UDACITY

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

DISCUSS ON STUDENT HUB

Extended Kalman Filters

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

This is a perfect submission. You have a very good understanding of underlying concepts. Congratulations on successfully completing the 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.

Great job here! I was able to successfully compile your program without any compilation errors!

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].

Great job! You were able to achieve the required RMSE with:

```
Accuracy - RMSE:
0.0973
0.0855
0.4513
0.4399
```

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.

Your algorithm correctly initializes with the respective sensor measurement and its respective state 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.

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.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review