Extended Kalman Filter

Behavioral Cloning Project

The goals/steps of this project are the following:

- To utilize a Kalman filter to estimate the state of a moving object of interest with noisy lidar and radar measurements.
- The RMSE values that are lower than the tolerance outlined in the project rubric.

Files Submitted & Code Quality

My project includes the following files:

- main.cpp containing the script to read measurements and interact with the simulator.
- FusionEKF.cpp for initializing variables, initializing the Kalman filters, and calling functions that implement the prediction step and update step.
- kalman_filter.cpp that contains the predict and update equations for lidar and radar data.
- Tools.cpp to calculate the RMSE and the Jacobian matrix used for radar measurements.

Implementation approach

FusionEKF.cpp

- The measurement covariance matrices, R_laser_ and R_radar_ , for laser and radar data respectively, are initialized.
- The measurement matrix, H_laser_ for laser measurements is initialized.
- The state transition matrix ekf_.F_ and state covariance matrix ekf_.P_ are initialized.
- The code for the same is available between lines 21-57 in FusionEKF.cpp.

- When we get the first measurement, the Kalman filter will try to initialize the object's location with the sensor measurement.
- If the measurement comes from the radar, the measurements are converted from polar coordinates to cartesian coordinates to get the initial positions in x and y-direction. The initial velocities are set to 0, as it is not directly measured.

- If the measurement comes from the laser, initial positions in x and y-direction are directly set using the measurement data. The initial velocities are set to 0, as it is not directly measured.
- The code for the same is available between lines 70-100 in FusionEKF.cpp.

- The state transition matrix, ekf_.F_ is updated based on the elapsed time between subsequent measurements. This needs to be done at every step as the time between measurements is not constant.
- process noise covariance matrix, ekf_.Q_ is initialized using the following equation:

$$Q = GQ_\nu G^T = \begin{pmatrix} \frac{\Delta t^4}{4}\sigma_{ax}^2 & 0 & \frac{\Delta t^3}{2}\sigma_{ax}^2 & 0 \\ 0 & \frac{\Delta t^4}{4}\sigma_{ay}^2 & 0 & \frac{\Delta t^3}{2}\sigma_{ay}^2 \\ \frac{\Delta t^3}{2}\sigma_{ax}^2 & 0 & \Delta t^2\sigma_{ax}^2 & 0 \\ 0 & \frac{\Delta t^3}{2}\sigma_{ay}^2 & 0 & \Delta t^2\sigma_{ay}^2 \end{pmatrix}$$

- The code for the same is available between lines 121-134 in FusionEKF.cpp.

- Next, the Kalman filter prediction step is called.
- To call Kalman filter update/measurement step we need to check if the measurements are from radar or lidar, as radar measurements are non-linear and we need to use Extended Kalman filter equations for radar, while standard Kalman filter equations are used for lidar.
- Before calling the Extended Kalman filter update step, we need to calculate the Jacobian matrix used in EKF equations in place of measurement matrix, H.
- The code for the same is available between lines 136-159 in FusionEKF.cpp.

kalman_filter.cpp

- The predict and update functions of the Kalman filter are defined.
- The predict function is the same for both the radar and laser measurements.
- The prediction step of the Kalman filter is defined using the following equations:

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

- The code for the prediction step is available between lines 28-33 in kalman_filter.cpp.

- As lidar uses linear equations, the update step for lidar data uses the standard Kalman filter equations shown below:

$$y = z - Hx'$$
$$S = HP'H^T + R$$
$$K = P'H^T S^{-1}$$

$$x = x' + Ky$$
$$P = (I - KH)P'$$

- The code for the lidar update step is available between lines 36-49 in kalman_filter.cpp.

- As radar uses non-linear equations, so the update step involves linearizing the equations with the Jacobian matrix. Also, the measurements are in polar coordinates and these are mapped into measurement space using h(x′) function. Therefore, in the EKF equation, the error equation becomes:

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

Where h(x′) is calculated using the following equation:

$$h(x') = \begin{pmatrix} \sqrt{p_x'^2 + p_y'^2} \\ \arctan(p_y'/p_x') \\ \frac{p_x' v_x' + p_y' v_y'}{\sqrt{p_x'^2 + p_y'^2}} \end{pmatrix}$$

The Jacobian matrix, Hj is used in place of measurement matrix, H. The Jacobian matrix is calculated using the following equation:

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

- The code for radar update step is available between lines 52-85 in kalman_filter.cpp.

tools.cpp

- The calculations for root-mean-square error and the Jacobian matrix is implemented in this file.
- The RMSE is calculated for the predicted state vector and ground truth vector using the following equation:

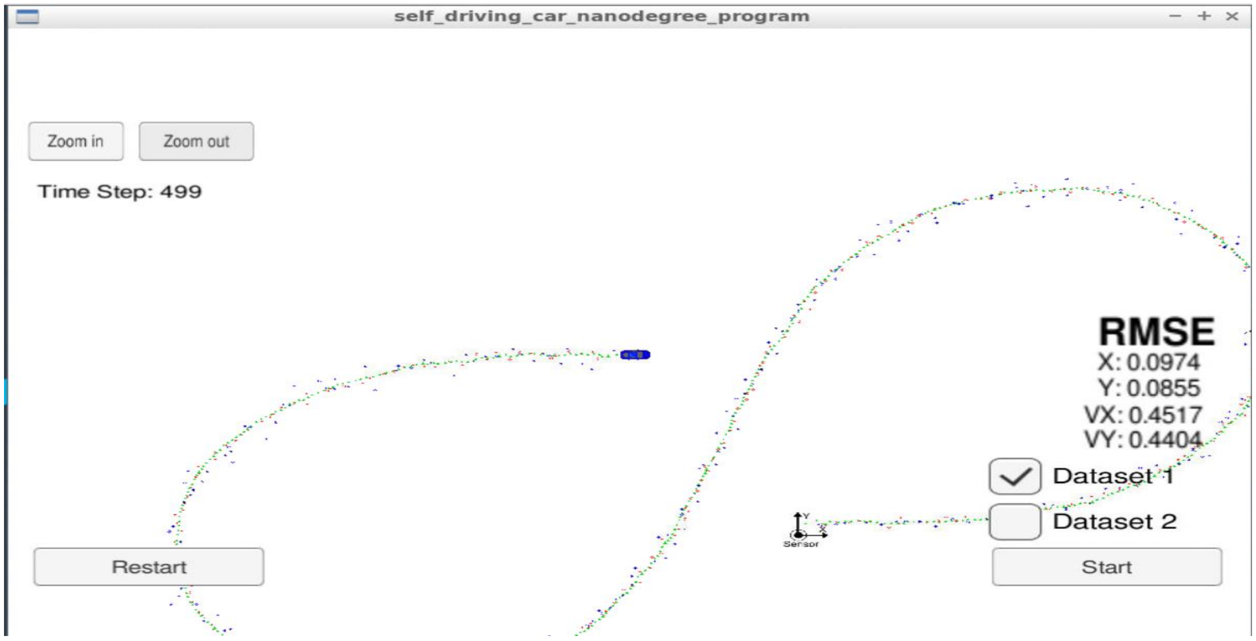
$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(x_t^{est} - x_t^{true})^2}$$

- The code for rmse calculation is available between lines 13-45 in tools.cpp.
- Next, the Jacobian matrix needed for radar measurements is calculated using the Jacobian matrix shown before.
- The code for Jacobian calculation is available between lines 50-81 in tools.cpp.

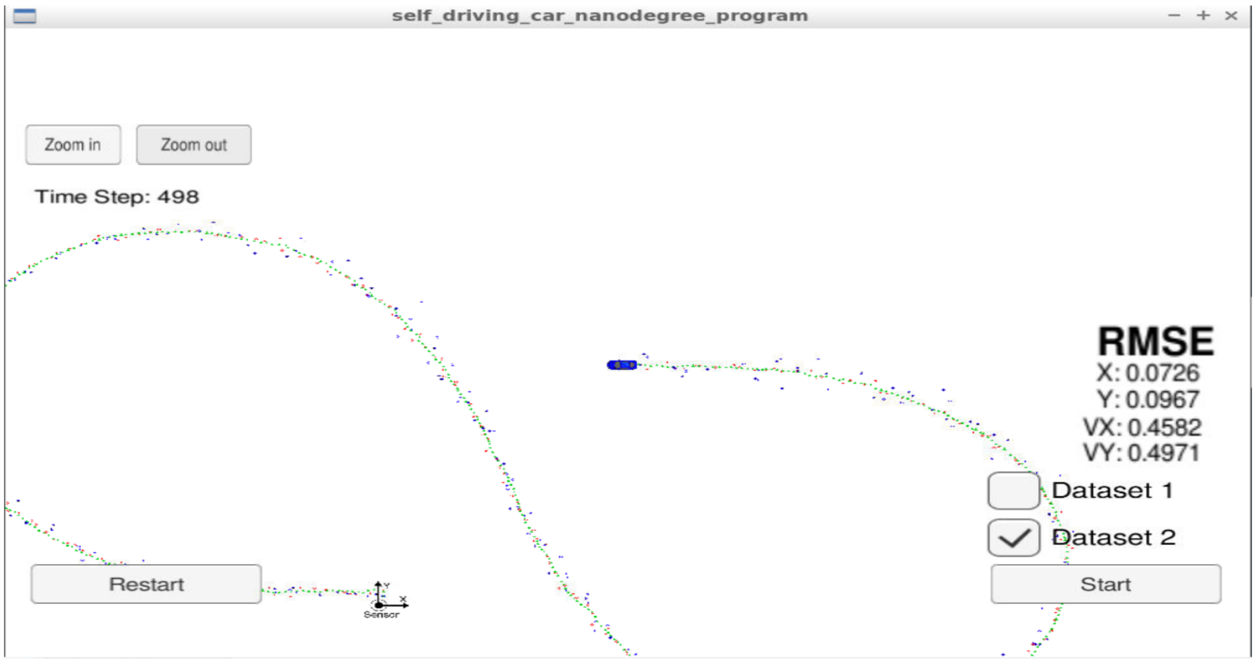
Output

The screenshots below show the tracking performance using the Kalman filter equations. Lidar measurements are red circles, radar measurements are blue circles with an arrow pointing in the direction of the observed angle, and estimation markers are green triangles.

As, it can be seen from the screenshots the RMSE value are within the threshold specified in the project rubric.



Estimation on Dataset 1



Estimation on Dataset 2