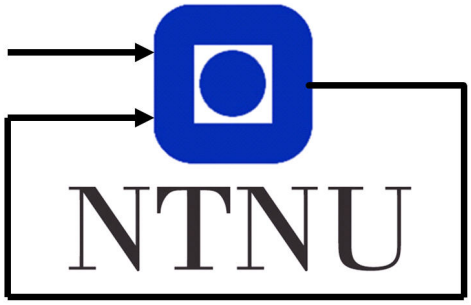
TTK4115 Helicopter Lab Report

Group 131
Simon 538244
Henry 537942

18th November, 2020



Department of Engineering Cybernetics

Contents

1 Part I - Monovariable Control

1.1 Task 1 - Manual Control

As a start, the joystick controller was connected directly to the motors. The helicopter was difficult to manoeuvre and the control's y-axis was inverted. The gain on the y-axis was inverted to a positive value.

Listing 1: Matlab code of the gain variables.

```
1 Joystick_gain_x = 1;
2 Joystick_gain_y = 1;
```

The gain of $[1, 1]$ was about as controllable as one could get when the motors were directly controlled by joystick. Anything more or less amde it more difficult to control.

1.2 Task 2 - Parameter Identification

The elevation angle was incorrect as it was desired to let the equilibrium point be at the point where the helicopter was horizontally level. The elevation value was measured by lifting the helicopter from the table and to the desired equilibrium point. The measured value at the desired equilibrium point was

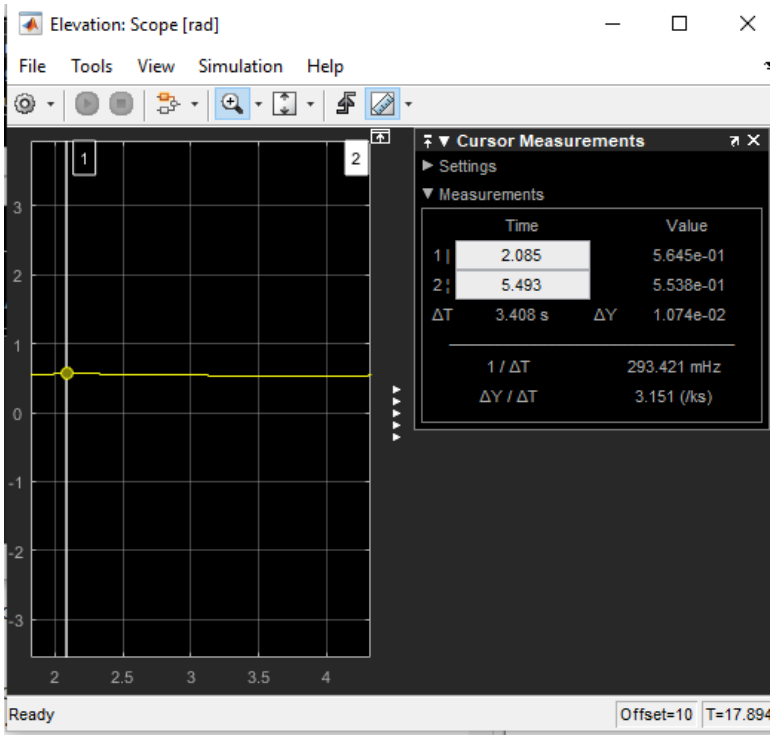


Figure 1: e value when the helicopter was horizontal

measured to be about 0.56. This offset was subtracted from the measured value of the elevation angle. The corrected elevation angle was only correct if the helicopter started from the table, which was important to consider whenever tests had to be done. The reason why the elevation angle had to be corrected was to make sure the values given from the real system would match the mathematical model.

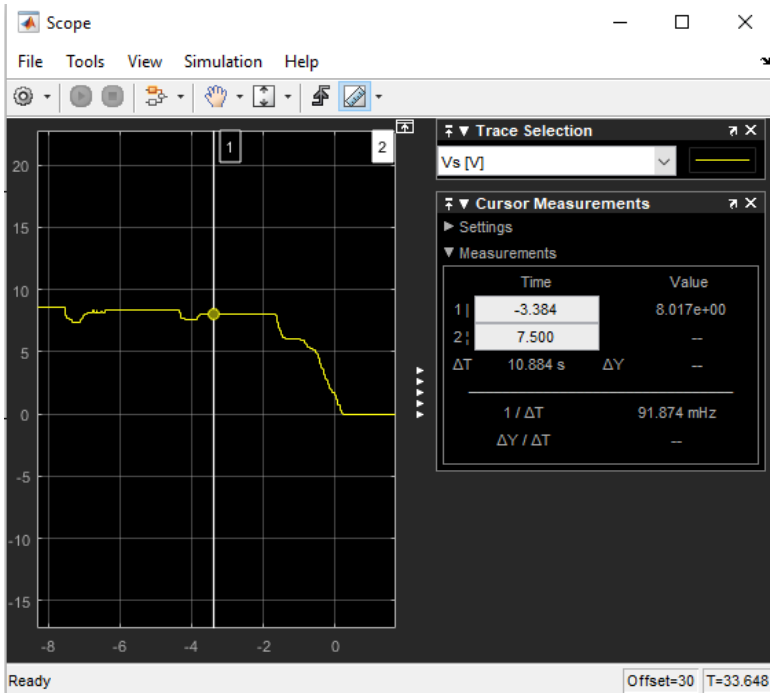


Figure 2: V_s value when the helicopter was horizontal

The offset was found to be about 8.0 by letting the helicopter hover at the desired equilibrium point. This offset was inserted into the Simulink model and was used in further development.

An offset for the voltage had to added to the voltage value to make sure the value of the voltage would make the helicopter hover whenever the helicopter was level. This is done because it is desirable to make the equilibrium point of the helicopter at that point instead of on the table. After this, the helicopter would attempt to stabilise itself at the new equilibrium. The new voltage variable with offset will from here on be known as V_{s0}.

After the variable V_{s0} has been found, one can find the motor force constant K_f. The equation for K_f is as follows:

$$K_f = \frac{L_h 2 m_p g - L_c m_c g}{V_{s0} L_h L_p} \tag{1}$$

L_h	Distance between elevation axis to helicopter head	0.66 (m)
L_c	Distance between elevation axis to counterweight	0.46 (m)
L_p	Distance from pitch axis to motor	0.175 (m)
m_p	Mass of motor	0.72 (kg)
m_c	Mass of counterweight	1.92 (kg)
g	Gravity constant	9.81 ($\frac{m}{s^2}$)

Explanation of Variables

The variable K_f was found to be 0.713 after plugging in the variables. This variable was used in further development of the system.

1.3 Task 3 - PD Control

In the lab preparations, the pitch controller was calculated to be:

$$Vd(s) = Kpp * (Pc - P) - Kpd * P' \tag{2}$$

This controller was implemented in the already made Simulink model as a subsystem.

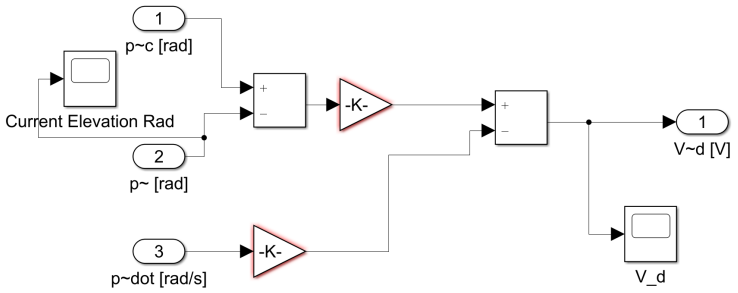


Figure 3: The Pitch Controller

When the pitch controller was implemented, the next step was to try out different pole placements. Finding the perfect poles is not something that is trivial, but finding a tuning that keeps the helicopter in the air was not too difficult. Multiple poles where tested and the results can be seen in table 1.3.

heightPoles tested	Observations/Result
Pole 1: -1 Pole 2: -0.3	System is stable but the response is a little slow
Pole 1: -3 Pole 2: -1	System had some unwanted oscillations (under-damped), However the response was faster then the previous test
Pole 1: 0 Pole 2: 0	System is uncontrollable and unstable. This checks out with the theory of system poles
Pole 1: -0.1 Pole 2: -0.5	System is stable, but not easy to control. Pitch control was sluggish.
Pole 1: -9 Pole 2: -5	The pitch oscillated with a growing amplitude. This is unstable and not possible to control.

The system was controllable, but the user was required to compensate with their input if they would like for the helicopter to stabilise at a reasonable time.

2 Part II - Multivariable Control

2.1 Task 1 - Implementation

In the previous lab a controller was only implemented for the pitch angle. The controller used was a PD controller. In this lab the goal was to implement a state feedback controller with a feed forward reference. The feedback controller uses the states of interest with a gain matrix K, that will later be used for tuning the controller. The feed forward part takes the reference states and weighs them with a matrix F that is calculated so that the states of the system will converge towards our reference. The equation for the controller is as follows:

$$u = F * r - K * x \tag{3}$$

The matrix u, contains the input voltages $\tilde{V_s}$ and V_d:
$u = \begin{bmatrix} \mathrm{V}_s \\ \mathrm{V}_d \end{bmatrix}$
Matrix x contains the following states:
$x = \begin{bmatrix} \mathrm{p} \\ \mathrm{p'} \\ \mathrm{e'} \end{bmatrix}$
The reference matrix r gets its values from the joystick and is expressed as:
$r = \begin{bmatrix} \mathrm{p}_c \\ \mathrm{e'}_c \end{bmatrix}$
The feed forward gain matrix F was calculated in the preparation part of the lab. The resulting matrix is as follows:
$F = \begin{bmatrix} \mathrm{k}_{11} & \mathrm{k}_{13} \\ \mathrm{k}_{21} & \mathrm{k}_{23} \end{bmatrix}$
$K = \begin{bmatrix} \mathrm{k}_{11} & \mathrm{k}_{12} & \mathrm{k}_{13} \\ \mathrm{k}_{21} & \mathrm{k}_{22} & \mathrm{k}_{23} \end{bmatrix}$
As shown, the F matrix values are dependent on the gain matrix K. The gain matrix will be chosen using pole placement in the next part.
The controller was implemented in Simulink as shown in figure 4.

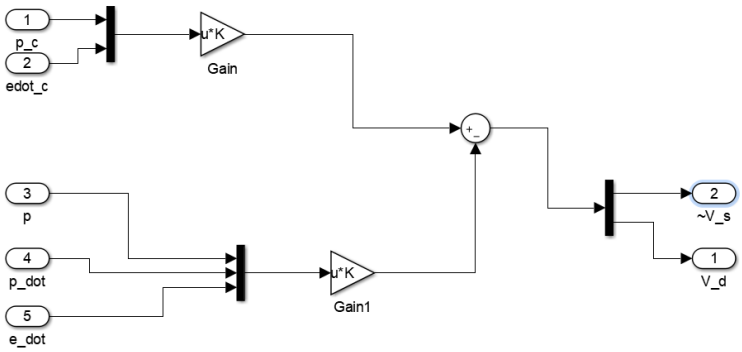


Figure 4: The Simulink diagram of the state feedback controller with feed forward reference. Note that the gain blocks in the picture are not correct since they both use the K matrix as gain. The upper gain block contains the F matrix, and the one below contains the K matrix.

2.2 Task 2 - Pole Placement

The matrices A and B was implemented in Matlab, and the the K values were computed for some given pole placements using the Matlab command place(A,B,p), where p is a vector of poles. The pole vector $[-0.7, -1.6, -1.9]^T$ was determined to give the best results after a few experiments. It gave a controllable system which converged quickly, but it had some issues with minor oscillation in the pitch angle. A system without oscillation was also found but the convergence was slow and a higher convergence rate was preferred over no oscillations.

In graph 5, one can see the oscillations on the pitch axis caused by the controller.

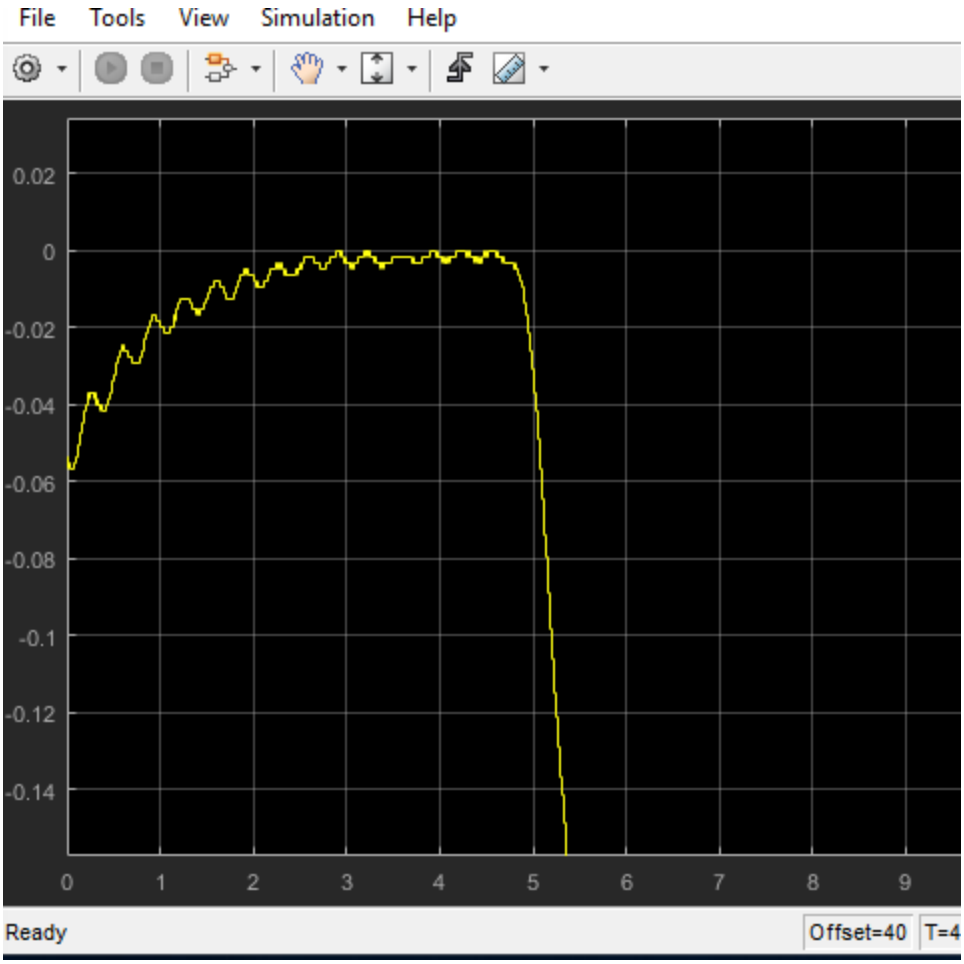


Figure 5: Scope of the pole placement experiment.

2.3 Task 3 - LQR

The manual pole placement of K was instead replaced by the LQR algorithm found in Matlab. The weighting matrices Q and R were made diagonal for simplicity as shown in the code snippet below.

2.4 Task 4 - LQR With Integral Action

The LQR controller was further developed to include integral action as well. When the integral action was included two new states was introduced in x. The K matrix was also extended so that zeta and gamma also had a gain. The new matrices look like this:

$$x = \begin{bmatrix} \mathrm{p} \\ \mathrm{p'} \\ \mathrm{e'} \\ \gamma \\ \zeta \end{bmatrix} \quad K = \begin{bmatrix} \mathrm{k}_{11} & \mathrm{k}_{12} & \mathrm{k}_{13} & \mathrm{k}_{14} & \mathrm{k}_{15} \\ \mathrm{k}_{21} & \mathrm{k}_{22} & \mathrm{k}_{23} & \mathrm{k}_{24} & \mathrm{k}_{25} \end{bmatrix}$$

Listing 2: Matlab code of the gain matrices.

```
Q = diag([1;1;1;1;1]);
R = eye(2);
K = lqr(A,B,Q,R)
F = [K(1,1) K(1,3); K(2,1) K(2,3)]
```

The weighting matrices were at first identity matrices for simplicity's sake. The matrices were then tuned one at a time, through trial and error.The system eventually got stable and converged fast. The matrices which ended up giving the best system was one with the Q and R matrices:

$$\mathrm{Q} = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 100 \end{bmatrix}, \mathrm{R} = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix}$$

These matrices gave a system which was converging quickly towards the origin and was very easy to control, but it suffered from a small amount of oscillations in the pitch angle. These oscillations quickly disappeared out after a short period of time. This could have been solved by increasing the pitch cost in the Q matrix, however the group prioritised a quicker convergence rate.

When testing the system with and without integral action we did not see any big difference. Integral action is a good tool for eliminating disturbances. When we don't have any integral action we depend on modelling a feed forward gain matrix that eliminates the effects of disturbance. This however works well in theory, but in actuality we get some deviation in the actual system. The integral solves this problem by integrating over the error. When integrating over the error the value grows larger until the input u reacts. This means that the steady state error is eliminated. Since the integral action removes steady state error, the F matrix no longer needs to be calculated to remove the steady state error. However when changing the F matrix, the control was not as good. It is recommended to keep the F matrix as it is for a good tuning.

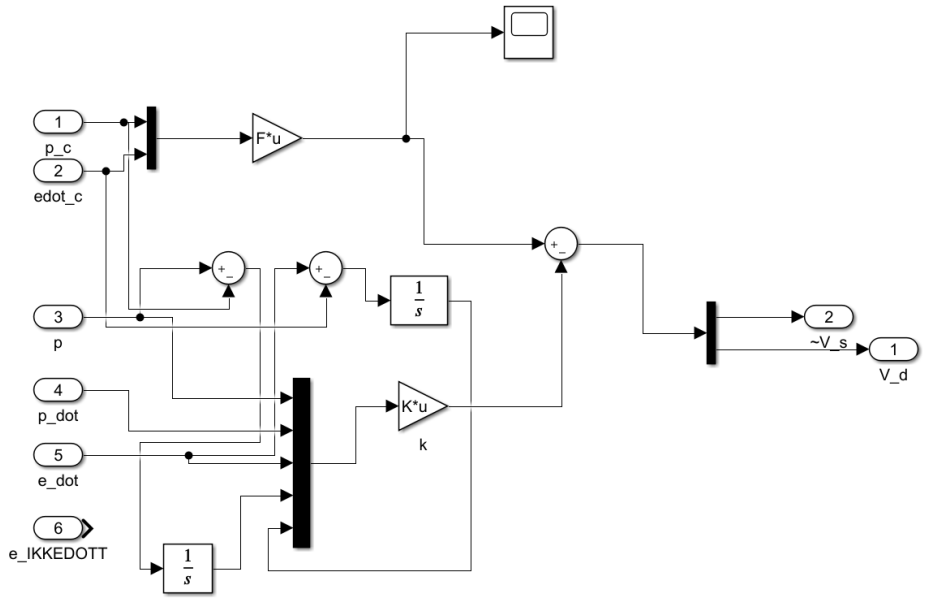


Figure 6: The Simulink diagram of the LQR controller with integral action

3 Part III - Luenberger Observer

3.1 Task 1 - IMU Characteristics

In this task, the data gathered from the IMU was compared with the data produced by the encoders. The first test was rotating pitch, elevation and travel manually around the equilibrium point. The encoder values where the same as the data from the IMU when measuring the rates, other then small deviations. The angles however had a constant deviation. Apart from the constant deviation the signals where similar. However when rotating two angles at the same time the constant deviation got worse. The deviation is caused by the fact that the coordinate system for the encoders is fixed, but the coordinate system for the IMU will move, as we turn the helicopter.

When looking at the accelerometer output, it was possible to see the gravity constant expressed as a three-dimensional vector which changed values as the helicopter was rotated.

3.2 Task 2 - Transformations

In the previous task it was discovered that the angles measured from the IMU had a constant deviation when compared to the encoder values. This was handled by implementing a transformation block that was already created provided through Blackboard.

To measure the angles p and e, equations were created in the preparations to calculate the angles, with the accelerometer values as input. The equations for the angles was expressed as follows:

$$p = arctan(a_y/a_z) \tag{4}$$

$$e = arctan(a_x/\sqrt{y^2 + z^2}) \tag{5}$$

These equations are correct as long as the helicopter is stationary. Small deviations from the real value will occur when the helicopter is accelerated by other forces than the gravity.

The function block was implemented in the Simulink model (see Appendix A.1). Here the function block has an added exception in cases where the denominator for $tan^{-}1()$ would be equal to zero. After implementing the transformation block for the gyro, and implementing the necessary calculations to compute the angles from the accelerometer, the values where checked and verified. The angles and angle rates from the IMU matched with the data from the encoders. However, since arctan was used to calculate the angles. Arctan has a range of: [- $\frac{\pi}{2}$, $\frac{\pi}{2}$]. This means that if the angle exceeds the range, the angle will appear to move π rad.

3.3 Task 3 - Theoretical Discussion

In problem 2 in the preparations, the observability of the matrices, where only the angle rates are taken into account, was calculated.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ k3 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6}$$

$$\tag{7}$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

Where $k_3 = 1$. The observability matrix with only the rates was calculated to be:

$$O = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{9}$$

The observability matrix has a rank of 4, while the observability matrix requires a rank of 6 for the system to be observable. The observable states are: [p,$\dot{p}$,$\dot{e}$,$\dot{\lambda}$]. The reason for that p is observable is that it is expressed in:

$$\ddot{\lambda} = k_3 p. \tag{10}$$

λ and e is not observable.

When including all states in the C matrix:

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{11}$$

The observability matrix is:

$$O = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{12}$$

The matrix has rank 5, which means that all the states are observable with λ as a exception.

From experimenting and theory learned in the course, it can be concluded that a state is observable if it can be measured by the output of the system. It is also possible that a state is indirectly observable if the state is represented as shown for p when the only directly observable states where the angle rates. To observe all of the states of the system, it was not sufficient to only use the angle rates since e was not observable. Therefore the C matrix was extended to make the whole system observable except λ.

3.4 Task 4 - State Estimator

The IMU measurements contain allot of noise which is not desired. A linear observer was introduced, which in theory will smooth out the signal. A linear observer uses a model of the system. In our case, the system matrix A, B, and C is required. The observer will take in the measurements y from the IMU as well as the input u. u and y is fed into the model which produces a estimate $\hat{x}$ of the system. A matrix L is also introduced to tune the estimator. The equation for the estimator is as follows:

$$\hat{\dot{x}} = A\hat{x} + Bu + L(y - C\hat{x}) \tag{13}$$

As seen in equation 12, the L matrix weighs the difference of measured output and estimated output, and is used for tuning the estimator. The implementation in simulink is as follows:

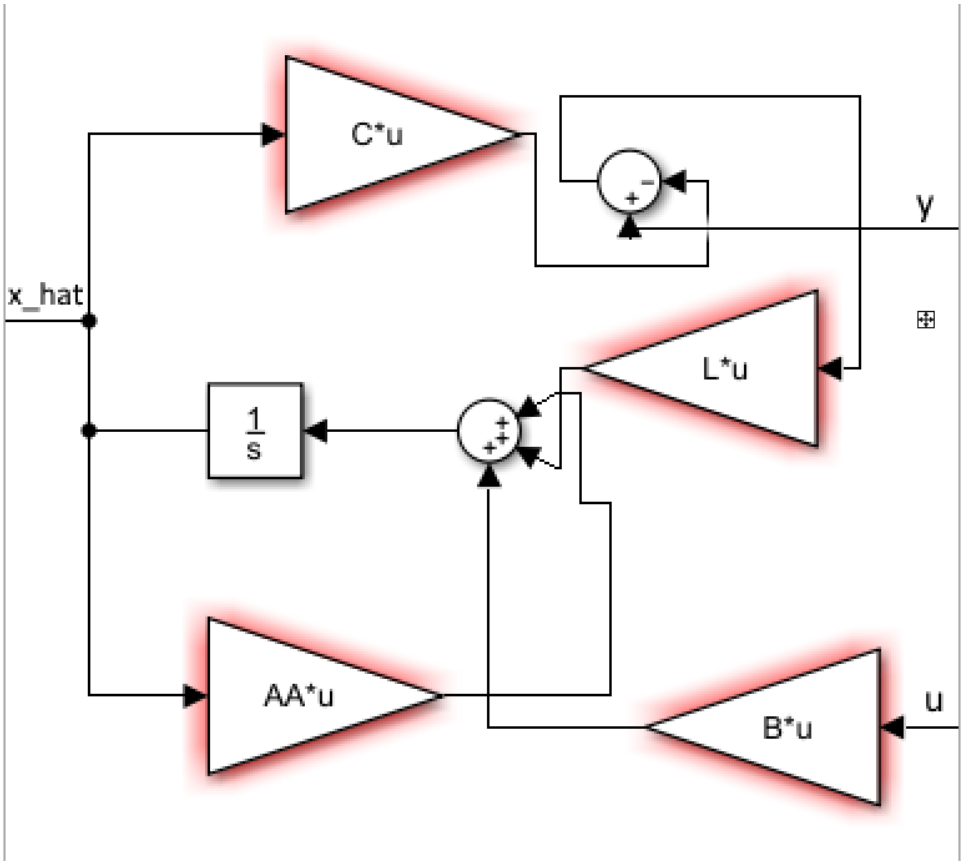


Figure 7: Linear observer

When the observer was implemented some tests was performed to see if the estimator tracked the states properly.

When comparing the estimated states with the encoder, there was some deviation, but the estimated states followed the encoder data nicely. It was also observed that the estimator was a little slow in response to changes.

When comparing the estimated states with the IMU, it was observed that the estimators data was allot smoother then the data from the IMU, as demonstrated in this graph:

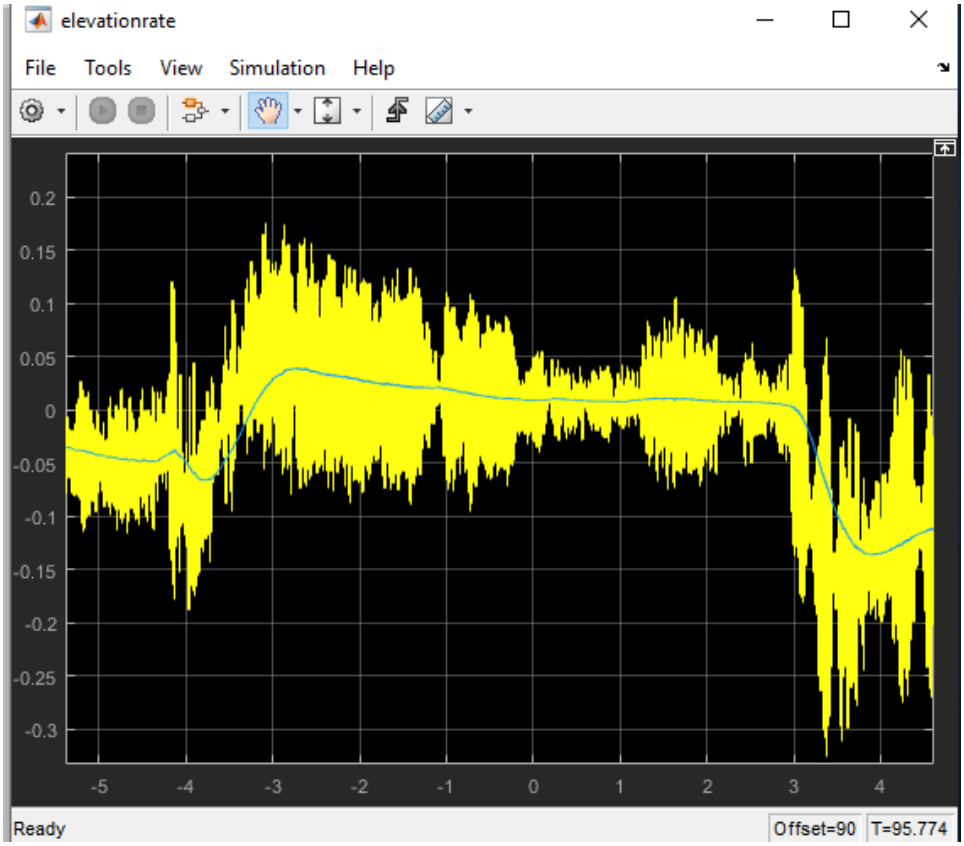


Figure 8: Blue is data from the estimator, while yellow is data from the IMU

Pole placement was used to tune the observer. In Matlab the function

L = place(A',C',p')

was used to choose the gain matrix. After experimenting with different poles it was observed that large poles produced a highly responsive, but noisy prediction. On the other hand, small poles produced a slower response with

better noise suppression. When choosing the optimal poles, it was desired to suppress some noise, but not sacrifice the responsiveness of the system. Therefor a middle ground was found for poles at

$$p = [-10, -8, -7, -8] \tag{14}$$

This resulted in an estimation that was responsive and suppressed some of the noise.
At first it seemed like the estimator worked fine, and flying the helicopter worked well. However, it was later discovered that the system was slightly unstable. If the helicopter is left to hover, the elevation angle would start to deviate and the helicopter would slowly approach the table of the setup. The group attempted to solve this by changing the tuning of the gain matrix L, as well as tuning the LQR. Another problem arrived when trying to stabilise the estimator. Previously the pole in equation 14 worked pretty well, but when implementing the same pole later, the helicopter laid docile on the table without any input to the motors. The only way to make the estimator work again was to use very big poles. The group managed to get the helicopter flying again for the evaluation, but there was still some instability.

4 Part IV - Kalman Filter

4.1 Task 1 Noise Estimate

The measurement noise was at first checked when the helicopter was laying still on the table and with the motors disconnected. The covariance of the different signals were calculated by using the cov() function. The covariance matrix while the helicopter was on the ground is as follows:

$$R_{ground} = \begin{bmatrix} 0.0651 & 0.0035 & 0.0054 & 0.0041 & -0.0047 \\ 0.0035 & 0.0668 & -0.0009 & 0.0032 & 0.0049 \\ 0.0054 & -0.0009 & 0.0776 & 0.0018 & -0.0023 \\ 0.0041 & 0.0032 & 0.0018 & 0.0736 & 0.0006 \\ -0.0047 & 0.0049 & -0.0023 & 0.00060.1646 & \end{bmatrix} 10^{-5} \tag{15}$$

As seen here, the covariance is close to zero, which is the definition of white noise. A second set of measurements were done while the helicopter was flying at the linearisation point. The LQR controller was used for this part. The covariance matrix while flying:

$$R_{air} = \begin{bmatrix} 0.0015 & -0.0014 & -0.0017 & 0.0011 & 0.0026 \\ -0.0014 & 0.0031 & -0.0000 & -0.0020 & -0.0039 \\ -0.0017 & 0.0000 & 0.0092 & -0.0001 & 0.0002 \\ 0.0011 & -0.0020 & -0.0001 & 0.0021 & 0.0024 \\ 0.0026 & -0.0039 & 0.0002 & 0.0024 & 0.0183 \end{bmatrix} \tag{16}$$

Here the covariance matrix R_{air} has significantly higher values than R_{ground}, but the covariance matrix is close enough to zero for it to be still defined as white noise. The reason why the covariance is higher is most likely because of the disturbances coming from the motors.

4.2 Task 2 - Discretization

The Kalman-filter that is to be used in this lab requires the system to be discrete. The system was discretized using the Matlab function c2d(sys). The new system matrices are as follows:

$$A_d = \begin{bmatrix} 1 & 0.002 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.002 & 0 & 0 \\ 0.0005 & 0 & 0 & 0 & 1 & 0.002 \\ 0.4832 & 0.0005 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B_d = 10^{-3} \begin{bmatrix} 0 & 0 \\ 0 & 0.2495 \\ 0 & 0 \\ 0.1594 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

The matrix C_d is the same as C

4.3 Task 3 -Implementation

A correction step was implemented as a Matlab block in Simulink (See Appendix A.2). Here the code updates the correction when new data has been received, while setting $\hat{x}[k+1]$ and $\hat{P}$ equal to $\bar{x}$ and $\bar{P}$ when no new data has arrived.

4.4 Task 4 - Experimentation

Q_d was first set to zero during experimentation. The estimated pitch rate diverged from the measured value and the error increased linearly towards infinite. The other estimated values did not match up nicely and it was concluded that the filter would be rather useless in this situation.The value also seemed to have a low variance.

Q_d was then set to 1. The pitch and elevation followed the encoder signal but was very noisy. Some extremely high noise spikes several magnitudes higher than the normal noise level was observed.

Q_d was then set to an arbitrarly high number. The estimated position became extremely noisy and seemed to follow the noise signal exactly while the mean value seemed to roughly follow the measured position from the encoder. After doing several experiments. The Q_d matrix was set to a value where the noise would not create too large spikes in the estimated value, while keeping the variance of the matrix somewhat close to the variance of the encoder values. A manual switch was added to the Simulink model to make sure it was possible to force the correction to not update.

As seen in figure 9, the value deviates from the measured value from the encoder. This is because the new values need to use old data to predict the new position, which results in a straight line.

The Kalman filter expresses the error dynamics as a normal distribution with a covariance. The goal is to calculate a gain matrix K with a minimised variance of the error dynamics. This will provide the optimal gain matrix. In the prediction step, a prediction on the states of the system is made, as well as a prediction on the error covariance. These are calculated based on the the previous estimate $\hat{x}$ and the covariance of the error. When

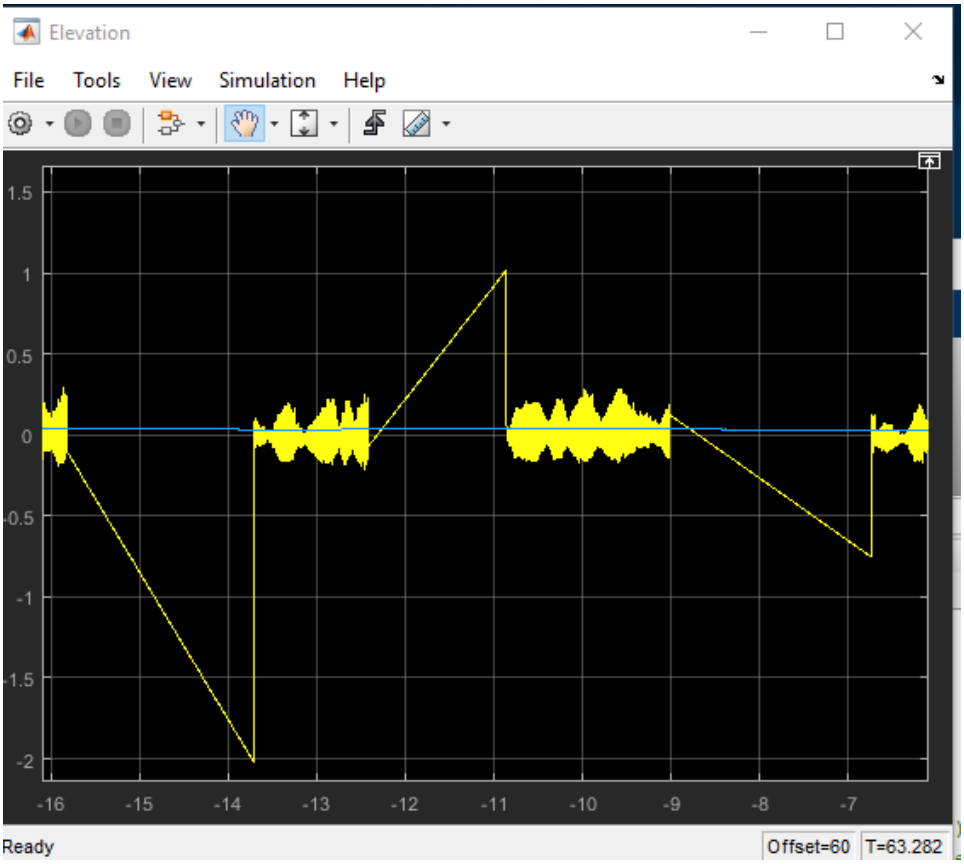


Figure 9: Scope of elevation during manual switching of new data

a prediction is made, the kalman gain is calculated based on the predicted error dynamics. When the kalman gain is calculated, a estimate of the states is calculated based on the predicted states, the kalman gain, and the system measurements. This loop continues. The equation

$$\hat{x}(k) = \bar{x}(k) + K(k)(y - C\bar{x}) \tag{17}$$

will sum the predicted states with the with the difference between the measured outputs and the estimated outputs. If the kalman gain has a good tuning, the error dynamics will have a small covariance leading to a good estimate of the states. The linear observer does not use this statistical approach when tuning the estimator. The kalman filter minimises the covariance of the error dynamics to produce a good gain matrix, but the linear estimator uses methods like pole-placement.

4.5 Task 5 - Tuning

The Kalman filter proved to be hard to tune. The filter was tuned in such a way that the pitch angle became very responsive and stabilised quickly. The elevator angle however, proved hard to tune. The measured $\dot{e}$ angle inverted itself every time λ was changed by about $\frac{\pi}{2}$. This problem was fixed after going through the Simulink diagram thoroughly. It is far more stable after the fix. The system was able to perform better and was less prone to noise than the Luenberger Observer. A more fine-tuned system would further improve the performance.

5 Conclusion

The first part featured a monovariable controller. The result was adequate for controlling the helicopter, though improvements were desired. The multivariable controller brought improvements to the system by implementing an LQR controller. The LQR controller made tuning the controller much easier. The resulting system with the LQR controller was easier to control than the monovariable one, but the user input was slow. The convergence rate of the LQR controller was rather slow as well. The Luenberger Observer was implemented and gave a controllable system. It was observed that the estimator smoothed out the noise that was present in the IMU readings at the expense of responsiveness. It was also possible to tune the estimator to be more responsive, but the noise suppression suffered. The Kalman filter was the final part of the lab. It made it possible to fly the helicopter with only the measurements from an IMU, just like ordinary drones. The system was able to fly with the same measurements as the Part III system, while being less prone to the noise from the IMU.

A MATLAB Code

A.1 AccelToAngles.m

```
function [e,p]= fcn( a_x,a_y,a_z)
%#codegen
e = 0;
p = 0;
if sqrt((a_y)^2 +(a_z)^2) == 0 || a_z == 0
    if sqrt((a_y)^2 +(a_z)^2) == 0
         e = 0;
    end
    if  a_z == 0
        p = 0;
    end
else
    e = atan(a_x/(sqrt((a_y)^2 +(a_z)^2)));
    p = atan(a_y/a_z);
end
```

A.2 CorrectionStep - xHatt

```
function [x_h1,P_h] = fcn(x_,y,C_d,R_d,P_,newData)
if(newData)
K_k = P_*C_d'*(C_d*P_*C_d' + R_d)^-1;
P_h = (eye(6) -K_k*C_d)*P_*(eye(6) - K_k*C_d)' +K_k*R_d*K_k';
x_h1 = x_+K_k*(y-C_d*x_);
else
x_h1 = x_;
P_h = P_;
end
```