Read Me File
Project Extended Kalman Filter, Project 1, SDCND Term 2:

Initialization
The task is to finish the FusionEKF and set the noises.
I initialize H_laser as a 2 by 4 matrix ans set the values. From section 10 in lesson 5 we learned that

"**H** is the matrix that projects your belief about the object's current state into the measurement space of the sensor. For lidar, this is a fancy way of saying that we discard velocity information from the state variable since the lidar sensor only measures position."

That's why we set the matrix as 1,0,0,0
0, 1,0,0. So only the positions get noticed and vx and vy are disregarded.

I further set the ekf_.P_ and ekf._F_ matrices and I initialize ekf._F_ matrix as a diagonal 4by4.

I set the noises both noise_ax and noise_ay to 9.

Next I initialize the state ekf._x_ with the first measurements. the state is

ekf._x_ = 1,1,1,1 and the first two elements get overwritten by the measurements. In the Q&A Video the person talks about the last two positions which can have an impact on RMSE. In my case I leave it 1, 1 and RMSE works fine for me.

If the measurement comes from **Laser**, the algorithm just uses the x and y measurements as shown here:

```
ekf_.x_ <<
measurement_pack.raw_measurements_[0],measurement_pack.raw_measurements_[1], 0, 0;
```

and if the measurement comes from **Radar** the polar measurements have to be converted to cartesian :

```
float ro = measurement_pack.raw_measurements_[0];
float phi = measurement_pack.raw_measurements_[1];
ekf_.x_ << ro * cos(phi), ro * sin(phi), 0, 0;
```

also make sure to set:

```
previous_timestamp_ = measurement_pack.timestamp_;
```

This finishes the initialization.

After initializing the FusionEKF file we are ready for the Kaman Filter which goes through a two step process of prediction and update.

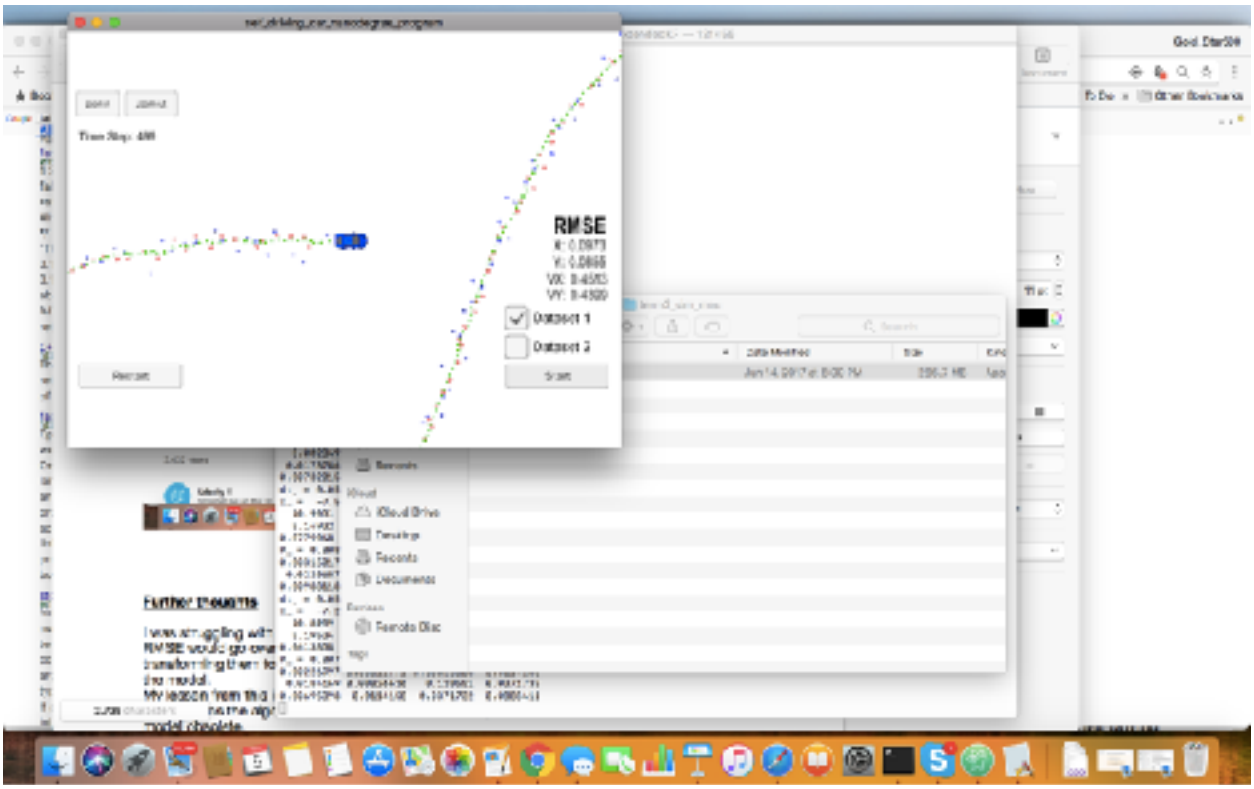
Prediction:
The Kaman Filter has the Predict and Update Functions.
Predict uses the classroom equations for predict in kalman_filter.cpp lines 25-27.

Update distinguishes between Radar and Lidar. The Lidar Update is straightforward as we are dealing with linear data.
H_ * x gives the z prediction. We calculate the error in line 37 (kalman_filter.cpp) y=z-H*x.
In the Radar Update H turns into Hx, which is the Jacobean matrix. Key here is to normalize the data between -pi and pi. The error is y=z-Hx as shown in line 73. This is the key difference between Radar and Lidar.

Tools

The tools.cpp file contains the functions to calculate the Jacobean and the to calculate RMSE.

My RMSE values are shown in the screenshot below.



Further thoughts

I was struggling with the Kaman Filter because after about half the simulation was running my RMSE would go over the limit. It turns out I was not normalizing the polar data points when transforming them to cartesian. I believe this was the problem. It took me a long time to debug the model.
My lesson from this is that the quality of the data and the proper management of the data is as important as the algorithms. In fact, it is even more important since bad data can make the model obsolete.

One question I have is still not clearly answered, which is WHY IS RADAR DATA NONLINEAR. I am not sure about this. It has to be driven by the fact that Radar as opposed to light is a different type of wave form. But this is not clear to me. It would be useful to explain this a bit more.

The link to my code is:

https://github.com/kdelko/Udacity

I have several versions of FusionEKF and kaman_filter, they are named with copy. Please ignore them, just focus on the code in the actual files like FusionEKF.cpp and klaman_filter.cpp