Exercises for 34757

Unmanned Autonomous Systems 34757

Department of Electrical and Photonics Engineering • Technical University of Denmark

PURPOSE:

The purpose of this project assignment is to practice the topics learned in the Unmanned Autonomous System course 34757 on a realistic unmanned aerial vehicle application.

INTRODUCTION:

In this course we will have a combination of lectures, exercises, self-study material, and we aim at a final demonstration of your skills and competences gained, using a real Crazyflie quadrotor UAV performing a drone race against other groups. The UAV is a nano quadcopter that weighs only 27g. This has many advantages, including that it is ideal for flying inside a lab, office, or your living room without damaging or harming. Even though the propellers spin at high RPMs, they are soft and the torque in the motors is very low when compared to a brushless motor. In any case, when you work with the hardware, we recommend you to use safety precautions as described in the practical exercise document.
Even if in this course you are expected to complete all the predefined theoretical and practical exercises and prepare a small demonstration, we do not want you to spend too much time on reporting, rather we prefer you use your extra time for self learning. Your report should include (at least) your solution to part 7, and possibly a documentation of your final demo/race.

ABOUT THIS MATERIAL:

This paper describes a project assignment tasks that consists of 7 parts, denoted 'Part 1' to 'Part 7', available in the material on DTU Learn. The parts are a mixture of basic problems on the different aspects constituting the foundations of the course (these problems will help you refreshing previously studied topics), and more advanced exercises that focus on reaching the learning objectives of the course. Furthermore, Part 6 and Part 7 will give you hands on experience on the course topics through simulation and real experiments.
We recommend you to complete the assignment using Matlab and Simulink.

CONTENTS:

Part 1 - Rotations focuses on refreshing your knowledge on 3D rotations and minimal representation. You will solve problems related to angle representation using several approaches, from Euler angles to quaternions.
Part 2 - Modeling focuses on refreshing your knowledge on kinematics and dynamics modeling. You will end up with a dynamic model of a quadrotor UAV.
Part 3 - Control focuses on refreshing your knowledge on control theory. You will control a non-linear dynamic system such as a quadrotor UAV.
Part 4 - Path Planning focuses on testing and comparing several path planning algorithms that you will need to find the 3D path of a UAV from a starting point to a target one.
Part 5 - Trajectory Planning focuses on defining time-dependent trajectories that allow the UAV to generate commands leading to the goal.
Part 6 - Simulation Exercises simulations will give you hands on experience and an excellent

starting point to test your knowledge and implementation, before you use the real system. *Part 7 - Experimental Exercises* this is the hands-on part of the course.

A GROUP WORK APPROACH:

We believe that group work is beneficial for your personal learning as you are required to plan, communicate, monitor and evaluate your work together with other students who might have different views, or even a different background. Group will consist of 5 team members, no exceptions.
When doing group work, it is necessary to kick in the work by sharing personal motivations and expectation from the self and your colleagues. Make sure that you discuss how are you going to work together, identify potential risks and think of mitigation well in advance. We encourage you to write down how much (in percentage) everyone has contributed to solving the exercises and making the report. This value will be used to differentiate the grades if needed, so please make sure that everyone in the group agrees with the differentiation (if any) and let the whole group sign the report.

SOLVING AND REPORTING THE PROJECT ASSIGNMENT:

There are many exercises, which will help you develoing the knowledge needed to excel in this course. However, we do not expect you to report on all of them, as this may require a lot of your time! To ensure that you gained the competences expected by this course, while reducing the time that you spend on reporting, we recommend preparing a detailed report of part 7, where you show the combination of your gained hands on competences and the application of the theoretical approach leading to your solution.
We will have a demo day in the last day of the course and you are expected to demonstrate all the exercises. However, main focus will be on a drone race, where every group compete on time.

The report is assessed as a whole based on the quality of the explanatory text and the correctness of the answers. The last page of the report should be signed by the participant(s). Remember your 'study registration number'. The project assignment needs to be carried out by team of two or three persons, the individual contribution to the project work must be clearly indicated contributions to the answer for each of the problems presented in the 5 parts of the project assignment report. Specify the work contribution each problem by the percentage scale, e.g. example Elisabeth 30% / Peter 30% / Jack 40%.

DEADLINES:

The project report and demonstration video must be submitted on DTU Learn by each of the group members, NO LATER THAN the midnight of July 4th, current year.

PART 1: Rotations

Exercise 1.1

Find the rotation matrix corresponding to the set of Euler angles ZXZ. Describe the procedure used to find the solution.

Exercise 1.2

Discuss the inverse solution for the Euler angles ZYZ in case $s_\theta = 0$.

Exercise 1.3

Discuss the inverse solution for Roll-Pitch-Yaw angles in the case $c_\theta = 0$.

Exercise 1.4

Given a pair of unit vectors v and w (v =from and w =to, find the minimal rotation that brings v in w.
(SUGGESTION: use your knowledge on axis-angle representation)

Exercise 1.5

Answer the following questions with explanations:

- What is the quaternion q_1 that represents the rotation of 180 degree about the x-axis?
- What is the quaternion q_2 that represents the rotation of 180 degree about the z-axis?
- What rotation is represented by composite quaternion $q = q_1 q_2$? Answer by specifying its rotation angle and axis.
-

Exercise 1.6

Compare the number of additions and multiplications needed to perform the following operations:

- Compose two rotation matrices.
- Compose two quaternions.
- Apply a rotation matrix to a vector.
- Apply a quaternion to a vector (as in Exercise 4).

Count a subtraction as an addition, and a division as a multiplication.

PART 2: Modeling

Exercise 2.1

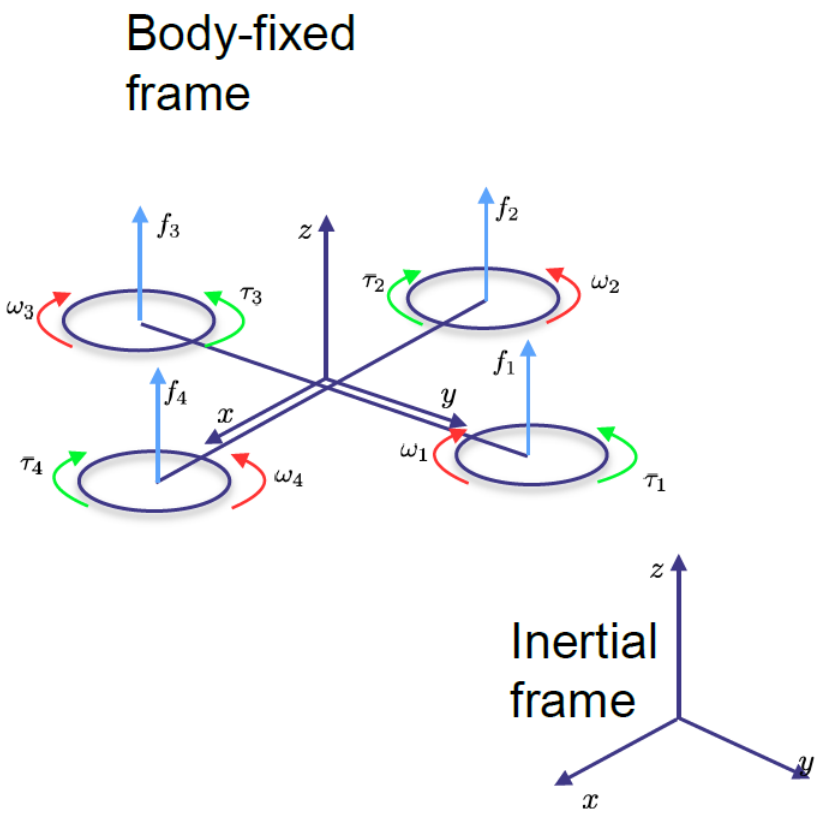

Figure 1: The free-body diagram of the quadrotor UAV.

Given the rigid body diagram as in Figure 1, derive the dynamic equation of the drone, given:

- m=0.5 Kg is the drone mass in the center of gravity of the drone
- L=0.225 m is the length of each arm of the quadrotor frame, i.e. the distance of the motors from the CoG.
- k=0.01 $N\frac{s^2}{rad^2}$ the overall aerodynamic coefficient necessary to compute the lift of the propellers as $f_i = b\omega_i$.
- b=0.001 $Nm\frac{s^2}{rad^2}$ is the overall aerodynamic coefficient necessary to compute the drag torque of the propellers as $\tau_i = b\omega_i$.
- $D = diag([D_x,\ D_y,\ D_z]^T)$ is matrix representing the the drag on the UAV moving in air with velocity $\dot{\mathbf{p}}$, being $D_x = D_y = D_z = 0.01\ Ns/m$
- $Ixx = Iyy = 3e^{-6}\ Nms^2/rad$ are the moment of inertia on the principal axis x and y

- $Izz = 1e^{-5}\ Nms^2/rad$ is the moment of inertia on the principal axis z
- The matrix of moment of inertial of the UAV is $I = diag([I_x x,\ I_y y,\ I_z z]^T)$
- gravity acts along the z-axis of the inertial frame of reference, and the gravitational acceleration $\mathbf{g}$ is given by $\mathbf{g} = [0,\ 0,\ -9.81]^T\ m/s^2$

Please consider the following notation:

- $\mathbf{p}$ [m] is the position of the CoG of the UAV w.r.t. a fixed inertial frame of reference. Let's denote $\mathbf{p} = [x,\ y,\ z]^T$
- Θ [rad] is the representation of the rotation of the body-fixed frame w.r.t. the fixed inertial frame of reference, according to the roll-pitch-yaw angular representation. Let's denote $\Theta = [\phi, \theta, \psi]^T$
- ω [rad/s] is the angular velocity of the body-fixed frame w.r.t. the inertial frame.
- Ω is the vector of the angular speed of the four propellers, $\Omega = [\Omega_1,\ \Omega_2,\ \Omega_3,\ \Omega_4]^T$

1. Define the rotation matrix representing the orientation of the body-fixed frame w.r.t. the inertial frame.

2. Define the relation between the angular velocity $\dot{\Theta}$ and the rotational velocity of the body-fixed frame ω.

3. Write the linear and angular dynamic equation of the drone in compact form, and clearly show each component of the equation explicitly

4. Make a MATLAB/Simulink model of the drone, given initial conditions $\mathbf{p}(0) = [0,\ 0,\ 0]^T$, $\dot{\mathbf{p}}(0) = [0,\ 0,\ 0]^T$, $\Theta(0) = [0, 0, 0]^T$ and $\dot{\Theta}(0) = [0, 0, 0]^T$, and report the following:

- Make a plot of $\mathbf{p}$ and Θ, given $\Omega = [0,\ 0,\ 0,\ 0]^T$ and explain the result
- Make a plot of $\mathbf{p}$ and Θ, given $\Omega = [10000,\ 0,\ 10000,\ 0]^T$ and explain the result
- Make a plot of $\mathbf{p}$ and Θ, given $\Omega = [0,\ 10000,\ 0,\ 10000]^T$ and explain the result

Exercise 2.2

Given the same UAV model as in Exercise 2.1, write the equations using the unit quaternions to represent the rotations and answer the following questions:
1. Define the rotation matrix representing the orientation of the body-fixed frame w.r.t. the inertial frame.

2. Define the relation between the angular velocity Θ and the rotational velocity of the body-fixed frame ω.

3. Write the linear and angular dynamic equation of the drone in compact form, and clearly

show each component of the equation explicitly

4. Make a MATLAB/Simulink model of the drone, given initial conditions $\mathbf{p}(0) = [0,\ 0,\ 0]^T$, $\dot{\mathbf{p}}(0) = [0,\ 0,\ 0]^T$, $\Theta(0) = [0, 0, 0]^T$ and $\dot{\Theta}(0) = [0, 0, 0]^T$, and report the following:

- Make a plot of $\mathbf{p}$ and Θ, given $\Omega = [0,\ 0,\ 0,\ 0]^T$ and explain the result
- Make a plot of $\mathbf{p}$ and Θ, given $\Omega = [10000,\ 0,\ 10000,\ 0]^T$ and explain the result
- Make a plot of $\mathbf{p}$ and Θ, given $\Omega = [0,\ 10000,\ 0,\ 10000]^T$ and explain the result

Exercise 2.3

Using the model in Exercise 2.1 (or 2.2), linearize the dynamic model of th eUAV in hovering conditions. Compare the linearized model with the non-linear one under the same input conditions as in previous exercises (2.1 and 2.2 if solved):

- Make a plot of $\mathbf{p}$ and Θ, given $\Omega = [0,\ 0,\ 0,\ 0]^T$ and explain the result
- Make a plot of $\mathbf{p}$ and Θ, given $\Omega = [10000,\ 0,\ 10000,\ 0]^T$ and explain the result
- Make a plot of $\mathbf{p}$ and Θ, given $\Omega = [0,\ 10000,\ 0,\ 10000]^T$ and explain the result

PART 3: Control

Exercise 3.1

Using the non-linear model of Exercise 2.1 (or 2.2), close a feedback control loop to control the attitude, to allow setting roll, pitch, yaw and altitude references. Choose the gains of the PID by tuning on the linearized system, by minimizing the settling time while ensuring that the controlled system is critically damped (no overshoot), and plot the step response for a step of:

- 1. $\Theta^\star = [10,\ 0,\ 0]^T\ [deg]$, $z^\star = 0[m]$
- 2. $\Theta^\star = [0,\ 10,\ 0]^T\ [deg]$, $z^\star = 0[m]$
- 3. $\Theta^\star = [0,\ 0,\ 10]^T\ [deg]$, $z^\star = 0[m]$
- 4. $\Theta^\star = [0,\ 0,\ 0]^T\ [deg]$, $z^\star = 1[m]$.

Show your results and explain them.

Exercise 3.2

Apply the controller of Exercise 3.1 to the non-linear model and compare the step responses for the controller applied to the linearized model and to the non-linear one. Show the comparison and explain the differences for each of the references setpoints of Exercise 3.1.

Exercise 3.3

Close a position control loop for the x and y components. Tune the gains using proper considerations and showing an appropriate methodology. Explain the method used and discuss your results.

PART 4: Path planning

Exercises for the path planning part. It is recommended to print the graphs used for these exercises and draw on them to run through the algorithms by hand. These drawings can be handed in as solutions. The graphs can be found at the end of part 4.

Exercise 4.1: Depth-first search (DFS) algorithm

Figure 2 shows a map of cities connected with roads. Each circle on the map is a node which represents a city, and the connections between them are edges representing roads. You need to get from city s0 to city s23. Use the DFS algorithm to find a route from city s0 to city s23.

You can read more about the DFS algorithm here `https://en.wikipedia.org/wiki/Depth-first_search` and in the book "Introduction to Algorithms" by Cormen, Leierson, Rivest and Stein. 3rd edition.", page 603-612, which can be found here: `https://edutechlearners.com/download/Introduction_to_algorithms-3rd%20Edition.pdf`.

1:
Start by stacking the lowest of the s numbers when it is possible to stack multiple. Start with s0 as the first frontier in the stack.

Write down the steps taken by the algorithm. In which order the nodes are expanded and what frontiers are queued at that point.

When the goal is reached draw the found route to the goal or write down the route from s0 to the goal, draw the layer numbers on each node or specify in which layer each node belong. Specify the total number of expanded nodes and the total number of frontiers generated. Show the path, as well as, the length of the path.

2:
Try to start by stacking the highest of the s numbers when it is possible to stack multiple. Start with s0 as the first frontier in the stack.

Write down the steps taken by the algorithm. In which order the nodes are expanded and what frontiers are queued at that point.

When the goal is reached draw the found route to the goal or write down the route from s0 to the goal, draw the layer numbers on each node or specify in which layer each node belong. Specify the total number of expanded nodes and the total number of frontiers generated. Show the path, as well as, the length of the path.

3:
Notice any difference in the paths obtained using the two different DFS methods? Why is one route longer than the other?

Exercise 4.2: Breadth-first search (BFS) algorithm

Figure 2 once more shows a map of cities connected with roads. This time the roads are toll roads. Each of the roads have the same fee. Use the BFS algorithm to find a route city s0 to city s23, where you have to pay the least amount of money.

You can read more about the DFS algorithm here `https://en.wikipedia.org/wiki/Breadth-first_search` and in the book "Introduction to Algorithms" by Cormen, Leierson, Rivest and Stein. 3rd edition.", page 594-603, which can be found here: `https://edutechlearners.com/download/Introduction_to_algorithms-3rd%20Edition.pdf`.

1:
Start by queuing the lowest of the s numbers when it is possible to queue multiple. Start with s0 as the first frontier in the queue.

Write down the steps taken by the algorithm. In which order the nodes are expanded and what frontiers are queued at that point.

When the goal is reached draw the found route to the goal or write down the route from s0 to the goal, draw the layer numbers on each node or specify in which layer each node belong. Specify the total number of expanded nodes and the total number of frontiers generated. Show the path, as well as, the length of the path.

2:
Comment on the advantages and disadvantages of using BFS vs DFS

Which type of algorithm is preferred for robots?

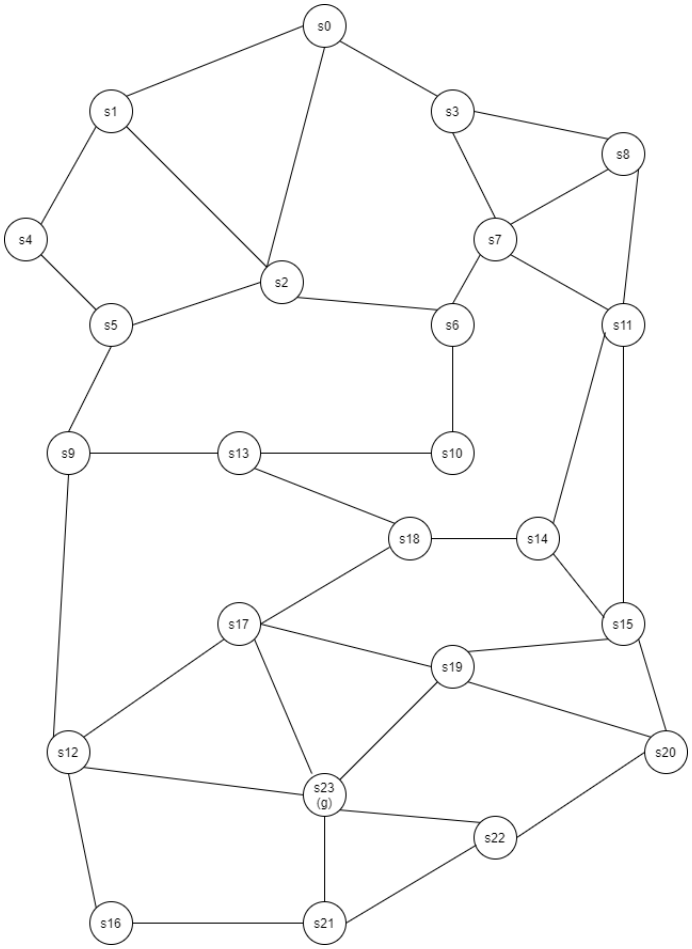

Figure 2: The graph used for Exercise 4.1 and 4.2. A larger picture can be found in ... (Appendix or end of Document?)

Exercise 4.3: Dijkstra's algorithm

Figure 3 shows the same city as Figure 2, but now the length of the roads have been added to the map. You once more want to get from city s0 to city s23, but this time you are in a rush and want to get to city s23 as fast as possible. You are allowed to drive the same speed on all roads. Use Dijkstra's algorithm to find the fastest route from city s0 to city s23.

You can read more about the Dijkstra algorithm here `https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm` and in the book "Introduction to Algorithms" by Cormen, Leierson, Rivest and Stein. 3rd edition.", page 658-664, which can be found here: `https://edutechlearners.com/download/Introduction_to_algorithms-3rd%20Edition.pdf`.

Write down the steps taken by the algorithm. In which order the nodes are expanded and what frontiers are generated at which point with what distance to it. Specify if a frontier is updated and write the new distance.

When the goal is reached draw the found route to the goal or write down the route from s0

to the goal, draw the distance to each node from s0 or specify the distance to each node from s0. Specify the total number of expanded nodes and the total number of frontiers generated. Show the path, as well as, the length of the path.

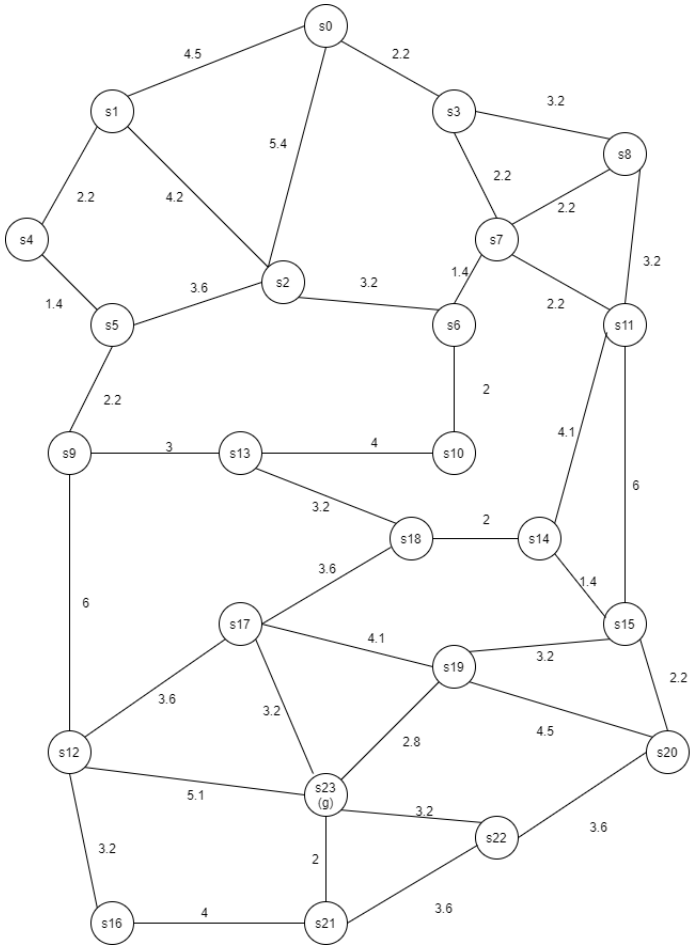

Figure 3: The graph used for Exercise 4.3. A larger picture can be found in ... (Appendix or end of Document?)

Exercise 4.4: Greedy best-first search algorithm

Figure 4 now shows the map of the cities with the distance from the cities to the goal city added in red. Use the Greedy best-first search algorithm to find a route from city s0 to city s23,

You can read more about the Greedy best-first search algorithm here `https://www.javatpoint.com/ai-informed-search-algorithms` and in the book "Artificial Intelligence Modern Approach" by S. Russell and P. Norvig.", page 92-96, which can be found here: `https://www.cin.ufpe.br/~tfl2/artificial-intelligence-modern-approach.9780131038059.25368.pdf`.

Write down the steps taken by the algorithm. In which order the nodes are expanded and

what frontiers are generated at which point with what distance to it. Specify if a frontier is updated and write the new distance.

When the goal is reached draw the found route to the goal or write down the route from s0 to the goal, draw the distance to each node from s0 or specify the distance to each node from s0. Specify the total number of expanded nodes and the total number of frontiers generated. Show the path, as well as, the length of the path.

Exercise 4.5: A* search algorithm

Now use both the direct distance and the length of the roads to get the shortest route from city s0 to city s23. Use the A* algorithm to obtain the shortest route.

You can read more about the A* algorithm here `https://en.wikipedia.org/wiki/A*_search_algorithm` and here `https://www.javatpoint.com/ai-informed-search-algorithms` and in the book "Artificial Intelligence Modern Approach" by S. Russell and P. Norvig.", page 96-101, which can be found here: `https://www.cin.ufpe.br/~tfl2/artificial-intelligence-modern-9780131038059.25368.pdf`..

Write down the steps taken by the algorithm. In which order the nodes are expanded and what frontiers are generated at which point with what distance to it. Specify if a frontier is updated and write the new distance.

When the goal is reached draw the found route to the goal or write down the route from s0 to the goal, draw the distance to each node from s0 or specify the distance to each node from s0. Specify the total number of expanded nodes and the total number of frontiers generated. Show the path, as well as, the length of the path.

This route is the same as the one found by Dijkstra's algorithm. What is the advantage of using A*?

Exercise 4.6: Advantages and Disadvantages of Greedy best-first search and A* search algorithms

1:
What are the advantages and disadvantages of Greedy best-first search algorithm?

2:
What are the advantages and disadvantages of A* search algorithm?

3:
Which algorithm is best for aerial robots and why?

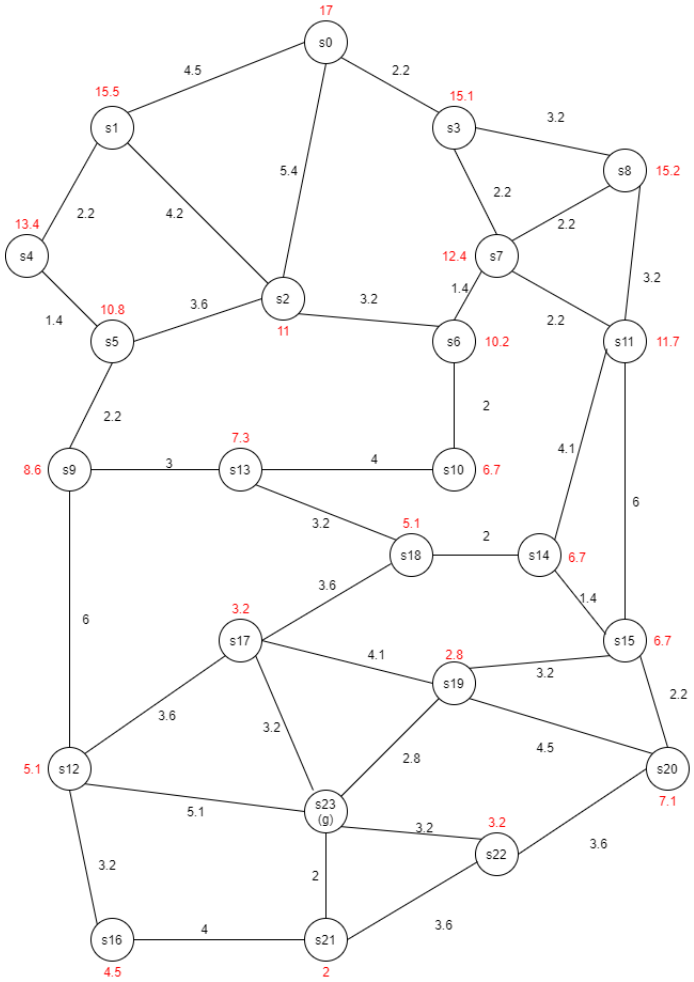

Figure 4: The graph used for Exercise 4.4 and 4.5. A larger picture can be found in ... (Appendix or end of Document?)

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

A search algorithm can be used to navigate through a grid based map. Such a map have been implemented in the file called *map_script.m*.

Open the file called *map_script.m*. The first part of the the file defines the map, including start and end position. The middle part runs the function called *greedy_2d*, and the last part draws the map and found route.

Run the Matlab script and watch the algorithm find the solution.

The *greedy_2d* function is an implementation of the Greedy best-first search algorithm in Matlab. Open the *greedy_2d.m* file and read through the function. Try to understand the implementation.

In order for a drone to navigate in 3D, the grid needs to be in 3D, and the algorithm

therefore needs to be upgraded to work in 3D.

Open the file called *map_script_3d.m*. This file is the same as *map_script.m* but it is in 3D.

Upgrade the greedy_2d function from 2D to 3D, and use it to solve the maze.

Hint: Before the nodes expanded in a square pattern, now it needs to expand in a cubic pattern.

Exercise 4.8: Modify the implementation of the Greedy best-first search algorithm into a A* search algorithm

The Greedy best-first search algorithm does not always find the optimal solution. To do this an A* algorithm is needed.

Modify the implementation of the Greedy best-first search algorithm to be a A* search algorithm

What changes have to be made?

Did it affect the running time of the algorithm?

Hint: 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.

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

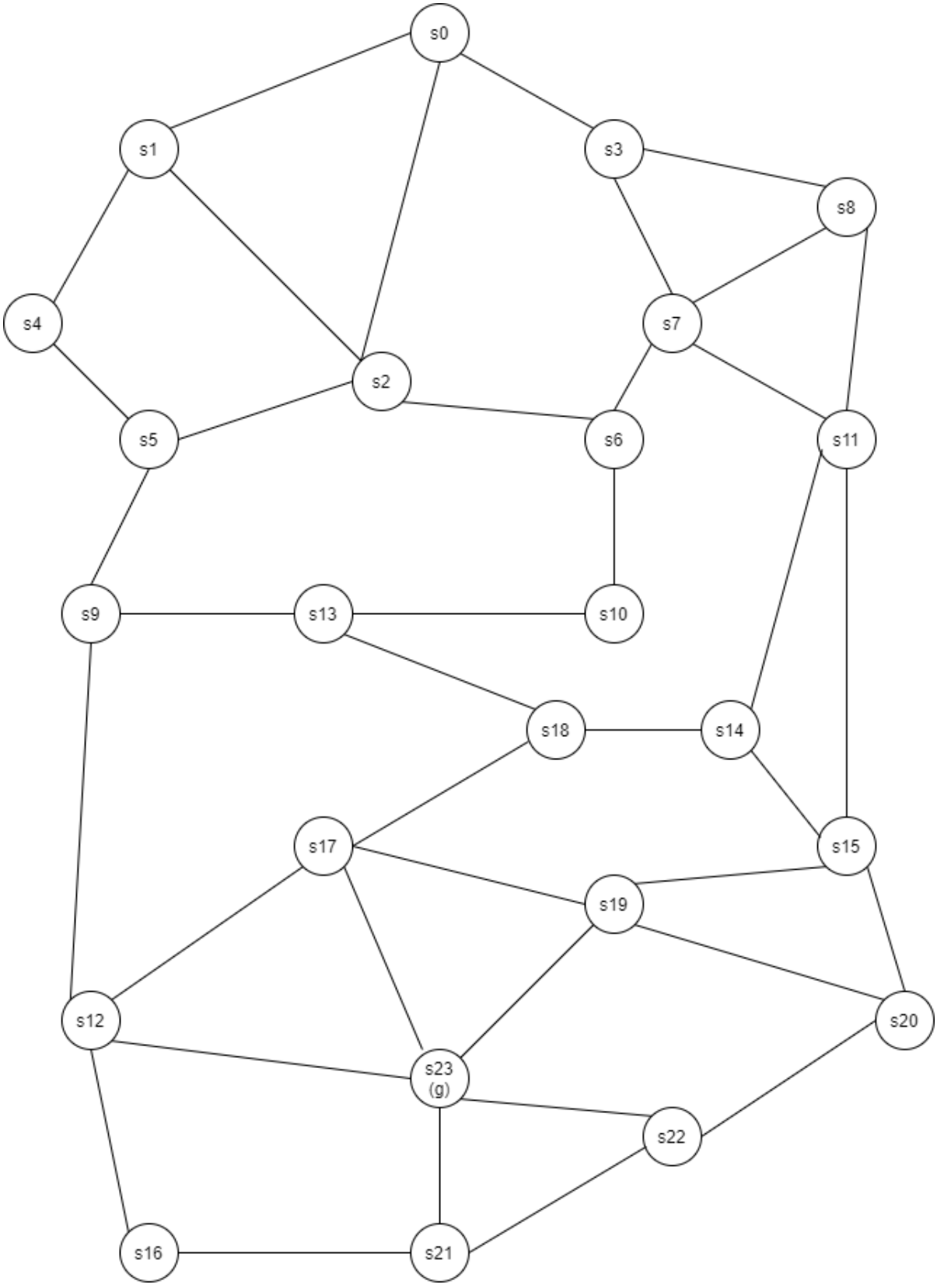

Figure 5: The graph used for Exercise 4.1 and 4.2

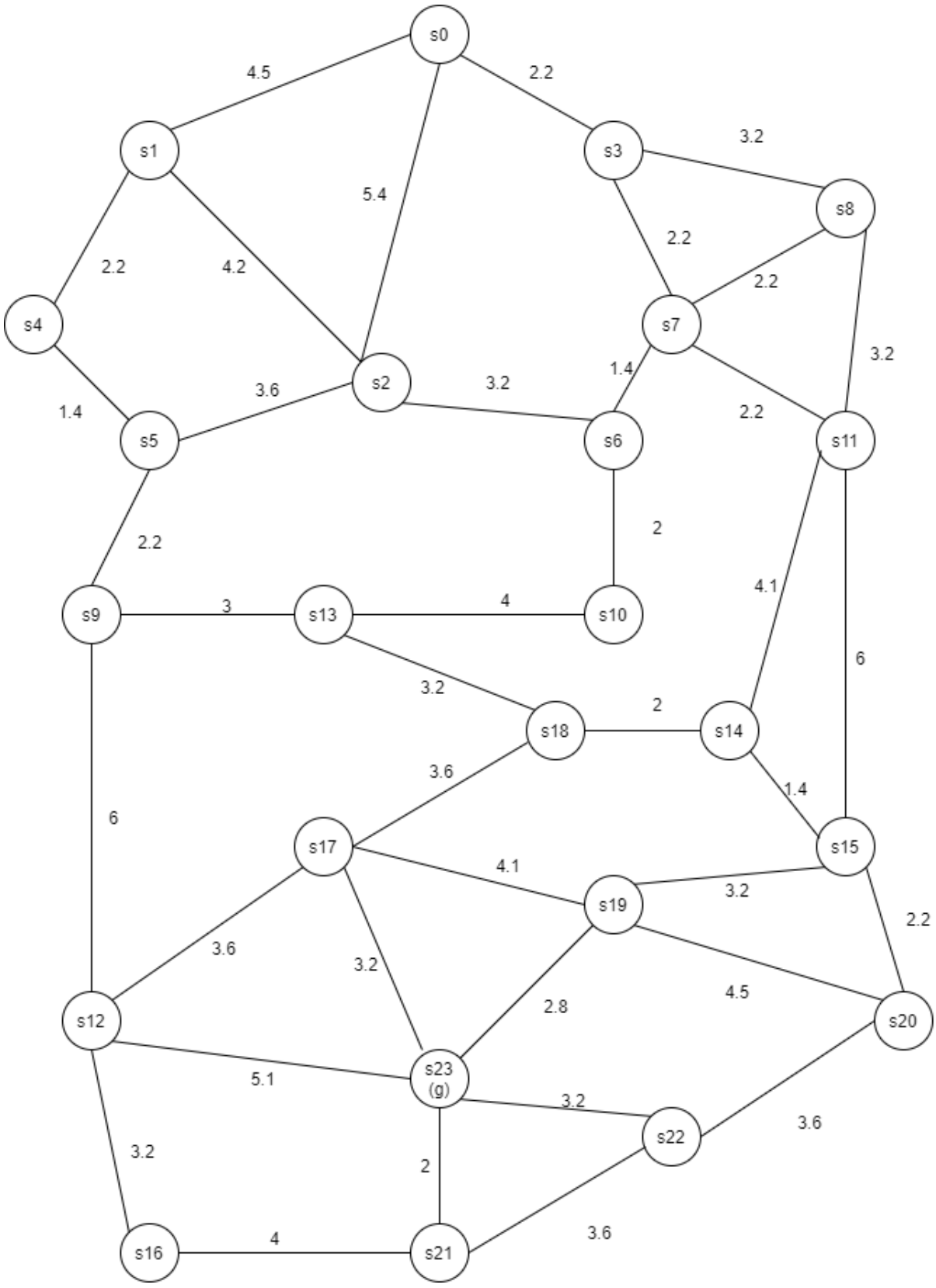

Figure 6: The graph used for Exercise 4.3

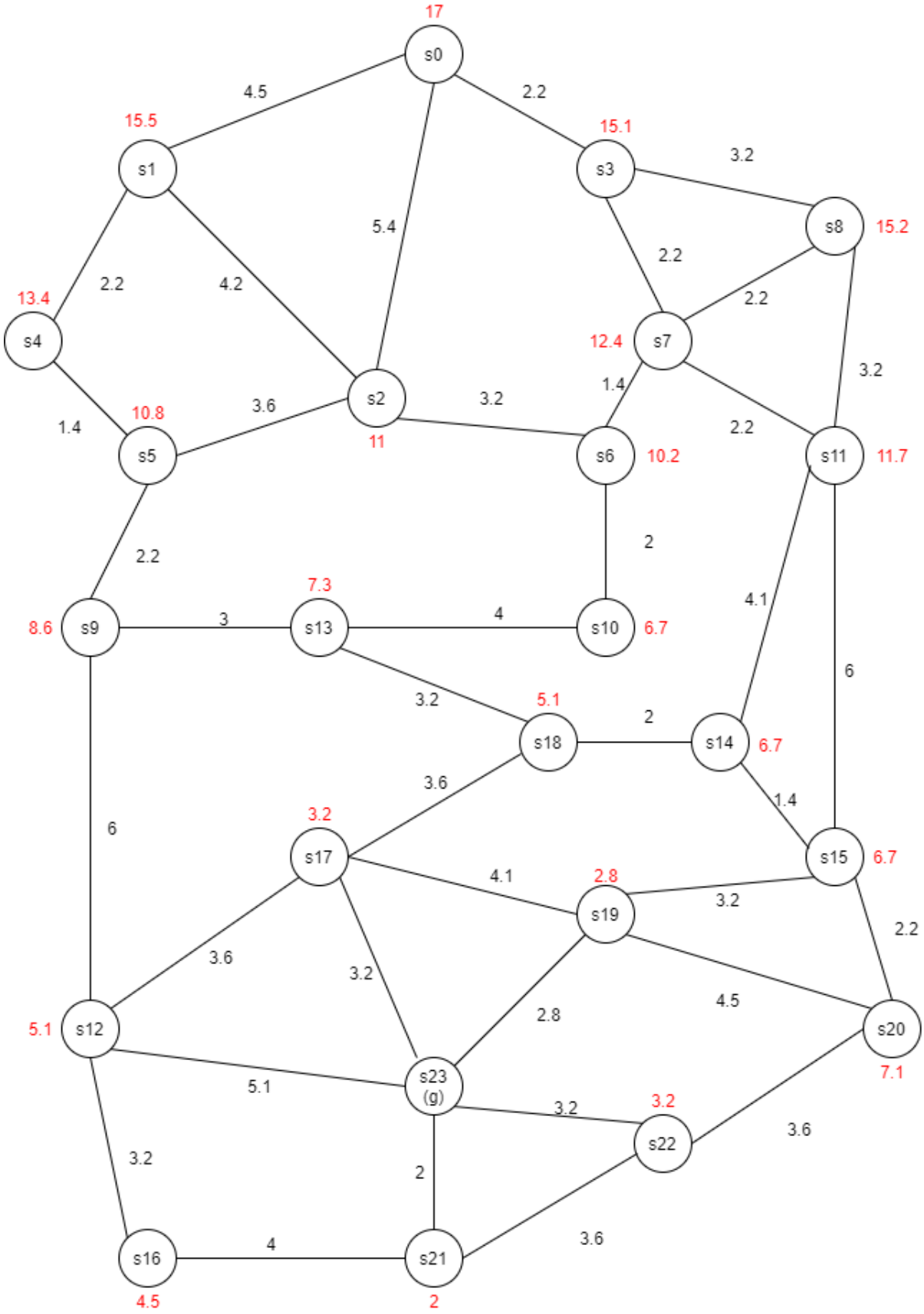

Figure 7: The graph used for Exercise 4.4 and 4.5

PART 5: Trajectory generation and control

Exercise 5.1: quintic splines

Consider the following 1-D quintic spline:

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

Find the constants a_i, $i = 0, 1, 2, 3, 4, 5$, to respect the constraints $x(0) = 0,\ \dot{x}(0) = 0,\ \ddot{x}(0) = 0,\ x(1) = 1,\ \dot{x}(1) = 0$, and $\ddot{x}(1) = 0$.

Exercise 5.2: B-splines

Given the *Cox-de Boor recusion formula*:

$$N_{i,0}(t) = \begin{cases} 1 & \text{if } t_i \leq t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$
$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t),$$

Compute the two first order basis functions $N_{0,1}(t)$ and $N_{1,1}(t)$ with $t_i = i$ and combine them linearly to interpolate between the two points $(1, 2)$ and $(2, 3)$.

Exercise 5.3: computation of a DCM

In Mellinger and Kumar (2011),

Mellinger D, Kumar V. Minimum snap trajectory generation and control for quadrotors. In2011 IEEE International Conference on Robotics and Automation 2011 May 9 (pp. 2520-2525). IEEE.

a Direction Cosine Matrix (DCM) is found knowing the upward direction of the drone z_B and the yaw angle ψ:

$$R_B = \begin{bmatrix} x_B & y_B & z_B \end{bmatrix},\ y_B = \frac{z_B \times x_C}{||z_B \times x_C||},\ x_C = \begin{bmatrix} \cos\psi \\ \sin\psi \\ 0 \end{bmatrix},\ x_B = y_B \times z_B.$$

However, this is based on the Z-X-Y Euler angles convention, and the convention we are using in this course is Z-Y-X, i.e. $R_B = R_z(\psi)R_y(\theta)R_x(\phi)$ with the roll, pitch and yaw angles ϕ, θ and ψ (respectively), and

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix},\ R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \text{ and } R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Compute R_B based on z_B and ψ in the Z-Y-X Euler angles convention.

PART 6: Simulation

This section is based on a MATLAB/Simulink environment available on GitLab:

`https://gitlab.gbar.dtu.dk/Courses/34757.git`

Follow the installation instructions to get started. Note that the main model file is `uas_main.slx` and that it relies on variables in the base workspace that can be loaded by running the script `uas_parameters.m` first.

Exercise 6.1: Navigating a 2D maze

When the simulation environment is up-and-running, you should be able to see a drone flying through a maze from one end to the other. The waypoints are containted in the variable `route`, and the corresponding scaling and offset are 1 m and 0 m (respectively).

Task: Recompute `route` using the code from exercise 4.1 to navigate from points (0, 0, 1) to (3, 5, 1).
Report: Provide a X-Y plot of the achieved trajectory.

Exercise 6.2: Re-implementing the position controller

In the `quadcopter` block, there is a position controller cascaded on an attitude controller. These two controllers are based on `papers/lee2011control.pdf`. The inputs of the position controller are setpoint positions, velocities and accelerations, and the outputs are roll, pitch, yaw and thrust commands.

Task: Replace the position controller with your own implementation. You can leave out the setpoint velocities and accelerations.
Report: Explain your approach and show the responses corresponding to step inputs of magnitude 1 m, 3 m, and 9 m in the x direction.

Exercise 6.3: Re-implementing the attitude controller

The inputs of the attitude controller are setpoint roll, pitch, yaw and thrust, and the outputs are force and torques commands that are translated into motor velocities by the mixer.

Task: Replace the attitude controller with your own implementation.
Report: Explain your approach and show the responses corresponding to step inputs of magnitude 5°, 15°, and 30° m in the forward pitch direction.

Exercise 6.4: Aggressively navigating a 2D maze

In the purple area, there are some blocks that output positions, velocities and accelerations using the variables generated by the script `uas_trajectory.m`. This script is based on

`papers/mellinger2011minimum.pdf` and the API's documentation can be found here:

`https://se.mathworks.com/matlabcentral/fileexchange/74573-traj_gen-matlab`

Task: Connect the blocks from the purple area to the quadcopter, comment the unused blocks, replace the position controller with the default one in case you didn't implement velocity and acceleration setpoints, and fix `uas_trajectory.m` to generate a trajectory that allows the drone to fly from (0,0,1) to (9,9,1) in less than 5 s without touching any of the walls.
Report: describe your solution and show the plots of the time profile as well as the x-y plot of the trajectory.

PART 7: Demonstration

This part is to be performed in the μASTA arenas. The following three exercises can be completed by adapting the Simulink model from part 6 or using high-level python scripts. To do so, uncomment and wire the ROS2 communication blocks (red area) plus Arming crazyflie block, and comment the blocks related to the simulated drone.
Before trying anything on the crazyflie, you can test your program in simulation. Refer to: Crazyswarm2 Simulation Guide. For flying the crazyflie, make sure that your laptop/system is connected to the OptiTrack wifi, not the OptiTrack 5G.
Make sure that the CF you are using carries reflective markers according to fig. 8. You have to design your own carrier for reflective markers. It is really important that carrier is mounted as indicated in the figure and the reflective markers should not be outside of the propeller guards.

Figure 8: An example figure illustrating mounting of the reflective markers, red marking indicates the mounting point for markers carrier

Exercise 7.1: Porting the position controller to the real system

Complete this exercise in the μASTA close from the pool using the Optitrack 3D motion tracking system.

This exercise can be completed either with your own position controller or with the default one. The real drone's attitude controller accepts roll and pitch commands as in the simulation (these commands just need to be converted from radiant to degrees). However, it tracks yaw rate commands instead of yaw, and thrust commands are expressed as PWM commands, which are integer values ranging from 0 to 60'000. These values essentially correspond to the average voltage applied to the motors.
Task: Identify the relationship between the thrust commands calculated by the position controller and the PWM commands accepted by the real drone's attitude controller. Fitting a first order polynomial (i.e., $y = ax + b$) on a data-set consisting of PWM commands and vertical accelerations should do the trick. You can gather this dataset by flying the drone manually in attitude mode using a joystick or a gamepad. Then, implement this relationship in the Simulink model and try to get a CF to:

1. hover stably on point (1, 1, 1) for 10 s with a maximum error of 10 cm, and
2. move from setpoint (1, 1, 1) to (2, 1, 1) while ensuring that the overshoot is lower than 30 cm.

Report: For each experiment, show:

1. a x-y plot,
2. a plot of x, y, and z versus time (with references), and
3. a plot of the x, y, and z error versus time.

Exercise 7.2: Navigating a 3D maze

Complete this exercise in the μ ASTA, where a custom maze has been created. The goal of this exercise is to be able to navigate the 3D maze. Use either the greedy best-first or A* search algorithm to obtain a path which the drone can follow. The path must be scaled to match the real maze. An offset is also needed. How the scaling and offset is applied can be seen in the bottom of the `map_3d.m` script. Adjust the scalings and offsets as needed. The code provided in exercise 4 can be used, but it must be upgraded from 2D to 3D if this is not already done. Use way-points to control the drone.

The starting position is marked E, F, G or H and it can be seen on the floor of the maze. The goal is to reach A, B, C or D, also marked on the floor of the maze. A map of the maze is contained in the file called `wall.txt`.

On the demonstration day a new maze will be made. The map of the demonstration day maze will be provided. The format of the map will be the same as with the `wall.txt` file.

Exercise 7.3: Aggressively navigating through hoops

This exercise can be completed either with your own position controller (if it accepts velocity and acceleration setpoints) or with the default one. Also, it requires the thrust-PWM

relationship from exercise 7.1.

Task: Modify the code in `uas_trajectory.m` to navigate through the hoops standing in the μASTA close to the pool.
Report: Show the x-y plot of the reference trajectory superimposed with the achieved one.