Laboratory Exercise 3 - Modeling and controlling a quadcopter

Theodora Gaiceanu

1 Estimating k

First of all, one should have inside **Omega_in.signals.values** some equally spaced points between the lower limit of omega and the upper limit of omega. This can be done using the *linspace* function.

```
Omega_in.signals.values = linspace(omega_min_lim,
    omega_max_lim,length(Omega_in.time))'*ones(1,4)
```

Listing 1: Omega values

Then, one should create data objects for the following three models.

$$f_i = k * \Omega$$
$$f_i = k * \Omega^2$$
$$f_i = k * \Omega^3$$

This means that the second parameter of the data objects (corresponding to the input) should be modified accordingly to the equations above. After, the process models should be estimated using the data objects. The estimated models have zero poles.

```
dat1 = iddata(out.acc.data(:,3)+g,4*out.Omega.data(:,1),
    inner_h);
dat2 = iddata(out.acc.data(:,3)+g,4*out.Omega.data(:,1).^2,
    inner_h);
dat3 = iddata(out.acc.data(:,3)+g,4*out.Omega.data(:,1).^3,
    inner_h);
```

```
sys1 = procest(dat1,'p0')
sys2 = procest(dat2,'p0') %p0 means zero poles
sys3 = procest(dat3,'p0')
```

Listing 2: Building the estimations

The identified process has the form from Equation(1).

$$\sigma = \frac{k}{m}u \tag{1}$$

Considering this, one can say that the identified process equals to a constant multiplied with the input, where the constant is $K_p = \frac{k}{m}$. Consequently, $k = K_p * m$.

```
k_est = sys2.Kp*m
```

Listing 3: Estimation of k

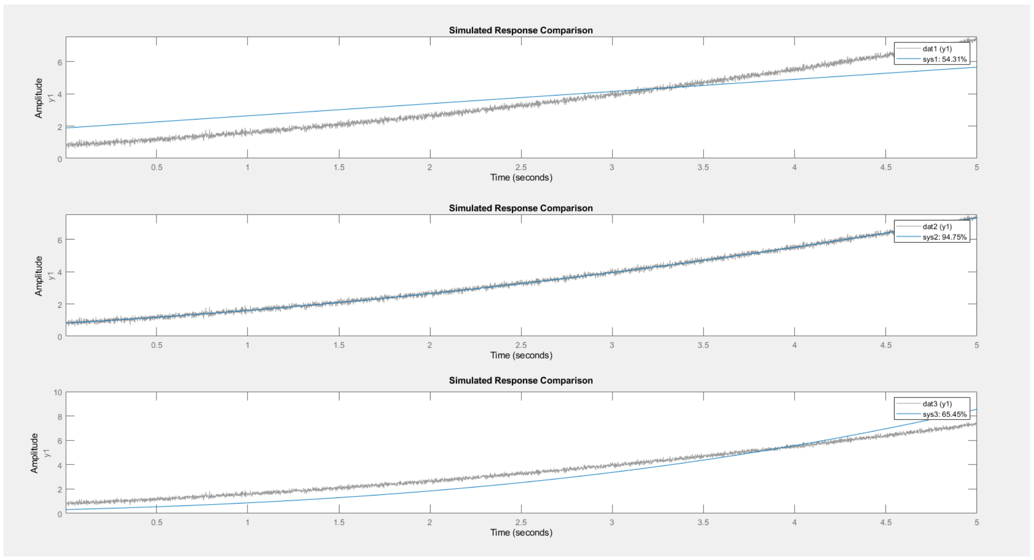

Figure 1: Results for the identified systems

Analysing Figure 1, it can be noticed that the model with the highest accuracy is system 2. This is why model 2 was used in order to estimate k.

From Figure 2, one can see that the quadcopter is moving only in the z direction.

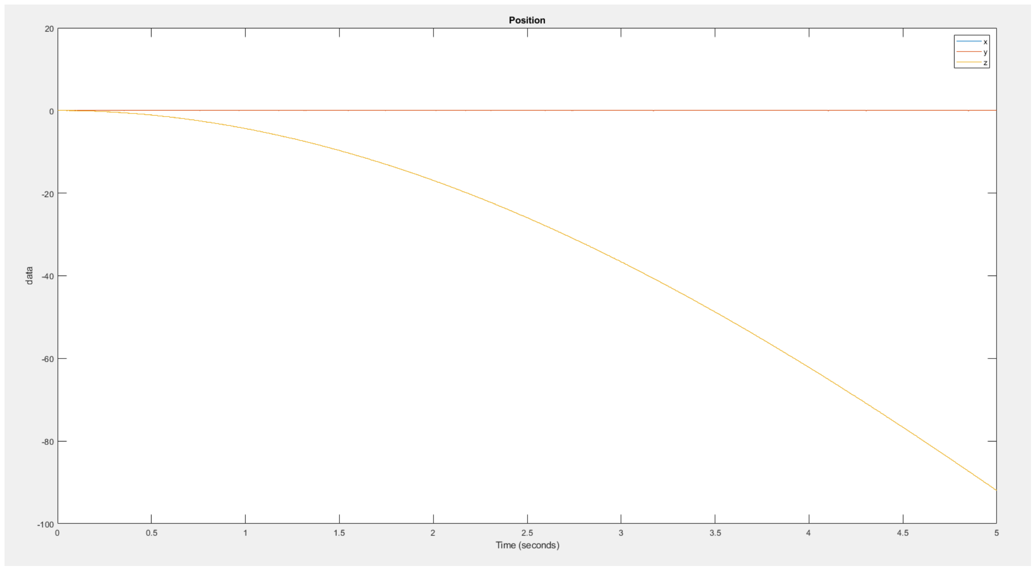

Figure 2: Position

```
k_est =  2.2011e-08

k = 2.2000e-08
```

Listing 4: Values for k

As it can be observed from Listing (4), the value of the estimated k is almost the same as the real value.

2 Estimating I_3 and b

After several experiments, I chose the value 0.68 for c and 20 for the number of segments. Actually, first of all I wrote the code for Ω_1 and I analysed the result of the identification. I noticed that the accuracy was the best around a value for c lower than 0.7 and for a value of 20 for the number of segments.

One knows that when the quadcopter hovers, the following Equation is true:

$$mg = k(2\Omega_1^2 + 2\Omega_2^2) \tag{2}$$

Using Equation (2) and the fact that $\Omega_2 = c\Omega_1$, one can express Ω_1 as in Equation (3).

$$\Omega_1 = \sqrt{\frac{mg}{2k(1+c^2)}} \tag{3}$$

```
c = 0.68
Omega1 = abs(sqrt(m*g/(k*(2+2*c^2))));
...
segments = 20
```

Listing 5: Completed code

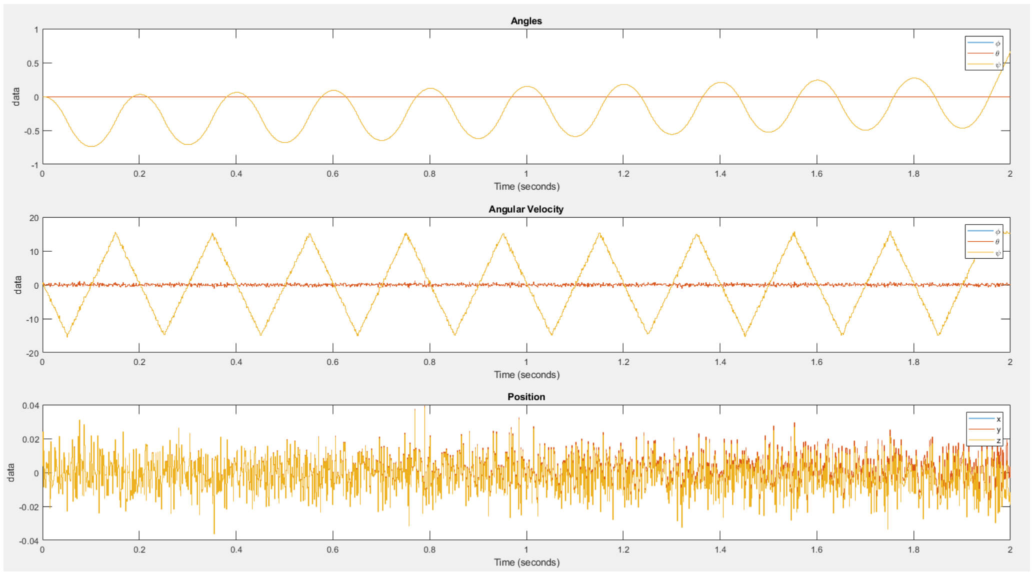

Figure 3: Angles, angular velocity and position of the quadcopter

From Figure 3, it can be seen that the yaw angle is inside the interval $(-\pi, \pi)$ (it is actually inside the interval $(-1, 1)$). Moreover, the other angles are 0. Also, the angular velocity is not zero only for the yaw angle.

In Figure 4 one can see the result of the identification. The model approximates very good the real process, having 95.2% accuracy.

Now, the dynamics are given by Equation (4).

$$\frac{d}{dt}\dot{\Psi} = \frac{b}{I_3}u \tag{4}$$

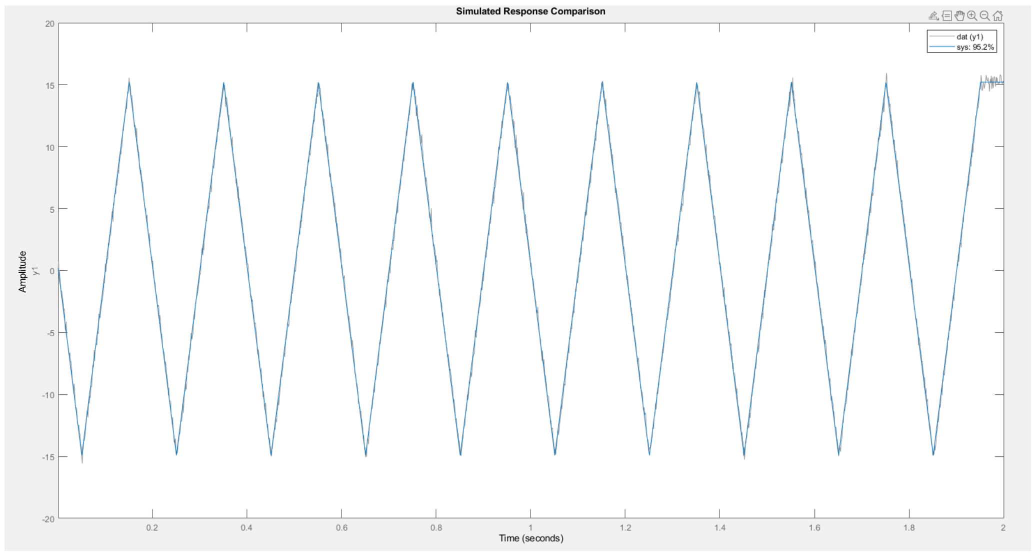

Figure 4: Result of the identification

This corresponds to an integrator in Laplace. Therefore, one can actually say that $K_p = \frac{b}{I_3}$. So b can be easily computed as $K_p * I_3$. We have K_p from the identification. The reason why we can not use this experiment for both b and I_3 is that we have only one K_p from the identified model. So we can only use it to compute either b or I_3.

```
b_est = sys.Kp*I(3)
```

Listing 6: Code for computing the estimation

```
b_est = 1.9939e-09
true_b = 2.0000e-09
```

Listing 7: Results for the estimation

Again, from the numerical results, one can notice that the estimation is very similar to the real value.

3 Estimating I_1 and I_2

This task is very similar to the previous one. After conducting some experiments I chose 0.6 for c and 20 segments. Like in the previous task, first I complete the code for computing Ω_H and Ω_2, then I completed the code for identification. I analysed the results and I adjusted the values for c and the number of segments in order to obtain the best accuracy.

Here it is known that $\Omega_H = \Omega_1$. And in this case, the condition for the quadcopter to hover is:

$$\Omega_2^2(1+c^2) = 2\Omega_H^2 \tag{5}$$

Using Equation (5), one can state that Ω_2 has the following expression:

$$\Omega_2 = \sqrt{\frac{2\Omega_H^2}{1+c^2}} \tag{6}$$

Therefore, the code for finding Ω_H and Ω_2 can be observed in the Listing (8).

```
Omega_H = Omega1; %TODO
c = 0.6; %TODO
Omega2  = abs(sqrt(2*(Omega_H^2)/(1+c^2))); %TODO
Omega4 = c*Omega2;
Omega_in.time = (0:inner_h:2)';
nbr_samples = length(Omega_in.time);
Omega_in.signals.values = zeros(nbr_samples,4);
segments = 20; %TODO
segment_size = floor(nbr_samples/segments)
switch_time = [floor(segment_size/2):segment_size:nbr_samples
    , nbr_samples]
```

Listing 8: Code for finding Ω_H and Ω_2

The next step is to identify the model. This is done like in the previous tasks, creating a data object and then estimating the process. But now the output of the data object is $\dot{\Phi}$ and the input is $-\Omega_2^2 + \Omega_4^2$. The model has zero poles. The code for doing the identification can be seen in Listing (9).

```
Phidot = out.deta.data(:,1);
dat = iddata(Phidot, -out.Omega.data(:,2).^2 + out.Omega.data
    (:,4).^2 ,inner_h ) %TODO
sys = procest(dat,'p0I') %TODO
```

Listing 9: Code for identification

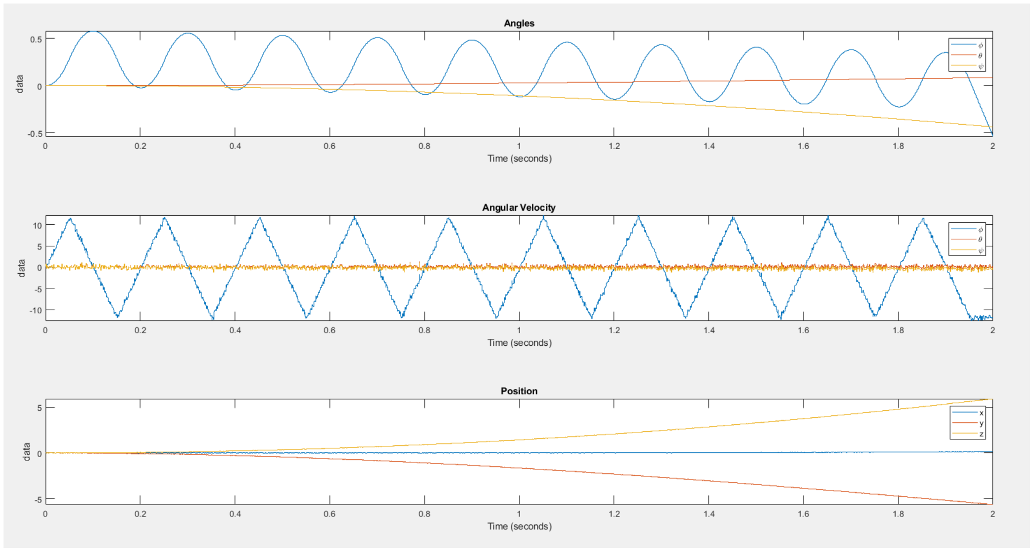

Figure 5: Angles, angular velocity and position

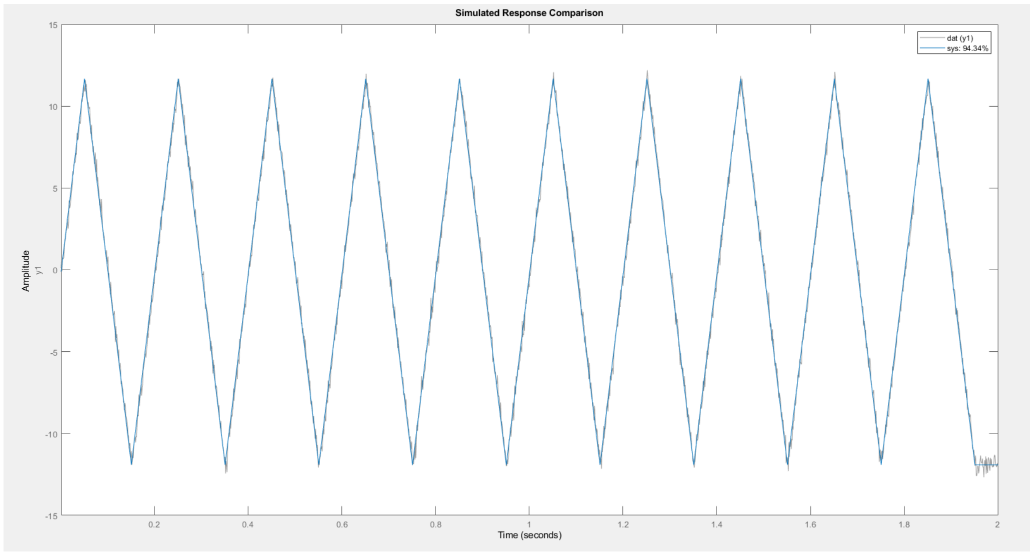

Figure 6: Results of the identification

From Figure 5, it can be seen that the pitch angle is inside the interval $(-\pi, \pi)$ (it is actually inside the interval $(-0.5, 0.5)$). Moreover, the roll angle is 0. Also, the angular velocity is not zero only for the pitch angle.

In Figure 6 one can see the result of the identification. The model approximates very good the real process, having 94.34% accuracy.

Now, the dynamics are given by Equation (7).

$$\dot{\Phi} = \frac{kl}{I_1}\frac{1}{s}u \tag{7}$$

This corresponds again to an integrator. Therefore, one can actually say that $K_p = \frac{kl}{I_1}$. So I_1 can be easily computed as $\frac{kl}{Kp}$. We have K_p from the identification.

```
I1_est = (k_est*l)/sys.Kp %TODO
```

Listing 10: Code for computing the estimation

```
I1_est = 1.6628e-05
true_I = 1.6600e-05
```

Listing 11: Results for the estimation

Again, the result for the estimation is very similar to the real one.

4 Closing the Loop

Analysing Figure 7, it can be observed an overshooting for both the Position and the Angles. For the Position, the transitory period lasts 6.5 seconds. After this, the stationary value for y is 0.1 and 0.0 for x and z. This means that the goal was reached, the quadcopter moved 0.1 m in the y direction, while maintaining the same height z. Only the pitch angle has a transitory period of 4 seconds, the others being 0 all the time. And this is also correct, because we actually control the pitch angle.

5 Estimating a disturbance model

After trying several values, I chose the value of 10.44 for R, as it seemed to be the best. Again, like in the previous tasks, this value was adjusted after observing the results of the estimation (more precisely, after comparing the denominator of the ARMA model with the real denominator and after

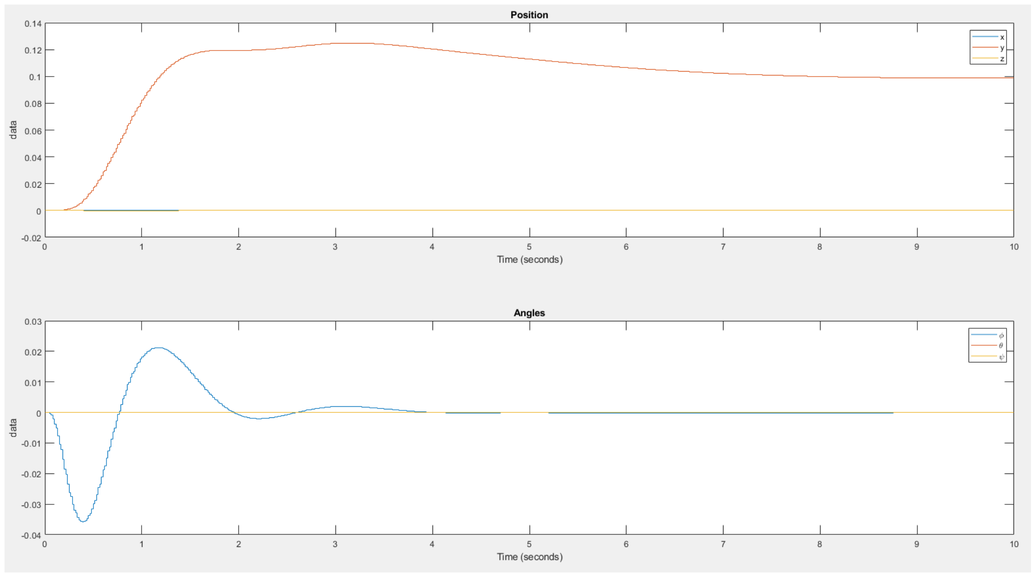

Figure 7: Controlling the quadcopter

observing the Bode plots).

One should actually choose the parameters for the ARMA model. This can be easily done in Maltab, using the *armax* function. This function has two parameters: the estimation data given as an IDDATA object and the order of the ARMAX polynomials. The ARMAX model has the structure from Equation (8).

$$A(q)y(t) = B(q)u(t - nk) + C(q)e(t), \tag{8}$$

where na is the order of A polynomial, nb is the order of B polynomial, nc is the order of C polynomial, and nk is the input delay.

But we do not need an ARMAX model, we actually need an ARMA model. This is a special case of an ARMAX model without input channels. It has the following structure:

$$A(q)y(t) = C(q)e(t), \tag{9}$$

Therefore, one needs to select only the orders na and nc.

```
R = 10.44;%TODO might be needed to change
```

```
...
na = 2;
nc = 1;
armax2 = armax(dat,[na nc]) %compare armax with wind.denum
```

Listing 12: Code for the ARMA model

After trying other values also, I noticed that the best results are achieved when using order 2 for **A** polynomial and order 1 for the second polynomial. This is in accordance with the real model of the wind.

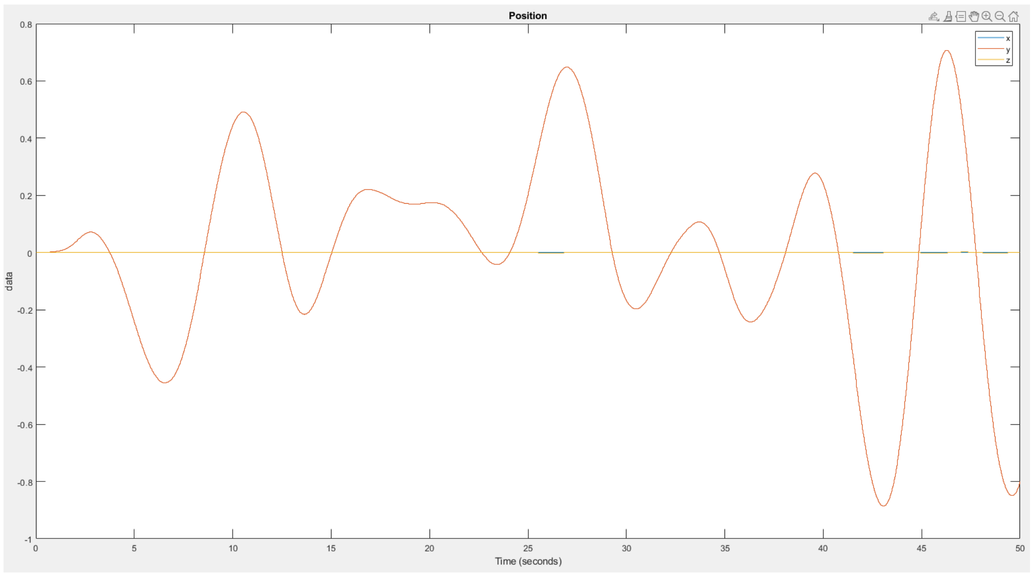

Figure 8: Position of the quadcopter

From Figure 8, it can be seen that the quadcopter moves only on the y axis. This is where the disturbance is. So the quadcopter tries to keep its position, accelerating or decelerating, depending on the disturbance.

From Figure 9, it can be noticed that the model estimates the process very good for all frequencies (the accuracy can be seen in Listing 13). There is a little overfitting when considering the magnitude, but the approximation is still good enough.

```
Status:
Estimated using ARMAX on time domain data "dat".
Fit to estimation data: 97.05% (prediction focus)
FPE: 1.859e-07, MSE: 1.821e-07
```

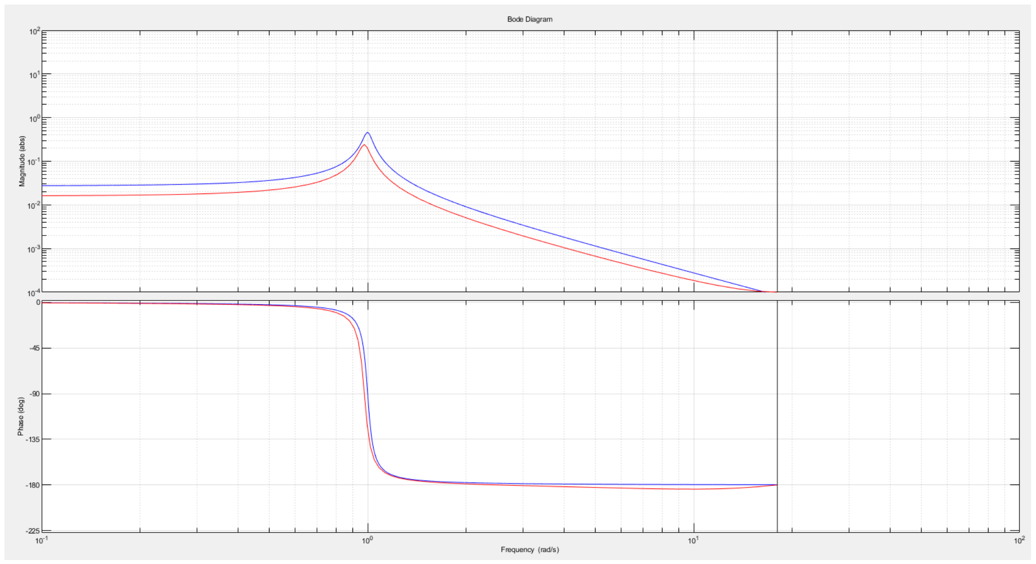

Figure 9: Bode plots

```
ans =

      2.672 s + 35.89
  ------------------------
  s^2 + 0.06502 s + 0.9555
Continuous-time transfer function.
```

Listing 13: Result for the ARMA model

```
>> wind.denum
ans = 1.0000    0.0600    1.0000
```

Listing 14: True denominator

From Listing 14, it can be seen that the true denominator is similar to the denominator of the ARMA model. The ARMA model has (1, 0.06502, 0.9555) for the denominator, whereas the real one is (1, 0.06, 1).

6 Using the Disturbance Model in a control design

From the manual, the disturbance model has the following transfer function:

$$\frac{w_0^2}{s^2 + 2\epsilon w_0 s + w_0^2} \tag{10}$$

From the previous section, the estimation of the disturbance model has the following transfer function:

$$\frac{2.672s + 35.89}{s^2 + 0.06502s + 0.9555} \tag{11}$$

Using Equations (10) and (11), one can identify ϵ as $\frac{0.06502}{2w_0}$ and w_0 as $\sqrt{0.9555}$. These values were added in the code.

```
w0_wind = abs(sqrt(0.9555)); % TODO, change this
epsilon = 0.06502/(2*w0_wind); % TODO, change this
```

Listing 15: Values for w_0 and ϵ

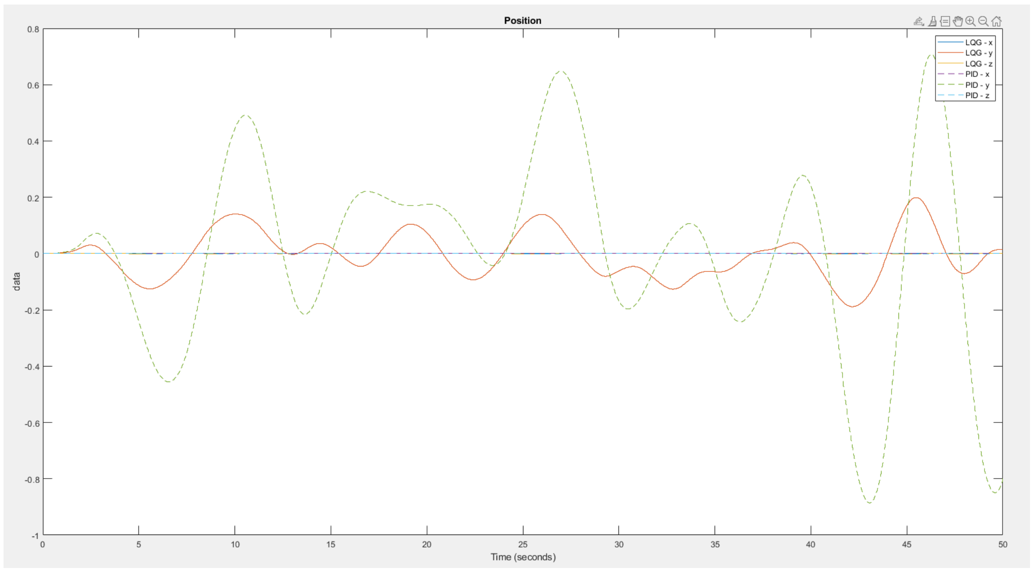

Figure 10: Position of the quadcopter

As it can be observed in Figure 10, the quadcopter moves only on the y axis. But it can be noticed that the LQG controller acts better, because the line is more smooth. The PID controller has large over shootings, the LQG controller and the Kalman filter give a more stable behaviour.