README Documentation

How to Execute the Program

This program is designed to be run from the command prompt. The input of our Python program is training data consisting of a hexbug moving around in a box. This is gathered by reading a list of data points representing the coordinates of the centroid of the hexbug robot in each frame of the video file. Our program returns a prediction of the coordinates of the centroid of the robot's blue region for 60 frames in the form of 60 integer pairs, each separated by a newline, and elements of a pair separated by a comma. The first element of each pair is the x-coordinate of the centroid, given by the number of pixels from the left side of the video window. The second element of each pair is the y-coordinate of the centroid, given by the number of pixels from the top of the video window.

1. Open the command prompt.
2. Navigate to the directory where the program files are stored
3. On the command prompt, enter:
   $ python predictor.py -f "filename" (if you omit the -f flag, it will default to reading in the file `testing_video-centroid_data`)

Optional Flags

```
-c, --clean            Do not use previously learned data
-d, --debug            Write debug output while running
-f, --file=FILE        Read coordinates from FILE
                       Defaults to testing_video-centroid_data
-h, --help             Print this help screen and exit
-l, --learn            Learn from data and create a learned file
-o, --output=FILE      Write output to FILE
                       Defaults to prediction.txt
-p, --predict=FRAMES   Predict FRAMES number of frames
                       Defaults to 60
-v, --visualize        Show a visual of the predictions after processing
                       Includes the last 120 points of the original data
                       (or all of it if less than 120 exists)
-y, --yaw=YAW_LIMIT    Limit yaw noise
                       Defaults to 0.015
                       Set to zero to turn off
```

The program will default to reading in pre-learned data created from the training file `training_video1-centroid_data`. The pre-learned data includes the location of the 4 walls and the final, learned, covariance matrix from the input data. This pre-learned

data is used to enhance the prediction of the system. To run the program without using pre-learned data, use the -c flag. To force the program to learn new data, use the -l flag.

To see a visual representation of the data, use the -v flag. The visualization uses the turtle package, and will plot the last 120 points of the data (or all available if less than 120) in red, and the predicted points in blue.

eg. enter:

```
$ python predictor.py -f "filename_data" -v
```

The predictions will be output to the file `prediction.txt` by default. This can be changed with the -o flag.

Dependencies:

Before you run the program, be sure to have the following installed:
Python version 2.7
NumPy version 1.9.1

Brief Overview of How Our Algorithm Works

Our prediction algorithm utilized the Extended Kalman Filter. A Kalman filter utilizes a series of measurement and noise data taken at regular time steps to create educated estimates of the system's state. With a Kalman filter, a prediction step first produces estimates of multiple state variables and uncertainty values for the system. Next, an update step takes incoming (noisy) measurement data and updates the system variables based on the previously estimated state variables as well as the newly measured data. This filter runs continuously as more data comes in. A standard Kalman filter can only be used with linear data, so to account for the nonlinearity of the hexbug's movement we used an extended Kalman filter (EKF).

The extended Kalman filter is very similar to the standard Kalman filter, and produces the same state variables and estimates as a standard Kalman filter for linear data. However, the difference is that the EKF can use nonlinear, differential equations for the measurement and state-transition models. At each time step, the EKF algorithm linearizes these equations using a Jacobian matrix, and uses the modified system of equations to make its prediction.

In order to model the potential collisions of the hexbug, we determined the state space from the training data based on a min-max of the known x and y coordinates in the training data, adjusting these values as needed for any new data points.

We also realized that since we were predicting the next step and only checking to see whether we hit a wall or not at that point, our prediction for a wall collision could move beyond the wall before it bounced back. Therefore we modified our algorithm to simply remove those erroneous points. Since our prediction uses the centroid of the robot as input data, we bounce the robot off of the walls when it is within 10 pixels from a wall in order to account for the robot's size. This seems to be a good estimate based on the data.

Since there was so much noise in the input data, the filter can occasionally diverge and predict yaw rates that are too high, resulting in prediction that moves around in a tight circle. As a result, we decided to limit the yaw rate to within a reasonable range. Restricting the yaw allows for nonlinear motion predictions, but keeps the predictions within a reasonable range in the case that the filter diverges. After many test runs with different yaw rates, we decided to limited the yaw rate to a default -0.015 to 0.015 (rad/s) in angular velocity.

Justification for the Extended Kalman Filter

The Kalman filter is a very good method for localization and prediction for position and velocity with noise in 2 dimensional x-y space. We originally attempted to use a standard Kalman Filter and the resulting output images always erroneously moved in a straight line. Since our system is nonlinear, we realized that a standard Kalman Filter, which requires a linear system, would not be appropriate. Therefore, we needed to apply the Extended Kalman Filter (EKF) instead, to account for this nonlinearity. The EKF uses a Jacobian matrix created from the measurement and noise data to linearize about an estimate of the current mean and covariance. Our EKF localizes and predicts heading, as well as yaw rate and velocity for each time step.
We considered other methods, such as a particle filter, for this project, but ultimately decided on an extended Kalman filter due to its accuracy and effective handling of nonlinear data. No filter would be able to produce a perfect prediction for the hexbug's noisy motion, but the EKF does a good job of extracting useful information from past measurements and reducing it to reasonable predictions.

Test Results with Images

After running a number of tests on sample data and plotting the results, we achieved reasonably accurate looking predictions.

We ran several tests with different data in order to find the best solution. Using a yaw limit of 0.015 rad/s we were able to accomplish an L2 error as low as 497.47, which is demonstrated in the following figures.

Figure 1: The last 120 frames of a test file. The actual hexbug movement.

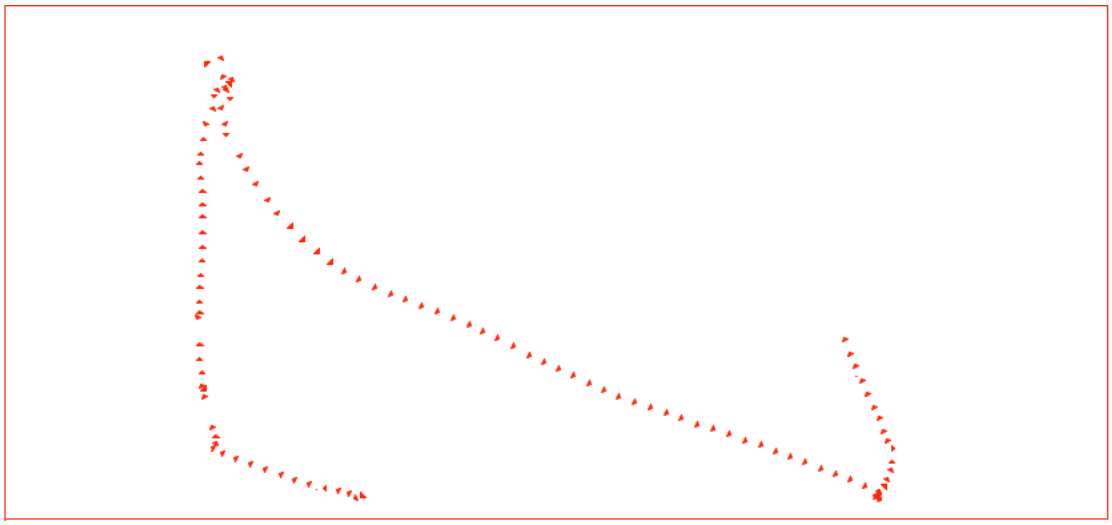

Figure 2: Prediction of 60 frames (in blue) using the test file with the last 60 frames removed.