Extended Kalman Filters

审阅

代码审阅 4

HISTORY

Meets Specifications

Congratulations 👏

You did a great job implementing the Extended Kalman Filter algorithm in C++! Your code is structured well, builds without any problems and meets all of the required RMSE values!

I added a few comments:

- Excellent velocity initialization for **RADAR** using `rho_dot` 👍
- `divide by zero` protection is always needed whenever there is a division
- You are properly Normalizing the angle after angle calculation 👏 I added a comment regarding an alternative way for `angle normalization`

For further reading. I am recommending the following resources to know more about Kalman Filter and Non-linear Kalman Filter:
Unscented Kalman Filter
introduction of Kalman Filter
Kalman Tutorial
Kalman Filtering Tutorial

Keep learning and good luck

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.

✅ Code properly compiles with `cmake` and `make`.

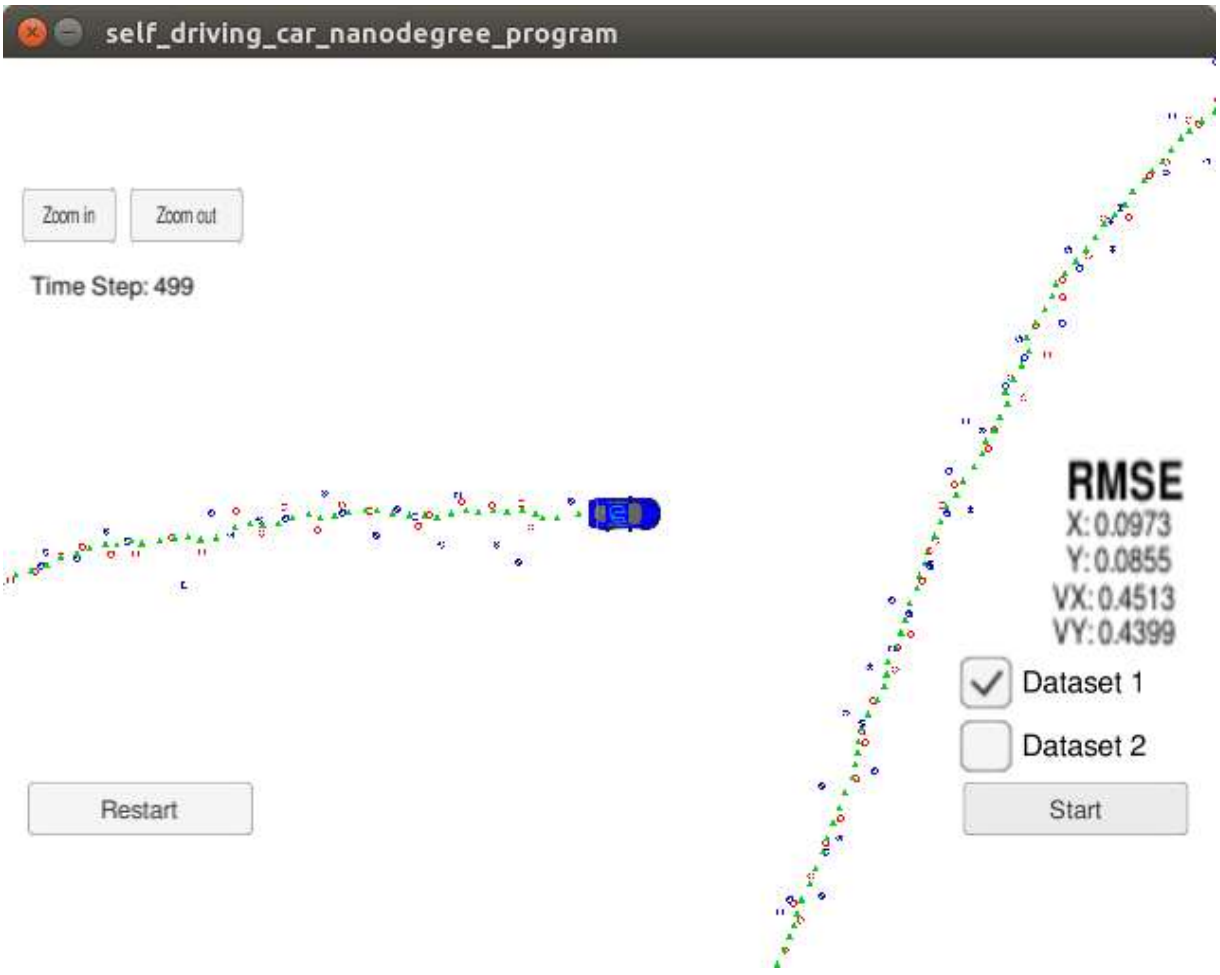
Here are some links to know more about `cmake`:
runningcmake
cmake-guide

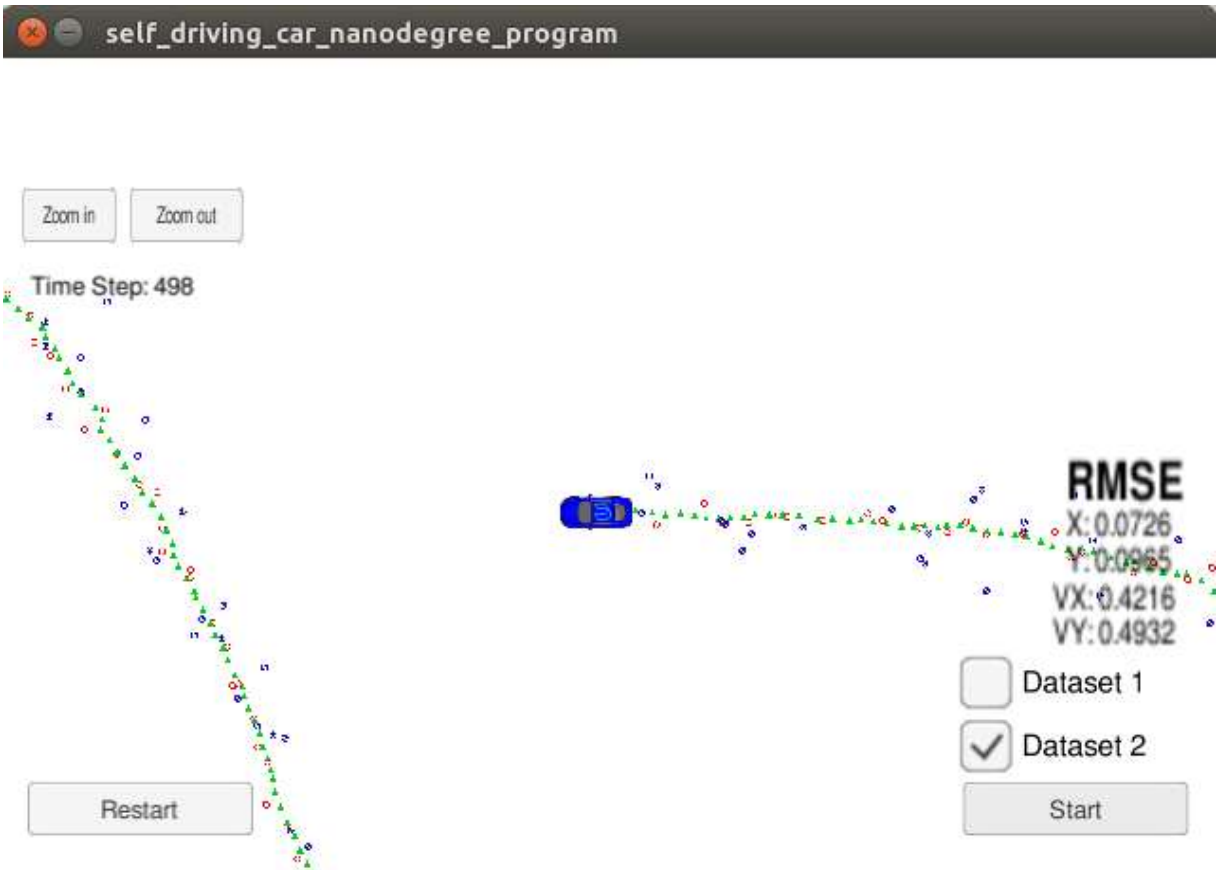
```
hive/gpfvic-CarND-Extended-Kalman-Filter-Project-e59f517/build$ make
Scanning dependencies of target ExtendedKF
[ 20%] Building CXX object CMakeFiles/ExtendedKF.dir/src/main.cpp.o
[ 40%] Building CXX object CMakeFiles/ExtendedKF.dir/src/tools.cpp.o
[ 60%] Building CXX object CMakeFiles/ExtendedKF.dir/src/FusionEKF.cpp.o
[ 80%] Building CXX object CMakeFiles/ExtendedKF.dir/src/kalman_filter.cpp.o
[100%] Linking CXX executable ExtendedKF
[100%] Built target ExtendedKF
```

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].

✅ When running your filter on the included datasets, I get the following RMSE values which are well within the requirements! `MSE <= [.11, .11, 0.52, 0.52]`
Great work!


self_driving_car_nanodegree_program
Zoom in
Zoom out
Time Step: 499
RMSE
X: 0.0973
Y: 0.0855
VX: 0.4513
VY: 0.4399
Dataset 1
Dataset 2
Restart
Start


self_driving_car_nanodegree_program
Zoom in
Zoom out
Time Step: 498
RMSE
X: 0.0726
VX: 0.4216
VY: 0.4932
Dataset 1
Dataset 2
Restart
Start

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.

✅ You correctly implemented the Extended Kalman Filter architecture!

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

✅ You correctly initialize the filter with the first measurement update!
👍 Using `rho_dot` for **RADAR velocity** initialization:

```
ekf_.x_(0) = rho * cos(phi);
ekf_.x_(1) = rho * sin(phi);
ekf_.x_(2) = rho_dot * cos(phi);
ekf_.x_(3) = rho_dot * sin(phi);
```

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 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.

✅ You correctly set up the noise covariance matrices and calculate the Jacobian for the RADAR case!

You are properly taking care of angle normalization 👍 See **code review** for suggestions on different ways angle normalization using: `atan2(sin(), cos()`

🌱 Whenever there is a division, it's always a good idea to add `divide by zero` protection unless you are ensuring that the denominator is not `zero`. See code review for more

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.

下载项目

4 代码审阅评注 ›

返回 PATH

给这次审阅打分

开始