DIETI UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL'INFORMAZIONE

Automation and Robotics Engineering

Field and Service Robotics Course Technical Report

Minimum Snap Trajectory and Control Strategies for a Racing Drone Flight

---

Student: Salvatore Granata P38000219

Professor: Fabio Ruggiero

Abstract

This report focuses on the development of a racing drone tracking system, starting from the generation of an optimal trajectory planning method defined as "minimum snap trajectory" to enhance lap completion time. The proposed approach integrates a Nonlinear Model Predictive Control (NMPC) system designed to ensure optimal performance, which is then compared with other controllers studied during the course. The trajectory planning algorithm optimizes the drone's path to minimize the trajectory over laps by accounting for corridor constraints. The results underscore the potential of advanced trajectory planning and controls in pushing the boundaries of competitive drone racing.

The GitHub repository cointaining the report and the all the code sources of the developed homework can be found at the following link: https://github.com/Salvatoregr/project_FSR.

Contents

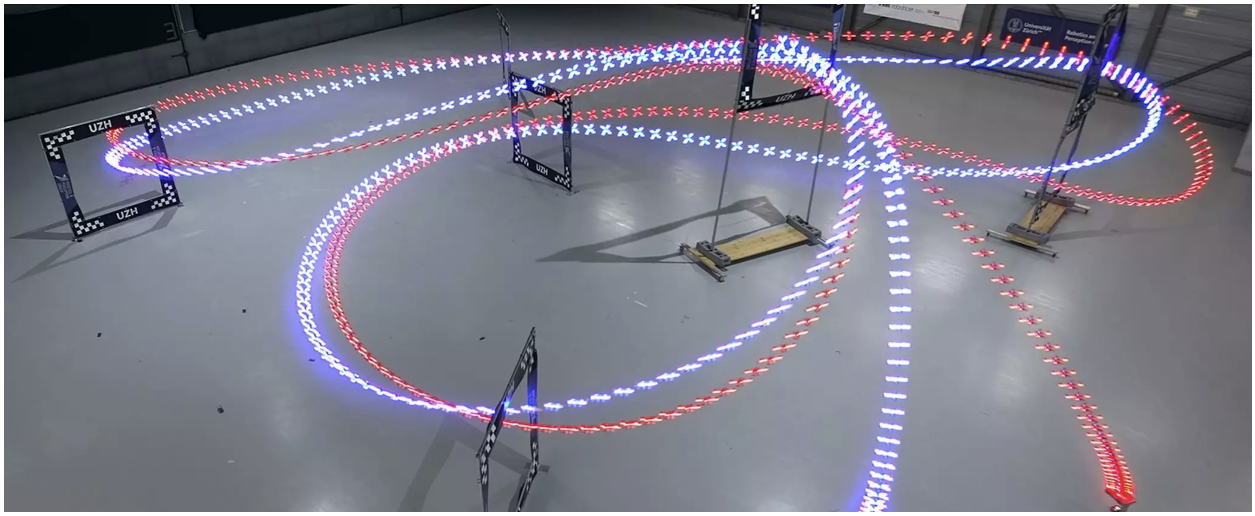


Figure 1: An example of trajectory in real-world drone racing application. The complete video can be found here

1 Introduction

Multirotor drones possess exceptional agility, making them indispensable for time-sensitive missions such as search and rescue or aerial delivery. Over the past decade, research on autonomous multirotor drones has increasingly focused on enhancing their speed and agility. Competitions have been established to foster the development of autonomous systems capable of surpassing the performance of expert human pilots.
This paper addresses the challenge of navigating a quadrotor through multiple waypoints with the shortest possible path, a crucial capability for missions and inspection tasks, and a fundamental goal in search and rescue operations as well as drone racing. Professional drone racing pilots demonstrate extraordinary proficiency, maneuvering their quadrotors through race tracks at speeds that are currently challenging for autonomous systems to match.

Achieving path-optimal multi-waypoint flight presents fundamental challenges in robotics, including planning, control, aerodynamics, and modeling. These challenges intensify with the demand for high accelerations and aggressive attitude changes. The study of new methods for minimum-path trajectory generation and tracking is essential not only to address these challenges, but also to expand the physical capabilities of quadcopters, given their limited actuation capability. The main objective of this project is to use an optimal trajectory planning method to obtain a minimum snap trajectory [1] with corridor constraints, then to use a Nonlinear Model Predictive control (NMPC) system, a Geometric control system and then an Hierarchical control system respectively, to ensure the tracking of the given trajectory. The results obtained with the control methods will be discussed by comparison with each other and with those of what can be considered the vanguard in the aforementioned field, i.e. the application of a similar trajectory planning method and subsequently a Model Predictive Contouring Control (MPCC) method proposed by Dr. Davide Scaramuzza from UZH [2].

The minimum snap method is proposed to achieve an optimal path quadrotor flight, taking into account constraints on the quadrotor's position, velocity, and acceleration. The nominal path can be any continuously differentiable 3D trajectory, parameterized by time, to ensure precise passage through the waypoints, their positions are encoded within the constraints in the form of continuity position ones and desired position at a certain time. The proposed controllers aim to accurately track the generated position and yaw, such that the drone is always facing the heading direction along the path.

2 Trajectory Planning in Autonomous Drone Racing

Trajectory planning is a critical aspect in drone operations, particularly in the field of autonomous racing drones, where the main goal is to navigate through a complex path in the shortest possible time. For this reason, efficient trajectory planning becomes fundamental.

2.1 Polynomial Representation of Trajectories

In order to ensure continuity and differentiability up to the N-2th derivative of position, trajectories are generally represented by Nth-order polynomials, i.e.:

$$\begin{aligned} p(t) &= p_0 + p_1 t + p_2 t^2 + p_3 t^3 + p_4 t^4 + ... + p_n t^n \\ &= \textstyle\sum_{i=0}^{N} p_i t^i \end{aligned}$$

where $p_0, p_1, p_2, ..., p_n$ are the parameters of the polynomial.
Defining the vector $\boldsymbol{p} = [p_0 \ p_1 \ ... \ p_n]^T$ as the parameter vector and $\boldsymbol{t} = [1 \ t \ t^2 \ ... \ t^n]$ as the time vector, then at any moment t, the trajectory's parameters (position, velocity, acceleration, jerk and snap) can be calculated as follows:

- Position $p(t) = [1 \ t \ t^2 \ ... \ t^n] \ \mathbf{p}$;
- Velocity $v(t) = p'(t) = [0 \ 1 \ 2t \ 3t^2 \ 4t^3 \ ... \ nt^{n-1}] \ \mathbf{p}$;
- Acceleration $a(t) = p''(t) = [0 \ 0 \ 2 \ 6t \ 12t^2 \ ... \ n(n-1)t^{n-2}] \ \mathbf{p}$;
- Jerk $jerk(t) = p^{(3)}(t) = [0 \ 0 \ 0 \ 6 \ 24t \ ... \ \frac{n!}{n-3!}t^{n-3}] \ \mathbf{p}$;
- Snap $snap(t) = p^{(4)}(t) = [0 \ 0 \ 0 \ 0 \ 24 \ ... \ \frac{n!}{n-4!}t^{n-4}] \ \mathbf{p}$;

In order to describe the whole path through the gates exploiting the polynomial expression, the latter has been divided into a sequence of polynomial traits. Continuity constraints become fundamental to make it work.

$$p(t) = \begin{cases} [1, t, t^2, ..., t^n] \cdot p_1 & t_0 \leq t < t_1 \\ [1, t, t^2, ..., t^n] \cdot p_2 & t_1 \leq t < t_2 \\ \dots & \\ [1, t, t^2, ..., t^n] \cdot p_k & t_{k-1} \leq t < t_k \end{cases}$$

2.2 Minimum-Snap Trajectory Planning

The goal of the trajectory planning is to find the polynomial parameters of the trajectory. The problem is to understand how to navigate through different keyframes, whose position in space is known at specified times.
It is possible to set the trajectory to satisfy a series of constraints, imposing desired position, velocity, and acceleration at the starting and ending points and the passage through the gates' centers, ensuring that the connection between adjacent segments is smooth (continuous position, velocity, and acceleration). Additionally, in the interval between each keyframe, there is a safe corridor that the quadcopter must stay within. A trivial trajectory that satisfies these constraints is one that interpolates between keyframes using straight lines. Typically, there are numerous trajectories that satisfy the constraints, but the real challenge is to find the optimal trajectory within the feasible set.

For this reason, the problem can be modeled as a constrained optimization problem, which can be formulated as:

$$\begin{aligned} &\min f(p) \\ \text{s.t.} \quad & \\ &A_{eq}p = b_{eq} \\ &A_{ieq}p \leq b_{ieq} \end{aligned}$$

where p is a vector which includes all the coefficients of each polynomial defined for each interval of time.

From here, the aim is to minimize the function $f(p)$ keeping the equality and inequality constraints always satisfied. Since the trajectory planning is said to be "minimum snap", as the name suggest, the objective function to be minimized is the snap, hence:

$$minimum\ snap : \min f(p) = \min(p^{(4)}(t))^2$$

From here it's possible to plan the trajectory in the space.

2.2.1 Time Allocation and Computation of the Optimization Function

In order to carry out the trajectory planning it is fundamental to understand where the quadcopter needs to find itself at each specific time in space, so it is mandatory to allocate specific time instant to specific positions. Multiple waypoints are added in between the keyframe so that the what was before a single segment is now made up of many smaller ones. Starting from the desired total time length T, the latter is divided into smaller intervals after computing each segment distance.
The total time T is allocated to each segment based on its length. In conclusion, the minimum-snap optimization function can be computed as:

$$\begin{aligned} &\min \int_0^T (p^{(4)}(t))^2 dt \\ &= \min \sum_{i=1}^{k} \int_{t_{i-1}}^{t_i} (p^{(4)}(t))^2 dt \\ &= \min \sum_{i=1}^{k} \int_{t_{i-1}}^{t_i} ([0\ 0\ 0\ 0\ 24\ ...\ \tfrac{n!}{n-4!}t^{n-4}] \cdot \boldsymbol{p})^T\ [0\ 0\ 0\ 0\ 24\ ...\ \tfrac{n!}{n-4!}t^{n-4}] \cdot \boldsymbol{p}\ dt \\ &= \min \sum_{i=1}^{k} \boldsymbol{p}^T \int_{t_{i-1}}^{t_i} [0\ 0\ 0\ 0\ 24\ ...\ \tfrac{n!}{n-4!}t^{n-4}]^T\ [0\ 0\ 0\ 0\ 24\ ...\ \tfrac{n!}{n-4!}t^{n-4}] dt\ \boldsymbol{p} \\ &= \min \sum_{i=1}^{k} \boldsymbol{p}^T Q\ \boldsymbol{p} \\ &= \min \boldsymbol{p}^T Q\ \boldsymbol{p} \end{aligned}$$

which turns out to be the known form of quadratic programming (QP) problems.
The matrix **Q** will be a block diagonal matrix with the following structure:

$$\mathbf{Q} = \begin{bmatrix} Q_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & Q_k \end{bmatrix}$$

such that:

$$\mathbf{Q}_i = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1152T_i & 2880T_i^2 & 5760T_i^3 & 10080T_i^4 \\ 0 & 0 & 0 & 0 & 2880T_i^2 & 9600T_i^3 & 21600T_i^4 & 40320T_i^5 \\ 0 & 0 & 0 & 0 & 5760T_i^3 & 21600T_i^4 & 51480T_i^5 & 100800T_i^6 \\ 0 & 0 & 0 & 0 & 10080T_i^4 & 40320T_i^5 & 100800T_i^6 & 201600T_i^7 \end{bmatrix}$$

with $T_i = t_i - t_{i-1} \qquad , i = 1 \dots k$

2.2.2 Computation of the Equality Constraints

Defined the Objective function, there's the need to derive the linear constraints that the problem is subject to. Starting from the equality ones, in 2.2 there's the need of imposing desired starting and ending position, velocity and acceleration, and to impose continuity for the latters in correspondence of the juncture between two sequential polynomials. In order to do so, it's been relied on the definition of equality constraints according to the following reasons:

- The first polynomial evaluated at the starting time must be equal to the starting position, and by similar reasoning its time derivatives evaluated in the same instant must be equal to the starting velocity and acceleration:

$$\begin{bmatrix} [1\ t_0\ \dots\ t_0^7]\ \boldsymbol{p}_1 \\ [0\ 1\ 2t_0\ \dots\ 7t_0^6]\ \boldsymbol{p}_1 \\ [0\ 0\ 2\ 6t_0\ \dots\ 42t_0^5]\ \boldsymbol{p}_1 \end{bmatrix} = \begin{bmatrix} \textit{desired initial position} \\ \textit{desired initial velocity} \\ \textit{desired initial acceleration} \end{bmatrix}$$

- The difference between two consecutive polynomials evaluated at the time in which the first ends and the second starts must be zero to ensure continuity. This is done for position, velocity and acceleration:

$$\begin{bmatrix} [1\ t_i\ \dots\ t_i^7]\ \boldsymbol{p}_i - [1\ t_i\ \dots\ t_i^7]\ \boldsymbol{p}_{i+1} \\ [0\ 1\ 2t_i\ \dots\ 7t_i^6]\ \boldsymbol{p}_i - [0\ 1\ 2t_i\ \dots\ 7t_i^6]\ \boldsymbol{p}_{i+1} \\ [0\ 0\ 2\ 6t_i\ \dots\ 42t_i^5]\ \boldsymbol{p}_i - [0\ 0\ 2\ 6t_i\ \dots\ 42t_i^5]\ \boldsymbol{p}_{i+1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

- When passing through the keyframes, the continuity conditions are imposed and also the desired value for the position is defined:

$$\begin{bmatrix} [1\ t_i\ \dots\ t_i^7]\ \boldsymbol{p}_i \\ [1\ t_i\ \dots\ t_i^7]\ \boldsymbol{p}_i - [1\ t_i\ \dots\ t_i^7]\ \boldsymbol{p}_{i+1} \\ [0\ 1\ 2t_i\ \dots\ 7t_i^6]\ \boldsymbol{p}_i - [0\ 1\ 2t_i\ \dots\ 7t_i^6]\ \boldsymbol{p}_{i+1} \\ [0\ 0\ 2\ 6t_i\ \dots\ 42t_i^5]\ \boldsymbol{p}_i - [0\ 0\ 2\ 6t_i\ \dots\ 42t_i^5]\ \boldsymbol{p}_{i+1} \end{bmatrix} = \begin{bmatrix} keyframe \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- The last polynomial evaluated at the ending time must be equal to the final position, and by similar reasoning its time derivatives evaluated in the same instant must be equal to the final velocity and acceleration:

$$\begin{bmatrix} [1\ t_k\ \dots\ t_k^7]\ \boldsymbol{p}_k \\ [0\ 1\ 2t_k\ \dots\ 7t_k^6]\ \boldsymbol{p}_k \\ [0\ 0\ 2\ 6t_k\ \dots\ 42t_k^5]\ \boldsymbol{p}_k \end{bmatrix} = \begin{bmatrix} \textit{desired final position} \\ \textit{desired final velocity} \\ \textit{desired final acceleration} \end{bmatrix}$$

By following this rules the matrices A_{eq} and b_{eq} were computed through an iterative algorithm. In particular,

$$A_{eq} \in \mathbb{R}^{n \times m} \ , \ b_{eq} \in \mathbb{R}^{n} \text{ where:}$$

- $n = 3 \cdot \#waypoints + (\#keyframes - 2)$;
- $m = 8 \cdot \#segments$.

2.2.3 Computation of the Inequality Constraints

In order to prevent the generated trajectory from staying too far away from the straight line connecting the keyframes in the transition from one to the other, a geometric constraint was used in the form of inequality constraints. The idea is to define a series of cubes centred on the points of the path and impose that the position polynomial evaluated at the time instant allocated for that point is such that to guarantee that the trajectory remains within defined margins. The cube will be the result of the composition of the inequality constraints in the 3D space.

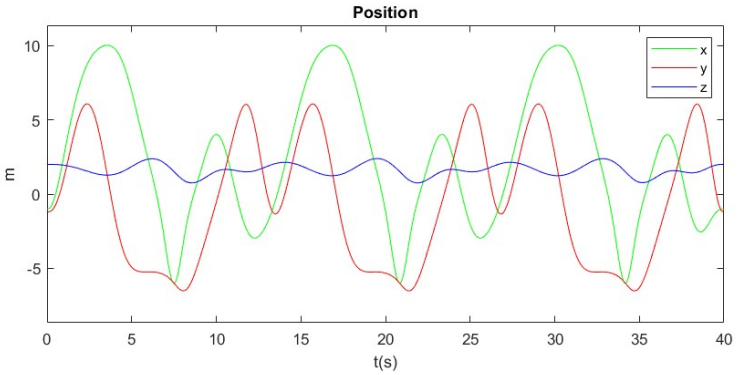
In other words, the inequality constraints were defined according to the following method: the polynomial evaluated at the time instant allocated for a specific point must be equal to a value belonging to the interval $[w_i - r, w_i + r]$, where w_i is the $i-th$ point component along the axis for which we are solving the problem and r is the semilength of the cube edge:

$$\begin{bmatrix} [1 \ t_i \ ... \ t_i^7] \ \boldsymbol{p}_i \\ -[1 \ t_i \ ... \ t_i^7] \ \boldsymbol{p}_i \end{bmatrix} \leq \begin{bmatrix} w_i + r \\ w_i - r \end{bmatrix}$$

By following this rules the matrices A_{ieq} and b_{ieq} where computed through an iterative algorithm. In particular, $A_{ieq} \in \mathbb{R}^{d \times m}$ and $b_{ieq} \in \mathbb{R}^{d}$ where:
- $d = 2 \cdot \#waypoints$

2.2.4 Minimum-Snap Trajectory Results

The position of the keyframes have been defined in order to be the closest as possible to the configuration shown UZH Robotics and Perception Group's video (UZH Robotics and Perception Group) and discussed in Scaramuzza's[2] work as well. The problem has been implemented in MATLAB and solved using the quadprog solver.
The following choices have been made:
- $T = 40$ s;
- $r = 1$ m;
- $n = 7$-th order.



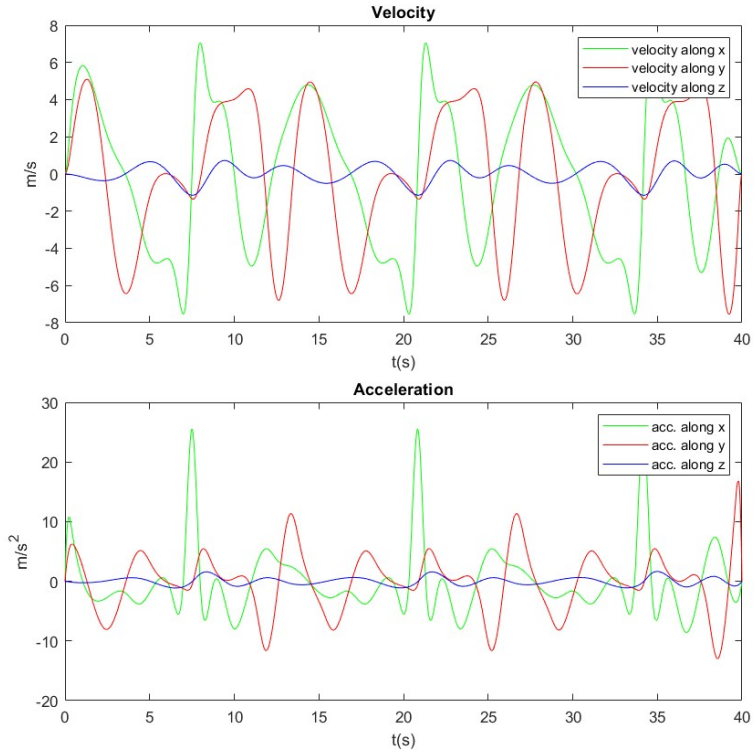
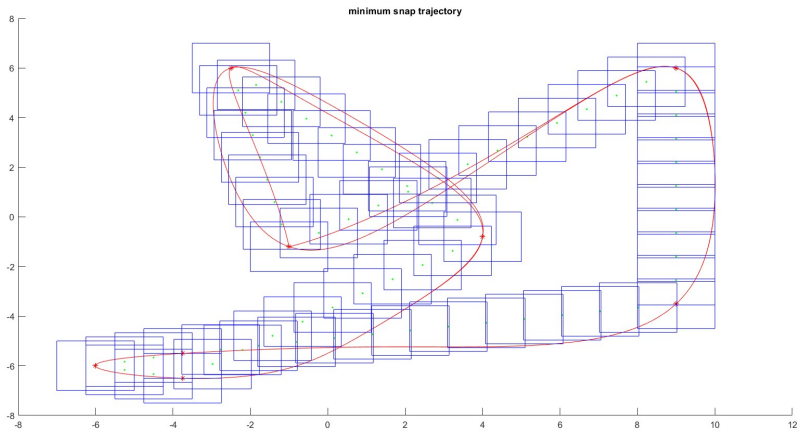


Figure 2: Position, velocity and acceleration trends obtained by considering the polynomial coefficients computed by solving the QP problem

With this configuration, the problem was solved trying to accomplish a number of three lap in the given time T. The problem was successfully solved, leading to a minimum-snap trajectory which fulfills continuity constraints on position, velocity and acceleration (whose trends are shown respectively in Fig. 2) and corridor constraints, as it can be observed in Fig. 3.
Attempts to reduce the time and the dimensions of the corridors showed how the problem is extremely sensitive to the choice of:

- The total time T;
- The semi-edge of the cube r;
- The polynomial order n.



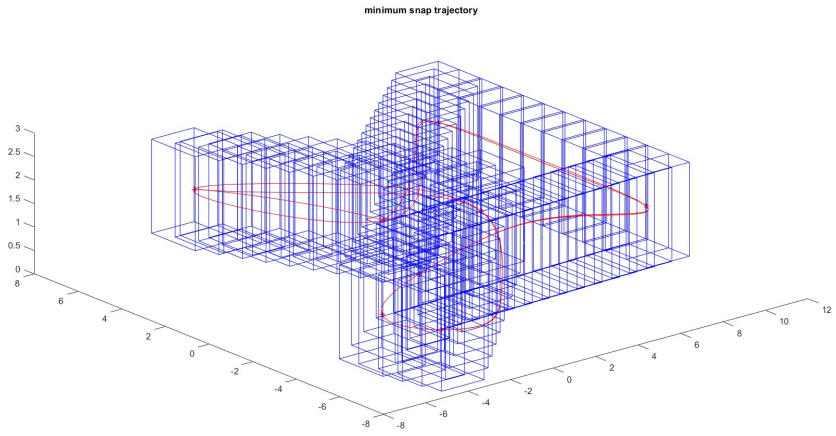


Figure 3: 3D trajectory obtained by considering the polynomial coefficients computed by solving the QP problem, respectively top view (top) and isometric view (down). In blue the cube resulting from the inequality constrains.

Reducing the semi-edge allowed to satisfy stricter inequality constraints, but such problems were not minimum-snap ones anymore, so such solutions have not been considered in the prosecution of this work.

3 Control Approaches and Results

Once the trajectory to be tracked has been generated, the focus switches onto the development of the controller for the quadcopter. For this reason, some control strategies have been developed and compared.

3.1 Nonlinear MPC

One of the possible approaches for this work is a NMPC: as a traditional linear MPC, the NMPC compute the control actions to be applied to the system at each control interval by using a combination of model-based prediction and constrained optimization. The key differences are:

- The use of a nonlinear prediction model, which could eventually also include time-varying parameters;
- The equality and inequality constraints can be nonlinear;
- The scalar cost function to be minimized can be a nonquadratic function of the decision variables, linear or nonlinear.

Before delving into the description of the structure of the NMPC, it is worth recalling the characteristics of the system and its representation in terms of equations, as they won't be the same for each control strategy

3.1.1 Dynamic Model of the Quadcopter

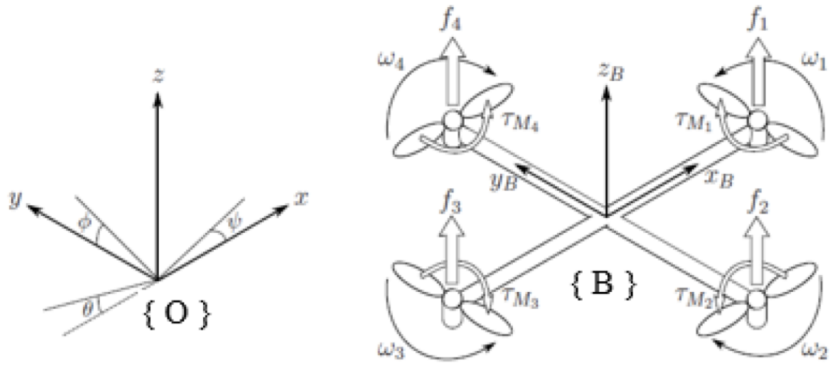


Figure 4: Scheme for the quadcopter dynamics

Defined f_i ,$i = 1 \dots 4$ as the thrust exerted by the quadcopter's i-th propeller and d the distance of the propellers from the centre of the drone, by referring to Fig. 4 and by considering positive anticlockwise rotations, the roll rotation around the body axis x_B results in:

$$\tau_x = d(f_4 - f_2)$$

Similarly for the pitch rotation around the body axis y_B:

$$\tau_y = d(f_3 - f_1)$$

For what regards instead the yaw rotation around the body axis z_B, the latter is determined by an unbalance in the aerodynamic resistance of opposed propellers, leading to:

$$\tau_z = c(f_1 - f_2 + f_3 - f_4)$$

where c is the drag coefficient. The translation along the body axes x_B and y_B are determined by the projection of the total thrust along the same axes, while the translation along z_B must take into account also the weight force. Said ϕ the roll angle, θ the pitch angle and ψ the yaw angle, the thrust component along the three body axes will be:

$$F_x = (f_1 + f_2 + f_3 + f_4)\sin\theta$$
$$F_y = -(f_1 + f_2 + f_3 + f_4)\sin\phi$$
$$F_z = (f_1 + f_2 + f_3 + f_4)\cos\phi\cos\theta - mg$$

where m and g are respectively the mass of the quadcopter and the gravitational acceleration. It's possible to define the state vector as follows:

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \phi \\ \theta \\ \psi \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{bmatrix}$$

while the input vector is defined as:

$$\boldsymbol{u} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

The ODE vector describing the quadcopter dynamics can be written as:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \\ \frac{\sin(x_8)}{m}(u_1 + u_2 + u_3 + u_4) \\ -\frac{\sin(x_7)}{m}(u_1 + u_2 + u_3 + u_4) \\ \frac{\cos(x_7)\cos(x_8)}{m}(u_1 + u_2 + u_3 + u_4) - g \\ x_{10} \\ x_{11} \\ x_{12} \\ \frac{d}{I_{xx}}(u_4 - u_2) \\ \frac{d}{I_{yy}}(u_3 - u_1) \\ \frac{c}{I_{zz}}(u_1 - u_2 + u_3 - u_4) \end{bmatrix}$$

where I_{xx}, I_{yy} and I_{zz} are the inertia moments around the body axes.

Table 1: Quadrotor Parameters

Parameter	Value
m (kg)	0.85
d (m)	0.3
diag($[I_{xx}, I_{yy}, I_{zz}]$)	$[2.5,\ 2.1,\ 4.3]\cdot 10^{-2}$

3.1.2 Background Theory

By applying the NMPC approach, the aim is to solve an optimization problem which can be generally formulated as follows:

$$\begin{aligned} J_0^*(x(t)) = \min_{U_0} & \ p(x_N) + \sum_{k=0}^{N-1} q(x_k, u_k) \\ & \text{s.t.} \\ & x_{k+1} = f(x_k, u_k) \\ & x_k \in X \ , \ u_k \in U, \ k = 0, ..., N-1 \\ & x_N \in X_f \\ & x_0 = x(t) \end{aligned}$$

where $x_{k+1} = f(x_k, u_k)$ is a discretized version of the nonlinear system to be controlled, X is the set of values which are admittable for the states variables, U is the set of values which are admittable for the control inputs, X_f is the terminal set, x_0 is the state initial condition, $p(x_N)$ is the terminal cost and $q(x_k, u_k)$ is the stage cost.
Said T_s the sampling time, T_p the prediction horizon and T_c the control horizon, the NMPC approach can be summarized in the following repeated steps:

1. The state is sampled at time t;
2. The optimization problem is solved on the finite prediction horizon T_p and $u_t^*(x(t))$ are applied to the system;
3. Only the first T_c elements of $u_t^*(x(t))$ are applied to the system;
4. The process starts again at time $t + T_s$

The controller solves the control problem numerically, which doesn't guarantee asymptotic stability, but empirically its capable of good performances. By this context it is possible to understand the importance of simulation in order to assess the controller performance.

3.1.3 NMPC Implementation

The NMPC has been developed in MATLAB and Simulink exploiting the Model Predictive Control Toolbox [3] which provides functions and Simulink blocks in order to develop model predictive controller to simplify their adoption in simulations. Fig. 5 shows the block scheme in Simulink with the NMPC block enveloped in a closed loop control. The development of the control solution focused on the choice of the parameters regarding the prediction and control horizons, as well as the weight's gains and the boundaries.
The cost function has been defined as:

$$J = \sum_{k=0}^{N} (y_{d,k} - y_k)^T \boldsymbol{Q} (y_{d,k} - y_k) + u_k^T \boldsymbol{R} u_k$$

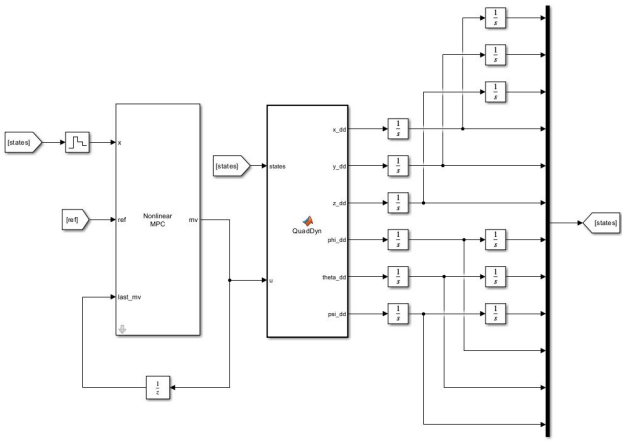


Figure 5: Simulink block scheme for the application of the NMPC controller to the quadcopter dynamic model

where the outputs of the system have been defined through a dedicated function as:

$$y_k = \begin{bmatrix} x \\ y \\ z \\ \psi \end{bmatrix}$$

The ψ reference has been computed from the references for the positions along x and y in order for the quadcopter to always face the heading direction along the trajectory. For what regards the choice of the controller parameters, it has been based on the following reasons:

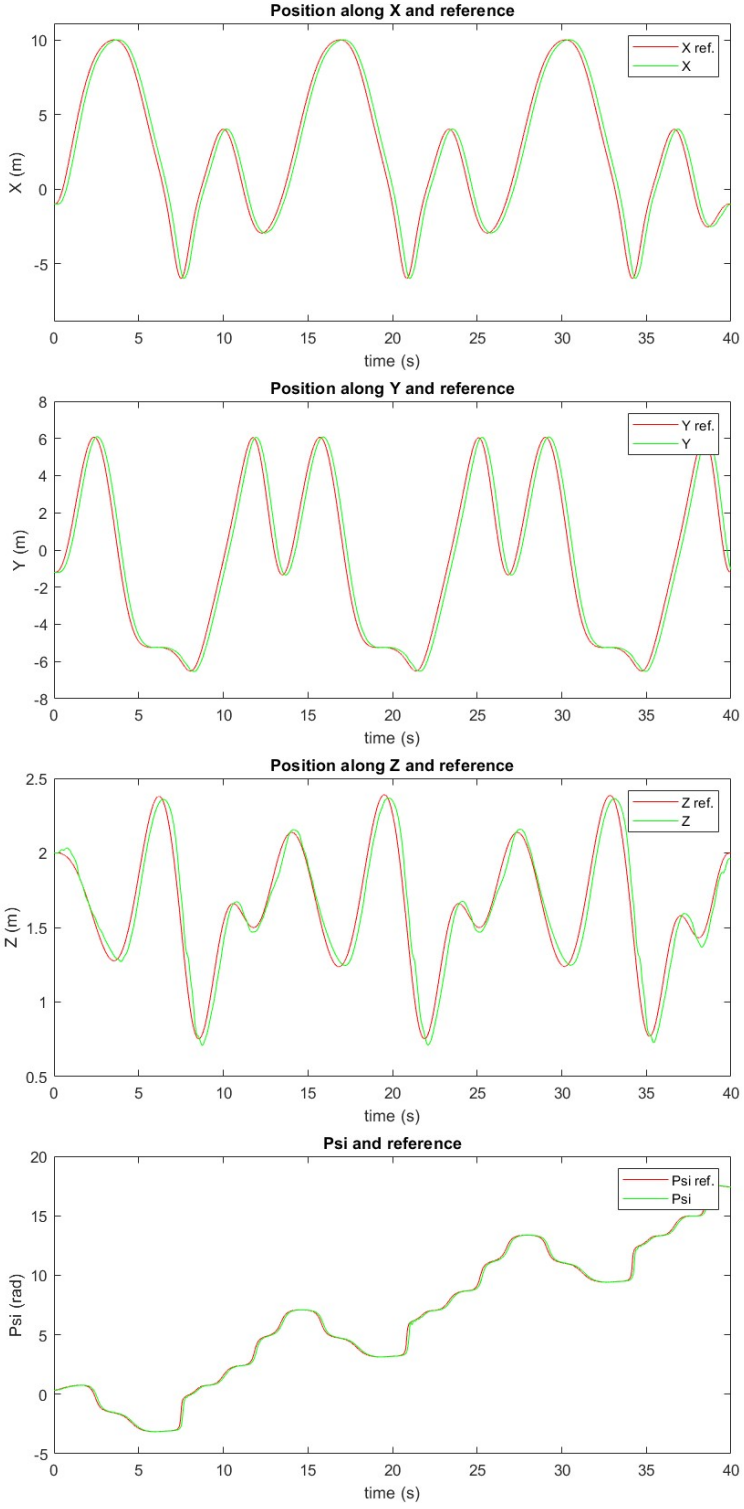
- The sampling time T_s, which is relevant due to the controller operating in discrete time. A too small T_s would lead to an unsustainable computational load due to the cost function depending upon nonlinear bounds and a considerable number of variables. Anyway, due to the subject model requiring a fast controller in order to be able to exhibit good performances, such sampling time must be small enough. A good compromise has been found which led to T_s=0.1 s;
- The prediction horizon T_p has been chosen by progressively increasing it until a further increase results in worse performance. The delay, the transient error and the exhibition of an oscillatory behavior have been mainly considered as indicators of the quality of the controller. By this approach, the prediction horizon has been chosen as $T_p = 20$;
- The control horizon T_c is generally chosen to be much smaller than the prediction one since this leads to fewer variables to compute in the cost function solved at each control interval (which fasten up computations) and promotes (but doesn't guarantee) an internally stable controller. By this considerations, the control horizon has been chosen as $T_c = 5$.

A scaling operation has been computed in order to bring all values of state, input and output variables in between 0 and 1, after which the weights matrices have been empirically chosen as:

$$\boldsymbol{Q} = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \qquad \boldsymbol{R} = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}$$

3.1.4 Results and comparison with MPCC

From the previous implementation and definition, simulations of the control were carried out, and the results are as follows:



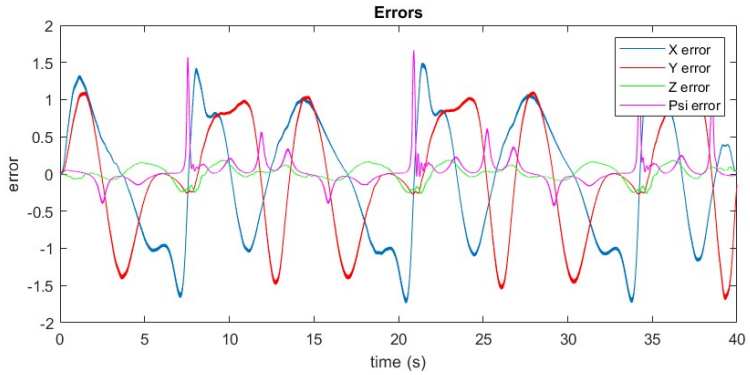


Figure 6: From up downward: position trend along x; position trend along y; position trend along z; ψ trend; position and angular error trends

As it's possible to see, Fig. 6 shows how the defined NMPC controller has been able to follow the angular and position references with a maximum transient error of about 1.5 meters, which is acceptable considering the challenging trajectory to be tracked.

Lower bounds have been defined for the control inputs and z in order for them to be always greater than zero.
All these parameters and constraints, as well as the functions defining the dynamic constraints and the output to be controlled are used to define the fields of a *nlmpc* object defined in MATLAB environment which is used to communicate such data to the controller block. (Fig. 5)

The required control inputs to obtain such results, which can be considered to be feasible and coherent with the task can be represented as follows:

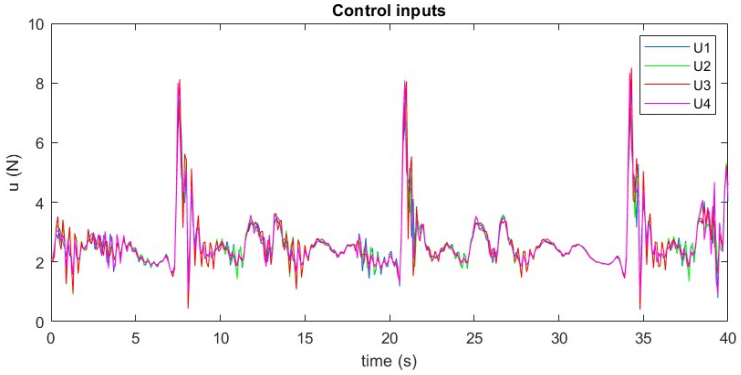
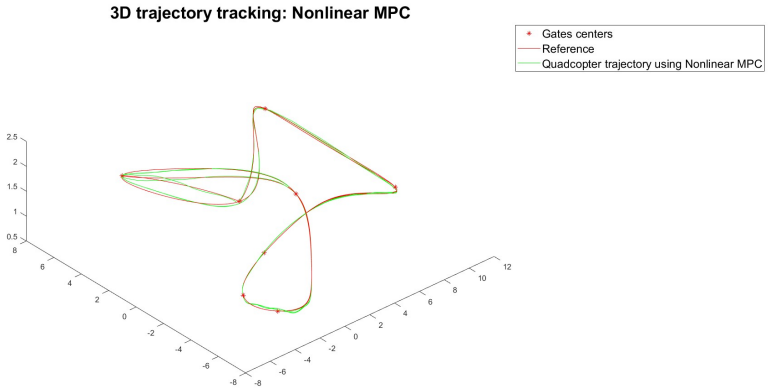


Figure 7: Control inputs trend



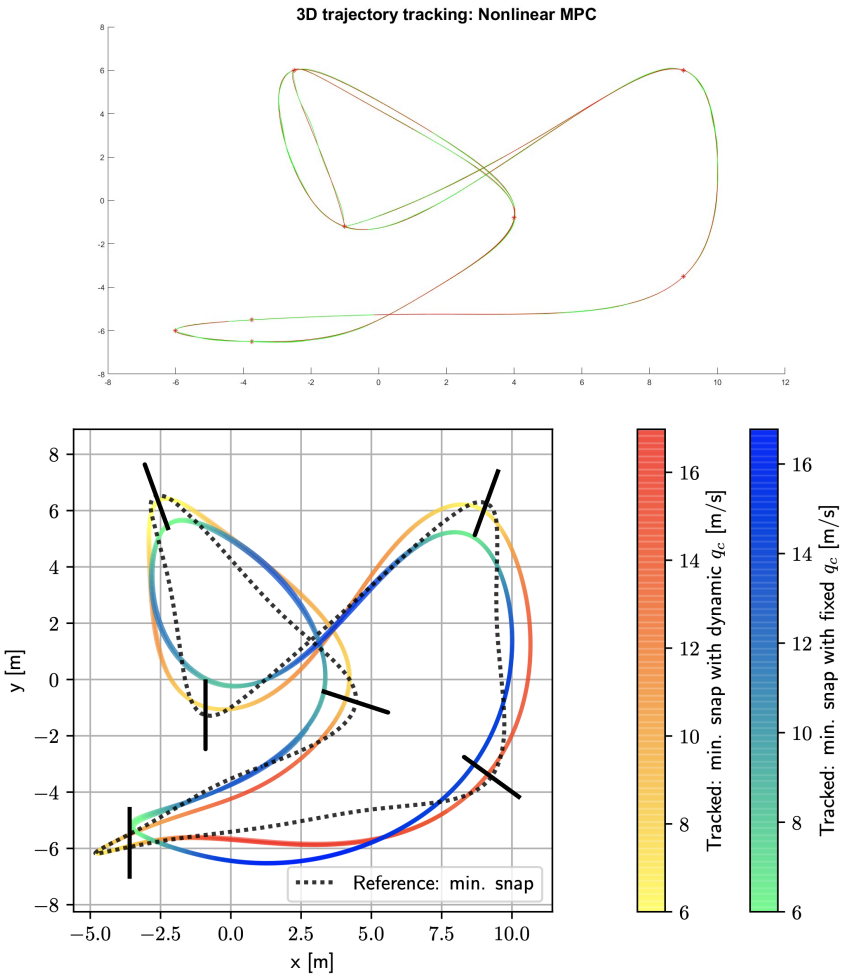


Figure 8: Comparison between the results obtained by employing the NMPC approach to track the previously generated minimum-snap constrained trajectory (top) and the results obtained by Scaramuzza employing a MPCC (down) with static contouring weight (blue-green color) and dynamic contouring weight (orange-yellow color). The minimum-snap trajectory reference followed by Scaramuzza is represented through a black dotted line.

From the figures, it's possible to see the effective 3D trajectory of the controlled drone over the 3D trajectory reference and the comparison with the results obtained by Scaramuzza in his work. It can be observed how the generated trajectories in both works are pretty similar, while there's a substantial difference in the performances of the controllers. This is due to the fact that, while the proposed NMPC tries to faithfully follow the reference, the MPCC proposed by Scaramuzza carries out both trajectory planning and tracking at the same time, noticeably changing the described path at runtime in order to minimize the lap time and maximize the progression along the path. It can be observed how by applying this approach with a static contouring weight, the MPCC fails in passing through some of the keyframes, so performing worse than the NMPC proposed in this work. On the contrary, a dynamic contouring weight ensures the quadcopter to pass through the gates but staying further from the reference in order to also achieve more and different optimal goals.

3.2 Geometric Control

Geometric Control is based on a coordinate-free dynamic model[1] using rotation matrices in SO(3) [4]. In particular, it constructs a dynamic model of the errors directly in SO(3) (where there are no singularities) and applies feedback linearization on this error model with a two-loop control structure:

- an *outer loop* for the thrust vector control that gets as inputs the desired values for linear position, velocity and acceleration from the planner as well as the actual linear position, velocity and acceleration of the system, and computes, using a PD plus gravity compensation control, plus an acceleration feedforward term, the total vertical thrust u_T, and the current direction of the z_b axis of the body frame, useful for the inner loop controller.
- an *inner loop* for the for attitude control, which gets as inputs the vector $x_{b,d}$ that stays inside the horizontal plane of the UAV (so stays in the plane containing all the propellers), the desired and current angular velocities and accelerations. The controller for the angular part is a PD+ like controller (remembering that the derivative term is not the direct derivate of the error of the proportional term) made of a proportional term for e_R (for the error on the rotation matrix), minus a proportional term for the term of the angular velocity e_ω , plus a term that compensates the one in the dynamic model, minus a term that compensate for the desired angular velocity and the desired angular acceleration.

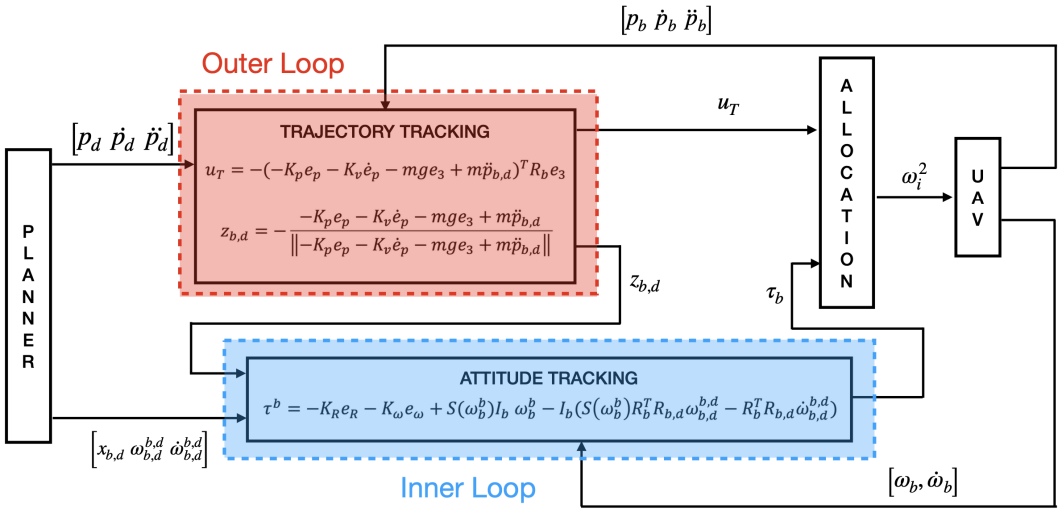


Figure 9: Geometric Control structure

3.2.1 Geometric Control Implementation

The implementation of the geometric control was carried out in Simulink environment[2] and the scheme follows exactly the one shown in Fig. 9.

To achieve satisfying results in terms of reference tracking, the gains K_p and K_v of the outer loop, and K_R and K_ω of the inner loop were tuned through empirical trials. The gains were chosen as follows[3]:

[1] The theory related to the Geometric Control has already been widely studied during the FSR course, therefore the development won't be discussed again, but just cited

[2] The control scheme from FSR course, homework n.3, was used as the starting model

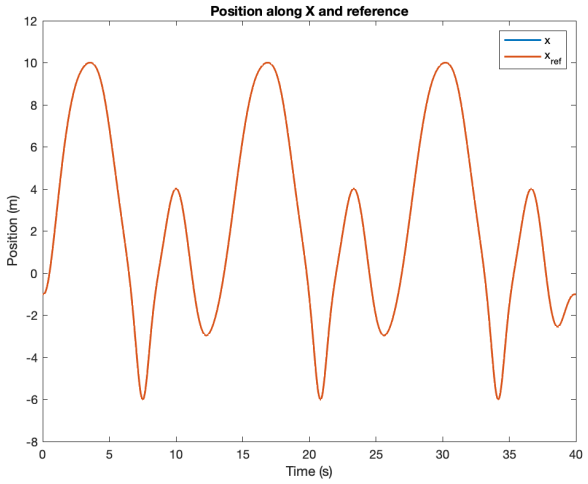
[3] The gains were chosen using a trial and error method to obtain good performance and do not necessarily represent the optimal choice

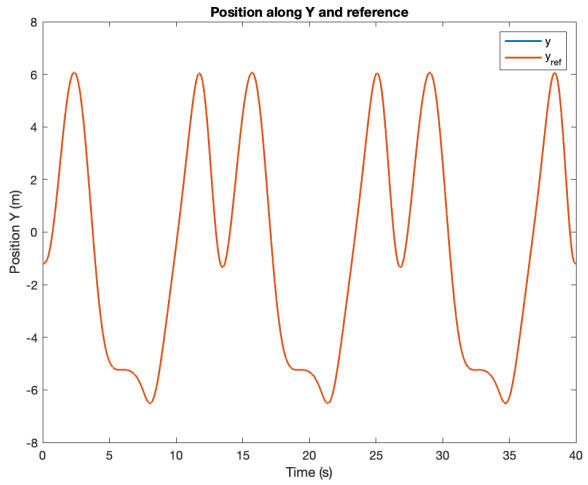
$$K_p = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 20 \end{bmatrix} \qquad K_v = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

$$K_R = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 20 \end{bmatrix} \qquad K_\omega = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

3.2.2 Results

From the previous implementation and definition, simulations of the control were carried out, with the following results:





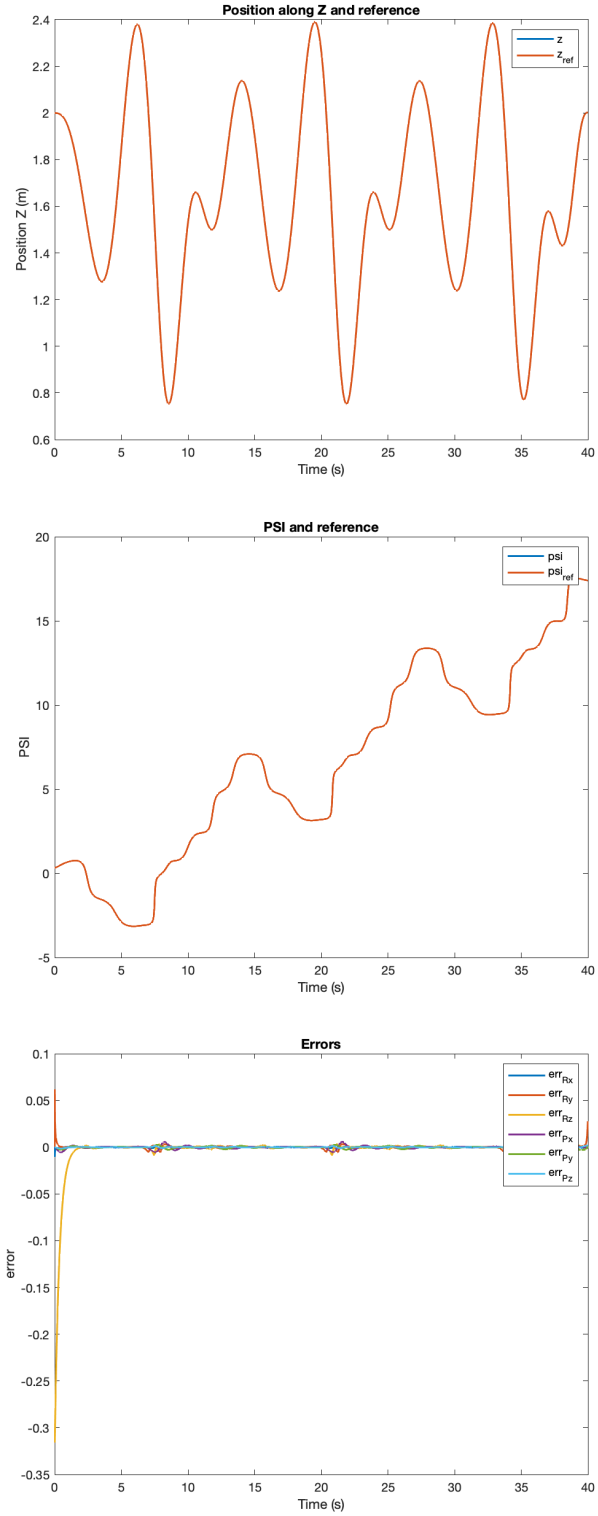


Figure 10: From up downward: position trend along x; position trend along y; position trend along z; ψ trend; position and angular error trends

As it's possible to see, Fig. 10 shows how the Geometric controller has been able to perfectly follow the references with a maximum transient error for the position of about 3 centimeters, which is a great performance.

The obtained results demonstrate that this type of control is very versatile and suitable for dynamic and acrobatic maneuvers due to better handling of large initial attitude errors. For this reason,

making a comparison with the NMPC control, the geometric one performs much better as it's capable to track the reference, even when this changes very fast.

The required control inputs to obtain such results, can be represented as follows:

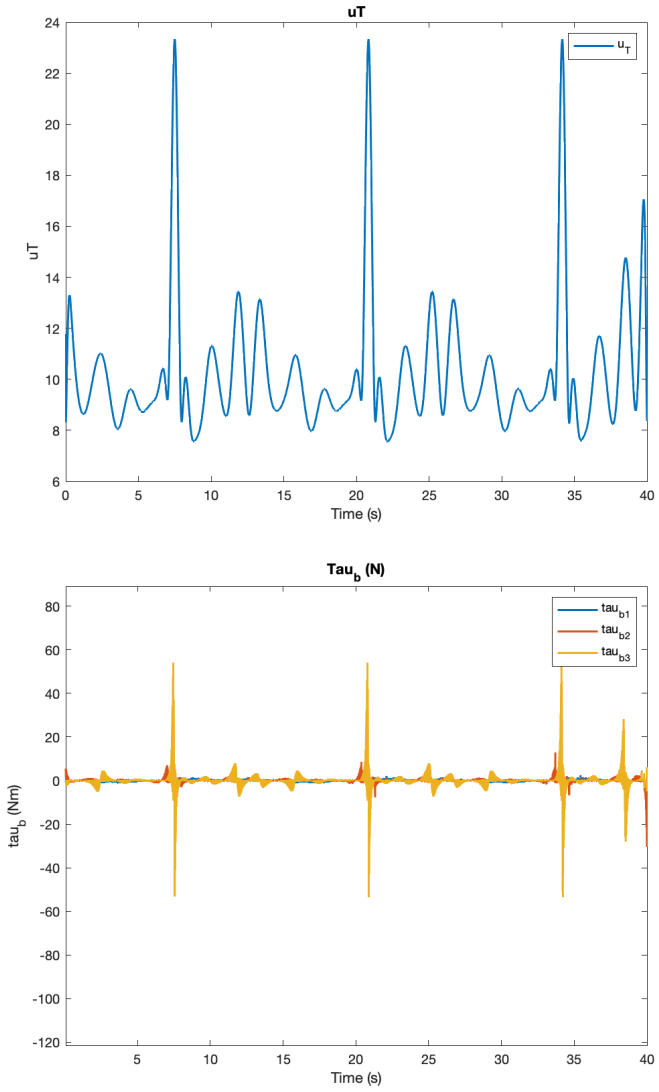


Figure 11: Control inputs trend

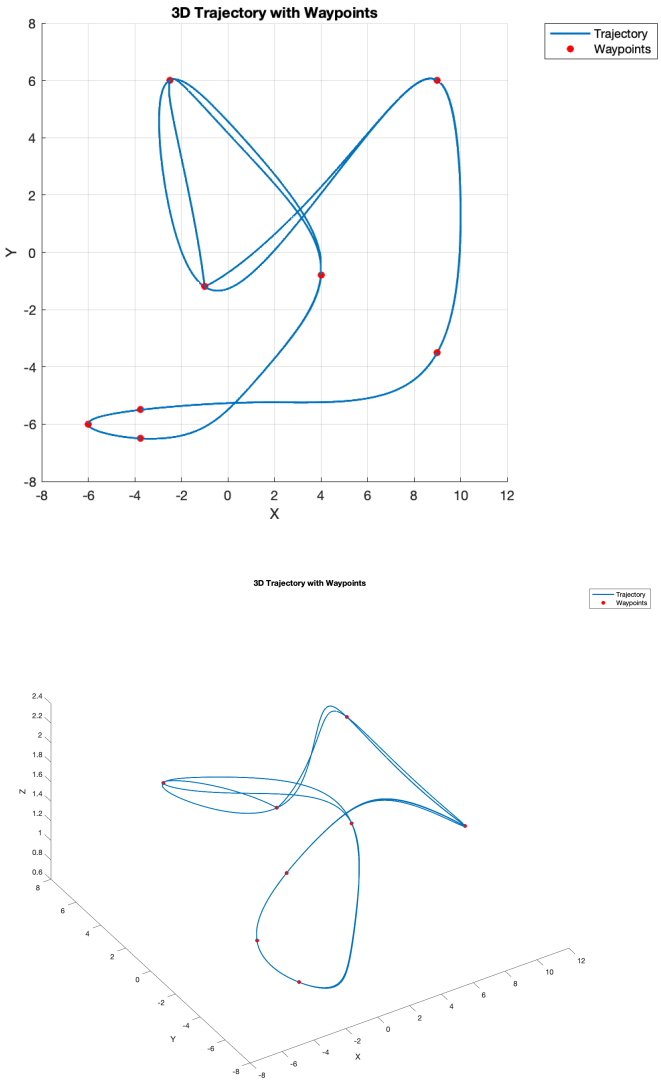


Figure 12: Trajectory obtained by employing the Geometric Control approach to track the generated minimum-snap constrained trajectory

As it's possible to see, the effective 3D trajectory is almost the same of the one generated by the solution of the minimum-snap optimization problem. From Fig. 11 it can be observed how, in order to keep those high performance, the required control inputs in terms of total thrust and torque is noticeable higher than the NMPC controller, so it's more demanding, which could represent a problem in terms of power consumption of the drone.

3.3 Hierarchical Control

Hierarchical Control relies on the RPY (roll, pitch, yaw) dynamic model [4] and uses a two-loop structure where[4]:

- the *outer loop* for position control retrieves the desired values of linear position, velocity and acceleration from the planner and the current linear position and velocity of the system and

[4]The theory related to the Hierarchical Control has already been widely studied during the FSR course, therefore it won't be discussed again, but just cited

the desired yaw angle, and computes, using a PD control plus a feedforward action for the desired linear acceleration the total vertical thrust u_T to be sent to the drone and also the desired values for ϕ and θ useful for the angular control;

- the *inner loop* for attitude control retrieves the desired values for the angles and computes the desired values for the angular position, velocity and acceleration. Then, through a PD control plus a feedforward action for the desired angular acceleration and a feedback linearization it computes the control torques τ_b to be sent to the drone.

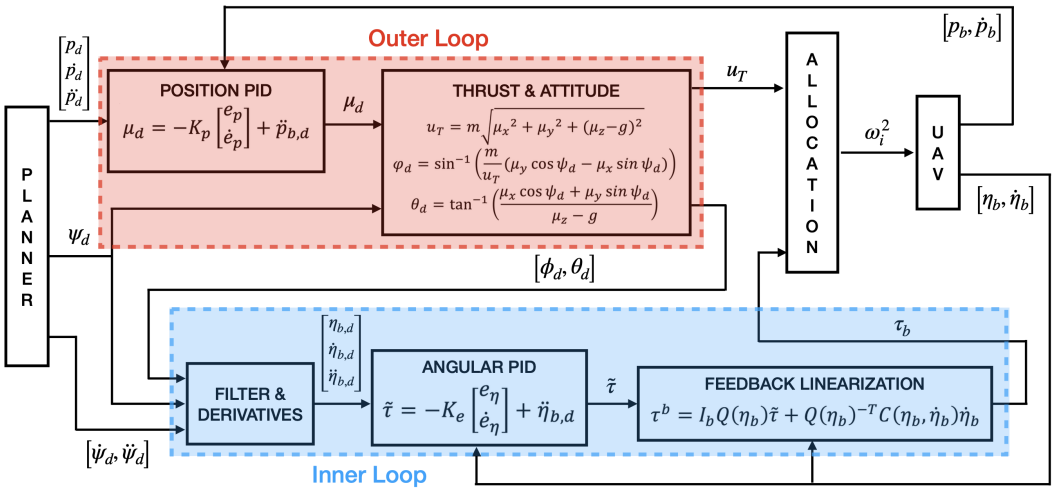


Figure 13: Hierarchical Control structure

3.3.1 Hierarchical Control Implementation

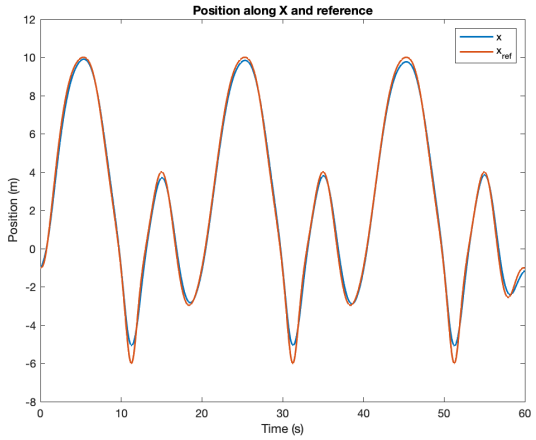
The implementation of the hierarchical control was carried out in Simulink environment and the scheme follows exactly the one shown in Fig. 13. To achieve satisfying results in terms of reference tracking, the gains K_p and K_v of the outer loop for the position control, and the gains K_R and K_ω of the inner loop for attitude control were tuned through empirical way. After some simulations, the gains were chosen as follows [5]:

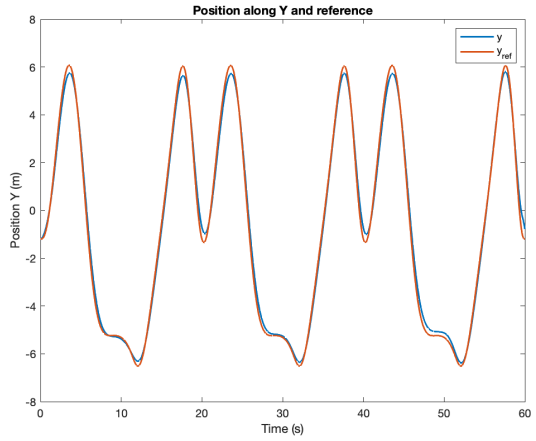
$$K_p = \begin{bmatrix} 12 & 0 & 0 \\ 0 & 12 & 0 \\ 0 & 0 & 18 \end{bmatrix} \qquad K_v = \begin{bmatrix} 0.7 & 0 & 0 \\ 0 & 0.7 & 0 \\ 0 & 0 & 2.5 \end{bmatrix}$$

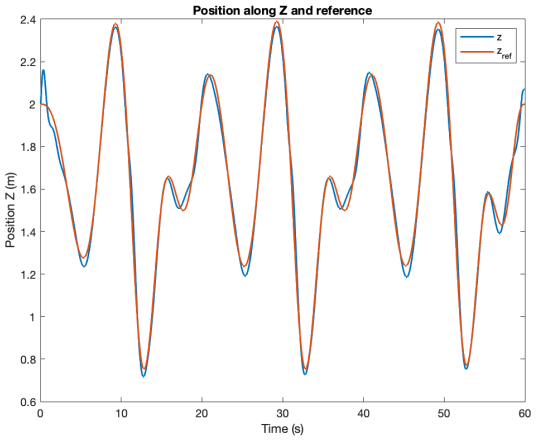
$$K_R = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix} \qquad K_\omega = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 15 \end{bmatrix}$$

[5] The gains tuning has been achieved after increasing the trajectory duration, as will be better explained in section 3.3.2

3.3.2 Results


Position along X and reference
x
x_ref
Position (m)
Time (s)


Position along Y and reference
y
y_ref
Position Y (m)
Time (s)


Position along Z and reference
z
z_ref
Position Z (m)
Time (s)

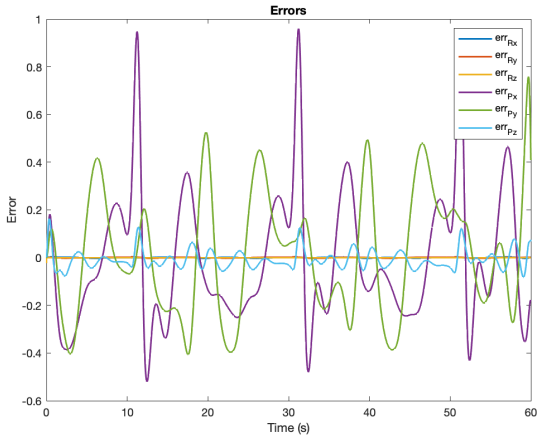


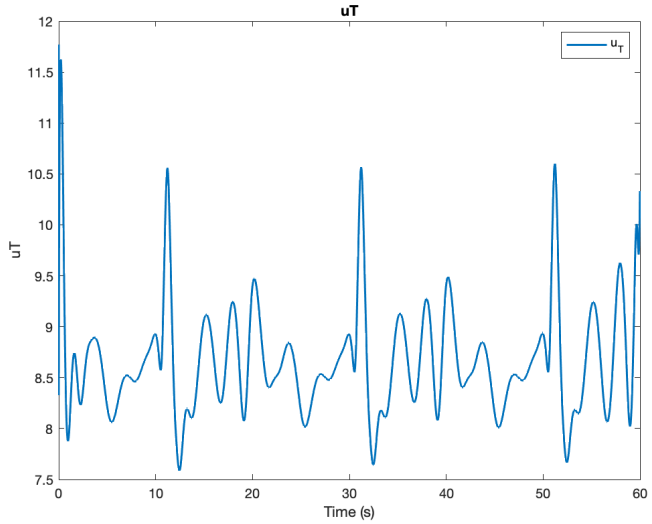
Figure 14: From up downward: position trend along x; position trend along y; position trend along z; ψ trend; position and angular error trends

From the results obtained, it is important to specify that, after a large number of simulations, as expected from what studied during the FSR course of the hierarchical control and its limitations, no configuration made the trajectory admissible and trackable by the controller. This because, unlike Geometric Control, which is very robust and avoids singularities (making it ideal for applications like racing), hierarchical control is better suited for stationary or hovering operations where small movement angles are more common, which is not the case in this type of task. Consequently, this type of control cannot guarantee the same performance as the previous ones. This problem arises when dealing with sudden changes in the system dynamics, such as high accelerations, tight turns, or extreme angles, where the model can easily encounter singularities.

For the reason just described, to make this type of control functional, it was necessary to scale the trajectory increasing the duration time. From the positive results obtained, the drone manages to complete the track 20 seconds longer than with the previous controls.

As can be seen, Fig. 14 shows how the hierarchical control was able to follow the position and angle references with a maximum error of about 1 meter.

The required control inputs to obtain such results, which can be considered feasible and coherent with the task, can be represented as follows:



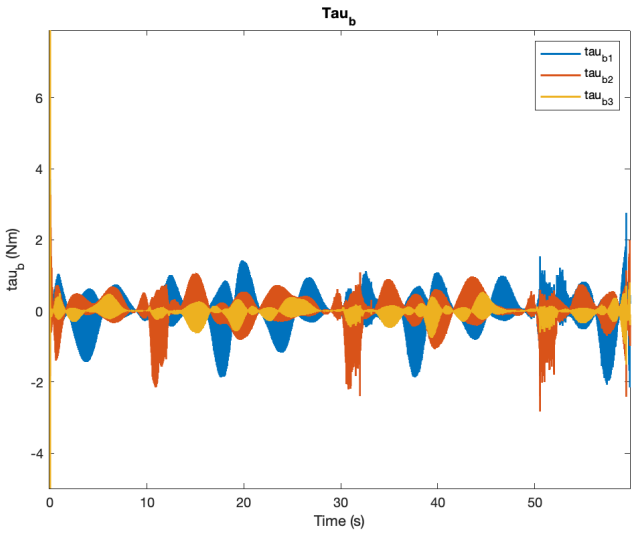
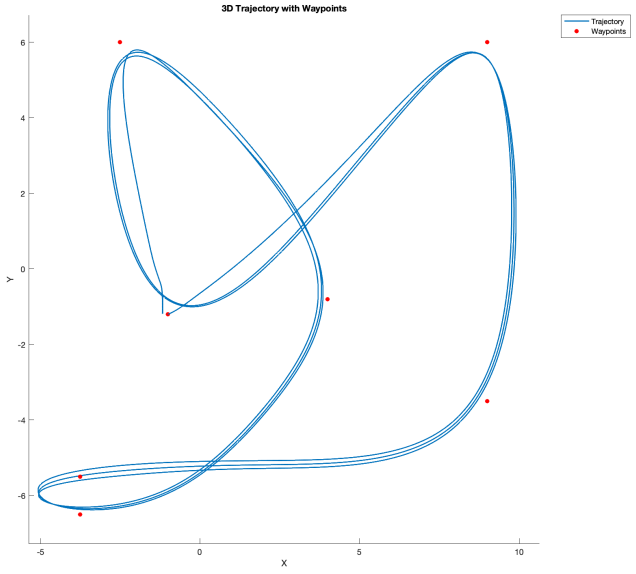


Figure 15: Control inputs trend



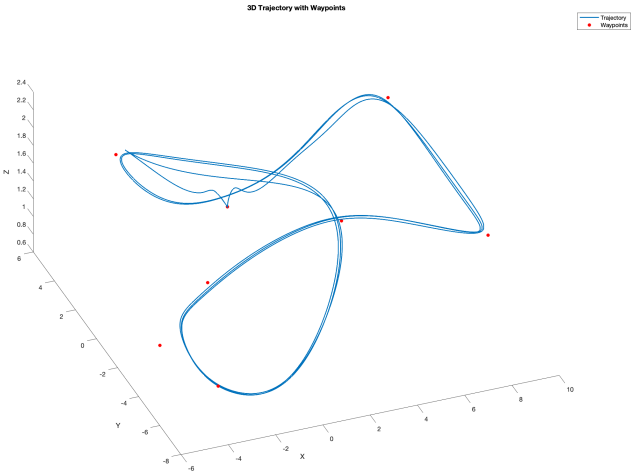


Figure 16: Trajectory obtained by employing the Hierarchical Control approach to track the generated minimum-snap constrained trajectory in a longer time

From the figures, it is evident that the drone was not able to pass through all the generated waypoints, and thus the final trajectory is slightly modified. This is due to the fact that this type of control struggles with changes in direction, which is precisely where the waypoints are positioned, making this controller unsuitable for racing drones.

4 Conclusions

This report proposes a method to generate a minimum-snap optimal constrained trajectory and some controllers such as the Nonlinear Model Predictive Control, Geometric and Hierarchical Control formulations in the framework of full quadcopter dynamics.

The proposed work is compared to an advanced and innovative Model Predictive Contouring Control, highlighting the differences between the approaches. The results from this work aims to show the ongoing developments in the field of autonomous drone racing and to stimulate the reader inventive by highlighting how such context can be extremely challenging. Even considering the innovative results obtained by other reasearchers used as comparison in this report, classical control theory can still perform very well and there are still challenges ahead mainly related to onboard state estimation and perception at high speeds. The proposed controllers try to prove to be stable approaches to fly very aggressive and collect maneuvers at very high speeds.
Finally, the results shown in this work can be successfully applied in other applications in the field of Field and Service robotics for exploration purposes and on other kind of robots. Further developments could lead to the integration of sensor-retrieved data in order to carry out an on-line optimal trajectory generation.

5 Acknowledgements

The author Salvatore Granata wants to thank the Professor Eng. Fabio Ruggiero for pushing him into such innovative and still unexplored field, confronting with expert works, as this has enabled him to gain new awareness of the challenges offered by the Robotics field, application and cutting-edge skills.

References

[1] D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors,” in IEEE Int. Conf. Robot. Autom. (ICRA), 2011.

[2] Romero, Angel & Sun, Sihao & Foehn, Philipp & Scaramuzza, Davide, 2021. Model Predictive Contouring Control for Near-Time-Optimal Quadrotor Flight

[3] MathWorks. Nonlinear MPC, https://it.mathworks.com/help/mpc/ug/nonlinear-mpc.html

[4] Fabio Ruggiero, Aerial Robotics Chapter, FIELD AND SERVICE ROBOTICS SLIDES, 2024