UDACITY

‹ Return to "Self-Driving Car Engineer" in the classroom

DISCUSS ON STUDENT HUB

Unscented Kalman Filters

REVIEW

CODE REVIEW 6

HISTORY

Meets Specifications

Congrats! You made a strong submission.
You did a great job tweaking your noise parameters and your code is very well structured.

Here are a few articles. I recommend you check them out to see how fellow students deal with issues like NIS consistency.

State Estimation with Kalman Filter
Credits: Malintha Fernando
How a Kalman filter works
Credits: Arun
Kalman filter: Intuition and discrete case derivation
Credits: Vivek yadav

I wish you all the success for your next submission of Kidnapped Vehicle Project.

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.

Good! The setup is smooth.

Accuracy

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].

Good job! Your results meet the requirements.
These are the values I get when testing your algorithm:

```
 Accuracy - RMSE:
0.0698
0.0839
0.3625
0.2208
```

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.

Your algorithm should use 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.

Good job using the standard equations for the linear LIDAR case.

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.

Overall good job! But check out the suggestions in the code review.

DOWNLOAD PROJECT

6 CODE REVIEW COMMENTS ❯

RETURN TO PATH

Rate this review