DTU Electrical Engineering

31390 - Unmanned Autonomous Systems

Final Project

Author:
Andreas Tuxen (s153408)
Sandra Karlsdóttir Andreassen (s202088)
Victor Azpeitia Haavik (s163865)

Date: 06/24-2021

Contents

List of Figures

1 Part 1: Rotations

1.1 Exercise

The rotation matrix corresponding to the set of Euler angles ZXZ, is found using the individual rotation matrices corresponding to the Z-rotation and the X-rotation. The matrix product of these matrices ZXZ, produces the compound ZXZ rotation. See equation 1.

$$ZXZ = \begin{bmatrix} cos(\psi_1)cos(\psi_2) - sin(\psi_1)cos(\phi)sin(\psi_2) & cos(\psi_1)sin(\psi_2) + sin(\psi_1)cos(\phi)cos(\psi_2) & sin(\psi_1)sin(\phi) \\ -sin(\psi_1)cos(\psi_2) - cos(\psi_1)cos(\phi)sin(\psi_2) & -sin(\psi_1)sin(\psi_2) + cos(\psi_1)cos(\phi)cos(\psi_2) & cos(\psi_1)sin(\phi) \\ sin(\phi)sin(\psi_2) & -sin(\phi)cos(\psi_2) & cos(\phi) \end{bmatrix} \tag{1}$$

.

1.2 Exercise

In the case when sin(theta) = 0, the body is restricted to an orientation of either 0 or Pi radiant around the y-axis. Thus a ZYZ rotation will only affect the body as an ZZ rotational operation. When calculating the rotation matrix corresponding to the set of Euler angles ZYZ, restricting sin(theta) = 0, it is verified that the matrix corresponds to two Z rotations. See equation 2.

$$ZYZ = \begin{bmatrix} cos(\psi_1)cos(\psi_2) - sin(\psi_1)sin(\psi_2) & cos(\psi_1)sin(\psi_2) + sin(\psi_1)cos(\psi_2) & 0 \\ -sin(\psi_1)cos(\psi_2) - cos(\psi_1)sin(\psi_2) & cos(\psi_1)cos(\psi_2) - sin(\psi_1)sin(\psi_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

.

1.3 Exercise

In the case when cos(theta) = 0, the body is restricted to an orientation of either Pi/2 or 3*Pi/2 radiant around the y-axis. Thus a ZYX rotation will only affect the body as an XZ rotational operation. When calculating the rotation matrix corresponding to the Roll-Pitch-Yaw angles, restricting cos(theta) = 0, it is verified that the matrix corresponds to a Z and an X rotation. This will result in a *gimbal lock*, where we loose a degree of freedom. See equation 1.

$$ZYX = \begin{bmatrix} 0 & cos(\phi)sin(\psi) + sin(\phi)cos(\psi) & sin(\phi)sin(\psi) - cos(\phi)cos(\psi) \\ 0 & cos(\phi)cos(\psi) - sin(\phi)sin(\psi) & cos(\phi)sin(\psi) + sin(\phi)cos(\psi) \\ 1 & 0 & 0 \end{bmatrix} \tag{3}$$

1.4 Exercise

Using the axis-angle representation, we can describe the rotation from v to w, using a single operation instead of a combination of matrix operations. To find the axis of rotation that "turns" v into w, the cross product is helpful. The cross product allow to calculate the orthogonal vector from the plane spanned by v and w, this will be the axis on which we can turn v into w.

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = a \times b \tag{4}$$

There will exist two directions to rotate v into w. To find the shortest angle, the following equation is used. The length of the two vectors do not need to be considered since they are unit vectors.

$$\theta = arccos(a \cdot b)$$

The quaternion corresponding to the minimal rotation that brings v into w, is thus:

$$\begin{bmatrix} v_x \\ v_y \\ v_z \\ \theta \end{bmatrix} \tag{5}$$

1.5 Exercise

1.5.1

The quaternion q1 that represents the rotation of 180 degree about the x-axis, is found simply by inserting the angle of rotation (180), and defining the vector to be in the direction of 1 unit i, where i is in the direction of x:

$$\left[cos(180) + sin(180) * (i, 0j, 0k)\right] \tag{6}$$

1.5.2

Similarly, the quaternion q2 that represents the rotation of 180 degree about the z-axis, is found by inserting the angle of rotation (180), and defining the vector to be in the direction of 1 unit k, where k is in the direction of z:

$$\left[cos(180) + sin(180) * (0i, 0j, k)\right] \tag{7}$$

1.5.3

The rotation represented by the composite quaternion *q1q2*, is found by multiplying the q1 on the q2. As all the vector elements in the two quaternions are 0 elements, the resulting quaternion is:

$$\left[1 + 0i + 0j + 0k\right] \tag{8}$$

1.6 Exercise

1.6.1 Compose two rotation matrices

For the composition of two rotation matrices, two 3x3 matrices are multiplied by each-other, resulting in a total of 27 multiplications and 18 additions.

1.6.2 Compose two quaternions

To compose two quaternions, a total of 16 multiplications and 12 additions are needed.

1.6.3 Apply a rotation matrix to a vector

For applying the rotation matrix on a vector, a 3x3 matrix is multiplied by a 3x1 vector, resulting in a total of 9 multiplications and 6 additions.

1.6.4 Apply a quaternion to a vector

To apply a quaternion *q1* to a vector, the vector is turned into a pure quaternion p by adding 0 as a fourth element. By multiplying q*p with the conjugate of q, a new pure quaternion is calculated. Removing the real part (the 0), the vector part remains, which corresponds to the vector after applying the rotation. As the pure quaternion p has a zero-element, only 12 multiplications and 8 additions are performed for each quaternion multiplication. This means a total of 2*12=24 multiplications and 2*8=16 additions are needed for applying a quaternion to a vector.

2 Part 2

2.1 Exercise

2.1.1

The rotation matrix defined in the inertial coordinate system, is given by the multiplication of the rotational matrix of all three axes as shown in the equation 9.

$$\boldsymbol{R}(\gamma) = \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(\phi) = \begin{bmatrix} cos_\psi cos_\theta & cos_\psi sin_\theta sin\phi - sin_\psi cos_\phi & cos_\psi sin_\theta sin\phi + sin_\psi cos_\phi \\ sin_\psi cos_\theta & sin_\psi sin_\theta sin\phi + cos_\psi cos_\phi & sin_\psi sin_\theta cos\phi + cos_\psi cos_\phi \\ -sin_\theta & cos_\theta sin_\phi & cos_\theta cos_\phi \end{bmatrix} \tag{9}$$

2.1.2

The relation between the angular velocity and the rotational velocity is given by the equation in equation 10.

$$\vec{\omega} = \begin{bmatrix} 1 & 0 & -sin_\theta \\ 0 & cos_\theta & cos_\theta sin_\phi \\ 0 & -sin_\phi & cos_\theta cos_\phi \end{bmatrix} \dot{\vec{\theta}} \tag{10}$$

2.1.3

The linear dynamic equations are written in compact form in equation 11 and afterwards each component of the equation are written in equation 12 and equation 13.

Linear

$$m\dot{x} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} + RF_b + F_D \tag{11}$$

Where R is the rotation matrix in equation 9 and:

$$F_b = k \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} w_i^2 \end{bmatrix} \tag{12}$$

$$F_D = -k_d \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{13}$$

Angular The angular dynamic equations are written in compact form in equation 14 and afterwards each component of the equation are written in equation 15 and equation 16.

$$\dot{w} = I^{-1}(\tau_b - w \times (Iw)) \tag{14}$$

Where:

$$\dot{w} = \begin{bmatrix} \dot{w}_x \\ \dot{w}_y \\ \dot{w}_z \end{bmatrix} \tag{15}$$

$$\tau_b = \begin{bmatrix} Lk(w_1^2 - w_3^2) \\ Lk(w_2^2 - w_4^2) \\ b(w_1^2 - w_2^2 + w_3^2 - w_4^2) \end{bmatrix} \tag{16}$$

2.1.4 Simulink

The model was made in Simulink according to equations presented above. Figure 1 shows the model. Thereafter, the simulation was run given three different values for the angular speed of the four propellers of the drone. For each value of Ω, a plot of both the position of the drone and it's angle was made. These plots are shown below.

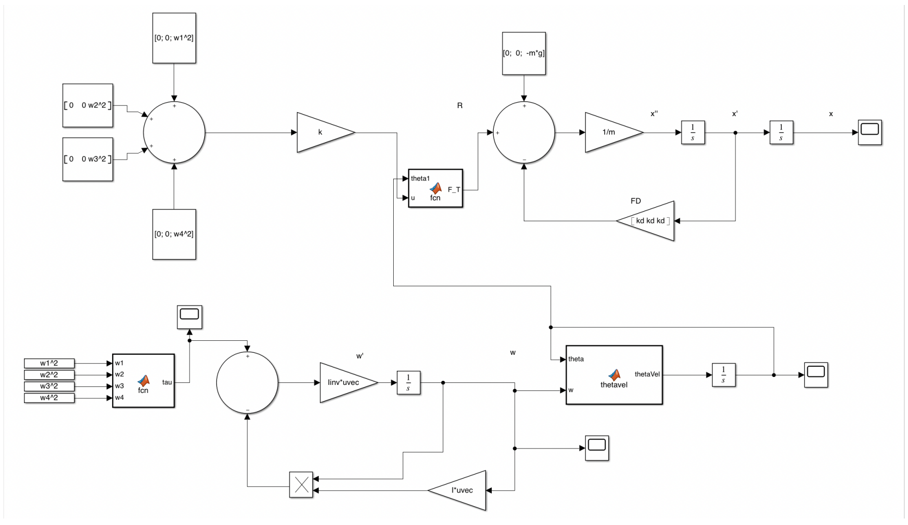

Figure 1: Simulink System

First run
For the first run the following input where used.

$$\Omega = [0; 0; 0; 0]^T \tag{17}$$

Which means all of the propellers where turned off, and therefore, there is nothing that has an impact on the system besides the mass and gravity $-mg$. In figure 2 it can be seen that the drone will fall in the Z-direction, because of the propellers being turned off, and no ground is defined in the model.

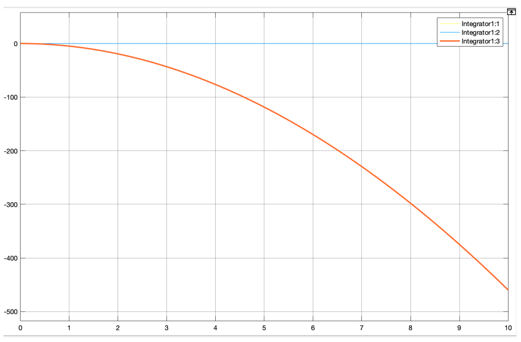

Figure 2: $\Omega = [0;0;0;0]^T$

Furthermore no change is taking place at rotations.

Second run
For the second run the following input where used.

$$\Omega = [10000;0;10000;0]^T \tag{18}$$

This time two of the opposite propellers are turned on, and thus a lift and rotation occurs on the body of the drone. The elements that are affected are lift direction in Z in figure 3 and its rotation round Z in figure 4

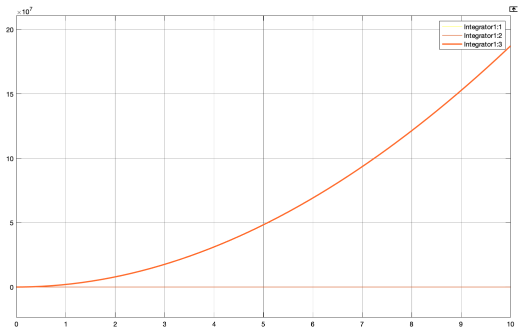

Figure 3: Z-direction $\Omega = [10000;0;10000;0]^T$

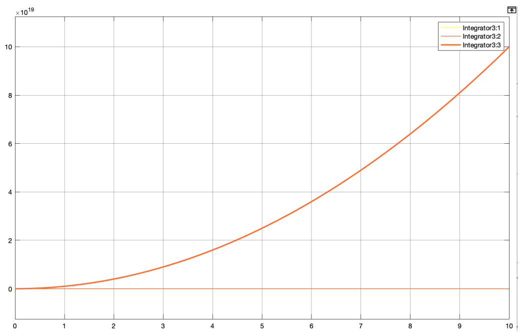

Figure 4: Z-rotation $\Omega = [10000;0;10000;0]^T$

Third run
For the second run the following input where used.

$$\Omega = [0;10000;0;10000]^T \tag{19}$$

Relative to the second round, it is now the last 2 propellers that are activated, and therefore the overall outcome will be the same as seen in figure 5 and figure 6

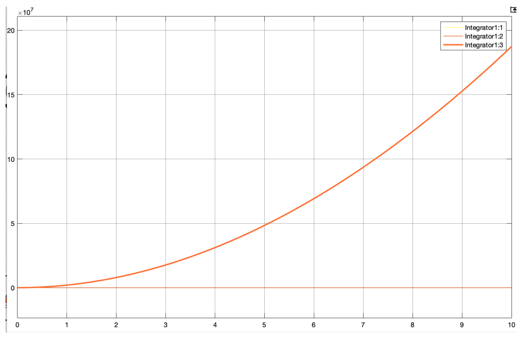

Figure 5: Z-direction $\Omega = [0; 10000; 0; 10000]^T$

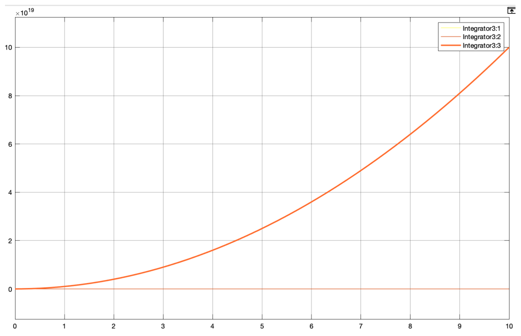

Figure 6: Z-rotation $\Omega = [0; 10000; 0; 10000]^T$

2.2 Exercise

Quaternion rotation from body to inertial.

To rotate from the inertial frame to the body-fixed frame, two vectors are initialised. The first one being the reference in the inertial coordinate system, the other being attached to the body-fixed frame. Similarly to exercise 1.4, it is possible to use the cross product to find the axis of rotation, and the the dot product for finding the rotation.

2.3 Exercise

For the linearization of the model, the assumption of small angles were used. For small angles we can assume cos(θ) = 1-θ and sin(θ) = θ. To linearize the model, all instances of cosine and sinus were substituted using this assumption.

3 Part 3

3.1 Exercise: PID control on the non-linear system

To add the attitude PID controller to the system, the non-linear simulink model from Part 2 (*figure: 1*) was made into a subsystem block diagram in Simulink, with the input being the angular speeds of the propellers *wi*, and the outputs being the rotational angles *theta*, the updated rotation matrix R and the xyz poistion of the body frame. See fig. 7

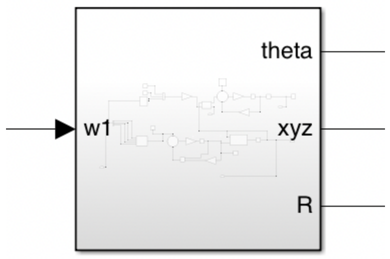

Figure 7: Subsystem

The control loop for controlling the attitude parameters of the drone, *theta*, consisted of a PID controller taking the error in the error functions of the three rotational angles as inputs, and a function relating the PID controller outputs to the propeller speeds. In figure 8 the controller loop can be seen. The Fc input for z-direction height and the set points for the attitude is controlled from a constant block.

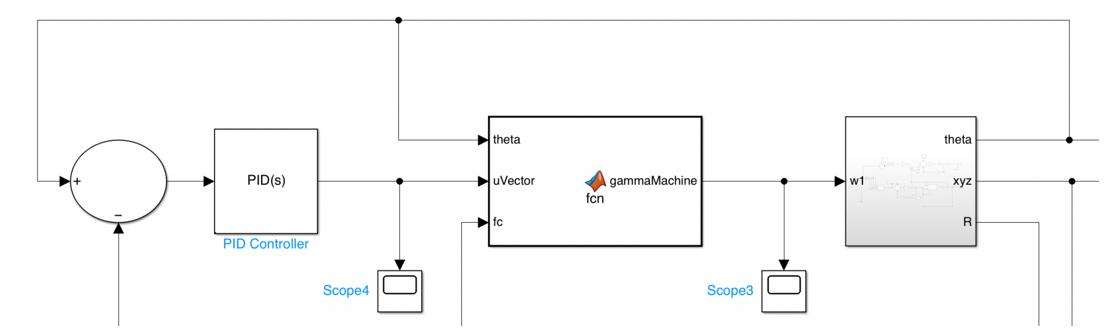

Figure 8: PID attitude controller

The function relating PID control outputs to propeller speeds, is based on the formula given in figure 9. Here the u corresponds to the output of the PID controller, phi, theta and psi being the inputs angles from the subsystem, and the additional variables being constants specific for this drone setup.

$$\gamma_1 = \frac{mg}{k4cos\theta cos\phi} - \frac{2bu_\phi I_{xx} + u_\psi I_{zz} kL}{4bkL}$$

$$\gamma_2 = \frac{mg}{k4cos\theta cos\phi} - \frac{2bu_\theta I_{yy} - u_\psi I_{zz} kL}{4bkL}$$

$$\gamma_3 = \frac{mg}{k4cos\theta cos\phi} - \frac{-2bu_\phi I_{xx} + u_\psi I_{zz} kL}{4bkL}$$

$$\gamma_4 = \frac{mg}{k4cos\theta cos\phi} - \frac{-2bu_\theta I_{yy} - u_\psi I_{zz} kL}{4bkL}$$

Figure 9: Gamma equation

Tuning the PID controller to a critical damped state, was done through an experimental approach. The most optimal gain values for the PID was found to be P = 1, I = 0.2 , D = 4 . Below the rotational and height responds for the four states can be seen.

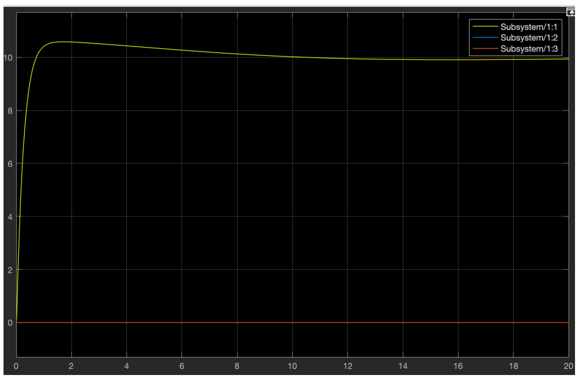

Figure 10: $\Omega = [10; 0; 0]^T z = 0$

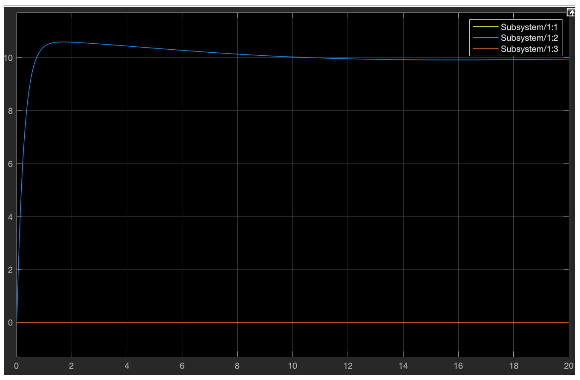

Figure 11: $\Omega = [0; 10; 0]^T z = 0$

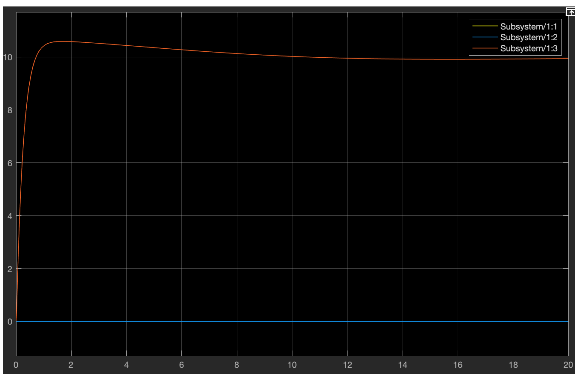

Figure 12: $\Omega = [0; 0; 10]^T z = 0$

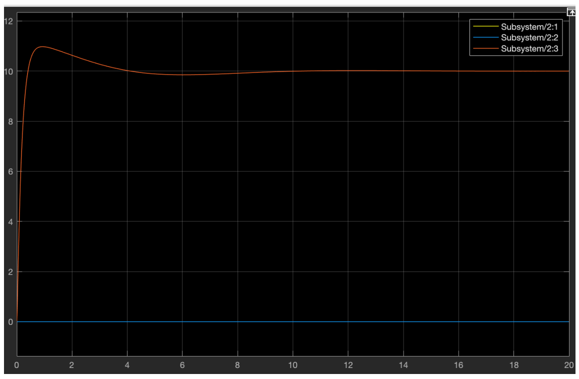

Figure 13: $\Omega = [0; 0; 0; 0]^T z = 1$

3.2 Exercise: Comparison of the control of the linear and non-linear system

Comparing the linear and non-linear system designed in terms of control responds revealed that the control responds for the linear system were...

In the figures below the graphed control responds from the linearized model can be seen.

3.3 Exercise: Position control of X and Y

To add control of the drone through x and y set points, an additional PID controller is added to the controller system. This controller controls the set points of the attitude controller through the relation given by:

$$\begin{bmatrix} \phi^\star \\ \theta^\star \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} R \begin{bmatrix} PID(e_x) \\ PID(e_y) \\ 0 \end{bmatrix}$$

Figure 14: [x,y] world coordinates related to roll and pitch set points

As seen in the equation a PID output from the error function of x and y, are needed. These are calculated through the new PID control loop, and afterwards the output of the controller is signaled in to a function block that calculates the new rotational set points using the equation. The rotation matrix comes from the subsystem, and the PSI setpoint from a constant. The PID for the e(z) signal is used as input in the Fc controller, as described in:

$$\Sigma \gamma_i = \frac{\mathrm{mg} + f_{c,z}}{\mathrm{k}\cos\vartheta\ \cos\varphi}$$

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

Figure 15: PID for x and y

The new control loop can be seen:

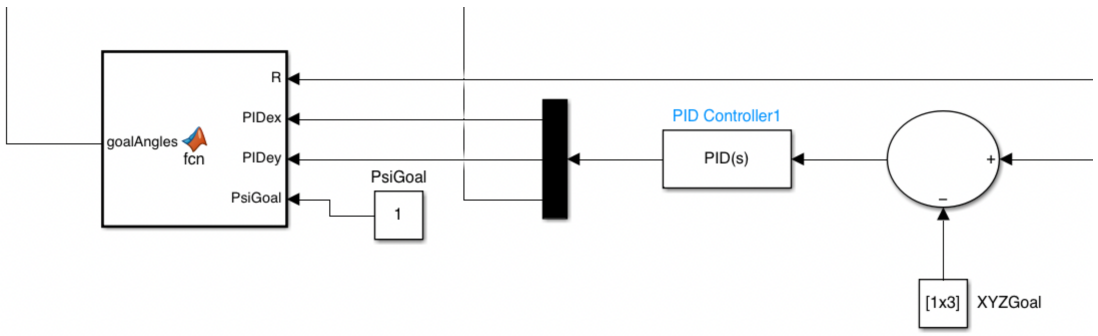

Figure 16: PID for x and y

The complete control loop for controlling the drone from x and y setpoints is seen in the fig 17:

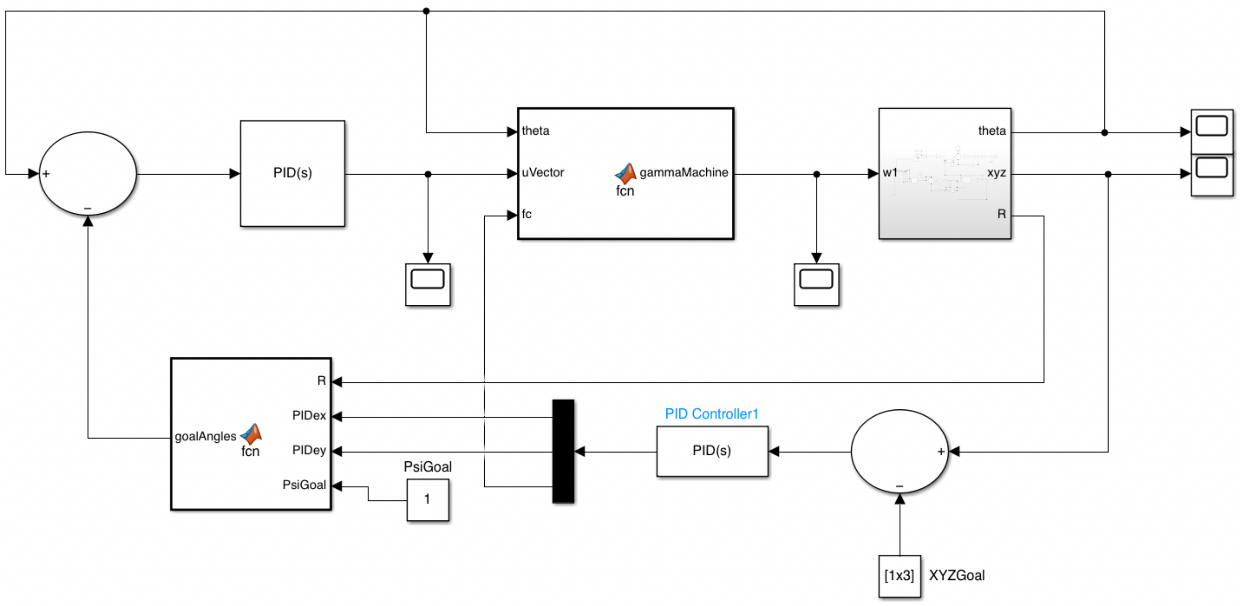

Figure 17: Complete PID loop

4 Part 4

4.1 Depth- first search (DFS) algorithm

The DFS algorithm starts at the designated starting node and travels down each branch as far as possible before backtracking. The algorithm backtracks when it reaches a node it has visited before. After backtracking, another node is chosen to travel to and this is repeated until the end node is found. The sorting mechanism of the algorithm is FIFO (First one in, first one out), feasibility is guaranteed but not optimality.

4.1.1 Forward run

In the forward run, the algorithm always to chooses to explore the node with the lowest number first. Thus, when moving down the path it chooses to visit the nodes with the lowest number until it has to backtrack. Since the end node in this graph has a high number, this resulted in a very long path to the end point. The path is shown in black and the backtracking of the algorithm is shown in red in figure 18.

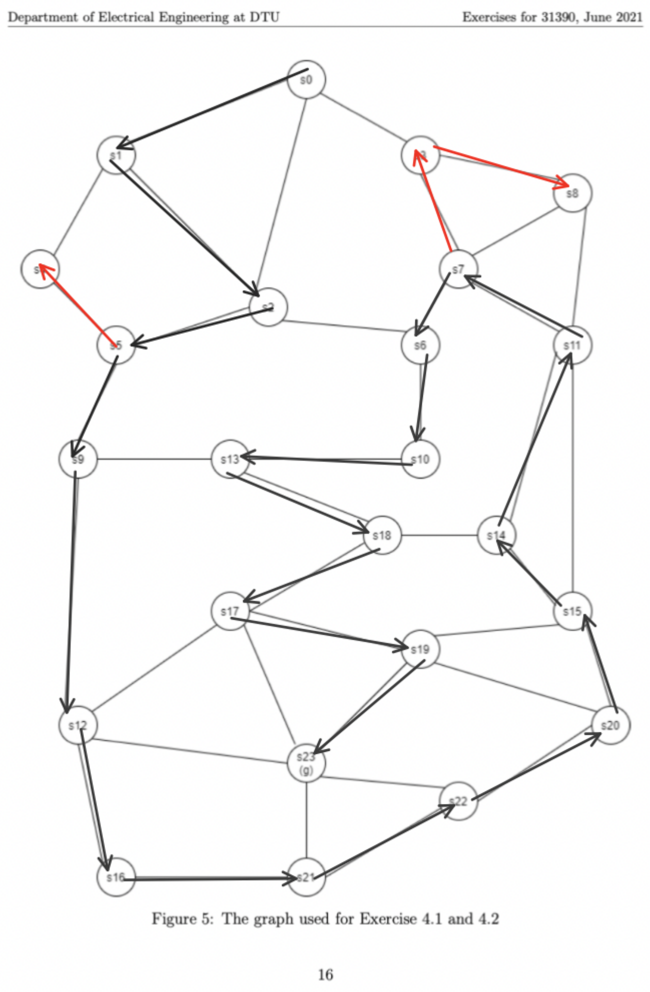
Department of Electrical Engineering at DTU
Exercises for 31390, June 2021
s0
s1
s7
s8
s6
s11
s9
s13
s10
s18
s14
s17
s15
s19
s12
s20
s23
(g)
s22
s16
s21
Figure 5: The graph used for Exercise 4.1 and 4.2
16

Figure 18: DFS forward run

4.1.2 Backward run

In the forward run, the algorithm always to chooses to explore the node with the highest number first. This run was a lot more efficient then the forward run, since the end node has the highest number of all nodes in the graph. The path is shown in black in figure 19, where no backtracking was needed.

Department of Electrical Engineering at DTU Exercises for 31390, June 2021

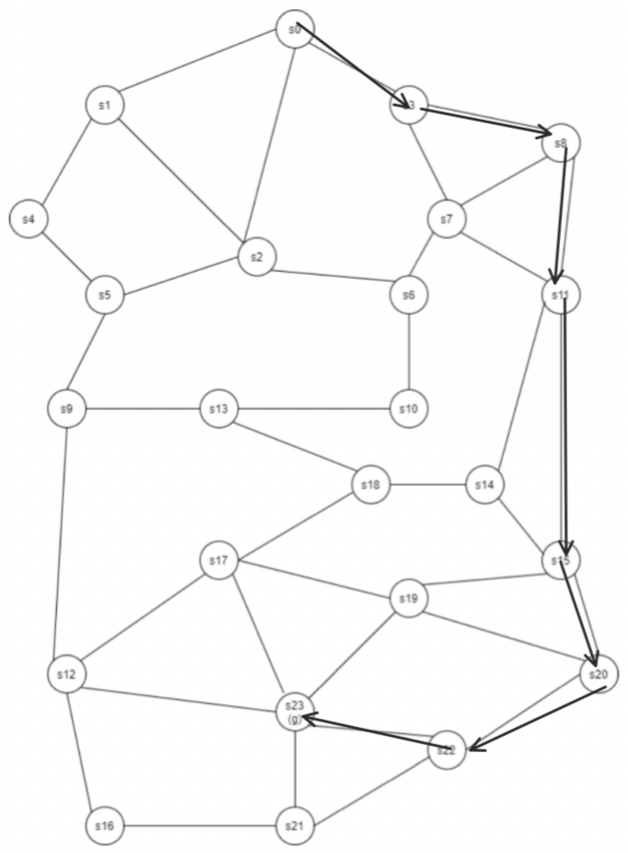

Figure 5: The graph used for Exercise 4.1 and 4.2

16

Figure 19: DFS backward run

4.1.3 Difference in paths

It is obvious that there is a significant difference in the paths of the two different DFS methods. The reason being that when the forward method is run, the algorithm will favor nodes of small numbers to those with larger numbers. The algorithm might, therefore, be positioned next to the end node but chose to move in the direction away from it, towards a node with a lower number. The backwards run will, however, always favor the end node over all other nodes since its number is the highest of all nodes.

4.2 Breadth- first search (DFS) algorithm

The BFS algorithm uses queue data structure to explore the given map. This means that it follows the LIFO sorting mechanism (Last one in, first one out).The algorithm does not guarantee optimality and is feasible for finite state spaces. It starts at the starting point and explores all of the neighbor nodes before moving on to the nodes at the next depth level. The algorithm terminates when the goal node is found. The path found with DFS is shown in figure 20.

Department of Electrical Engineering at DTU Exercises for 31390, June 2021

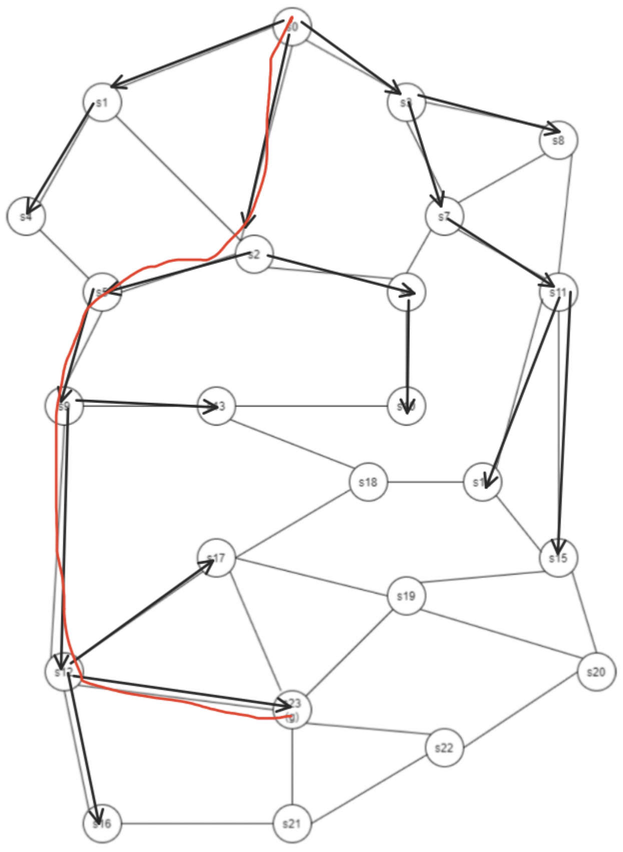

Figure 5: The graph used for Exercise 4.1 and 4.2

Figure 20: BFS forward run

4.2.1 DFS vs. BFS

The main advantage of using BFS is that if a solution exists, the algorithm will find it. Additionally, if more than one path exists, it will find the shortest one. The disadvantage is memory constraints since it stores all the connected nodes. The main disadvantage of DFS is that it is not guaranteed to find the solution and the time complexity may be more if it gets trapped searching a useless path. However, it consumes less memory and the solution can be found using fewer searches.

Since a robot can fall into a limitless loop investigating a single node's depth when using DFS, BFS is preferred since it will always find the shortest path.

4.3 Dijkstra's algorithm

Dijkstra's algorithm can be used to find the shortest path through the graph. The algorithm starts at node 0 and registers the distance from every neighbor to the starting point, 0, if the distance is smaller than the currently registered distance. In the first round every distance is set to infinity and, therefore, all distances should be switched out. The algorithm also notes down from which node the current node derived from. Once the algorithm has visited a specific node, it does not visit it again. Therefore, a record of Visited and Unvisited nodes is held. Furthermore, the algorithm always visits the node that has not yet been visited and currently holds the lowest distance to the starting point.

Visited = [0, 3, 7, 1, 2, 8, 6, 11, 4, 10, 5, 9, 14, 15, 13, 18, 20, 19, 23]

Unvisited = [16, 21]

When the algorithm is finished running, table 1 has been formed and can be used to find the shortest path through the map. In this case, the shortest path is: 0-3-7-11-14-15-19-23. This path can also be seen in figure 21

Vertex	**Shortest distance from 0**	**Previous vertex**
0	0	0
1	4.5	0
2	5.4	0
3	2.2	0
4	6.7	1
5	8.1	4
6	5.8	7
7	4.4	3
8	5.4	3
9	10.3	5
10	7.8	6
11	6.6	7
12	16.3	9
13	11.8	10
14	10.7	11
15	11.1	14
16	INF	
17	16.3	18
18	12.7	14
19	14.3	15
20	13.2	15
21	INF	
22	15.8	20
23	17.1	19

Table 1: Dijkstra's Algorithm

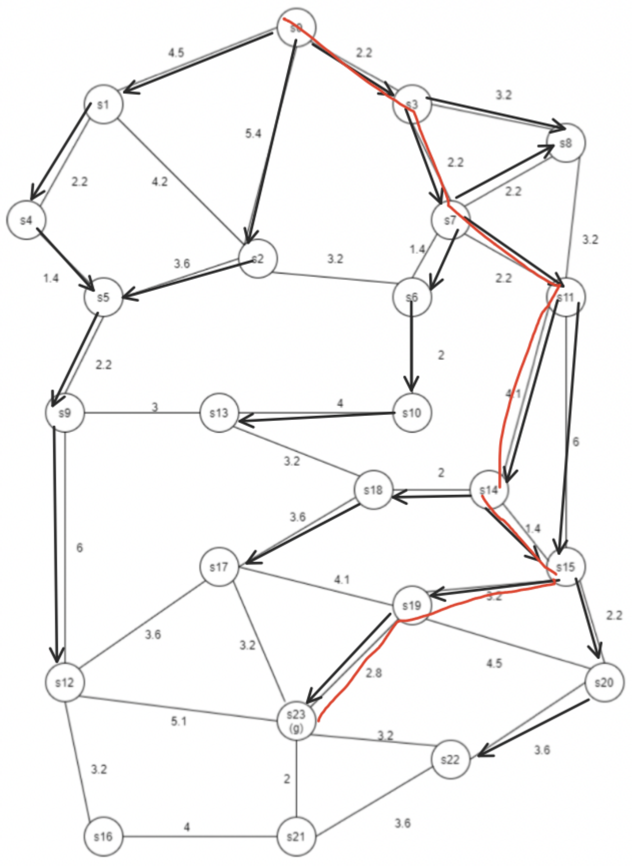

Figure 6: The graph used for Exercise 4.3

Figure 21: Dijkstra forward run

4.4 Greedy best- first search algorithm

The Greedy algorithm takes into account each nodes' direct-line distance to the end point. This node is thought to be most promising since it leads the algorithm to the closest known node to the end node. The algorithm uses a priority queue as a sorting mechanism were it prioritises nodes with the shortest distance to the end goal. Feasibility is guaranteed but not optimality since it does not take the cost into account. Figure 22 shows the path found with the Greedy algorithm.

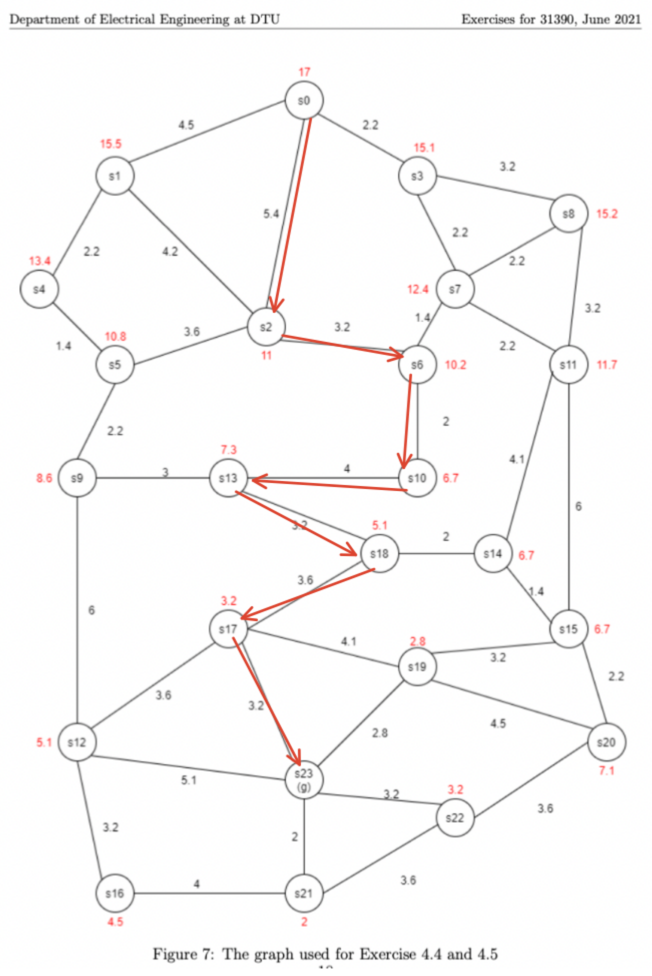

Figure 22: Greedy algorithm forward run

4.5 A* search algorithm

Compared to the other algorithms that have been used before, the A* algorithm is dependent on both the direct distrance to the end goal and the total travel length so far. Which means that there are several parameters that determine the fastest route.
Thus, the algorithm can be written up in the following way:

- G-cost = distance from starting node
- H-cost = distance from end node
- F-cost = G cost + H cost

The path with the lowest F-cost is thus the most optimal route and the F-cost structure can be seen in figure 23

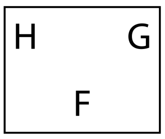

Figure 23: A* algorithm structure

The full path of the A* algorithm is seen in figure 24,

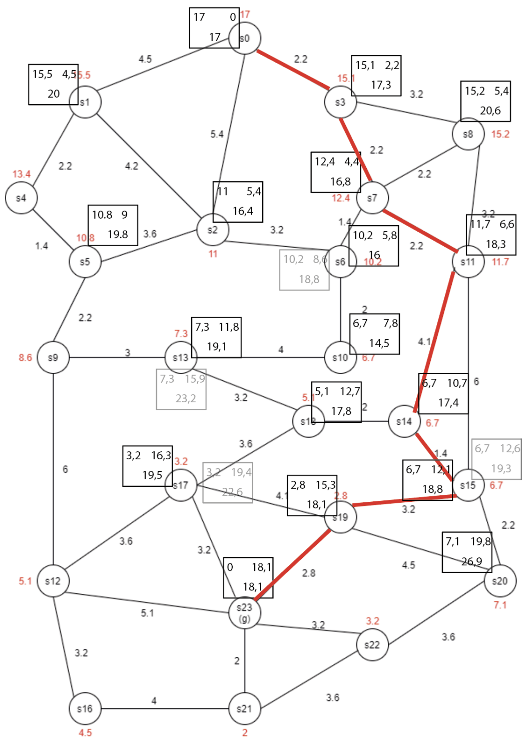

Figure 24: A* algorithm path

When looking at figure 23 and figure 21, it is seen how they have both taken the same route from S_0 to S_23. One of reasons to use the A* algorithm is, that it makes use of both start and end distance, relative to Dijkstra that only uses the total distance traveled.
Which means that the A* algorithm will take the shortest route by evaluating along the way, while the Dijkstra would keep unnecessarily exploring the map.

4.6 Greedy best-first vs. A*

4.6.1 Greedy best-first

The disadvantage of Greedy Best First Search is that it is not always optimal nor complete. The advantages are that it's approach is simple to implement, the algorithm typically has less time complexity and it is in most cases very efficient.

4.6.2 A*

Contrary to Greesy best-first, the A* algorithm is both complete optimal. It is very efficient in the way that no other algorithm is guaranteed to expand fewer nodes than A*. Its limitations are that the speed execution of A* search is dependant on the accuracy of the heuristic algorithm that is utilized to compute h(n), the actual cost of the path from the current state to goal state. Furthermore, it uses more memory than Greedy best-first.

4.6.3 Best for robotics

Since Greedy BFS is neither complete, nor optimal whereas A* is both, A* is often preferred in robotics.

4.7 Upgrade the Greedy best-first search from 2D to 3D

The provided code which implemented the Greedy best-first approach in 2D was changed to 3D by adding the z-axis to the program. The *map_script_3d.m* was used and the function *calc_dist_3d* which were both given. It was then only a matter of added the third *for_loop* when searching for the all the neighbors of the current node. This third loop was the z-axis. The conditions of every *if_statement* then needed to updated to also take the z-axis into account. The resulting map and path is shown in figure 25.

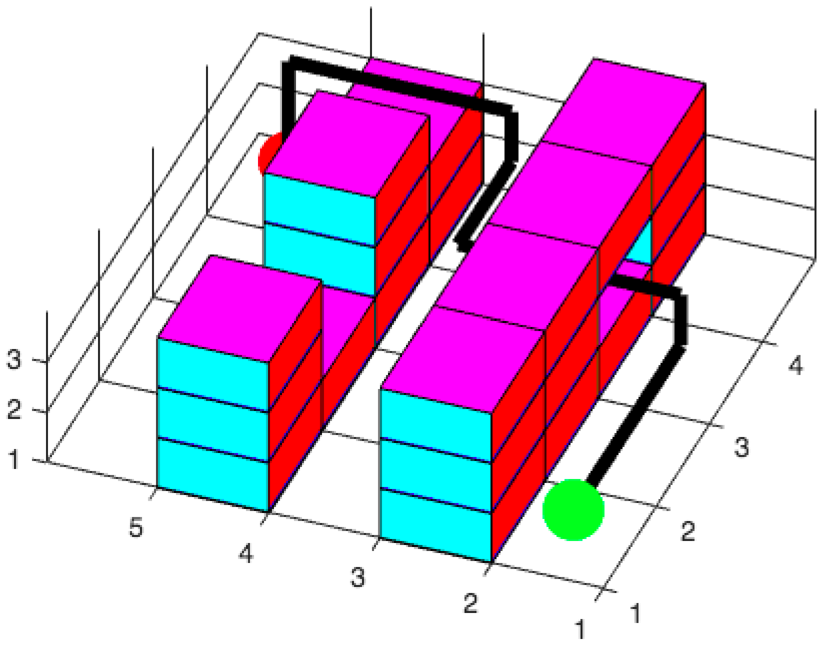

Figure 25: Greedy best search algorithm run on map with walls in 3D

4.8 Modify Greedy best-first into A* search

For the Greedy algorithm the cost was simply f(n) = h(n), but for the A* search method, the cost is now as f(n) = h(n) + g(n). Where h(n) is the distance to the goal, and g(n) is the cost between nodes.

This was implemented in Matlab by changing the variable f(n) to also take into account g(n) and adding a code that calculates the cost between nodes. Since every step in the map is of length 1, diagonal movements are not allowed, the cost between nodes is always one. The function g(n) is therefore calculated as a counter of how many steps were taken from the start point to get to the current point. In the code this is done by using the g variable of the parent node and adding 1 to it. The parent node is then updated to be the child node with the lowest f value until the goal is found. The results can be seen in figure 26. It can be seen that the algorithm takes a different route than Greedy best-first, however the step count is the same. Both algorithms have the final step count of 11 steps.

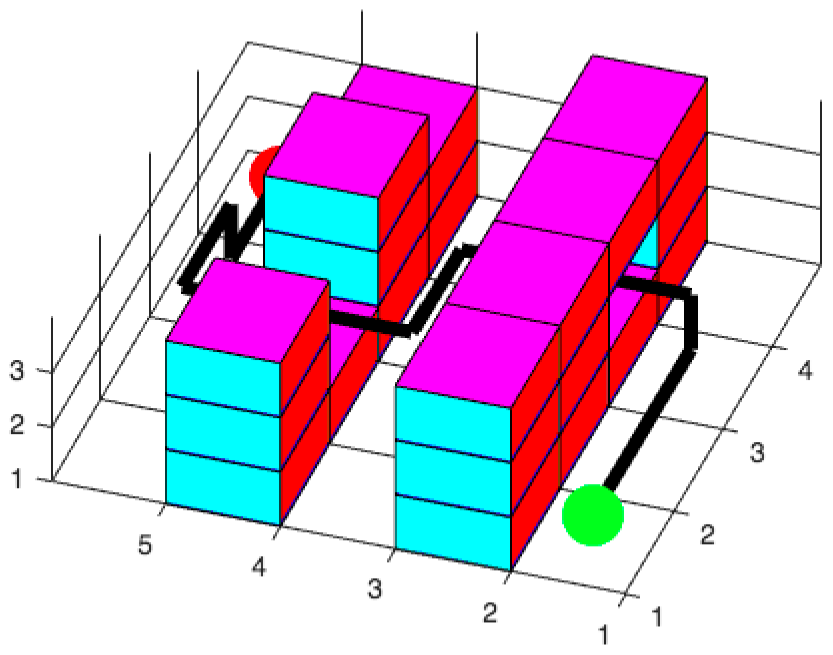

Figure 26: A* algorithm run on map with walls in 3D

What algorithm would be implemented if the formula was f(n)=g(n) instead?

If we were to change to code so that the cost formula would be f(n)=g(n) instead, the algorithm would be only look for the path which takes the fewest steps. Meaning the Dijkstra's algorithm would be implemented.

5 Part 5

5.1 Quintic splines

The following equations were given and the goal is to compute the constants α_i where i= 0:5.

$$x(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \alpha_3 t^3 + \alpha_4 t^4 + \alpha_5 t^5 \tag{20}$$

6 equations are made, since 6 constraints were given.

$$x(0) = 0; \dot{x}(0) = 0; \ddot{x}(0) = 0; x(1) = 1; \dot{x}(1) = 0; \ddot{x}(1) = 0 \tag{21}$$

These constraints were used to compose a matrix of differential equations that was said to be equal to the constraints. The end result was the following matrix.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 2 & 6 & 12 & 20 \end{bmatrix} \vec{\omega} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{22}$$

After calculating the results, the following answers for the α_i variables were gotten.

$$\vec{\omega} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 10 \\ -15 \\ 6 \end{bmatrix} \tag{23}$$

5.2 B-splines

5.3 Computation of a DCM

To calculate R_B some assumption are made to simplify the overall system. By saying z_B are placed in the center of the coordinate system, and because x_C, is 0 at the Z axis the x_C plane is placed an arbitrary place on z_B. This first step is thus to find y_C, by using x_C which is given in the task.

$$x_C = \begin{bmatrix} cos(\psi) \\ sin(\psi) \\ 0 \end{bmatrix} y_C = \begin{bmatrix} cos(\psi + \pi/2) \\ sin(\psi\pi/2) \\ 0 \end{bmatrix} = \begin{bmatrix} -sin(\psi) \\ cos(\psi) \\ 0 \end{bmatrix} \tag{24}$$

As seen in 24, y_C is found by finding the opposite vector plan of x_C. Which is still crossing the z_B vector.
This now allows to find the x_B, by assuming y_C is an arbitrary vector plan which passes trough z_B, as shown in figure 27 and finding the cross product between y_C and z_B

$$x_B = z_B \times y_C \tag{25}$$

Since y_C is assume to be placed on the entire surface of Z_B thwn X_B must be orthogonal to y_C and X_B as shown in figure 27.
Now is possible to find the last vector Y_B, by using the cross product from X_B and Z_B.

$$y_B = z_B \times x_B \tag{26}$$

The output is seen in figure 27

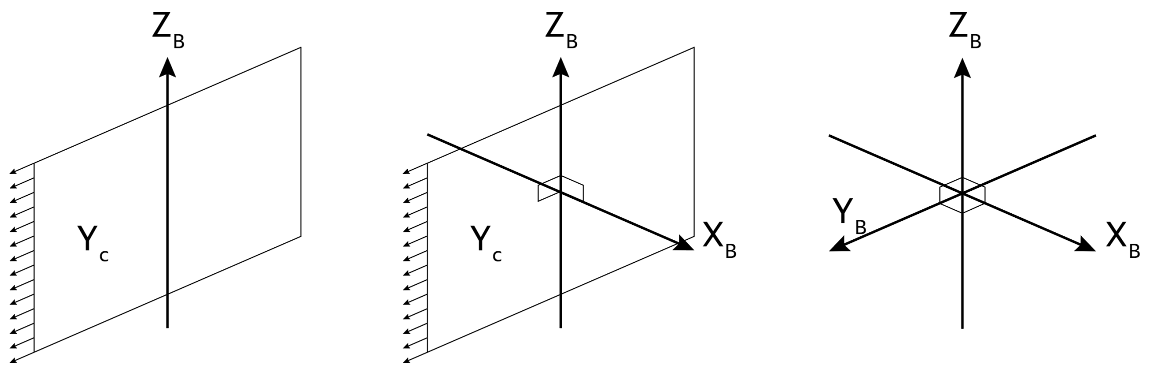

Figure 27: Computation of a DCM

Thus, we have found all the vectors x_B, y_B and z_B, and thus is possible to construct the rotations matrix from the body frame R_B

$$R_B = \begin{bmatrix} x_B & y_B & z_B \end{bmatrix} \tag{27}$$

6 Part 6

6.1 Navigating a 2D maze

To find a route from (0,0,1) to (3,5,1), the A* algorithm from exercise 4.8 were used for path planning. As the drone is having a steady position in the Z direction, the route is planned using the 2D path planner.

The full route can be seen in figure 28

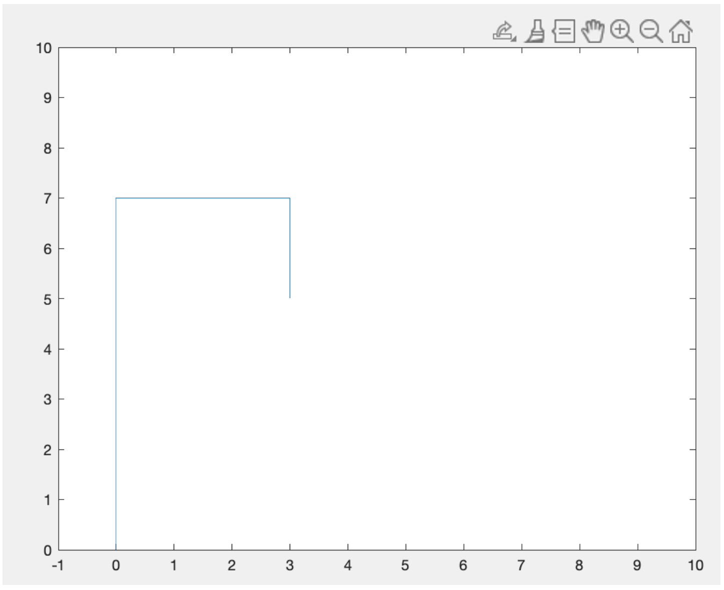

Figure 28: Route from 2D A* planner. Horizontal axis is y.

Table 2: Paths

Table 3: Coordinates

X	Y	Z
0	0	1
9	0	1
9	9	1

Table 4: A* algorithm

X	Y	Z
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1
5	0	1
6	0	1
7	0	1
7	1	1
7	2	1
7	3	1
6	3	1
5	3	1

6.2 Re-implementing the position controller

For the implementation of the position controller, new subsystem taking position, the rotation matrix, position set points and yaw angle as inputs. See figure 29.

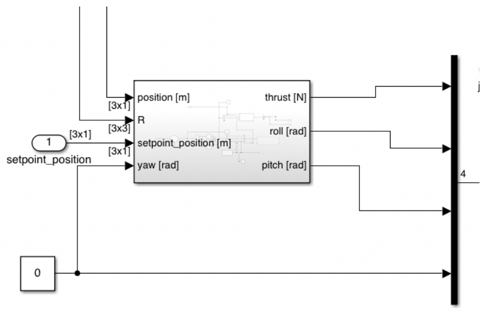

Figure 29: The new controller subsystem

For the rotation matrix input, the rotation matrix function used in exercise 2 is inserted, taking the angular rotations as inputs. See figure 30.

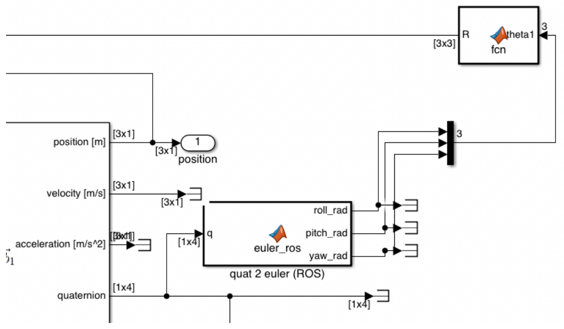

Figure 30: Rotation matrix inputs

Inside the controller subsystem from 29, three PID controllers are inserted on the error calculator. The PIDx and PIDy are signaled into the roll and pitch calculator from exercise 3. For the thrust calculation, the gravity compensation has to be taking into account by adding the mass*g to the thrust signal. See figure 31.

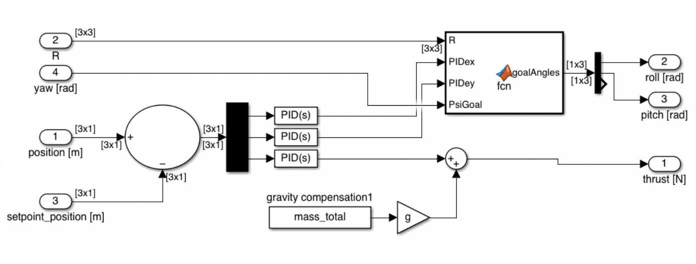

Figure 31: The new position controller

After the implementation a test were made, setting the position set points to make the drone hovering in Z=1, and flying to respectively x=1, x=3 and x=9. See figures 32 to 34. As can be seen on the picture there are both overshoot and undershoot, hover very low steady state error. The over and undershoot could be solved by tweaking the P and D parameters.

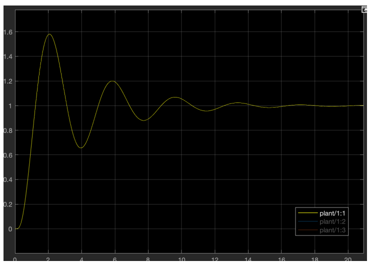
Figure 32: 1m in the x-direction

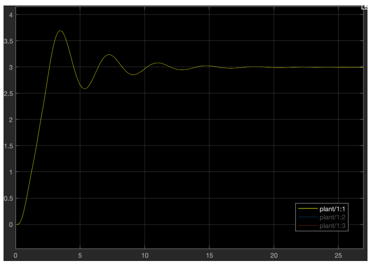
Figure 33: 3m in the x-direction

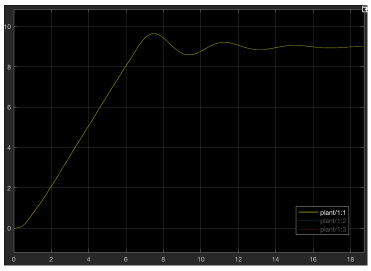
Figure 34: 9m in the x-direction

6.3 Re-implementing the attitude controller

6.4 Aggressively navigating the maze

For aggressively navigating the maze, the traj-gen matlab plugin was used. The plugin creates a spline trajectory based on volumes and waypoints to visit at certain time stamps. For navigating the maze, two waypoints were positioned, one at the start and end coordinates. Near the corner of the maze two corridors were positioned, to control how the drone cuts the corner. An order of 7 was sufficient to calculate the route, with the above contraints and a 5 second knots window. See figure 35 and 36 for the planed and actual trajectories. Figure 37 shows the drone's position over time and can be used to see that the drone managed to move to point (9,9,1) within 5 seconds.

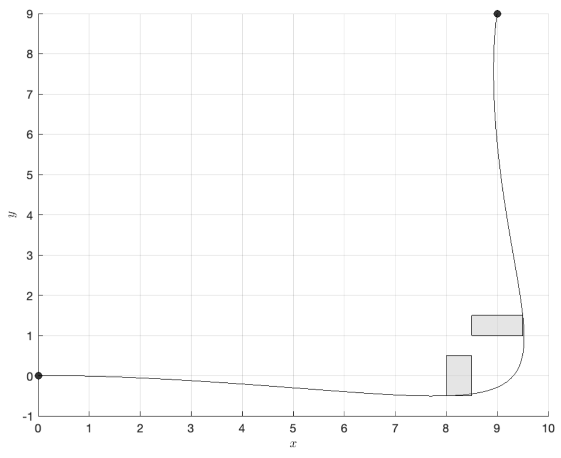

Figure 35: Trajectory

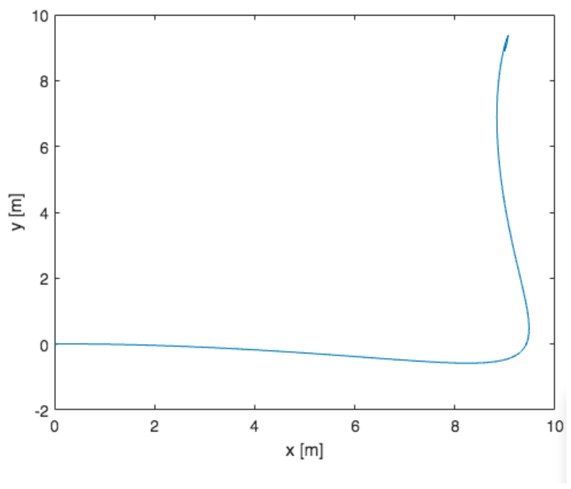

Figure 36: X-Y plot

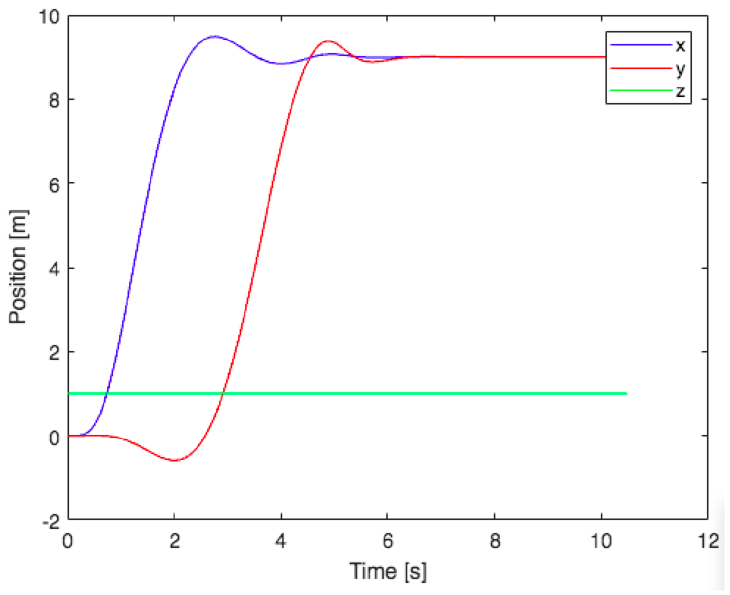

Figure 37: Trajectory

7 Part 7

This exercise was completed using the default position controller.

7.1 Navigating a 3D maze

For the navigating the 3D maze in practice, the A* algorithm where used to navigate through the obstacle maze. The path that the drone were to take was recreated in matlab, and then implemented into the drone route.

The map and the route can be seen in figure 38

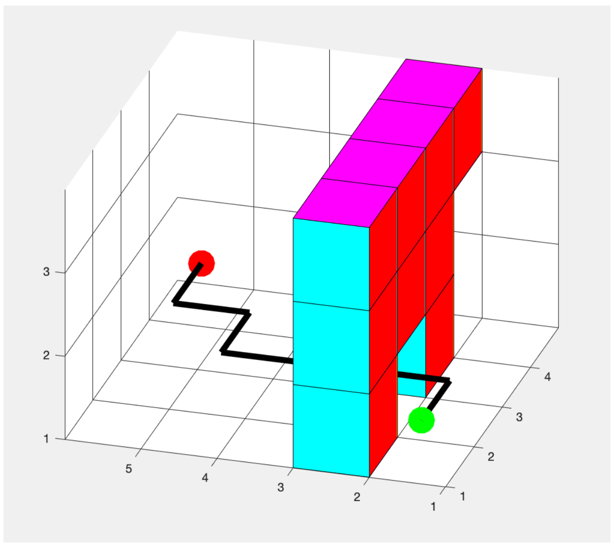

Figure 38: Trajectory

The controlling used for the drone were the one handed out for the course, thus minimizing the risk of errors that could occur under the test flight.

To see out result from figure 38 press the following: https://youtu.be/I5w5I3CLj7U

Various routes To check the stability of A* algorithm and the controller, different routes where created to investigate the various routes the drone would take and how it would behave under flying. Two examples of different routes is seen in figure 39 and figure 40.

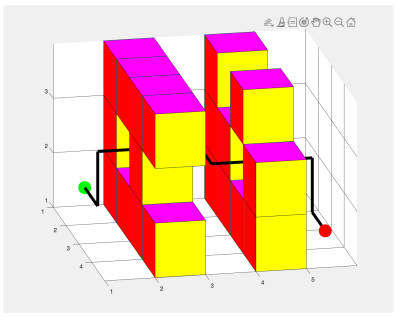
Figure 39: Test Route 1

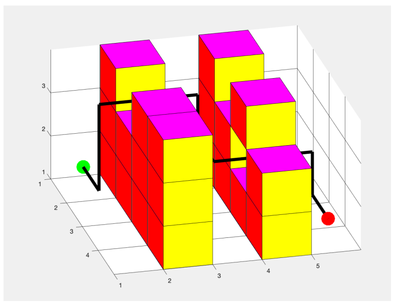
Figure 40: Test Route 2

The issues that arose by changing the route, where the scaling in the Z directing, and thus had to be fined tuned to the different heights which could occur on the different routes.

To see out result from test route 2 press the following: https://youtu.be/6EH4EAUrhok

7.2 Porting the position controller to the real system

In order to be able to feed the drone correct values corresponding to what it was needed to do, the input signal to the drone was altered. This was done by identifying the relationship between the thrust commands calculated by the position controller and the PWM commands accepted by the real drone's attitude controller.

This relationship was calculated by using a joystick to fly the drone manually in attitude mode. The data gathered from this experience was then used to fit a linear model in Matlab. The result was the following equation:

$$y = -11.5898x + 11.7986 \tag{28}$$

These two variables, slope (p2) and intersection (p1) at the y-axis, are used in the model in the subsystem called thrust mapping.

7.2.1 Hover stably on point

The setpoint for hovering was set to (0, -2, 1). Figure 41 shows the x,y and z positions over time and can be used to see how the drone hovered in the correct place.

The error is shown in figure 42 and is calculated as the position of the drone minus the setpoint. Whilst the drone is hovering, the three errors hover around zero as well. Figure 43 shows how the drone hovered at the point (x,y)=(0,-2) with the error of approximately ±0.02m in both directions.

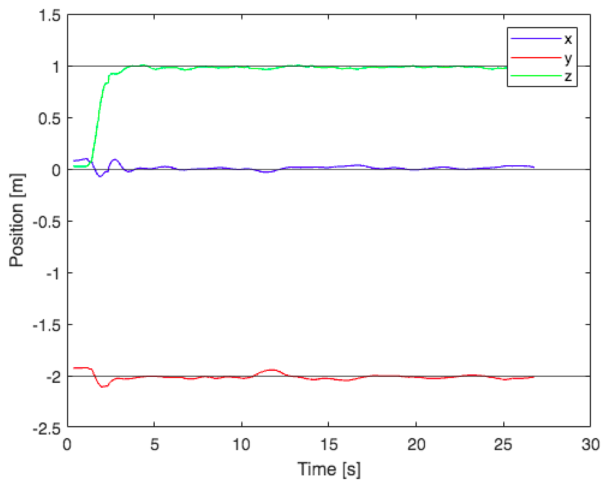

Figure 41: x,y and z values over time

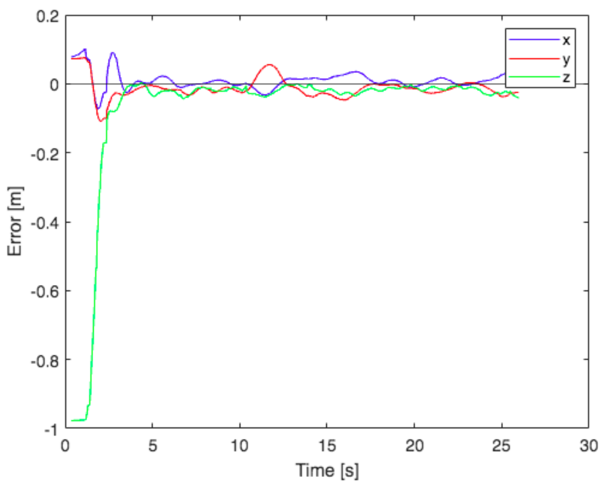

Figure 42: x,y and z errors over time

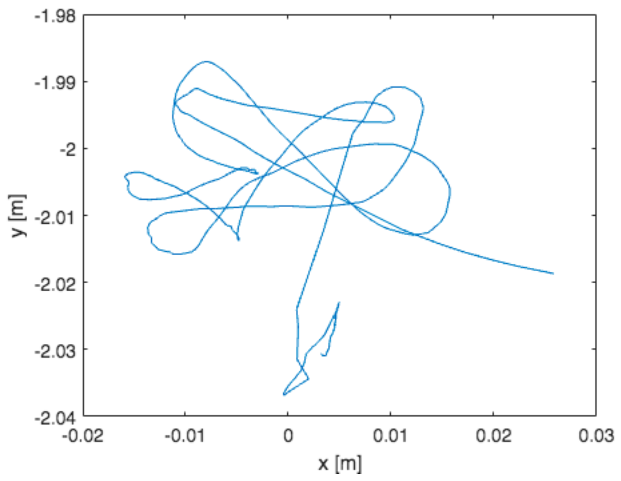

Figure 43: Plot of x versus y

To see out result from figure 41 press the following: https://youtu.be/K7d6sdqMfTE

In another attempt, we placed the drone in an random stop at the field, to see how it would hover over to the center position, https://youtu.be/nyfRcZuOyfQ

7.2.2 Move from setpoint to setpoint

Here the initial setpoint was to be the same as before and then after 20 seconds, a step input moved the setpoint by 1m in the positive x-direction. Figures 44 and 45 show that the drone moved by 1m with very little overshoot, approximately 0.1m.

Figure 45 shows the error of each axis during the hovering and moving. The error is quite stable around zero, until the setpoint is changed at time 20s. Within 5 seconds the values have reached their new setpoint and the error is again stable at around zero until the drone is shut off.

Finally, figure 46 shows how the drone hovered at x=0 and y=-2, moved towards x=1 and y=-2 and hovered there.

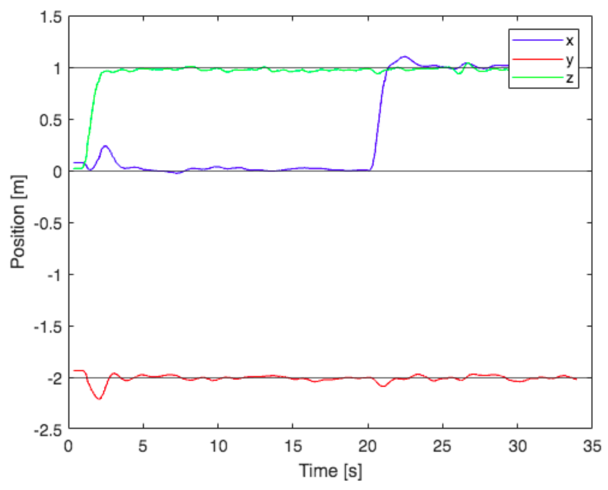

Figure 44: x,y and z values over time

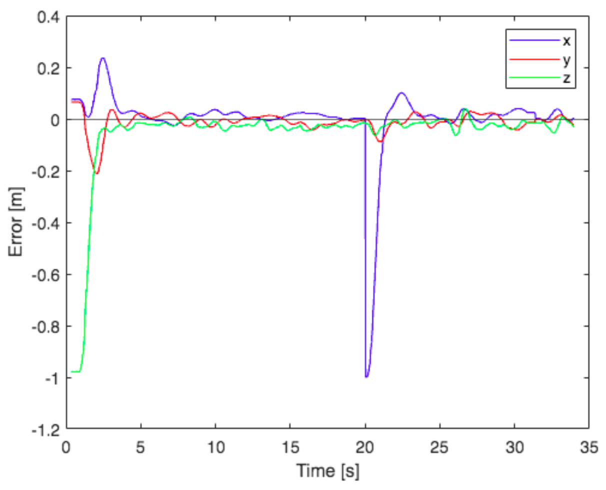

Figure 45: x,y and z errors over time

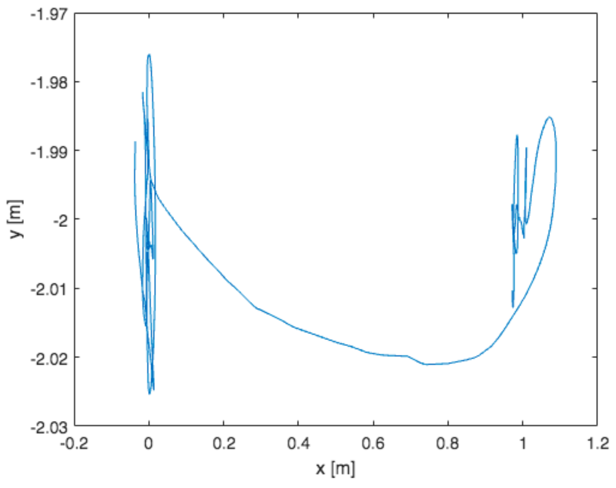

Figure 46: Plot of x versus y

To see the result from figure 46 go to the following link https://youtu.be/Do2fCgye_6I

7.3 Aggressively navigating through hoops

For aggressively navigating the maze through the hoops, the traj-gen plugin from exercise 6.4 were used. Corridors were positioned both at the hoops, to control the drone through the hoop, but also at the corners make sure the drone enters the hoops right in front. An order of 14 was sufficient to calculate the route, with the above constraints, and a 30 seconds knots window. See figure 47. The timestamps can be seen in figure 48.

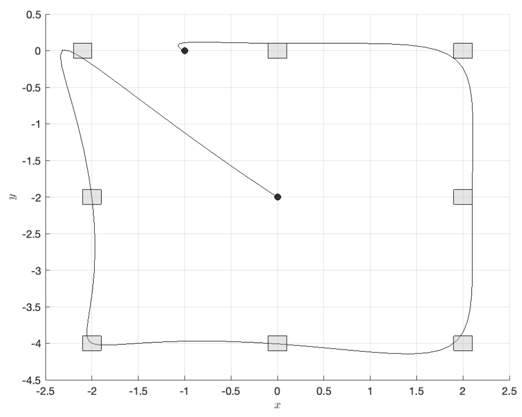
Figure 47: Trajectory

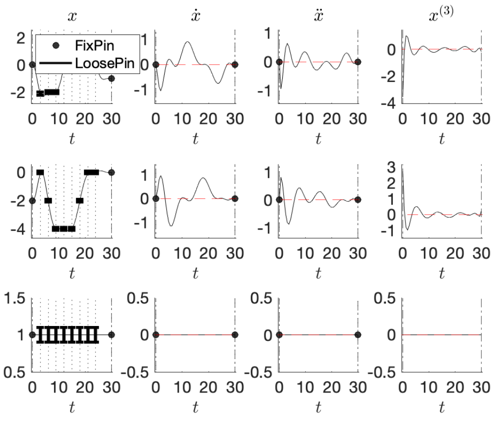

Figure 48: Timestamps

As the opti-track setup in ASTA, could not identify the reflectors correctly, after multiple tries in the last days we were not able to complete the practical part of this exercise.