Intro

Contents

This report looks at Kalman filters and how they can be used with the aim to improve undergraduate experiments. Kalman filters are a optimal estimation algorithm used to predict the true state (a description of the system) from a set of noisy measurements. The Kalman filter uses the current state along with a model, that reflects the physics of the system, to predict the next state. This is then compared with measurements to improve on this prediction. The model in the Kalman filter is dependent on how the system behaves, whereas most filters use the same algorithm with a couple of parameters that are tuned. Often the Kalman filter will use measurements from multiple types of sensors to improve accuracy, sensor fusion.

Experiment I uses a 6 axis IMU to track position and attitude. The IMU measures acceleration, in the x, y and z directions and angular velocity in the yaw, pitch and roll directions. The position and attitude estimates are inaccurate due to drift associated with integration. Using Kalman filters to fuse sensor data between the accelerometer and gyroscope as well as non inertial sensors like magnetometer and GPS would greatly improve the results of this experiment.

Contents

1 Basics Of Filtering

Before looking at how Kalman filters worked basic recursive filters were tested. Later the performance between these and the Kalman filter will be compared. The purpose of a filter is to consider measurments and use these to form an estimate of the true state $\mathbf{x}_k$ which is a column vector describing the system, in these example the state is 1D. Measurements of the true state, z_k are corrupted by noise. The filters output an estimate of the true state $\hat{x}_k$.

1.1 Average filters

For a signal where x_k is constant, e.g. calculating the output from a battery, computing the mean is a reasonable way to determine $\hat{x}_k$. The average is computed as:

$$\hat{x}_k = \frac{z_1 + z_2 + z_3 + \ldots + z_k}{k} \tag{1}$$

In this case $\hat{x}_k$ is the average. However $\hat{x}_k$ needs to be computed at every time-step, (1) isn't going to be very efficient to compute. Written in a more computationally efficient form:

$$\hat{x}_k = \left(\frac{k-1}{k}\right)\hat{x}_{k-1} + \frac{z_k}{k} \tag{2}$$

Let $\alpha = \frac{k-1}{k}$, (2) can be rewritten as:

$$\hat{x}_k = \alpha\hat{x}_{k-1} + (1-\alpha)z_k \tag{3}$$

This will be referred to as the recursive average filter. The code for this filter can be found in github. The programme is made up from 3 files: `GetVolt.py`, `RcsAvgFilter.py` and `Test.py`. The first file generates a noisy signal, the second contains the filter and the third runs the filter on the generated signal.

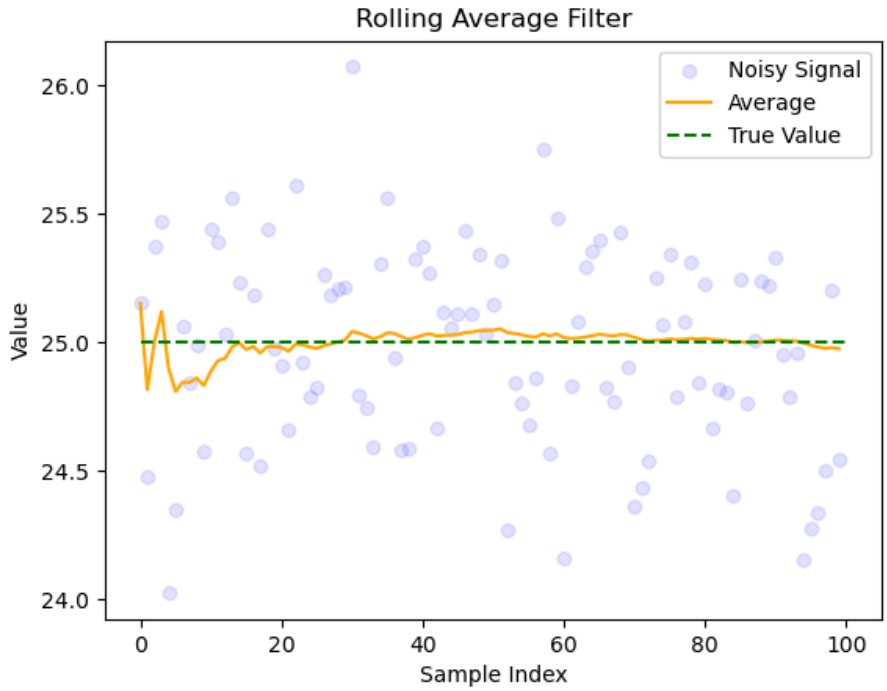

Fig. 1 A noisy signal with mean 25 and standard deviation 0.5, fitted with a moving average filter.

The filter is efficient and converges to the correct value very quickly. Due to random variation the filter fluctuates around the correct value. The average filter is very good when $\hat{x}_k$ is constant, however it won't converge to a changing signal.

1.2 Moving Average filters

The moving average is used to remove noise over a constantly varying signal. Here $\hat{x}_k$ represents the moving average at time t_k and n represents the window size which is a parameter to be tuned. The moving average can be written as:

$$\hat{x}_k = \frac{z_{k-n+1} + z_{k-n+2} + \ldots + z_k}{n} \tag{4}$$

$\hat{x}_k$ can be written recursively using the previous estimate $\hat{x}_{k-1}$ and the new measurement z_k:

$$\hat{x}_k = \hat{x}_{k-1} + \frac{z_n - z_{k-n}}{n} \tag{5}$$

for more efficient computation. The code for the following graphs can be found in github. The programme is made up from 3 files: `GenTestSig.py`, `MovAvgFilter.py` and `Main.py`. The first file generates a signal (the true signal) which is a combination of 3 $\sin$ waves and then adds random gaussian noise to this signal, the noisy signal. The second contains the filter algorithm which will be used to fit the noisy signal and the third runs the filter on the generated signal.

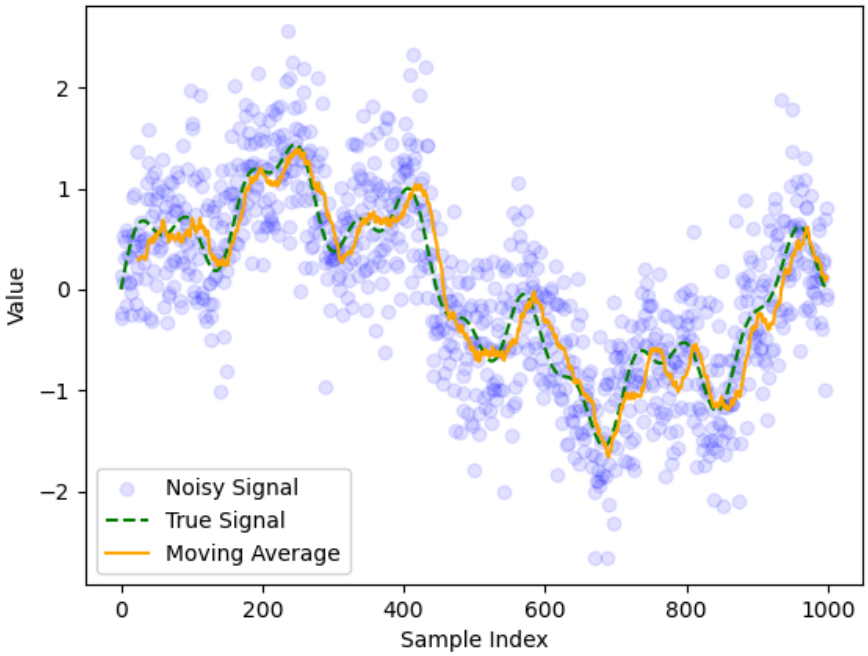

Fig. 2 Moving average filter applied to a noisy signal with $n = 25$.

The moving average lags behind the true signal but has roughly the right shape. This is expected as the moving average relies on previous estimates.

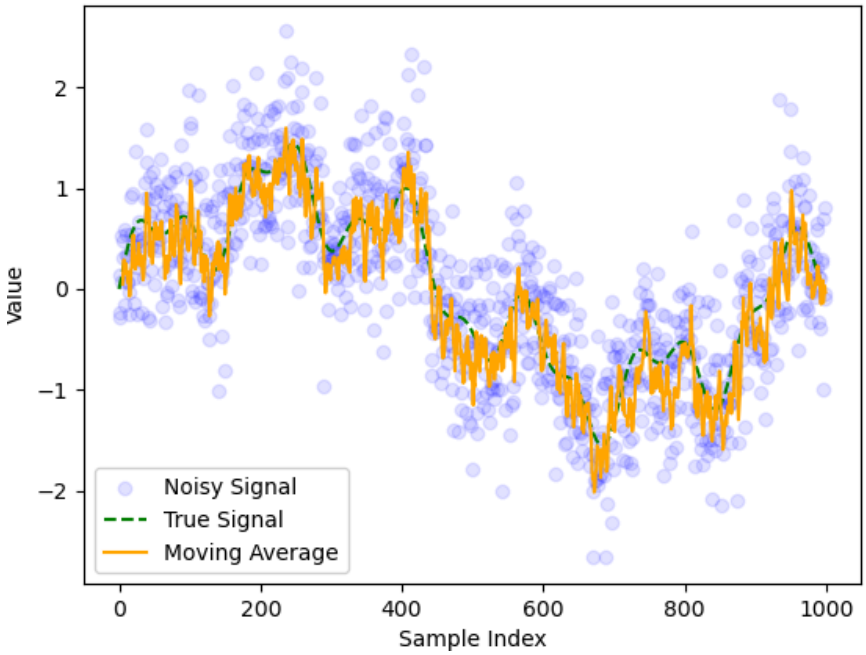

Fig. 3 Moving average filter applied to a noisy signal with $n = 5$.

In this example the delay is smaller but less noise is removed. There is a tradeoff between noise reduction and minimizing delay limiting the accuracy of the moving average filter.

1.3 Exponential Moving Average filter (Low Pass)

All terms in (4) have equal weighting ($1/n$), however it makes more sense to give more recent terms a larger weighting, this should help minimize delay whilst preserving the smoothing effect. A result of this is it allows low frequencies to pass through but filters out high frequencies. Noise is usually high frequency. Below is an example of a first order exponential moving average, low pass filter (EMALPF):

$$\hat{x}_k = \alpha\hat{x}_{k-1} + (1-\alpha)z_k \quad 0 < \alpha < 1 \tag{6}$$

Looks similar to (3) except here α is a parameter to be tuned. It is also true that:

$$\hat{x}_{k-1} = \alpha\hat{x}_{k-2} + (1-\alpha)z_{k-1} \quad 0 < \alpha < 1 \tag{7}$$

Combining these equations helps to overcome some problems associated with the moving average since more distant terms disappear exponentially quickly:

$$\hat{x}_k = \alpha^2\hat{x}_{k-2} + \alpha(1-\alpha)z_{k-1} + (1-\alpha)z_k \quad 0 < \alpha < 1 \tag{8}$$

Due to the restriction on alpha larger n means greater weighting on $\hat{x}_n$ since $\alpha(1-\alpha) \leq 1-\alpha$. Previous data gets weighted exponentially less.

The code for this section can be found in github. Similarly to the other two examples the code is made up of 3 files: `GenTestSig.py`, `LowPassFilter.py` and `Main.py`. The first file generates a signal (the true signal) which is

again a combination of 3 $\sin$ waves and then adds random gaussian noise to this signal, the noisy signal. The second contains the filter algorithm which will be used to fit the noisy signal and the third runs the filter on the generated signal.

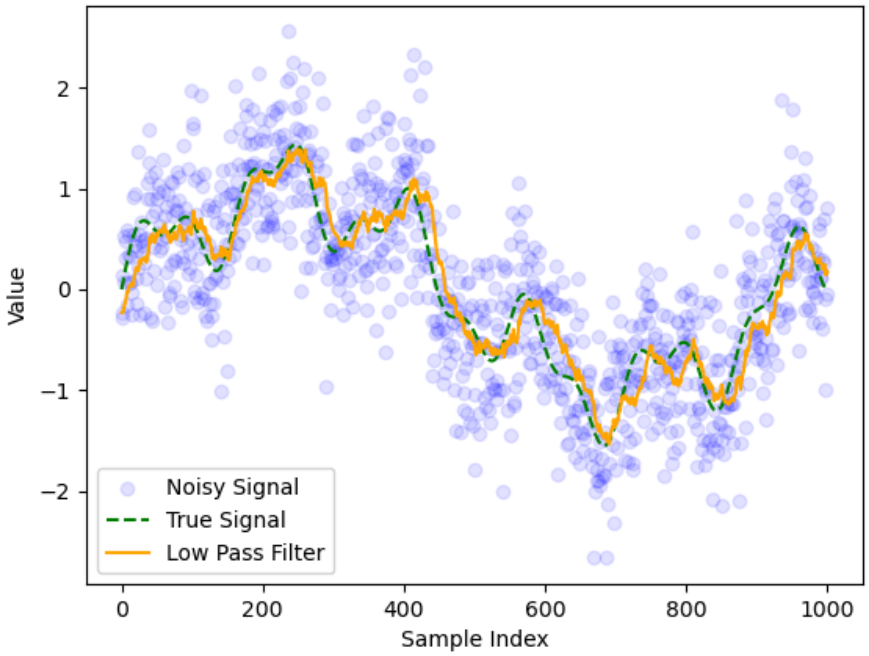

Fig. 4 Figure 3.1: EMALPF with optimized $\alpha = 0.93$ (visually).

The low pass filter has a smaller delay compared to the moving average filter, but was nosier. This makes sense as the moving average filter gives equal weight to all previous estimates, so noisy measurements will have a smaller effect on the fit.

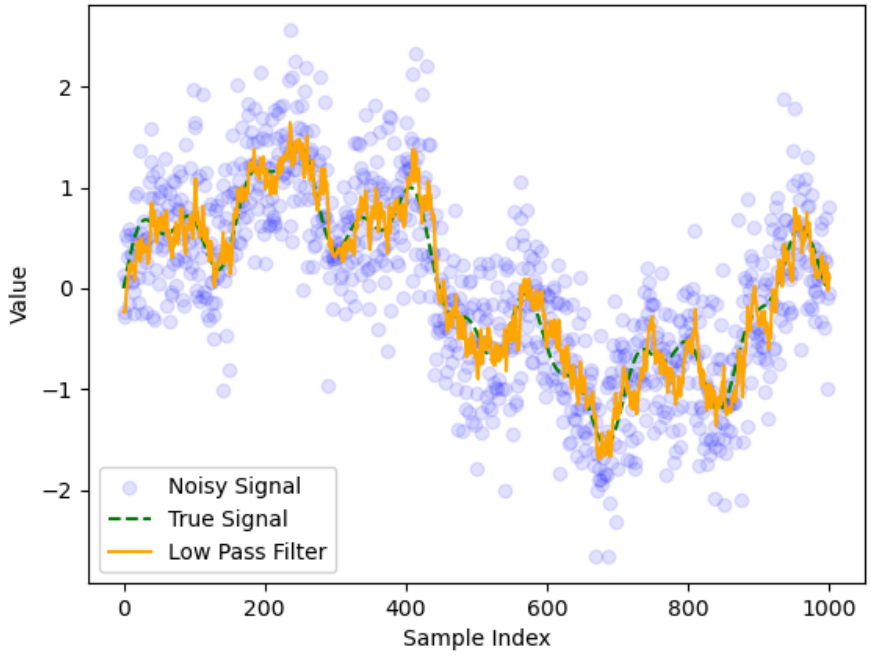

Fig. 5 Figure 3.2: Low pass filter with $\alpha = 0.8$.

Here the delay is less significant but the filter doesn't remove as much noise compared to Fig. 4, since the weighting on previous estimates is smaller.

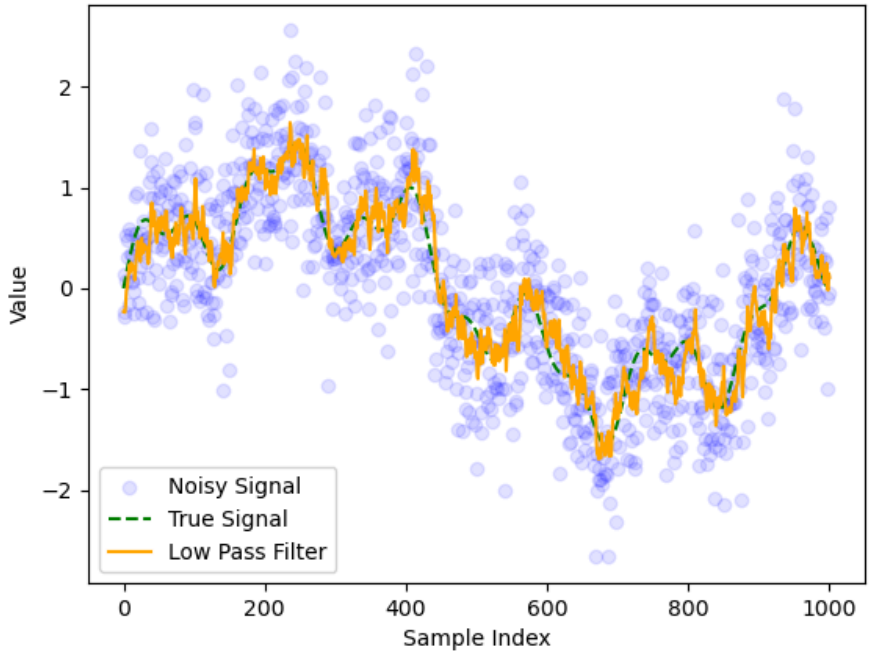

Fig. 6 Figure 3.3: Low pass filter with $\alpha = 0.97$.

Here the delay is more significant since previous results are given larger weightings. The choice of α represents a tradeoff between a noisy signal and a delayed signal, which means its difficult to find optimal α.

1.4 Exponential Moving Average Filter (High Pass)

An exponential moving average high pass filter (EMAHPF) works by subtracting the EMALPF $\hat{x}_k^{LP}$ from z_k. This corresponds to removing the low frequencies from the signal. The high pass filter estimates the state $\hat{x}_k^{HP}$ as:

$$\hat{x}_k^{HP} = z_k - \hat{x}_k^{LP} \tag{9}$$

Subbing in (6) gives:

$$\hat{x}_k^{HP} = \alpha(\hat{x}_{k-1}^{LP} + z_k) \tag{10}$$

Then subbing (9) rearranged into (10) gives:

$$\hat{x}_k^{HP} = \hat{x}_{k-1}^{HP} + z_k - z_{k-1} \tag{11}$$

Which is the recursive formula for the EMAHPF. These are useful for removing low frequency background noise, such as integration drift.

1.5 Summary

The above are examples of passive filters as they filter data without using feedback from a physical model. Kalman filters are active filters which compare measurements with feedback to determine an improved estimate of the state.

2 Kalman filters

A Kalman filter compares the predicted state, $\hat{x}_k^-$, with measurements, z_k, to create an estimate of the true state, $\hat{x}_k$. This section introduces the Kalman filter, its algorithm and why its useful. The state is a vector that contains the variables required to describe the system at a specific time.

> **Important**
>
> Predictions and estimates are not interchangeable when talking about Kalman filters. A prediction is a forecast of the next state based on the previous state and the mathematical model. Whereas a estimate is an update of the predicted state once new measurements have been taken.

> **Note**
>
> In all future sections scalars will be written as plain text a and vectors will be written in bold $\mathbf{a}$. Here the symbols are context free so so are all in plain text.

2.1 Dictionary

The Kalman filter deals with the linear state model. Where the state evolves linearly according to A:

$$x_{k+1} = Ax_k + w_k \tag{12}$$

As long as no forcing function is present. The measurement can be described as:

$$z_k = Hx_k + v_k \tag{13}$$

Where w_k and v_k are both distributed normally with mean 0.

- x_k is the actual (but unknown) state vector at time t_k e.g. the true position and velocity of a car. x_k. $n \times 1$ **column vector**.
- z_k is the measurement of the state from the sensor at time t_k, but contains noise and errors e.g. the GPS position of the car. $m \times 1$ **column vector**.
- A is the state transition matrix. A represents the "rules of motion" of the system. It describes how $\hat{x}_k$ evolves from one time step to the next, assuming there is no external factors changing it (a forcing function). For example if you knew a cars velocity at time t_k you could use A to predict where it would be at time t_{k+1}. $n \times n$ **matrix**.

A forcin
that **del**
system
car rolli
determi
breaks
function

> **Note**
>
> A isn't always a constant matrix, it can be dependent on time, or measurements from other sensors.

- $\hat{x}_k$ and $\hat{x}_k^-$ represent the best estimate and prediction of x_k respectively. $\hat{x}_k^-$ is a prediction of the next state based on the physics of the system which is modeled using A. $\hat{x}_k$ is a updated estimate which blends the model based prediction $\hat{x}_k^-$ and the measurement z_k. $\hat{x}_k$ is the end output from the Kalman filter.
- H is the state to measurement matrix. H is the translator between the system's state and what can be measured. It explains how, if there were no noise or errors, the true state would appear in the sensor. E.g. if the model uses position to predict velocity but the sensor only measures position how H only picks out position. $m \times n$ **matrix**.
- $\hat{z}_k^-$ and $\hat{z}_k$ is the what the model predicts and estimates the measurements should be respectively:

$$\begin{aligned} \hat{z}_k &= H\hat{x}_k \\ \hat{z}_k^- &= H\hat{x}_k^- \end{aligned} \tag{14}$$

(14) is useful for determining H.

- w_k is the linear process noise vector or noise associated with the prediction. w_k is white sequence noise (random noise uncorrelated with time), which makes the models prediction imperfect. E.g. unexpected bumps in the road for a moving car. $n \times 1$ **column vector**.
- v_k is the linear measurement noise vector, noise associated that corrupts z_k. Even if the true state is fixed, the measurements can change due to sensor imperfections. E.g. temperature measurements using a thermometer will be slightly different each time you measure. $m \times 1$ **column vector**.

- Q is the covariance matrix of w_k i.e. how much the true state is expected to deviate from the predictions made by the state transition model. Large Q assumes the measurements are more reliable than the model and puts a larger weighting on z_k compared to $\hat{x}_k^-$ when computing $\hat{x}_{k+1}$. Smaller Q puts more trust in the model $\hat{x}_k^-$ compared to z_k. $n \times n$ **matrix**.

$$Q_k = E[w_k w_k^T] \tag{15}$$

[BH12] (chapter 4)

- R is the covariance matrix of v_k. i.e. how much the measured state is effected by noise in the state transition model. R tells the Kalman filter how much to "trust" the measurements compared to model predictions. Large R suggests the model is more reliable than the measurements and puts more emphasis on $\hat{x}_k^-$ compared to z_k when computing $\hat{x}_{k+1}$, small R puts more emphasis on z_k compared to $\hat{x}_k^-$. $m \times m$ **matrix**.

$$R_k = E[v_k v_k^T] \tag{16}$$

[BH12] (chapter 4)

- e_k and e_k^- are the error in the estimation and the error in the prediction respectively defined as $e_k = x_k - \hat{x}_k$ and $e_k^- = x_k - \hat{x}_k^-$. $n \times 1$ **column vector.**
- P_k and P_k^- are the associated error covariance matrices for e_k and e_k^- respectively defined by: $n \times n$ matrix.

$$P_k = E[e_k e_k^T] \tag{17}$$

and

$$P_k^- = E[e_k^-(e_k^-)^T] \tag{18}$$

[BH12] (chapter 4)

- K_k is the Kalman gain which has similar effects to α in the low pass filter. $n \times m$ **matrix**. It is a blending factor that determines how much of the prediction and measurement goes into the updated estimate $\hat{x}_k$. The Kalman gain is determined by the covariance matrices P_k^-, H, R and Q.
- x_0 is the initial state estimate. provided at the start of the estimation process.
- P_0 is the initial error covariance matrix estimate.

> **Important**
>
> The Kalman filter parameters are the parameters set by the user before fitting the data they are: A, Q, H, R, $\hat{x}_0$ and P_0.

2.2 Estimation step

The prediction $\hat{x}_k^-$ and measurement are combined using the blending factor K_k (yet to be determined) below to determine the updated estimate $\hat{x}_k$.

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{19}$$

this is known as the **update step** K_k determines how much of each z_k and $\hat{x}_k^-$ goes into $\hat{x}_k$ i.e. how much the measurement is trusted compared to the prediction.

> **Aside: Connection between Kalman filter and low pass filter**
>
> Equation (19) can be rewritten as:
>
> $$\hat{x}_k = (I + K_k H)\hat{x}_k^- + K_k z_k \tag{20}$$
>
> Letting $H = I$ and $\alpha = 1 - K_k$ a first order low pass filter is recovered similar to (7).
>
> $$\hat{x}_k = \alpha\hat{x}_k^- + (1-\alpha)z_k \tag{21}$$
>
> A low pass filter smooths out noisy data by blending the previous estimate with new data using a fixed weighting factor α. The Kalman filter automatically adjusts its version of α essentially K_k to decide how to weight the previous estimate and the new data for each time step depending on how much uncertainty there is in the prediction and measurement.

The mean square error (MSE) between $\hat{x}_k$ and x_k is the performance criterion for the Kalman filter. The MSE is related to the terms in the leading diagonal of P_k. Here an expression for K_k is derived that minimizes the MSE. An expression from the definition of P_k from (17) in terms of our Kalman parameters is required. The problem is (17) contains e_k which can't be computed since x_k is unknown.

First an expression for $\hat{x}_k$ was determined by subbing (13) into (19), to rewrite the estimation update equation:

$$\hat{x}_k = \hat{x}_k^- + K_k(Hx_k + v_k - H\hat{x}_k^-) \tag{22}$$

then an expression for P_k was obtained by subbing (22) into (17):

$$P_k = E\{[(x_k - \hat{x}_k^-) - K_k(H_k x_k + v_k - H_k\hat{x}_k^-)][(x_k - \hat{x}_k^-) - K_k(H_k x_k + v_k - H_k\hat{x}_k^-)]^T\} \tag{23}$$

In depth expectation simplification

By comparing (23) and (17) the estimation error e_k is built from the error in prediction and the error effect of measurement noise. The estimation error after the update step is:

$$e_k = (x_k - \hat{x}_k^-) - K_k(H_k x_k + v_k - H_k \hat{x}_k^-) \tag{24}$$

Initially e_k is the error between the true state and the prediction. The Kalman gain adjusts this based on what is actually measured $z_k = H_k x_k + v_k$ and what the expected measurment is $\hat{z} = H_k \hat{x}_k^-$ based on the model. Grouping the terms more intuitively by substituting in the definition for the prediction error $e_k^- = x_k - \hat{x}_k^-$ gets:

$$e_k = e_k^- \Omega_k - K_k v_k \tag{25}$$

Where $\Omega_k = I - K_k H_k$ describes how much of the prediction error remains after the update. Subbing (25) into (17) gives:

$$P_k = E[(e_k^- \Omega_k - K_k v_k)(e_k^- \Omega_k - K_k v_k)^T] \tag{26}$$

After expanding the brackets 4 terms are obtained:

$$P_k = E[\Omega_k, e_k^-(e_k^-)^T \Omega_k^T - \Omega_k, e_k^- v^T K_k^T - K_k v (e_k^-)^T \Omega^T + K_k v v^T K_k^T] \tag{27}$$

Then using the useful identity $E[Ax + By] \equiv AE[x] + BE[y]$ (27) can be rewritten:

$$P_k = \Omega_k, E[e_k^-(e_k^-)^I]\Omega_k^I - \Omega_k E[e_k^- v^I] K_k^I - K_k E[v(e_k^-)^I]\Omega^I + K_k E[v v^I] K_k^I \tag{28}$$

The first term represents the uncertainty left from the prediction after the update. The last term represents the uncertainty added by the measurement noise. The two middle terms represent the interaction between the prediction error and the measurement noise they are both zero since e_k and v_k are both have mean of 0 and are independent. Subbing in (18) into the first term, (16) into the last and our definition of Ω_k term gets (29).

Performing the expectations leaves an equation for P_k in terms of Kalman filter parameters:

$$P_k = (I - K_k H_k) P_k^- (I - K_k H_k)^T + K_k R_k K_k^T \tag{29}$$

Which is a general expression for the updated error covariance matrix for suboptimal K_k. Now optimize K_k by minimizing the MSE corresponding to finding the value of K_k that minimizes the individual terms along the leading diagonal of P_k as these terms represent the estimation error variances for the elements of the state vector elements being selected. This leaves:

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \tag{30}$$

For a m

In this case K_k is the optimum Kalman gain. When considering the optimum Kalman gain (29) can be simplified by ignoring the last two terms:

$$P_k = P_k^- - K_k H P_k^- \tag{31}$$

[BH12] (chapter 4).

Whereas the low pass filter passes $\hat{x}_{k-1}$ directly between time steps t_{k-1} and t_k the Kalman filter predicts the next step before a measurement is carried out and compares this with the measurement to give new estimate.

> **Note**
>
> H and R are only used in the prediction step.

The equations (19), (30) and (31) together describe the estimation step of the Kalman filter.

2.3 Prediction step

Unlike the low pass filter Kalman filters also consider the physics of the system, the predictions which are used alongside measurements for state estimation. The system dynamically changes over time and is modelled using A, which describes how the state is predicted to change between t_k and t_{k+1}:

$$\hat{x}_{k+1}^- = A\hat{x}_k \tag{32}$$

The error covariance matrix associated with this prediction $\hat{x}_{k+1}^-$ is obtained from:

$$P_k^- = AP_kA^T + Q \tag{33}$$

Q also appears here as this is how much the true state is expected to vary from the predictions. Together equations (32) and (33) equations encode the prediction phase of the filter.

> **Note**
>
> A and Q are only used in the prediction step.

[BH12]

If μ_x is t
its cova
transfor
[BAOOI2

2.4 Summary

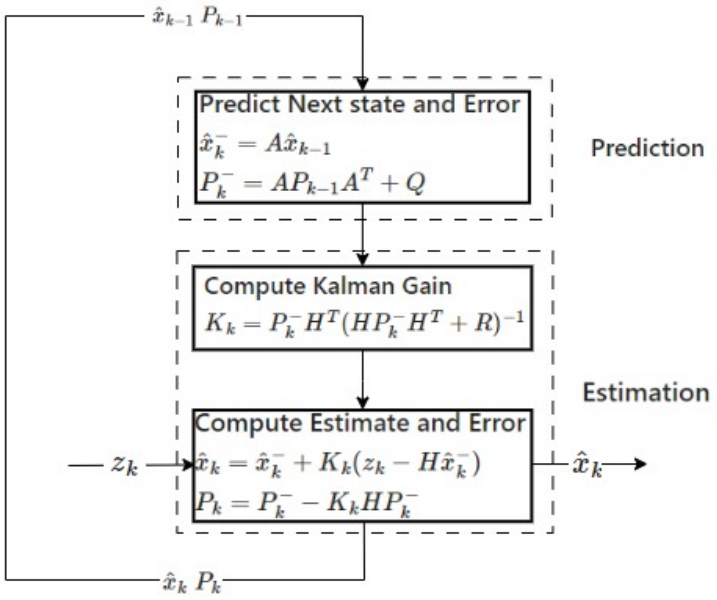

Fig. 7 Block diagram of the Kalman filter process.

- A and H are essential for the Kalman filter if these are incorrect the filter will not converge.
- Q and R are used for tuning and ensure an optimum fit.
- x_0 and P_0 are initial estimates the filter will still converge even if these are wrong.
- K_k and P_k are internal parameters.
- z_k are the inputs (measured state)
- $\hat{x}_k$ are the outputs (estimated state)

3 Example: Battery output

This example looks at how a Kalman filter can be used to fit a constant signal in this case the output from a battery. The aim of this simple example is to experiment with changing the Kalman parameters A, H, Q, R, $\hat{x}_0$ and P_0. This example was inspired by Dr Shane Ross a link to the video can be found here.

3.1 Model

- $\hat{x}_k$ is the estimate of the true battery output in volts, $n = 1$,
- z_k is the measured battery output in volts, $m = 1$ In this very simple example A, H, Q and R are all scalars since $m = n = 1$
- $A = 1$ the battery output is constant which means $x_{k+1} = x_k$ comparing with (32) shows $A = 1$
- $Q = 0$ Since the battery output is known to be constant.
- $H = 1$ since estimate corresponds directly to measurement see (14).
- $R = 4$ initially but was determined through tuning.
- x_0 was set to 5
- P_0 was set to 1

3.2 Testing

The code for this example can be found here. The code is broken up into 3 files: `GenTestSig.py` which generates the true signal (a constant value) and adds gaussian noise to it; `SimpleKalman.py` contains the algorithm for the kalman filter for the 1D case; and `TestWidget.py` calls functions from the other files and plots the results, with a widget so the effects of the kalman parameters can be easily visualized.

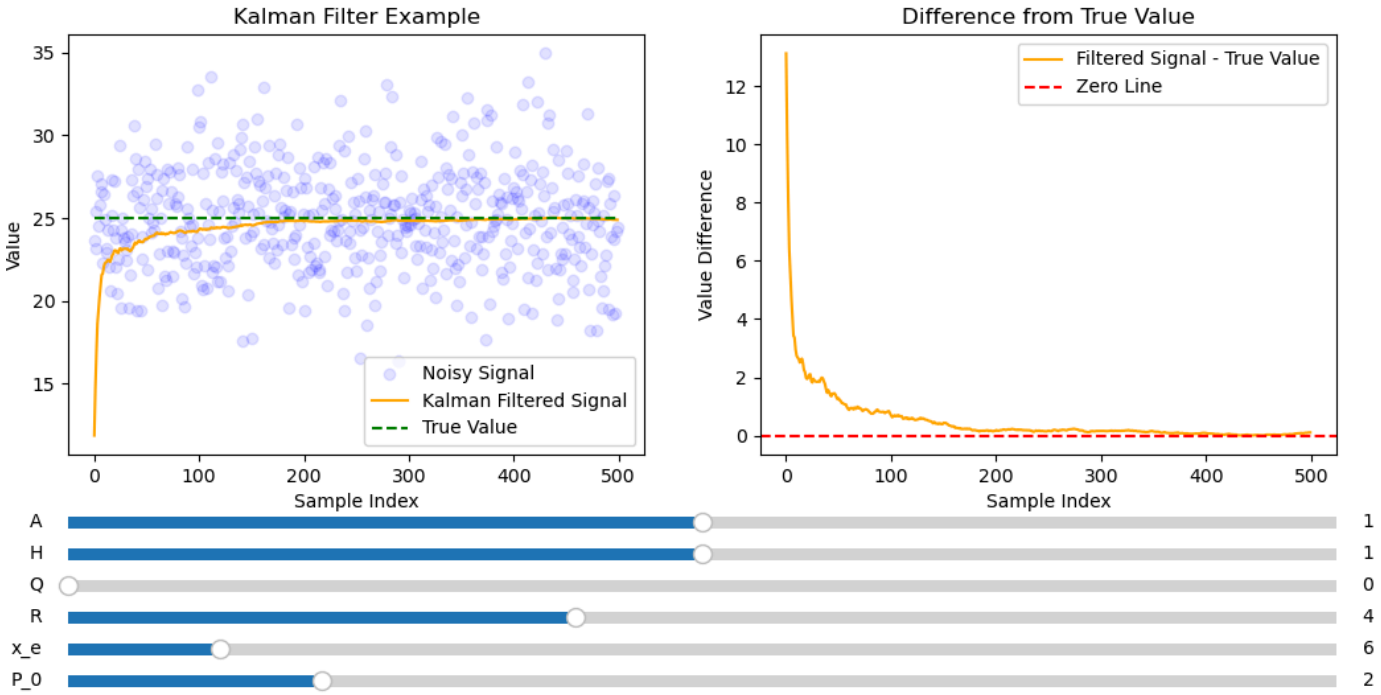

Fig. 8 Kalman filter used to fit a non-varying signal with a large amount of noise.

Fig. 8 shows that convergence is faster and smoother than the average filter. This makes sense as the kalman filter looks at both the model and the data to reduce noise.

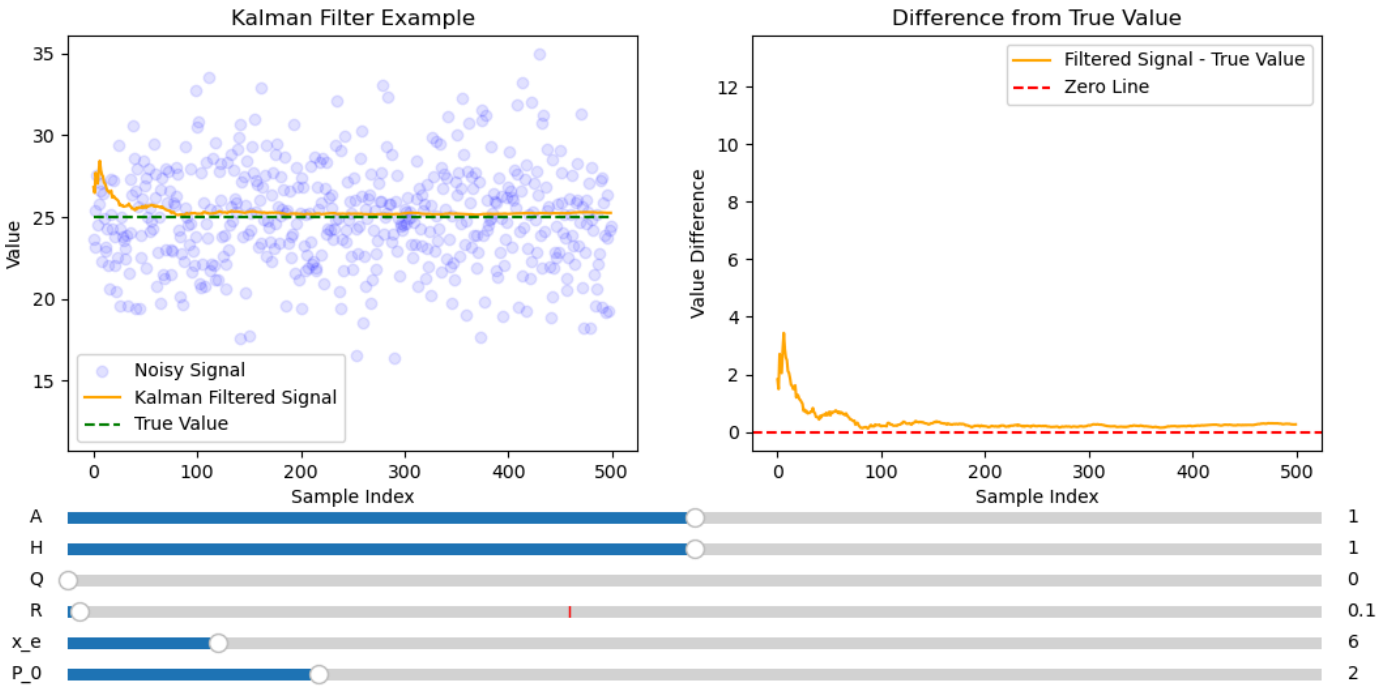

Fig. 9 Fig. 8 but with smaller R.

Fig. 9 shows smaller R makes convergence less smooth. This puts more faith in noisy z_k compared to less noisy $\hat{x}_k^-$ so in the correction stage of the kalman filter more emphasis will be put on noisy z_k than on less noisy $\hat{x}_k^-$. The signal will converge faster when R is smaller as the initial guess is far from x_k, this means initially x_k^- will be less accurate than z_k since the prediction phase assumes that $\hat{x}_k^- =\hat{x-1}_k$ which for the initial guess is incorrect. The downside of this is that the signal is more effected by noise.

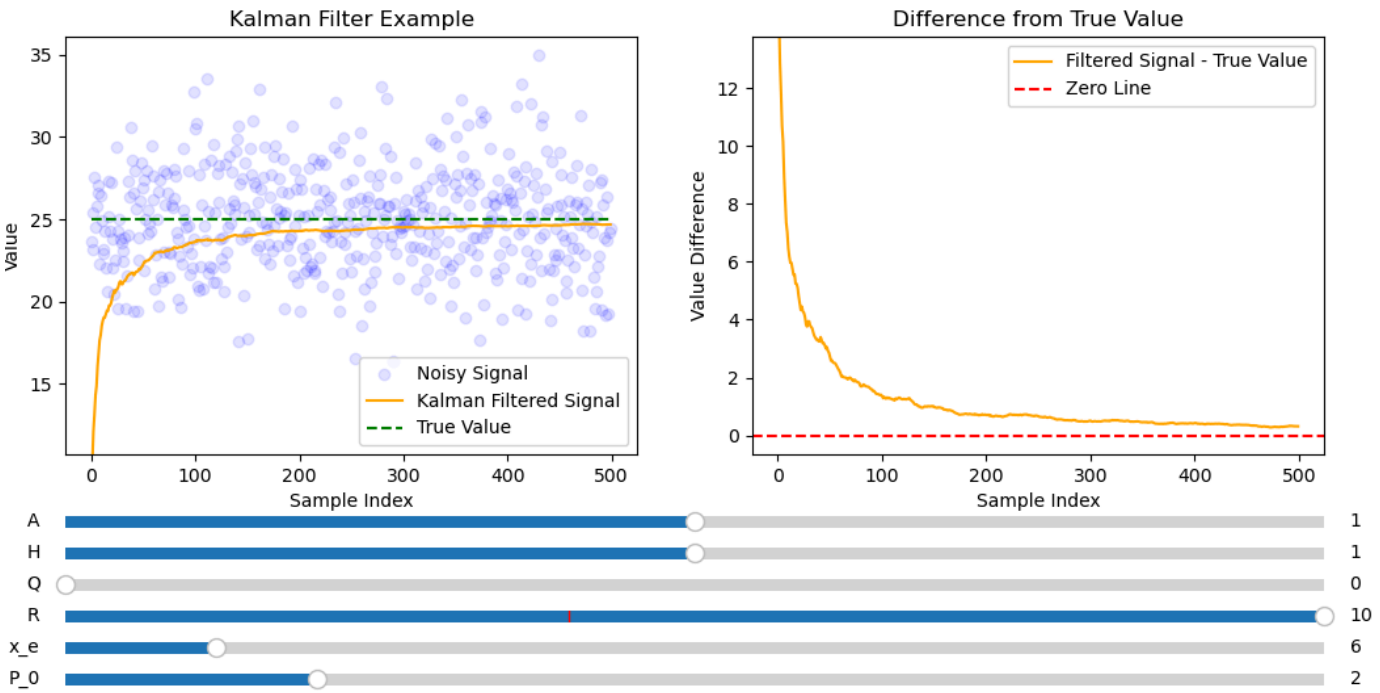

Fig. 10 Fig. 8 but with larger R.

Fig. 10 shows slower convergence than figures 4.1 and 4.2 but has the smoothest convergence of the three. Fig. 8 seems to be best as it has a good tradeoff between a noise free filtered signal and fast convergence.

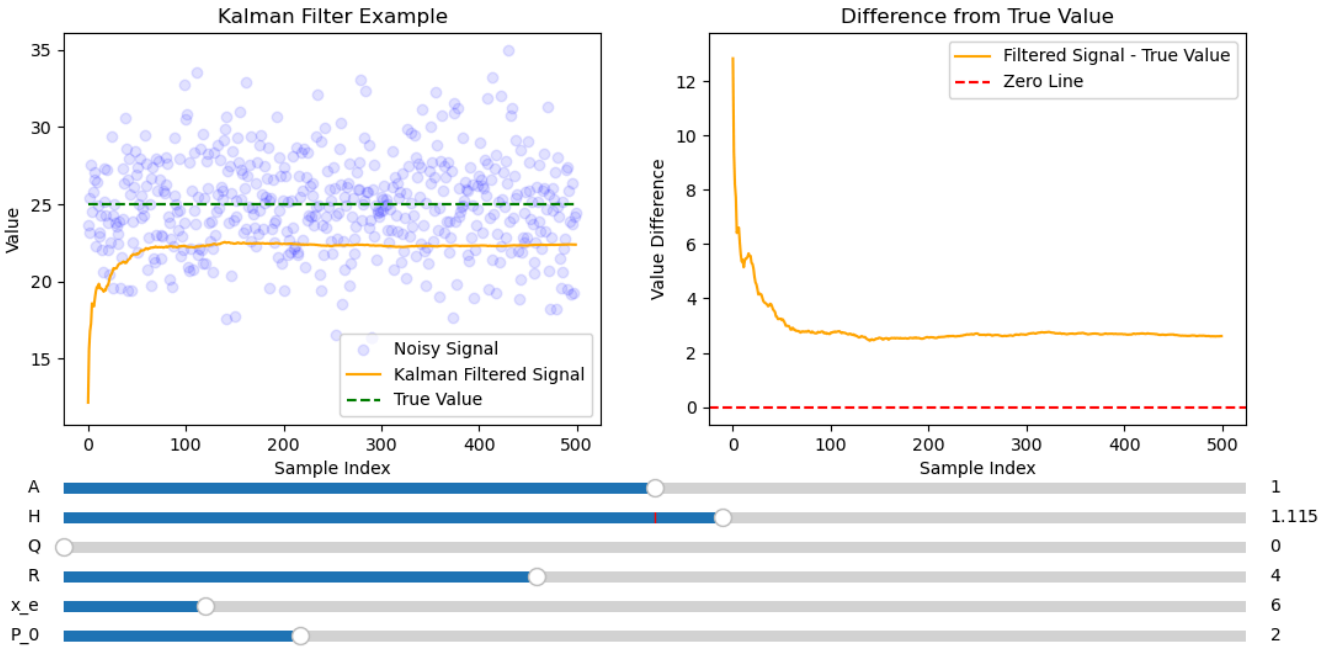

Fig. 11 Fig. 8 with H set to 1.115 rather than 1

Varying H will lead to the Kalman filter converging to the wrong value since the measurement directly measures the state only satisfied by $H = 1$.

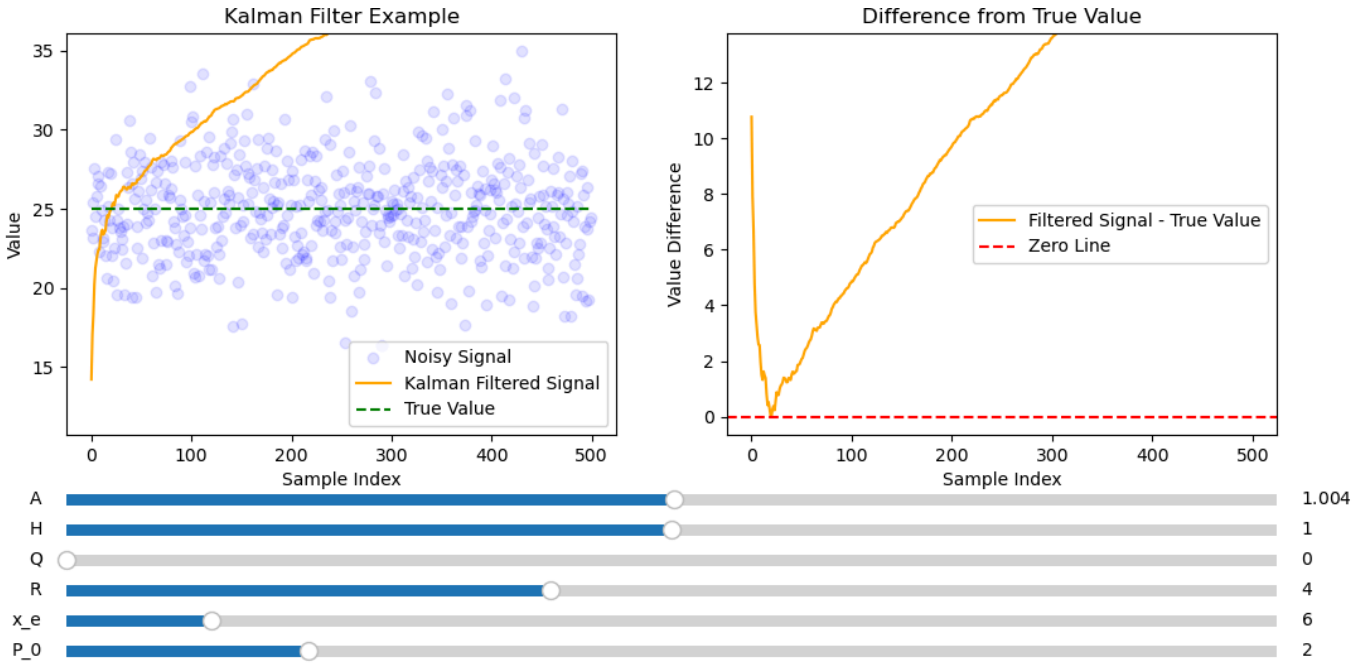

Fig. 12 Fig. 8 with A set to 1.004 rather than 1.

This causes the Kalman filter to diverge since only $A = 1$ describes a straight line. It diverges quickly since $Q = 0$ puts more weighting on $\hat{x}_k$ compared to $\hat{z}_k$.

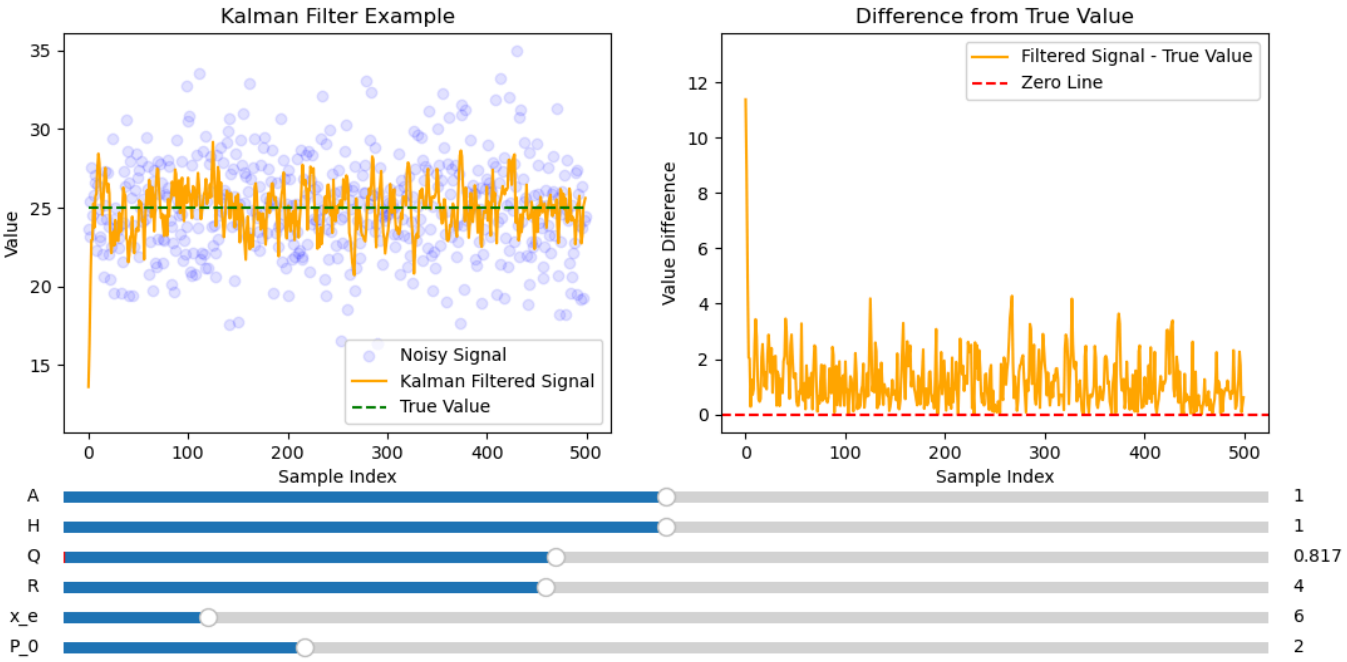

Fig. 13 Fig. 8 except with $Q = 0.817$ rather than 1.

Setting Q to anything other than 0 in this example makes the filtered signal noisy, because it assumes the model is noisy. Increasing Q puts more trust $\hat{z}_k$ over $\hat{x}_k^-$ when predicting $\hat{x}_k$ which means the signal becomes noisy like the data.

3.3 Summary

A describes the model and H describes how the measurement relates to the state. If these are incorrect the fit will converge to the wrong value or diverge completely. Q and R are covariance matrices, these are "tuned" to achieve optimum fit.

4 Example: Velocity from position

This example looks at applying a Kalman filter to estimate the true position and velocity from noisy position data. This example was inspired by Dr Shane Ross a link to the videos are here: Video 1 and Video 2.

4.1 Model

In this case $\hat{x}_k = \begin{bmatrix} s_k \\ v_k \end{bmatrix}$ where s_k and v_k are the position and velocity (in the same direction) respectively at time t_k. The correction measurement is position therefore $z_k = s_k$. The measurement and the state are related by (14) from this H can be calculated.

$$\begin{aligned} z_k &= H \begin{bmatrix} s_k \\ v_k \end{bmatrix} \\ \implies H &= \begin{bmatrix} 1 & 0 \end{bmatrix} \end{aligned} \tag{34}$$

For simplicity and to avoid needing to use a forcing function the model assumes no acceleration. The equations of motion are:

$$\begin{aligned} s_{k+1} &\approx s_k + \nu_k \Delta t \\ \nu_{k+1} &\approx \nu_k \end{aligned} \tag{35}$$

why use $\approx$ not $=$

There is a small error associated with discretizing dt since Δt isn't infinitesimally small. This is negligible term to term but over a large number of terms such as in this example the error accumulates and causes drift.

Where $\Delta t = t_{k+1} - t_k$. Now writing the update equations in matrix form the following are obtained:

$$\begin{bmatrix} s \\ \nu \end{bmatrix}_{k+1}^{-} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ \nu \end{bmatrix}_k \tag{36}$$

Q is more complicated to calculate and is going to be a 2×2 matrix of the form below:

$$Q = E[w_k w_k^T] = \begin{bmatrix} \mathrm{VAR}(s) & \mathrm{COV}(s,\nu) \\ \mathrm{COV}(\nu,s) & \mathrm{VAR}(\nu) \end{bmatrix} = \sigma_Q^2 \begin{bmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & \Delta t^2 \end{bmatrix}$$

[BAOOI21] (chapter 2.3) [BAOOI21] (chapter 10.2) where σ_Q^2 is the variance in the true acceleration, which is the model assumes is zero. σ_Q^2 will be used as a tuning parameter. $R = \mathrm{VAR}(z_k)$ is going to be the variance in the measurements of the position which will be denoted as σ_R [BAOOI21] (chapter 10).

The following parameters will be used for the Kalman filter.

- $\hat{x}_k = \begin{bmatrix} s_k \\ \nu_k \end{bmatrix}$
- $z_k = s_k$
- $A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$
- $H = \begin{bmatrix} 1 & 0 \end{bmatrix}$
- Q and R will are tuned using the parameters σ_a and σ_s.
- $\hat{x}_0 = 0$ and $P_0 \approx \begin{bmatrix} P_s & 0 \\ 0 & P_v \end{bmatrix}$ initially, these don't matter so much as the Kalman filter will eventually find the correct values as k increases.

Note

While optimal P_0 isn't necessarily always diagonal matrix this is a good enough initial approximation as the kalman filter will converge if A and H are correct.

4.2 Implementation

The code for this example can be found here. The code is broken up into 3 files: `GenTestSig.py` which generates the true signal (a combination of sine waves of different frequencies) and adds gaussian noise to it; `AdvKalman.py` contains the algorithm for the Kalman filter for the multidimensional case; and `TestUpdated.py` calls functions from the other files and plots the results, with a widget so the effects of the Kalman parameters can be easily visualized.

The true signal in the graphs below is generated using a combination of $\sin$ waves of varying amplitudes and speeds, the velocity version is the analytical derivative of this. The signal the Kalman filter is being used to fit is generated by adding random noise to the true signal and its analytical derivative. The Kalman filter is applied to the noisy position data and outputs the filtered position data and filtered velocity data. Which are graphed below. The r^2 value, mean squared error (MSE) and mean absolute error (MAE) are given for both velocity and position.

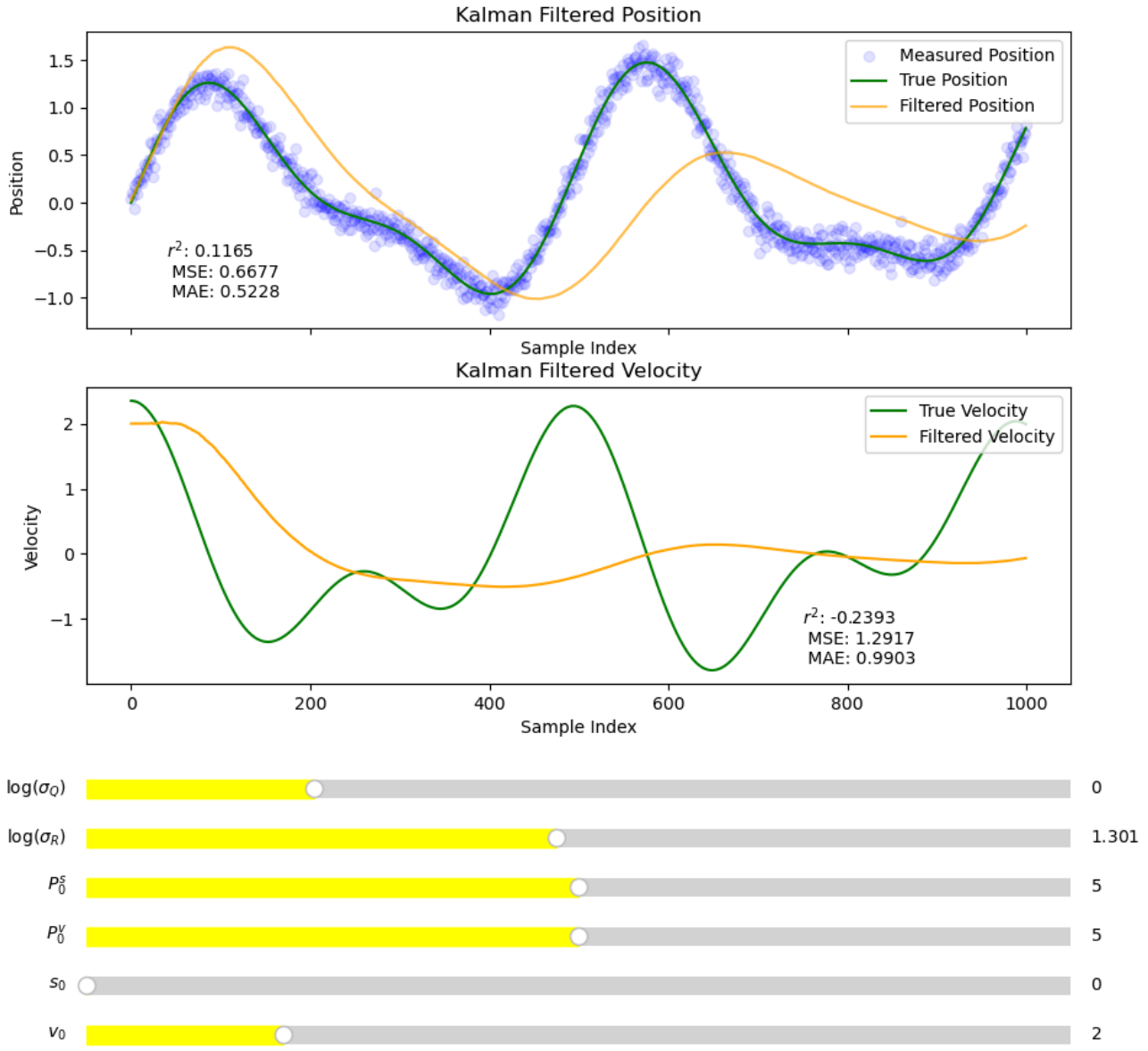

Fig. 14 Filtered, unfiltered and true position and velocity plotted against index.

In both cases the kalman filtered fit is lagging behind the true and noisy signal. This is because too much emphasis is being put on the predicted values which assumes that velocity stays the same. The true signal shows this clearly this isn't the case. This means the velocity prediction is consistently delayed so the position will also be delayed. Since the model is erroneous it was assumed that w_k was larger and therefore σ_Q should be increased. A similar effect can be achieved by assuming smaller measurement noise (smaller v_k) therefore σ_R should be decreased.

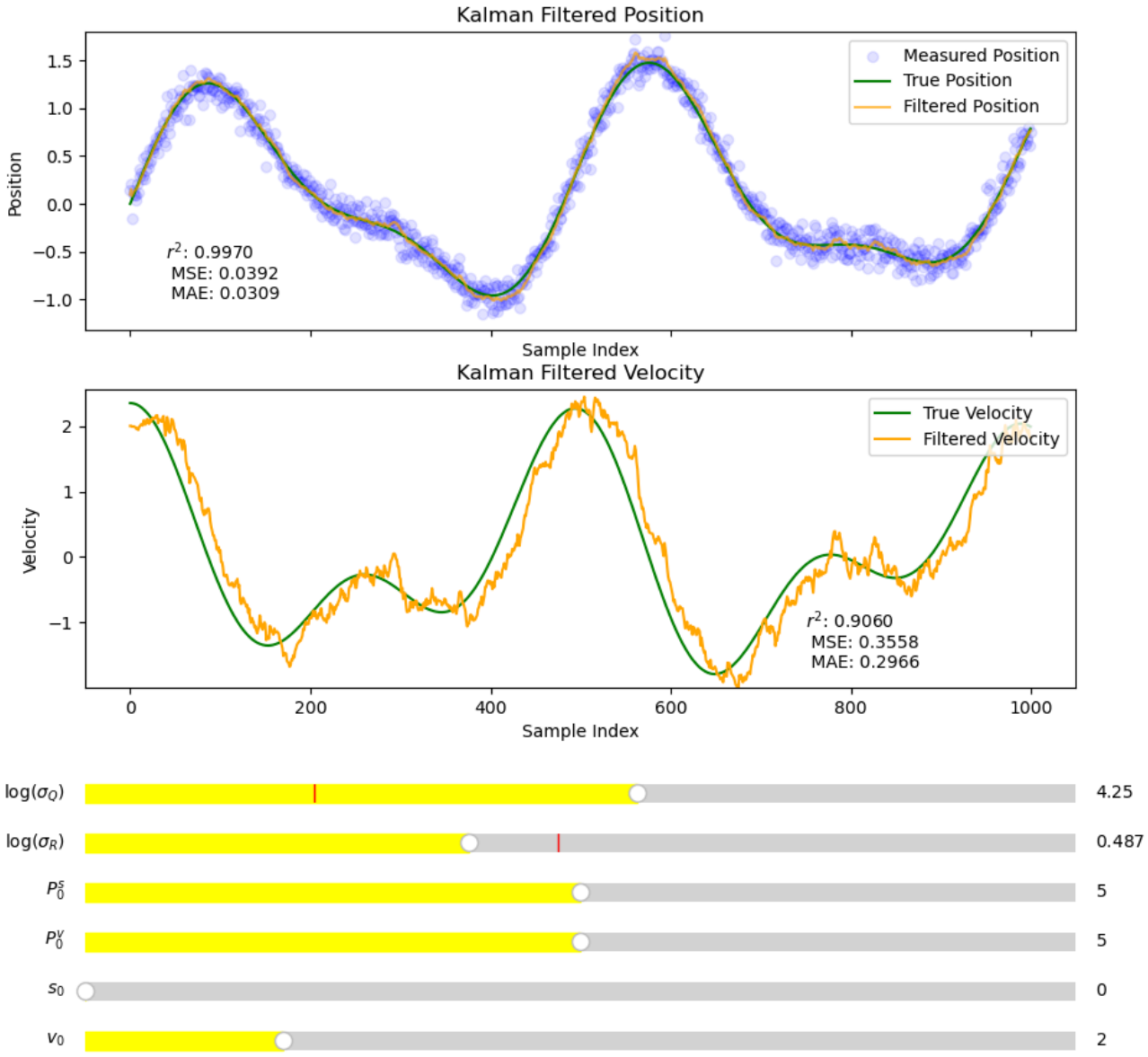

Fig. 15 See Fig. 14 with increased σ_Q and decreased σ_R.

Figure Fig. 15 shows a slightly improved fit with position and a significantly improved fit with velocity. This makes sense for the position data since decreasing R increases the weighting for the measurement. However the velocity fit is now significantly less smooth since more emphasis is being put on the measured positions which is always nosier than the prediction. There is a clear tradeoff between having a a smooth fit and having an accurate fit. Smoother fits more accurately represent the true shape of the data, but will often result often this comes with drift.

Important

If the Kalman filter fit is too noisy, due to an over emphasis on $\hat{z}_k$ compared to $\hat{x}_k^-$ to calculate $\hat{x}_k$ increasing R or decreasing Q will make the fit smoother. If the Kalman filter fit is delayed increasing Q or decreasing R will reduce the delay. Increasing Q has a similar effect to decreasing R.

4.3 Summary

The kalman filter does a good job of estimating the true position from the data and the model. The velocity fit is less accurate and significantly delayed, but is good considering there are no measurements for the velocity. The fit for velocity would be greatly improved using an additional sensor to easily measure velocity or acceleration to be used in the prediction step.

5 Example: Attitude using a gyroscope and accelerometer

This section calculating the attitude of an object (yaw-pitch-roll), using real world sensor data. Firstly using the gyroscope to measure the angular velocity and then using Euler's method for integration to obtain a prediction for the attitude, then improving this using a kalman filter, however this estimate drifts and becomes less accurate over time. To reduce drift accelerometer is used and the Kalman filter fuses data from the accelerometer and gyroscope.

The code for this section can be found in here. The code is broken up into 3 files: `Test.py` which runs the filter and plots the results; `AdvKalman.py` which contains the kalman filter algorithm; and `Integrate.py` which contains the integration algorithm which calculates the attitude without the kalman filter.

This example and data was provided by Dr Shane Ross a link to the video can be found here.

5.1 Euler's method

Using gyroscope (measures angular velocity, ω) and knowing the attitude at t_0 it is possible to determine the attitude of a craft at t_k. The relationship between the body angular velocity $\omega = (\omega_1, \omega_2, \omega_3)$ and the time derivatives of the Euler angles $\alpha = (\psi, \theta, \phi)$ in the 3-2-1 (yaw-pitch-roll) sequence is given by the kinematic differential equation (KDE) below:

$$\begin{bmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \frac{1}{\cos\theta} \begin{bmatrix} 0 & \sin\phi & \cos\phi \\ 0 & \cos\phi\cos\theta & -\sin\phi\cos\theta \\ \cos\theta & \sin\phi\sin\theta & \cos\phi\sin\theta \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{37}$$

ϕ, θ and
describe
space. W
rotation
rate.

Or more simply :

$$\dot{\alpha} = \Phi(\alpha)\omega \tag{38}$$

ψ, yaw:
the vert
head lef

θ, pitch:
the side

where $\Phi(\alpha) = \frac{1}{\cos\theta}\begin{bmatrix} 0 & \sin\phi & \cos\phi \\ 0 & \cos\phi\cos\theta & -\sin\phi\cos\theta \\ \cos\theta & \sin\phi\sin\theta & \cos\phi\sin\theta \end{bmatrix}$

The rates of rotation (provided by the gyroscope) are integrated step by step to predict the craft's attitude at any later time t_k:

$$\alpha_{k+1} \approx \alpha_k + \dot{\alpha}_k \Delta t \tag{39}$$

Subbing the relationship between $\dot{\alpha}_k$ and ω (38) into the update rule (39) gives:

$$\alpha_{k+1} \approx \alpha_k + \Phi(\alpha_k)\omega_k \Delta t \tag{40}$$

where $\Delta t = t_k - t_{k-1}$.

> **why use $\approx$ not $=$**
>
> There is a small error associated with discretizing dt since Δt isn't infinitesimally small. This is negligible term to term but over a large number of terms such as in this example the error accumulates and causes drift.

your he
motion.

ϕ, roll: r
(x) axis.
your ea

The Eul
same as
orientat
the obje

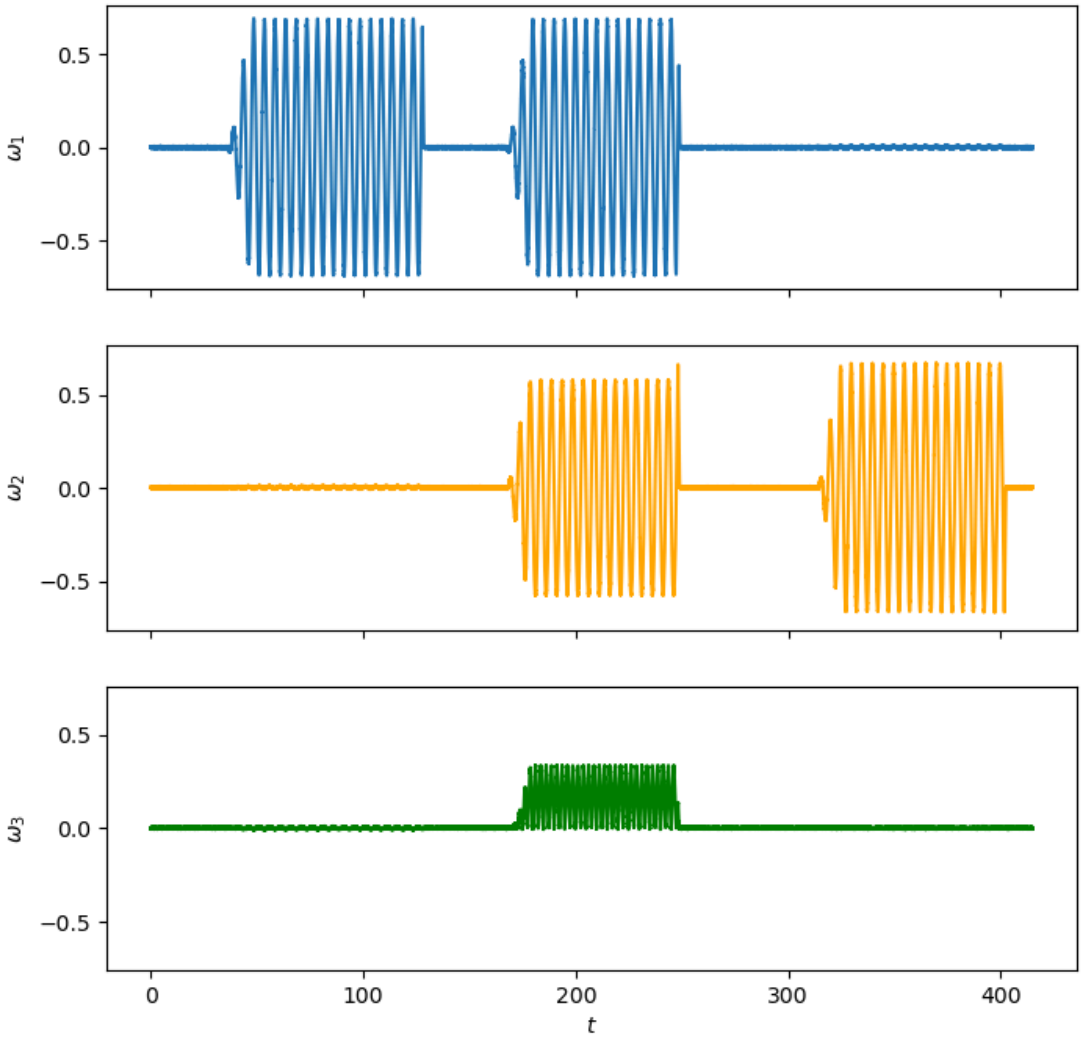

Fig. 16 Calibration signal involved shaking the device in just the ω_1 direction then pausing and shaking the device in the ω_1 and ω_2 directions before pausing again and shaking the device in the ω_3. The observed motion in the ω_3 direction was erroneous. View in Github

Using the calibration data in Fig. 16 and knowing the initial attitude (40) can be used to estimate the attitude over time:

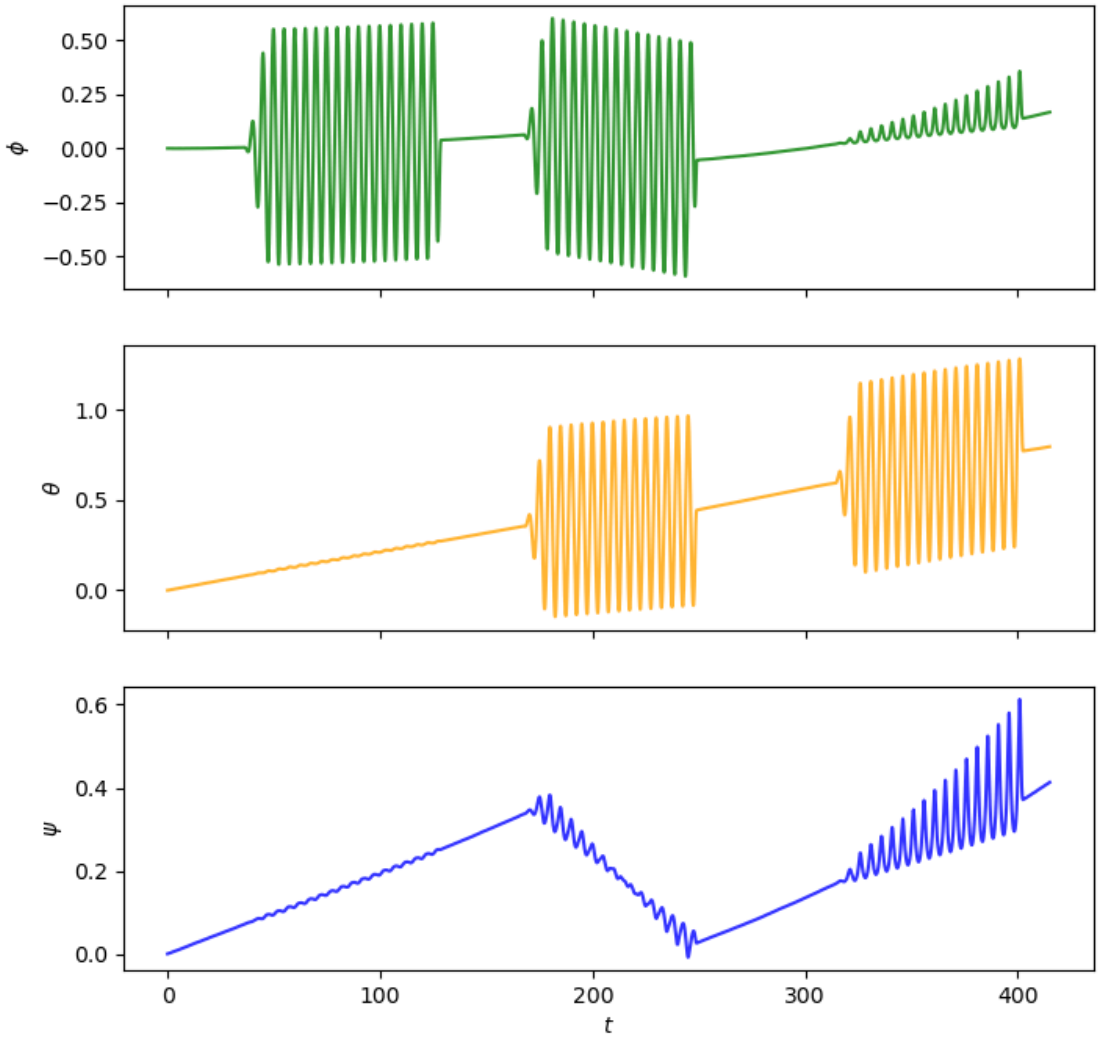

Fig. 17 α_k in each direction plotted against t_k. View in Github

Overall the attitude α_k follows the generic shape of the true attude. The drift is least significant in the ϕ direction but was predicting oscillations in the third part of the calibration even though there were no oscillations in the ω_3 direction at that time, and the drift also changes direction randomly. Drift was most pronounced in the θ direction therefore the estimate becomes less accurate over time.

Drift is caused by the error associated with the numerical integration accumulating over time. The unexpected oscillations are likely from the gyroscope being sensetive to noise. Furthermore the oscillations could be coupled meaning small oscillations in one direction can be amplified in another.

5.2 Kalman filters

The model can be improved using a Kalman filter. However there is a problem as its not possible to put our update equation (40) into the form required for the kalman filter (32). To fix this the attitude was instead

written in quaternions.

> **Tip**
>
> Use Euler parameters:

$$\begin{aligned}
\beta_0 &= \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} + \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} \\
\beta_1 &= \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\
\beta_2 &= \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \\
\beta_3 &= \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2}
\end{aligned} \tag{41}$$

Below is the corresponding Euler parameters KDE:

$$\begin{bmatrix} \dot{\beta}_0 \\ \dot{\beta}_1 \\ \dot{\beta}_2 \\ \dot{\beta}_3 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & \omega_3 & -\omega_2 \\ \omega_2 & -\omega_3 & 0 & \omega_1 \\ \omega_3 & \omega_2 & -\omega_1 & 0 \end{bmatrix}\begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \tag{42}$$

[BAOOl21] (chapter 11.3) or more simply:

$$\dot{\beta} = \Psi(\omega)\beta \tag{43}$$

Where:

$\Psi(\omega) =$

$\dot{\beta}$ was integrated using the same method as (39).

$$\beta_{k+1} \approx \beta_k + \dot{\beta}_k \Delta t \tag{44}$$

$\approx$ was used instead of $=$ for the same reason as in the previous example the numerical integration will mean β_k drifts further away from its true value as k increases. Now sub in (43):

$$\beta_{k+1} \approx (I + \Delta t \Psi(\omega))\beta_k \tag{45}$$

Rewriting (45) with β_{k-1} on the right hand side side being the previous estimate estimate of the state and β on the left hand side becoming the prediction of the state:

$$\hat{x}^-_{k+1} = (I + \Delta t \Psi(\omega))\hat{x}_k \tag{46}$$

Which is in the form required by (32) with $A = (I + \Delta t \Psi(\omega))$ It follows from the euler parameters (41) and $\alpha_0 = 0$ that $\hat{x}_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$.

The gyroscope data was used in the prediction step of the kalman filter, since it relies on the previous state to be able to predict the next state. The correction measurement used in this example will be the previous state $\hat{x}_k$, this will help reduce noise in the estimate.

Important

A correction measurement measures the state without needing to rely on previous measurements or approximations its error should be random. A prediction relies on previous measurements. In this case the prediction measurement error is systematic because of integration drift.

Since both z_k and $\hat{x}_k$ represent euler parameters $H = I_{4\times4}$ which can be understood from (14). Finally the tuning parameters were set $Q = qI_{4\times4}$, $R = rI_{4\times4}$ and $P_0 = pI_{4\times4}$ to allow for simple tuning.

Warning

It is unlikely that optimal Q, R and P_0^- are scalar multiples of the identity, but this method reduces the number of parameters that need to be tuned.

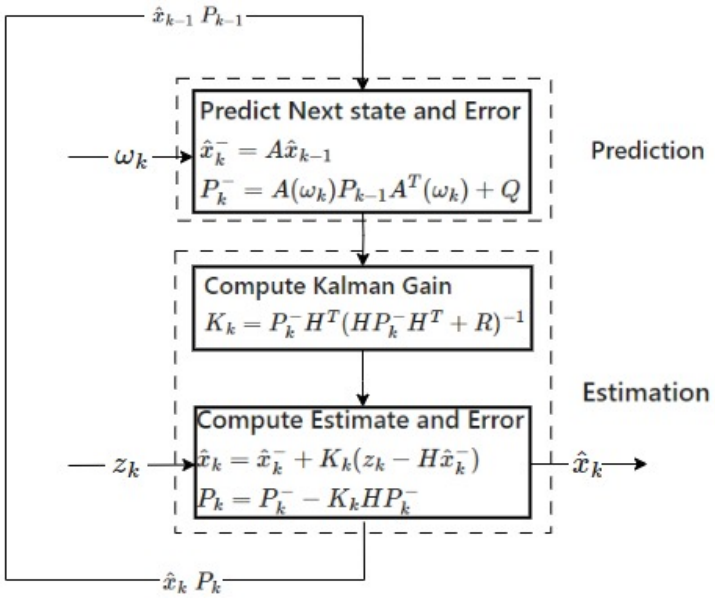

Fig. 18 Kalman filter block diagram specific to attitude determination.

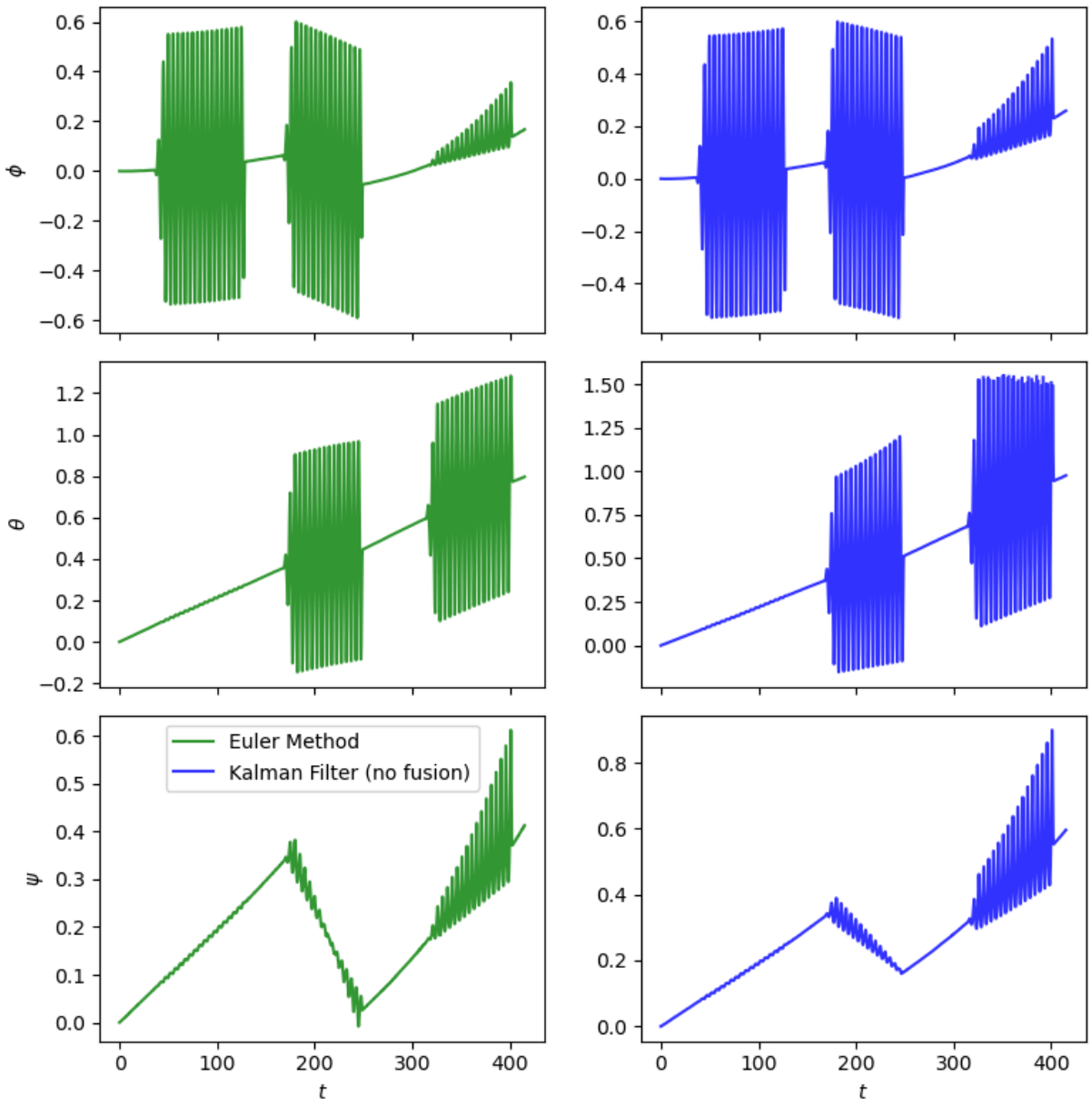

Fig. 19 The euler method for calculating attitude alongside the kalman filtered example discussed above.
View in Github

This kalman filter example hasn't improved the fit. The integration drift hasn't been corrected for. This is because the measurement in this case didn't contain any corrective information so didn't correct for drift. **The only difference the kalman filter makes in this case is it puts a greater emphasis on previous measurements**. A better choice would be to use a sensor which doesn't rely on the previous measurement to calculate attitude.

5.3 Kalman filters with sensor fusion

Sensor fusion involves combining different sensors to get a better estimate. A typical six axis IMU will contain a gyroscope and an accelerometer. Accelerometer data to calculate attitude but is noisier than gyroscope data. However accelerometer data can be used to calculate attitude without drift as it doesn't involve numerical

integration. The aim of this part is to use the kalman filter to do sensor fusion to produce a filtered signal with less noise than the accelerometer and no drift.

Accelerometer Data

Here accelerometer data is used to calculate z_k which represents the attitude in terms of euler parameters as measured using the accelerometer. The accelerometer measures the acceleration, $\bar{a}$ in the x, y and z directions. A accelerometer moving at a constant velocity can always identify which direction is down due to the acceleration from gravity. This means it can determine θ and ϕ but not ψ.

The acceleration in the body fixed frame, the frame of the craft as seen by a stationary observer on earth is given by:

$$[a]_B = [\dot{v}]_B - [g]_B \tag{47}$$

Where $\dot{v}$ is the translational acceleration and g is the acceleration due to gravity. **Assumption: the translational acceleration of the body is zero and the accelerometer is located at the center of rotation for this example.**

Note

B is the linear transformation matrix which transforms form the frame of the earth to the frame of the device.

$$B = \begin{bmatrix} \cos\psi\cos\theta & \sin\psi\cos\theta & -\sin\theta \\ \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \sin\psi\sin\theta\sin\phi + \cos\phi\cos\psi & \cos\theta\sin\phi \\ \cos\psi\sin\theta\cos\phi + \sin\phi\sin\psi & \sin\psi\sin\theta\cos\phi - \sin\phi\cos\psi & \cos\theta\cos\phi \end{bmatrix}$$

This can be written in terms of unit vectors describing the effects on the x, y and z components.

$$B = [\hat{n}_x \quad \hat{n}_y \quad \hat{n}_z]$$

where $\hat{n}_x$, $\hat{n}_y$ and $\hat{n}_z$ are 3 dimensional unit vectors in the x, y and z directions.

The acceleration in the frame of the body can be calculated from its position relative to the earth:

$$[a]_B = -Bg = -g\hat{n}_z = g\begin{bmatrix} \sin\theta \\ -\cos\theta\sin\phi \\ -\cos\theta\cos\phi \end{bmatrix} \tag{48}$$

In component form $a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$ rearranging (48) gets:

$$\theta = \arcsin(\frac{a_1}{g}) \quad \phi = \arcsin(\frac{-a_2}{g\cos\theta}) \tag{49}$$

From the accelerometer data alone it is possible to directly calculate θ and ϕ but not ψ. In this example let $\psi = 0$ since no oscillations were performed in the ω_3 direction.

Warning

Additional noise has been added to the accelerometer readings to make the effects of the kalman filter more visible. Usually the accelerometer is more sensitive to noise than the gyroscope. Although the accelerometer data in the next few examples is definitely nosier it is hard to visualize, therefore additional gaussian noise has been added.

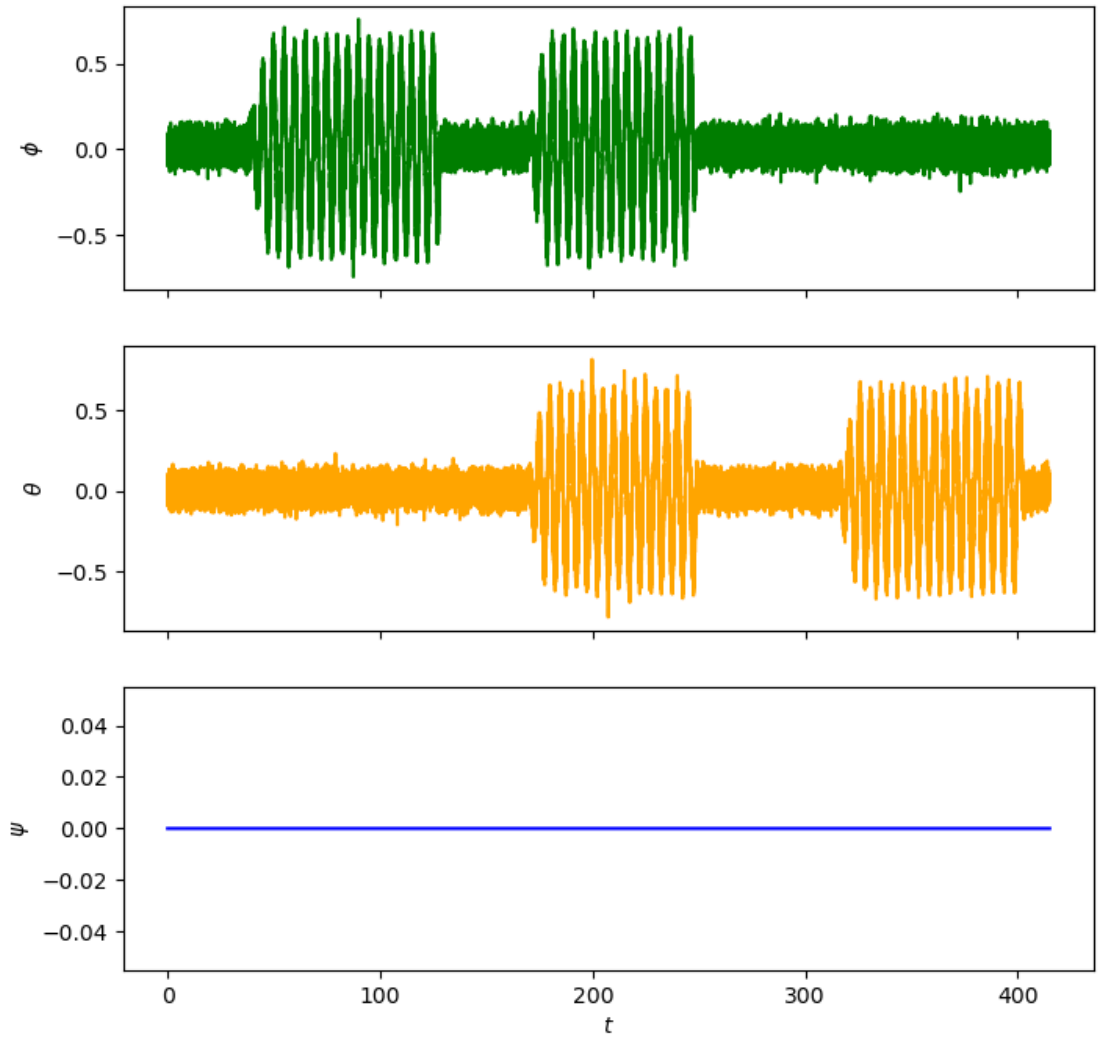

Fig. 20 yaw-pitch-roll against time using only accelerometer data for the same calibration mentioned above.
View in Github

Fig. 20 is much nosier than the predicted data from the gyroscope, see Fig. 17. For small k the gyroscope is more accurate as the drift is less significant compared to the noise from the accelerometer however for large k the accelerometer is more accurate as the gyroscope measurements are subject to drift.

Improved Kalman Filter

Using the real world accelerometer and gyroscope data Q and R were tuned to obtain the optimal fit for the data.

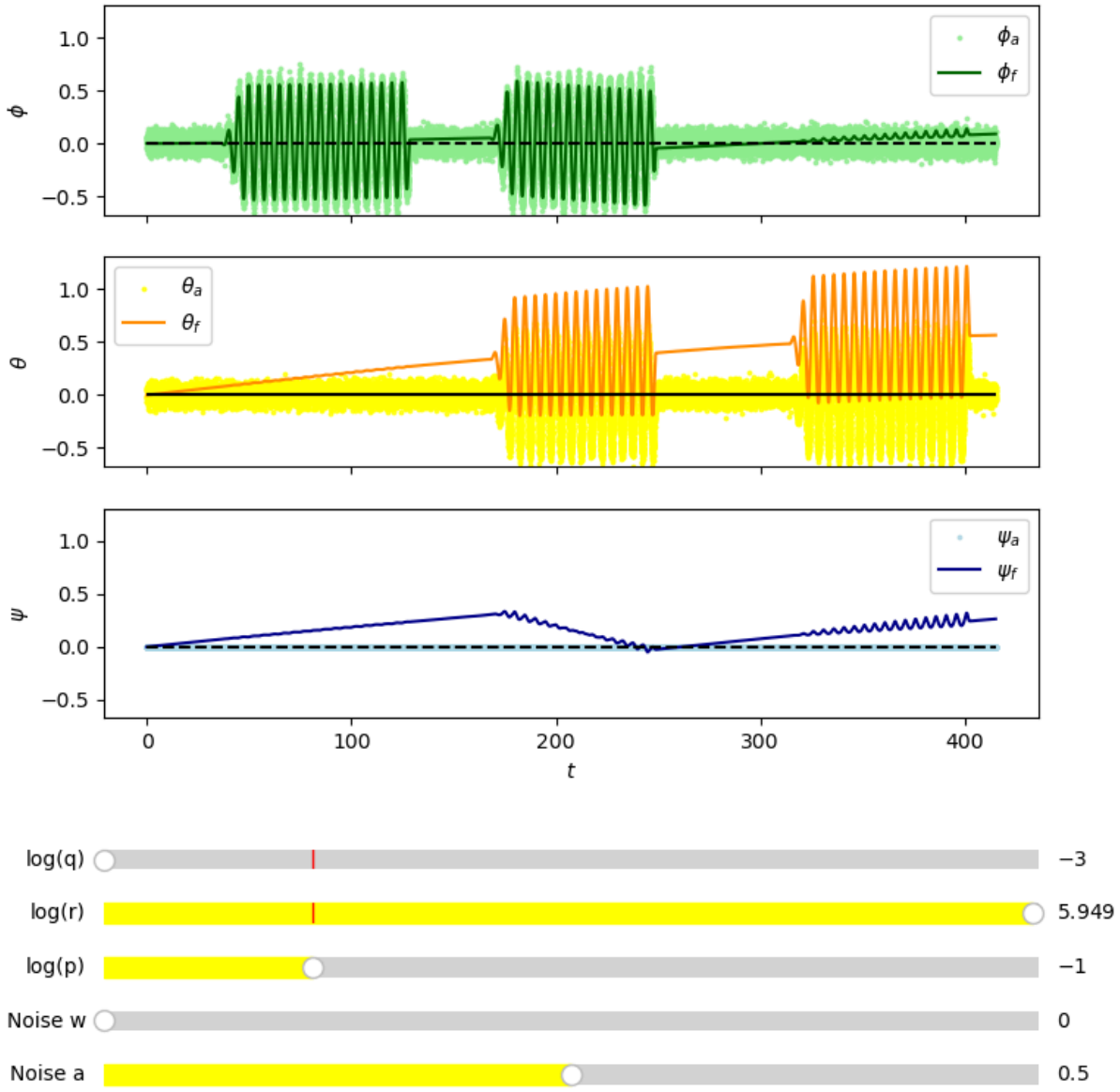

Fig. 21 Testing the kalman filter with small q and large r. ϕ_a represents ϕ measured from accelerometer data. ϕ_f represents ϕ from the kalman filter with sensor fusion. View in Github

Fig. 21 is similar to the predicted data in Fig. 17 which would suggest the filter is working correctly as small q and large r mean the filter gives more weighting to predicted (gyroscope) data compared to measured (accelerometer) data.

Given q
β_2, and
can be c

$\phi = a$

$\theta = a$

$\psi = a$

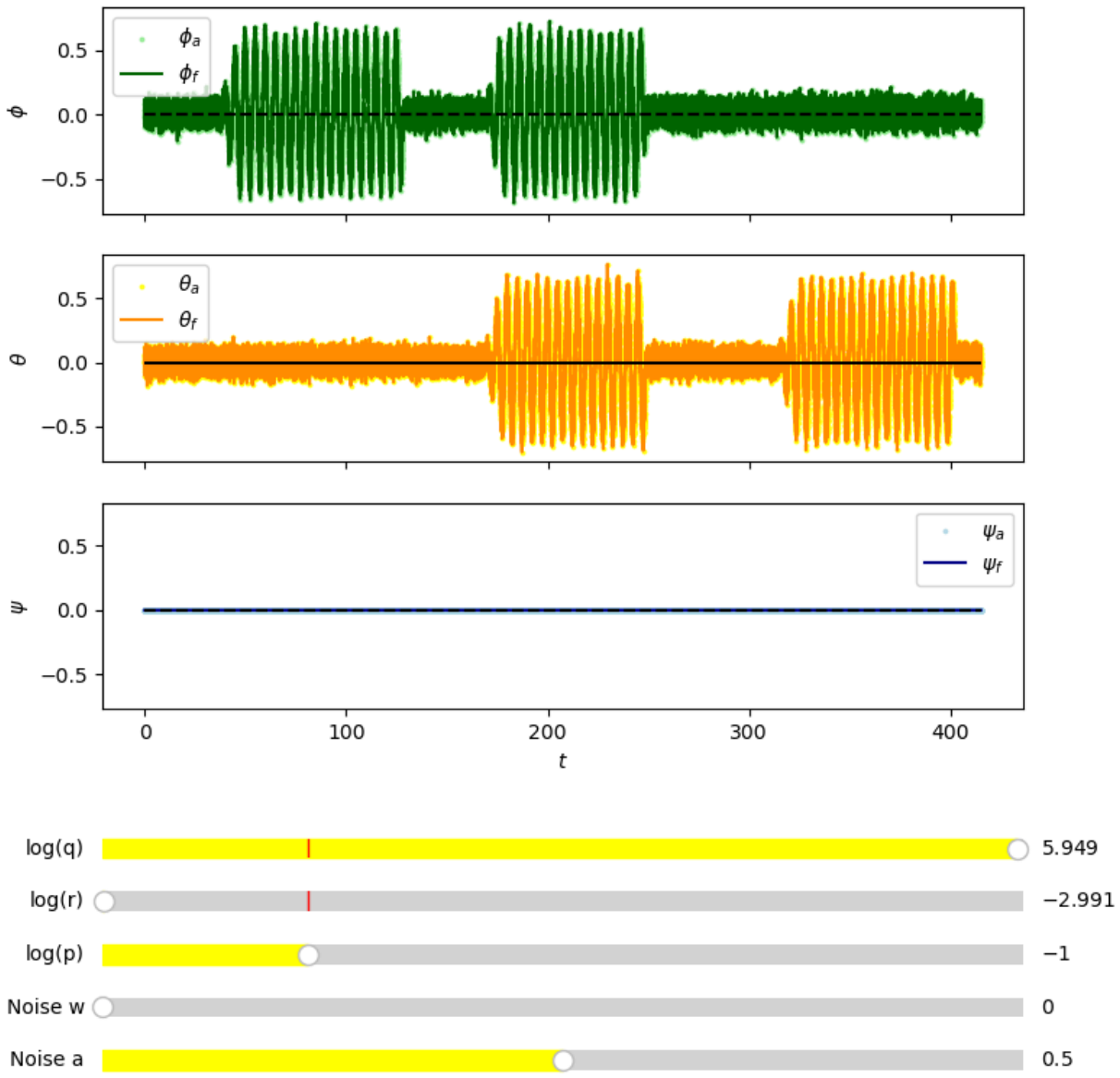

Fig. 22 Testing the kalman filter with large q and small r. ϕ_a represents ϕ measured from accelerometer data. ϕ_f represents ϕ from the kalman filter with sensor fusion. View in Github

Fig. 22 is very noisy and is similar to the predicted data in Fig. 20, similarly this would suggest the filter is working correctly since large q and small r mean the filter gives more weighting to measured data compared to predicted data.

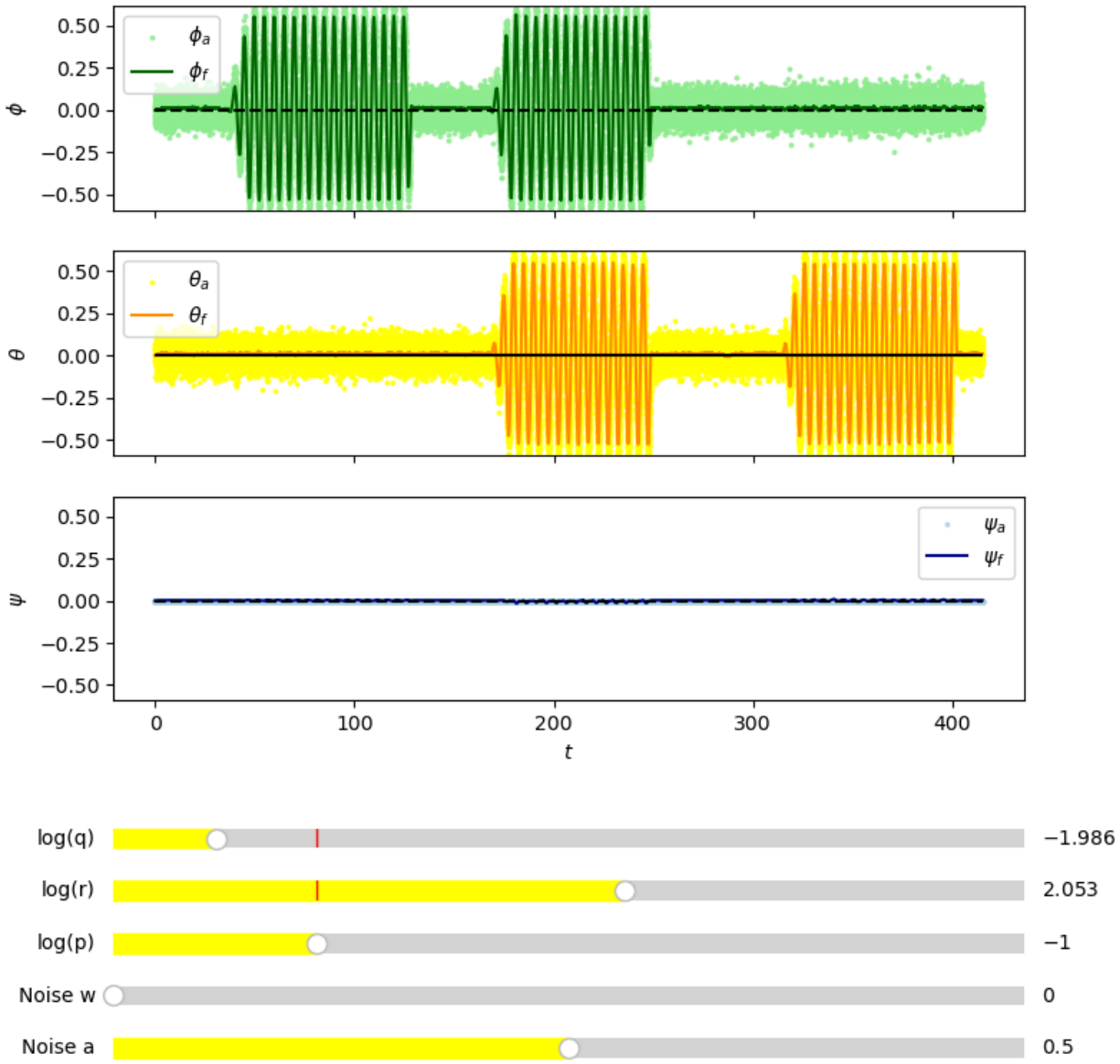

Fig. 23 Kalman filter tuned optimally by eye. ϕ_a represents ϕ measured from accelerometer data. ϕ_f represents ϕ from the kalman filter with sensor fusion. View in Github

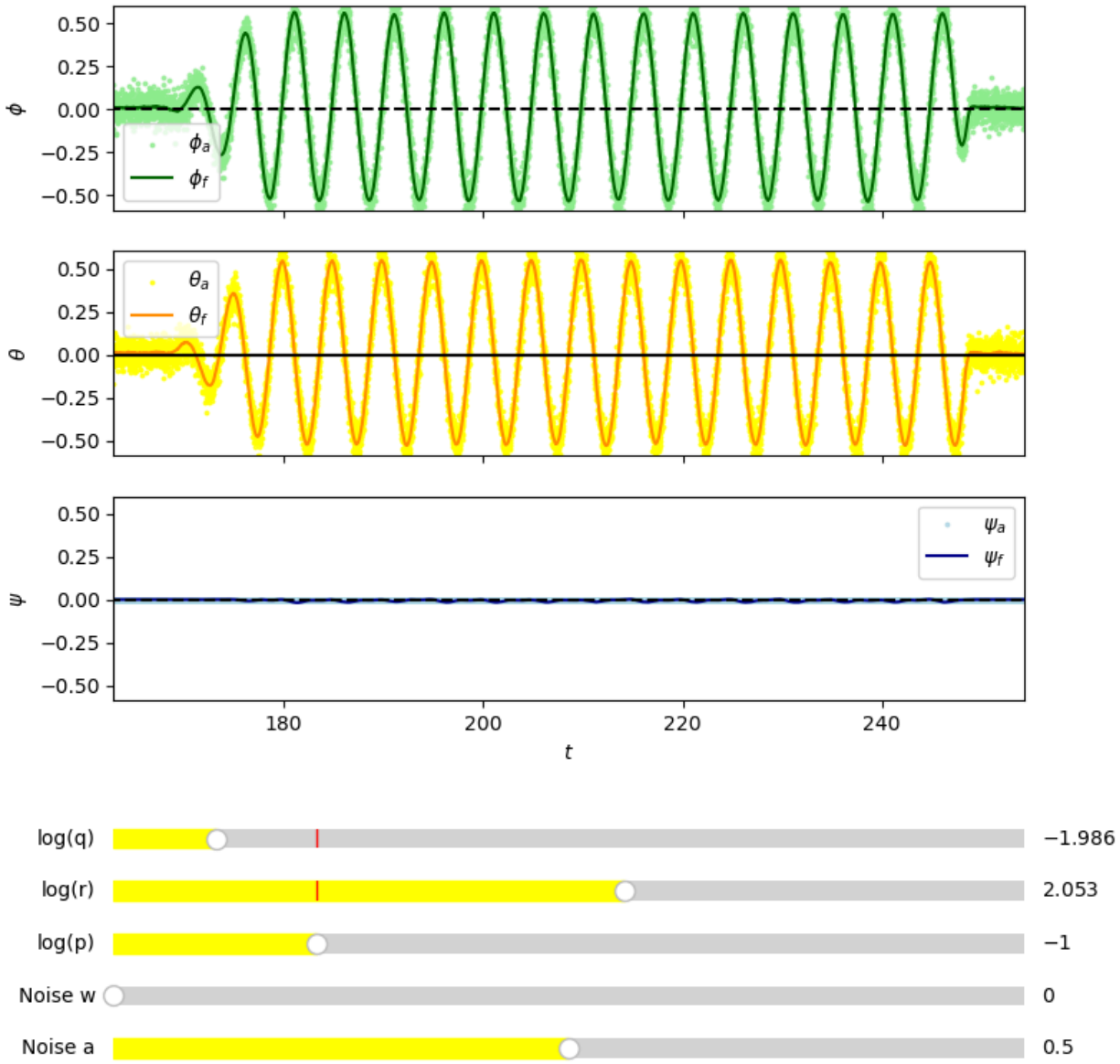

Fig. 24 Zoomed in Fig. 23. ϕ_a represents ϕ measured from accelerometer data. ϕ_f represents ϕ from the kalman filter with sensor fusion. View in Github

Fig. 23 and Fig. 24 show the kalman filter has produced a very good fit. The filtered signal appears both noise, drift and delay free.

5.4 Summary

Lets compare all three filters side by side.

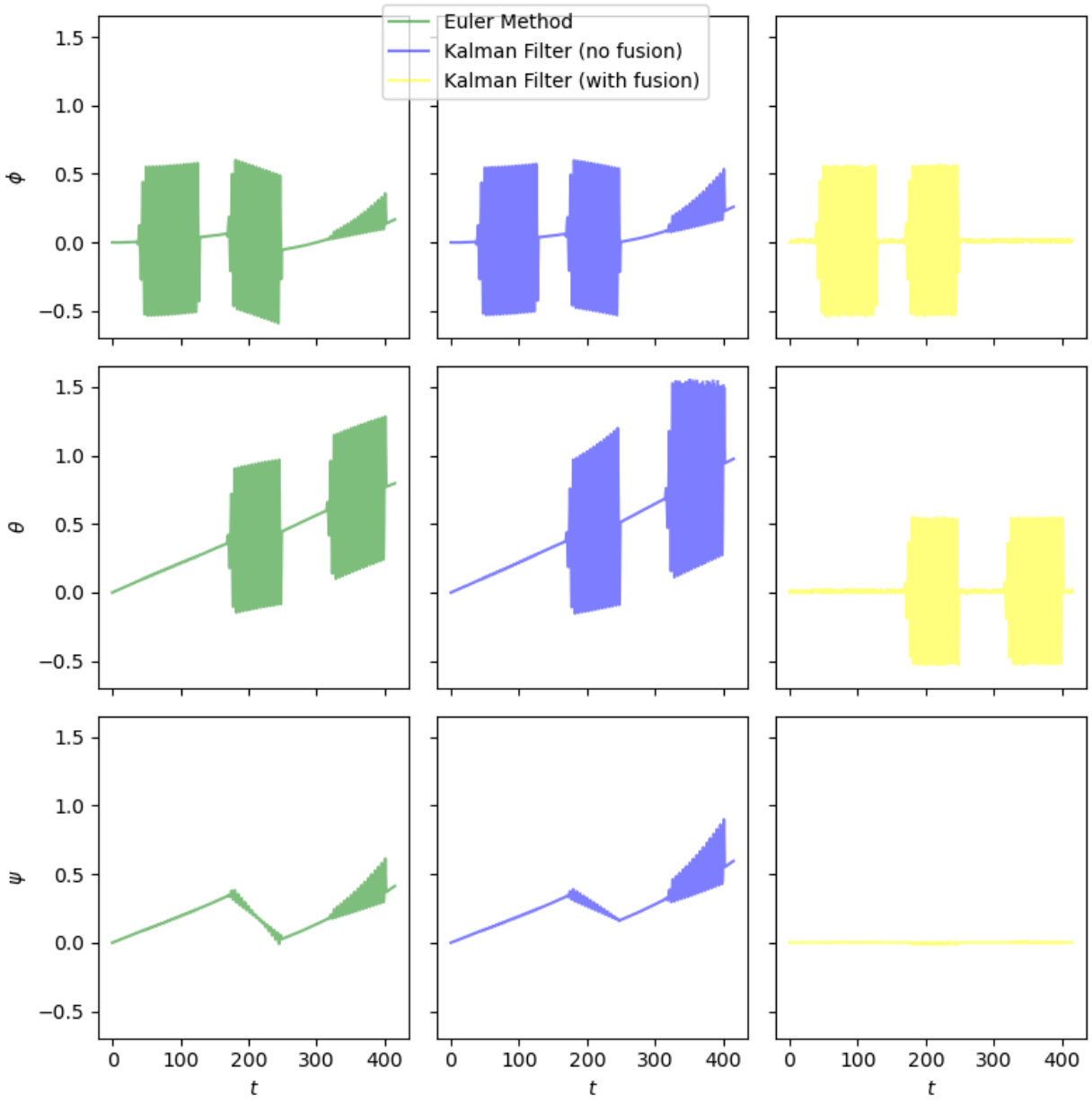

Fig. 25 All three filters side by side. View in Github

Figure Fig. 25 shows the only the kalman filter (with fusion) is accurate for determining attitude in the longrun. Even though the error from numerical integration is small if it isn't regularly corrected for will be subject to integration drift.

The Kalman filter produced an excellent fit in this case as the prediction (from gyroscope) and measurement (from accelerometer) were complimentary to each other. The gyroscope was less susceptible to noise but was susceptible to drift whereas the accelerometer was more susceptible to noise and less susceptible to drift. Sensor fusion gets the best of both worlds.

Idea

Rewrite the part of the code that carries out kalman filter calculations and determines A in C++ as the programme runs really slowly.

6 Example : Position using GPS and accelerometer data

This section improves on the the velocity from position example example by using sensor fusion. While the system using only position data (theoretically measured using GPS) works well for predicting the position, its not so good at predicting the velocity. The current model (35) represents an oversimplification as it assumes no acceleration (the acceleration doesn't change between steps) which means that the velocity has to be corrected for by the measurements which is what causes lag. The model could be improved using real world accelerometer data which can be integrated to find velocity and position. There are other reasons for including the accelerometer data for example when GPS isn't available due to some form of blocking e.g. being in a tunnel, the device can still roughly determine its position.

6.1 Model

Starting with the 1D case the new model is built on (35) with an additional 2nd order term:

$$\begin{aligned} s_{k+1} &\approx s_k + v_k \Delta t + \frac{1}{2} a_k \Delta t^2 \\ v_{k+1} &\approx v_k + a_k \Delta t \end{aligned} \tag{50}$$

The parameters from the velocity from position model remain the same, except for the model A:

- $\hat{x}$ is the column vector of position and velocity
- z is the measurment of position from the GPS
- $H = [1 \quad 0]$
- $Q = \sigma_a^2 \begin{bmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & \Delta t^2 \end{bmatrix}$ Where σ_a will be the standard deviation in the acceleration measurements.
- $R = \sigma_s^2$ Where σ_s will be the standard deviation in the position measurements. A needs to be a 2×2 matrix, since it has the same number of rows and columns as the number of entries in $\hat{x}$, but this isn't possible since (50) contains 3 terms.

If μ_x is t
its cova
transfor
[BAOOI2

Extended Kalman Filters

Its not possible to write the prediction stage of the kalman filter as a linear transformation. The extended kalman filter predicts the next state using:

$$\hat{x}_{k+1}^{-} = A\hat{x}_k + Bu_k \tag{51}$$

Where u_k is the forcing function and B is its associated control matrix where u_k is the forcing function and B is its associated control matrix.

$$P_{k+1}^{-} = A_k P_k A_k^{\prime} + B_k R^u B_k^{\prime} + Q \tag{52}$$

Where R^u is the associated error covariance matrix for u_k. Equation (52) is the updated form of (31) with the final term $B_k R^u B_k^T$ corresponding to the covariance update for R^u. R^u becomes one of our kalman parameters when using the extended kalman filter.

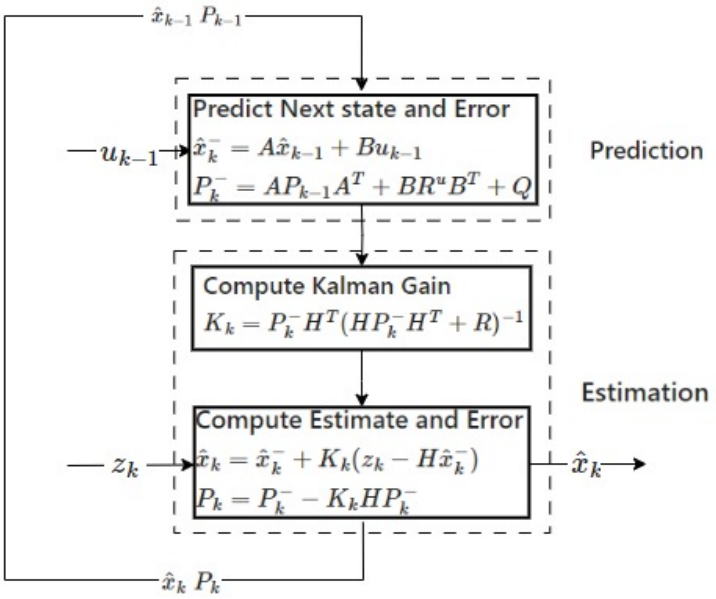

Fig. 26 Block diagram for the extended kalman filter. The estimation phase is equivalent to Fig. 7 but the prediction stage has been updated.

So (50) was rewritten in the form of (52) to determine u_k and B.

$$\begin{bmatrix} s \\ v \end{bmatrix}_{k+1}^{-} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ v \end{bmatrix}_k + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} a_k.$$

Which gives $u_k = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix}$ and $u_k = a_k$. The tuning parameter will be $R^u = \sigma_a^{\prime 2}$ determine by tuning.

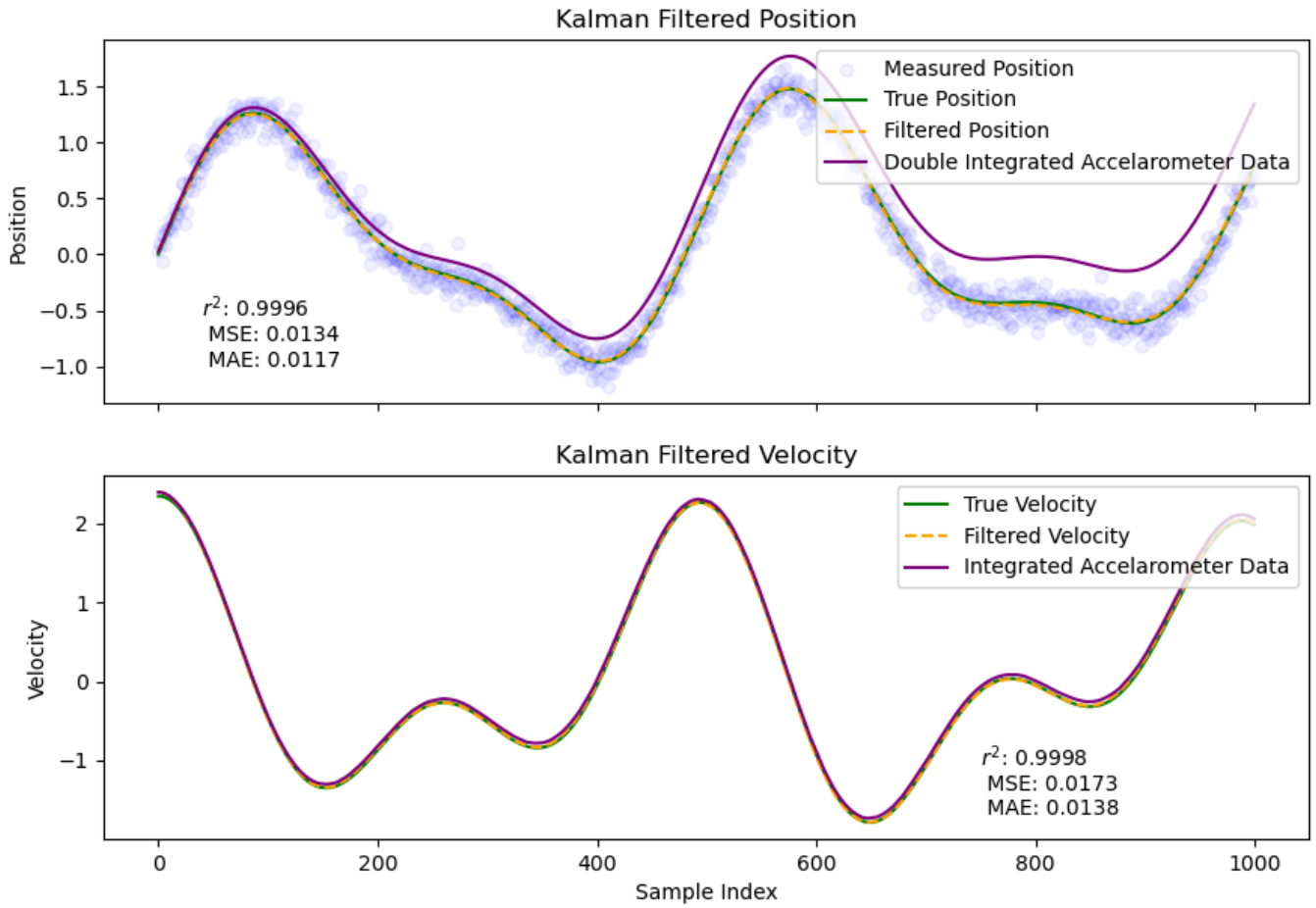

Fig. 27 Velocity and position as a function of time plotted for the extended kalman filter using the same parameters in Fig. 15, tuned by eye. View in Github

Compared to Fig. 15 the extended kalman filter with acceleration measurements gives a better fit for position and a significantly better fit for velocity, helped by the significantly better model. Even without measurement corrections the accelerometer gives a surprisingly good fit although there is a tiny bit of drift visible at the end. However the drift is significantly larger when integrated twice.

6.2 Smartphone experiment

> **Warning**
>
> Incomplete section. It is left here as a placeholder for future work. The experiment would have involved using kalman filters to determine the real world position and velocity of a smartphone using accelerometer and GPS data. This would have needed to consider the GPS and accelerometer having different sampling rates and noise characteristics.

7 Experiment: Attitude Using a 6/9 axis IMU

This section explores examples similar to section 5 but using real data. Initially using a gyroscope and an accelerometer, a 6 axis IMU, and later using a magnetometer as well, a 9 axis IMU. The 6 axis example uses

the MPU6050 with an arduino controller and the 9 axis example used the mobile phone sensors with the sensor logger app.

Implementation

The code for this section can be found here. The main file is `Analysis.py` which contains `AnalyseMPU` class which is the parent class for `AnalysePhone` and `AnalyseTest`, these read data from the MPU6050, sensor logger and test data (from section 5) respectively.

Raw Data from MPU6050

Each file with measurements came with its own calibration file, which was used to determine the offset in each direction. Different tests were performed with the MPU6050, the data for these are contained in the folders:

- `YawPitchRoll`: Involves fast oscillations in the yaw direction followed by a short break, then oscillations in the roll direction followed by a short break, then oscillations in the pitch direction.
- `FullYaw`: Contains data where the MPU6050 was rotated in a full circle in the yaw direction and then rotated back in the opposite direction.
- `FullPitch`: Contains data where the MPU6050 was rotated in a full circle in the pitch direction and then rotated back in the opposite direction.

> **Warning**
>
> Currently the Kalman filter will only correctly fit `YawPitchRoll` data. When the sensor was rotated beyond 180^o in the yaw and roll directions or beyond 90^o in the pitch direction the Kalman filter will need to re-normalize the attitude, which made it difficult to fit the Kalman filter. This wasn't a problem with `YawPitchRoll` data as no full rotations were performed.

Raw Data From Sensor Logger

The data collected can be found here. Each folder contains a different test and contains the following files:

- *In all cases time was measured in seconds.*
- `Gravity.csv`: Contains the acceleration readings without the acceleration due to gravity removed. *The x, y and z components are measured in* ms^{-2}.
- `Magnetometer.csv`: Contains the magnetometer readings in the phone's frame of reference. *The x, y and z components are measured in* μT.
- `Gyroscope.csv`: Contains the gyroscope readings in the phone's frame of reference. *x, y and z are measured in radians* $rads^{-1}$
- `Orientation.csv`: Contains the orientation of the phone in yaw-pitch-roll measured in radians. We will refer to this as the true orientation as its the orientation calculated by the phone. All other files are not used in this analysis.

Tests were performed and stored in the folders:

- `PitchRollCalibration`: Fast movement in the pitch direction followed by a short break, then motion in a combination of the pitch and roll directions followed by a short break, then motion in just the roll direction, this was the same test as in section 5.
- `YawRollPitchCalibration`: Contains data for a calibration test where the phone was moved in each direction alone followed by a short pause.
- `FullYaw`: Contains data where the phone was rotated in a full circle in the yaw direction and then rotated back in the opposite direction.
- `FullPitch`: Contains data where the phone was rotated in a full circle in the pitch direction and then rotated back in the opposite direction.

Programme structure

The programme is structured with three files:

- `AdvKalman.py`: Contains the Kalman filter implementation and handles the conversion between quaternions and Euler angles.
- `OrientationKalman.py`: Contains the `run` function which runs the Kalman filter on the data and returns the filtered signal, Euler angles from the accelerometer and gyroscope, and the magnetometer readings. It acts as an interface between Analysis.py and AdvKalman.py.
- `Analysis.py`: Contains the `AnalysePhone` class which is used to read in the data, run the Kalman filter and plot the results.

Theory, 9 axis

In section 5 gyroscope data and accelerometer data were fused to determine the attitude of the phone. However the accelerometer on its own was unable to measure the yaw direction. When phone is lying flat in the xy plane z^{ψ}, can be determined easily:

$$z^{\psi} = \arctan\left(\frac{m_y}{m_x}\right) \tag{53}$$

Where m_x and m_y are the x and y components of the magnetometer reading. The magnetometer reading is in the phone's frame of reference this is normally not the xy plane so the accelerometer data to is required to write the corrected magnetometer readings m_x' and m_y'.

$$\begin{aligned} m_x' &= m_x \cos(z^{\theta}) + m_y \sin(z^{\theta}) \sin(z^{\phi}) + m_z \sin(z^{\theta}) \cos(z^{\phi}) \\ m_y' &= m_y \cos(z^{\theta}) - m_x \sin(z^{\theta}) \sin(z^{\phi}) + m_z \sin(z^{\theta}) \cos(z^{\phi}) \end{aligned} \tag{54}$$

Where z^{ϕ} and z^{θ} are the roll and pitch angles calculated using the accelerometer.

Then as before z^{ψ} and z^{ϕ} were calculated using the accelerometer and were used to correct z^{ψ} when the magnetometer isn't lying flat in the xy plane. To summarize the update measurement will be formed from the accelerometer, for pitch and roll and the magnetometer for yaw. The prediction step will use measurements from the gyroscope in the same way as section 5.

Results

Fast Yaw Pitch Roll

These examples involved fast oscillations in the yaw directions followed by a short break, then oscillations in the roll direction followed by a short break, then oscillations in the pitch direction.

> **Note**
>
> Here yaw-pitch-roll is normalized as follows: yaw is in the range $[-\pi, \pi]$, pitch is in the range $[-\pi/2, \pi/2]$ and roll is in the range $[-\pi, \pi]$.

MPU6050 6-axis IMU

So that the correction was normalized the correction in the yaw direction was set to always be zero, the pitch and roll directions came from accelerometer measurements. The prediction came from gyroscope measurements.

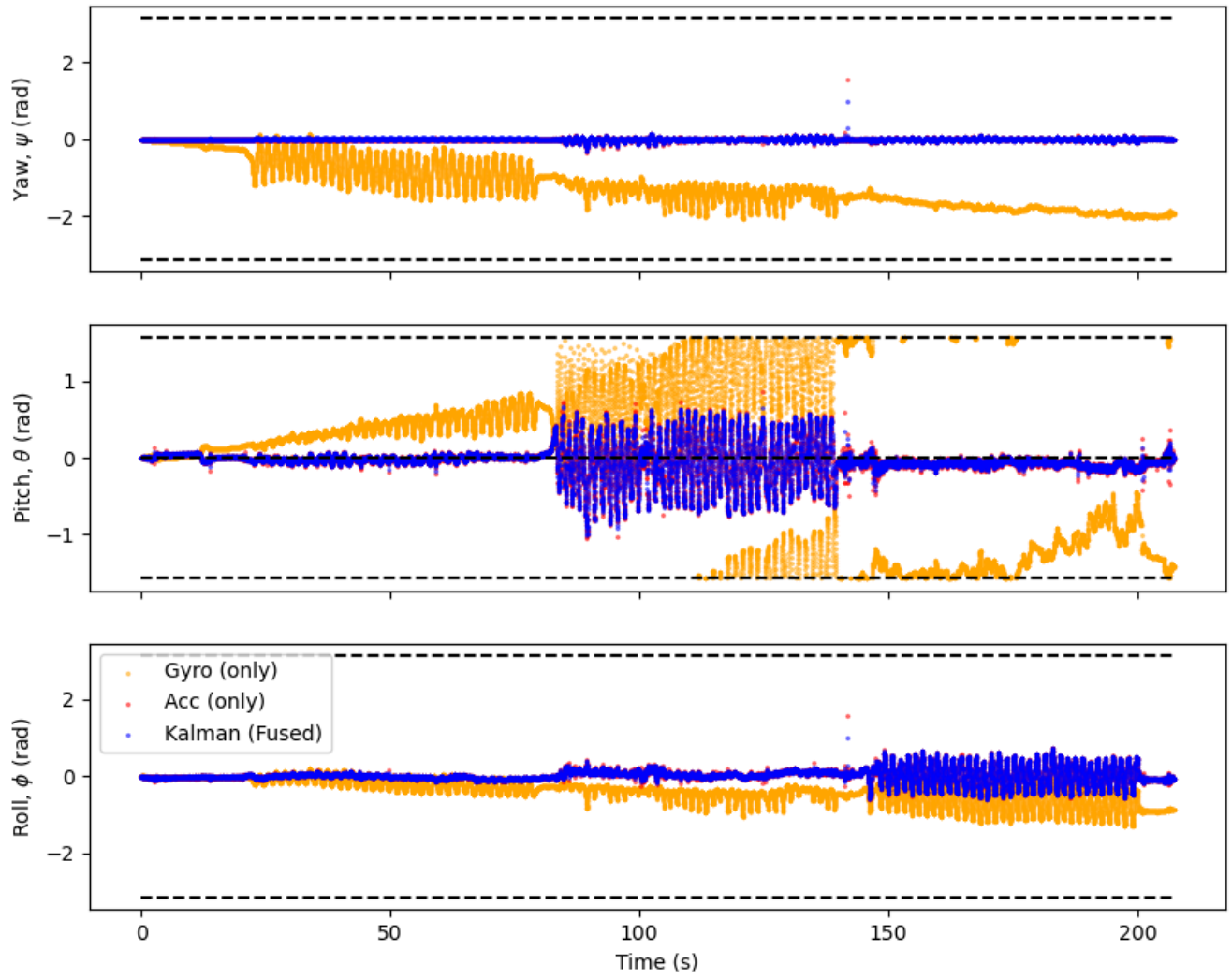

Fig. 28 Fast oscillations in the yaw, roll and pitch directions. The orange line is the measurements from integrating the angular velocities from the gyro and red line is the measurements from the accelerometer. The blue line is the Kalman filtered attitude which fuses data from the gyroscope and accelerometer/magnetometer.

The drift is clearly visible from the gyro. But the accelerometer measurements seem to be much more accurate. However it is unclear if Kalman filter improves on the accelerometer measurements with the current Q and R.

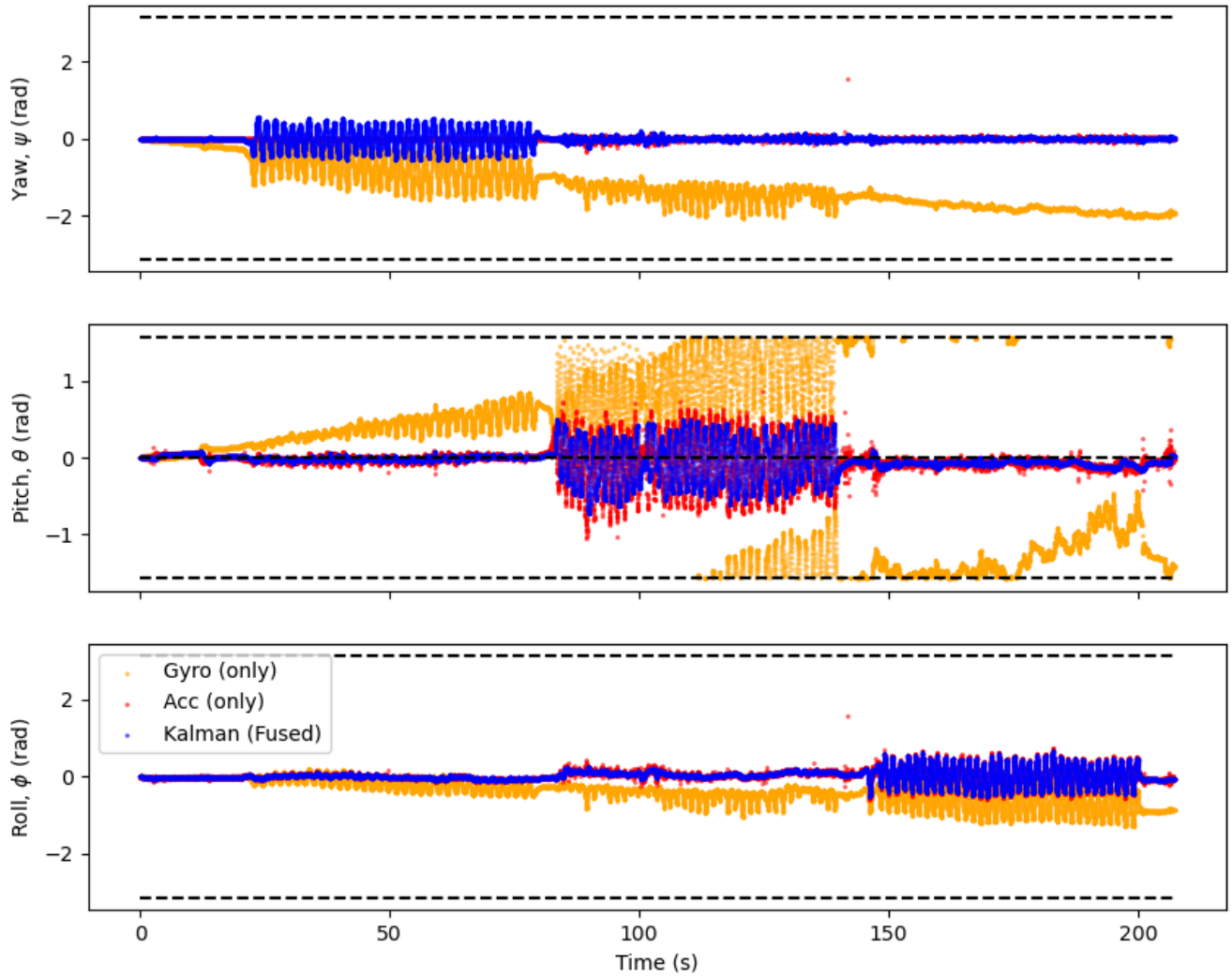

Fig. 29 The same data as Fig. 28 but with the Kalman filter tuned, with larger R and smaller Q, increasing the weighting on the prediction, the gyro data.

Here the Kalman filter has a much better shape for the yaw direction since there is a greater weighting on the prediction meaning oscillations from the gyro are much more pronounced in the Kalman filter. The Kalman filter also still filters out the drift from the gyro data making it a very accurate fit.

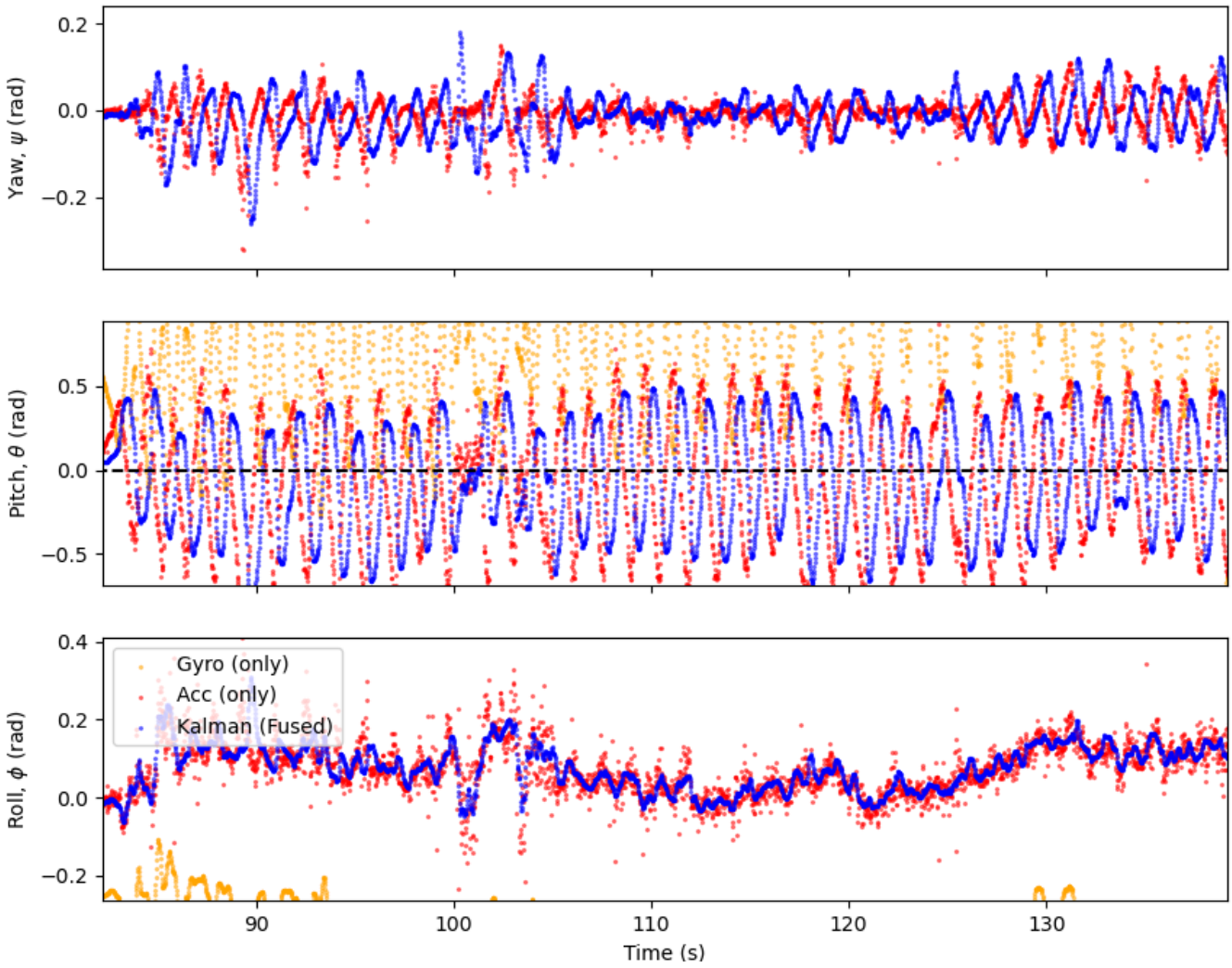

Fig. 30 Zoomed in on the pitch oscillations in Fig. 28.

There is a problem in the oscillations in all directions are now delayed as the gyroscope is slower to to changes than the accelerometer, hence by putting more faith in the prediction although the fit more accurately resembles the shape of the true data it is slightly delayed.

The Kalman filter is able to filter out some of the noise from the accelerometer data. The accelerometer data is typically noisier than the gyroscope data, so putting more emphasis on the gyroscope will help to filter out noise.

Phone, 9-axis IMU

Introducing magnetometer measurements means there is 2 measurements for each axis, reducing the total drift.

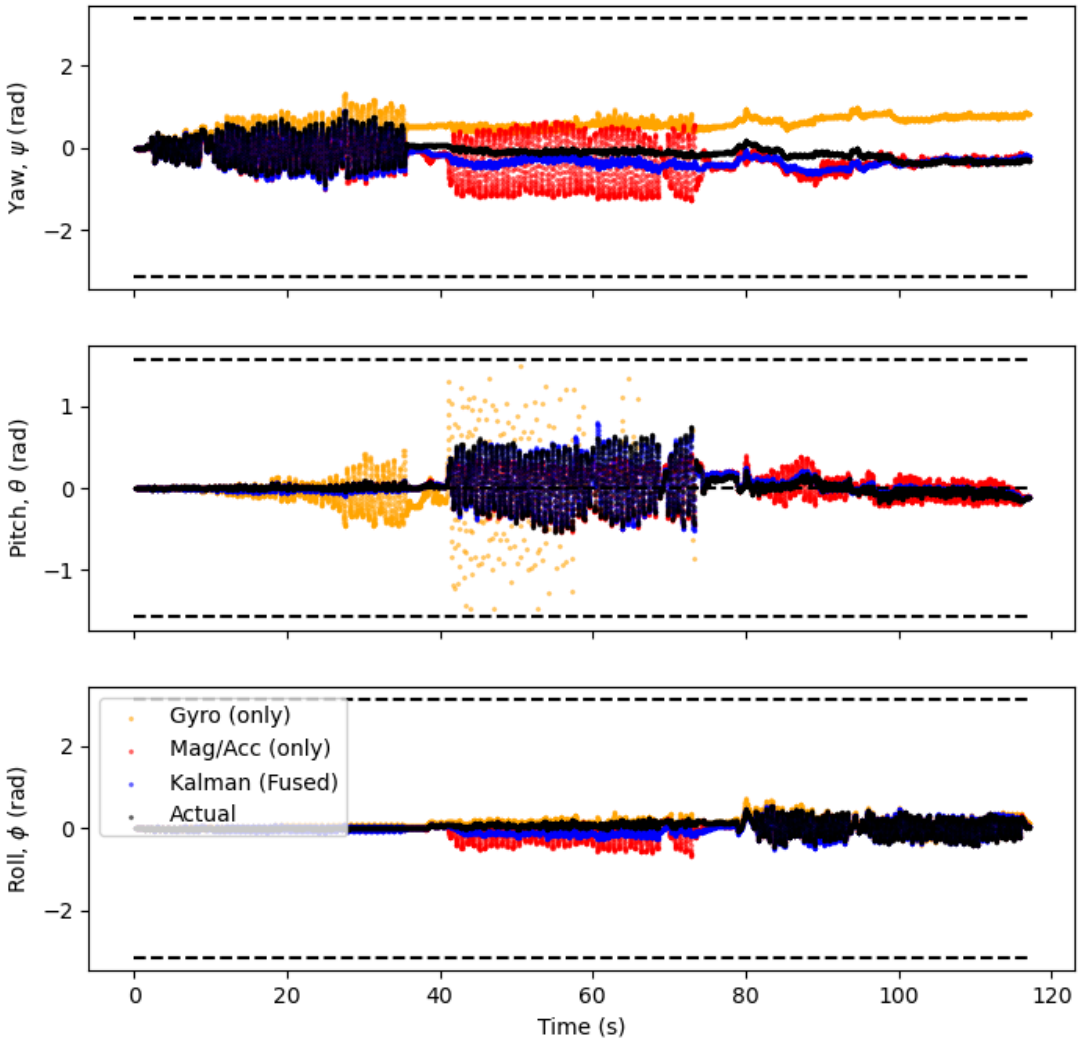

Fig. 31 Fast oscillations in the yaw, roll and pitch directions. The orange line is the measurements from integrating the angular velocities from the gyro and red line is the measurements from the magnetometer and accelerometer. The blue line is the Kalman filtered attitude which fuses data from the gyroscope and accelerometer/magnetometer. The black line is the phones own attitude measurements.

There is significant drift in the measurements from the gyro alone the end measurements from the gyro alone has drifted by approximately 60^{o} from the true value where as the Kalman filter and the magnetometer/accelerometer data are much better fits. Also the amplitude of the measurements oscillations in the roll direction is much smaller than the yaw and pitch directions. This was because its because of the way the phone had to be held.

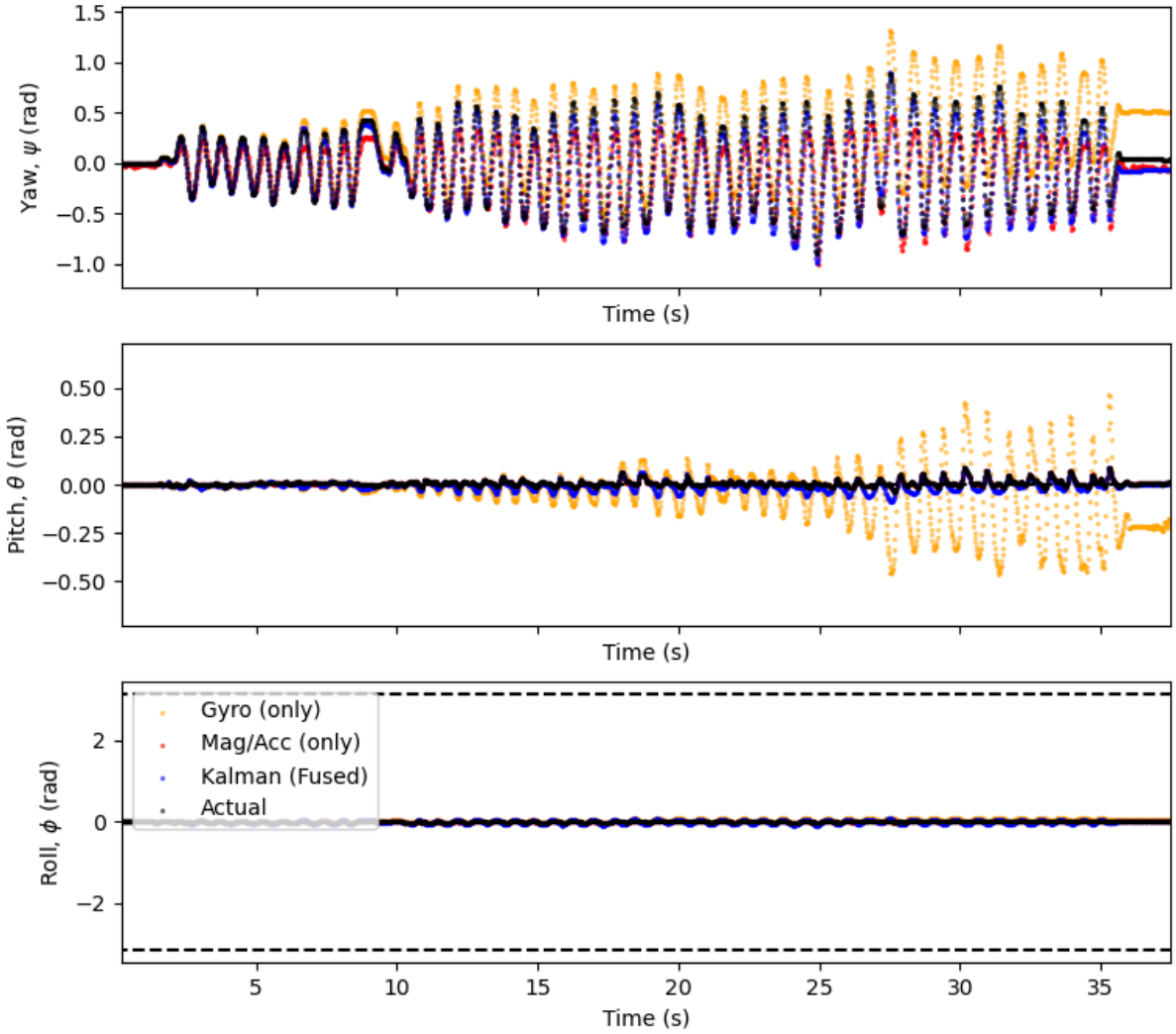

Fig. 32 Zoomed in on the yaw oscillations in Fig. 31.

Here the Kalman filter is able to correct for both the drift from the gyro and noise from the magnetometer making it a very good fit for yaw. The is interference with the gyroscope in the pitch direction but the Kalman filter successfully filters this out.

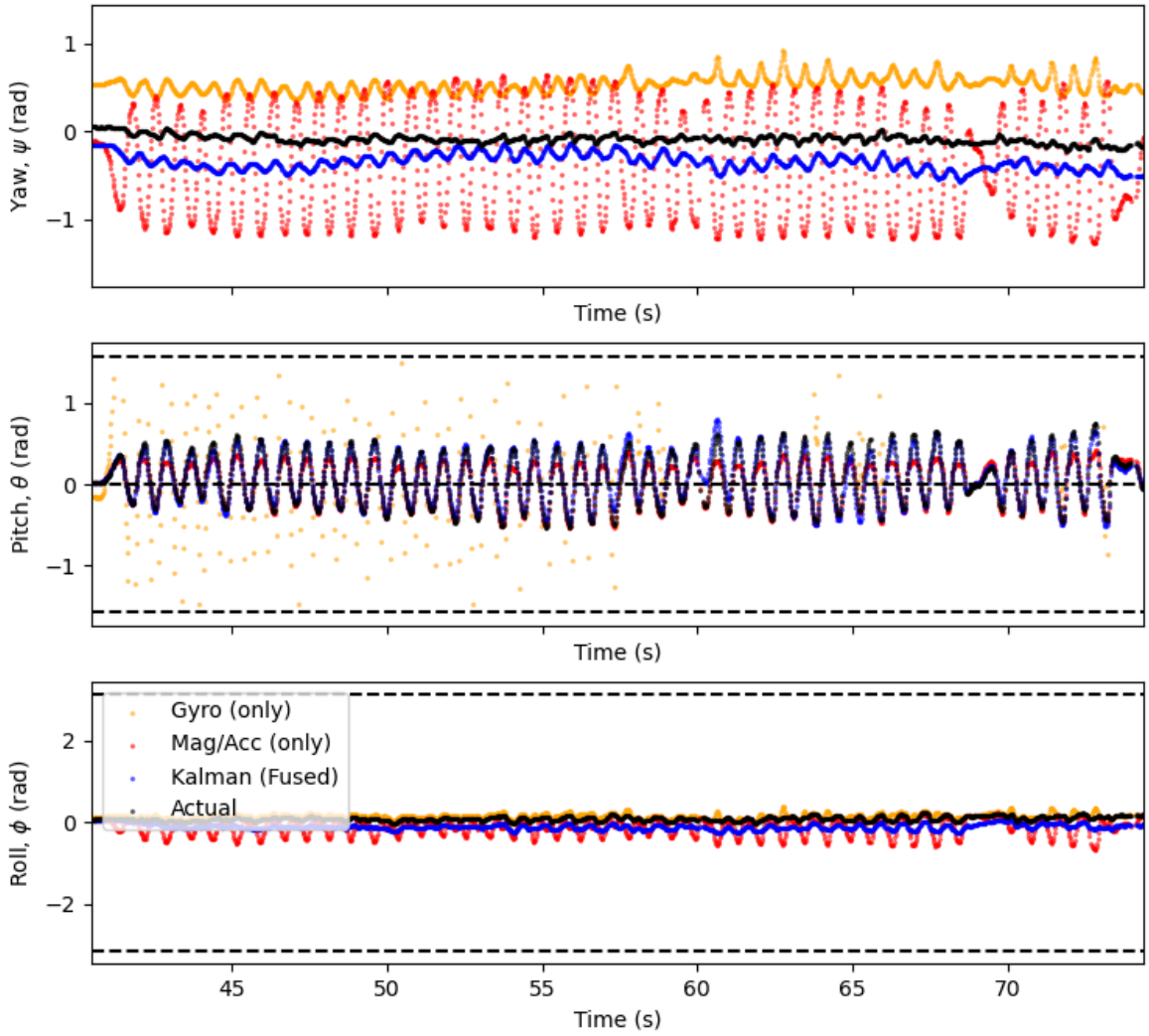

Fig. 33 Zoomed in on the pitch oscillations in Fig. 31.

There is interference in the magnetometer/gyroscope measurements but the Kalman filter filters this out successfully but this does cause drift in the Kalman filter measurements. The gyroscope measurements for Pitch are incredibly noisy possibly due to oversensitivity of the gyroscope in the phone, but the Kalman filter remains a good fit. The Roll direction remains fairly stable.

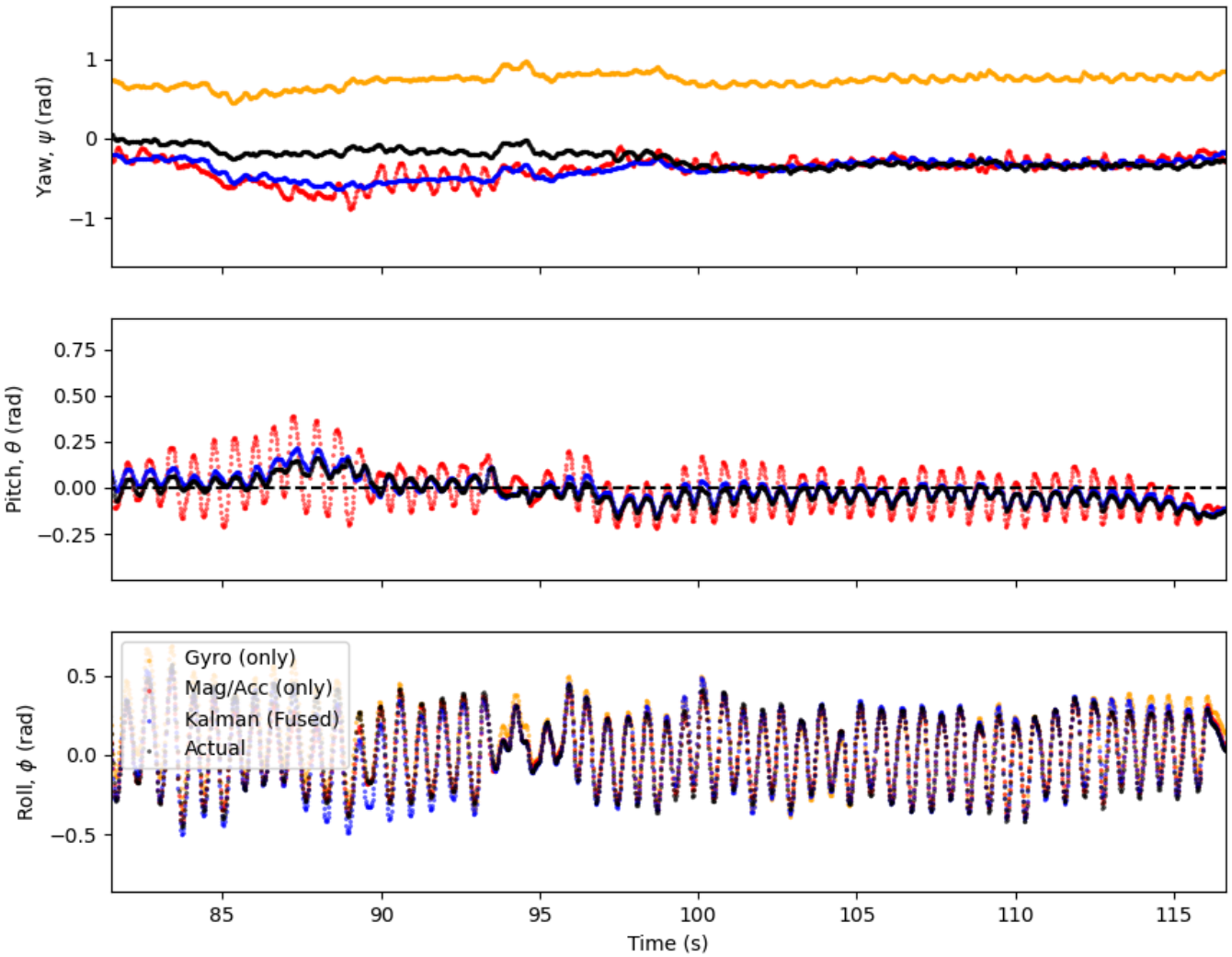

Fig. 34 Zoomed in on the roll oscillations in Fig. 31.

For the yaw direction both sets of instruments are indicating the correct shape but there is drift especially in the gyro. There is interference in the pitch direction, again the Kalman filter does a good job of filtering this out.

Full Rotation, Yaw

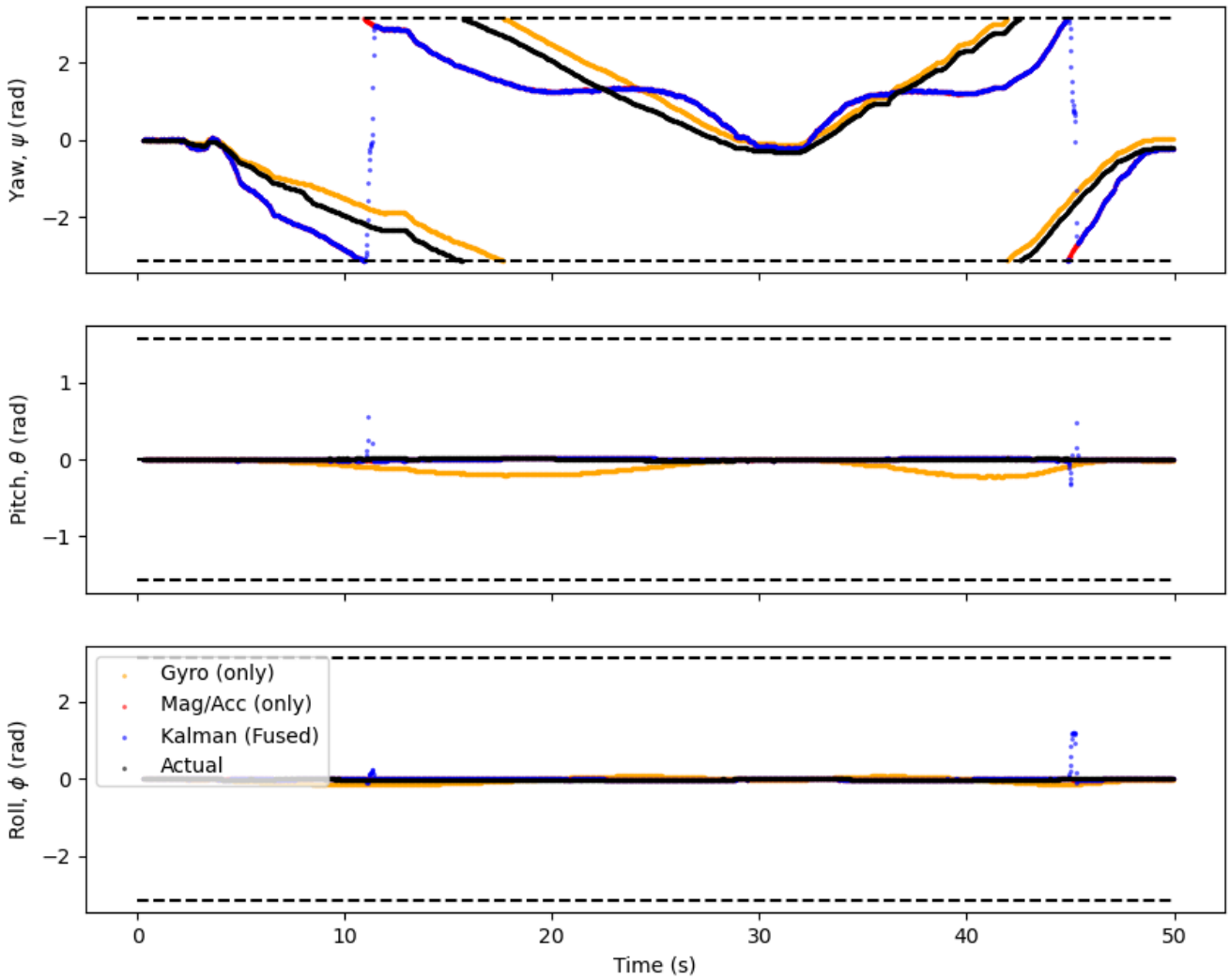

Fig. 35 The phone was rotated in a full circle in the yaw direction and then rotated back in the opposite direction. The orange line is the measurements from integrating the angular velocities from the gyro and red line is the measurements from the magnetometer and accelerometer. The blue line is the Kalman filtered attitude which fuses data from the gyroscope and accelerometer/magnetometer. The black line is the phones own attitude measurements.

The Kalman filter is able to reduce noise in the magnetometer measurements but the gyroscope seems to provide the best fit here. This could be because the magnetometer is overly sensitive. However it's plausible that the phone reliable more on its gyroscope for measurements of orientation since magnetometer data isn't as reliable due to additional magnetic fields meaning it can be poorly calibrated. Whereas a gyroscope with a small amount of drift will always work.

Note

I didn't have enough time during the placement to test the Kalman filter for full rotations.

8 Experiment: Quantitative Comparison of Kalman Filter

Performance

Using two sensors (magnetometer and gyroscope), four filters (a high pass, a low pass, a polynomial fitting and the kalman filter) were used to calculate estimates for the yaw angle, ψ, of a mobile phone. The results were compared to the data from the phones in built filter. The experiment involved two tests designed to see how the filters perform over a single frequency then a range of frequencies. In the first part of the experiment the phone oscillated with high frequency which the parameters were tuned, to test how the filters performed at a single frequency. The second part of the experiment the phone completed a full rotation at a low frequency without the parameters being re-tuned, to see how the filters responded when not tuned to a specific frequency.

8.1 Implementation

Data was recorded using the sensor logger app allowing the phone to function as a 9-axis IMU. Only ψ was measured so only gyroscope and magnetometer data were necessary. The correction measurement was calculated using the magnetometer data m_x and m_y. m_z wasn't required as the IMU was assumed to be in the plane perpendicular to the gravity vector, so didn't need to be corrected. ψ was determined using (53).

The gyroscope data was integrated using the euler method:

$$\psi_k \approx \psi_{k-1} + \omega_k \Delta t \tag{55}$$

This time ω_k is the gyroscope measurement of angular velocity at time k in the z direction. This formed the prediction measurement. The form required by the Kalman filter is $\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$. So $A = 1$ and $B = dt$ and $u_k = \omega_k$. The state to measurement matrix $H = 1$ since z_k and $\hat{x}_k$ are both the yaw angle.

The code for this section can be found in github. The programme comprises of one file `Main.py`. Each of the filters is written as a separate function and can be analyzed using the `AnalysePhone` class, which contains methods to generate interactive plots.

8.2 Filters

The following filters were compared as well as unfiltered data from the gyroscope and the magnetometer:

Kalman Filter (KF): A standard Kalman filter which **fused** data from the magnetometer with gyroscope data with a constant process noise covariance and measurement noise covariance. The tuning parameters were:

- Q: Process noise covariance.
- R^m: Measurement noise covariance for the magnetometer measurements.
- R^g: Measurement noise covariance for the gyroscope measurements.

Exponential Moving Average Low Pass Filter (EMAHPF): Was applied to the **magnetometer** data where the Where α^{HP} is the tuning parameter, to reduce noise.

Exponential Moving Average High Pass Filter (EMAHPF): Was applied to **gyroscope data** to cutout integration drift (low frequency). α^{LP} was the tuning parameter.

Savitzky-Golay Filter (SGF): Was applied to the **magnetometer data** to help mitigate random noise. It has the following parameters:

- W: The length of the filter window.
- O: The order of the polynomial used to fit the samples. The SGF works by fitting a polynomial of order O to a window of length W. The center of the window is the fitted value.

8.3 Results

High frequency oscillation test

The phone was initially held in a fixed position for around 5 seconds then oscillated at a high frequency for around 20 s, each of the filters was tuned to optimise the fit.

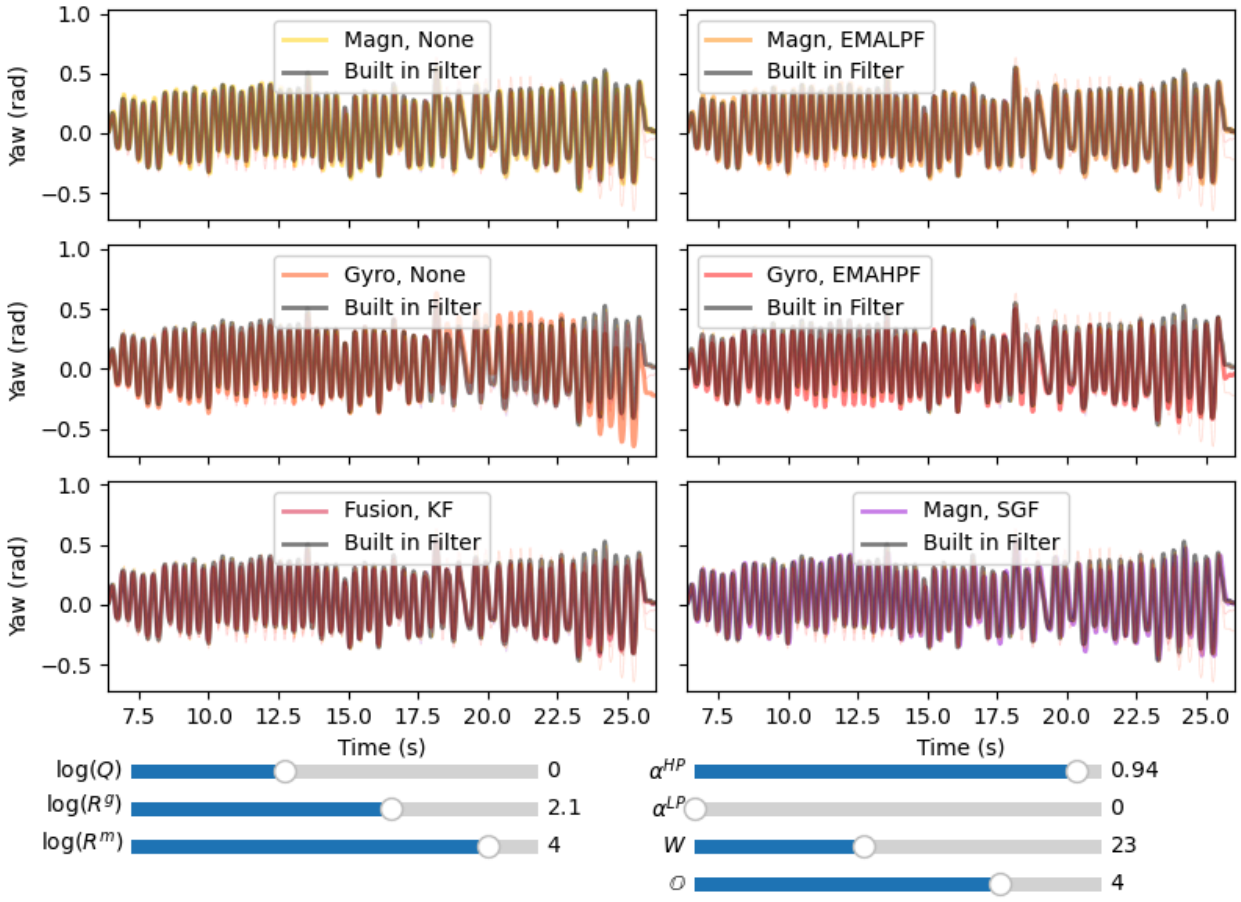

Fig. 36 Shows the signal used and gives an overview of the results and compares the phones own filters against the filters described above. Tuning parameters were set below.

Fig. 36 shows integration drift is visible in the unfiltered gyroscope data. But all other filters seem to be a good fit from a distance.

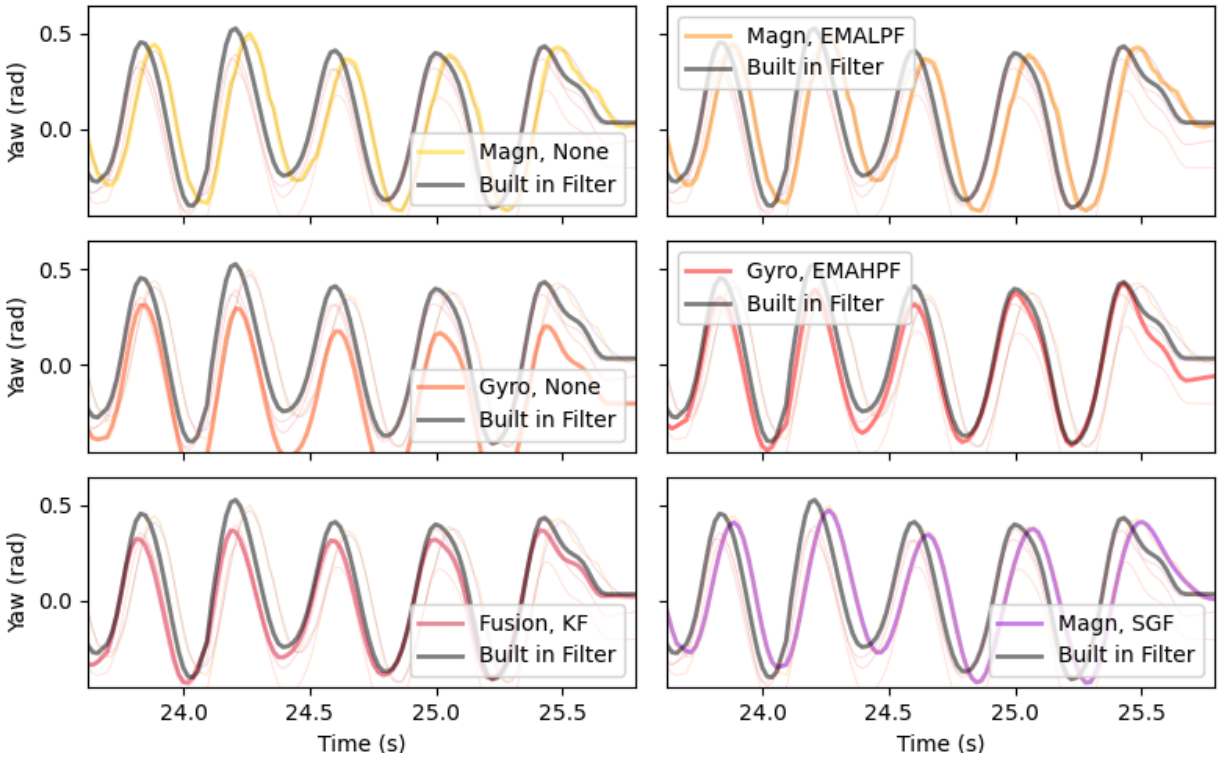

Fig. 37 Zoomed in view of the final few oscillations of the high frequency oscillation experiment with a visually optimized fit.

Fig. 37 shows the there is integration drift from the unfiltered gyroscope data but not with the EMAHPF since the drift is low-frequency. The magnetometer measurments are lagging behind the true signal which also effects the EMALPF and SGF. Surprisingly the Kalman filter isn't lagged as it relies heavily on the gyroscope measurements. The Kalman filter assumes a large amount of noise in the magnetometer, R_m, two orders of magnitude larger than R_g. There is also a small amount of linear process noise.

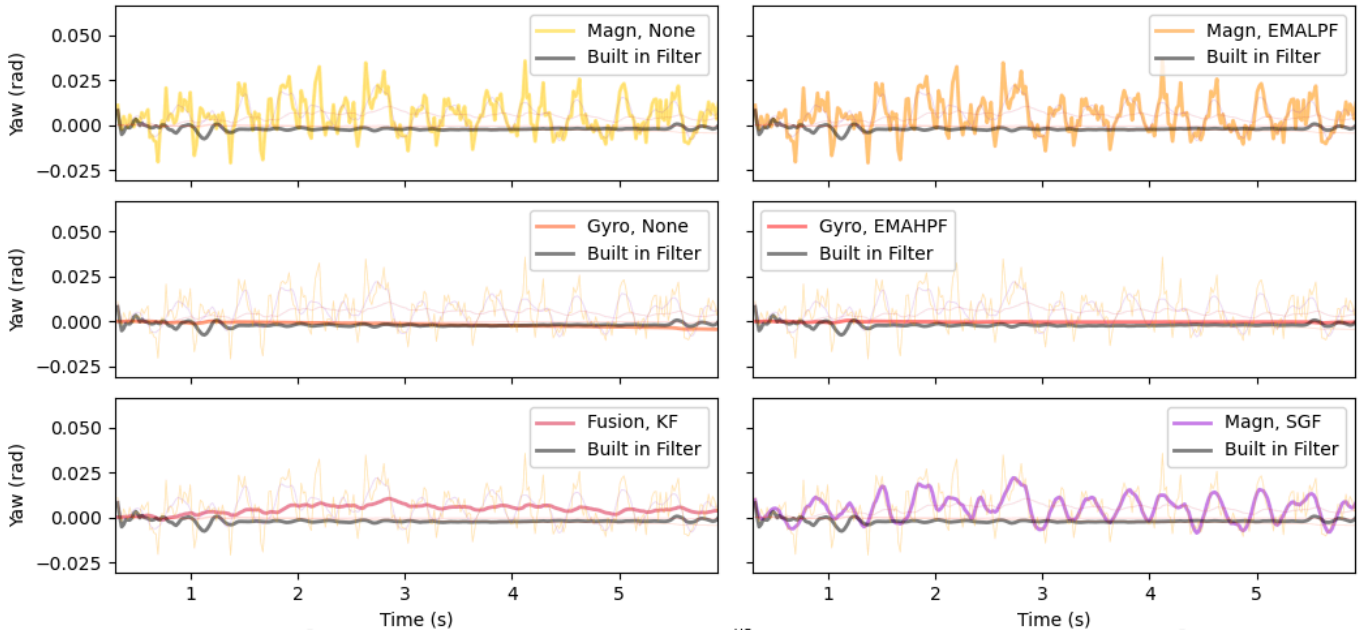

Fig. 38 Zoomed in view of the flat proportion of the graph, at the beginning.

Fig. 38 shows the noisiest sensor is the magnetometer. The SGF reduces some of this noise. The EMALPF actually made the fit worse since the noise from the magnetometer was significantly less significant than its lag, which applying a low-pass filter actually worsened.

> **Note**
>
> R^2 measures the pattern-matching ability of the filter and is sensitive to phase shifts. MSE (mean squared error) measures absolute accuracy and heavily penalizes large errors due to squaring.

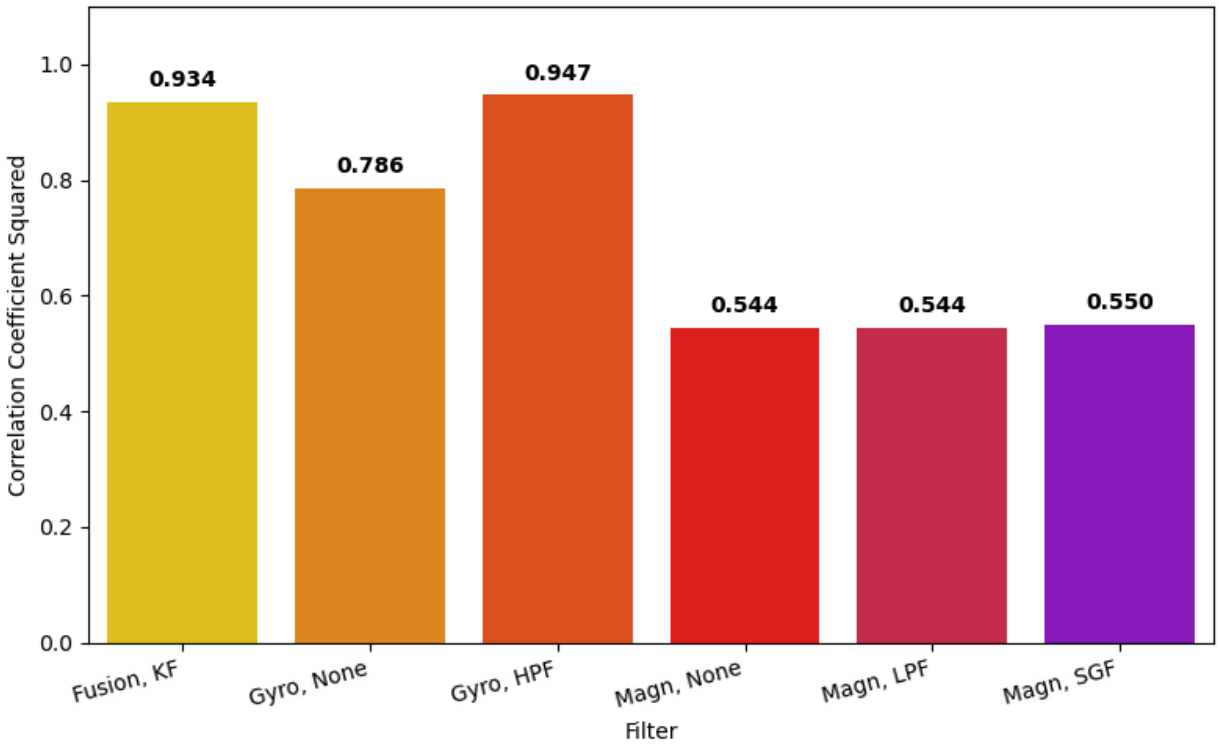

Fig. 39 Correlation coefficient squared, R^2 for each of the filters against the phones built in filter.

None of the measurements using the magnetometer perform well. The best of the magnetometer measurements is the SGF, as it removes a small amount of noise and doesn't cause significant lag. Fig. 39 suggests that the EMALPF fits the shape the best on the gyro data than the fused KF did, likely a result of lagging magnetometer data which wasn't corrected for.

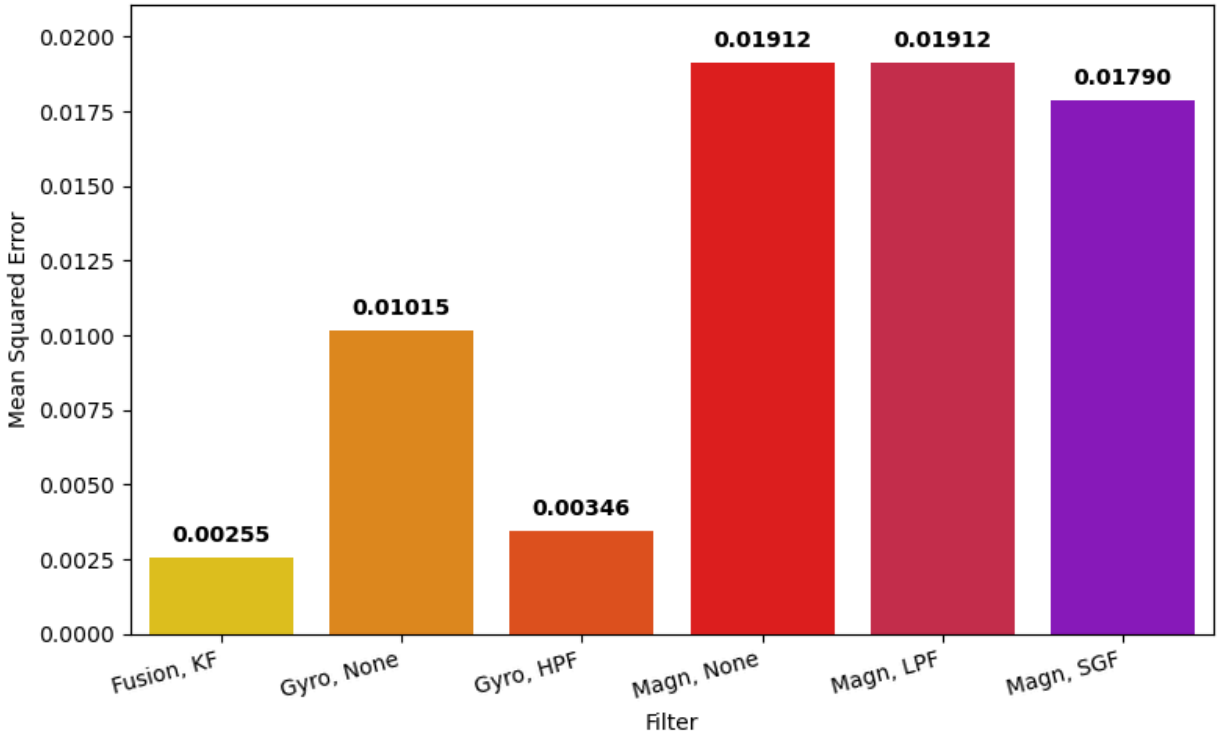

Fig. 40 MSE for each of the filters against the phones builtin filter for the first test.

Conversely Fig. 40 suggests that KF has a more accurate fit. This suggests that EMALPF fits the shape of the data better than KF but KF is on average more accurate.

Low frequency rotations test

The phone was held still for around 8s and then was rotated a full circle over the space of the next 15s. The tuning parameters for this experiment are those given in Fig. 36.

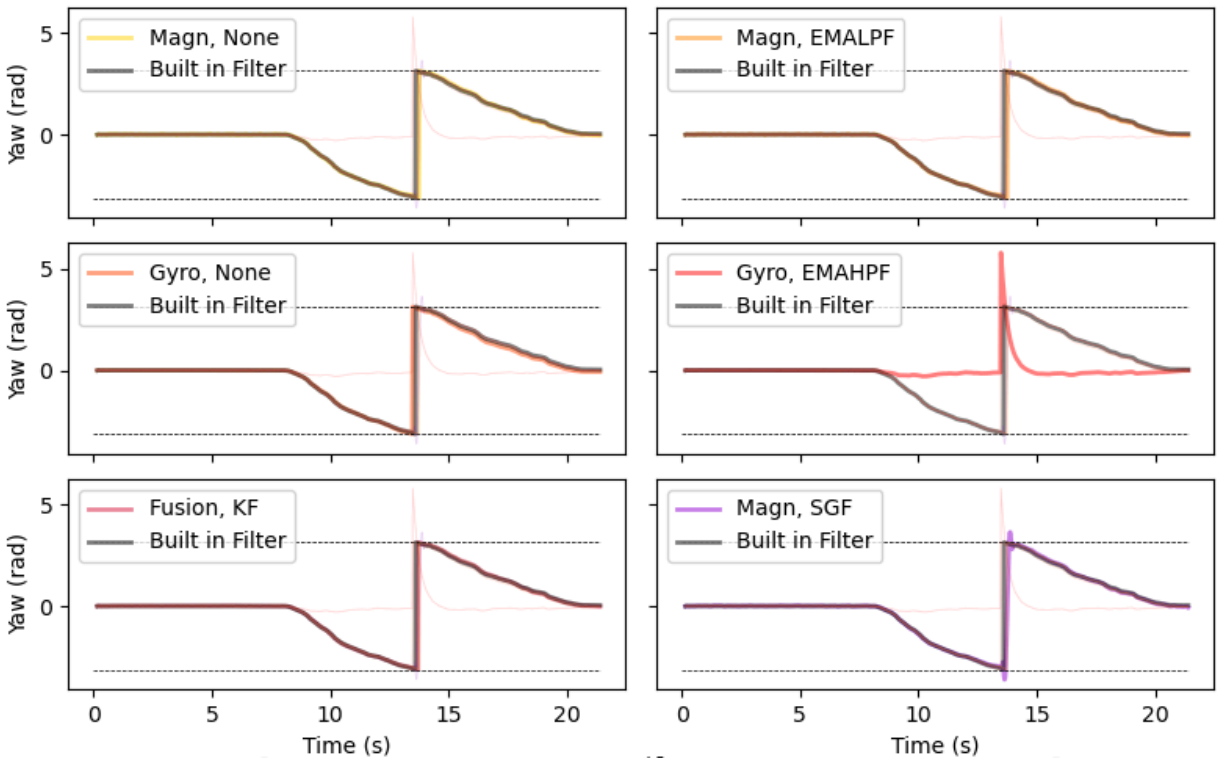

Fig. 41 Comparison of the phones own filters against the filters described above. With tuning parameters the same as Fig. 36.

Fig. 41 shows that all filters are a good fit for the data except the EMAHPF which filters out the low frequencies even if they are not related to noise, in this case the slow rotation.

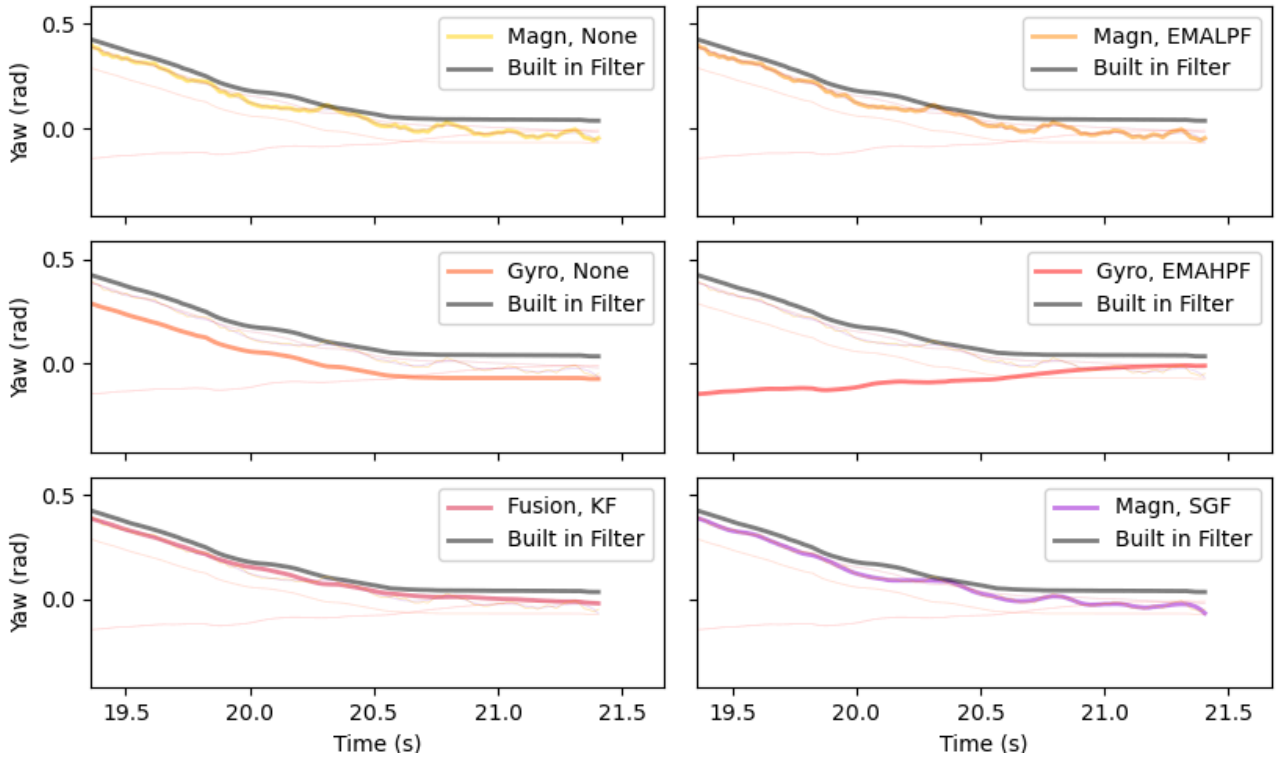

Fig. 42 Zooming in on the last couple of seconds of Fig. 41.

Fig. 42 shows that there has been some drift from all the filters away from the true yaw. But the Kalman filter is closest to the true value. As expected the gyro produces a much smoother fit compared to the magnetometer. Applying a low pass filter to the magnetometer data would make the fit for gyro data better but this would result in some of the higher frequencies being lost which will result in a worse fit for the first test.

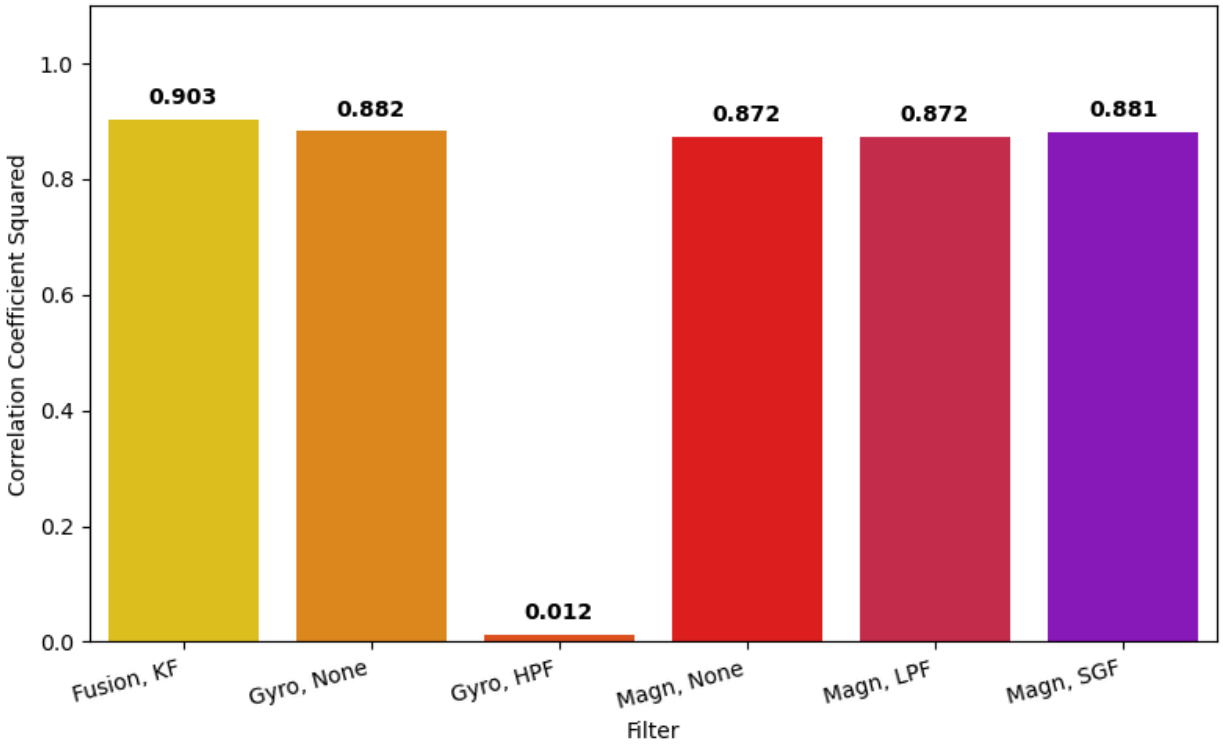

Fig. 43 R^2 for each of the filters against the phones built in filter.

Fig. 43 shows the Kalman filter produced the best fit as did all the other filters except the EMAHPF which had a very poor fit. The magnetometer data gave a much better fit in these examples compared to the previous test as the oscillations are significantly slower, the lag is less significant. In general the second signal is much easier to fit than the first.

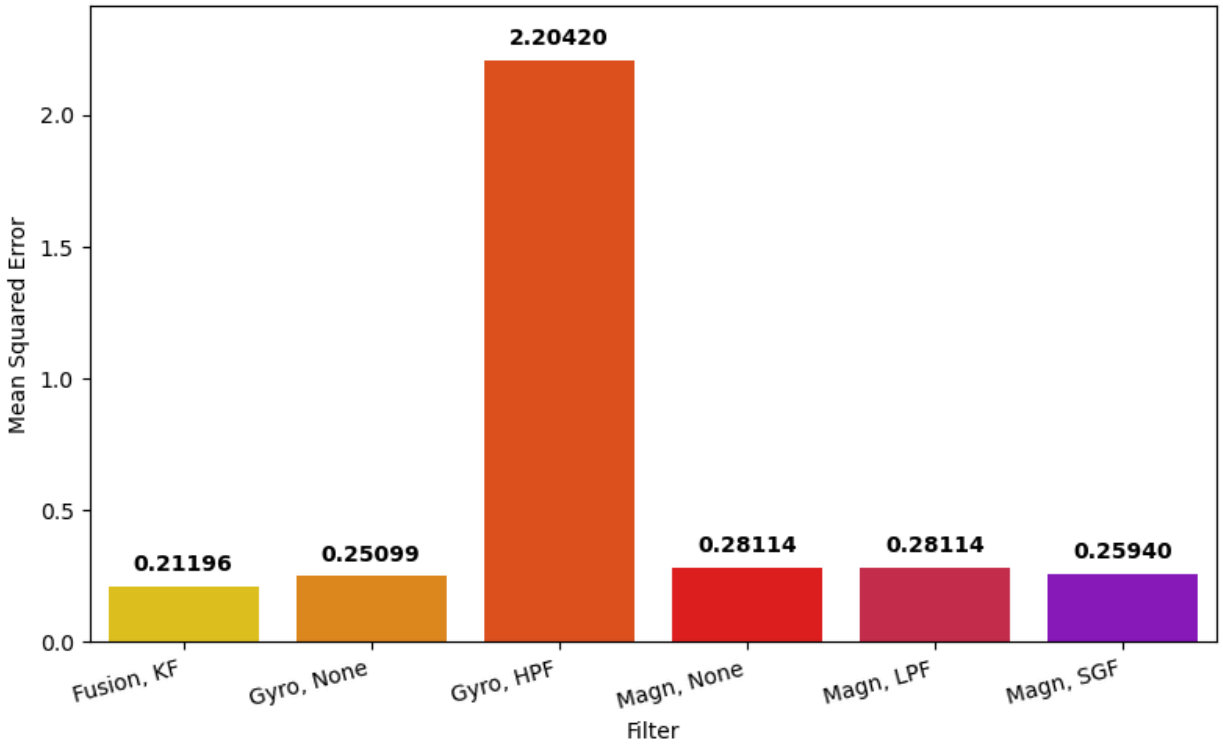

Fig. 44 MSE for each of the filters against the phones builtin filter for the second test.

Fig. 44 indicates the same as Fig. 43, the Kalman filter still provides a marginally better fit than the other filters. The EMAHPF, which had the best fit in the first test now has by far the worst fit as discussed earlier.

8.4 Conclusion

Frequency based filters, like the EMAHPF and EMALPF work well for signals which are oscillating at a constant or near constant frequency, especially when there is only one sensor available. However they work less well for a signal made up from a range of different frequencies, like real world motion. Kalman filters work well in this situation as the combination of measurements minimizes noise from both sensors (which are subject to different types of noise) to achieve a reasonably accurate state estimate, without ignoring any specific frequency data. The parameters of the KF are independent of frequency whereas the EMAHPF and EMALPF have frequency dependent tuning parameters.

Bibliography

Resources

The videos from Dr Shane Ross act as a good introduction to Kalman filters and are what the basics of this research is based from. They are linked here: video 1, video 2 and video 3.

References

[BAOOl21] Armando Barreto, Malek Adjouadi, Francisco R. Ortega, and Nonnarit O-larnnithipong. "intuitive understanding of kalman filtering with matlab". 2021. URL: https://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=6236305, doi:10.1201/9780429200656.

[BH12] Robert Grover Brown and Patrick Y. C. Hwang. Introduction to random signals and applied kalman filtering : with matlab exercises. 2012. URL: http://proquest.safaribooksonline.com/9780470609699.