CASE STUDY: QUADCOPTER

MORITZ WOLTER (GROUP 19)

```
-------------------------------------------
   Quadcopter exercise - Simulation Report
-------------------------------------------

Full state vector: estimated by an Observer
Number of checkpoints reached: 7/7
Payload: 0.1 kg

Timing:
-------
From initial pos. to checkpoint 1: 3.550 s
From checkpoint 1 to checkpoint 2: 3.700 s
From checkpoint 2 to checkpoint 3: 4.150 s
From checkpoint 3 to checkpoint 4: 4.450 s
From checkpoint 4 to checkpoint 5: 4.100 s
From checkpoint 5 to checkpoint 6: 2.350 s
From checkpoint 6 to checkpoint 7: 1.200 s

Average time: 3.357 s

-------------------------------------------
```

Project Report

Supervised by Prof. Bart de Moor
Mauricio Agudelo
January 5, 2015 – Version 1

CONTENTS

LIST OF FIGURES

1 MODELING

In this report a state space model for a quadcopter is going to be derived. After testing the model with some simple simulations, a LQR controller based in this model will be designed. In oder to tackle a weight disturbance, integrators are going to be added to this controller. And finally to reach more realistic scenarios Kalman-Filtering will be included into the design.

1.1 FINDING A STATE SPACE MODEL

We are going to use equations from Newtonian mechanics to derive the model for the quadcopter. However since these equations are nonlinear and a linear model is desired the equation system will have to be linearized. To do that an equilibrium point hast to be determined. This point comes from the physics of the quadcopter, it will be in equilibrium, when it is hovering at the origin with zero roll, pitch and yaw angles. Mathematically this means:

$$x = y = z = 0 = \phi = \theta = \psi \tag{1}$$

With this definition we can find our set of linear equations by first order approximation:

$$\frac{d\triangle x}{dt} = \frac{\partial f}{\partial x}|_{x_e,u_e}\triangle x + \frac{\partial f}{\partial u}|_{x_e,u_e}\triangle u. \tag{2}$$

In our state space model the matrix A will be multiplied with the change of the states $\triangle \mathbf{x}$ summed together with the variation of the input $\triangle \mathbf{u}$ times the B matrix it will give the next state of the system at a given point in time. Thats why we can read of these two matrices from equation 2. In practice a lot of partial derivatives have to be found in order to compute A, and B. For brevity these will be skipped here. The desired matrices are:

$$A = \begin{pmatrix} 1 & & & & & & & & & & & \\ & 1 & & & & & & & & & & \\ & & 1 & & & & & & & & & \\ \frac{-k_d}{m} & & & & & g & & & & & & \\ & \frac{-k_d}{m} & & & -g & & & & & & & \\ & & \frac{-k_d}{m} & & & & & & & & & \\ & & & & & & 1 & & & & & \\ & & & & & & & 1 & & & & \\ & & & & & & & & 1 & & & \\ & & & & & & & & & & & \end{pmatrix} \tag{3}$$

In matlab A is given by :

```
A = zeros(12);
A(1,4) = 1;
A(2,5) = 1;
A(3,6) = 1;
A(4,4) = -kd/m;
A(4,8) = g;
A(5,5) = -kd/m;
A(5,7) = -g;
A(6,6) = -kd/m;
A(7,10) = 1;
A(8,11) = 1;
A(9,12) = 1;.
```

Twelve different states are included in the model. The system matrix is therefore element of $\mathbb{R}^{12\times12}$.

In matlab B *is given by:*
```
B = zeros(12,4);
B(6,:) = (k*cm)/m;
B(10,1) = (L*k*cm)/Ixx;
B(10,3) = -(L*k*cm)/Ixx;
B(11,2) = (L*k*cm)/Iyy;
B(11,4) = -(L*k*cm)/Iyy;
B(12,1) = (b*cm)/Izz;
B(12,2) = -(b*cm)/Izz;
B(12,3) = (b*cm)/Izz;
B(12,4) = -(b*cm)/Izz;.
```

$$B = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ \frac{kc_m}{m} & \frac{kc_m}{m} & \frac{kc_m}{m} & \frac{kc_m}{m} \\ & & & \\ & & & \\ & & & \\ \frac{Lkc_m}{I_{xx}} & & -\frac{Lkc_m}{I_{xx}} & \\ & \frac{Lkc_m}{I_{yy}} & & -\frac{Lkc_m}{I_{yy}} \\ \frac{bc_m}{I_{zz}} & -\frac{bc_m}{I_{zz}} & \frac{bc_m}{I_{zz}} & -\frac{bc_m}{I_{zz}} \end{pmatrix} \tag{4}$$

The input matrix is multiplied by the input vector $\mathbf{u}$ and is added to the product of system matrix and state vector. In this case it must be of dimension 12×4.

With the system and input matrix A, B known all that is left to do is to find the output and feedtrough matrices and the state space representation is complete. In real systems the feedtrough matrix is always zero. This leads to $D = 0$. As we have six outputs and 4 inputs the zero is the zero matrix of dimension 6×4. As the output is defined to be $\mathbf{y} = [x, y, z, \phi, \theta, \psi]^T$. We choose our output matrix to pick these values out of the state vector $\mathbf{x} = [x, y, z, v_x, v_y, v_z, \phi, \theta, \psi]^T$:

C may be implemented using the code:
```
C = zeros(6,12);
C(1,1) = 1;
C(2,2) = 1;
C(3,3) = 1;
C(4,7) = 1;
C(5,8) = 1;
C(6,9) = 1;.
```

$$C = \begin{pmatrix} 1 & & & & & & & & & & & \\ & 1 & & & & & & & & & & \\ & & 1 & & & & & & & & & \\ & & & & & 1 & & & & & & \\ & & & & & & 1 & & & & & \\ & & & & & & & 1 & & & & \end{pmatrix} \tag{5}$$

As the model has 6 outputs and 12 states C is element of $\mathbb{R}^{6\times12}$. So far the report included the state space matrices only in their symbolic form. A numerical form is given in figure 1.

1.2 DISCRETIZATION AND SIMULATION

As the controller is going to be implemented on a small computer within the quadcopter, a discrete state space model is required. Before discretizing the model it makes sense to perform some small simulations to make shure the model leads to realistic values and is

```
>> A

A =

         0         0         0    1.0000         0         0         0         0         0         0         0         0
         0         0         0         0    1.0000         0         0         0         0         0         0         0
         0         0         0         0         0    1.0000         0         0         0         0         0         0
         0         0         0   -0.5000         0         0         0    9.8100         0         0         0         0
         0         0         0         0   -0.5000         0   -9.8100         0         0         0         0         0
         0         0         0         0         0   -0.5000         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0    1.0000         0         0
         0         0         0         0         0         0         0         0         0         0    1.0000         0
         0         0         0         0         0         0         0         0         0         0         0    1.0000
         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

>> B

B =

         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
    0.0600    0.0600    0.0600    0.0600
         0         0         0         0
         0         0         0         0
         0         0         0         0
    1.5000         0   -1.5000         0
         0    1.5000         0   -1.5000
    0.1000   -0.1000    0.1000   -0.1000

>> C

C =

     1     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     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     1     0     0     0     0
     0     0     0     0     0     0     0     0     1     0     0     0
```

Figure 1: The state space Matrices in the matlab command window.

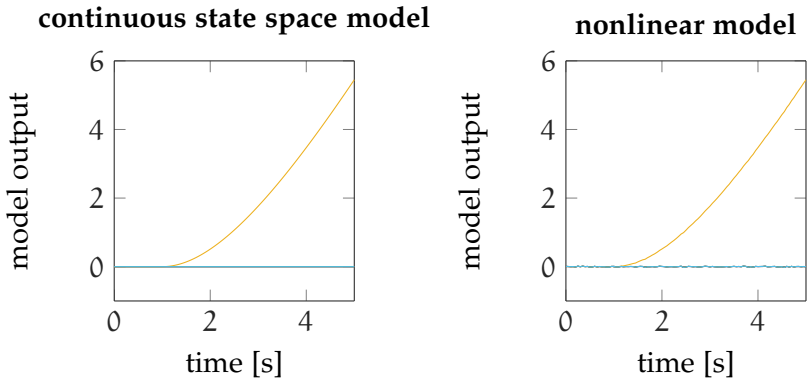

Figure 2: step response of the linear and nonlinear model

free of mistakes. Simulation results are given in figure 2. The linear model responds to a step input like the nonlinear model. This is a first indication, that the derived model is actually correct.

The discretization is done using bilinear transformations according to the formulae:

$$A_d = (I - \frac{AT_s}{2})^{-1}(I + \frac{AT_s}{2}). \tag{6}$$

$$B_d = (I - \frac{AT_s}{2})^{-1}BT_s. \tag{7}$$

$$C_d = C(I - \frac{AT_s}{2})^{-1}. \tag{8}$$

$$D_d = D + C(I - \frac{AT_s}{2})^{-1}\frac{BT_s}{2}. \tag{9}$$

The result is identical to the return values of the built in *matlab* function `c2d` when it is run with the correct sample time $T_s = 0.05\,s$ and the `'tustin'` option, which refers to the bilinear transformation.

2 CONTROL DESIGN

Armed with the discretized model we can now proceed to design a controller for the quadcopter. The challenge is to track a given reference signal as accurately as possible, in order to do that the reference will have to be introduced properly into the controller.

2.1 FULL STATE LQR

The reference signal will be handled by two matrices. $N_x \in \mathbb{R}^{12\times3}$ which links the twelve states to the three entries in the reference vector and $N_u \in \mathbb{R}^{4\times3}$, which links the three reference signals to the four plant inputs. In oder to compute these crucial matrix gains we consider the system:

$$\begin{pmatrix} A - I & B \\ \hat{C} & \hat{D} \end{pmatrix} \begin{pmatrix} N_x \\ N_u \end{pmatrix} = \begin{pmatrix} 0 \\ I \end{pmatrix} \tag{10}$$

However if C and D remain unchanged this approach will lead to false dimensions. Thats why they are shown with hats in equation 10 above. $\hat{C}$ and $\hat{D}$ are truncated versions of C and D containing only their first three rows. With the dimensions fixed after computing a pseudo inverse N_x and N_u may be found from:

In the code $\hat{C} =$ `C(1:3,:)` *and* $\hat{D} =$ `D(1:3,:)`.

$$\begin{pmatrix} N_x \\ N_u \end{pmatrix} = \begin{pmatrix} A - I & B \\ \hat{C} & \hat{D} \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ I \end{pmatrix} \tag{11}$$

When looking at this equation it is important to keep in mind that taking the pseudo inverse flips the dimensions to 16×15 for the augmented matrix, which is inversed. With the two reference gain matrices found only the feedback gain Matrix K remains to be determined. As in this project LQR control is used this gain is fund by minimizing the cost function:

$$V = \sum (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u}) \tag{12}$$

In oder to find reasonable Q and R matrices, weights are introduced. They are based on physical properties and may be used to change a certain characteristic of the controller. The following weights where used: A position weight w_{pos}, a height weight w_z a angle weight w_{angle}, a velocity weight w_v, an angular velocity weight w_{angv}, a stability weight w_{stab} and finally an input weight w_{inp}. This leads to the weighting matrix $Q \in \mathbb{R}^{12\times12}$ with the vector $\mathbf{q} = (w_{pos}, w_{pos}, w_{pos}$.

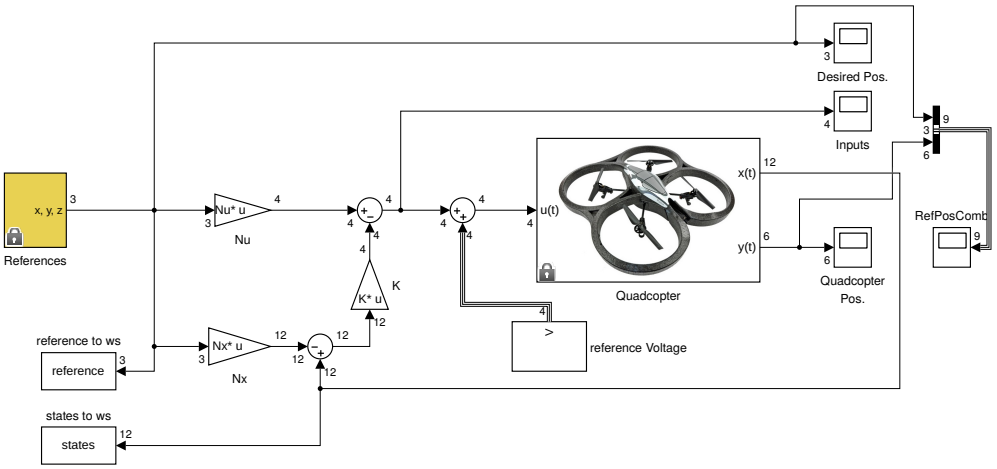

Figure 3: The simulink template used for LQR-Control with full state feedback.

In matlab this matrix in implemted using: `Q = diag(q);`.

$w_z, w_v, w_v, w_v, w_{angle}, w_{angle} \cdot w_{stab}, w_{angle}, w_{angv}, w_{angv} \cdot w_{stab}, w_{angv}$) on the diagonal. The input weight sits on the diagonal of the $R \in \mathbb{R}^{4\times4}$ matrix. Increasing a weight will make that property more important, as the optmization algorithm will put more effort into minimizing that particular variable. With these tuning parameters found by systematic trial and error. The quadcopter is able to reach the waypoints on it's path in within a two second average. Selected results are shown in figure 4[1]. However in a real setting one would probably put more weight on the inputs and on stability to make sure actuator saturation or flipping does not occur. However in this assignment the goal was to finish the course as fast as possible so a very agressive controller was designed. If a payload of 0.1 kg is added to the quadcopter the results are not satisfactory. Actuators are saturated, proper flight height is not reached and consequently all checkpoints are missed.

1 The following weights where used: $w_z = 25, w_{pos} = 9.5, w_{angle} = 0.4, w_v = 0.4, w_{angv} = 1.1, w_{stab} = 2, w_{inp} = 0.1$.

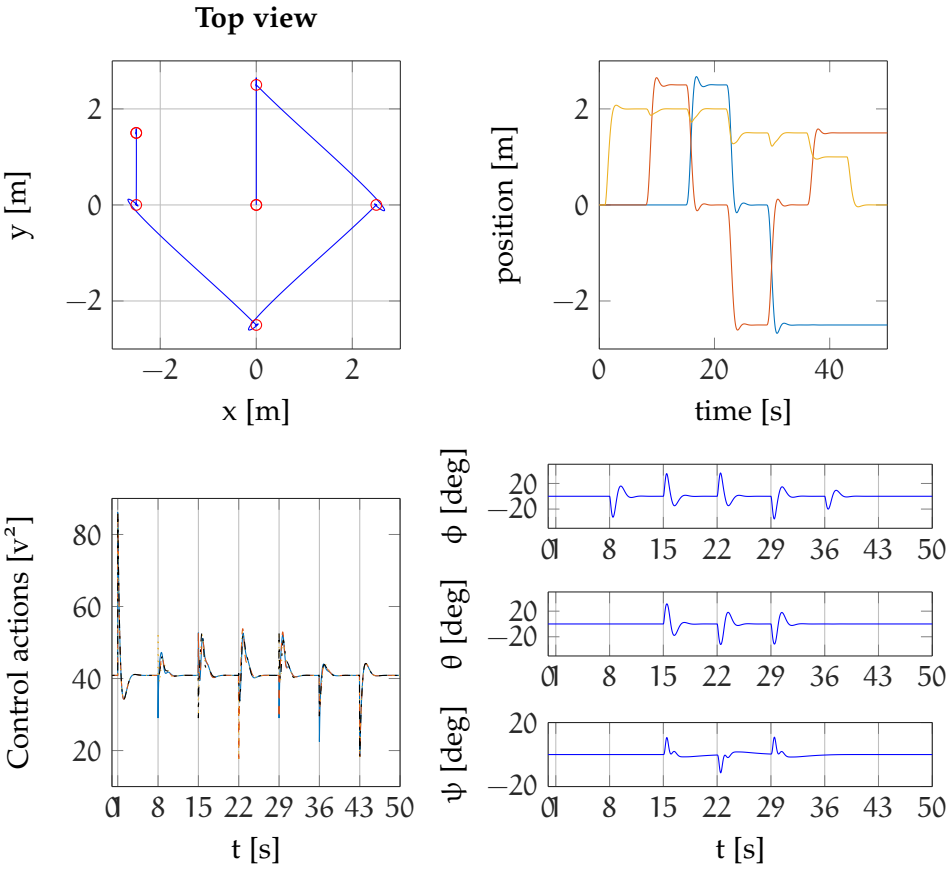

Figure 4: Selected simulation results of the LQR controller without integration. The top view of the flight is shown in the top left. The evolution of the x, y and z coordinates are shown in the top right. The control actions are shown in the bottom left. The development of the quadcopter angles are shown in the bottom right.

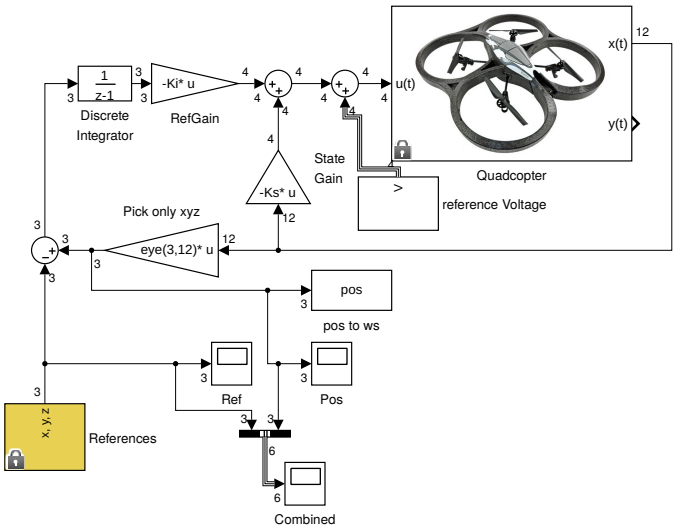

Figure 5: The simulink template of the controller with integral action.

2.2 FULL STATE LQR WITH INTEGRATOR

In order to be able to fly with payload and possibly even speed up the flight time of the quadcopter without payload a controller with integrator is going to be set up. In order to do that extra states are added to the system which integrate the control error $e_k = y_k - r_k$. This leads to the augmented state equations:

$$\begin{pmatrix} x_{I_{k+1}} \\ x_{k+1} \end{pmatrix} = \begin{pmatrix} I & \hat{C} \\ 0 & A \end{pmatrix} \begin{pmatrix} x_{I_k} \\ x_k \end{pmatrix} + \begin{pmatrix} \hat{D} \\ B \end{pmatrix} u_k - \begin{pmatrix} I \\ 0 \end{pmatrix} r_k \tag{13}$$

Again $\hat{C}$ and $\hat{D}$ denote the truncated versions of C and D. The system remains controllable[2]. The layout used to add integral action to the controller is given in figure 5. It implements the control law:

$$u_k = -\begin{pmatrix} K_i & K_s \end{pmatrix} \begin{pmatrix} x_{I_k} \\ x_k \end{pmatrix}. \tag{14}$$

Again $K = (K_i \; K_s)$ is computed by minimizing the cost function 12. However an additional weight for the integrated error w_{int} has to be introduced. These weights occupy the first three spots of $Q \in \mathbb{R}^{15}$.

2 The rank of the controllability matrix $C(A, B) = (B \; AB \; A^2B \; \ldots \; A^{n-1}B)$ remains 15.

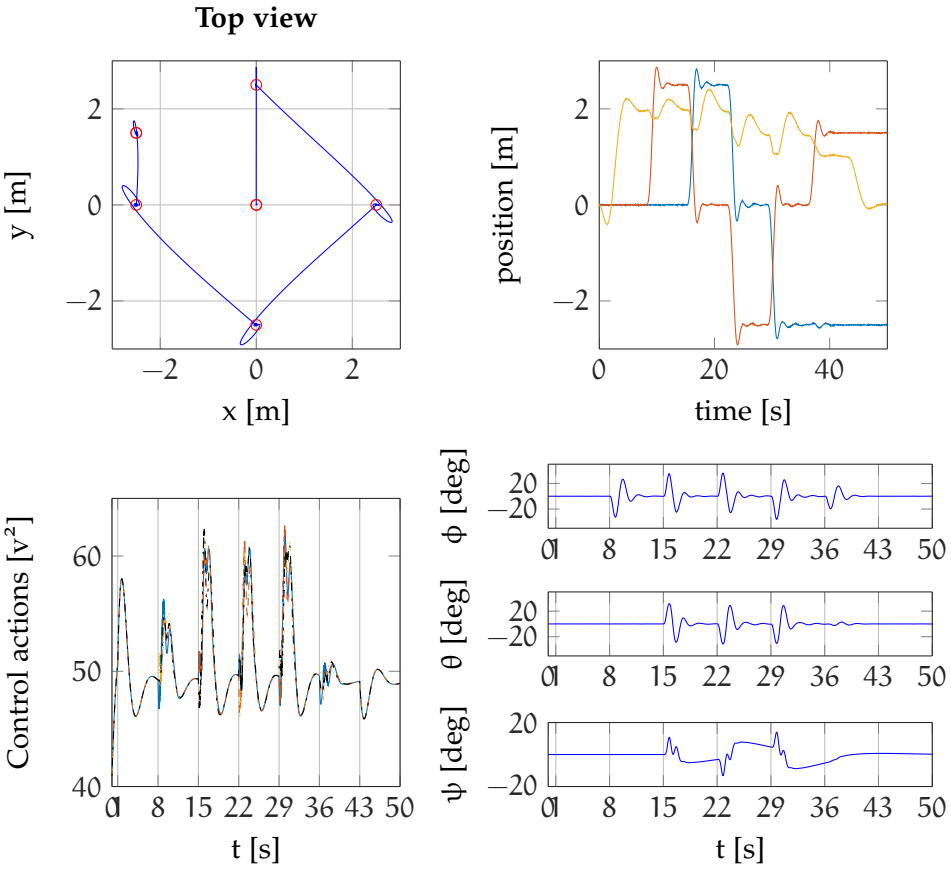

Figure 6: Selected simulation results of the LQR controller with integration and a payload of 0.1 kg. The top view of the flight is shown in the top left. The evolution of the x, y and z coordinates are shown in the top right. The control actions are shown in the bottom left. The development of the quadcopter angles are shown in the bottom right.

Results with the optimized K with payload are shown in figure 6[3]. The quadcopter is able to handle the payload with integral control. However the average time until a checkpoint is reached increases to 4.46 s without payload and to 4.6 s with payload.

2.3 LQG WITH KALMAN-FILTERING

In the previous sections we where feeding back the state vector. However in practice it is often impossible to measure the entire state vector. Therefore some states have to be estimated before the state vector can be fed back to the controller. This estimation is done by using Kalman-Filtering. Kalman filtering estimates the current state by using measurement and model information. It starts by computing the

3 The following weights where used: $w_{int} = 2$, $w_z = 1$, $w_{pos} = 1$, $w_{ang} = 5$, $w_v = 1$, $w_{angv} = 1.1$, $w_{stab} = 10$, $w_{inp} = 2$

model output to the inputs that are actually fed into the plant. The plant output is fed back into the filter, where it is compared to the predicted output. The estimated states are then corrected according to the prediction error found from the predicted output and the measured output. In order to do this correction properly information on the noise in the system is required, namely the covariance matrices of the process and measurement noise is necessary to compute the Kalman-gain. The Kalman gain weights the effect that observation and prediction have and the state estimation. If the measurements are of very high quality a large gain should put the weight here. On the other hand if there is a lot of noise on the measurements and little noise in the process a low gain should put more weight on the model prediction.

2.3.1 *Matlab's default filter block*

Covariance is generalisation of correlation with mean adjustment

In order to choose the noise covariance matrices in this example white Gaussian noise is assumed, since no further information on the noise distribution is available. Therefore as Gaussian white noise is only correlated with itself the covariance matrices will be diagonal, with specific noise variances on the diagonal. For the measurement noise the specific variances are known, $\sigma^2_{pos} = 2.5 \cdot 10^{-5}$ on the position measurements and $\sigma^2_{ang} = 7.57 * 10^{-5}$ on data from the angle sensors, this leads to the measurement noise covariance matrix with the vector:

$$q_{var} = \begin{pmatrix} \sigma^2_{pos} & \sigma^2_{pos} & \sigma^2_{pos} & \sigma^2_{ang} & \sigma^2_{ang} & \sigma^2_{ang} \end{pmatrix}^T \tag{15}$$

on the diagonal. No data is available for the process noise. After some testing with low variance values on the diagonal it turns out that the process covariance matrix with $\sigma^2_{state} = 10^{-6}$ on the diagonal produces good results as shown figure 8. The simulation was run with a payload of 0.1kg. The filter turns out to be very robust. After some retuning of the controller the quadcopter takes almost no additional time to fly trough the parcour with payload in comparison to integral control with full state feedback. However the noise is clearly visible in the angle measurements. If the payload is removed the quadcopter is able to fly faster, overall tracking performance remains unchanged.

2.3.2 *Manual Kalman filter design*

In order go gain additional insight into the way the Kalman filter works the default block will now be replaced by a combination of standard blocks which produce the same result. In order to obtain the required setup we implement the equation:

$$\hat{x}_{k+1|k} = A\hat{x}_{k|k-1} + Bu(k) + L_k(y_k - C\hat{x}_{x|k-1}). \tag{16}$$

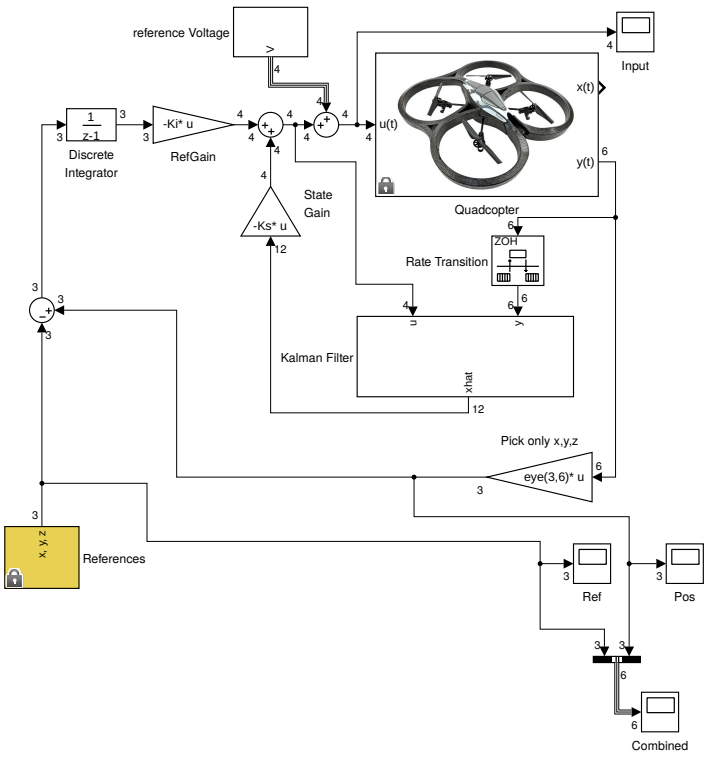

Figure 7: Control template with Kalman filter.

Where A, B and C are the discretized system matrices. L_k denotes the Kalman gain, which weights the plant measurements as described earlier. The simulation template is given in figure 9. The simulation results are similar to what was obtained by using the default block. However the size of the process noise covariance matrix changes. In this setting it is assumed, that the process noise enters the filter trough the inputs. Additionally this implementation is not as robust as the default block. In order to be able to correctly fly with the payload the noise covariance matrix has to be rescaled. Which leads to the new process noise matrix $Q = 3 \cdot 10^{-2} \cdot I_{4\times4}$.

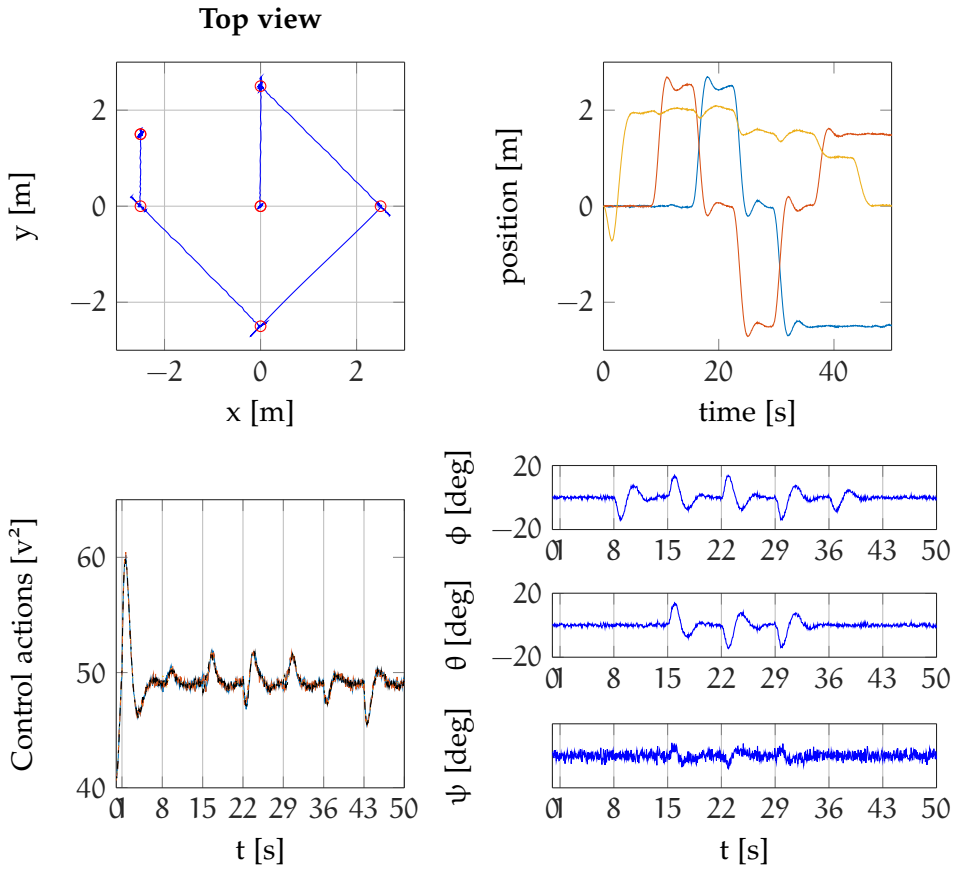

Figure 8: Selected simulation results of the LQR controller with integration, a payload of 0.1 kg and state estimation trough Kalman-filtering. The top view of the flight is shown in the top left. The evolution of the x, y and z coordinates are shown in the top right. The control actions are shown in the bottom left. The development of the quadcopter angles are shown in the bottom right.

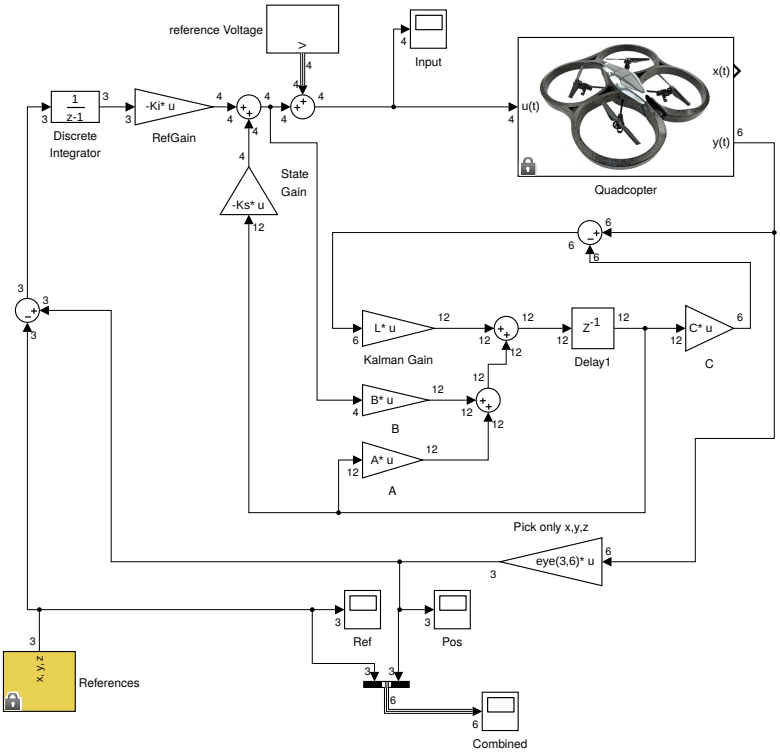

Figure 9: Kalman filtering using only delay, gain, and summation blocks

3 CONCLUSION

In this report went trough all the major steps of the control-design-process. It started with model generation and linearizion, continued with discretization and finally looked at controller design in three different settings:

1. LQR control without integrator and full state feedback.
2. LQR control with integrator and full state feedback.
3. LQR control with integrator and LQE.

In all three settings satisfactory control results where achieved. However only in the first setting was the controller tuned for speed. Furthermore did it turn out to be impossible to come up with a controller that could handle the payload without using an integrator. So in the first case excellent results where observed under optimal conditions. Unfortunately did the controller turn out very vulnerable to disturbances without an integrator. With integration added disturbances like the payload could be handled well by the controllers that where set up. Unfortunately the agility that came from pure LQR control was no achieved with integration, however spending more effort an tuning a more aggressive controller might change the situation. Surprisingly did the situation change little with state estimation. Only small delays occurred when the full state vector was estimated from the model and the plant outputs before it was fed back into the controller. Overall the project turned out very helpful in gaining some first experience in digital control design.