# Student AirRace Simulation Platform

Amin Seffo Francisco Fonseca Luca Dalle Sasse Simon Pokorny

***Abstract*—This document presents the development of an autonomous drone racing simulation environment for the "Student AirRace" initiative. In this challenge, students design, build, and program an autonomous racing drone that can navigate through an unknown course as quickly and accurately as possible. This report demonstrates a solution for autonomous drone racing by proposing a navigation/exploration strategy and trajectory optimisation based on a continuously improving racetrack map. The performance of the algorithms is evaluated in a Hardware-In-The-Loop simulation environment. This project demonstrates the feasibility of autonomous drone racing for student-teams and provides an example of how to tackle the challenge of autonomous drone racing.**

*(Amin Seffo: Exploration and Navigation, Francisco Fonseca: Trajectoy, Luca Dalle Sasse: Perception and Mapping, Simon Pokorny: Simulation)*

***Index Terms*—Student AirRace**

## I. Introduction

The architecture of this project can be divided into four parts corresponding to individual ROS packages: A simulator with ros bridge, a navigation/exploration strategy, dynamic map generation, and trajectory optimisation. Figure 1 illustrates the functional interaction of these components. The emphasised nodes are developed in this project.

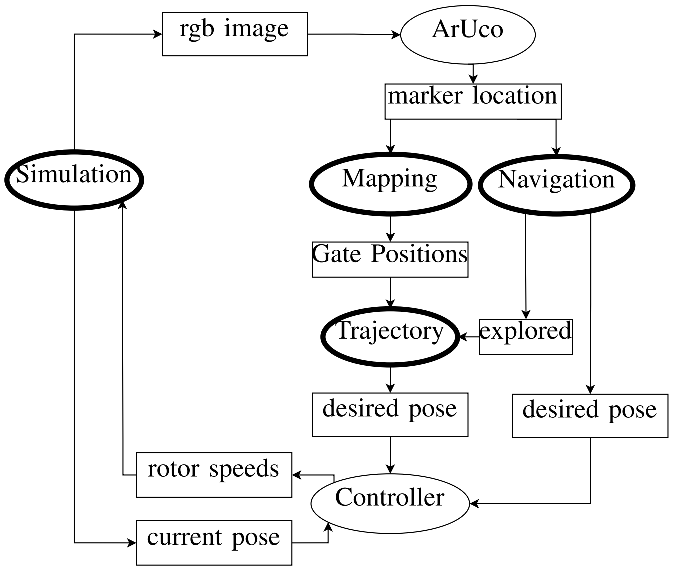

Fig. 1. Architecture

The simulation models the racetrack and the drone dynamics. The forces and torques acting on the drone depend on the angular rotor speed velocities that are being received from the controller via a ROS topic. The simulator publishes the current pose of the drone and an image of the camera attached to the drone. The camera image is analysed by an aruco_ros package, which publishes the position and orientation of the ArUco markers. This information is subscribed by the navigation and by the mapping component. The navigation node is responsible for navigating the drone through the racetrack during the first lap so that the mapping component can create a map of the racetrack. As soon as the racetrack exploration is complete, the trajectory optimisation requests the map's information to calculate the track's optimised trajectory.

## II. Simulation

The simulation is based on Unity and implements an integration to ROS via a ROS bridge, which Prof. Dr Markus Ryll developed. The project can be started with a single bash script (`./run.bash`), which launches the Unity executable and runs a docker container with the previously mentioned ROS nodes.

### A. The Racetrack

The racetrack is composed of ten gates, each consisting of two pylons. The pylons are equipped with ArUco markers at the front and the back so that the orientation and position of the pylon can uniquely identified. Figure 2 illustrates the pylons' dimensions and the position of the ArUco marker.

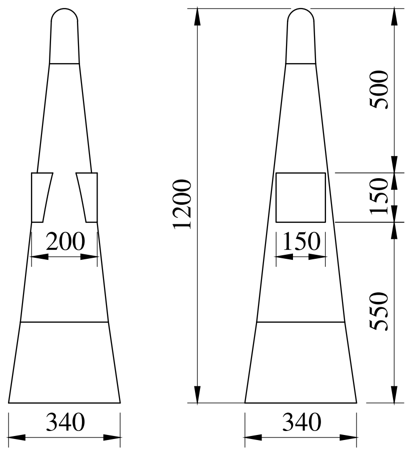

Fig. 2. Pylon size in cm

A 15cm white border surrounds each ArUco marker to improve the detection success rate. That results in an ArUco size of 120cm. ArUco markers are encoded using an original dictionary. The two pylons of a gate are located 10m apart, and the ArUco IDs follow a simple rule to ensure that the drone can uniquely identify the gate as soon as one ArUco marker is detected. The markers in the front always have an even number, while the markers at the back encode an odd number. Figure 3 illustrates the IDs of the ArUco marker on the gate, while $n$ encodes the gate number.

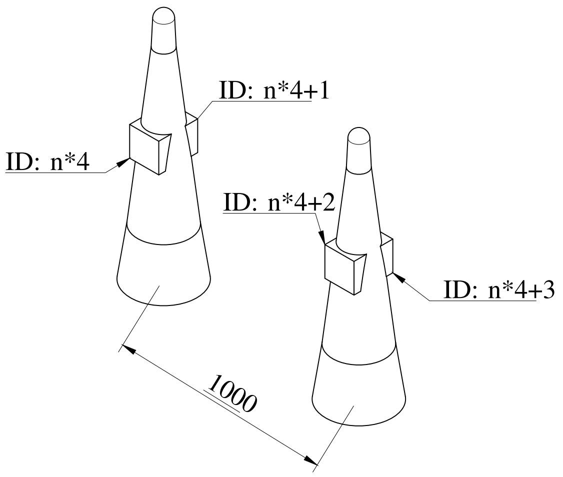

Fig. 3. Gate Marker IDs

The racetrack forms a closed loop, and the gates are located so that the left pylon of the next gate is on a line perpendicular to the line connecting the two pylons of the previous gate and crossing its left pylon. The angle of the gates is either 45° or -45° with respect to the previous gate. That ensures that the ArUco marker can be detected by the drone. Figure 4 illustrates two consecutive gates from a top view.

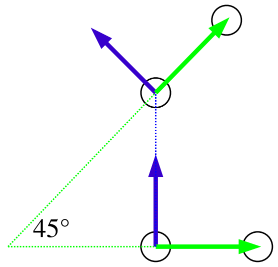

Fig. 4. Gate orientation

## B. QuadrotorDynamics

The simulation supports two types of drones. A standard quadrotor model and the official Student AirRace demonstrator drone called "StarOne". The default drone, as shown in figure 5, has four propellers and a symmetrical frame layout. The first and the third motor spin counterclockwise while the remaining motors spin clockwise.

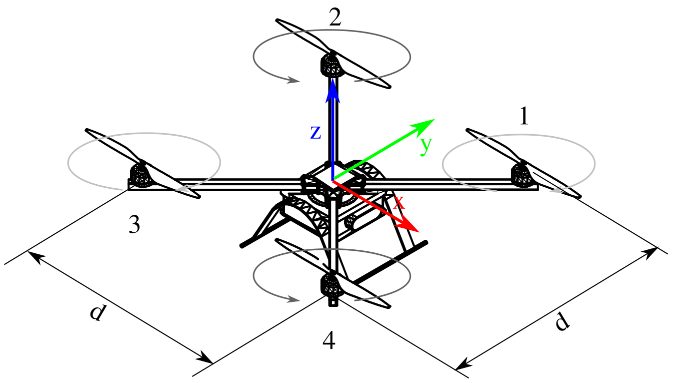

Fig. 5. Propeller Direction and Coordinate Frame

We assume that the force $f_i$, which is generated by the spinning propeller, is quadratic to the angular velocity $\omega_i$ and depends on the pitch and the size of the propeller, which is described with a constant factor $c_f$. Based on these assumptions, we can model the force generated by a single motor as $f_i = c_f * \omega_i^2$. Furthermore, we assume that a motor's torque is linear to the force with a scaling factor of $c_f$. Therefore as described in [4] we assume that the torque of the $i$-th propeller can be written as $\tau_i = (-1)^i c_{\tau f} f_i$. The sum of forces and torques acting on the system can be described as:

$$\begin{bmatrix} f \\ M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -\frac{d}{2} & -\frac{d}{2} & \frac{d}{2} & \frac{d}{2} \\ \frac{d}{2} & -\frac{d}{2} & -\frac{d}{2} & \frac{d}{2} \\ -c_{\tau f} & c_{\tau f} & -c_{\tau f} & c_{\tau f} \end{bmatrix} \times \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \qquad (1)$$

The StarOne demonstrator is an octocopter in a coaxial configuration illustrated in figure 6.

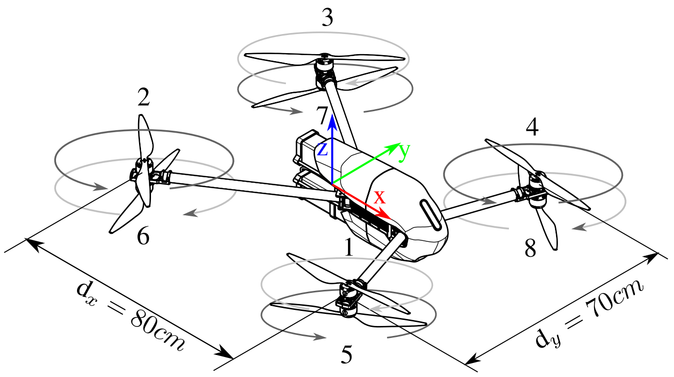

Fig. 6. Propeller Direction and Coordinate Frame

The dynamics are similar to the quadrotor configuration, but the four additional motors result in four additional columns in the force-to-wrench matrix below. Because this model is not symmetric, the distance between the motors in the x direction differs from that in the y direction. To account for the difference, $d$ was replaced by $d_x$ and $d_y$

$$\begin{bmatrix} f \\ M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -\frac{d_y}{2} & -\frac{d_y}{2} & \frac{d_y}{2} & \frac{d_y}{2} & -\frac{d_y}{2} & -\frac{d_y}{2} & \frac{d_y}{2} & \frac{d_y}{2} \\ -\frac{d_x}{2} & \frac{d_x}{2} & \frac{d_x}{2} & -\frac{d_x}{2} & -\frac{d_x}{2} & \frac{d_x}{2} & \frac{d_x}{2} & -\frac{d_x}{2} \\ c_{\tau f} & -c_{\tau f} & c_{\tau f} & -c_{\tau f} & -c_{\tau f} & c_{\tau f} & -c_{\tau f} & c_{\tau f} \end{bmatrix} \times \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \end{bmatrix} \tag{2}$$

Because the matrix is not invertible, a Moore-Penrose pseudo inverse was calculated in the geometric controller to compute the angular velocities of the motors.

The run.bash start script accepts a parameter to specify the drone that should be used. When starting the project, e.g. with the command `./run.bash -d StarOne`, results in a simulation with the StarOne drone model and an adjusted ros bridge considering the increased number of motor velocities. If the parameter `-d Default` or no parameter is set, it will start with the default quadrotor model.

### C. Hardware in the Loop (HITL) Simulation

This project supports a Hardware-In-The-Loop simulation, where the simulation, including the ROS bridge, runs on an x86 workstation while the whole autonomy stack (perception, navigation, mapping, trajectory optimisation, controller) runs on an attached ARM-based companion computer. The concept was validated using an Nvidia jetson nano, which serves as the companion computer for the StarOne drone. The companion computer is required to be attached to the simulation workstation via ethernet. The simulation can be started with a single bash script located on the simulation workstation. The script initiates an ssh connection to the companion computer and executes a script that builds the docker image on the companion computer and runs the same ros launch file as in the SITL simulation but with a parameter that disables the instantiation of the Unity bridge. The run.bash start script accepts a parameter to specify whether HITL or SITL simulation should be used. Starting the project, e.g. with the command `./run.bash -s HITL`, results in a Hardware-In-The-Loop simulation. If the parameter is set to SITL or omitted, SITL simulation is used.

## III. Perception and Mapping

The perception stage of our system employs the ARUCO Library [2]. The library, which has been developed by the Ava group of the University of Cordoba (Spain), allows us to obtain real-time marker-based 3D pose estimation using aruco markers. It takes as input camera RGB image and camera info (calibration parameters) topics, and it publishes the detected marker; as a result, it provides the corresponding id, x,y, and z coordinates and the marker's orientation. In the project's setup, the drones are equipped with a monocular camera that provides 1280x720px resolution images already calibrated and rectified (see the D, K, R, P matrices within camera info topic). In the most general case, images must be rectified before being used by the aruco marker detector and one way to do this is by exploiting the functionalities of the *image_pipeline* package. Coordinates and orientation are projected into the desired frame considering the camera's extrinsic and the drone's position. In this context, every measurement has been expressed with respect to the world frame.

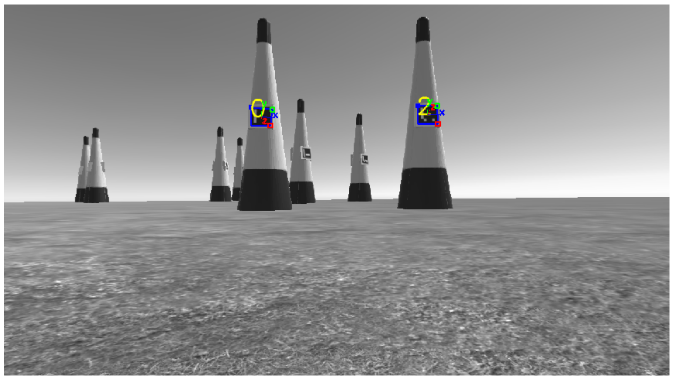

Fig. 7. Resulting aruco marker detection

Throughout the simulation, the identified coordinates of each pylon are saved through a ROS service. A client subscribes to the markers topic and sends the obtained messages to the service. There, each new coordinate set is stored accordingly to the id and subsequently utilised to enhance the estimation of the complete racetrack by computing the mean and variance of all the values for each x, y, and z coordinate of each pylon. Moreover, to further improve the accuracy of the estimation, an outlier analysis has been introduced not to consider unreliable marker measurements that could substantially affect the overall estimation. Specifically, measurements whose coordinate values deviated from the mean value, calculated thus far, by more than three times the standard deviation are discarded. Supplementary data is also recorded, including the aggregate count of identified markers, the count of erroneously out-of-range identified markers, a corresponding error rate computed based on these values and the number of outliers detected for each marker's estimation. The service sends back as a response only the updated values of each marker's coordinate estimation, while the whole set of the resulting statistical values is stored in a file that gives a whole picture of how the estimate is progressing and updating over time. The map of the track is thus subject to continuous updates as additional markers are identified and subsequently utilised to compute the optimal trajectory. It can be noted that in the case of marker's ids which have been identified many times, a more accurate estimate of the position is obtained with x,y and z values that tend to stabilise with a decreasing variance.

## IV. Exploration and Navigation

The main objective of this part is to develop an autonomous drone navigation system that can navigate through a racetrack without any prior knowledge of its layout. To achieve this goal, a set of waypoints is generated between each pair of pylons on the track (which acts as gates). The waypoints consist of midpoints between two pylons, and additional waypoints are added when the distance between two gates is too long to be covered by a straight path. The waypoint set is then used to create a non-optimal trajectory for the drone to explore the environment and navigate through the track.

The exploration and navigation pipeline is illustrated in figure 8 and consists of the following components:

### A. Calibration

It is essential to calibrate the drone position before the flight and verify that the camera is aligned with the simulated track to ensure the drone takes off in the right direction. It is crucial to do so because the simulation may cause the drone's orientation to become tilted, leading to an incorrect flight path.

### B. Exploration and Search Strategies

Once the calibration process is complete, the exploration phase is initiated, and the drone proceeds to locate the Aruco marker array. The drone hovers at the beginning of the flight until the marker is detected. In case the marker is not detected, the drone switch between these two search strategies until both markers are detected:

*1) Step Forward:* This search strategy deals with distant gates which cannot be detected due to the limited focal length of the camera and its field of view. To mitigate this issue, the drone executes small steps to search for the Aruco marker. The first step of this manoeuvre is waiting for the drone to return to the HOVER mode, which occurs when the controller error falls below a specified threshold. At this point, the drone advances forward in small increments to detect the marker.

*2) Step Around:* If the drone detects only one marker, it executes a backward step to correct its orientation. This method is necessary to ensure that the drone is aligned with both pylons in case it was tilted during the initial detection. By correcting its orientation, the drone can ensure accurate detection of the markers and improve the overall precision of its flight.

### C. Gate Detection and Navigation

After detecting both Aruco markers, a new gate is added to a list containing its ID, position, and orientation. The geometric controller [4] uses the *MultiDOFJointTrajectory* message to obtain the midpoint between the two pylons. The message provides the midpoint coordinates and the angle $(x, y, z, \alpha)$. This trajectory consists of a sequence of waypoints the drone follows without optimising its path or minimising the time to reach each waypoint.

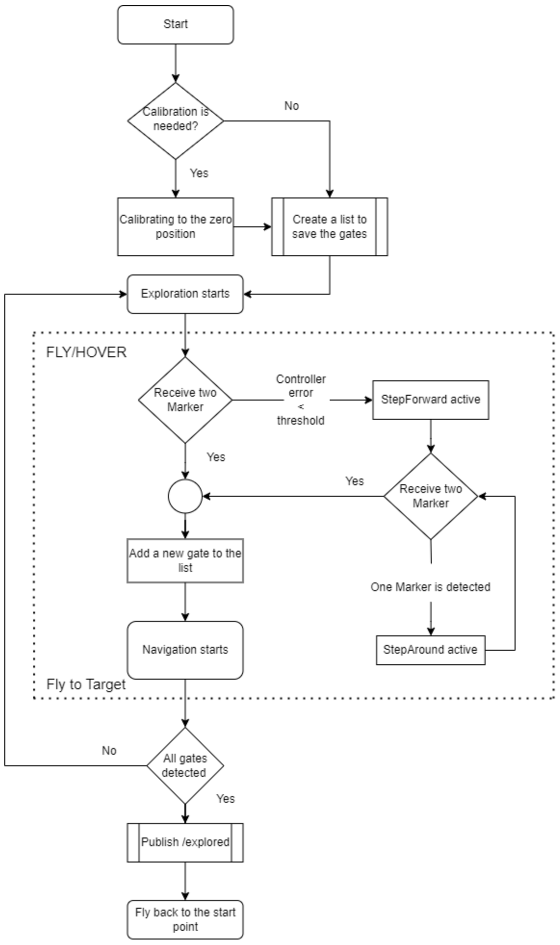

Fig. 8. Flowchart of the Exploration and Navigation Algorithm

Figure 9 illustrates the racetrack that the drone has explored, which includes all of the gates on the map. However, the transition between the two points is not continuous, emphasising the importance of optimising the drone's trajectory. By doing so, the drone can navigate the racetrack more efficiently and accurately.

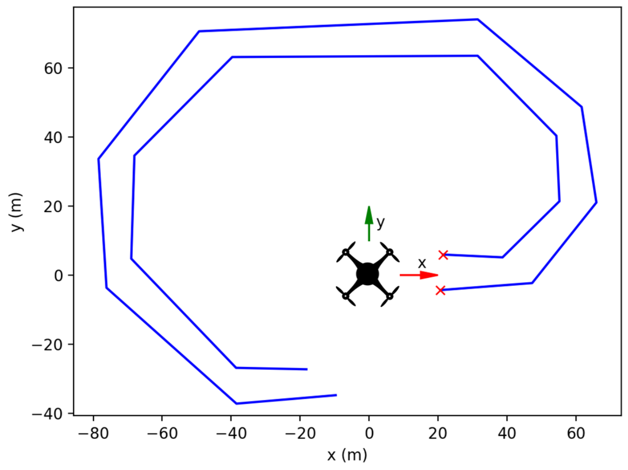

Fig. 9. Explored racetrack

## V. Trajectory Optimization

The main objective of trajectory optimisation is to generate an optimal and smooth trajectory based on the given map. In addition to this, after each lap, the trajectory is updated with the most recently improved map waypoints.

### A. *Minimum Snap Trajectory Optimization*

In this project, the trajectory optimisation method minimises the fourth derivative of the positions while also setting constraints for several waypoints [3]. Figure 10 is an example of optimising the second derivative for three segments.

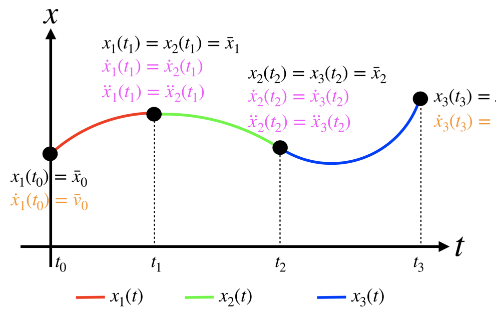
$x$
$x_1(t_1) = x_2(t_1) = \bar{x}_1$
$\dot{x}_1(t_1) = \dot{x}_2(t_1)$
$\ddot{x}_1(t_1) = \ddot{x}_2(t_1)$
$x_2(t_2) = x_3(t_2) = \bar{x}_2$
$\dot{x}_2(t_2) = \dot{x}_3(t_2)$
$\ddot{x}_2(t_2) = \ddot{x}_3(t_2)$
$x_3(t_3) = \bar{x}_3$
$\dot{x}_3(t_3) = \bar{v}_3$
$x_1(t_0) = \bar{x}_0$
$\dot{x}_1(t_0) = \bar{v}_0$
$t$
$t_0$
$t_1$
$t_2$
$t_3$
$x_1(t)$
$x_2(t)$
$x_3(t)$

Fig. 10. Multi-segment minimum acceleration (r = 2) trajectory optimization with 3 segments [3]

The method applies this principle by setting waypoint constraints for the centre of each gate, calculated from the pylon's locations obtained during mapping. The *mav_trajectory_generation* [5] is used to generate the optimal trajectory and can be visualised in the image of the trajectory markers provided in figure 11, which are in rviz.

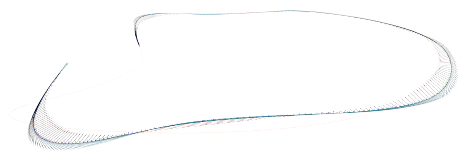

Fig. 11. Trajectory markers in rviz. Orange lines are previous trajectories, and arrows represent the position and orientation of the drone in the trajectory.

### B. *3-Dimensional Trajectory*

In 3-dimensional trajectory optimisation, the objective is to optimise the trajectory in $x$, $y$, and $z$ (position). For this method, we only give constraints for the intermediate waypoints of each gate and optimise the trajectory according to the position variables. For a race with ten laps, this lead to an average of $40,60s$ per lap (the best time of the three approaches). However, the main issue with this approach is that the drone's orientation concerning the yaw is not guaranteed to have the camera pointing forward in the direction of motion, which is problematic since we must detect the aruco markers to get better estimates of the gate's positions. Furthermore, in the case of a changing environment we want to ensure a constant forward vision.

### C. *4 and 6-Dimensional Trajectory*

In 4 and 6-dimensional trajectory optimisation, the trajectory optimises in four $(x, y, z, \psi)$ or six dimensions $(x, y, z, \phi, \theta, \psi)$. This optimisation is more comprehensive than 3-dimensional trajectory optimisation and considers the constraints and characteristics of the vehicle to generate a trajectory that is optimal in 4 or all dimensions. The 4-dimensional trajectory approach already allows for the goal of always having the camera facing forward, leading to an average lap time of $41.52s$. The slower average time is due to the time lost to get the correct yaw position at each gate. Since there are no restrictions for the other two dimensions, in some conditions the camera might not be ideally oriented.

The 6-dimensional trajectory optimisation leads to the a better result by a small margin with an average time of $41,21s$. Although not a huge improvement it has the benefit of using all possible degrees of freedom which allows us to fully utilise all of the possible motions of the drone to achieve the best possible trajectory given our constraints (which in this case are the gate midway position, yaw always facing the movement when reaching a gate and the other angles to 0 to make sure the camera is in the horizontal position).

## VI. Conclusion

In conclusion, this paper presents a solution for an autonomous drone racing that includes a navigation/exploration strategy and trajectory optimization based on a continuously improving racetrack map. The project demonstrates the feasibility of autonomous drone racing for student-teams and provides an example of how to tackle the challenge of autonomous drone racing.

## References

[1] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A Flexible Quadrotor Simulator," in Proceedings of the 2020 Conference on Robot Learning, 2021, pp. 1147–1157.

[2] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, M.J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," Pattern Recognition, Volume 47, Issue 6, 2014, pp. 2280-2292.

[3] V. Tzoumas, L. Carlone, "16.485: Visual Navigation for Autonomous Vehicles (VNAV) Fall 2020, Lecture 9: Trajectory Optimisation I," Massachusetts Institute of Technology, 2020.

[4] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," 49th IEEE Conference on Decision and Control (CDC), 2010.

[5] Markus Achtelik, Michael Burri, Helen Oleynikova, Rik Bähnemann, Marija Popović. *MAV Trajectory Generation*, Autonomous Systems Lab, ETH Zurich. https://github.com/ethz-asl/mav_trajectory_generation. Rik Bahnemann, maintainer, email: brik@ethz.ch.