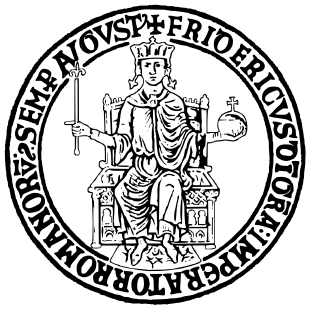
Link alla cartella GitHub:

https://github.com/alessandragranata/Homework$_{3F}SR.git$

Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea Magistrale in Automazione e Robotica

Fields and Service Robotics

Homework 3

Relatore
Prof. Fabio Ruggiero

Candidata
Alessandra Granata P38000279

Anno Accademico 2024–2025

1. Degrees of Freedom, Underactuation, and Allocation Matrix of an Octocopter

Let's consider the octocopter composed of 8 propellers arranged in a coplanar configuration. When evaluating the degrees of freedom of the system, it is necessary to distinguish between two possible scenarios:

Octocopter Fixed to the Ground

When the octocopter is fixed to the ground, it cannot move in space, so the only possible movement is that of the eight propellers. Each propeller can be modeled as a *revolute joint*, since it rotates around its own axis.
To determine the number of degrees of freedom of the system, we apply the **Grübler formula** for spatial mechanisms:

$$\text{DoF} = 6 \cdot (N - 1 - J) + \sum_{i=1}^{J} f_i \tag{1}$$

where:

- $N = 9$; $J = 8$ is the number of joints (one for each propeller), and each joint has $f_i = 1$ degree of freedom (a single rotation).

$$\text{DoF} = 6 \cdot (9 - 1 - 8) + 8 = 6 \cdot 0 + 8 = 8 \tag{2}$$

Therefore, the system has **8 degrees of freedom**, which are determined by the propellers of the drone.

Configuration Space

Since each propeller rotates about its own axis, it behaves like a revolute joint, and its configuration is described by an angle (i.e., a rotation over a circle, which is topologically equivalent to S^1). Therefore, the configuration space is:

$$\mathcal{C} = S^1 \times S^1 \times S^1 \times S^1 \times S^1 \times S^1 \times S^1 \times S^1 = \mathbb{T}^8 \tag{3}$$

Octocopter in Flight

When the octocopter is in flight, it behaves as a rigid body free to move in three-dimensional space. In addition to the degrees of freedom of the drone when is fixed on the ground, it is necessary to also account for the translational and rotational motion of the drone's body. In particular, 3 translational degrees of freedom and 3 rotational degrees of freedom are added. Therefore, the total number of degrees of freedom of the octocopter in flight is:

$$\text{DoF} = 8 + 6 = 14$$

Configuration Space

As the number of degrees of freedom increases, the configuration space of the system also expands accordingly. In addition to the rotations of the 8 propellers, already modeled as $\mathbb{T}^8$ we must now include:

- the position of the drone's center of mass in three-dimensional space, represented by $\mathbb{R}^3$,
- the orientation of the rigid body, which can be described as a combination of:
  - $\mathbb{S}^2$: the direction of the oriented axis,
  - $\mathbb{S}^1$: the rotation about that axis.

Thus, the complete configuration space of the system in flight becomes:

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

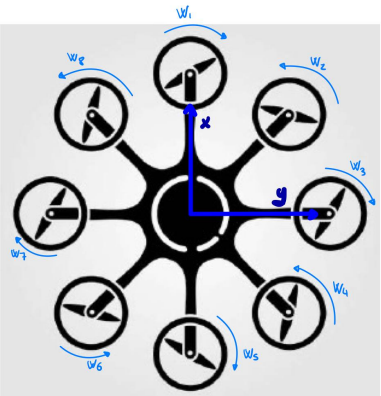
Underactuation

In the this octocopter , all propellers are arranged in a **coplanar configuration**, meaning they lie on the same horizontal plane and are **oriented vertically**. Each propeller generates a *lift force* perpendicular to the propeller plane, resulting in thrust forces that are parallel to each other and directed upward, along the z-axis of the body frame. In this configuration, the system can produce a resultant force along the vertical axis passing through the center of mass, allowing the drone to move only *upward or downward*. **Horizontal motion** in the x-y plane is not achievable unless the drone body is tilted. Tilting modifies the direction of the net thrust, projecting a component onto the horizontal plane and enabling forward, backward, or lateral motion. To obtain **direct horizontal thrust**, the propellers would need to be *tiltable*, capable of changing the inclination of the generated force. In a standard octocopter, where the propellers are fixed and vertical, this is not possible without modifying the orientation of the entire body. Therefore, *no combination of forces generated by coplanar and vertical propellers* can directly produce a **net horizontal force**. This constraint implies that **arbitrary instantaneous accelerations in all directions of 3D space are not possible**, and the system is thus **underactuated**. The octocopter is also **intrinsically underactuated**: despite having 8 propellers, the system has only **4 effectively controllable independent inputs**, which are:

- the total thrust along z,
- and the three control torques around the body frame axes: τ_x, τ_y, τ_z.

The number of independent control inputs is lower than the number of degrees of freedom, confirming the *underactuated nature* of the system.
Indeed, if the system's allocation matrix is computed, it turns out not to be full rank, precisely because the first two rows of the matrix will be zero, due to the fact that the tilting angles of the propellers are zero. This provides direct evidence that the system is underactuated. This matrix will be computed in detail in the next paragraph.

Allocation Matrix



In the configuration under consideration, the octocopter features eight propellers symmetrically arranged around the body frame. The direction of rotation of each propeller follows an alternating pattern: if one rotates clockwise, the adjacent one rotates counterclockwise, and so on around the structure. Each propeller is associated with a local reference frame, fixed to the brushless motor (and not to the rotating blades). In this local frame, the z-axis is oriented upward, according to the ENU (East-North-Up) convention, and remains fixed with respect to the drone body. The body frame of the drone, on the other hand, is defined according to the NED (North-East-Down) convention, with the z-axis pointing downward. Consequently, the direction of the thrust force

generated by each propeller, expressed in the body frame, is given by:

$$\mathbf{z}_b^{p_i} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

Each propeller generates a thrust force aligned with $\mathbf{z}_{p_i}$, resulting from its interaction with the airflow, and a drag moment due to aerodynamic resistance to rotation. The analysis is carried out under quasi-static assumptions, neglecting flapping effects the inertia of the propellers, and other secondary aerodynamic effects. In this context, the net force applied at the drone's center of mass, expressed in the body frame, is computed by summing the thrust contributions of all propellers:

$$\mathbf{f}^b = \sum_{i=1}^{8} c_f \cdot |\omega_i| \cdot \omega_i \cdot \mathbf{z}_{p_i}^b$$

Since $\mathbf{z}_{p_i}^b = \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T$, the negative sign is already included in the vector, considering that the propeller frames follow the ENU convention, whereas the body frame is defined in NED.The total torque with respect to the center of mass includes two contributions: the first is the drag moment generated by each propeller, and the second is the moment induced by the position of the propeller with respect to the center of mass. The total torque in the body frame is given by:

$$\boldsymbol{\tau}^b = \sum_{i=1}^{8} |\omega_i| \cdot \omega_i \cdot \left(-k_i c_m \cdot \mathbf{z}_{p_i}^b + c_f \cdot S(\mathbf{p}_{p_i}^b) \cdot \mathbf{z}_{p_i}^b \right)$$

where ω_i is the angular velocity of the i-th propeller, c_f is the thrust coefficient, c_m is the drag coefficient, and $k_i \in \{+1, -1\}$ a sign factor determined by the blade geometry and the direction of rotation of the propeller: $k_i = +1$ for counterclockwise rotation and $k_i = -1$ for clockwise rotation. Assuming the brushless motors are controlled via angular velocity, we define:

$$u_{\omega_i} = |\omega_i| \cdot \omega_i$$

as the control input associated with the i-th propeller. This quantity is real and continuous, and directly proportional to the thrust and drag moment produced by the rotor. The actuation vector of the system, consisting of the total force $\mathbf{f}^b$ and torque $\boldsymbol{\tau}^b$, can be written as:

$$\begin{bmatrix} \mathbf{f}^b \\ \boldsymbol{\tau}^b \end{bmatrix} = \mathbf{f}_u(\alpha_i, \beta_i; u_{\omega_i})$$

where α_i and β_i are the angles that describe the orientation of the i-th propeller's rotation axis relative to the body frame. In the configuration considered, all propeller axes are parallel to each other and aligned with the body frame's z-axis, which corresponds to setting $\alpha_i = \beta_i = 0$ for all $i = 1, \ldots, 8$. As a result, the total thrust vector has no components along the x and y axes: only the component along the z-axis is active:

$$f_z = -c_f \cdot (u_{\omega_1} + u_{\omega_2} + u_{\omega_3} + u_{\omega_4} + u_{\omega_5} + u_{\omega_6} + u_{\omega_7} + u_{\omega_8})$$

To build the allocation matrix, a 6×8 matrix is defined, where the six rows correspond to the components of the total force and torque vectors in the body frame, while the eight columns represent the inputs of the individual propellers. The torque components along the x, y, and z axes are:

$$\tau_x = l \cdot c_f \cdot \left(-\frac{\sqrt{2}}{2} u_{\omega_2} - u_{\omega_3} - \frac{\sqrt{2}}{2} u_{\omega_4} + \frac{\sqrt{2}}{2} u_{\omega_6} + u_{\omega_7} + \frac{\sqrt{2}}{2} u_{\omega_8} \right)$$

$$\tau_y = l \cdot c_f \cdot \left(u_{\omega_1} + \frac{\sqrt{2}}{2} u_{\omega_2} - \frac{\sqrt{2}}{2} u_{\omega_4} - u_{\omega_5} - \frac{\sqrt{2}}{2} u_{\omega_6} + \frac{\sqrt{2}}{2} u_{\omega_8} \right)$$

$$\tau_z = c_m \cdot (-u_{\omega_1} + u_{\omega_2} - u_{\omega_3} + u_{\omega_4} - u_{\omega_5} + u_{\omega_6} - u_{\omega_7} + u_{\omega_8})$$

Finally, the allocation matrix is expressed as:

$$\mathbf{G}_q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -c_f & -c_f & -c_f & -c_f & -c_f & -c_f & -c_f & -c_f \\ 0 & -\frac{\sqrt{2}}{2} c_f l & -c_f l & -\frac{\sqrt{2}}{2} c_f l & 0 & \frac{\sqrt{2}}{2} c_f l & c_f l & \frac{\sqrt{2}}{2} c_f l \\ c_f l & \frac{\sqrt{2}}{2} c_f l & 0 & -\frac{\sqrt{2}}{2} c_f l & -c_f l & -\frac{\sqrt{2}}{2} c_f l & 0 & \frac{\sqrt{2}}{2} c_f l \\ -c_m & c_m & -c_m & c_m & -c_m & c_m & -c_m & c_m \end{bmatrix}$$

Here, $\mathbf{G}_q \in \mathbb{R}^{6\times 8}$ is the allocation matrix. This matrix is not full rank due to the first two zero rows, which correspond to the f_x and f_y components. This confirms that the system is underactuated, as it is not possible to generate horizontal forces in the x-y plane without changing the orientation of the body.

2. Ground effect and Ceiling effect

Aerial robots can fly in open environments and interact with the surrounding area through sensors. However, many applications—especially those related to inspections—require the UAV to operate close to surfaces or obstacles. In these cases, the environment can significantly affect the flight behavior of the drone. One phenomenon of particular interest is the interaction between the airflow generated by the propellers and nearby surfaces. When the propeller spins rapidly, it generates a flow that, upon reflecting off the ground, interacts again with the drone, altering its dynamics. This phenomenon is known as *ground effect*, and it has been studied using *potential flow models*. This modeling approach relies on the following assumptions: the fluid is inviscid, incompressible, irrotational, and steady. To model ground effect, the *method of images* is commonly used. This method consists in introducing a "virtual" propeller symmetrically located with respect to the ground plane. If the real propeller is at the point $[0\ 0\ h]^T$, then the virtual propeller is placed at $[0\ 0\ -h]^T$. This configuration allows for simulating the reflected airflow, helping to evaluate the forces acting on the drone.We define the following quantities: V_{IGE} and T_{IGE}: induced velocity and thrust in the presence of ground effect; V_{OGE} and T_{OGE}: induced velocity and thrust in the absence of ground effect.
Assuming a single-propeller UAV initially flying far from the ground (where ground effect is negligible), the acting forces are T_{OG} and V_{OGE}. As the drone approaches the ground, the reflected flow alters these quantities. If we assume that the power supplied to the system remains constant, the following relation holds:

$$T_{\text{IGE}} \cdot V_{\text{IGE}} = T_{\text{OGE}} \cdot V_{\text{OGE}}$$

Applying the method of images and introducing some simplifications, the following theoretical expression is obtained to estimate the thrust in the presence of ground effect for a single propeller:

$$\frac{T_{\text{IGE}}}{T_{\text{OGE}}} = \frac{1}{1 - \left(\frac{\rho}{4Z}\right)^2}$$

where ρ is the radius of the propeller and z is its distance from the ground. The ground effect becomes negligible and it can be ignored when $z > 2\rho$. In the case of multirotor systems, the method of images is applied to each propeller, but the mutual interactions between real and virtual propellers must also be considered. For instance, propeller 1 is influenced not only by

its own image, but also by the images of propellers 2, 3, and 4. As a result, the thrust ratio in the presence and absence of ground effect becomes a complex function of ρ, z, and the distance d between propellers. Assuming that the power remains constant for the complete multirotor system, the following expression holds:

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

Where, d is the distance between the propellers. The greater the value of d, the lower the mutual influence between the propellers. The complete expression accounts not only for the reflection of the flow from each propeller's own image, but also for the contributions from the other propellers and their virtual counterparts. If the drone is tilted, it is possible that only some propellers are within the ground effect zone. This can generate destabilizing torques—or, in some cases, may even help stabilize the drone, depending on the situation. In any case, these effects are important considerations in the design of the control system. In conclusion, ground effect is a crucial phenomenon to consider when designing and controlling multirotor drones, especially for operations in confined environments, near surfaces, or with external payloads.
In addition to the well-known ground effect, a very common phenomenon in inspection and maintenance applications is the so-called **ceiling effect**. This effect is, in a certain sense, the opposite of the ground effect: it occurs when flying very close to an upper surface. Also in this case, the *method of images* can be used to model the phenomenon: a "virtual" propeller is imagined above the surface, while the real propeller is just below it. This allows for the estimation, similarly to the ground effect, of the ratio between the thrust generated in the presence and absence of the ceiling effect for a single propeller. This ratio depends on the **radius of the propeller** ρ, the **distance** z between the propeller and the ceiling, and two parameters k_1 and k_2, which must be tuned based on various conditions, including the characteristics of the ceiling surface. The theoretical model describing this phenomenon is the following:

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

The closer the drone gets to the ceiling, the stronger the ceiling effect becomes. Unlike the ground effect, where the reflected airflow helps generate additional upward thrust, the ceiling effect acts in the opposite way: as the propeller spins rapidly and approaches the surface, it removes the air particles between the propeller and the ceiling, creating a sort of *vacuum effect*. This causes a *low-pressure region* between the propeller and the surface, which can lead the drone to be "sucked" upward, risking sudden collisions with the structure. To counteract the ceiling effect, it is necessary to reduce the rotational speed of the propellers in order to maintain constant thrust and avoid the undesired pressure drop. The effect is therefore exactly the opposite of the ground effect: while in the latter the reflected air "pushes" the drone upwards, in the case of the ceiling effect, the drone is "attracted" towards the upper surface. If not properly modeled or compensated for, this can lead to dangerous collisions, especially during operations in confined spaces or near critical infrastructure.

3. Estimation of External Disturbances on a Quadrotor Using a Momentum-Based Estimator

Introdution

In this exercise, the flight of a quadrotor is analyzed using the data provided in the file `ws_homework_3_2025.m`
The goal of the exercise is to implement a momentum-based estimator of order r, with a sampling time of 1 ms. The estimator should provide an estimate of the external forces and torques acting on the UAV during the flight, to be compared with the actual applied disturbances.

Estimator-based control

An external disturbance estimation method based on momentum was implemented using flight data from a UAV. This approach, commonly integrated into control architectures such as hierarchical, geometric, and passivity-based schemes, allows for the compensation of unmodeled dynamics. The estimation relies on comparing the measured momentum with that predicted by the nominal model: any discrepancies are attributed to external forces and torques. This enables the estimation of disturbances caused, for instance, by unmodeled aerodynamic effects, inaccuracies in physical parameters, or interactions with the environment. The dynamic model of the quadrotor in RPY coordinates is then considered, including external wrench disturbances modeled as additional forces and torques:

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

The terms $\mathbf{f}_e$ and $\boldsymbol{\tau}_e$ represent the external perturbative forces and torques, respectively, and they are the target of estimation through the momentum-based estimator. The generalized momentum vector $\mathbf{q} \in \mathbb{R}^6$ is defined as:

$$\mathbf{q} = \begin{bmatrix} m\mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & M(\eta_b) \end{bmatrix} \begin{bmatrix} \dot{p}_b \\ \dot{\eta}_b \end{bmatrix}$$

Then, the time derivative of the generalized momentum is computed. At this point, we define the wrench vector $\begin{bmatrix} \mathbf{f}_e \\ \boldsymbol{\tau}_e \end{bmatrix}$, which represents the combination of external forces and torques acting on the system. This is the real wrench, which cannot be directly measured. The goal is therefore to estimate this vector by constructing the estimated wrench $\begin{bmatrix} \hat{\mathbf{f}}_e \\ \hat{\boldsymbol{\tau}}_e \end{bmatrix}$, which should asymptotically converge to the real wrench over time. A linear relationship between the estimated external wrench and the real wrench is desired. To this end, the analysis is carried out in the Laplace domain, where such a relationship can be written as:

$$\mathcal{L}\left\{ \begin{bmatrix} \hat{\mathbf{f}}_e \\ \hat{\boldsymbol{\tau}}_e \end{bmatrix} \right\} = \mathbf{G}(s)\, \mathcal{L}\left\{ \begin{bmatrix} \mathbf{f}_e \\ \boldsymbol{\tau}_e \end{bmatrix} \right\}$$

where $\mathbf{G}(s) \in \mathbb{C}^{6\times 6}$ is a diagonal matrix of transfer functions. In the case of a first-order momentum-based estimator, the transfer functions are chosen as:

$$G_i(s) = \frac{k_0}{s + c_0}$$

with k_0 and c_0 being positive gains. Since the objective is for the estimate to asymptotically converge to the real value of the disturbance, the static gain must be equal to 1. This implies the condition: $k_0 = c_0$. To make the estimate as accurate as possible, it is important to avoid filtering out the dynamic components of the signal. This suggests selecting a sufficiently large value of the gain in order to preserve the high-frequency components of the real disturbance. However, since the estimator relies on measurements that are typically affected by noise, choosing a gain that is too high may allow the noise to pass through as well, thereby degrading the quality of the estimation. Therefore, a trade-off must be found between estimation accuracy and robustness to noise, by appropriately tuning the value of the gain.
The final form of the first-order momentum-based estimator is given by:

$$\begin{bmatrix} \hat{\mathbf{f}}_e \\ \hat{\boldsymbol{\tau}}_e \end{bmatrix} = k_0 \left(\mathbf{q} - \int_0^t \left[\begin{bmatrix} \hat{\mathbf{f}}_e \\ \hat{\boldsymbol{\tau}}_e \end{bmatrix} + mg\mathbf{e}_3 - u_T \mathbf{R}_b \mathbf{e}_3 + \mathbf{C}^T(\boldsymbol{\eta}_b, \dot{\boldsymbol{\eta}}_b)\, \dot{\boldsymbol{\eta}}_b + \mathbf{Q}^T(\boldsymbol{\eta}_b)\, \boldsymbol{\tau}_b \right] dt \right)$$

To compute the estimator, it is necessary to have access to the commanded thrust u_T and the commanded torques $\boldsymbol{\tau}_b$, both of which are provided by the instructor through the `signals` fields of the `thrust` and `tau` structures, respectively. In addition, the estimation process requires the knowledge of several system parameters: the mass m, the inertia matrix $\mathbf{I}_b$,which is essential for computing both the matrix $\mathbf{C}$ and the generalized momentum vector $\mathbf{q}$, and the Euler angles along with their time derivatives. These angular variables are used to compute the matrices $\mathbf{C}$, $\mathbf{Q}$, and the rotation matrix $\mathbf{R}_b$, all of which are necessary to model the system's dynamics accurately.
In the context of this exercise, an *order-r estimator* has been implemented using the following recursive formulation:

$$\gamma_1(t) = K_1 \left(\mathbf{q} - \int_0^t \left[\begin{bmatrix} \hat{\mathbf{f}}_e \\ \hat{\boldsymbol{\tau}}_e \end{bmatrix} + mg\mathbf{e}_3 - u_T \mathbf{R}_b \mathbf{e}_3 + \mathbf{C}^T(\boldsymbol{\eta}_b, \dot{\boldsymbol{\eta}}_b)\dot{\boldsymbol{\eta}}_b + \mathbf{Q}^T(\boldsymbol{\eta}_b)\boldsymbol{\tau}_b \right] dt \right) \tag{4}$$

$$\gamma_i(t) = K_i \int_0^t \left(\mathbf{q} - \begin{bmatrix} \hat{\mathbf{f}}_e \\ \hat{\boldsymbol{\tau}}_e \end{bmatrix} + \gamma_{i-1}(t) \right) dt, \quad \text{for } i = 2, \ldots, r \tag{5}$$

In the MATLAB code, the continuous-time formulation was discretized using Euler integration with a sampling time of 1 ms. To determine the gains K_i to be used in the recursive formulation of the estimator, a normalized Butterworth filter of order r, with a cutoff frequency of 1 rad/s, was adopted. Using the `butter` function in MATLAB, the coefficients a_j of the denominator of the transfer function were computed. Based on these coefficients, the gains K_i were calculated to satisfy the following condition:

$$\prod_{i=j+1}^{r} K_i = a_j, \quad \text{for } j = 0, \ldots, r-1$$

This condition ensures that the dynamics of the estimator accurately reproduce those of the Butterworth filter, providing a desirable frequency response and allowing an appropriate trade-off between noise attenuation and estimation fidelity. In the MATLAB implementation, the estimator order r is a parameter specified by the user. To perform the recursive formulation of the estimator, the following dynamic matrices need to be computed:
$Q(\eta_b)$, $M(\eta_b)$, q, $C(\eta_b, \dot{\eta}_b)$, R_b.
All these matrices are recomputed at each time step within the main loop of the script, using the updated values of the Euler angles and their derivatives. This allows the system dynamics to be updated in real-time, enabling the recursive estimator to operate correctly.

Plots and real mass computation

The following plots analyze the estimation results of the external forces for different values of the estimator order r. In particular, for each value $r \in \{1, 3, 8\}$, are shown: the comparison between the estimated external force ($\hat{f}_x$, $\hat{f}_y$, $\hat{\tau}_z$) and the true value. In the case $r = 1$, the transients are fast and without overshoot, with stable convergence within approximately 5 seconds. The estimation errors are small, and no significant oscillations are observed, highlighting a good responsiveness of the system, although potentially more sensitive to noise (even if not visible in simulation). When increasing the order to $r = 3$, the rise time increases slightly. Small overshoots can be observed, especially in the estimation of the torque $\hat{\tau}_z$, as well as minor oscillatory phenomena during the transient phase. In this case, a compromise between responsiveness and robustness is achieved, with still acceptable performance. In the case $r = 8$, the estimator shows a markedly slower behavior, with longer transients, significant delays, and more noticeable oscillations in the estimation errors. Although the sensitivity to noise could be reduced, the overall performance is penalized, especially in terms of speed and accuracy during the initial phase. In conclusion, the plots clearly show that as r increases, the estimator dynamics slow down, introducing delays and overshoots. It is observed that already from $r = 3$, the additional benefits in terms of smoothness are marginal, while convergence times increase significantly.

Figure 1: Estimated wrench r=1

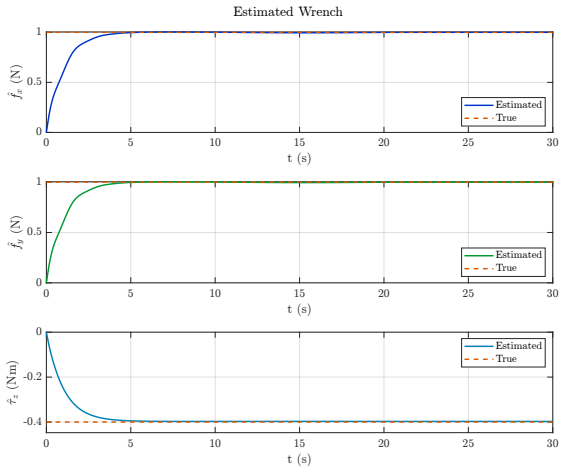


Figure 2: Estimated wrench r=3

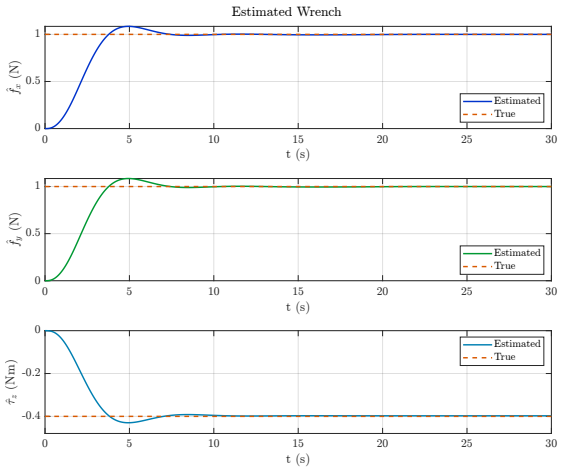
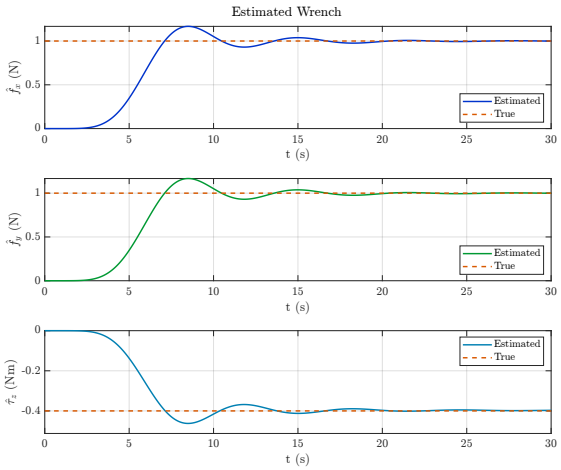


Figure 3: Estimated wrench r=8



Based on the estimated disturbance along the z-axis, the actual mass of the UAV was computed using the following relation:

$$\bar{m}g = mg + f_z$$

where $\bar{m}$ represents the actual mass and f_z is the steady-state value of the disturbance along the z-axis, evaluated at the final sample N. The resulting value of $\bar{m}$ is 1.25 kg, compared to the nominal mass of 1.5 kg. This result is consistent with expectations: since the estimated external force is negative, it is reasonable to assume that the effective weight acting on the drone is lower than the nominal one.

4. Implementation of Geometric Control

In this exercise, a geometric control strategy for a quadrotor is implemented using a provided Simulink template. The objective is to complete both the inner and outer control loops, simulate the closed-loop system, and analyze the most relevant results.

In geometric control, a coordinate-free dynamic model is adopted, where the translational dynamics are described in the world frame, while the angular dynamics are expressed in the body frame:

$$\begin{cases} m\ddot{\mathbf{p}}_b = mg\mathbf{e}_3 - u_T\mathbf{R}_b\mathbf{e}_3 \\ \dot{\mathbf{R}}_b = \mathbf{R}_b S(\boldsymbol{\omega}_b^b) \\ \mathbf{I}_b\dot{\boldsymbol{\omega}}_b^b = -S(\boldsymbol{\omega}_b^b)\mathbf{I}_b\boldsymbol{\omega}_b^b + \boldsymbol{\tau}^b \end{cases} \tag{6}$$

In geometric control, the drone's orientation is represented using rotation matrices instead of Euler angles, thereby avoiding singularity issues. Although the system remains underactuated, the control architecture is structured in two loops: the outer loop manages the linear dynamics and provides a reference to the inner loop, which controls the angular dynamics. Unlike hierarchical control, the outer loop does not provide roll and pitch angles, but directly generates the desired rotation matrix $R_{b,d}$. The construction of $R_{b,d}$ starts from the vector $\mathbf{z}_{b,d}$, which represents the desired thrust direction and forms the third column of the matrix. The vector $\mathbf{x}_{b,d}$, provided by the planner, represents the forward direction, but it may not be orthogonal to $\mathbf{z}_{b,d}$; for this reason, it is projected onto the plane orthogonal to $\mathbf{z}_{b,d}$. The corrected version of $\mathbf{x}_{b,d}$ thus obtained allows the computation of $\mathbf{y}_{b,d}$ as the normalized cross product between $\mathbf{z}_{b,d}$ and $\mathbf{x}_{b,d}$:

$$\mathbf{y}_{b,d} = \frac{S(\mathbf{z}_{b,d})\,\mathbf{x}_{b,d}}{\|S(\mathbf{z}_{b,d})\,\mathbf{x}_{b,d}\|}$$

This ensures that $\mathbf{y}_{b,d}$ is a unit vector and orthogonal to the other two. Finally, the corrected $\mathbf{x}_{b,d}$ axis is obtained. In this way, an orthonormal triplet $[\mathbf{x}_{b,d}, \mathbf{y}_{b,d}, \mathbf{z}_{b,d}]$ is obtained, which forms the desired rotation matrix $\mathbf{R}_{b,d}$, representing the orientation configuration that the drone's body frame must assume with respect to the world frame:

$$\mathbf{R}_{b,d} = \begin{bmatrix} S(\mathbf{y}_{b,d})\mathbf{z}_{b,d} & \frac{S(\mathbf{z}_{b,d})\mathbf{x}_{b,d}}{\|S(\mathbf{z}_{b,d})\mathbf{x}_{b,d}\|} & \mathbf{z}_{b,d} \end{bmatrix}$$

The objective of the geometric controller is to cancel the tracking error, that is, to bring the current variables to coincide with the desired ones: $\mathbf{p}_{b,d}$, $\dot{\mathbf{p}}_{b,d}$, and Proj$[\mathbf{x}_{b,d}]$.
The tracking errors are defined as follows:
Translational motion error: $\mathbf{e}_p = \mathbf{p}_b - \mathbf{p}_{b,d}$, $\dot{\mathbf{e}}_p = \dot{\mathbf{p}}_b - \dot{\mathbf{p}}_{b,d}$.
Rotational motion error: $\mathbf{e}_R = \frac{1}{2}\left(\mathbf{R}_{b,d}^T\mathbf{R}_b - \mathbf{R}_b^T\mathbf{R}_{b,d}\right)^\vee$, $\mathbf{e}_\omega = \boldsymbol{\omega}_b^b - \mathbf{R}_b^T\mathbf{R}_{b,d}\,\boldsymbol{\omega}_{b,d}^b$.
Where: $\mathbf{e}_R$ represents the error between the current rotation of the drone and the desired one, and is defined in the space $SO(3)$. The operator $\vee$ allows extracting the vector from a skew-symmetric matrix. $\mathbf{e}_\omega$ represents the difference between the current angular velocity of the drone, expressed in the body frame, and the desired one, properly transformed into the same reference frame. It is important to note that the error $\mathbf{e}_\omega$ cannot be obtained by differentiating the error $\mathbf{e}_R$ over time, as it is necessary to account for the rotational dynamics of the system. In particular, the desired angular velocity $\boldsymbol{\omega}_{b,d}^b$ is originally defined in the desired frame,it is first transformed into the world frame by multiplying it by $\mathbf{R}_{b,d}$, and then mapped into the current body frame by multiplying by $\mathbf{R}_b^\top$. The difference with respect to the measured angular velocity $\boldsymbol{\omega}_b^b$ gives the error angular velocity $\mathbf{e}_\omega$.

Outer Loop Control

Once the tracking errors for the translational motion have been defined, it is possible to compute the control action of the **outer loop**. In geometric control, the goal of the outer loop is to compute the **total thrust** u_T and the desired direction $\mathbf{z}_{b,d}$, which represents the axis along which the thrust force must be applied, while the goal of the inner loop is the computation of the torque $\boldsymbol{\tau}_b$. The control action is computed using a PD+ action with gravity compensation and feedforward of the desired acceleration:

$$\mathbf{A} = -K_p\mathbf{e}_p - K_v\dot{\mathbf{e}}_p - mg\mathbf{e}_3 + m\ddot{\mathbf{p}}_{b,d}$$

From this expression, the two fundamental quantities are derived:

$$u_T = -\mathbf{A}^T\mathbf{R}_b\mathbf{e}_3, \quad \mathbf{z}_{b,d} = \frac{\mathbf{A}}{\|\mathbf{A}\|}$$

The term u_T represents the total thrust to be applied along the direction of the axis $\mathbf{z}_b$, and is obtained by simply projecting the desired force vector $\mathbf{A}$ onto the axis $\mathbf{z}_b$, that is, by computing the dot product between $\mathbf{A}$ and $\mathbf{R}_b\mathbf{e}_3$, which corresponds to the last column of the drone's rotation matrix. The desired direction $\mathbf{z}_{b,d}$ is instead obtained by normalizing the vector $\mathbf{A}$. Under ideal conditions (i.e., in the absence of error), the force $\mathbf{A}$ exactly compensates for gravity and provides the desired acceleration, thus aligning with the vertical axis. An additional interpretation of the thrust term u_T highlights an important property of the geometric controller. The total thrust can in fact be rewritten as:

$$u_T = -\|\mathbf{A}\| \left(\mathbf{R}_{b,d}\mathbf{e}_3\right)^\top \mathbf{R}_b\mathbf{e}_3 = -\|\mathbf{A}\| \, \mathbf{z}_{b,d}^\top \mathbf{z}_b$$

The term u_T depends on the cosine of the angle between $\mathbf{z}_{b,d}$ and $\mathbf{z}_b$, making the total thrust proportional to the norm of vector $\mathbf{A}$ multiplied by $\cos(\theta)$. This naturally limits the maximum thrust that can be delivered, making the controller self-limiting: in the presence of large attitude errors, the thrust is reduced, thus preventing excessive stress on the motors.
The position $\mathbf{p}_b$ and linear velocity $\dot{\mathbf{p}}_b$ are obtained from the model feedback, while the corresponding desired quantities $\mathbf{p}_{b,d}$, $\dot{\mathbf{p}}_{b,d}$, and $\ddot{\mathbf{p}}_{b,d}$ are provided by the planner and are inputs to the `Outer-loop control` function, where the control described above is implemented. The gains K_p and K_v represent the proportional and derivative coefficients, respectively, of the PD+ controller for the outer loop, and have been chosen as:

$$K_p = \text{diag}(35,\ 35,\ 45), \quad K_v = \text{diag}(25,\ 25,\ 35)$$

The performance of the geometric controller strongly depends on proper tuning of the gains in the outer and inner control loops. Regarding the outer loop, which is responsible for controlling the translational motion, the proportional and derivative gains were selected to ensure good responsiveness in trajectory tracking and effective damping of velocity errors, following a systematic tuning process. These moderate values allow for generating smooth and consistent commands for the inner loop.

Inner Loop Control

For the Inner-Loop the torque is computed using the following expression:

$$\boldsymbol{\tau}_b = -K_R\mathbf{e}_R - K_\omega\mathbf{e}_\omega + S(\boldsymbol{\omega}_b^b)\mathbf{I}_b\boldsymbol{\omega}_b^b - \mathbf{I}_b\left(S(\boldsymbol{\omega}_b^b)\mathbf{R}_b^\top\mathbf{R}_{b,d}\dot{\boldsymbol{\omega}}_{b,d}^b - \mathbf{R}_b^\top\mathbf{R}_{b,d}\dot{\boldsymbol{\omega}}_{b,d}^b\right) \tag{7}$$

The construction of the desired rotation matrix $\mathbf{R}_{b,d}$, once the vector $\mathbf{z}_{b,d}$ is obtained, has been implemented inside the `Inner-Loop control` MATLAB function, as described earlier. The parameters K_R and K_ω represent the gains of the PD controller for the angular part, and were selected as:

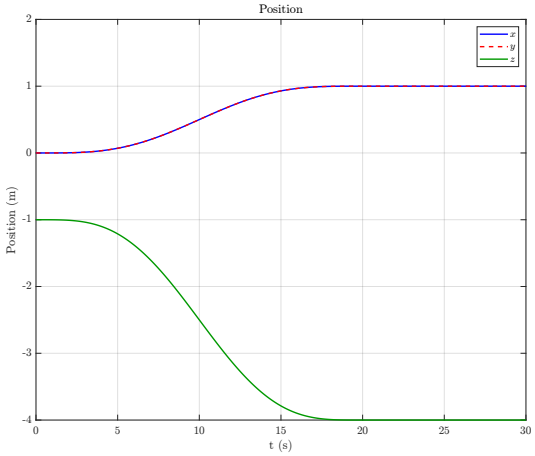
$$K_R = \text{diag}(120,\ 120,\ 120), \quad K_\omega = \text{diag}(80,\ 80,\ 80)$$

For the inner loop, which governs the rotational dynamics, it is crucial to ensure a faster response. In fact, the stability and effectiveness of the outer loop directly depend on the ability of the inner controller to quickly align the attitude with the desired configuration. To this end, higher gains were selected for the orientation and angular velocity errors. In summary, the gains were tuned according to a hierarchical logic, ensuring rapid stabilization of the attitude and, consequently, accurate trajectory tracking.

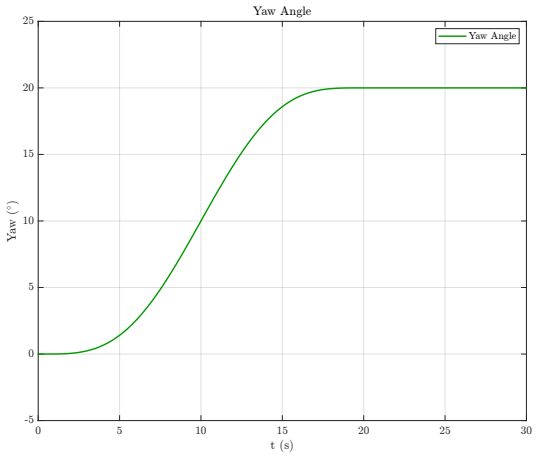
Simulation Results

Below are the plots obtained from the simulation, useful for evaluating the performance of the implemented control system: The **Position** plot shows the trend of the drone's center of mass position along the x, y, and z axes over time. The trajectory is well tracked and the final position is correctly reached, confirming the effectiveness of the outer controller. The **Yaw Angle** plot

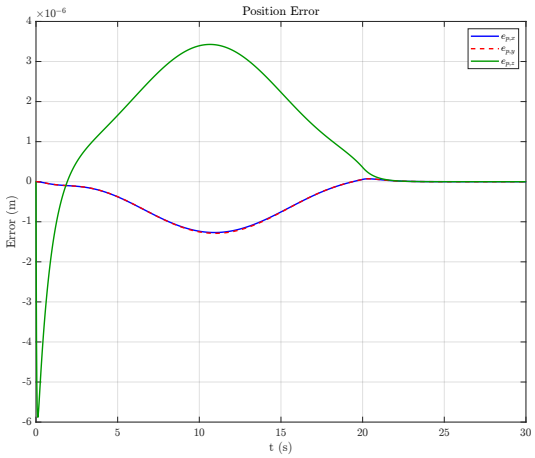
highlights the evolution of the yaw angle over time. The final desired orientation is correctly achieved, confirming the accuracy of the imposed angular trajectory. In the **Position Error** plot, the position error along each axis is shown. It presents an initial excursion that is still limited (on the order of 10^{-6} m), and it goes to zero over time, so a good convergence towards the desired trajectory. In the **Orientation Error** plot, the error $\mathbf{e}_R$ along the roll, pitch, and yaw components is represented. Although slight initial oscillations are present, the errors always remain on the order of 10^{-6} rad and converge to zero, demonstrating the accuracy of the inner controller in tracking the desired attitude configuration. The **Linear Velocity Error** plot shows the error on the linear velocity. The initial error values are also limited (around 10^{-5} m/s) and tend to stabilize over time, with full settling occurring around 15 seconds. This indicates a damped behavior of the outer controller. The **Angular Velocity Error** plot shows the error between the current and desired angular velocity. The errors remain within $\pm 1 \times 10^{-5}$ rad/s and show good convergence, consistent with accurately controlled dynamics. The **Thrust** u_T plot reports the trend of the total thrust generated by the outer controller. The thrust value remains almost constant around 12 N (between 11.7 and 12), consistent with hovering flight or small accelerations. This indicates an effective balance between weight force and imposed accelerations. Finally, the **Torque** $\boldsymbol{\tau}_b$ plot shows the trend of the control torque components applied along the body axes. The torques remain within a narrow range (± 0.02 Nm), confirming that the system maintains the desired orientation with limited effort. This is indicative of smooth and well-controlled flight.



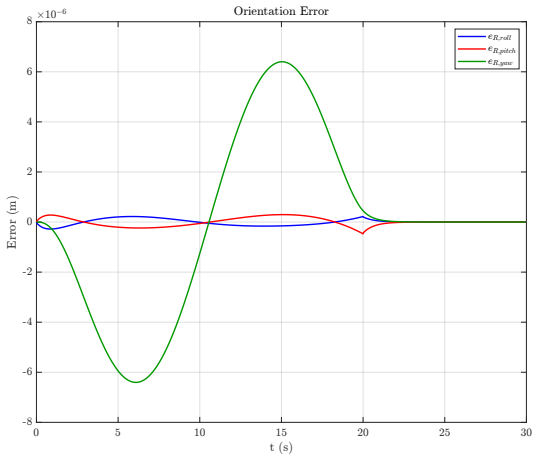
(a) Position tracking.



(b) Yaw angle tracking.

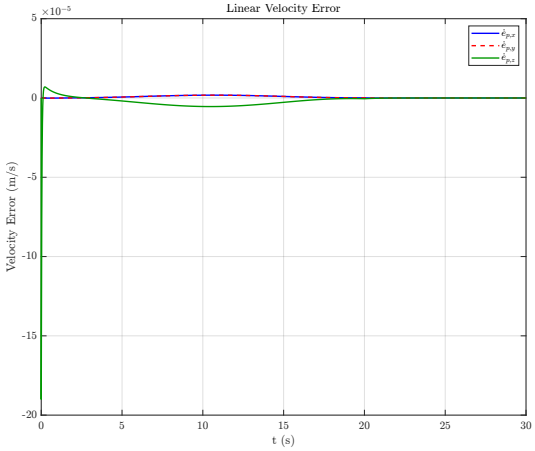


(c) Position error.

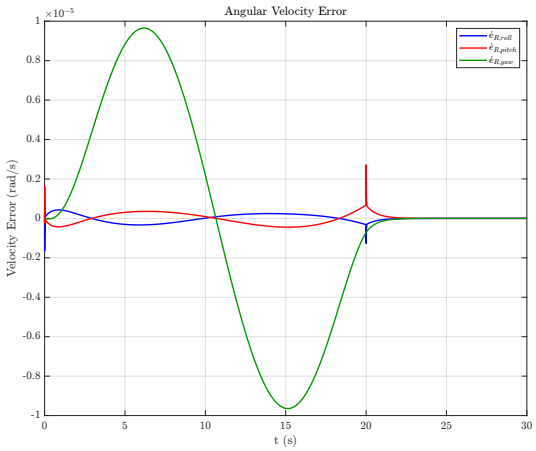


(d) Orientation Error.

Figure 4: Tracking performance and associated errors.

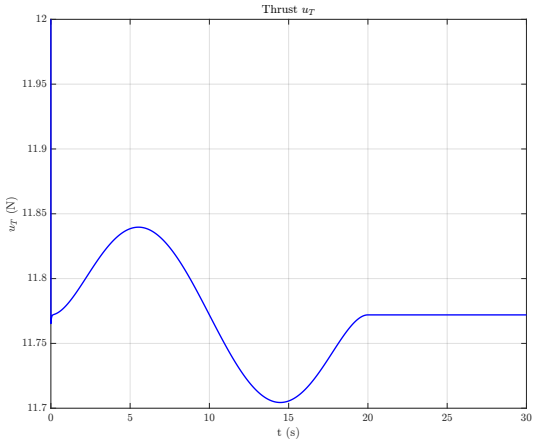


(a) Linear velocity error.

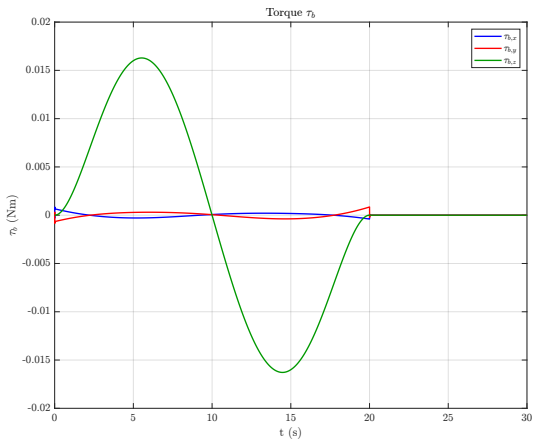


(b) Angular velocity error

Figure 5: Linear and Angualr velocities errors



(a) Thrust u_T.



(b) Control torque $\boldsymbol{\tau}_b$.

Figure 6: thrust and body torque.

5. Tilting UAV control

In this exercise, the control problem of a quadrotor with tiltable propellers is addressed using the *Voliro approach*. A Simulink model is provided, containing a partially completed simulation scheme. The goal is to complete the implementation of the controller, simulate the system behavior, and analyze the resulting data.

In tilting control, the main objective is to overcome the limitations imposed by the underactuation typical of conventional quadrotors. This becomes possible thanks to the ability to tilt the porpellers, that is, to introduce a tilting angle α_i for each rotor, thus obtaining a **fully actuated** system. In this new scenario, the allocation matrix depends on the tilt angles. In particular, the first two rows of the matrix, which were zero in the classical case, now include trigonometric terms such as $s_i = \sin(\alpha_i)$ and $c_i = \cos(\alpha_i)$, and the matrix itself is explicitly divided into two blocks: one related to forces and the other to torques. The angle α therefore becomes a new control variable, which allows direct action on components that were previously unmanageable, making the system fully actuated. However, the introduction of the input u_α – which controls the rate of change of the tilting angles α introduces a structural issue: since it acts at a higher derivative level than u_ω (the rotor angular velocities), it does not appear directly in the second-order dynamic equations. As a result, the extended allocation matrix contains zero columns, compromising its rank and once again leading to an underactuated condition. To overcome this limitation, several

approaches exist. One consists of rewriting the dynamic model at the jerk level, but in this exercise, the *Voliro approach* has been chosen.

Voliro approach Simulink implementation

The Voliro approach is based on a nonlinear transformation of the allocation problem into a linear one through a change of variables. In practice, new control inputs are introduced, defined as:

$$u_{v,i} = c_f \, u_{\omega,i} \cos(\alpha_i), \quad u_{l,i} = c_f \, u_{\omega,i} \sin(\alpha_i)$$

for each $i = 1, \ldots, 4$. The control was implemented within the MATLAB function `Voliro`, called by the Simulink scheme. The main objective is to generate the control inputs – namely the rotor angular velocities u_ω and the tilting angles α – required for the drone to follow a desired trajectory in terms of position and orientation, as defined by the planner. The function receives as input the current state of the drone, including position, linear velocity, rotation matrix R_b, and angular velocity, along with the desired state provided by the planner block and the physical parameters of the system: mass, inertia matrix I_b, distance of the motors from the center l, and aerodynamic coefficients c_f and c_m. In the control implemented using a PD with feedforward action, the process begins with the computation of tracking errors and their derivatives. In particular:

$$e_p = p - p_d \quad \dot{e}_p = \dot{p} - \dot{p}_d$$

$$\mathbf{e}_R = \frac{1}{2}(R_{b,d}^\top R_b - R_b^\top R_{b,d})^\vee \quad \mathbf{e}_\omega = \boldsymbol{\omega}_b - \mathbf{R}_b^\top \mathbf{R}_b^d \boldsymbol{\omega}_b^d$$

The mass and inertia of the system are combined into the matrix $M = \left[\begin{smallmatrix} m \cdot I_3 & 0 \\ 0 & I_b \end{smallmatrix}\right]$.
The g term is given by $g = \left[\begin{smallmatrix} mge_3 \\ -S(\omega_b) \cdot I_b \cdot \omega_b \end{smallmatrix}\right]$. The allocation matrix :

$$G_{q,\text{static}} = G \in \mathbb{R}^{6\times 8}$$

The vector v:

$$v = \begin{bmatrix} \ddot{p}_d - K_p e_p - K_d \dot{e}_p \\ \dot{\omega}_d^b - K_R e_R - K_\omega e_\omega \end{bmatrix}$$

The controller gains were selected to ensure a balance between responsiveness and stability. In particular, the gains $K_p = \text{diag}(10, 10, 10)$ and $K_d = \text{diag}(6, 6, 6)$ enable good tracking of the desired position trajectory, with effective damping of velocity errors. For the rotational dynamics, the gains $K_R = \text{diag}(8, 8, 8)$ and $K_\omega = \text{diag}(6, 6, 4)$ ensure smooth and stable convergence to the desired orientation. The dynamic model is expressed as:

$$A = M^{-1} \begin{bmatrix} R & 0 \\ 0 & I_3 \end{bmatrix} G, \qquad b = M^{-1} g$$

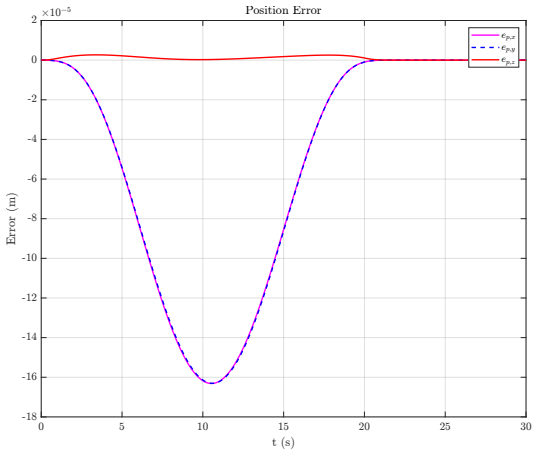
The virtual input $u \in \mathbb{R}^8$ is computed by solving: $u = A^\dagger (v - b)$. Finally, the new inputs required by the Voliro approach are computed as:

$$u_{\omega,i} = \frac{1}{c_f} \sqrt{u_{l,i}^2 + u_{v,i}^2}, \quad \alpha_i = \text{atan2}(u_{l,i}, u_{v,i}), \quad \text{for } i = 1, \ldots, 4$$

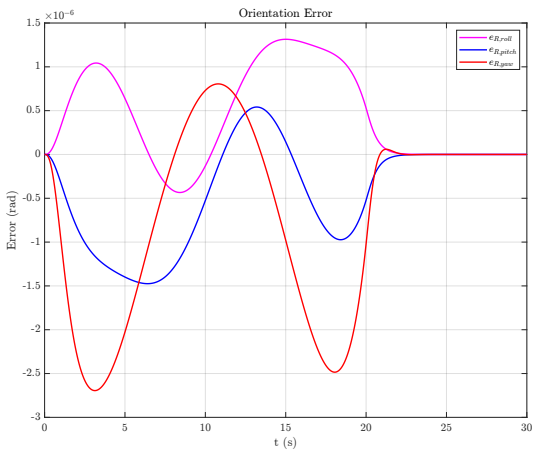
These represent, respectively, the angular velocity of each rotor and the corresponding tilting angle.

Plots

Position error: the component z shows error that remain very close to zero, while the x and y components show a more pronounced error, still on the order of 10^{-5} m, which progressively reducing and vanishing. This behavior is consistent with a maneuver along the horizontal plane, driven by the tilting control strategy that prioritizes fast repositioning in x and y.**Orientation error:** the components of the orientation error start from values initially different from zero, though still small (on the order of 10^{-6}), and gradually converge to zero. This behavior confirms that the controller is able to smoothly correct initial attitude deviations. **Linear velocity error:** shows oscillations around zero, also small (on the order of 10^{-5} m/s), which diminish over time until disappearing. **Angular velocity error:** although it starts with small initial peaks (still on the order of 10^{-6} rad/s), it progressively decreases as well. **Position tracking:** this plot shows the actual position in the three spatial components x, y, and z, compared with the desired trajectory over the 30 seconds of the simulation. The tracking is accurate, with the actual trajectories closely following the reference. **Orientation tracking:** the roll varies approximately between $-20°$ and $40°$, the pitch between $0°$ and $60°$, while the yaw spans a wide range from $-200°$ to $200°$, indicating complex rotational maneuvers around the vertical axis. In the yaw dynamics, an apparent discontinuity is observed starting around $t = 20$ s, with the desired trajectory jumping from $+180°$ to $-180°$. This behavior is due to the discontinuity introduced by the limited angular representation in the interval $[-180°, +180°]$. When the yaw angle exceeds $180°$, it is automatically wrapped to $-180°$ to avoid ambiguity. The windows observed in the actual trajectory reflect this numerical wrap-around and do not indicate a control error; the system correctly adapts to the discontinuity and continues to follow the desired orientation. **Control inputs:** the rotor speeds u_w are stable and symmetric, indicating a balanced thrust distribution. The tilting angles α of the motors, on the other hand, show oscillations between approximately ± 1.5 rad, confirming that the system actively uses the tilting motors to orient the thrust direction as expected in the Voliro approach. During the simulation, the tilting-motor quadrotor follows a dynamic three-dimensional trajectory. In the initial phase, a rapid displacement along the horizontal plane is observed, enabled by motor tilting and the resulting reorientation of the thrust. The vertical position along the z axis is maintained with high precision. During the maneuver, the drone adjusts its spatial orientation, performing rotations, particularly around the yaw axis, to correctly align with the desired trajectory. In the final phase, the vehicle stabilizes both position and orientation, remaining close to the target configuration. Position and orientation errors progressively cancel out, while rotor speeds and tilting angles converge to stable values, confirming the effectiveness and stability of the control strategy.

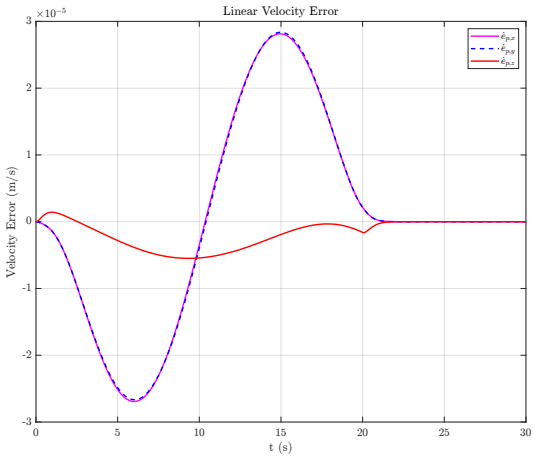


(a) Position error.

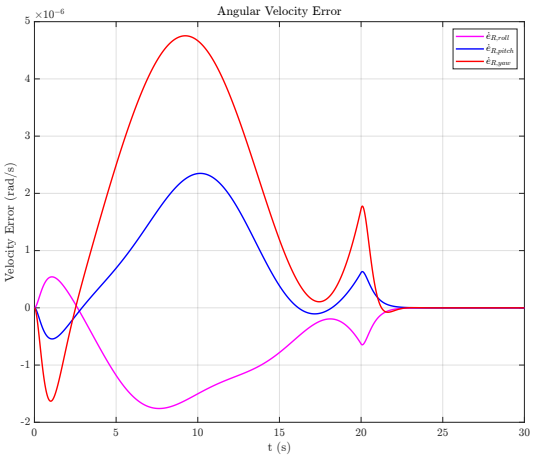


(b) Orientation error.

Figure 7: Tracking errors in position and orientation.

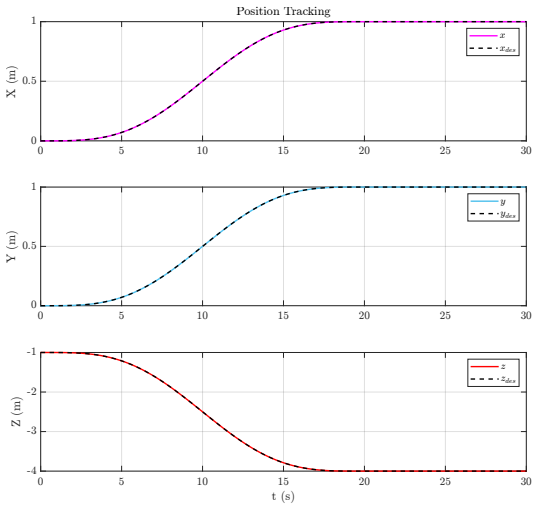


(a) Linear velocity error.

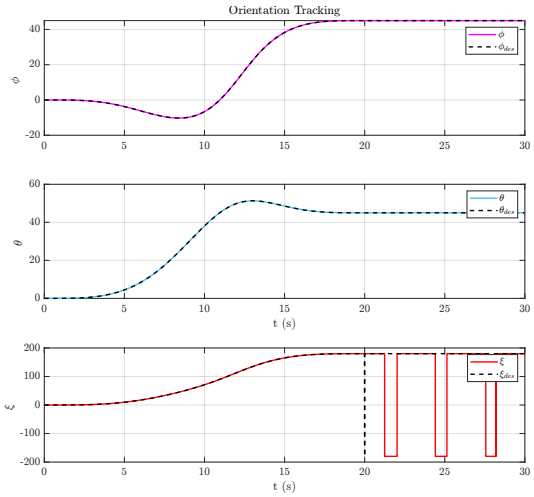


(b) Angular velocity error.

Figure 8: Tracking errors in linear and angular velocities.

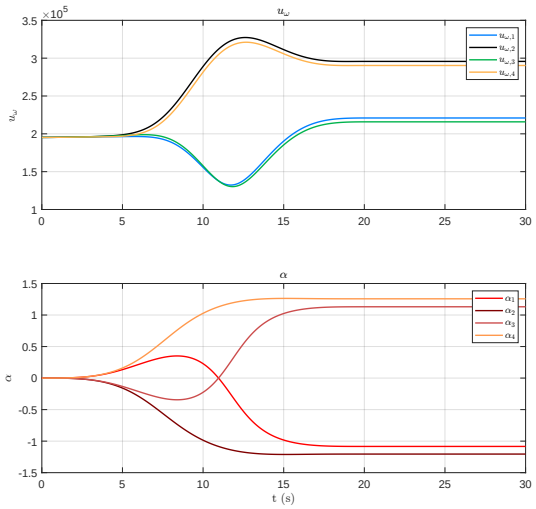


(a) Position tracking.



(b) Orientation tracking (roll, pitch, yaw).

Figure 9: Tracking of actual vs. desired trajectories in position and orientation.



(a) Rotor speed and tilting angles.

Figure 10: Control inputs to the tilting propellers.