CarND-Extended-Kalman-Filter-P1

Overview:

This project implement an Extended Kalman Filter with C++.

A simulator provided by Udacity generates noisy RADAR and LIDAR measurements of the position and velocity of an object.

They communicate using uWebSockets. The goal of Extended Kalman Filter is to predict the position of the object.

ELK Algorithms: (criteria #3, #4, #5, #6)

Data will flow in from simulator. The data is either from Lidar or Radar. 1) The 1st data point (Lidar or Radar) will be added in measurement list 2) The P matrix is initialized 3) previous time stamp is set

For the rest of data points from data stream: 1) calculate F 2) caluculate Q 3) run Predict() 4.1) if the data point is from Laser Initialize H = H_laser (predefined) Initialize R = R_laser (predefined) run Update() 4.2) if the data point is from Radar Initialize H = H_Jacobian = CalculateJacobian(x) Initialize R = R_radar (predefined) run UpdateEKF()

Criteria #1: Compiling and executing the project

emily@emily-OptiPlex-790:~/TERM2/PROJ_01_KF/CarND-Extended-Kalman-Filter-Project$ (cd build/; make) Scanning dependencies of target ExtendedKF [20%] Building CXX object CMakeFiles/ExtendedKF.dir/src/FusionEKF.cpp.o [40%] Linking CXX executable ExtendedKF [100%] Built target ExtendedKF

CRITERIA #2: RMSE(px, py, vx, vy) << [.11, .11, 0.52, 0.52].

Previous version failed to meet the RMSE requirements. Changes made : to use atan2() instead of atan and make sure -pi < phi < pi improved outcome.

data set #1 : new RMSE: 0.0973178 0.0854597 0.451267 0.439935

data set #1 : Old RMSE: 1.70509 3.33037 2.93471 4.08678

data set #2 : new RMSE: 0.0725678 0.0964738 0.421634 0.493199

data set #2 : Old RMSE: 2.4963 2.44957 3.20215 2.80432

Criteria #7: Code efficiency

The function update() and updateEKF() could be factor out but I decide to leave it there for easier to reader or debug

Additional functions are created to make code easier to read and debug:

MatrixXd Tools::CalculateJacobian(const VectorXd& x_state) MatrixXd Tools::Calculate_Q(float noise_ax , float noise_ay , float dt) MatrixXd Tools::Calculate_F (float dt) MatrixXd Tools::Calculate_P (float p_value=1000) VectorXd Tools::Convert_Polar2Cartesian (float rho, float rho_dot, float phi) VectorXd Tools::Convert_Cartesian2Polar (const VectorXd& x_vector)