Exercise 3: Frequency and Kalman filters

Alexander Rasch
alexander.rasch@chalmers.se

Pierluigi Olleja
ollejap@chalmers.se

September 13, 2021

Introduction

In this assignment you will apply your knowledge on signal processing to filter data. This assignment consists of two parts. In the first part, you will design and apply frequency filters (low- and high-pass) to real world data. In the second part you will apply a Kalman filter to estimate the motion of a vehicle. In this assignment you will be using MATLAB.

Note: Grader will only assist you in the second part with the Kalman filter, since the first part gives you more freedom to create your own filter.

Note: You are encouraged to discuss your solutions with other groups but you must write and submit your own code implementation and answers in your group.

Deadline: September 27, 2021 (23:59)

Learning objectives

After having performed this assignment, you shall be able to:

- Plot a signal in the time domain
- Plot a signal in the frequency domain
- Analyze a signal in the frequency domain
- Recognize unwanted frequencies from a signal spectrum
- Design and apply filters (low- and high-pass) to cut-off unwanted frequencies
- Express the kinematic equation of motion of a vehicle in the state space form
- Design and apply a Kalman filter to estimate the motion of a vehicle

Preparations

1. Download the material from Canvas.

   (a) `Exercise_frequency_filters.m` is the template script to solve the first part (A) of the assignment. To solve the first part of the assignment you will also need the function `spectrumanalysis.m` and the dataset `signals.mat`.

   (b) `Exercise_kalman_filter.m` is the template script to solve the second part (B) of the assignment.

2. In both scripts, fill in your group number in the header of the file, and start the tasks below.

Tasks

Part A - Frequency filters

1. Open the script `Exercise_frequency_filters.m`. The script contains some guidelines to complete the exercise and the boilerplate.
   Complete the parts marked by `% ======== YOUR CODE HERE ========`

2. Load the data `signals.mat` in the workspace. Use the function `load`.

3. Once the data is loaded, in your workspace you have two variables: Speed and Acceleration. Plot the two signals (using the plot command) with respect to time (in seconds). As the signals were sampled at 120Hz, you need to create a proper time vector to be able to have the x-axis in seconds. You should get a figure that looks like Figure 1.

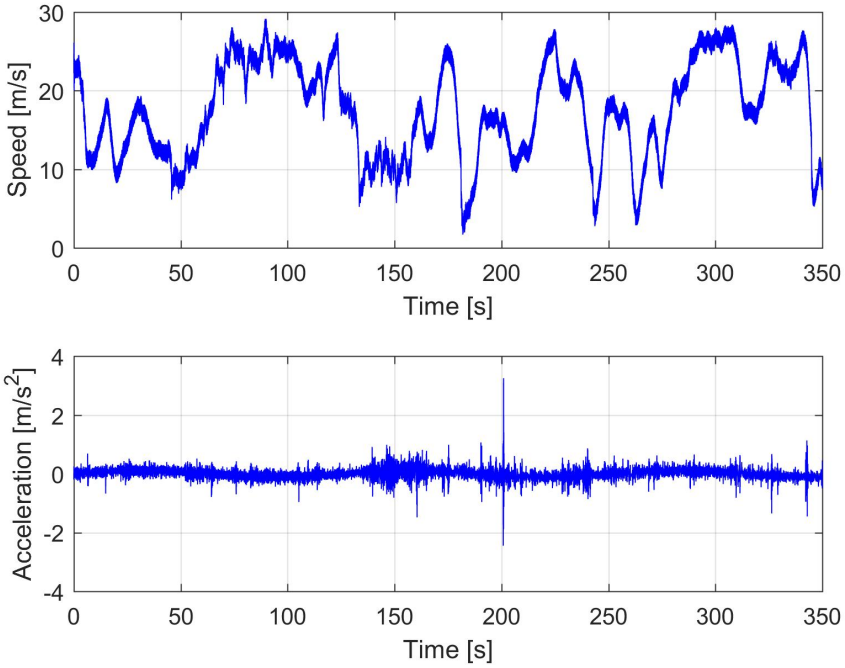


Figure 1: Raw signals for speed and acceleration

4. Start by analyzing the `Speed` signal:

   (a) To inspect the signal, use the function `spectrumanalysis`. Type `help spectrumanalysis` in the console to understand which arguments you need to provide and how to run the function. Remember that the signals were sampled at 120 Hz.

   (b) Speed was acquired in a test field when testing a prototype of an active safety system. This prototype was installed in close proximity of the speed sensor, and it may have introduced noise on the recorded data. It is known that the prototype was powered with a 50 Hz AC power. By looking at the spectrum of the speed, can you see any trace of a potential 50-Hz noise? To eliminate the noise, you could use a low-pass filter. What cut-off frequency would you use?

   (c) There are several ways to design a filter in MATLAB. The function `designfilt` is one of the options. Look at the MATLAB documentation for `designfilt` to understand how it works. In general, you may need to select the filter response (e.g., *LowPassIIR* or *HighPassIIR*), the frequency constraints (e.g., *PassbandFrequency* and *StopbandFrequency*, see also Figure 2), and the design method you want (e.g., *butter* to design a Butterworth IIR filter, i.e. an infinite impulse response filter). You can retrieve more information on the filter you have designed by using the commands `info` (to get general information), `filtord` (to get the order), and `fvtool` (to display the frequency response).

   (d) Once you have designed the filter, apply it on the Speed signal. Use the command `filtfilt` to filter the signal with a zero-phase filtering (i.e., remove the phase distortion).

(e) Finally plot the raw signal and the filtered one. Is you filter effective at removing the noise? Does your figure look like Figure 3a?

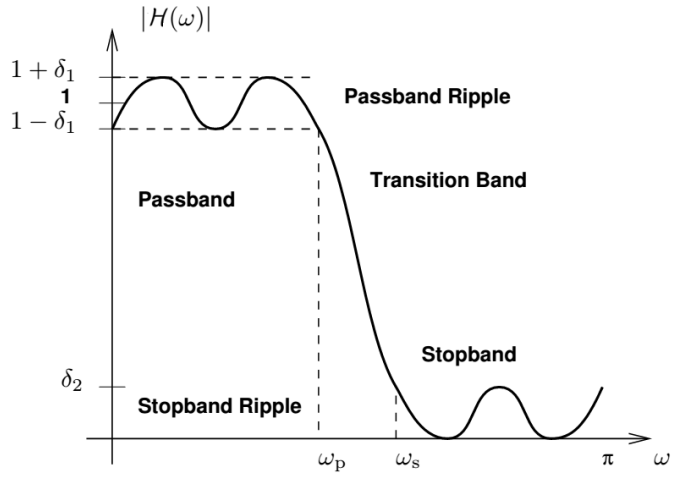


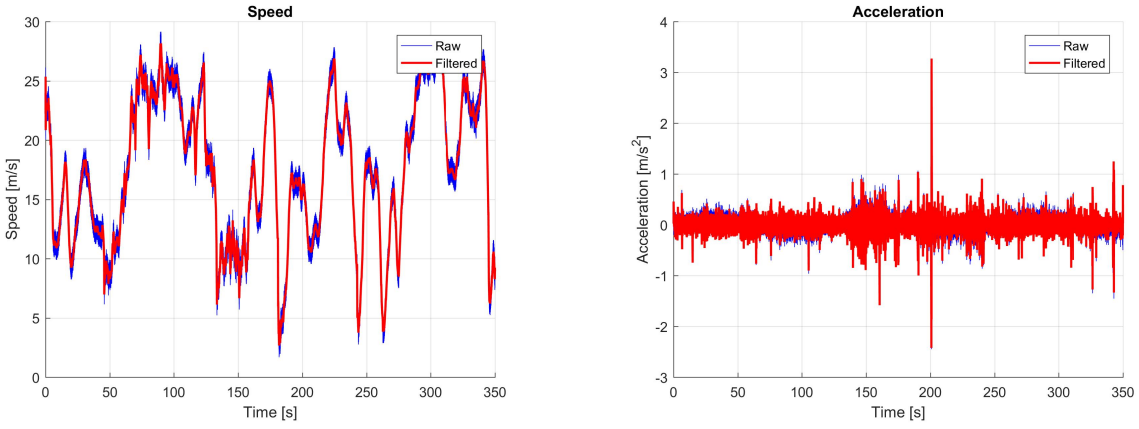
Figure 2: Parameters for designing a filter. Image taken from [1].

5. Next, move on to the `Acceleration` signal. Apply the same workflow as before.

   (a) `Acceleration` was acquired in a test field when temperature was slowly fluctuating. By looking at the spectrum of the acceleration, can you see any trace of a potential slow drift of the signal due to temperature changes?

   (b) To eliminate the low frequency drift, you could use a high-pass filter (e.g., a Butterworth *HighPassIIR*). What cut-off frequency would you use?

   (c) Design the filter and apply it to the `Acceleration` signal. Does your figure look like Figure 3b?



(a) Speed signal

(b) Acceleration signal

Figure 3: Speed and acceleration signal, raw and filtered signals are plotted.

Part B - Kalman filter

1. Open the script `Exercise_kalman_filter.m`. In the beginning of the script, we are simulating the motion of a vehicle travelling on a line. That is, we are simulating the true state of the vehicle across time. However, in practical applications, the true state of the vehicle is usually not observable directly. If the state of the vehicle cannot be observed, it can be estimated with the Kalman filter. Simulating the true state of the vehicle—instead of using real-world data—has the advantage that we can then assess the performance of the Kalman filter. We will, for example, quantify the estimation error given the estimated position and the true position. With the Kalman filter we are trying to estimate the state $\mathbf{x}$ of a vehicle travelling on a line from noisy measurements $\mathbf{z}$. Specifically, we are trying to estimate position and velocity of the vehicle measuring only its position from a GPS sensor. The system we are trying to estimate motion is governed by the state-space equations:

$$\mathbf{x}_k = \mathbf{A}\,\mathbf{x}_{k-1} + \mathbf{B}\,\mathbf{u}_{k-1} + \mathbf{w} \tag{1}$$

$$\mathbf{z}_k = \mathbf{H}\,\mathbf{x}_k + \nu \tag{2}$$

   where $\mathbf{A}$ is the state transition matrix, $\mathbf{B}$ is the control input vector, u is the acceleration input (from the throttle pedal), $\mathbf{w}$ is process noise (perturbation to the system like bumps on the road and wind gusts), H is the measurement matrix (it relates the state $\mathbf{x}$ to the measurement $\mathbf{z}$), and v is the measurement noise (GPS sensor noise, which has a standard deviation of 5 m). The subscript k indicates the time stamp. Variables without subscript are assumed to be constant across time. Moreover, w is assumed to be normally distributed around 0 with process noise covariance Q. Further, v is assumed to be normally distributed around 0 with measurement noise covariance R. The vehicle motion is simulated in the code section `%% SIMULATE THE VEHICLE MOTION.`

2. Your task is to complete the design of the Kalman filter (please, also refer to the slides presented at lecture). The script `Exercise_kalman_filter.m` contains some guidelines to complete the exercise and the boilerplate. Complete the parts marked by `% ======== YOUR CODE HERE ========.`

3. A Kalman filter consists of two steps: Predict and Correct, see Figure 4. Luckily, the MATLAB language allows (almost) a direct translation of the equations in Figure 4 into code.

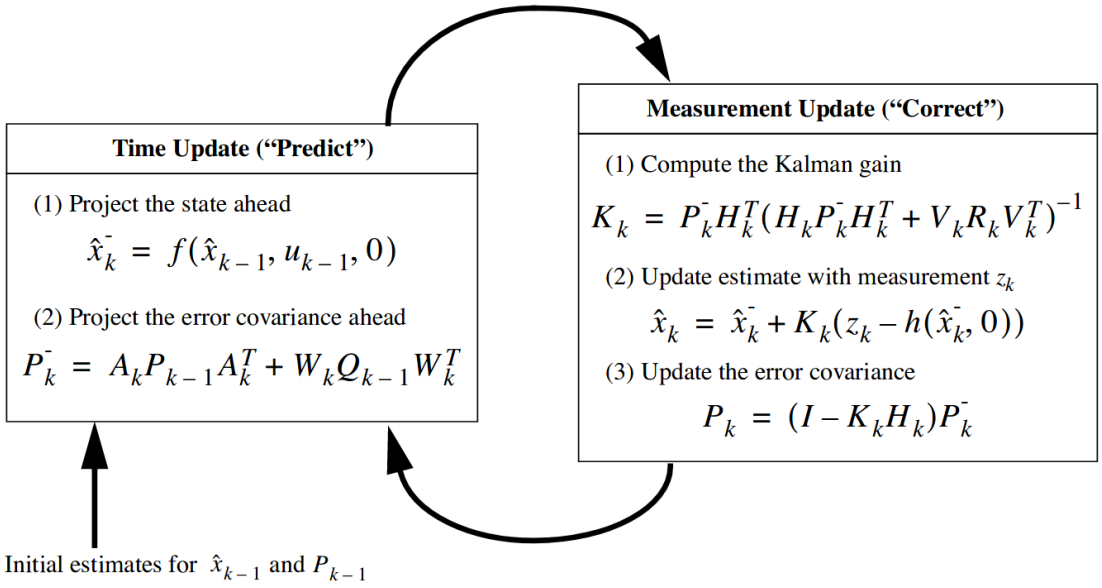


Figure 4: The equations for the Kalman filter, grouped with respect to the *Predict* and *Correct* step. Image taken from [2].

4. First, complete the *Predict* step:

   (a) Project the state ahead (*predict* step):

$$\hat{\mathbf{x}}_k^- = \mathbf{A}\,\hat{\mathbf{x}}_{k-1} + \mathbf{B}\,\mathbf{u}_{k-1} \tag{3}$$

   where $\hat{\mathbf{x}}$ is the vector of state estimate. The superscript $-$ indicates predictions. The initial conditions for $\hat{\mathbf{x}}_{k-1}$ are given. Also, the matrices A and B, and the input vector u are given. $\hat{\mathbf{x}}$ is named `x_hat` in the script. `x_hat` has two rows, corresponding to the two states we want to estimate (i.e., position and velocity), and it has as many columns as the total number of time stamps in the simulation (i.e., the values at the timestamp k can be accessed with `x_hat(:, k))`. To simplify the code, we will not store the prior estimate $\hat{\mathbf{x}}^-$ in a separate variable. That is, we will keep updating the variable `x_hat` within the loop. Thus, the code would look more like $\hat{\mathbf{x}}_k = A\,\hat{\mathbf{x}}_{k-1} + B\,u_{k-1}$.

   (b) Project the error covariance ahead (*predict* step):

$$\mathbf{P}_k^- = \mathbf{A}\,\mathbf{P}_{k-1}\,\mathbf{A}^T + \mathbf{Q} \tag{4}$$

   where $\mathbf{P}$ is the error covariance matrix. The initial conditions for $\mathbf{P}_{k-1}$ are given. The Q matrix is also given. As for `x_hat`, in the script we will not store the prior estimate $\mathbf{P}^-$ in a separate variable. That is, we will keep updating the variable P within the loop. Additionally, we will not store the value for P at each timestamp, but we will keep overwriting it at every step. Thus, the code would look like $P = A\,P\,A^T + Q$. The command to transpose a matrix in MATLAB is the single quotation mark `'`.

5. Second, complete the *Correct* step:

   (a) Compute the Kalman gain:

   $$K_k = \mathbf{P}_k^- \mathbf{H}^T \left(\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R}\right)^{-1} \tag{5}$$

   Matrix $\mathbf{H}$ and $\mathbf{R}$ are given. Remember the simplification we made above for $\mathbf{P}_k^-$.

   (b) Update the estimate with the measurement $\mathbf{z}_k$:

   $$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^-) \tag{6}$$

   The measurements $\mathbf{z}$ are given. Remember the simplification we made above for $\hat{\mathbf{x}}_k^-$.

   (c) Update the error covariance:

   $$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H})\mathbf{P}_k^- \tag{7}$$

   $\mathbf{I}$ is the identity matrix. It can be computed with the MATLAB command `eye`. Remember the simplification we made above for $\mathbf{P}_k^-$.

6. Finally, run the script from the beginning. If your implementation is correct, you should obtain something like Figure 5. Inspect the graphs and note how the estimation error quickly drops to zero. We managed to estimate the state of the vehicle using only its (noisy) GPS position!

Submission

1. Run and submit your code for part B (`Exercise_kalman_filter.m`) in MATLAB Grader to verify that the Kalman filter output is correct.

2. Submit your solutions (both files, `Exercise_frequency_filters.m` and `Exercise_kalman_filter.m`) in the assignment in Canvas. It is sufficient if at least one group member submits your solution in Canvas. See deadline above.

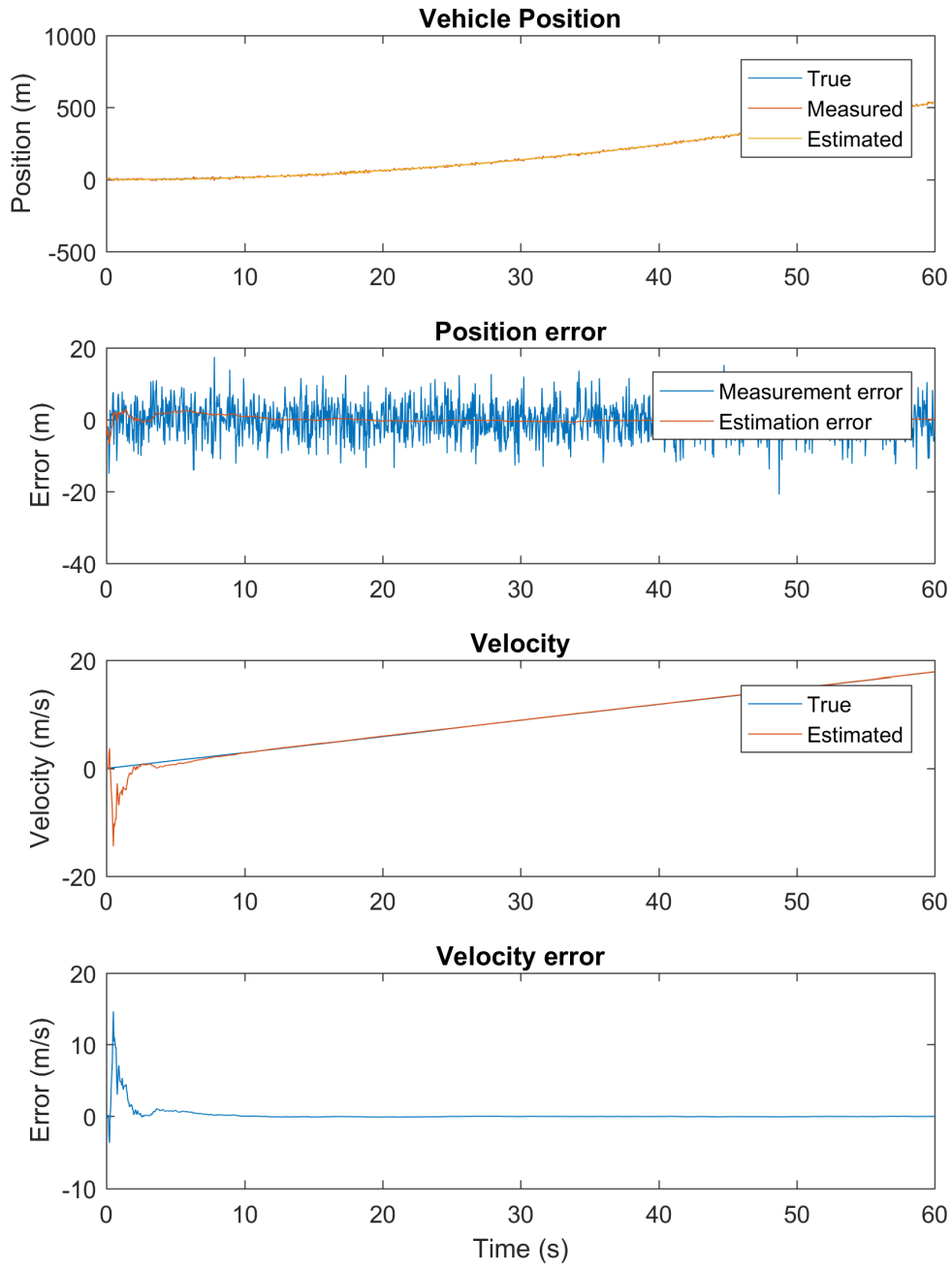


Figure 5: Results from the Kalman filter

References

[1] Jeffrey Fessler. *Design of Digital Filters*. 2004. URL: `https://web.eecs.umich.edu/~fessler/course/451/l/pdf/c8.pdf`.

[2] Greg Welch and Gary Bishop. *An Introduction to the Kalman filter*. Tech. rep. TR95-041. The article has also been translated into Chinese by Xuchen Yao, a student at The Institute of Acoustics of The Chinese Academy of Sciences. See also our Kalman filter web site at https://www.cs.unc.edu/ welch/kalman/in-

dex.html. University of North Carolina at Chapel Hill, Department of Computer Science, 2006. URL: https://sreal.ucf.edu/wp-content/uploads/2017/02/kalman_intro.pdf. published.