UDACITY

PROJECT

Extended Kalman Filters

A part of the Self Driving Car Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW 6

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Congratulations on completing this challenging C++ project! You should be pleased with what you have accomplished here.

Best of luck with the UKF project and the rest of Term 2!

If you want to learn more about Bayesian and Kalman filters (this material covers KF, EKF, UKF, and particle filter algorithms), this github project is a great resource.

NB: *The nature of this project and rubric means that elements of this review will be similar or identical to the reviews for other students. Please rest assured that I have looked through all of the code you have written and submitted looking for issues with your implementation and places where I could suggest improvements.*

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.

The project compiles using `cmake` and `make` without errors.

Accuracy

✓

For the new data set, 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 [.11, .11, 0.52, 0.52].

For the older version of the project, your algorithm will be run against "sample-laser-radar-measurement-data-1.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 [0.08, 0.08, 0.60, 0.60].

The filter passes the threshold for data-1 (old version):

```
Accuracy - RMSE:
0.0651667
 0.060539
 0.530413
  0.54425
```

Nice one!

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.

Each step of the EKF algorithm is present and accounted for. Your code is well organised and easy to read. Nicely done.

✓

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

The initialisation step is executed for the first measurement and only executed once. Exactly as it should.

✓

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.

The prediction step looks correct and the appropriate update steps are called for each of the two measurement types.

✓

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

All of the matrices are correctly initialised and initialised with appropriate values.

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.**

One of the more resource expensive steps in the EKF is the update step where multiple matrix operations are calculated. This is especially true if the state space is larger than it is for this project. It is important to implement these calculations in an efficient manner and avoid repeat calculations (the first point in the list of inefficiencies above).

For your code:

```
        MatrixXd Ht = H_.transpose();
        MatrixXd S = H_ * P_ * Ht + R_;
        MatrixXd Si = S.inverse();
        MatrixXd PHt = P_ * Ht;
        MatrixXd K = PHt * Si;
```

`P_ * Ht` is executed twice. It would be better to store this in a variable (as you have) and then reference it for these two calculations (`S` and `K`).

This issue has been copied in the `UpdateEKF()` method too. I recommend fixing this in both places, or better yet, refactor your implementation to keep it DRY.

I have made a few suggestions in the code review. I hope you find them helpful and constructive.

To make your code even more efficient, when defining variables that remain unchanged throughout the execution of the code, it is best to declare them using the `const` keyword. This allows the compiler to optimise memory allocation when compiling the code. The first answer from this stack overflow question goes into greater detail.

DOWNLOAD PROJECT

6 CODE REVIEW COMMENTS ❯

RETURN TO PATH

Student FAQ