TOPICS IN ROBOT LOCOMOTION: STATE ESTIMATION AND FEASIBLE-PATH GENERATION FOR WHEELED AND LEGGED ROBOTS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Atif I. Chaudhry

August 2015

TOPICS IN ROBOT LOCOMOTION: STATE ESTIMATION AND FEASIBLE-PATH GENERATION FOR WHEELED AND LEGGED ROBOTS

Atif I. Chaudhry, Ph.D.

Cornell University 2015

The four areas of research discussed are all related to the common theme of robot locomotion, relating to present-state estimation, feasible-path generation, and mechanical locomotion.

BIOGRAPHICAL SKETCH

Atif I. Chaudhry attended the Brunswick School from Pre-kindergarten through 12th grade. He received two Bachelors degree, one in Aeronautical and Astronautical Engineering and another in Electrical Engineering, from the Massachusetts Institute of Technology (MIT) in May, 1997. This was followed by a Master of Science Degree in Aeronautical and Astronautical Engineering from MIT in January, 1999. After working at the Charles Stark Draper Laboratory in Cambridge, MA for a few years he enrolled in the PhD program in Mechanical and Aerospace Engineering at Cornell University.

This document is dedicated to those who have passed beyond
and to those who remain and persevere.

"After all, it's the chasing....that I really love."
- Chiyoko, Millenium Actress (Satoshi Kon, 2001)

ACKNOWLEDGEMENTS

I would like to express my gratitude to my Committee Chair, Prof. Andy Ruina, for his support, advice, guidance, and most of all his patience. I would like to thank Prof. Strogatz and Prof. Rand for serving on my doctoral committee.

I must also thank all my many fellow graduate students and staff members at Cornell, with a special focus on my labmates in Prof. Ruina's lab, Anoop Grewal, Jason Cortell, Petr Zaytsev, Matt Kelly, and Matt Sheen. Chapter two of this dissertation would not have been possible without the suggestions and input of Anoop Grewal. I am grateful to Marcia Sawyer, Graduate Field Administrator for M&AE, for her years of advice and support.

The research presented in chapter 4 is based upon work supported by the DARPA Advanced Research Projects, Information Exploitation Office (DARPA/IXO) and the United States Air Force Research Laboratory under Contract No. F33615-01-C-3149. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA or the United States Air Force. My helpful collaborator with chapter 4 his was primarily Kathy Misovec.

This research presented in chapter 5 was supported by NASA Langley through a Graduate Studies Research Program (GSRP) fellowship. I thank Tomas Kalmar-Nagy for inviting me to a CDC 2006 Control and Decision conference.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Overview

This dissertation covers four topics, one per chapter, related to robot locomotion with a focus on state estimation, path planning, or path following. This research was conducted at different times during my non-contiguous presence at Cornell University.

Cornell Ranger

The first two chapters (2 & 3) discuss research done at the Cornell Biorobotics and Locomotion Lab, headed by Prof. Andy Ruina. Both topics involve working with the lab's latest walking robot, Cornell Ranger. A description of Ranger, including photos, can be found in the beginning of chapter 2. The research done with Ranger combined theory, simulation, and real world engineering challenges.

Heel-strike Estimation

Chapter 2 focuses on one aspect of Ranger's onboard state estimator. Efficient walking depends on many well coordinated movements. The Biorobotics and Locomotion Lab's research indicates that, for robots with actuated feet and capable of rotating around, that the timing of the ankle to push off against the

ground is crucial. This is especially true for the robot Ranger. For the most efficient walking, the stance foot (the one on the ground) pushes off just before the swing foot (previously in the air) impacts the ground. This impact is referred to as heel-strike.

Humans do not have direct sensors for how high their feet are above the ground, we do not have sonars or laser range finders in our feet. By learning to walk as babies, we internalized models which allow us to push off at the right time to walk efficiently, even with our eyes closed. To match this ability, Ranger must have an internal state estimator for the timing of heel-strike. Ranger's present heel-strike estimator is based on sensors and kinematics alone. Engineering intuition suggests that using some kind of dynamic model for Ranger could only help its estimator. Such a new, model-based, estimator is proposed and demonstrated through Matlab simulation in chapter 2.

Self Steering

Chapter 3 describes Ranger's self-steering, which I, with help from others, implemented. Cornell Ranger was designed to coordinate and control its walking motion by itself, however it required a human operator to control steering. One of the challenges of Ranger's world record setting marathon walk was having someone constantly steer Ranger. There had been some early work on making Ranger self steering but other issues took precedence and self-steering was not used on Ranger's marathon walk. After the marathon, the lab was invited to bring Ranger to a robotics exhibition, Robots on Tour, in Switzerland (Spring 2013). This gave us a reason to reinvestigate giving Ranger self-steering capa-

bilities.

Chapter 3 describes the design, testing, and implementation approach for a low power, low weight, line-following system. The center-piece of the system is an off-the-shelf camera board running fast and efficient custom video firmware (software). For the line-following system to work well, the camera software needs to reliably detect a line. Chapter 3 explains, in detail, the process of selecting a tape which can be easily applied to a floor, and then how to detect it with the selected camera board. The final piece of the line-following system is the algorithm to generate steering commands based on the camera board's line detection output. The entire system was tested in Ithaca before taking Ranger to Robots on Tour in Zurich, Switzerland in the spring of 2013.

Path Planning for a UAV

The rest of my dissertation covers work done before I joined the Biorobotics and Locomotion lab when I was nominally-advised by Raff D'Andrea, working Kathy Misovec. Chapter 4 discusses research into path planning for unmanned air vehicles (UAV's) and was presented at the 43rd IEEE Conference on Decision and Control. The work was done in coordination with Alphatech, now part of BAE Systems.

The researchers at Alphatech formulated a problem related to the detection of an unmanned air vehicle (UAV) by radar, which is dependent on many variables including range, altitude, and relative orientation. Given a radar location and appropriate model for the likelihood of detection, a path plan can be created for an unmanned air vehicle which constrains the probability of detection.

I worked on a path-planning approach that involved using a linearized detection model. The detection model and the UAV's dynamics are represented as a linear program subject to mixed integer constraints. This mixed integer linear program was solved with commercial software, traditionally used by the Operations Research community. This approach searches for all feasible solutions and produces the best path plan based on the user specified parameters.

Fast Path Planning

Early in my time at Cornell, I was inspired by the fact that humans learn by repetition and by using past experiences. Chapter 5 covers one approach to seeing if it is possible for robots to act in a similar fashion. I found that by representing past path traversal experiences with matrices, a new path could be generated without relying on calculations of complex dynamics or control laws. A paper was written and presented at the 45th IEEE Conference on Decision and Control, which discussed this approach for using past experience to generate new paths and control actions. The method relies on using several matrices to associate each new input value with previous robot states. An example is provided and analyzed which shows a successful simulated implementation. Using infrastructure developed for Cornell's RocoCup team, a real world test of the approach was conducted. It demonstrated that the implementation not only worked, but was extremely fast even with limited computational power.

CHAPTER 2

MODEL-BASED STATE ESTIMATION OF A 2-D BIPEDAL ROBOT

2.1 Abstract

The motivation for the research in this chapter was to answer this question: *Is using combined dynamics and kinematic models for state estimation for a walking robot more accurate than just relying on just a kinematic model?* To study this question, simulation of a proxy model of the walking robot was employed. Several factors affecting the combining of a dynamic model with a kinematic model were studied. In the end it was determined that optimally combining sensor data with a dynamic-model predictor just slightly improves state estimation over just a kinematic model, assuming the statistics of environmental factors are sufficiently-accurately characterized. For real world implementation, hand tuning would be required and would only lead to small improvement over a system that is entirely based on sensors and model kinematics (with no use made of robot dynamics in the state estimation).

2.2 Background

The Biorobotics and Locomotion Lab at Cornell is trying to make better walking robots. Two key interests are energy efficiency and robustness. The robot of concern here is the Cornell Ranger, figure 2.1.

Ranger has a stiff pair of outer legs and a stiff pair of inner legs. Their paired nature makes four-legged Ranger functionally a planar biped. We'll refer to

Figure 2.1: Cornell Ranger in Barton Hall during a record setting 65 km walk [1] in 2011, before this research was begun. Note the paired outer legs and paired inner legs making Ranger effectively a planar biped robot.

each pair of legs as a single leg. The feet have a powered push off, then flip up, at the beginning of leg swing.

A key event in walking is heel-strike, when the swing leg hits the ground. In Ranger, the stance foot begins to push off from the ground shortly before the swing-leg heel-strike. This push-off has two purposes. First, energy is added to the system compensating for collisional losses. Second, pushing off just before the heel-strike (as opposed to, say, just after heel-strike) reduces the heel-strike

collisional energetic losses [2]. Experience with Ranger shows that the efficiency of walking is highly correlated with the timing of the push-off [3]. The most efficient stride on Ranger occurs about when the push off takes place within a 20 millisecond window before the heel-strike [4]. To push-off within this window, Ranger's software makes an estimate of when heel-strike is going to take place. If this estimate is too far off, the efficiency of Ranger's walk degrades. More significantly, the state of the robot during the next step is highly sensitive to the time of push off at the end of the present step. Errors in push off timing, relative to heel-strike, lead to erratic walking and possibly falling.

Presently, Ranger estimates when heel-strike is going to occur using the on-board sensors and simple filtering. Before the work presented in this chapter, the Ranger estimator only uses a kinematic model of Ranger, $\theta = \int \omega \cdot \delta t$ (ω is the sensed angular rate), and not a dynamics model (pendulum ODEs).

This chapter concerns an investigation of the benefits of using, instead of just kinematics, a dynamics model-based state estimator. Based on simulations, the possibilities and limitations of this approach are discussed. Understanding the issues related to using a model-based estimator in Ranger should not only help any further development of Ranger, but should also give insight into the utility of using dynamics-based state estimation for future walking robots.

2.3 Heel-strike State Estimation in Ranger

Ranger's present (before this research) heel-strike estimation software is simple. It estimates the time to impending heel-strike by estimating the swing foot's height above the ground. To estimate the swing-foot's height, various sensors

are used. Each set of feet has relatively accurate encoders which provide the ankle angle, the angle between the foot and the leg. A digital encoder between the legs provides a relatively accurate measurement of the angle between the stance leg and the swing leg. In addition there is an *inertial measurement unit* (IMU) mounted on the outer legs of Ranger. One of the rate gyros in the IMU provides a measure of absolute angular velocity, ω, of Ranger's outer legs from which the angle, θ, of the outer leg can be found by integration in time, assuming a good initial condition (to be discussed at length below). This gyro-based angle estimate is the least accurate part of the sensor chain. Figures 2.2 and 2.4 show where these sensors are located on the robot.

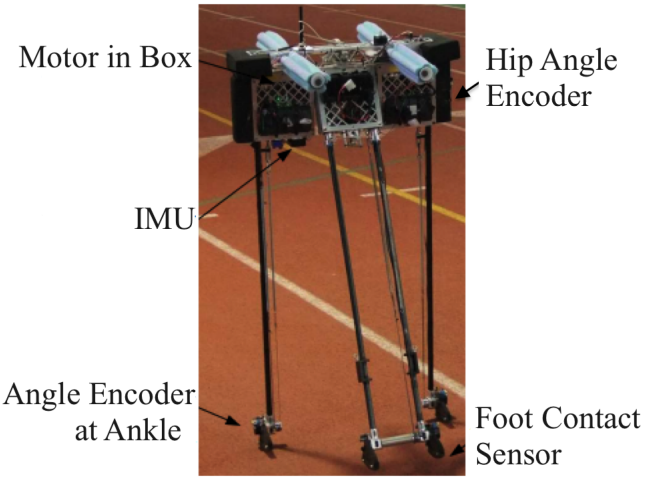

Figure 2.2: The Cornell Rangerrobot, with main motor and angle-sensor locations labeled [1].

2.3.1 Ranger Sensor Properties

The digital angle encoders used on each set of feet, and between the legs, have negligible noise relative to the IMU. They have 13 bit resolution per revolution

which gives a discretization error of only $2\pi/2^{13}$, which is approximately 0.00077 radians, or only 0.044°. After going through the leg geometry, this leads to about a (practically speaking, infinitely fine) resolution of swing-leg foot height of about 0.15 mm foot-height, (using Ranger's 0.35 m step length).

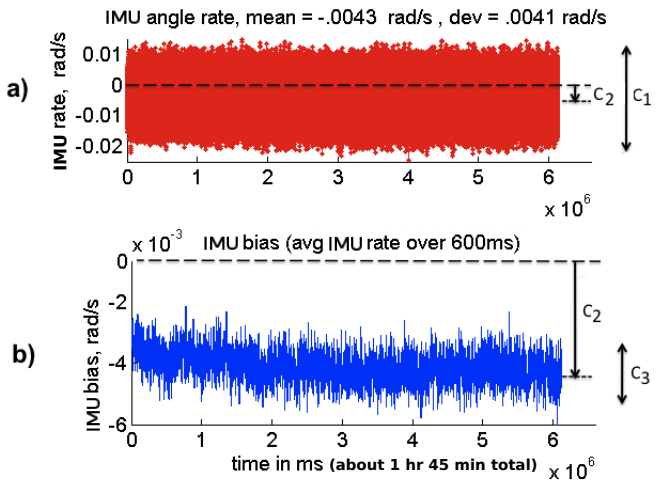

Figure 2.3: Raw IMU data plotted in (a) with mean given by C_2 and noise given by C_1. In (b) the same data is processed with a 600ms (300 sample) box-car average, producing the same mean, C_2, but a noise, C_3 reduced by a factor of about $17 \approx \sqrt{300}$.

The rate-gyro in the IMU has two main sources of error, random noise and bias-drift (a bias which drifts over longer times). These are evident in the sample data given in figure 2.3 [4]. The top plot, (a), is sample data collected from the rate gyro as it sat stationary for 105 minutes. The mean of this data, labeled C_2, is the rate-gyro bias and ideally should be constant at zero. The random noise in (a) has a peak to peak amplitude labeled C_1. This data shows the rate gyro has an almost uncorrelated gaussian noise with a variance of 0.0041 rad/s [4]. Bias-drift over time is not very evident in the top plot. To better show the drift, we define bias as the mean sensor reading over the period of one Ranger step, which is 600 milliseconds. Accordingly, the data was processed with a

box car average of 600 milliseconds. Ranger's controller frequency of 500 Hz gives a sample time step of 0.002 seconds, so for a 600 millisecond box car, there are 300 samples. Averaging sets of 300 samples produces figure 2.3 (b). Note that the mean rate-gyro bias, C_2, corresponds to an IMU angular rate mean of -0.0043 rad/s. This bias drifts slowly over time, (note: the bias at the end of (b) is close to the mean bias). The peak-to-peak magnitude of noise in (b), labeled C_3, has been reduced in comparison to C_1 by approximately $1/\sqrt{300}$ $(\approx 1/17) rad/s$, which is to be expected for uncorrelated random noise. There are ways to compensate for the rate-gyro bias drift if necessary, for example modeling the bias as a state to be estimated. For the Ranger estimator here, bias is assumed to be fully compensated one way or another, so we use $C_2 = 0$ for our modeling of the state estimator.

2.3.2 Present (before this research) Ranger Heel-strike Estimator

Using the sensors, the known geometric parameters of Ranger, and simple kinematics, the presently implemented estimator on Ranger assumes a flat walking surface to estimate the swing leg's foot's height above the ground. Because the angle encoder at the hip joint is comparatively accurate, the relative position and relative angular velocity of the legs is well measured. For this reason, the absolute angular velocity of the swing leg can be measured directly with, or derived from, the IMU rate gyro using the leg-relative-angle encoder, depending on which leg the IMU is on. The orientation of the feet, relative to the legs they are attached to, is measured using the ankle angle encoder. That is, no mat-

ter which leg the rate-gyro is on, the estimate the swing-foot height above the ground, requires the angle θ of the stance leg with respect to gravity, and relative angle of the ankle and hip. Note again, and ankle angles (angles of feet relative to legs) are measured relatively accurately by the ankle angle encoders. The main job of the estimator is to find the absolute angle of the stance leg. It does so by integrating the stance angle rate measured by the rate gyro on the IMU (or the IMU with the hip-relative-angle encoder). Note, again, the estimator uses (and trusts because of its relatively fine resolution) the relative-leg-angle encoder.

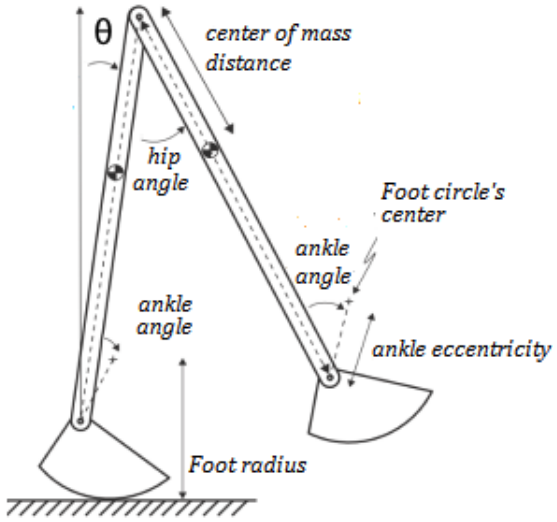

Figure 2.4: Ranger schematic with angles labeled. [4]

To better understand the presently implemented estimator, start with the following definitions for a given time-step k (k increments every 0.002 seconds)

and consult figure 2.4

θ_k = *Stance Leg Angle, Relative to Gravity*

$\hat{\theta}_k$ = *Estimate of Stance Leg Angle*

ω_k^s = *Gyro-rate derived, or directly measured, Stance-leg angular velocity*

$\hat{\omega}_k$ = *Model-based estimate, of stance leg angular velocity*

Δt = *Time since most recent sensor reading and model-estimate update (typically 0.002s)*

The estimator for leg angle takes the basic form:

$$\hat{\theta}_k = \hat{\theta}_{k-1} + \omega_k^s \cdot \Delta t \tag{2.1}$$

However, using the estimator above results in errors from noise in the rate-gyro readings. To reduce these errors, Ranger presently uses the estimator below:

$$\hat{\theta}_k = \hat{\theta}_{k-1} + \left(\frac{\hat{\omega}_k + \hat{\omega}_{k-1}}{2}\right) \cdot \Delta t \tag{2.2a}$$

$$\hat{\omega}_k = \omega_k^s \tag{2.2b}$$

The equation above can be interpreted two ways. First, it is a trapezoidal rule for integrating the stance angle rate. Second, because it takes the average of two angular velocity values, it slightly reduces the affect of noise. In this sense, it is a two-value box-car filter. A longer box-car filter, which uses more than just the most-recent two readings (i.e. a higher order filter), could be implemented for more noise cancellation. Unfortunately, higher order filters also introduce larger time lag in the estimates. The two-value box-car filter, as in equation (2.2a), is simple and worked well enough, so a more sophisticated filter was not

developed. Note that these filters, while smoothing the angular rate signal, do not improve the accuracy of the estimate of the angle equal to $\int \dot{\theta} \cdot \delta t$ because, but for the most recent value, the numerical integration is identical using, or not using, the averaging.

Heel-strike Reset

The stance-leg angle estimator uses integration and thus depends on an initial condition. An important feature of the present Ranger heel-strike estimator is the setting, and resetting, of this initial condition using 'heel-strike resetting'. If the angular velocity is integrated to produce the angle estimate $\hat{\theta}_k$ as described in equation (2.2a), over time, the errors due to integration, and more importantly drift due to sensor bias, will accumulate and cause errors in the integrated angle estimate. To prevent the accumulation of error, heel-strike reset is used. Immediately after the swing foot hits the ground (heel-strike), both legs are in contact with the ground and the swing leg is no longer swinging. This gives the estimator extra information, $\tilde{\theta}_k$, where $\tilde{\theta}_k$ is the value for θ_k derived from the unique geometry that occurs at heel-strike. Ideally, when Ranger has a heel-strike, the estimator's value, $\hat{\theta}_k$, would simply be replaced with $\tilde{\theta}_k$ (ˆfor estimate,˜for heel-strike reset). Unfortunately, heel-strike is not gentle. Parts of Ranger, such as springs, rods, and beams, experience an impact and thus vibrate. This impact causes the sensors to briefly give highly fluctuating readings. Because of these fluctuations, sensors that normally give accurate measurement of the joint angles are, just after heel-strike, relatively less accurate. Ranger uses a new value for the theta estimate, $\hat{\theta}_k^+$, using a weighted average of the present value with the geometrically derived value. Thus $\tilde{\theta}_k$ is calculated by equation (2.3). Unfor-

tunately, any real ground is not mathematically flat. This is another reason that the reset value is not reliable to predict swing foot height for the next step. The nine to one weighting used in equation (2.3) was found through the real world tuning (experimental trial and error) of Ranger.

$$\hat{\theta}_k^+ = \left(\frac{9}{10}\right)\hat{\theta}_k^- + \left(\frac{1}{10}\right)\tilde{\theta}_k \tag{2.3}$$

Where

$\hat{\theta}_k^-$ = *Estimate of stance leg angle from integrating up to* t_k

$\tilde{\theta}_k$ = *Derived stance leg angle from double-stance geometry at the present heel-strike*

This heel-strike estimator, equations (2.2a) & (2.3), was used during the world-record-setting marathon walk in May, 2011. It was determined that, with this estimator, Ranger can estimate swing-foot heigh accurately within ±3 mm, for its step length of about 35 cm and assuming high accuracy of the joint angle sensors. This is equivalent to an error in stance leg angle, θ, of about ± 0.009 radians (= 0.003/0.35).

2.3.3 Simplified proxy model

For this study of estimates, we use a simplified, proxy model of Ranger. While walking, Ranger can be thought of as a double pendulum where the stance leg, attached with a pivot to the ground, acts as the first link in a double pendulum. The swing leg, a second link in a double pendulum, is attached to the stance leg through the hips. For designing some controllers, this double pendulum model is used [5], however for most controllers a simpler model, an inverted single pendulum, is used [6]. The majority of Ranger's mass is its body, the swing leg and foot mass do not significantly impact the angular velocity of the body.

These aspects of Ranger can be captured in an inverted single pendulum model. For controllers, like Ranger's, that are most concerned with the swing leg's foot placement, one needs to know the angle of both the swing and stance legs. As previously noted, because of the relatively accurate angle encoders, the angle between the stance leg and the swing leg is well measured, but the absolute angle and angular velocity of the stance leg is less well known. And estimating the stance leg angle is the main purpose of the heel-strike estimator we discuss here. The foot position can then be measured using this angle estimate and relative angle in the hip joint. The inverted single pendulum model (figure 2.5) captures the dynamics of Ranger and estimating its state is the most important task of a heel-strike estimator.

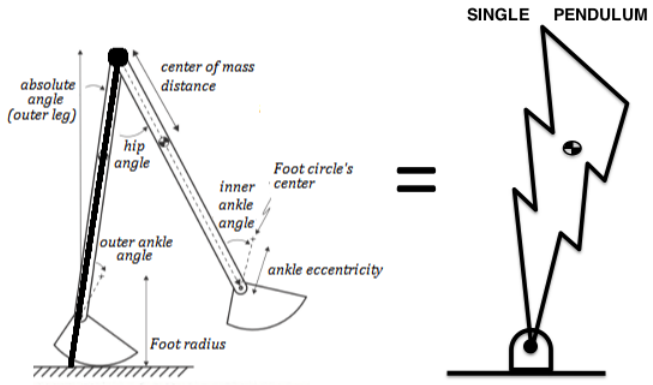

Figure 2.5: Inverted single pendulum model for Ranger does not have to be uniform or have a single point mass.

This single inverted pendulum model, a rigid object with a hinge, has several advantages over more complex models. By capturing the factors affecting heel-strike estimation with a simpler dynamics model, the relative effect of each remaining factor is more apparent and is not masked, or diluted, by behavior of the dynamics model.

For purposes of investigating heel-strike state estimators, a proxy model for this already-simplified inverted pendulum is used. An inverted pendulum needs a controller to keep it from falling over. By using a simple single, unforced, undamped, *non-inverted pendulum* model for the stance leg, no controller is needed, fig. 2.6 in our studies. Such a pendulum will continuously swing by itself around the stable position with no energy dissipation. This simplifies the description and testing of the state estimation here, and allows a simulation to focus on the heel-strike estimator, without concern for a controller and any possible influence a controller could have on the estimator. This is not a model of Ranger, but a proxy on which to test estimator ideas. While it is interesting to have a good model for Ranger, for purposes of this study it not necessary. To investigate the issues related to estimator design, any reasonable model system, incorporating repeated motions, discrete measurements, and mildly non-linear dynamics, seems acceptable.

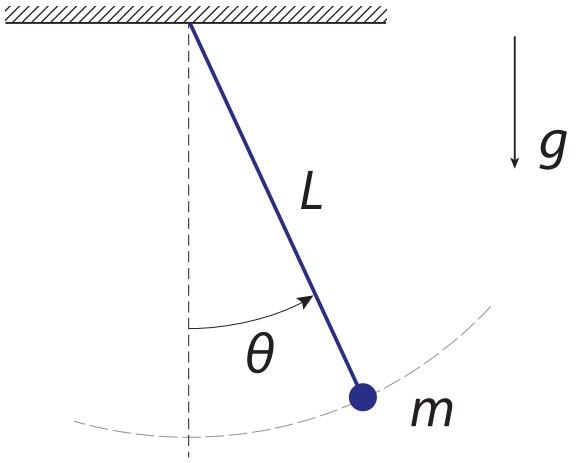

Figure 2.6: Simple, unforced, undamped, non-inverted, pendulum model. This pendulum model is a proxy for the inverted pendulum of the stance leg.

Truth model for simplified proxy model

To generate synthetic data in our estimation simulations we use a model, which we call the "truth model". Estimate errors are the difference between the "truth model" and the estimator outputs. This truth model is based on the fundamental differential equations for a simple, unforced, undamped, non-inverted pendulum as given below. Note that $F^p(t)$ is a time varying function that represents the process noise (such as extraneous torques), which will be discussed later in detail.

$$\dot{\omega}^s = -\left(\frac{g}{L}\right)\sin\theta + F^p(t) \tag{2.4a}$$

$$\dot{\theta} = \omega^s \tag{2.4b}$$

Where

g = *gravity constant*

L = *leg length*

θ = *Stance Leg Angle Relative to Gravity*

ω^s = $\dot{\theta}$ = *Stance Leg Angular Velocity*

$F^p(t)$ = Process noise

2.3.4 Why not use a Kalman filter?

A Kalman filter [7], designed to optimally remove noise using dynamic models, may appear to be a good method for combining a model and sensors to create an estimator for Ranger, or for the simplified proxy model described above. Unfortunately, several properties of Kalman filtering make it a less-than-ideal approach. A Kalman filter (and its extended versions) works best when sen-

sor and process noises are Gaussian and uncorrelated. This is not the case for the model described above, where the sensor noises and process errors are not strictly Gaussian. They are also not uncorrelated, as the effects of heel-strike are time correlated to each heel-strike event. In addition, Kalman filters do not handle model uncertainty well, and our walking does have significant model uncertainty (e.g., friction). Finally, a typically-formulated Kalman filter will try to make best estimates at all times. For a walking robot's heel-strike estimator, the main goal is to have an accurate estimate right before heel-strike. For these reasons, a direct Kalman filter was not pursued. However, using a linear combination of model-based and sensor-based estimates is an aspect of Kalman filtering with merit, and is used in the estimator proposed below. Kalman filters choose the analytical optimal linear combination to minimize the error in the estimate, assuming Gaussian noises. We will choose a numerically-found optimal linear combination, for the errors of the type we actually have.

2.3.5 Proposed model-based heel-strike estimator

A dynamic model of Ranger, plus initial conditions, could be used by itself to make heel-strike predictions. If an initial state is known, a perfect model predicts the future perfectly. However, a perfect model-based prediction is only good for finite time because any real model is not exact and thus deviates from reality at least linearly in time. Even if the model was mathematically exact, and started with the exactly correct initial conditions, model-based estimates would still drift from truth over time as numerical integration errors would accumulate, causing model and reality to diverge. More reasons as to why a system, and its model-based prediction, can differ, are discussed below.

In this chapter, we investigate if appropriately combining a model, with the sensor readings, can produce a better heel-strike estimate than is possible with just sensors and a kinematic model (as is used in Ranger's present estimator discussed in the previous section).

To create a dynamics model-based heel-strike estimator, that combines both sensor-based and model-based predictions, we start with the fundamental differential equations for each. Then we combine them. The simple, unforced, undamped, non-inverted pendulum model is described by the differential equations (2.4) and includes $F^p(t)$ which represents the process noise. These equations are used as the basis for the model-based estimator. However, as we have no model for $F^p(t)$, this term is left out of the estimator equations. The implications of this omission for the estimator will be discussed later. The purely model-based estimator is described by the following equations:

$$\dot{\omega}^m = -\left(\frac{g}{L}\right)\sin\theta^m \tag{2.5a}$$

$$\dot{\theta}^m = \omega^m \tag{2.5b}$$

Where

θ^m = *Model stance-leg-angle, relative to gravity*

ω^m = *Model stance-leg angular velocity*

For a system that has a rate-gyro sensor which produces a value for the angular velocity, ω^s, the purely sensor-based differential equations are as follows.

ω^s = *Gyro-rate-sensor-derived stance leg angular velocity*

$$\dot{\omega}^s = \textit{nothing, not a concept in the sensor-based estimate} \tag{2.6a}$$

$$\dot{\theta}^s = \omega^s \tag{2.6b}$$

The non-existence of equation (2.6a) makes the creation of differential equa-

tions for an estimator, that combines model and sensor values, a little more involved. Equations (2.6) can be rewritten using a new variable, ω^{es}, to form equations (2.7) as follows.

$$\begin{aligned} \omega^{es} &= \textit{sensor-based Stance Leg Angular Velocity Estimate} \\ \dot{\omega}^{es} &= S_{FD} \cdot (\omega^{s} - \omega^{es}) \qquad (2.7a) \\ \dot{\theta}^{s} &= \omega^{es} \qquad (2.7b) \end{aligned}$$

Equations (2.7) form a first order filter for the sensor output, ω^s, where S_{FD} determines the characteristic decay time, ***filter decay***, which is $1/S_{FD}$. As $S_{FD} \to \infty$ the characteristic decay time goes to zero and equations (2.7) revert to equations (2.6). As discussed before with equation (2.2a), the (presently implemented) Ranger estimator also filters the sensor data, using a two-value box car filter. In a similar way, S_{FD} (filter decay) is analogous to (the reciprocal of) the width of the box in the box-car filter in equation (2.2a). A discussion of the best values for S_{FD}, is given later.

The differential equations for a pure model estimator, equations (2.5), and for a pure sensor-based estimator, equations (2.7), can now be combined to produce differential equations for the proposed combined model and sensor estimator, in continuous (not yet discretized for time steps),

$$\begin{aligned} \hat{\theta} &= \textit{Estimate of Stance Leg Angle} \\ \hat{\omega} &= \textit{Estimate of Stance Leg Angular Velocity} \\ \dot{\hat{\omega}} &= (1 - S_{MS}) \underbrace{[-\left(\frac{g}{L}\right) \sin \hat{\theta}]}_{\text{MODEL}} + S_{MS} \underbrace{[S_{FD}(\omega^{s} - \hat{\omega})]}_{\text{SENSOR}} \qquad (2.8a) \\ \dot{\hat{\theta}} &= \hat{\omega}. \qquad (2.8b) \end{aligned}$$

The factor S_{MS} determines the weighting between the ***model*** and the ***sen-***

sor. The term ***model-based estimator*** will be used to refer to this estimator (equations (2.8)), which combines sensor data and a dynamics model. Likewise, all plots labeled ***"Est. Err."*** refer to errors in the estimated leg angle data, $\hat{\theta}_k$ (equations (2.8)), relative to the truth model θ_k (equations (2.4)), from this model-based estimator.

When $S_{MS} = 1$ (model sensor), equation (2.8a) reverts to equation (2.7a), the (filtered) sensor-only estimator. Conversely, when $S_{MS} = 0$, equation (2.8a) reverts to equation (2.5a), the model-only estimator. For clarity S_{MS} will be referred to as the model-sensor weighting factor (1 for all sensor, 0 for all model).

To simulate and implement this model-based estimator on Ranger, the discrete versions of equations (2.8) are needed. Assuming a time-step index k and a fixed time step size Δt, we use Euler integration for the discrete versions of these equations. The remainder of this chapter, and the primary topic of interest, is the properties of these equations compared to using a pure model (or with these same equations with $S_{MS} = 0$).

$$\hat{\omega}_{k+1} = \hat{\omega}_k + \Big[(1 - S_{MS}) \underbrace{[-\left(\frac{g}{L}\right) \sin \hat{\theta}_k]}_{\text{MODEL}} + S_{MS} \underbrace{[S_{FD}(\omega^s_{k+1} - \hat{\omega}_k)]}_{\text{SENSOR}}\Big]\Delta t \quad (2.9a)$$

$$\hat{\theta}_{k+1} = \hat{\theta}_k + \hat{\omega}_k \cdot \Delta t \quad (2.9b)$$

These equations are not complete, however, as the initial conditions for the integration are not yet specified.

Heel-strike Reset

As with the presently implemented heel-strike estimator on Ranger, when heel-strike occurs, the $\hat{\theta}^+_k$ is replaced with a new value. Similar to equation (2.3), this

new value is a combination of $\hat{\theta}_k$ and $\tilde{\theta}_k$ (estimator and geometry based estimates at heel-strike), effectively resetting the θ part of the estimator at heel-strike. The relative weighting of $\hat{\theta}_k$ and $\tilde{\theta}_k$ is another controllable factor in the model-based estimator investigated here. In contrast to equation (2.3), used by the present Ranger estimator, the model-based estimator ***heel reset*** equation contains the free parameter S_{HR}, not necessarily using $S_{HR} = 0.1$ (as was used in Ranger).

$$\hat{\theta}_k^+ = (1 - S_{HR}) \cdot \hat{\theta}_k^- + S_{HR} \cdot \tilde{\theta}_k \tag{2.10}$$

Where

$\hat{\theta}_k^-$ = *Estimate of swing leg angle from integrating up to* t_k

$\tilde{\theta}_k$ = *Derived swing leg angle from double-stance geometry*

For the proxy truth-model, given a smooth floor, heel-strike occurs when the simple pendulum is pointed straight down, defined at $\theta = 0$. This happens twice per period, once in each direction, so the angular velocity is either a maximum or minimum value. Figure 2.7 shows a short simulation of a pendulum swinging with the ”heel-strike” and corresponding velocities marked. When the floor is not smooth, floor height variations will cause heel-strike to occur at θ values close to 0, but not exactly 0.

2.3.6 Summary of Estimator Factors

Three different factor values determine the performance of the model-based estimator as given in equations (2.9a) and (2.10).

1. The first, S_{FD}, is the filter decay-factor which is related the characteristic decay time of the sensor filter. S_{FD} can range from zero (infinitely slow

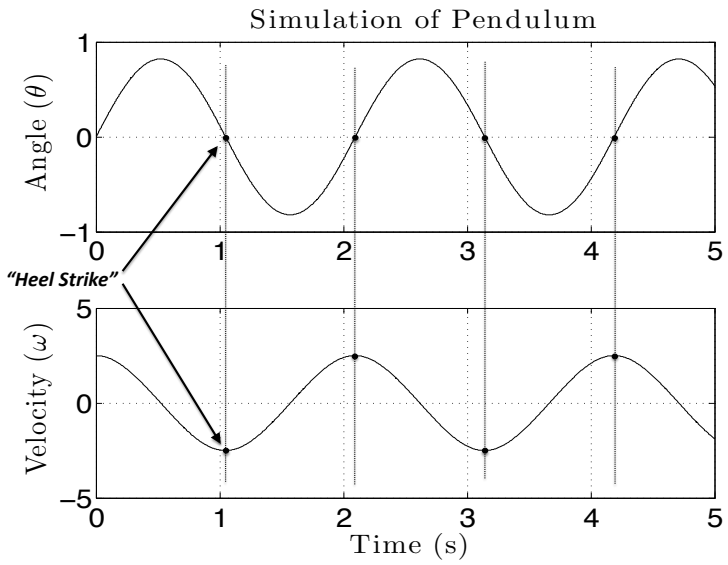

Figure 2.7: Simple, unforced, undamped, non-inverted, pendulum simulation with heel-strike, $\theta = 0$, marked with black dots.

decay) to infinity (instantaneous). As we will show, there are practical bounds on S_{FD} to allow the model-based estimator to work in the context of the finite-difference calculation.

2. The next factor is S_{MS}, which controls how much the model-based estimator relies on its model versus the sensors. When $S_{MS} = 0$ only the model is used, and when $S_{MS} = 1$ only the sensors are used. The effect of various error sources, on the model and sensors, will determine the best value for S_{MS}.

3. The final factor is S_{HR} which is the the heel-strike reset factor. This factor determines how much of the geometry based, versus the estimator based, value of θ to use when reseting the estimate of θ at heel-strike. If $S_{HR} = 1$, a full reset takes place at each heel strike, it is all geometry based. If $S_{HR} = 0$, no heel-strike reset takes place and the angle integration is never reset.

The value of S_{HR} is determined by the unevenness of the floor. The more uncertain (uneven) the floor, the smaller S_{HR} should be so that the floor height is effectively averaged over many steps. This floor-height variation effects both types of estimators the same way.

2.4 Simulation of the proposed combined estimator

Our hope is that using a dynamics model, in conjunction with sensor data, can produce an estimator that is better than one which uses just a model, or just sensors. To gain insight into using such an approach, the simplified Ranger proxy model described above was used. We think this proxy model captures the relevant features of Ranger's walking dynamics, at least for evaluating state-estimation schemes. A Matlab simulation uses this simplified model to compare this model-based estimator (using various values for the estimator constants) with Ranger's present kinematics-based estimator. In the Matlab simulation, a Runge Kutta 4th order integration algorithm (RK4) is used to create the "truth" in the simulation, and the Euler-Cromer method is used for the model, mimicking calculations that can be done on a robot microprocessor, in the model-based estimator.

The simulation includes simulations of the noises and other major causes of error that Ranger's estimator must contend with, in the real-world. The two main types of sensor errors are rate-gyro noise and collision induced errors. The main additional error is "process noise" cause by an assortment of environmental and robot conditions (wind, actuator imperfictions, etc). Finally there is floor height uncertainty, as no real walking surface is completely flat; the er-

rors of concern here, a few millimeters, are comparable to height variations on nominally-flat linoleum floors.

2.4.1 Matlab simulation

Matlab was used to simulate the simple pendulum that represents, in an abstract proxy way, Ranger. The simulation time step, Δt, is 0.002 s, which corresponds to Ranger's 500 Hz controller frequency. This is the frequency at which Ranger's processors run through the controller code. A simple non-inverted pendulum of length 1 m is unforced and undamped. The user selects an initial angle and initial velocity. Although an analytic solution is available, it is easier, and just as good for our purposes, to use for the "truth" model, the motion of the pendulum as simulated using a Runge Kutta 4th order integration algorithm (RK4), as given below. The estimates from the present Ranger estimator and the model-based estimator are compared to the values of the truth model. This gives a measure of how well the estimators function. The process noise, $F^p(t)$, will be discussed later and is added directly in the angular acceleration calculation in the truth model, as repeated below.

$$\begin{aligned} \dot{\omega} &= -\left(\frac{g}{L}\right)\sin\theta + F^p(t) \quad (Truth\ Model) \\ \dot{\theta} &= \omega \end{aligned} \tag{2.11}$$

Starting from this base, various factors and noises are added to simulate the actual Ranger walking environment and modeled $\dot{\theta}$ sensor errors. These noises make estimation of heel-strike less accurate. These factors and noises were extracted from examining the physical characteristics, and the logged data, of Ranger while in motion, and while experiencing repeated ground impacts

during heel-strike. The following sections discuss each source of error in detail, and how it impacts the simulation and the effectiveness of the present Ranger estimator and proposed model-based estimator.

2.4.2 Integration errors in the model-based estimator

The Runge Kutta 4 (RK4) algorithm used to calculate the trajectory of the "truth" model is a fourth order method with truncation error on the order of $O(\Delta t^5)$. For our Δt, it gives an error of about 10^{-9} compared to an accurate Matlab ODE45 simulation (using AbsTol and RelTol set to 10^{-15}). However, for a given step size, Runge Kutta 4 is more computationally intensive than other methods, e.g. Euler, Euler-Cromer, etc. The dynamics model used in a model-based estimator must be fast to be of use for real time estimation on a robot microprocessor. The normal Euler approximation for a swinging undamped and unforced pendulum, as given below in a generic form, has a well known problem, energy is not conserved.

$$
\begin{aligned}
g &= \textit{gravity constant} \\
L &= \textit{leg length} \\
\hat{\omega}_{k+1} &= \hat{\omega}_k - \left(\frac{g}{L}\right) \sin\hat{\theta}_k \cdot \Delta t \qquad (2.12a) \\
\hat{\theta}_{k+1} &= \hat{\theta}_k + \hat{\omega}_k \cdot \Delta t \qquad (2.12b)
\end{aligned}
$$

Thus, the equally simple Euler-Cromer method for solving the differential equations is used for the model in the model-based estimator. While Euler-Cromer is as simple (hence fast) a calculation as Euler, it does a somewhat better job of conserving energy [8]. Over one cycle, the energy increase using Euler is proportional to Δt but for Euler-Cromer it is proportional to Δt^3 [9]. The only

term that changes is in equation (2.13b) below, which solves for the new $\hat{\theta}_{k+1}$ using the new $\hat{\omega}_{k+1}$, as opposed to the Euler method, whereas the old $\hat{\omega}_k$ value is used.

$$\hat{\omega}_{k+1} = \hat{\omega}_k - \left(\frac{g}{L}\right)\sin\hat{\theta}_k \cdot \Delta t \tag{2.13a}$$

$$\hat{\theta}_{k+1} = \hat{\theta}_k + \hat{\omega}_{k+1} \cdot \Delta t \tag{2.13b}$$

Equation (2.13b) replaces Euler-method equation (2.9b) in our model-based estimator formulation. Thus, the discrete version of the differential equations that describe the proposed model-based estimator, equations (2.14), is given below, where there is only the slight change in finding $\hat{\theta}_{k+1}$.

$$\hat{\omega}_{k+1} = \hat{\omega}_k + \left[(1 - S_{MS})[-\left(\frac{g}{L}\right)\sin\hat{\theta}_k] + S_{MS}[S_{FD}(\omega^s_{k+1} - \hat{\omega}_k)]\right]\Delta t \tag{2.14a}$$

$$\hat{\theta}_{k+1} = \hat{\theta}_k + \hat{\omega}_{k+1} \cdot \Delta t \tag{2.14b}$$

$$\hat{\theta}^+_k = (1 - S_{HR}) \cdot \hat{\theta}^-_k + S_{HR} \cdot \tilde{\theta}_k \quad \textit{(if k = heel-strike)} \tag{2.14c}$$

As figure 2.8 shows, even without heel-strike, this approximation is able to match the truth model well for a few steps. For this short simulation there are no noises, and the estimator only uses the model, $S_{MS} = 0$ (all model, no sensor), and $S_{HR} = 0$ (no heel-strike correction). In figure 2.8, the top plot gives the angle versus time plot for the truth model (the Runge Kutta 4th order solution). Below is a plot of the model-based estimator's error in angle (relative to the truth model) versus time starting with perfect initial conditions. After 20 seconds the error is still less than 10^{-4}. The growth in error is due to the difference between the accurate RK4 "truth" solution and the simpler Euler-Cromer solution. As discussed before, any small difference between the model and truth, such as period mismatch or wrong phase, will cause the model-based estimate to drift further and further from truth just as figure 2.8 illustrates.

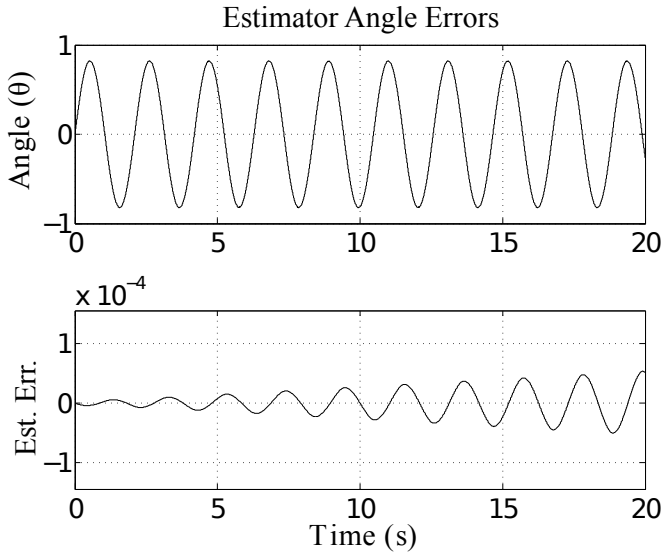

Figure 2.8: Plot on top shows the angle versus time data as output by the pure model-based estimator, $S_{MS} = 0$ (all model, no sensor), and $S_{HR} = 0$ (no heel-strike correction). The bottom plot shows the error, initially small ($\sim 10^{-4}$) but growing, between the model's estimate and the truth model. This is just the numerical error in the Euler-Cromer integration. To check this growth, we need a sensor or a reset capability in the integration.

2.4.3 Sensor errors

The angular-rate sensor error has two major contributors. The first is the inherent electrical sensor noise and drift (*the rate-gyro noise*). The second is due to mechanical vibrations exacerbated by heel-strike collisions (*the collision noise*).

Rate-gyro noise

Generally, rate-gyro absolute-angle sensors are much less accurate than relative-angle encoders. Both the present Ranger estimator, and the proposed model-based estimator, must contend with rate-gyro inaccuracies. To simulate the out-

put of this sensor, the truth model output is combined with simulated sensor error noise to create the simulated sensor reading. Because gyro-bias drifts slowly, so is assumed to be negated, the additive gyro noise is a random variable with gaussian distribution (on Ranger the standard deviation is 0.0041 rad/s, see figure 2.3) and bias = 0.

$$\omega_k^s \;=\; \omega_k - \; mean\ sensor\ noise + \underbrace{bias}_{0}$$

This ω_k^s sensor value is then used to test the presently implemented Ranger estimator equation (2.2a). Similarly ω_k^s is also used with our new model-based estimator, equation (2.14a).

The effects of rate-gyro noise can be seen in figure 2.9 below, which shows the output of two short simulations, using the present Ranger estimator, where the only noise or error is the rate gyro noise. In the simulation, the present Ranger heel reset value, $S_{HR} = 0.1$, was used. A noise with a larger standard deviation of $\sigma = 0.0051 rad/s$, which is even larger than Ranger's rate-gyro noise, was used in the simulation.

Collision-noise error

Another cause of error is the noise generated at each heel-strike. After the initial impact, components, such as springs and beams within Ranger, continue to vibrate. The effect of the heel-strike collision on the leg angle-encoder has been discussed, and explains why the heel-strike reset requires equation (2.3), a weighted combination of the estimated and geometrically derived θ, even if the floor is exactly flat.

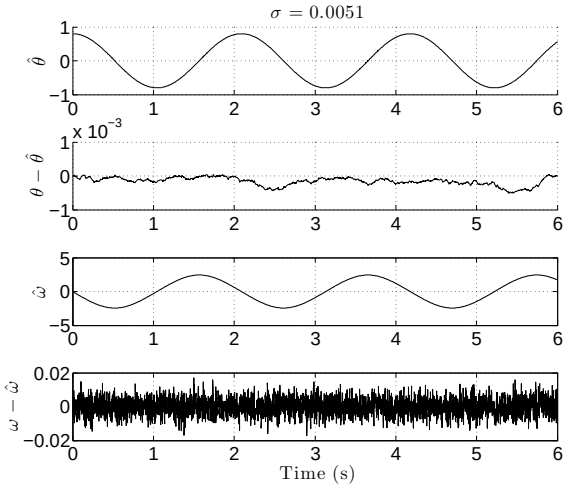

Figure 2.9: The presently implemented Ranger estimator, applied to the simple pendulum model, where the sensor is only affected by gyro noise. The heel-strike reset is $S_{HR} = 0.1$ There is no model (only sensor) being used in the Ranger estimator, that is $S_{MS} = 1$. The top plot is the angle as estimated by Ranger, the second plot shows the error when compared to the truth model. The third plot is the estimated angular rate ω and the forth plot is the estimator rate error, relative to the truth model. Note that the angle error is as large as about 3×10^{-4}, corresponding to about 0.1 mm error in foot height.

The collisions also affect the rate gyro. On Ranger, the corresponding error has been characterized as having a maximum amplitude of 0.5 rad/s and dies down to almost zero within one quarter of a swing phase [4]. Each noise error value also has a 0.995 correlation with the previous value, 0.002 seconds earlier [4]. In the simulation, error values are generated corresponding to collision noise and added to the rate gyro sensor values.

The effects of collision noise can be seen in figure 2.10 below, which shows the output of a short simulation using the presently implemented Ranger estimator, where the only error is sensor error, due to collision noise. The present

Ranger heel reset value, $S_{HR} = 0.1$ and a noise amplitude of $0.5 rad/s$ were used in the simulation.

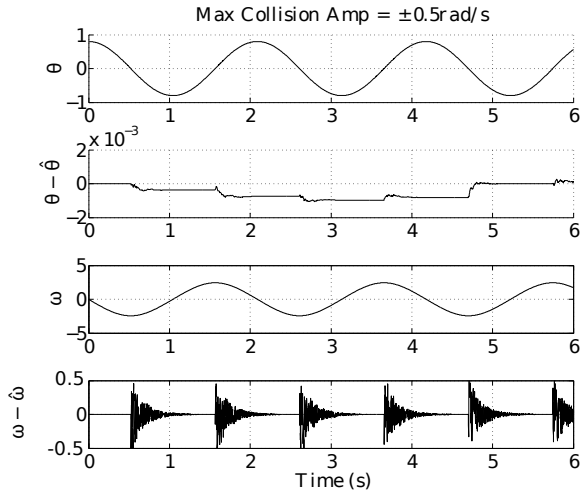

Figure 2.10: Ranger Estimator results with $S_{HR} = 0.1$ and collision noise error has a maximum amplitude $0.5 rad/s$. As in figure 2.9, the sensor is only affected by gyro noise. The top plot is the true angle, θ, and the second plot is the estimator angle error, $\theta - \hat{\theta}$. The third plot is the true angular rate, ω, and the fourth plot is the estimator error for the angular rate, $\omega - \hat{\omega}$. Note angle errors up to about 10^{-3}, corresponding to foot-height errors of about 0.3 mm.

2.4.4 Process-noise error

Process noise is represented by the $F^p(t)$ term, first introduced in equation (2.4a). This noise accounts for the errors observed in the logged Ranger data; these are not attributed to any of the sources previously discussed. Process noise has several causes, including wind, environmental disturbances, internal bumping of the motors, and motor non-linearities. This noise is directly added to the truth model via the $F^p(t)$ term in the Runge Kutta 4 (RK4) truth calculation presented

earlier in equation 2.11. It is modeled as a constant disturbance torque that lasts for a short period of time before randomly changing amplitude. The magnitude of the process noise, and duration that the process noise, is held constant as will be discussed later. Because process noise directly affects the system, it is roughly equivalent to un-modeled dynamics for the proposed model-based estimator.

2.4.5 Floor Uncertainty

If the floor is perfectly flat and smooth and geometric calculations are accurate, it is clear that at heel-strike the estimated swing leg angle, $\hat{\theta}$, should be completely replaced with the geometrically derived value, $\tilde{\theta}$. This behavior corresponds to $S_{HR} = 1$ as seen in equation (2.14c). The converse, completely ignoring $\tilde{\theta}$, corresponds to $S_{HR} = 0$ and, ultimately with long time, ever growing errors. Unfortunately the floor is never perfectly flat and the geometry based calculation to find $\tilde{\theta}$ is never perfect due to the angle sensor errors and floor roughness. Choosing a value for S_{HR}, which combines $\hat{\theta}$ and $\tilde{\theta}$, in a model-based estimator is affected by the relative errors in $\hat{\theta}$ and $\tilde{\theta}$. Also, in selecting a value for S_{HR}, the weighting of the model component and the filter component, S_{MS}, is important as the choice of S_{MS} affects the error in $\hat{\theta}$.

The error in the geometrically derived angle, $\tilde{\theta}$, is modeled as an equivalent floor roughness, the floor height variation at heel-strike. The floor height is sampled from a uniform distribution from zero to a fixed maximum magnitude. This height error is converted into the equivalent θ value assuming a leg length of 35 cm, approximating Ranger's leg length, and is added to estimators' error at heel-strike.

2.4.6 Summary of estimator error sources

There are several sources of errors for the model-based estimator and the purely sensor-based estimator. The model-based estimator is affected by model integration error (very small) and by process noise (large). Their are two main types of sensor error, rate-gyro noise and collision error. Both the model-based estimator and the sensor-based estimator are affected by sensor noise, but the sensor-based estimator is affected to a much larger degree as the model-based estimator can use its internal model to mitigate sensor errors. Floor uncertainty, the last source of error, affects both types of estimators in the same way. In the next section we investigate how to choose the values of estimator factors to best mitigate the sources of estimator errors.

2.5 Investigating best values for S_{FD}, S_{MS}, & S_{HR}

The proposed model-based estimator, equations (2.14), has three user defined factors, S_{FD} (Filter Decay), S_{MS} (Model-Sensor Ratio), & S_{HR} (Heel-Strike Reset). The effects of integration error, sensor error, and process noise error, on the model-based estimator are investigated in this section of the chapter. By studying these effects we gain insight into the relationship between the factors and their individual and relative sensitivities to the various errors.

2.5.1 Filter Decay, S_{FD}

To begin examining values for the relative filter decay parameter, S_{FD}, first note that S_{FD} concerns only the sensor aspect of the model-based estimator, so sensor noise is our main concern in this section. The best value of S_{FD} (the one minimizing the estimator error) is mainly unaffected by process noise, which is modeled as a constant torque that is applied for a set amount of time before changing value. We model the process noise to be low frequency, held constant for a relatively long amount of time compared to the constantly noise perturbed sensor signal. A filter which best mitigates the high frequency sensor noise, a low pass filter, has little affect on the process noise. In selecting the best value of S_{FD}, our concern is high frequency sensor noise, not the low frequency process noise.

Finding the best value of S_{FD} starts with equation (2.14a) using $S_{MS} = 1$ (no model, only sensor). Equation (2.14a) becomes the following.

$$\begin{aligned} \hat{\omega}_{k+1} &= \hat{\omega}_k + [S_{FD} \cdot (\omega^s_{k+1} - \hat{\omega}_k)]\Delta t \\ &= (1 - S_{FD}\Delta t) \cdot \hat{\omega}_k \; + \; (S_{FD}\Delta t) \cdot \omega^s_{k+1} \end{aligned} \tag{2.15}$$

We use the above expression to calculate the effect of S_{FD} in simulation.

First, we find the range of S_{FD} values, for which the sensor filter is stable. Note, the expression (2.15) is a 1st order difference equation with respect to the filter estimate $\hat{\omega}_k$. In order for the equation to be stable, the linear coefficient $(1 - S_{FD})$, the eigenvalue, of the equation, has to be smaller than 1 in magnitude.[1]

[1] More generally, for a digital filter to be stable the poles of the difference equation must lie within the unit circle [10], which is equivalent to noting that the eigenvalues must be less than one in magnitude

Therefore we get,

$$|1 - S_{FD}\Delta t| < 1 \quad \Rightarrow$$

$$-1 < 1 - S_{FD}\Delta t < 1 \quad \Rightarrow$$

$$0 < S_{FD} < \frac{2}{\Delta t}$$

For all the simulations presented here, $\Delta t = 0.002$, giving the lower and upper bounds of S_{FD} as,

$$0 < S_{FD} < 1000/s$$

These bounds are confirmed in figure 2.11 below, which contains two simulated outputs of the model-based estimator. The simulations are each only two seconds long, have $S_{MS} = 1$ (all filter, no model), $S_{HR} = 0$ (no heel-strike resetting), and no process or sensor noises or model errors. In one simulation (middle plot) $S_{FD} = 0$ and there is no filtering and the estimator produces only a constant output (we set $\hat{\omega}_0 = 0$). In another (bottom plot) $S_{FD} = 1005$ and the estimator output diverges and starts to grow without bound - the filter is unstable.

Best Value for S_{FD}

With the bounds ($0 < S_{FD} < \frac{2}{\Delta t}$) established, the filter can be tested in simulation with S_{FD} value varying from the lower bound to upper bound to see which value of S_{FD} produces the smallest angle estimate error. The angle-estimate error is measured by taking the mean of all the estimated foot-height errors at heel-strike. The results of a parameter study are in figure 2.12. Each simulation was run for 3000 seconds with no noise or other error sources. In each simulation both the Ranger estimator and the model-based estimator were used, while the S_{FD} value was varied from 33 to 983 in increments of 50. The present Ranger estimator produced the same results in each run (because it does not use the S_{FD}

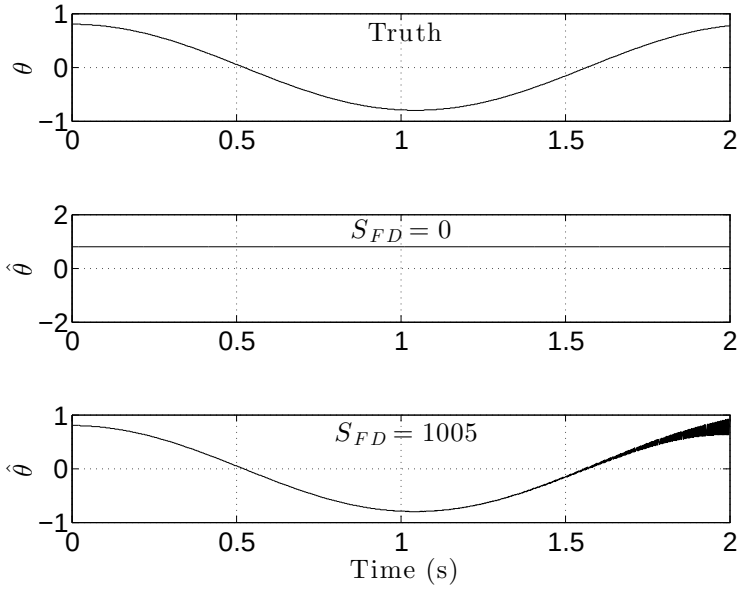

Figure 2.11: The estimator filters the rate data from the gyro/sensor, ω^s, with a first order filter, and then integrates it to produce the estimate of the angle, $\hat{\theta}$. S_{FD} acts as the coefficient of a filter for ω. If this filter coefficient is too large, $S_{FD} > 1000$, it causes the output of the filter to increase with time, the filter is numerically unstable. This is shown in the bottom plot. If the filter coefficient is set to zero, $S_{FD} = 0$, there is no useful output at all as shown in the middle plot. The top plot is the true angle, θ. All plots are the result of a two-second long simulation, with no process or sensor noise and no model error.

parameter), however, the model-based estimator had its smallest error around $S_{FD} = 333$ and the error curve is quite shallow after this point. Based on this plot we can expect that the S_{FD} that gives the minimal error in most cases, depending on the different combinations of noises and floor uncertainties, will be in the 333 neighborhood.

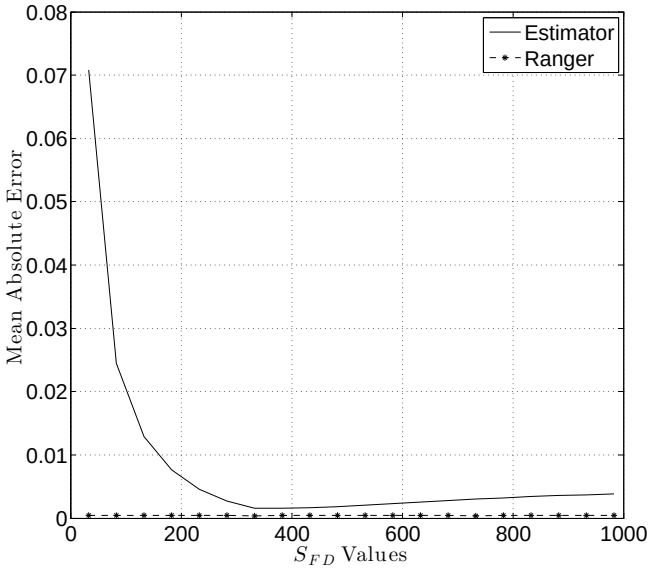

Figure 2.12: The figure above shows the optimal filter factor, S_{FD}, value to produce minimum error in angle estimate for the case of no sensor noise, process noise, or floor uncertainty. Simulations were run for 3000 seconds with S_{FD} varying from 33 to 983 in increments of 50. The mean absolute error for heel-strike is plotted for the various S_{FD} values for the model-based estimator and the present Ranger estimator, which has a fixed S_{FD} value. Because the Ranger estimator's fixed S_{FD} value, the Ranger estimator error is a constant line in the plot, while the proposed estimator does best when S_{FD} is around 333. Due to the shallow nature of the curve around $S_{FD} = 333$ we expect a range of optimal S_{FD} in the 333 neighborhood.

2.5.2 Model-to-sensor ratio, S_{MS}

The model-to-sensor ratio factor, S_{MS}, as first given in equation (2.8a) on page 20, determines the weighting between the model and the filtered sensor in the proposed model-based estimator. When $S_{MS} = 0$ only the model is used and when $S_{MS} = 1$ only the sensor is used, with values in between combining the two. Intuitively, whichever is more accurate on its own, the model or sensor,

should have the larger weighting. As described previously, process noise degrades the accuracy of the model where as sensor error degrades the accuracy of the filtered sensor output. Through simulation, we will see how the optimal S_{MS} is influenced by the relationship between process noise and sensor error. In all the simulations in this section $S_{FD} = 333$ will be used based on our results in section 2.5.1. For S_{HR} a value of 0.1 will be used to stay consistent with the presently implemented Ranger estimator. Later, we will look at the affects of varying S_{HR}..

Process noise is modeled as a torque (normalized by the moment of inertia) that is directly added to the truth model, as shown in equation (2.4a). We assume that this normalized torque is a piece-wise constant random variable. It has a random constant magnitude for a fixed period of time before randomly changing value for the next time interval. Thus, the torque is described by two parameters, the maximum allowed magnitude and the fixed period of time it is held constant. Nominal values based on our experience with Ranger are maximum magnitude of 0.5 rad/s^2 and fixed time period of 0.25 seconds. In the simulations below, examining process noise's effect on S_{MS}, the maximum magnitude varies from 0.1 to 0.9 rad/s^2 in 0.2 increments. The fixed time period varies from 0.1 to 0.4 seconds in 0.1 increments.

Sensor error is produced by a combination of rate-gyro noise, which is present throughout the step, and collision noise, which is large after heel-strike but then dies down. Nominal values based on experience with Ranger are rate gyro noise standard deviation of 0.0041 rad/s and collision noise error maximum amplitude of 0.5 rad/s. In the simulations below the rate gyro noise standard deviation will range from 0.001 to 0.007 rad/s in 0.002 increments. The

collision noise error maximum amplitude ranges from 0.1 to 0.9 rad/s in 0.2 intervals.

Process noise effects on S_{MS}

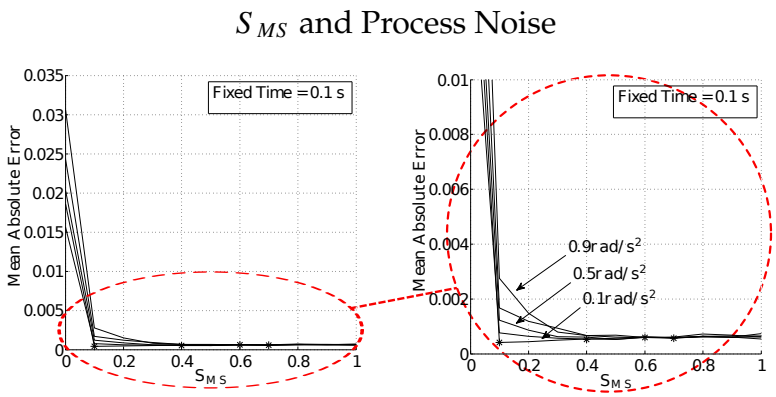

Figure 2.13: The figures above show the best model-sensor ratio, S_{MS}, to produce the minimum error in the angle estimate in the presence of process and sensor noise. The sensor noise was held fixed at its nominal value (rate gyro standard deviation = 0.0041 rad/s and collision-noise error maximum amplitude = 0.5 rad/s), while the process noise is varied. Process noise has two parameters, time period and maximum magnitude. In the plots above only the maximum magnitude of process noise varied, ranging from 0.1 to 0.9 rad/s^2 in 0.2 increments, while process time period was always 0.1 seconds. The best estimate for S_{MS} in each case is marked with an asterisk. Note that the graph is flat, minimum possible error at the optimal S_{MS} is not much different from the error in the case when no model was used. In all the simulations S_{HR} = 0.1 was used, to match Rangers present estimator.

First, we examine the process noise's effect on S_{MS}. To accomplish this we do several simulations, each 3000 seconds long, where the sensor noise characterizations are fixed, but process noise parameters vary. Figure 2.13 shows how the mean estimator angle-error at heel-strike changes as S_{MS} is varied from 0 to 1 in 0.1 increments, and process noise magnitude varies from 0.1 to 0.9,

while process noise time period is held fixed. Each curve corresponds to a different value of the process noise magnitude. The value of S_{MS} that gives the smallest model-estimate error, for the given curve, is marked with an asterisk and will be referred to as S^*_{MS}. Keep in mind that in all these simulations the sensor error-statistics are fixed to nominal values. Examining figure 2.13, we note that when the maximum process noise amplitude is small (bottom curves), a small S_{MS} produces the best estimate. However, the left plot clearly shows large error when $S_{MS} = 0$, i.e. when no sensor data is used. This pure-model error would drift to larger and larger values if the time of test was arbitrarily extended, because it has no reset at heel-strike So, because of process noise, a pure model-based estimator always benefits from being combined with some filtered sensor data, at least for heel-strike reset.

The set of simulations in figure 2.13 were repeated with different fixed process noise time periods, ranging from 0.1 to 0.4 in steps of 0.1 seconds, generating four plots as shown in figure 2.14. The plots show that as the process noise time periods increases, so does the mean absolute error right before heel-strike and that the S_{MS} values that gives the best result also increase.

In figure 2.15 the asterisks in each plot from figure 2.14 are taken and plotted together. Each curve on figure 2.15 corresponds to a single plot on figure 2.14, i.e. to a fixed time period. The lines show the relationship between the optimal S_{MS} (along the y axis) and process noise amplitude (along the x axis) for process noise time period (each line). The larger the process noise time period (which corresponds to noise auto-correlation), the larger the amount of noise because it reduces self cancellation. Note that when the process noise is small, both in magnitude and time period, the model part of the estimator is more heavily

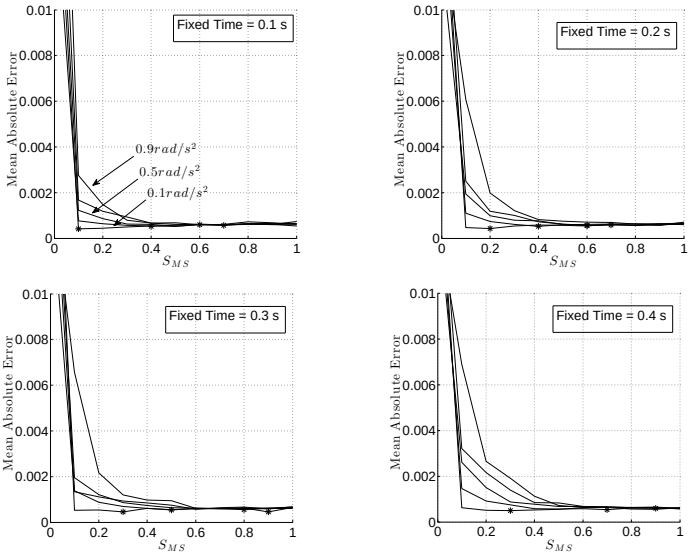

Figure 2.14: Figure above shows the best model sensor ratio, S_{MS}, to produce the minimum error in the angle estimate in presence of process and sensor noises. The sensor noise is held fixed at its nominal value (gyro deviation = 0.0041 rad/s, collision noise amplitude = 0.5 rad/s), while both the process noise parameters were varied. Note that the graph is flat, minimum possible error at the optimal S_{MS} is not much different from the error in the case when the sensor is used alone (right edge of graphs)

favored, S^*_{MS} is small for all the lines. However, as the process maximum amplitude initially increases, the variability of S^*_{MS} also increases, but as it continues to get larger the S^*_{MS} values no longer increase.

Sensor Error affects on the best S_{MS}

To examine the sensor error's affect on the best S_{MS} we repeat the simulations, but the process noise properties are kept the same throughout while the sensor noise properties are altered. In these simulations S_{MS} is varied from 0 to 1 in smaller 0.05 increments while the collision noise error maximum amplitude

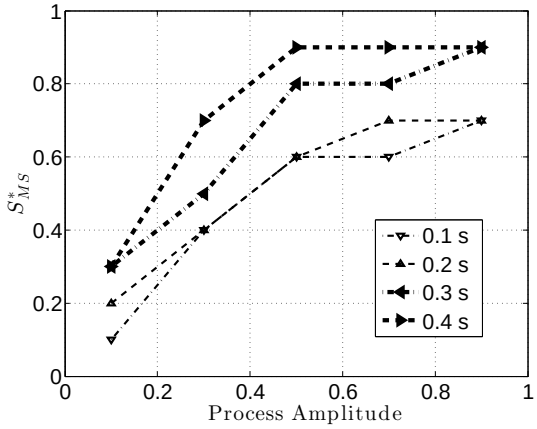

Figure 2.15: Figure above shows the optimal values of model sensor ratio S_{MS} needed to produce the lowest error in angle estimate, as a function of process noise parameters. The process noise parameters are its amplitude and process noise time period. The sensor noise is fixed at its nominal value. The figure shows the expected result that as process noise increases, the sensor is favored over the model (S_{MS} goes to 1). However, note that the actual errors in the angle estimates are not very different for each of these cases, as was seen in figure 2.14

varies from 0.1 to 0.9 in 0.2 intervals. In the following simulations, the process noise has a fixed nominal value. Note the relatively flat nature of the lines in figure 2.16. This indicates that S_{MS} can vary widely with little impact on the model-based estimator's performance (mean absolute error).

A set of four plots is shown in figure 2.16. Each plot corresponds to a different simulated rate-gyro standard deviation, and each curve to a different collision-noise error maximum-amplitude. In these four plots, the asterisks show the smallest mean absolute error right before heel-strike (hence, the best S_{MS}) for each combination of sensor noise parameters (different rate gyro standard deviations and collision noise amplitude). In figure 2.17, the asterisks in each plot are taken and plotted together, where the x axis is the collision noise

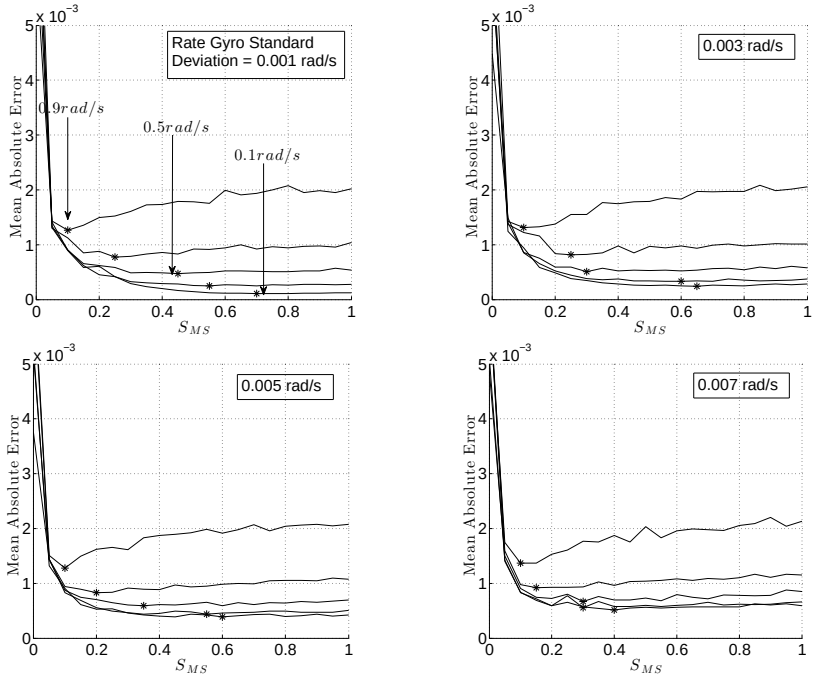

Figure 2.16: Figure above shows the optimal model-sensor ratio, S_{MS}, to produce the minimum error in the angle estimate, in the presence of process and sensor noises. The process noise is held fixed at its nominal value (amplitude = 0.0041 rad/s^2, amplitude time = 0.5 rad/s^2), while both the parameters of the sensor noise are varied. The parameters of sensors are gyro noise amplitude and the collision noise amplitude. Note that the graph is flat, except for high sensor noise values. The flat graph shows that the minimum possible error, at the optimal S_MS, is not much different from the error in the case if the sensor is used alone. As the sensor noise values get higher the trade-off between sensor and process noise becomes more clear. The model is favored relatively more, although never at the expense of total neglect of the sensors.

maximum amplitude, and the y axis is the S^*_{MS}, the S_{MS} value that gave the best result. This produces four lines, one for each rate gyro standard-deviation of 0.001, 0.003, 0.005, and 0.007 rad/s. Note, when the sensor error is large, both in rate-gyro error and collision-noise magnitude, the model part of the estimator is better, S^*_{MS} is small. Also, once the rate gyro error gets larger (larger standard

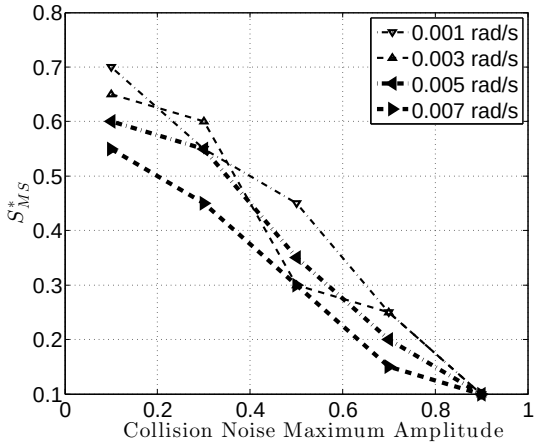

Figure 2.17: The figure above shows the optimal values of model sensor ratio S_{MS} needed to produce the lowest error in angle estimate, as a function of sensor noise parameters. The sensor noise parameters are the rate gyro amplitude standard deviation and collision noise maximum magnitude. The process noise is fixed at its nominal value. The figure shows the expected result that sensor noise increases, the models is favored over the sensor (S_{MS} goes to 0). However, note that the actual errors in the angle estimates are about the same except when the collision noise becomes higher. The relatively overlapping nature of the lines show that the S_{MS} does not depend as much on the gyro noise amplitude, as much as it depends on the collision noise amplitude.

deviation), the affect of varying collision amplitude magnitude decreases (the lines appear to converge as collision-noise max-amplitude increases).

2.5.3 Heel-Strike Reset, S_{HR}

The longer the simulation runs, the more the integration error, the sensor error, the model error, and the process noise error accumulate. But, at heel-strike, more information is available to the estimator. The leg angle can be estimated

based on geometry alone, using the relatively accurate joint sensors and assuming a flat and smooth ground. This extra, periodically available, information can be used to constrain the accumulation of errors. The heel-strike factor, S_{HR}, determines how to combine $\hat{\theta}$ at heel-strike (the estimated θ) and $\tilde{\theta}$ (the geometry derived value at heel-strike) to get a better estimate. The present Ranger estimator combines two parts, a sensor filter and the heel-strike information, using a hard coded S_{HR} value of 0.1 (where 0 means no reset and 1 is a full reset). However, the proposed model-based estimator combines three parts, a model, a filter, and the heel-strike information, using both S_{HR} and S_{MS}.

Two sets of simulations were conducted with a different fixed maximum magnitude random floor error but varying S_{MS} and S_{HR}. In each set there are four combinations of error, low sensor error and low process noise, low sensor and high process, high sensor and low process, and finally high sensor error and high process process noise. In one set of simulations the floor roughness was small, 1 mm, and in the other the floor roughness was large, 4 mm. For each specific combination of sensor error, process noise, and floor roughness, simulations were run where S_{MS} varied from 0 to 1 in 0.05 intervals and S_{HR} varied from 0 to 1 in 0.05 intervals. For each S_{MS}, the value of S_{HR} which gave the smallest estimator error was recorded as S^*_{HR}.

In the plot on the right in figure 2.18 all the S^*_{HR} values (y axis) are plotted versus the S_{MS} values (x-axis) for each of the four combinations of sensor error and process noise, yielding a plot with four lines. In the right plot of 2.18 the floor aways had a large roughness of 4mm. Floor roughness translates into a θ error when heel-strike occurs. As expected due to the large floor roughness, the $\tilde{\theta}$ at heel-strike is not very useful and the S^*_{HR} are mostly smaller than $S_{HR} = 0.1$

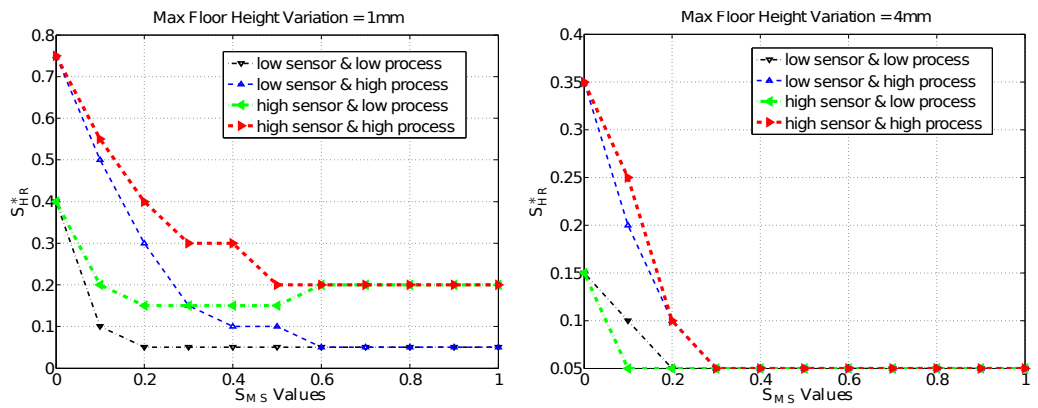

Figure 2.18: Four cases showing which S_{HR} value gave the lowest estimator error for S_{MS} ranging from 0 to 1. The four cases are the combination of low or high levels of sensor error combined with either low or high levels of process noise. Low sensor error is when the rate gyro standard deviation is 0.001 rad/s and the collision maximum amplitude is 0.1 rad/s where as high sensor error has a rate gyro standard deviation of 0.004 rad/s and collision amplitude of 0.5 rad/s. Low process noise is when the process noise fixed time interval is 0.1 s and the maximum process magnitude is 0.1 rad/s^2. High process noise has a fixed time interval of 0.25 s and maximum process magnitude of 0.5 rad/s^2. In both cases, when S_{MS} is small the low process noise made the model more desirable, as indicated with a higher S_{HR} value. The large the floor height variation, the quicker S_{HR} goes to zero as S_{MS} goes to one.

presently used in Ranger. The two lines with large process noise required the most heel reset, despite the rough floor, when S_{MS} is small. This follows from the fact that small S_{MS} means the model-based estimator is weighing the model more than the filter, and the model is most affected by process noise. Conversely, the two lines with low process noise require less heel-strike resetting when S_{MS} is small. In all four cases, once S_{MS} moves away from zero, meaning more of the filter output is used in the estimate, only a very small heel reset is desirable, again because the floor is very rough.

The interaction between S_{MS} on heel-strike S_{HR} is clearer in the case when the floor roughness is small. The two plots in figure 2.18 give the results of the same simulations, except the floor roughness was always small, maximum height change of 1 mm, for the simulations in the left plot. When S_{MS} is small (more model), the value of S^*_{HR} is driven by the amount of process noise. The two cases with large process noise are close to each other with large S^*_{HR} values. As S_{MS} increases, favoring the filter output more in the model estimate, the lines with the same level of sensor error approach each other. The two lines with high sensor error require a larger S^*_{HR} than the lines with low sensor error.

Figure 2.18 confirms the basic intuition that the optimal amount of heel-strike reset, S_{HR} is dependent on the relative errors in the model-based estimator output and the floor roughness. The figures also support the choice of a small heel reset, $S_{HR} = 0.1$, used on the present Ranger estimator. Recall that no model is used in the present Ranger estimator, i.e. $S_{MS} = 1$. They make clear the importance of having some sense of the relative error sizes induced by either process noise or sensor error.

2.6 Results

In the previous section, the values of S_{FD}, S_{MS}, and S_{HR} were investigated and considered individually. To produce the best model-based estimator, all the factors need to be optimized simultaneously, in the same simulation. In this section we put all the factors together and vary them to see under what conditions the model-based estimator does best, and how it compares to the present Ranger kinematic estimator. This is accomplished through simulations in which all the

factors and noises are varied together, and then comparing results.

2.6.1 Confirmation of Best S_{FD}

Above, it was found that S_{FD} in the neighborhood of 333 produced the best results in the model-based estimator. To get a sense of the best values of S_{FD} for all conditions a multi-factor simulation was run. In this simulation the collision maximum amplitude was varied from 0.0 to 0.8 rad/s, the process fixed time varied from 0.1 seconds to 0.5 seconds, the process maximum amplitude varied from 0.0 to 0.08 rad/s^2, gyro noise standard deviation varied from 0.0 to 0.008 rad/s, and the floor noise varied from 0 to +/- 4 mm. For each of these combinations numerous simulations were run where the S_{MS} and S_{HR} varied from 0 to 1 and S_{FD} varied from 33 to 633. The simulation with the lowest mean absolute error was found and the factors which produced this best model-based estimator was recorded. In figure 2.19 below, a plot is shown with all the best simulation results giving their error (y axis) and the S_{FD} (x axis) that produced that smallest error. The figure confirms that for all different sources of error, the combination of factors that produce the best estimator all usually had $S_{FD} = 333$. This confirms our earlier results that for the model-based estimator with the structure described in this chapter, the best value of S_{FD} is 333, or in the 333 neighborhood.

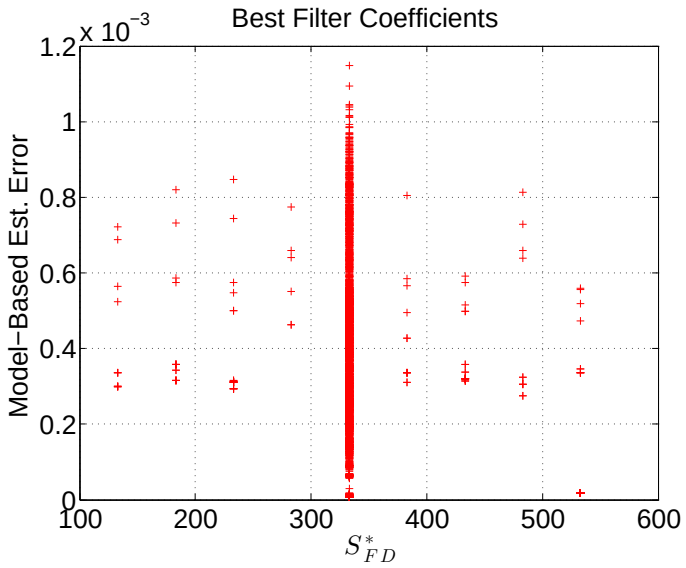

Figure 2.19: Initial large set of simulations, for each simulation the combination of S_{HR}, S_{MS}, and S_{FD} which gave the smallest error was recorded and the error and corresponding S_{FD} plotted above. Therefore, a "+" in the plot represents the best S_{FD} for one combination of error source values. The plot confirms that the vast majority of smallest model-based estimator errors were given when, at least approximately, $S_{FD} = 333$.

2.6.2 Multi-factor Simulation Results

Using the large data set generated by the multi-noise and multiple valued S_{FD}, S_{MS}, and S_{HR} simulation, it is possible to finally get a sense of the usefulness of a model-based estimator compared to a purely sensor-based estimator. To compare the two, it have to find the best performing versions of each, for each combination of noises and disturbances. Note that a non-smooth floor, which introduces a floor height error at each heel-strike, affects both estimators in the same way. Process noise has a large detrimental affect on the model-based estimator as the model does not account for process noise. Conversely, sensor noises affect the sensor-based estimator much more than the model-based esti-

mator (which has uses both a model and sensors). The two types of sensor noise are the gyro noise (which is always present while the robot is walking) and collision noise (which occurs immediately at heel-strike and then dies down). In figures 2.20 and 2.21 the affect of process noise is compared to collision noise where as in figures 2.22 and 2.23, process noise is compared to gyro noise. There is nothing to be gained by comparing collision noise and gyro noise as they are both sensor noises, and without process noise the model-based estimator is favored with big sensor noises.

The plots illustrate that it takes very little process noise to degrade the model-based estimator's performance to almost the same level as the sensor-based estimator's performance. As process noise gets very large, the model-based estimator becomes a more sensor-based estimator as S_{MS} will go to 1 (pure sensor). For this reason, a properly tuned, model-based estimator will always be just as good as, or a little better than, a purely sensor-based estimator. It is important to note that the model-based estimator has to be properly tuned, meaning the best possible value for S_{FD}. In the simulations used to generate figures 2.20, 2.21, 2.22, and 2.23, ALL possible values of S_{FD}, S_{MS}, and S_{HR} were used to find the best possible combination of values. In a real world setting the best possible combination would not be known because the exact nature of the various noises and sources of errors would not be known. For this reason a model-based estimator would need to be "tuned" through trial and error to find the best S_{FD}, S_{MS}, and S_{HR} values.

In comparing figures 2.20, 2.21, 2.22, and 2.23, the over-riding affect of floor height noise is clear. The advantage a model-based estimator might have over a sensor-based estimator is overshadowed by the errors induced by the floor

uncertainty. In fact, once any estimator's error is reliably smaller than the variation of floor height, not much is gained by improving estimation accuracy. That is, the present Ranger functions as well as it does given its present sensor only based heel-strike estimator.

2.7 Conclusions

The present Ranger estimator works well enough for Ranger to walk for record setting distances. The proposed model-based estimator can improve upon the present Ranger estimator, to a small extent, but by one or two percent, not by orders of magnitude, at least in real world cases (where the sensor errors are not large). Compare the lengths of the dashes in Figure 2.23 with the height of the points As the results above illustrate, to implement a model-based estimator it is important to know, from experience or experimentation, the characteristics of the various sources of errors. Undoubtedly for a real world implementation there would be some fine tuning of the model-based estimator factors through trial and error. Given the small improvement in performance, it will rarely be worth the extra time, effort, and wear and tear on the robot, required to tune a model-based estimator compared to tuning a simpler sensor-based estimator. In short, the simple conclusion from this chapter: it is not worth changing from sensor-and-kinematics-based estimation to encorporating the model-based estimation considered here.

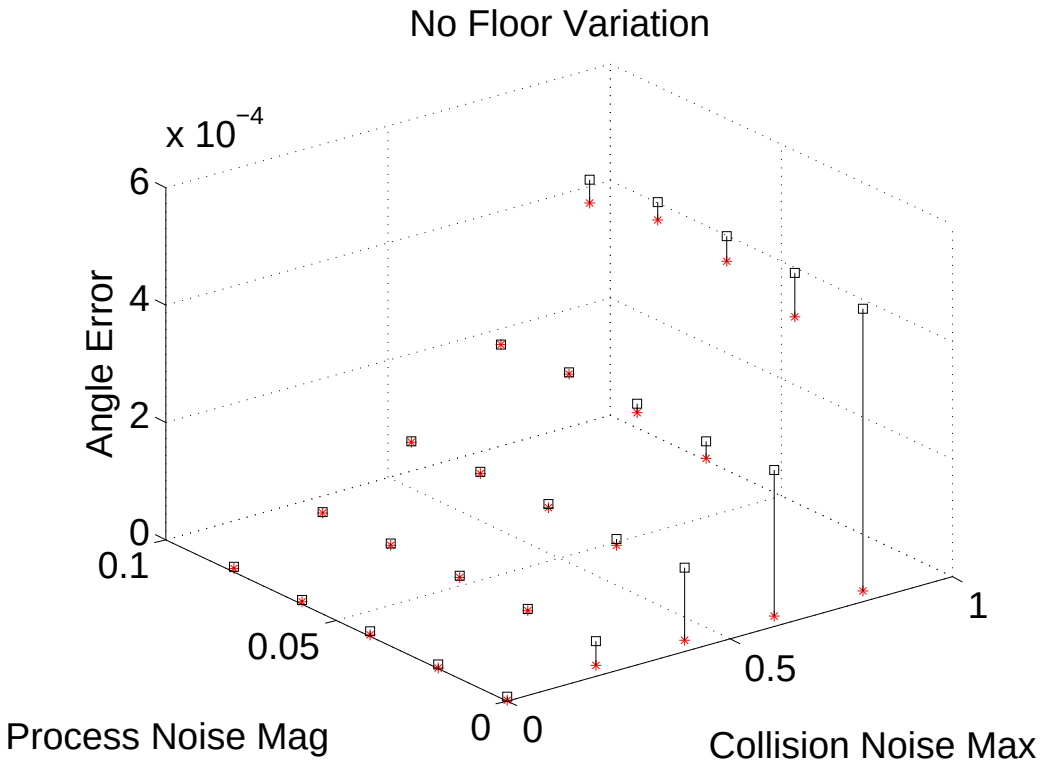

Figure 2.20: Large set of simulations with process noise and collision noise, with no gyro noise, and a completely smooth floor with no height variations (no floor noise). The process noise magnitude and collision maximum magnitude were allowed to vary and all combinations of S_{FD}, S_{MS}, and S_{HR} were run. The combination that produced the lowest estimator errors, for each combination of noises, were recorded and plotted above. The model-based estimator error is given with the asterisks and sensor-based estimator with the square. A line connects the the two for the same combination of noises to facilitate comparing them. It is interesting to note that when there is no process noise the model-based estimator does very well, but with just a little process noise the model-based estimator's performance degrades, so that it is not significantly better than the sensor-based estimator, no matter how large the collision noises.

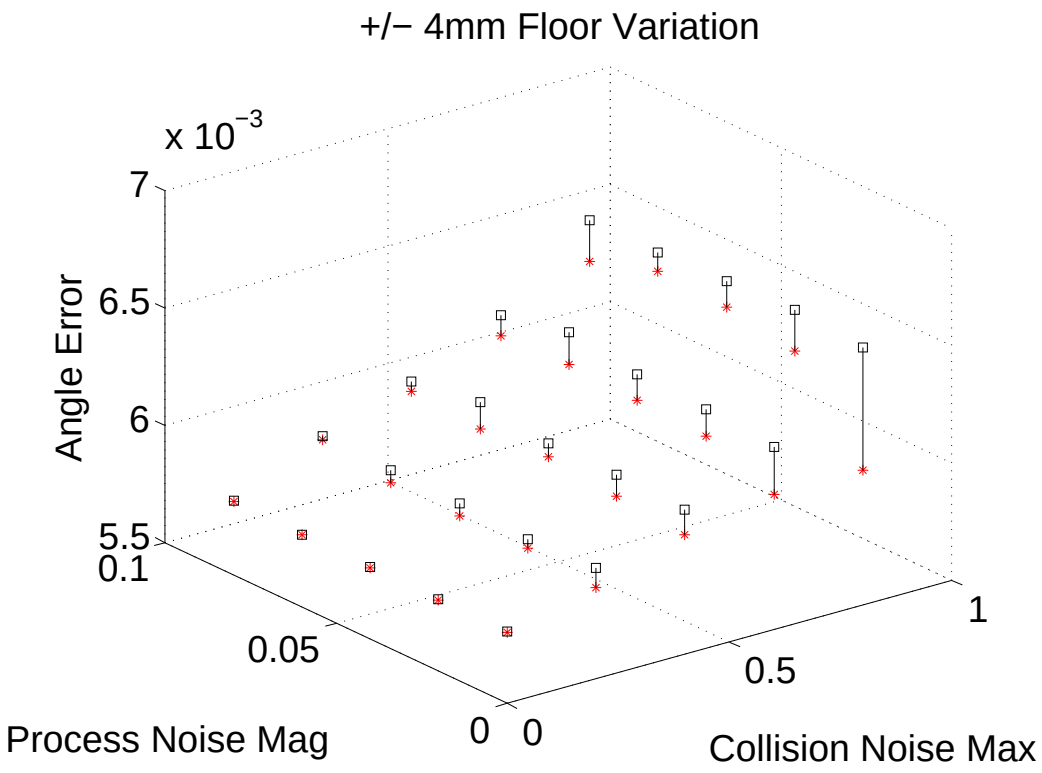

Figure 2.21: Illustration of a large set of simulations, with process noise and collision noise, with no gyro noise, and a rough floor with height variations in the ± 4mm range. The process-noise magnitude and collision maximum-magnitude noise were allowed to vary and all combinations of S_{FD}, S_{MS}, and S_{HR} were run. The combination that produced the lowest estimator errors, for each combination of noises, were recorded and plotted above. The model base estimator error is given with the asterisks and sensor-based estimator with the square. A line connects the the two for the same combination of noises to facilitate comparing them. It is interesting to note that the error induced by the rough floor in both estimators is much larger than the errors caused by process noise and collision noise when there was no floor noise, as shown in figure 2.20. This plot clearly indicates that when the floor is not smooth, there is little advantage in using a model-based estimator. Remember that the model-based estimator performance in this plot was produced with the best possible combination of S_{FD}, S_{MS}, and S_{HR}.

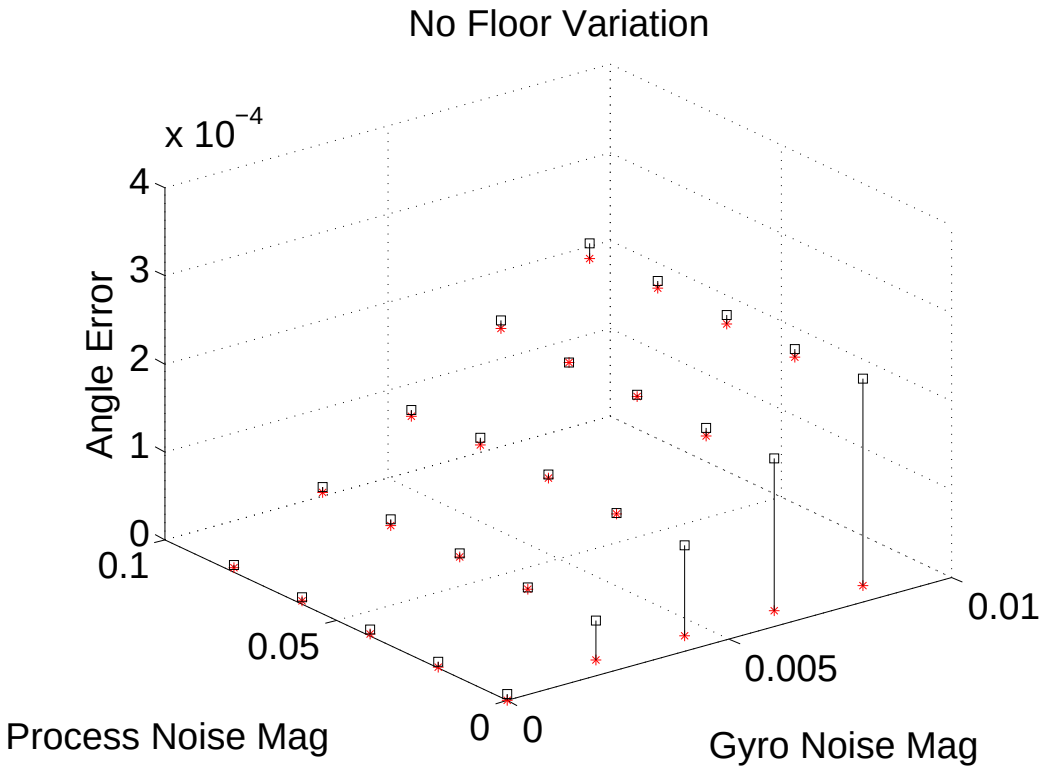

Figure 2.22: Large set of simulations with process noise and gyro noise, with no collision noise, and a completely smooth floor with no height variations (no floor noise). The process noise magnitude and gyro noise were allowed to vary and all combinations of S_{FD}, S_{MS}, and S_{HR} were run. The combination that produced the lowest estimator errors, for each combination of noises, were recorded and plotted above. The model base estimator error is given with the asterisks and sensor-based estimator with the square. A line connects the the two for the same combination of noises to facilitate comparing them. It is interesting to note that when there is no process noise the model-based estimator does very well, but with just a little process noise the model-based estimator's performance degrades, so that it is not significantly better than the sensor-based estimator, no matter how large the gyro noises.

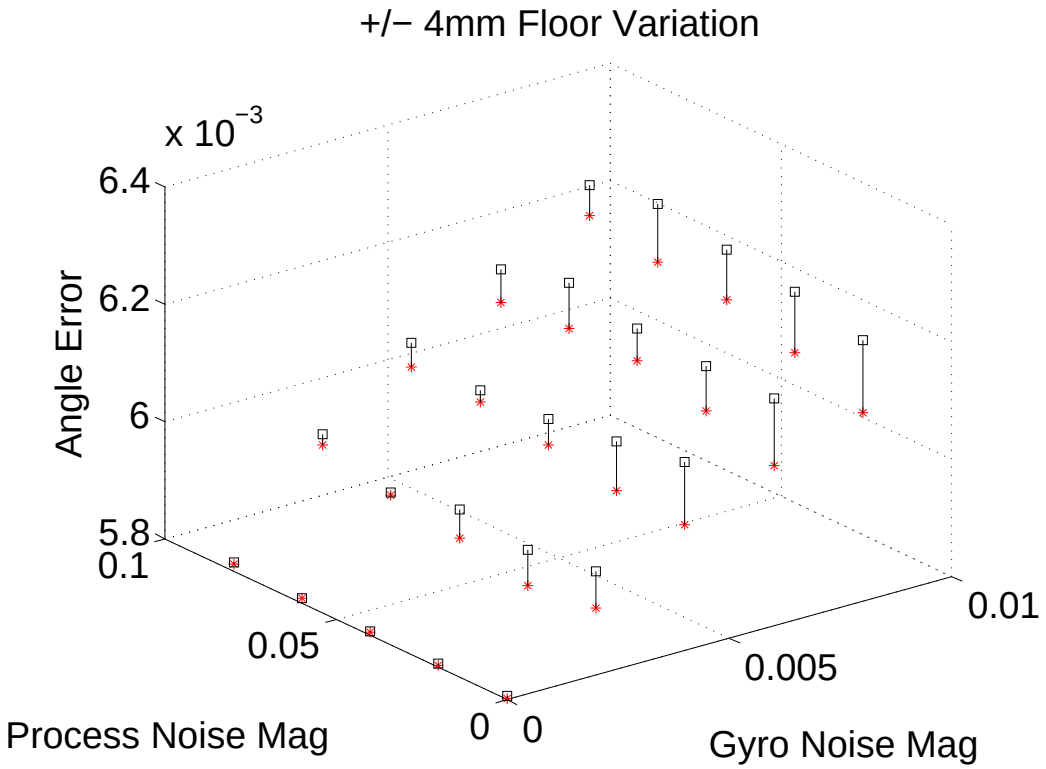

Figure 2.23: Large set of simulations with process noise and gyro noise, with no collision noise, and a rough floor with height variations in the +/- 4mm range. The process noise magnitude and gyro noise were allowed to vary and all combinations of S_{FD}, S_{MS}, and S_{HR} were run. The combination that produced the lowest estimator errors, for each combination of noises, were recorded and plotted above. The model base estimator error is given with the asterisks and sensor-based estimator with the square. A line connects the the two for the same combination of noises to facilitate comparing them. It is interesting to note that the error induced by the rough floor in both estimators is much larger than the errors caused by process noise and gyro noise when there was no floor noise, as shown in figure 2.22. This plot clearly indicates that when the floor is not smooth, there is little advantage in using a model-based estimator. Remember that the model-based estimator performance in this plot was produced with the best possible combination of S_{FD}, S_{MS}, and S_{HR}.

CHAPTER 3

WALKING ROBOT VISION SYSTEM FOR PATH FOLLOWING

3.1 Introduction

This chapter discusses the process and challenges in developing a path following vision system for a walking robot. The background and history of the effort are given to explain the motivation for the task. The following topics are then covered in detail:

- Camera Board Hardware
- Floor Line Creation
- Interfacing the Robot and Vison System
- Results and Conclusions

3.1.1 Background

Cornell Ranger is a dynamic walking robot that is almost completely autonomous with all walking control and batteries carried onboard. In early May of 2011, during Ranger's world record setting marathon walk, human steering was required. The steering input was done by a human operator via a wireless controller, of the type often used by model aircraft hobbyists. At least one operator had to follow Ranger at all times, close to 31 hours, while Ranger walked almost 308 laps around Cornell's Barton Hall running track. Figure 3.1, a photo from the marathon walk, shows a human steering Ranger via the large black R/C controller. The need for an R/C controller detracts from the sense of com-

plete autonomy.

Ranger is capable of turning because the inner legs can rotate relative to the body and outer legs. While the inner legs are swinging and Ranger is standing on its outer legs, no turning is taking place. This causes a turning lag time if a turn is commanded via the R/C controller while the inner legs are swinging. Accounting for this requires some skill on the part of the human operator. However, during the long straight sections of the Barton track, little steering input was required. From a human factors perspective, this was a difficult situation as it entailed stretches of boredom punctuated with short, stressful, moments of concentration. This experience reinforced an earlier notion that enabling Ranger self steering would be useful, despite the fact that it is not directly tied to dynamic walking, the main focus of the Biorobotics and Locomotion lab. Later, in Spring of 2013, Ranger was invited to participate in "Robots on Tour" being hosted by the University of Zurich [11]. There was interest in having Ranger walk during this event, and the organizers designed the exhibition space to include a track for Ranger. Participation in Robots on Tour gave urgency to adding self steering to Ranger. While we were able to accomplish Ranger self steering, as detailed in this chapter, Ranger was not able to walk continuously throughout all of Robots on Tour. The failure was not due to problems with self steering, but to other hardware difficulties.

3.1.2 Initial Efforts

The earliest effort to enable Ranger self-steering, before the May 2011 marathon walk, tried to use the unique colors and features of the Barton track. As figure

Figure 3.1: Ranger in early May of 2011 during world record setting marathon walk with undergraduate researchers Lauren Min (left) and Violeta Crow (right). Violeta is holding the R/C controller used to steer Ranger around the Barton Hall track.

3.1 indicates, the track floor is multicolored. The red-brown lanes are separated by white lines, and the areas outside the lanes are green. It was theorized that by adding two RGB digital color sensors (ADJD-S371-QR999), one pointed left and one pointed right, the change in color, from red-brown to green, could be detected and then used to steer Ranger around the track [12]. Several obstacles were encountered during this early effort, the largest being the reliable detection of the color difference by the RGB digital color sensor under different conditions. Variable lighting affected the contrast between the green and red-brown areas of the track. The lab was unable to get the sensor to reliably distinguish between the two as lighting conditions changed.

Another considered solution was to use the Global Positioning System (GPS)

to allow Ranger to know its own location at all times. If also given a precise map of the track, Ranger could steer itself around the track. Unfortunately GPS is not feasible for indoor locations, like Barton. The use of pseudo-satellite beacons to allow such a GPS solution to work indoors was considered but dismissed as impractical. Such a solution, like many other indoor navigation solutions, requires a significant investment in hardware, setup time, and the ability to position the hardware throughout the walking space. In addition, there were concerns that the large number of electronics and robotic systems present at the Robots On Tour conference would interfere with these wireless based indoor navigation systems. For these reasons, it was decided to revisit the idea of using a vision system on Ranger, thus eliminating the need for external sensing or computing.

3.2 On Board Camera

In late 2012, a review of present robotic vision systems was conducted. Many sophisticated robotic vision systems were found. Cornell's own SkyNet, which competed in Darpa's Urban Grand Challenge, has a vision system that allowed the vehicle to navigate a difficult and dynamic environment [13]. Unfortunately these vision systems suffered from multiple issues that made them unsuitable for Ranger, specifically weight, size, large power requirements, and cost. Ranger was designed to minimize energy use. Given Ranger's 16 watts total usage for walking, sensing, and control, adding, for example, a 5 watt vision system, such as the Microsoft Kinect [14], would kill its record breaking abilities.

On the other extreme are simple cameras which are small and low power. Driven by the mobile device industry (mobile phones and tablets), many light,

small, and low power, off the shelf camera options are available. However, these cameras are primarily optics packages designed for integration into the electronics of a mobile device. Their use would necessitate interfacing with Ranger's onboard processors, requiring advanced device drivers or an additional processing board. The largest concern was the time and effort needed to design, from scratch, a capable vision system which uses little energy. Fortunately, a good solution was found in the area of small, self built, robotic systems created by hobbyists and academics.

3.3 CMUcam4

There is a growing community of hobby roboticists, and companies and organizations who support their efforts. The CMUcam project (www.cmucam.org) is one such organization. They have developed a series of open source programmable embedded color vision sensors. The stated goal of these sensors is to make them low-cost, low-power, and suitable for mobile robots at the hobbyist level [15]. We purchased a CMUcam4, as shown in figure 3.2, for US $100 to investigate its possible use on Ranger.

3.3.1 General Specifications

The CMUcam4 packages an OmniVision 9665 CMOS camera and a Parallax P8X32A (Propeller chip) microprocessor on one board which is Arduino Shield compatible. The CMUcam4 comes ready to use, but it is also possible to customize it by changing the Propeller chip's firmware. The CMUcam4 board can

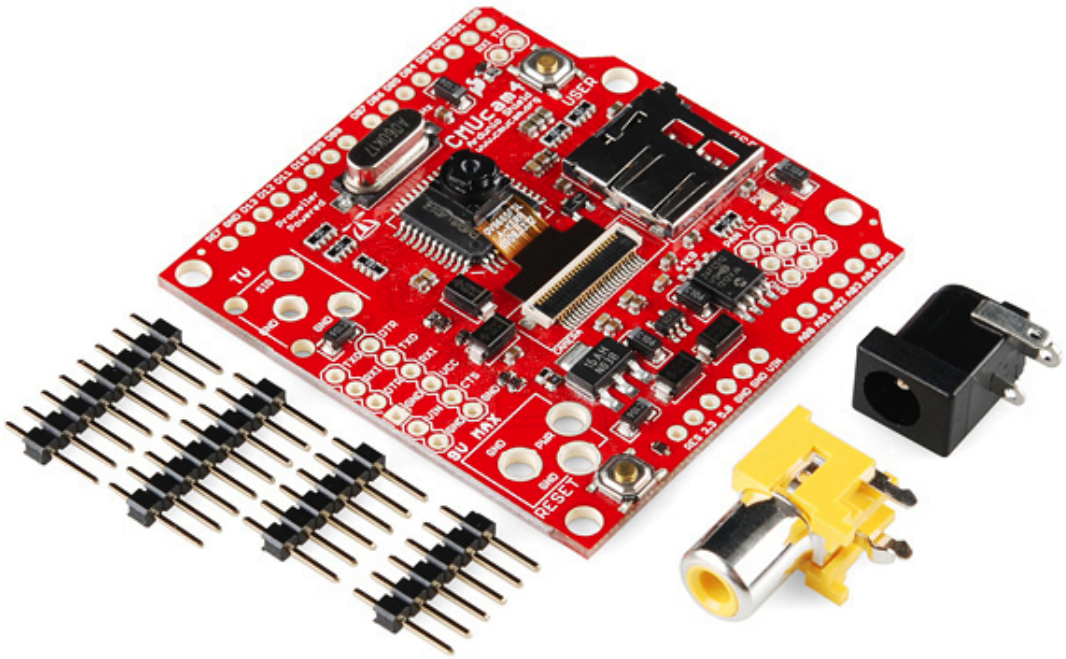

Figure 3.2: Picture from cmucam4.org showing the CMUcam4 basic board and provided connectors for user to customize board.

be connected to a computer's USB port using a Prop Plug, shown in figure 3.3. Once connected, proprietary software can be used to download new firmware directly to the Propeller chip. Parallax, the maker of the Propeller chip, supplies the Parallax Propeller Tool for Microsoft Windows platform machines [17]. For MacOS and Linux machines, Brad's Spin Tool is available on-line from a third party [18].

With its original firmware, or once it has custom firmware, the CMUcam4 board is ready to go and has ample input/output options available while it is in use. Depending on their requirements, the user can select input/output options and then solder the corresponding connectors to the board. In addition

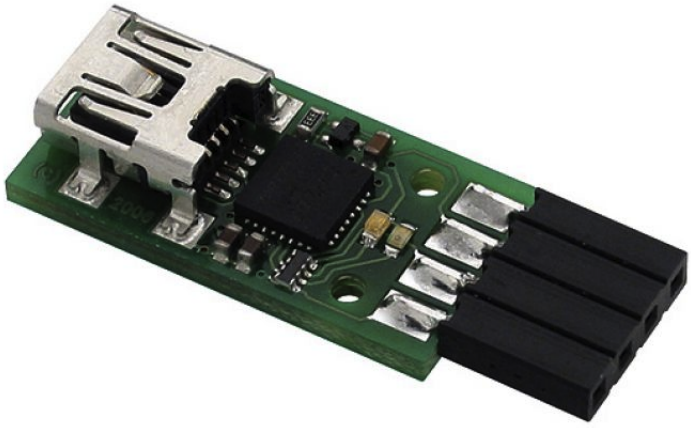

Figure 3.3: Propeller Plug tool used to connect the CMUcam4 board to a laptop via USB port.

to normal digital signal outputs, the CMUcam4 has outputs suitable for driving digital servo motors with pulse width modulated signals. This was helpful for integration onto Ranger, as described later. The board also contains a place to solder an RCA jack, allowing the CMUcam4 to be connected to an external TV monitor. These features are shown on the schematic given in figure 3.4. When active the entire package typically uses 227.5 mW, an acceptable power drain on Ranger's batteries [19]. Its small size and low weight also makes it well suited for use on Ranger.

The board comes with the Propeller chip preprogramed with the firmware necessary to drive the CMOS camera and process the output various ways. The majority of users connect the CMUcam4 to an Arduino micro-controller, and then use the supplied Arduino interface library to communicate with the board to receive camera data or issue board commands. For Ranger, the use of an Arduino in addition to the CMUcam4 was undesirable as energy and weight of the vision system needed to be minimized. In addition to the weight and power

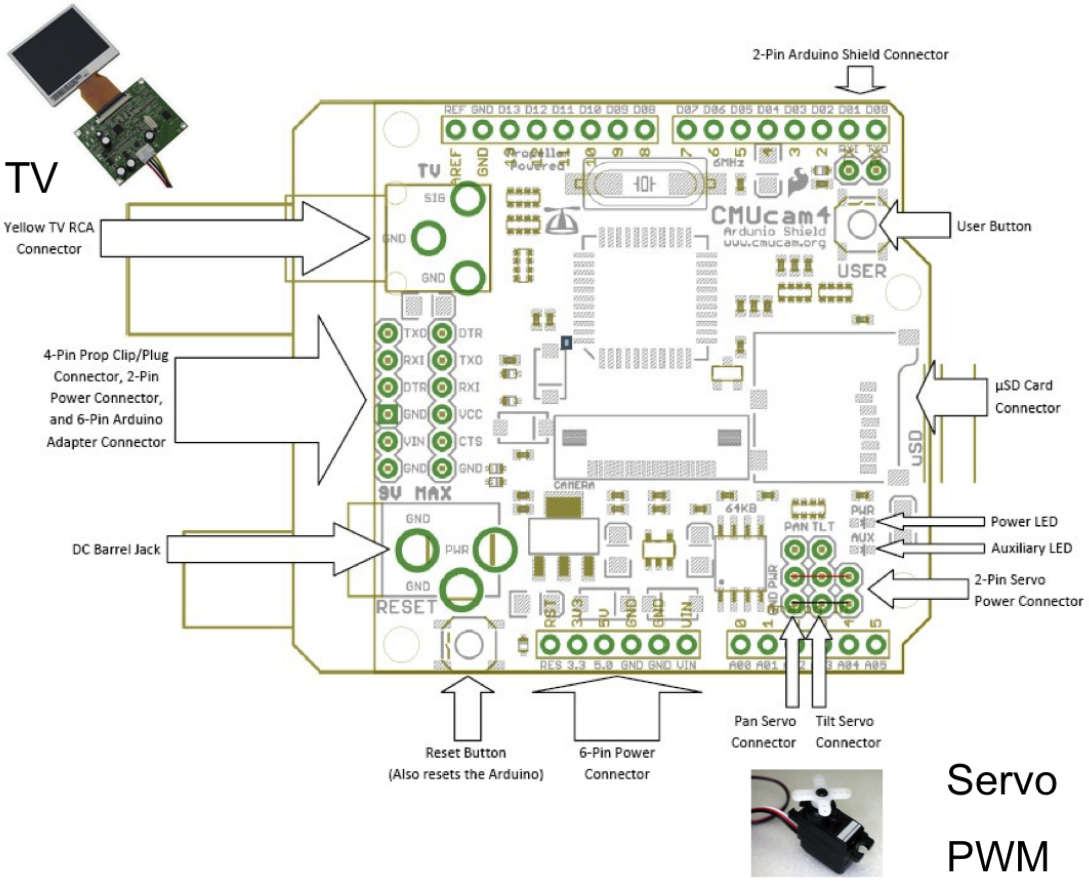

Figure 3.4: Schematic of CMUcam4 with relevant I/O portions emphasized [20].

penalty, adding an Arduino would also increase the complexity of the vision system. It was decided to use the CMUcam4 board directly with Ranger by taking advantage of the Propeller chip and the servo-control outputs. This necessitated making important alterations to the supplied Propeller micro-control code, the firmware, of the CMUcam4 board.

3.3.2 Firmware Rewrite

The firmware supplied on the Propeller chip of the CMUcam4 board already does many crucial tasks. The most important is interfacing with the OmniVision 9665 CMOS camera. The camera needs constant power and constant control signals to enable it to function properly and provide raw vision data. This low level camera code was kept unchanged in the firmware. The supplied firmware also takes the raw output of the camera and processes it into a suitable RCA signal for output to an external TV monitor. The ability to use an external monitor was crucial in testing and debugging and was left in the firmware code.

In addition, the Propeller chip runs the software necessary to allow an external Arduino micro-controller to receive data and send commands. This capability was not needed for Ranger and removed. Also removed was the auto color calibration done upon power up. Calibrating the camera is a crucial, but difficult, part of making the CMUcam4 board truly useful. It makes sense to sell the board with self calibration software that runs automatically on power up. This allows someone without a background in electronics or programming to use the CMUcam4 quickly without much effort. For our purposes this self calibration made the board less useful as its characteristics would be determined by lighting, and other environmental conditions, at power up. Self calibration on power up causes the camera board to function differently each time it is turned on. This code was replaced with hard coded calibration values as discussed below.

The board comes with a demonstration program that is activated after power up by pressing the user button, shown in the upper right corner of figure 3.4. Once activated, the demonstration program automatically analyzes the color of

the pixels in the center of the camera's view for several seconds. When this is done, the demonstration program switches to continuous tracking. During tracking, all pixels from the camera's CCD are analyzed and the ones matching the initial color are noted. The centroid of these pixels is calculated. While the centroid is not in the center of the camera's view, control commands are found using a proportional integral derivative (PID) controller. The outputs of the PID controller are x and y servo positions which are encoded on pulse width modulated (PWM) servo command signals. These signals are sent to the two dedicated servo outputs, one for x and one for y, on the CMUcam4 board. If the board and servos are connected properly to a frame that is free to rotate around its x and y axis, the servo commands from the PID controller will move the frame until the centroid is in the center. If the color being followed is a line on the walking surface, such as one of the white lane lines on the Barton track, Ranger could follow the line and steer itself around the track. For this reason, the demo program served as a template for writing the Ranger Line Following (RLF) firmware.

To create the Ranger Line Following (RLF) firmware, the automatic color analysis was removed and replaced with hardcoded color tracking values. The code that tracks the color pixels' centroid across the camera's view was altered so that it would track the color only along the x axis within a window that is in the middle third of the camera's view as illustrated in figure 3.5. The reasoning for this is given in the discussion of calibration and testing. Figure 3.6 shows the TV output from the CMUcam4 board running the Ranger Line Detection firmware when a line is detected. The y axis is of no use when following a line. The centroid's x position is then used to generate a servo output. Unlike the demo program that used a PID controller, the servo output of the RLF firmware

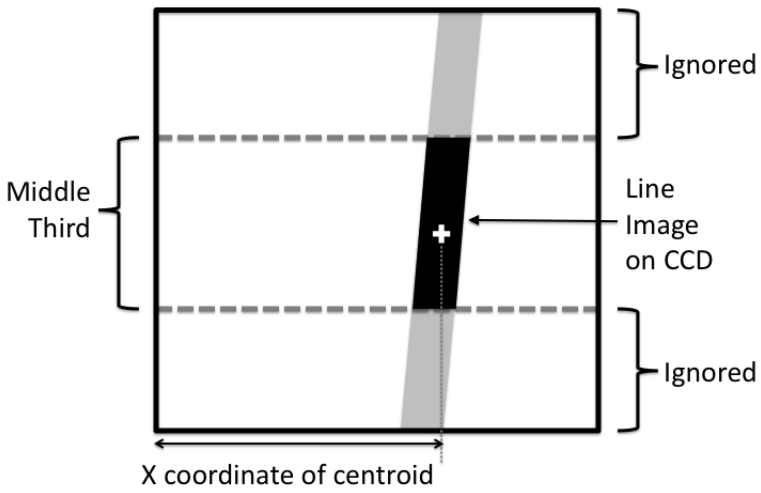

Figure 3.5: Depiction of the camera view shown. Ranger Line Following (RLF) firmware's interpretation of the view which outputs the x coordinate of the centroid.

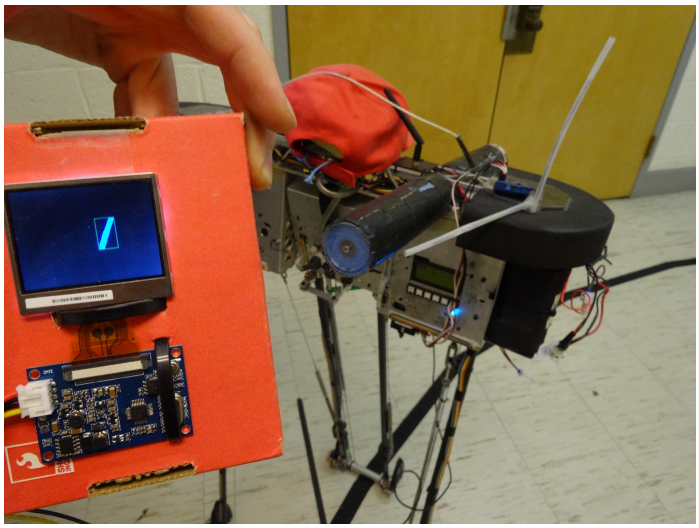

Figure 3.6: TV output from CMUcam4 board running Ranger Line Following firmware with line detected.

was changed to simply be a value indicating to Ranger where the tracked line was located within the camera view, or if no line was visible. This was done so that Ranger's far more powerful processors could be used to run a custom

steering control algorithm, which is discussed below. The Ranger Line Following (RLF) firmware code is available online [16].

3.3.3 Ranger Line Following Firmware Testing

Before altering Ranger's hardware or software to accommodate the CMUcam4 board, it was important to test the board with our Ranger Line Following (RLF) firmware to see if its performance would be adequate. The easiest solution was to create the hand held test box, shown in figure 3.7. The CMUcam4 board was mounted to the front of the box. A 2.4 inch LCD TV was mounted to the back of the box and connected to the CMUcam4 via the RCA jack. The TV shows exactly what the camera board is seeing and could be used to verify if the color calibration was working properly. The RLF firmware marks the centroid of the color being tracked with a red "+" and all pixels that match the color are white and all other pixels are black. To see if the RLF was properly encoding the centroid's coordinates into PWM signals, two servo motors were attached to the side of the box and connected to the servo output of the CMUcam4 board. Batteries to power the board, the TV, and the servos, were located inside of the box. The test box was useful for fast and easy firmware editing and board testing iterations. This proved helpful when testing and calibrating various tapes, as described below.

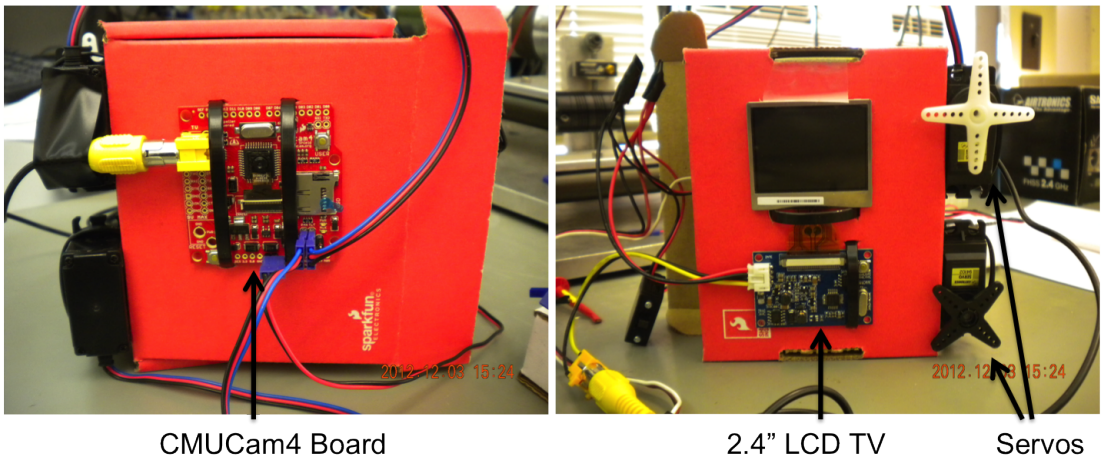

Figure 3.7: The front (left) and back (right) of the test box created for the CMUcam4.

3.4 Creating the Line

Once the CMUcam4 test box was completed, focus shifted to how to make the best line for tracking. The organizers of Robots on Tour had selected a refurbished warehouse as the venue for the event. The walking surface was a sealed cement slab and permanent changes were not allowed. This eliminated the use of paint as it would be too difficult to remove after the event. Tape then became the best option.

3.4.1 Selecting Tape

Detectability by the vision system was the most important criteria in selecting a tape to make the line for Ranger to follow. The tape had to be detectable in varying lighting conditions. The first tape test involved tapes in an assortment of colors and textures. The Bovay Laboratory Complex, located in the basement

of Thurston Hall at Cornell University, has a cement floor that is similar to the one at the Robots on Tour venue. Many tape samples were taken to the Bovay Laboratory and placed on the floor. The test box was then used to see how well the CMUcam4 and the RLF firmware could see the tapes at a distance and angle consistent with use on Ranger. With proper color calibration, all the tapes were visible in well lit conditions. Unfortunately varying the lighting greatly affected how well they could be tracked. It was thought that on a grey cement floor, light shiny tape would be easy to detect. Surprisingly at first, it was observed that the least affected by the lighting changes were the darkest tapes with matte finishes.

Advantages of Black

With further thought it became clear that, because of the physics of light and color, black tape would generally be the best choice for the line. Black tape absorbs light, the more light it absorbs, the darker it appears. Even as the ambient light increases, black tape remains black. White and color tape reflect light and their brightness depends on the ambient light. The more light, the brighter they appear, but black always remains black. In the theoretical case where we have an infinitely black tape that absorbs all light, it is always possible, no matter the lighting conditions, to adjust a camera's exposure level to distinguish between the black tape and any background. This is can be illustrated with histograms of real black tape. In figure 3.8, a black tape sample is placed over a white background and a dark grey background. The histogram for the image shows distinct peaks at three tones. From left (darkest) to right (lightest), they correspond to the black tape, the grey background, and the white background. For

comparison, in figure 3.9 the white background has been removed and the camera exposure has been altered. With the change in exposure, the peak for the black tape is still on the far left, but the grey background now appears lighter in the photo and its peak has shifted further away, towards the center of the histogram. With a theoretical perfect black tape, the left most peak would stay all the way at zero no matter the lighting conditions and the camera exposure could be adjusted to separate it from its background. With real tape, there are practical limits. Notice that the left most peak in figure 3.9 shifted to the right and is wider. If the black tape and the background are too close in color, as the exposure is adjusted the peaks would spread out and possibly overlap. The overlap would make it more difficult to distinguish the two and reliably detect the tape. With this understanding of why black tape was the best choice, the next step was to find the blackest feasible tape.

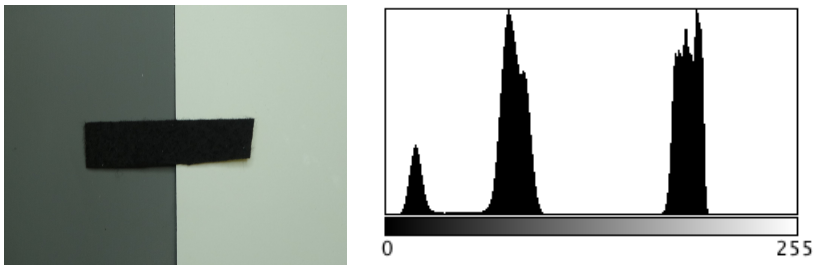

Figure 3.8: Black tape sample on grey and white backgrounds(left). Histogram with peaks, left to right, for the tape, grey background, and white background (right).

Candidate Products

Many activities rely on having black material that absorbs light. The two most common are photography and astronomy. Both these fields rely on the detection

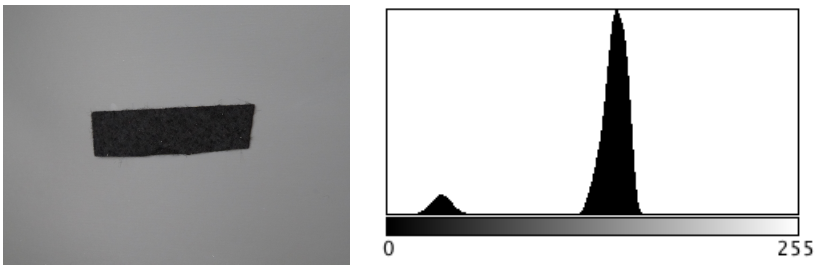

Figure 3.9: Black tape sample on grey backgrounds(left). Histogram with peaks, left to right, for the tape and grey background (right). Note that due to camera exposure differences, the peak for the same grey background as in figure 3.8 has shifted to the right.

and focusing of light. Many products exist to cater to their needs and an assortment of samples were obtained to determine suitability for use with Ranger. The best product at absorbing light was ProtoStar flocked paper [21] which is often used inside of telescopes. Unfortunately it was at least twice as expensive as the other candidate tapes. The flocked paper was sold in rolls that are 30 inches wide and up to 500 inches long, so cutting them it into a usable width would take some effort. For these two reasons the ProtoStar flocked paper was no longer considered. The initial candidate samples that were chosen all had acceptable cost, availability, durability, and usability. These candidates, shown in figure 3.10, were Duve Pro Duvetyne tape [22], Shurtape photo black tape [23], black masking tape, and gaffer's tape.

The best way to evaluate the candidate tapes was to take a picture of them in the same frame so they would be in the same lighting conditions. Then using image editing software, such as iPhoto, the histogram of the photo could be produced and the relative darkness of each sample could be compared. An example of this is shown in figure 3.11 along with the corresponding histogram. Note that the auto exposure levels are shown at the bottom of the histogram.

Figure 3.10: From left to right, the candidate tapes considered for use during Robots on Tour: Duvetyne tape, Shurtape, masking tape, and gaffer's tape.

The black triangle gives the darkest level, the white triangle gives the lightest level, and the grey triangle in between represents middle grey. The histogram clearly shows five distinct peaks. The one farthest to the right is the lightest and corresponds to the background. The four peaks left of center on the histogram each represent a different tape. Each vertical division on the histogram roughly corresponds to a doubling of light intensity, so the background luminosity is eight times as bright as the lightest tape.

By adjusting the auto exposure levels in the histogram it can be shown that the peak further to the left, the darkest, corresponds to the Duvetyne tape, which is on the far left of the photo. Figure 3.12 shows the same photo as figure 3.11, however the auto exposure levels have been adjusted in the histogram. Now the lightest level is just to the right of the darkest peak and middle grey is just to the left of the darkest peak. Adjusting the auto exposure levels causes the three samples on the left of the photo to appear white and the Duvetyne to

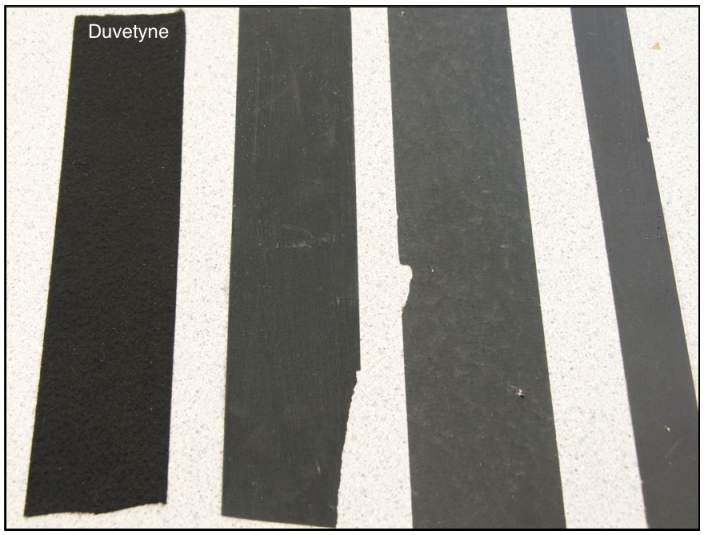

Photo of tape samples, from left to right:
Duvetyne tape, Shurtape, masking tape, and gaffer's tape.

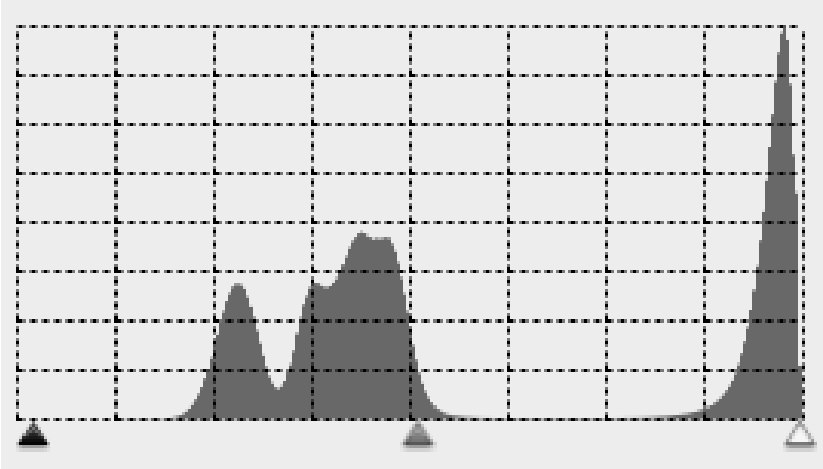

Figure 3.11: Corresponding histogram of the photo of tape samples above with unaltered auto exposure.

appear grey. This change confirms that the Duvetyne tape corresponds to the darkest peak on the histogram and that it is about twice as dark as the second darkest tape sample.

Duve Pro Duvetyne Tape was chosen as it was the darkest of the candidate tapes, with the best light absorbing properties as shown on the histogram. It is

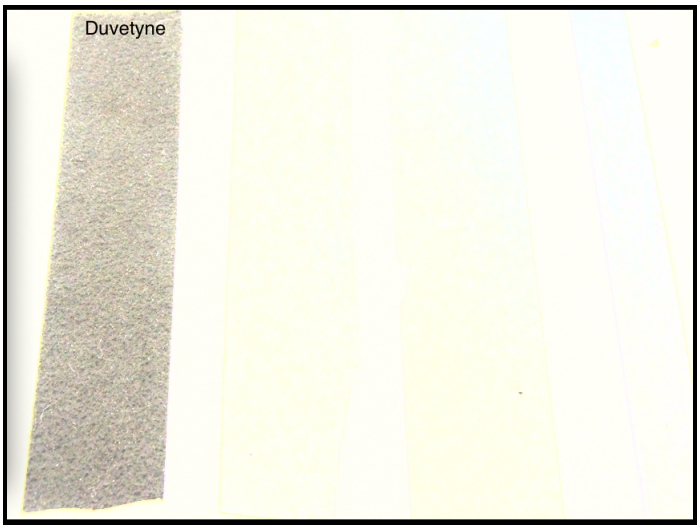

Photo of tape samples, from left to right:
Duvetyne tape, Shurtape, masking tape, and gaffer's tape.

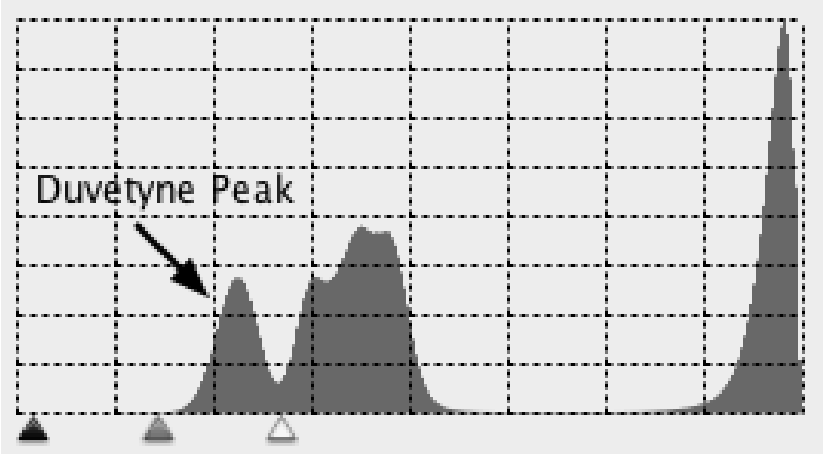

Figure 3.12: Corresponding histogram of the photo of tape samples above with altered auto exposure.

a fabric tape that comes in rolls that are 2 inches wide. It's felt like texture helps it absorb more light than the other tapes and have the blackest appearance in a wide variety of lighting conditions.

3.4.2 Calibration

Once the Duvetyne tape was selected, the Ranger Line Following (RLF) firmware was hard coded with the calibration values that best allowed the CMUcam4 to detect the tape in various lighting conditions. The calibration process was straight forward, but required a few steps. The original demo firmware was reinstalled onto the CMUcam4 board. It has the code to enable the camera to take photos which are stored to a microSD card. The test box was used to take photos of Duvetyne tape in various lighting conditions. These files were then transferred to a computer where software was used to analyze the photos. An example of a photo taken with the CMUcam4 is given in figure 3.13. Using the histogram function on a simple image editing program, the RGB component histograms were generated as shown in figure 3.13. The histograms indicate that the black tape has red, green, and blue components in the mid-30's. A perfectly black tape would have all zero values. For comparison, the light background (speckled grey linoleum) in the same photo has component values in the 160 to 170 range.

In different lighting conditions the auto exposure caused the same tape to have pixels with higher or lower red, green, and blue components. Because of this variation, the analysis had to be done in different conditions. For the RLF to work well it was important that a set of threshold values was selected that was between the tape's RGB component values and the floor's values, for a large range of lighting conditions. This set of threshold values was then hard coded in the Ranger Line Following software. When the RLF is running, each pixel of the camera output is analyzed to determine its RGB components. Any pixels whose three components are lower than the corresponding RGB thresholds, is

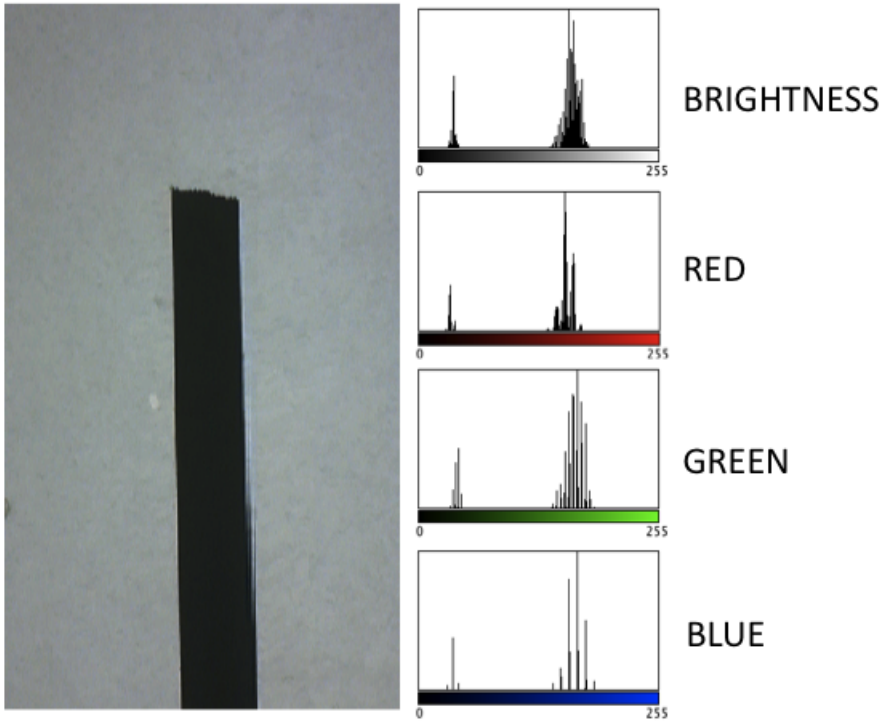

Figure 3.13: Picture of Duvetyne tape taken with CMUcam4 board's camera along with associated histograms: brightness, red, green, and blue.

considered part of the tape. The x coordinate of the centroid of all the pixels that are part of the tape is encoded on the PWM servo output. If too few pixels are below threshold, a special value is sent to the servo output to signal to Ranger that the tracking has failed.

3.5 Ranger Interface

Once the CMUcam4 was reliably tracking a line created with the Duvetyne tape, it had to be integrated with Ranger's hardware and software. Not only did it need to work on Ranger, but work in a way to allow for quick and easy switch-

ing between self steering and human steering.

3.5.1 Hardware and Mounting

A suitable location on the front of Ranger's center body segment was found. The board was mounted to this location but at an angle determined to give the camera a view of the walking surface several feet in front of Ranger, a roughly 45° angle. While Ranger is walking, the camera's view of the floor also changes as the inner legs are attached to the center body segment. When the inner legs are swinging forward, the camera sees further ahead. When the inner legs are stationary, the camera looks directly in front of Ranger. To minimize the chance of seeing Ranger's own feet, or looking too far ahead, the Ranger Line Following code uses a window that is only one third of the camera's total view. The camera board mount (left) and a close up of the camera board (right) are shown in figure 3.14.

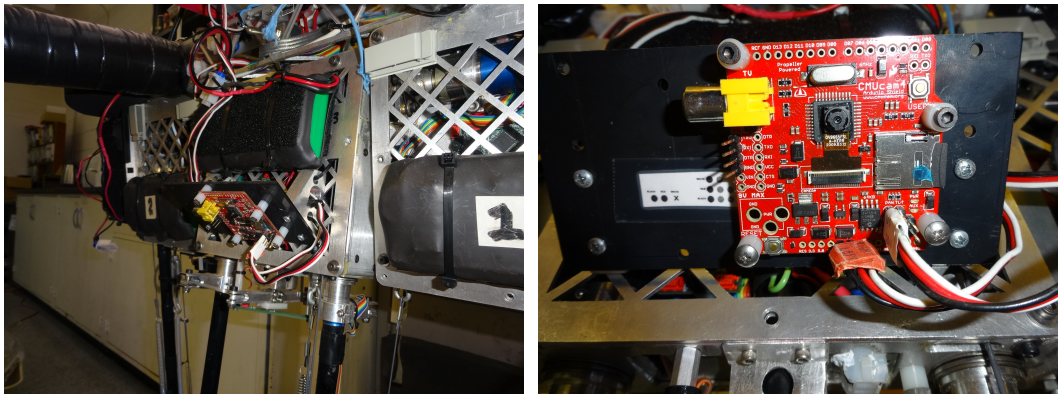

Figure 3.14: Side view of the camera board mounted on Ranger (left) and a close up front view (right).

In the close up photo of figure 3.14, two wire connectors are shown; one is

to power the board, and the other connects the servo output to one of Ranger's ports on its dedicated I/O board. While powered, the board is constantly running the Ranger Line Following software and constantly outputting the x coordinate of the line, or a flag value if no line is visible.

3.5.2 Software and User Interface

The picture on the left in figure 3.15 shows the R/C controller with the steering selection switch and steering joystick labeled [24]. These were chosen to allow the R/C controller to be held with the operator's left hand while using the switch and joystick with their right hand. When the steering selection switch is flipped toward the front of the controller, pointing towards the operator, the R/C steering commands from the joystick are used by Ranger. When the switch is pushed back, away from the operator, Ranger steers itself. No matter the steering selection switch position, Ranger is constantly receiving a signal from the CMUcam4, either containing the location of the line, or a flag indicating no line is visible. This is done so that as soon as the switch goes from operator control to Ranger steering control, Ranger is ready to self steer using RLF output from the CMUcam4 board.

If self steering is being used and line tracking is lost, it is vital for the operator to know immediately so they can take over steering control. For this reason all six of the rear facing status LED's flash red if the CMUcam4 board loses track of the line. This is shown in the photo on the right in figure 3.15.

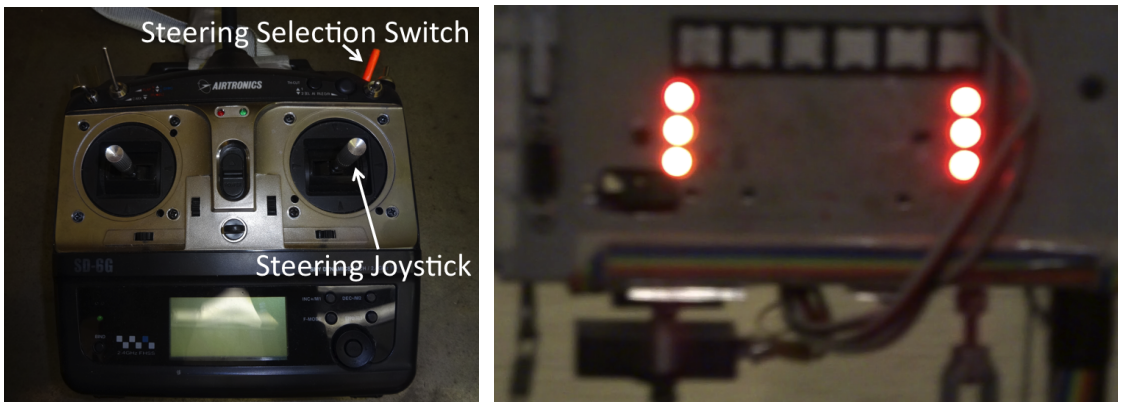

Figure 3.15: The R/C controller with steering selection switch and steering joystick labeled (left). All of Ranger's rear LED light panel flashing red to signal line tracking failure (right).

3.5.3 Steering Algorithm

The CMUcam4 board, running the RLF firmware, is a line detection sensor which produces one sensor value, the position of the line relative to Ranger. Using this sensor value, x, Ranger's on board processors produce a steering command to make Ranger follow the line. This steering command, θ, is found using a steering algorithm designed by Prof. Ruina. The algorithm combines the present sensor value with the previous sensor value and the previous steering command. If a, b, and c, are all weighting factors, then equation 3.1 shows how the steering command at time step k is calculated.

$$\begin{aligned} \theta &= \textit{steering angle} \\ x &= \textit{position of line (x coordinate of centroid)} \\ \theta_k &= \boldsymbol{a} \cdot x_{k-1} + \boldsymbol{b} \cdot x_k + \boldsymbol{c} \cdot \theta_{k-1} \end{aligned} \tag{3.1}$$

A Matlab simulation was used to find good values for a, b, and c. The simulation contains a simple model of Ranger including random sensor noise and

steering bias. The model accounts for the height of the camera, the angle of the camera from vertical, the camera's look ahead distance, and Ranger's step length. In the simulation, Ranger takes 200 steps using a single set of weighting factors, and at each step Ranger's distance from the line is measured. The simulation was run 1000 times with random weighting factors, a, b, and c, each time. The weighting factors from the best performing steering equation, where Ranger stayed closest to the line, were saved. This random search approach found the initial values used for the steering equation. The Matlab steering simulation code is available online [25].

Several experiments were done with Ranger at the venue the day before Robots on Tour. Through these experiments, the weighting factors were tuned, which allowed Ranger to follow a line smoothly and with minimal control effort. When following a straight line, the control algorithm which minimized overshoot was judged to be the best. To turn, Ranger uses energy to power its steering motor. Minimizing overshoot reduces the number of steering commands, saving energy and wear and tear on Ranger.

3.6 Results and Conclusions

The first test of Ranger's line following capability was performed at the local Home Depot. The floor at Home Depot is a sealed cement surface, like the Robots on Tour venue in Zurich and the Bovay lab at Cornell. The test took place late at night when the store was not crowded. A curved tape line was put down with the same 9 meter turn radius that Ranger would need to handle at Robots on Tour. The self steering system worked well with Ranger often able to

keep the line between its two inner feet as seen in figure 3.16. One advantage of Home Depot is that it is uniformly well lit and, because of the lack of windows, the lighting does not change throughout the day or night.

Figure 3.16: Ranger self steering test conducted at the Ithaca Home Depot.

In March of 2013, Ranger was packed up and taken to Zurich to participate in Robots on Tour. Figure 3.17 shows part of Ranger's track around the outside of the exhibition space. The vision system worked well and Ranger was able to completely self steer for much of the exhibition. The only vision problems occurred at lighting extremes. At night, there was one dark corner where the overhead lighting was deficient and Ranger would lose line tracking. Thickening the line with extra tape helped Ranger distinguish between the line and the floor in the darkness. During the day, there was a short period of time where the bright sun was in a location such that Ranger's own shadow contrasted enough with the floor to cause confusion. One way to address these problems in the future would be to have lighting that is more uniform. Due to mechan-

ical problems unrelated to the path following vision system, Ranger was not able to walk continuously throughout all of Robots on Tour. However, at different times throughout Robots on Tour, Ranger self steered for several consecutive laps around the exhibition track without operator intervention. Given the constantly changing environment in Zurich, Ranger's self steering capability worked remarkably well considering its low power, weight, and processing capabilty.

Figure 3.17: Part of Ranger's track at Robots on Tour in Zurich, Switzerland.

CHAPTER 4

PATH PLANNING USING MIXED INTEGER LINEAR PROGRAMMING

4.1 Introduction

Optimal path planning for Unmanned Air Vehicles (UAVs) is a difficult task due to the intrinsically non-convex nature of path optimization. Recently, new approaches have been used which utilize techniques traditionally used in the field of Operations Research. By using linearized dynamic models and mixed integer constraints, researchers have used Mixed Integer Linear Programming (MILP) for different path planning applications [26] [27] [28] [29] [30] [31] [32]. MILP problem formulations can be solved using commercial software that employs a branch and bound algorithm. This algorithm, by looking at all path possibilities, returns the optimal solution, if one exists.

Previous efforts [26] [27] have concentrated on path planning for multiple vehicles and multiple goals. Constraints have included time to waypoints and total effort, or fuel. In this paper the MILP approach for path planning is expanded to include constraints derived not only from the relationship, but also the interactions, between the UAV and adversaries.

Radar guided surface to air missiles (SAMs) are a major threat to UAVs. While the loss of a UAV does not result in a human casualty, it can jeopardize the mission and results in the loss of an important battlefield resource. Knowledge of a SAM's location and a good model of its attack capabilities can be used to create a UAV path plan with acceptable risk. A good model for a SAM's probability of detecting a UAV takes many factors into consideration. The most

basic is distance between the SAM and the UAV. Another important factor in determining whether a SAM site detects a UAV is the signature presented by the UAV to the SAM's radar. Once detected, it is possible for a UAV to avoid destruction by taking advantage of the possibility of lock loss. Lock loss occurs when radar detects a UAV but is unable to continue detection, or lock on, long enough for a SAM to be fired at the UAV.

Non-convexities, path dependencies, and sharp gradients, are introduced to the path generation problem when taking into account UAV flight dynamics, the probability of radar detection based on UAV state, and the occurrence of lock loss. Such problems have been previously approached using a gradient descent based method which can result in local minima [33]. Discrete approximations to continuous shortest paths have also been investigated in relation to SAM avoidance [34]. The approach outlined in this paper uses a linearized model of the probability of detection, lock loss phenomenon, and UAV dynamics, to create a large MILP formulation. Once solved, this MILP formulation results in the global solution. Examples of possible path plans, as a function of acceptable probabilities of detection, are also presented.

4.2 General Problem Formulation

The model is based on the Open Experimental Platform (OEP) developed by Boeing [35]. Throughout this report, it is assumed that the aircraft maintains a fixed altitude and that the radar is on the ground. The model is presented in three parts: the vehicle dynamics, the probability of detection model, and the lock loss model.

4.2.1 Vehicle Dynamics

The inputs to the aircraft model include measurements of the aircraft position and velocity as well as the destination waypoint position. A constant speed for travel between waypoints is also input. The output for the aircraft model is the aircraft position and attitude in inertial coordinates (north, east, up). The model is highly simplified and does not accurately portray the true physics of the aircraft. However, the model was developed for research in mission planning systems, and it is assumed that these planning controls will be designed and tested in conjunction with more complex models of the vehicles and their flight management systems. The model assumptions include constant altitude (2D) flight, instantaneous changes in speed, heading and bank angle, and simplified turning assumptions. During turns the bank angle is ±45°, while during stead level flight, the bank angle is zero. The following equations are used for the aircraft state for $t_i \leq t \leq t_{i+1}$ where t_i denotes when the UAV is at waypoint i

$$\begin{aligned} x(t) &= x(t_i) + U(t_i)\cos(\psi(t_i))(t - t_i) \\ y(t) &= y(t_i) + U(t_i)\sin(\psi(t_i))(t - t_i) \\ h(t) &= h(t_i) \\ \psi(t) &= \psi(t_{i+1}) \\ \phi(t) &= \frac{\pi}{4}\delta_{pt} - \frac{\pi}{4}\delta_{nt} \end{aligned} \tag{4.1}$$

where x, y, and h are the aircraft positions along the north, east, and up axes, respectively and $U(t)$ is the speed. The heading and bank angles are ψ and ϕ respectively. The integer variables, δ_{pt} and δ_{nt}, are defined for positive (right)

and negative (left) turns as follows.

$$\delta_{pt} = 1 \leftrightarrow t \leq t_i + T_{turn} \quad \text{and} \quad \psi(t_{i+1}) - \psi(t_i) \geq \varepsilon$$

$$\delta_{nt} = 1 \leftrightarrow t \leq t_i + T_{turn} \quad \text{and} \quad \psi(t_{i+1}) - \psi(t_i) \leq -\varepsilon$$

where ε is a small positive constant. The heading, $\psi(t_i)$, is the angle between the nose and north and is positive clockwise about the up axis and is defined below.

$$\psi(t_i) = \tan^{-1}\left(\frac{y(t_{i+1}) - y(t_i)}{x(t_{i+1}) - x(t_i)}\right)$$

The steady level flight turn rate equation is used to compute the turn time, T_{turn}, as follows

$$T_{turn} = \frac{\mid \psi(t_{i+1}) - \psi(t_i) \mid}{\frac{g \tan\left(\frac{\pi}{4}\right)}{U(t_i)}}$$

To compute the magnitude of the turn angle, the dot product between the velocity vector, $v(t_i) = v_x(t_i)[\underline{x}] + v_y(t_i)[\underline{y}]$, and the difference between the destination waypoint and the present waypoint

$$\Delta R = (x(t_{i+1}) - x(t_i))[\underline{x}] + (y(t_{i+1}) - y(t_i))[\underline{y}]$$

is used as follows

$$\cos(\mid \psi(t_{i+1}) - \psi(t_i) \mid) =$$

$$\min\left\{1, \frac{v_x(t_i)(x(t_{i+1}) - x(t_i)) + v_y(t_i)(y(t_{i+1}) - y(t_i))}{\sqrt{(v_x(t_i))^2 + (v_y(t_i))^2}}\right.$$

$$\left. \cdot \frac{1}{\sqrt{(x(t_{i+1}) - x(t_i))^2 + (y(t_{i+1}) - y(t_i))^2}}\right\} \tag{4.2}$$

For the sign of the turn angle, the cross product is used. Specifically if $y(t_{i+1})v_x(t_i) - x(t_{i+1})v_y(t_i) > 0$, then the change in turn angle is negative, $\psi(t_{i+1}) -$

$\psi(t_i) = -|\psi(t_{i+1}) - \psi(t_i)|$. This is a "left" turn, and is achieved by rolling the aircraft with roll angle, $\phi = -45°$, with the left wing down. Similarly if $y(t_{i+1})v_x(t_i) - x(t_{i+1})v_y(t_i) < 0$, then the change in turn angle is positive, $\psi(t_{i+1}) - \psi(t_i) = |\psi(t_{i+1}) - \psi(t_i)|$. This is a "right" turn, and is achieved by rolling the aircraft with roll angle, $\phi = 45°$, with the right wing down.

Note that there could be cases where there is no turn. This happens if the turn angle is less than a small amount specified in the OEP. Also, there could be cases where the turn time is greater than or equal to time between waypoints $t_{i+1} - t_i$. In this case, the OEP model assumes turning for the entire segment.

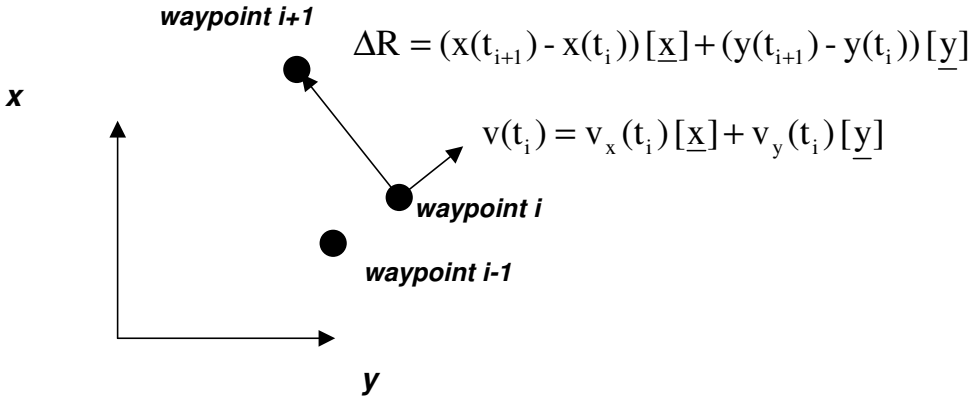

Figure 4.1: The turn angle is computed from the velocity vector and the vector between the destination and present waypoints.

4.2.2 Probability of Detection

The second component of the model is the detection model. This model computes the probability of detection of a UAV by an opponent SAM radar. The inputs of the detection model are the outputs of the aircraft model, specifically,

the aircraft position and attitude. The positions of opponent radars are also input if the aircraft is within an engagement range of the radar. For each radar, the signature and probability of detection of the aircraft is included in the output of the detection model. Signature is an intermediary variable that is related to radar cross section.

For each radar, a vector from the aircraft to the radar is transformed to aircraft body axis. This vector is then transformed to spherical coordinates for azimuth (Az), elevation (El) and range (R).

The signature is computed with a table look-up as function of azimuth and elevation. The tables used in this paper are shown in Table 1 and Table 2. Note that the signature model has sharp changes between azimuths magnitudes between 30^o and 31^o and also between 159^o and 160^0. When the nose of the aircraft points directly to the radar or directly away from the radar, and the elevation magnitudes are relatively low, there is a region of low signature. The probability of detection is computed as a function of signature and range using another table look-up. For the probability of detection model, note that probability of detection increases with increasing signature and decreasing range. To determine probability of detection as a function of signature and range, interpolation is used. The maximum and minimum probability of detection are .99 and .01, respectively.

In the OEP, a random number, uniformly distributed between 0 and 1 is generated. If this number is greater than or equal to the probability of detection, the UAV is detected. If the generated number is less than the probability of detection, the UAV is not detected. We assume that the problem is deterministic, and that a detection occurs if the probability of detection is greater than a threshold

Table 4.1: Signature is a function of azimuth and elevation

		Az					
		0	± 30	± 31	± 159	± 160	± 180
El	90°	1e0	1e0	1e0	1e0	1e0	1e0
	45°	5e-3	5e-3	1e0	1e0	5e-3	5e-3
	20°	5e-4	5e-4	5e-1	5e-1	5e-4	5e-4
	0°	5e-5	5e-5	5e-1	5e-1	5e-5	5e-5
	−20°	5e-4	5e-4	5e-1	5e-1	5e-4	5e-4
	−45°	5e-3	5e-3	1e0	1e0	5e-3	5e-3
	−90°	1e0	1e0	1e0	1e0	1e0	1e0

value.

4.2.3 Lock Loss

A mixed logical dynamical (MLD) representation of the lock loss model is presented here and shown in figure 4.2. Define four states: disengage, engage, launch and damage. The integer, binary variables δ_e, δ_{LL}, δ_{launch}, δ_{damage} are used to signal events that transition the opponent SAM radar system from one state to another. When $\delta_e = 1$, the opponent SAM engages the UAV. The variables δ_{LL}, δ_{launch}, δ_{damage} are used in conjunction with timers described below for lock loss, launch and time-to-target. When a timer surpasses a threshold value, a change in state is triggered. When the lock loss timer exceeds threshold T_{LL}, the variable δ_{LL} is set to 1 and the state transitions to disengage. When the engage timer exceeds $Teng$, the variable δ_{launch} is set to one, indicating that the SAM is

Table 4.2: Probability of Detection Table: This table relates signature, range in km, and probability of detection.

		Probability of Detection			
		.99	.5	.1	.01
Signature	1	380	481.2	555.6	656.6
	1e-1	213.7	270.6	312.5	369.2
	1e-2	120.2	152.2	175.7	207.6
	1e-3	67.6	85.6	98.8	116.8
	1e-4	38	48.1	55.6	65.7
	1e-5	21.4	27.1	31.2	36.9
	1e-6	12	15.2	17.6	20.8

launching a missile at the UAV. When the time-to-target timer exceeds T_{target}, the variable δ_{damage} is set to one, indicating that the missile has had enough time to reach the target. These conditions are detailed by the following logical equations described below:

$$\begin{aligned} \delta_e &= 1 \leftrightarrow \text{engagement conditions met} \\ \delta_{LL} &= 1 \leftrightarrow x_{LL} \geq T_{LL} \\ \delta_{Launch} &= 1 \leftrightarrow x_{eng} \geq T_{eng} \\ \delta_{damage} &= 1 \leftrightarrow x_{launch} \geq T_{target} \end{aligned} \tag{4.3}$$

The variables x_{LL}, x_{eng}, and x_{launch} are the timer variables, and are updated using the equations below.

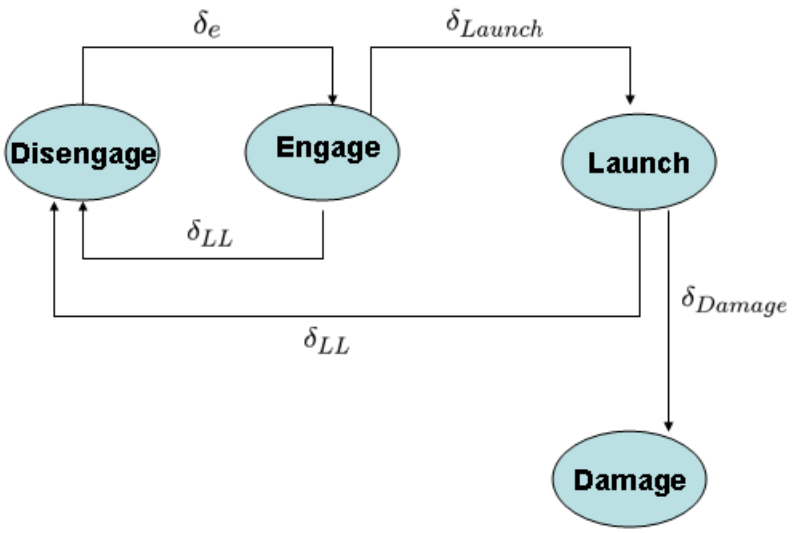

Figure 4.2: Lock Loss four state model.

A clock is defined with the equation below.

$$\begin{aligned} t(j+1) &= t(j) + T_s \qquad j = 0, 1, 2, ... \\ t(0) &= 0 \end{aligned} \tag{4.4}$$

Where T_s is the sample time.

For the lock loss timer, if there is no detection, the lock loss timer is incremented. If there is a detection, the lock loss timer is reset to zero. The lock loss threshold, T_{LL}, in (4.3) is specified in the OEP model.

$$x_{LL}(t(j+1)) = x_{LL}(t(j))(1-\delta_e) + (1-\delta_e) \tag{4.5}$$

The engage timer increments when the system is engaged and is reset to zero if there is a lock loss event.

$$x_{eng}(t(j+1)) = (x_{eng}(t(j)) + \delta_e)(1-\delta_{LL}) \tag{4.6}$$

Similarly, the launch timer increments when a launch has occurred and is reset to zero if there is a lock loss event.

$$x_{launch}(t(j+1)) = (x_{launch}(t(j)) + \delta_{launch})(1 - \delta_{LL}) \tag{4.7}$$

4.3 MILP Problem Formulation

In this section, a MILP representation of the problem is presented. The first subsection pertains to developing a vehicle dynamics model that is suitable for the MILP approach. The second subsection addresses the detection model, and the third pertains to the lock loss model.

In order to represent the model described in the previous section as a Mixed Integer Linear Programming (MILP) problem, several simplifications are made. The MILP formulation, like the model, assumes the UAV maintains constant altitude. In addition, the bank angle of the UAV is always zero. This simplification is valid if the turn time is small compared to the size of the discrete time steps used in MILP.

4.3.1 Dynamics Constraints

The total flight time is represented throughout the formulation as T discrete time steps, which is a parameter the user specifies in the input data file. The UAV is given T time steps to reach the desired end point.

There are T values for the UAV's x, y position. These T values are indexed

with variable k. At any individual time step, k, the UAV's position at that time step, $x[k]$ and $y[k]$, is calculated to be the sum of the starting x and y position, called x_0 and y_0, plus each velocity component, $v_x[k]$ and $v_y[k]$ multiplied by the length of time of each time step, which is a constant T_s, for all values of k from zero to the present time step. Written as equations this gives:

$$x[k] = x_0 + \sum_{j=0}^{k} (v_x[j] \cdot T_s) \tag{4.8}$$

$$y[k] = y_0 + \sum_{j=0}^{k} (v_y[j] \cdot T_s) \tag{4.9}$$

Velocity components, v_x and v_y, must be constrained so that the magnitude of the velocity vector is not greater than the maximum velocity of the UAV, v_{max}.

Calculating the total velocity, using v_x and v_y results in a nonlinear function, due to the square root. For a MILP representation a linearized approach must be taken to constrain v_x and v_y. One such approach, outlined in [27], is to constrain v_x and v_y separately to be less than some v_m.

$$| v_x | \leq v_m \quad \text{and} \quad | v_y | \leq v_m \tag{4.10}$$

Plotted on a v_x versus v_y graph, as in Fig. 4.3, these constraints create a box that is inscribed by the circle corresponding to all feasible v_x and v_y values. This box is a poor approximation to the circle.

To achieve a better approximation to the circle, more constraints can be added.

$$v_x[k] \sin\left(\frac{2\pi m}{M}\right) + v_y[k] \cos\left(\frac{2\pi m}{M}\right) \leq v_{max} \cos\left(\frac{\pi}{M}\right) \tag{4.11}$$

This constraint has to hold for all integer values of m in the set from 1 to M,

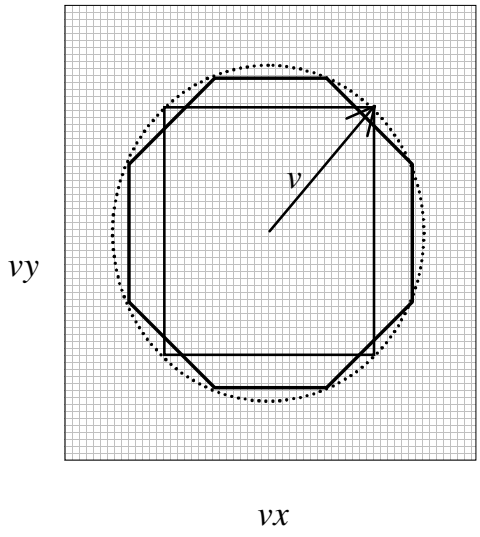

Figure 4.3: Polygon circle approximation for M = 4 and M = 8.

where M is the number or sides of the polygon approximation of the circle. Because the polygon approximation is inscribed by the circle, v_{max} must be multiplied by the cosine factor. Without this factor, the circle would be inscribed by the polygon and there would be v_x and v_y combinations allowed by the approximation, which combine to produce a v greater than v_{max}. This is easiest to see in the case where M equals four because there is a large difference between the box inscribed by the circle, and the box that inscribes the circle.

The above constraints only ensure that v_{max} is not violated. To find the actual velocity at any time step k, $v[k]$, the v_x and v_y components must be used. Normally, calculating the magnitude of a vector from its components relies on the nonlinear square root.

$$v[k] = \sqrt{(v_x[k])^2 + (v_y[k])^2} \tag{4.12}$$

A linear approximation uses three constraints and several binary variables, b_v. For any time step k and for all m in the range of 1 to M, where L is an arbitrarily large number:

$$v_x[k]\sin\left(\frac{2\pi m}{M}\right) + v_y[k]\cos\left(\frac{2\pi m}{M}\right) \leq v[k] \tag{4.13}$$

$$\left(v_x[k]\sin\left(\frac{2\pi m}{M}\right) + v_y[k]\cos\left(\frac{2\pi m}{M}\right)\right)\cdot 1.01 \geq v[k] - L(1 - b_v[k,m]) \tag{4.14}$$

and

$$\sum_{j=1}^{M} b_v[k,j] = 1 \quad , \quad b_v[k,j] \in \{0,1\} \tag{4.15}$$

The true value of $v[k]$, corresponding to the UAV velocity, will satisfy these condition for only one value of m. First equation 4.13 constrains $v[k]$ to be a value that is bigger than any possible combination of $v_x[k]$ and $v_y[k]$, for any value m. Equation 4.14 also constrains $v[k]$ to be smaller than any combination multiplied by 1.01, or else the corresponding binary variable, $b_v[k]$, must be zero. With no other constraints, many values of $v[k]$ could satisfy equations 4.13 and 4.14 by having all zero values of $b_v[k]$. To prevent this, equation 4.15 constrains $b_v[k]$ such that it must be non-zero for one m value. This m corresponds to one of the corners of the M sided polygon appoximation. When $b_v[k]$ does equal 1, equations 4.13 and 4.14 can only be satisfied with a $v[k]$ which is a close approximation to the actual velocity value.

The heading value is important in determining the radar signature of the aircraft. The heading angle is tracked with a variable which will later be constrained to ensure that a small enough cross section is presented to any radar to avoid detection. Given the model for UAV dynamics, it can be assumed that the UAV always faces the direction it is traveling. Therefore the heading is related to the velocity vector. In finding $v[k]$ there are m binary variables, $b_v[k]$, for each time step k. These variable all equal zero except for the one that corresponds to the m which relates how $v_x[k]$ and $v_y[k]$ combine to form $v[k]$. This value of m is directly related to the heading angle of the UAV using the following

$$\sum_{j=1}^{M} b_v[k, j] \cdot j \cdot \frac{360}{M} = \text{heading angle} \tag{4.16}$$

To force the UAV to go from its starting position to the final, or desired position, more constraints and binary variables are necessary. Again assuming that L is an arbitrary large value we can write the following:

$$x[k] - x_f \leq L \cdot (1 - b_f[k]) \tag{4.17}$$

$$x[k] - x_f \geq -L \cdot (1 - b_f[k]) \tag{4.18}$$

$$y[k] - y_f \leq L \cdot (1 - b_f[k]) \tag{4.19}$$

$$y[k] - y_f \geq -L \cdot (1 - b_f[k]) \tag{4.20}$$

and

$$\sum_{j=1}^{T} b_f[j] = 1 \quad , \quad b_f[j] \in \{ 0, 1\} \tag{4.21}$$

The first four constraints force $b_f[k]$ to be 0 unless the UAV is at the final location, x_f and y_f, in which case $b_f[k]$ can be 1. The final constraint says that when the binary variable b is summed over the entire time of the flight, it must equal 1. These constraints force the UAV to the final position.

None of the constraints presented so far force the UAV to reach the final position as quickly as possible. To do this the final step is to create a definition for what makes a path optimal. This is done with MILP by creating a metric that characterizes each path and then either minimizing or maximizing that metric. Note that in the constraints to force the UAV to the final position, $b_f[k]$ only equals 1 when the UAV is at the final position and that k is a time step index. The optimal path is one for which the k is the smallest. This can be represented by defining the cost as:

$$\sum_{k=1}^{T} \left(b_f[k] \cdot \sum_{1}^{k} T_s \right) = \text{cost} \tag{4.22}$$

Thus the optimal path is the one that satisfies all other constraints and has the lowest cost.

4.3.2 SAM Constraints

Given a model for a SAM's ability to detect a UAV, several more constraints can be placed on the UAV's path to reduce the probability of detection to a predetermined acceptable level.

As described in the model, two important parameters in determining a SAM's impact on the UAV's path is the distance between the two, and the UAV's signature to the SAM's radar. The calculations in the model for distance and azimuth are nonlinear. The following MILP formulation is an equivalent linear approximation.

Distance between a UAV and SAM is an important factor in determining the probability of detection. The distance can be found, and then constrained, in

a similar fashion as maximum velocity, using constraints and binary variables. For any time step k, where L is an arbitrarily large number the distance, $dist[k]$, is constrained as follows,

$$x[k]\sin\left(\frac{2\pi m}{M}\right) + y[k]\cos\left(\frac{2\pi m}{M}\right) \leq dist[k] \tag{4.23}$$

$$\left(x[k]\sin\left(\frac{2\pi m}{M}\right) + y[k]\cos\left(\frac{2\pi m}{M}\right)\right) \cdot 1.01 \geq dist[k] - L(1 - b_d[k,m]) \tag{4.24}$$

and

$$\sum_{j=1}^{M} b_d[k,j] = 1 \quad , \quad b_d[k,j] \in \{0,1\} \tag{4.25}$$

$x[k]$ and $y[k]$ are combined so that they are less than or equal to $dist[k]$ yet are greater than or equal $dist[k]$ when multiplied by 1.01. This must hold for all values of m, where m ranges from 1 to M. The binary variables are used to pick out the one true value of distance that makes the two constraints true.

Another factor in determining whether a SAM detects a UAV is if the UAV is nose in or nose out relative to the SAM. Nose in refers to when the UAV's exposure to radar is minimal. Nose out refers to when the exposure is greater. This is determined by comparing the UAV's heading angle and the line of sight (LOS) angle between the SAM and the UAV. The LOS angle can be found from the x, y, and distance in the same way that heading was found from v_x, v_y, and velocity. Using equations 4.25, 4.26, and 4.27 to find the non-zero value of $b_d[k]$ the line of sight angle is given by

$$\sum_{j=1}^{M} b_d[k, j] \cdot j \cdot \frac{360}{M} = \text{Line of Sight angle} \tag{4.26}$$

With the line of sight angle and the previously calculated heading angle, the azimuth angle at any time step k, can be found by comparing the two. The line of sight angle and the heading angle are each one of M discrete values, where M is the size of the polygon approximation. A static M by M table can be used to find the azimuth angle as a function of both the line of sight angle and the heading angle. To illustrate, consider if the UAV has a heading of 0°, heading north, the azimuth table entry would be 180° if the line of sight value is also zero; the UAV is north of the SAM and heading away. However if the line of sight value is 180°, than the azimuth angle would be 0°, the UAV is south of the SAM and heading directly towards it. To access the correct table entry binary variables b_{los} and b_h are used as indices.

The next factor to consider is elevation angle between the UAV and the SAM. Because a constant altitude is assumed, the elevation angle is simply a function of range.

$$dist \cdot \tan(\text{elevation}) = \text{altitude} \tag{4.27}$$

However tangent is a non-linear function. Because the elevation table given in the model for SAM detection is discrete, a discrete linear approximation can be used for the tangent function in this case. For any time step, k, an index, d, that ranges from 1 to the size of the tangent approximation array, and the binary variables b_{el} can be used to create the following constraints:

$$alt \leq dist[k] \cdot el_{table}[d+1] + M \cdot (1 - b_{el}[k,d]) \tag{4.28}$$

$$alt \geq dist[k] \cdot el_{table}[d] - M \cdot (1 - b_{el}[k,d]) \tag{4.29}$$

and

$$\sum_{d=1}^{el_{size}} b_{el}[k,j] = 1 \quad , \quad b_{el}[k,j] \in \{0,1\} \tag{4.30}$$

$$\sum_{d=1}^{el_{size}} b_{el}[k,d] \cdot d \cdot \frac{180}{el_{size}} = el[k] \tag{4.31}$$

Using elevation, azimuth, and acceptable probability of detection set by the user, the tables given in section 2.2 can be used to find the minimum safe distance between the UAV and the SAM radar. In the MILP formulation the two tables given in section 2.2 are combined into one three dimensional table. First the elevation angle is used to find the correct azimuth versus probably of detection sub-table. Then the azimuth variable is used with the user supplied, discrete, probability of detection threshold variable, to look up the minimum distance. The final MILP constraint is that the UAV distance from the SAM is greater than this look up distance. This constraint is consistent with the assumption that if a path violates the probability of detection threshold, a detection is assured. This simplification is necessary to eliminate the random element of SAM radar detection. Due to this assumption, for any path to be valid it must be true that for any given time the distance between the UAV and the SAM is greater than the look up distance based on all the factors.

4.3.3 Lock-Loss Constraint

The MILP formulation of lock loss assumes that the first time the UAV is detected, the lock loss timer is activated. During the lock loss time if the vehicle continues to be detected, a SAM is fired and the UAV is destroyed. If the UAV is not detected for as long as the lock loss time, than the engage timer is reset and it takes another detection to start it again. This means that a UAV can take advantage of lock loss by flying a path that may allow it to be detected, but not for longer than the lock loss time limit. In MILP this can be specified using binary variables. Suppose the lock loss time is 4 time steps. The distance constraint is changed to include b_{ll}, binary variables for the lock loss such that at any time step k and for L being an arbitrary large number:

$$dist[k] + (L \cdot b_{ll}[k]) \geq \text{table value for distance} \tag{4.32}$$

If the distance at time k, is less than the table value for distance, this constraint can only be true if $b_{ll}[k]$ is 1, otherwise it can be 0. To ensure lock loss the final constraint is added, assuming the lock loss time is 4:

$$\sum_{j=0}^{3} b_{ll}[k+j] = 3 \quad , \quad b_{ll}[k+j] \in \{0, 1\} \tag{4.33}$$

Which states that for any time, k, and the next 3 consecutive time steps, b_{ll} must be equal to zero once. This can only happen if for any 4 consecutive time steps, the distance between the UAV and the SAM is greater than the table value at least once. This ensures that the UAV is not detected for any longer than the lock loss time.

4.4 Implementation

The AMPL modeling language is well suited for representing the optimizations presented above. With AMPL two files are used, the model file details the various constraints. Another file holds the model data. Thus data can be changed, such as starting point or UAV velocity, without changing the constraints. The optimization problem coded in the two AMPL files is solved with ILOG CPLEX [36]. The output values are saved to a third file, which can be opened in Matlab for data analysis. The following examples were solved on a 1.8 GHz Pentium II computer with 512 MB of RAM, running the Windows XP operating system.

4.5 Examples

4.5.1 Single UAV vs. Single SAM

Using the data in the tables presented in Tables I and II a single UAV was started at position x_o = 350 and y_o = 125. The UAV was given a final destination of x_f = 340 and y_f = 0. A single SAM was placed at the origin and an acceptable probability of detection of 0.5 was used. The MILP formulation of this path planning problem contains 15363 binary variables, 72 linear variables and 10359 linear constraints. As shown in figure 4.4, the UAV does not fly directly to the final position, but instead must stay far enough away from the SAM until it reaches a position where it can turn towards the SAM site and approach the final position. The circle in the figure represents the minimum distance the UAV must maintain if it has a high signature, corresponding to a nose out orientation.

The circle appear elliptical due to the unequal scaling of x and y. The UAV only turns toward the destination when doing so will provide a small cross section to the SAM's radar, allowing it to get closer without violating the acceptable probability of detection.

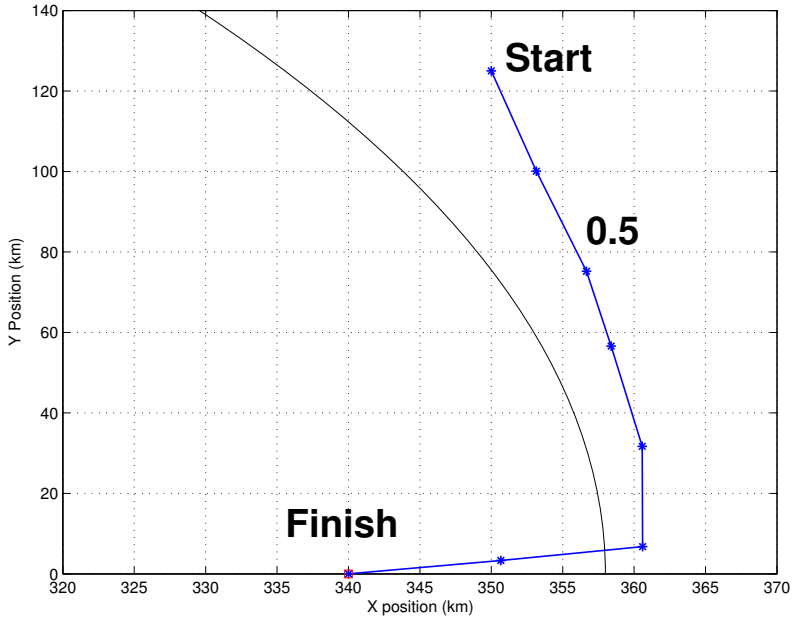

Figure 4.4: UAV path, SAM at origin and Probability of Detection = 0.5

For size comparison, a two UAV scenario, with the second UAV at x_o = -350 and y_o = 125, resulted in a MILP formulation containing 25037 binary variables, 116 linear variables, and 14354 linear constraints.

4.5.2 Effects of Probability of Detection

By varying the acceptable probability of detection, a more direct path plan can be generated. Two additional path plans were calculated using all the same parameters as the example given above, except that the probability of detection

threshold values of 0.6 and 0.7 were used. All three paths are plotted in figure 4.5, which clearly shows that when a higher risk of detection is acceptable, the UAV can use a more direct and risky path. The additional circles correspond to the smaller minimum distance for high signature approaches. The smaller the probability threshold, the closer the UAV can be to the SAM at the origin while presenting it with a high signature.

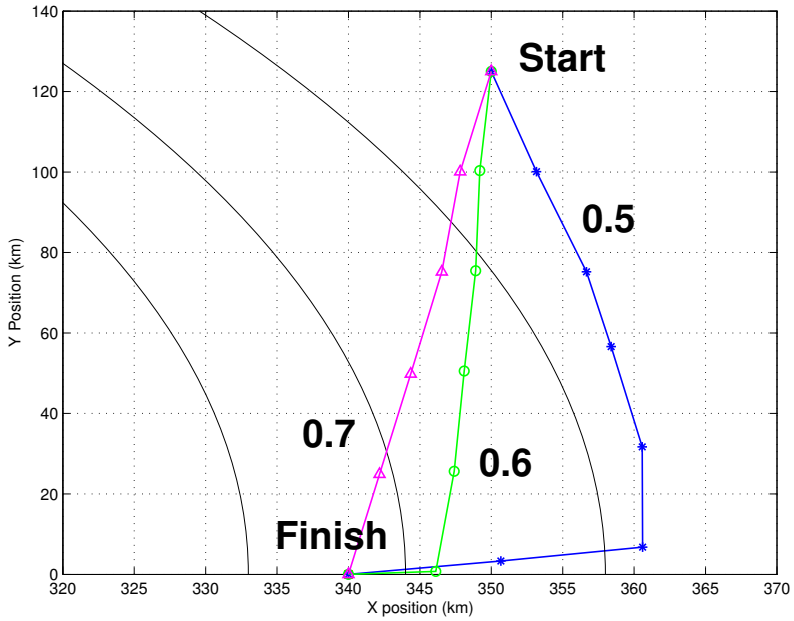

Figure 4.5: UAV paths, SAM at origin and Probability of Detection = 0.5, 0.6, and 0.7

4.5.3 Calculation Time

Each path given in figure 4.5 took approximately 3 minutes to compute. This time was needed by the CPLEX solver to find all feasible solutions. Due to the large and complex nature of the MILP formulation there are many branches for the solver to traverse in search of the optimal solution. While not fast enough

for realtime applications, it should be noted that this calculation time is greatly influenced by the computational capabilities of the computer used to solve the MILP. Realtime application may be possible with a computer optimized to solve CPLEX problems.

4.5.4 Effects of Lock Loss

Shown in figure 4.6 is the same path plan from figure 4.4, with a 0.5 probability of detection. Also shown is another path that takes advantage of the lock loss, where the lock loss time is set to four time steps. This means as long as the UAV is not detected for four consecutive time steps, the UAV will not be destroyed. Due to the extra complexity that the lock loss modeling adds to the MILP formulation the computation time increased to approximately 3.5 mintues.

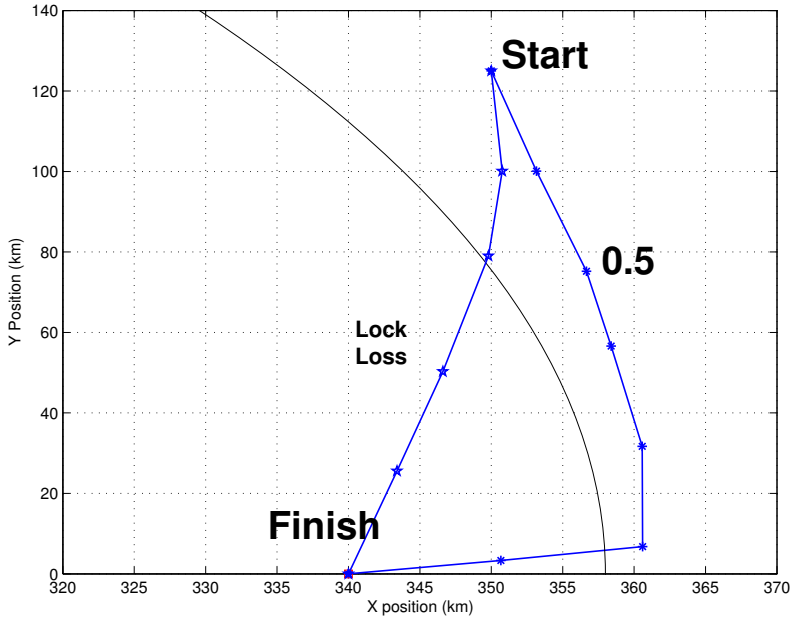

Figure 4.6: UAV paths, lock loss and non-lock loss with Probability of Detection = 0.5, SAM at origin.

4.6 Conclusion

As the use of UAVs increases, so does the importance of efficient path plans. In complex situations, such as the threat of a SAM, finding a feasible path plan can be difficult, knowing it is also optimal, even more so. The MILP approach outlined in this paper finds the most efficient path. Due to the computation time, the approach outlined above is best suited as a top level path planning algorithm. It may also be possible to use this algorithm in a receding horizon fashion, where new path plans are generated for shorter intermediary waypoints and recalculated as new sensor information is available.

The MILP approach given in this paper does not require the path planning to be computed by the vehicle itself. One possible scenario could entail using a computational facility to solve the MILP and calculate a path plan that is then sent over a communications link to a UAV.

CHAPTER 5

PATH GENERATION USING MATRIX REPRESENTATIONS OF PREVIOUS ROBOT STATE DATA

5.1 Introduction

Humans do not move by calculating on a conscious level the physics behind motion. Humans do not throw a ball by calculating a force and angle, taking into account the effects of gravity. Humans learn by repetition and using past experience as a template to create input-output mappings. It is possible for robots to act in a similar fashion. Instead of generating paths by performing complex computations, a robot can use past travel experiences. Such a memory based approach does not rely on developing an accurate physics or dynamics model of the robot and its environment. In the same way that a human does not calculate the moment of inertia of an arm swinging around a shoulder joint, a robot does not need to calculate how a control action will affect its dynamics. In both cases, by using past experiences, the result of an action can be predicted without calculation

An example of this approach can be seen by examining how a human learns to throw a ball at a target. The first time they are presented with this challenge, it takes many tries to learn the sequence of muscle contractions that allow the ball to travel with the correct trajectory. Once they have learned to throw a ball with satisfactory results, the appropriate body motions, or control inputs, are remembered. If given a heavier ball to throw, a human simply makes adjustments to the remembered body motion from their previous experience, instead of re-learning how to throw all over again. While it may take a few tries to learn

how to hit the target with the heavier ball, they will learn this new task much quicker than before because they have their past experience to rely on. Similarly for path planning, if a robot has successfully traversed a path in the past, there is a way to encode this experience to allow future use. Given the task of generating a new path plan which is similar, but not identical, to a previous path, a robot can use past experience to quickly form a solution.

As with humans, there is no guarantee that this solution will be optimal, but it need not rely on heavy mathematical modeling or calculation.

5.1.1 Hierarchical Bayesian Model

There are several theories on how the human brain and nervous system function to allow fluid motion and learning from past exerience. A recent one is presented by Jeff Hawkins in his book "On Intelligence." [37] In the book, it is theorized that the human brain creates and remembers series of sequences that are associated together at each level of the brain. This approach can be represented with a Hierarchical Bayesian Network model. Such a model was used to demonstrate an effective image recognition system, using training images to teach the network what different images are, and how they may vary in size, shape, and orientation, and yet still be the same type [38]. This approach not only accurately recognized varying images, but was able to fill in holes in test images to make them better match the type of image that they were categorized as [39].

5.1.2 Simplicity & Speed

This approach shows great promise in illustrating how the brain may function. The formulation relies on Bayesian belief propagation equations which use message passing between nodes to propagate beliefs [40]. Unfortunately this extensive message passing and the numerous required matrices make it ill-suited for real time use with present computer technology. Because silicon based computers work in a fundamentally different way than neuron based organic brains, they can not implement algorithms in the same way and at the same speed. For real-time path planning, a different approach is required that still encompasses the same idea of utilizing past experience. Presented below is a system that uses matrices to capture past experience, but runs fast enough to enable real time path planning. For purposes of this research, a path is defined as traveling from a start point to an end point and finishing with a specific orientation without regard to obstacle or terrain issues. This system approach benefits from the concept of associating various inputs to form representations of past experiences. However by using fewer matrices and fewer levels, it does not suffer from the extensive message passing and speed problems of Bayesian networks.

5.1.3 Outline

In the next section, the system configuration is described with a discussion of an appropriate coordinate frame. The assumed robot dynamics are given, as is a description of the discretization used. A simple example illustrates the fundamentals of the approach. In the third section, results are given and discussed. This section includes a description of the real world system used and several

plots of both the training paths, and the paths generated by the approach. Concluding remarks and ideas for future work are presented in section four.

5.1.4 Robot Kinematics

The discussion below assumes a traditional kinematic unicycle robot, one which can only travel in the direction it is heading. In cartesian coordinates, the dynamics of such a robot are given by the following equations:

$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$\dot{\theta} = \omega$$

Here v and ω are the control variables.

5.2 System Configuration

The system must have previous path data before being able to generate new paths. Previous path data is composed of robot state values along a viable path. This data is contained in matrices which are then used to associate incoming robot state data with the closest state data stored from any previous path. The control effort from this stored closest state data is then applied as the present control effort. After exerting this control effort, a new set of robot state information is taken and again compared against the matrix. This process is explained in greater detail below.

5.2.1 Coordinate Frame

As a robot moves from a start position to an end position, it creates a path connecting the two. At each point along the path, the robot has a specific state. This state is composed of the robot's position, orientation, and the commanded control effort used to move the robot along the path. The position state can be given in various coordinate frames. For this research a world coordinate frame fixed at the end point is best. With such a coordinate frame, the robot's present position is described relative to the ending point, specifically the distance away and the difference between present and desired robot heading. By using an end point centered coordinate frame, relating the robot state to the end state, the same dynamics are true even if the absolute position of the robot changes, as long as the relative position of the robot to the endpoint does not.

This assumes a uniform environment without obstacles or varying terrain, otherwise the dynamics of a robot traveling between two points may change. A robot will travel a different path between two points if the points are translated to a location where an obstacle falls between the points. Given a uniform environment, translating or rotating a path does not change the robot's position states or its dynamic states along the path.

The control efforts at each point are the commands used by the robot to move itself at that point in the path. The form of the control commands depend on the specifics of the robot. At the lowest level, the robot's controller may produce exact voltage values to be sent to each motor. Alternatively, a very high level controller may just determine way points to the final destination. An intermediate level is one where the robot determines a desired velocity vector to travel. This velocity vector is composed of a magnitude and an angle relative to the

robot's present heading, θ.

Each point along a path then has five state values associated with it as illustrated in figure 5.1. In the figure, distance is represented with d, heading θ, heading offset $\psi = \theta - \psi_f$, commanded velocity magnitude v, and velocity rotation ω. As shown in the figure, ψ_f is ninety degrees, which means the robot should reach the end point facing up.

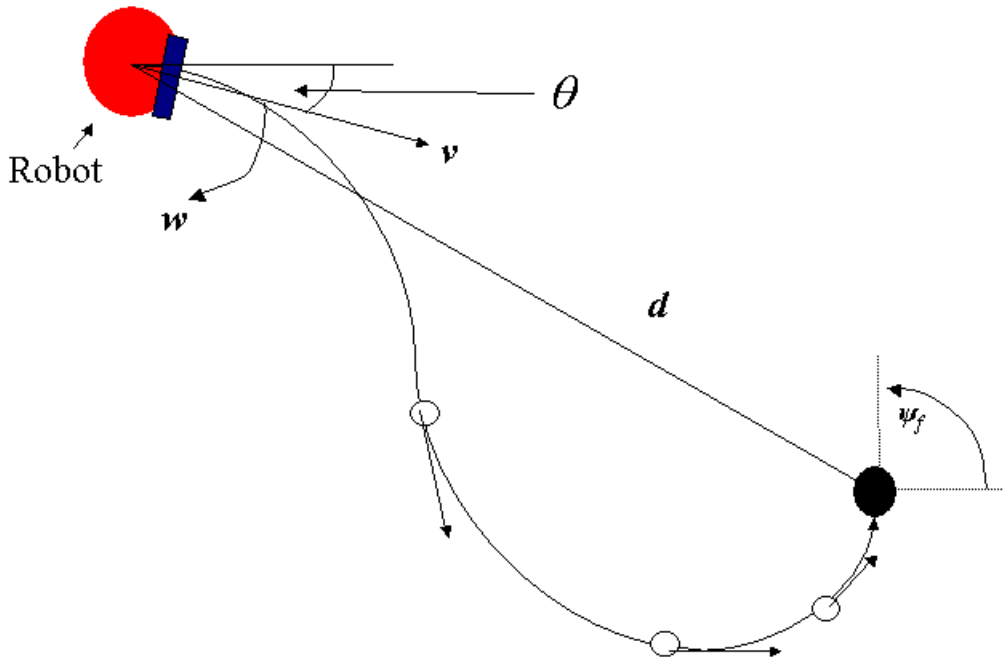

Figure 5.1: At any point along a path a robot has five state values, distance d, heading θ, heading offset ψ, velocity v, and velocity angle ω.

5.2.2 Training Data Discretization

Several viable robot paths are needed. These training paths are analogous to throwing a ball several times to learn how to throw it to hit a target. To keep the training data a finite size, different points along the length of each path must be selected. At each point, the complete vehicle state (position d, heading θ, heading offset ψ, velocity magnitude v, and velocity rotation ω) are recorded. If there are 5 training paths with 100 points on each path, that would give 500 distinct robot states. For each parameter a discretization scheme is needed. For the angles this is straight forward as each angle can vary from +pi to -pi. The range of possible values can be discretized to the desired fidelity. A coarse discretization would be $[-\pi, \frac{-3\pi}{4}, \frac{-\pi}{2}, \frac{-\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}]$. A finer discretization would be $[-\pi, \frac{-9\pi}{10}, \frac{-8\pi}{10}, ..., 0, ..., \frac{8\pi}{10}, \frac{9\pi}{10}]$. For both distance and velocity magnitude values, a discretization would have to be chosen that spans the range of values of all the distinct robot state values. Once discretization schemes are selected for each parameter, the robot states are processed so that each parameter in a state is rounded to the nearest discretization value for that parameter.

Example - State Discretization

In the following example we have three training paths with three path points each. The states at a path point are represented as $p_{path\#,point\#} = [d, \theta, \psi, v, \omega]$

Path 1

$$
\begin{aligned}
p_{1,1} &= [3.2, -\pi, -\pi, 2, -\pi] \\
p_{1,2} &= [2.4, \frac{-6.2\pi}{10}, \frac{-8.8\pi}{10}, .9, \frac{-8.6\pi}{10}] \\
p_{1,3} &= [1.1, \frac{-\pi}{10}, \frac{-5.7\pi}{10}, .2, \frac{-5.8\pi}{10}]
\end{aligned}
$$

Path 2

$$
\begin{aligned}
p_{2,1} &= [3.6, \frac{5\pi}{10}, 0, 2.6, 0] \\
p_{2,2} &= [2.1, \frac{2\pi}{10}, \frac{-3.3\pi}{10}, 1.7, \frac{-3.4\pi}{10}] \\
p_{2,3} &= [0.8, 0, \frac{-4.9\pi}{10}, 0.3, \frac{-4.8\pi}{10}]
\end{aligned}
$$

Path 3

$$
\begin{aligned}
p_{3,1} &= [2.9, \frac{-7.7\pi}{10}, \frac{\pi}{10}, 2.2, \frac{\pi}{10}] \\
p_{3,2} &= [1.5, \frac{-3\pi}{10}, \frac{-4.3\pi}{10}, 1.3, \frac{-4.2\pi}{10}] \\
p_{3,3} &= [0.2, 0, \frac{-5\pi}{10}, 0.4, \frac{-4.9\pi}{10}]
\end{aligned}
$$

Now discretize the state values such that the permissible values are

$$
\begin{aligned}
d &= [0, 1, 2, 3, 4] \\
\theta &= [-\pi, \frac{-9\pi}{10}, \frac{-8\pi}{10}, ..., 0, ..., \frac{8\pi}{10}, \frac{9\pi}{10}] \\
\psi &= [-\pi, \frac{-9\pi}{10}, \frac{-8\pi}{10}, ..., 0, ..., \frac{8\pi}{10}, \frac{9\pi}{10}] \\
v &= [0, 0.5, 1, 1.5, 2, 2.5, 3] \\
\omega &= [-\pi, \frac{-9\pi}{10}, \frac{-8\pi}{10}, ..., 0, ..., \frac{8\pi}{10}, \frac{9\pi}{10}]
\end{aligned}
$$

Using this discretization the path points become

Path 1

$$
\begin{aligned}
p_{1,1} &= [\mathbf{3}, -\pi, -\pi, 2, -\pi] \\
p_{1,2} &= [2, \frac{-6\pi}{10}, \frac{-9\pi}{10}, 1, \frac{-9\pi}{10}] \\
p_{1,3} &= [1, \frac{-\pi}{10}, \frac{-6\pi}{10}, 0, \frac{-6\pi}{10}]
\end{aligned}
$$

Path 2

$$
\begin{aligned}
p_{2,1} &= [4, \frac{5\pi}{10}, 0, 3, 0] \\
p_{2,2} &= [2, \frac{2\pi}{10}, \frac{-3\pi}{10}, 2, \frac{-3\pi}{10}] \\
p_{2,3} &= [1, 0, \frac{-5\pi}{10}, 0, \frac{-5\pi}{10}]
\end{aligned}
$$

Path 3

$$
\begin{aligned}
p_{3,1} &= [3, \frac{-8\pi}{10}, \frac{\pi}{10}, 2, \frac{\pi}{10}] \\
p_{3,2} &= [2, \frac{-3\pi}{10}, \frac{-4\pi}{10}, 1, \frac{-4\pi}{10}] \\
p_{3,3} &= [0, 0, \frac{-5\pi}{10}, 0, \frac{-5\pi}{10}]
\end{aligned}
$$

As will be shown below, this discretization and enumeration of the path parameter states can now be used to assign control effort based on new position inputs.

5.2.3 Input Matrix & Processing

New paths can be generated by taking input parameters and associating them with previous path points. The input parameters are the robot position states,

d, θ, ψ. For each of these position states, an input matrix is created. These input matrices have a row for each possible robot state as described above, and one column for each discrete value of either d, θ, or ψ. For the example given above, the three paths have three points each, so there are nine robot states. The discretization of d has five values, so the input matrix for d would be nine by five. The input matrix is filled by placing a 0.5 in the column of each row that corresponds to the exact d, θ, or ψ value that composes that state. A 0.25 is then placed in the columns adjacent to this column. This is done to account for the possibility that a specific state may correspond to slightly different d, θ, or ψ values. For the d input matrix, the first row would correspond to the first point on Path 1, which has a discretized value of 3. Thus the first row of the d input matrix, M_d, would be [0 0 .25 5 .25].

Once these matrices are constructed they can be used to generate control commands for the robot. The first step is to measure the d, θ, ψ for a robot that is trying to get to the final position. A column vector is created with the length of the discrete parameter values and contains two values associated with how close the measurement is to the discrete values. For instance, using the possible distances from the example above, d = [0 1 2 3 4], if a distance of 3.2 is measured, the associated column vector would be $[0\ 0\ 0\ 0.8\ 0.2]^T$. This column vector reflects that the measured value is between 3 and 4, but closer to 3. A matrix multiplication is then done between the input matrix and this column vector. The resulting row vector, λ_d, has the length of the number of possible states. A corresponding row vector, λ_θ and λ_ψ are created in the same way for the measured θ and ψ.

5.2.4 Control Output

The lambda row vectors are combined by doing an element by element multiplication to create one lambda. The largest element is found and this value corresponds to the most likely robot state associated with the inputs d, θ, and ψ. The velocity magnitude and rotation velocity with this state are then used as the control outputs to the robot.

Example - Creating Control Outputs from State Inputs

Continuing with the example given above, there are 9 path points and distance is discretized as d = [0, 1, 2, 3, 4]. The input matrix for distance has a row for each path point, p. Using the method described earlier, column values for each row are assigned. The input matrix for the example above would be

$$M_d = \begin{bmatrix} 0 & 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0.25 & 0.5 & 0.25 & 0 \\ 0.25 & 0.5 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0.25 & 0.5 \\ 0 & 0.25 & 0.5 & 0.25 & 0 \\ 0.25 & 0.5 & 0.25 & 0 & 0 \\ 0 & 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0.25 & 0.5 & 0.25 & 0 \\ 0.5 & 0.25 & 0 & 0 & 0 \end{bmatrix}$$

Multiplying M_d by the column vector $[0\ 0\ 0\ .8\ .2]^T$, which was derived from the input d = 3.2, gives a λ_d of $[0.45\ 0.2\ 0\ 0.3\ 0.2\ 0\ 0.45\ 0.2\ 0]^T$;

In a similar fashion a M_θ and M_ψ are created and multiplied by the column vector derived from an inputs θ and ψ to produce a λ_θ and λ_ψ.

If $\lambda_\theta = [0\ 0\ 0\ 0.3\ 0.2\ 0.1\ 0.4\ 0.3\ 0.1]^T$ and $\lambda_\psi = [0\ 0\ 0\ 0.3\ 0.1\ 0.1\ 0.5\ 0.2\ 0.1]^T$, then multiplying λ_d element by element by λ_θ and λ_ψ gives $[0\ 0\ 0\ 0.027\ 0.004\ 0\ 0.09\ 0.012\ 0]^T$.

Because the seventh element, 0.09, is the largest the seventh path position, in this case $p_{3,1} = [3, \frac{-8\pi}{10}, 2, \frac{\pi}{10}]$, is chosen and the control efforts are $v = 2$ and $\omega = \frac{\pi}{10}$. These control commands are used by the robot and a new input d, θ, and ψ are measured. The process repeats to generate the next set of control efforts. Note that there is no restriction that consecutive control efforts must come from the same previous path.

5.3 Real World and Simulation Results

The approach was validated through both simulated and real world implementation. The simulation results are given below and were generated with Cornell University's Robocup simulator. Cornell's Robocup system was used for real world testing.

5.3.1 Robocup System

Cornell's Robocup team has competed in the RoboCup Small-Sized League since 1999 and has won the international championship four times. The system consists of a playing field, robots, an overhead vision system, and computer

hardware and software [41]. The availability and sophistication of the Cornell Robocup infrastructure made it ideal for testing this new path generation approach. The Robocup robots can travel omni-directionally, able to move in a direction other than the one they are presently facing, which is an important competitive ability. For the purposes of this research, the robots were made to behave in a kinematic unicycle manner.

5.3.2 Simulation

As part of the Robocup system, the Cornell team developed a Robocup simulator [42]. Figure 5.2 shows a partial screen capture of the simulator. Note that in this picture only one robot is on the field, Robot 0 of the blue team, the rest are on the sideline at the the top of the screen. Robot 0 is below the ball at the center of the field and is facing up.

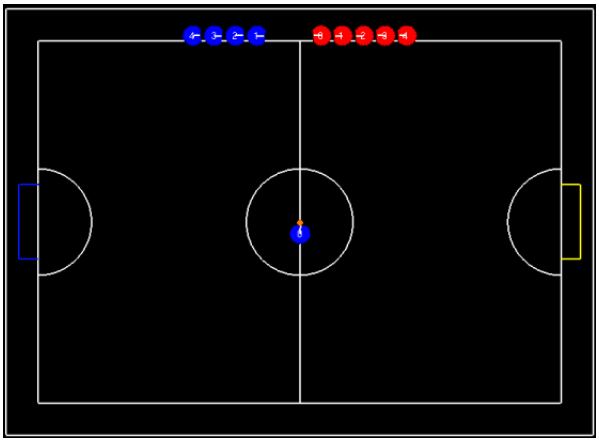

Figure 5.2: Screen capture of Cornell's Robocup simulator. Robot 0 is in the center below the ball facing the top.

Training Paths

This simulator contains several robot skills which are used by the robot team members during competition. These skills use one of several sophisticated path generation algorithms based on traditional approaches to robot dynamics and control. One of these path generation algorithms were used to create training paths. The training paths all start with the robot on the right side of the field facing toward the ball at the center of the field. The robot then travels to the ball, ending with the orientation shown in figure 5.2. Training paths on the left side of the field are unnecessary due the symmetric nature of the field. Different behavior between the left and right sides of the field is possible as long as the training data reflects this difference. Figure 5.3 gives a plot of all the training paths used. These training paths were chosen to cover the entire half field. Note that there are larger gaps between the top and bottom paths and the middle three paths. This leads to behavior that is described below. There is no requirement that the training paths do not cross. Even if two paths cross, the robot state at the crossing point is different for each path because the robot orientation is different. If the robot finds itself near this crossing point, the robot state with the closer orientation will be selected as the closest match.

The total field is roughly 5 meters long and 3 meters wide. The training paths show that the omni directional capability of the robots is not exploited. Using omni directional capabilities would allow the Robocup robot to take a more direct approach to the ball and change orientation independently of the path heading.

The training paths resulted in 126 path position states. The following discretizations were used,

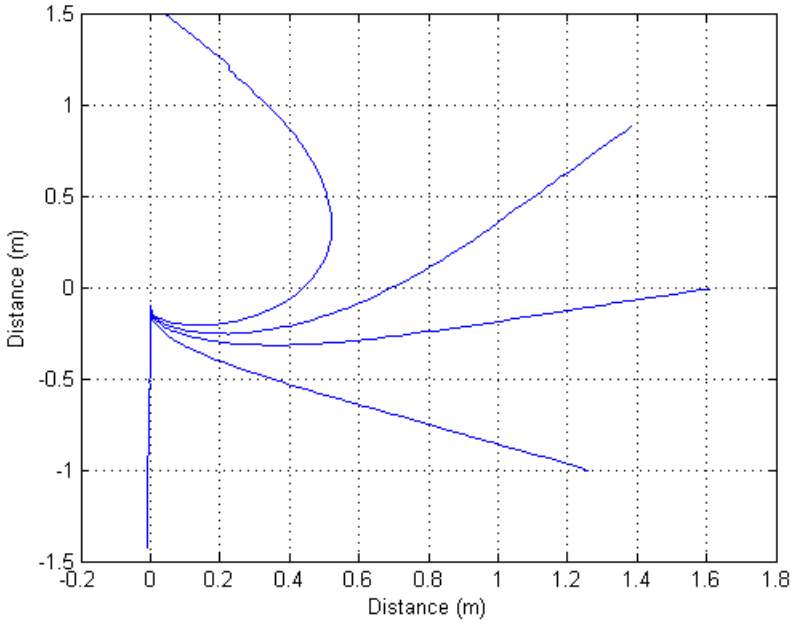

Figure 5.3: Plot of training data. Training paths on the right half of the field.

d = [1.6 1.5 1.4 1.3 ... 0.4 0.3 0.2 0.1]

θ = [0 0.1 0.2 ... 3.0 3.1 3.2]

This resulted in a M_d matrix of size 126 by 16 and a M_θ of 126 by 32.

Generated Paths

Using the 126 position states and the M_d and M_θ generated from the training data, many different path generations were run. Figure 5.4 through figure 5.9 show different sample paths that the robot traveled.

In figure 5.4, the robot starts off directly below the ball but is slightly offset from a position used in a training path. The robot is given the same commands

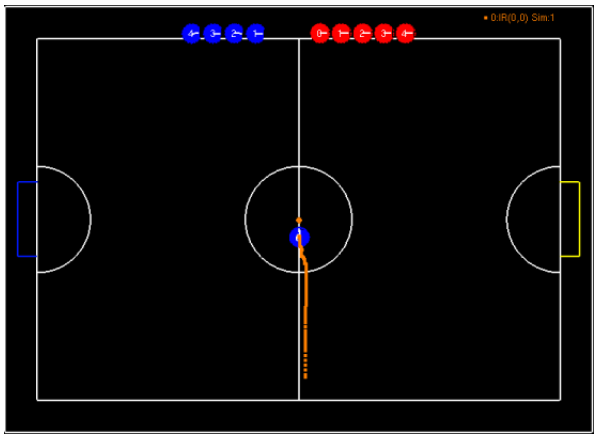

Figure 5.4: Sample path showing course correction near ball.

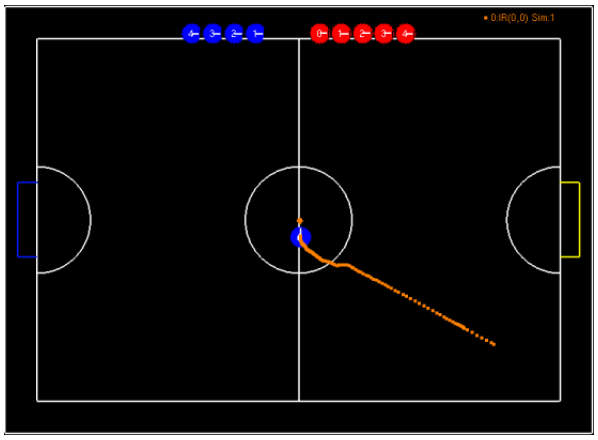

Figure 5.5: Sample path showing path switching.

that were used in the training path which results in a straight up path that is parallel to the training path. Once the robot gets close to the ball, it becomes closer to a state from another training path and the commanded velocity finally corrects for the offset.

Figure 5.5, figure 5.6, and figure 5.7 also show evidence of this path switching. The robot starts by using the control commands of one training path, but then switches control as it approaches another training path. It switches back

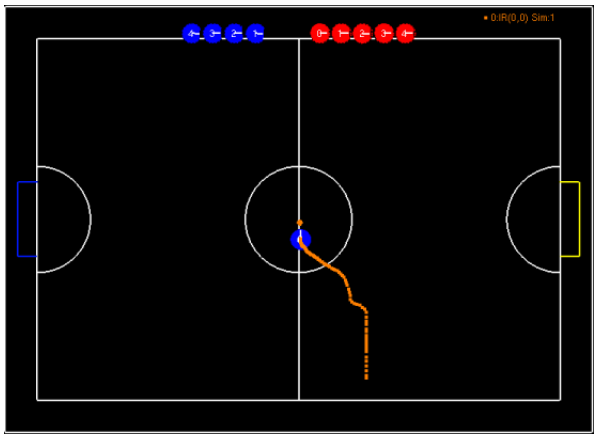

Figure 5.6: Sample path.

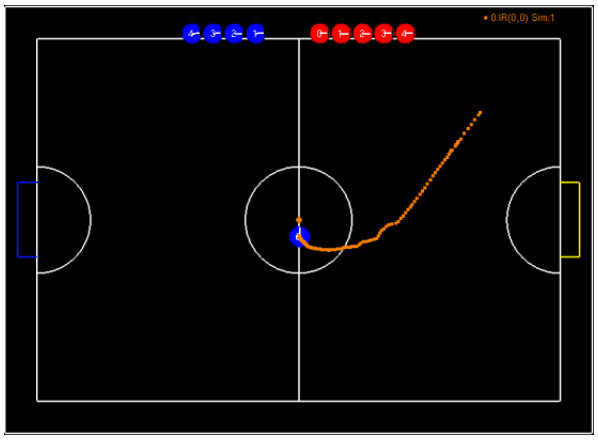

Figure 5.7: Sample path.

and forth between states from either training path as it approaches the ball.

The path shown in figure 5.8 is very smooth. The robot starts at a point that is near the top training path. As it approach the ball it remains closest to this same training path and thus uses the control commands corresponding to this path. This illustrates that increasing the number of training paths and spreading them uniformly would allow for smoother generated paths.

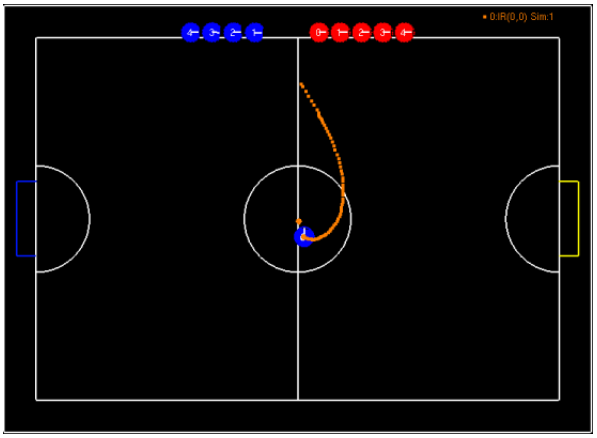

Figure 5.8: Sample path.

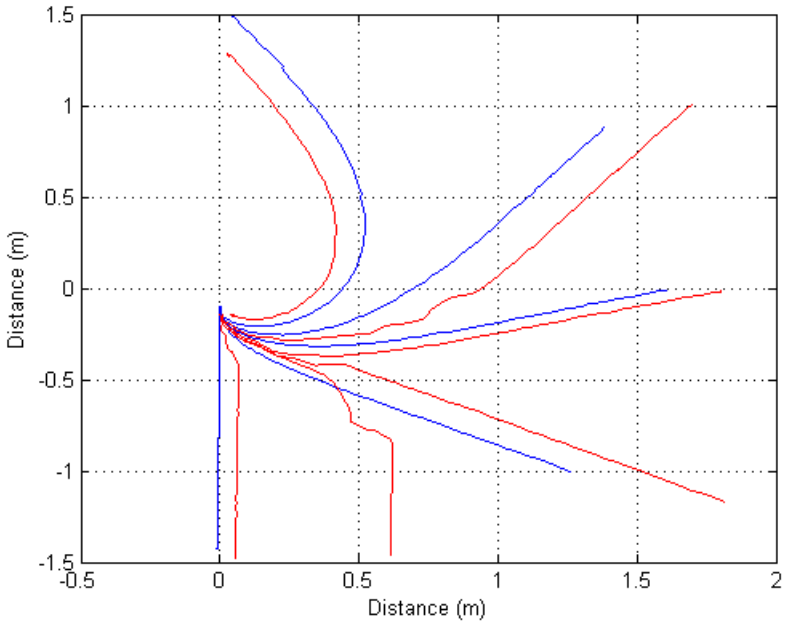

Figure 5.9: Plot of training data in blue and several generated paths in red

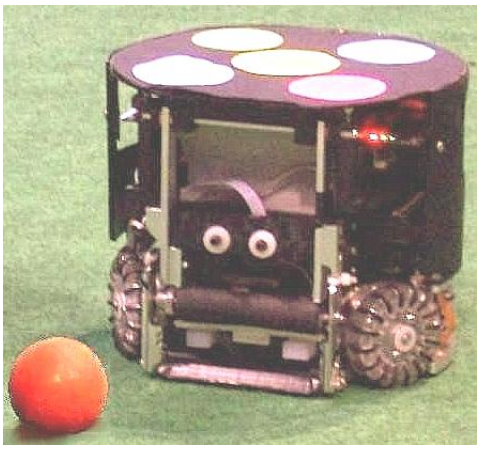

Figure 5.10: Cornell Robocup team member.

Figure 5.9 is a plot showing the original training paths in blue and several generated paths in red. It is clear on this plot that as a generated path starts approaching a different training path, the control efforts change and creates a bump in the generated path. In several cases, it is this bump that acts as a correction to force the robot to take a more direct approach to the ball. In other cases, the change causes the robot to approach the original training path and another switch occurs. More training paths would be needed to minimize these bumps and generate smoother paths.

5.3.3 Real World

The path generation method was also tested with a Cornell Robocup robot, an example of which is shown in figure 5.10. The robot was successfully able to generate and follow a correct path plan to the center of the field where the ball was located. This demonstrates that the generation method could be executed quickly enough for real time application for fast moving robots.

5.4 Conclusions and Future Work

Generating a path plan based on previous successful paths is here demonstrated to be feasible. Doing so has the advantage of being independent of the methods used to create the training paths. The training paths could be generated by different algorithms. They could also be generated directly by a human operator of a robot. A situation can be envisioned in which a complicated machine could be guided by a human being a few times, and then left to operate by itself based on the human training runs. This would avoid the time and cost of developing and coding a traditional algorithm based on system dynamics.

The next step in this process is to devise a way to allow a robot to create its own training data. In much the same way that a human baby learns to walk by initial trial and error a robot could create a successful solution to a path problem. Then, again like a child, it could practice and learn from experience how to expand and refine its movements.

BIBLIOGRAPHY

Chapter 2

[1] A. Ruina. *Biorobotics and Locomotion Lab Media* [Online] Available: http://ruina.tam.cornell.edu/research/topics/locomotion_and_robotics/ranger/Ranger2011/index.html

[2] P. A. Bhounsule, J. Cortell, A. Ruina. "Design and control of Ranger: an energy-efficient, dynamic walking robot" in *Proceedings CLAWAR* p. 441-448, 2012

[3] P. A. Bhounsule, J. Cortell, A.Grewal, B. Hendriksen, J.G.D. Karssen, C. Paul, A. Ruina. "Low-bandwidth reflex-based control for lower power walking: 65 km on a single battery charge." in *International Journal of Robotics Research* [In Press], 2014

[4] A. Grewal, "Appendix B: Estimation Module for Cornell Ranger" *Energy Efficient Trajectory Stabilization of Biped Robots* [In Press], 2014

[5] M. Garcia, A. Chatterjee, A. Ruina, M. Coleman. "The Simplest Walking Model: Stability, Complexity, and Scaling" in *ASME Journal of Biomechanical Engineering*, Vol. 120, pp. 281-288, 1998

[6] A. Kuo, J. Donelan, A. Ruina. "Energetic Consequences of Walking Like an Inverted Pendulum: Step-to-Step Transitions" in *Exercise Sport Science Review*, Vol. 22, No. 2, pp. 88-97, 2005

[7] R. Brown and P. Hwang, *Introduction to Random Signals and Applied Kalman Filtering*, p. 242, John Wiley & Sons, New York, NY; 1997

[8] University of Cape Town. (2005) *Topic 2: The pendulum* [Online] Available: http://www.phy.uct.ac.za/courses/phy321f/cpman2.pdf

[9] University of Deleware. (2011) *The Euler-Cromer method* [Online] Available: http://www.physics.udel.edu/~jim/PHYS460_660_11S/Ordinary\%20Differential\%20Equations/Euler-Cromer\%20Method.htm

[10] J. Fessler. (2004, May 27) *Design of Digital FIlter* [Online PDF]. Available: http://web.eecs.umich.edu/~fessler/course/451/l/pdf/c8.pdf, p. 8.2

Chapter 3

[11] University of Zurich. (2012) *Robots on Tour* [Online]. Available: http://www.robotsontour.com/en/

[12] J. Kuriloff. (2011, May 19) *Mechanical Design of Biped Hardware* [Online PDF]. Available: `http://ruina.tam.cornell.edu/research/topics/locomotion_and_robotics/ranger/ranger_paper/Reports/Ranger_Robot/mechanical/Inventor_files/Sensor_Box_2011.zip`

[13] I. Miller, M. Campbell, D. Huttenlocher, A. Nathan, et. al., "Cornell, Skynet: Robust Perception and Planning in an Urban Environment," in *The DARPA Urban Challenge*, Springer Berlin Heidelberg, 2009, pp. 257-304

[14] B. Klug. (2010, December 9) *Microsoft Kinect: The AnandTech Review* [Online]. Available: http://www.anandtech.com/show/4057/microsoft-kinect-the-anandtech-review

[15] CMUcam. (2013) *CMUcam: Open Source Programmable Embedded Color Vision Sensors* [Online]. Available http://www.cmucam.org

[16] A. Chaudhry. (2013) *Ranger Line Following firmware* [Oneline]. Available: http://ruina.tam.cornell.edu/research/AtifChaudhry/code.html

[17] Parallax Semiconductor. (2011) *Parallax Semiconductor Software* [Online]. Available: http://www.parallaxsemiconductor.com/software

[18] B. Campbell. (2010, April 23). *BST - The multi-platform Propeller Tool Suite* [Online]. Available: http://www.fnarfbargle.com/bst.html

[19] K. Agyeman, A. Rowe. (2012) *CMUcam4 Electrical Characteristics* [Online PDF]. Available: http://www.cmucam.org/attachments/688/CMUcam4 Arduino Shield DOC EC - v10.pdf

[20] K. Agyeman, A. Rowe. (2012) *CMUcam4 Board Layout and Ports* [Online PDF]. Available: http://www.cmucam.org/attachments/689/CMUcam4 Arduino Shield DOC BL&P - v10.pdf

[21] Protostar (2013) *Flocked light trap material* [Online]. Available: http://www.protostar.biz/flock.htm

[22] Pro Tapes and Specialties (2013) *Duve pro* [Online]. Available: http://www.protapes.com/Catalog/Items/DuvePro.aspx

[23] Shurtape Technologies, LLC (2014) *Shurtape CP743* [Online]. Available: http://www.shurtape.com/node/292

[24] SANWA denshi kiki (2011) *SG-6G* [Online]. Available: http://sanwa-denshi.com/rc/sky/propo/sd-6g.html

[25] A. Ruina (2013) *Ranger Steering Simulation* [Oneline]. Available: http://ruina.tam.cornell.edu/research/AtifChaudhry/code.html

Chapter 4

[26] M. G. Earl and R. DAndrea, A Study in Cooperative Control: The RoboFlag Drill, *in Proc. American Control Conference*, Anchorage, AK, 2002

[27] A. G. Richards and J. P. How, Aircraft Trajectory Planning With Collision Avoidance Using Mixed Integer Linear Programming, *in Proc. American Control Conference*, Anchorage, AK, 2002

[28] M. G. Earl and R. DAndrea, Modeling and Control of a Multi-Agent System Using Mixed Integer Linear Programming, *IEEE CDC*, Los Vegas, Nevada, Dec. 2002

[29] Y. Kuwata, Real-time Trajectory Design for Unmanned Aerial vehicles using Receding Horizon Control, *Masters Thesis*, Massachusetts Institute of Technology, June 2003

[30] C. Tomlin, G. J. Pappas, and S. Sastry, Conflict Resolution for Air Traffic Management: A Case Study in Multi-Agent Hybrid Systems, *IEEE Transactions on Autonomous Control*, vol. 43, no. 4, pp. 509-521, April 1998.

[31] A. Bemporad, F. Borrelli, and M. Morari, Piecewise Linear Opti- mal Controllers for Hybrid Systems, *American Control Conference*, Chicago, USA, Vol. 2, pp. 1190-1194, June 2001.

[32] A. G. Richards, J. P. How, T. Schouwenaars, and E. Feron, Plume Avoidance Maneuver Planning Using Mixed Integer Linear Programming, *AIAA Guidance, Navigation and Control Conference*, Montreal, August 2001.

[33] K. Misovec, T. Inanc, J. Wohletz, R.M. Murray, Low Observable Nonlinear Trajectory Generation for Unmanned Air Vehicles, *IEEE CDC*, Maui, Hawaii, Dec. 2003

[34] J Kim, J Hespanha, Discrete Approximations to Continuous Shortest- Path: Application to Minimum-Risk Path Planning for Groups of UAVs, *IEE CDC*, Maui, Hawaii, Dec. 2003

[35] J.KnuttiandD.Corman, *OEP documentation*,The BoeingCompany, 2002

[36] ILOG, Inc. CPLEX 7.1 http://www.ilog.com/products/cplex/

Chapter 5

[37] J. Hawkins and S. Blakeslee, *On Intelligence*, Times Books, New York, NY; 2004.

[38] D. George and J. Hawkins, Invariant Pattern Recognition using Bayesian Inference on Hierarchical Sequences, *Technical Report*, Redwood Neuroscience, Menlo Park, CA; 2004.

[39] D. George and J. Hawkins, A Hierarchical Bayesian Model of Invariant Pattern Recognition in the Visual Cortex, *in Proceedings of the International Joint Conference on Neural Networks - IJCNN*, Montreal, Canada; 2005

[40] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufman Publishers, San Francisco, CA; 1988.

[41] R. D'Andrea. *The Cornell RoboCup Soccer Team: 1999 - 2003.* In B. Levine and D. Hristu, editors, Handbook of Networked and Embedded Control Systems. Birkhauser, 2005.

[42] R. D'Andrea and J.W. Lee, *Cornell BigRed : Small Size League Winner*, AI Magazine, p. 41-44, 2000