law:

$$\dot{\phi}(t) = c_\phi (c(t) - d_o)$$

(2.6)

where $\dot{\phi}(t)$ is the head tilt velocity, and $d_o$ is the desired distance to the nearest obstacle and should be greater than $d_{\text{min}}$. This will stabilize $c(t)$ about the desired operating point causing the robot to continuously drive forward at near constant velocity, except when the limits of head tilt are reached. Unfortunately, at the time of writing, the pan-tilt unit had trouble smoothly actuating, so the camera head had to be locked in a static downward tilt for the system tests results discussed in this thesis. The negative impact this had on overall system functionality is discussed in chapter 7.

### 2.3 Subsystem Analysis

Horswill’s algorithm preforms well under a variety of real indoor environments where noise caused by carpet stains, specular reflection off metallic surfaces, irregular or non-uniform lighting resulting in strong shadows, violate the algorithm’s idealizations. The algorithm’s failure modes under these less-than-perfect conditions usually results
in safe behavior since the robot’s image operators prefer false-positives (the floor is really clear, but something is detected) to false-negatives (the floor is obstructed, but nothing is detected).

Horswill’s algorithm is a purely reactive control mechanism: it forgets about obstacles once they are out of view. Combined with other perceptual processes and reactive controllers in a subsumption architecture [2] a robot agent can be directed to traverse hallway corridors and open atriums all the while relying on the obstacle avoidance routine to steer it toward freespace. However, navigation requires more than just reacting to what can be directly sensed: it requires some memory of recent experience, and a good sense of how to efficiently use this knowledge to achieve its goal oriented task.
Chapter 3

Depth Recovery

In order to aggregate visual information over time, the robot transforms a radial depth map \( RDM_t(u) \) into approximate metric depth information. It will then integrate this transformed data into its temporary occupancy grid, the discrete 2-D map in floor coordinates that represents the robot's current beliefs about where obstacles are in the world. The inverse of the perspective projection transform found from doing a linear constrained optimization with known calibration points proved to be accurate enough for this purpose.

3.1 Perspective Projection Camera Model

Figure 3.1 illustrates the perspective projection camera model. In this model perspective rays or lines between points in the real world \( M(x, y, z) \) and a point of convergence known as the center of projection (COP) intersect the image or retinal plane \( \mathcal{R} \) at points \( m(u, v) \). The perspective ray running perpendicular to \( \mathcal{R} \), and passing through \( c \) is called the optical axis, and the COP sits at a distance \( f \), a focal length away. We are free to choose the origin \( C \) of our world coordinate system to be fixed at the COP with the z-axis pointing along the optical axis. This configuration is called the standard coordinate system of the camera, and defines the focal plane \( \mathcal{F} \) to be the \( xy \)-plane. The perspective transform will allow us to describe the relationship between points in the image and points in any world coordinate system more easily.
3.1.1 Perspective Transform: Affine Equations

Following closely the derivation of the perspective projection transform described in [4], we see from our camera model that the relationship between image coordinates and 3-d space coordinates via similar triangles can be written as:

\[
\frac{u}{x} = \frac{v}{y} = \frac{-f}{z}
\]

(notating \(c\) sits at \(-f\) along the \(z\) - axis). The affine equations which then describe a point in the image are given by

\[
\begin{align*}
    u &= \frac{-f}{z}x \\
    v &= \frac{-f}{z}y;
\end{align*}
\]

however, if the axes of the retinal plane have different scalings due to the geometry of the lens or imaging array elements, and the retinal plane origin is not aligned with
c, then these equations become

\begin{align*}
    u &= \frac{-fk_u}{z} x + u_0 \quad (3.1) \\
    v &= \frac{-fk_v}{z} y + v_0, \quad (3.2)
\end{align*}

where \((u_0, v_0)\) are the image coordinate offsets of \(c\), and \(k_u, k_v\) are the proper scalings of the retinal plane axes. Letting \(\alpha_u = -fk_u\) and \(\alpha_v = -fk_v\), we can write this equation linearly as

\[
\begin{bmatrix}
    U \\
    V \\
    S
\end{bmatrix} =
\begin{bmatrix}
    \alpha_u & 0 & u_0 & 0 \\
    0 & \alpha_v & v_0 & 0 \\
    0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix} 
\] (3.3)

where

\[ u = U/S \quad v = V/S \text{ if } S \neq 0. \quad (3.4) \]

According to the equation, as \(z\) approaches zero, the 3-d point \(M\) will be in the focal plane of the camera, and \(u\) and \(v\) will approach infinity. The set of points for which \(S = 0\) are called the *points at infinity* and illustrate that the equation is projective, i.e., defined up to a scale factor [4].

If we allow for a different choice of world coordinate system, we can describe \(M'\), our 3-d point \(M\) expressed in terms of the standard coordinate system, by the equation

\[ M' = R_x M + t, \quad (3.5) \]

where \(R_x\) is a rotation matrix about the 3-d coordinate vector \(\vec{x}\) and \(t\) is the world coordinate system origin's translation vector offset in the *standard coordinate* system's frame.
3.1.2 Projective Geometry

Projective coordinates allow us to embed this affine transformation in our imaging equation using a linear algebra operation, by defining our world coordinates relative to a scale factor \( T \). Using the projective coordinate \((X, Y, Z, T)\) of \( M(x, y, z) \) we can rewrite the imaging equation as:

\[
\begin{bmatrix}
U \\
V \\
S
\end{bmatrix} =
\begin{bmatrix}
\alpha_u & 0 & u_0 & 0 \\
0 & \alpha_v & v_0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z \\
T
\end{bmatrix}
\] (3.6)

which we write in matrix form as

\[
\tilde{m} = \tilde{S}\tilde{M}
\] (3.7)

The tilde notation is used here to denote a matrix or vector that lies in a projective space. Note this differs from equation 3.3 in that both image and world coordinates are defined up to a scale factor. We can always recover the true world coordinate by dividing through by \( T \), in other words, \( M(x, y, z) = M(x/T, y/T, z/T) \). Our coordinate transformation in equation 3.5 is invariant with respect to scale factor \( T \), so we can write it as:

\[
\tilde{M}' = \tilde{R}\tilde{M} + \tilde{t}
\] (3.8)

where

\[
\tilde{R} =
\begin{bmatrix}
R_{xyz} & 0_3 \\
0_3 & 1
\end{bmatrix}
\]
We can concatenate these expressions into a single 4x4 matrix

\[
\mathbf{K} = \begin{bmatrix}
R_{xyz} & t \\
0_3 & 1
\end{bmatrix}
\]

and the imaging equation for arbitrary choice of retinal and world coordinate systems can then be written more simply as

\[
\tilde{m} = \tilde{P}\tilde{M}
\]  

(3.9)

where \( \tilde{P} = \tilde{S}\mathbf{K} \). Faugeras\[4\] gives a geometric interpretation of the row vectors of the matrix \( \tilde{P} \); they define the focal plane and the line joining the optical center to the origin of the coordinates in the retinal plane.

### 3.2 Camera Calibration

The process of camera calibration is the estimation of the matrix \( \tilde{P} \) from a set of known 2-d image and 3-d world point correspondences. If we think of the 3x4 matrix \( \tilde{P} \) as \([ P \ \tilde{p} \) the concatenation of a 3x3 matrix \( P \) and a 3x1 vector \( \tilde{p} \) we again see the imaging equation as the affine transform

\[
\begin{bmatrix}
U \\
V \\
S
\end{bmatrix} = \begin{bmatrix}
x \\
y \\
z
\end{bmatrix} + \tilde{p}.
\]  

(3.10)

For \( N \) calibration points, let \( \tilde{m}_i \) be the set of homogeneous image reference points, and \( M_i \) their corresponding 3-d world points. The projection matrix can be found
Reprojection of Calibration Points

Figure 3-2: Calibration Pattern: Black tape reference points, and red circles show their calculated reprojection.

by minimizing a least-squares cost function of the form

\[ E = \sum_i ||\tilde{m}_i - (PM + \tilde{p}) ||^2, \]  

(3.11)

with respect to \( P \) and \( \tilde{p} \). By writing the relationship between the affine coordinates of the reference points \( M_i \) and \( m_i \) linearly, and adding the constraint that the plane of points at infinity has unit energy to avoid the trivial zero solution, this equation can be solved as a constrained linear least squares problem.

Let \( p_i \) be the \( i \)th row of matrix \( P \), and \( \tilde{p}_i \) be the \( i \)th element of row vector \( \tilde{p} \); then affine equation 3.10 can be rewritten as:

\[ p_i^T M_i - u_i p_3^T M_i + \tilde{p}_1 - u_i \tilde{p}_3 = 0 \]
\[ p_2^T M_i - v_i p_3^T M_i + \tilde{p}_2 - v_i \tilde{p}_3 = 0. \]

\( || \cdot || \) denotes here the \( L_2 \) energy norm.
This defines a system of \(2N\) homogeneous linear equations in the unknowns of the elements of the projection matrix for our \(N\) calibration point pairs. In matrix notation we write

\[
A\hat{p} = 0
\]

where \(A\) is a \(2N\times12\) matrix depending on the 3-D and 2-D calibration point pairs, and \(\hat{p}\) is the \(12\times1\) vector \([p_1^T, p_2^T, p_2, p_3^T, \hat{p}_3]\). Since \(\hat{p}\) is defined up to a scale factor, we avoid the trivial solution \(\hat{p} = 0\), by finding a solution as the minimization of \(\|Aq\|\) subject the constraint that \(\|p_3\|^2 = 1\).

Glossing over some details, we have \(2N\) equations for each point pair correspondence, and 12 unknowns, therefore for \(N \geq 6\) points a solution to our equation exists. In practice, the camera should be calibrated about its operating position, i.e. tilted down at some fixed angle, and more many more points than needed, widely scattered to cover the complete viewing area should be used to improve the accuracy of the results. In addition, these points should be sampled at various heights from the floor plane to avoid ill-conditioned or badly scaled matrix inversions in finding a solution. Table 3.1 shows the calibration point correspondences, the found projection matrix, and error statistics for the calibration pattern shown for a fixed camera tilt position of \(\phi = -0.8\text{rad}\) downward from horizontal at a height of 1.245m. Figure 3.2 shows the calibration pattern, and reprojection of the 3-D reference points. A Matlab code implementation of this calibration technique is included in Appendix A, and an in depth look at the process of constrained minimization and the conditions under which it will succeed for the equations described here can be found in [4].

### 3.3 Recovering Floor Coordinates

The robot's camera is mounted facing forward on a “Directed Perception” pan-tilt unit which sits atop the robot’s body only a few centimeters behind its front face. With the camera at a fixed tilt, the estimated projection matrix \(\hat{P}\) accounts for any
Table 3.1: Calibration Data and Estimation Statistics

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>u</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.6000</td>
<td>1.0050</td>
<td>0</td>
<td>31.4677</td>
<td>166.1140</td>
</tr>
<tr>
<td>-0.3000</td>
<td>0.9000</td>
<td>0</td>
<td>88.2419</td>
<td>190.6754</td>
</tr>
<tr>
<td>-0.3000</td>
<td>0.9000</td>
<td>0.2000</td>
<td>81.6060</td>
<td>168.2193</td>
</tr>
<tr>
<td>-0.3000</td>
<td>1.5000</td>
<td>0</td>
<td>107.4124</td>
<td>91.7281</td>
</tr>
<tr>
<td>-0.3000</td>
<td>1.5000</td>
<td>0.2500</td>
<td>102.2512</td>
<td>40.5000</td>
</tr>
<tr>
<td>-0.1000</td>
<td>1.8000</td>
<td>0.2500</td>
<td>145.7535</td>
<td>27.1667</td>
</tr>
<tr>
<td>0.3000</td>
<td>1.5000</td>
<td>0.3800</td>
<td>226.1221</td>
<td>44.0088</td>
</tr>
<tr>
<td>0.5350</td>
<td>1.5000</td>
<td>0.3800</td>
<td>271.8364</td>
<td>44.0088</td>
</tr>
<tr>
<td>0.3000</td>
<td>0.9000</td>
<td>0.2000</td>
<td>301.3295</td>
<td>169.6228</td>
</tr>
<tr>
<td>0.3000</td>
<td>0.9000</td>
<td>0</td>
<td>225.3848</td>
<td>194.8860</td>
</tr>
</tbody>
</table>

Projection matrix:

\[
\tilde{P} = \begin{bmatrix}
276.5031 & 108.6529 & -109.2647 & 94.1431 \\
9.8268 & -136.7732 & -263.1310 & 356.7305 \\
0.0199 & 0.6519 & -0.7580 & 0.6346
\end{bmatrix}
\]

Intrinsic Camera Parameters:

\[
[\alpha_u = 273.6204 \quad \alpha_v = 275.3820 \quad u_0 = 159.1541 \quad v_0 = 110.4814]
\]

Extrinsic Parameters:

\[
R = \begin{bmatrix}
0.9990 & 0.0179 & 0.0416 \\
0.0277 & -0.7582 & -0.6514 \\
0.0199 & 0.6519 & -0.7580
\end{bmatrix}, \quad t = \begin{bmatrix}
-0.0250 \\
1.0408 \\
0.6346
\end{bmatrix}
\]

Pixel Error Statistics:

\[
MSE = \|m_{calculated} - m_{reference}\|^2 = \begin{bmatrix}
16.6652 & 1.4978 \\
1.4978 & 16.2601
\end{bmatrix}
\]

MaxError(u, v) = [1.3910, 2.1572], MinError(u, v) = [-2.2050, -2.1157]
deviation of the camera's mounting from perfect on-center alignment. The goal here is to recover the 3-d world coordinates of points in the image given that the imaged point came from a point on the floor. According to the projective transformation above, we can find our world coordinates based on the homogeneous image coordinates from the following recovery equation:

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} = \mathbf{P}^{-1} \begin{bmatrix}
  U \\
  V \\
  S
\end{bmatrix} - \mathbf{P}^{-1} \hat{\mathbf{p}}.
\]  

(3.12)

However, the robot only knows the affine image coordinates \((u, v)\), and must determine the homogeneous scale factor \(S\). Using a 3-D world coordinate system fixed to the ground plane with z-axis normal and pointing up as in Figure 3.3, the scale factor \(S\) can be recovered by setting the equation for \(z\) equal to zero. Let \(\mathbf{W} = \mathbf{P}^{-1}\), \(w_{ij}\) its
Recovery of Image Points from World Coordinates

$i^j$th component, and $\tilde{p}_i$ be the $i^j$th component of $\tilde{p}$, then for $z = 0$ we have

$$w_{31}(U - \tilde{p}_1) + w_{32}(V - \tilde{p}_2) + w_{33}(S - \tilde{p}_3) = 0,$$

which upon substitution for $U = uS$ and $V = vS$ and some rearranging of terms gives the solution

$$S = \frac{w_{31}\tilde{p}_1 + w_{32}\tilde{p}_2 + w_{33}\tilde{p}_3}{w_{31}u + w_{32}v + w_{33}},$$

from which the floor coordinates $x$ and $y$ can be found from backsubstitution of $S$ into the recovery equation. However, recovery of the floor coordinates changes with translation and rotation of the robot. A standard change of coordinates is performed to map floor coordinates $(x, y)$ obtained from the recovery equation, into true global floor coordinates $(x', y')$, based on the robot's estimated heading $\theta$ and

Figure 3-4: Accuracy of Floor Coordinate Recovery
position \((x_r, y_r)\):

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
= \begin{bmatrix}
  \cos(\theta) & \sin(\theta) \\
  -\sin(\theta) & \cos(\theta)
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
+ \begin{bmatrix}
  x_r \\
  y_r
\end{bmatrix}.
\]

In the current implementation, this odometric information is taken from shaft encoders in the robot’s base, but could be made more robust if estimated from optical flow, or by map based realignment. Figure 3.4 illustrates the reprojection of the floor coordinates calculated from a radial depth of the styrofoam block calibration image. From this picture we see the accuracy of our calibration, i.e., where the robot thinks these objects are in the world.
Chapter 4

Occupancy Grid Map

4.1 Constructing Occupancy Grids

Robust navigation requires planning, and planning requires having a map. With the ability to recover the real world coordinates of image pixels lying on the floor, the robot builds a map of the floor space for its surrounding environment. The mapping technique employed was derived from the occupancy grid method pioneered by Moravec and Elfes [8]. An occupancy grid is a finite element 2-D grid which models the accumulation over time and space of information collected from a sensor or group of sensors. This technique, which has been employed by numerous robot researchers, discretizes the floor space of the room into uniform blocks, or grid cells, and assigns to each a value between 0.0 and 1.0 measuring the degree to which the robot believes there is an obstacle somewhere in that cell. While occupancy grids are easy to build and maintain, the accuracy of the robot’s sensing information decays over long distances, and so we intend to use the occupancy grid fairly locally, for crossing a room, for example then throw them away. The robot will build more abstract, topological maps for large-scale navigation.
4.1.1 Background on Use and Application to Vision

While the occupancy grid mapping technique has been used extensively for sonar sensing robots, its employment for vision based robots has not had as wide appeal. One reason for this maybe the added complexity of transforming image pixel data into metric depth information, or because until recently cheap on-board off-the-shelf hardware capable of processing such information in real time has not been available. A more likely reason is the awkwardness of using vision to construct occupancy grids. Sonar sensing robots generally get information about a bounded 360 degree view around the robot at each time instance, while a robot with a forward looking camera must look around to acquire similar information. Some researchers have even gone to the extent of using a ceiling facing camera with an ultra-wide field of view and then accounting for image warp in software, as a means of dealing with the irregularity of the robot’s sense-space geometry. Regardless of the reason for its lack of widespread appeal, vision based grid maps can be as robust as their sonar equivalent.

The nature of vision data also differs significantly from sonar data. In vision, the resolution of a single pixel and the overall field of view is dependent only on the camera’s focal length. Sonar data is generally sparse: the field of view sampled from a single sonar is approximately a 30° conic wedge. Vision is also a passive sensor, recovering depth information indirectly. Sensor noise in vision, apart from that resulting from the camera’s pixel acquisition electronics, is dependent on the degree to which the depth recovery model’s assumptions match the given environment. The algorithm used in this paper prefers false-positives to false-negatives. On the other hand, sonar suffers from the problem of “missing objects”, because reflections off corners known as specular reflection and wave absorption by roughly textured surfaces makes obstacles seem far away or absent. Since this sensor noise is usually modelled as an additive gaussian process with variance as an increasing function of range, the average of several adjacent sonar sensor readings is usually used to reduce its chances of error over time. Erik is typical of most grid mapping research robots, having 24 sonars arranged in a ring around its midsection. Although the height of the robot’s
sonar ring dictates the maximum sensing distance as the boundary where the sonar’s conic wedge intersects the floor, its effective sensing distance may be considerably smaller given the sensitivity of the sonar’s receiver electronics.

4.1.2 Integrating Occupancy Evidence

In this thesis work, we applied a Bayesian update rule for the integration of newly acquired freespace/obstacle information into the occupancy grid. The update rule we used is like the one employed by Thrun at CMU [11] for use on sonar sensing robots, but is modified for use with visual sensing.

The update proceeds as follows. Each radial depth map (RDM) that the robot gets gives information about occupied and free areas in its field of view. We recover floor coordinates for pixels in the image which generated the RDM under the assumption that all pixels were imaged from the floor, and map them into grid cells. Now we have to combine the newly sensed evidence with our previous beliefs about the state of the world. We make the assertion that data is “new” if and only if the robot’s pose has changed significantly. This helps simplify the update rule by allowing us to assert a greater degree of conditional independence between sensor samples given the state of the world. Formally speaking, we only perform an update if

$$\Delta s > \Delta s_{\text{thresh}}$$

where $$\Delta s = s_t - s_{t-1}$$ is the change in robot pose since the previous update, $$\Delta s_{\text{thresh}}$$ is some user defined threshold, and $$s_t = (x, y, \theta_{\text{base}}, \theta_{\text{head}}, \phi_{\text{head}})$$ is the robot’s pose at time $$t$$. Next we define

$$X(u, v) : (u, v) \mapsto (x, y)$$

as the mapping from image pixel $$(u, v)$$ to the grid cell containing floor coordinate $$(x, y)$$ under the supposition that every pixel lies on the ground; then we can find the set of grid cells containing pixels in $$I(t)$$, the robot’s image at time $$t$$. This allows us
to construct observations for each grid cell \((x, y)\) given the robot’s pose, which we describe mathematically via the function

\[
O_{(x,y)}(s_t) = \begin{cases} 
\text{occupied} & \text{if } \exists u \text{ s.t. } \mathbf{X}(u, RDM_t(u)) = (x, y), \\
\text{unoccupied} & \text{if } \forall v \geq RDM_t(u) \text{ s.t. } \mathbf{X}(u, v) = (x, y), \\
\text{unknown} & \text{otherwise.}
\end{cases}
\]

This function says that we should declare a grid cell: occupied if there exist any pixels in the RDM which map into it, unoccupied if and only if all pixels that map into it come from pixels below the height of the RDM, and unknown if pixels mapping into it are above the RDM or otherwise. Figure 4.1 illustrates for us the observation of an RDM.

It is convenient to interpret the output of this observation function as sampling a random process where each sample yields an i.i.d random variable having

\[
Pr(occ_{(x,y)} | O^{(k)}_{(x,y)})
\]
(the notation $O_{(x,y)}^{(k)}$ denotes the $k$th observation for grid cell $(x, y)$). This probability of occupancy given the $k$th observation is extremely difficult to model. Although such models may be learned by training neural networks, we have chosen to design our model by hand. Lacking a proper probability distribution of conditional occupancy given observation, a simpler approach was taken in this thesis, whereby the user chooses update rates $\alpha$ and $\beta$ that satisfy the constraints

\[
Pr(\text{occ}(x,y) \mid O_{(x,y)}^{(k)} = \text{occupied}) \leq \alpha
\]

\[
1.0 - Pr(\text{occ}(x,y) \mid O_{<x,y>} = \text{unoccupied}) \geq \beta.
\]

The parameters $\alpha$ and $\beta$ allow us to assign a degree of trust to our observations. We'll assign larger values of $\alpha$ and $\beta$ if we have reason to believe that our sensor can be trusted, and lower values in noisier environments where our obstacle detector is more likely to make mistakes. This simple modelling allows us to approximate a measure of the true probability of occupancy for grid cell $(x, y)$, as the probability of occupancy for grid cell $(x, y)$ conditioned on all observations,

\[
Pr(\text{occ}(x,y) \mid O_{(x,y)}^{(1)}, O_{(x,y)}^{(2)}, \ldots, O_{(x,y)}^{(N)}).
\]

If we assume that each “process sample” is taken at significantly different robot pose, we can assume a modest degree of conditional independence, in other words, $Pr(\text{occ}(x,y) \mid O_{(x,y)}^{(k)}) \neq Pr(\text{occ}(x,y) \mid O_{(x,y)}^{(k')})$, if $k \neq k'$, even though the statistics of observation $Pr(O_{(x,y)}^{(k)})$ and $Pr(O_{(x,y)}^{(k')})$ are usually dependent. This assumption is difficult to justify. In fact, we argue that it is probably not true and will depend on the correlation decay time of our measurements. The difficulty in determining a good sensor model hinges on the fact that this is a complicated function of the robot’s velocity and varies spatially in the image plane for different tilt angles, and lighting conditions. The remaining derivation of the Bayesian update process treats our pseudo-sensor model as though it were legitimate according to the assumption of conditional independence.
Bayesian Updating

Given a sensor model whose output for update $k$ is

$$Pr(occ(x,y) \mid \mathcal{O}^{(k)}_{(x,y)})$$

we can estimate the probability of occupancy given $N$ observations iteratively by successive application of Bayes’ rule and the assumption of conditional independence. Without detailing the derivation, the resulting update is captured by the formula

$$\begin{align*}
Pr(occ(x,y) \mid \mathcal{O}^{(1)}_{(x,y)}, \mathcal{O}^{(2)}_{(x,y)}, \ldots, \mathcal{O}^{(N)}_{(x,y)})
&= 1 - \left(\frac{1 - Pr(occ(x,y) \mid \mathcal{O}^{(1)}_{(x,y)})}{1 - Pr(occ(x,y))} \prod_{k=2}^{N} \frac{Pr(occ(x,y) \mid \mathcal{O}^{(k)}_{(x,y)})}{1 - Pr(occ(x,y) \mid \mathcal{O}^{(k)}_{(x,y)})} \frac{1 - Pr(occ(x,y))}{Pr(occ(x,y))}\right)^{-1}
\end{align*}$$

(4.2)

where $Pr(occ(x,y))$ is the prior probability for occupancy, sometimes referred to as the background occupancy, and is representative of the average occupancy of a grid cell in a mature map [11]. In our experiments, we found that good occupancy grids are constructed when the background occupancy is approximately 0.35 for hallways, but should be higher in more cluttered environments like the lab room where Erik resides.
Modifications to the Rule

The following change to the occupancy probability update equation was applied, to accommodate for the lack of a direct sensor model:

\[
Pr(\text{occ}(x,y) | O^{(1)}_{(x,y)}, O^{(2)}_{(x,y)}, \ldots, O^{(N)}_{(x,y)}) \approx \begin{cases} 
1 - \left(1 + \frac{\alpha}{1-\alpha} \prod_{k=2}^{N} \frac{Pr(\text{occ}(x,y) | O^{(k)}_{(x,y)})}{1-Pr(\text{occ}(x,y) | O^{(k)}_{(x,y)})} \frac{1-Pr(\text{occ}(x,y))}{Pr(\text{occ}(x,y))}\right)^{-1} & \text{if } O_{(x,y)}(t) = \text{occupied}, \\
1 - \left(1 + \frac{\beta}{1-\beta} \prod_{k=2}^{N} \frac{Pr(\text{occ}(x,y) | O^{(k)}_{(x,y)})}{1-Pr(\text{occ}(x,y) | O^{(k)}_{(x,y)})} \frac{1-Pr(\text{occ}(x,y))}{Pr(\text{occ}(x,y))}\right)^{-1} & \text{if } O_{(x,y)}(t) = \text{unoccupied}, \\
1 - \left(1 + \prod_{k=1}^{N} \frac{Pr(\text{occ}(x,y) | O^{(k)}_{(x,y)})}{1-Pr(\text{occ}(x,y) | O^{(k)}_{(x,y)})} \frac{1-Pr(\text{occ}(x,y))}{Pr(\text{occ}(x,y))}\right)^{-1} & \text{if } O_{(x,y)}(t) = \text{unknown}. 
\end{cases}
\]  

(4.3)

Since the occupancy grid probability represents all previous applications of the update rule, the previous value can be substituted for the probability of occupancy conditioned on all previous observations. In the case where \(O_{(x,y)}(t) = \text{unknown}\), the occupancy probability is unchanged. Figure 4.2 gives an example of the evolution of the occupancy grid as the robot drives around the lab room. The images on the left show a snapshot of what the robot sees (the pink shading indicates pixels above the RDM). The images on the right depict the occupancy grid map. The blue square marks the robot’s position at the labelled control step, white areas indicate freespace, black areas are obstacles, and grey areas have not yet been explored. Between control steps \(t = 4\) and \(t = 90\), not shown in the sequence, the robot has driven around the room and discovered the back wall. Control steps \(t = 90\) and \(t = 91\) shows the robot viewing the same styrofoam blocks from an alternate viewing angle. We can clearly distinguish the layout of these blocks on the floor; they are the dark obstacles at near the bottom of the explored space in the map at step \(t = 91\).
Figure 4-2: Evolution of an Occupancy Grid
Chapter 5

Navigation: The Planning Module

5.1 The Agent-Environment Feedback Model

Consider the problem of robot navigation as represented by an agent in an environment for which a state-space description can be written. The state-space evolves according to a Markov process, and the action of the agent is described as a function of the current state at every point in time. For the system described in this thesis,
the occupancy grid captures the robot’s belief about the underlying structure of this process. As the robot drives around it continuously gains information which it aggregates into its occupancy grid. In the limit of a large number of observations, grid cell occupancies approach their final or mature values with probability 1. Therefore, any description of the state of the world that has a counter-factual belief for a grid cell’s occupancy in the interim, must have a transition probability of 0. For example, the state of the world would be contradicted if the robot occupied a group of grid cells that it earlier claimed were occupied by something else. The universe of belief states about the world is therefore constantly shrinking as more grid cell’s occupancy becomes known with certainty; the Markov process is converging to a steady-state and is in fact converging to a single unique state of the world we call the full-information state. Explicitly speaking, as it drives around integrating information collected at different poses, the occupancy grid becomes increasingly refined according to a Bayesian updating process, and the robot’s belief about the global structure becomes known with certainty.

The robot’s behavior is affected by its current state of knowledge about the world. Conversely, the rate of flow and content of information about the world are affected by the robot’s action choices in each period. For example, the appearance of a box in a narrow hallway may mean that the most direct path is blocked and that the robot should look for an alternate route to its goal or desired destination, possibly through another room or hallway corridor it has not yet seen. Clearly, its actions (actuator velocities) alter its pose, possibly exposing previously unseen details of the world which may cause it to react differently in the future. In addition, the robot may pay a penalty for not finding an alternative route quickly enough. Perhaps it is in need of a recharge and every second it spends looking for a way around the hallway obstacle, the longer it will have to spend at the charging dock. The robot’s navigation strategy should therefore try to seek out the best possible alternative as the route that allows the robot to get to the goal in the shortest amount of time. Navigating efficiently in this sense is then tantamount to finding a trajectory through the state-space that minimizes the cumulative cost for getting to the goal. Since the
robot's future is largely dependent on its current actions, position, and knowledge of the global space, we look to the theory of Markov decision processes (MDP)s and optimal control for finding good navigation strategies.

5.1.1 Overview of Markov Decision Processes

An MDP models the synchronous interaction that takes place between a goal seeking agent and its environment as it tries to minimize some cumulative cost function. In general it is defined by a set of states, action, the one-step dynamics of the environment, and a penalty signal. Formally we define this penalty signal which passes from the environment to the agent as a sequence of numbers, \( c_t \in \mathbb{R} \). Since the agent's goal is to minimize the total amount of penalties it receives while interacting with the environment, we define the cumulative cost \( C_t \) as the sum of penalties paid at each future time step:

\[
C_t = c_{t+1} + c_{t+2} + c_{t+3} + \cdots + c_T
\]

where \( T \) is a final time step, or the end of an episode (finite-horizon). We can also define the discounted cost for continuing or non-episodic tasks \( T = \infty \) (infinite-horizon), as:

\[
C_t = c_{t+1} + \gamma c_{t+2} + \gamma^2 c_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k c_{t+k+1}
\]

where the discount factor \( \gamma \in [0, 1] \) measures the degree to which future costs will impact current decisions.
We represent a finite-state MDP by the tuple, \((\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C})\) where:

- \(\mathcal{S}\) the finite set of states in the world.
- \(\mathcal{A}\) the finite set of actions for every state.
- \(\mathcal{T}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]\) is the set of transition probabilities, and \(T(s, a, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)\), defines the transition probability function, which quantifies the one-step dynamics of the state-space as the probability of going to state \(s'\) from state \(s\) by taking action \(a\).
- \(\mathcal{C}: \mathcal{S} \times \mathcal{A} \to \mathcal{C}\) defines the costs for being in each state \(s\), and acting according to some behavior policy.

\[
C(s, a) = E[c_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s']
\]

represents the immediate or expected value of the next penalty paid for transitioning to \(s'\) taking action \(a\) in state \(s\).

In general, the expected cumulative cost will be dependent on the trajectory that the agent follows over the course of an episode or lifetime in the infinite-horizon case. The agent’s trajectory is the direct consequence of its actions, which are selected by the agent in each state according to some control or policy \(\pi(s, a)\): a mapping from situations (states) to actions defined with respect to a value function – a function of state (or of state-action pairs) that estimates how good it is for an agent to be in a particular state (or to perform a particular action in a given state) [10]. The policy function \(\pi^*\), characterizes optimal behavior for the agent, and is the unique mapping of situations (states) to actions that minimizes the expected discounted future cost

\[
E\left[\sum_{t=0}^{\infty} \gamma^t c_t\right]
\]
for the agent starting in every state.

### 5.1.2 Sidebar: On the Applicability of the MDP Framework

The navigation task considered in this paper can still be modelled as an MDP by asserting that the state of the agent and environment collectively satisfy the Markov property – the probability of transitioning to some state in the next period is only dependent on the previous state and action taken, and not on the agents trajectory through the space. However, the MDP model is not directly applicable to the robot navigation problem. We mention here that the state of the robot and environment taken collectively as prescribed by the cross space of position (grid cell location), and knowledge state (the values of the occupancy grid at time $t$) is continuous and in fact infinite. There exists an optimal policy in this large space, but it is too hard to find. Even with a discretization of the values of the occupancy grid to approximate this optimal solution, the problem is immense as the number of states would be exponential in the number of discretization levels. For reasonably sized occupancy grids even a binary discretization would be too large to handle reasonably by current solution methods. Blei and Kaelbling have employed such techniques to solve similar but much smaller Bridge Problems, in which an agent tries to find paths of shortest expected length through a number of islands given prior probabilities of connecting bridges being intact, but have noted the inapplicability of their methods to problems of even a moderate number of agent locations [1]. Therefore, while viewing occupancy grid and pose as our state-space is optimal, it is nonetheless computationally impossible. Instead, we choose not to model the fact that the robot's sensing will in fact change the robot's map, and simply treat the map as static in each period. Under this simplified model, the robot solves for the optimal policy in the static map, assuming occupancy probability is the probability that the robot will be able to traverse that particular grid cell. It then takes one step of that policy, gets new sensor data which it uses to update the occupancy grid, and computes a new plan.

It is important to make the distinction that the overall behavior of the robot is sub-optimal in the infinite-state MDP sense. In general the robot's policy is non-
stationary, it evolves over time and under the worst circumstances may change drastically at any moment. More specifically, Erik’s algorithm continuously re-solves a finite-state MDP at each control step, using the value iteration algorithm [7], a form of dynamic programming which has been popularized for its use as an any time planner in stochastic domains. Since the robot continuously gains information, by reseeding the planner’s information about the optimal policy in the previous period, the policy function converges to the optimal policy found under full information as the occupancy grid map matures. The extent to which this is true is clearly debatable: previously occluded obstacles can break apart well connected land masses, and uncovering freespace can bridge seemingly remote islands causing the policy to thrash around as the robot loses track of the state of the overall process. For example, consider a partial dividing wall between the robot and its goal. There are two gaps which the robot might pass through in trying to get to the goal. Suppose the gap on the right is the beginning of a corridor, and the gap on the left leads into the other half of the room. The robot will choose either the path to left side of the wall or the path on the right. If it sees an obstacle near the passage on the left it will choose to
shoot through the gap on the right. As it passes through the opening on the right at
sees that it cannot pass through the corridor wall on its left, we hope it will update
its map, recompute a new plan, and decide to back up and try the passage on the
left rather than traverse down the unknown corridor. Approaching the passage on
the left the robot may get lucky and realize that the obstacle seen earlier does not
connect with anything else, and it can safely maneuver around it to reach the goal.

Figure 5.3 illustrates a similar problem using an artificial map. The robot is trying
to get to a goal in the center of the map. It knows there is an enclosure going around
the interior of the room, but cannot see the blockade in the eastern corridor until it
turns around a corner, the resulting policy change shows the severity of this change
of information state. The robot’s choice of action is to take a step in the direction of
steepest descent in the value function, i.e. a manhattan action in the direction taking
it to the next cooler colored grid cell. The maps shown here have many fewer states
then the maps constructed by the robot, and are intended to help the reader gain
insight into the algorithm.

States and Actions

An MDP admits a model for an agent acting with uncertain actions, however, the
Markov property implies that the agent’s state is fully known at all times. By assum-
ing that the robot’s odometry does not drift, there is never any uncertainty about its
current position. Therefore, the state of the robot at time $t$ apart from its belief about
the world is captured completely by its grid cell location $(x, y)$. In reality this is not
true, the odometry can drift and the robot’s belief about the structure of the world
can become skewed. However, we assume that drift is slow enough for navigation in
the spaces in which the algorithm is employed, or can be corrected to within sub-grid
cell resolution. The more realistic partially observable POMDP model, would treat
the robot’s odometry as an observation of the current state, and has been studied in
great detail by Kaelbling et al. [7]. However methods for dealing with such models are
considerably more complex and do not scale well to the robot’s world representation
as described in this thesis.
Figure 5-3: Evolving belief state: cooler colors indicate lower map occupancy, and lower total expected cost to goal in the corresponding plan.
**Transition Function**

The robot's belief about the world is captured in the model by $T$, the transition function, which models the robot's physical movement in the world based on grid cell occupancy. In the current implementation Erik has at most four actions available to it from any state. Since we make the *static world* assumption, the robot's actions are deterministic and we define $\text{succ}(s, a)$ as the deterministic successor state that results from applying action $a \in A(s)$ to state $s \in S$, where $A(s) \subseteq A$ is the set of actions moving the robot one grid cell step in one of four directions (NORTH, EAST, SOUTH, WEST). In general, the occupancy of a particular grid cell indicates the degree to which the robot will be unable to traverse a particular location in the world. Therefore, the robot will enter $\text{succ}(s, a)$ with probability $1 - \text{occ}(\text{succ}(s, a))$, and will not be able to enter with probability $\text{occ}(\text{succ}(s, a))$, where $\text{occ}(s = (x, y))$ is the probability of occupancy for grid location $(x, y)$.

If we view attempts to traverse a particular grid cell as Bernoulli trials, with the probability that the robot will succeed in entering grid cell $s$ is $1 - \text{occ}(s)$, then over an infinite number of trials the probability of success approaches 1. Therefore, the robot can enter any grid cell with probability of occupancy less than 1. However, this model for crossing grid cells is clearly inaccurate. For higher occupancy values the robot can be fairly certain that repeated attempts to traverse a particular grid cell will always end in failure (it is impossible to walk through walls). We therefore threshold to 1 values in the occupancy map above some value $\text{occ}_{\text{thresh}}$. In addition, the robot moves slowly enough that the transition probability for moving more than one grid cell away is effectively 0. Formally, we write this sparse transition function as:

$$T(s, a, s') = \begin{cases} 
1 - \text{occ}(\text{succ}(s, a)) & \text{if } s' = \text{succ}(s, a) \text{ and } \text{occ}(\text{succ}(s, a)) < \text{occ}_{\text{thresh}}, \\
\text{occ}(\text{succ}(s, a)) & \text{if } s = s', \\
0 & \text{otherwise.}
\end{cases}$$
The Cost Function

The desired robot behavior should be to drive toward the goal along the clearest and shortest length path possible. The cost function encourages this behavior by penalizing the robot for every action such that $\text{succ}(s, a)$ is not the goal state $s_{\text{goal}}$. Formally,

$$C(s, a) = \begin{cases} 
1 & \text{if } s \neq \text{goal} \\
0 & \text{otherwise.} 
\end{cases} \quad (5.1)$$

Given two paths of equal path length, the robot will choose the clearer path as the one that minimizes the number of steps to reach the goal. Alternatively, we speculate that better behavior, although still far from optimal, can be exacted by implementing a cost function that also penalizes the robot for taking actions which bring it into highly cluttered or difficult to navigate areas of the state-space. This is something akin to risk aversion, since cluttered areas are more complex the likelihood that the robot will end up bumping into an obstacle is greater.

5.1.3 Solving finite MDPs

The solution to our finite MDP is a stationary policy, $\pi : S \rightarrow A$, a situation-action mapping that specifies the behavior of the robot acting over our infinite-horizon discounted cost model. In reality, we are actually solving $\pi_t$ the policy used to choose the action on the $t^{th}$ to-last step of the non-stationary policy in the infinite state-space. We omit the subscript $t$ here for convenience and to avoid confusion with the more difficult solution of acting optimally over changing belief state/occupancy grid.

We begin by defining the value function $V_{\pi}(s)$: the cumulative cost the agent can expect to pay for starting in state $s$ and executing stationary policy $\pi$. This function can be defined recursively:

$$V_{\pi} = C(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_{\pi}(s'), \quad (5.2)$$
the immediate cost $C(s, \pi(s))$ paid today for executing policy $\pi$ in current state $s$ and the discounted cost of the future evaluated as an expectation over the expected discounted sum of future costs for being in each next-state $s'$ weighted by its likelihood of occurrence $T(s, \pi(s), s')$. Here we see the importance of the discount factor $\gamma$. For small values of $\gamma$ the agent is myopic, it is more concerned with the return that it currently expects based on its next state transition, but for values closer to 1 the agent gains some "vision", and is equally interested in the return it expects further down the road. The value function is found explicitly by solving this recursion for the unique simultaneous solution of a set of linear equations, one for each state $s$.

This illustrates how a value function is computed from a policy $\pi$. We can go the other way too and compute a policy with respect to a value function. Since the robot always exploits its environment, we say that it uses a greedy policy obtained by taking the action that minimizes the expected immediate cost plus the expected discounted value of the next state computed from $V$. We write the greedy policy computed with respect to value function $V$ as

$$\pi_V(s) = \arg \min_a \left[ C(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V(s') \right].$$

By thresholding the occupancy map, the robot’s transition model makes the claim that it is impossible to enter grid cells that it is fairly certain are occupied. The same constraint must be matched by the robot’s choice of action at each step. Simply following a greedy policy is incorrect. When the robot is close to the goal, the value function indicates that it is reasonable for the robot to pass from a state occupied by the wall into an adjacent clear cell which takes it closer to the goal. Since it is clearly impossible for the robot to be in a state occupied by the wall, actions which might take it into these states are eliminated by setting the costs to infinity. Thus the modified policy followed by the robot draws it along least cost paths toward the goal by forcing it to take actions of steepest descent in the modified $V$ function, eliminating actions which would allow it to cross into an occupied state.
initialize $V_k(s) := \infty$ initially for all $s$
$k := 1$
loop
  $k := k + 1$
  loop for all $s \in S$
    loop for all $a \in A(s)$
      $Q_k^a(s) := C(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_{k-1}(s')$
    end loop
  $V_k(s) := \max_a Q_k^a(s)$
end loop

until $|V_k(s) - V_{k-1}(s)| < \epsilon$ for all $s \in S$

Table 5.1: The value-iteration algorithm for finite state-space MDPs.

5.2 Value Iteration and Planning Algorithm

The value-iteration algorithm finds $\pi^*(s)$, the unique optimal behavior policy for the finite MDP conditioned on the robot’s current belief state at time $t$. It is a greedy policy with respect to the value function whose unique solution is defined by the set of equations

$$V^*(s) = \max_a \left[ C(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right].$$

Table 5.1 details the value-iteration algorithm. Although we seek an optimal value function to the infinite-horizon discounted model, the state-space is discrete and the goal can be reached exactly. Therefore, we can approximate for the infinite-horizon by computing the sequence $V_k$ of discounted finite-horizon optimal value functions. For a more in-depth discussion of this algorithm we refer the interested reader to [7].

5.2.1 Stopping Criteria: Planning Time verse Control Time

Value iteration is called an anytime planner because each iteration finds a value function $V_k(s)$ that is guaranteed to be a closer approximation to $V^*$. Fortunately, it frequently happens that the greedy policy for $V_k$ with fairly large error is in fact optimal. Value-iteration works by pushing cost out from the goal state in the direction
of least resistance in the map, in our case paths of lower occupancy. Since the planner is always being reseeded with the cost function found in the previous period, a radical change in the map can have deleterious effects on the time necessary to find the new optimal policy because the robot must essentially first unlearn the previous value function. To see why, take for example the effect of a radical change in plan optimality based on the discovery of a new obstacle as illustrated earlier in Figure 5.3. This is a worst case scenario for the planner: before the new plan can be found the planner must push cost information back through the only clear passages before it can build up the cost along paths that it previously thought were clear but are in reality now known to be blocked. Since the software on Erik runs serially, both planner and control run once per sensor update. Even in a threaded implementation, in which these processes could be interleaved on the robot’s cpu (a closer approximation to the parallel computation that most living organisms are capable of), the maximum velocity at which the robot can drive safely is dependent on the rate at which it can form new optimal policies. The problem is that the robot’s velocity is determined during the previously control cycle. If the robot chooses to spend time planning until it has an optimal policy given the new obstacle information, it runs the risk of driving more than one grid cell, possibly into an obstacle. This is a violation of the transition function model and therefore compromises the optimality of the robot’s prior choice of action. If, instead, it chooses to end planning before its policy is optimal, and maintain the rate at which it can choose new actions, it runs the risk of choosing an action that is no longer optimal given the current state of the world.

We were able to mediate this dilemma by letting the planner run until the optimal policy could be found or until a timer expired, whichever came first. In addition, we limit the robot’s base actuator velocities to be proportional to the average time spent in planning to avoid driving more than one grid cell between control cycles. Since the policy can only change with the introduction of new information in the occupancy grid, the robot only runs value-iteration if the robot’s pose changes significantly. This helps alleviate the problem of making the robot drive to slowly in more complex environments where the robot’s planning time is longer on average. In practice,
we found that planning time could be reduced significantly, without disturbing the optimality of the robot’s policy found in each planning phase, for most environments. This result stems from the fact that the unique policy $\pi^*$ often converges long before $V_k$ approaches $V^*$, and therefore allows the robot to terminate planning with a near-optimal policy in bounded time as a function of $\gamma$ and the magnitude of the Bellman error [7]. The rich structure of a value function, the occupancy grid which generated it, and the resulting policy it produces are illustrated in Figure 5.4. The negative of the value function is pictured for clarity.
5.2.2 Incorporating Proprioceptive Knowledge

Using the computed policy at each time step, the robot chooses the best possible action given its current state. In general, the resolution of the robot’s occupancy grid map will influence the usefulness of this policy. If the grid cells are too large the robot’s grid map will cause more drastic changes to the policy, but will allow the planner to run faster since there are fewer states; on the other hand, small grid cells allow for finer planning, but require much more computation if the optimal plan is significantly different from the one found in the last period. A good grid size will trade off flexibility and performance, and should take into account the precision, accuracy and density of information retrieved by the robot’s sensors. The best size determined empirically has been found to be about 0.1m which is about the maximum error calculated for the robot’s depth recovery system. Since each grid cell \( (x, y) \) is much smaller than the robot, a means of incorporating knowledge about robot geometry into the policy is necessary to prevent the robot from coming too close to obstacles in the world. The modification we proposed and implemented accounts for the robot’s geometry by modifying the transition function in the following way. First, the robot calculates \( \mathcal{R}(s) \subseteq \mathcal{S} \), the set of grid cells occupied by the robot when its center is in state \( s \). If the longest dimension (diameter) of the robot in the xy-plane is \( \max_a B(s, a) \), where \( B(s, a) \subseteq \mathcal{S} \) defines the set of grid cells along the perimeter of \( \mathcal{R}(s) \) adjacent to the edge of the robot perpendicular to the direction of the vector \( \text{succ}(s, a) - s \), then the robot’s probability of taking action \( a \) from state \( s \) and entering \( s' \) is 1 minus the joint occupancy of grid cells \( B(s, a) \). Formally we write

\[
U(s, a) = 1 - \prod_{s_b \in B(s, a)} (1 - \text{occ}(s_b))
\]
Figure 5-4: The value function and policy mature with time and exploration. The maps on the left are occupancy grids with stable wall obstacles clearly defining the boundary of the explored space. The associated value function which it generates is to its right.
Figure 5-5: The Modified Transition Function: Erik’s diameter is three grid cells in length.

and the modified transition function $T(s, a, s')$ becomes,

$$T(s, a, s') = \begin{cases} 
1 - U(s, a) & \text{if } s \neq s' \text{ and } U(s, a) < \text{occ}_{\text{thresh}}, \\
U(s, a) & \text{if } s = s', \\
0 & \text{otherwise.}
\end{cases} \quad (5.3)$$

This transition function models the robot’s physical ability to shoot gaps, and encourages the robot to steer wide of obstacles. Figure 5.5 depicts the robot’s situation in the occupancy grid, and the calculation of joint occupancy for deterministic successor states $\text{succ}(s, a)$.

### 5.2.3 Optimal Behavior: Exploitation versus Exploration

In each control interval, the robot applies action $a = \pi^*(s)$, gains new information, and recomputes $\pi^*(s)$. However, as stated earlier, this policy is only optimal around a fixed point in the belief space. As the occupancy grid map matures, the robot becomes more certain about the structure of its environment. While it is difficult to quantify how much structure of the world needs to be known before the robot can formulate the optimal policy given full information, in some sense we don’t really care whether it finds the truly optimal policy in the limit; we just want it to get to the goal. The degree to which this will be true is dependent on the complexity of the
environment’s global structure and the robot’s ability to trade off *exploitation* (actions leading directly toward the goal), and *exploration* (actions taken to gain information). This tradeoff between exploitation and exploration is handled implicitly on Erik by an emergency obstacle avoidance routine. If an obstacle appears at close range along the robot’s path, it backs up a small amount and turns in away from the obstacle directly in front of it. This exploratory action is difficult to model because it only occurs when the robot’s assumptions about the state of the world do not coincide with reality. However, in most cases it will only trigger a small fraction of the time, and may lead the robot to new discovery, allowing it to recover from model inconsistencies.
Chapter 6

Actuator Control

The Actuator Control subsystem controls the base and pan-tilt head movement of the robot in response to the greedy action computed from the current policy given the robot’s current grid cell location. Although the robot’s policy allows it to determine the best action from any state, the model was built under the assumption that robot never transitions more than one grid cell away in any given period, and takes mostly deterministic actions. The base actuation is grounded in this MDP model and therefore attempts to adhere to its assumptions as closely as possible. Given that the robot take Manhattan actions from any grid cell, the controller attempts to decouple as much as possible rotation and translation so that actions approximate the discrete actions in the grid.

6.1 Base Actuation

In the current implementation the robot can only move at most one grid cell (NORTH, SOUTH, EAST, WEST), in the map. In terms of movement in the real world this corresponds to translating \([(0, \rho), (0, -\rho), (\rho, 0), (-\rho, 0)]\) in the \(xy\)-plane respectively, where \(\rho\) is an incremental step along an axis dimension. Decoupling of rotation and translation is accomplished by forcing the robot’s translation velocity to zero while the absolute difference between the robot’s heading and a desired direction is greater than some small delta, \(\delta_r\). As an added sanity check to the system, the
translational velocity is proportional to the normalized height of the lowest obstacle in the center of the image at time $t$, with a constant of proportionality, $\rho = \frac{\text{mesh width}}{\text{control.dt}}$, the grid cell width divided by control.dt, the elapsed time since the previous control cycle. This helps ensure that the robot’s velocity is proportional to the frequency at which it can make control decisions for the robot travelling at most one grid cell per control cycle.

Formally we describe the base actuation by the equations:

$$\dot{\theta}(t) = c_\theta(\theta - \theta_{\text{desired}}) \quad (6.1)$$

$$v(t) = \begin{cases} 
\rho \dot{c}(t) & \text{if } |\theta(t) - \theta_{\text{desired}}| < \delta_r, \\
0 & \text{otherwise} 
\end{cases} \quad (6.2)$$

where $\dot{c}(t)$ is the normalized image center minimum $c(t)$ defined in chapter 2, and $c_\theta$ is a positive gain factor, and $\theta_{\text{desired}}$ is the angle of the desired action direction.

### 6.2 Pan-Tilt Actuation

The head tilt control attempts to keep $c(t)$ above a minimum value, and uses the control law described in section 2. Unfortunately, because of a buggy serial interface to the pan-tilt unit, velocity control did not function smoothly and so we conducted all system tests in this thesis with the head tilt locked at $\phi = 0.8$ radians downward tilt from horizontal. This severely impacted the performance of the system, because it forced competition between the robot’s grid map actions and its emergency obstacle avoidance routine. When the robot’s policy takes it close to walls its obstacle avoidance routine sometimes triggers, overriding the policy’s greedy action and causing the robot to rotate away from the wall until it is out of sight. However, the robot’s policy may not necessarily change so quickly and therefore instructs the robot to turn back toward the wall, sometimes failing to pick up the lower edge and putting erroneous freespace into the grid where in fact the wall should be. To make matters worse, the incorrect data can serve to generate a policy where the robot thinks it desirable to
pass through the grid cells where the wall used to be. If the pan-tilt unit were working properly this would not happen, since the robot would be able to look further down in response to its proximity to the wall, thereby strengthening its initial assertion that there is in fact a wall in that part of the room, and generating a policy that will not take it any closer. The ability of the robot to get up right near obstacles is dependent on its ability to continue picking them up at close range. A similar problem happens with obstacles are shorter than the robot, whereby its policy may steer it near a wide box or table, and then it turns, failing to pick up the true bottom edge, and integrating freespace where the obstacle truly is, and placing edges picked up from the top of the obstacle at a further distance away.
Chapter 7

Results

We tested the robot’s navigation system on different sized maps, varying and tuning the system parameters for best observable behavior given different environment conditions. Without a doubt we observed the best robot behavior in environments that closely matched the constraints of the obstacle detector. However, that the system still works reasonably well even in the presence of substantial violations of these constraints, i.e., tables, chairs, shadows, and reflections off shiny floor surfaces, that are ever present in the various rooms in which we tested the system, seems encouraging for this approach.

7.1 System Analysis

Erik successfully navigated rooms in three very different office-like settings within the AI building, two of which were actually carpeted. However, we found the system fairly difficult to tune optimally. The algorithm, for reasonably sized grids, requires a fair amount of computation, rendering it incapable of moving very quickly most of the time. Therefore, the number of tests we were capable of running given the time available was limited. Moreover, given the large number of parameters, any attempt to analyze the system systematically given the often complex cross-interaction between these parameter would require hundreds, possibly thousands of testing hours to prove anything statistically significant. We tested the most important system parameters
over reasonable ranges for the given environment, holding the other parameters constant at either the midpoint of each range or at the value that gave best performance in previous tests.

7.1.1 The Problem with Camera Tilt

Adding to the difficulty of testing and analyzing system performance under different parameter values was the problem with having a fixed camera head tilt. Many times the system failed to work correctly, and even during the best test runs we often had to nudge the robot or manually steer it away from an imminent crash when its obstacle detector failed. As discussed in the previous chapter, many of the failure modes we witnessed were directly related to the robot’s inability to tilt its camera head. The kinds of error in perception introduced by this software bug proved hazardous to the overall system performance. Figure 7.1 illustrates the consequences of not having camera tilt actuation, even in the most pristine office environment in which we tested the system. As the robot drives back toward its starting position, it clearly sees the file cabinet reinforcing its belief that the location where the file cabinet sits is indeed occupied. However, its desired path takes it north in the map, very close to the file cabinet. Since it cannot tilt its head down any further, as it gets close to the cabinet its emergency collision avoidance causes it to back up slightly. Eventually the planner regains control and the robot turns back toward the file cabinet. This time the obstacle detector misses the boundary edge where the carpet and cabinet meet, because it has moved too close. The robot now sees freespace which it integrates into the occupancy map at the position where it once believed it saw the file cabinet. This change in belief causes the robot’s planner to further disagree with the actual state of the world. By manually backing the robot up, and turning it this malfunction can often be corrected fairly quickly, and the robot will then continue on its path toward home.
Figure 7-1: Obstacle Detector fails leading to imminent collision.
7.1.2 Map Size Parameters

The navigation system can tractably support square occupancy grids slightly larger than 200 grid cells on a side, or 40,000 cell states. However, since we only intend to use the system for room level navigation, the ability to support grids much larger than this is unnecessary. We experimented with a few different sized grids and mesh sizes appropriate for the sizes of the rooms, and the diameter of the robot. A high density of grid cells generally leads to more clearly defined maps, but a low-density map is less demanding on the planner. We varied the grid cell width over several different values ranging from about a third of the robot’s radius, 0.1m, to little more than a full robot radius or about 0.3m, and tried grid maps spanning 5m to 20m on a side, which seemed larger than the amount of space the robot could fully-explore for most spaces, but also prevented the robot from “running out of map”. Most of these tests were conducted in the playroom on the 8th floor of the AI lab building. Figure 7.2 shows some maps generated for the lab room for different mesh widths (top 3), and for different sized maps of several other spaces using a mesh width of 0.1m (bottom 4). Since these images were collected over different test runs in which either the robot was driving autonomously or we drove the robot around manually, there is substantial variation between the maturity of the maps of the clean lab room. Also, the deviation in the average grey coloring in the bottom images represents different background occupancy levels, with lighter coloration being lower occupancy. Note the blue square represents the space occupied by the robot at the time the snapshot was taken. Additionally, we observed that the robot had trouble navigating cluttered spaces with the larger grid cells. Intuitively this makes sense, large grid cells approximate the world with less precision. A more flexible system might store several grid maps of varying cell densities and compute plans using more refined grid maps for more cluttered spaces.
Figure 7.2: Different occupancy grids: images scaled to show relevant detail.
7.1.3 Probability and Certainty Parameters

We also experimented with changes to the occupancy update parameters, $\alpha$ and $\beta$. While these are all continuous-valued variables that can take on values in the range $[0,1]$ it is not necessary to test the entire range; intuitively we know that the algorithm will function poorly over a substantial portion of this parameter space. For example, $\alpha$ and $\beta$ represent the maximum and minimum amount of occupancy respectively that we should permit the robot's model of the world to accept for any given single observation. Since, our vision sensor is fairly accurate, the model should be fairly accepting of its declarations. Also, we want the robot to err on the safe side, it is better to have the model be more willing to increase occupancy grid values than to decrease them. However, making $\alpha$ too close to 1 will make it difficult for the robot to eliminate noisy readings. Therefore it is necessary to choose values that properly tradeoff acknowledgement and forgetfullness. Table 7.1 rates qualitatively the robot's behavior based on the number of steps or control cycles spent trying to drive to the goal position, as illustrated, for some sample values of $\alpha$ and $\beta$. Good values, not listed in the table, that seemed to work well across a broad range of environment conditions were found to be $\alpha = 0.8$ and $\beta = 0.1$. Fixing these parameter values to be $\alpha = 0.8$ and $\beta = 0.1$, we experimented with different occupancy background levels, $Pr(occ(x,y))$, and obstacle certainty thresholds $occ_{\text{thresh}}$. In general, the background level should be set as low as possible for the given environment to allow for fastest possible replanning in the worst-case scenario. A good value in most cases is in the range $[0.35,0.45]$, using larger values for more cluttered spaces. For the robot task illustrated above, we found that the robot planned better and was less prone to running into things that were truly obstacles with a fairly low obstacle certainty threshold of about 0.8, and ran into many obstacles when the threshold was above 0.9.
7.2 Conclusion and Future Work

Although we had to run the algorithm with the camera head at a fixed tilt angle, we were able to garner a sufficient number of tests to demonstrate algorithm functionality. While Erik had to be unwedged from situations where its obstacle detector failed, he generally was able to generate and follow a reasonable path to the goal position after sufficient exploration of the environment. In instances where the goal was not directly in sight, Erik often generated a path toward unexplored regions to gain information. This behavior often led the robot toward areas of the map where the ground plane constraint or background texture constraint does not hold as well. The explanation for this phenomenon is simple: as the robot drives closer to obstacles in these areas, like chairs and short tables, its obstacle detector triggers on their top edges causing the robot to clear out space close to it, and integrate obstacle data at a distance further away. It may be possible to alleviate this problem by blurring obstacles in the map so that the robot will never attempt to drive too close to any of these obstacles,
but that would greatly reduce its ability to navigate in more constrained areas like narrow or cluttered hallways.

With head tilt ability, and optimally tuned parameters, it is believed the robot will be less prone to make these kinds of mistakes. Indeed, the results of the experiments conducted with the robot point toward the need to find optimal parameter tunings for different environment conditions to achieve robust room-scale navigation. One possible improvement for the system may be to integrate obstacle information taken from other image segmentation routines. We have considered adapting the system to integrate obstacle depth information using optical flow. Since different image segmentation algorithms embody different assumptions about the environment, we conjecture that it may be to the robot’s advantage to switch between different routines based on the context of the environment, or possibly integrate information from several routines running in parallel to achieve more accurate occupancy maps. This seems to be a likely next step to improving the capabilities of this navigation system in the future.

\footnote{Several MPEG movies of the robot driving around rooms of the AI lab, and the evolution of its occupancy grid and predicted path planning can be found at http://www.ai.mit.edu/people/jsg/RoomNavigation.html}
Appendix A

Calibration Code

function [S, R, t] = calibrate (D)
% This function performs constrained linear least-squares estimation
% of the intrinsic and extrinsic camera parameters as
% described in "3 Dimensional Vision" by O. Faugeras

% It takes a matrix D of Calibration Points N x 5,
% where each row is [x y z u v]

% And returns S -- the projection matrix of the standard coordinate frame
% R the rotation matrix, and t the translation offset which can be
% combined into a matrix K = [R t; 0 0 0 1], to give a single matrix
% transformation and pre-multiplied by S to yield [P p] = S*K;

npts = size(D, 1);

M_i = D(:, 1:3);
Mtilde = [ M_i, ones(npts, 1)];

u_i = D(:, 4);
v_i = D(:, 5);
\[ m_i = [u_i, v_i]; \]
\[ \mathbf{mtilde} = [m_i, \text{ones(npts, 1)}]; \]

\[
\begin{align*}
A_u &= [M_i, \text{ones(npts,1)}, \text{zeros(npts,4)}, -\text{diag}(u_i)\cdot M_i, -u_i ]; \\
A_v &= [\text{zeros(npts, 4)}, M_i, \text{ones(npts, 1)}, -\text{diag}(v_i)\cdot M_i, -v_i];
\end{align*}
\]

\[
A_u \_\text{ext} = [\text{dyadup}(A_u, 'r', 0); \text{zeros}(1, 12)]; \\
A_v \_\text{ext} = \text{dyadup}(A_v, 'r', 1); A_v \_\text{ext} = A_v \_\text{ext}(1:(2\cdot\text{npts}), :); \\
A &= A_u \_\text{ext} + A_v \_\text{ext};
\]

\% \ Aq = Cy + Dz, \min_y,z (||Aq||^2) \ s.t. ||z||^2 = 1
\[
\begin{align*}
C &= [A(:, 1:8), A(:, 12)]; \\
D &= A(:, 9:11);
\end{align*}
\]

\[
[Z, e] = \text{eigs}(D'\cdot D - D'*C*\text{inv}(C'*C)*C'*D);
\]
\% Eigenvector associated with smallest eigenvalue will minimize
\[z = Z(:,3);\]
\[y = -\text{inv}(C'*C)*C'*D*z;\]

\[
\begin{align*}
\mathbf{Ptilde} &= [(y(1:4))', (y(5:8))'; z' y(9)]; \\
Q1 &= (\mathbf{Ptilde} (1,:))'; q1 &= Q1 (1:3); q14 = Q1(4); \\
Q2 &= (\mathbf{Ptilde} (2,:))'; q2 &= Q2 (1:3); q24 = Q2(4); \\
Q3 &= (\mathbf{Ptilde} (3,:))'; q3 &= Q3 (1:3); q34 = Q3(4);
\end{align*}
\]

\[
\begin{align*}
u_o &= q1'*q3; \\
v_o &= q2'*q3;
\end{align*}
\]

\[
\begin{align*}
\alpha_u &= \sqrt{q1'*q1 - u_o*u_o}; \\
\alpha_v &= \sqrt{q2'*q2 - v_o*v_o};
\end{align*}
\]
\[ S = [\alpha_u 0 u_o 0; 0 \alpha_v v_o 0; 0 0 1 0]; \]

\[ r1 = (q1' - u_o*q3') / \alpha_u; \]
\[ r2 = (q2' - v_o*q3') / \alpha_v; \]
\[ r3 = q3'; \]
\[ R = [r1; r2; r3]; \]

\[ tz = q34; \]
\[ tx = (q14 - u_o*tz) / \alpha_v; \]
\[ ty = (q24 - v_o*tz) / \alpha_v; \]

\[ t = [tx; ty; tz]; \]

\% Calculate MSE and Max and Min Errors
\[ m = (Ptilde*Mtilde')'; m = diag (1.0 ./ m(:, 3))*m; \]
\[ e = m - mtilde; \]
\[ MSE = e'*e \]
\[ MaxError = max(e, [], 1) \]
\[ MinError = min(e, [], 1) \]
Bibliography


