UDACITY

PROJECT

Unscented Kalman Filters

A part of the Self-Driving Car Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Hi,
This is a very good submission. Well done. Please keep up the good work.

- I'd like to invite you to look at this link to this documents on UKF. I personally use it a lot in order to better understand these filters.
https://www.seas.harvard.edu/courses/cs281/papers/unscented.pdf

Compiling

Code must compile without errors with `cmake` and `make`.

Given that we've made CMakeLists.txt as general as possible, it's recommended that you do not change it unless you can guarantee that your changes will still compile on any platform.

Excellent work, your code compiles successfully with no mistake.

Accuracy

For the new data set, your algorithm will be run against "obj_pose-laser-radar-synthetic-input.txt". We'll collect the positions that your algorithm outputs and compare them to ground truth data. Your px, py, vx, and vy RMSE should be less than or equal to the values [.09, .10, .40, .30].

For the older version of the project, your algorithm will be run against "sample-laser-radar-measurement-data-1.txt". We'll collect the positions that your algorithm outputs and compare them to ground truth data. Your px, py, vx, and vy RMSE should be less than or equal to the values [0.09, 0.09, 0.65, 0.65].

For the older version, your algorithm will also be run against "sample-laser-radar-measurement-data-2.txt". The RMSE for the second data set should be <=[0.20, 0.20, 0.55, 0.55].

Great job here, when the algorithm is run against "obj_pose-laser-radar-synthetic-input.txt", your px, py, vx, and vy RMSE is less than the ground truth. When I run it, these are the results I obtained.

Follows the Correct Algorithm

While you may be creative with your implementation, there is a well-defined set of steps that must take place in order to successfully build a Kalman Filter. As such, your project should follow the algorithm as described in the preceding lesson.

Well done, all the steps required to implement a Kalman Filter has been well defined and correctly implemented.

Your algorithm should use the first measurements to initialize the state vectors and covariance matrices.

Excellent work here. Your algorithm effectively uses the first measurements to initialize the state vectors and covariance matrices.

Upon receiving a measurement after the first, the algorithm should predict object position to the current timestep and then update the prediction using the new measurement.

Nice, the algorithm predicts the object position at the current time step and then update it using the new measurement.

Your algorithm sets up the appropriate matrices given the type of measurement and calls the correct measurement function for a given sensor type.

Excellent, the algorithm sets up the appropriate matrices depending on the type of the measurement. It efficiently calls the correct measurement function for each sensor.

Code Efficiency

This is mostly a "code smell" test. Your algorithm does not need to sacrifice comprehension, stability, robustness or security for speed, however it should maintain good practice with respect to calculations.

Here are some things to avoid. This is not a complete list, but rather a few examples of inefficiencies.

- **Running the exact same calculation repeatedly when you can run it once, store the value and then reuse the value later.**
- **Loops that run too many times.**
- **Creating unnecessarily complex data structures when simpler structures work equivalently.**
- **Unnecessary control flow checks.**

Well done, there's is no unnecessary loop that runs too many times, and no unnecessary control flow checks.

DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ