UDACITY

PROJECT

Extended Kalman Filters

A part of the Self Driving Car Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW 9

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

2 SPECIFICATIONS REQUIRE CHANGES

Amazing submission. You are so close to the end result. With a little more tuning of your noise values, or your initial P, you should be able to get your RMSE down below the rubric. Please don't be disheartened, but instead imagine that this small an RMSE difference could mean the difference between your car localising a child crossing the road correctly or missing it. Since the things we do are so safety related, we also need extreme diligence in doing them. I'm sure your next submission is going to be amazing.

Compiling

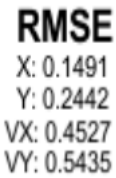
Code must compile without errors with `cmake` and `make`.

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.

Accuracy

Your algorithm will be run against Dataset 1 in the simulator which is the same as "data/obj_pose-laser-radar-synthetic-input.txt" in the repository. 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 [.11, .11, 0.52, 0.52].

The RMSE I got running your code were `0.14 0.24 0.45 0.54`. Unfortunately this does not meet the rubric. I've given some hints in the code annotations which should help improve.



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!

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

Changes required

Reset the noise values of `ax` and `ay` back to 9 in this case! You are right, that ideally, these are tunable parameters, but in this case, to reduce the complexity of the project, 9 has been chosen as the optimum noise variance. With 9, and the rest of the code implemented correctly, you should be able to achieve the rubric RMSE!

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.

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

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.**

Good job overall! I've given some suggestions as part of the code annotations to make your solution even better!

RESUBMIT

DOWNLOAD PROJECT

9 CODE REVIEW COMMENTS ›

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH

Rate this review

Student FAQ Reviewer Agreement