# Extended Kalman Filters

审阅

代码审阅 3

HISTORY

## Meets Specifications

Dear Learner,
I can't believe you made it on the first try... Congrats!👏👏
This was a brilliant submission and I enjoyed reviewing your work. By carefully going through the project, it shows a lot of effort, diligence and above all, understanding of the project. Excellent work! You have successfully passed all the rubrics of this project. All the efforts you put in to complete the project are very much appreciated and it was my pleasure reviewing this great implemented project. Remember, Perfect practice makes perfect. So keep practicing on these projects and I wish you all the best. Please keep up the fantastic work. 💪

### 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.**

By compiling the code using cmake and make, no errors were found as the compilation was 100% successful. Well done!

```
[ 40%] Building CXX object CMakeFiles/ExtendedKF.dir/src/tools.cpp.o
[ 60%] Building CXX object CMakeFiles/ExtendedKF.dir/src/FusionEKF.cpp.o
[ 80%] Building CXX object CMakeFiles/ExtendedKF.dir/src/kalman_filter.cpp.o
[100%] Linking CXX executable ExtendedKF
[100%] Built target ExtendedKF
```

# Tips

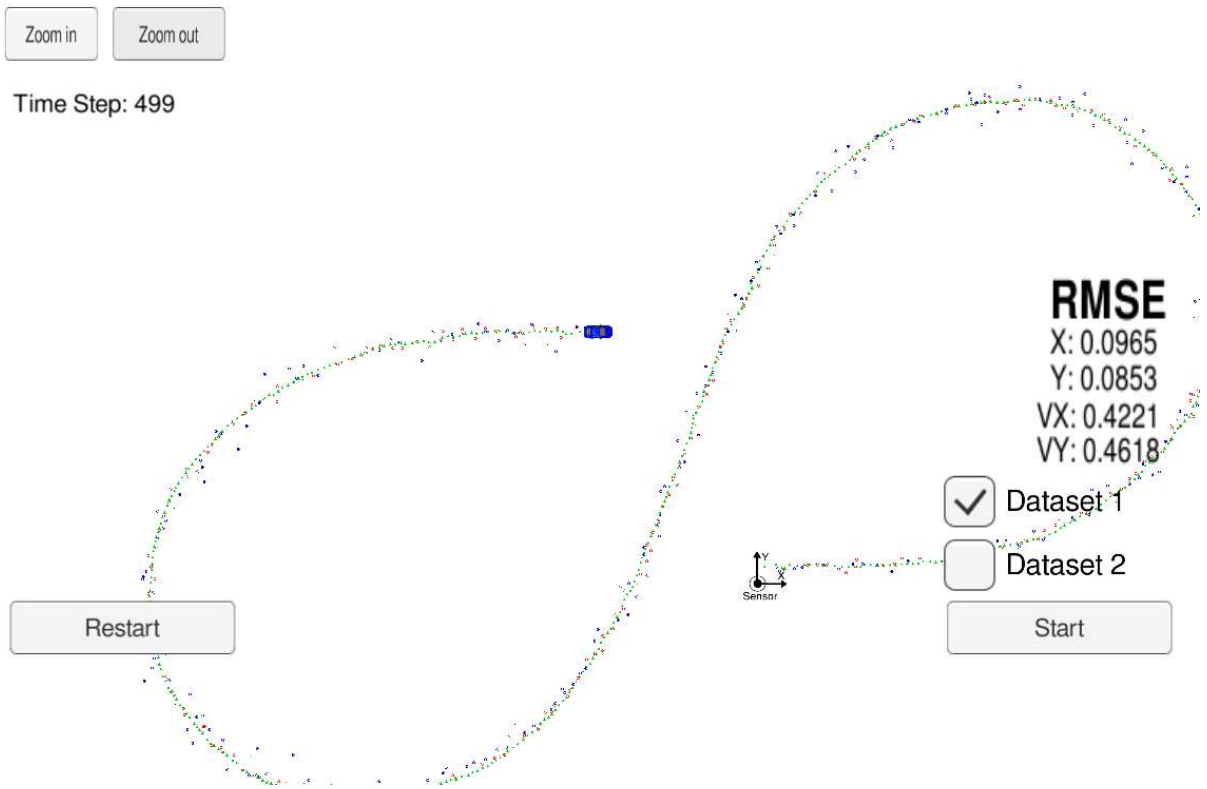
Below are some links on `cmake` and `make` which I think will give you more indepth knowledge. Please, feel free to check them out.

- Cmake FAQS.
- Using make and writing Makefiles
- Youtube set of tutorials on using make and writing Makefile.
- MakeFiles

## Accuracy

**Your algorithm will be run against Dataset 1 in the simulator which is the same as "data/obj_pose-laser-radar-synthetic-input.txt" in the repository. 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 [.11, .11, 0.52, 0.52].**

By running the algorithm against Dataset 1 in the simulator, it produces very low RMSE values that clearly fall within the limits specified, as indicated in the image below. Well done implementing the changes that were specified in the previous review.!👏




## 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.**

Fantastic implementation!!. The algorithm clearly follows the suggested flow of this lesson to successfully build a Kalman Filter for sensor fusion.

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

The algorithm correctly initializes all the state vectors and covariance matrices.
It is interesting to see that you used `is_initialized_` boolean variable to ensure that the first measurement is used to initialize state vector and covariance matrix. Well done!👏

**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.**

Good work! The algorithm correctly predicts the object position to the current timestep and then goes forward to updating the prediction using the new measurement. This is done upon receiving a

forward to updating the prediction using the new measurement. This is done upon receiving a measurement after the first.

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

A step-wise implementation of the algorithm correctly sets up the appropriate matrices given the type of measurement ( radar or laser) and calls the correct measurement function for a given sensor type. Nicely done(update and updateEKF functions).

## 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 overall code structure of the algorithm is good. It does not need to sacrifice comprehension, stability, robustness or security for speed. 👍

# Tip

Check out the suggested `CommonUpdate()` function under code review that does the matrix calculations common to the `Update()` and `UpdateEKF()` functions for lidar and radar measurements. This will avoid repeated calculations and code.

I will like to share some resources on code optimization with you which may be of great interest:

- 10 Tips for C and C++ Performance Improvement Code Optimization).
- C++ Optimization Strategies and Techniques

下载项目

3 代码审阅评注 ❯

返回 PATH