Field and Service Robotics
Final Project

Planning and Control for a Fully Actuated Hexacopter

Federico Esposito P38000142
Riccardo Aliotta P38000143

July 2023

Contents

1 Introduction

The aim of this paper is to study a fully actuated tilted hexacopter and to then plan a trajectory for it and control it using several control techniques.

The subject of this study has been inspired by Voyles et al. [1], that described how the addition of a fixed cant angle in the orientation of a hexarotor's propellers could let it reach a full actuation with the minimum number of actuators.

Having 6 propellers which aren't coplanar, the hexarotor is able to generate accelerations both along and around its three axes. A mathematical proof of that will be provided in Chapter 2, where we will show that the allocation matrix of the robot is 6×6 and is full rank.

The goal we've imagined for the robot is that of Inspection and Maintenance in an industrial environment: the robot will inspect a machine, then fly back to its ground station avoiding pipes and walls which separate the different rooms. The environment will obviously be approximated for the sake of simplicity.

The paper will start with a more in depth geometrical description of the robot and with the mathematical steps needed to compute the aforementioned allocation matrix. After that we will focus on the path and trajectory planning in 3D space, using a modified RRT method and then high order polynomials to plan all the time derivatives needed. Chapter 4 will instead describe and compare the different control techniques we've applied on the robot: slightly modified versions of the three controllers seen in class and then a full Feedback Linearization Controller which makes use of the drone's full actuation.

2 Allocation Matrix

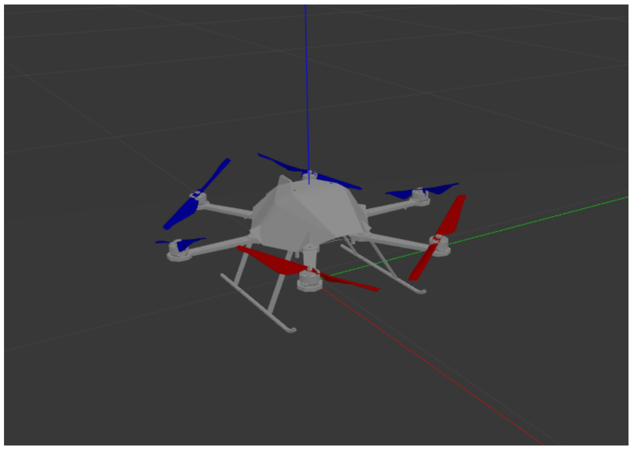

Figure 1: Tilted hexacopter as seen in Gazebo.

The robot we've considered merges the concepts in Voyles et al. [1] with the structure of the Firefly coplanar hexacopter present in the ROS package *RotorS*. This means that we have assumed that the radial distance from the center of the robot and each one of the propellers is $l = 0.215\,m$, that the propellers have a vertical displacement of $z_{disp} = 0.037\,m$ going up from the center of mass of the robot, and that the cant angle is of $\beta = \pm 20°$.

The cant angle represents the rotation that each propeller was given around the radius that connects it to the center of the robot. A null cant angle would result in coplanar propellers. For symmetry adjacent propellers have been canted with opposite angles, starting with the propeller denoted as "0" in the .urdf file having a cant angle of $-20°$. In Figure 1 this propeller is the red one on the left, and it spins counterclockwise. As always we assume that adjacent propellers spin in opposite directions.

We also need to know how the rotation of a propeller translates into forces and torques: to do so we need the thrust factor and the drag factor, which luckily we could derive from the .urdf file of the drone we considered: we found that the thrust factor is $cT = 8.54858 \cdot 10^{-6}\,kg\,m/s^2$, while the ratio between the drag factor and the thrust factor is $\gamma = 0.016\,m$.

With this information it's now possible to start the mathematical derivation of the allocation matrix.

First we assume that the body frame is NED, with the x-axis along the bisector of the angle formed by the two propellers in red in Figure 1. We also add a reference frame for each propeller, with the z-axis along the positive direction of the thrust and the x-axis along the radius, pointing outside the robot.

The following computations will be guided by Rashad et al. [2].

For the time being, we will consider that the wrench W is a column vector that includes torques and forces, in this order; at the end of the chapter, for consistency with the conventions used in class, we will swap the first and last rows of the matrix to use a wrench vector that starts with forces and ends with torques.

Rather than mapping the wrench directly to the square of the rotor speeds, Rashad et al. [2] finds a mapping between the thrusts generated by the rotors and the wrench that the UAV feels.

The i-th column of the matrix as defined in the reference paper is:

$$\begin{bmatrix} \mathrm{S}(\xi_\mathrm{i})\,\mathrm{R}_\mathrm{i}\,\hat{z} + \gamma\,(-1)^{i+1}\,\mathrm{R}_\mathrm{i}\,\hat{z} \\ \mathrm{R}_\mathrm{i}\,\hat{z} \end{bmatrix}, \quad i = 0, \cdots, 5$$

Where ξ_i is the position of the i-th propeller in the body frame, S indicates the skew-symmetric operator,

R_i is the rotation matrix that describes the orientation of the i-th propeller in body frame, $\hat{z}$ is the vector $[0\,0\,1]^T$ and the term $(-1)^{i+1}$ is needed to account for the alternating rotation directions of the propellers.

Both ξ_i and R_i have been computed with the help of the *Rviz* and the *tf* ROS package, although they could also be obtained manually considering that the propellers are spaced 60° from each other, with the first one being at 30° from the x-axis, and that the physical dimensions and the cant angle are known.

After changing the order of the columns we get:

$$\begin{bmatrix} 0.1710 & -0.3420 & 0.1710 & 0.1710 & -0.3420 & 0.1710 \\ 0.2962 & 0 & -0.2962 & 0.2962 & 0 & -0.2962 \\ -0.9397 & -0.9397 & -0.9397 & -0.9397 & -0.9397 & -0.9397 \\ 0.1092 & 0.1966 & 0.0873 & -0.0873 & -0.1966 & -0.1092 \\ 0.1639 & 0.0127 & -0.1766 & -0.1766 & 0.0127 & 0.1639 \\ 0.0886 & -0.0886 & 0.0886 & -0.0886 & 0.0886 & -0.0886 \end{bmatrix} \tag{1}$$

Which is a constant 6×6 matrix of full rank, proving the full actuation mentioned before.

If we want to convert the desired control wrench in rotor speeds we have to multiply the inverse of this matrix by the desired wrench to get the vector of the rotor thrusts, then we must divide the thrusts by cT to get the squares of the velocities, and finally we can extract the root squares to get the velocities of the rotors. In case one of the rotor thrusts ends up being negative we have to invert the rotation of the propeller, so we use the absolute value when computing the square root and then change the sign at the end.

The 3th row of the matrix shows that, if all the propellers spin in their positive direction, the UAV feels a negative force on the z-axis; this makes sense, since we're using a NED frame. In the controllers, however, we'll use a positive force on z to go upwards. To compensate for this, and for the similar effects on the 1st and 2nd row, we will need to invert the sign of the control forces (but not the control torques) before using this matrix to compute the rotor speeds.

3 Environment and Planning

As already said, the purpose imagined for the drone is that of Inspection and Maintenance, but while in Voyles et al. [1] the drone was bigger and equipped with a parallel 6-DoF manipulator, in our case we assumed the robot's task is only that of inspection (mainly due to its modest dimensions, which would make it hard to carry a manipulator).

The algorithm applied for the path planning is a 3D extension of the *Bilateral RRT* developed during the course with a few minor variations from the standard one. For the environment description, instead, a simulation of a LIDAR generated cloud point has been obtained by using custom meshes, and then the *OccupancyMap* class in *MATLAB* has been used to visualize it, as it also allowed to *inflate* the obstacles to account for the drone's dimensions.

After having generated the path, the trajectory has to be computed. In our case, we have implemented that by using 5-th order polynomials, assuming that the drone fully stops in every point generated through RRT, while for the angle's specifications we assumed only to vary the orientation a fixed amount every fixed point. We will now go into more detail about the 3D Path planning and the Trajectory planning.

3.1 Environment and Path planning

An extremely simplified, industrial-like environment has been described to simulate how the drone would move in real applications: a wide room with a cuboidal obstacle, which could be a piece of machinery to be inspected, a wall with a rectangular opening and a full-height cylinder in front of it, assumed to be a pipe; on the other side we assumed to be open grounds, so free of obstacles. The starting point is placed at a height from the floor, nearing the cuboidal obstacle, as if mid inspection, while the goal is on the other side of the wall, where our ground station could be placed.

The *meshes* have been defined by hand to describe the obstacles' structure, and placed in the environment as to create a non-trivial path. After the creation of the *OccupancyMap (OMap)* and its inflation to account for the actual drone dimensions, a discretized map has been generated by checking certain positions for occupation. This has been done to connect the path planning algorithm to the newly generated map, which work with different types of data.

For the RRT, the checks are made over a *3D matrix* of dimensions varying on the environment's size (20 m x 20 m x 10 m) and on a chosen resolution (as definitive parameters we have chosen a 0.25 m resolution, leading to an 80x80x40 matrix). We then take a random point in the matrix, identify the closest point to it (initially only the goal and the starting point are part of the *roadmap*) and proceed to generate the actual target at a set distance δ from the latter. If no collision is detected on the line connecting the two, then the target is added to the roadmap, else it is discarded. This leads to the creation of two subtrees, one from the goal and one from the starting point.

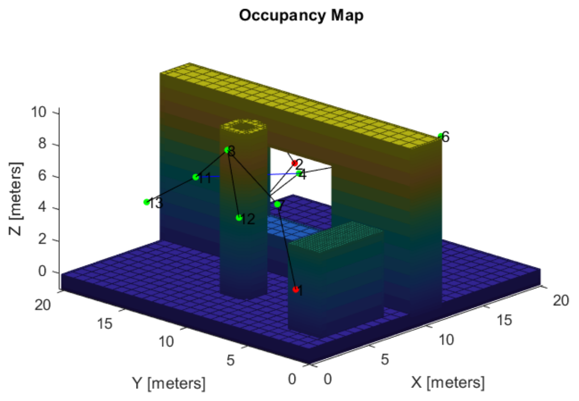

Figure 2: Resulting path in the described environment

After a number of iterations, the algorithm tries to connect the two closest points of the two different subtrees: if a collision-free connection is found, the algorithm stops, else it performs more iterations. If after a maximum number of iterations no result is achieved, the algorithm stops. To test consistency of the algorithm, a path with a set seed (*rng(1)*) has been generated while debugging, and then a final version has been tested with numerous random seeds.

3.2 Trajectory planning

On the points generated by the Path planner, the Trajectory planner generates the time law that the drone has to track. In reality, the points generated contribute only to the translational part of the drone, while the angular part has been designed in a different, simpler way: from point to point, the value of the controllable angles is increased by a fixed value (in this case, 5°) with a constant 'spacial' rate, which of course applies only to the yaw for every controller, except for the Feedback Linearization Controller that accounts for the full actuation, in which all angles are controllable.

For the trajectory planning itself, the chosen method is the 5-th order polynomial to obtain the time law of the path. To simplify the trajectory, in every fixed point the drone will stop, generating a continuous and easy to track path, even though it will not be the fastest. Every segment connecting two fixed points will be covered by the drone in the same time, set in our case as 5 seconds. After reaching the goal, the drone assumes to be stationary for 20 seconds, which will give us the chance to properly see what happens in the steady state.

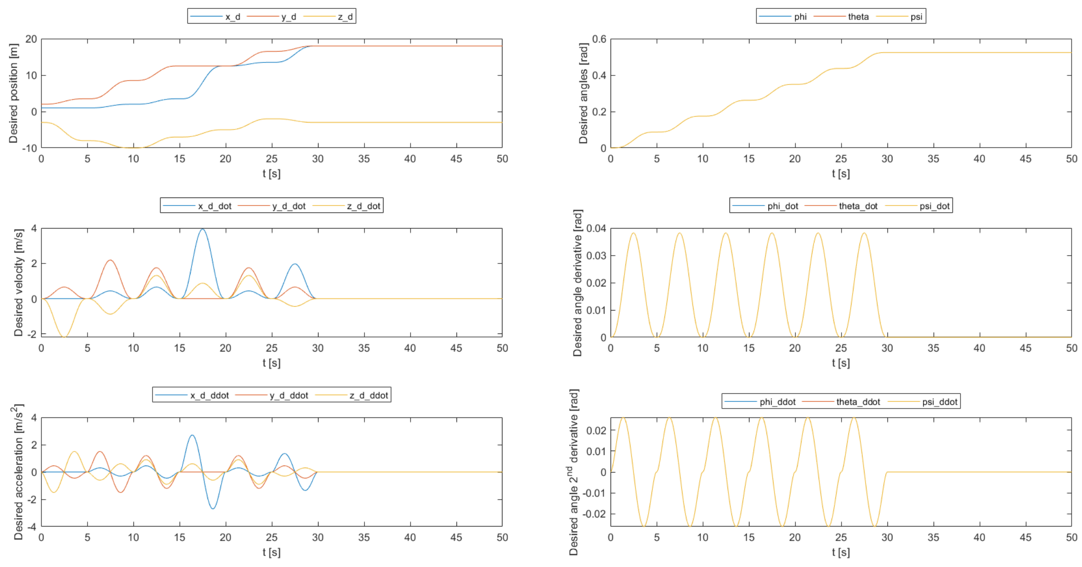

Figure 3: Resulting trajectory from the computed path. The first plot shows the desired x, y and z positions; the second plot shows the desired ϕ, θ and ψ angles (they are planned in the same way, so their values overlap); the third plot shows the desired velocities on x, y and z; the fourth plot shows the first derivative of the desired angles, the fifth plot shows the desired accelerations on x, y and z; the sixth plot shows the second derivative of the desired angles

4 Controllers

To control the drone, first we had a look at the already seen control schemes for underactuated multicopters, in particular Geometric Control, Hierarchical Control and Passivity-based Control, expanding on the basic concept by applying them on a tilted configuration hexacopter, and then by robustifying them with the addition of estimators and integral action terms. To use these control schemes on our model we simply assign a null value to the x and y components of the control force u_D, and use the control input u_T as the component along z.

After that, a new controller was designed by us to account for and exploit the full actuation of the drone via a Feedback Linearization of the drone model. In this case, too, the estimator and integral actions have been added to robustify the control. Some parameters remained the same across all controllers, but in some cases some tweaking and tuning was necessarily different case by case.

It is noteworthy that all the external disturbances applied to the system are in world frame, and so are the estimated wrenches; the only exception to this is in the Geometric Controller, where it is easier to estimate the force in world frame but the torque in body frame.

The controller has been designed by keeping in mind the model of a Firefly drone, as can be seen in Figure 1, which has a mass $m = 1.56779\,kg$ and an inertia matrix $I_b = diag([0.0347563, 0.0458929, 0.0977])$, as stated on the .urdf file of the model.

4.1 Geometric Control

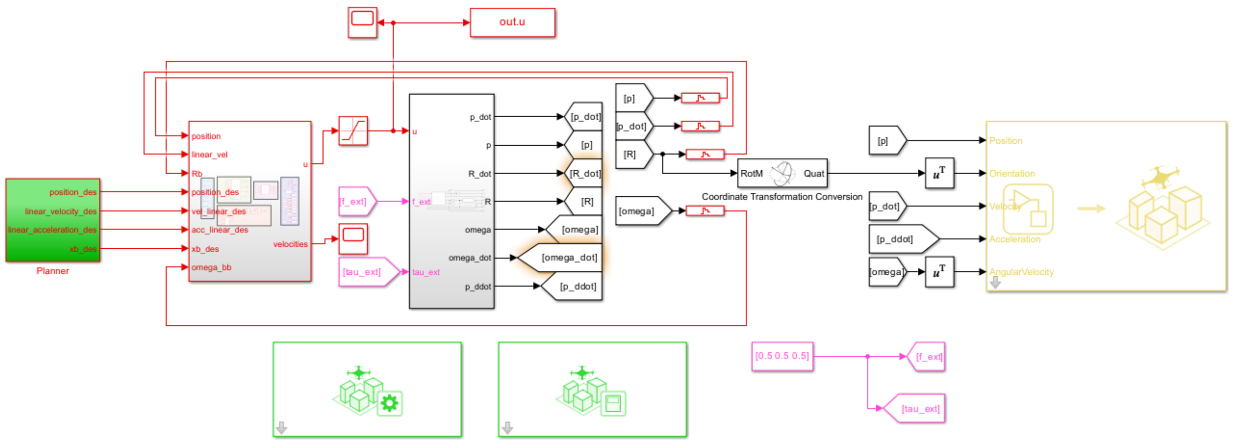

Figure 4: Overview of the control scheme used for the Geometric Control.

Figures 4, 5 and 6 show the *Simulink* model created to implement the Geometric Controller on the Coordinate-free UAV model.

In Figure 4 we have on the left the block that contains all the outputs of the planner, in the center the controller and the UAV model, while on the right and in the bottom part of the image we have some *Simulink* blocks needed for the visualization of the simulation in the 3D environment. The colours used in the figure highlight whether the block is working in discrete time (red and green, respectively associated with a sampling time of $1\,ms$ and of $0.1\,s$ for the planner, the controller and the blocks used for the 3D visualization of the UAV's movement) or in continuous time (black, for the UAV model); the pink is used for the constant external forces, while the yellow corresponds to the fact that one of the visualization blocks is multirate since it uses continuous inputs as well as the sampling time of $0.1\,s$ mentioned before.

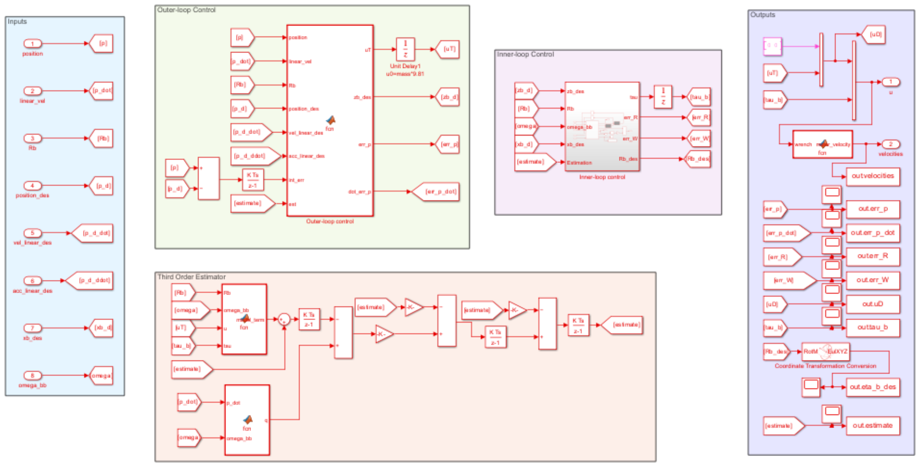

Figure 5: Control scheme used for the Geometric Control, divided in Outer-Loop for the linear part and Inner-Loop for the angular part.

In Figure 5 we can see the whole controller structure, divided in five subareas: while the cyan and the violet areas contain respectively input and output variables, including the computation of the rotor velocities using the allocation matrix obtained in Chapter 2, the light green and magenta ones contain the actual controller as they respectively incapsulate the outer-loop control and the inner-loop control; the orange area contains a third order estimator that robustifies the control to external disturbances and parametric uncertainty. Due to more sensitivity to parameter tuning, the disturbances applied while testing the Geometric Control were lower ($0.5\,N$ and $0.5\,Nm$ on the whole wrench) than those for the other schemes.

As already said, the controller is mainly divided in three parts: the outer-loop control, the inner-loop control and the estimator.

The estimator is in its typical form, in particular of the third order: starting from the Coordinate-free model (2), the momentum q and its derivative are computed, and from there (4) has been implemented in discrete-time fashion.

$$\begin{cases} m\ddot{p}_b = mge_3 - R_b u_D + f_e \\ I_b \omega_b^b = -S(\omega_b^b) I_b \omega_b^b + \tau^b + \tau_e^b \end{cases} \tag{2}$$

$$q = \begin{bmatrix} mI_3 & 0_3 \\ 0_3 & I_b \end{bmatrix} \begin{bmatrix} \dot{p}_b \\ \omega_b^b \end{bmatrix}, \quad \dot{q} = \begin{bmatrix} mge_3 - R_b u_D + f_e \\ S(\omega_b^b) I_b \omega_b^b + \tau^b + \tau_e^b \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} f_{est} \\ \tau_{est} \end{bmatrix} = K_3 \left(\int_0^t - \begin{bmatrix} f_{est} \\ \tau_{est} \end{bmatrix} + K_2 \left(\int_0^t - \begin{bmatrix} f_{est} \\ \tau_{est} \end{bmatrix} + K_1 \left(q - \int_0^t \begin{bmatrix} f_{est} \\ \tau_{est} \end{bmatrix} + \begin{bmatrix} mge_3 - R_b u_D \\ -S(\omega_b^b) I_b \omega_b^b + \tau^b \end{bmatrix} dt \right) dt \right) dt \right) \tag{4}$$

where the gains have been chosen as to get a dynamic with three poles in -50. The estimated values are then passed on to the inner-loop and outer-loop controls.

The outer-loop control takes as input the desired and actual position and velocity $p_{b,d}, p_b, \dot{p}_{b,d}, \dot{p}_b$, the body rotation matrix R_b, as well as the estimated wrench, the integral of the position error and the desired acceleration, from which it computes the desired thrust along the z axis u_T, as well as a vector describing the desired z axis of the drone $z_{b,d}$. While the former will directly be sent as an input to the model, the $z_{b,d}$ will be given as input to the inner-loop controller. The formulas applied into the *MATLAB function* are

$$u_T = -(-K_p e_p - K_v \dot{e}_p - \int_0^t K_i e_p \, dt - mge_3 + m\ddot{p}_{b,d})^T R_b e_3 \tag{5}$$

$$z_{b,d} = -\frac{-K_p e_p - K_v \dot{e}_p - \int_0^t K_i e_p \, dt - mge_3 + m\ddot{p}_{b,d}}{\| - K_p e_p - K_v \dot{e}_p - \int_0^t K_i e_p \, dt - mge_3 + m\ddot{p}_{b,d} \|} \tag{6}$$

where $e_p = p_b - p_{b,d}$ and $\dot{e}_p = \dot{p}_b - \dot{p}_{b,d}$ and the gains K_p, K_i and K_v were tuned through trial and error to achieve stability of the controller.

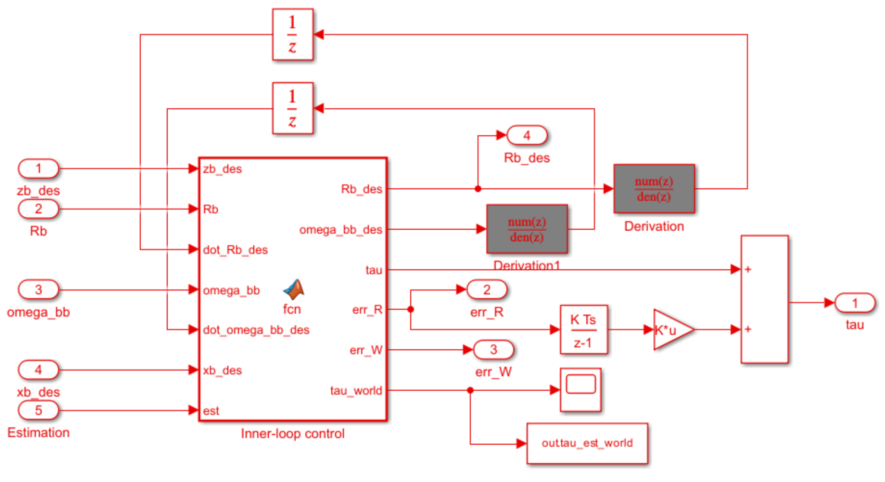

Figure 6: Block diagram of the Inner-Loop of the Geometric Control.

The inner-loop control takes as input R_b, $z_{b,d}$, the heading direction from the planner $x_{b,d}$, the body angular velocities expressed in body frame ω_b^b, the estimated wrench, as well as the derivatives of the desired body rotation matrix and body angular velocities $\dot{R}_{b,d}, \dot{\omega}_{b,d}^{b,d}$, and from these computes the desired body rotation matrix $R_{b,d}$, the desired body angular velocities $\omega_{b,d}^{b,d}$ and the control torques in body frame τ^b. The integral action is present also in this control loop, particularly on the computation of the control torque, but has been obtained via blocks external to the *MATLAB function*, nonetheless the mathematical formulas remain unvaried. The equations applied inside the *MATLAB function* block are

$$y_{b,d} = \frac{z_{b,d} \times x_{b,d}}{\|z_{b,d} \times x_{b,d}\|} \tag{7}$$

$$R_{b,d} = \left[S(y_{b,d}) z_{b,d} \ \frac{S(z_{b,d}) x_{b,d}}{\|S(z_{b,d}) x_{b,d}\|} \ z_{b,d} \right] \tag{8}$$

$$\omega_{b,d}^{b,d} = (R_{b,d}^T \dot{R}_{b,d})^V \tag{9}$$

$$\tau^b = -K_R e_R - K_\omega e_\omega - \int_0^t K_{i_{ang}} e_p \, dt + S(\omega_b^b) I_b \omega_b^b - I_b(S(\omega_b^b) R_b^T R_{b,d} \omega_{b,d}^{b,d} - R_b^T R_{b,d} \dot{\omega}_{b,d}^{b,d}) \tag{10}$$

where $e_R = \frac{1}{2}(R_{b,d}^T R_b - R_b^T R_{b,d})^V$ (the superscript V indicates the Vee operator, inverse of the Skew Symmetric operator $S(\omega)$), $e_\omega = \omega_b^b - R_b^T R_{b,d} \omega_{b,d}^{b,d}$, and the gains K_R, K_ω and $K_{i_{ang}}$ were tuned through trial and error together with the previous ones in order to achieve the controller's stability, resulting in the following values: $K_p = 100 I_3$, $K_v = diag([1\,1\,10]), K_i = 100 I_3$, $K_R = 2.5 I_3$, $K_\omega = 0.2 I_3$, $K_{i_{ang}} = I_3$. Increasing too much the gains lead to instability of the controller.

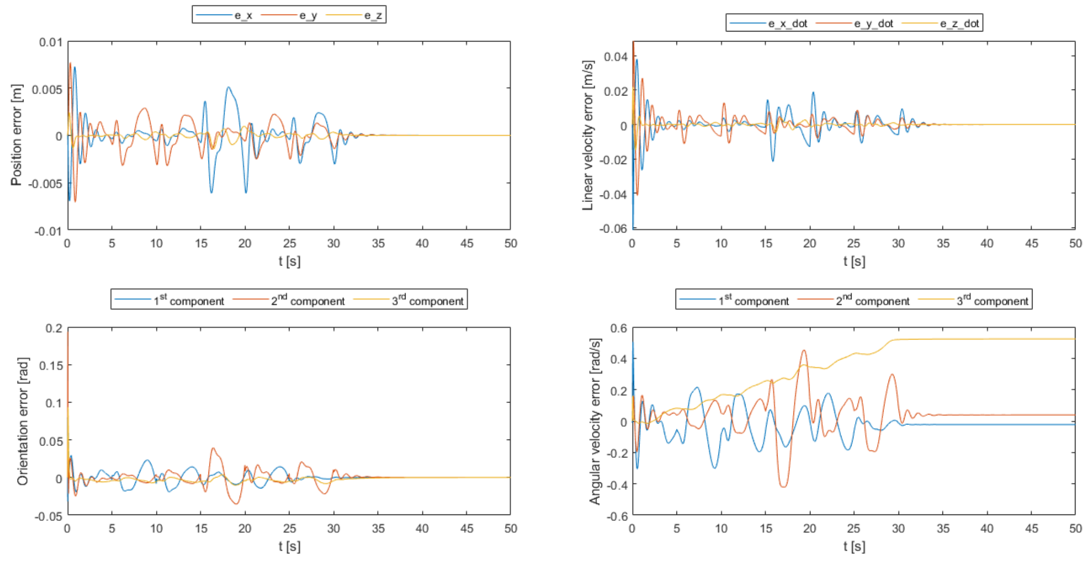

Figure 7: Control errors in the Geometric Control. The first plot shows the position error on the x, y and z axis; the second plot shows the velocity error on the x, y and z axis; the third plot shows the error associated to the rotation matrices; the fourth plot shows the error associated to the body angular velocities.

In Figure 7 it is possible to see the control errors on which the controller works. The position and linear velocity error are highly oscillatory, and the position error is kept under a centimeter, which for the intended application of the drone is more than acceptable. At steady state, after a little oscillation, all the errors converge to zero thanks to the integral actions and the estimator, except for the e_ω, which has a constant error: this however is most likely related to the non-easily comprehensible meaning of such measure, as in reality the orientation reached is fixed and exactly what was planned, as can be confirmed by the other error e_R. Also, as already stated during the course, constant errors at steady state on the angular part are acceptable for the Geometric Control.

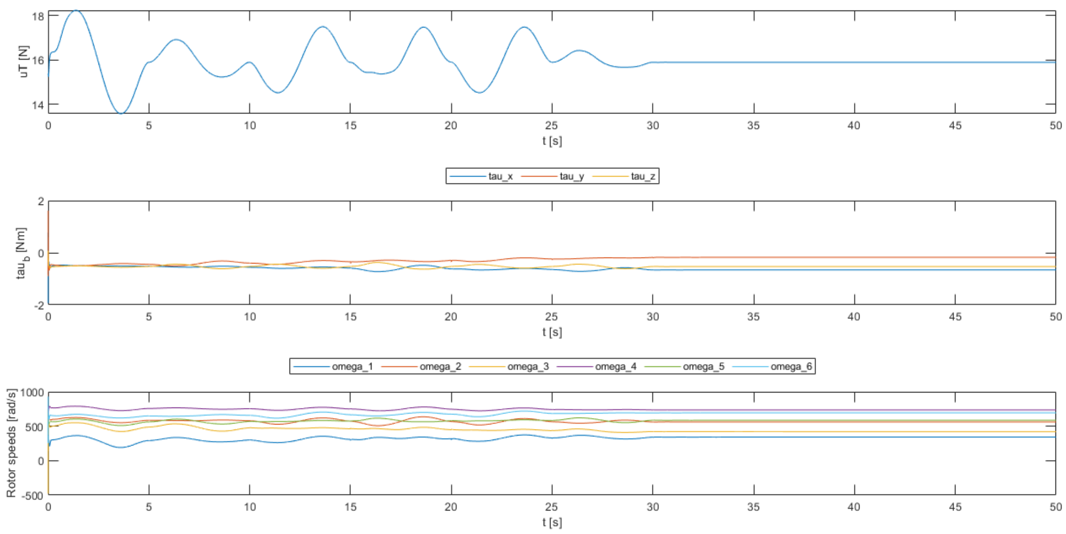

Figure 8: Control inputs in the Geometric Control. The first plot shows the thrust u_T; the second plot shows the control torques τ^b; the third plot shows the rotor speeds.

Figure 8 shows the control inputs that the controller generates in order to have the UAV follow the desired trajectory. We can see that, while the thrust has a smooth evolution without any spike, the control torques contain an initial spike, which however is sufficiently modest (less than 2 Nm); this has an effect also on the rotor speeds, which has a similar, negative spike on some rotor's speed. However, these speeds are all achievable, as according to the Firefly model we are referencing the limit top speeds are of about 1000 rad/s.

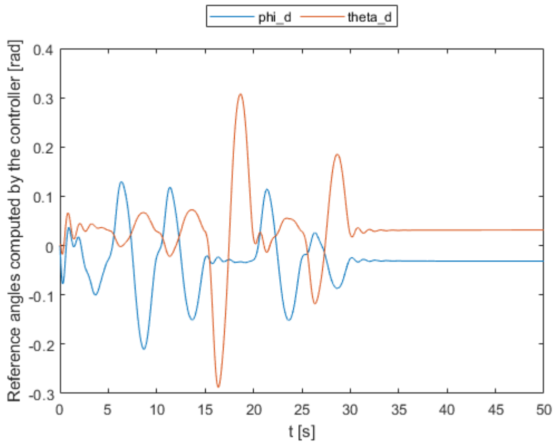

Figure 9: Reference angles generated in the Geometric Control.

Since we're ignoring the full actuation of the system, we only use the angle ψ from the planner, which is converted into a desired heading direction and have to instead compute the desired rotation matrix in the control loop. Figure 9 shows the evolution of the non controllable angles extracted from the computed desired rotation matrix.

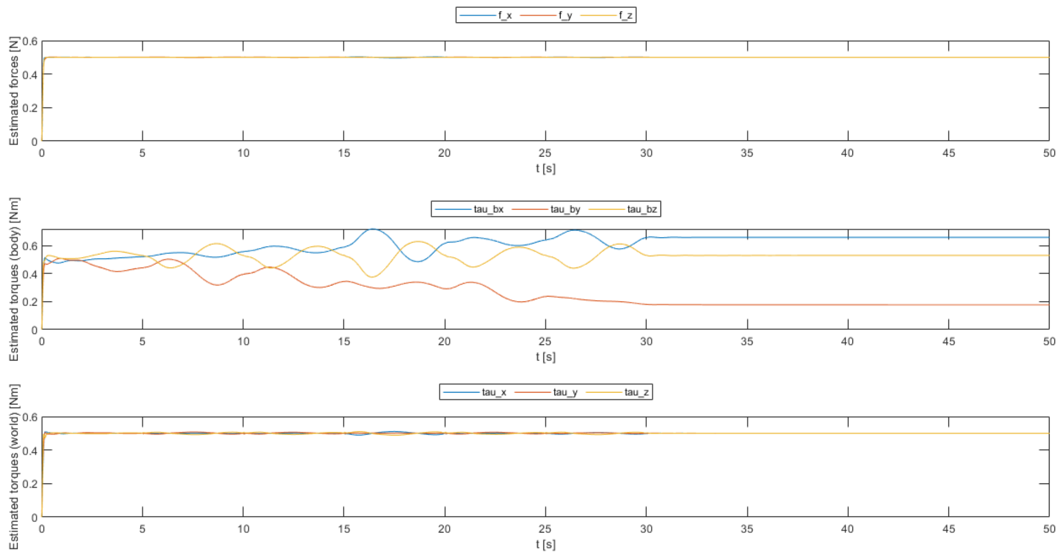

Figure 10: Estimate of the external wrench in the Geometric Control. Since the numerical values of the external force disturbances are the same, the plots overlap.

Finally, Figure 10 shows that the third order estimator is able to very quickly, and without overshoot, compute the external disturbances applied to the system, but as mentioned before the estimation of the torque is done in body frame, and not in world frame: moreover, we can assume that the small oscillations that can be seen on the world frame projection, which are in the order of $10^{-3}\,Nm$, are due to the presence in the transient of a non-zero e_R, which however converges to zero at steady state, as does the estimate in world frame.

4.2 Hierarchical Control

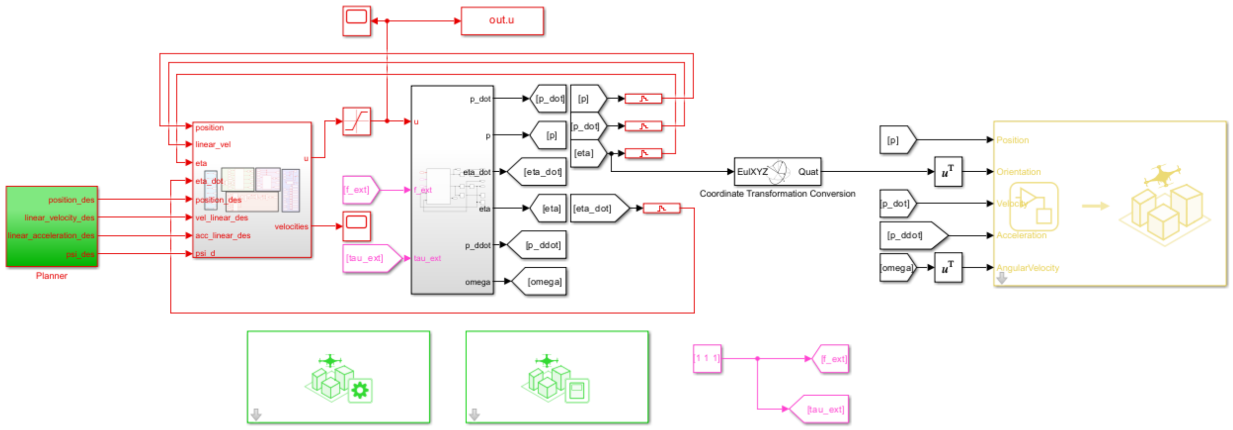

Figure 11: Overview of the control scheme used for the Hierarchical Control.

Figures 11, 12 and 13 show the *Simulink* model created to implement the Hierarchical Controller on the RPY UAV model.

The description of the overview of the scheme (Figure 11) results identical to that seen in the previous controller, with the only real difference being that the angular variables which are feedbacked from the model are this time referred to the Euler angles $\eta_b = [\phi\ \theta\ \psi]^T$ rather than to the rotation matrix as a whole.

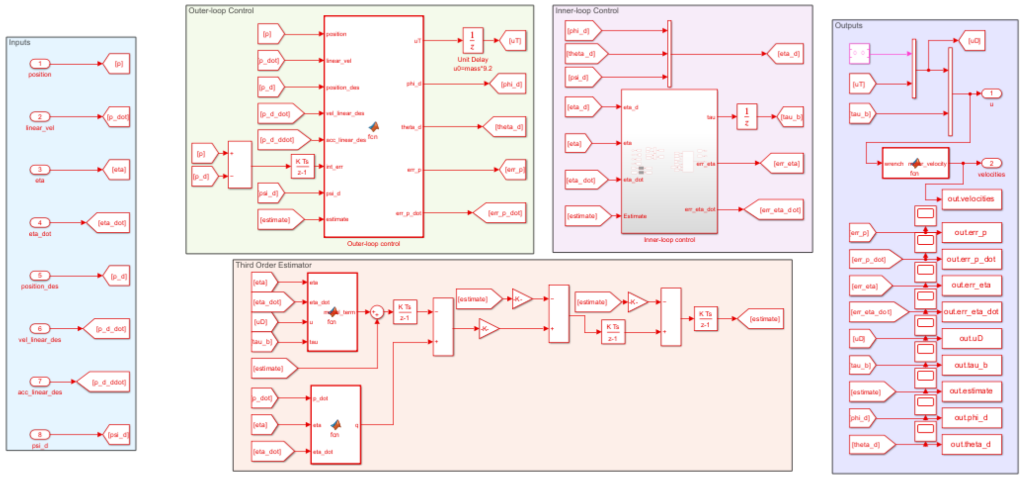

Figure 12: Control scheme used for the Hierarchical Control, divided in Outer-Loop for the linear part and Inner-Loop for the angular part.

The different subareas of the control scheme (Figure 12) keep the same role they had in the previously analyzed controller. This time the estimator gives result already in world frame, so comparing the estimates to the external wrench is trivial.

The outer-loop shown in the control scheme is realized with a single *MATLAB function* in which, from almost the same inputs as seen for the Geometric Control we compute the control force u_T and the angular references ϕ_d and θ_d that the inner-loop needs. The only difference in the inputs is that we don't use the rotation matrix R_b and that we instead need to use ψ_d to compute the other reference angles. The formulas we use for the control force and the angular references come from the differential flatness of a coplanar multicopter; to be able to use them for our tilted drone as well we force it to be underactuated by setting the x and y components of the control force u_D to zero. With the addition of an integral action, the equations of the outer-loop become:

$$\mu_d = -K_p \begin{bmatrix} e_p \\ \dot{e}_p \end{bmatrix} - K_i \int_0^t e_p dt + \ddot{p}_{b,d} - \frac{f_{est}}{m} \tag{11}$$

$$u_T = m\sqrt{{\mu_{d_x}}^2 + {\mu_{d_y}}^2 + (\mu_{d_z} - g)^2} \tag{12}$$

$$\phi_d = \arcsin\left[\frac{m}{u_T}\left(\mu_{d_y}\cos\psi_d - \mu_{d_x}\sin\psi_d\right)\right] \tag{13}$$

$$\theta_d = \arctan\left[\frac{\mu_{d_x}\cos\psi_d + \mu_{d_y}\sin\psi_d}{\mu_{d_z} - g}\right] \tag{14}$$

Where $e_p = p_b - p_{b,d}$, $\dot{e}_p = \dot{p}_b - \dot{p}_{b,d}$ and where the subscripts x, y and z are used to indicate the x, y and z component of the desired acceleration vector μ_d.

In order not to divide by zero, we have to saturate the z component of μ_d at a value that is slightly smaller of the 9.81 used for g.

The gain matrix K_p has been obtained placing a dominant pole in -2 on the error system seen in class that has the matrices:

$$A = \begin{bmatrix} 0_3 & I_3 \\ 0_3 & 0_3 \end{bmatrix} \tag{15}$$

$$B = \begin{bmatrix} 0_3 \\ I_3 \end{bmatrix} \tag{16}$$

The gain matrix K_i, instead, has been tuned through trial and error and has a value of $250I_3$.

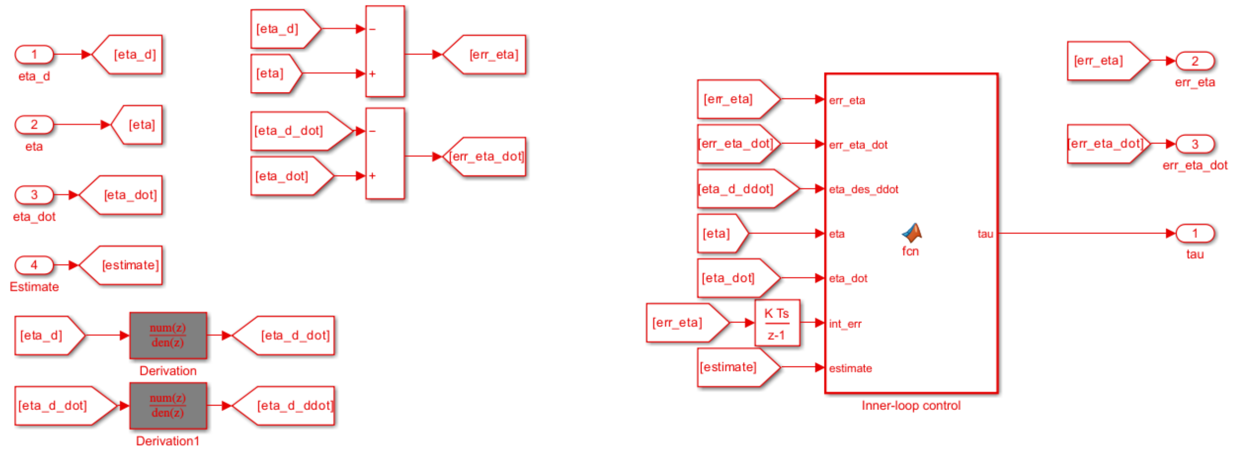

Figure 13: Block diagram of the Inner-Loop of the Hierarchical Control.

In Figure 13 we have the realization of the inner-loop controller; once again to guarantee a null error in the steady state we've added an integral action, so the feedback linearization action on the angular part becomes:

$$\tilde{\tau} = -K_e \begin{bmatrix} e_\eta \\ \dot{e}_\eta \end{bmatrix} - K_{i_{ang}} \int_0^t e_\eta dt + \ddot{\eta_{b,d}} \tag{17}$$

$$\tau^b = I_b Q \tilde{\tau} + Q^{-T} C \dot{\eta}_b - Q^T \tau_{est} \tag{18}$$

where $e_\eta = \eta_b - \eta_{b,d}$, $\dot{e}_\eta = \dot{\eta}_b - \dot{\eta}_{b,d}$, Q and C are respectively the conversion matrix between $\dot{\eta}$ and ω and the Coriolis matrix in the UAV model, K_e is a gain matrix which has been obtained similarly to the previously mentioned K_p, but choosing a dominant pole of -20, and finally $K_{i_{ang}}$ has been tuned with trial and error and has a value of $5000I_3$.

In Figure 13 we can see that the values of $\dot{\eta}_{b,d}$ and $\ddot{\eta}_{b,d}$ have been obtained by putting ϕ_d, θ_d and ψ_d in a *multiplexer* to then derivate them; in theory we could and should have used this derivatives just for ϕ and θ, since we already have the derivatives of ψ from the planner, however tests have shown that the derivatives of ψ that we compute in the inner-loop are the same as the ones that the planner provides, so there are no visible changes in using one over the other. We've decided to proceed in a similar way in the following controllers as well, in order to generate a less chaotic *Simulink* model.

The estimates f_{est} and τ_{est} come from the third order estimator, which works identically to what was described in the chapter about the Geometric Control, but obviously uses the RPY model in the computation of the model term and of the variable $\dot{q}$. Both estimates are in world frame.

The gains for the estimator in the RPY model have been chosen so that it realizes a transfer function with three poles in -100, which is slightly faster than what we've done in the Geometric Control, since we didn't have as many stability problems with the RPY controllers.

$$\begin{cases} m\ddot{p}_b = mge_3 - R_b u_D + f_e \\ M\ddot{\eta}_b = -C\dot{\eta}_b + Q^T\tau^b + \tau_e \end{cases} \tag{19}$$

$$q = \begin{bmatrix} mI_3 & 0_3 \\ 0_3 & I_b \end{bmatrix} \begin{bmatrix} \dot{p}_b \\ \omega_b^b \end{bmatrix}, \quad \dot{q} = \begin{bmatrix} mge_3 - R_b u_D + f_e \\ C^T\dot{\eta}_b + Q^T\tau^b + \tau_e \end{bmatrix} \tag{20}$$

$$\begin{bmatrix} f_{est} \\ \tau_{est} \end{bmatrix} = K_3\left(\int_0^t - \begin{bmatrix} f_{est} \\ \tau_{est} \end{bmatrix} + K_2\left(\int_0^t - \begin{bmatrix} f_{est} \\ \tau_{est} \end{bmatrix} + K_1\left(q - \int_0^t \begin{bmatrix} f_{est} \\ \tau_{est} \end{bmatrix} + \begin{bmatrix} mge_3 - R_b u_D \\ C^T\dot{\eta}_b + Q^T\tau^b \end{bmatrix} dt\right) dt\right) dt\right) \tag{21}$$

The physical parameters of the robot have been kept the same for all the controllers. In all the controllers which are based on the RPY UAV model the external disturbances have been set at $1\,N$ along all the axes and $1\,Nm$ around all the axes, but in testing we've also seen that the controller is able to reject values that go up to 100 without issues.

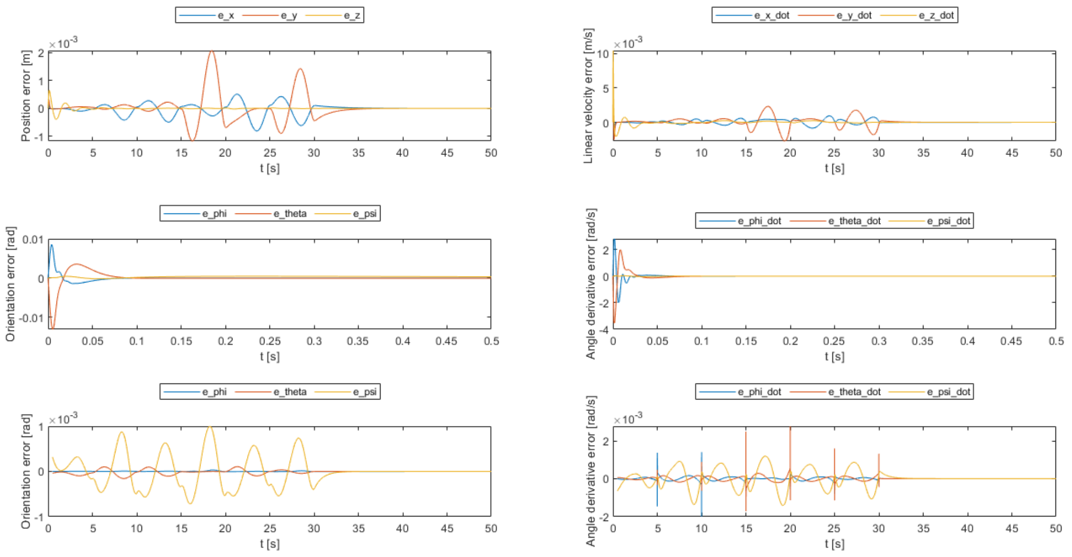

Figure 14: Control errors in the Hierarchical Control. The first plot shows the position error on the x, y and z axis; the second plot shows the velocity error on the x, y and z axis; the third plot shows the orientation error on ϕ, θ and ψ in the first $0.5\,s$; the fourth plot shows the angle derivative error on $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$ in the first $0.5\,s$; the fifth plot shows the continuation of the third one; the sixth plot shows the continuation of the fourth one.

In Figure 14 it is possible to see the control errors on which the controller works. We can see that the linear and angular position errors always stay low enough to be negligible, and that thanks to the integral actions and the estimator all the errors converge quickly to zero at steady state. For the orientation and angle derivative errors we've decided to split the plots in two, highlighting both the initial peak and then the low values which are reached shortly after. In the angle derivative errors we can also note some small peaks every $5\,s$, which are connected with the trajectory planning and could be further decreased by extending the time that the robot has to go from a point to the next.

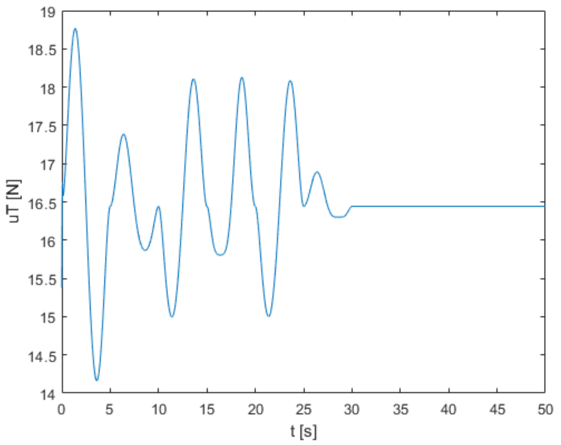

Figure 15: Control input u_T in the Hierarchical Control.

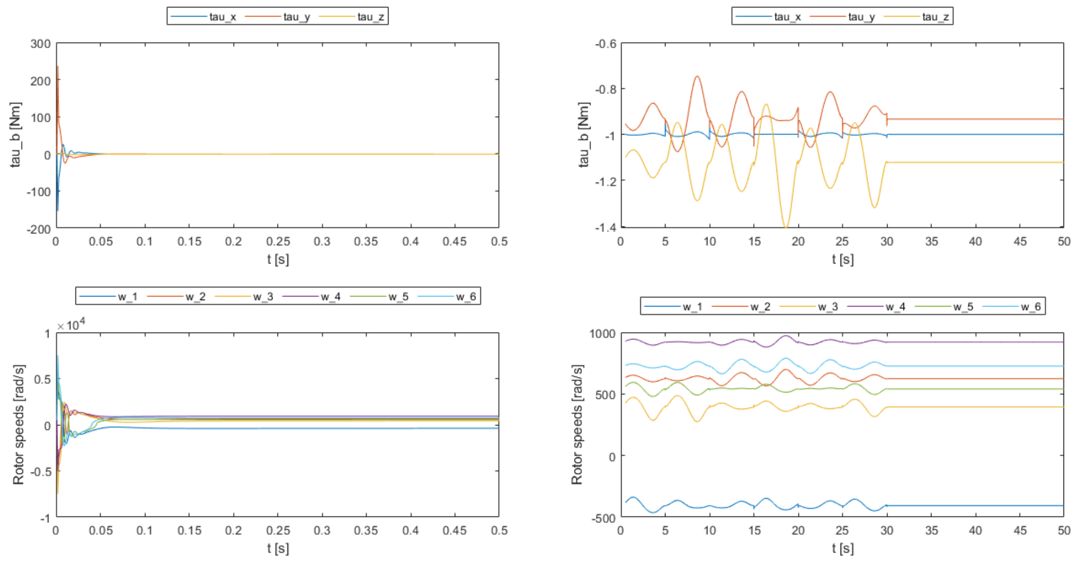

Figure 16: Other control inputs in the Hierarchical Control. The first plot shows the control torques τ^b in the first $0.5\,s$, while the second plot shows the continuation of that; the third plot shows the rotor speeds in the first $0.5\,s$, while the fourth plot shows the continuation of that.

Figures 15 and 16 show the control inputs that the controller generates in order to have the UAV follow the desired trajectory. It is possible to note an initial jump in u_T, caused by the fact that the force is initialized at a value equal to the weight of the robot, but thanks to the estimator the control loop quickly realizes it has to compensate the external disturbance as well. The control torques have a high peak at the start, which would obviously need to be saturated not to damage the actuators (and Figure 11 shows that a saturation between -50 and 50 has already been included in the *Simulink* model for both forces and torques), but after that their values stay low. The error peaks showed in Figure 14 cause the torques to have some sudden jumps as well; it is important to note, however, that these peaks don't correspond to discontinuities and in fact happen over the course of several samples. The computed rotor speeds follow a similar evolution to that of the control torques, and after an initial phase in which they'd need to be saturated we have values that could realistically be commanded to the drone we're considering.

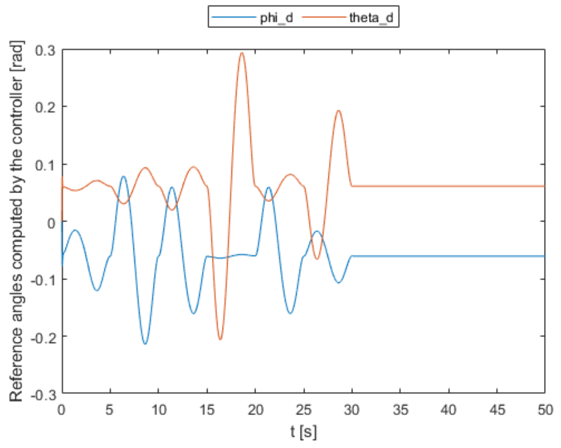

Figure 17: Reference angles generated in the Hierarchical Control.

As already said for the Geometric Control we are assuming not to exploit the full actuation, so we plan only ψ_d, while ϕ_d and θ_d (with which this time we operate directly) are computed in the outer-loop. Figure

17 shows the evolution of said computed angles, and from it we can also notice that we never end up even close to the singularity conditions on θ.

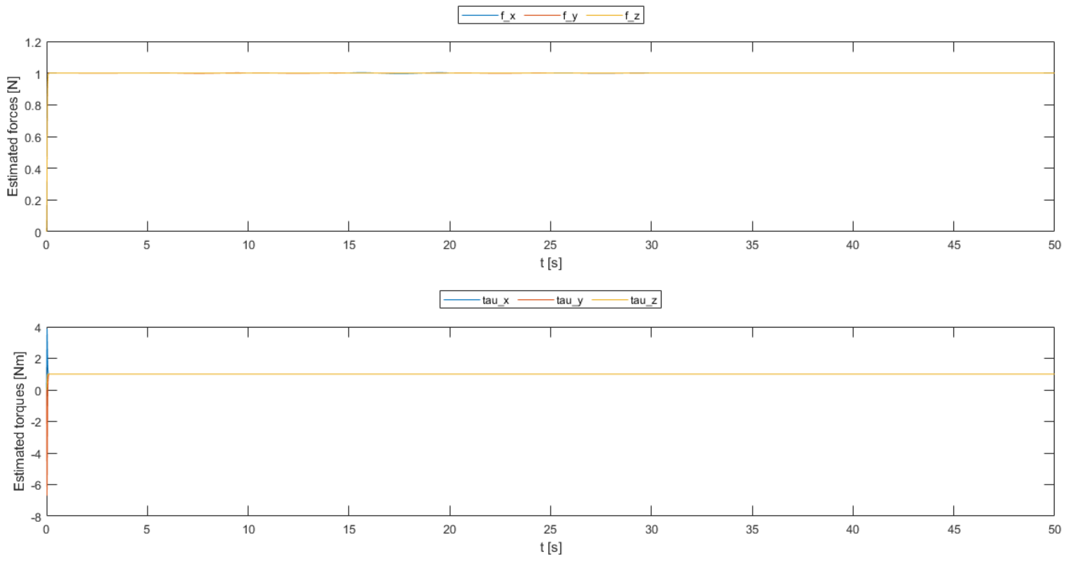

Figure 18: Estimate of the external wrench in the Hierarchical Control. Since the numerical values of the external disturbances are the same the plots overlap.

Finally, Figure 18 shows that the third order estimator is able to very quickly compute the external disturbances applied to the system. In the first moments of the simulation the estimate has a big peak, which is probably the cause of the larger peak on τ^b seen in Figure 16, but in about $0.1\,s$ the correct value is reached.

4.3 Passivity-based Control

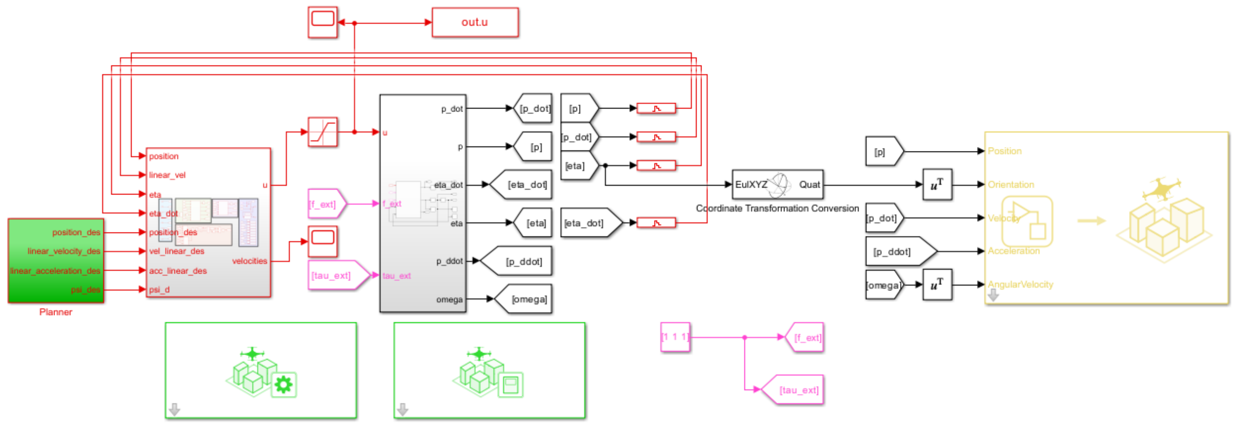

Figure 19: Overview of the control scheme used for the Passivity-based Control.

Figures 19, 20 and 21 show the *Simulink* model created to implement the Passivity-based Controller on the RPY UAV model.

The overall scheme (Figure 19) is identical to that seen in the Hierarchical Control, since we use the same UAV model. All the changes are hidden in the controller block.

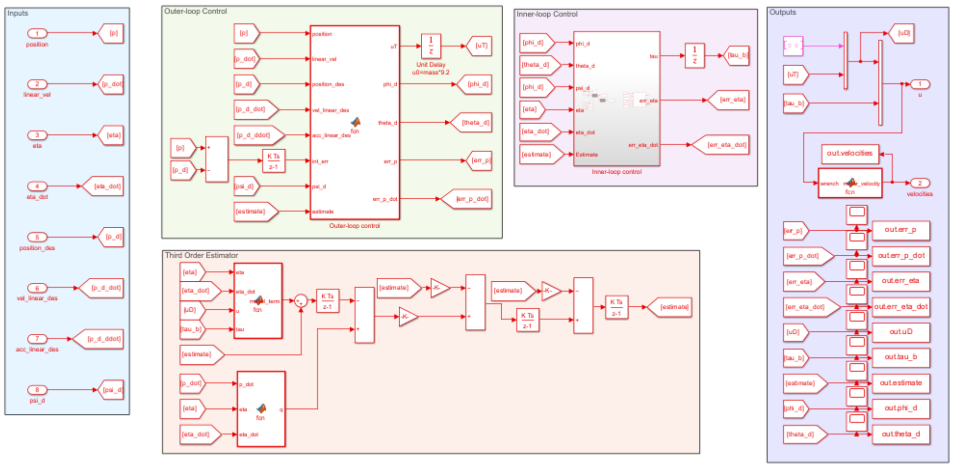

Figure 20: Control scheme used for the Passivity-based Control, divided in Outer-Loop for the linear part and Inner-Loop for the angular part.

Once again the subareas of the block scheme keep their previously described meaning.

The similarities between the Hierarchical and the Passivity-based Control continue as the entire outer-loop of both controllers is the same. We've also chosen to tune the gain matrices in the same way, to have performances that are as similar as possible.

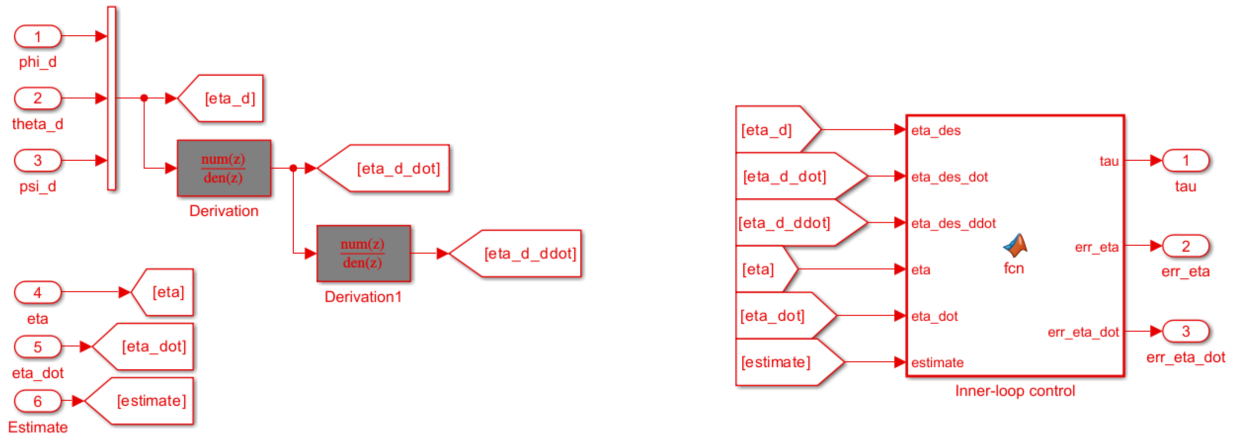

Figure 21: Block diagram of the Inner-Loop of the Passivity-based Control.

In Figure 21 we see the inner-loop of the Passivity-based Control, which is where it differentiates itself from the Hierarchical Control. One thing that can be already noticed from the image is the lack of an integral action: we've found that thanks to the passivity this controller is able to get a null angular error even with external disturbances, so adding an integrator was deemed superfluous.

The equation that guarantee the property of passivity in the inner-loop is:

$$\tau^b = Q^{-T}\left(M\ddot{\eta}_{b_r} + C\dot{\eta}_r - \tau_{est} - D_0\upsilon_\eta - K_0 e_\eta\right) \tag{22}$$

Where we need to define $\dot{\eta}_r = \dot{\eta}_{b,d} - \sigma e_\eta$, $\ddot{\eta}_r = \ddot{\eta}_{b,d} - \nu \dot{e}_\eta$, $e_\eta = \eta_b - \eta_{b,d}$, $\dot{e}_\eta = \dot{\eta}_b - \dot{\eta}_{b,d}$ and $\upsilon_\eta = \dot{e}_\eta + \sigma e_\eta$ as reference quantities for the controller.

As suggested in class, we've chosen $K_0 = \sigma D_0$, to reduce the number of parameters to tune. After some trial and error, the final values for the control gains are $\sigma = 10$, $\nu = 10$ and $D_0 = 10I_3$.

The estimator used for the Passivity-based Control is the same as the one seen in the Hierarchical Control. The simulation has been executed using the same parameters and disturbances seen in the Hierarchical Control.

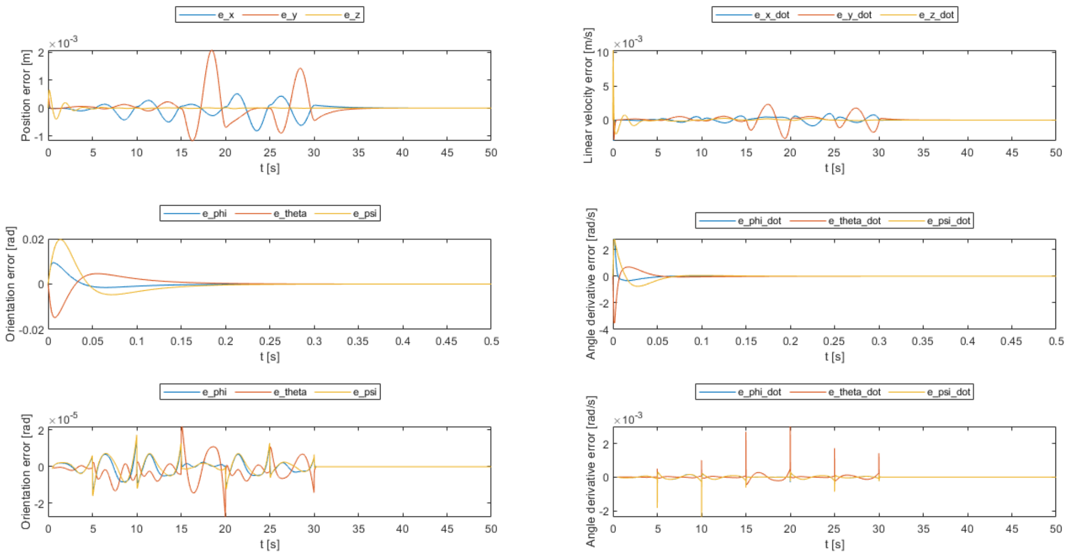

Figure 22: Control errors in the Passivity-based Control. The first plot shows the position error on the x, y and z axis; the second plot shows the velocity error on the x, y and z axis; the third plot shows the orientation error on ϕ, θ and ψ in the first $0.5\,s$; the fourth plot shows the angle derivative error on $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$ in the first $0.5\,s$; the fifth plot shows the continuation of the third one; the sixth plot shows the continuation of the fourth one.

Figure 22 shows that, as expected due to how we tuned the control parameters, the linear position and velocity errors are sufficiently low and have a similar evolution compared to those in the Hierarchical Control. The angular position error starts with a slightly higher peak, but then it soon reaches minuscule values in the order of 10^{-5}; the evolution of the error on the angle derivatives is also very similar to that of the Hierarchical Controller, and features the same little peaks every $5\,s$.

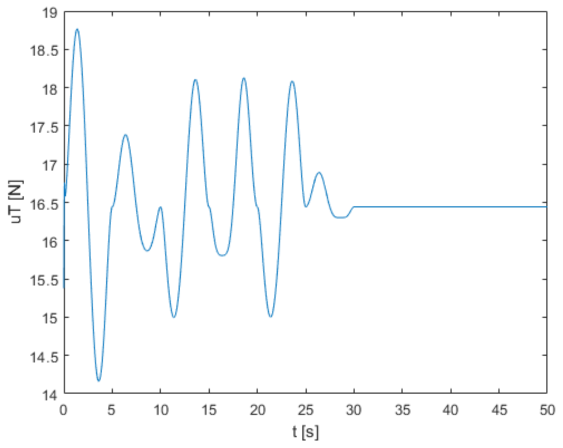

Figure 23: Control input u_T in the Passivity-based Control.

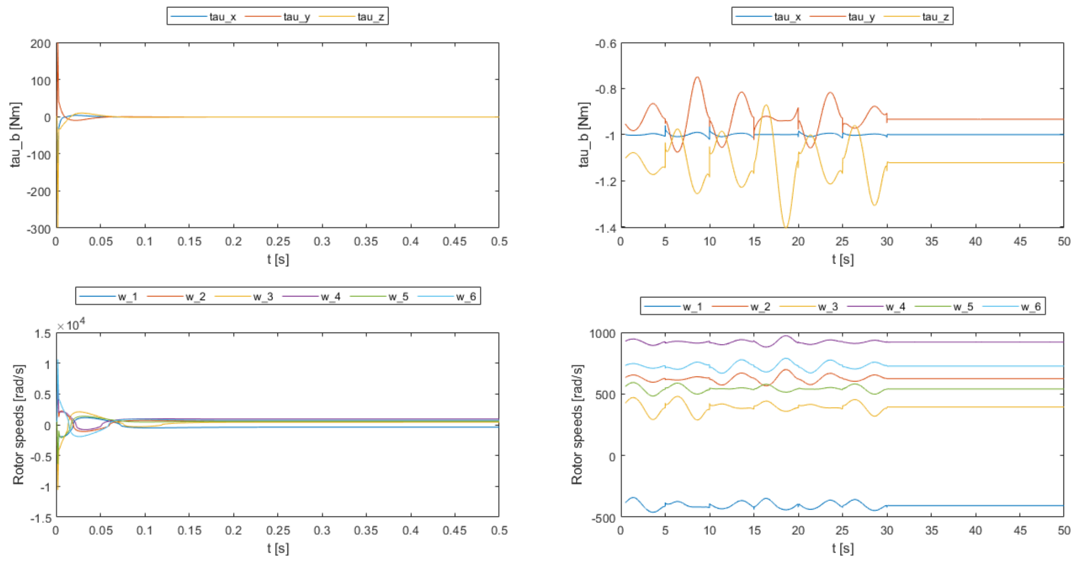

Figure 24: Other control inputs in the Passivity-based Control. The first plot shows the control torques τ^b in the first $0.5\,s$, while the second plot shows the continuation of that; the third plot shows the rotor speeds in the first $0.5\,s$, while the fourth plot shows the continuation of that.

It's pretty much impossible to spot any difference between u_T in the Passivity-based Control (Figure 23) and the one in the Hierarchical Control (Figure 15). The same comparison can be done about τ^b and the rotors' speeds in Figures 16 and 24: other than the initial peak, that would need to be saturated, the control inputs result pretty much identical.

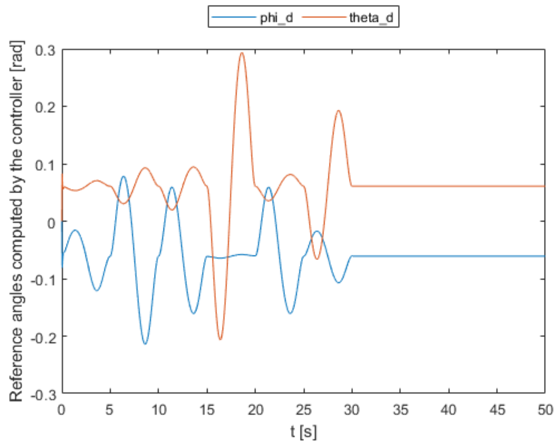

Figure 25: Reference angles generated in the Passivity-based Control.

Having seen the previous plots and remembering that the outer-loop, which computes the reference angles for the inner-loop, is the same in both the Hierarchical and the Passivity-based Controller it's no surprise that Figure 25 looks the same as Figure 17.

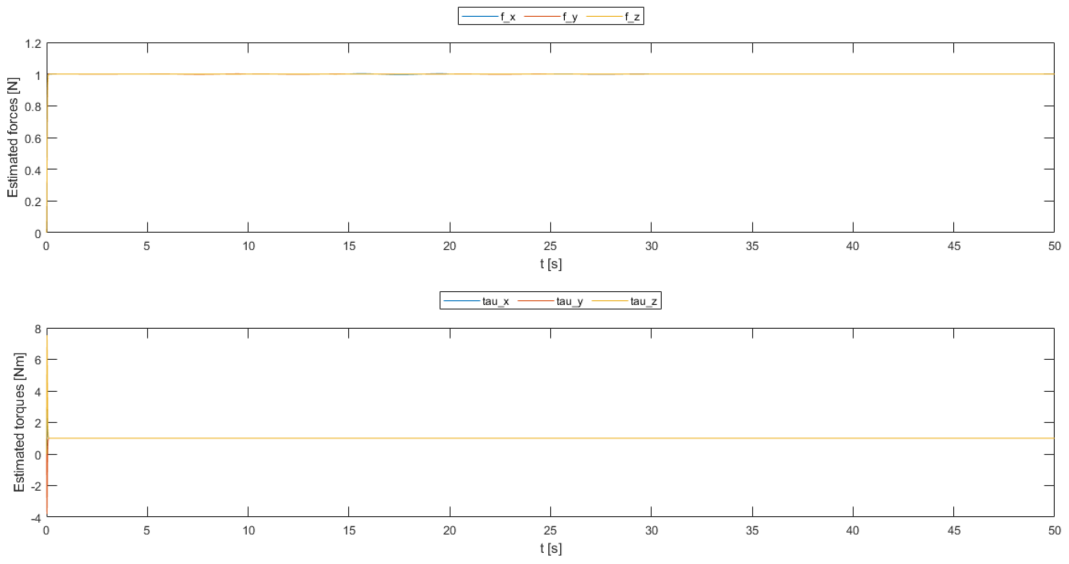

Figure 26: Estimate of the external wrench in the Passivity-based Control. Since the numerical values of the external disturbances are the same the plots overlap.

In Figure 26 we can once again see that the estimator is able to quickly find the value of the external forces and torques so that the control loop can cancel them. The initial peak is similar in amplitude to the one seen in the Hierarchical, and disappears just as fast.

4.4 Feedback Linearization Control for Full Actuation

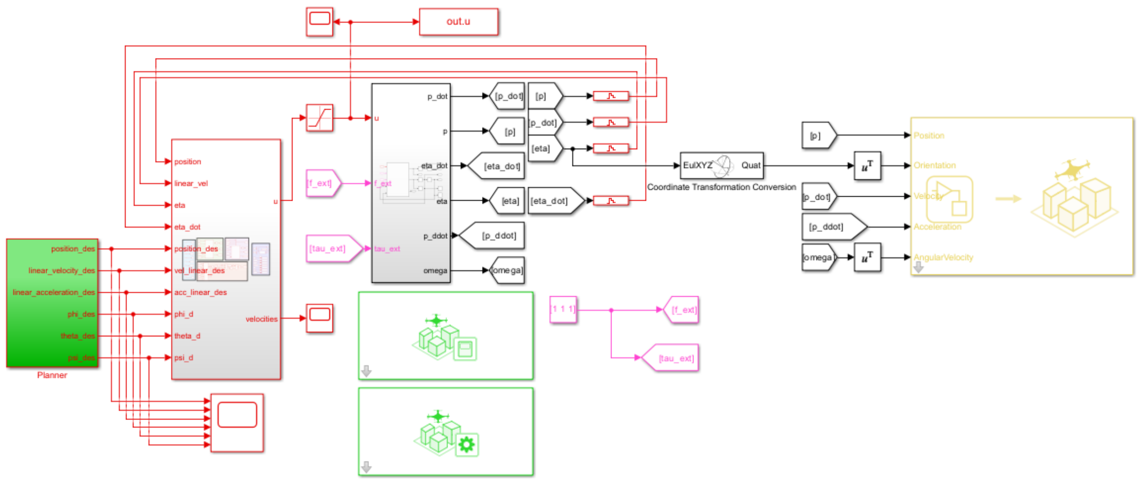

Figure 27: Overview of the control scheme used for the Feedback Linearization Control.

Figures 27, 28 and 29 show the *Simulink* model created to implement the Feedback Linearization Controller on the RPY UAV model. The last controller we will showcase will also be the only one to properly use the UAV's full actuation, so it will be the only one where we plan all the three angular coordinates and where the outer-loop gives us a force with components on the x, y and z axis rather than just u_T as we've done so far.

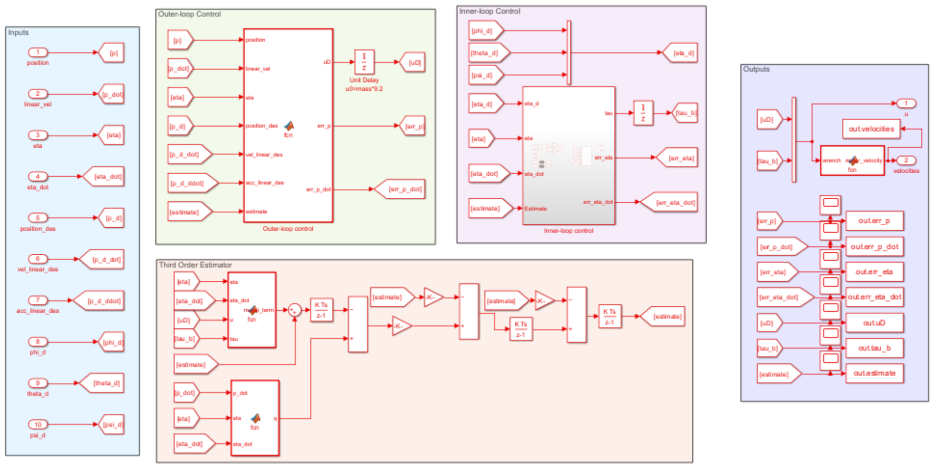
Figure 28: Control scheme used for the Feedback Linearization Control, divided in Outer-Loop for the linear part and Inner-Loop for the angular part.

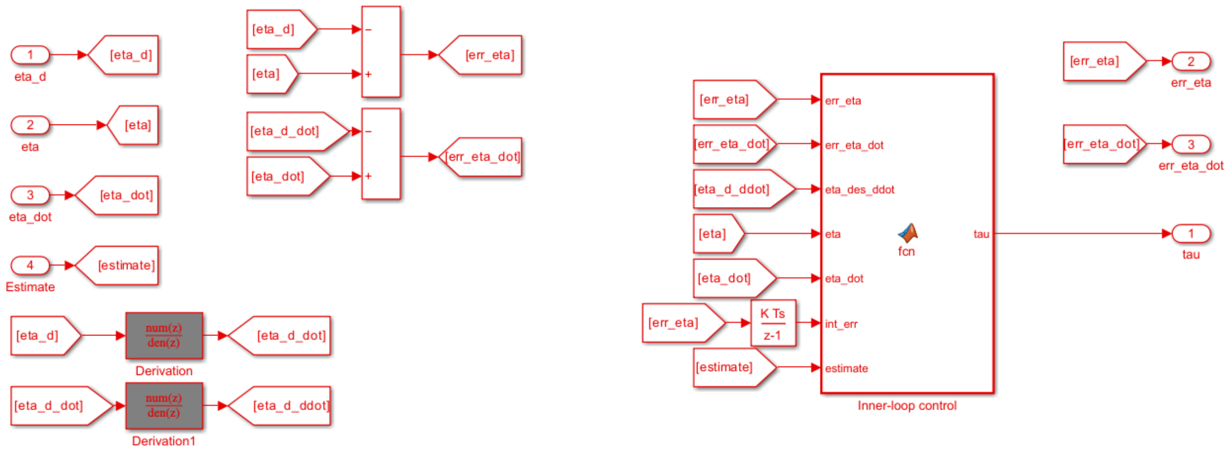

Figure 29: Block diagram of the Inner-Loop of the Feedback Linearization Control

Both the overall control scheme of Figure 28 and the inner-loop scheme of Figure 29 work in a similar fashion to what has already been explained so far. In fact the inner-loop for this controller is exactly the same as the one of the Hierarchical Control, since it already contained a feedback linearization on the angular part, with the only difference that the gain used for the integral action in the angular part is $K_{i_{ang}} = 10000 I_3$. Using a smaller gain could also be acceptable since the only change in performances is connected to how fast the orientation errors converge to zero at steady state, and they are in the order of 10^{-4} anyways.

The novelty of this controller lies in the outer-loop, where we cancel all the dynamics of the system using the following equation:

$$u_D = R_b^{-1} m \left(\begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \frac{f_{est}}{m} + K_p e_p + K_d \dot{e}_p - \ddot{p}_{b,d} \right) \tag{23}$$

In which R_b must be computed from the Euler angles, since we don't have a direct feedback of the rotation matrix, and where e_p and $\dot{e}_p$ follow the definitions mentioned in the previous controllers.

To achieve similar performances to the other controllers, K_p and K_e have been tuned in the same way used for the analogous gain matrices in the Hierarchical and Passivity-based Controllers.

Once again the physical parameters, the external disturbances and the estimator are the same as what has been presented in the Hierarchical and Passivity-based Controller. Since the Feedback Linearization Control exploits the robot's full actuation, however, we also plan and use the references for ϕ and θ rather than having to compute them online in the outer-loop.

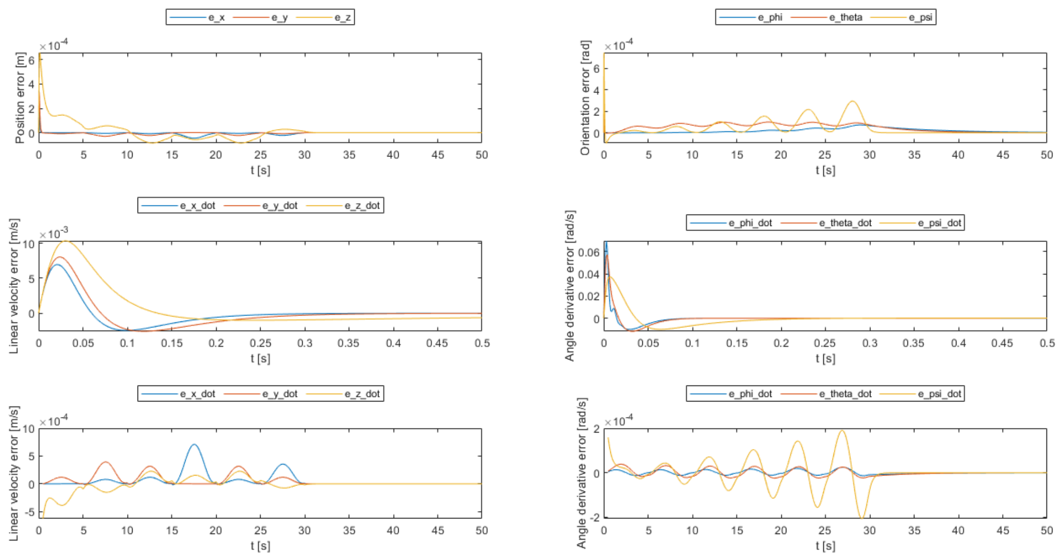

Figure 30: Control errors in the Feedback Linearization Control. The first plot shows the position error on the x, y and z axis; the second plot shows the orientation error on ϕ, θ and ψ; the third plot shows the linear velocity error on the x, y and z axis in the first $0.5\,s$; the fourth plot shows the angle derivative error on $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$ in the first $0.5\,s$; the fifth plot shows the continuation of the third one; the sixth plot shows the continuation of the fourth one.

Figure 30 shows the control errors on which this controller works. The position and orientation errors are always small enough to be more than acceptable, and the use of integral actions and the estimator makes all the errors go to zero at steady state. It is interesting to note that the linear position error goes to zero without the need of an added integral action, just with the combined effort of the Feedback Linearization and the estimator. The plots about linear velocity and angle derivative error have been split in two to highlight both the initial peak and then the low values which are reached right after that. The angle derivative error lacks the peaks that happen every $5\,s$ that we've seen in the two previous controllers, and even though there are some oscillations they're too small in amplitude to be a problem.

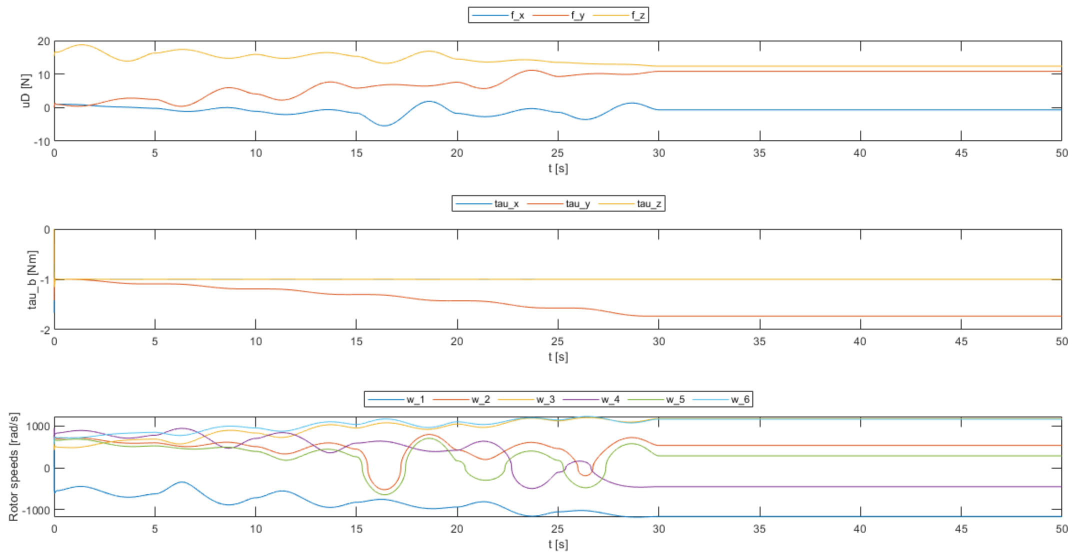

Figure 31: Control inputs in the Feedback Linearization Control. The first plot shows the control force u_D on the x, y and z axis; the second plot shows the control torque τ^b (the torque on the x and on the z axis overlap); the third plot shows the rotor speeds.

Thanks to the full actuation, in Figure 31 we have three control forces on the three axes rather than just u_T. We note that, as the drone's orientation changes in time, the distribution of the forces changes. A similar effect can be seen on the rotor speeds as well, as their values vary much more compared to the last two controllers, since the robot now has to use its propellers to generate a full wrench in 3D. From the same figure we can also see that the evolution of τ^b is much smoother compared to the ones seen so far, with two components being constant after the initial peak and one following the same shape of the angular references from Figure 3. It's possible to notice that the only torque that is not equal and opposite to the external disturbances is the one around the y axis: this peculiar effect is actually a coincidence dependent on the posture the drone achieves and the effect of the projection of the external torques in body frame, as we've varied the desired posture while keeping the same disturbances and the results varied, with differently distributed control torques.

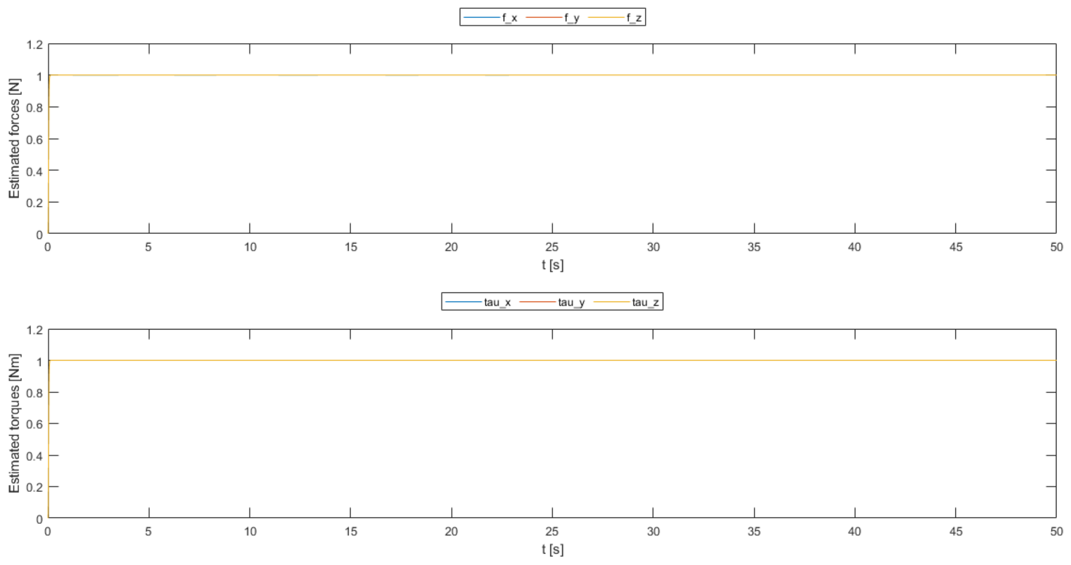

Figure 32: Estimate of the external wrench in the Feedback Linearization Control. Since the numerical values of the external disturbances are the same the plots overlap.

As mentioned before, the Feedback Linearization Control also uses the estimator to reject disturbances and uncertainties. Unlike what we've seen in the other RPY controllers, Figure 32 shows that the estimate doesn't have any initial peak and is able to quickly find the correct value of the external disturbances.

5 Conclusion

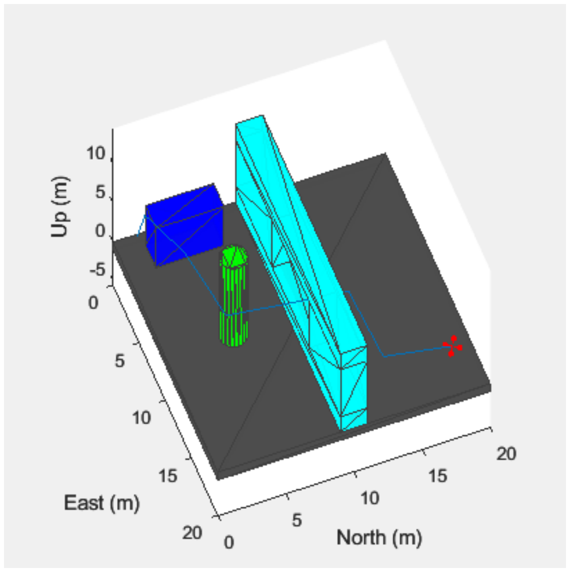

Figure 33: Plot of the UAV moving in the 3D environment, generated using the *Simulink* blocks in the *UAV Scenario* package. The plot shows the path that the drone follows using the Feedback Linearization Control; due to the small scale of the errors and the fact that the path remains the same it is hard to distinguish with the naked eye the simulations done with the different controllers. Nonetheless, videos of the different simulation results can be found at the following link: FSR_FinalProjectVideo_AliottaEsposito

Figure 33 and the videos linked in the caption show the (sped up to mimic real time) actual trajectory that the UAV follows in the execution of the simulations. It is hard to notice meaningful differences between the controllers from such a visualization, but the plots provided in Chapter 4 can be used to make a comparison and possibly answer the question of which controller is best suited for the control of the drone in the described scenario.

As we've proven multiple times, the Hierarchical and the Passivity-based Control are extremely similar in almost all regards. The former should be easier to tune, while the latter should avoid the problems of low robustness given by the Feedback Linearization in the inner-loop, but in our testing we haven't had too many issues finding parameters that could make the Passivity-based controller work, and we already have a good way of dealing with uncertainties and disturbances thanks to the estimator and the integral actions. Between these two controllers, the best option would be the Passivity-based one, because it's able to achieve a much smaller angular error with a similar control effort, and without the need of any modification on its inner-loop.

The Geometric Controller's main advantage is the lack of representation singularities, but in the case analyzed we don't need acrobatic movement, and the high sensitivity that we've found to the variations of the control gains, of the external disturbances and of the physical parameters make this choice less than ideal.

The Feedback Linearization controller, instead, seems to be the perfect fit for the tilted hexarotor: the control errors are perfectly acceptable, the control inputs are smooth and don't require any initial saturation, and finally the use of the full actuation lets us move the robot in a much more dexterous way and lets us plan the whole orientation rather than just the yaw angle. The addition of an integral action and the estimator solve the robustness problem inherent with this kind of controller, meaning that we can enjoy the simplicity and efficiency of this technique without worrying about its cons.

With the rotor velocities that the controllers already provide thanks to the allocation matrix, it should be possible to control either an actual drone or a simulated one in the *ROS/Gazebo* environment, for example by using the *RotorS* package and by converting the *Simulink* models in *C++* code.

6 References

[1] R. Voyles and G. Jiang, "Hexrotor UAV platform enabling dextrous interaction with structures—Preliminary work," in Proc. IEEE Int. Symp. Saf., Secur., Rescue Robot. (SSRR), Nov. 2012, pp. 1–7.

[2] R. Rashad, J. Goerres, R. Aarts, J. B. Engelen, and S. Stramigioli, "Fully actuated multirotor uavs: A literature review," IEEE Robotics & Automation Magazine, vol. 27, no. 3, pp. 97–107, 2020.