Analysis of Regression Tree Models on IMU Data for Positional Tracking of Cyborg Insects

Roger Wang, University of California Los Angeles, the United States
LiveMechX Morishima Lab, Prof. MORISHIMA Keisuke
Graduate School of Engineering, Osaka University

Abstract

Positional tracking is an integral feature of Urban Search and Rescue (USAR) cyborg robots. Efficient and low-cost inertial measurement systems, such as IMU sensors (magnetic, angular rate, and gravity), are frequently implemented onto cyborg robotic platforms to provide essential movement and orientation data. These sensors provide information in 9DoF: 3 axes each for acceleration, magnetic field, and gyroscope rate. However, due to inherent measurement noise, bias, and drift, it is not possible to directly integrate sensor data to provide meaningful estimations of position or orientation. Existing sensor fusion and filtration algorithms are already implemented on board some commercial chips, such as the Adafruit BNO055, to provide highly accurate measurements of absolute orientation but not position. One proposed solution for localization is the usage of a machine learning model. This paper aims to analyze the effectiveness of regression trees in directly predicting positional changes of a cyborg cockroach during short periods of time using aggregated data from an onboard IMU sensor. A microcontroller 'backpack' is mated to a cockroach via an existing surgical attachment. This backpack contains the IMU sensor, wireless transceiver, and Arduino microcontroller. The cockroach is placed in a square arena with an overhead camera which records video to use as ground truth of cockroach position. During experimentation, IMU data: 9DoF and euler angle orientations, are transmitted to another Arduino board which acts as a receiver. Sensor data is merged with video data to create a labeled dataset which is then grouped into time intervals of 1s long with 0.25s overlap. Augmented data from each time interval is collected as features into the final labeled dataset for use in model selection and training. Features include mean, standard deviation, and kurtosis for each of the 9DoF measurements as well as the euler angles. The top 10 predictors are chosen for the model by correlation score with the target variables, which, for each time interval, are the change in x,y position from the first datapoint and the last datapoint, representing the total change of position during that time period. Several models are considered and tested, including Decision Trees, Random Forests, and Gradient Boosted Trees. Each is tuned via Grid Search and a final fit using the best parameters is done. The final model is then run through a separate 10-Fold Cross Validation and its scores are recorded. Then, comparison of trees and various preprocessing scaling methods are completed. The best model was a Random Forests Tree for x-position with score of 0.750 and a similar Random Forests Tree for y-position with score of 0.729. These results show that regression trees can be effective for localization, but further work needs to be done to refine such models.

Introduction

Natural disasters are unpredictable and potentially catastrophic, as events such as earthquakes and hurricanes can quickly devastate urban areas. Destruction of buildings in particular can cause many individuals to become trapped under rubble and other debris. The survival of these individuals depends on the speed that they can be located and extracted. Such relevant missions are termed Urban Search and Rescue (USAR), and specialized USAR teams and technologies can save human lives during disasters [3].

Many small, robotic systems have been proposed for these missions with advantages such as size, maneuverability, and search speed. Automated robots can quickly search dangerous environments that human rescuers would not be able to traverse. However, robotic systems are incredibly complex, and require precise machinery in order to achieve reasonable mobility in unknown areas with various obstacles. Therefore, researchers have proposed using cyborg platforms as a solution to difficulties in locomotive engineering. Specifically, the use of cockroaches as mobile platforms for USAR applications has been demonstrated to be efficient and competitive with fully robotic machines [1]. Cockroaches provide several drastic advantages to robotic systems: they have incredible mobility, obstacle navigation, and self-righting abilities, all available without the complex engineering in fully artificial systems [1].

These cyborg systems demand several requirements in order to be feasible for real-world USAR missions. These include human detection, automated navigation, wireless transmission, etc.. Another highly important feature that cyborg robots must implement is self-localization, the ability for the system to relay its own position. Without the cyborg's location, rescuers can't track searched and unsearched areas, and won't be able to find a survivor that the system has detected. Consequently, positional tracking of the cyborg cockroach is absolutely essential for a successful mission.

In order to achieve accurate positional data, the cockroach is equipped with a 9DoF IMU sensor. These sensors are called an Inertial Measurement Unit (IMU) that collects magnetic in addition to standard acceleration and gyroscope data. Theoretically, positional and orientational information can be obtained from these measurements by directly integrating the measured acceleration and gyroscope rate respectively. Unfortunately, the heavy noise and large bias inherent within IMU sensors prevents useful integration from occurring [3]. Over time, noisy data and errors accumulate during integration and the resulting information is nonsensical. In the case of orientation, this is the infamous 'gyro drift' problem.

Several algorithms have been thoroughly researched to correct noisy sensor measurements, especially for the purpose of orientation [5]. Kalman and Madgwick filters as well as other sensor fusion methods have been successful in generating accurate estimations of absolute orientation. These functions are directly integrated into several commercial IMU sensors, such as the Adafruit BNO055 Absolute Orientation Sensor, which is used in this study. However, similarly performing algorithms are not yet widely used for the purpose of positional estimation.

One proposed solution to positional tracking using IMU sensors is machine learning. Several regression and classification models have been developed using traditional ML methods as well as deep learning neural networks, with varying performances [1][2][3]. These models have shown the capability of machine learning to predict motion states in cyborg insects as well as speeds and headings [1][3][4]. In particular, this paper aims to

study the feasibility of Regression Trees as a data-driven ML model to predict direct positional changes using IMU data.

The general function of Regression Trees is to split the dataset into branches where each node in the tree represents a split along a set of predictors. At the bottom of the tree, leaf nodes exist that are used for prediction. A new data vector falls down through the tree, entering nodes based on the splitting criteria of previous nodes until a leaf node is reached. Then, the values of the response variable in the leaf node is used to generate the model prediction. Three types of Regression Trees were considered in this study, Decision Trees, Random Forests, and Gradient Boosting Trees.

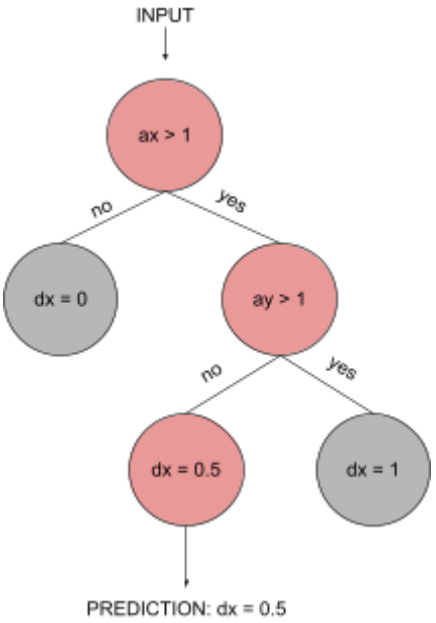

Figure 1: Example of very simple decision tree that predicts dx based on ax and ay (accelerations)

Decision Trees are the simplest of the three, using one tree with simple decision rules at each node to predict a final response. These trees are also easy to interpret, since the decision rules define the characteristics of the response and predictor relationships.

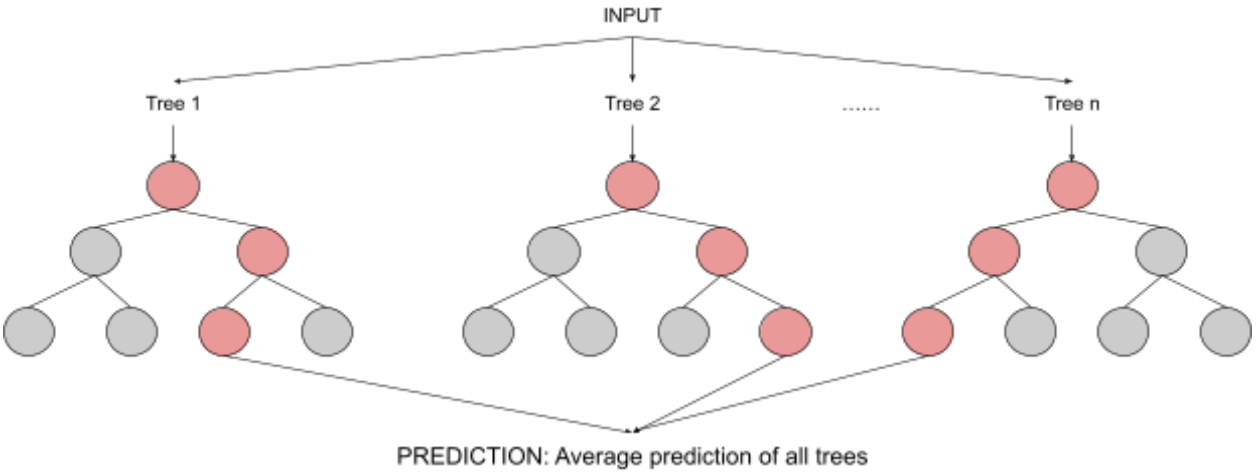

Figure 2: Example of Random Forest, prediction is based on average of trees

In contrast, Random Forest models use a random set of decision trees in order to predict a response. The final prediction is usually a weighted average of the predictions of all trees within the model's forest. Furthermore, Random Forests use bagging, a sampling technique on training data, as well as random subsets of features at each split, in order to create the most accurate predictions.

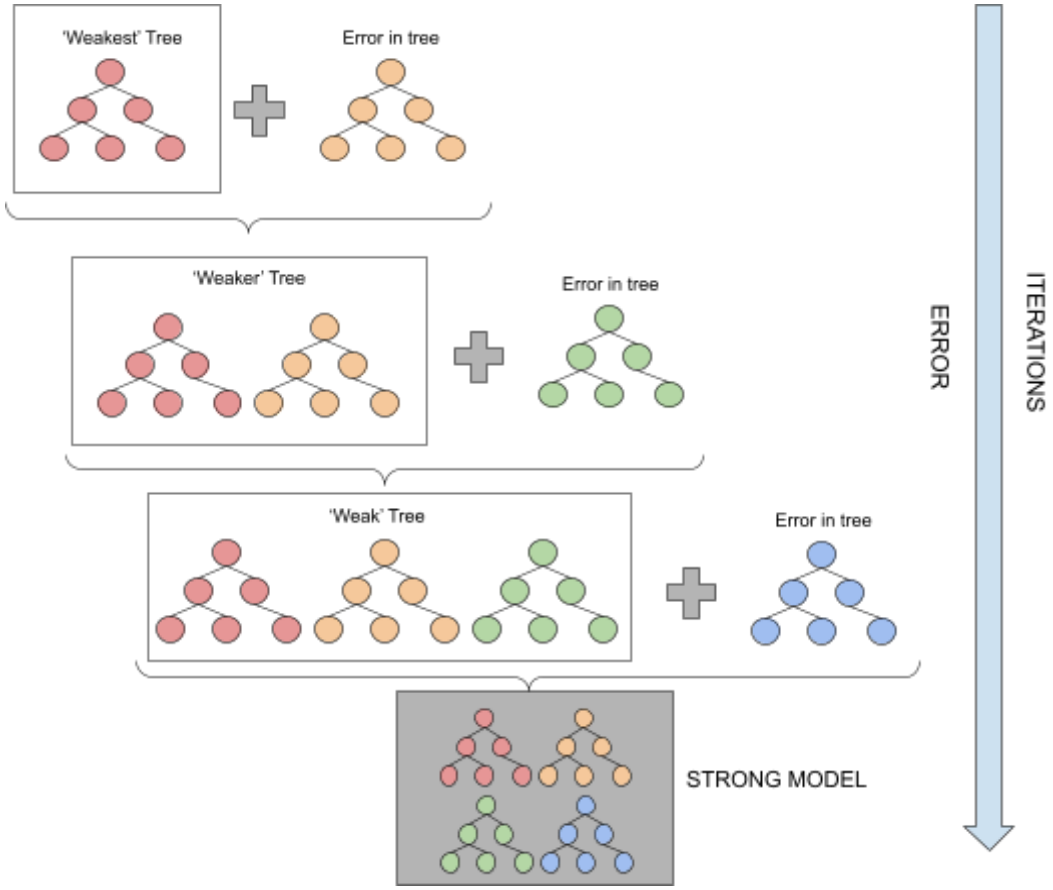

Figure 3: Diagram of Gradient Boosting trees, notice error goes down with more iterations

Finally, Gradient Boosting Trees use a combination of 'weak' decision trees to progressively predict the error of the previous tree, these 'weak' trees are aggregated into one 'strong' model.

These models were trained on IMU + video data to directly predict the change in x,y position during a 1s time interval. The hope is that a direct sum of x,y changes can be used to determine the total dx,dy of the cyborg insect during its operation time and localization is then achieved. The following sections of this paper detail the experimental method as well as model results including R^2 scores and parameters.

Materials and Methods

Cyborg Insect and Data Transmission/Receiving

Madagascar hissing cockroaches were selected as the cyborg platform for their desirable qualities and behaviors as detailed by Ariyanto et al. [1]. The cockroaches were raised through a standard procedure and then had surgical implantations for the attachment of microcontrollers [1]. The implantations allowed for electrical stimulation of cockroaches' antennae and cerci, which is intended to control the insects movement. These connections are not utilized in this study. Rather, only the microboard attachment is used as a mounting

platform for the microchip ‘backpack’, which includes the Adafruit Seeeduino Xiao board, NRF24L01 wireless chip, and Adafruit BNO055 Absolute Orientation Sensor.

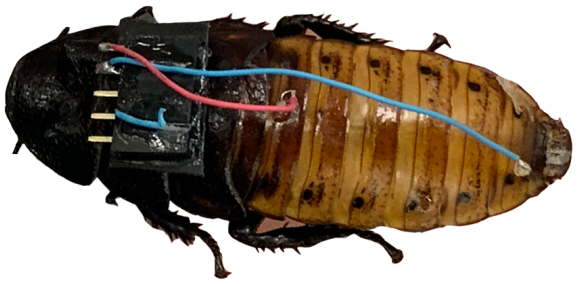

Figure 4: Cockroach with surgical implants, the black unit is used as ‘backpack’ mount

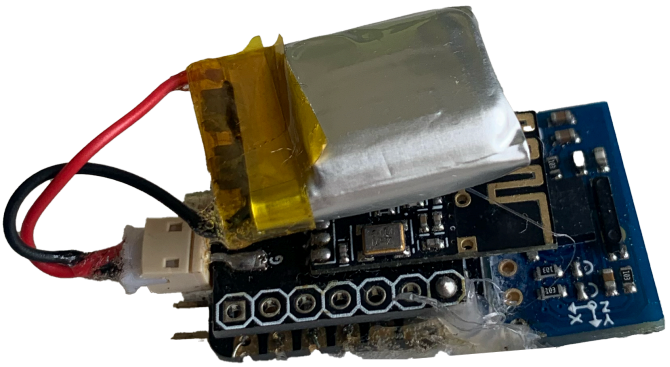

Figure 5: Microchip backpack assembly

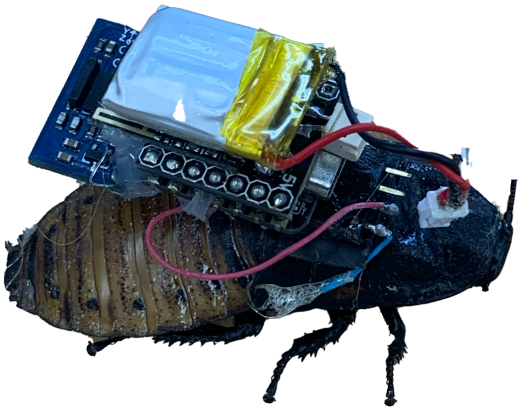

Figure 6: Cyborg Cockroach with ‘backpack’ mounted

A simple data-collection and transmission program written in the Arduino C++ IDE was uploaded to the Seeeduino board. This program connects the ‘backpack’ transmitter to a separate receiver board. This receiver, an Arduino MKR WiFi 1010, is connected to a local laptop in order to collect and store the data. A separate receiver program is uploaded to the MKR board.

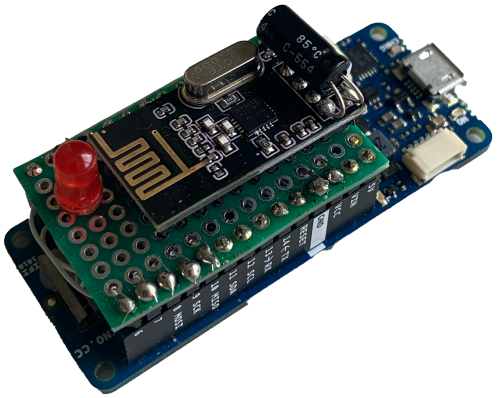

Figure 7: Receiver module

The Transmitter program initializes the RF24 radio object (wireless chip connected to pins 6 and 7 of Seeeduino board) and connects to the channel at number 108. The data sampling rate is set to 10ms or 100Hz. At every interval of the data sampling rate, a max 32-bit package is sent via channel 108 to the receiver board. This package contains the IMU sensor data. The data includes: 3 axis of accelerometer, 3 axis of magnetometer, 3 axis of gyroscope, and 3 principle Euler angles, for a total of 12 points of data. Consequently, due to the max 32-bit size of the transmission package, the data points cannot be sent as precise floats or doubles. Instead, each data point is converted into a signed short int, which reduces the data package to 24-bits, well within the max size. In addition, direct conversion to short int data type involves the truncation of decimal values, a valuable information source when working with slow moving and accelerating cockroaches. Thus, accelerometer, magnetometer, and gyroscope data points are multiplied by 1000 and rounded in order to preserve decimal information before being typecast to short int. This procedure is ignored for Euler angles since they are described in degrees and decimal precision is not required.

The Receiver program has the same initialization steps for the wireless chip. Every time a data package is available from the transmitting board, which comes at intervals set by the data sampling rate, it is read and displayed onto the serial monitor at baud rate of 115200. Before display on the monitor, the reverse conversion formula is applied to the accelerometer, magnetometer, and gyroscope data points. After the experiment, data from the serial monitor is collected and saved into a .csv format for use in data augmentation and model training in Python.

Data Collection

The data collection setup involves placing the cyborg robot into a square arena, 0.88mx0.88m, with a video camera mounted overhead. The camera begins recording and data collection from the cyborg transmitter/receiver module also begins, simultaneously. Data is recorded in ranges of 45s to 90s. During this time period, the cyborg insect is allowed to move freely and without stimulation. At the end of the video recording, data collection is also stopped. This way, the video and sensor data describe exactly the same time period which will be useful for data processing.

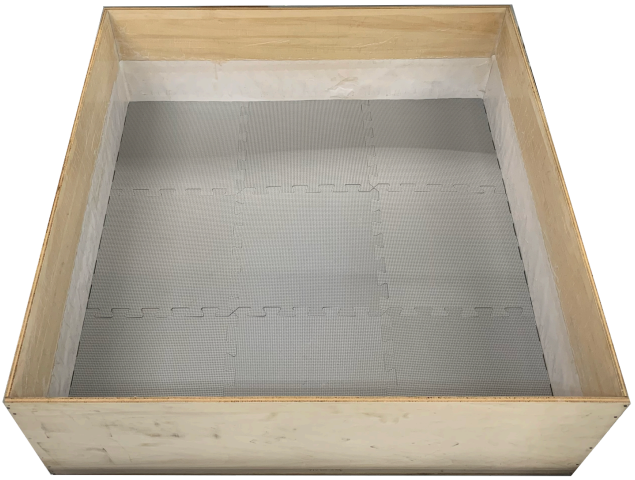

Figure 8: Square arena for data collection experiments, 0.88m x 0.88m

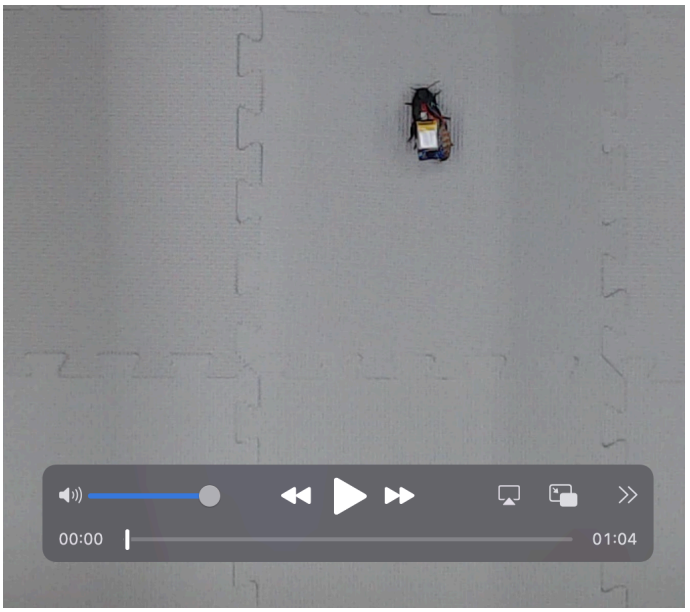

Figure 9: Screenshot of recorded data collection video

This process creates a sensor dataset and corresponding video. The video is uploaded to Tracker, a video analysis tool, and the corresponding ground truth positions are extracted from the video. Importantly, the x and y axis for these experiments are determined by the initial cockroach orientation at the start of the experiment. The x-axis is aligned with the initial insect’s heading, and the y-axis is orthogonal to this. Predicted dx and dy values are now in the frame of the cockroach's initial heading. This was done in order to provide interpretability to the model prediction, as in real-world applications, it may be difficult to decide a standard frame for all cyborg robots to reference during a USAR mission. By aligning the xy axis to each cyborg's initial orientation, predicted dx and dy values can be understood in relation to this heading, which would be known during application.

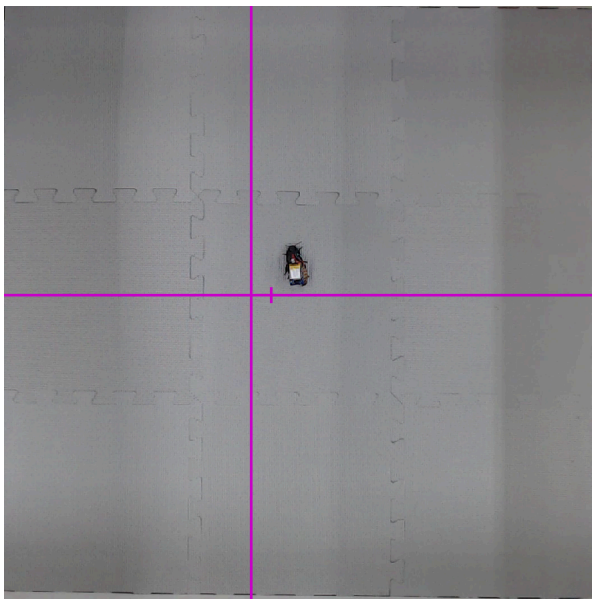

Figure 10: Initial Axes in Tracker

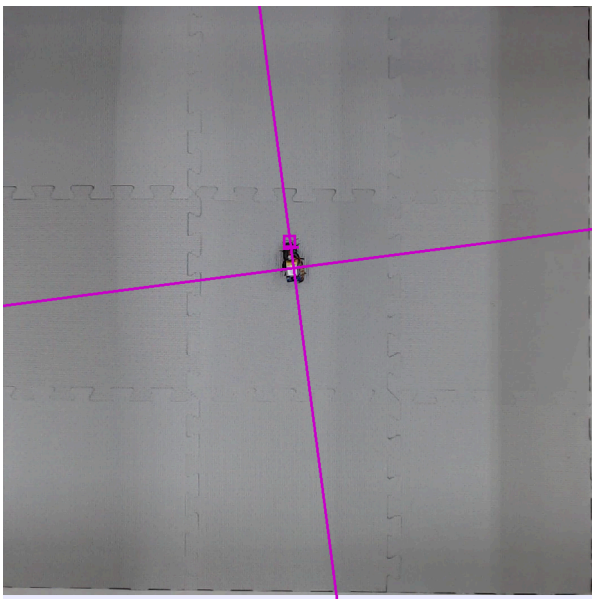

Figure 11: Rotated axes to align with initial position and heading

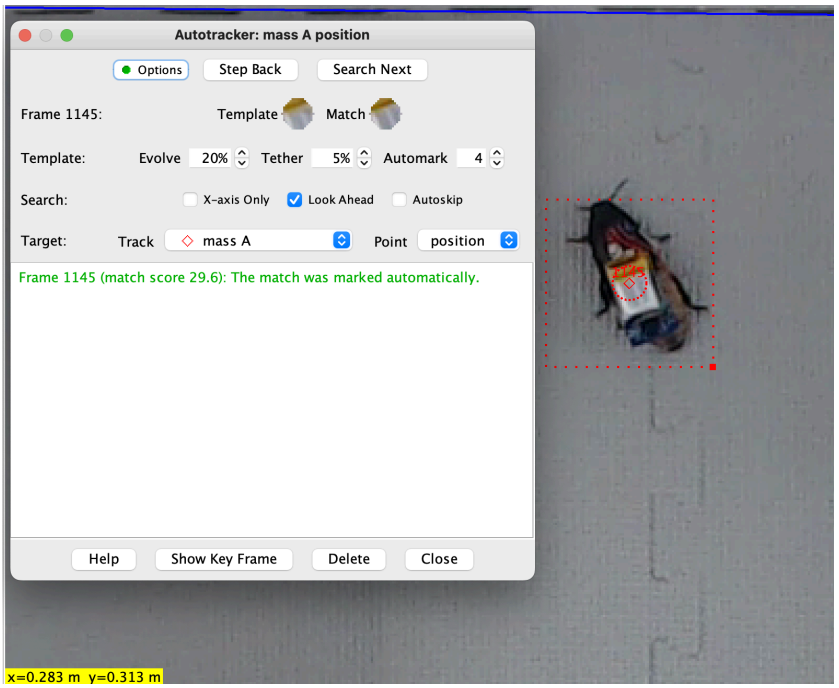

Figure 12: Tracker software automatically follows cyborg insect and records position

The Tracker software produces a positional dataset for each frame of the uploaded video. However, the frame rate of the video capture and the data sampling rate of the IMU sensor do not align. The video records at 30fps, so ground truth data is available at a frequency of 30Hz. Furthermore, after examining the IMU sensor datasets in Python, it is discovered that the true data collection rate is slower than the intended data sampling rate, at around 16ms vs 10ms, or 62.5Hz vs 100Hz. This discrepancy could result from the time of wirelessly transmitting and receiving packages between modules. Consequently, the difference in data rates between video and sensor data means that accurate ground truth data is only available for about half of the sensor observations (every other measurement).

In practice, this will not affect the accuracy of ground truth data and sensor data alignment since changes from one observation to the next are still very small.

Once video data has been collected, it is paired and merged to corresponding sensor data. This process removes about half of the sensor data that is left without ground truth position and results in a single dataset that contains sensor data and position. This is the 'Initial Dataset'. This data collection procedure was completed 9 times, for 9 separate sets of data. Each will be referred to as Experiment 1, 2, … etc.

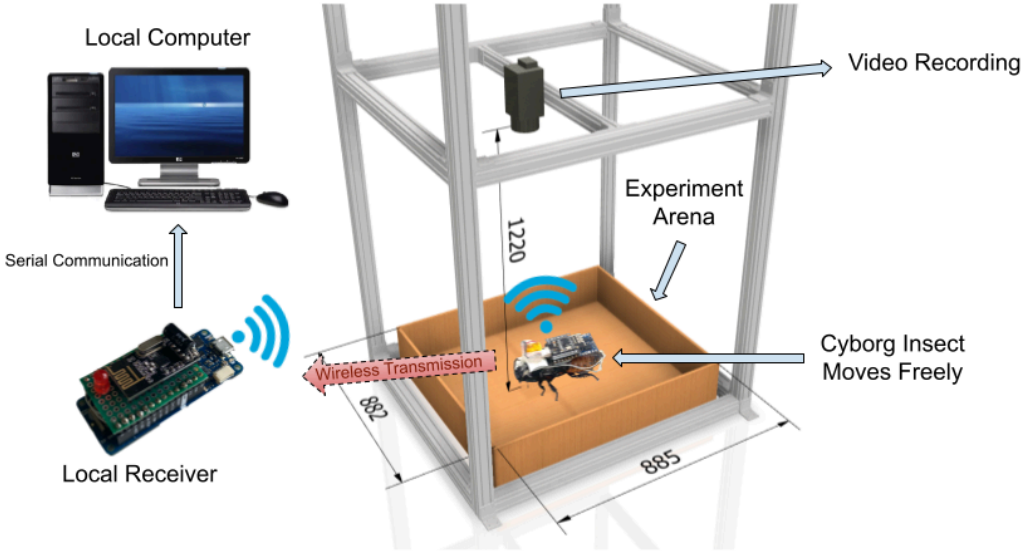

Figure 13: Diagram of data collection procedure

Data Augmentation

Before the creation of the 'Initial Dataset', i.e. before video and sensor data are merged, a low pass filter is run on the raw sensor data for the accelerometer, magnetometer, and gyroscope measurements. The chosen filter is 2nd order, with critical frequency 5Hz and digital sampling frequency 100Hz. This filter was done in order to help remove some of the noise in sensor measurements before further data augmentation.

After the 'Initial Dataset' for an experiment is created, several steps are taken in order to divide the dataset into sets of short time intervals. The chosen interval is 1s long with a 0.25s overlap between consecutive intervals. This allows for the analysis of a small segment of the cyborg robot's movement. For each time interval, several statistics are collected and become the predictor variables for this time interval. These statistics are: mean, standard deviation, and kurtosis for each of the 12 variables. Other statistics such as variance, root mean square, median, and zero/mean crossing rates were also considered, but not included in the final data augmentation process due to poor predictive ability. Finally, the response variable, or label, of each time interval is computed as the change in x,y position between the first and last observations during the time interval, representing the total dx and dy of this period.

This process is repeated for each of the 9 data collection experiments, and all but one of these datasets are concatenated together to form one large dataset, the 'Final Dataset', which model training and selection will be performed on. The data corresponding to Experiment 5 is excluded from the 'Final Dataset' for reasons that will be discussed later in this paper.

Data Preparation and Initial Preprocessing

The 'Final Dataset' has 36 predictors, 3 each for the 12 data points from the sensor, which is likely to be too much for a model to be trained on. Thus, only the top 10 predictors are chosen by comparison of correlation with target variables dx and dy.

The training dataset is then run through an initial Standard Scaler before use in hyperparameter tuning and k-Fold cross validation.

Model Tuning and Cross Validation

Hyperparameter tuning is performed for each of the 3 chosen models with the use of GridSearchCV, which exhaustively tests models using a combination of predefined ranges of parameters. The GridSearchCV is initialized with simple cross validation of 10 splits. In addition, the cross validation used with this function is done without shuffling, so another cross validation is done with the best fit model after tuning.

This second cross validation is a 10-Fold validation with shuffling and random state of 42 for reproducibility across calls.

Parameters tuned for are as follows:

Table 1: Regression Trees and parameters searched during GridSearchCV tuning

Parameter	Decision Tree	Random Forest	Gradient Boosting
max_features	[None, 1, 5, 'sqrt', 'log2']	[None, 1, 5, 'sqrt', 'log2']	[None, 1, 5, 'sqrt', 'log2']
min_samples_split	[2, 5, 7, 10, 15]	[2, 5, 7, 10, 15]	[2, 5, 7, 10, 15]
min_samples_leaf	[1, 5, 7, 10, 15, 20]	[1, 5, 7, 10, 15, 20]	[1, 5, 7, 10, 15, 20]
n_estimators	n/a	[5, 10, 20, 30, 50, 100]	[5, 10, 20, 30, 50, 100]

The scores of the 10-Fold validation, which are R^2, are collected and used as an evaluation of the model at best parameters.

Preprocessing Testing

After best parameter models are found, further analysis is done in order to study the effect of different preprocessing scalers on model performance. Namely, the scalers tested were MinMax, Normalizer, and Robust scaler. Each scaler has different properties and scales the data in different ranges with different regards to outliers.

Each scaler is run on training data and another 10-Fold validation is done with the new scaled data and best parameter model. The scores of these validations are collected and used as evaluation of models with different preprocessing implementations.

Results

The best predictors for both dx and dy include statistical values of acceleration, magnetic field, gyro rate, and yaw. Surprisingly, the average of y-axis acceleration was a top predictor in both dx and dy models whereas the average of x-axis acceleration was weak in both. Also, measures of yaw were important for both predictors which was expected since this represents the direction of motion for the cyborg insect.

Table 2: Top 10 predictors for ‘dx’ and ‘dy’ model by correlation

‘dx’ Model Predictors	‘dy’ Model Predictors
avg_az	avg_ay
avg_ay	std_mx
std_mx	std_yaw
avg_gy	avg_az
std_gx	kurt_mx
std_my	avg_yaw
std_yaw	kurt_az
std_gz	kurt_ay
avg_yaw	kurt_ax
std_mz	avg_pitch

For **models predicting dx**, the Random Forest Regressor performed the best, with a score of 0.750 using Standard Scaler. The best parameters for this model and others tested are as follows:

Table 3: Three Regression Trees for ‘dx’, best parameters and final score

Parameter	Decision Tree	Random Forest	Gradient Boosting
max_features	None	5	None
min_samples_split	2	10	15
min_samples_leaf	15	1	7
n_estimators	n/a	100	10
Scores:	**0.688**	**0.750**	**0.586**

In terms of preprocessing testing, Standard Scaler, MinMax, and Robust all perform with the same accuracy. However, Normalizer scores considerably worse in all three models.

Table 4: Three Regression Trees on ‘dx’ with different scalers applied and their scores on best model

Scaler	Decision Tree	Random Forest	Gradient Boosting
Standard	0.688	0.750	0.586
MinMax	0.688	0.750	0.586
Normalizer	0.539	0.696	0.496
Robust	0.688	0.750	0.586

For **models predicting dy**, the Random Forest Regressor performed the best as well, with a score of 0.729 using Standard Scaler. The best parameters for this model and others tested are shown below.

Table 5: Three Regression Trees for ‘dy’, best parameters and final score

Parameter	Decision Tree	Random Forest	Gradient Boosting
max_features	None	5	None
min_samples_split	2	10	2
min_samples_leaf	10	1	5
n_estimators	n/a	100	20
Scores:	**0.618**	**0.729**	**0.628**

Similar preprocessing relationships as dx are also shown for dy models, namely the poor performance of Normalizer and identical performance of other scalers.

Table 6: Three Regression Trees on ‘dy’ with different scalers applied and their scores on best model

Scaler	Decision Tree	Random Forest	Gradient Boosting
Standard	0.618	0.729	0.628
MinMax	0.618	0.729	0.628
Normalizer	0.470	0.633	0.547
Robust	0.618	0.729	0.628

Discussion

The best performing model for both dx and dy performed similarly with scores from 0.73 ~ 0.75. Although these scores seem satisfactory it is worth noting that they represent cross validation scores only, as model testing on entirely new datasets was not thoroughly conducted. In fact, preliminary generalization testing of these models on new experiments

showed very poor results. Thus, there is a likely possibility of overfitting to the training data already seen. Although cross validation with shuffling ensures that a random set of indexes are used for each train/test split and that new models are fit for each split, overfitting may still occur as patterns in an experiment may be learned during training which allows the model to earn good test scores on testing data from the same experiment. For example, if a section of Experiment 3 is randomly included in the training set, it may have enough information for the model to learn patterns specific to Experiment 3, causing a high score on the rest of the dataset used during testing.

The poor generalization performance of these models could also be due to the relatively small datasets used during training. The total length of the training dataset is only 792, which may not be enough to quantify all the different movements that cyborg insects are capable of. Therefore, new datasets with new cyborg motions, faster than usual, more stopping time, etc., may not be represented well in the training data, which leads to lack of generalization.

Increasing the size of the dataset may help the model generalize. Despite this reasoning, Experiment 5 was omitted from the final training dataset as described earlier. Although more data should be better, in models trained using Experiment 5, the testing scores decrease drastically, to around 0.18 - 0.2. Accordingly, this experiment was not used in the final procedure.

Conclusion

This study has shown the potential effectiveness of Random Forest Regression Trees in predicting the location of cyborg robotic systems using IMU sensor data. Further research is necessary to refine these models and expand the training datasets so that generalized predictions can be made on new data. In addition, this paper focused on the use of ML models that are not implemented on board the cyborg system. In the current configuration, cyborg data must be transmitted to an off-site computer for processing and location prediction, a function that can ideally be performed on-board the cyborg chip. Continued work in converting the data augmentation and Random Forest model into C++ will allow the positional data of the cyborg insect to be transmitted directly to operators. Finally, this model builds predictors for the xy-axes only, as the experimental arena used was flat and allowed the cyborg insect to move only within a 2D plane. Real-world applications are bound to be three dimensional, with vertical movements over obstacles as well as elevation changes expected. As such, the efficiency of these models on the z-axis is yet to be determined.

References

1. Ariyanto M., Refat C.M.M., Hirao, K., Morishima, K. *Movement Optimization for a Cyborg Cockroach in a Bounded Space Incorporating Machine Learning*, Cyborg Bionic Syst., 15 March 2023
2. Chen, C., Pan, X. *Deep Learning for Inertial Positioning: A Survey*, 20 March 2023
3. Cole, J.y, Bozkurt, A., Lobaton, E. *Localization of Biobotic Insects Using Low-Cost Inertia Measurement Units*, Sensors, Department of Electrical and Computer Engineering, NC State University, Raleigh, NC 27695, USA, 11 August 2020
4. Tran-Ngoc, P.T., Le, D.L., Chong, B.S., Nguyen, H.D., Dung, V.T., Cao, F., Li, Y., Kai, K., Gan, J.H., Vo-Doan, T.T., Nguyen, T.L. and Sato, H., *Intelligent Insect–Computer Hybrid Robot: Installing Innate Obstacle Negotiation and Onboard Human Detection onto Cyborg Insect*. Adv. Intell. Syst., 5: 2200319. https://doi.org/10.1002/aisy.202200319, 27 January 2023
5. van Dijk, M.P., Kok, M., Berger, M.A.M., Hoozemans, M.J.M., Veeger, D.H.E.J. *Machine Learning to Improve Orientation Estimation in Sports Situations Challenging for Inertial Sensor Use*. Front. Sports Act. Living 3:670263. doi: 10.3389/fspor.2021.670263, 3 August 2021