UDACITY

PROJECT SPECIFICATION

Extended Kalman Filters

Compiling

CRITERIA	MEETS SPECIFICATIONS
Your code should compile.	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

CRITERIA	MEETS SPECIFICATIONS

CRITERIA	MEETS SPECIFICATIONS
px, py, vx, vy output coordinates must have an RMSE <= [.11, .11, 0.52, 0.52] when using the file: "obj_pose-laser-radar-synthetic-input.txt" which is the same data file the simulator uses for Dataset 1.	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].

Follows the Correct Algorithm

CRITERIA	MEETS SPECIFICATIONS
Your Sensor Fusion algorithm follows the general processing flow as taught in the preceding lessons.	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.

CRITERIA	MEETS SPECIFICATIONS
Your Kalman Filter algorithm handles the first measurements appropriately.	Your algorithm should use the first measurements to initialize the state vectors and covariance matrices.
Your Kalman Filter algorithm first predicts then updates.	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 Kalman Filter can handle radar and lidar measurements.	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

CRITERIA	MEETS SPECIFICATIONS

CRITERIA	MEETS SPECIFICATIONS
Your algorithm should avoid unnecessary calculations.	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.

Suggestions to Make Your Project Stand Out!

There are two ways we think you could make your project stand out:

1. While we're giving this project to you with starter code, you are not actually required to use it! If you think you can organize your Kalman Filter better than us, go for it! Also, this project was templatized in an object-oriented style, however it's reasonable to build a Kalman Filter in a functional style. Feel free to start from scratch with a functional algorithm!

    - Keep in mind that your code *must* compile. If your changes necessitate modifying CMakeLists.txt, you are responsible for ensuring that *any* reviewer can still compile your code given the dependencies listed earlier in the instructions - platform specific errors will *not* be debugged by graders.

2. There is some room for improvement with the Kalman Filter algorithm. Maybe some aspects of the algorithm could be combined? Maybe some could be skipped under certain circumstances? Maybe there are other ways to improve performance? Get creative!

creative!

3. Analyze what happens when you turn off radar or lidar. Which sensor type provides more accurate readings? How does fusing the two sensors' data improve the tracking results?