Estimating position and velocity using an accelerometer and a GPS

4B25 - Coursework 6 - By Eivind Roson Eide

Problem Description and importance:

I made a tracking device to estimate the position and velocity accurately by fusing noisy sensor data from the accelerometer and an external GPS. The output should be displayed on a small screen such that the application can be moved around. Devices like these can be used to help positioning of autonomous systems and can be used to extract useful metrics in wearables for sport applications.

Current state of the art:

Position estimation is widely used in many of the above mentioned applications. Linear Filters, like Kalman Filters is the optimal approach as long as the assumptions about the distribution of the noise is correct and the prediction and update steps are linear transformations of the state and the sensor readings respectively [1]. However, as most systems in the world are not linear, other methods like the Extended Kalman Filter have been developed to deal with this. They work by linearizing the system to the first or second degree around your current state. However, they are no longer optimal, and can often show problems of divergence if the system is not modelled correctly. Other more sophisticated method have been developed, like the particle filter [2], that is a form of genetic algorithm. Instead of the typical predict-update cycle, it uses a mutate-select cycle. This has shown to be more robust when the systems are unstable, the sensor data is too big or the non-linearities are not smooth (such that they can be approximated with taylor series). Particle filters are therefore regarded as the state of the art in the more complex cases.

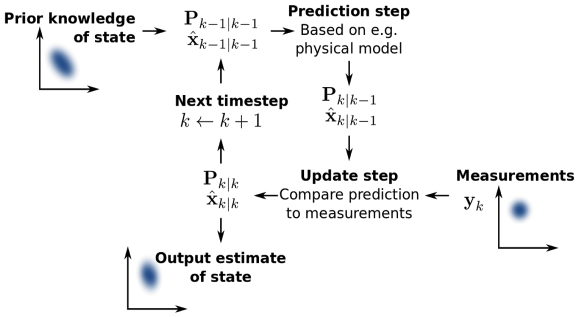


Fig 1: The kalman predict-update cycle

Deriving the kalman filter:

I chose to approach the problem with a simple linear kalman filter as linear motion can be modelled exactly with this approach. However, rotation is not linear, this caused problems as we cannot estimate the 3-axis acceleration without knowing the orientation. I solved this by assuming the acceleration is mostly in the z-direction (up-down), thereby allowing me to estimate the vertical acceleration as the difference between the magnitude of the measured acceleration and the gravitational constant, and ignoring the acceleration in the other directions. Another big problem with acceleration data is that it drifts quite significantly, due to integration errors. To mitigate this, I introduced a decay rate that assumes that the acceleration in one time instant is largely uncorrelated with previous time steps. Lastly, I computed the gravitational acceleration by averaging a set of acceleration measurements at startup.

The second set of inputs is GPS position data. I converted this to a local coordinate system by setting the first reading as the origin. I could also extract ground velocity from the data, and use this to improve the estimate of the positioning in the x and y direction.

Explanation of the software:

The software is organized into various modules.

- **Kalman Filter:** Implements a Kalman Filter using the Linear Algebra library Eigen and a convenient API to pass data to and from the kalman filter.
- **Accelerometer:** Implements a API for the Freescale library for communicating with the accelerometer. The board communicates with the accelerometer over I2C.
- **Display:** Contains a low level driver for communicating over SPI with the display. It also implements a hardware specific interface for the Adafruit GFX library.
- **Positioning:** Contains a low level driver for the GPS module. The board communicates over UART, and buffers the data in software using a interrupt call when it receives data. It also implements the Adafruit_GPS NMEA message parser. It also has an API for interfacing accessing the GPS data.

Results:

Simulating typical input with the accelerometer updating 10 times more often with realistic noise models show good performance of the tracker. In figures 2 and 3 we can see that the estimator represented with the green line is able to track the actual path in the blue line on simulated data. We can also notice how it can accurately use the accelerometer data to estimate the velocity and foresee the change of direction before the GPS updates. The estimator quickly gets rid of initial uncertainty and stabilizes at approximately the uncertainty of our sensors. As seen in figure 5 however, the noise in the readings from the accelerometer is quite large when it is being rotated. Using the time series I computed that the standard deviation to be approximately 0.4 m/s^2. However, despite the significant error, this is much less than trying to maintain the accelerometer angled correctly.

Fig 2: Simulated position estimate

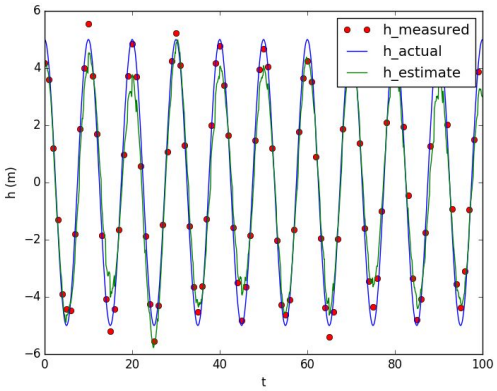
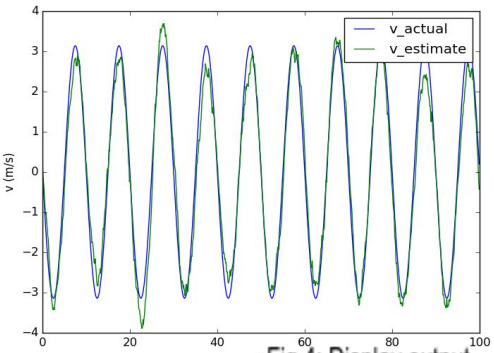


Fig 3: Simulated velocity estimate



Another important criteria is the speed of computation. Testing on the real device showed that the required time for computing the predict and update loop is below the 10ms update rate for the accelerometer, which is sufficient. However, writing on the screen takes more time than 10 ms, hence slowing down the predict step, and there by missing out on accelerometer data that could be useful. To reduce the effect of this it only updates the screen at 1Hz. One of the major constraints was memory space as we only have 128k byte of flash memory. Increasing the level of optimizations I achieved to compress the code to 111k byte, just small enough to fit on the device.

Fig 4: Display output

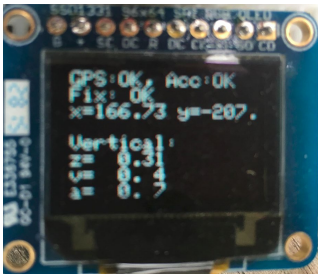
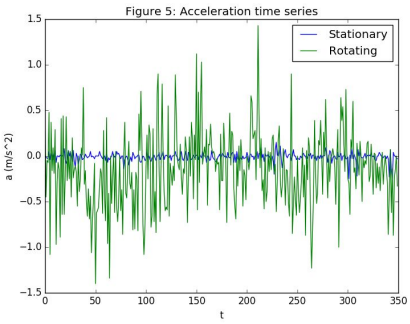


Figure 5: Acceleration time series



Lastly, we want to convey the information effectively to the user. In figure 4 we can see the screen output in operation. It includes information about the status of the system, as well as our estimate for vertical position, velocity and acceleration as well as horizontal position.

References:

[1] Faragher, R., 2012. Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]. *IEEE Signal processing magazine*, *29*(5), pp.128-132.

[2] Van Der Merwe, R., Doucet, A., De Freitas, N. and Wan, E.A., 2001. The unscented particle filter. In *Advances in neural information processing systems* (pp. 584-590).

Code repository:

https://github.com/ereide/position_estimator-FRDM-KL25Z4