Exercises for 31390, June 2020

Unmanned Autonomous Systems 31390

Department of Electrical Engineering • Technical University of Denmark

04 June 2020

PURPOSE:

The purpose of this project assignment is to practice the topics learned in the Unmanned Autonomous System course 31390 on a realistic unmanned aerial vehicle application.

INTRODUCTION:

NOTE: this introduction refers to the original course, before the Covid-19 lock down of DTU. In 2020, you will work on a simulator instead. The intro is kept in this file, to give you an idea of how the course would have looked in regular condition

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 simulated Crazyflie-2.1 quadrotor UAV. 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 encourage you to use any extra time to perform self-study activities and additional competence development using the available systems and infrastructures. In the course material, beside the lecture slides and references to the lectures, we will provide additional study material to allow you deepening the main aspects of the course. If you complete the exercises on time, you can than take advantage of the available time to learn more and experiment more.

ABOUT THIS MATERIAL:

This paper describes a project assignment tasks that consists of 5 parts, denoted ‘Part 1’ to ‘Part 5’, 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.

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 Exercise 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. We recommend a group to have 3 members. Groups will be defined on Day 1. If you are not present on that day, you will be assigned to a group by the teachers and the list of groups will be made available on the course page. 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 three type of exercises, propaedeutic, mandatory and advanced.
Mandatory exercises are indicated with the red star symbol $\star$.
Advanced exercises are indicated with the red double star symbol $\star\star$
All the other exercises are propaedeutic.
Solving the mandatory problems in this note carries out the project assignment. The answers shall be supported by a sufficient number of intermediate calculations and explanatory text, so that the principles and method used are clear. Plots should be numbered consecutive $A0, A1, ...A12$.

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 (2 copies) must be submitted to Matteo Fumagalli (Building 326, ground floor), NO LATER THAN 5pm, 10.July.2020. Please leave the project reports in the mailbox.

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⋆

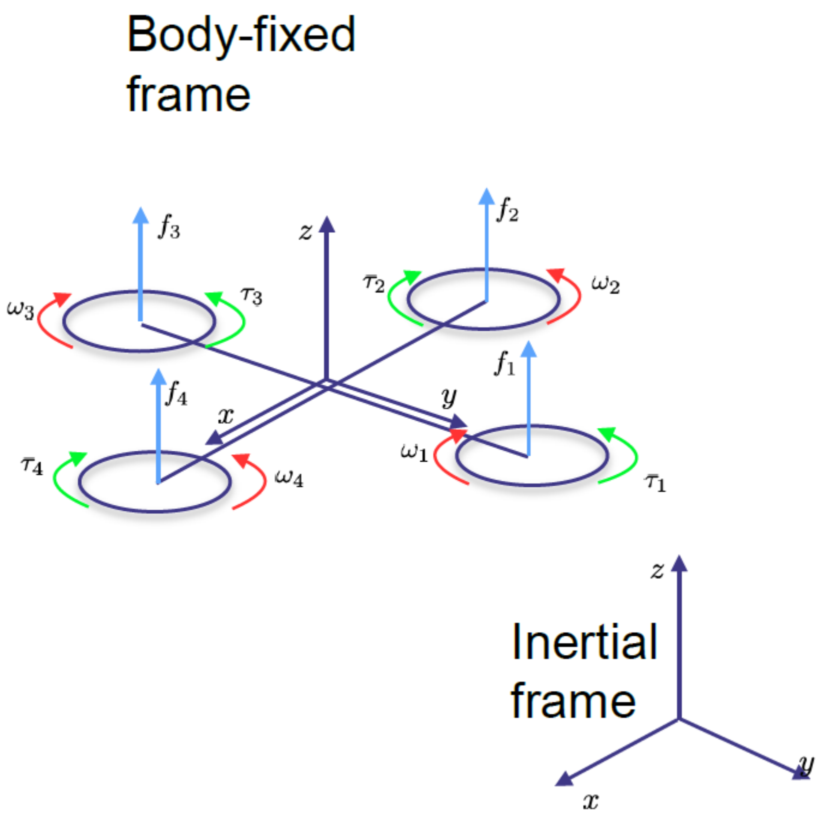

Figure 1: The free-body diagram of the quadrotor UAV.

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$\star$

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$\star$

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$\star$

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$\star$

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 ... (Appendix or end of Document?)

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 the second route so long compared to the first?

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?

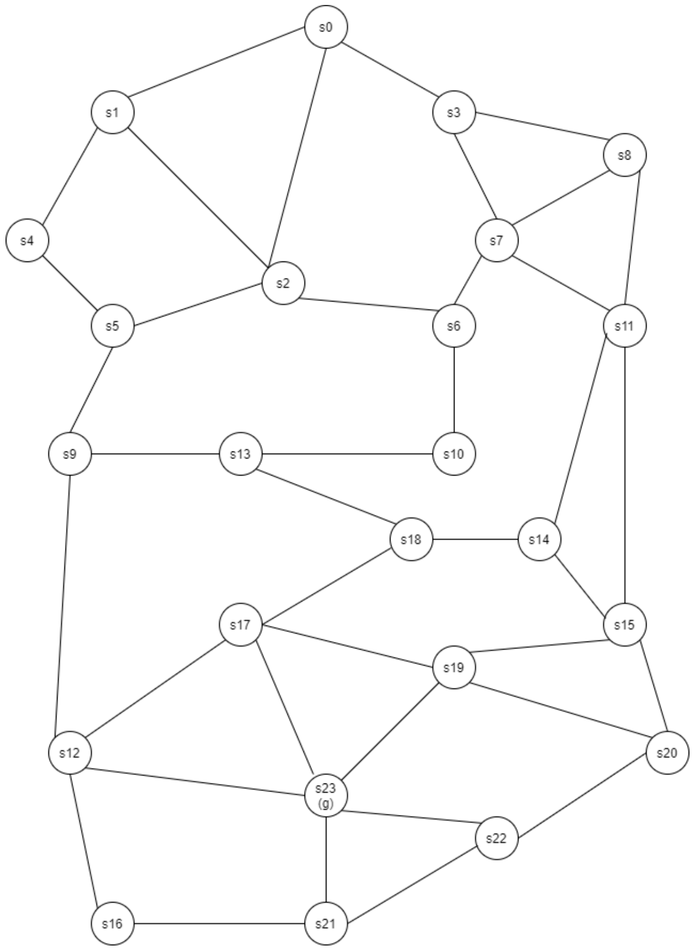

Figure 2: The graph used for Exercise 4.1 and 4.2. A larger picture can be found in ... (Appendix or end of Document?)

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.

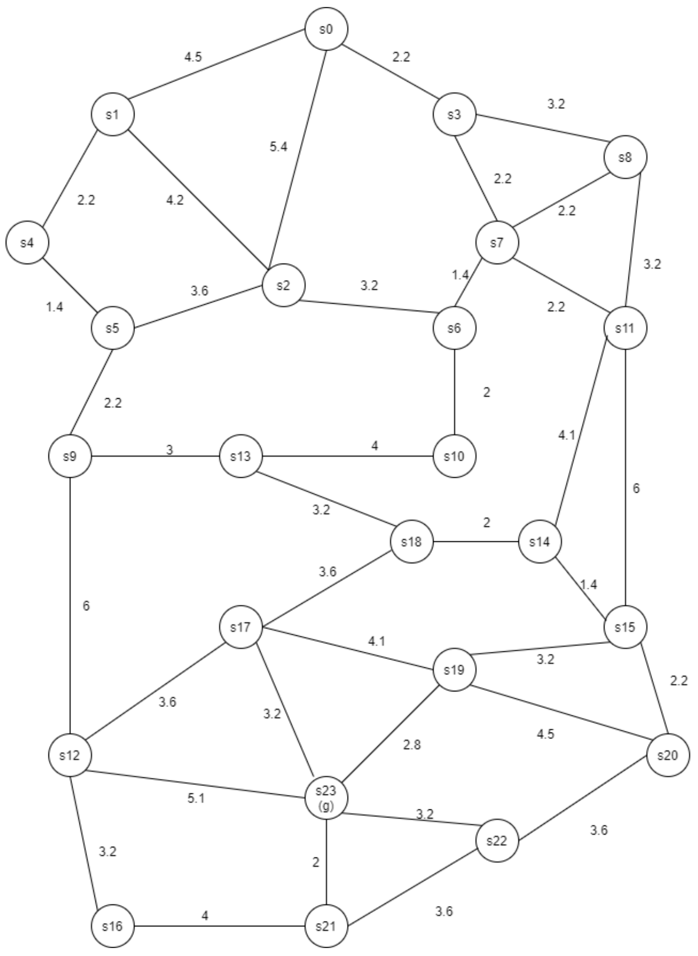

Figure 3: The graph used for Exercise 4.3. A larger picture can be found in ... (Appendix or end of Document?)

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?

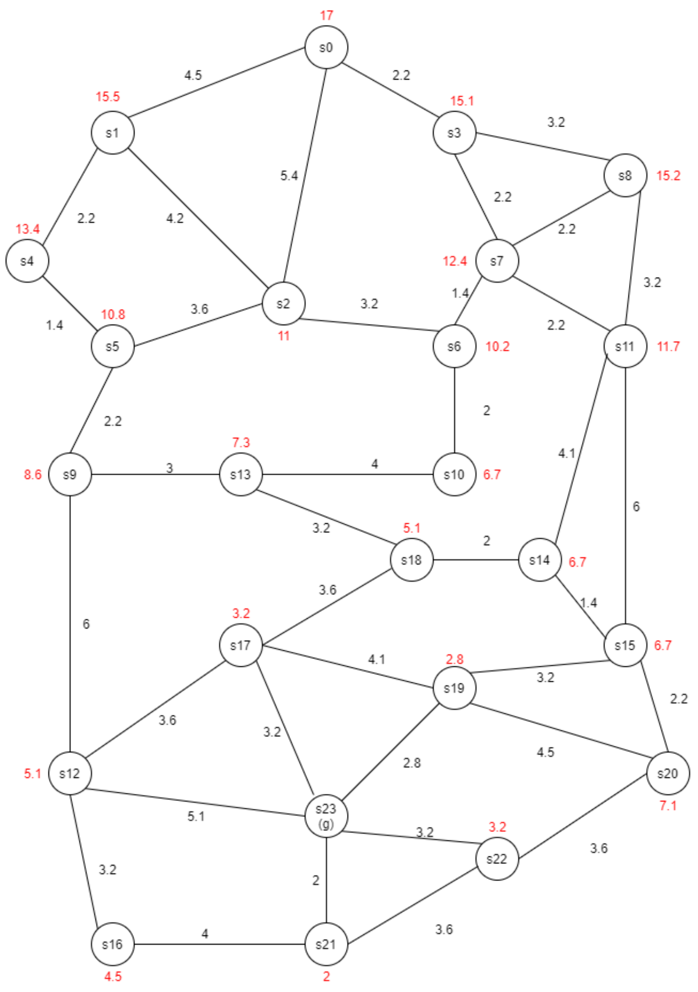

Figure 4: The graph used for Exercise 4.4 and 4.5. A larger picture can be found in ... (Appendix or end of Document?)

Exercise 4.7: Run through the implementation of the Greedy best-first search algorithm⋆

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

Open the file called *map.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.

Exercise 4.8: Upgrade the Greedy best-first search from 2D to 3D⋆

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

Open the file called *map_3d.m*. This file is the same as *map.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 4.9: Modify the implementation of the Greedy best-first search algorithm into a A* search algorithm (Optional)⋆⋆

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?

Exercise 4.10: Modify the implementation of the A* search algorithm into a weighted A* search algorithm (Optional)⋆⋆

An A* algorithm is sometimes too slow. Especially in big maps. A good compromise between A* and Greedy best-first would be a weighted A* algorithm.

Try to modify the A* search algorithm into a weighted A* search algorithm

What changes have to be made?

Try playing around with the weight. How does it affect the running time of the algorithm?

What does the algorithm become when the weight goes towards 0? What about when it goes towards ∞ ?

Exercise 4.11: Modify the implementation of the A* search algorithm into a Recursive best-first search algorithm (Optional)⋆⋆

Sometimes space is limited on drones, and the current A* implementation uses up a lot of space. The amount of space used can be reduced at the cost of speed by using a recursive best-first search algorithm.
Try to modify the A* search algorithm into a Recursive best-first search algorithm

What changes have to be made?

Did it affect the running time of the algorithm?

What is the advantage of using a Recursive best-first search algorithm compared to the A* algorithm?

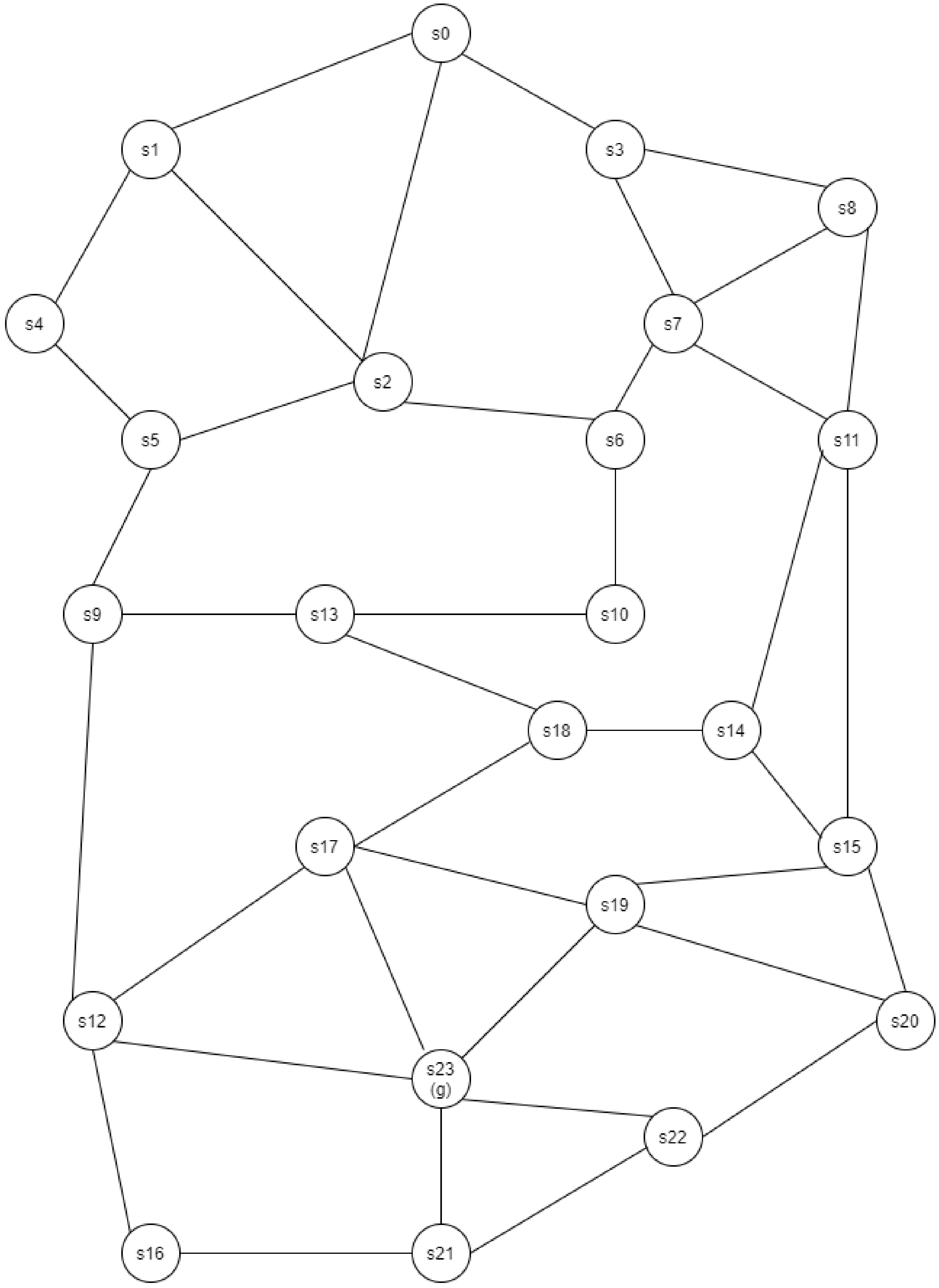

Figure 5: The graph used for Exercise 4.1 and 4.2

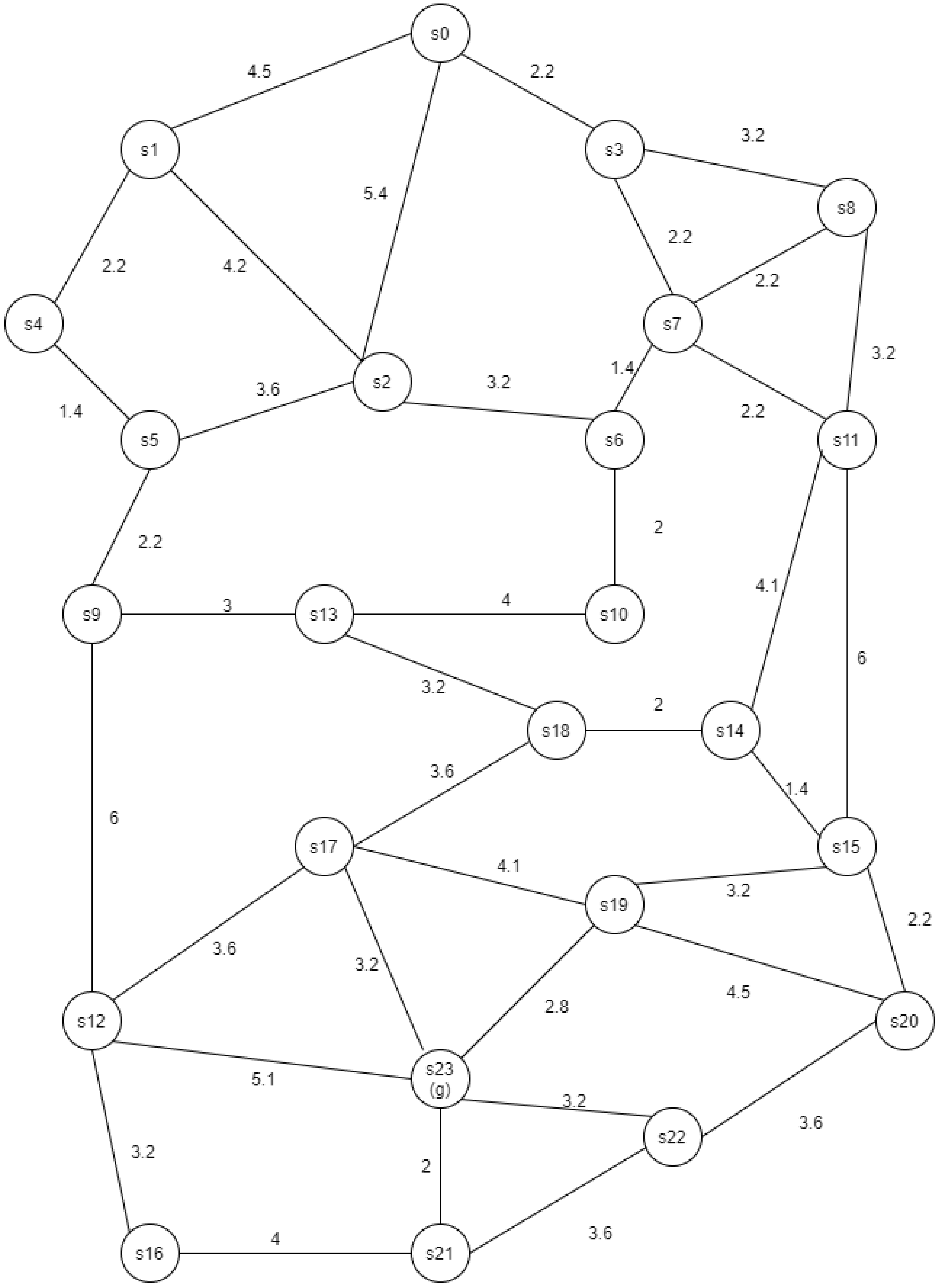

Figure 6: The graph used for Exercise 4.3

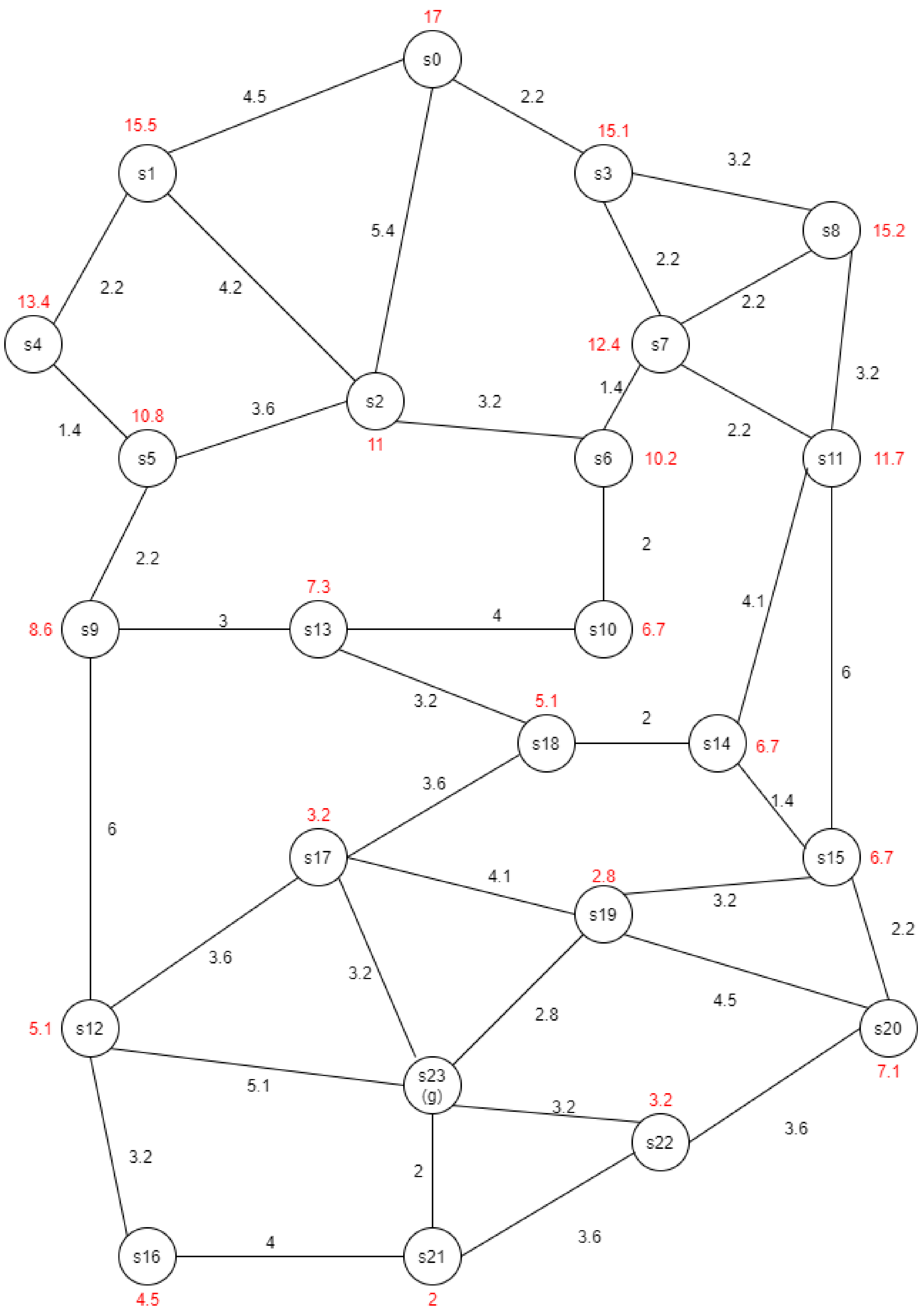

Figure 7: The graph used for Exercise 4.4 and 4.5

PART 5: Trajectory generation and control

Exercise 5.1: cubic splines ⋆

Consider the following 1-D cubic spline:

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3.$$

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

Exercise 5.2: quintic splines ⋆

Consider the following 1-D quitic 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 $x(0) = 0,\ \dot{x}(0) = 0,\ \ddot{x}(0) = 0,\ x(1) = 0,\ \dot{x}(1) = 0$, and $\ddot{x}(1) = 0$.

Exercise 5.3: B-splines ⋆

Given the *Cox-de Boor recusion formula*:

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

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 5.4: 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 "compass" direction x_C:

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

Knowing that

$$x_C = R_z(\psi)\begin{bmatrix}1\\0\\0\end{bmatrix} \quad \text{and} \quad z_B = R_z(\psi)R_y(\theta)R_x(\phi)\begin{bmatrix}0\\0\\1\end{bmatrix}$$

with

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

show that x_C is indeed perpendicular to z_B, that is, $z_B \cdot x_C = 0$.

Exercise 5.4: aggressive trajectories ⋆

This exercise is to be done in MATLAB and SIMULINK. First, clone the corresponding Git repository:

```
git clone https://github.com/DavidWuthier/minimum-snap-quadrotor-matlab.git
```

Then, switch to the `exercise` branch:

```
cd minimum-snap-quadrotor-matlab
git checkout execise
```

Make sure that the following dependencies are met: Simscape Multibody, Robotics Toolbox, Optimization Toolbox and traj_gen. If not, you can use the Add-on Explorer on the home tab of MATLAB.

To run this simulation:

1. execute the file `minimum-snap.m`
2. run the model `minimum-snap.slx`

If everything's good, you should just see the quadcopter falling in the Mechanics Explorer.

(a) In `miminum-snap.m`, compute the `mix_matrix` on line 58 based on the parameters for the *hummingbird* quadrotor. You can see how this matrix is used in `minimum-snap.slx`. When that works, the quadcopter should hover while drifting a little.

(b) In `minimum-snap.slx`, fill in the `lee` function block from the `attitude controller` subsystem according to Mellinger and Kumar (2011). When that works, the quadcopter should hover with no drift at all.

(c) In `minimum-snap.slx`, implement the `position controller` subsystem according to Mellinger and Kumar (2011). When that works, the quadcopter should be able to follow the aggressive trajectory computed at the end of `minimum-snap.m`.

PART 6: Gazebo Simulation

The following exercises will get you started on working with the Gazebo simulator together with Simulink.

Watch the following video to get an introduction to the Simulation environment:
************ Make a video and insert a link *****************

Exercise 6.1: Handling waypoints in Simulink

The first exercise will make you familiar with the Gazebo / Simulink interface. Use the files in the `rotorS_position_interface` as a templete for the exercise. The Simulink model will provide you with a subscriber to get the current position and a subscriber to get the current attitude. Then you get a publisher so you can send desired positions to the drone simulator. You can also take a look at the `initial_test` simulink model to see how it is set up.

Using the position_interface model, make a model that can follow a sequence of points (route) stored in a vector. Make it so the next point is sent once the drone is currently within some distance, d, of the target point.

- Make a plot of the position together with the target positions using the following route vector with 4 waypoints:

  ```
  route = [0 0 1; 3 0 2; −1 2 1; 1 −2 1]
  ```

  where $route = [x_1, y_1, z_1; x_2, y_2, ...]$. Send the next target when the drone is within 10 cm of the target position.
  Use yaw = 0°
  Hint: You can have the full route as a constant and have a counter for handling the current point.

Simulink template: `rotorS_position_interface`
Launch file: `mav_with_position_controller.launch`
Launch options: None

Exercise 6.2: Making a hovering attitude controlled drone

In this exercise we will make a stable hovering controller for your attitude controlled drone. For this we will be using the attitude controlled drone. To get started use the `rotorS_attitude_interface` files as a template. When using the attitude controlled drone we have 4 control inputs - roll, pitch, yawrate and thrust. For this exercise we are only going to make and tune the thrust controllers.

- Make a height controller so you can control the height width a z reference in meters.

- Make a step response of the height with your controller for a step from 0 til 1 meter.

  When running the simulation set the input of roll/pitch/yawrate to 0.

Thrust is an value ranging from -15 to 15 with 0 being approximately hovering thrust, 15 being full thrust and -15 being motors off.

Simulink template: `rotorS_attitude_interface`
Launch file: `mav_rpy_thrust_controller.launch`
Launch options: None

Exercise 6.3: Attitude controlled position controller

Using the work form Exercise 6.2 upgrade the controller to include a position controller using the attitude commands. Remember to also have a yaw controller.

- Make a plot of a 2 meter step response in x
- Make a plot of a 2 meter step response in y
- Make a plot of a 90° step response of the yaw controller$x$

  Perform all steps while hovering at 1 meter

Simulink template: `rotorS_attitude_interface`
Launch file: `mav_rpy_thrust_controller.launch`
Launch options: None

Exercise 6.4: Testing the position controller with a simple route

Expand your model from Exercise 6.3 so that it can follow a sequence of points. You could combine it with your work from Exercise 6.1.

- Make a plot of your position controller where you follow the following route and the corresponding yaw position(in degrees):
- The plot should show the current position and attitude together with the current setpoint value.

```
route = [0 0 1; 2 0 1; 2 2 1; 0 2 1; 0 0 0]
yaw = [0; 90; 180; -90; 0]
```

Simulink template: `rotorS_attitude_interface`
Launch file: `mav_rpy_thrust_controller.launch`
Launch options: None

Exercise 6.5: Making a hovering drone from motor velocities

Now we are going to have the full drone control done in Simulink. This means that we are going to be sending the desired motor velocities to the Gazebo Simulator.
For this exercise use the `rotorS_motor_velocities_interface` as a template. See the Simulation instruction document for motor numbering in Gazebo.

- Make a hovering controller (height controller)
- Make a plot of a step response from 0 to 1 meter.

Simulink template: `rotorS_motor_velocity_interface`
Launch file: `mav_without_controller.launch`
Launch options: None

Exercise 6.6: Closing the attitude control loop

With a hovering controller working from Exercise 6.5 extend your controller to include attitude control.

- Make a plot of a roll step response of 10°
- Make a plot of a pitch step response of 10°
- Make a plot of a yaw step response of 90° Make the plots while hovering at 1 meter.

Simulink template: `rotorS_motor_velocity_interface`
Launch file: `mav_without_controller.launch`
Launch options: None

Exercise 6.7: Closing the position control loop on top of the attitude controller

With a attitude controller working from Exercise 6.5 repeat Exercises 6.3 and 6.4 using your attitude controller instead of the simulators.

Simulink template: `rotorS_motor_velocity_interface`
Launch file: `mav_without_controller.launch`
Launch options: None

Exercise 6.8: 2D Path following

In this exercise we will combine the work from Exercise 6.1 with path planning. Run your A* on the 2D map for `maze_1`. From your work with path planning use your path planning function to find a path in `maze_1`, see Figure 8.

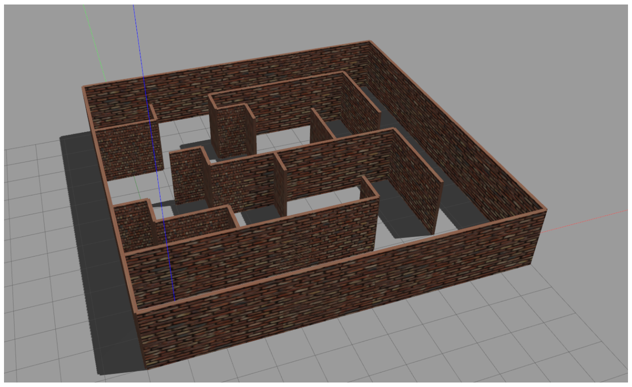

Figure 8: `Maze_1`

`Maze_1` is a 12x12x2 meter maze with an inner grid if 10x10x1 so that it can be used for 2D path planning and following. Running the `maze_1.m` file produces a variable, `map`, that contains the information about walls in the maze. A 1 indicates a wall and a 0 a free path. `map(1,1)` will give you the information about point $(0, 0)$ in the maze because of MatLab using 1-indexing - remember this when converting your route through the map into position commands for the drone. The map indexing is `map(x,y)`.

- Make the drone go from $(0, 0)$ in world coordinates to $(3, 6)$ using your A* search and route following scheme. Fly the route at a height of 1 meter.
- Make a xy-plot of the flown path with the planned route.

Simulink template: `rotorS_position_interface`
Launch file: `mav_with_position_controller.launch`
Launch options: `world_name:=maze_1`

Exercise 6.9: 3D Path following

This exercise is similar to the previous but this time we will move to 3D path planning and following. You will be working with the `maze_3D`, see Figure 9. `Maze_3D` is a 12x12x6 tower with an inner grid of 10x10x3. The usable flying heights in the maze is 1, 3, and 5 meters corresponding to the 3 levels in the map. So `map(2,2,2)` would correspond to the point $(1, 1, 3)$. Remember to account for this shift when passing the route to drone.

Use your 3D A* algorithm to fly around in the maze.

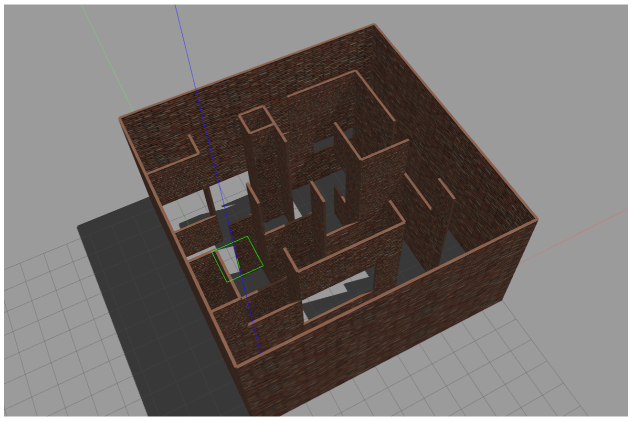

Figure 9: `Maze_3D`

- Make the drone go from $(0, 0, 1)$ in world coordinates to $(8, 9, 5)$ using your A* search and route following scheme.
- Make a 3D-plot of the flown path with the planned route.

Simulink template: `rotorS_position_interface`
Launch file: `mav_with_position_controller.launch`
Launch options: `world_name:=maze_3D`

Exercise 6.10: Combining your work

The last exercise is to test the the controllers you developed in Exercise 6.3+6.4 or Exercise 6.5-6.7 in the maze.

If you have made a working Simulink position controller(either from 6.3+6.4 or 6.5-6.7) repeat Exercise 6.9 using that controller.