AI4R Final Project Report

Quansheng Yang, Anthony Lee, Xin Xiao, Robert Houck

Introduction

In this project, we are given 60 seconds of movement video of Hexbug nano micro robot with a size of 1.5 mm in a wood box in size of about 350 pixels x 600 pixels. Based on the first 60 seconds of video data, the behavior of the robot could be learned to build a robot movement model. This model in turn is used to predict next 2 seconds movement of the robot. Each second of data are divided into 30 frames with date format of x and y coordinates, the central position of the robot.

Analysis of the robot movement behavior

Use the provided training_data.txt, which includes 36,000 data points, it clearly shows that the robot could explore entire wood box space, with coverage ratio of 0.215. This means that the robot almost never visits the same position twice in the data. We then plot its movement tracks for the dataset. The entire free space is almost covered by the robot with little movement pattern.

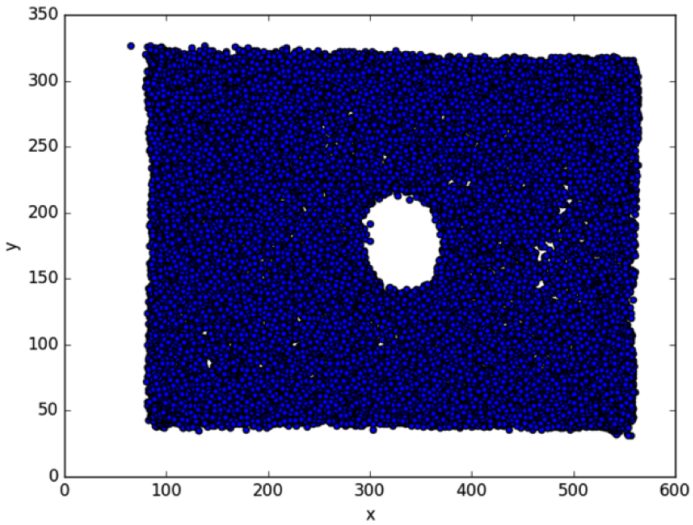


Figure 1 Movement of the Hexbug robot in the wood box with a size of 350 x 600. Each spot represents a location of the robot in a frame. The walls of boundary and central candle could be inferred as wide white space in the image.

Algorithms explored to model the robot behavior

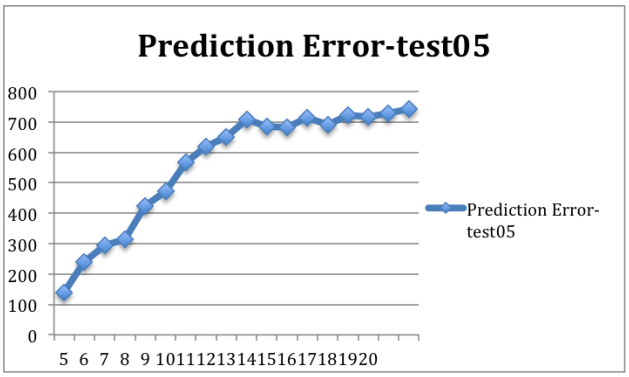
I. KNN Algorithm

I.1. Introduction

First approach we try is to cluster the robot behavior into several categories based on four parameter, velocity, turning angle, x and y positions using k Nearest Neighbor algorithm (KNN). First, the robot behavior could be clustered into k clusters based on the first 1800 data points. To predict the next 60 points, the behavior of the next movement is extracted from one of the clusters based on the most recent robot movement. The KNN algorithm provides the information of k nearest neighbors of a moving object. The KNN algorithm finds a set of nearest moving objects to a given location at a given moment.

The complete process of KNN algorithm is shown as follows:

- Normalize x, y coordinate of each training data to the range ([0,1], [0,1]).
- Calculate the velocity and angle from each point to the previous point.
- Build a new 4D training data (x coordinate, y coordinate, velocity and angle).
- Set up 60 vectors (dx,dy) up to 60 time steps from each known coordinate.
- From the given starting point, find k most similar points in the training data.
- Calculate the mean dx and dy of these k points.
- Add the mean dx and dy to the given starting point.
- Return 60 denormalized predicting x, y coordinates.



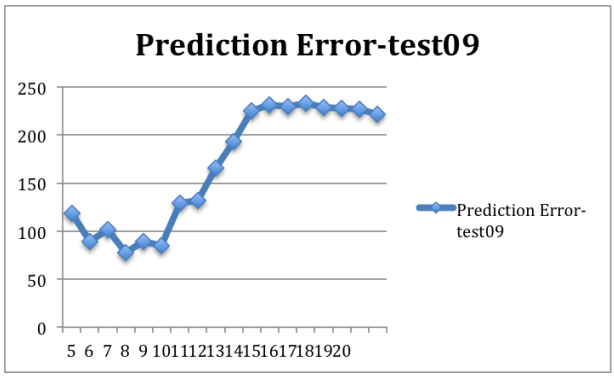


Figure 2. Plot of k value vs average Prediction Error predicting 1700-1759 points, 1701-1760 points and 1702-1761 points. Left panel from test05.txt and right panel from test09.txt.

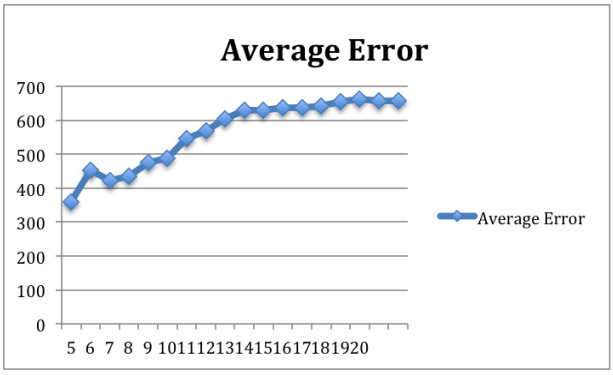


Figure 3 Plot of k value vs average Prediction Error predicting 1700-1759 points, 1701-1760 points and 1702-1761 points from test01.txt to test10.txt.

From three figures above, k value influencing prediction error will change if the given starting pointing is different, especially when moving object is around the corner or around obstacles. In general, when k value is between 5-8, the error is relatively low. So the setting value of k is 7.

I.2. Algorithm Results

Prediction Error	Running Time
529.6838942	*0.271251917*
534.0883458	*0.218225956*
1325.612573	*0.220170975*
1119.882261	*0.224828005*
157.8361492	*0.214833021*

177.5087635	*0.212402105*
72.61542536	*0.216380119*
445.9527254	*0.252189875*
73.57554696	*0.208832979*
167.5015754	*0.210287094*
Average: 460.425726	**Sum**: 2.249402046

Table 1 Prediction Error and running time predicting 1700-1759 points from test01.txt to test10.txt

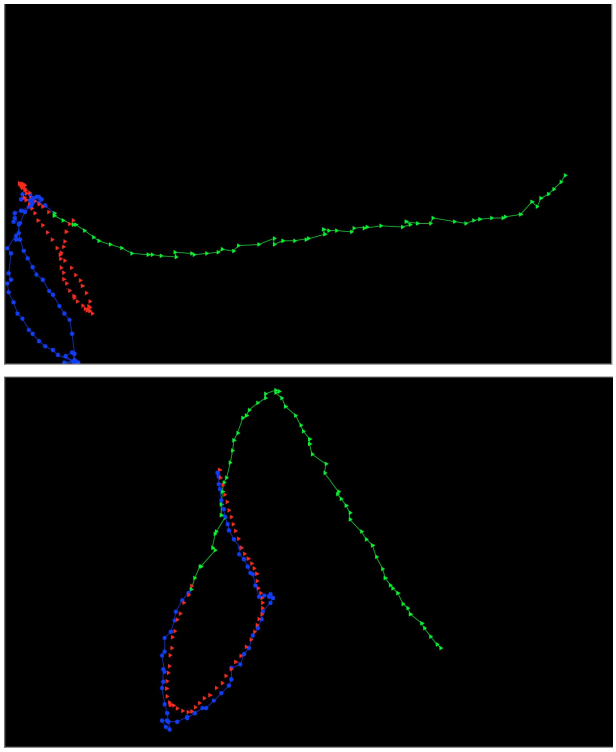


Image 1 Green point-60 old points, Red point-60 predicting point and Blue point- 60 actual points. Top, from test03.txt. Bottom from test09.txt

I.3. Algorithm Performance

Based on results, KNN algorithm obviously is effective especially when the training data is large. And KNN algorithm is greatly robust to noisy training data.

But KNN algorithm also presents some difficulties. It is hard to determine k value especially when the object moves randomly and obstacles exist. Computation cost is high when training data is very large. For example, running time was close to 15 seconds when running 10 input files created from training_data.txt.

II. Other Algorithms

II.1 Defined distance and angle

The algorithm for a robot in an ideal condition is that the robot moves a defined distance and angle in a free space. When the robot is collided with the boundary such as the walls of the wood box or the candle in the central position, it chooses a new angle. Although it is clearly not the case for the Hexbug nano robot, as it is described in the project document, it displays “persistent random behavior”.

One algorithm is to choose a fixed distance using average distance the robot travels, and the change angle could be either the average turning angle of the robot or randomly sampled based on the angle change distribution in the range of - pi to pi. Set the range such that the robot has to move forward and thus backward movement is not allowed. Next position could be easily predicted as x’ = x + d * cos(angle) and y’ = y + d * sin(angle). (x, y are the given coordinate, x’, y’ are the predicted coordinate, d is the average travel distance, and angle is the bearing of the robot.) For this algorithm to work accurately, the boundary positions have to be determined. Otherwise, the robot could either move well outside of the wood box or move directly through the central candle.

There are two approaches to learn the obstacle positions. First approach is to extract the boundary from the movement data points in the training_data.txt. From figure 1, it is obvious where is the boundary and where is the candle. We design a function that gives whether a position is an obstacle or not by counting how many grids in its adjacent 25 grids the robot has already visited. Based on the coverage ratio of 0.215 of the training

data, the robot average visits about 25 * 0.215 ~ 5 grids. We use 3 and above as a threshold for free space and 2 and below counts as obstacles.

The second approach is to learn the distinct behavior that the robot exhibits when the robot collides with a boundary. This behavior is not discernible from our studies. The robot could be struck in the corner for a while or scuttle away quickly. This may be caused by persistent random angle change of the robot. Our theory is that the robot has no special behavior on the boundary. It randomly chooses a new turning angle and tries to move to the new position. If the next possible position is on the boundary, it just fails to move. So this algorithm heavily depends on the learning of the boundary from the training data set. Without the training data set this algorithm is not reliable and could move far away into immovable boundary space.

II.2 Histogram Filtering

We decided to attempt to use a histogram filter to solve the problem. It was expected that this method would not be very accurate due to the observed, seemingly random, motion of the hexbot. It was seen as a good idea to continue working on it for comparison against the KNN algorithm.

II.2.1 Models

Three models would be needed, one for the arena and two for the hexbot. The arena model was used to identify if a point was within the arena proper and if it was in a hindered location or unhindered location.

II.2.1.1 Arena Model

The arena model was created using edge points to generate lines for the edge detection and an ellipse in the center. Once this model eliminated outlying points (max of 30 points) a buffer was added creating two different areas: hindered and unhindered. Since the hexbot would operate differently when connecting with a surface, the hindered buffer was created. Points identified in this area were treated differently.

II.2.1.2 Hexbot Models

Using the points from the training data, histograms were created from in two areas. As stated in the Arena Model section, the two areas, hindered and unhindered, were separated and histograms created. Two histograms were generated for each area: distance and angle of change. The angle of change is the difference of bearing from the previous set of points. Due to the time it takes to generate these histograms, they were hard coded to reduce run time.

II.2.1.3 Algorithm Execution

With our models in place, the algorithm could be executed. This method is naive in its approach. Other than if a points is in the hindered area or not are the only things taken into account. Taking the last two points the algorithm calculates the bearing and begins predicting new points.

Using the last available point, it is decided if it is in the hindered or unhindered area. This will determine which histograms are used to predict the next point. A random value is selected based on the histogram weights for both distance and angle of change. After updating the bearing with the angle of change, changes to the previous point are made to generate the next predicted point. In the event that the generated point is outside of

the arena, the algorithm is ran on the same points again until the generated point is in the arena.

II.2.1.4 Algorithm Results

When running all 10 input files, completion time was close to 6 seconds with an error rate between 1150 and 1450. The random nature of the algorithm lends itself to inaccuracy as shown by the error rate. The lack of context inherent in this model leaves little room for improvement.

II.2.2.1 Mean Selection with Histogram Filtering

The second method we attempted was more deterministic. While continuing to use the models presented in the previous section, it dealt with the means of distance and angle of change instead of the histograms.

II.2.2.2 Algorithm Execution

The algorithm's only deviation from the above is the use of the means instead of random weighted histograms. The use of the means will produce a consistent prediction path. If a generated point is outside of the arena, the random weighted histogram method is used to generate a point that is in the arena. It then will resume using the means.

II.2.2.3 Algorithm Results

This algorithm also takes approximately 6 seconds to complete with an error rate between 1250 and 1310. The reduced error range is expected as less use of random elements. No increase in accuracy was expected.

III. Algorithm Comparison

Robot model	Fixed distance and fixed travel angle	Fixed travel distance and random angle	Random distance and random angle	KNN
Average error	1677	1250 - 1310	1150 - 1450	772
Model property	Deterministic	Random	Random	Deterministic
Run time	4 s	6 s	6 s	15 s

Table 2 Test results on 10 test cases created from training_data.txt

IV. Conclusion

To solve this problem, we propose an effective KNN algorithm in the obstructed space. The 2D input data is transformed into 4D features, velocity, turning angle, x and y positions. L2 error is used as a tool to validate the accuracy of KNN algorithm. We evaluate KNN algorithm by comparing other proposed algorithms. The KNN algorithm is shown to outperform other algorithms significantly in our all tests.