AI For Robotics Final Project Report

Ben Emmett
Team #35
7/30/2016
User ID: bemmett3
GTID: 903182385

1. Introduction

1.1 Problem Space

The objective of this project is to develop a python program to characterize and predict the motion of the hexbug in various test cases. A number of different techniques were employed to analyse the measurement data, develop a model of hexbug motion, and implement the model in hexbug path prediction. This report describes the techniques used, their rationale, and test results.

1.2 Approach to the problem

Fundamentally, this project is centered around modelling and characterizing the hexbug in order to predict future movements. I chose to use a physics-based model. This model uses a kalman filter to track and estimate position and velocity. Propagation uses the position and velocity estimates along with two types of accelerations. A forward acceleration which is always in the direction of the current velocity estimate, as well as a periodic acceleration which is time dependent, but independent from the hexbug position or orientation.

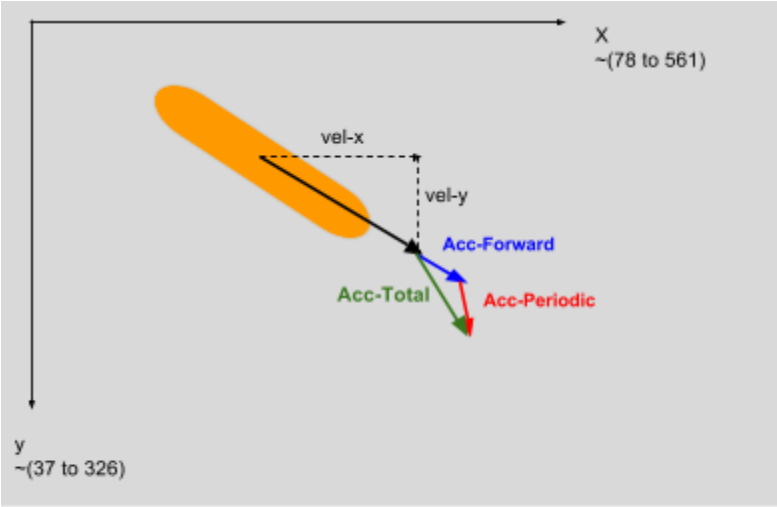


Figure 1.2-1 Hexbug State Dynamics Diagram

Testing is critical to the success of this project, so early on I created visualization and testing tools. These tools operate in the test cases and extra training data in order to evaluate using L2 errors as well as visualize the measurements, tracker estimates, and predictions. Optimization of parameters was performed using the Twiddle algorithm.

I analysed the training data looking for clues about a proper motion model. I found there to be periodic acceleration information which I leveraged for the propagation motion model. My initial theory about this periodic acceleration is that it came from an external source (either tilting the box or applying a time-varying magnetic field). For this reason, I use the terms external acceleration and periodic acceleration interchangeably in my code. I also use the term internal acceleration to mean the normal forward acceleration of the hexbug. In the report I will use the terms periodic acceleration and forward acceleration.

Note also that I refer to the box walls and the candle as the “obstacles” throughout the paper.

2. Data Analysis

By watching the videos of the hexbug, it is apparent that there is some kind of varying acceleration aside from the normal forward acceleration due to the inclined bristles and vibration. In order to better characterize the hexbug and it’s apparent random motion, I analysed the measurement data looking for correlations between motion, time, hexbug orientation, and hexbug location.

I first tested whether the box floor was tilted or curved in any way as this would affect the motion of the robot based on its position. In order to asses motion and position correlation, I estimated the hexbug acceleration at each measurement, then averaged over location bins as seen for the y-dimension in Figure 2-1. Ignoring the sporadic outliers, it is clear that there is no strong correlation between acceleration and position in the box.

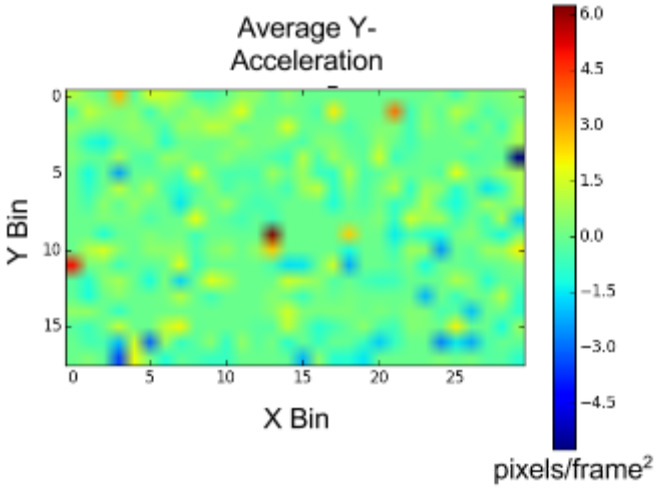


Figure 2-1 Local Y-Dimension Acceleration

I also tested the acceleration and velocity over time. Acceleration estimates were very noisy due to being the second derivative of already noisy data. However, the velocity over time clearly fluctuates in a sinusoidal nature as seen in figure 2-2. This indicates that, although not measurable by our data, acceleration must also be sinusoidal in nature. As I will discuss later, periodic acceleration is estimated on each test run and used in my propagation model.

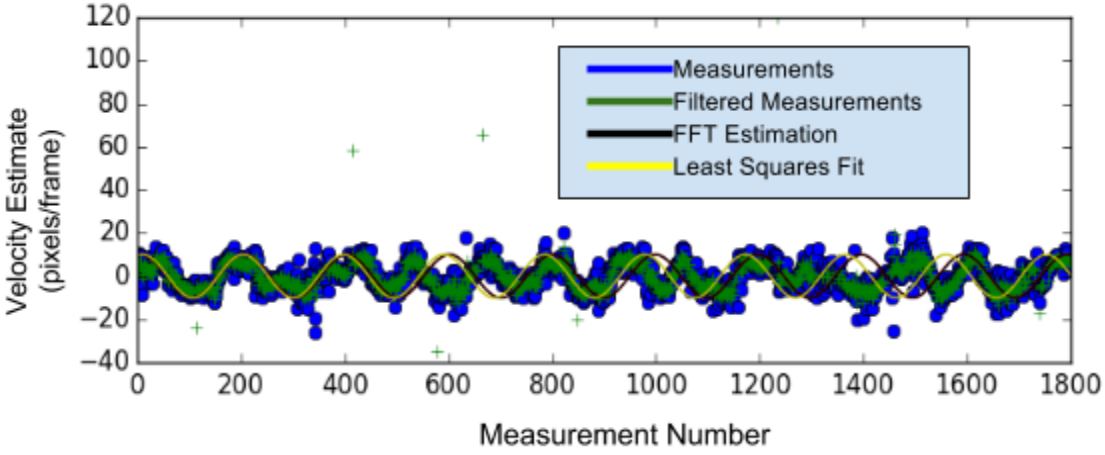


Figure 2-2 Velocity of Hexbug over time

3. Design

3.1 Overview

The file finalproject.py has the following general process flow: Read Measurements, Kalman Tracking and State Estimation, Periodic Acceleration Characterization, State Propagation, Write Predication. Each step is described in more detail below.

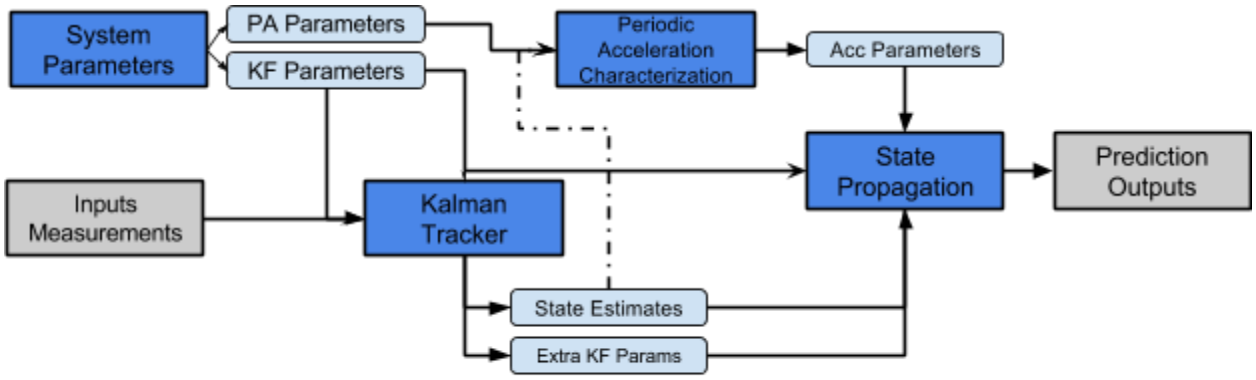


Figure 3.1-1 System Design Overview

3.2 State Estimation

I use a basic Kalman filter with extra modifications to track and estimate position and velocity of the hexbug for the duration of the measurement data. The kalman filter was not my first choice for tracking and state estimation because of the difficulty of modeling the sporadic movements of the hexbug. A particle filter would be much less rigid to the random wandering of the hexbug. However, the advantage of using the kalman filter is that that it will converge to a good estimate given a good enough model to follow and given enough measurement updates. Another difficulty of the Kalman filter is recovering to a good state estimate soon after an obstacle collision. The rigidity of the Kalman filter was remedied by utilizing a position depended process noise matrix. For propagation without measurements, additional acceleration and collision models are applied.

3.2.1 Kalman Filter

The Kalman Filter state vector consists of only position and velocity in X and Y directions. Acceleration is modelled separately and injected only during the state propagation when no more measurements are available. For the sake of simplicity, the Kalman filter uses pixels as the position units, pixels per frame (pixels/frame) as the velocity units, and pixels per frame^2 (pixels/frame^2) for acceleration. This avoids the need to convert time units to seconds and potential errors.

One modification made to the Kalman filter to help recover a good state estimate after an obstacle collision is that process noise matrix (Q) is modified with respect to the hexbug proximity to an obstacle. At the boundary of an obstacle the noise is set very high but falls off linearly as distance to the obstacle increases until a fixed distance away where Q is set to it's minimum values. The higher process noise near obstacles helps the tracker to recover a good state estimate more quickly after an obstacle collision by giving it less confidence in it's own state predictions and causing it to rely more heavily on the measurements.

The tracker also attempts to detect irregular velocity. This is done by comparing the estimated velocity with an expected velocity value and normalizing by the velocity covariance as follows:

$$norm\_delta\_vel = abs(tracker\_vel\_estimate - expected\_hexbug\_vel)/sqrt(vel\_X\_var^2 + vel\_Y\_var^2)$$

If the normalized delta velocity becomes too great (>2.0pixels/frame) for more than 10 measurements in a row it is determined that the velocity is irregular. When this occurs a new velocity is estimated and passed on to State Propagation and overrides the fixed value. Also, when this occurs the acceleration model is turned off for State Propagation.

The tracker function returns state estimates for all time steps. The velocity estimates over time are used for Periodic Acceleration Estimation. Only the final state estimate is used in State Propagation.

3.2.2 Alternative Tracking and State Estimation Techniques

Initially, I used a particle filter for state estimation. The particle Filter seemed appropriate for tracking the movements of the hexbug which didn't seem to follow a linear model. I used a state space with position (x,y), speed, heading, and yaw-rate. I found that the particle filter could do well with smoothing the hexbug's position, however the dynamic state variables (speed, and yaw-rate) would not converge to a good estimate. Perhaps using position and velocity in the state space would have worked better. However, I was not able to get it working to my satisfaction. The particle filter code is available but not used.

I also tried increasing the state vector of the Kalman filter to include acceleration and jerk. Overall, these both worked very well. However, I found it difficult to create a way to handle the obstacle collisions. Most collisions did not cause issues, however in a few cases collisions caused the state estimates to oscillate wildly. Given more time, I would like to find a remedy to this issue, however it came up too late into the assignment and I was forced to use the 4-state Kalman filter in order to finish in time.

I also considered using an Extended Kalman filter. This would have been able to more accurately estimate and predict the periodic acceleration. However, I spent most of my time with the particle filter and other Kalman filters so I did not have time to implement this.

3.3 Periodic Acceleration Estimation

3.3.1 FFT Periodicity Estimation

Data analysis shows that velocity (and therefore acceleration also) has a sinusoidal component with a period of roughly 6.3 seconds. The frequency of this periodic acceleration is estimated for each test case in periodicAccModel.py using a FFT to find the dominant frequency and the corresponding phase of that frequency. Large amounts of noise make it difficult to determine the precise phase of the signal. I found that in this case, a small error in phase can cause the signal to be wildly off after 1800 samples. For this reason, the FFT is only performed on the last 256 measurements of the series. I chose to use a very large FFT of length 2^13 in order to produce very fine frequency bins and therefore get a more precise frequency estimate as well.

The velocity estimates from the tracker are used to estimate the frequency, phase, and amplitude of the sinusoidal component. From the velocity component, acceleration parameters are derived using the following relationship:

a(t) = dv(t)/dt, where

$v(t) = A_v*cos(w_v*t + phi_v)$, *Therefore*

$a(t) = -w_v*A_v sin(w_v*t + phi_v) = -A_{acc}*sin(w_{acc}*t + phi_{acc})$

The acceleration parameters (A_{acc},w_{acc},phi_{acc}) for each dimension (X & Y) are passed from the Periodic Acceleration Estimation to State Propagation where they are injected into the Kalman state update step through the input effect matrix called BU.

Analysis with the training data showed a lot of promise. However, in practice results of the periodic acceleration estimation are often disappointing. Phase errors can be easily seen in Figure 3.3-3 below where the hexbot accelerates in the x direction while the prediction accelerations in the -x direction. I spent a good amount of time testing alternative periodicity estimation techniques and would have liked to spend more time on this as I'm sure it would improve propagation accuracy greatly. Because phase estimates are often incorrect, optimization of parameters showed that applying a small scale factor to reduce the amount of periodic acceleration actually improves average performance over many test cases.

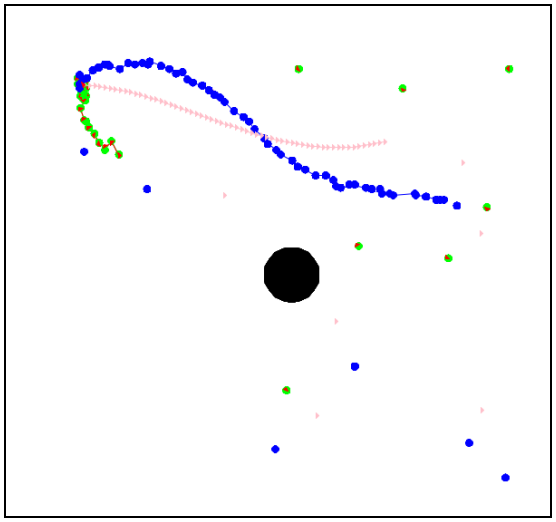


Figure 3.3-2 - In-Phase Estimate

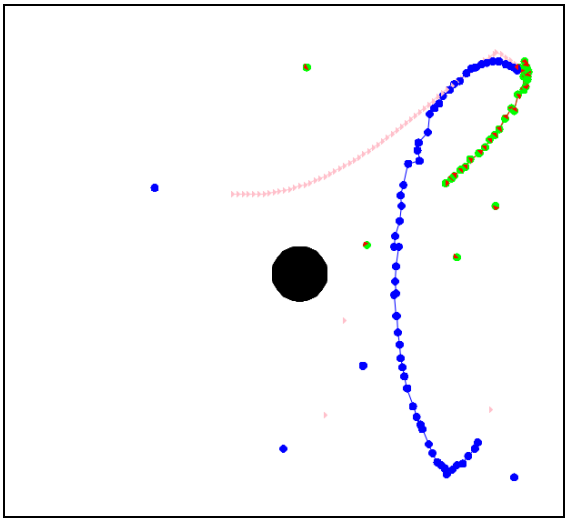
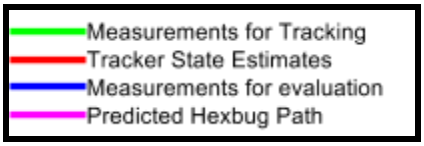


Figure 3.3-3 Out-of-Phase Estimate



3.3.2 Alternative Periodicity Estimation Techniques

I tested several approaches beside the FFT. None of the techniques were able to consistently estimate the frequency and phase of the velocity estimates as well as the FFT did.

Another approach I used instead of a FFT was to first apply a median filter to the velocity estimates to remove high frequency outliers, then remove measurements that occurred very close to an obstacle. This removed sharp changes in velocity which were caused by collisions. After both filtering steps, the remaining measurements had uneven time spacing and therefore the FFT could not be used on them. In this case, instead of a FFT I used a sinusoidal least squares fit to find frequency and phase. This alternative method also worked well, however the FFT method generally produced better results when evaluated by visual inspection.

I also explored the Lomb-Scargle method which produces a periodogram of a given signal. As I understand it, the Lomb-Scargle method is fundamentally very similar to least squares fit. The results of Lomb-Scargle were also promising but not as accurate as the traditional FFT in this case.

3.5 Prediction

Hexbug path predictions are made by propagating the tracker's final state estimate for 60 frames. The propagator and the tracker are intimately tied together, in fact as mentioned before, the propagator uses the same function call as the tracker does to updated states. The only difference is that in the propagator, acceleration models and obstacle collision models are turned on. The acceleration and collision models are described below.

3.5.1 Acceleration Models

Two types of acceleration are used when propagating the state estimate beyond the given measurements, forward acceleration and periodic acceleration. The two are calculated separately then summed together before being injected into the state update via the BU matrix.

3.5.1.1 Forward acceleration

Forward acceleration is assumed to be constant and always in the forward direction. This models the acceleration due the vibration and inclined bristles which drives the hexbug forward. Forward direction is estimated using the hexbug's most recent velocity estimate using the following:

$$Acc\_fwd = acc\_magnitude*V/sum(V)$$

Where V is a 2x1 velocity vector.

3.5.1.2 Periodic Acceleration

Periodic acceleration parameters are applied along with a "boost" factor which is optimized and accounts for uncertainty in parameter estimation.

$$Acc\_periodic = boost\_factor*(-1*Acc\_mag)*sin(w_{acc}*t + phi_{acc})$$

Forward and periodic acceleration terms are added together then used in the input effect matrix, BU, in the Kalman filter update as follows.

Acc = Acc_fwd + Acc_periodic
*BU = [0,0,acc[0]*dt,acc[1]*dt]*

3.5.2 Collision Models

The results of collisions with walls and the candle appear to be somewhat random. Given more time I would like to have characterized them with further analysis using machine learning techniques and/or more refined physics models. However, due to lack of time I implemented a simple physics model which assumes a specular reflection type of collision where the exit angle (theta_out) equals the entrance angle (theta_in). Energy loss is accounted for by decrementing the velocity estimate by a fixed value (speed_dec = 0.5) as follows.

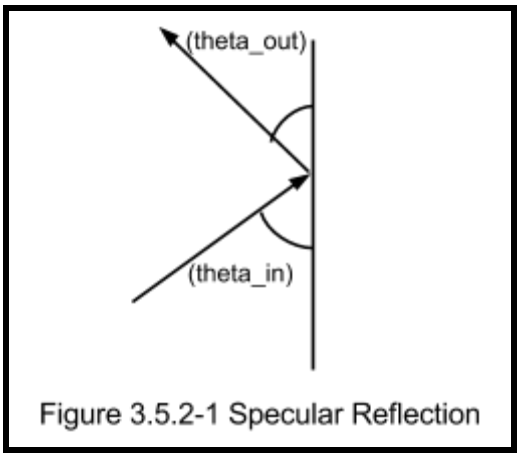


Figure 3.5.2-1 Specular Reflection

3.5.2.1 Wall Collision Model

The specular reflection type collision is implemented as follows

if(left or right wall collision):
    V = speed_dec[-Vx,Vy]*
if(top or bottom wall collision):
    V = speed_dec[Vx,-Vy]*

3.5.2.2 Candle Collision Model

The same specular reflection model is used for the candle but was a bit more difficult to derive for a circular column than for a flat wall. I took the approach of rotating the velocity vector by the angle of impact such that the angle of impact was then effectively 0 radians. I then applied the same specular reflection method as used in the wall collision model described above. After applying the specular reflection model, the velocity vector is rotated back to the original frame. This is illustrated by Figure 3.5.2-2 below. The result is that the velocity out is as if it had reflected on the tangent line where the collision occurred.

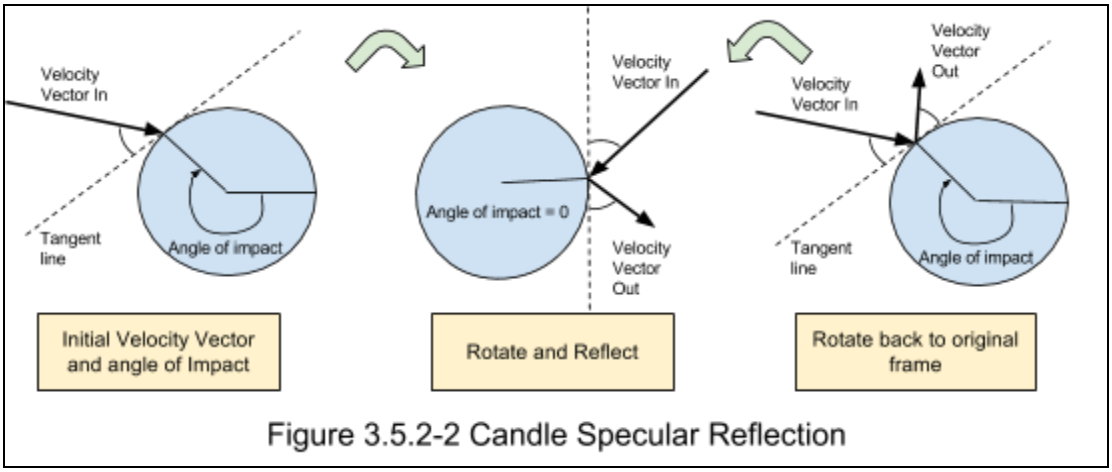


Figure 3.5.2-2 Candle Specular Reflection

4 Testing

4.1 Unit Testing

Unit testing for the minor functions was performed to ensure correct and expected functionality. The unit testing for these was done in an informal manner utilizing debug mode of PyCharm which allows pausing and evaluating sections of code manually. I altered variable values in this manner to find and fix design issues and bugs until I was satisfied that each subsection of code was working properly.

4.2 System Testing

4.1.1 Test Cases

According to the Piazza forum poll, I configured a file called finalproject_test.py which uses the first 1740 measurements and predicts the following 60. The L2 score was calculated for each run, then the mean of all but the highest and lowest L2 scores was averaged. The final average using this method is 673.8.

4.1.2 Further Testing

A similar approach was taken utilizing the provided training_data.txt to further evaluate the overall system performance in various test cases. The file dev_finalproject.py runs on 1800 measurements at a time, predicts the next 60, and calculates an L2 score for evaluation. This process is performed iteratively a configurable amount of times and a mean score is calculated at the end.

This metric was used to generally prove out design changes. This same general method was also used in optimization as described below.

5 Optimization

Optimization was performed using Twiddle. The scoring metric is the average L2 score from running the design described above on the measurements found in training_data.txt. The scoring function runs on 1800 measurements at a time to obtain a state estimate, propagates the state estimate 60 frames, and compares the propagation with subsequent 60 measurements in training_data.txt. The average L2 score of 250 test runs was used as the evaluation metric to be minimized.

Optimal parameters were obtained for the following:

1. **max_speed** - Used in propagation when acceleration models are activated to limit the maximum predicted speed (pixels/frame) of the hexbug. This value is sometimes overridden by the tracker.
2. **meas_noise** - The diagonal values of the Kalman filter R matrix.
3. **cov_init** - The initial diagonal values of the Kalman filter covariance matrix P.
4. **max_q** - The diagonal values of the Kalman filter Q matrix when distance from the hexbug to any obstacle is 0.
5. **min_q** - The diagonal values of the Kalman filter Q matrix when distance from the hexbug to any obstacle is at least 25 pixels.
6. **q_sf** - A 2x1 array of scale factors applied to the diagonal values of the Q Kalman filter Q matrix. The first value is applied to position process noise, the second is applied to velocity process noise.
7. **int_acc** - The forward acceleration (i.e. Internal Acceleration) (pixels/frame^2) value applied during state propagation when the acceleration model is active.
8. **ext_acc_boost** - A scale factor applied to the periodic acceleration component (i.e. External Acceleration) (unitless) to either boost or attenuate its influence on the predicted path.

Optimization of these parameters from their original values reduced the average L2 score by about 400.

An interesting note about optimization is that it sometimes shows where weakness exists in the model itself. For example, in my optimization both of the acceleration parameters (int_acc, and ext_acc_boost) were optimized to be smaller than the physics model would predict. There are two reasons for this. First, the periodic acceleration estimation phase is sometimes wrong, and therefore it is sometimes better to reduce its influence. However, it is evident that the acceleration model does more good than harm because ext_acc_boost was not optimized to zero even when its initial value was set to zero in Twiddle. The second reason that the acceleration parameters are set low is due to the low fidelity collision modelling. Sometimes the hexbug does not “bounce” off a wall or candle collision in a specular way. In these cases, where my collision model is incorrect, it is sometimes more optimal to allow the hexbug to stay slow

moving and not go very far in the wrong direction. These kinds of “red flags” from optimization shed light on which areas of the model itself could use further refinement.

6 Conclusion

Using the techniques taught in the AI for Robotics course as well as researching other common techniques and algorithms, I was able to create a program to characterize and predict the hexbug motion reasonably well. I took a simple Kalman Filter track and augmented it with additional process noise adaptation as well as acceleration and collision models to fit this particular project. Many approaches were considered and several were implemented. In the end, several algorithm classes were employed to work in unison and provide reasonably accurate position estimates for the hexbug robot.