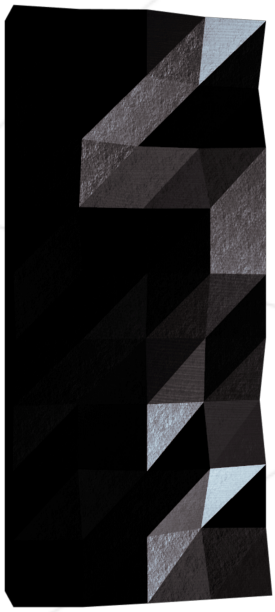


ft_kalman

Some more tools to find charly

Tiago Lernould (tlernoul),
Eliott Suits (esuits),
Damien Cuvillier (dacuvill)

Summary: For this project, you will need to create a kalman filter to retrace a trajectory with incomplete and flawed information

Table des matières

Chapitre I

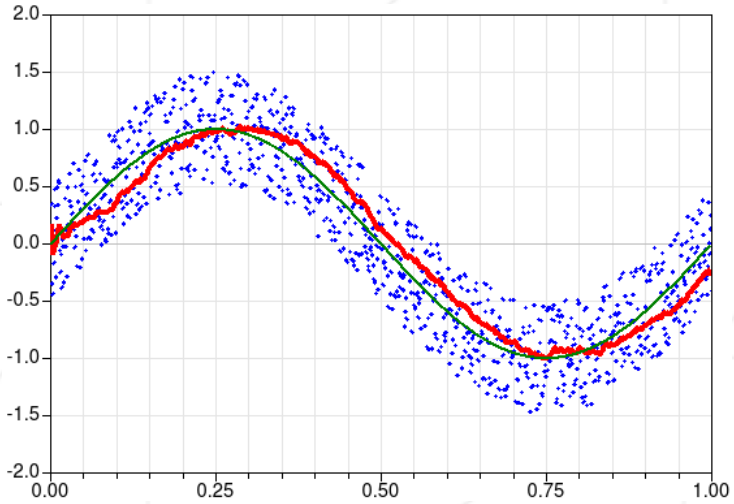
Foreword

The missile knows where it is at all times.
It knows this because it knows where it isn't.
By subtracting where it is from where it isn't, or where it isn't from where it is (whichever is greater), it obtains a difference, or deviation.
The guidance subsystem uses deviations to generate corrective commands to drive the missile from a position where it is to a position where it isn't, and arriving at a position where it wasn't, it now is.
Consequently, the position where it is, is now the position that it wasn't, and it follows that the position that it was, is now the position that it isn't.
In the event that the position that it is in is not the position that it wasn't, the system has acquired a variation, the variation being the difference between where the missile is, and where it wasn't.
If variation is considered to be a significant factor, it too may be corrected by the GEA. However, the missile must also know where it was. The missile guidance computer scenario works as follows.
Because a variation has modified some of the information the missile has obtained, it is not sure just where it is. However, it is sure where it isn't, within reason, and it knows where it was.
It now subtracts where it should be from where it wasn't, or vice-versa, and by differentiating this from the algebraic sum of where it shouldn't be, and where it was, it is able to obtain the deviation and its variation, which is called error.

Chapitre II

Introduction

The goal of this project is to create a Kalman Filter, which is an algorithm used in navigation (in the air, on the ground, in space, wherever really), economics, and even CV !
You will be introduced to concepts such as co-variance matrices, signal theory, and some things in between...
It is highly recommended to have completed matrix before doing this project.



You are the Inertial Measurement Unit of a generic vehicle (not a car, not a plane, not a boat) moving in a generic environment (with no air, no gravity).
The only information you have are the following :

- Your true initial position (X,Y,Z),
- Your true initial velocity (in km/h),
- Your current inertia (in m/s^2, every 1/100th of a second),
- Your current direction (in euler angles),
- Your current GPS position every 3 seconds (X,Y,Z),

But since we live in an imperfect world, some of those measurements are affected by gaussian white noise :

- Accelerometer : $\sigma = 10^{-3}, \upsilon = 0$
- Gyroscope : $\sigma = 10^{-2}, \upsilon = 0$
- GPS : $\sigma = 10^{-1}, \upsilon = 0$

Chapitre III

Goal

Included in the project files is a CLI program that, when launched, creates a listener on a UDP port which will be your main communication channel.

After connecting to that listener, you should send a handshake which will be the signal that starts the trajectory generation.
Once this is done, you will receive your first input which will give you the necessary input to initialize your Kalman filter, after which you should send your first estimation.

Thus should begin a dialogue between your filter and the sensor-stream, which will wait for your position estimation before sending you updated data until the run's end. Obviously, your position estimation must be close to the real coordinates. If your estimation is further than 5 meters of the real position, or if your filter take more than 1 second to answer, then the sensor-stream will return an error and stop the communication.

As of the rest of the information, you'll have to figure out ! (-h)

```
./imu-sensor-stream -s 42 -d 42 -p 4242
```

Chapitre IV

General Instructions

- You Must implement a Kalman Filter, no machine learning, no other algorithm
- Your code Must be in C, C++ (standard C++17 or less) or Rust
- Your code Must be able to compile statically
- If the language requires it, you must submit a Makefile which will compile your source files to the required output with the flags -Wall, -Wextra and -Werror, and your Makefile must not relink.
- You obviously cannot use any library that does the filtering for you, mathematical libraries are otherwise allowed
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, panic, etc) apart from undefined behaviors.
- All heap allocated memory space must be properly freed when necessary. No leaks will be tolerated.

Chapitre V

Mandatory Part

Your filter must always work with trajectories of durations up to 90 minutes with default noise amount.

The following points only apply to trajectories shorter or equal to 90 minutes. Beyond that duration all bets are off.

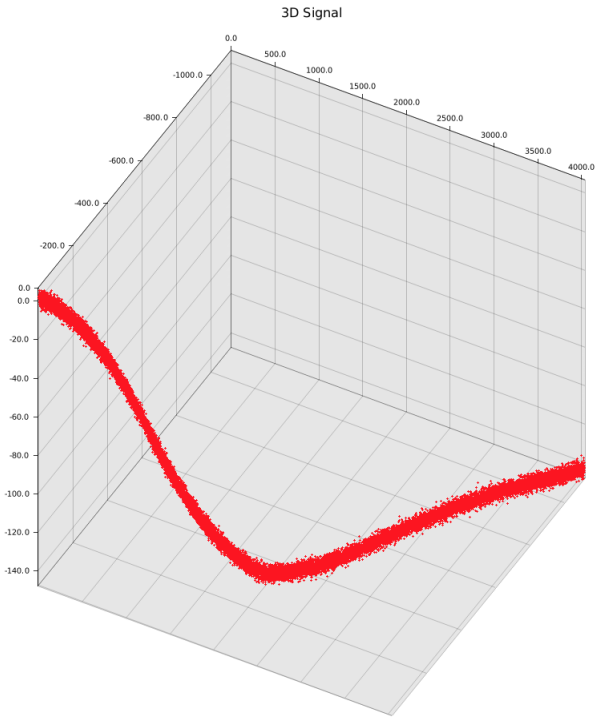
Your program should be decently optimized and thus should never timeout, and if it should timeout it should be handled gracefully (no crash, segfault, leaks will be tolerated).

Likewise, your program should be precise enough that it should never create an estimation that is further than allowed, and if it does it should also be handled gracefully.

Chapitre VI

Optional Part

- A trajectory visualizer (from a basic 2d plot to a 3d visualizer with HUD and variance display).
- A very fast filter which compute meanly in milliseconds (see the "meanspeed" argument to estimate this).
- A really precise filtering which can handle more noise than default amount (check the "noise" argument to do this).
- A new functionality we didn't think of (aesthetic-only features doesn't count).



Perhaps this could be a decent 3d plot visual.

Or not...Who knows.

Chapitre VII

Submission and peer evaluation

We know this is a highly technical project with a lot of complexity, which is all the more reason why you should really understand why and how the Kalman Filter works, and because this aspect will be extremely important during your peer evaluation.

Submit your work on your GiT repository as usual. Only the work on your repository will be graded.