Extended Kalman Filter

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

Project Objectives

The goals / steps of this project are the following:

1. Clone/fork the project's template files from the **project repository**. (Note: Please do not submit your project as a pull request against our repo!)

2. Clone the visualization and data generation utilities from the **utilities repository**.

3. Build an Unscented Kalman Filter by applying the general processing flow as described in the previous lesson.

4. Test your code!

5. When you achieve an RMSE below the required values, submit your project!

Rubric Points

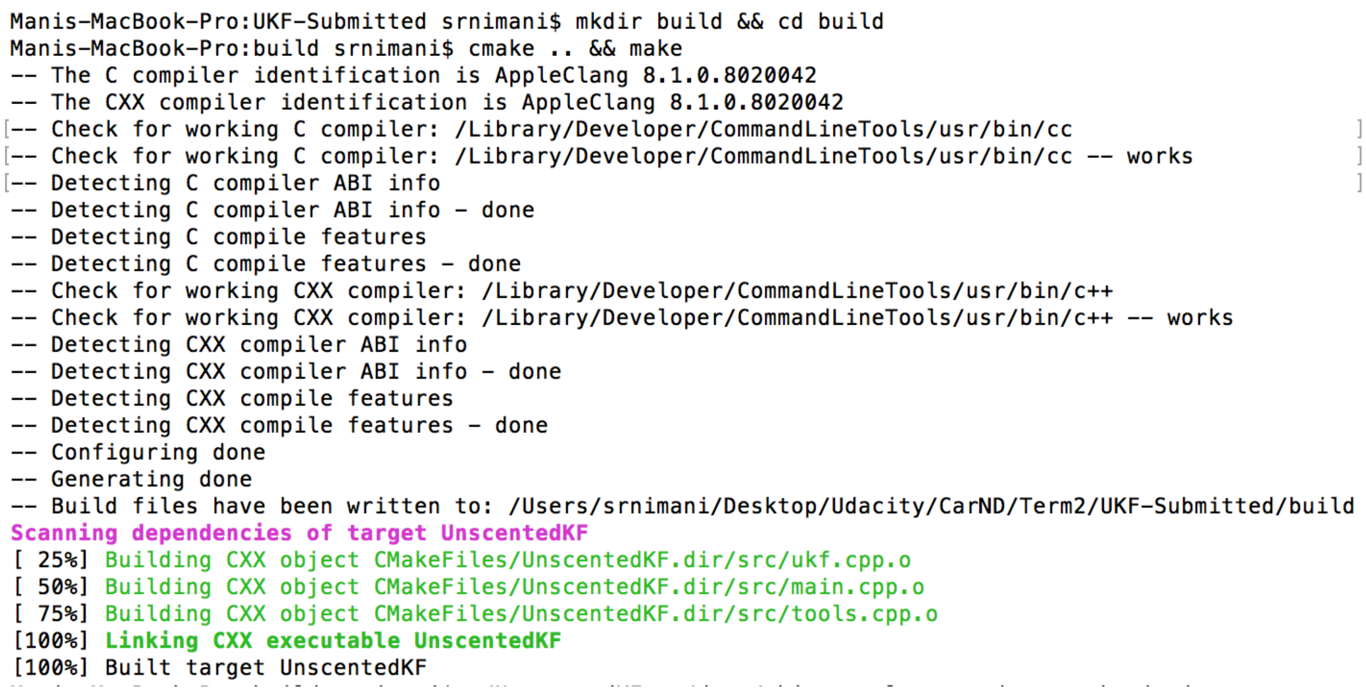
Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

1. Compiling

Your code should compile.

Code compiles without errors with cmake and make..

See the output of the cmake and make steps below.



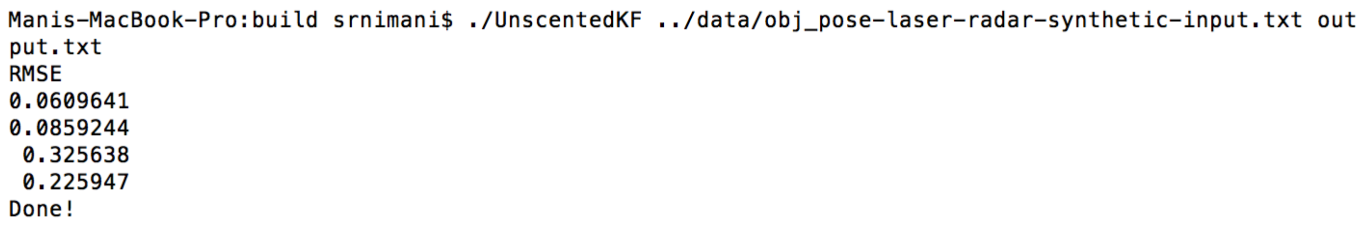
```
Manis-MacBook-Pro:UKF-Submitted srnimani$ mkdir build && cd build
Manis-MacBook-Pro:build srnimani$ cmake .. && make
-- The C compiler identification is AppleClang 8.1.0.8020042
-- The CXX compiler identification is AppleClang 8.1.0.8020042
[-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc
[-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc -- works
[-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /Library/Developer/CommandLineTools/usr/bin/c++
-- Check for working CXX compiler: /Library/Developer/CommandLineTools/usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/srnimani/Desktop/Udacity/CarND/Term2/UKF-Submitted/build
Scanning dependencies of target UnscentedKF
[ 25%] Building CXX object CMakeFiles/UnscentedKF.dir/src/ukf.cpp.o
[ 50%] Building CXX object CMakeFiles/UnscentedKF.dir/src/main.cpp.o
[ 75%] Building CXX object CMakeFiles/UnscentedKF.dir/src/tools.cpp.o
[100%] Linking CXX executable UnscentedKF
[100%] Built target UnscentedKF
```

2. Accuracy

Critria

For the new version of the project, there is now only one data set "obj_pose-laser-radar-synthetic-input.txt". px, py, vx, vy output coordinates must have an RMSE <= [.09, .10, .40, .30] when using the file: "obj_pose-laser-radar-synthetic-input.txt"

See the output below (RMSE) :



```
Manis-MacBook-Pro:build srnimani$ ./UnscentedKF ../data/obj_pose-laser-radar-synthetic-input.txt out
put.txt
RMSE
0.0609641
0.0859244
 0.325638
 0.225947
Done!
```

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.

1. Initialization
2. Prediction
3. Update

I have used the sample code given in the course material while adding my own to complete the project work.

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.