Extended Kalman Filter

Overview:

In this project we build Kalman filter and its extended version to track an object's location and velocity relative to our car.

Simulated lidar and radar measurements for detecting a bicycle that travels around our vehicle is given by Udacity.

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. The simulator provides the script the measured data (either lidar or radar), and the script feeds back the measured estimation marker, and RMSE values from its Kalman filter.

Steps involved in Kalman filter:

Below are the three main steps for programming a Kalman filter:

- Initializing Kalman filter variables
- predicting where our object is going to be after a time step Δt
- updating where our object is based on sensor measurements

Then the prediction and update steps repeat themselves in a loop.

To measure how well our Kalman filter performs, we will then calculate root mean squared error comparing the Kalman filter results with the provided ground truth.
These three steps (initialize, predict, update) plus calculating RMSE encapsulate the entire extended Kalman filter project.

Project code:

Here is the overview of files in the project:

- `main.cpp` - communicates with the Term 2 Simulator receiving data measurements, calls a function to run the Kalman filter, calls a function to calculate RMSE.
- `FusionEKF.cpp` - initializes the filter, calls the predict function, calls the update function. Takes the sensor data and initializes variables and updates variables. The Kalman filter equations are not in this file. `FusionEKF.cpp` has a variable called `ekf_`, which is an instance of a `KalmanFilter` class. The `ekf_` will hold the matrix and vector values. `ekf_` instance is used to call the predict and update equations.
- `kalman_filter.cpp`- defines the predict function, the update function for lidar and the update function for radar
- `tools.cpp`- function to calculate RMSE and the Jacobian matrix
- `Measurement_package.h` – Holds sensor type, raw measurements and timestamp details

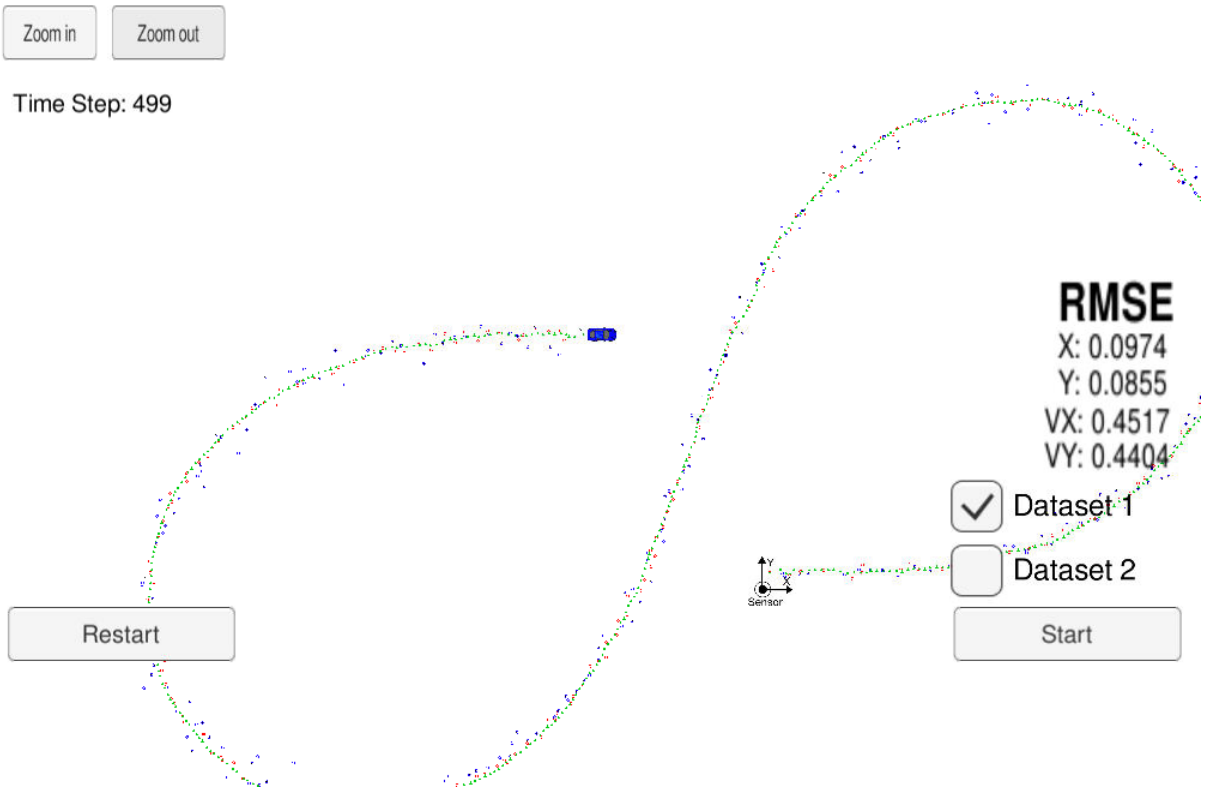
The Kalman Filter algorithm will go through the following steps:

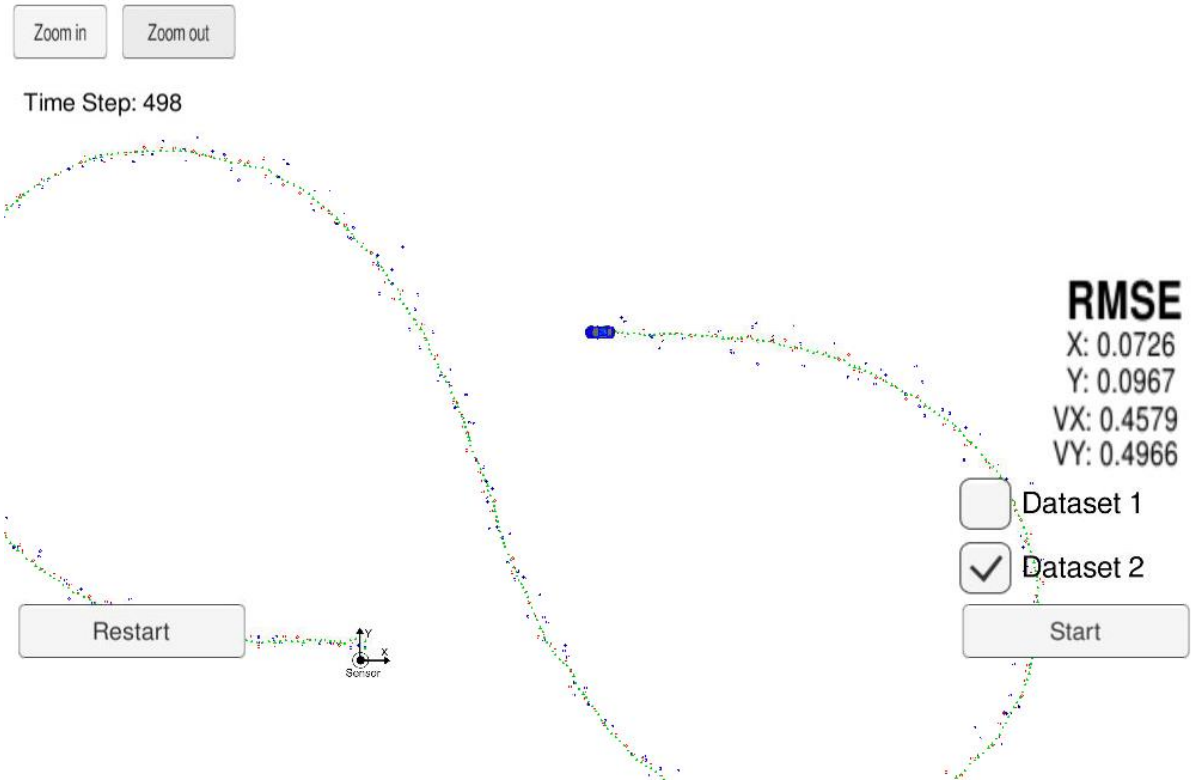
- **first measurement** - the filter will receive initial measurements of the bicycle's position relative to the car. These measurements will come from a radar or lidar sensor.
- **initialize state and covariance matrices** - the filter will initialize the bicycle's position based on the first measurement.
- then the car will receive another sensor measurement after a time period Δ*t*.
- **predict** - the algorithm will predict where the bicycle will be after time Δ*t*. One basic way to predict the bicycle location after Δ*t* is to assume the bicycle's velocity is constant; thus the bicycle will have moved velocity Δ*t*. In the extended Kalman filter lesson, we will assume the velocity is constant.
- **update** - the filter compares the "predicted" location with what the sensor measurement says. The predicted location and the measured location are combined to give an updated location. The Kalman filter will put more weight on either the predicted location or the measured location depending on the uncertainty of each value.
- then the car will receive another sensor measurement after a time period Δ*t*. The algorithm then does another **predict** and **update** step.

Results and Conclusion:

EKF models the object state x : (px, py, vx, vy) where px, py represent objects position wrt to x and y axis on a 2D plane and vx and vy represent velocity respectively. State is initialized with (px, py, 1, 1) when first measurement is received. Covariance matrix P is initialized with (1,1,1000,1000) to reflect the uncertainty.

Following images show the results from simulator on DataSet1 and DataSet2:





The effect of individual sensor measurements can be analyzed by turning off the updates for a particular sensor. It is seen that the overall RMSE value for position in the absence of one sensor is higher than the position RMSE values when both sensors are used. In particular, the RADAR sensor does a poor job of position estimation due to the sensing mechanism used by the sensor i.e. Doppler effect to directly measure velocity. The LIDAR does a relatively better job of estimating position, however, velocity measurements are not better in the absence of RADAR data.

This demonstrates the importance of an EKF which is able to take measurements from multiple sensor types and generate a state estimation more accurate than estimations from individual sensors by combining the Gaussian probability distributions of the measurements with its own prediction of where the tracked object is located.