PROJECT

Extended Kalman Filters

A part of the Self Driving Car Engineer Nanodegree Program

PROJECT REVIEW | CODE REVIEW | NOTES

Requires Changes

SHARE YOUR ACCOMPLISHMENT

1 SPECIFICATION REQUIRES CHANGES

Dear Student,

You put a lot of effort into this project and it shows off. Your algorithm needs a few changes.
Please check my suggestions for that.
I believe you will certainly pass the rubric next time.
I wish you all the success for your next submission. 😃

Few Articles

State Estimation with Kalman Filter
Credits: Malintha Fernando
How a Kalman filter works
Credits: Arun
Kalman filter: Intuition and discrete case derivation
Credits: Vivek yadav

Suggestion

- To expand ones knowledge in this area, it might be good in ones extra time to look at the following materials:
  - Kalman filter from The Extended Kalman Filter: An Interactive Tutorial for Non-Experts.
  - Experimental Comparison of Sensor Fusion
  - Particle/Kalman Filter for Efficient Robot Localization
  - Some expert approaches can be sort here Adaptive approaches to nonlinear state estimation for mobile robot localization: An experimental comparison
  - Great content for Extended Kalman Filter and Sigma Point Filter Approaches to Adaptive Filtering
  - A Robust Kalman Framework with Resampling and Optimal Smoothing

Credits: My Fellow Reviewer.

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.

Excellent. The code compiles and links without any errors with `cmake` and `make`.
You have done a good job here.

Suggestion

- Some C++ debugging tips to consider as a developer, see
  - setting CMAKE_BUILD_TYPE to release

Rate this review

    - setting CMAKE_BUILD_TYPE to debug
    - In case of any failure or difficulty, most commonly Understanding why CMAKE_BUILD_TYPE cannot be set
- When cmake includes symbol tables in the generated file, GDB now can come in to assist in debugging critical faults like the famous segmentation fault, why some variables are behaving weired, etc, see
    - How to Debug Using GDB
    - GNU GDB Debugger Command Cheat Sheet
    - Debugging with GDB By Alexandra Hoffer

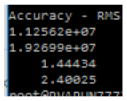
Credits: My fellow reviewer

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].

Your algorithm was run against "sample-laser-radar-measurement-data-1.txt". The positions of your algorithm outputs were collected and was compared with the 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]. Your values are:



Suggestion

- Check that noise models (i.e. R for each sensor type, noise_ax, noise_ay, and Q) are correctly modeling the respective sensor types - refer back to the Udacity lectures for the correct models to use.
- Check the initialization values of important matrices are appropriate and correct (i.e. F, H and P) - look for typos, and outright errors.

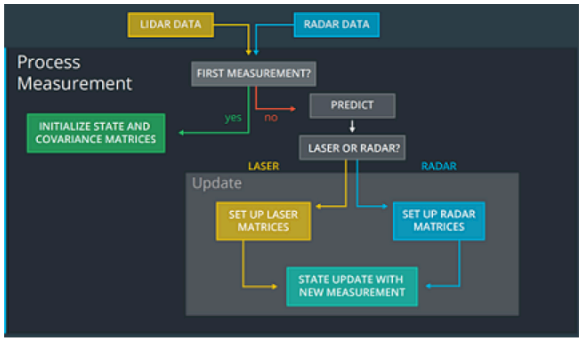
Follows the Correct Algorithm

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.

The filter implements the basic steps in the correct order.

- initialization
- loop until done
    - prediction of new state (and state estimate error covariance) given elapsed time
    - update using new measurements data

For in detail steps please check here



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

The state and co-variance matrices are correctly initialized upon receiving the first measurements from each sensor.

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 steps are each implemented properly for both LASER and RADAR sensors. The time difference since the prior measurement is calculated and the necessary updates are made to both the state transition matrix F and the process noise co-variance matrix Q. New predictions of the state vector (x) and state estimate error co-variance matrix (P) are correctly made.

This one is a bit obscure, but if `dt` is very small (i.e. two measurements are approximately coincident in time) we should not make a prediction as it can led to arithmetic problems. A solution is:

```
if ( dt > 0.001 )
{
    ekf_.Predict();
}
```

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

- When a measurement is available the correct matrices are updated based upon the measurement type, and the correct measurement update functions are called for each sensor type.
- The update steps are each implemented properly for both LASER and RADAR sensors.
- The correct H and R matrices are used based upon the measurement sensor type.
- For radar measurements polar coordinates are successfully translated to cartesian coordinates and the Jacobian measurement prediction matrix (Hj) is correctly updated.
- The error between the measurement and the current prediction is used to correctly update the state vector (x) and the state estimate error covariance matrix (P).

Together with the prediction step, these two steps the hardest part of implementing a successful Extended Kalman Filter and you have accomplished them with ease. Excellent 💯

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.

Further Improvement

- This work is good in the light of this rubric. There is always room for improvement. Here are some suggested materials to further improve this section
  - C++ Optimization Strategies and Techniques
  - A discussion on C++ Optimization Techniques
  - A full document with details on Tips for Optimizing C/C++ Code
  - 10 Tips for C and C++ Performance Improvement
  - Optimizing C++ A book about improving program performance
  - Optimizing C++/Writing efficient code/Performance improving features
  - Optimizing C and C++ Code

    Credits: My fellow reviewer

RESUBMIT PROJECT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Student FAQ