



Artificial Intelligence for Robotics - Project 2

Fall 2016

Team 14

Dhiraj Dhule	ddhule3@gatech.edu
Pavan Karkun	pkarkun3@gatech.edu
Piyush Makhija	pmakhija3@gatech.edu
Vishal Krishna	vkrishna7@gatech.edu

Overview

This project provides methods for prediction of the motion of a hexbug robot trapped in a box. The obstacles are the edges of the box and also a circular (from top view) candle placed at the centre of the box. We use AI and machine learning methods to process the motion of the hexbug for certain amount of time. Supervised learning methods are used to generate models of the motion characteristics of the hexbug. The models are then used to predict the trajectory of the robot. We use the centroid data provided in the `train_data.txt` to train the models and predict motion for next 60 frames (2 seconds) for each of the 1800 frame (60 second) `test##.txt` files.

Problem Statement

The problem statement can be briefly defined as prediction of movements of a commonly available micro-robot (Hexbug) for each of the given 10 test scenarios, given sufficient data for us to observe its behaviour and learn from it. The hexbug is placed in a wooden box with a candle at the center of the box. The hexbug runs in a straight line and bounces off any obstacle it encounters.

We are provided with Video data for 60 seconds taken at 30 frames per second(fps) of hexbug's movements in each of the 10 test cases and are tasked with prediction of the next 2 seconds (60 frames) of its movements based on this knowledge. We are also provided with extracted centroid integer coordinates for all test case videos and 1200 seconds of training data in video and extracted centroid format.

The objective of this problem is to apply methods learnt in the course and to apply them to a practical problem where data is messy and noise prone.

The final code file is named "*finalproject.py*" and can be run from command line by specifying the input test case file. E.g: `python finalproject.py test01.txt`

Goals

Given the centroid locations/video data representing/showing the locations/movements of hexbug in each frame of the video worth of 60 seconds (1800 frames), predict the location of the robot in next 2 seconds (60 frames).

Hexbug Robot

The HEXBUG Nano robot is micro-bot which operates on physics of vibration to propel forward. It responds to stimulus from its environment and adjusts course/turns based on this stimulus and continues moving straight ahead in absence of external stimulus. Its motion seems to be ergodic which suggest statistical methods may prove to be effective in predicting its motion.

Approaches considered

1. 2D Kalman filters
2. Machine Learning approach using Gradient Boosting
3. Machine Learning approach using KNN algorithm (different values of K)
4. Machine Learning approach using Decision Tree Regression
5. Machine Learning approach using Extra Tree Regression
6. Machine learning approaches using Support Vector Machine

Python Libraries Used

Generic Python libraries such as numpy and scipy have been used. We have refrained from using opencv or image/video processing in general since centroid data for all test cases and training file has already been provided.

Scikit-Learn (sklearn) library has played a major role in our implementation. Sklearn library's implementation of different machine learning algorithms was directly used in our solution to Project 2.

Proposed Solution

This section describes the flow of our solution code. The solution code file *"finalproject.py"* takes a command line input of *"test##.txt"* where ## can be any number between 0~10. It reads *"training_data.txt"* provided in project resources and writes the prediction results for next 60 frames in *"prediction.txt"* file in the same folder as *"finalproject.py"*.

Flow of Code is as follows:

1. Read training data from *"training_data.txt"*.
2. Get input test file (*"test##.txt"*) from the command line input.
3. Instantiate a *"regressorBasedPredictor"* class. This class contains all operations related to prediction of next 60 frames for given command line input file.
4. *"Process_training_data"* method from the *"regressorBasedPredictor"* class is called. This method is called for storing/operating over training data.
 - a. Here some trivial calculations like finding max/min of input x and y coordinates is done.
 - b. Input x,y coordinate data is normalized and used to calculate velocity and heading direction. All of the following 4 features are then stored together in list:
 - i. X coordinate at previous time frame
 - ii. Y coordinate at previous time frame
 - iii. Hexbug's current velocity
 - iv. Hexbug's current heading direction
 - c. More data processing and storing to different variable operations to be used for model fitting operations.
5. *"Process_test_data"* method is called next. This method is called for operating over command line input test data.
 - a. Store test data.
 - b. Normalize input x,y coordinate data. Calculate velocity and heading direction and store as a list with these 4 features
6. Next step is to select which sklearn library's Machine Learning regression method is to be used. *"regressor"* variable is used to select that.
 - a. *regressor = 0 => KNN*
 - b. *regressor = 1 => Decision Tree*
 - c. *regressor = 2 => Expression Tree*
 - d. *regressor = 3 => PassiveAggressiveRegressor*
 - e. *regressor = 4 => LogisticRegression*
 - f. *regressor = 5 => Boosting*
 - g. *regressor = 6 => NuSVR*
 - h. *regressor = 7 => LinearSVR*
7. Now we run our *"regressorPredictor"* method and store results in a list. This method runs the machine learning regression using the training and testing data we have already provided as input to the selected machine learning model.
 - a. Select the type of machine learning regressor based on *"regressor"* variable input.
 - b. Do model fitting using the training and test file data.
 - c. Predict the next action/coordinate location for hexbug

- d. Denormalize or change the output data format to integer x,y coordinates and return coordinate value.
8. Write prediction results from machine learning model to "*prediction.txt*" output file

Testing Method

For the purpose of verification, we modified the our code to predict last 2 (59th and 60th) seconds for each of our 60s input test file. All figures, graphs and tables produced in this report are using the VM environment provided in project resources and follow this testing methodology to check for accuracy and running time analysis of our code with different approaches. This change has been undone in the version of code submitted.

Comparison of approaches

We need to predict the coordinate location values of the hexbug. Using machine learning regression techniques, we can predict the X values and Y values for next 60 frames respectively. We have experimented with various ML techniques to generate a regression model which could predict the hexbug's movement. Some of them were found to be not suitable due to factors like long time taken to run model and predict, bad prediction accuracy, etc. Few of the promising ones are described below with appropriate data and comparison.

Kalman Filters

We considered kalman filters with 2D tracking using x , y , dx , dy initially, where (x, y) represent coordinates and (dx, dy) represent rate of change(velocity) in respective directions. This approach was taught and discussed extensively in this course . Also, Kalman Filter implementation worked for the runaway robot project, but there were no obstacles or external stimuli presented to the runaway robot. Now we have to deal with box edges and corners, obstruction due to candle and most importantly unpredicted bounce off angles from these obstacles.

Modifying a regular Kalman filter code to deal with these conditions seemed to be less promising and highly effort demanding task. Since, by means of observation of given input and training data, we had already concluded that hexbug's motion is ergodic in nature, we moved towards statistical methods.

Gradient Boosting

We used scikit learn's ensemble method - [Gradient Boosting Regression](#) to predict the required centroids. Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

We used least squares regression loss function inside Gradient Boosting and limited the number of nodes in the tree to 3. Other parameters were left as default parameters in sklearn's implementation. Link to the sklearn's Gradient Boosting Regression page is provided above.

Results:

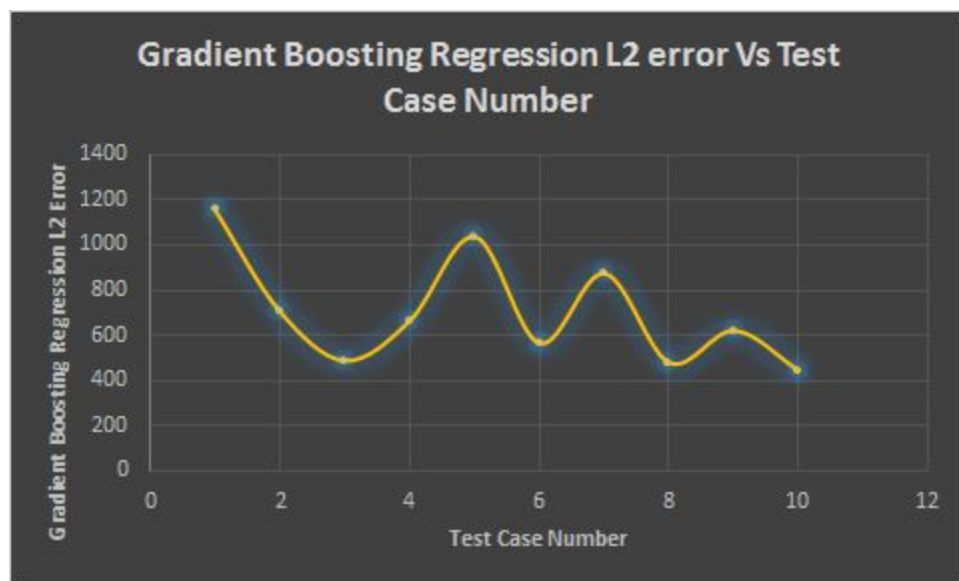
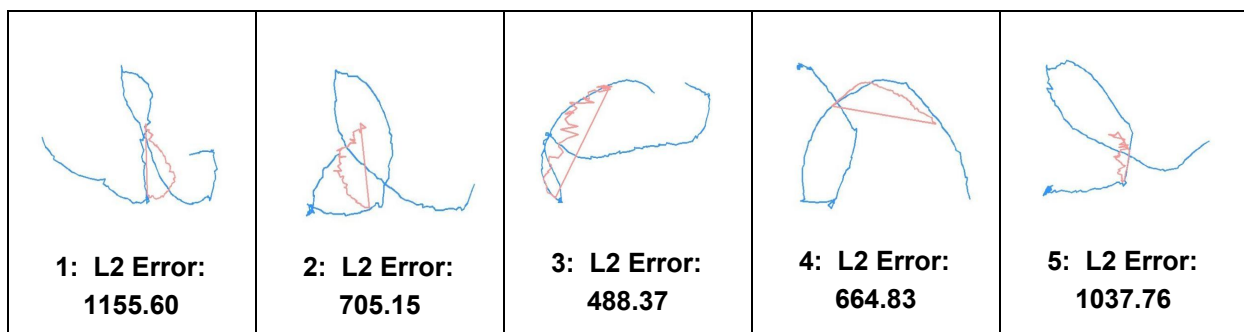


Figure 1: Gradient Boosting Regression L2 error Vs Test Case Number



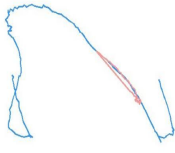

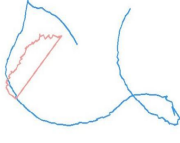
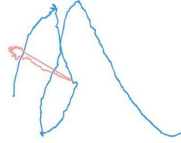
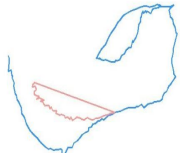
				
6: L2 Error: 565.39	7: L2 Error: 875.34	8: L2 Error: 479.14	9: L2 Error: 621.53	10: L2 Error: 443.96

Table 1: Path Visualization for all Test Cases using Gradient Boosting Regression Method

From Figure 1, we observe that L2 error per test case for Gradient Boosting algorithm, while Table 1 provides us with a visualization of the path out Gradient Boosting algorithm predicts for our hexbug. In the path images we can see that our predictor closely resembles the actual path. With average error as 703.7, this was the best performing approach out of more than 10 different approaches we tried, however time taken by this algorithm exceeded 60 seconds for 10 Test Cases. KNN with neighbours set to 95 was close with average error of 726.0.

K Nearest Neighbors Regression

From `scikit-learn.neighbor`, we used [KNeighborsRegressor](#). k -NN is a type of [instance-based learning](#), or [lazy learning](#), where the function is only approximated locally and all computation is deferred until regression. In k -NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors. We used KD-Tree as the underlying algorithm for our KNN regressor. We have considered many different “number of neighbors” K , and compared the L2 error to estimate the optimal number of neighbors. Figure 2 provides a comparison of KNN Regression performing on different values of K .

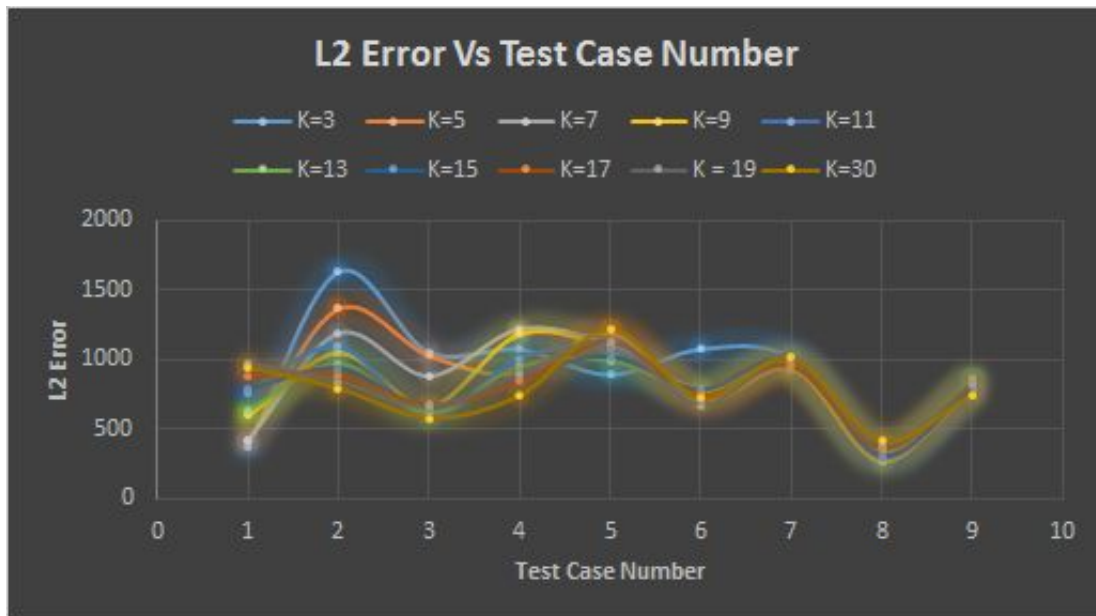


Figure 2: L2 Error Vs Input Test Cases for different values of K using KNN algorithm



Figure 3: Average L2 Error over 10 Test Cases Vs K value chosen for KNN Method

As we can see in Figure 3, KNN with K= 95 is giving us the lowest L2 error. There is a minima observed at K=13, but average L2 error seems to be slowly moving down as K increases further. Table 2 provides a quick visual comparison between KNN with K=95 and gradient boosting algo's path:

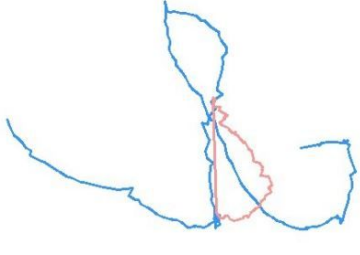
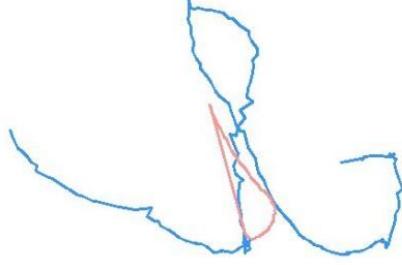
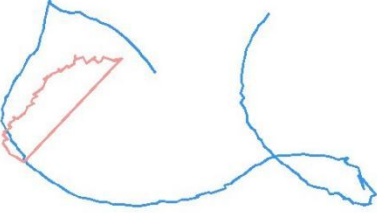
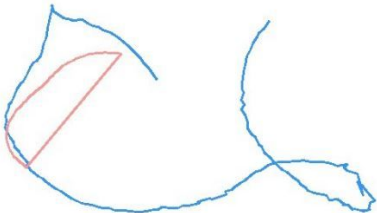
Gradient Boosting	KNN, K = 95
	
	

Table 2: Comparative Path Visualization for Test Case 1& 8 respectively using Gradient Boosting Regression and KNN Regression Methods

Key difference to be noted is, Gradient boosting path is jagged line whereas KNN is smooth. This is expected because KNN output is actually average of 95 nearest neighbors whereas in gradient boosting, each point is a regression learnt output generated by the algorithm.

Decision Tree and Extra Tree Regressor

Decision tree learning uses a decision tree as a predictive model which maps observations about an item (represented in the branches) to conclusions about the item's target value. Decision trees where the target variable can take continuous values are called regression trees. We used Scikit-learn's implementation of decision trees and extra tree regression in our implementation and got the predicted values.

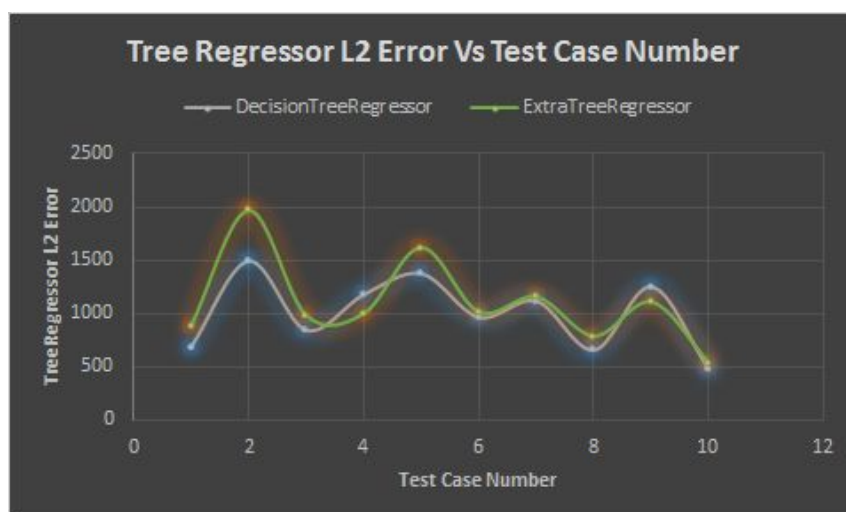


Figure 4: Tree Regressor Method's L2 Error Vs Test Case Number

Figure 4 provides a comparison between the 2 kinds of Tree learning regression methods. As we can see, decision tree and extra tree performs fairly decent, and average error is lower than the baseline. But still they lose in comparison to KNN and gradient boosting techniques we discussed earlier.

Other Approaches

We also tried [Support Vector Regressors](#), [Passive Aggressive Regressor](#) (Type of linear regression), [Neural Networks](#) and their variations. Mostly these were found to be not suitable for our purpose as they took huge amount of time for training and predicting (Eg: SVM and Neural Net) or because of high error in predictions they produced (Eg: Linear class of regressions).

Comparison of machine Learning Methods

Figure 5 provides a comparison between the average L2 error of different ML methods we tried over all 10 test cases. We observe that, Passive-Aggressive Regression is worst in performance, worse than baseline performance. Performance of Tree based regression algorithms is better than baseline but not the best. KNN regression and Gradient Boosting regression provide the best results with Gradient Boosting methods being slightly better.

Figure 6 provides running time performance of KNN Regression method with different K values over all 10 test cases. The grading specifications require all 10 test cases to execute with 1 minute when run over VM. K=13 seems to be the best choice in terms of running time, but K=95 was found to be the best in terms of accuracy.

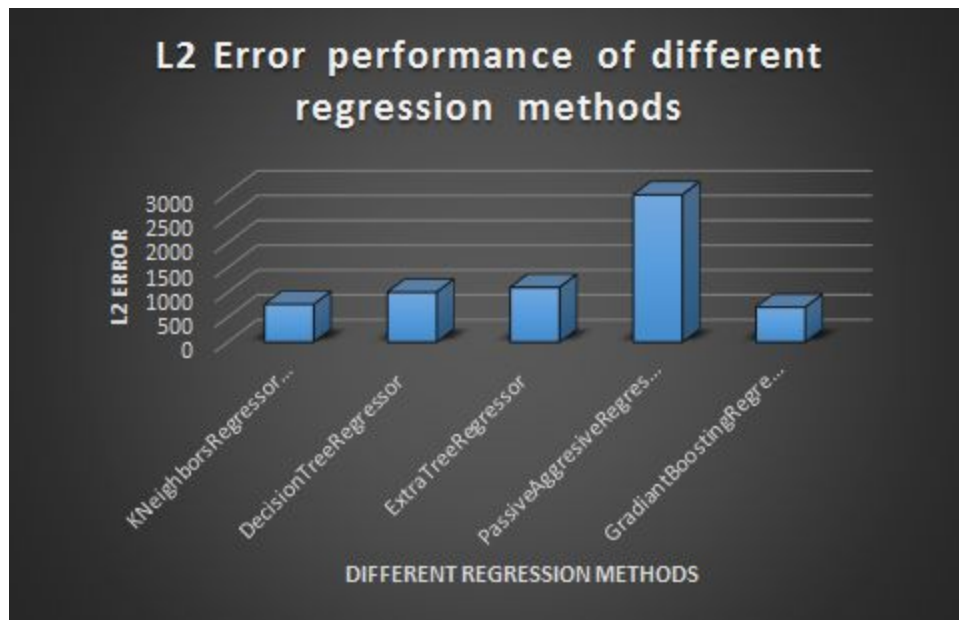


Figure 5: L2 Error performance of various Machine Learning Regression methods experimented with

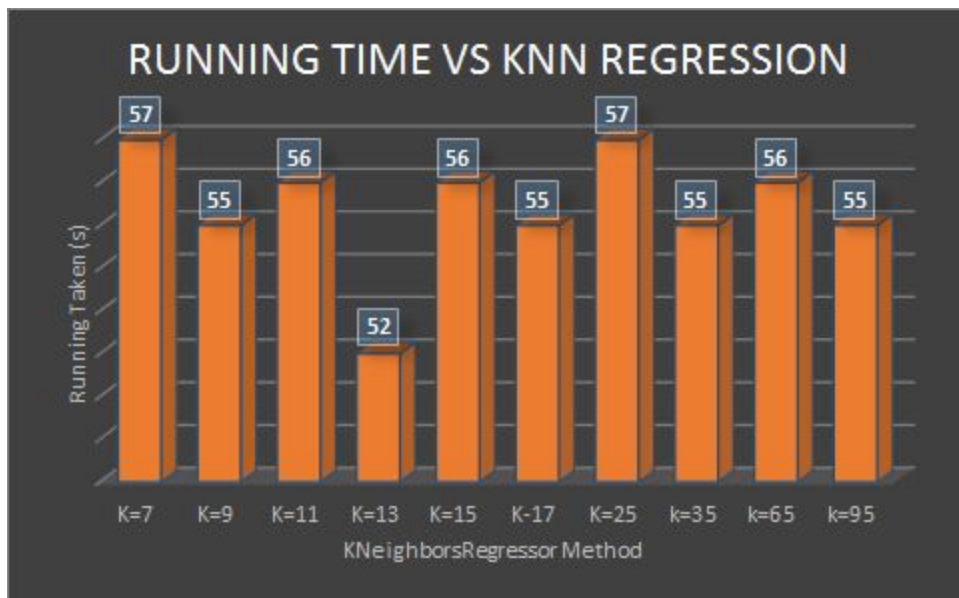


Figure 6: Running Time performance for all 10 test cases in seconds Vs different K values for K-NN Regression Method

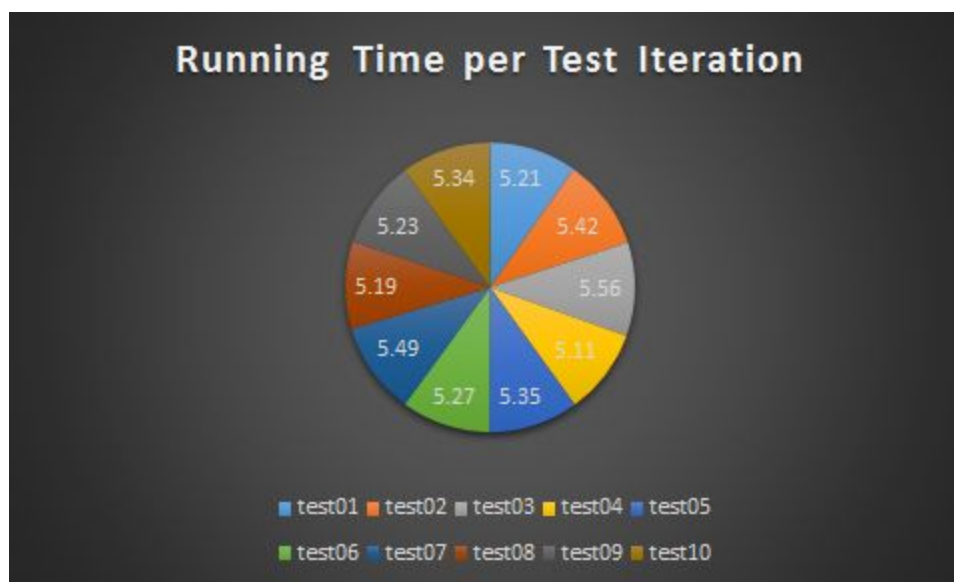


Figure 7: Running Time in seconds per test case iteration

Figure 7 provides details about running time per test case iteration taken by a KNN regression method with $K = 95$ over VM. All test cases seem to be taking less than 6 seconds to execute.

Why Machine Learning Techniques Worked?

Given the data points, it is evident that there's a pattern which can be identified using centroid locations. It is also clear that the next position of hexbug can be calculated if we know its current velocity, its heading direction and its current position. Leaving aside the randomness, if we input these 3 parameters as "features" to our feature vector and try to regress, we should get good approximation of next position. The point to be noted is that when the hexbug collides with the walls or candle, there's a lot of randomness involved as to where the bug will go next. And due to this randomness, our ML models won't be able to predict which path it will take next. If the error is to be seen from the test files, you would note that our models are good at predicting the fixed trajectory of hexbug's movement and bad at predicting the movement after collision.

Further Improvements

We have used only the centroid data extracted and given to us. In contrast, a video processing approach would have enabled us apply object tracking algorithms clubbed with our ML algorithms. In such an implementation we would have extracted a filter which identifies the hexbug in frames and use particle filter to track the hexbug. It could

potentially be used as extra input features to our prediction model. This would have improved the ML models' performance and hence the predictions.

Other possible improvement involves addition of deterministic variables along with ML output. I.e., if we could explicitly set the coordinates of walls and candle making it as "no-predict" region, our algorithm could make a random choice for the next trajectory after collision. This way our algorithm would be able to predict after collision randomness as well with some confidence.

Thirdly, one could use combination of various methods, and then use machine learning ensemble techniques to combine the prediction of various smaller classifiers. We did use Gradient Boosting regression which is a form of ensemble method, but a combination of particle filters with ML techniques is something which we could not implement in the required timeframe.

Conclusion

Statistical methods such as found in sklearn library e.g.: K-nearest neighbor and Gradient Boosting performed really well to predict object motion trajectories. Both these methods were able to learn the motion model from training data and able to predict motion of hexbug for the next 60 frames with good accuracy. While accuracy of Gradient Boosting regression method was found to be better than that of KNN based regression method, running time performance was found to be better for the latter.

Since KNN regressions' better running performance is attributed to its lazy learner nature while Gradient boosting regressions methods better accuracy is attributed to its weak ensemble learning method which increases its accuracy with every iteration.

In case of KNN, as the number of neighbors grow, we let more number of unique trajectories represent themselves instead of forcibly grouping them together with distantly identical trajectories. The experimental results confirm that models with higher value of k perform better in predicting trajectories.

References

1. Dizan Alejandro Vasquez Govea, Thierry Fraichard. Motion Prediction for Moving Objects: a Statistical Approach. Proc. of the IEEE Int. Conf. on Robotics and Automation, Apr 2004, New Orleans, LA (US), France. pp.3931–3936, 2004. <https://hal.inria.fr/inria-00182066/PDF/893.pdf>
2. <http://people.csail.mit.edu/dannyf/traj.pdf>
3. <http://www.cs.cmu.edu/~pearl/bennewitz02icra.pdf>
4. T. M. Mitchell. Machine Learning. McGraw-Hill, 1997



5. <http://scikit-learn.org/stable/modules/classes.html#module-sklearn.neighbors>