UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria dell'Automazione e Robotica

Field and Service Robotics

HOMEWORK 3

Academic Year 2024/2025

Relatore
Ch.mo prof. Fabio Ruggiero
Candidato
Emmanuel Patellaro
P38000239

Exercise n°1

Let us consider the *Octocopter* in Figure 1.

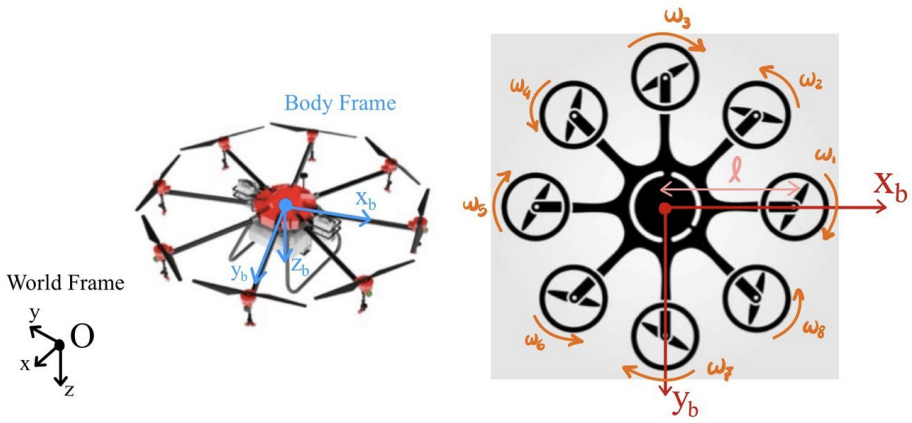

Figure 1: Octocopter

It is composed by 8 rotating propellers (1 link + 1 rotational joint each). So it has **14 DoFs**:

- 8 given by the propellers
- 6 given by the spatial rigid body

So, the **Configuration Space Topology** for this robot can be written as

$$T^8 \times \mathbb{R}^3 \times S^2 \times S^1$$

The case of the drone fixed to the ground is not of great interest, as it does not have any manipulator that would allow it to interact with the environment. Moreover, in this case, the drone would only have the 8 degrees of freedom due to the rotation of the propellers, with a configuration space topology T^8.
"Is the system underactuated?": **YES**, the system is **Underactuated** because all the propellers are co-planar and so aligned to the same plane. Because of this, basically, there are *8 forces* always aligned and perpendicular to the plane itself, and so I can have accelerations only in this direction but not, for example, laterally. Furthermore, as explicitly highlighted in *Homework 1 (Exercise 3, point c.)*, this condition holds regardless of the number of propellers.
A Fully Actuated system is possible, for example, in the case of *tilting propellers*.

Finally, let us derive the **Allocation Matrix** for the drone considering the *Body Frame* and the *World Frame* according to the *NED* convention, as can be easily seen in Figure 1. In order to not introduce any moment that can destabilize the drone, an alternation in the rotation direction of the propellers is necessary. This is just to not have *aereodynamics effects* that can tilt the platform but, at the end, the *thrust* force will always positive and aligned with respect to $z^b_{p_i} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$[1]. Using

$$f^b = \sum_{i=1}^{n} c_f \left|\omega_i\right| \omega_i z^b_{P_i} \qquad \tau^b = \sum_{i=1}^{n} \left|\omega_i\right| \omega_i (-k_i c_m z^b_{P_i} + c_f S(p^b_{P_i}) z^b_{P_i})$$

with $c_f > 0$ and $c_m > 0$ respectively the *thrust constant* and the *drag factor* and $k_i = 1$ for descending chord and $k_i = -1$ for ascending chord, it is possible to compute the *Allocation Matrix* G_q such that [2]:

$$\begin{bmatrix} f^b \\ \tau^b \end{bmatrix} = f_u(\alpha_i, \beta_i, u_{\omega_i}) \Rightarrow \begin{bmatrix} u_T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = G_q u_\omega$$

Let us find f_z, τ_x, τ_y and τ_z, thanks to the right-hand rule, in this way:

$$f_z = u_T = -(f_{P_1} + f_{P_2} + f_{P_3} + f_{P_4} + f_{P_5} + f_{P_6} + f_{P_7} + f_{P_8})$$

$$\tau_x = l(\frac{\sqrt{2}}{2} f_{P_2} + f_{P_3} + \frac{\sqrt{2}}{2} f_{P_4} - \frac{\sqrt{2}}{2} f_{P_6} - f_{P_7} - \frac{\sqrt{2}}{2} f_{P_8})$$

$$\tau_y = l(\frac{\sqrt{2}}{2} f_{P_8} + f_{P_1} + \frac{\sqrt{2}}{2} f_{P_2} - \frac{\sqrt{2}}{2} f_{P_4} - f_{P_5} - \frac{\sqrt{2}}{2} f_{P_6})$$

$$\tau_z = -(m_{P_1} + m_{P_3} + m_{P_5} + m_{P_7}) + (m_{P_2} + m_{P_4} + m_{P_6} + m_{P_8})$$

[1] There is -1 because the z axis of a propeller is always pointing upwards, while the Body Frame is a NED Frame and so with the z axis pointing downwards

[2] For co-planar propellers it is always $\alpha_i = \beta_i = 0$. So, to f_x and f_y correspond 0 rows in the Allocation Matrix and $f_z = u_T = \sum_{i=1}^{4} f_{P_i} > 0$

At the end, the *Allocation Matrix* $G_q \in \mathbb{R}^{4x8}$ is:

$$G_q = \begin{bmatrix} -c_f & -c_f & -c_f & -c_f & -c_f & -c_f & -c_f & -c_f \\ 0 & \frac{\sqrt{2}}{2}lc_f & lc_f & \frac{\sqrt{2}}{2}lc_f & 0 & -\frac{\sqrt{2}}{2}lc_f & -lc_f & -\frac{\sqrt{2}}{2}lc_f \\ lc_f & \frac{\sqrt{2}}{2}lc_f & 0 & -\frac{\sqrt{2}}{2}lc_f & -lc_f & -\frac{\sqrt{2}}{2}lc_f & 0 & \frac{\sqrt{2}}{2}lc_f \\ -c_m & c_m & -c_m & c_m & -c_m & c_m & -c_m & c_m \end{bmatrix} \quad (1)$$

Exercise n°2

Let us describe the differences between the **Ground Effect** and the **Ceiling Effect**. In *Aerial Robotics* there are a lot of applications, especially related to *inspection* and *maintenance*, that require the UAVs and UAMs to stay close to *surfaces* and to the *environment*. Of course, this can affect the flight of the drone and its rotors. Two of the main effects caused by flight not in Free-Space are precisely these two.

In order to model the effect of different surfaces like the ground or the ceil, the **method of images** is employed. Thanks to this method, we suppose to have, virtually, the same propellers placed at the same amount of distance but under the ground in the first case or over the ceil in the second case. In both cases we want to understand what is the effect of the *air flow* when the robot is close to the two surfaces.

In the case of the **Ground Effect** the particles of air hit somewhere the ground and come back to the drone because of the fact that it is very close to the surface and because of the high velocities of the propellers. So, the high-speed airflow generated by the propellers, directed downward, once in contact with the ground, returns upward. Then, once the propellers are reached, it produces extra lift forces, in such a way to generate a pushing force interacting with the UAV from below. Basing on the so-called *potential aerodynamic assumption* and using the *power preserving theory*, it is possible to estimate the point in which the propellers are affected by this[3]. In particular, when there are more than one propeller (in most cases), we have to consider also the effects caused by the *interaction* between them. The greater the distance between them, the lower the mutual influence will be.

[3]When $z > 2\rho$ there is no Ground Effect

Instead, in the case of the **Ceiling Effect**, the result is completely the opposite with respect the previous one. In fact, when the drone is close to the ceil, the air flow, obviously, continuously goes down, but what happens is that the propellers move the particles of the air creating a sort of **Vacuum Effect**. In this last case, the propellers start to increase faster the velocity to maintain the same thrust and this create a *pressure* with the ceil in such a way that it is attracted to the ceil causing a possible crash with the environment. It is important to take into account this effect when it is necessary to inspect, for example, bridges or structure from below. To counteract this last effect, the solution could be to decrease the spinning of the propellers close to the ceil in such a way to maintain the same thrust. In this way, in addition to avoiding the Vacuum Effect, it is also possible to save battery.

Now let us show some formulas[4]:

$$\frac{T_{IGE}}{T_{OGE}} = \frac{1}{1 - (\frac{\rho}{4z})^2}$$

$$\frac{u_{T,IGE}}{u_{T,OGE}} = \frac{1}{1 - (\frac{\rho}{4z})^2 - \frac{z\rho^2}{\sqrt{(d^2+4z^2)^3}} - \frac{z\rho^2}{2\sqrt{(2d^2+4z^2)^3}}}$$

$$\frac{T_{ICE}}{T_{OCE}} = \frac{1}{1 - \frac{1}{k_1}(\frac{\rho}{z+k_2})^2}$$

where T is the thrust of the rotor, ρ is the radius of the propeller and z is the height of the propeller from the ground or, in the second case, the distance between the drone and the ceil.

Furthermore, **other considerations** can be made. In fact, other cases with significant impact on the drone's performance may also arise, as can be seen from the figure below. One example could be the presence of a *manipulator arm* under the drone itself which could serve, if positioned below the propellers, the same function as a *ground*.

[4] IGE=Inside the Ground Effect, OGE=Outside the Ground Effect, ICE=Inside the Ceiling Effect, OCE=Outside the Ceiling Effect

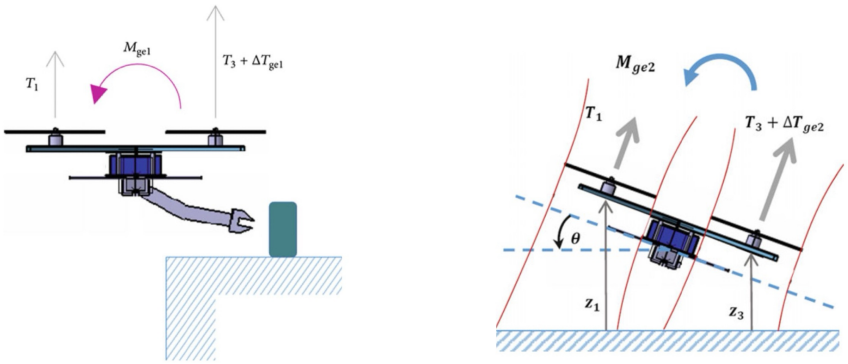

Figure 2: Particular Cases for Ground Effect (for the other effect is the same but with a ceil)

Alternatively, it is possible that a part of the propellers are positioning over (or under in the case of the Ceiling Effect) a surface, while the other part not, causing a *moment around the center of mass* of the UAV. In this last condition, only for the *Ground Effect*, this disturbance tries to stabilize the drone (for the other one the destabilization is increased).
At the end, another strange behavior can be noticed in small indoor place, where the recirculation of the air from the ground to the ceil can push down the UAV after a certain amount of time. This is called **Indirect Ground Effect**.
To conclude, to ensure that the drone behaves as desired, these effects must be taken into account in order to avoid unexpected behaviors that could interfere with the robot's performance, potentially causing irreversible damage in some cases. To do this, one can apply either a **Model-Based Approach** or a **Data-Driven Approach**, depending on the requirements and available resources.

Exercise n°3

Let us implement the **Momentum-Based Estimator** of order r to estimate the *external disturbances* acting on the UAV during the flight considering the given workspace *ws_homework_3_2025.mat*.
Let us start from the *RPY Quadrotor Dynamic Model*[5] with external wrench

[5] C is the Coriolis-like Matrix, Q is the Transformation Matrix, R_b is the Rotation Matrix from the Body Frame to the World Frame and $\eta_b = \begin{bmatrix} \varphi & \theta & \psi \end{bmatrix}^T$

disturbance given by f_e and τ_e

$$\begin{cases} m\ddot{p}_b = mge_3 - u_T R_b e_3 + f_e \\ M(\eta_b)\ddot{\eta}_b = -C(\eta_b, \dot{\eta}_b)\dot{\eta}_b + Q^T(\eta_b)\tau^b + \tau_e \end{cases}$$

These two lumped terms are due to various reasons like *uncertainties of the model*, *interaction with the environment* and other stuff. Considering the generalized **Momentum Vector** $q = \begin{bmatrix} mI_3 & O_3 \\ O_3 & M(\eta_b) \end{bmatrix} \begin{bmatrix} \dot{p}_b \\ \dot{\eta}_b \end{bmatrix} \in \mathbb{R}^6$ and its time derivative $\dot{q}$ the estimator's goal is to estimate the **Wrench Disturbance** $\begin{bmatrix} f_e \\ \tau_e \end{bmatrix}$ through $\begin{bmatrix} \hat{f}_e \\ \hat{\tau}_e \end{bmatrix}$[6]. Let us consider the **r-order transfer function**

$$G(s) = \frac{k_0^r}{(s + c_0)^r}$$

choosing $k_0 = c_0$ in such a way to obtain $G(0) = 1$ and choosing a c_0 in order to gather as much information as possible. In fact, it is a low-pass filter, but the goal is not to filter, rather to *extract information*. So the cutoff frequency has to be as high as possible, but not too high, otherwise also *noisy signals* from the *IMU* are captured, in particular if this is a very cheap one. For this reason, the value $c_0 = 10$ has been considered. For the r^{th}*-order estimator* the important formulas are the following and they are implemented with a recursive MATLAB Script

$$\sum_{i=0}^{r} \prod_{j=-r}^{-(i+1)} K_{-j} \frac{d^i}{dt^i} \begin{bmatrix} \hat{f}_e \\ \hat{\tau}_e \end{bmatrix} = \prod_{i=1}^{r} K_i \begin{bmatrix} f_e(t) \\ \tau_e(t) \end{bmatrix} \qquad \prod_{i=j+1}^{r} K_i = c_j \quad j = 0, ..., r-1$$

$$\begin{cases} \gamma_1(t) = K_1 \left(q - \int_0^t \begin{bmatrix} \hat{f}_e \\ \hat{\tau}_e \end{bmatrix} + \begin{bmatrix} mge_3 u_T R_b e_3 \\ C^T(\eta_b, \dot{\eta}_b)\dot{\eta}_b + Q^T(\eta_b)\tau^b \end{bmatrix} dt \right) \\ \gamma_i(t) = K_i \int_0^t - \begin{bmatrix} \hat{f}_e \\ \hat{\tau}_e \end{bmatrix} + \gamma_{i-1} dt \qquad i = 2, ..., r \end{cases}$$

It is important to highlight that, in the presence of such disturbances, one needs to be a little bit more *careful* with the performance of both the **Inner** and **Outer Loop**, because, since this compensation is necessary, if the estimator is slowed down, the inner loop must also be slowed down, and consequently the outer loop

[6]The Laplacian Domain is considered in order to simplify things because of the Linear Relationship given by the Laplacian Transform between the external wrench and its estimation $\mathcal{L}\left[\begin{bmatrix} \hat{f}_e \\ \hat{\tau}_e \end{bmatrix}\right] = G(s)\mathcal{L}\left[\begin{bmatrix} f_e \\ \tau_e \end{bmatrix}\right]$

as well in cascade. So, in the end, what is effectively being done is *degrading the controller's performance* with a longer convergence time, and it is not possible to demand very fast movements from the planner. Let us show some results. Comparing the estimation results with **different values of r** (many examples are included in the folder associated with this project), it was possible to observe that, **the lower the order of the filter, the shorter the settling time**; however, obviously, **the more abrupt the transient and the greater the overshoot**. This overshoot almost completely disappears around the value of $r = 35$, with a settling time of $5s$. For higher values, the behavior remains smooth as before, but only the settling time is extended. But, also for some previous values the overshoot is very low.

So, it is possible to say that, **the value of r from which the estimation results do not improve too much is r=5, with a settling time to 1% $\mathbf{T_{s,1}}$ equal approximately to 2.75s**. The *evaluation criterion*, used to verify this value of r, was to consider a **maximum overshoot** of **5%**. In fact, for **r=5**, the overshoot obtained is **4.92%**, while, for the previous value **r=4**, it is **5.31%**.

Instead at a value of **r=90**, the estimator begins to show *strange steady-state behaviors* and starts having small *oscillations*; oscillations that are *exponentially* amplified as the order increases. For **r=110**, as can be seen from the figures below, the oscillations tend to reach values even an order of magnitude higher than the actual disturbance affecting the system.

Furthermore, **by increasing the value of $\mathbf{c_0}$, the estimation will be faster but with more overshoot and a less smooth transient**. These results will not be shown.

In the MATLAB Script, also an implementation with a Transfer Function with a specific settling time has been made (commented) (not shown in the following but useful if it is necessary to control the settling time of the estimation).

Let us compare the obtained estimation with the **disturbances of 1N along the x- and y- axis of the World Frame**, and of **-0.4Nm around the Yaw Axis**. Also the **z** one is shown for the next point of the exercise (for the other, the values are very small and so there are no disturbances).

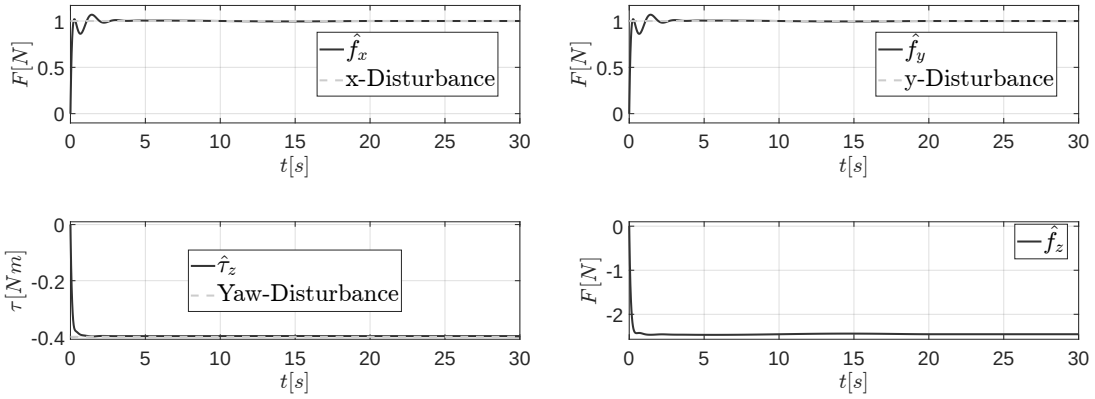

Figure 3: Wrench Estimation Comparison with the Real Disturbances with $r = 1$

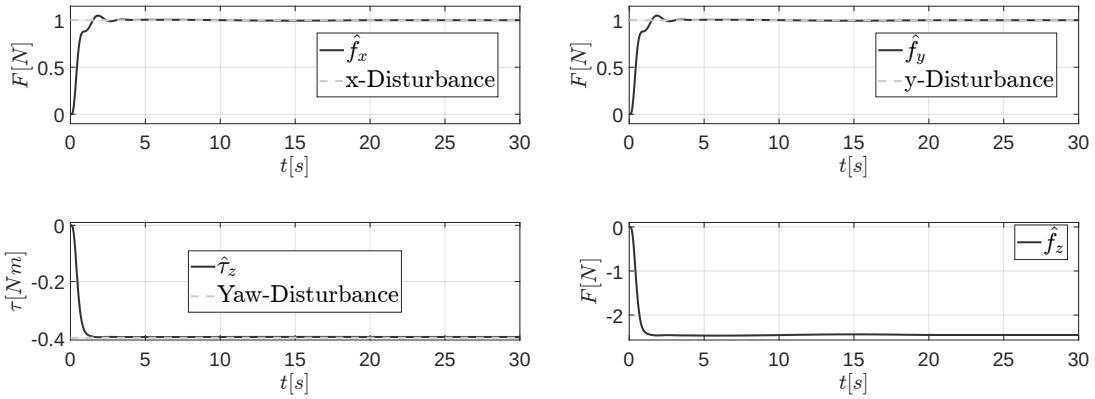

Figure 4: Wrench Estimation Comparison with the Real Disturbances with $r = 5$

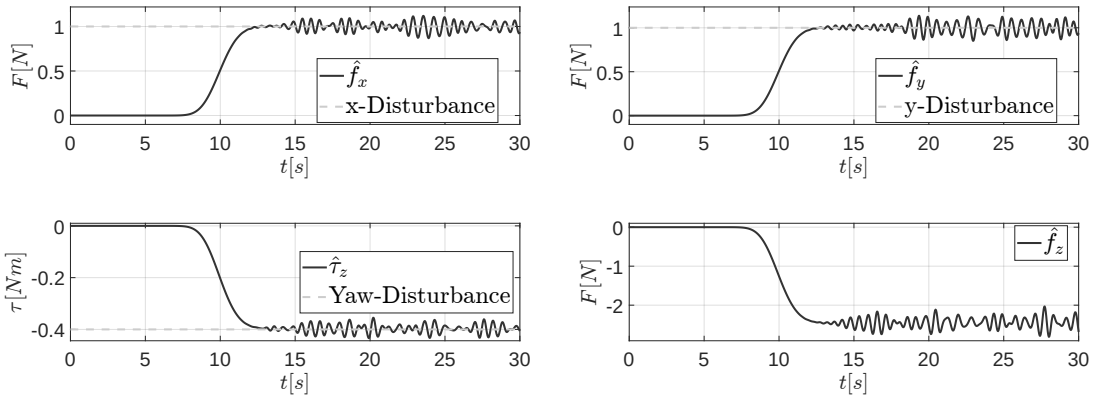

Figure 5: Wrench Estimation Comparison with the Real Disturbances with $r = 100$

Now, by studying the **estimated disturbance along the z-axis**, it is possible to compute the **real mass** of the UAV. In fact, as can be observed from

the previous figures, there is a *disturbance along the z-axis* which is therefore not caused by external disturbances but rather by **model uncertainties**, and specifically by an *incorrect estimation of the real mass of the UAV*. Thus, it is possible to compute it considering

$$\tilde{m} = \frac{\hat{f}_z}{g} \Rightarrow m_r = m_s + \tilde{m} = 1.5Kg - 0.25Kg = 1.25Kg$$

with m_s the supposed mass and m_r the real mass. So, **the real mass of the UAV, computed by the estimated disturbance along the z-axis, is 1.25Kg**

Exercise n°4

Let us implement the **Geometric Control** of a Quadcopter[7]. Neglecting the external disturbances it is necessary to consider the *Coordinate-Free Quadcopter Dynamic Model*

$$\begin{cases} m\ddot{p}_b = mge_3 - u_T R_b e_3 \\ \dot{R}_b = R_b S(\omega_b^b) \\ I_b \dot{\omega}_b^b = -S(\omega_b^b) I_b \omega_b^b + \tau^b \end{cases}$$

In the **Outer Loop** the position error and its derivative

$$e_p = p_b - p_{b,d} \qquad \dot{e}_p = \dot{p}_b - \dot{p}_{b,d}$$

are computed. The **Outer Loop Control** consist in the computation of the Desired z-axis of the Body Frame $z_{b,d}$ and the Commanded Total Thrust u_T

$$z_{b,d} = -\frac{-K_p e_p - K_v \dot{e}_p - mge_3 + m\ddot{p}_{b,d}}{\|-K_p e_p - K_v \dot{e}_p - mge_3 + m\ddot{p}_{b,d}\|}$$

$$u_T = -(-K_p e_p - K_v \dot{e}_p - mge_3 + m\ddot{p}_{b,d})^T R_b e_3$$

So $z_{b,d}$ is, obviously, a *Unit Vector*, while the Control Law consist in a **PD+ Control with Gravity Compensation and an Acceleration Feedforward**[8].

[7] The quadrotor is a differentially flat system with the position variables and the Yaw Angle as flat output: (x, y, z, ψ). This is a very notable property useful for the planner

[8] $R_b e_3$ is used to consider only the last column of the Rotation Matrix since $e_3 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$

Instead, in the **Inner Loop** the Desired Rotation Matrix can be computed

$$R_{b,d} = \begin{bmatrix} S(y_{b,d})z_{b,d} & \frac{S(z_{b,d})x_{b,d}}{\left\|S(z_{b,d})x_{b,d}\right\|} & z_{b,d} \end{bmatrix}$$

with $x_{b,d}$ given by the planner and $z_{b,d}$ given by the **Outer Loop**. Furthermore, in the **Inner Loop** also the Rotational Error[9] and the Angular Velocity Error

$$e_R = \frac{1}{2}(R_{b,d}^T R_b - R_b^T R_{b,d})^\vee \qquad e_\omega = \omega_b^b - R_b^T R_{b,d}\omega_{b,d}^{b,d}$$

are computed. At the end, besides the computation of the desired angular velocity expressed in the body frame $\omega_{b,d}^{b,d}$, the **Inner Loop Control** is given by

$$\tau^b = -K_R e_R - K_\omega e_\omega + S(\omega_b^b)I_b\omega_b^b - I_b(S(\omega_b^b)R_b^T R_{b,d}\omega_{b,d}^{b,d} - R_b^T R_{b,d}\dot{\omega}_{b,d}^{b,d})$$

It is possible to notice that, basically, a **Feedback Linearization of the Angular Part** has been done by the compensation of the Dynamic Model. The last two terms consist in the **Feedforward** for the velocity and for the acceleration of the angular part.

The benefits of the Geometric Control are that there are *no more filtering and numerical derivation*[10] and there are *no representation singularities*, since *Rotation Matrices* are used. So, with this controller it is possible to do, for example, acrobatic flight with the drone without problems.

On the other hand, the control is *not very robust* due to the *Feedback Linearization of the Angular Part.*

Studying the Closed-Loop System[11] and thanks to the Lyapunov Theory, it is possible to demonstrate the *exponential convergence of the error to zero* if the initial attitude error is less then 90°, but this is not a problem since usually the drone starts form the ground. Let us see some results through several plots.

The gains chosen were

$$K_p = diag([30, 30, 60]) \quad K_v = diag([4, 4, 20])$$

$$K_R = diag([40, 40, 80]) \quad K_\omega = diag([10, 10, 20])$$

[9] The Angular Error is defined in $SO(3)$ since we are working with Rotation Matrices

[10] Like for example in the Hierarcical Control

[11] $m\ddot{e}_p + K_v\dot{e}_p + K_p e_p = -\frac{u_T}{e_3^T R_{b,d}^T R_b r_3}[(e_3^T R_{b,d}^T R_b e_3)R_b e_3 - R_{b,d}e_3]$

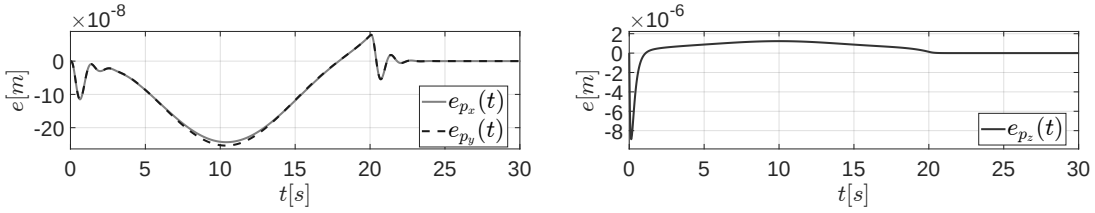

Figure 6: Linear Position Error

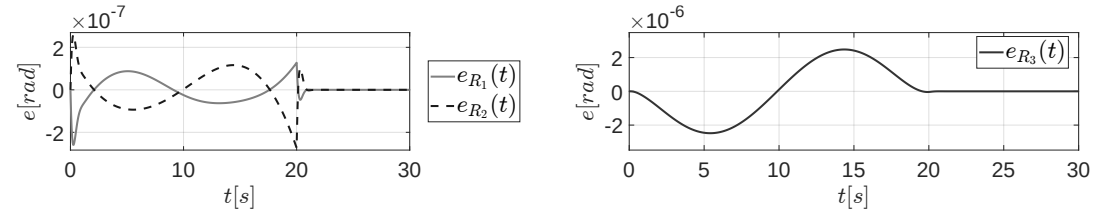

Figure 7: Angular Position Error

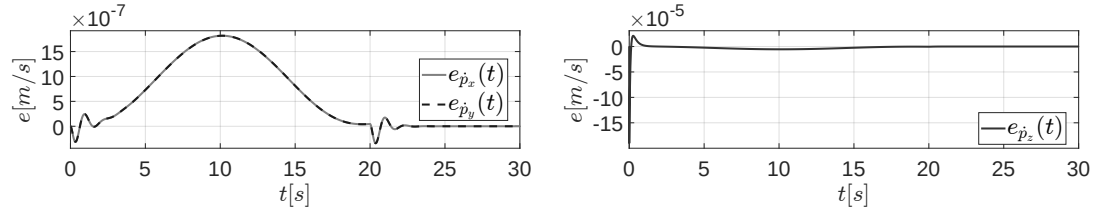

Figure 8: Linear Velocity Error

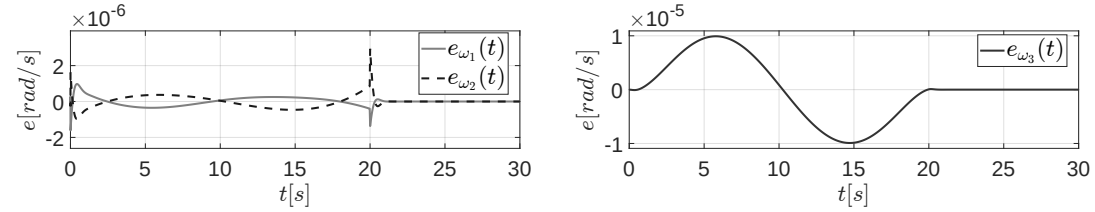

Figure 9: Angular Velocity Error

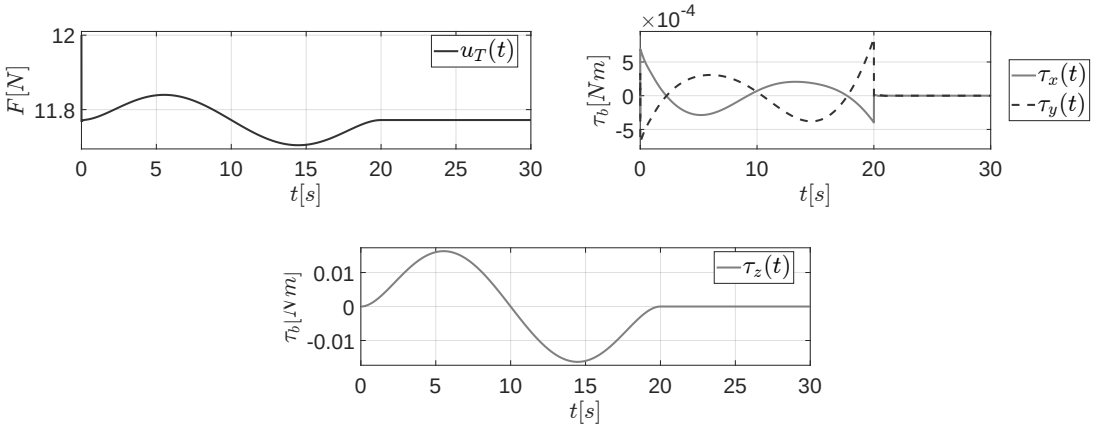

Figure 10: Total Thrust u_T and Control Torque τ^b

As can be seen from the graphs, **all the errors remain very low**. The $z-errors$ are slightly higher, in particular in the initial phase. The initial z position error, in fact, is more high to recover with respect the other two components but, after a small transient, also it becomes very small until an order of 10^{-7}. The same can be said for the velocity errors. This is due to the fact that the drone starts with an u_T[12] equal to approximately $12N$, while the steady-state value is $11.7720N$, with which the drone reaches a stationary position with a balance of all the forces acting on the UAV. Moreover, the torques are also very small, due to the fact that *the drone does not need to perform rotations around x and y Body Axis*. Only τ_z reaches higher value with respect the other two, with a maximum of $0.0163Nm$ with, obviously, a $0Nm$ value at the steady-state. At the end, the UAV reaches **steady-state** after approximately $20s$ for the control inputs and $21/22s$ for the other variables. This can also be seen in the video of the linear trajectory execution shown in the following.

YouTube Link: `https://youtu.be/WhszwPdAYqA?si=X_bzuT7MxtdR0AIk`

Exercise n°5

The UAV considered in the previous exercise was an *underactuated system*. To make a drone a **fully-actuated system** it is necessary to have the possibility of *tilting* it. In fact, as can be seen from the Allocation Matrix $G(q)[\ 1]$ in the first exercise, the first two rows are always null (and for this reason not included). This make the system *underactuated* in any case because this matrix can never

[12]Remember that, for a Non-Tilting Drone, $u_T = f_z$

be *Full-Rank*. So, in order to have a *fully-actuated system*, it is necessary to have also the **first two rows different from 0**.

An useful approach is the **Voliro Approach** (another one is the Holocopter Approach[13]), with which it is possible to obtain a **constant Allocation Matrix**, and therefore **always Full-Rank**. This is done by designing the control through f^b and τ^b and computing u_ω and α by transforming the nonlinear allocation problem into a **linear problem** through a **variable transformation** given by the formulas[14]

$$u_{v,i} = c_f u_{\omega_i} c_i \qquad u_{l,i} = c_f u_{\omega_i} s_i \qquad i = 1, ..., 4$$

Doing this, the new **Allocation Matrix** $G_{q,static} \in \mathbb{R}^{6\times 8}$ will be **constant** and

$$\begin{bmatrix} f_x \\ f_y \\ f_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ -1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & -l & -\frac{c_m}{c_f} & 0 & 0 & l & \frac{c_m}{c_f} \\ l & \frac{c_m}{c_f} & 0 & 0 & -l & -\frac{c_m}{c_f} & 0 & 0 \\ -\frac{c_m}{c_f} & l & \frac{c_m}{c_f} & -l & -\frac{c_m}{c_f} & l & \frac{c_m}{c_f} & -l \end{bmatrix} \begin{bmatrix} u_{v,1} \\ u_{l,1} \\ u_{v,2} \\ u_{l,2} \\ u_{v,3} \\ u_{l,3} \\ u_{v,4} \\ u_{l,4} \end{bmatrix} = G_{q,static} u(\alpha, u_\omega)$$

The only thing to do is to retrieve, from these new inputs, independently u_{ω_i} and the angles α. It is possible to write

$$\underbrace{\begin{bmatrix} mI_3 & 0 \\ 0 & I_b \end{bmatrix}}_{\overline{M}} \begin{bmatrix} \ddot{p}_b \\ \dot{\omega}_b^b \end{bmatrix} = \underbrace{\begin{bmatrix} mge_3 \\ -S(\omega_b^b) I_b \omega_b^b \end{bmatrix}}_{\overline{g}} + \begin{bmatrix} R_b & 0 \\ 0 & I_3 \end{bmatrix} G_{q,static} u \Rightarrow$$

$$\begin{bmatrix} \ddot{p}_b \\ \dot{\omega}_b^b \end{bmatrix} = \underbrace{\overline{M}^{-1}\overline{g}}_{\mathbf{b}} + \underbrace{\overline{M}^{-1} \begin{bmatrix} R_b & 0 \\ 0 & I_3 \end{bmatrix} G_{q,static}}_{\mathrm{A}} u \Rightarrow v = b + Au \Rightarrow u = A^\dagger(-b + v)$$

where v can be design as a **PD Control with Feedforward**. In fact

$$\ddot{p}_b = -K_P e_p - K_V \dot{e}_p + \ddot{p}_{b,d}$$

[13] In this case the allocation matrix is not constant but it is α dependent $\begin{bmatrix} G_{q,f}(\alpha) \\ G_{q,\tau}(\alpha) \end{bmatrix}$

[14] With $s_i = sin(\alpha_i)$ and $c_i = cos(\alpha_i)$

$$\dot{\omega}_b^b = -K_R e_R - K_\omega e_\omega + \dot{\omega}_{b,d}^{b,d}$$

In this way, it is possible to retrieve $u(\alpha, u_\omega)$ from f^b and τ^b. The rest is all very simple, in fact

$$u_{\omega_i} = \frac{1}{c_f}\sqrt{u_{l,i}^2 + u_{v,i}^2} \qquad \alpha_i = atan2(u_{l,i}, u_{v,i}) \qquad i = 1, ...4$$

It is important to highlight that one possible drawback of the *Voliro Approach* due to the *Pseudo-Inversion of the matrix A*. In fact, because of this, large actuator command can be generated that may result in inadmissible rotor speed that are higher than physically possible. Let us see some results

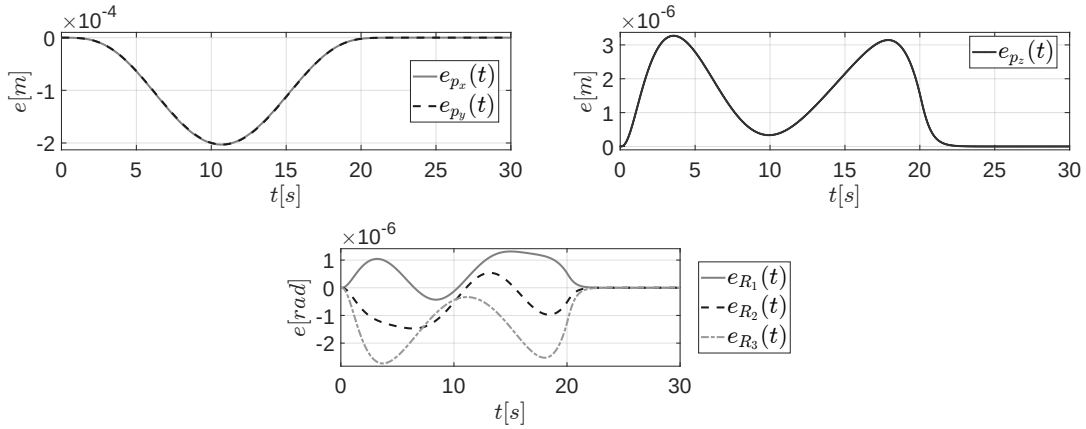

Figure 11: Linear and Angular Position Error

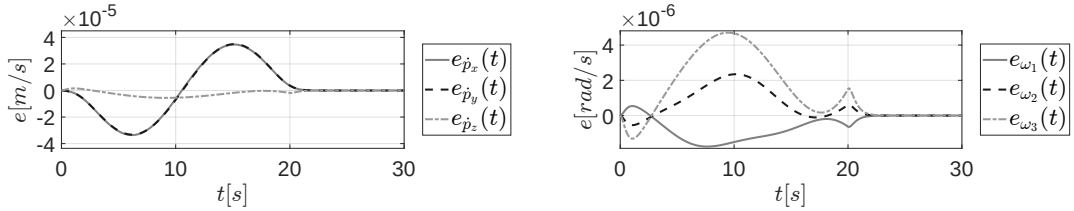

Figure 12: Linear and Angular Velocity Error

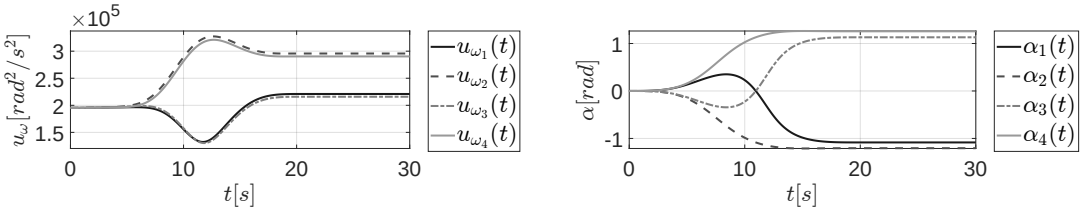

Figure 13: u_{ω_i} and Rotor Orientation α_i

From the following plot, it is possible to notice that **all the errors are very small**. From Figure 13 we can retrieve the Actual Rotor Speed[15]. The rotors reach a maximum speed of 572 rad/s (5464 RPM) (reached by rotor 1) down to a minimum of 360.96 rad/s (3448 RPM)(reached by rotor 3). Giving to ChatGPT these data, and considering a UAV, for example, of $1.25Kg$, it says that it is possible to perform maneuvers and hovering without any difficulty.
Instead, the Tilting Angles of the propellers reach some values between 1.2634 rad and -1.2113 rad.
Then, the *Linear z Position and Velocity Errors* do not exhibit the *initial transient* as in the previous exercise, due to the reasons described earlier and due to the tilting characteristics of the UAV.
At the end, the UAV reaches **steady-state** after approximately $18s$ for the control inputs and $21/22s$ for the other variables. Furthermore, it can be possible to see in the video linked at the end of this section, that *the trajectory followed by the center of mass of the tilting quadrotor is a Linear one* and it seems to reach the same *final position* as in the previous exercise.
The difference with respect this one is that here, since the drone is a tilting one, it can have **different desired orientations** during the movement. In fact the robot first start to rotate around the *Body z-Axis* and then, **in the last seconds of the trajectory, it perform some rotations also around the other two Body Axis, up to reaching the position shown at the end of the video**, with the final **RPY Angles** $\varphi = 0.7854rad = 45°$, $\theta = 0.7854rad = 45°$ and $\psi = 3.1416rad = 180°$.
Then, as mentioned, in the final part all the errors converge to zero, thus allowing a **perfect tracking** of both position and orientation variables.
Thanks to its characteristics, the UAV is able to reach a **final tilted position with respect to the initial one, and it remains stably in it while hovering**.
YouTube Link: `https://youtu.be/g0UQ6E7wnBU?si=cSS7kXi5ydt4JRPQ`

[15]Remember that $u_{\omega_i} = \omega_i|\omega_i|$