Extended Kalman Filter

Write up of work done for Udacity CarND Extended Kalman Filter Term 2, Project 1, by Mani Srinivasan

Project Objectives

The goals / steps of this project are the following:

1. Read **the repo's README** for more detailed instructions.
2. Complete the Extended Kalman Filter algorithm in C++.
3. Ensure that your project compiles.
    - From the root of the repo:
        1. mkdir build && cd build
        2. cmake .. && make
        3. ./ExtendedKF path/to/input.txt path/to/output.txt
4. Test your Kalman Filter against the sample data. Ensure that the px, py, vx, and vy RMSE are below the values specified in the rubric.
5. Submit your project!

Rubric Points

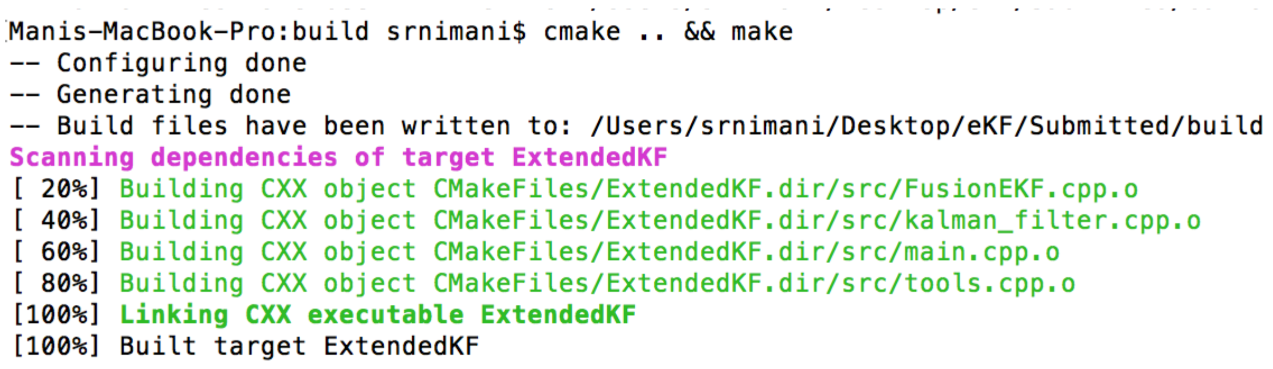
Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

1. Compiling

Your code should compile.

Code compiles without errors with cmake and make..

See the output of the cmake and make steps below.

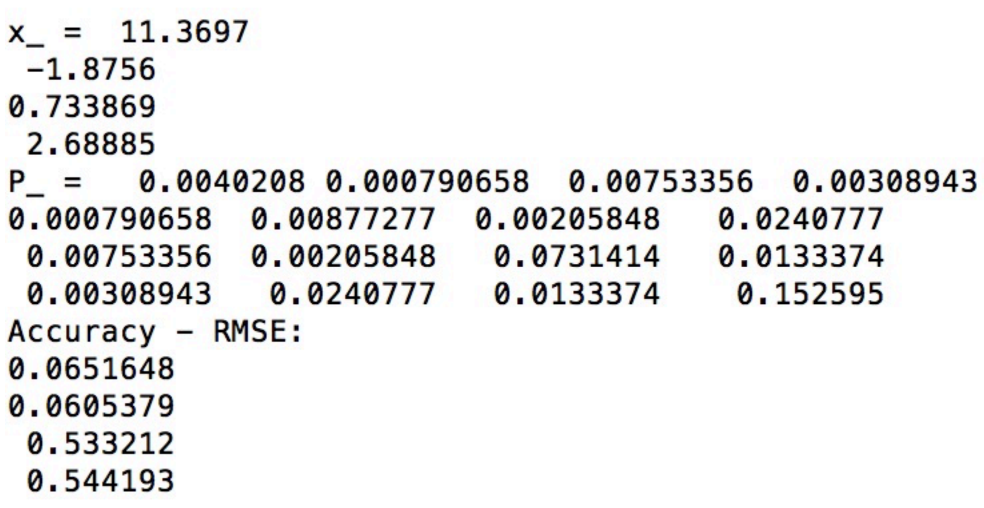


```
Manis-MacBook-Pro:build srnimani$ cmake .. && make
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/srnimani/Desktop/eKF/Submitted/build
Scanning dependencies of target ExtendedKF
[ 20%] Building CXX object CMakeFiles/ExtendedKF.dir/src/FusionEKF.cpp.o
[ 40%] Building CXX object CMakeFiles/ExtendedKF.dir/src/kalman_filter.cpp.o
[ 60%] Building CXX object CMakeFiles/ExtendedKF.dir/src/main.cpp.o
[ 80%] Building CXX object CMakeFiles/ExtendedKF.dir/src/tools.cpp.o
[100%] Linking CXX executable ExtendedKF
[100%] Built target ExtendedKF
```

2. Accuracy

The px, py, vx, vy output coordinates have an RMSE <= [0.08, 0.08, 0.60, 0.60] when using the file: "sample-laser-radar-measurement-data-1.txt".

See the output below (last run):

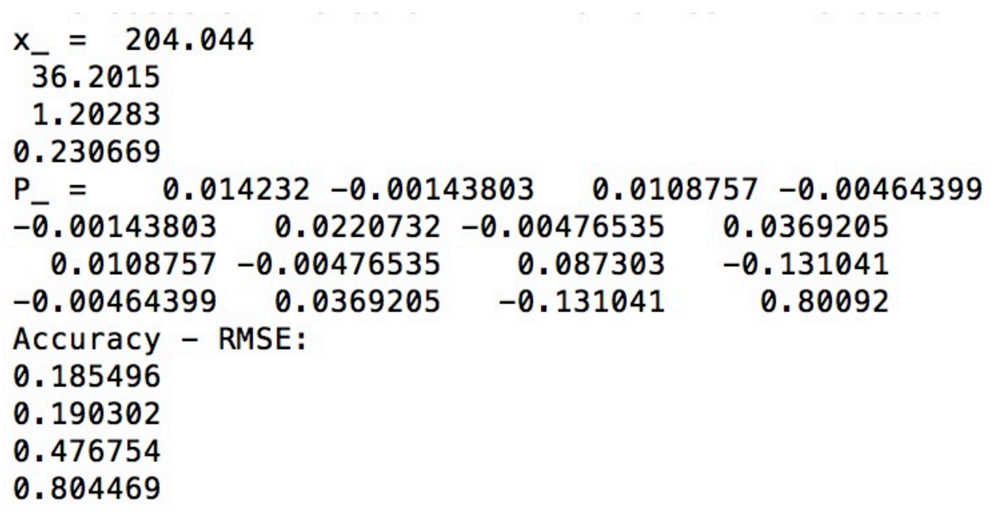


```
x_ =  11.3697
 -1.8756
0.733869
 2.68885
P_ =   0.0040208 0.000790658  0.00753356  0.00308943
0.000790658  0.00877277  0.00205848   0.0240777
 0.00753356  0.00205848   0.0731414   0.0133374
 0.00308943   0.0240777   0.0133374    0.152595
Accuracy - RMSE:
0.0651648
0.0605379
 0.533212
 0.544193
```

Accuracy is well within the limits.

The px, py, vx, vy output coordinates have an RMSE <= [0.20, 0.20, .50, .85] when using the file: "sample-laser-radar-measurement-data-2.txt".

See the output below:



```
x_ =  204.044
 36.2015
 1.20283
0.230669
P_ =    0.014232 -0.00143803   0.0108757 -0.00464399
-0.00143803   0.0220732 -0.00476535   0.0369205
  0.0108757 -0.00476535    0.087303   -0.131041
-0.00464399   0.0369205   -0.131041     0.80092
Accuracy - RMSE:
0.185496
0.190302
0.476754
0.804469
```

Accuracy is well within the limits.

3. Follows the Correct Algorithm

Your Sensor Fusion algorithm follows the general processing flow as taught in the preceding lessons.

The implementation, follows the well-defined set of steps as in the course.

Your Kalman Filter algorithm handles the first measurements appropriately.

The algorithm use the first measurements to initialize the state vectors and covariance matrices.

Your Kalman Filter algorithm first predicts then updates.

Upon receiving a measurement after the first, the algorithm predicts the object position to the current timestep and then update the prediction using the new measurement.

Your Kalman Filter can handle radar and lidar measurements.

The algorithm sets up the appropriate matrices given the type of measurement and calls the correct measurement function for a given sensor type.

4. Code Efficiency

Your algorithm should avoid unnecessary calculations.

I guess my code avoids the following:

- Running the exact same calculation repeatedly when you can run it once, store the value and then reuse the value later.
- Loops that run too many times.
- Creating unnecessarily complex data structures when simpler structures work equivalently.
- Unnecessary control flow checks.