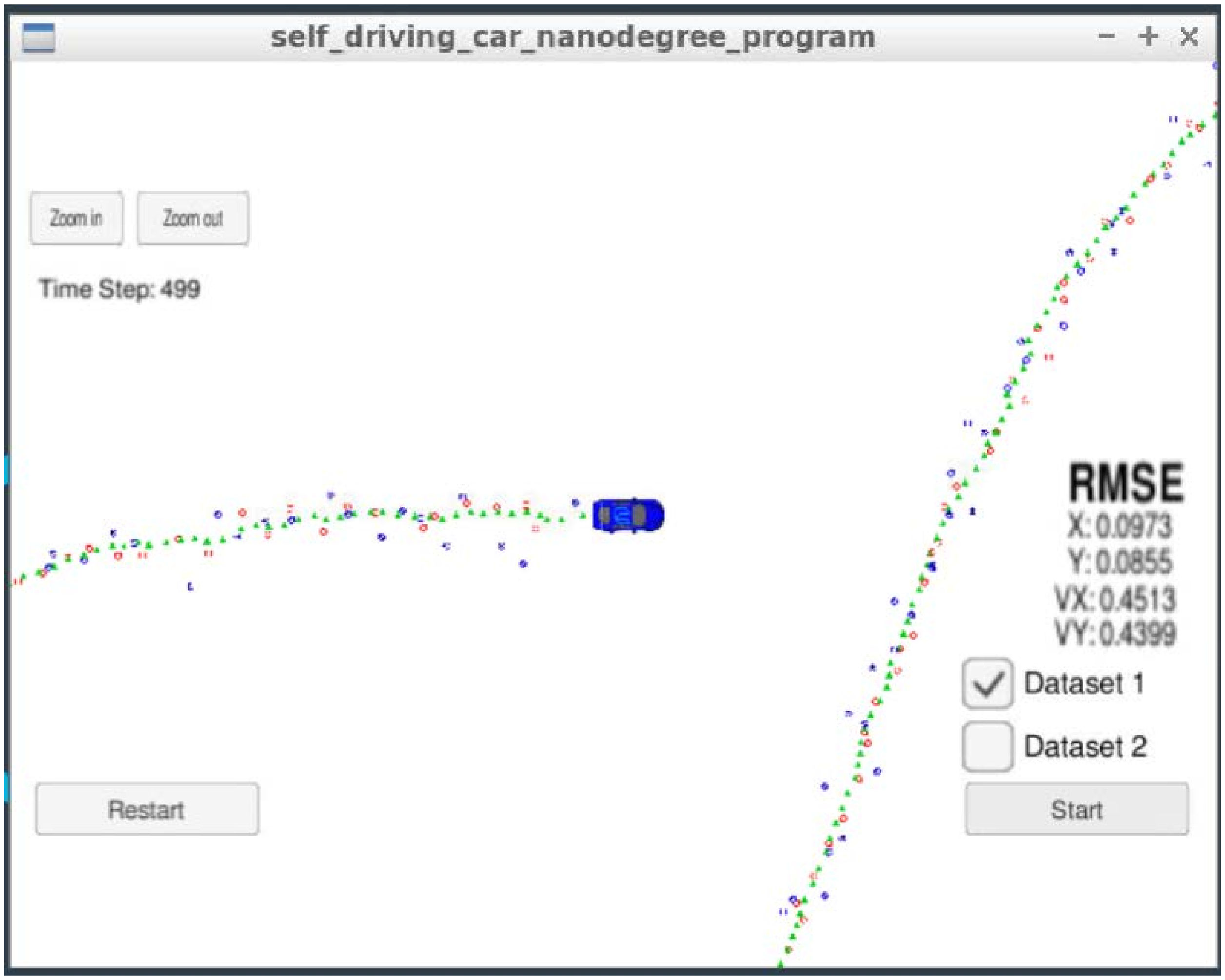
Udacity Self Driving Car Nanodegree: Extended Kalman Filter project

Objective:

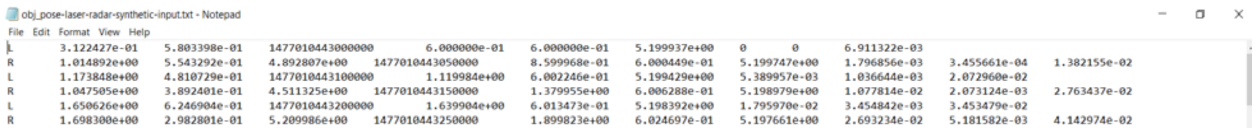
The objective of this project was to implement an extended Kalman filter in C++. The project provided simulated Lidar and Radar measurements detecting a bicycle that travels around a vehicle in a Simulator. We needed to use a Kalman filter, Lidar measurements and Radar measurements to track the bicycle's position and velocity.

Project Details and Approach:



In the picture above, Lidar measurements are red circles, radar measurements are blue circles with an arrow pointing in the direction of the observed angle, and estimation markers (determined by my program) are green triangles.

Data file:
The data file provided for the project had Lidar and radar measurement data. The data file looks like the following image:



obj_pose-laser-radar-synthetic-input.txt - Notepad
File Edit Format View Help

```
L	3.122427e-01	5.803398e-01	1477010443000000	6.000000e-01	6.000000e-01	5.199937e+00	0	0	6.911322e-03
R	1.014892e+00	5.543292e-01	4.892807e+00	1477010443050000	8.599968e-01	6.000449e-01	5.199747e+00	1.796856e-03	3.455661e-04	1.382155e-02
L	1.173848e+00	4.810729e-01	1477010443100000	1.119984e+00	6.002246e-01	5.199429e+00	5.389957e-03	1.036644e-03	2.072960e-02
R	1.047505e+00	3.892401e-01	4.511325e+00	1477010443150000	1.379955e+00	6.006288e-01	5.198979e+00	1.077814e-02	2.073124e-03	2.763437e-02
L	1.650626e+00	6.246904e-01	1477010443200000	1.639904e+00	6.013473e-01	5.198392e+00	1.795970e-02	3.454842e-03	3.453479e-02
R	1.698300e+00	2.982801e-01	5.209986e+00	1477010443250000	1.899823e+00	6.024697e-01	5.197661e+00	2.693234e-02	5.181582e-03	4.142974e-02
```

For a row containing radar data, the columns are: sensor_type ("R" for radar), rho_measured, phi_measured, rhodot_measured, timestamp, x_groundtruth, y_groundtruth, vx_groundtruth, vy_groundtruth, yaw_groundtruth, yawrate_groundtruth.

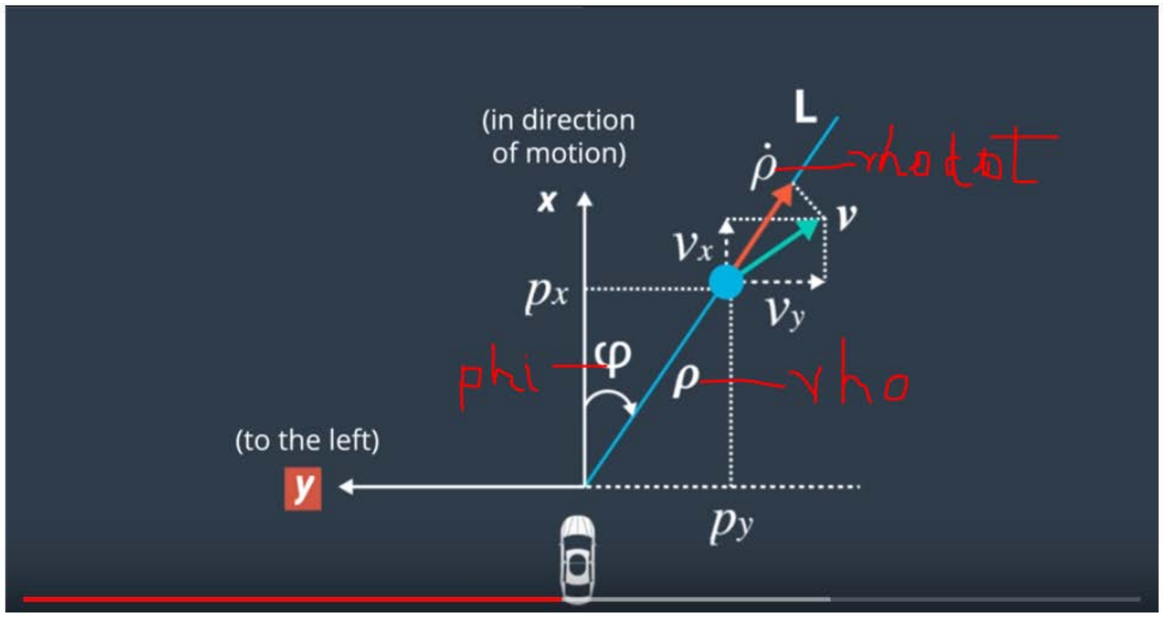
For a row containing Lidar data, the columns are: sensor_type ("L" for Lidar), x_measured, y_measured, timestamp, x_groundtruth, y_groundtruth, vx_groundtruth, vy_groundtruth, yaw_groundtruth, yawrate_groundtruth.

Yaw and yaw rate are to be ignored for this project.

<u>Lidar and radar measurements</u>
In the picture below, x axis is the direction of vehicle movement and y axis is to the left.

Lidar measurements are in Cartesian coordinate system: px and py in the below picture. Lidar does not measure velocity.

Radar measurements are in the polar coordinate system. The radar can directly measure range (i.e. radial distance from origin, symbol is rho), bearing (i.e. angle between range and x axis; symbol is phi) and range rate (i.e. radial velocity, symbol is rho dot).



<u>Kalman Filter and Extended Kalman Filter</u>
Kalman Filter, in summary, involves a continual loop of State Prediction and Measurement Update functions. In State Prediction step, the algorithm uses the information it has to predict the state (of

bicycle around the car, in this case) until the next measurement arrives. In the Measurement Update, a new measurement (from the sensors Lidar/Radar) is used to adjust our belief of the state of the bicycle.

While the Kalman Filter can handle linear motion and measurement, an Extended Kalman Filter can be used to handle nonlinear motion and measurement models. In case of a LIDAR measurement update, we will apply a Kalman Filter because the measurements from the sensor are Linear. But in case of a Radar measurement update, we need to apply Extended Kalman Filter because it includes angles that are nonlinear.

A mathematical equation called Taylor Series (as shown in picture below) is used to get a Linear Approximation of the Non Linear Function.

$$f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f'''(a)}{3!}(x-a)^3+\cdots.$$

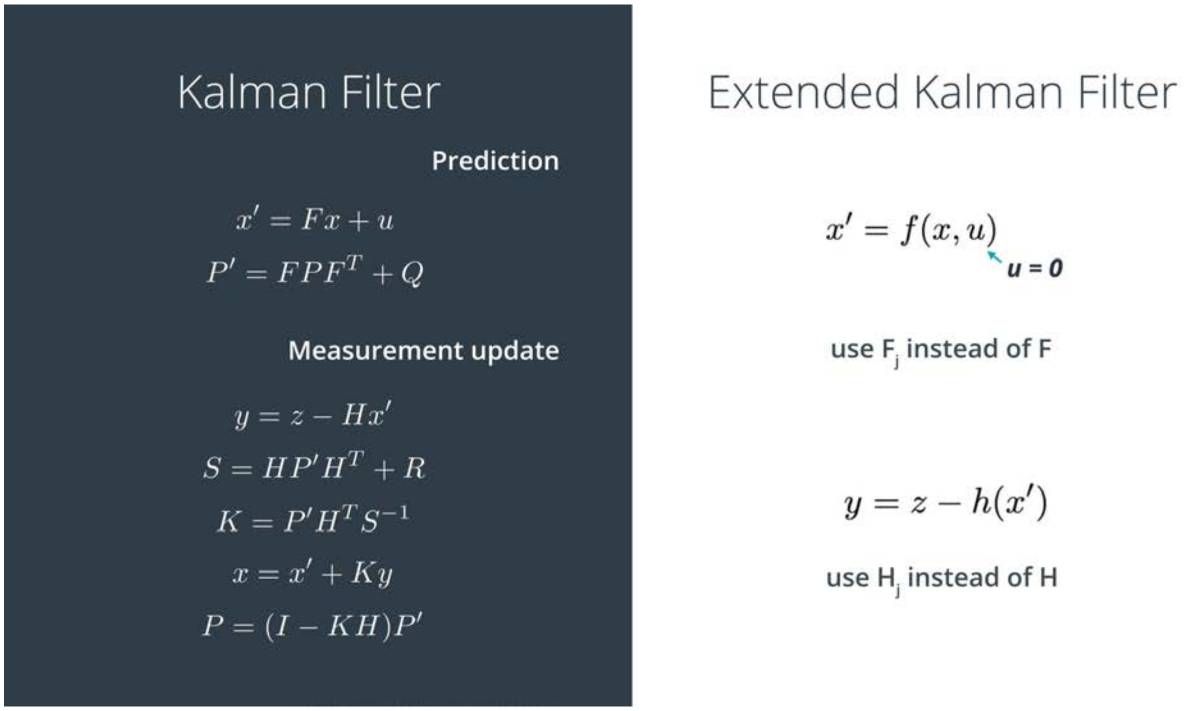
Since we are interested in linearizing, we just consider the first derivative of Taylor series. The first derivative is implemented thru a Jacobian matrix. The Jacobian for EKF looks like the following:

$$H_j=\begin{bmatrix}\frac{\partial\rho}{\partial p_x} & \frac{\partial\rho}{\partial p_y} & \frac{\partial\rho}{\partial v_x} & \frac{\partial\rho}{\partial v_y}\\ \frac{\partial\varphi}{\partial p_x} & \frac{\partial\varphi}{\partial p_y} & \frac{\partial\varphi}{\partial v_x} & \frac{\partial\varphi}{\partial v_y}\\ \frac{\partial\dot{\rho}}{\partial p_x} & \frac{\partial\dot{\rho}}{\partial p_y} & \frac{\partial\dot{\rho}}{\partial v_x} & \frac{\partial\dot{\rho}}{\partial v_y}\end{bmatrix}$$

Which calculates to following:

$$H_j=\begin{bmatrix}\frac{p_x}{\sqrt{p_x^2+p_y^2}} & \frac{p_y}{\sqrt{p_x^2+p_y^2}} & 0 & 0\\ -\frac{p_y}{p_x^2+p_y^2} & \frac{p_x}{p_x^2+p_y^2} & 0 & 0\\ \frac{p_y(v_xp_y-v_yp_x)}{(p_x^2+p_y^2)^{3/2}} & \frac{p_x(v_yp_x-v_xp_y)}{(p_x^2+p_y^2)^{3/2}} & \frac{p_x}{\sqrt{p_x^2+p_y^2}} & \frac{p_y}{\sqrt{p_x^2+p_y^2}}\end{bmatrix}$$

Equations for Kalman Filter and Extended Kalman Filter are as per the below image:


Kalman Filter

Prediction

$$x' = Fx + u$$
$$P' = FPF^T + Q$$

Measurement update

$$y = z - Hx'$$
$$S = HP'H^T + R$$
$$K = P'H^T S^{-1}$$
$$x = x' + Ky$$
$$P = (I - KH)P'$$

Extended Kalman Filter

$$x' = f(x, u)$$

u = 0

use F_j instead of F

$$y = z - h(x')$$

use H_j instead of H

Notations:
X=State matrix
F=State Transformation Matrix
u=Control variable matrix
P=Process Covariance Matrix (represents error in the estimate/process)
K=Kalman Gain
R=Sensor Noise Covariance Matrix (Measurement Error)
I=Identity matrix
z=Measurement of the state
H=Measurement transition matrix

x’ and P’ are Predicted state and Predicted Process Covariance matrix respectively.

As seen from the image above, Kalman filter equations and extended Kalman filter equations are very similar. The main differences are:

- The F matrix will be replaced by Fj when calculating P'.
- The H matrix in the Kalman filter will be replaced by the Jacobian matrix Hj when calculating S, K, and P.
- To calculate x’, the prediction update function, f, is used instead of the F matrix.
- To calculate y, the h function is used instead of the H matrix.

-

For this project, however, we do not need to use the f function or Fj. This is because we are using a linear model for the prediction step. So, for the prediction step, we can still use the regular Kalman filter equations and the F matrix rather than the extended Kalman filter equations.

The equation y=z−Hx′ for the Kalman filter does not change to y=z−Hjx for the extended Kalman filter. Instead, for extended Kalman filters, the equation will be y=z-h (x'); we'll use the h function directly to convert our Cartesian space to polar space as per the image below:

$$h(x') = \begin{pmatrix} \rho \\ \phi \\ \dot{\rho} \end{pmatrix} = \begin{pmatrix} \sqrt{p'^2_x + p'^2_y} \\ \arctan(p'_y / p'_x) \\ \frac{p'_x v'_x + p'_y v'_y}{\sqrt{p'^2_x + p'^2_y}} \end{pmatrix}$$

The angle phi (which is the second value in the h (x') matrix) needed to be normalized in the y vector so that its angle is between –pi and pi.

Files for the project:

Following files were provided for the project:

1. obj_pose-laser-radar-synthetic-input- data file (Lidar and radar measurements; and the ground truth data)
2. main.cpp-Communicates with the Udacity Simulator; receives the data measurements; call a function to run the Kalman Filter; calls a function to calculate RMSE
3. FusionEKF.cpp- initializes the filter, calls the predict function, calls the update function
4. kalman_filter.cpp- defines the predict function, update function for Lidar, and the update function for radar
5. tools.cpp- function to calculate RMSE and the Jacobian matrix
6. kalman_filter.h - defines the KalmanFilter class
7. measurement_package.h - defines the MeasurementPackage class

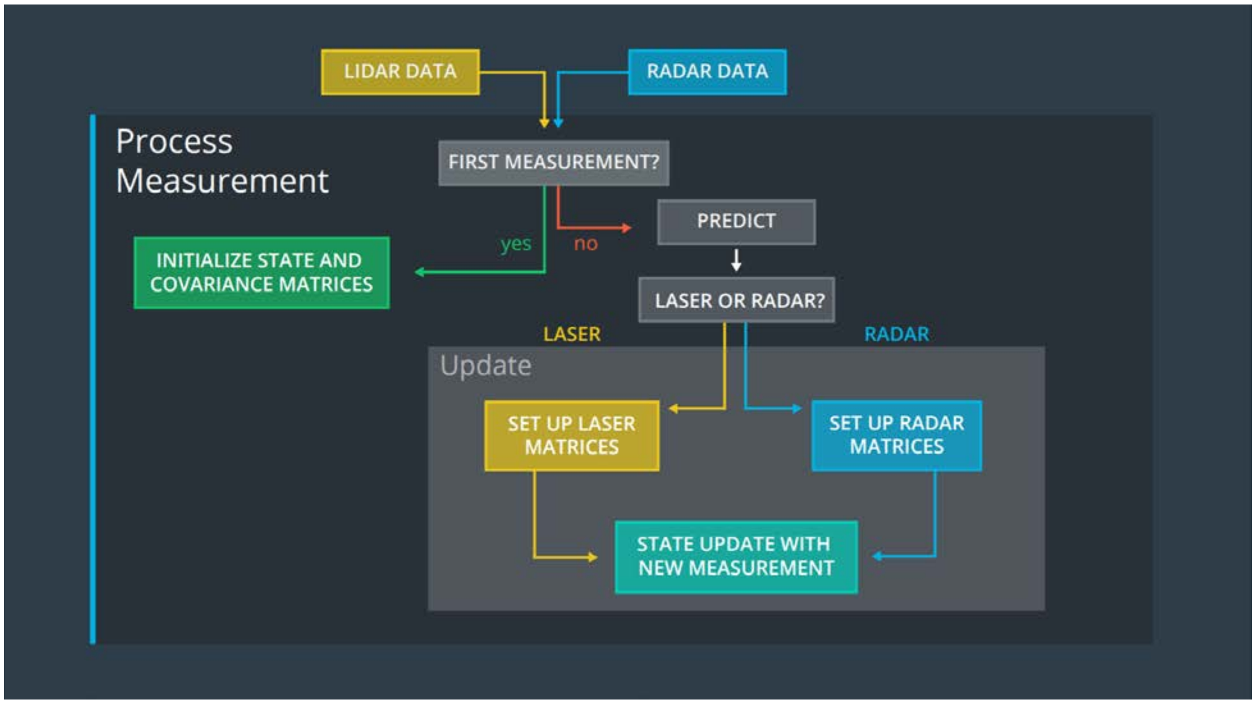
Project work summary:

Requirement for the project was to write the code in the following files so that in the EKF tracking the bicycle around the car; the RMSE values for px,py,vx and vy are less than .11,.11,.52 and .52 respectively for the dataset provided in the : "obj_pose-laser-radar-synthetic-input.txt" file which is used by the Udacity simulator.

1. In tools.cpp, fill in the functions that calculate root mean squared error (RMSE) and the Jacobian matrix.
2. Fill in the code in FusionEKF.cpp. You'll need to initialize the Kalman Filter, prepare the Q and F matrices for the prediction step, and call the radar and Lidar update functions.
3. In kalman_filter.cpp, fill out the Predict(), Update(), and UpdateEKF() functions.

Results:

The program followed the suggested flow (as per the image below) to build the Kalman filter for sensor fusion and resulted in desired RMSE values for px, py,vx and vy.


LIDAR DATA
RADAR DATA
Process Measurement
FIRST MEASUREMENT?
yes
no
INITIALIZE STATE AND COVARIANCE MATRICES
PREDICT
LASER OR RADAR?
LASER
RADAR
Update
SET UP LASER MATRICES
SET UP RADAR MATRICES
STATE UPDATE WITH NEW MEASUREMENT