CS 8803: AI for Robotics
Final Project Report - Fall 2015

Introduction

In this project, our team produced an algorithm to predict the motion of a hexbug robot moving in a wooden box. These predictions were based on ten 60 second video clips. We were provided with real-world data files containing the centroid coordinates that were extracted from the video clips to be used as input for the algorithm we implemented. The goal of the project was to predict the position of the robot as accurately as possible for next 2 seconds (the following 60 frames) after the end of the provided inputs.

Attempted Approaches

In order to accomplish this, our team first considered using a particle filter to predict the motion of the hexbug. However, after extensive research looking into the particulars we discovered that the particle filter was better suited for determining the current location rather than predicting a future position. We also decided against a particle filter since it is resource intensive and the project required the algorithm to complete in a limited time frame.

In the next approach, our team decided to pursue the challenge using a library called FilterPy [1]. FilterPy is a Python library that implements a number of Bayesian filters including a basic Kalman Filter, an Extended KF, an Unscented KF, a Fading Memory Filter, and an Adaptive Filter. We began by utilizing FilterPy's Kalman filter by implementing a 2D filter that only tracked the X coordinate using very basic input values: 1, 2, 3, and 4. We were successful in having the filter produce predictions very close to what we would expect. However, when we tried to install FilterPy on the VM provided for the project, we ran into compatibility issues. After a great deal of time spent researching, we were unable to determine an easy way to have the latest version of the library install onto the VM and we

didn't want to risk any incompatibility with using an older version of the library. As a result, we decided to abandon this approach.

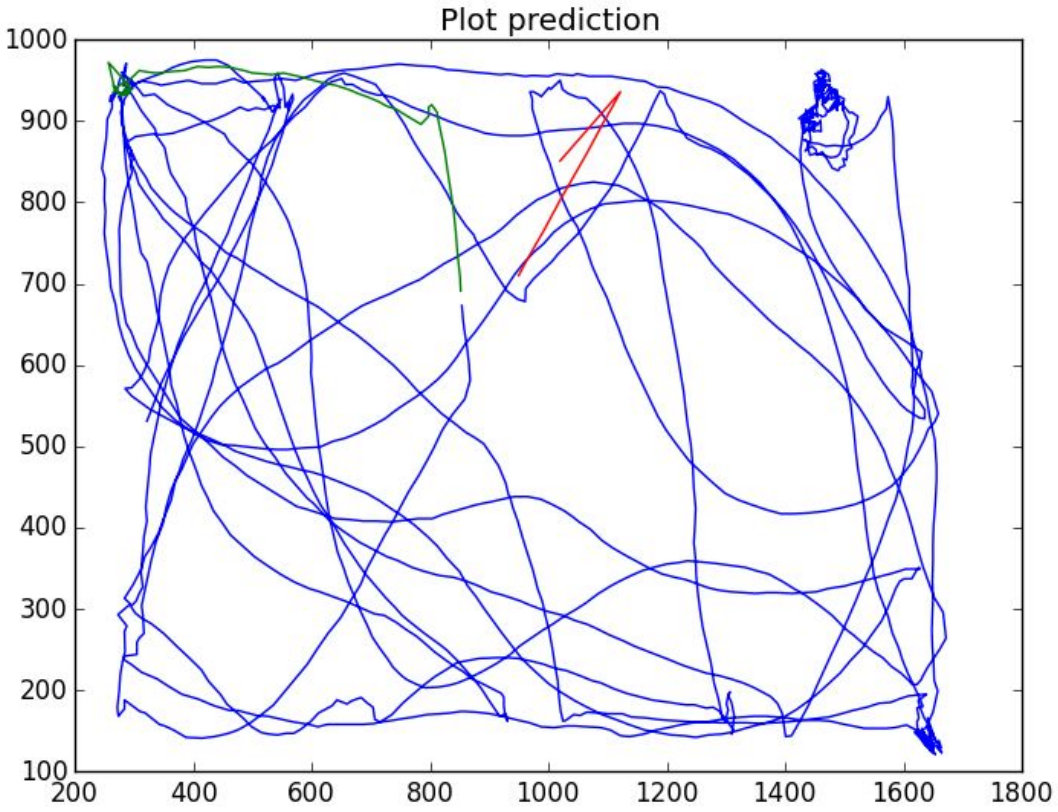
The next approach we tried was to enhance the 2D Kalman filter that we implemented at the beginning of the project. Our familiarity with the Kalman filter and related concepts due to the lessons we covered in class, made us feel comfortable continuing to work with it. We first enhanced our implementation by changing it to a 4D filter to also track the Y coordinate and its velocity. This implementation was successful, so we proceeded to introduce acceleration into our equations by converting our matrices to support 6 dimensions. While running the filter with all of the data in the input files we discovered that these updates actually made the estimation worse. We were disappointed in these results and recalled that the Basic Kalman filter is better suited for linear motion only, so we spent several days researching how to implement either a UKF or EKF as our algorithm. The issue with these approaches is that the mathematical complexity involved in coding these algorithms proved to be beyond the abilities of all of our team members. However, the during research into these filters we learned more about the Fading Memory filter and the UKF. It was discovered that the Unscented Kalman Filter deals with nonlinearity by simplifying the problem into its most basic linear form by taking each observation and producing a new linear prediction [3]. The Fading Memory filter takes a similar approach and only uses a small number of the most recent data points as input into the filter in order to generate a prediction [4]. This discovery provided us with the insight we needed to determine how to proceed.

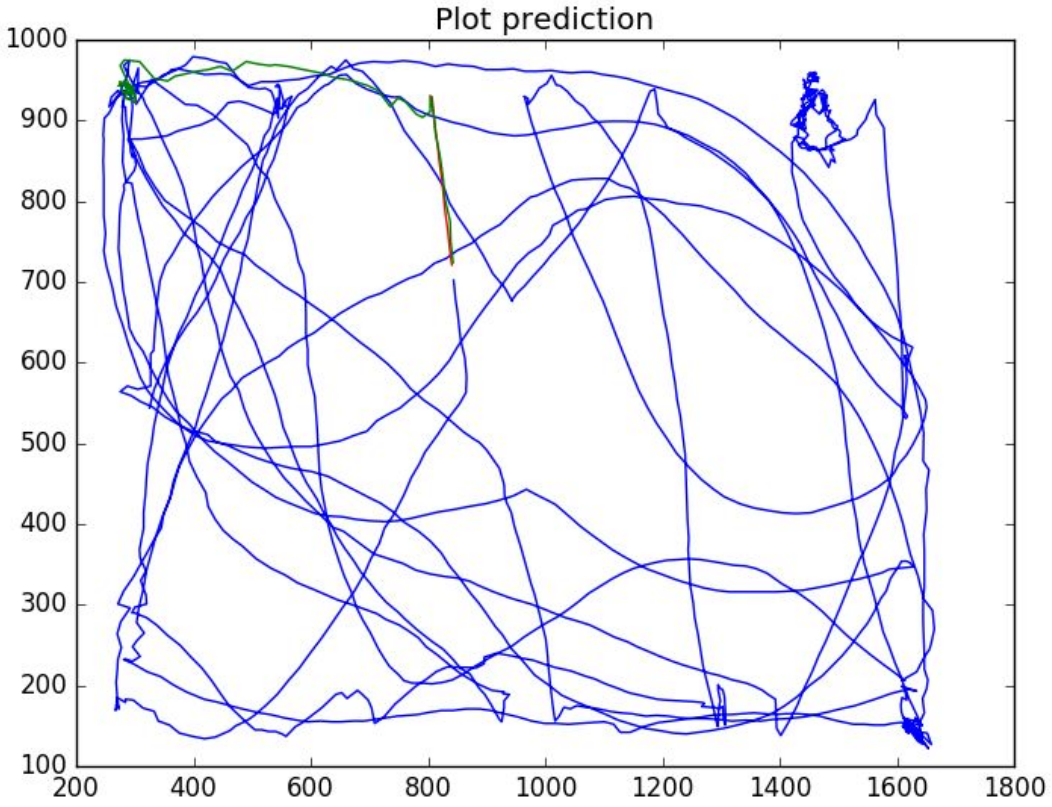
Final Algorithm Functionality

Taking all of this into consideration, we decided to implement our own simple version of a Fading Memory filter by utilizing a 4D Kalman Filter and only looking back a fixed number of frames into the data in order to generate our predictions. For every iteration of the loop that generates our predictions, we re-initialize the x and P matrices and only run

the filter with the last 4 frames worth of data. During that loop, we append our own prediction to the list of measurements, so that it then becomes part of the data that is used to make the next prediction.

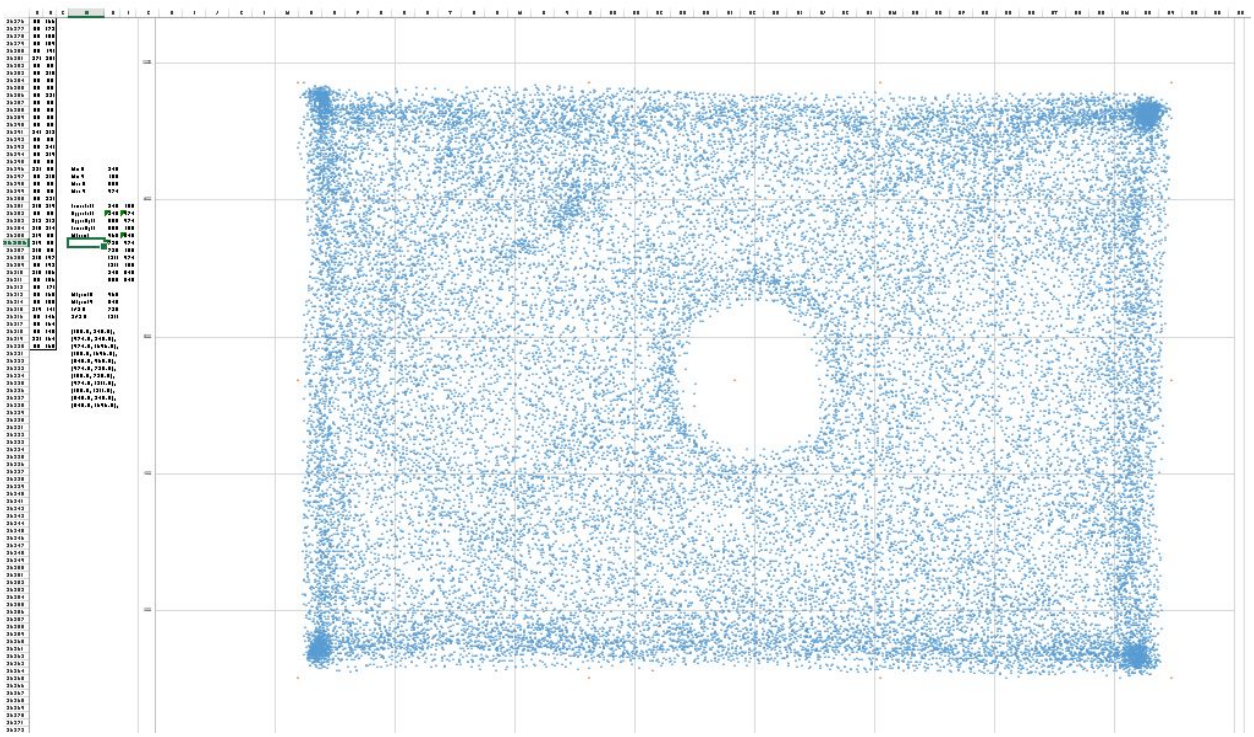
Our first version of this algorithm initialized the x matrix with coordinates of (0, 0), but we soon realized that with so few data points it was imperative to initialize the x matrix with the first actual coordinate that makes up the incoming measurements. This small change greatly improved the accuracy of the results as can be seen in the before and after plots below where blue is the input data, green is the expected data and red is our prediction.



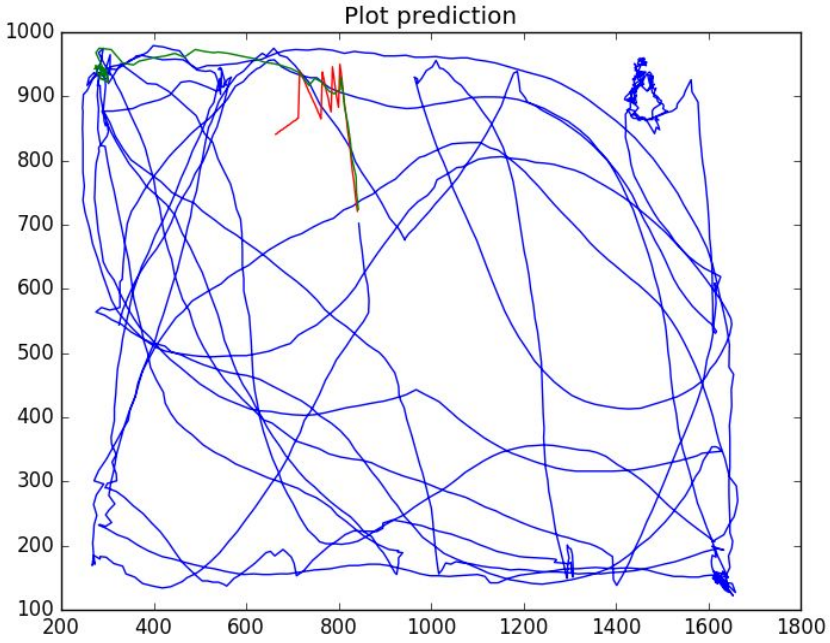


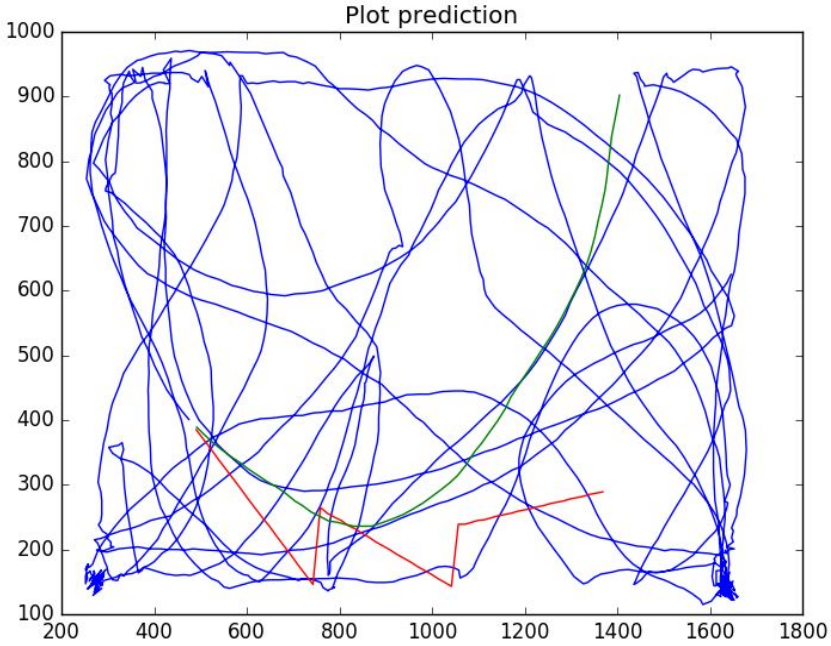
The predicted values are now almost congruent with the expected actual values such that the red and green lines are almost indistinguishable. This change allowed us to improve the result of running grading.py by approximately 660 points (from 3592 to 2928).

Next, we added a feature to simulate the hexbug bouncing off the wall. We analyzed all of the training data in Microsoft Excel as shown below in order to determine the equations of the lines for the top and bottom of the box.



We felt that the left and right sides of the box were close enough to vertical that we could use a single max and min value to define the boundaries of the X coordinate. When a collision is detected, with a particular side of the box the algorithm then looks back at the last 4 measurements to determine the average distance between them. Then depending on which side of the box is hit, we add or subtract the calculated average distance that the hexbug moves from the appropriate coordinate. For example, if the robot hits the top of the box, we assume that the velocity of the robot remains the same in the X direction and we simply subtract the average distance from the Y coordinate in order to determine the location that we predict the hexbug will be during the next time interval. Adding this feature allowed us to further improve our previous predictions and achieve the following results:





This change also improved the result of running the grading.py code by approximately 700 points (from 2856 to 2188).

Possible Future Enhancements

Due to time constraints, our team only modeled the boundaries of the box. So, the predictions could be improved if the model incorporated the obstacle in the center of the box. We could try to implement smoothing on our predictions in order to bring them more in line with the actual future movements of the hexbug. Machine learning techniques could be applied in order to extract information that could be used to further refine the predictions. Perhaps the observed trajectory of the hexbug over time could be analyzed in order to detect the pattern of motion of the robot in an attempt to reverse engineer the actions it will take at a certain time. For example, if the turning angle is repeated every 10 seconds this pattern could be detected. We could also use machine learning techniques to take advantage of all of the training data in order to determine the most likely angle of reflection based on the angle of incidence when the hexbug collides with an obstacle. All of this additional knowledge could then be leveraged to improve the results of a filter implementation.

Conclusion

At first, predicticting the location of the hexbug seemed like it would be an impossible task. Our team encountered many challenges on our journey to implementing a solution and in the end we were very happy to have achieved what we consider to be a satisfactory result. It was very rewarding to have the chance to apply the concepts we learned in class to a real-world scenario.

References:

[1] http://pythonhosted.org/filterpy/
[2] http://nbviewer.ipython.org/github/balzer82/Kalman/blob/master/Kalman-Filter-CA.ipynb?create=1
[3] http://nbviewer.ipython.org/github/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/10-Unscented-Kalman-Filter.ipynb
[4] http://nbviewer.ipython.org/github/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/14-Adaptive-Filtering.ipynb