CS8803-O01: Artificial Intelligence for Robotics
Final Project Report

Maruf Md Maniru Abbasi Jeremy S Fairbank John W Heath Barrett H Jones

1 Introduction

In the world of robotics, predicting a moving object's future location is a challenging but important path-planning task. Predicting movement is incredibly useful for capturing rogue robots such as the hypothetical robot in our Runaway Robot Project or rendezvousing with a self-driving car for refueling purposes [1]. Predicting movement necessarily requires historical data or measurements from the object to be tracked. Additionally, with most robots, one of the greatest tracking challenges is dealing with noisy, stochastic measurements. Therefore, prediction depends upon the use of techniques such as filtering, smoothing, or machine learning in order to remove noise and find patterns in the data.

In order to obtain real world experience with path-planning, we have undertaken the objective of predicting the movement of a small HEXBUG robot in a wooden box. In this report, we present an approach based on historical trajectory smoothing and similarity matching. We believe that despite the HEXBUG's seemingly erratic movements, it demonstrates repetitive motion patterns that can provide useful historical data for predicting its future location.

We discuss our main algorithm in Section 3. Section 2 explores the problem of predicting HEXBUG movement more thoroughly. Section 4 describes other algorithms and approaches we considered. Section 5 provides the results of our main algorithm on test data. Finally, Section 6 compares our main algorithm with the other algorithms, expounding upon its effectiveness over the others.

2 Problem

As mentioned, the problem in this project is predicting the movement of a tiny HEXBUG Nano robot. The provided data for the problem are 10 videos of the HEXBUG's moving within a wooden box and text files that contain the HEXBUG's centroid position relative to the box's boundaries at each frame of each video. There is also a candle placed in the middle of the box to additionally disrupt the movement of the HEXBUG. In this problem, we must utilize the HEXBUG's centroid position data in order to predict its positions over the final two seconds of the video.

The HEXBUG moves by vibrations, creating erratic but forward-progressing movement. Figure 1 depicts a few HEXBUG Nano robots courtesy of [2]. Notice that the HEXBUG Nano does not lend itself to precise

movements thanks to its shape and vibratory propulsion.

Figure 1: Various HEXBUG Nano robots.

Even though it appears that it can stay on a straight path, due to the nature of its motion, the HEXBUG has a tendency to veer itself off course. From a section of the video `test01.mp4`, Figure 2 depicts the HEXBUG's moving in a seemingly straight trajectory, only to suddenly curve to the right. This appears to suggest that the HEXBUG's movement may be partially stochastic.

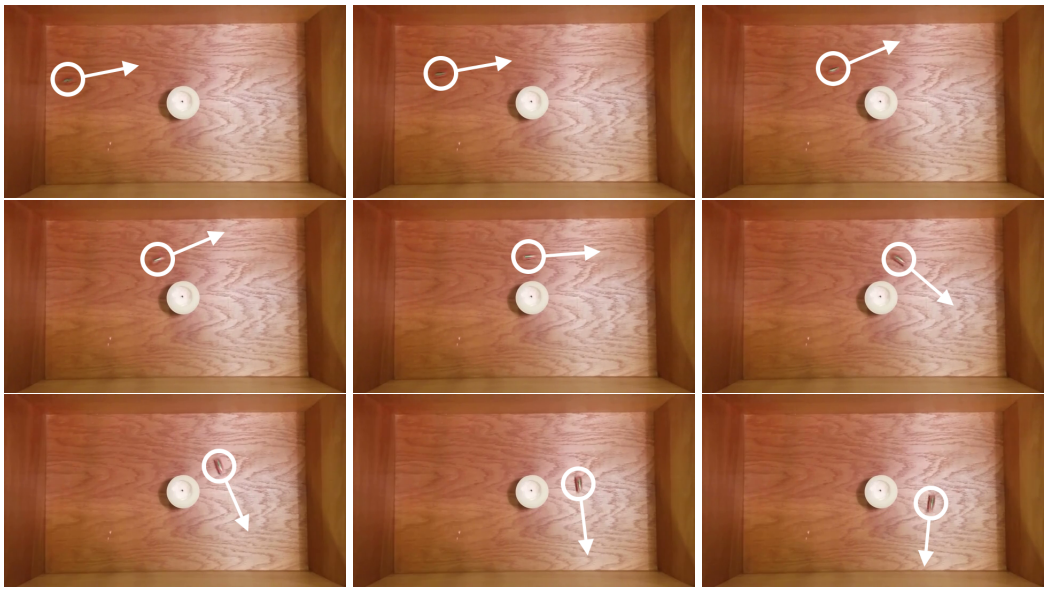

Figure 2: HEXBUG veers to right from a mostly straight path without hitting an obstruction. Taken from test01.mp4 between 15.5 and 17.5 seconds.

Collisions with the walls of the wooden box and with the candle further complicate the movement of the HEXBUG. Collisions with the wall and corners of the box either cause the HEXBUG to change its heading direction immediately or after a brief moment of being "stuck" on the wall. Whenever the HEXBUG does change its heading direction, it does not appear to always be by the same amount. That is, sometimes the HEXBUG appears to change its heading by say 135°, while other times it seems to only change by a few degrees. Collisions with the candle appear to cause the same stochastic change in heading direction as well. Figure 3 shows a sample of the HEXBUG's becoming stuck in the top left corner for about 0.5 seconds.

Finally, the HEXBUG occasionally flips over on its side or back, causing it to stay mostly static. This greatly disrupts trajectory information, injecting additional noise into the data. This is most pronounced in the

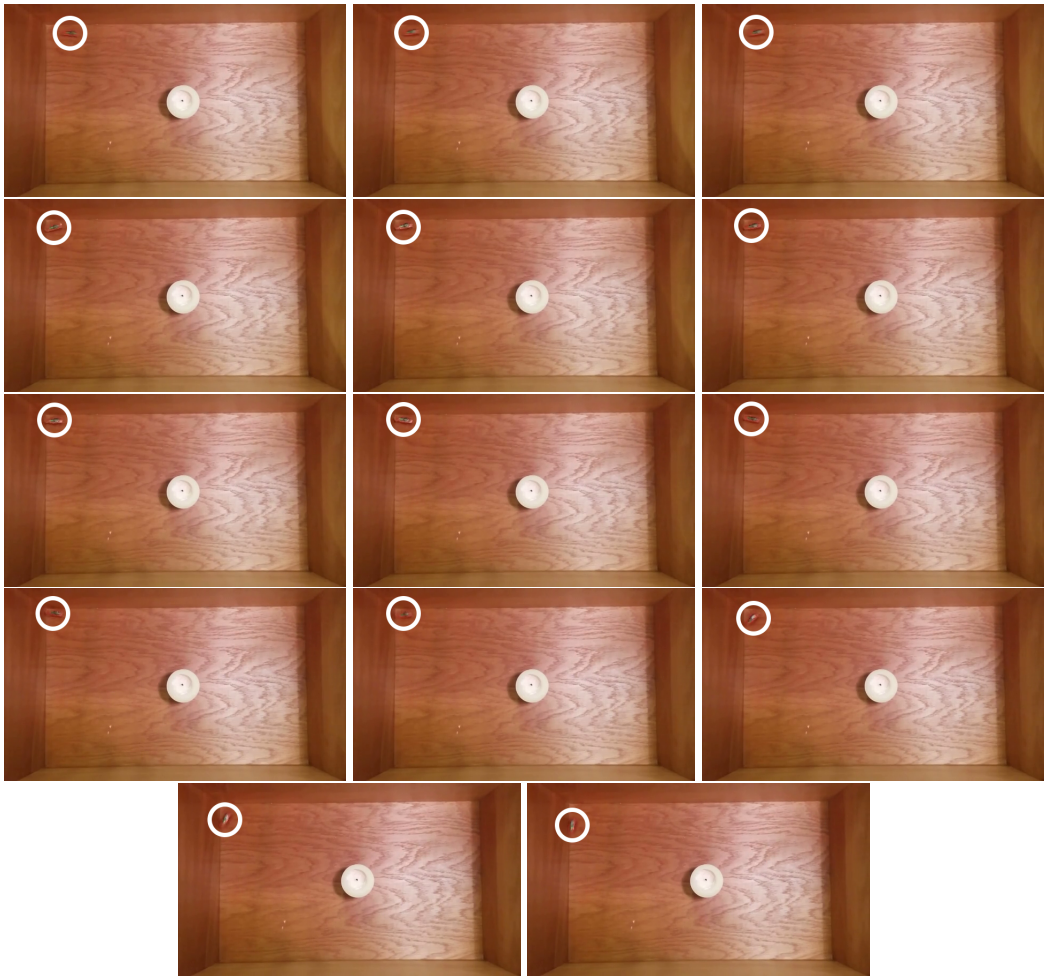

Figure 3: HEXBUG becomes stuck on the wall for a brief moment. Taken from test01.mp4 between 2.5 and 4.5 seconds.

`test07.mp4` video where at approximately 35 seconds, a collision with the top wall causes the HEXBUG to flip and remain on its side and back for the remainder of the video. Figure 4 showcases an example of the flipped HEXBUG from this video.

Given its stochastic nature and its surroundings, the HEXBUG might appear to have unpredictable movement. However, closer examination of the centroid data reveals some common trajectory patterns of movement. Figure 5 provides the raw centroid data for the HEXBUG in the first 3 videos. Even though each video contains unique positions, notice the trajectory patterns that seem to emerge in the scatter plots. We see that many trajectories appear very similar from one video to the next. This isn't a rigorous examination, but we believe the HEXBUG repeats certain motion patterns, which we can exploit to predict future motion.

Figure 4: HEXBUG flipped on its back at approximately 39 seconds into the test07.mp4 video.

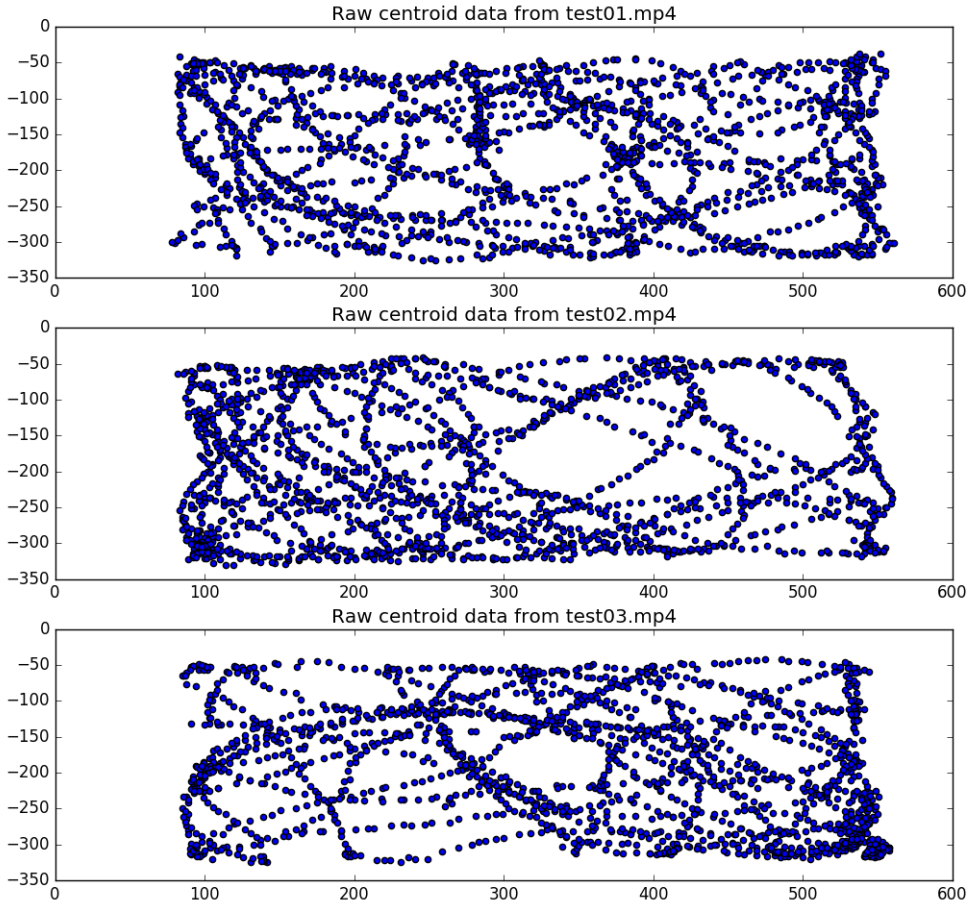

Figure 5: Scatter plots of HEXBUG centroid data from test01.mp4, test02.mp4, and test03.mp4 from top to bottom.

3 Main Algorithm

3.1 Overview

As briefly described in Section 1, our main algorithm utilizes historical trajectory smoothing and similarity matching. Our algorithm is based on ideas from [1] where they present a "data-driven approach to predicting motion". They suggest that "moving objects often follow typical motion patterns and that historical data about their movement can be used to predict their future locations" [1]. They show that moving objects typically follow trajectories that can be divided into repetitive subtrajectories, or motion patterns. They identify and cluster these subtrajectories via a four-step algorithm that is based on expectation-maximization and k-lines projection algorithms. Once they cluster the subtrajectories, they compute representative line segments for each cluster and feed the representatives into a Hidden Markov Model (HMM) in order to predict future motion.

Our approach is similar in spirit to [1], but uses a drastically simpler algorithm. We incorporate the idea of trajectories and subtrajectories but don't cluster subtrajectories into representative line segments. Instead, we build a historical knowledge of subtrajectories and the longer trajectories to which they belong. Also, rather than predicting future motion with an HMM, we match subtrajectories with the historical subtrajectories and reuse the best matching trajectory for our prediction.

We cover this more in Sections 5 and 6, but our results show that the HEXBUG's motion patterns are definitely repetitive, allowing our algorithm to match very similar trajectories. Figure 6 provides sample results for video `test10.mp4`. Yellow points indicate the actual path of the HEXBUG of the last 60 frames, and red points indicate the predicted path. Notice how close these trajectories match, especially for the several initial points in each trajectory. Most of the other videos had very similarly matching trajectories as well.

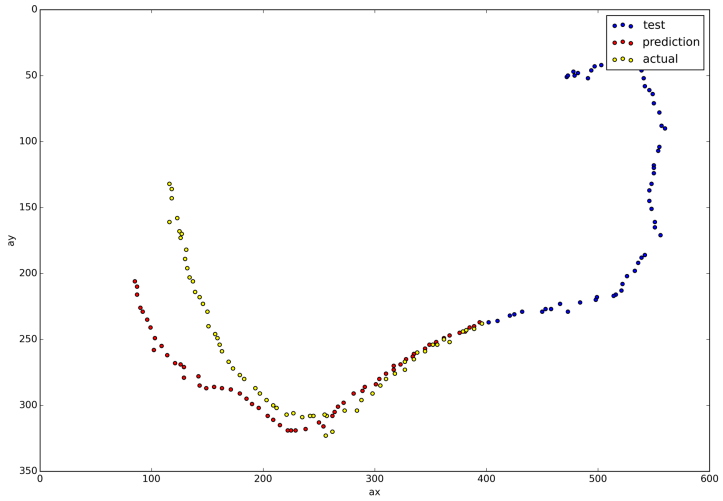

Figure 6: Unsmoothed actual and predicted trajectories from the last 60 frames of test10.mp4.

Figure 7 supplies additional insight. Note that although the actual and predicted trajectories never overlap, they have very similar arcs. This shows that despite its erratic motions, the HEXBUG can still move in repeatable patterns.

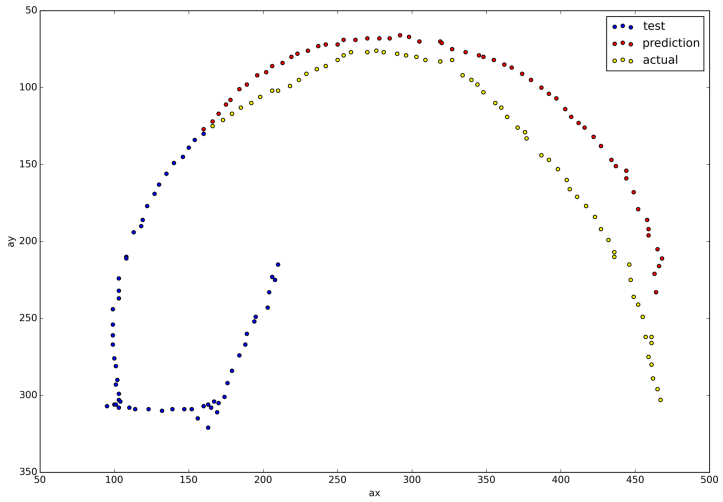

Figure 7: Unsmoothed actual and predicted trajectories from the last 60 frames of test04.mp4.

3.2 Implementation

Our algorithm is straightforward and short, coming in at only 83 lines of Python code. Our algorithm contains four steps: reading training and input data, calculating the heading angle of the HEXBUG at each data point, building a knowledge base of trajectories and subtrajectories, and making a prediction based on matching the last subtrajectory to the closest subtrajectory in the knowledge base.

3.2.1 Reading Data

Reading in training and input data is a simple file-reading approach. We use a helper function called `read_data` that takes a file name as an argument. The function expects the file to be in the form of comma-separated x-y coordinates on each line of the file. The function reads each line and creates a coordinate pair as a tuple, ensuring to cast the values to integers. Listing 1 shows the source code for the `read_data` function.

Listing 1: Function for reading in comma-separated x-y coordinates from a file.

```
def read_data(filename):
    with open(filename) as f:
        lines = f.readlines()
        lines = [line.split(',') for line in lines]

        data = [(int(x.strip()), int(y.strip())) for x, y in lines]

    return data
```

3.2.2 Calculating Heading

Our algorithm works quite well using only x-y coordinates for subtrajectory matching. However, we were able to slightly improve our results by including the HEXBUG's heading direction at each data

point. After reading in training data and the input data, we calculate the heading for each point via the `calculate_headings` function. Listing 2 provides the source code for the `calculate_headings` function. We convert the raw data into a numpy array and then compute the change between coordinates by subtracting each point by the point before it. Finally, we calculate the arctangent of the change in x and y values to yield a heading angle for a given point.

Listing 2: Function for calculating the heading directions of a list of points.

```
def calculate_headings(raw_data):
    data = np.array(raw_data)

    prior = data[:-1]
    leading = data[1:]

    deltas = leading - prior

    thetas = np.arctan2(deltas[:, 1], deltas[:, 0])[:, None]
    thetas = np.vstack((thetas, thetas[-1]))

    new_data = np.hstack((data, thetas))

    return map(tuple, new_data.tolist())
```

3.2.3 Building the Knowledge Base

Our algorithm's knowledge base is comprised of the twenty-minute training data as well as the test data from the 10 input videos. Once we load up all the aforementioned data, we chunk it into a list of tuples that associate a subtrajectory with the longer trajectory to which it belongs. The subtrajectory is always the first 7 points of the longer trajectory. (Through experimentation, we obtained mostly the best results by using the subtrajectory length of 7.) Listing 3 shows a snippet from our `build_knowledge` function, creating the initial knowledge base from the training data.

Listing 3: Snippet that shows chunking data points into subtrajectories associated with larger trajectories.

```
knowledge = (
    [(training_data[i:i + SUBTRAJECTORY_LENGTH],
      training_data[i + SUBTRAJECTORY_LENGTH:
                    i + SUBTRAJECTORY_LENGTH + 60])
     for i in range(len(training_data) - (60 + SUBTRAJECTORY_LENGTH))]
)
```

Notice that we loop over all but the last 67 (`60 + SUBTRAJECTORY_LENGTH`) points. For each point in the loop, we grab the next 6 points, creating the starting subtrajectory. The subtrajectory becomes the first element in the tuple. Then, we grab the next 60 points after the subtrajectory and use those as the second element in the tuple. These 60 points will be the paths used for making a prediction for the last 60 frames of a video.

3.2.4 Making a Prediction

Finally, in order to make a prediction we match the last 7 points of a video with the closest subtrajectory in our knowledge base. To deal with the noisy, erratic measurements, we smooth over the last 7 points prior to matching.

Our matching process calculates the L2 distance between the smoothed 7 points and every subtrajectory in the knowledge base via the `calculate_error` function. We use three dimensions in the L2 distance calculation: x coordinate, y coordinate, and heading angle. We then pick whichever subtrajectory yields the smallest L2 distance, or error. Once we've selected the subtrajectory, we can pull out the remaining 60 points for the remainder of that trajectory. To improve our results and reduce noise, we smooth the 60 points and use the smoothed path as the final prediction. Our smoothing function is called `smooth` and is based on the path smoothing lectures and function from our lesson on PID control.

4 Other Algorithms

Our team took a divided approach to the prediction problem whereby each person worked independently, providing ideas and feedback to one another. This approach yielded multiple algorithms, some of which were also great solutions to the problem.

4.1 Trajectory Creation from Constant Velocities

One algorithm was based on human expectations of the robot's motion. We noticed that the HEXBUG typically followed smoothed curved paths when running in open space (i.e. not colliding with the walls of the box or the candle). This approach is similar to our main algorithm in that it recognizes that the HEXBUG will repeat certain motion patterns. Additionally, this approach recognizes that collisions with the walls and corners of the box and with the candle result in unpredictable behavior. We had wanted to incorporate ideas from [3] to deal with collisions better but did not have time.

The gist of the algorithm is to create an entirely new trajectory based on the position, heading, translational velocity, and angular velocity of the HEXBUG. We compute the heading of the HEXBUG at every position by computing the arctangent of its change in position over every few frames. We assume relatively constant translational and angular velocities and compute them over the last several frames of the video. We then use the last available position of the HEXBUG along with its heading to build a new trajectory. We loop 60 times, taking the current position and heading and updating those values by the angular and translational velocities.

After generating the new trajectory, we check for boundary conditions to ensure the trajectory does not go beyond the walls of the box. Because we had not yet incorporated special handling for walls and corners, we simply mirror the position of the HEXBUG at a boundary if it goes beyond the boundary. For example, if the x coordinate of the HEXBUG is 601, but the boundary value along the x axis is 600, then we change the x coordinate to 599.

4.2 Trajectory Creation from Hidden Markov Model

Another algorithm was also based on creating a new trajectory from historical data but incorporated an HMM for predicting each position of the new trajectory. After thorough analysis of the videos, training data, and test data, we determined that there were 755 distinct turning angles possible. Thus, we built a state transition matrix comprised of the probability of transitioning from one turning angle to another for any given position of the HEXBUG. Take this example state transition matrix:

	t1	t2	t3	t4
t1	A		C	
t2				
t3				
t4				

Here we have 4 different turning angles: `t1`, `t2`, `t3`, and `t4` for a given point x_i, y_j. The example state transition matrix shows the probabilities `A` and `C` of transitioning from `t1` to `t1` and `t1` to `t3`, respectively. We compute probabilities via maximum likelihood. Assume we see the following sequence of turning angle changes in the data:

```
t1->t3->t3->t2->t4->t1->t1
```

Then, we would say that `A = 0.5` and `C = 0.5` since we have two transitions from `t1`. We tried to use the entire data set to reach a stationary distribution for each direction.

We use an expectation maximization algorithm to estimate the model parameters and construct a Markov Chain (the state transition matrix). We did not try Laplacian smoothing but still performed some smoothing of the training data before training the model. Then, we take the last two frames in the test input and compute the heading of the HEXBUG at that point. We also take each possible turning angle from the 755 angles and calculate the probability (via total probability and Bayes rule) of the next turning angle. We utilize the heading and expected turning angle along with a constant translational velocity to compute the expected next position. We repeat this process until we have the last 60 points, yielding us a probability of the HEXBUG taking this path.

Next, we perform an exhaustive search to find a candidate prediction. At each point, we move the HEXBUG toward all 755 turning angles, producing at most 755 possible next positions (this number may be less based on the boundaries of the wooden box and rounding to same location). From each of these new positions, we again send the HEXBUG toward all 755 turning angles. As we progress along each of these exponential paths, if the probability drops below 0.3, then we stop and only keep 100 paths to keep performance in check. Finally, we select the most likely candidate path based on probability. Figure 8 depicts a graph example of picking a path.

4.3 Trajectory Clustering and Hidden Markov Model

We attempted a hybrid approach of [1] and another trajectory clustering algorithm from [4]. The ideas presented in [4] suggested clustering subtrajectories based on position and a labeled heading. The labeled headings generalized specific heading angles to a finite list of `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, and `NW`. Confining angles

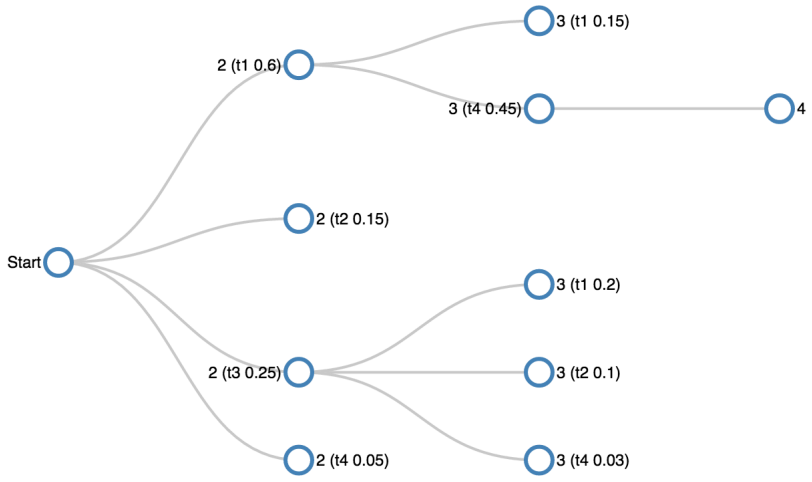

Figure 8: Path selection example. We first turn by t1 and then t4 to reach position 4.

to finite labels helped lump trajectories closer to one another, recognizing that small angle differences were insignificant in finding overall motion patterns.

The original idea for this approach was to utilize the clustering approach from [4], compute representative line segments from it, and then feed those representatives into an HMM similar to [1]. Unfortunately, due to time constraints and inexperience with HMM, we could not expand on this approach to create a useable algorithm for predicting the HEXBUG's motions.

5 Results

We obtained our results with the 10 original input files by removing the last 60 points in each file and only using the first 1740 points as input to our algorithm. This allowed us to mimic not having the last 60 frames of data and thus make a prediction with what data was available. We also used a subtrajectory length of 7. We discuss this further in Section 6, but we feel that these results might indicate a degree of overfitting.

We obtained our results by running our algorithm on the class provided VM on a MacBook Pro host machine. The host machine used OS X El Capitan with a 2.2 GHz Intel Core i7 processor and 16 GB 1600 MHz DDR3 RAM. Typical run times to run a modified grading script for all 10 input text files were approximately 30 seconds. The modified grading script was the instructor-provided grading script with additional plotting code to generate the plots for this report.

Our algorithm's average error on the test videos was **426.575**. To calculate this value, we first compute the error between the predicted 60 points and actual 60 points of each video via L2 distance. The lower the error, the more accurate the prediction is. Then, we drop the worst and best errors (highest and lowest, respectively), leaving only eight error values. Finally, we compute the average of those eight error values to yield the average error. The errors for all 10 videos are available in Table 1.

Table 1: Main algorithm raw results. Error between predicted and actual points for last 60 frames of each test video.

Test Video	Error
test01.mp4	347.253
test02.mp4	430.377
test03.mp4	331.223
test04.mp4	360.053
test05.mp4	1861.788
test06.mp4	476.118
test07.mp4	152.908
test08.mp4	923.146
test09.mp4	220.699
test10.mp4	323.734

Most of the error values were very close to one another. Videos 1, 2, 3, 4, 6, 9, and 10 all fall within a range of 200–500. Videos 5, 7, and 8 were the outliers with errors of 1861.788, 152.908, and 923.146, respectively.

6 Discussion

6.1 Discussion of Results

Looking further at the results in Table 1, we notice some interesting trends. As we saw earlier, 7 out of the 10 videos had an error within the range of 200–500. These are all extremely good results. For example plots for Videos 4 and 7, please refer back to Figures 7 and 6. Note that these figures depict **unsmoothed** paths, but they are still generated from our main algorithm. Again, these examples show how well our algorithm can find similar trajectories and reuse them to make a prediction.

Regardless, our results also potentially indicate some overfitting. If we modify our testing setup to use a different set of 60 frames from each video instead of the last 60, we begin to see wildly different results, usually worse than the previously presented results. When we changed which 60 frames we used over several trials, we saw our average error grow to values as large as 1167.974. Over these trials, we also experimented with changing the subtrajectory length. We took the average of the average error over all trials and found that a subtrajectory length of 10 might yield slightly better results. However, a subtrajectory length of 10 performed worse than a subtrajectory length of 7 when we examined the original setup that removed the last 60 frames of each video. Thus, it was hard to decide what subtrajectory length might work the best for the real missing 60 frames that we were not provided. Because the average with a length of 7 was not orders of magnitude worse than a length of 10, we decided to stay with a length of 7.

Even with the subtrajectory length of 7, we still had outlier videos that either performed poorly or surprisingly well. Video 8 and especially Video 5 performed poorly. Figures 9 and 10 depict the plotted results for Videos 8 and 5, respectively.

In Video 8, we start off matching very well, but the actual and predicted trajectories begin to diverge

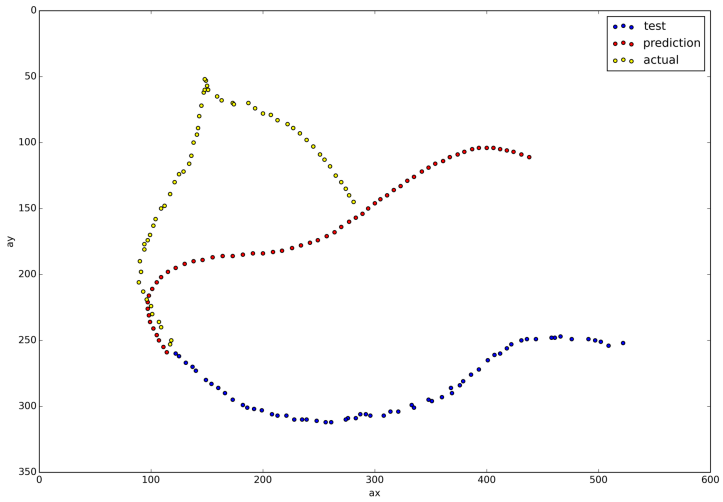

Figure 9: Actual and smoothed predicted trajectories from the last 60 frames of test08.mp4.

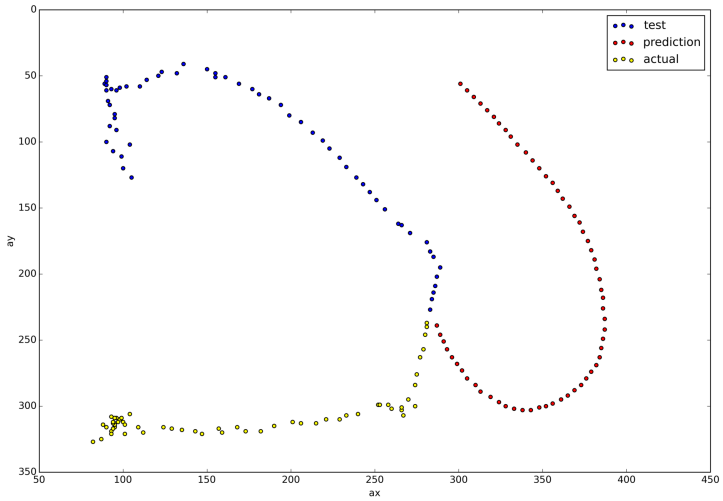

Figure 10: Actual and smoothed predicted trajectories from the last 60 frames of test05.mp4.

significantly. In Video 5, while the subtrajectories match, they belong to vastly different overall trajectories that go in different heading directions! This is the downside to our algorithm and the erratic nature of the HEXBUG. It's possible to have strongly matching subtrajectories but different overall trajectories because the HEXBUG could randomly change direction from what we expect according to historical data. It's possible that a different subtrajectory length could have produced better results on this video.

In contrast, Video 7 performed exceptionally well. As we noted about Video 7 before in Section 2, from 35 seconds to the end of the video, the HEXBUG remains on its back, randomly vibrating around the top left of the box but in a very limited area. Because the training data for our test includes all of Video 7 except the last two seconds that are test data, we have approximately 23 seconds of training data showing the bug moving similarly in a very limited area. Our algorithm is easily able to match the final two seconds of Video 7 to training data, yielding a very small error compared to errors determined for a "running" bug. Even though we like the low error for this video, it was also our best scoring video, meaning it did not figure into the average error. Please refer to Figure 11 to see the erratic clustering of points from Video 7. Note that the scale differs

significantly from all the other scatter plot figures provided because the points shown are tightly clustered.

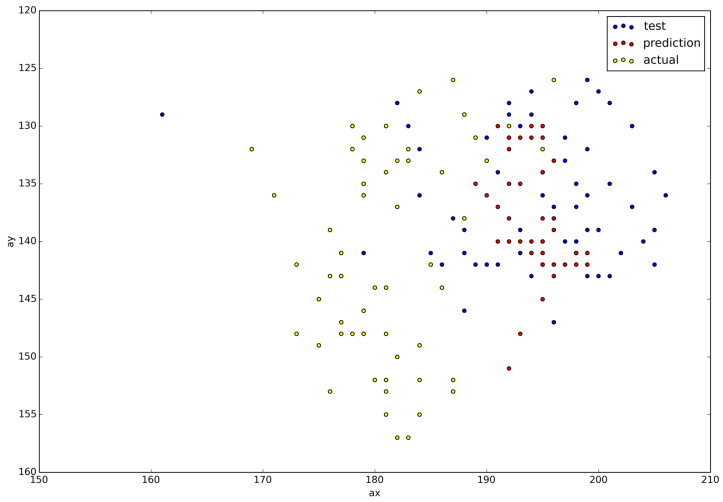

Figure 11: Actual and smoothed predicted trajectories from the last 60 frames of test07.mp4.

6.2 Comparison to other Algorithms

For this section, we will refer to the "Trajectory Creation from Constant Velocities" algorithm as Algorithm 1 and the "Trajectory Creation from Hidden Markov Model" algorithm as Algorithm 2.

While our other algorithms were strong candidates for predicting the HEXBUG'S future motion, we ultimately decided on our main algorithm of matching subtrajectories from historical data due to its stronger performance on our test data. Recall from Section 5 that our main algorithm scored an average error of **426.575** when we tried to predict the last available 60 frames of data from each video. From the same test dataset, our average error with Algorithm 1 was **621.779**, and our average error with Algorithm 2 was **913.85**. While both of these approaches scored well too, we had more confidence in our main algorithm as the most general approach.

Algorithm 1 did perform extremely well on certain videos. For example, on Video 4, it scored 316.605 for its error. Figure 12 depicts these results from Video 4. Notice how well the predicted path (red points) matches the actual path (green points).

However, Algorithm 1 does not perform so well on other videos such as Video 1, receiving a score of 1560.641. Figure 13 depicts the results on Video 1. Notice that the predicted path heads in vastly different direction from the actual path and exhibits a dissimilar arc in its trajectory. This could be attributed to lack of sufficient training data along with noise in measurements.

Ultimately, we think that Algorithm 1 is still a viable approach. We believe that with more training data and potentially incorporating filtering and ideas from an HMM like Algorithm 2, we could refine this approach to work even better.

Unfortunately, we don't have any more raw data or plots to provide for Algorithm 2, but we believe further research and experimentation could prove that the prediction problem is ergodic and that an HMM could solve it with a drastically smaller average error.

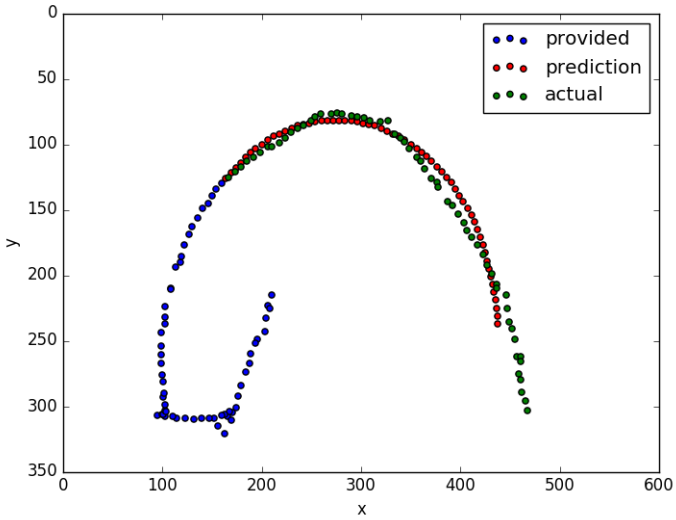

Figure 12: Actual and predicted trajectories from the last 60 frames of test04.mp4 via Algorithm 1 ("Trajectory Creation from Constant Velocities").

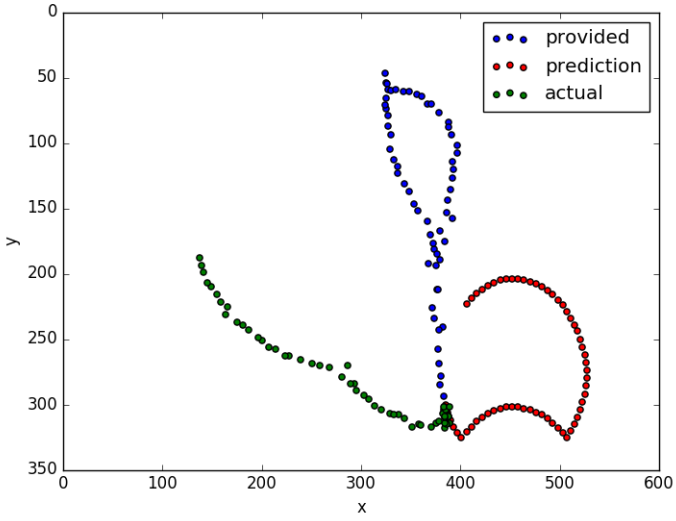

Figure 13: Actual and predicted trajectories from the last 60 frames of test01.mp4 via Algorithm 1 ("Trajectory Creation from Constant Velocities").

Given that Algorithm 1 had not incorporated any filtering or path-planning techniques introduced in our class and that Algorithm 2 had a high error, we decided to stick with our main algorithm, especially since it scored the best on the test data.

7 Conclusion

As we have shown in this paper, predicting the future movement of an erratically-moving HEXBUG is a nontrivial task. We have presented multiple algorithms and their results and justified the selection of our main algorithm. Our main algorithm takes a data-driven approach that hinges on historical trajectory smoothing and similarity matching. Thanks to the wealth of data from the training data and other test videos, we are able to build a knowledge base that allows us to select similar subtrajectories and make predictions about the overall trajectory that the HEXBUG will take.

8 References

[1] C. Sung, D. Feldman, and D. Rus, "Trajectory clustering for motion prediction," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012 [Online]. Available: http://people.csail.mit.edu/dannyf/traj.pdf

[2] Available: http://bit.ly/2gHPzig

[3] P. P. Choi and M. Hebert, "Learning and predicting moving object trajectory: A piecewise trajectory segment approach," *Carnegie Mellon University Research Showcase @ CMU*, 2006 [Online]. Available: http://repository.cmu.edu/cgi/viewcontent.cgi?article=1336&context=robotics

[4] O. Ossama, H. M. Mokhtar, and M. E. El-Sharkawi, "An extended k-means technique for clustering moving objects," *Egyptian Informatics Journal*, vol. 12, no. 1, pp. 45–51, 2011.