UDACITY

‹ Return to "Self-Driving Car Engineer" in the classroom

DISCUSS ON STUDENT HUB

Unscented Kalman Filters

REVIEW

CODE REVIEW 4

HISTORY

Meets Specifications

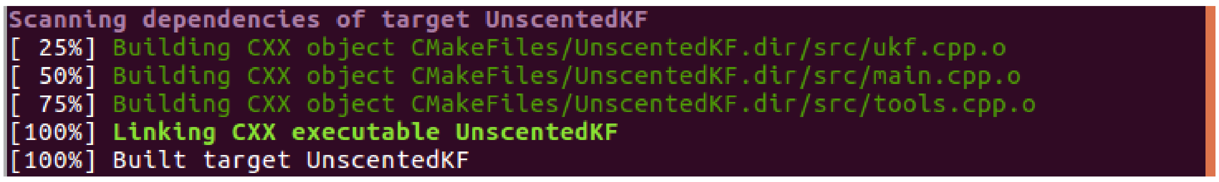
Dear Learner,
I must say this submission was indeed enjoyable to review. I appreciate and commend the efforts and hard work put into this piece. Congratulations👏👏for making it pass this stage of learning with us and I wish that this spirit is carried forward in subsequent projects. You should be proud of yourself because success is no accident. It is hard work, perseverance, learning, studying, sacrifice and most of all, love of what you are doing or learning to do. Please keep practicing on these projects and I wish you all the best.💪

Compiling

Code must compile without errors with `cmake` and `make`.

Given that we've made CMakeLists.txt as general as possible, it's recommended that you do not change it unless you can guarantee that your changes will still compile on any platform.

Good start! The project compiles without errors with `cmake` and `make`, `CMakeLists.txt` was also included in the submission.

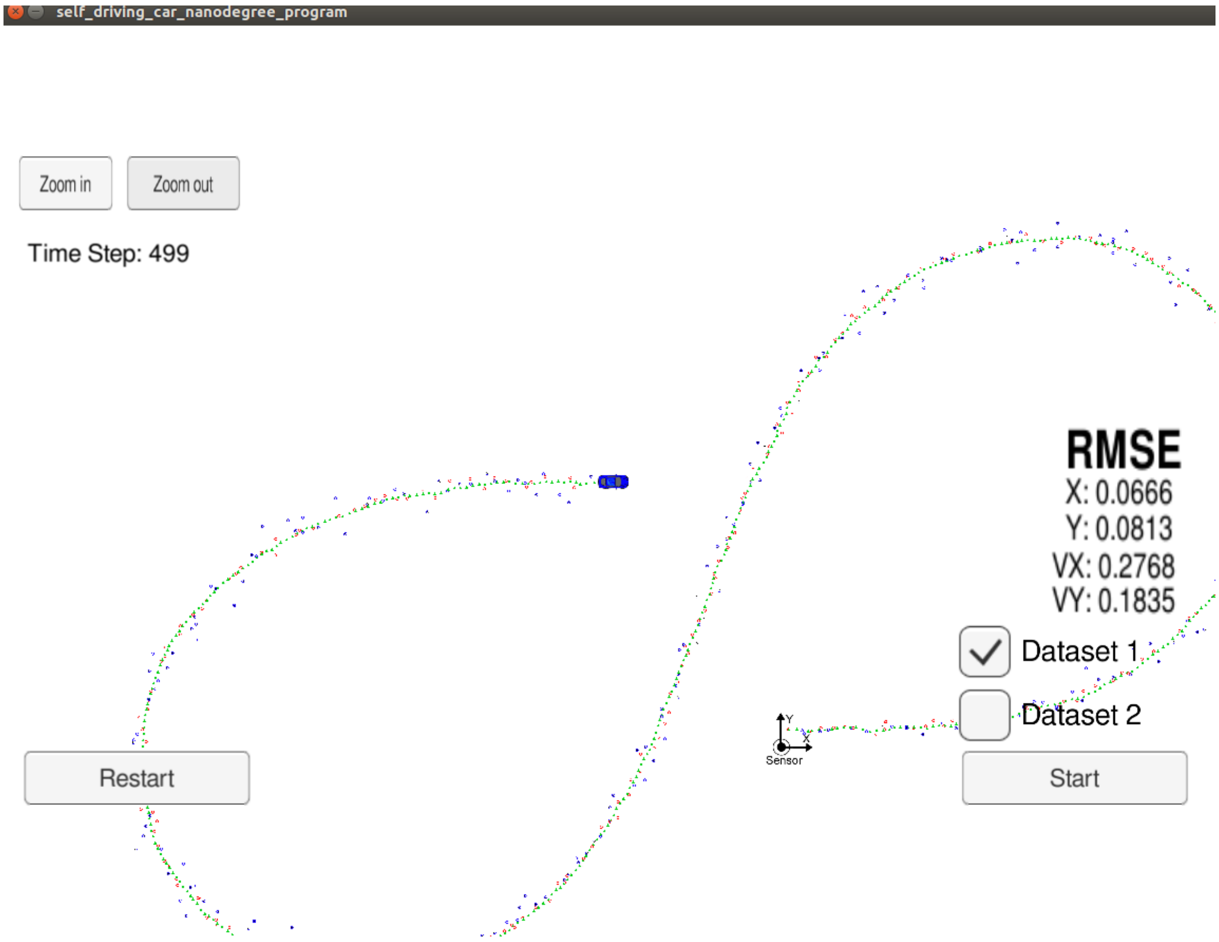


```
Scanning dependencies of target UnscentedKF
[ 25%] Building CXX object CMakeFiles/UnscentedKF.dir/src/ukf.cpp.o
[ 50%] Building CXX object CMakeFiles/UnscentedKF.dir/src/main.cpp.o
[ 75%] Building CXX object CMakeFiles/UnscentedKF.dir/src/tools.cpp.o
[100%] Linking CXX executable UnscentedKF
[100%] Built target UnscentedKF
```

Accuracy

Your algorithm will be run against "obj_pose-laser-radar-synthetic-input.txt". We'll collect the positions that your algorithm outputs and compare them to ground truth data. Your px, py, vx, and vy RMSE should be less than or equal to the values [.09, .10, .40, .30].

On running the project against Dataset 1 in the simulator, the resulting RMSE values meet specifications as shown on the screenshot below. Excellent results!



Follows the Correct Algorithm

While you may be creative with your implementation, there is a well-defined set of steps that must take place in order to successfully build a Kalman Filter. As such, your project should follow the algorithm as described in the preceding lesson.

Good job with the implementation of the UKF algorithm. The work doesn't deviate from the well-defined set of steps for building a Kalman filter. Keep up the good work!👍

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

Nice work! The first measurements were used to initialize the state vector and covariance matrices as expected in ukf.cpp. 👏

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

On receiving measurements after the first, object positions to the current time step are correctly predicted and then updated with new measurements. Good work on implementing the measurement update as a combination of prediction and model update steps.👍

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

A wonderful job is done in this implementation as the algorithm sets up the correct matrices based on the type of measurement (radar or lidar) and accurately calls the correct function (UpdateLidar and UpdateRadar)for that type. Great!

Code Efficiency

This is mostly a "code smell" test. Your algorithm does not need to sacrifice comprehension, stability, robustness or security for speed, however it should maintain good practice with respect to calculations.

Here are some things to avoid. This is not a complete list, but rather a few examples of inefficiencies.

- **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.**

The code was optimized and used simple structures and no unnecessary control flow checks. 👏

Pro Tips

Here are a few tips for improving on code efficiency and optimization:

- Optimizing C++/Writing efficient code/Performance improving features
- Efficient C++ Performance Programming Techniques
- 10 Tips for C and C++ Performance Improvement Code Optimization

DOWNLOAD PROJECT

4 CODE REVIEW COMMENTS ›

RETURN TO PATH

Rate this review