CS 8803: AI For Robotics
Final Project - Spring 2016

Team Members:
Joe Conard (jconard8@gatech.edu)
Darin Itdhanuvekin (ditdhanuvekin3@gatech.edu)
Zach Singleton (zsingleton@gatech.edu)
Wilson Tsao (wtsao6@gatech.edu)

This report explains our understanding of the problem, steps we took to analyze and learn the robot's motion, the approaches we considered and ultimately went with, and how our approach performed with our test sets.

The Problem

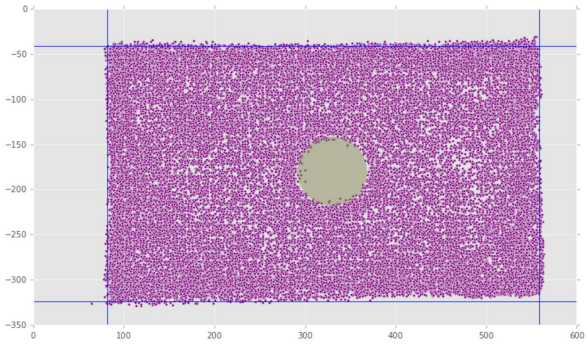
For the final project, we are provided with 10 partial video clips of a HEXBUG Nano micro robot moving in a wooden box, each 62 seconds long and recorded at 30 frames per second. We are given the first 60 seconds (1800 frames) of training data for each video. From that, we must predict the positions of the robot in each of the following 2 seconds. There is a candle inside the box that must be accounted for. There is also noisy data and information we must account for and/or handle throughout this project.

Analyzing the robot

We studied the robot's behavior by analyzing all of the data provided in the 10 test videos as well as all of the data provided in the training set. Based on the analysis we came up with 1) Constraints we used for coding, 2) overall stats we could use to inform how the robot behaves, and 3) helpful parameter guides for the robot.

1. **Constraints**

   We plotted every single point that the robot traveled across all of the test and training data sets in order to get a sense of the boundaries in which the robot could be in. The following is that plot:



We drew reference lines as well as an approximation of where the candle was in relation to all of the points the robot traveled.
Note: We can see from the above that there's noise in the measurement data, but we can confidently say that based on the above, we can use these constraints for our robot step predictions:

MIN_X = 82, MAX_X = 558, MIN_Y = 41, MAX_Y = 324
CANDLE_MID_COORD = (330, 180) ; CANDLE_RADIUS = 38

2. **Overall stats**

We had a supporting ipython notebook that allowed us to make many cuts and splits on robot data. One of the interesting splits was a summary of the robot stats, being:

"Open range" stats: (where open range means that it's more than 5 away from our defined walls and more than 5 away from our defined candle)

- Average distance traveled per step: 6.6
- Average turn: 0.000 (our takeaway from this was that it moved left just as much as it moved right)

Wall interaction stats:

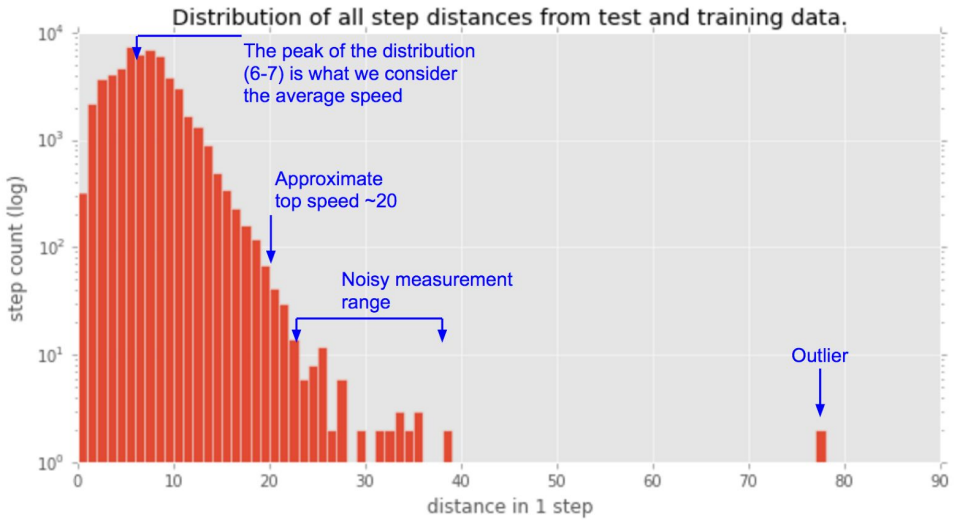
- Wall interaction percentage: 7% (i.e. the robot hit the wall 7% of the time)
- Average turn per move: 0.000 (our takeaway from this was that the robot did <u>not</u> favor moving left or right based on a wall collision)

Candle interaction stats:

- Candle interaction percentage: 1%
- Number of left turns: 94
- Number of right turns: 95 (Because the turn off a circle varies, we ended up looking at left vs. right turns which allowed us to conclude that the robot also does not favor moving left or right based on a candle collision)

3. **Parameter guides**

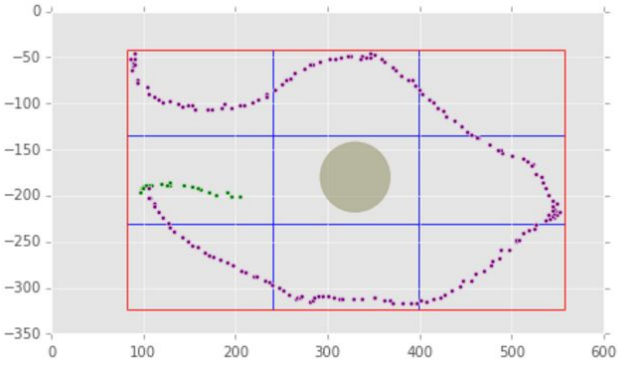
One major parameter we looked at was the distribution of distance traveled by the robot. This had a very wide range, but it's a clear unimodal distribution that allowed us to know the "average" step size of the robot and a reasonable approximation of the maximum step size of the robot.



Assumptions

Naturally following our analysis, we made assumptions about the robot that constantly informed our approach and adjustments made to our predictions. Our assumptions were made through 1) data-based analysis, 2) snapshots of many "interesting" states of the robot, and 3) observations of the robot through all of the videos. The main assumptions are:

- The robot is <u>not</u> sensing where it is, and therefore is not aware of where it is nor is it aware of its surroundings including the walls or candle.
- The robot, for the most part, attempts to travel in somewhat circular motions. We believe that if the space had no obstructions, the robot would most often be traveling in curves that could be mapped to arcs of a circle.



The above plot is an example "snapshot" that we used to analyze the robot's motion. Each snapshot is 200 steps of the robot (in the above example, for steps 401-600 of test10). The green dots are the first 20 steps of this set. The purple

dots are the rest of the steps. The red lines represent the walls. The candle is the circle in the middle. The blue lines are merely “gridlines” that we can refer to.

In this example, we see the circular (albeit inconsistent) motion that the robot follows.

- The robot has both 1) a max speed, and 2) more or less of a constant acceleration.
- On the topic of max speed, this is informed by the distance distribution above where we can comfortably say that the robot would likely not travel beyond 20 in one step.
- On the topic of acceleration, we noticed from watching videos that the robot seems to “lose steam” when it collides with the wall or the candle. We were not able to mathematically prove or arrive at an estimate of acceleration but we do believe that the robot has a constant acceleration that gives it a pretty consistent trajectory.

The Approach

The approach we went with was logical, simple and backed by our analysis. It consists of mainly 4 parts: 1) Starting with a strong Kalman Filter similar to what we all learned from the lecture videos, 2) Expanding on the Kalman with an Extended Kalman Filter in order to better simulate this robot’s motion, 3) Reducing noise in some of areas where it made sense, 4) Handling collision types that we discuss later in this paper in the ‘Collision Handling’ section.

1. **Starting with a strong Kalman Filter**

   We initially discussed as a team which approaches we thought could be used for this assignment. Because a few teammates had success using Kalman Filters (KF) with the Runaway Robot project and the uncertainty of how to apply the other concepts, we decided to use KF. The thinking here was that if time allowed, we could implement additional algorithms with this approach.

   Input to KF

   To use the KF, we needed an input data trend analysis. Using all the data points we quickly found was not a solution. Observation of video told us that a general motion trend could be picked up in a very short period of time. Consequently, for each input file, we used trend analysis of the last 10 steps of the robot, basically 1/3 of a second. This provided rough data for the angle and current heading of the robot. Longer bases actually introduced more

unpredictability as the box and obstacles created an environment that changed angles and headings often.

Prediction Start Point

Because the trend analysis caused an offset error in the first prediction point, special logic was added to determine the first prediction point by calculating the heading of the last two actual points from the input data. This had a small positive impact on our results for all KF and EKF tests.

Drawbacks

It was soon obvious we had to look at the Extended Kalman Filter. Because the KF doesn't account for noise in a non-linear environment, we found the accuracy to be insufficient.

2. Expanding on the Kalman with an Extended Kalman Filter

Due to the accuracy issues with the KF, we ended up using an Extended Kalman Filter (EKF). this allowed the application of process noise along with an associated covariance matrix. We were able to fine tune the matrix for what appeared to be our best results through testing. We also found that adding an average angle calculation in radians to the prediction of the next x in the EKF aided in determining headings and curvature. Finally, we set the next state matrix change in time (Δt) value as a speed control. We adjust this diagonal in the matrix during the run if the speed exceeds what we determined to be average speed maximum or falls below what we determined to be average speed minimum. These values were calculated based on the training data provided. The filter "tuning" values were set as follows:

1. **Covariance Matrix Q** was defined as shown below. These values were arrived at through trial and error. Given more time, analysis of data may have led us to better selection, but observational results led us to believe these values introduced appropriate noise to the prediction model.

[1.0, 0.0, 0.0, 0.0]
[0.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 0.3, 0.0]
[0.0, 0.0, 0.0, 0.3]

2. The average angle was added to the **y velocity of the 2D x matrix** during prediction phase of the filter, which allowed for directional change predictions.

3. **Next state matrix F** has the Δt value set to 1.5 at the start and then governed by a maximum average speed of 7 and minimum of 6. We found that at times our speed would become too fast or too slow based on actuals. Often speed would get extremely high and distances between points became so great that accuracy was adversely impacted. Note that these values only slowed or sped up our prediction. They did not limit the distances between points to be within a range of 6 to 7.

It should be pointed out that we assumed the Q matrix would be enough to allow the EKF to properly account for variances in speed and heading, however we found we still need the average angle tuning and the speed control with the next state matrix. Additionally, the R matrix was employed during testing with the Q but results for velocity control were inconclusive. Consequently, we ended up with what we call a modified EKF.

Modified EKF Flow (with "noise reduction"):

Find the "best" 10 positions from the file.
Analyze this known data by:

- EKF Prediction (applying covariance)
- EKF Update
- Calculate angle between last two points
- Use angle and previously calculated angles to determine angle average

After 10 positions are processed, start generating predicted positions:

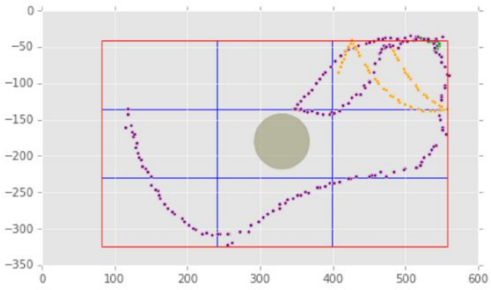
- Use last two actual points from the file to determine first prediction measurement
- EKF Prediction (applying covariance and average angle update)
- Calculate distance between last two points
- Use distance and previously calculated distances to determine distance average
- Update F matrix (next state function) to speed up or slow down the subsequent prediction

3. Reducing noise in some areas

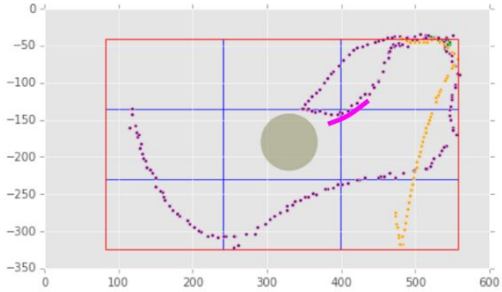
One area we noticed could potentially be improved was when the last 10 steps used for the (E)KF were noisy (either because it was just interacting with a

wall, or it just had noisy measurement data). One example where this is very apparent, is if we test against predicting the 120th-60th last step of test10.

EKF



EKF (with noise reduction)



In the above 2 images, the left is the EKF, and the right is the EKF with noise reduction. Both images share the same actual path (with green dots indicating the first 20 steps, and the purple dots showing the remaining 190 steps). The orange is the prediction in both images. In the EKF we can see that the prediction is based off a tighter grouping of 10 points leading to a stunted and slightly misdirected prediction. In the EKF with noise reduction scenario, we “walk back” through the prior 30 steps to see if there’s a set of 10 steps with a better spaced curve that is in the same heading direction as the original 10 steps. If there is a better space curved (which we see there is with the underscored pink above), we superimpose that to the last robot position and apply EKF to that set of 10 steps.

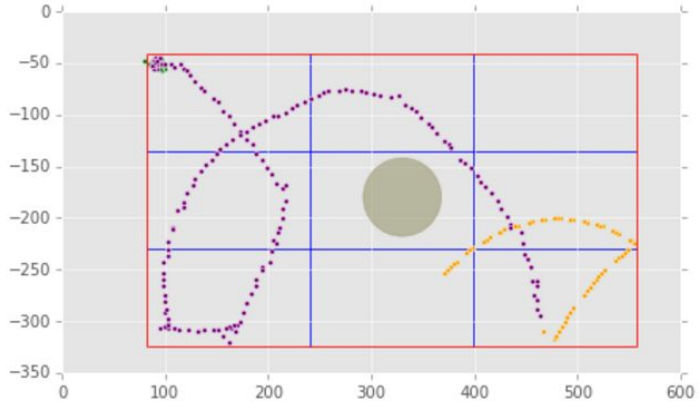
In the above example, the EKF had an L2 error of 830.8 while the EKF with noise reduction had an L2 error of 378.7. We can also see that visually, the EKF with noise reduction tracks better with the robot’s actual next 60 steps.

Collision Handling

The robot and our predictions encounter 3 main collision scenario, and 1 pseudo collision scenario, which we categorized as 1) colliding with one of the four walls, 2) colliding with the candle, 3) encountering a corner (i.e. colliding with 2 walls in a small time range), and 4) being in a fallen state. Collision scenarios 1 & 2 take place independently from the collision, whereas 3 & 4 (at times) play a factor in to the prediction. The way our team accounted for each of these collision scenarios was the following:

1. **Collision with one of the four walls.**

   Wall collisions were handled entirely by reflecting the path off the wall once there was a point in the path that went beyond one of our defined walls.

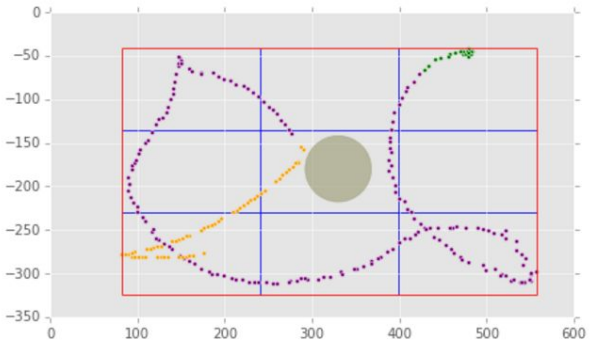


We can see in the above example (using test04) that our prediction (the orange dots) collides with 2 walls. The 1st collision with the bottom wall takes place near the beginning of the prediction and the 2nd collision with the right wall takes place shortly before the midpoint of the prediction. Had we not factored in collision handling, this orange dots would have extended beyond the south wall with no bounds.

2. **Collision with the candle:**

   There were 2 possible candle collision outcomes; One was “grazing” the candle; the other is “bouncing off” the candle.

   - The grazing scenario checks that the path does not go within the circle for more than 4 out of 5 consecutive steps. If it meets this criteria, we essentially “nudge” those steps that fell within the candle, to go outside of the candle and we treat the remaining prediction points as unchanged.
   - The “bouncing off” the candle requires that at least 4 out of 5 consecutive steps of the prediction are within the candle. When this is met, we reflect (the same as we do with wall collisions) off of the axis that is closest to either the top most, right most, left most or bottom most part of the circle.



We can see in the above example (using test08) that our prediction (the orange dots) collides with the candle and reacts by bouncing off of it. The bounce is a reflection off approximately the vertical line at x=295. It knows to do the bounce because if it continued without a course correction, there would be more than 4 out of 5 consecutive steps that would have been in the candle.

3. **Encountering a corner**

   We don't do any path correction using "corner detection" logic. However, we do consider corners when trying to identify pockets of short average distance. While we didn't calculate the averages of the corner logic, we noticed (both from videos and from snapshots of data) that the robot has short distances between steps when it's in the corner. Basically, we wanted to identify corners in order to not mistake corners with pseudo collision 4 which we speak about next.

4. **Fallen state**

   This occurs when the robot flips over (as we can see at the end of test07). It is a rare state and when it happens, the distances traveled by the robot are much smaller relative to its other states. In order to detect this state, we look for periods (say 40 steps) of steps where the distance between (the max recorded X and max recorded Y) and (the min recorded X and the min recorded Y) are below 80. This actually allows us to apply a custom prediction for this state itself which we essentially output a controlled random scatter around the last point using a random gaussian distribution.

Testing and Validation

At many stages throughout our approach, we started testing early and often. We went with a champion and challenger testing method, where we started with the base KF as a champion and replaced it with a better (challenger) version only if it outperformed in a consistent, sound manner.

Our testing method consisted of stepping back in groups of 60 steps for each of the 10 test files, and comparing our 60 predicted steps with the actual 60 steps the robot took. For our comparisons, we used the same method we'll be graded on, i.e. the L2 error between our predicted data and the actual data. We did this for a total of 10 different groups, so we stepped back for the 60, 120, 180, 240, 300, 360, 420, 480, 540, and 600 steps. In total, this meant that each major iteration of our approach had 100 full set comparisons of 60 predictions vs. 60 actual measurements. This approach allowed us to be comfortable that our predictions were not overfitting, as long as we did not see wide ranges in the results across these sets.

Example of L2 test results with the EKF version

test file ->	1	2	3	4	5	6	7	8	9	10		min	max	average (of middle 8)
error score (last 60) ->	1606.9	924.4	438.2	836.1	707.6	465.4	92.6	342.5	921.8	520.0		92.6	1606.9	644.5

In the above we can see that L2 error from each test data set, along with the min, max and average for the entire “last 60” set. We repeated this same thing for “120-60”, “180-120”... all the way to “600-540”. So our entire result for the EKF portion of our approach was:

Min, max, and average of middle 8 for all 10 sets

grouping	60-0	120-60	180-120	240-180	300-240	360-300	420-360	480-420	540-480	600-540
minimum ->	92.6	115.4	80.7	117.4	196.3	248.2	256.6	217.8	380.8	382.3
maximum ->	1606.9	1445.7	2087.1	1860.9	1510.3	2441.6	1129.4	1430.2	1630.6	2501.7
average of middle 8 ->	644.5	820.8	843.1	760.1	790.0	658.5	802.5	802.9	826.4	864.5

Summarizing the “average of the middle 8”, we can get summary stats for EKF resulting L2 error being:

Avg Group L2: 781.3 Median Group L2: 802.7 Min: 644.5 Max: 864.5

Comparison of L2 summary stats for each iteration

Version	Avg	Median	Min	Max	Range
KF	880.6	904.7	613.5	1100.7	487.2
EKF	781.3	802.7	644.5	864.5	220.0
EKF+b10	751.8	764.5	640.6	840.3	199.7
EKF+aggressive_b10	762.1	778.2	644.5	864.7	220.2

The above are the L2 summary stats we used to inform our “champion vs. challenger” method. From the above, we can see that the EKF outperformed the KF in nearly every summary stat. Also, we can see that the “EKF+b10” which is our EKF+noise reduction, performed slightly better than EKF. The green highlighted cells show the best performance across all of the versions. This shows how we arrived at the

“EKF+b10” as the best version we could complete within the provided time. We explored the “EKF+aggressive_b10” version which was a more aggressive attempt at noise reduction, but because it did not the other versions in any significant way, we decided not to use it.

Alternative Methods

We considered earlier on in the project about many approaches we could use to predict the robot motion. One possible way to use a particle filter would be to train a particle filter for different velocities in the map. This approach sounds more error-prone than the Kalman filter, though, because it won't be possible to train the robot at all possible points, so instead, the training would be more like an unmoving robot with the map moving underneath with special-casing at the edges.

Using particle filtering is possible, but we were wary that the parameters to pull for particle filters would be less intuitive, and had the chance to be more computationally intensive. Measurement noise might also get exacerbated with particle filtering. For example, the videos likely don't cover the robot moving over the whole map. That can be remedied, but we’d also need to account for the robot's motion, so, basically, one will need to combine the results from several frames, thus, several PF results, further increasing the error potential.

We also looked at an averaging approach, similar to what some team members did for Runaway Robot. However, testing with that showed results worse than a straight KF. This likely is due to Runaway Robot having a very narrowly defined “cone of error” as it was moving roughly in a circle. In this case we are dealing with a problem more like Runaway Robot 5, where there is a widely defined noise set, making averaging far less successful.

Further considerations

With the allotted time for this project, we had to make some concessions knowing that we could not explore every avenue since we had to manage limited resources. Some thoughts about more areas we would explore include:

1. We intentionally decided to not use any non-native packages in our program as we wanted to keep things as simple as possible (for our own development & to reduce the risk of complications for the submission to TAs for grading). We’re only using native packages such as math, random, ast, os, and sys. Given more time, we would love to explore the use of powerful array packages such as numpy that would allow us to compute sets of curves in a very quick manner.
2. Our collision handling was a good balance of “practical physics” along with “straightforward implementation”. We essentially reflect off of walls, and have 2

different interaction types with the candle. Given more time, we would explore more precise collision handling include better interaction with a circular object as described here: http://math.stackexchange.com/questions/81269/angle-of-reflection-off-of-a-circle

3. We certainly feel that we can reduce noise even more so that we can provide the EKF with a very clean initial set. We attempted an aggressive noise reducer, but it resulted in performance that could not beat our original noise handler. If we were given more time, we could probably find additional ways to reduce noise perhaps by giving weight to a heading that was not done in the recent steps.
4. We had an action item at one point to explore K-nearest neighbors as a way to store "common reactions" to previously encountered scenarios. However, no one on our team had the chance to look at this in great detail. If we had more time, we'd probably further inform the robot's prediction with a randomly averaged sample of prior reactions to common scenarios like its current scenario.

Conclusion

This problem started off with a lot of ambiguity. We were able to work in logical phases to first understand the scope or limits of the problem, analyze manageable sets of data as well as aggregate stats over large sets of data, and come up with an iterable approach to predict the robot's path.

Our iterable approach started with a strong Kalman Filter, expanded into an Extended Kalman Filter, and finally ended up being an Extended Kalman Filter with noise reduction. At each step of the way, we confirmed our belief that the new version was better than previous by proving it had consistently better L2 performance stats over a large set of tests. We were happy to arrive at a fairly consistent, low-variation, good-performing EKF approach with a built in noise reduction.

References

The Extended Kalman Filter: An Interactive Tutorial for Non-Experts - Washington and Lee University

Robotics 2 Target Tracking - Kai Arras, Cyrill Stachniss, Maren Bennewitz, Wolfram Burgard

Localization Using Extended Kalman Filters in Wireless Sensor Networks - Ali Shareef, Yifeng Zhu