Unscented Kalman Filters

Compiling

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
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.	The code compiles and the executable(UnscentedKF) is created.

Accuracy

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
px, py, vx, vy output coordinates must have an RMSE <= [.09, .10, .40, .30] 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 "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].	The RMSE values ends with [0.0689, 0.0811, 0.3319, 0.2284] approximately. We can clearly see algorithm converging with ground truth as we get more measurements. Initially, with values of 30 for both std_a_ and std_yawdd_ the values have been high and

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
		after trying multiple values and trial and error around the current values of 1.5 and 0.5 we achieved significantly lower values.

Follows the Correct Algorithm

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
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.	Care is taken to follow the flow suggested in slide 3 of the project lesson. https://classroom.udacity.com/nanodegrees/nd013/parts/40f38239-66b6-46ec-ae68-03afd8a601c8/modules/0949fca6-b379-42af-a919-ee50aa304e6a/lessons/c3eb3583-17b2-4d83-abf7-d852ae1b9fff/concepts/5aecba15-634c-4e15-b994-30edb03d64f9
Your Kalman Filter algorithm handles the first measurements appropriately.	Your algorithm should use the first measurements to initialize the state vectors and covariance matrices.	The initializationsare done in lines 21-53 of ukf.cpp file.
Your Kalman Filter algorithm first predicts then updates.	Upon receiving a measurement after the first, the algorithm should predict object	This flow has been taken care in UKF::ProcessMeasurement function between

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
	position to the current timestep and then update the prediction using the new measurement.	lines 134-153 in ukf.cpp Additionally, the initialization for the first measurement is also taken care using the code in lines 107-132 in ukf.cpp.
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.	Most of the code for this part is reused from lectures. First up; Prediction: 1. Generate Sigma Points, 2. Predict Sigma Points, 3. Predict Mean and Covariance flow has been defined in UKF::Prediction() function in lines 161-316 in ukf.cpp Next, Update step has been applied for both lidar and radar measurements. Update: 1. Predict Measurement, 2. Update State has been applied. The code is present in lines 322-409 for UKF:UpdateLidar() function and in lines 415-522 for UKF::UpdateRadar() function.

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
		I have the code from lectures stored under lecture_code folder. Mostly it was straight forward and matter of variable name changes. Only major thing I found was the calculation of radar and lidar NIS calculations. But a bit of browsing on the Internet gave me the formula for calculating the NIS and used the same. I found the fomula in the following links. 1. https://core.ac.uk/download/pdf/95153642.pdf (section 5.4) 2. http://www.frc.ri.cmu.edu/projects/emergencyresponse/radioPos/KFtheory.html under measurement gating. 3. Adaptive Filtering for Single Target Tracking

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
		http://www.dtic.mil/dtic/tr/fulltext/u2/a507663.pdf section 2.3.2 One question/request I have is that, how can we use NIS to tune the UKF! As part of the part project instructions, it clearly says that I dont need to modify anything other than ukf.cpp and tools.cpp. Without modifying any other files, is it possible. My understanding is that I have to meddle around with the main.cpp for this particular tuning. Please clarify/advise. The other code I have added is in tools.cpp which I took from the lectures for CalculateRMSE functions and it is lines 12-48 in tools.cpp

Code Efficiency

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
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.	Since a lot of the code was reused from lectures, I didn’t have to do anything fancy. But I tried to incorporate the Tips given in slide 4 of the project lesson.