MULTIROBOT TETHERING FOR LOCALIZATION AND CONTROL

BY

BRAD A. BAILLIO

B.S., Brigham Young University, 2006

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2009

Urbana, Illinois

Adviser:

Professor Geir E. Dullerud

ABSTRACT

Particle filtering has proven to be an effective localization method for wheeled autonomous vehicles. For a given map, a sensor model, and observations, occasions arise where the vehicle could equally likely be in many locations of the map. Because particle filtering algorithms may generate low confidence pose estimates under these conditions, more robust localization strategies are required to produce reliable pose estimates. This becomes more critical if the state estimate is an integral part of system control. We investigate the use of particle filter estimation techniques on a hovercraft vehicle. The marginally stable dynamics of a hovercraft require reliable state estimates for proper stability and control. We use the Monte Carlo localization method, which implements a particle filter in a recursive state estimate algorithm. An H-infinity controller, designed to accommodate the latency inherent in our state estimation, provides stability and controllability to the hovercraft. In order to eliminate the low confidence estimates produced in certain environments, a multirobot system is designed to introduce mobile environment features. By tracking and controlling the secondary robot, we can position the mobile feature throughout the environment to ensure a high confidence estimate, thus maintaining stability in the system. A laser rangefinder is the sensor the hovercraft uses to track the secondary robot, observe the environment, and facilitate successful localization and stability in motion.

To my loving wife and daughter

ACKNOWLEDGMENTS

I would like to thank my professor for giving me the freedom to pursue these topics; my friend and collegue Vlad for always being there with advice, questions, and help; and my wife for sacrificing much so I could accomplish this goal.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

HoTDeC	Hovercraft Testbed for Decentralized Control
Lidar	Laser Rangefinder
EKF	Extended Kalman Filter
LMMSE	Linear Minimum Mean Squared Error
UKF	Unscented Kalman Filter
UT	Unscented Transform
PF	Particle Filter
MCL	Monte Carlo Localization

CHAPTER 1

INTRODUCTION

1.1 Motivation

There are many challenges associated with creating networked autonomous vehicles. A large subset of these challenges consists of strategies for providing autonomy to robotic vehicles. Two critical problems require solutions in order to endow a robotic vehicle with a certain level of autonomy. The first is localization, the second is control. Localization is the ability of a robot to ascertain its state of existence, or state. In robotics literature, this robot state is referred to as its pose. A typical robot pose consists of its relative x and y position, and orientation in an environment. In this study, we consider implementing localization strategies on a hovercraft vehicle. The hovercraft pose includes its relative position and orientation, as well as linear and angular velocities. To develop a localization strategy, we investigate probabilistic robotics, an emerging field that exploits the uncertainty inherent in robotic systems and their environments to find solutions to difficult problems. We can use probabilistic techniques, which are amply described in the literature, to solve the localization problem. One such solution is detailed in [1]. Their solution to localization incorporates global sensors which are rigidly fixed to the environment (an overhead camera system). Localization performance is good under this system, but is inherently restrictive to autonomy. The robot is limited to movement inside a lab environment. This restriction provides

motivation to develop a different localization strategy that allows free movement outside lab environments.

The problem of control consists of making sure the robotic vehicle maintains stability during movement along trajectories. The solution to this problem involves designing an appropriate controller for the hovercraft. Designing for a hovercraft introduces unique problems. Because most vehicles are wheeled, they benefit from inherent stability in motion. A hovercraft does not share this benefit. A hovercraft is a marginally stable vehicle. While floating, the hovercraft is highly susceptible to disturbances in the environment. It cannot maintain a set location without actuation, much less travel along a set trajectory. In order to provide control to movement (and a measure of stability), an accurate estimate of the pose is needed. Thus, the motivating question behind this study: Can we synthesize a controller around a robust localization method that allows a hovercraft vehicle to autonomously navigate environments that are not globally observed? We find that we can autonomously navigate these environments.

1.2 Contribution

In this thesis, several of the potential solutions for state estimation are discussed, including the underlying theory behind recursive state estimation. These potential solutions include the Kalman filter, unscented Kalman filter, and the particle filter. After a careful review of the benefits and limitations of each solution, a particle filter method is chosen for our system. The particle filter provides the right amount of accuracy, is robust to differing environments, and solves the localization problems associated with our hovercraft. Among these

localization problems are the global localization problem, and the kidnapped robot problem. The particle filter, as a localization strategy, is a Monte Carlo method. The MCL algorithm provides a means of predicting possible locations of the hovercraft, and judging the likelihood of each using observation data. The hovercraft uses range and bearing measurements of the environment, provided by a laser, for its observations. Due to the limitations of the hovercraft and environment, many design choices/tradeoffs are made in the implementation specifics of the localization strategy. Among these tradeoffs are the use of one thousand particles to represent the probabilistic statistics of the estimate, the use of a random walk as a system model of motion, and a biased sensor model.

From the different approaches to control, an H-infinity approach suits the hovercraft because of the inherent marginal stability. An H-infinity controller can minimize system perturbations. Since the hovercraft, while floating, is susceptible to such perturbations, an H-infinity controller is chosen and synthesized. The controller mitigates the susceptability of the hovercraft to instability. The total delay of the system, including data aquisition and processing time, is considered during synthesis of the controller. Simulation and modeling of the synthesized controller in MATLAB assure satisfactory stability of the system. After simulation, the controller is transplanted to the hovercraft. Utilizing the H-infinity controller and the particle filter, stable trajectory movement is achieved in the lab setting.

After testing autonomous movement along trajectories, and verifying stability in motion, the hovercraft is removed from the lab environment. The lab environment is feature rich. Features such as corners, walls, cabinets, and desks, create a unique map from which it is easy to localize. A hallway, where

subsequent test are conducted, is feature poor. The few features are lines representing the long hallway walls. Without many distinct features, the map is not unique. Moreover, different locations in the map become indistinguishable. This feature-poor hallway presents difficulty in producing a single hypothesis for the state estimate. A state estimate with high variance, or an estimate that results from multiple state hypotheses, leads to major instabilities in the hovercraft. To solve this problem, a multirobot network is established. The hovercraft acts as an overseer and worker to autonomously navigate both itself and a helper robot, the herdbot, down a hallway. The herdbot plays the role of mobile feature. Because the robots leapfrog their movement, we can avoid the complications of dynamically changing maps, and benefit from a recognizable, reliable feature to guarantee a state estimate with low variance. The final system solution is shown in Figure 1.1.

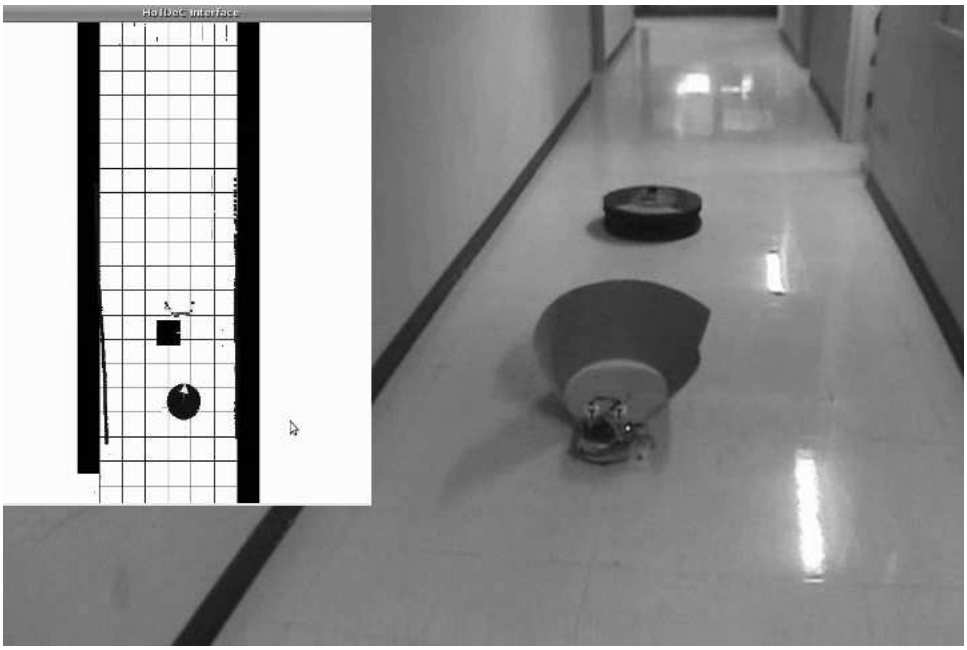

Figure 1.1: Multirobot Tethering Demonstration

1.3 Related Work

Most of the ideas related to probabilistic robotics presented in this paper are gleaned from [2]. Work with the hovercraft vehicles was established in [1] and [3] with the creation of HoTDeC.

1.4 Organization of Thesis

Chapter 2 discusses the testbed used for this study. It also describes the hardware and software architecture specifics. Chapter 3 discusses the localization problem and introduces recursive state estimation and the particle filter. Chapter 4 talks about the specific implementation details of the particle filter. Chapter 5 addresses the control problem. It discusses the design methodology and synthesis of an H-infinity controller. Chapter 6 shows how to use a secondary robotic vehicle to maintain stability in unique environments. Conclusions and future work are discussed in Chapters 7 and 8, respectively.

CHAPTER 2

HARDWARE/SOFTWARE ARCHITECTURE

The testbed is split into two parts, since there are two robotic vehicles. Each vehicle has specific software and hardware components that interact to create the multirobot tethering autonomous system.

2.1 Hovercraft

2.1.1 Hardware

Figure 2.1 shows our hovercraft vehicle. The hovercraft has a circular foam-mold body and a slightly inflatable rubber skirt. Inside are five thrusters, computing resources in the form of PC104 boards, batteries and a battery regulation system. It has about a 0.46 m (1.5 ft) diameter and weighs 2.84 kg. Four thrusters are positioned as seen in Figure 2.2. These thrusters provide movement in the x and y directions. They can also produce a torque to change the orientation of the hovercraft. The fifth thruster is the lift fan. It fills the

Figure 2.1: Hovercraft Vehicle

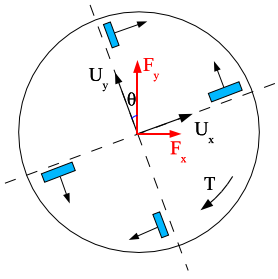

Figure 2.2: Hovercraft Thruster Configuration

skirt with a pocket of air which allows the vehicle to hover.

Figure 2.3: Lidar

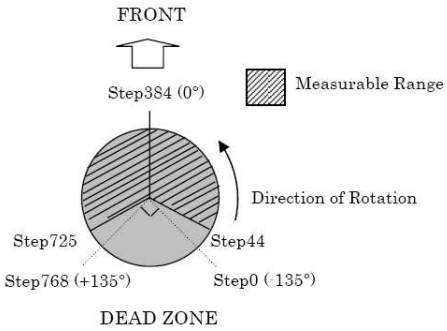

Figure 2.4: Measurable Range

The main observing sensor on the hovercraft, depicted in Figure 2.3, is a laser rangefinder, commonly called lidar. The lidar is a Hokuyo urg4lx model. It has 0.36^o angular resolution, scans a 240^o arc and services data every 100 ms. See Figure 2.4 for details. The lidar is placed on the hovercraft so that the arrow indicating the FRONT (in the figure) also points to the front of the hovercraft. This is useful in determining the vehicle orientation in its own frame of reference and with the global frame of reference. In Figure 2.5 we see that when the lidar is pointed down the global x-axis, the hovercraft has an orientation of 0^o.

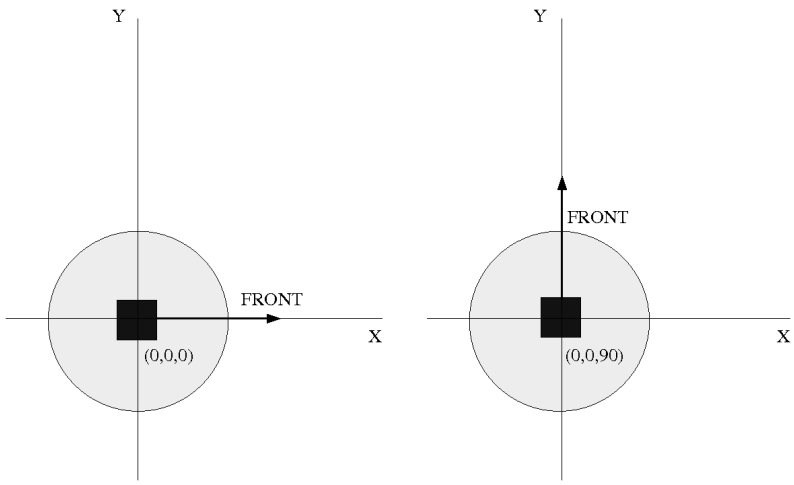

Figure 2.5: Orientation

The lidar communicates externally via the USB port, and internally via the RS232 communication protocol. The communication format is presented in Figure 2.6. The host, or hovercraft, requests data from the sensor, or lidar. The lidar responds by echoing the command and then providing the data. Because we are working over the serial port, any attempt to retrieve data, before it is all available, results in a blocking read. This can cause serious delays in the time it takes to get the lidar data and pass it on for processing. If we wait for all the data to become available, then it takes tens of microseconds to read the data. This is preferable. So, to maximize data throughput for the lidar, we send a request for data, we wait 100 ms, read all the data, then repeat the process. This gives the most recent data possible (100 ms old) to work with in our system.

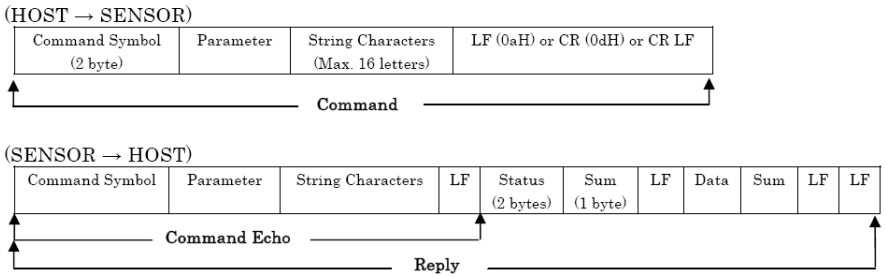

Figure 2.6: Communication Format for Lidar

The hovercraft computing power comes from a stack of PC104 expansion boards. One of them is a single board computer (SBC) with a Pentium class Transmeta Crusoe 800 MHz processor with 256 MB memory. Another contains a micro-controller unit (MCU) plus some shared memory. There is also a USB and PCMCIA board. We insert an Orinoco 802.11b PCMCIA wireless card so the hovercraft can communicate via wifi. The SBC acts as the embedded controller. It sends the thruster fan speeds to the MCU, which runs a PI controller by implementing the desired fan speeds and closing the loop by measuring real fan speeds via the Hall-effect sensors. The SBC and MCU

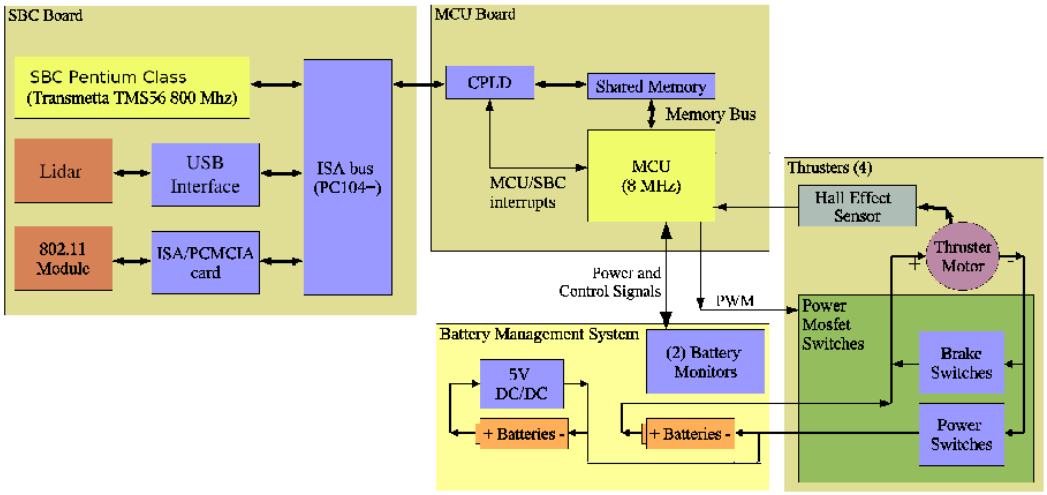

Figure 2.7: Electronic Control System

communicate via two SRAM ICs, which are treated as shared memory. Through the use of interrupts, the two processors are able to notify when data is ready to be accessed by the other. The SBC also reads the lidar data (over USB), performs all state estimation, and handles all communication (via wifi) and control of the secondary vehicle.

The electronic system setup is shown in Figure 2.7.

2.1.2 Software

The Linux kernel is the OS on the SBC. We are running the 2.6.15.3 kernel with certain real-time options enabled. We recompile the kernel to include a 1 ms system timer. This allows fast context switching, which is important for time-critical processes. We also enable the preemptible kernel. This means that if a high priority process needs to run, it can interrupt kernel tasks, which normally are uninterruptible. The final option we enable is Posix message queues. The Posix message queues are shared memory devices that are designed for real-time systems. The use of all these options is necessary since the

controller is designed around a specific time period, and control is therefore a real-time process. A host of other processes run in the background, computing the state estimate, collecting the lidar scan points, communicating with the secondary vehicle, and relaying commands and information to a human user. While these other processes are not real-time, they all share information through the message queues. The software setup is shown in Figure 2.8.

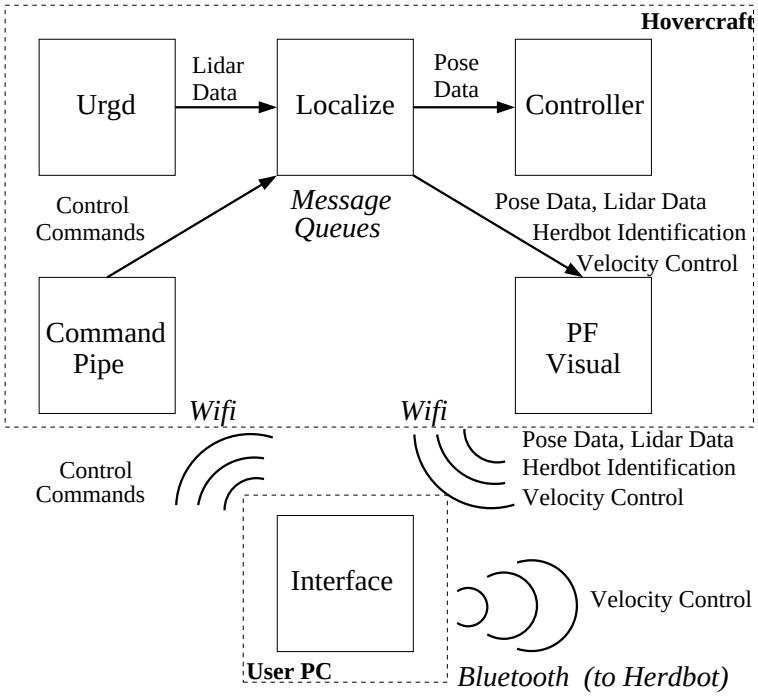

Figure 2.8: Software System on Hovercraft

Three main processes do all the important computation for localization and control: urgd, controller and localize. Two processes handle communication between a user and the hovercraft: pf visual and command pipe. The user runs a single process, called interface, to issue commands to the hovercraft, receive data for graphical viewing, and relay velocity commands to the herdbot. Urgd is the process that communicates with the lidar. It asks the lidar for data, retrieves it, packages it, and sends it on to the localize process. Controller is the

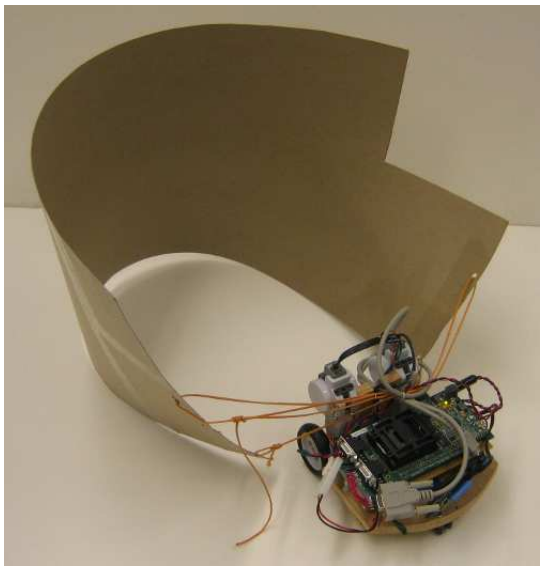

Figure 2.9: Herdbot

process that takes pose data from the localize process and calculates appropriate force values for the thrusters to keep the craft stable. Localize is the workhorse process for the hovercraft. It handles the localization effort, as well as tracking the herdbot, issuing commands, packaging data for viewing by the user, and other functions. Pf visual takes viewing data and herdbot commands from the localize process, and sends it to the user. Command pipe waits for commands from the user and pushes them on to the localize process. More of the specifics of these processes will be explained in subsequent chapters.

2.2 Herdbot

2.2.1 Hardware

The second robotic vehicle in the testbed, the three-wheeled herdbot, is shown in Figure 2.9. It has a base platform of wood. It uses two Lego Mindstorms NXT wheel motors for the differential drive, and the third wheel is a

ball-bearing caster wheel. The computing power comes from a TI DSP TMS320-F28335 development board and a gumstix Connex XM4-BT board. The gumstix board has a 400 MHz Intel XScale core (ARMv5) processor. The two main interfaces with the gumstix are the UART for communication with the DSP, and the Bluetooth module for communication with the hovercraft.

The herdbot is used in the tethering portion of the project. Its role is to help the hovercraft navigate a hallway. In order to better see the small robot with the lidar, we attach a cardboard parachute that the herdbot drags behind. Since the Mindstorms motors are not very powerful, we could not attach anything heavy or rigid to the vehicle. The parachute keeps the wheels unobstructed and is light enough to pull around without noticeable use of power.

2.2.2 Software

The gumstix processor also runs the Linux kernel, version 2.6.11. Figure 2.10 shows the software layout. Two processes handle all the communication needs: A Bluetooth daemon runs in the background, accepting Bluetooth connections, while a second process uses the open Bluetooth connections to listen for any command data being sent by the hovercraft or user. When verifiable command data is received, the second process packages it and sends it through the UART to the DSP. The DSP runs a single process that is the controller. It uses command data as well as encoder values to control the vehicle's speed. We use a speed controller to help the herdbot drive straight. Corrections are made via the hovercraft velocity control commands.

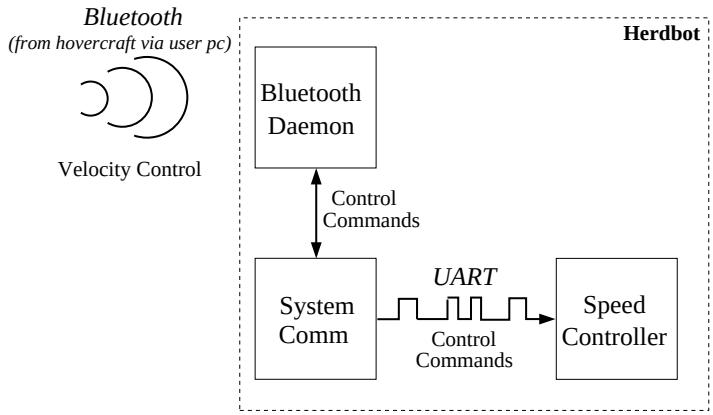

Figure 2.10: Software System on Herdbot

2.2.3 Control

The velocity controller on the herdbot is depicted in Figure 2.11. The block diagram represents a controller that is run for each of the two motorized wheels. The strategy for making the herdbot drive straight is straightforward. The left (motorized) wheel is kept at a fixed reference velocity. The right wheel starts out at the same fixed reference velocity. During the herdbot movement, the hovercraft keeps track of the error between the center of the hallway and the herdbot. The hovercraft passes the error term through a gain, adds it to the fixed reference velocity, and sends it to the herdbot as a modified right wheel reference velocity. Using this method, we can speed up or slow down one of the herdbot wheels to make minor course adjustments so its trajectory stays straight down the hallway.

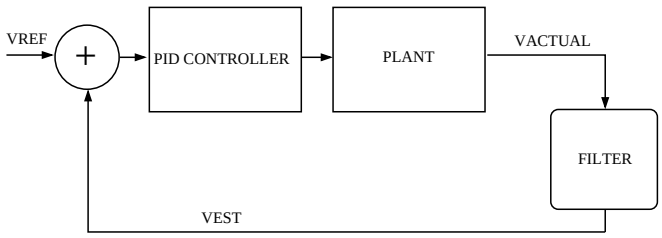

Figure 2.11: Velocity Controller on Herdbot

CHAPTER 3

LOCALIZATION

Our goal is to find a localization solution that produces an accurate pose estimate, or estimate of our x, y position and orientation, using the lidar. But before we can discuss localization, we need to understand the basics of recursive state estimation.

3.1 Recursive State Estimation

To get started, let us review a few basic ideas from probability theory.

Conditional probability

$$p(x|y) = \frac{p(x,y)}{p(y)} \tag{3.1}$$

Bayes rule

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{3.2}$$

where

$p(y) = \int (p(y|x')p(x')dx'$

Prior $:= p(x)$

Posterior $:= p(x|y)$

Conditional independence

$$p(x,y|z) = p(x|z)p(y|z) \tag{3.3}$$

Gaussian distribution

$$p(x) = det(2\pi\Sigma)^{-\frac{1}{2}} e^{\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\}} \tag{3.4}$$

We will use several other terms throughout the study of state estimation techniques. The first is the idea of a belief, $bel(x_t)$. A belief is a posterior probability distribution over the state variable, conditioned on available data. For all our experiments, a given map is part of the available data. When we take a prior and push it through a state transition equation, we get a prediction, $\overline{bel}(x_t)$. Pushing the prediction through a measurement update equation is called a correction, and results in a belief posterior. Equipped with these mathematical tools, we establish the basis of our state estimation problem: the Bayes filter.

3.2 Bayes Filter Algorithm

The Bayes filter is a direct application of the Bayes rule. We employ a recursive algorithm with two main steps: prediction and correction. The prediction step is calculated from the previous posterior and a state transition equation. The correction step calculates a belief posterior from the prediction and a measurement update equation. Figure 3.1, from [2], depicts this algorithm.

Bayes_filter$(bel(x_{t-1}), u_t, z_t)$:
for all x_t do
    $\overline{bel}(x_t) = \int p(x_t|x_{t-1}, u_t)p(x_{t-1}|z_{1:t-1}, u_{1:t-1})dx_{t-1}$
    $bel(x_t) = \eta\, p(z_t|x_t)\overline{bel}(x_t)$
endfor
return $bel(x_t)$

Figure 3.1: Bayes Filter Algorithm

All recursive state estimation problems follow this pattern. Now that we understand how all state estimation problems will be formulated (prediction from the state transition, then correction based on sensor observations), we approach the localization problem. Our first question is, what kind of localization problem are we trying to solve? Ultimately, we would like our hovercraft vehicle to start anywhere, and move around stably. This describes the global localization problem. So we concern ourselves with a localization approach utilizing unknown initial conditions (starting location and distribution). The ability to recover from serious failures, e.g., the kidnapped robot problem, would be an excellent feature as well. Our setup will involve mostly static environments, and we will be utilizing a more passive localization strategy. We will indeed use the pose estimates to ensure stability, but we will not actively try to minimize our localization error at this stage. And finally, we are concerned with localizing a single robot vehicle at the present time, with the intention to move to a multirobot system in the future.

Based on the answer to the question posed, we can narrow down our localization solution set. To satisfy our localization problem, we have available to us the extended Kalman filter, the unscented Kalman filter, and the particle filter.

3.3 Extended Kalman Filter

The extended Kalman filter (EKF) is the nonlinear extension to the Kalman filter, a robust method for estimation of linear systems with Gaussian noise characteristics. There are many ways to study the Kalman filter. The main idea for the Kalman filter is to estimate one random sequence from another, as can

be seen in Figure 3.2, from [4]. To do the estimation, we employ the linear minimum mean squared error (LMMSE) estimator. This is a method of taking the expectation of the random variable in question, conditioned on a collection of other random variables.

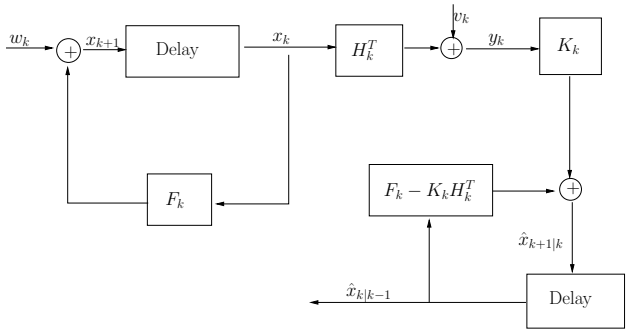

Figure 3.2: Block Diagram of Kalman Filter

In our mobile robot application, we find the expectation of the current pose, conditioned on all previous observations or measurements. However, like all state estimation problems, we can think of it as following the Bayes filter algorithm, that is, as having a prediction step and a correction step. We are simply taking the expectation of the belief. Since we are using Gaussian distributions to model our belief, we can easily represent them with the first two moments: mean and covariance. Thus, we do a prediction step by propagating the mean and covariance through the linear state update equations. The correction step employs the LMMSE estimator to produce a corrective term to the mean and covariance. The Kalman gain is formed to incorporate the measurement information to help minimize the trace of the covariance matrix. As mentioned before, the EKF replaces the linear equations for state and measurement updates with nonlinear ones. Nonlinear equations are much harder to work with, so we use Taylor series expansion to linearize these equations. We approximate around the mean, keep the first order term, and drop all the higher order terms.

The EKF is an extremely robust algorithm and a great way to do state estimation. However, there are some drawbacks to using the EKF for localization. One of these is the feature-based observation model. Another is the strict use of Gaussian distributions to represent the belief, which limits us to position tracking. We need to do better. Also, linearization using the Taylor expansion can be cumbersome in some applications, and disregards all higher order information about the belief, which can have negative effects on the estimate. This leads us to a different approach.

3.4 Unscented Kalman Filter

The unscented Kalman filter (UKF) is a variant on the EKF that addresses the problems with the EKF. The main problem with the estimation produced from the EKF is the inaccuracies in the linearization step which are caused by the Taylor series expansion approximation throwing away all higher order terms and keeping just the first term. These higher order terms can have a significant impact on the estimation in certain applications. The UKF tries to correct the inaccuracies by replacing the Taylor approximation with something else.

The first key idea to the UKF is that *it is easier to approximate a probability distribution than it is to approximate an arbitrary nonlinear function* [5]. To achieve this end, we use the unscented transform (UT), depicted in Figure 3.3, and a set of deterministic points to represent the statistics of our probability distribution as it undergoes the nonlinear transformations. These points are called sigma points, and are chosen so that their mean and covariance match the statistics of the original distribution. The nonlinear function is applied to each point, and the statistics of the transformed points can be calculated to form an

estimated distribution with appropriate mean and covariance. Because the points are deterministically chosen, they carry with them high order information about the distribution that is otherwise lost through a linearization of the nonlinear transformation.

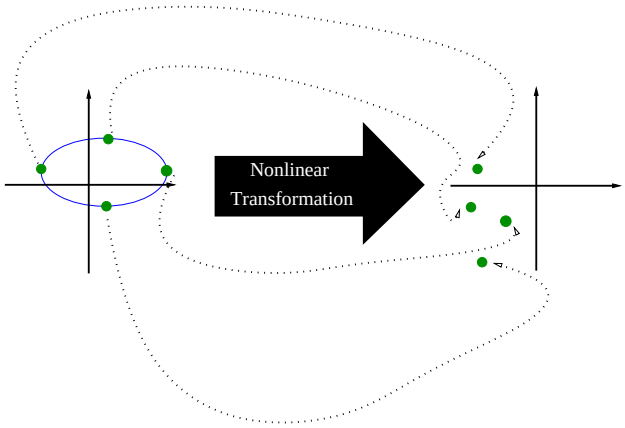

Figure 3.3: Unscented Transform

A sigma set is defined as p+1 vectors of sigma points and their associated weights, W. The weights must add to 1. The points are transformed via the nonlinear transformation. Then the estimated mean is the sum of all the products of the new sigma points and their weights. The covariance is likewise calculated. A typical symmetric set of 2n points can be calculated via the formulas shown in Figure 3.4 from [5].

We can exploit the higher order information by adding the mean of the original distribution as the first sigma point. Doing so will leave the mean unchanged, but the remaining sigma points would be scaled differently to maintain the covariance. This will create a different sigma set with the same mean and covariance, but with different high-order information. This new sigma set produces a better overall estimate of the covariance.

Sigma Set:

$$
\begin{aligned}
x^{(i)} &= \bar{x} + (\sqrt{N_x \Sigma_x})_i \\
W^{(i)} &= \frac{1}{2N_x} \\
x^{(i+N_x)} &= \bar{x} - (\sqrt{N_x \Sigma_x})_i \\
W^{(i+N_x)} &= \frac{1}{2N_x}
\end{aligned}
$$

where $(\sqrt{N_x \Sigma_x})_i$ is the ith column of the matrix square root, $\bar{x}$ is the mean of x, and Σ_x is the covariance.

Statistics:

$$
\begin{aligned}
z^{(i)} &= h[x^{(i)}] \\
\bar{z} &= \textstyle\sum_{i=0}^{p} W^{(i)} z^{(i)} \\
\Sigma_z &= \textstyle\sum_{i=0}^{p} W^{(i)} \{z^{(i)} - \bar{z}\}\{z^{(i)} - \bar{z}\}
\end{aligned}
$$

Figure 3.4: Unscented Transform Equations

Depending on the application, the UKF can outperform the EKF, providing better estimates of the mean and covariance. It does not add any more computational complexity, and can even be easier to implement because there are no difficult Jacobian matrices to calculate. The UKF solves the position tracking problem, but cannot adequately handle the global localization problem. It works best under Gaussian conditions on the initial distributions. It is also a feature-based implementation of the Bayes filter. All sensor readings are focused on picking out landmarks in the environment and matching them to the map. We can produce features from our lidar readings, but that adds more computation effort for the vehicle. We would like to use raw sensor readings. We would also like to solve global localization and be unrestricted in our distribution representation.

3.5 Particle Filter

Particle filters are one of the most robust and widely used techniques for mobile robot localization. The core of the technique is to approximate our posterior distribution by a finite number of parameters by choosing a set of random state samples drawn from the posterior distribution. The samples are called particles. The next major idea that ensures successful estimation using particle filters is the idea of importance sampling, as illustrated in Figure 3.5, taken from [2]. From Figure 3.5(a), we see that the ideal situation would be to sample particles from our target distribution, f, to obtain an accurate estimate. However, we do not have the ability to sample from this target distribution. We only have a proposal distribution, g, that we can sample from, as in Figure 3.5(b). We know that the empirical count of particles that fall into some set A converges to the integral of g under A. The strategy, then, is to weight the particles from the proposal distribution to model the target distribution. Taking our weight as $w^{[m]} = \frac{f(x^{[m]})}{g(x^{[m]})}$, we have

$$[\sum_{m=1}^{M} w^{[m]}]^{-1} \sum_{m=1}^{M} I(x^{[m]} \in A) w^{[m]} \longrightarrow \int_A f(x)dx \tag{3.5}$$

This mathematically validates the strategy, as illustrated in Figure 3.5(c).

When used in the localization problem, particle filters fit into an algorithm called Monte Carlo localization (MCL). It is a straightforward implementation of the basic particle filter algorithm, with our usual prediction/correction steps. To implement the prediction step, each particle is sent through the state transition probability equation to generate a new predicted pose sample. This new predicted pose sample is then sent through the measurement update probability equation to generate a likelihood, or weight, for the predicted pose. We now have our proposal distribution. To complete the estimation, the

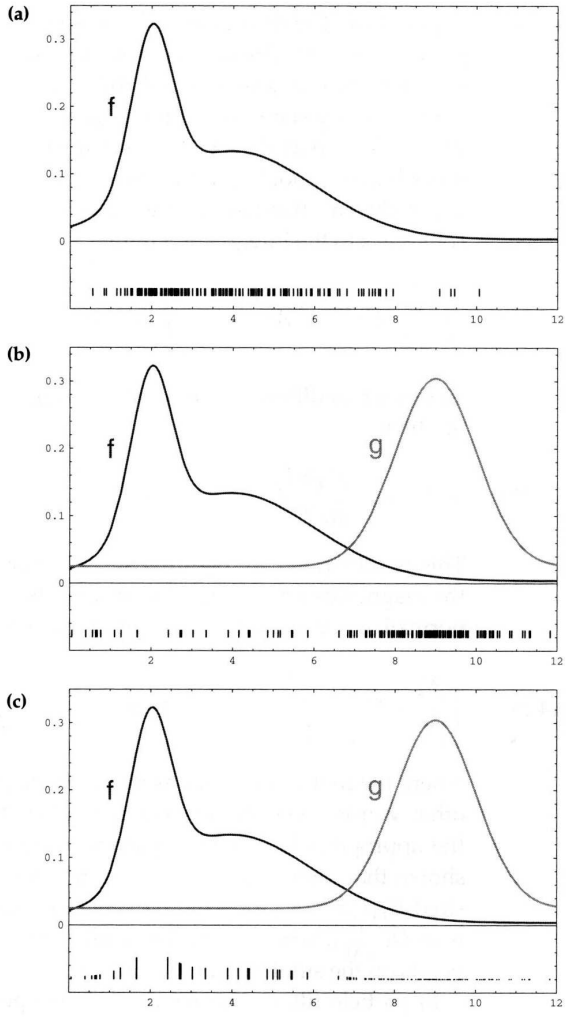

Figure 3.5: Illustration of Importance Sampling

correction is incorporated by resampling the particle set proportional to the weight of the particles.

Particle filters have many benefits we can exploit with our hovercraft vehicle: We can do global localization with particle filters, we can process raw sensor data from the lidar, we can model the measurement noise by any distribution. The MCL algorithm is easy to implement and very robust. The limitations of the particle filter are mostly computational: It requires more memory resources

and takes more time to execute. These limitations are weighty given our testbed resources; however, the benefits of the particle filter outweigh the disadvantages, which can be mitigated by several optimization variants on the MCL algorithm.

CHAPTER 4

LOCALIZATION ON THE HOVERCRAFT

4.1 Monte Carlo Algorithm

The MCL algorithm proves successful on the hovercraft vehicle. The MCL algorithm is shown in Figure 4.1, from [2], for discussion.

MCL_Algorithm$(\chi_{t-1}, u_t, z_t, map)$**:**
$\overline{\chi}_t = \chi_t = 0$
for $m = 1$ to M do
  sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$
  $w_t^{[m]} = p(z_t|x_t^{[m]}, map)$
  $\overline{\chi}_t = \overline{\chi}_t + < x_t^{[m]}, w_t^{[m]} >$
endfor
for $m = 1$ to M do
  draw i with probability $\propto w_t^{[i]}$
  add $x_t^{[i]}$ to χ_t
end for
return χ_t

Figure 4.1: Monte Carlo Localization Algorithm

4.2 Initial Conditions

One thousand particles are used to represent the posterior. The initial particle set, χ_0, is distributed uniformly over the pose space, constrained to the map, as shown in Figure 4.2.

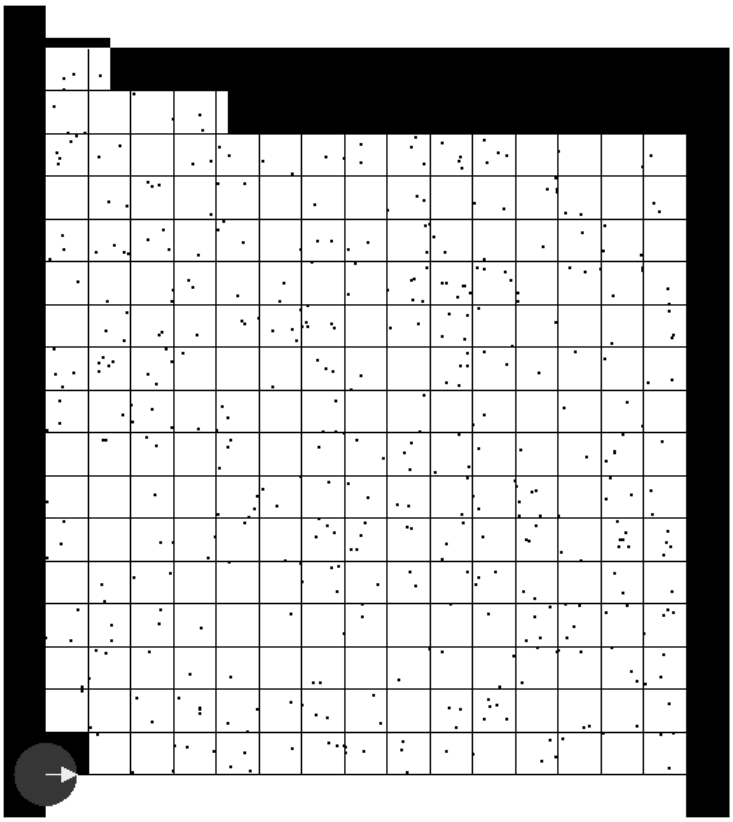

Figure 4.2: $Bel(x_0)$ is Uniformly Distributed

$$\dot{\mathrm{Y}} = \mathrm{AY} + \mathrm{Bu} + \mathrm{Gw}$$
$$\mathrm{z} = \mathrm{CY} + \mathrm{Hv}$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{\beta_x}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{\beta_y}{m} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -\frac{\beta_\theta}{J} \end{bmatrix}$$

$$Y = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \\ \theta \\ \dot{\theta} \end{bmatrix} \quad B = G = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{m} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{J} \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 4.3: Hovercraft Dynamics

4.3 Motion Model

To sample from $p(x_t|u_t, x_{t-1}^{[m]})$, we need equations that represent how the state evolves in time, given the previous state and any control actions. We can use the state equations that govern the hovercraft for the motion model. The state space representation of the hovercraft dynamics are shown in Figure 4.3, where w and v are plant and observation noise, respectively, and where m is the mass, J is the moment of inertia, and β is the friction coefficient. A more detailed discussion on how we formulate this linear model is discussed in the next chapter. By ignoring the noise and by solving the corresponding differential equations, we can get six equations that describe the evolution of the state. Three equations predict the position of the hovercraft (x, y, θ), and three

$$
\begin{aligned}
\acute{x} &= x - (\tfrac{m}{\beta_x^2})f_x + (\tfrac{m}{\beta_x})\dot{x} + ((\tfrac{m}{\beta_x^2})f_x - (\tfrac{m}{\beta_x})\dot{x})\exp\{-(\tfrac{\beta_x}{m})t\} + (\tfrac{f_x}{\beta_x})t \\
\acute{y} &= y - (\tfrac{m}{\beta_y^2})f_y + (\tfrac{m}{\beta_y})\dot{y} + ((\tfrac{m}{\beta_y^2})f_y - (\tfrac{m}{\beta_y})\dot{y})\exp\{-(\tfrac{\beta_y}{m})t\} + (\tfrac{f_y}{\beta_y})t \\
\acute{\theta} &= \theta - (\tfrac{m}{\beta_\theta^2})f_\theta + (\tfrac{m}{\beta_\theta})\dot{\theta} + ((\tfrac{m}{\beta_\theta^2})f_\theta - (\tfrac{m}{\beta_\theta})\dot{\theta})\exp\{-(\tfrac{\beta_\theta}{m})t\} + (\tfrac{f_\theta}{\beta_\theta})t \\
\acute{\dot{x}} &= (\dot{x} - (\tfrac{f_x}{\beta_x}))\exp\{-(\tfrac{\beta_x}{m})t\} + (\tfrac{f_x}{\beta_x}) \\
\acute{\dot{y}} &= (\dot{y} - (\tfrac{f_y}{\beta_y}))\exp\{-(\tfrac{\beta_y}{m})t\} + (\tfrac{f_y}{\beta_y}) \\
\acute{\dot{\theta}} &= (\dot{\theta} - (\tfrac{f_\theta}{\beta_\theta}))\exp\{-(\tfrac{\beta_\theta}{m})t\} + (\tfrac{f_\theta}{\beta_\theta})
\end{aligned}
$$

Figure 4.4: Hovercraft State Transition Equations

predict the velocity ($\dot{x}$, $\dot{y}$, $\dot{\theta}$). These are shown in Figure 4.4.

These equations require force feedback to correctly predict state updates. However, we encounter a problem with force feedback in implementation. Because the lab floor is not perfectly flat, how much force is applied to x, y and θ due to the gravity vector is dependent on the hovercraft location. Since our model does not take into account gravity, and because we cannot reliably model the effects of gravity, force feedback is detrimental to the localization effort. All predictions based on force feedback are erroneous. Consequently we set all force feedback to zero in the above equations. Doing this makes any velocity prediction unreliable. So we set all velocity equations to zero, and effectively only predict the position (x, y, θ). Our equations are now seriously crippled. To compensate for the lost effects of force feedback and velocity prediction, and to model plant disturbances, we introduce Gaussian noise with standard deviation of 0.1 m to our x and y predictions, and Gaussian noise with standard deviation 0.1 rad to θ. The final motion model update equations, from which we generate prediction samples, are shown in Figure 4.5. We have converted our motion model from the plant dynamics into a random walk. Although we have significantly reduced our ability to model the actual system, the random walk model is sufficient for successful localization. Because the hovercraft does not

$$\acute{x} = x + N(0, 0.1)$$
$$\acute{y} = y + N(0, 0.1)$$
$$\acute{\theta} = \theta + N(0, 0.1)$$

Figure 4.5: Random Walk

move very quickly, the spread of predicted particles produced from the 0.1 m noise is sufficient to capture the actual hovercraft movement.

If gravity models could be constructed and introduced into the system, it would be beneficial to stick with the original state update equations.

One final tweak is needed to ensure successful prediction using the motion model. We need to constrain any sampled particles to remain inside the bounds of the given map. In the software, we allow up to 100 attempts to sample a valid pose from the random walk model. See Figures A.9 and A.8 in Appendix A.

4.4 Sensor Model

Calculating the weights of our predictions involves exercising the sensor model. Finding $p(z_t|x_t^{[m]}, map)$ means calculating the probability of the observations, given the map and state. The first step to the sensor model is picking the appropriate observation data to process. The lidar collects 768 data points, of which a small set is used to keep computation time small. We use approximately 25 data points evenly spaced throughout the set. The motivation for this amount is given in the next chapter. Our objective in the sensor model step is to match up our actual measurements with our predicted particles and try to find the best fit. So our second step is to iterate over all the particles, using the Bresenham algorithm to match measured obstacles (z_t^k) in the environment with their actual conterparts (z_t^{k*}) in the map based on the particle state, and then

calculate a probability or likelihood of the match. The Bresenham algorithm takes the distance vector (from the hovercraft to the measurement point) and projects along that ray, in the map space, until an obstacle is reached. This new distance (from the hovercraft to the obstacle on the map) is the real distance. As is suggested in [2] and seen in Figure 4.6, we construct a probabilistic model that is a weighted combination of three probability distributions. We create a Gaussian distribution to model sensor noise. We create an exponential distribution to model unexpected objects. We create a uniform distribution to model random measurements. We combine these three distributions, and evaluate them using the real and measured distances previously calculated (z_t^{k*} and z_t^k). The result is the likelihood that the particle in question is a good estimate of the actual hovercraft state. After all predicted particles have been sent through the sensor model, the prediction step of the underlying Bayes filter is complete. For further reference, see Figures A.5, A.6, A.7 in Appendix A.

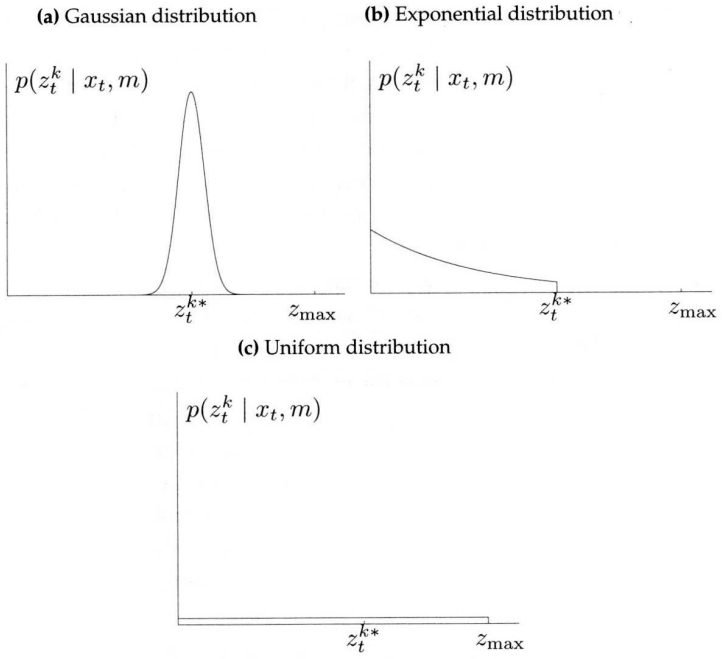

Figure 4.6: Three Probability Distributions

4.5 Resample

The resample stage is where the correction happens. We use a low-variance sampler to resample the particles. With the low-variance sampler, we avoid the pitfalls of the naive approach of resampling based on a particle's relative weight. The problem with this naive approach is that when a particle is chosen for resampling, it is randomly chosen every time. This means that potentially many bad particles can be resampled while good ones are not. The low-variance sampler fixes this approach by using only one random number in the resampling process. A single random number, r, is chosen. We add r to the inverse of the number of particles in the sample set. This new number, U, has a specific value that corresponds to a particle, i, which is chosen via the following equation:

$$i = \text{argmin}_j \sum_{m=1}^{j} w_t^{[m]} \geq U$$

Using this formulation, we continue to resample (with replacement) until we have a resampled particle set of 1000 particles.

CHAPTER 5

CONTROLLER

5.1 Plant Dynamics

The hovercraft has some interesting dynamics that make our system more challenging to work with. Because the hovercraft floats on a cushion of air contained in a rubber skirt, it can be represented by a 6 degree-of-freedom model. Previous work in the lab has developed this model [3]. We represent the system as a rigid body in the plane with translational and rotational viscous friction. This is a 3 degree-of-freedom model. The hovercraft is a holonomic system because we can achieve full actuation in all three degrees of freedom due to the actuator layout. A schematic of the four-thruster propulsion configuration is shown in Figure 2.2. The design choice of four propulsion thrusters permits the total translational force and total torque on the hovercraft to be simultaneously assigned, as desired, by appropriate choice of the four thrust values. This conversion from force to thruster velocities allows us to model the system with a linear model. The resulting model is depicted in Figure 5.1 and gives us the state equations of motion in Figure 4.3.

As seen in the state equations, we can only observe part of the state. To close the loop, we want to design a controller that can take the pose information from the state estimation algorithm and calculate forces to push the hovercraft into a stable state. For this discussion, a stable state consists of both maintaining a

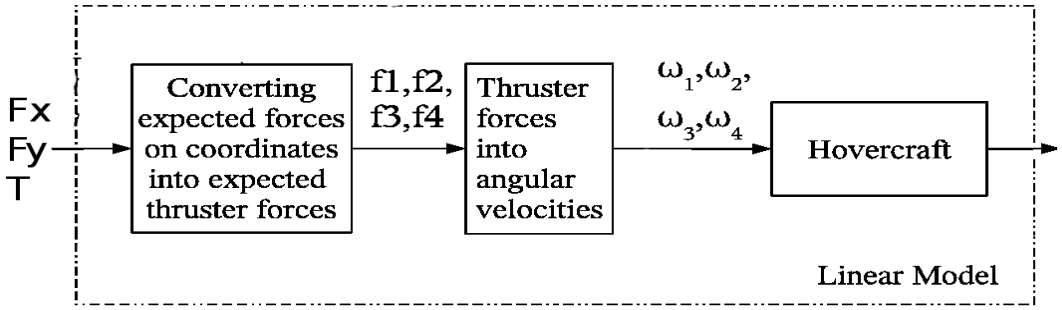

Figure 5.1: Linear Model

waypoint location without oscillations, and stability along a trajectory. We next investigate what type of controller best suits our testbed.

5.2 Controller

H-infinity control methods differ from conventional methods in their approach to stabilization. An H-infinity controller has two main goals: internally stabilize the system, and minimize the maximum gain of the system. Referring to Figure 5.2, we wish to create a controller K that maintains closed-loop stability. It also must minimize the worst-case effects of w on z. In other words, H-infinity

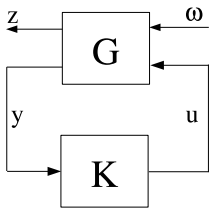

Figure 5.2: H-Infinity Controller Diagram

control minimizes the influence of perturbations on the closed-loop stability. This fits our testbed because we have a robotic vehicle that is highly susceptible to perturbations. Since there is no friction to hinder effects of system

disturbances, any amount of noise in the observations or perturbations from the environment can seriously affect system stability. We construct an H-infinity controller to mitigate all such negative effects. Our closed loop model is represented in Figure 5.3. To design the right controller dynamics, we need to study how the closed-loop model will be implemented and consider the timing characteristics of the system.

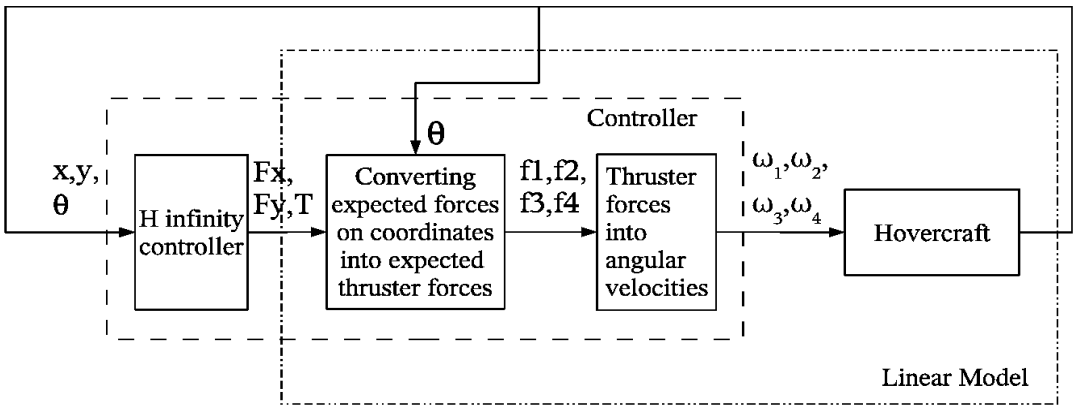

Figure 5.3: Closed Loop Model

If we take a look at the delays in the total system setup, we can see how our controller needs to be designed. If we study Figure 5.4, we see the closed-loop system is broken down into four main sections and a small communication delay. The first delay is introduced by the lidar. It has an operating frequency of 100 ms. This is the fastest it can scan, collect data, and deliver it to the SBC. The next big delay comes from the localization process. Running the particle filter is computationally expensive, as discussed in the previous chapter. The main bottleneck for computation comes in the measurement update, in which all 1000 particles have to be compared against a certain number of scanpoints. If we try and use all 768 points available, the delay can be orders of magnitude higher than 90 ms. For this reason, we dynamically adjust the number of

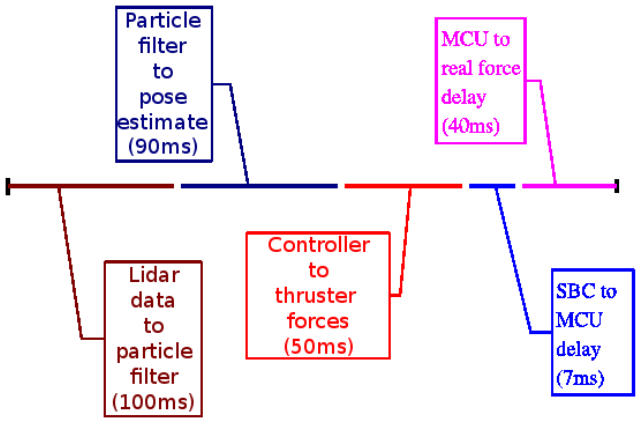

Figure 5.4: Total Closed-Loop System Delay

scanpoints used depending on the environment and the timing delay of the algorithm to stay close to 90 ms. The next delay is introduced in the controller. It has been designed to run at intervals of 50 ms, the time it needs to process the pose estimate and produce thruster forces. The last part of the delay owes to communication from the SBC to the MCU, and to the MCU actually delivering the new force commands to the thrusters. This gives an approximate total delay of 250 ms. As stated, through experimentation, a 50 ms sampling interval is chosen for the controller. Because the closed-loop delay is 250 ms, we can easily segment it into five 50-ms portions, or a five-step system delay. After discretization of the plant in 50 ms intervals, the closed-loop system is augmented to consider the five-step delay, as shown in Figure 5.5.

5.3 Simulation

Using the above formulation, a Simulink environment was created to model the hovercraft so we could design and synthesize the best H-infinity controller. See Figure 5.6. Because our pose variables are uncorrelated, we can decouple them in our controller design. Our final design has the following parameters (see Figure

$$x(t+1) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t-5)$$

and the state space form

$$\begin{bmatrix} x(t+1) \\ x(t) \\ x(t-1) \\ x(t-2) \\ x(t-3) \\ x(t-4) \end{bmatrix} = \begin{bmatrix} A & 0 & 0 & 0 & 0 & 0 \\ I & 0 & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ x(t-1) \\ x(t-2) \\ x(t-3) \\ x(t-4) \\ x(t-5) \end{bmatrix} + \begin{bmatrix} B \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & C \end{bmatrix} \begin{bmatrix} x(t) \\ x(t-1) \\ x(t-2) \\ x(t-3) \\ x(t-4) \\ x(t-5) \end{bmatrix}$$

Figure 5.5: System Delay Equation

5.2): We penalize the w signal by 2.5 for the x, y and θ variables. We penalize the control effort, u, by 0.0045 for x and y. We penalize the control effort, u, by 3 for θ. With this controller in place, we see stable motion from the hovercraft both in tracking waypoints and in trajectories. See Figures 5.7 and 5.8 for the performance of the hovercraft moving in a box shape in the lab environment.

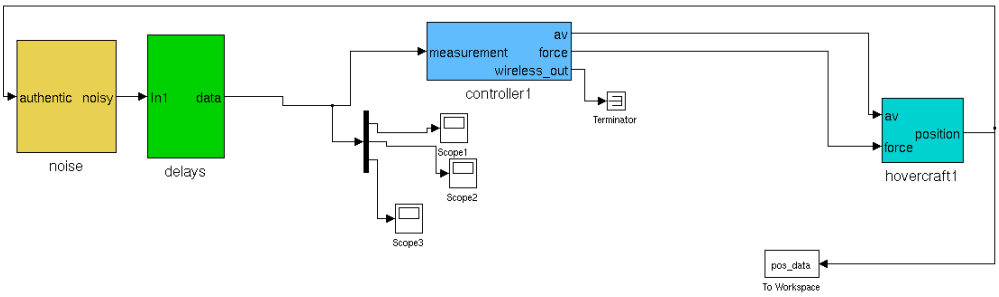

Figure 5.6: Simulink Closed-Loop System Model

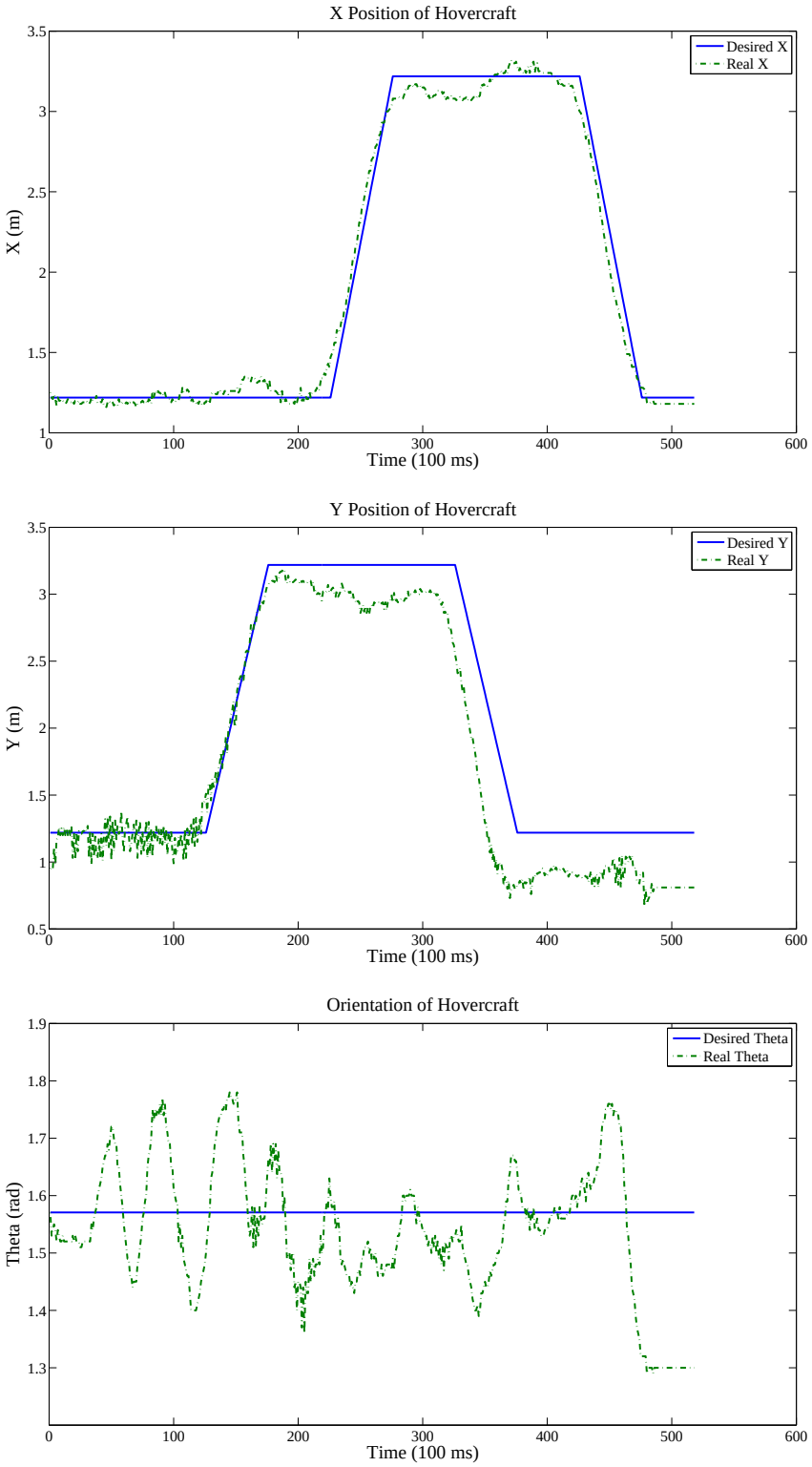

Figure 5.7: Test Results on Hovercraft

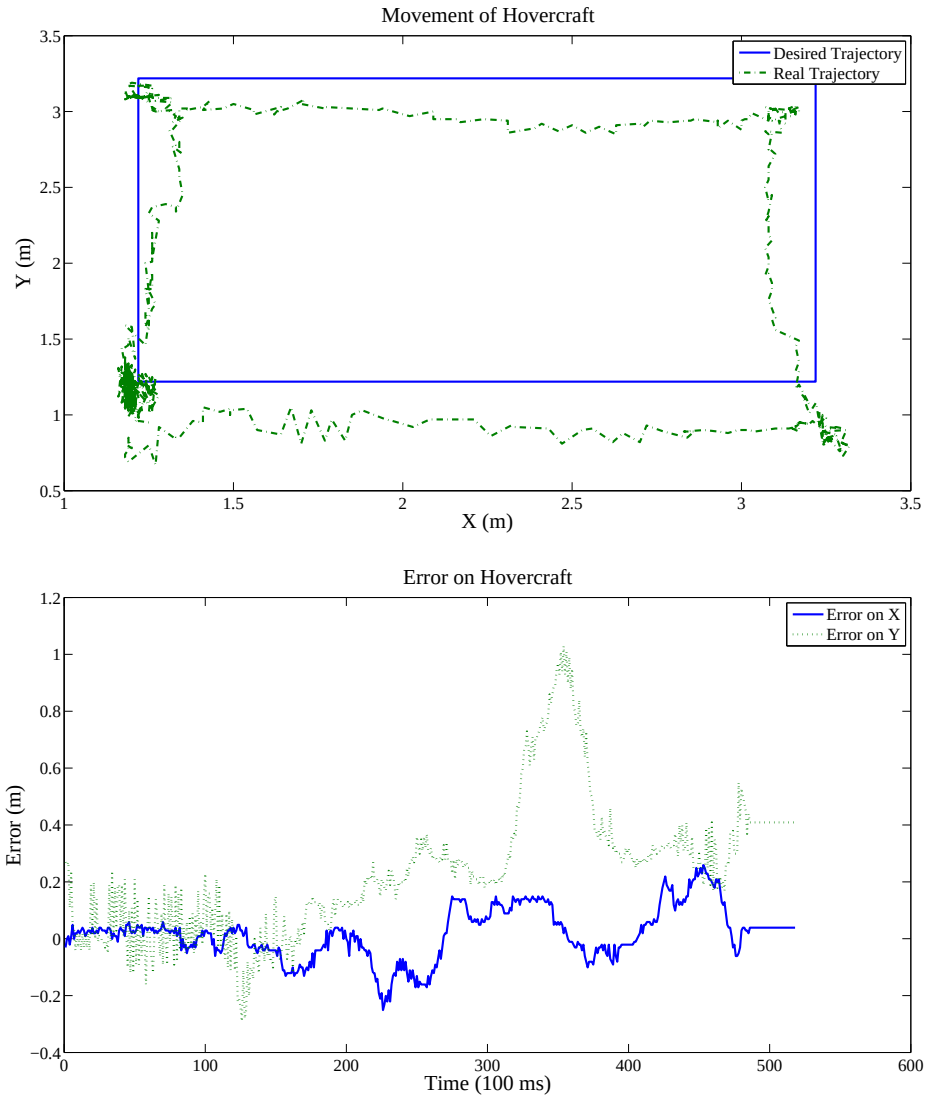

Figure 5.8: Test Results on Hovercraft

CHAPTER 6

TETHERING

6.1 Motivation

Equipped with the particle filter state estimator and the H-infinity controller, the hovercraft can navigate the lab environment smoothly and stably. When attempts are made to navigate through a hallway, the results are not what we expect. Because hallways are homogeneous, the particle filter produced multiple state hypotheses. From the perspective of the hovercraft, a hallway looks like two long, straight lines. There is little variation to make one location of the hallway seem different from another. Figure 6.1 illustrates the homogeneity of the hallway. The hovercraft cannot distinguish its pose in panel (a) from its pose in panel (b). This uncertainty results in multiple state hypotheses. An unstable

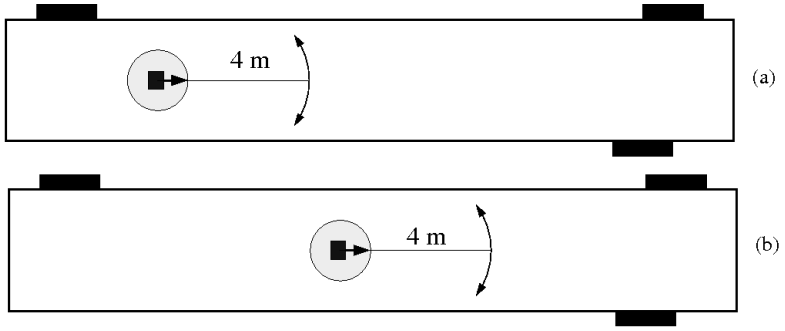

Figure 6.1: Hallway Homogeneity

estimate of the state leads to instability of the system as a whole, so the hovercraft loses control in this environment. In order to correct this shortcoming, a secondary robotic vehicle, the herdbot, was introduced into the system.

6.2 Hypothesis

If the hovercraft vehicle could recognize a distinct feature in the otherwise homogeneous environment, the state estimation would produce a single hypothesis. Since the hovercraft vehicle's sensing capabilities are limited to a 240^o arc at 4 m, the feature would need to be mobile. Our mobile feature is the herdbot. It is a small, wheeled robot whose stability and control are certain. Although easily controllable, there is still a problem with using the herdbot: A mobile feature conflicts with the static environment assumed by the particle filter. To overcome this problem, the hovercraft and herdbot leapfrog their movements. This leapfrog action is the crux of the multirobot tethering system.

6.3 Multirobot Tethering

6.3.1 Overview

The hovercraft operates in two modes: Overseer and Worker, depicted in Figure 6.2. In the Overseer mode, the hovercraft is at rest at a known location. From this stable state, it can track the changes in the environment, as well as issue commands to the herdbot. It communicates via Bluetooth to the herbot and issues velocity commands to move the herdbot forward a specified distance. Once the herdbot reaches its destination, the hovercraft commands it to stop moving, records the destination location, and modifies the map to reflect the change. At this point, the hovercraft is ready to switch into Worker mode.

In Worker mode, the hovercraft performs all the calculations to localize and move. This is the mobile autonomous phase for the hovercraft. It lifts off, moves toward the herdbot, settles into a stable location, and lands. It is during this

mode that the localization algorithms are exercised and the H-infinity controller does its job to keep the craft stable during movement. Once the hovercraft is at rest at a known location, it switches back to the Overseer mode, and the whole process starts again.

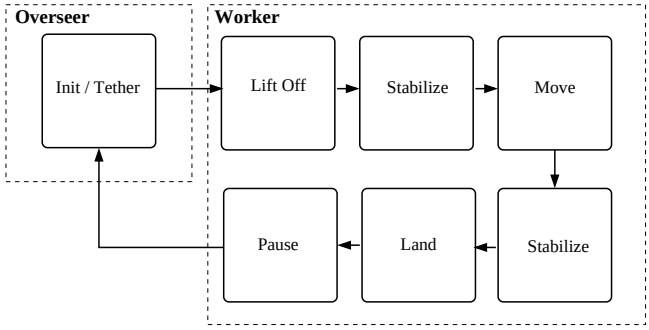

Figure 6.2: Two Modes

6.3.2 Worker

The first piece of Worker mode is tracking the herdbot. Tracking the herdbot is key to successfully navigating the hallway. In order to recognize the herdbot, we need to do some feature extraction from the lidar scan data. We utilize an algorithm called Split-and-Merge from [6]. To implement the split-and-merge algorithm, we pass through the scan data point-by-point and mark the points we consider discontinuous points. The discontinuous points represent more than 55 mm of space between scan points. Once we create a set of discontinuous points, we connect them into line segments. This is where the split portion of the split-and-merge algorithm begins. If a newly connected line segment curves or deviates more than 55 mm, then we break up the line segment in two, and try splitting the two new segments. If we do not need to split a segment, we calculate some line statistics and add it to our set of line segments. We do not worry about merging collinear segments as the algorithm suggests. The

implemented split-and-merge algorithm is shown in Figures A.1 and A.2 in Appendix A.

Once we have a set of line segments, we know that the herdbot corresponds to one of them. So, we take the last known location of the herdbot and compare it to the location of each line segment. The line segment that is closest is marked as being the herdbot. This is important for our MCL algorithm because we are going to bias our sensor model around the herdbot. When we do the Bresenham calculations and construct likelihoods, we are going to take more scan data from the line segment that represents the herdbot, than from the surrounding environment. We do this because the herdbot is the only distinct feature available to us. If we bias our sensor model around it, we are more likely to get higher weights for the particles closest to the true pose. Then during the resample stage of the MCL, we will propogate the correct particles. We are doing a mixture of scan-based localization and feature-based localization. Our leapfrog approach encourages this hybridization. See Figure 6.3 for reference.

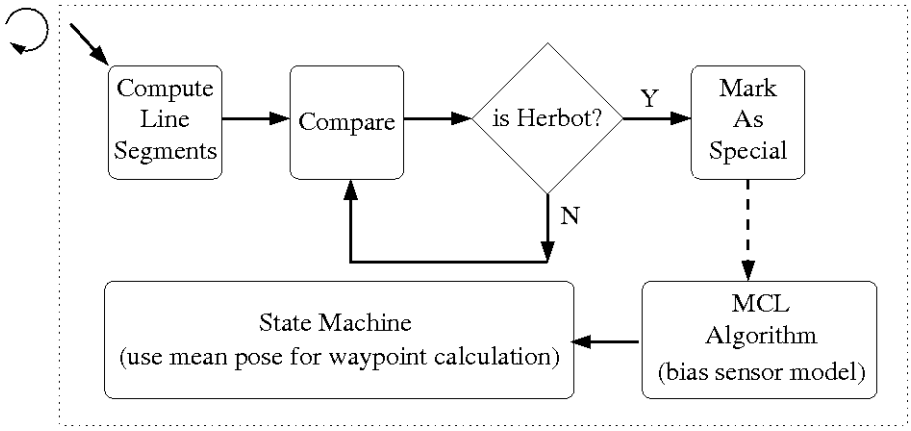

Figure 6.3: Software Flow for Worker Mode

After we find the herdbot and run the MCL algorithm, we take the mean of the resampled particle set to be the estimated pose. A state machine runs in Worker mode as shown in Figure 6.2. The state machine uses the mean pose to calculate the appropriate waypoints for each stage of movement. The mean pose is also syphoned off to the controller, which works to maintain stability at the waypoints and along trajectories.

As a fail-safe mechanism, we have designed a kill switch that will shut off the lift fans, effectively grounding the hovercraft. The kill switch activates if at any time the variance in the particle set gets sufficiently high. As stated above, when the variance is high—which can correspond to multiple hypotheses, among other things—the hovercraft destabilizes. The kill switch is programmed mostly for the safety of the hovercraft. When such an event occurs, it requires user interference to restart the vehicle autonomy.

6.3.3 Overseer

The Overseer mode is simpler than the Worker mode. The tasks switch from self-localization to managing the herdbot. The first task is to send movement commands to the herdbot. The hovercraft tells the herdbot to start moving forward, gives course corrections as discussed in Chapter 2, and tells the herdbot to stop. The other task is to track and update the herdbot location as it moves down the hall. We use the same line detection algorithms to locate the line segment corresponding to the herdbot. The only change is this: we update the last known location of the herdbot with the location of the corresponding line segment. We continually do this update until we tell the herdbot to stop. After issuing a stop command, and before switching back to Worker mode (where localization occurs), we need to redraw the static map. We erase the previous

position of the herdbot and insert the new location. In this way, the localization assumption of static maps is not violated because we change the map when localization does not occur. This is the crux of the leapfrog approach and is critical to successful localization and control of the hovercraft in the homogeneous environment. See Figure 6.4 for reference.

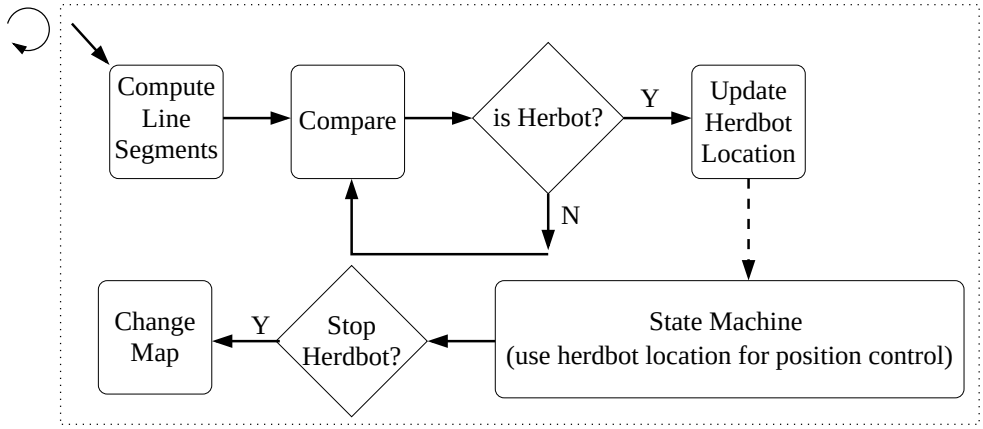

Figure 6.4: Software Flow for Overseer Mode

CHAPTER 7

CONCLUSIONS

We have shown that inherently unstable systems (hovercraft vehicles) can indeed achieve stability using state estimation techniques in conjunction with specially designed controllers. We find that the control of the hovercraft is largely dependent on a reliable state estimate. However, state estimation techniques have their shortcomings, placing our system in danger. Homogeneous environments are one of these shortcomings. Adding a secondary, tethered robotic vehicle to the system overcomes this problem and provides constant stability and control through reliable localization.

CHAPTER 8

FUTURE WORK

We had trouble with the batteries on the hovercraft. Previous work designed the battery management to kill power if the hovercraft attempted to draw too much current from the batteries. This made control a difficult problem to solve. The controllers, due to system delay, necessarily asked for powerful thruster velocities. Too frequent use caused the power to shut off. The battery management system or the batteries can be updated for the new system. Also, different controllers can be synthesized to better accomodate the battery shortcomings.

Instead of a wheeled robot for the secondary vehicle, another hovercraft can be used. This increases the challenge of maintaining control and stability. Also, the whole system can be augmented with increased autonomy. Currently, only small sections of hallway or lab space have been successfully navigated. Moving throughout a building, around corners, and into and out of rooms would add an extra layer of exploration, autonomy, and control to the system.

APPENDIX A

PSEUDO-CODE FUNCTIONS

All the following Figures A.1-A.9 are pseudo-code descriptions of important software functions for the multirobot tethering system.

get_track_points():
    Detect discontinuous points in the lidar scanpoints.
        (If 2 consecutive points are more than 55mm apart)
    Create the discontinuous points and calculate the point statistics.
    Call get_line_segs()
return

Figure A.1: Populate Set of Discontinuous Points

get_line_segs():
    Take the two discontinuous points and make a line.
    Calculate if the line curves/deviates more than 55mm.
        If it does, split the line up and call get_line_segs.
    Create a line segment and calculate the line statistics.
return

Figure A.2: Populate Set of Line Segments

find_herdbot():
    Search all line segments for the one which is closest to last known
        location of the herdbot.
    Mark the corresponding line segment as special.
    If in the tethering mode, update location of herdbot.
return

Figure A.3: Find Herdbot among Line Segments

control_herdbot():
    Calculate error. (Distance between herdbot and center of hallway)
    Calculate turn value. (K*error)
    Add turn value to right wheel.
return

Figure A.4: Herdbot Control Algorithm

measurement_model():
    For a certain amount of points:
        Call bresenham algorithm.
        Call probability_given_map algorithm.
    Weight of particle is result of $\prod$(probability_given_map)
return

Figure A.5: Measurement Model

bresenham():
    Create a vector from lidar to "measured" obstacle.
    Perform ray casting to find the real obstacle in the given map.
    Record this "real" distance.
return

Figure A.6: Bresenham Algorithm

probability_given_map():
    Create three probability distributions.
        A Gaussian to represent sensor noise.
        An Exponential to represent unexpected obstacles.
        A Uniform to represent environment white noise.
    Evaluate these distributions using the "real" and "measured" distances.
    Calculate a likelihood for the particle.
return

Figure A.7: Calculate Likelihood of Particle

sample_motion_model():
    Send particle through state equations, with noise.
    Only estimate the x,y,theta variables of the pose.
        (Force feedback is unreliable because of variations in gravity vector)
        (Sufficient noise takes care of $\dot{x}, \dot{y}, \dot{theta}$)
return

Figure A.8: Sample from Motion Model

motion_model():
    Call sample_motion_model.
    If new particle is outside map bounds, try again (up to 100x).
return

Figure A.9: Motion Model

REFERENCES

[1] Z. Di, "Control with structural constraints: Theory and implementation," Ph.D. dissertation, University of Illinois at Urbana-Champaign, Urbana, IL, 2007.

[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[3] A. Stubbs, "Estimation and control of autonomous vehicles over networks," Ph.D. dissertation, University of Illinois at Urbana-Champaign, Urbana, IL, 2005.

[4] "An exploration of random processes for engineers," class notes for ECE 534, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Winter 2008.

[5] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," in *Proceedings of the IEEE*, 2004, pp. 401–422.

[6] V. Nguyen, A. Martinelli, N. Tomatis, and R. Siegwart, "A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics," *Autonomous Robots*, vol. 23, pp. 97–111, August 2007.

AUTHOR'S BIOGRAPHY

Brad A. Baillio was born in Provo, Utah, though he lived most of his life in Texas. He began his college studies in 1999 at Brigham Young University (BYU) as a computer science major. After his freshman year, he took a two-year hiatus to perform missionary work in Spain for the Church of Jesus Christ of Latter-Day Saints. Upon returning to BYU in 2003, he continued his studies as a computer engineering major, with a focus in digital design and embedded systems. He received his B.S. degree in electrical and computer engineering from Brigham Young University in 2006.

In 2007, while attending the University of Illinois at Urbana-Champaign (UIUC), he joined Professor Geir Dullerud in the Networked Autonomous Vehicles Laboratory at UIUC where he pursued his interests in embedded systems and robotics. During his work he learned to appreciate the importance of control systems.

Brad met his wife while a senior at BYU. They married in 2005. In May of 2007, they had a beautiful baby girl. They are expecting a baby boy in April of 2009. Following the completion of his M.S., he will be working for General Dynamics Robotics Systems in Westminster, Maryland.