31388 Advanced Autonomous Robots

Group 12 - June 2021

Aksel Christian Møller, s173935, 25%
Mikkel Pedersen, s174831, 25%
Sebastian David Kusio, s174849, 25%
Victor Girardiner Flindt, s174822, 25%

I Part 01

A. *Exercise 1.1*

The rotation matrices for the Z- and X-dimensions are given by:

$$R_z = \begin{pmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

By multiplying the matrices in the Z·X·Z order, the rotation matrix corresponding to the Euler angels ZXZ can be found:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{pmatrix}$$

$$R_{total} = R_z * R_x * R_z =$$

$$\begin{pmatrix} \cos(\psi)^2 - \cos(\phi)\,\sin(\psi)^2 & \sigma_3 + \sigma_1 & \sigma_2 \\ -\sigma_3 - \sigma_1 & \cos(\phi)\,\cos(\psi)^2 - \sin(\psi)^2 & \sigma_4 \\ \sigma_2 & -\sigma_4 & \cos(\phi) \end{pmatrix}$$

where:

$$\sigma_1 = \cos(\phi)\,\cos(\psi)\,\sin(\psi) \quad \sigma_2 = \sin(\phi)\,\sin(\psi) \quad \sigma_3 = \cos(\psi)\,\sin(\psi) \quad \sigma_4 = \cos(\psi)\,\sin(\phi)$$

B. *Exercise 1.2*

First the rotation matrix corresponding to the set of Euler angles ZYZ is found:

$$R_z = \begin{bmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix}$$

$$R_{zyz} = R_z R_y R_z = \begin{bmatrix} c^2(\psi) * c(\theta) - s^2(\psi) & c(\psi) * c(\theta) * s(\psi) + s(\psi) * c(\psi) & c(\psi) * s(\theta) \\ -c(\psi) * c(\theta) * s(\psi) - s(\psi) * c(\psi) & -s^2(\psi) * c(\theta) + c^2(\psi) & -s(\psi) * sin(\theta) \\ c(\psi) * s(\theta) & -s(\psi) * s(\theta) & c(\theta) \end{bmatrix}$$

When $sin(\theta) = 0$ then $\theta = 0$. Thereby making the rotation matrix

$$R_{zyz} = R_z R_y R_z = \begin{bmatrix} c^2(\psi) - s^2(\psi) & 2 * c(\psi) * s(\psi) & 0 \\ -2 * c(\psi) * s(\psi) & -s^2(\psi) + c^2(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This means the drone would only rotate around the z axis.

C. Exercise 1.3

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{bmatrix}$$

To construct the rotation matrix the three matrices R_z, R_y and R_x are used.

$$R_{zyx} = R_z R_y R_x \begin{bmatrix} c(\psi) * c(\theta) & c(\psi) * sin(\theta) * s(\phi) - s(\psi) * c(\phi) & c(\psi) * s(\theta) * c(\phi) + s(\psi) * s(\phi) \\ s(\psi) * c(\theta) & s(\psi) * s(\theta) * s(\phi) + c(\psi) * c(\phi) & s(\psi) * s(\theta) * c(\phi) - c(\psi) * s(\phi) \\ -s(\theta) & c(\theta) * s(\phi) & c(\theta) * c(\phi) \end{bmatrix}$$

When $cos(\theta) = 0$ then $\theta = \pi/2$. We get the following rotation matrix:

$$R_{zyx} = R_z R_y R_x \begin{bmatrix} 0 & c(\psi) * s(\phi) - s(\psi) * c(\phi) & c(\psi) * c(\phi) + s(\psi) * s(\phi) \\ 0 & s(\psi) * s(\phi) + c(\psi) * c(\phi) & s(\psi) * c(\phi) - c(\psi) * s(\phi) \\ -1 & 0 & 0 \end{bmatrix}$$

By making the 90 degree rotation, a gimbal lock occurs. Because of this a degree of freedom is lost, since the x and z plane coincide. Therefore the plane is no longer in 3D but in 2D.

D. Exercise 1.4

This example assumes rotation from $\mathbf{v_1}$ to $\mathbf{v_2}$. In which the scalar of the quaternion can be found taking the dot-product of the 2 unit vectors.

$$\begin{bmatrix} \omega \end{bmatrix} = dot(v_1, v_2)$$

The vector part of the quaternion is calculated using the cross product of the 2 unit vectors.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = Crossproduct(v_1, v_2)$$

combining this as such:

$$q1 = \begin{bmatrix} \omega \\ x \\ y \\ z \end{bmatrix}$$

Results in the desired quaternion describing rotation from vector $\mathbf{v_1}$ unto vector $\mathbf{v_2}$. If the vectors are not normalised, the sum of the normalised lengths needs to be added when doing the cross product of the 2 vectors, this is not needed for this example since all vectors are assumed to be unit vectors.

E. Exercise 1.5.1 & 1.5.2

Finding the q1 quaternion, through the rotation matrix describing rotation of the X-axis.

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\psi) & -sin(\psi) \\ 0 & sin(\psi) & cos(\psi) \end{bmatrix}, \psi = 180°$$

where each cell in the rotation matrix is denoted with the following indexing:

$$\begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix}$$

Now the trace value can be calculated:

$$Trace(M) = M_{11} + M_{22} + M_{33}$$

finally the vector and scalar part of the quaternion can be calculated. q_1 denoting the scalar part, and q_2-q_4 the vector part of the quaternion.

$$|q_1| = \sqrt{(Trace(M)+1)/4} = 0$$
$$|q_2| = \sqrt{(M_{11})/2 + (1 - Trace(M))/4} = 1$$
$$|q_3| = \sqrt{(M_{22})/2 + (1 - Trace(M))/4} = 0$$
$$|q_4| = \sqrt{(M_{33})/2 + (1 - Trace(M))/4} = 0$$

$$q1 = [0, 1, 0, 0]^T$$

Calculating the q2 quaternion follows the same approach as above, though with the rotation matrix switched to a rotation matrix describing rotation around the z-axis.

$$M = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \psi = 180°$$

Then calculating the values for the quaternion

$$|q_1| = \sqrt{(Trace(M)+1)/4} = 0$$
$$|q_2| = \sqrt{(M_{11})/2 + (1 - Trace(M))/4} = 0$$
$$|q_3| = \sqrt{(M_{22})/2 + (1 - Trace(M))/4} = 0$$
$$|q_4| = \sqrt{(M_{33})/2 + (1 - Trace(M))/4} = \sqrt{2}/2$$

$$q_2 = [0, 0, 0, \sqrt(2)/2]^T$$

F. Exercise 1.5.3

Having found the quaternions q_1 and q_2 from previous calculations, determining the quaternion product from the 2 quaternions is done by following the quaternions product matrix:

$$\begin{pmatrix} a_1\,a_2 - b_1\,b_2 - c_1\,c_2 - d_1\,d_2 \\ a_1\,b_2 + a_2\,b_1 + c_1\,d_2 - c_2\,d_1 \\ a_1\,c_2 + a_2\,c_1 - b_1\,d_2 + b_2\,d_1 \\ a_1\,d_2 + a_2\,d_1 + b_1\,c_2 - b_2\,c_1 \end{pmatrix} = \begin{pmatrix} n_0 \\ n_1 \\ n_2 \\ n_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -\frac{\sqrt{2}}{2} \\ 0 \end{pmatrix}$$

Quaternions represent any rotation in 3D using a 4 element vector, the last three elements n_1-n_3 is the vector part of the quaternion. n_0 represents a scalar of the quaternion, especially it describes the amount of rotation that should be done around the vector part.

Since the product of the two quaternions q_1 and q_2 equals:

$$\begin{pmatrix} 0 \\ 0 \\ -\frac{\sqrt{2}}{2} \\ 0 \end{pmatrix}$$

it is possible to calculate both axis of rotation and amount of rotation. The third column of the vector specifies a rotation about Y-axis, and the following equations is used to calculate the specific amount of rotation.

$$q_y = a_y * sin(\frac{angle}{2}) \approx 1.5707, a_y = 1, angle = -\frac{\sqrt{2}}{2}$$

$$degrees = rad2deg(1.5707) \approx 90°$$

G. *Exercise 1.6*

nr.1:
The formula for composing 2 3x3 matrices is the following:

$$A * B = \begin{bmatrix} a_{1,1} \cdot b_{1,1} + a_{1,2} \cdot b_{2,1} + a_{1,3} \cdot b_{3,1} \\ a_{2,1} \cdot b_{1,2} + a_{2,2} \cdot b_{2,2} + a_{2,3} \cdot b_{3,2} \\ a_{3,1} \cdot b_{1,3} + a_{3,2} \cdot b_{2,3} + a_{3,3} \cdot b_{3,3} \end{bmatrix}$$

This means that we need a total of 27 multiplications and 18 additions
nr.2:
The formula for composing 2 quaternions is the following:

$$q1 * q2 = \begin{bmatrix} a1 * a2 - b1 * b2 - c1 * c2 - d1 * d2 \\ a1 * b2 + b1 * a2 + c1 * d2 - d1 * c2 \\ a1 * c2 - b1 * d2 + c1 * a2 + d1 * b2 \\ a1 * d2 + b1 * c2 - c1 * b2 + d1 * a2 \end{bmatrix}$$

This means that we need a total of 16 multiplications and 12 additions.

nr.3:
The formula for composing a 1x3 with a 3x3 (a vector and a matrix) is the following:

$$M * V = \begin{bmatrix} M_{1,1} \cdot V_{1,1} + M_{1,2} \cdot V_{2,1} + M_{1,3} \cdot V_{3,1} \\ M_{2,1} \cdot V_{1,1} + M_{2,2} \cdot V_{2,1} + M_{2,3} \cdot V_{3,1} \\ M_{3,1} \cdot V_{1,1} + M_{3,2} \cdot V_{2,1} + M_{3,3} \cdot V_{3,1} \end{bmatrix}$$

This is then 9 multiplications and 6 additions.
nr.4:

$$V_B = q_i^b \begin{bmatrix} 0 \\ v_x \\ v_y \\ v_z \end{bmatrix} (q_i^b)^{-1} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \begin{bmatrix} 0 \\ v_x \\ v_y \\ v_z \end{bmatrix} \begin{bmatrix} a \\ -b \\ -c \\ -d \end{bmatrix} = \begin{bmatrix} (b * v_x + c * v_y + d * v_z) * a \\ -(b * v_x + c * v_y + d * v_z) * b \\ -(b * v_x + c * v_y + d * v_z) * c \\ -(b * v_x + c * v_y + d * v_z) * d \end{bmatrix}$$

This results in 16 multiplications and 8 additions.

II Part 02

A. *Exercise 2.1*

$$R_{body-fixedwrtinitial} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & sin(\theta) \\ 0 & -sin(\theta) & cos \end{bmatrix}$$

In the modeling lecture the rotation matrix representing the orientation of the body-fixed frame w.r.t the inertial frame is given as:

$$\boldsymbol{R(\gamma)} = \begin{bmatrix} c(\psi) * c(\theta) & c(\psi) * s(\theta) * s(\phi) - s(\psi) * c(\phi) & c(\psi) * s(\theta) * c(\phi) + s(\psi) * s(\phi) \\ s(\psi) * c(\theta) & s(\psi) * s(\theta) * s(\phi) + c(\psi) * cos(\phi) & s(\psi) * s(\theta) * c(\phi) - c(\psi) * s(\phi) \\ -s(\theta) & c(\theta) * s(\phi) & c(\theta) * c(\phi) \end{bmatrix}$$

B. Exercise 2.2

From the lecture on modelling it can be found that the relation between angular velocity $\dot{\theta}$ and rotational velocity w:

$$w = \begin{bmatrix} 1 & 0 & -s(\theta) \\ 0 & c(\phi) & c(\theta) * s(\phi) \\ 0 & -s(\phi) & c(\theta) * c(\phi) \end{bmatrix} * \dot{\theta}$$

C. Exercise 2.3

The linear dynamic equation for the drone is given by:

$$\boldsymbol{m} * \ddot{\boldsymbol{x}} = \begin{bmatrix} 0 \\ 0 \\ -m * g \end{bmatrix} + R * F_B + F_D$$

Where the mass and the gravitational acceleration are $m = 0.5kg$ and $g = -9.81m/s$ respectively

Furthermore

$$\boldsymbol{m} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{bmatrix}$$

$$\ddot{\boldsymbol{x}} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}$$

And the rotation matrix is given by:

$$R = \begin{bmatrix} c(\psi) * c(\theta) & c(\psi) * s(\theta) * s(\phi) - s(\psi) * c(\phi) & c(\psi) * s(\theta) * c(\phi) + s(\psi) * s(\phi) \\ s(\psi) * c(\theta) & s(\psi) * s(\theta) * s(\phi) + c(\psi) * cos(\phi) & s(\psi) * s(\theta) * c(\phi) - c(\psi) * s(\phi) \\ -s(\theta) & c(\theta) * s(\phi) & c(\theta) * c(\phi) \end{bmatrix}$$

$$F_B = \sum_{i=1}^{4} f_i = k * \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} w_i^2 \end{bmatrix} = k * \begin{bmatrix} 0 \\ 0 \\ w_1^2 + w_2^2 + w_3^2 + w_4^2 \end{bmatrix}$$

where $k = 0.01N\frac{s^2}{rad^2}$

Lastly,

$$F_D = \begin{bmatrix} -k_D * \dot{x} \\ -k_D * \dot{y} \\ -k_D * \dot{z} \end{bmatrix}$$

where $k_D = 0.01\frac{N*s}{m}$.

The angular dynamic equation of the drone is given by:

$$\dot{w} = \begin{bmatrix} \tau_\phi * I_{xx}^{-1} \\ \tau_\theta * I_{yy}^{-1} \\ \tau_\psi * I_{zz}^{-1} \end{bmatrix} - \begin{bmatrix} \frac{I_{yy} - I_{zz}}{I_{xx}} w_y w_z \\ \frac{I_{zz} - I_{xx}}{I_{yy}} w_x w_z \\ \frac{I_{xx} - I_{yy}}{I_{zz}} w_x w_y \end{bmatrix}$$

Where τ_ϕ,τ_θ and τ_ψ are:

$$\tau_\psi = b(w_1^2 - w_2^2 + w_3^2 - w_4^2)$$

$$\tau_\theta = Lk(w_2^2 - w_4^2)$$

$$\tau_\phi = Lk(w_1^2 - w_3^2)$$

The constants are the following values:

$$b = 0.001Nm\frac{s^2}{rad^2}$$

$$k = 0.01Nm\frac{s^2}{rad^2}$$

$$L = 0.225$$

$$I_{xx} = I_{yy} = 3e^{-6}Nms^2/rad$$

$$I_{zz} = 1e^{-5}Nms^2/rad$$

To see the final drone model constructed in SIMULINK se figure 36 and figure 37 in appendix.

To test the simulation model of the quad-copter the we gave it 3 different inputs of ω First test with $\omega_1 = 0$, $\omega_2 = 0$, $\omega_3 = 0$, and $\omega_4 = 0$. The results can be seen in figure 1. In the first test the robots falls, indicated by the second graph. The drone does not move in the xy-plane, therefore the third graph is empty, indicating the the drone did not move. The first graph shows the angel of the robot. Here all angles are continuously 0, indicating that the drone does not change orientation during the test.

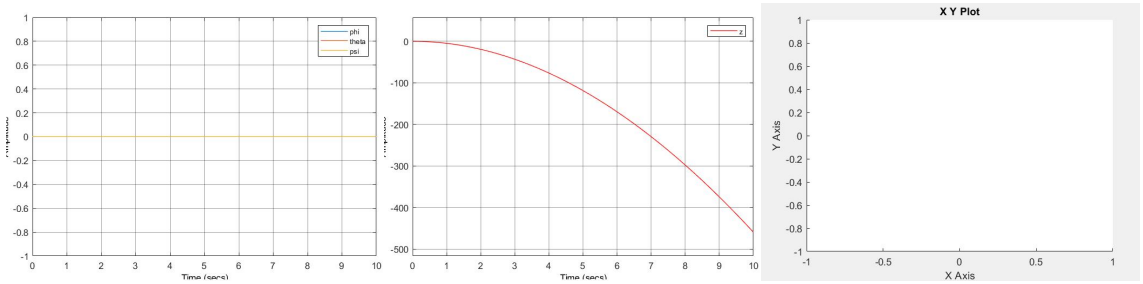


Figure 1: First graph shows the angels of orientation of the drone. The second graph shows the altitude of the drone. The third graph shows the movement of the drone in the xy-plane.

The second test with $\omega_1 = 10000$, $\omega_2 = 0$, $\omega_3 = 10000$, and $\omega_4 = 0$. The results can be seen in figure 2. In the second test the robots flies upward, while rotating around its own z-axis in a positive direction do to the momentum of the 2 rotors spinning in the same direction. This is indicated by the the first graph second graph. The drone does not move in the xy-plane, therefore the third graph is empty, indicating the the drone did not move in this plane.

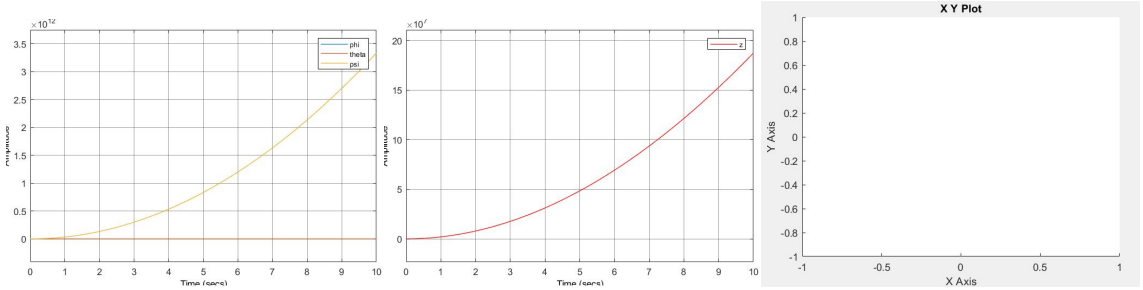


Figure 2: First graph shows the angels of orientation of the drone. The second graph shows the altitude of the drone. The third graph shows the movement of the drone in the xy-plane.

The second test with $\omega_1 = 0$, $\omega_2 = 10000$, $\omega_3 = 0$, and $\omega_4 = 10000$. The results can be seen in figure 2. In the second test the robots flies upward, while rotating around its own z-axis in a negative direction do to the momentum of the 2 rotors spinning in the same direction. The rotors are rotating opposite the rotors in test 2, therefore the drone is rotated in the opposite direction. This is indicated by the the first graph second graph. The drone does not move in the xy-plane, therefore the third graph is empty, indicating the the drone did not move in this plane.

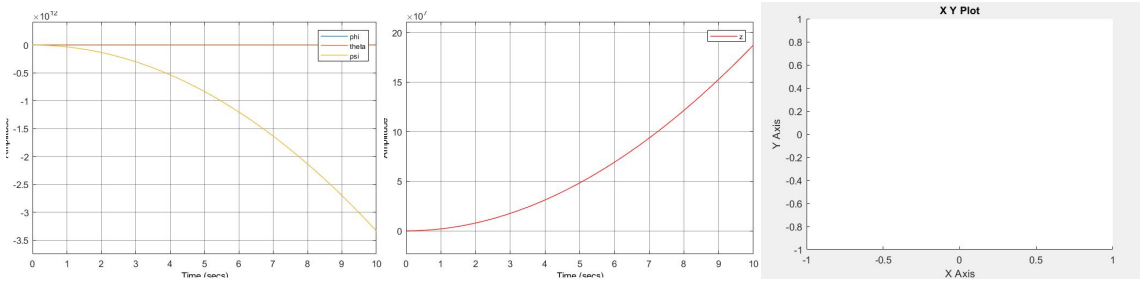


Figure 3: First graph shows the angels of orientation of the drone. The second graph shows the altitude of the drone. The third graph shows the movement of the drone in the xy-plane.

III Part 03

A. Exercise 3.1

The attitude controller adjusts the different angular velocities of the propellers and thereby changes the roll pitch and yaw of the quadrotor. The four velocities can be found by using the following equations:

$$\gamma 1 = \frac{mg}{k4cos(\theta)cos(\phi)} - \frac{2bu_\phi I_{xx} + u_\psi I_{zz} kL}{4bkL}$$

$$\gamma 2 = \frac{mg}{k4cos(\theta)cos(\phi)} - \frac{2bu_\theta I_{yy} - u_\psi I_{zz} kL}{4bkL}$$

$$\gamma 1 = \frac{mg}{k4cos(\theta)cos(\phi)} - \frac{-2bu_\phi I_{xx} + u_\psi I_{zz} kL}{4bkL}$$

$$\gamma 2 = \frac{mg}{k4cos(\theta)cos(\phi)} - \frac{-2bu_\theta I_{yy} - u_\psi I_{zz} kL}{4bkL}$$

Where u_ϕ, u_θ and u_ψ are:

$$\begin{bmatrix} u_\phi \\ u_\theta \\ u_\psi \end{bmatrix} = \begin{bmatrix} k_d e_\phi + k_p \int_0^T \dot{e_\phi} dt \\ k_d e_\theta + k_p \int_0^T \dot{e_\theta} dt \\ k_d e_\psi + k_p \int_0^T \dot{e_\psi} dt \end{bmatrix}$$

The signals e_ϕ, e_θ and e_ψ are the differences between the actual ϕ,θ and ψ values and their references. They are also known as error signals. The error signals are given by:

$$e_x = \int_0^T \dot{x} dt - x^*$$

where x^* is the reference.

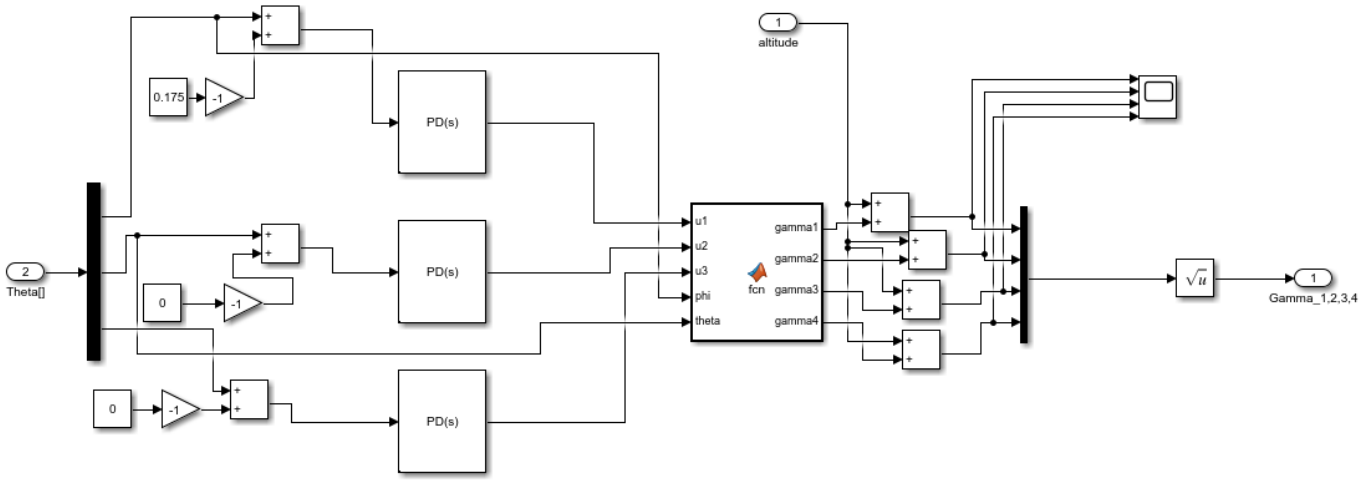


Figure 4: Simulink model of attitude controller. The ϕ reference is set to 10°(this is equal to 0.175 in radians), while θ and ψ references are set to 0

In order to control the altitude, an addition to the four gamma equations has to be made. By recalling the gamma equations, it can be observed that the following part affects the altitude:

$$\frac{mg}{k4cos(\theta)cos(\phi)}$$

By adding a PID control of z to the equation, the expression changes to:

$$\frac{mgf_{c,z}}{k4cos(\theta)cos(\phi)}$$

where

$$f_{c,z} = mPID(e_z)$$

After this implementation the model is now able to control both attitude and altitude.

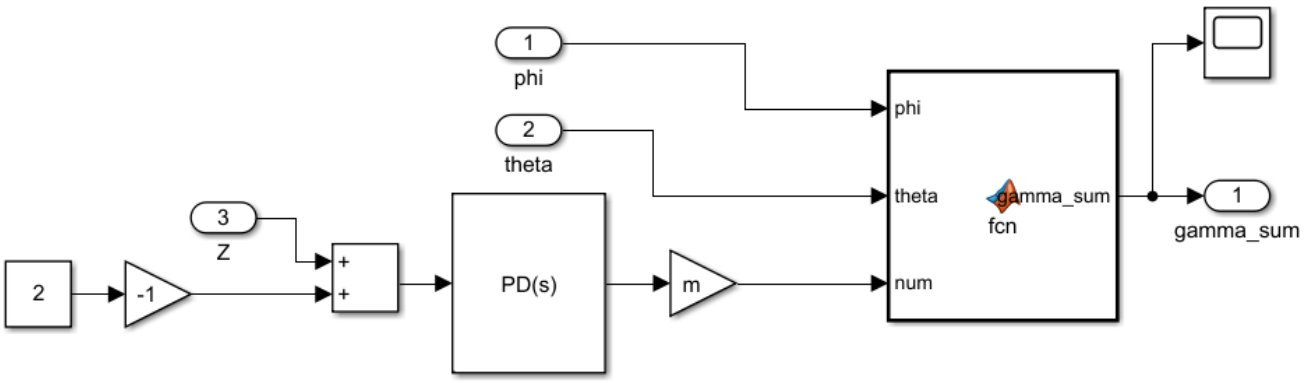


Figure 5: Simulink model of altitude controller where the reference is set to 2

Controllers were tuned one at the time using the Ziegler Nichols method. We used the approach of tuning a PID controller. But because the error we used was the error of the true angle subtracted from the reference angle and not the derivative of the angle subtracted from the derivative of the reference angle, we could just use a PD controller. However, in order to find kp and kd we could just set I to 0 in the PID controller and find the other values as a one would for a normal PID controller. Using this method for tuning we first needed to find the value of kp. Kp_max is the maximum value for kp where the step response oscillates around the final step value without being unstable, while the other constant in the PD controller are 0. We chose a final step value of 0.1 as the Ziegler Nicholos method states that the step used should be 10% of the max desired value the controller should be able to handle. As it makes very little sense to fly the drone at roll or pitch grater than 90 degrees a step of 0.1 was judged to appropriate. For the Phi and Theta controller kp_max turned out to be the same at around $9 * 10^{-5}$ giving a theoretical value of kp of $5.4 * 10^{-5}$ because the method states that the final kp value should be 60% of kp_max.

The kd value were calculated by the using the formula $kd = 0.125/f_0$ where f_0 is the oscillation frequency at kp_max. For phi and theta f_0 were between 14 and 15 hz giving the $kd = 0.0086$. Inserting the values into the gave a almost perfect step response for both the Phi angle and the Theta angle of the model. However adjusting kd slightly to 0.001 gave a slightly faster response to the final step value with very little extra overshoot and lowering kp to 0.00005 negated most of the overshoot. See figure 6 to see the step response of the phi and theta angles of the drone.

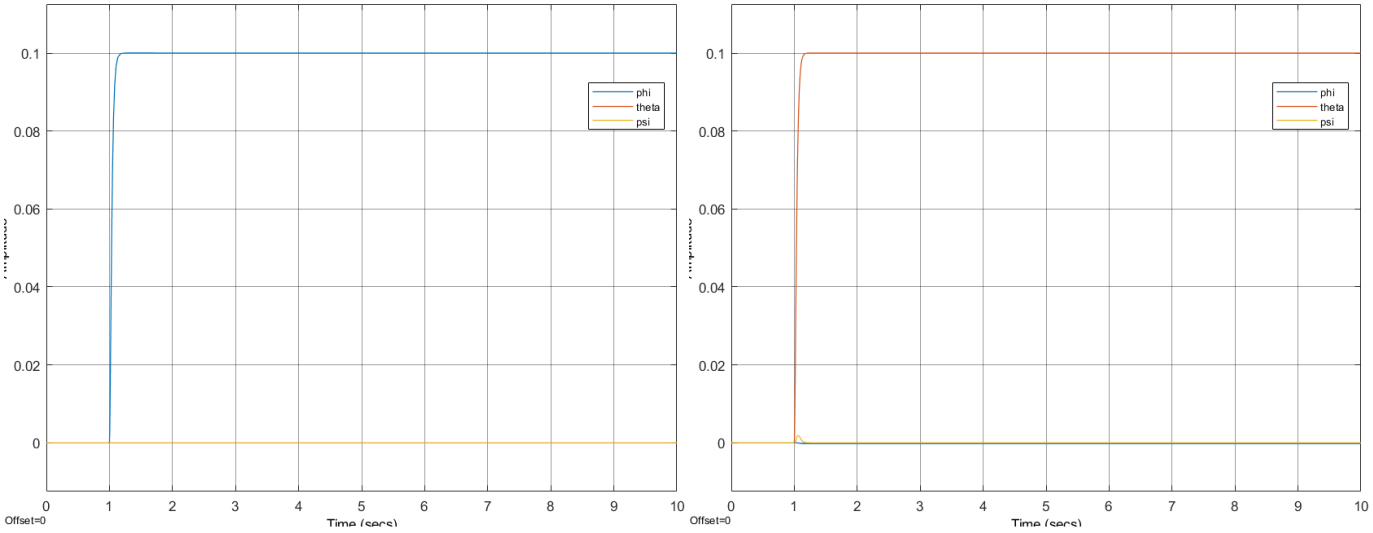


Figure 6: Step response of phi and theta of the drone. Kp = 0.00005 and kd = 0.001

To find the kp and kd values of the psi controller the same approach as for phi and theta were used. For psi a step of 0.1 were also used. Resulting in a kp_max of $1 * 10^{-5}$ again giving a kp of $6 * 10^{-5}$, and a theoretical kd of $kd = 0.125/0.12 = 1.04$. However, inserting these values into the PD controller resulted in a very unstable psi response, and kd were therefor hand tuned to 0.005. Resulting in a step with an ever so slight overshoot but otherwise a very good step response. Se figure 7.

In order to tune the altitude controller the Zeigler-Nicholos method were once again attempted, but did not give any useable values for kp and kd. Which let to this controller to be hand tuned to a kp of -0.135 and a kd of -0.045. We used a step with a final value of 1. Because we were not able to use the Zeigler-Nicholos method the resulting step response had a overshoot of 0.3, but we were not able to get this overshoot any lower. While having a stable controller and a fast enough response time. The best we could do with tuning by hand was a overshoot of 30 cm and a response time from step given to steady state final value in 3.2 seconds. This is far from a perfect response, but the drone would already by 1 meter + or - 5cm after 2.2 seconds. Which we judged to be an acceptable response time. See figure 8. We attempted to minimise the overshoot by lowering the kp value, but this only resulted in a very small change of the overshoot and a significantly longer response time.

1) The first test of the two controllers with the angles references set to $\Theta^* = \begin{bmatrix} 10 & 0 & 0 \end{bmatrix}^T$ gave the result seen on figure 9. As seen om the figure, the systems result in any overshoot and also rather rapidly goes into steady state at the correct value of 0.175 radians which corresponds to 10°.

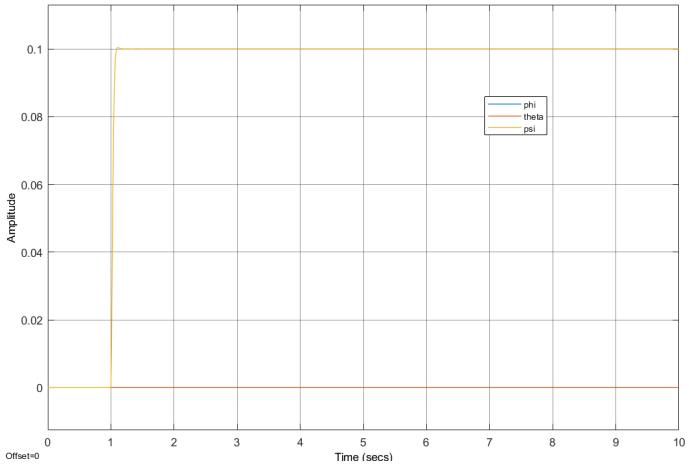


Figure 7: Step response of the psi angle of the drone. Kp = 0.00005 and kd = 0.005

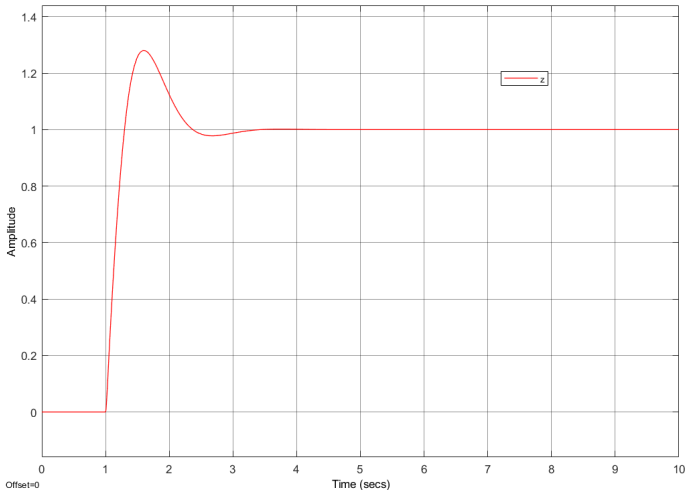


Figure 8: The altitude step response of the drone. Kp = -0.135 and kd = -0.045

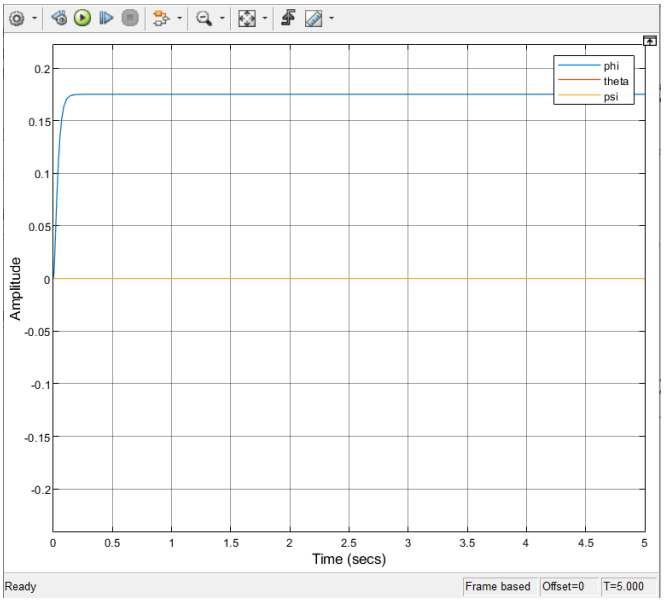


Figure 9: Plot of the angle values $\Theta^* = \begin{bmatrix} 10 & 0 & 0 \end{bmatrix}^T$

2) The second test of the two controllers with the angle references set to $\Theta^* = \begin{bmatrix} 0 & 10 & 0 \end{bmatrix}^T$ gave the result seen on figure 10. Again there is no overshoot and the signal gets to a steady state. The time is takes to get into a steady state is a bit longer than in figure 9.

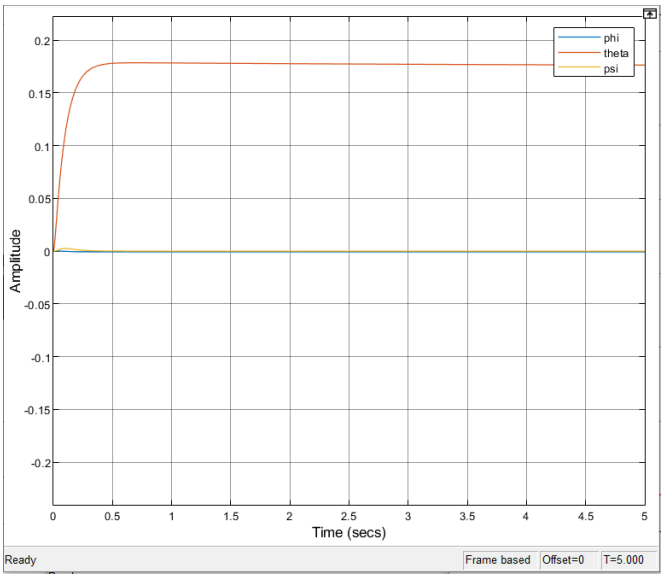


Figure 10: Plot of the angle values $\Theta^* = \begin{bmatrix} 0 & 10 & 0 \end{bmatrix}^T$

3) The third test of the two controllers with the angle references set to $\Theta^* = \begin{bmatrix} 0 & 0 & 10 \end{bmatrix}^T$ gave the result seen on figure 11. It can be seen that a slight overshoot occurs, but that the curve quickly falls into a steady state. The time it takes for signal to reach a steady state, is faster than seen in figure 9 and 10.

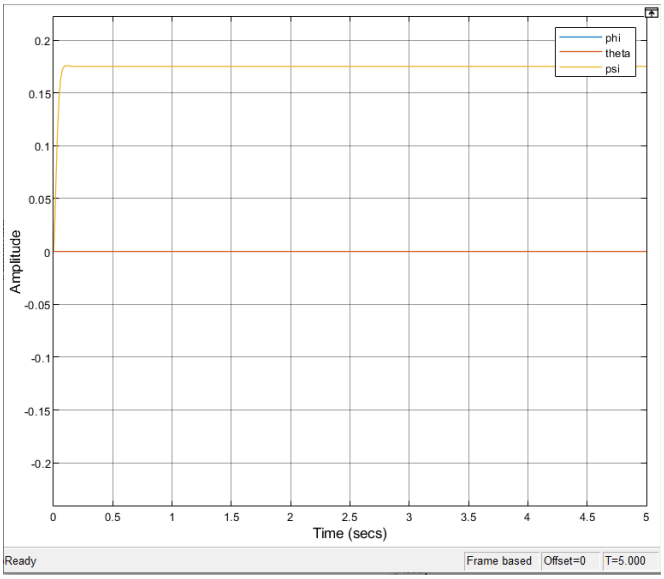


Figure 11: Plot of the angle values $\Theta^* = \begin{bmatrix} 0 & 0 & 10 \end{bmatrix}^T$

4) The fourth test of the two controllers with the angle references set to $\Theta^* = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ and the altitude reference set to $z^* = 1$ gave the result seen on figure 12. The control of the altitude results in a quite different signal than three signals regulated by the attitude controller. A noticeable overshoot can be seen, and it takes approximately 3 seconds to reach a steady state. The overshoot occurs because of the extreme angular velocity values that are seen at the beginning of the simulation. These values are

not realistic and therefore a lesser overshoot would most likely be seen if the controllers were applied on the quadrotor. This is the case because it is impossible for the propellers to spin at such a angular velocity in the physical world.

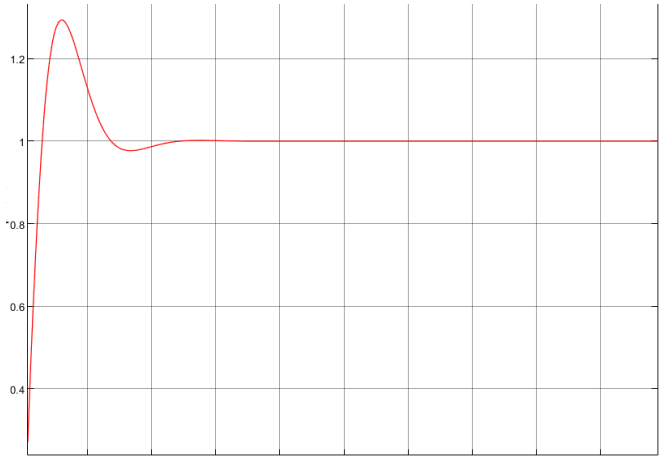


Figure 12: Altitude when reference is set to $z^* = 1$ seen over a time span of 10 seconds

IV Part 04

A. *Exercise 4.3*

The solution to the Dijkstra algorithm can be seen on 13. ”Parent” indicates the node that is observed, while ”node name” are the neighbour nodes to the observed node. The distance of the parent to the neighbour nodes are the ”node dist” values. If the parent node is not the start note, an arrow is located next to the parent node, which is used to show the path taken to the parent node. For example s11 ← s7 means that the shortest route to node s11 is via node s7.

If a distance to a node is longer than one already found earlier during the steps, the distance is marked with red and a ”X”. The goal is marked with a green colour. By looking at the goal and tracing back via the parent arrows we get the following route: s23 → s19 → s15 → s14→ s11 → s7 → s3→ s0 The final route can be seen in appendix figure 38

node name	node dist	parent		node name	node dist	parent		node name	node dist	parent
s1	4,5	s0		s7	2,2+2,2=4,4	s3 ← s0		s6	4,4+1,4=5,8	s7←s3
s2	5,4	s0	next step→	s8	2,2+3,2=5,4	s3 ← s0	next step →	s8	4,4+2,2=6,6 X	s7←s3
s3	2,2	s0						s11	4,4+2,2=6,6	s7←s3

node name	node dist	parent		node name	node dist	parent		node name	node dist	parent
s2	4,5+4,2=8,7 X	s1←s0		s5	5,4+3,6=9	s2←s0		s7	5,4+2,2=7,6 X	s8←s3
s4	4,5+2,2=6,7	s1←s0	next step →	s6	5,4+3,2=8,6 X	s2←s0	next step →	s11	5,4+3,2=8,6 X	s8←s3

node name	node dist	parent		node name	node dist	parent		node name	node dist	parent
s2	5,8+3,2=9 X	s6←s7		s14	6,6+4,1=10,7	s11←s7		s5	6,7+1,4=8,1	s4←s1
s10	5,8+2=7,8	s6←s7	next step →	s15	6,6+6=12,6	s11←s7	next step →			

node name	node dist	parent		node name	node dist	parent		node name	node dist	parent
s13	7,8+4=11,8	s10←s6		s2	8,1+3,6=11.7 X	s5←s4		s12	10,3+6=16,3	s9←s5
			next step →	s9	8,1+2,2=10,3	s5←s4	next step →	s13	10,3+3=13,3 X	s9←s5

node name	node dist	parent		node name	node dist	parent		node name	node dist	parent
s15	10,7+1,4=12,1	s14←s11		s9	11,8+3=14,8 X	s13←s10		s19	12,1+3,2=15.3	s15←s14
s18	10,7+2=12,7	s14←s11	next step →	s18	11,8+3,2=15 X	s13←s10	next step →	s20	12,1+2,2=14,3	s15←s14

node name	node dist	parent		node name	node dist	parent		node name	node dist	parent
s13	12,7+3,2=15,9 X	s18←s14		s19	14,3+4,5=18.8 X	s20←s15		s17	15,3+4,1=19,4 X	s19←s15
s17	12,7+3,6=16,3	s18←s14	next step →	s22	14,3+3,6=17,9	s20←s15	next step →	s20	15,3+4,5=19,8 X	s19←s15
								s23	15,3+2,8=18,1	s19←s15

Figure 13: All steps of the Dijkstra algorithm when starting in s0 and having the goal at s23

B. Exercise 4.4

The following will showcase how the Greedy best first search algorithm, on a step by step basis. No change in expanding s8 since all connecting points have alternatives routes which are faster.

Node Name	g(n)	total cost	Parent	Expansions	New parent	Inspected nodes
s11	6,6	6,6	s7	7	None	s0 s8
s4	6,7	6,7	s1		None	s3 s6
s5	9	9	s2		None	s7
s10	7,8	7,8	s6		None	s1
						s2

Figure 14: GBF searching s6's neighbours and ranking them in the priority queue

Node Name	g(n)	total cost	Parent	Expansions	New parent	Inspected nodes
s4	6,7	6,7	s1	8	None	s0 s8
s10	7,8	7,8	s6		None	s3 s6
s5	9	9	s2		None	s7 s11
s14	10,7	10,7	s11		None	s1
s15	12,6	12,6	s11			s2

Figure 15: GBF searching s11's neighbours and ranking them in the priority queue

Node Name	g(n)	total cost	Parent	Expansions	New parent	Inspected nodes
s10	7,8	7,8	s6	9	None	s0 s8
s5	8,1	8,1	s2		s4	s3 s6
s14	10,7	10,7	s11		None	s7 s11
s15	12,6	12,6	s11			s1 s4
						s2

Figure 16: GBF searching s4's neighbours and ranking them in the priority queue

Notice: Node s5 changes parent to s4

Node Name	g(n)	total cost	Parent	Expansions	New parent	Inspected nodes
s5	8,1	8,1	s4	10	None	s0 s8
s14	10,7	10,7	s11		None	s3 s6
s13	11,8	11,8	s10		None	s7 s11
s15	12,6	12,6	s11			s1 s4
		0				s2 s10

Figure 17: GBF searching s10's neighbours and ranking them in the priority queue

C. *Exercise 4.5*

The A* algorithm is closely related to the Dijkstra's algorithm but with the distinct feature that adds an extra cost via a heuristic term h(n). A good candidate for the this value, in this case, is the euclidean distance to the goal point.

The total cost of a node is calculated as:

$$f(n) = g(n) + h(n)$$

Each node in figure 39 is identified by a unique name spanning from s1-s23. Furthermore the distance from one note to its neighbouring notes is specified (in black), as well as the heuristic term (in red), in this example being the euclidean distance from each node to the goal.

As in Dijkstra's algorithm 2 queues are kept track of. Though the priority in the queue is calculated as the distance to Start node + heuristic cost.

Also we will be keeping track of the parent of each node as we wont care how many "jumps" it takes to reach each node, but rather the distance.

We will be Starting at s0, there is no distance to itself and the distance to the goal is 17. This is the first step of the algorithm can be seen in figure 18 and consists of noting the node in the priority queue and then inspecting its neighbours. Each figure describes a "step" of the algorithm, where an expansion has taken place and & or a node changes parent. The priority queue is sorted accordingly to the total cost, and the top element in a figure will represent the element being inspected in the next figure.

Node Name	Distance to Start	Distance to Goal	Total cost	Current parrent	total expansions	New Parent	Inspected Nodes
S	0	17	17	None	0	None	

Figure 18: Priority & Inspected queue, starting the algorithm

Node Name	Distance to Start	Distance to Goal	Total cost	Current parrent	total expansions	New Parent	Inspected Nodes
s2	5,4	11	16,4	S	1	None	s0
s3	2,2	15,1	17,3	S	1	None	
s1	4,5	15,5	20	S	1	None	

Figure 19: Priority & Inspected queue, inspecting node s0's neighbours

Node Name	Distance to Start	Distance to Goal	Total cost	Current parrent	total expansions	New Parent	Inspected Nodes
s3	2,2	15,1	17,3	s0	2	None	s0
s5	8,6	10,2	18,8	s2	2	None	S2
s6	9	10,2	19,2	s2			
s1	4,5	15,5	20	s0			

Figure 20: Priority & Inspected queue, inspecting node s2's neighbours

In figure 22 it is important to notice that s6 have now changed parent node to s7 and moved up so much in the queue that it will be the node inspected next. The reason for this is since s7 is also a neighbour node to s6, together with s2 there will be multiple ways of reaching s6 it just so happened that when reaching it through s2 it was further away. However reaching it through s7 made it a potential candidate for the optimal path.

Node Name	Distance to Start	Distance to Goal	Total cost	Current parrent	total expansions	New Parent	Inspected Nodes
s7	4,4	12,4	16,8	s3	3	None	s0
s5	8,6	10,2	18,8	s2	3	None	S2
s6	9	10,2	19,2	s2	3	None	S3
s1	4,5	15,5	20	s0	3	None	
s8	5,4	15,2	20,6	s3	3	None	

Figure 21: Priority & Inspected queue, inspecting node s3's neighbours

Node Name	Distance to Start	Distance to Goal	Total cost	Current parrent	total expansions	New Parent	Inspected Nodes
s6	5,8	10,2	16	s2	4	s7	s0
s11	6,6	11,7	18,3	s7	4	None	s2
s5	8,6	10,2	18,8	s2	4	None	s3
s1	4,5	15,5	20	s0	4	None	s7
s8	5,4	15,2	20,6	s3	4	None	

Figure 22: Priority & Inspected queue, inspecting node s7's neighbours

Node Name	Distance to Start	Distance to Goal	Total cost	Current parrent	total expansions	New Parent	Inspected Nodes
s10	7,8	6,7	14,5	s6	5	None	s0
s11	6,6	11,7	18,3	s7	5	None	s2
s5	8,6	10,2	18,8	s2	5	None	s3
s1	4,5	15,5	20	s0	5	None	s7
s8	5,4	15,2	20,6	s3	5	None	s6

Figure 23: Priority & Inspected queue, inspecting node s6's neighbours

Node Name	Distance to Start	Distance to Goal	Total cost	Current parrent	total expansions	New Parent	Inspected Nodes
s11	6,6	11,7	18,3	s7	6	None	s0 s10
s5	8,6	10,2	18,8	s2	6	None	s2
s13	11,8	7,3	19,1	s10	6	None	s3
s1	4,5	15,5	20	s0	6	None	s7
s8	5,4	15,2	20,6	s3	6	None	s6

Figure 24: Priority & Inspected queue, inspecting node s10's neighbours

Node Name	Distance to Start	Distance to Goal	Total cost	Current parrent	total expansions	New Parent	Inspected Nodes
s14	10,7	6,7	17,4	s11	7	None	s0 s10
s5	8,6	10,2	18,8	s2	7	None	s2 s11
s13	11,8	7,3	19,1	s10	7	None	s3
s15	12,6	6,7	19,3	s11	7	None	s7
s1	4,5	15,5	20	s0	7	None	s6
s8	5,4	15,2	20,6	s3	7	None	

Figure 25: Priority & Inspected queue, inspecting node s11's neighbours

Node Name	Distance to Start	Distance to Goal	Total cost	Current parrent	total expansions	New Parent	Inspected Nodes
s18	10	5,1	15,1	s14	8	None	s0 s10
s15	12,1	6,7	18,8	s11	8	s14	s2 s11
s5	8,6	10,2	18,8	s2	8	None	s3 s14
s13	11,8	7,3	19,1	s10	8	None	s7
s1	4,5	15,5	20	s0	8	None	s6
s8	5,4	15,2	20,6	s3	8	None	

Figure 26: Priority & Inspected queue, inspecting node s14's neighbours

Node Name	Distance to Start	Distance to Goal	Total cost	Current parrent	total expansions	New Parent	Inspected Nodes
s15	12,1	6,7	18,8	s14	9	None	s0 s10
s5	8,6	10,2	18,8	s2	9	None	s2 s11
s13	11,8	7,3	19,1	s10	9	None	s3 s14
s17	16,3	3,2	19,5	s18	9	None	s7 s18
s1	4,5	15,5	20	s0	9	None	s6
s8	5,4	15,2	20,6	s3	9	None	

Figure 27: Priority & Inspected queue, inspecting node s18's neighbours

Node Name	Distance to Start	Distance to Goal	Total cost	Current parrent	total expansions	New Parent	Inspected Nodes	
s19	15,3	2,8	18,1	s15	10	None	s0	s10
s5	8,6	10,2	18,8	s2	10	None	s2	s11
s13	11,8	7,3	19,1	s10	10	None	s3	s14
s17	16,3	3,2	19,5	s18	10	None	s7	s18
s1	4,5	15,5	20	s0	10	None	s6	s15
s8	5,4	15,2	20,6	s3	10	None		
s20	14,3	7,1	21,4	s15	10	None		

Figure 28: Priority & Inspected queue, inspecting node s15's neighbours

Node Name	Distance to Start	Distance to Goal	Total cost	Current parrent	total expansions	New Parent	Inspected Nodes	
s23 (Goal)	18,1	0	18,1	s19	11	None	s0	s10
s5	8,6	10,2	18,8	s2	11	None	s2	s11
s13	11,8	7,3	19,1	s10	11	None	s3	s14
s17	16,3	3,2	19,5	s18	11	None	s7	s18
s1	4,5	15,5	20	s0	11	None	s6	s15
s8	5,4	15,2	20,6	s3	11	None	s19	
s20	14,3	7,1	21,4	s15	11	None		
s17	19,4	3,2	22,6	s19	11	None		

Figure 29: Priority & Inspected queue, inspecting node s19's neighbours

Figure 29 describes the last step in the algorithm, which ends once the goal node is on top of the priority queue. Totalling 11 expansion, 2 parent switching and a optimal path found with distance of 18.1 from the start to the goal.
In figure 39 the optimal path found by the algorithm is drawn. Implementing the algorithm it would make sense keeping track of each nodes parent such that back-tracing from the goal would simply be back-tracing the parent nodes from the goal until it reaches the start node.

D. *Exercise 4.6*

Greedy first-seach

- Pro's
  - Memory efficient
  - Can switch between BFS and DFS. Getting pro's and cons from both.
- Con's
  - Neither complete, nor optimal.
  - Chance of getting stuck in a loop

A*

- Pro's
  - Complete and guarantees optimal path.
  - Various heuristics costs can be integrated to the algorithm without much change in the original code.
- Con's
  - Less memory efficient compared to less advanced algorithms.
  - Can be unpractical on large-scale problems.

No one algorithms is ever the "best" for all scenarios of UAV's, every algorithm has pro's and con's connected to them and the optimal algorithm must be chosen for what the purpose, environment and complexity of the UAV is going to be.

E. *Exercise 4.7*

The figure 30 showcases the provided obstacle map with end and start points at.

$$start = [2, 1, 2], end = [1, 5, 1];$$

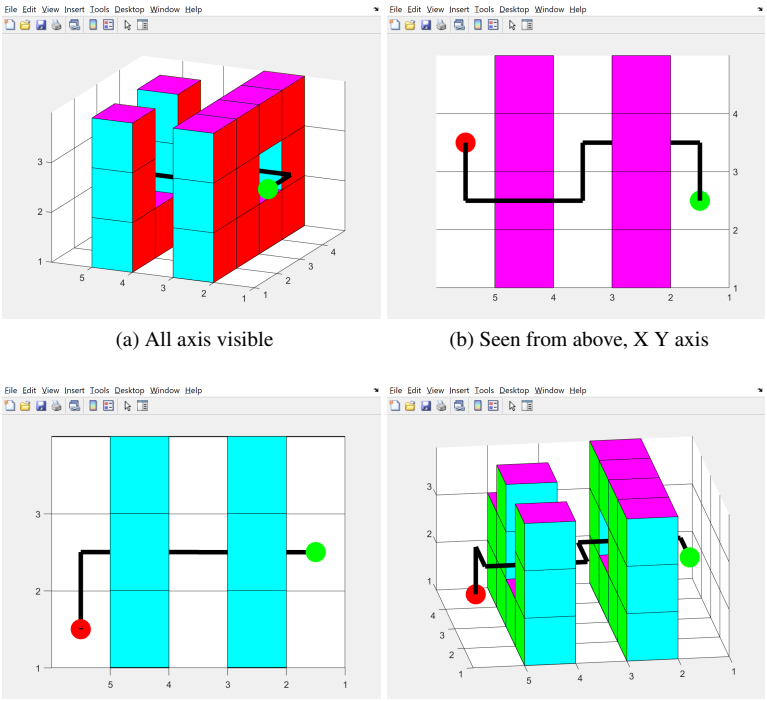
F. *Exercise 4.8*

In the greedy_3d.m a term describing distance back to the start point has been added.

```
% Calculate the distance from the node
% to the end point
temp_node.h = temp_node.calc_dist_3d(end_);
temp_node.g = temp_node.calc_dist_tostart_3d(start)
% Calculate the total cost of the node
temp_node.f = temp_node.h+temp_node.g;
```

Furthermore in the node.m file describing the node class a new functions has been made, calculating the distance to the starting point.

```
function g = calc_dist_tostart_3d(obj, start)
    g = [obj.position];
    while  ~(obj.position(1) == start(1) && ...
             obj.position(2) == start(2) && ...
             obj.position(3) == start(3))
       % Update the route by going backwards through the parents
             obj = obj.parent;
             g = cat(1,g,obj.position);
    end
    g=length(g);
end
```



(a) All axis visible

(b) Seen from above, X Y axis

(c) Seen from side, Y, Z axis

(d) All axis different perspective

Figure 30: Greedy best-first 3D path planning route

The implemented algorithm is now A* instead of Greedy best-first algorithm, the trace found by the algorithm is identical but that is to be expected for such a simple map.

If the implemented algorithm was of the form below, It would be the Dijkstra's Algorithm.

$$f(n) = g(n)$$

V Part 05

A. *Exercise 5.1*

We are given the equation

$$x(t) = a_0 + a_1 * t + a_2 * t^2 + a_3 * t^3 + a_4 * t^4 + a_5 * t^5$$

With the constraints

$$x(0) = 0, \dot{x}(0) = 0, \ddot{x}(0) = 0, x(1) = 1, \dot{x}(1) = 0, \ddot{x}(1) = 0$$

By solving the 6 equations, with the 6 unknowns a_i for $i = 0...5$, the following constants for the equations were found.

$$a_0 = 0 a_1 = 0 a_2 = 0 a_3 = 10 a_4 = -15 a_5 = 6$$

B. *Exercise 5.3*

Since the upward direction of the drone z_B, the yaw angle ψ and the Euler angle convention ZYX are known, the vector y_C can be found by:

$$y_C = \begin{bmatrix} cos\phi \\ sin\phi \\ 0 \end{bmatrix}$$

Now by having both z_B and y_C, x_B can be found:

$$x_B = \frac{z_B \times y_C}{||z_B \times y_C||}$$

Finally y_B is found:

$$y_B = x_B \times z_B$$

We now write R_B as:

$$R_B = \begin{bmatrix} x_B & y_B & z_B \end{bmatrix}$$

VI Part 06

A. *Exercise 6.1*

Instead of downscaling our A* algorithm to work in 2D we made the given map of the 3D maze into a 3D map. From here our A* 3D algorithm was used to find the optimal path.

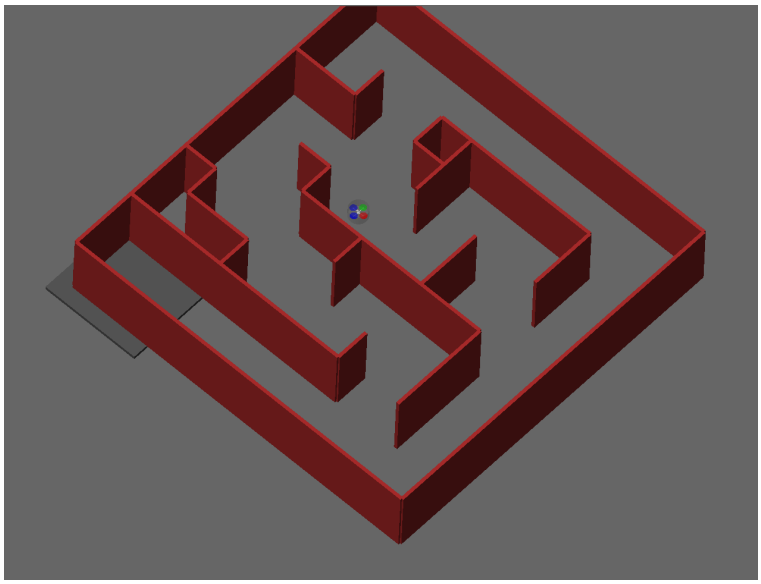


Figure 31: A star algorithm used for finding optimal path to point in maze

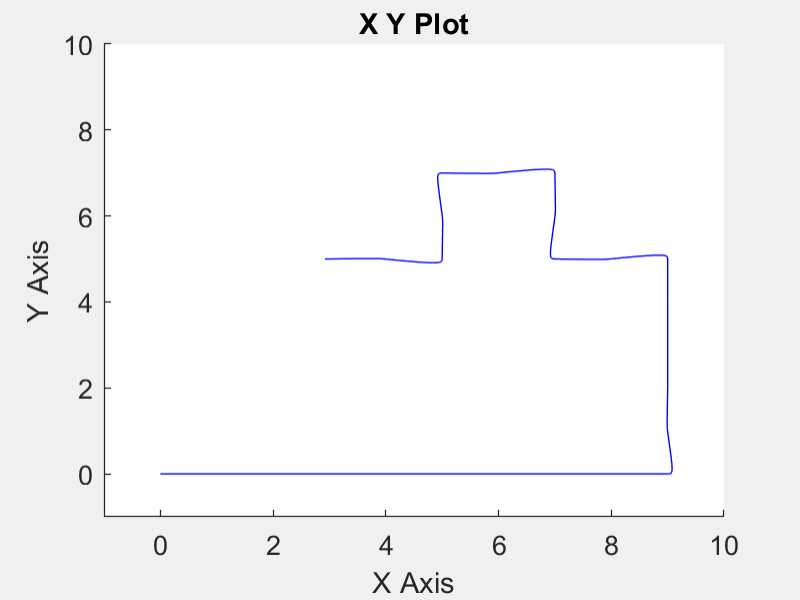


Figure 32: X-Y graph of the route taken by the robot

B. Exercise 6.4

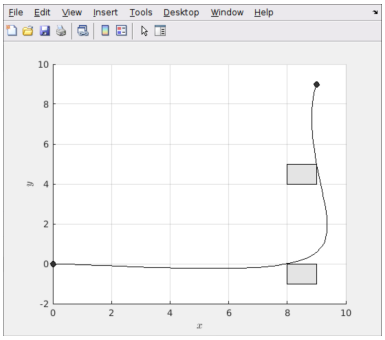
In the file uas_tracetory.m the following changes was made to the corridors as well as to the corridor times.

```
corridors.times = [2 3.5];
corridors.x_lower = [8 8];
corridors.x_upper = [9 9];
corridors.y_lower = [-1 4];
corridors.y_upper = [0 5];
corridors.z_lower = [0 0];
corridors.z_upper = [2 2];
```

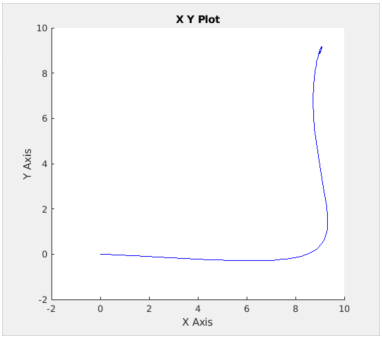
The corridors and waypoints are 2 ways for the trajectory planner to shape its route, corridors provide a ”soft” waypoint, meaning the path can pass through any part of the corridor to fulfil the requirements and waypoints are points that the path must pass through to fulfil the requirements.

Corridor times are the time at which the corridor requirements must be fulfilled on the path, meaning a higher time for the first corridor will give the UAV more time to get to the corridor. Changing corridor times has a large impact on the path.

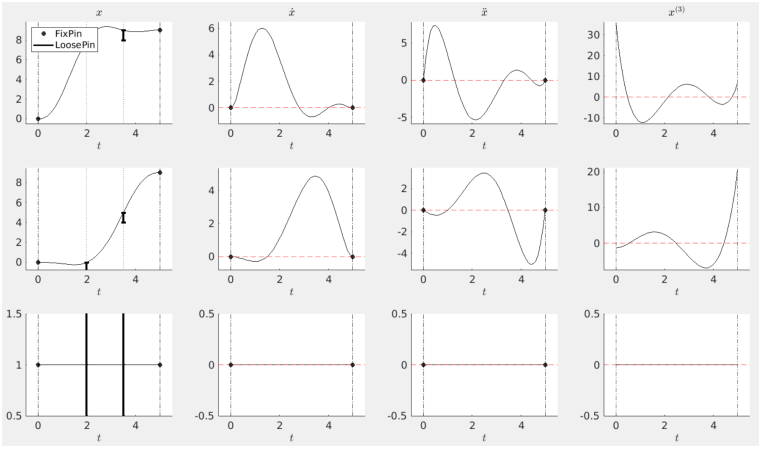
The placement of the corridors was chosen to improve the turn in the lower right corner, without this corridor the trajectory planner will make the path more circular and thus hit the walls when flying.



(a) Theoretical path.



(b) Actual path



(c) Force calculations from theoretical path

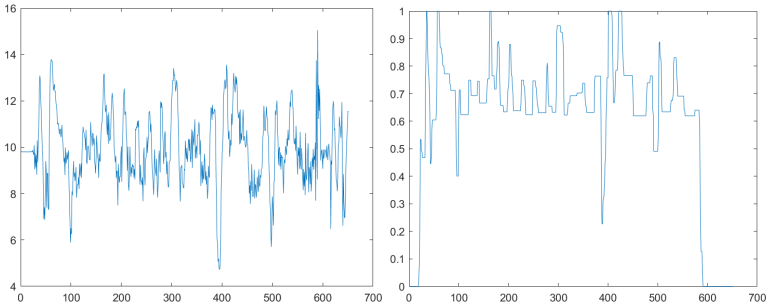
VII Part 07

A. *Exercise 7.1*

We were able to use our 3d maze algorithm from exercise 6.1 to find the correct path for the drone. Thereby we could use the algorithm to navigate the 3d maze without touching the walls or the obstacles. This was demonstrated on the demonstration day on the 25th of June. The drone can also be seen navigating the maze in our demonstration video also.

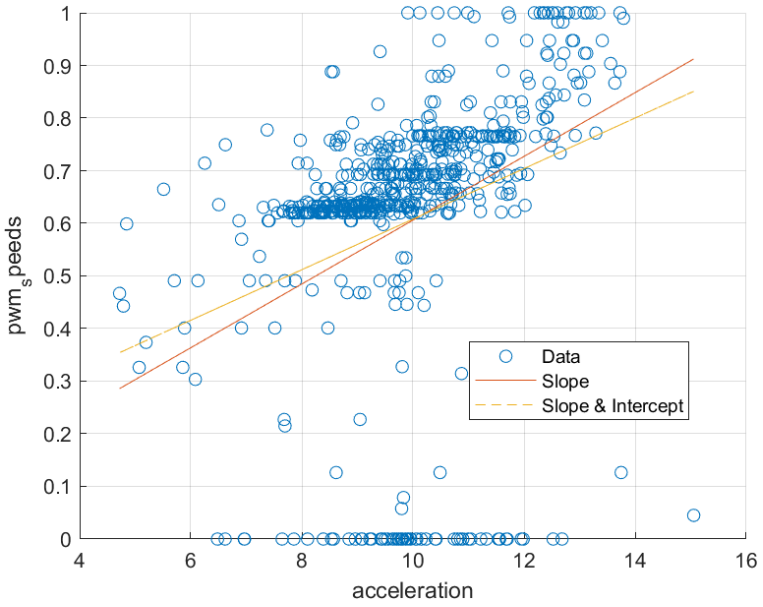
B. *Exercise 7.2*

To find the relationship between the thrust commands and the PWM commands we decided to gather data from the drone whilst trying to hover and then using PWM commands vs vertical accelerations to create a linear regression of the data in order for us to obtain the first order polynomial we would then use in the controller. The following graphs show the 2 data sets (Acceleration and PWM) and the correlation between the 2 data sets with the estimated first order polynomial.



(a) Acceleration graph for the drone in flight.

(b) PWM graph for the drone in flight.



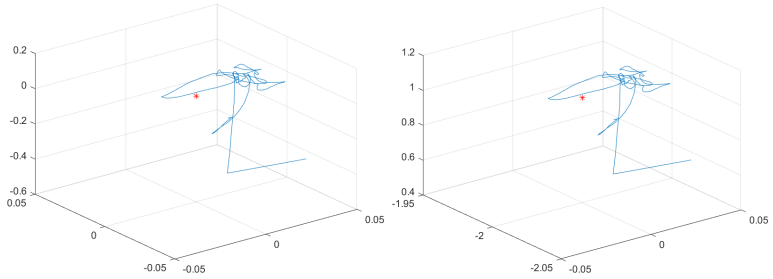
(c) Correlation between the 2 data sets (Linear regression).

Figure 33: Plots from data gathering and interpreting from exercise 7.2

This approach didn't get us the desired result but it was a starting point, we ended up "hand tuning" the values until the drone could hover on its own with the controller.

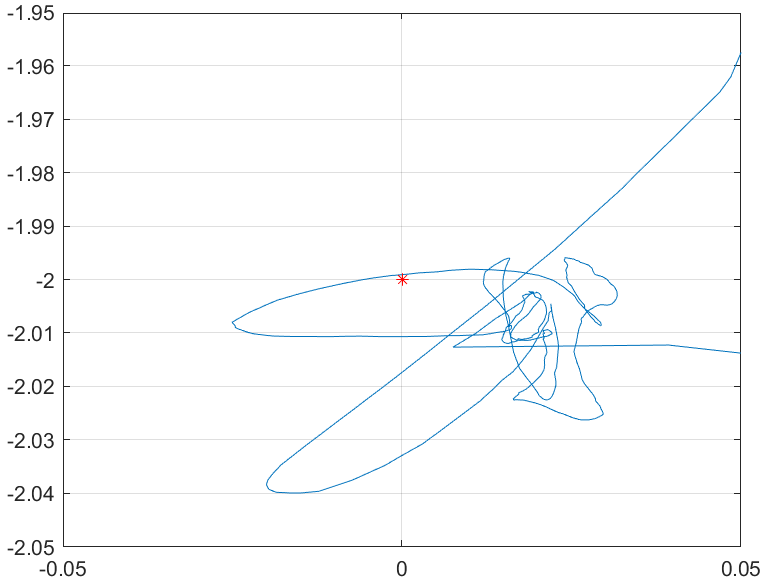
C. Exercise 7.2.1

In part 1 of this exercise the drone had to hover around the point [1, 1, 1] for 10 seconds with a maximum error of 10 cm.



(a) X-Y-Z error path w. reference point.

(b) X-Y-Z path with reference point.

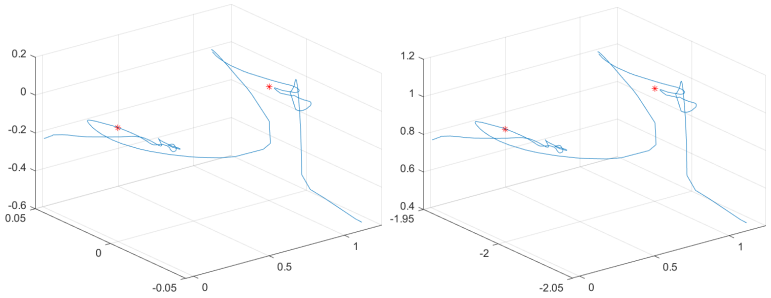


(c) X-Y path with reference point.

Figure 34: Plots of ex 7.2 part 1. Showing the trajectory of the drone hovering around the point [1, 1, 1]

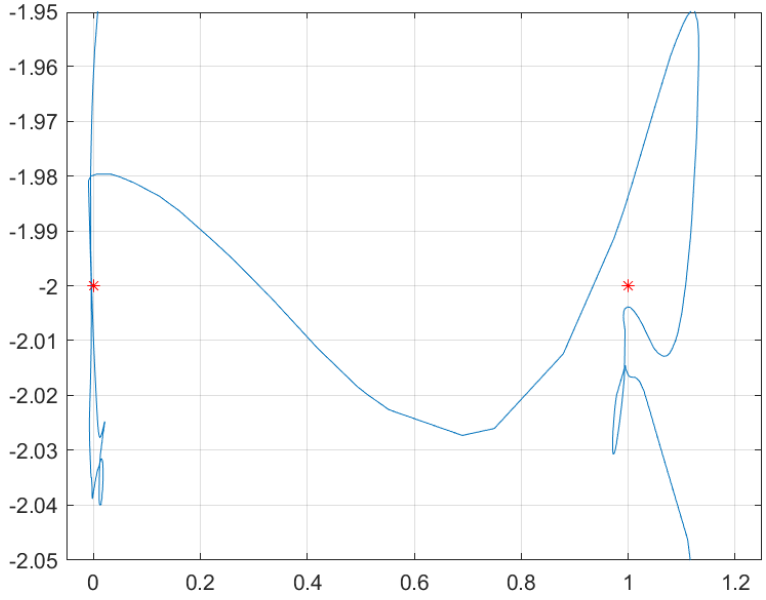
Figure 34 shows the actual trajectory of the drone during the entire flight, in this figure it can be seen that the error when hovering is less than the requirement of a maximum error of 10 cm.

in part 2 of this exercise the drone had to hover around the point [1, 1, 1] for 10 seconds and then move to [2, 1, 1] and again hover for 10 seconds, ensuring that the drone did not overshoot the second point by more than 30cm and that the robot had a maximum error of 10 cm when hovering.



(a) X-Y-Z error path w. reference point.

(b) X-Y-Z path with reference point.



(c) X-Y path with reference point.

Figure 35: Plots of ex 7.2 part 2. Showing the trajectory of the drone hovering around a point and then afterwards moving 1 meter.

Figure 35 shows the trajectory of the entire flight with markers showing the points at which the drone is supposed to hover around, it can be seen that the over shoot is less than 30 cm and the error is less than 10 cm.
Signed:

S173935

S174831 Mikkel Pedersen

S174849 Sebastian Kusio

S174822 Victor G. Flindt

VIII Appendix

Code written for the course can be found on our github here: https://github.com/S174831/uas-31390

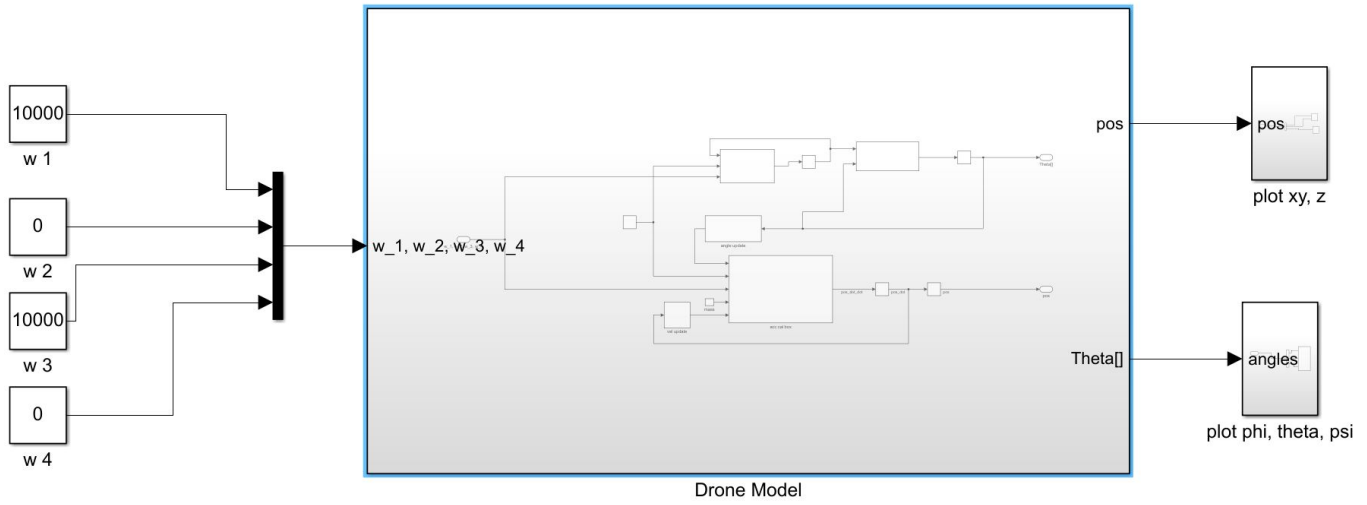


Figure 36: The model of the drone. With ω (rotational speed of the blades) as inputs position and angles of the drone as outputs.

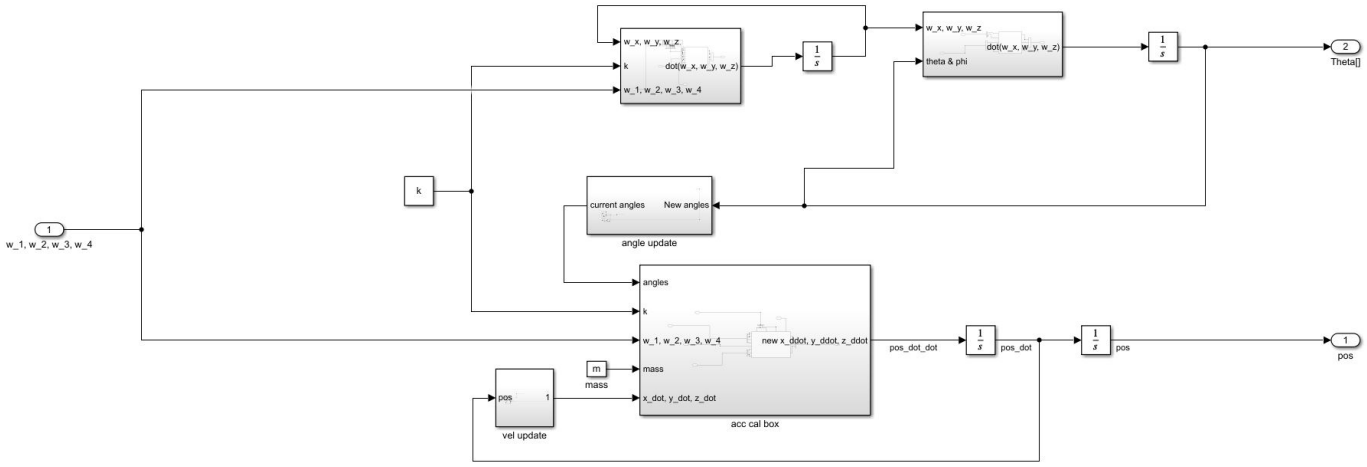


Figure 37: The internals of the model of the drone. With ω (rotational speed of the blades) as inputs position and angles of the drone as outputs. Here the structure of the drone model can be seen.

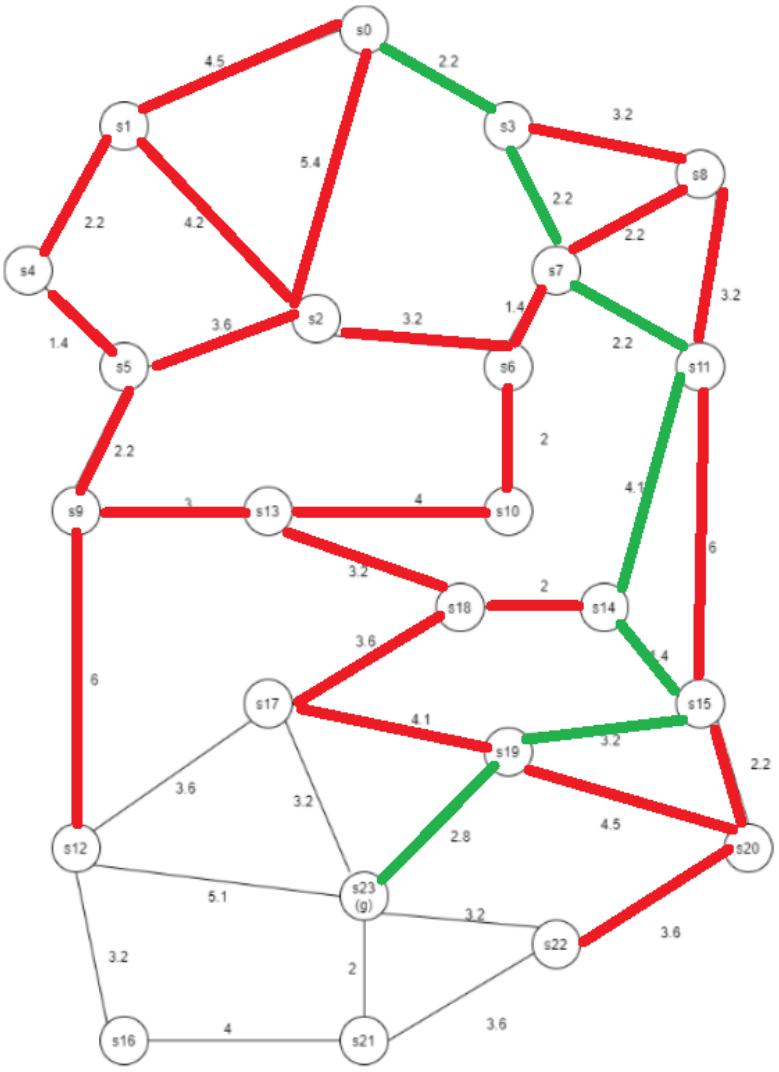


Figure 38: Map showing the fastest route to s23 from s0 when using the Dijkstra algorithm

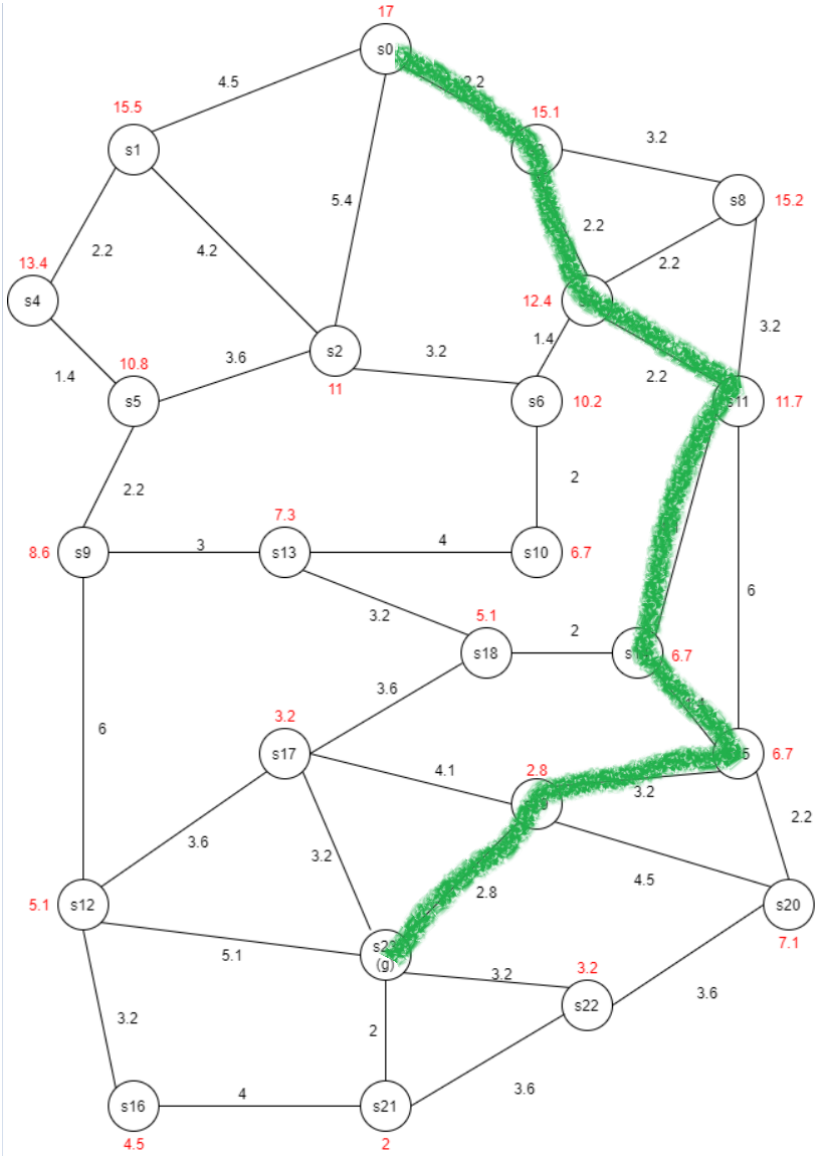


Figure 39: Optimal path found by A* algorithm