Contest description

Context and objective

This contest is a part of the project "Co4AIR - Computers, Cognition and Communication in Control: A strategIc paRtnership", funded by the European Commission through the Erasmus+ Key Action 203 Strategic Partnerships for Higher Education; see http://co4air.eu/. The contest is the result of a key project objective: to design and implement a challenge called "Co4-marathon", which will require student teams to design, build, and test efficient solutions for a current and practical control problem. This will challenge the brightest students from partners institutions and allow to compare the various teaching systems through the contest result.

Task

The objective of each team is to **race along a sequence of visual markers with a Parrot Mambo drone**, in a simulated environment. The markers are square, placed on the floor, and have colors that stand out. To prevent ambiguity when multiple markers are present in the camera view, they must be followed in order of their color: red, green, blue, red, green, blue, etc. The environment provides the feed of the down-facing camera of the drone, as well as accurate position and attitude signals for the drone; and it allows setting the motor commands. The drone parameters will be fixed, but the sequence of marker positions will be unknown to the teams until the day of the contest. The teams will have to design (a) a **computer vision algorithm** to detect the markers and produce waypoints for the controller; or (b) a **control algorithm** that ensures the drone reaches each waypoint in the sequence to within a prespecified tolerance; or **both** (a) and (b). Teams may choose to do only (a) or only (b), in which case a default implementation of the missing component is supplied by the organizers. The programming environment is Matlab/Simulink.

Optionally, you may also develop and demonstrate a **real-life solution** for the marker following task, with the true Parrot Mambo.

Logistics

When and where? The contest will take place on the 1st of July 2021, at 1PM CET, online via Microsoft Teams. Details will be announced later. Please book 4 hours, until 5PM CET.

Who? Teams of 2 or 3 students (PhD, MSc, or BSc) are accepted. All students in a team should preferably come from the same institution. Due to the nature of the task, which mixes control and computer vision, we welcome teams with expertise in either systems and control, computer science, or a mix of both.

How to register? Send an email to lucian.busoniu@aut.utcluj.ro, the competition lead organizer, at the latest **on the 20th of May**. Include the composition of your team, your institution, a team name (pick a good name that will make the audience believe in you!), and (preliminarily, you may revisit this later) whether you plan to solve task (a), (b), or both.

How much? There are no fees for participation. If in the end the contest takes place on-site, and you are a student of a Co4AIR partner institution, then you may be eligible for travel and accommodation refunds up to a limit specified by Erasmus rules; please contact the project leader at your institution to confirm.

Evaluation

Your solution will consist of certain Simulink blocks and Matlab code, as described under Solution Development below. Each team must submit by email a **draft solution two weeks** before the competition, to be checked for incompatibilities with the framework. Then, **the final solution is due one week** in advance.

On the day of the contest, you will be evaluated by a jury composed of one member from each Co4AIR partner institution. You will have to **present** your technical approach (up to 10 minutes per team, including questions from the jury), and then **demonstrate** your solution. Teams will be sorted in descending order of their **score,** computed with the following formula:

$$P + \left(1 - \frac{T}{T_{max}}\right) + \left(1 - \frac{M}{M_{tot}}\right) + B + R$$

where:

- P is your presentation score assigned by the jury

- *T* is the time to complete the task (via the safe landing of the drone), and T_{max} is a maximal time, which you will also get if you crash the drone or timeout
- *M* is the number of markers missed during your run (position not reached within the required tolerance), and M_{tot} is the total number of markers.
- *B* is a bonus which you receive if you performed both tasks (a) and (b).
- *R* is a bonus score that evaluates your real-life solution with the actual drone, if you choose to develop one.

The specific values for the range of P and R and the value of B will be published closer to the competition; we do not want students to overfit to the specific evaluation function used, your solution should work well in general! Also, keep in mind that the track (marker sequence) will be different from the one in the template.

Environment setup guide

Following this guide should result in a ready-to-use environment for Parrot minidrone control design, code generation and simulation.

MATLAB

MATLAB version 2019b must be used to guarantee compatibility with provided blocks/code. A fresh install is recommended. The following add-ons/toolboxes are needed:

- Computer Vision System Toolbox
- Aerospace Toolbox
- Aerospace Blockset
- Simulink Coder
- Simulink Control Design
- Simulink 3D Animation
- Robotics System Toolbox
- Robotics Systems Toolbox UAV Library

plus the:

Support package for Parrot minidrones

Install the Simulink® Support Package for Parrot® Minidrones, found here: https://www.mathworks.com/hardware-support/parrot-minidrones.html (alternatively, the package can be found by searching for it in the Add-on Explorer in MATLAB). Additional tutorials and examples can be found on this page. You can also use the documentation for the support package in MATLAB

After the package is installed, you will be prompted to run the setup for the package. This step is necessary (and can be completed) only if you have a Parrot minidrone available and you wish to generate code for the drone and perform real-world flights. The Simulations will work regardless of the completion of the Set-up for the package. If you wish to perform this action later, you can do so from the Add-on Manager in MATLAB.

Simulation template

Provided in the archive is a Simulink project template. Create a new project from the provided template and perform a test run, to verify all is in order. Simulations can be run from the main model only: asbQuadcopter.

Solution development

After you have created a new project, you can begin the design and implementation of your solution as follows.

Depending on your preference you can choose to implement your own algorithms for the machine vision system, the control system, or both. These are the models you will be working in:

Opening the project will automatically open the main model where you will find the flightControlSystem model referenced(FCS), which references the flightController model.

Regardless of your choice in solution development, the inputs and outputs of the blocks that you will modify should not be changed. Moreover, any constants you need for your algorithms should be initialized in the initConst.m file in the root folder of the project.

Solving task (a), Computer Vision

Your solution will consist of two blocks:

- The first block will replace the contents of the "flightControlSystem/Image Processing System" block. This block will receive the image data from the downward facing camera on the drone. The output of this block is not constrained as it will be processed by the second block you will design. However, you should be careful with the dimensions of the data being transferred between the two blocks as large amounts of data will slow the control system down.
- The second block will replace the contents of the "flightControlSystem/Flight Control System/Path Planning/Marker Follower" block. This block will receive the data from your first block alongside state data including (not limited to) drone position and orientation and will produce two outputs. The first output is the coordinates vector of the target point for the drone, and the second one is a completion flag. When the completion flag is set, the drone will begin its landing sequence.

Solving task (b), Control Design

Your solution will consist of one block which will replace the contents of the "flightController/Flight Controller" block. This block will receive the position waypoint from the machine vision system and state information and will output the motor commands.

In the appendix, we provide a quick guide to modeling and control design for the drone. Moreover, in order to help you test your controllers more easily, a simplified simulated control loop is provided in the Simulink model controllerDesign/Nonlinear_model.slx. This eliminates all higher-level components including computer vision, path planning etc. and just focuses on the control of the drone, which e.g. reduces compilation time.

Complete solution

If you choose to develop a complete solution, you will have to provide the blocks for both above-mentioned cases.

Submission of the simulation components for evaluation

For your solution to be evaluated you will have to provide the blocks you have created and a script file for any constants. These blocks will then be inserted into their respective places in a clean environment where the simulation will be run. Therefore, changes to the expected inputs and outputs of the blocks or other modifications made to the rest of the project will not be carried over, and if you make any such changes your solution might not work.

Real-life demonstration

Optionally, you may choose to also demonstrate a real-life solution of the marker following task, with the actual Parrot Mambo drone. To this end, you should provide a video recording of your drone running the track, as well as any explanations required (max. 4 minutes per video). Try to replicate the simulation task as closely as possible, but otherwise, there are no constraints on this solution. The jury will evaluate the overall quality of your demo and assign the value of the real-time bonus accordingly.

A short guide to modeling and control design for the Parrot Mambo mini-drone

November 11, 2020

1 Dynamical Modeling

The dynamical model for the Parrot Mambo presented in the sequel was obtained based on the Newton-Euler formalism and by considering a inertial reference frame in **N**orth-**E**ast-**D**own (NED) and a body-fixed coordinates initially aligned to this origin (**O**). The position vector (**B**) of the center of mass of the rotorcraft is denoted by $\xi = (x_B, y_B, z_B)^T$, it represents the position coordinates of the drone with respect to the NED inertial frame. The orientation vector of the aircraft with respect to the inertial frame is expressed by $\eta = (\psi, \theta, \phi)^T$ where ψ, θ and ϕ are respectively the yaw, pitch and roll Euler angles. For this development, we made the same assumptions as those done for the Simulink® Quadcopter simulation.

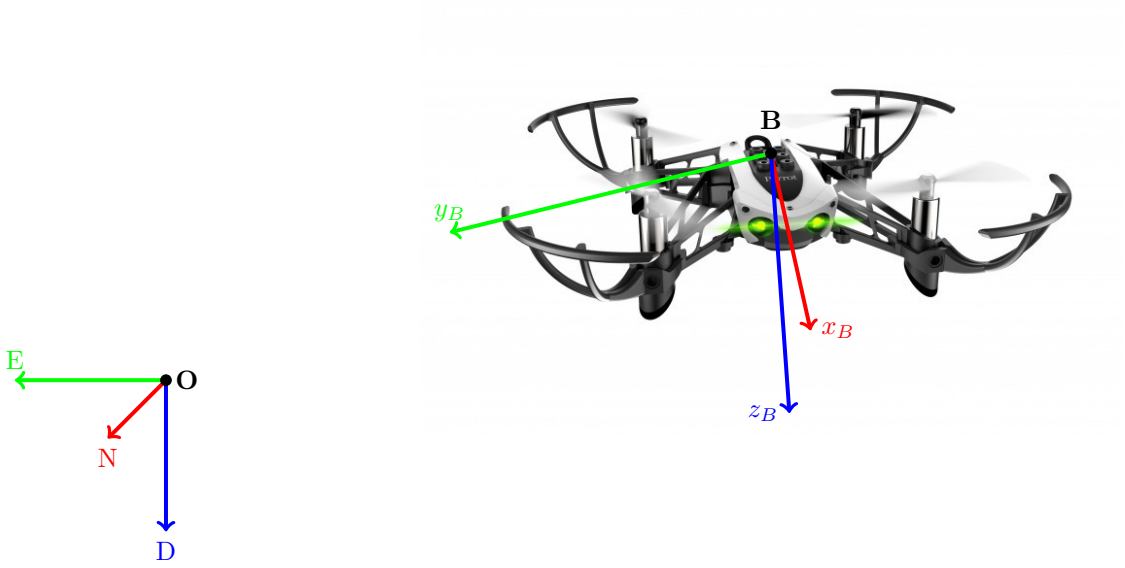

Figure 1: Coordinate Frames

1.1 Nonlinear Model

A simplified model to describe the quadrotor dynamics in its 6 Degrees of Freedom (DOF) configuration can be obtained from a standard Newtonian approach. Considering a total thrust force F acting on the center of mass and a body-frame torque τ as the inputs of the system, the drone's dynamics with neglected gyroscopic effects (e.g. see [1, 2]) can be written as:

$$m\ddot{\xi} = -mg\mathbf{D} + \mathbf{R}(\eta)F \tag{1}$$

$$I\dot{\Omega} = -\Omega \times I\Omega + \tau \tag{2}$$

where $R(\eta) \in SO(3)$ is a Z-Y-X Euler angles rotation matrix that associates body-fixed frame with the inertial frame, m represent the total mass, g denote the gravitational constant, $\Omega = (p, q, r)^T$ is the angular velocity of the vehicle expressed in body-fixed frame, $I = diag(I_{xx}, I_{yy}, I_{zz})$ is the inertia

matrix, and $\tau = \begin{bmatrix}\tau_\phi & \tau_\theta & \tau_\psi\end{bmatrix}^T$ is the total torque. Further, let τ_ψ be the total torque about the (body-frame) z_B-axis resulting from propeller drag, τ_ϕ and τ_θ be the resulting torque about y_B-axis and x_B-axis, respectively. Furthermore, let's define a W^{-1} matrix which transforms the body-angular rates about local x-y-z axes to Euler-rates to obtain the following relationship: $\dot{\eta} = W^{-1}\Omega$.

$$W^{-1} = \begin{bmatrix} 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \\ 0 & \cos\phi & -\sin\phi \\ 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \end{bmatrix} \tag{3}$$

Now, we can express the linear and angular momentum equations (4-5) and the quadrotor's angular rates (6):

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -\frac{F}{m}\begin{bmatrix} \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ \cos\theta\cos\phi \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \\ 0 & \cos\phi & -\sin\phi \\ 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \end{bmatrix}\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} q\frac{\sin\phi}{\cos\theta} + r\frac{\cos\phi}{\cos\theta} \\ q\cos\phi - r\sin\phi \\ p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} rq(I_{yy} - I_{zz}) \\ rp(I_{zz} - I_{xx}) \\ pq(I_{xx} - I_{yy}) \end{bmatrix} + \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{6}$$

These Equations of Motion (EOM) describe the rigid body dynamics on its 6 DOF and enable us to proceed with a model based controller design.

1.2 Linearized Model

In order to design a linear controller such as a Proportional-Derivative (PD) law, we must obtain a linearized representation of the quadrotor system. So, first, we will consider the system on its Equilibrium Hover Configuration by defining:

$$\xi = \xi_0,\ \theta_0 = \phi_0 = 0,\ \psi = \psi_0,\ \dot{\xi} = 0,\ \dot{\eta} = 0 \text{ and } F_0 = mg \tag{7}$$

Then, let us perform a simplification with the trigonometric functions $\cos(\cdot)$ and $\sin(\cdot)$ by assuming that, for a small variation of δ around 0, $\cos\delta$ is 1 and $\sin\delta$ is equal to δ. In the sequel, we can rewrite the linear momentum equation (4) as:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -\frac{F_0}{m}\begin{bmatrix} \theta\cos\psi + \phi\sin\psi \\ \theta\sin\psi - \phi\cos\psi \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{8}$$

Then, applying the above-mentioned procedure to linearize the trigonometric functions and considering the small angle approximation theory (i.e. the product of two terms that are approximately 0 results in 0), we can obtain the linear equation for the angular rates (5) near the hover position:

$$\begin{bmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} q\phi + r \\ q - r\phi \\ p + q\phi\theta + r\theta \end{bmatrix} = \begin{bmatrix} r \\ q \\ p \end{bmatrix} \tag{9}$$

Finally, applying the same assumption about the high order terms for the angular momentum equation (6) gives:

$$\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{10}$$

2 Cascaded Control

In this part of our tutorial, we are concern about how to control the motors in order to drive the vehicle along a desired trajectory.

We will adopt an hierarchical control approach. In this structure, we have three levels, the lowest one is the control of the rotor rotation speed, which will be assumed ideal and so unconsidered in this analysis. The next level (called inner loop from now on) is the control of vehicle attitude, and the top level (outer loop) is the position control along a trajectory. These levels form our nested feedback loops [3].

The inner control loops are depicted in Figure 2. Note that the outer, position control loop relies on the inner attitude control loop, so, in order to follow a desired trajectory we must guarantee a faster convergence of this inner controller.

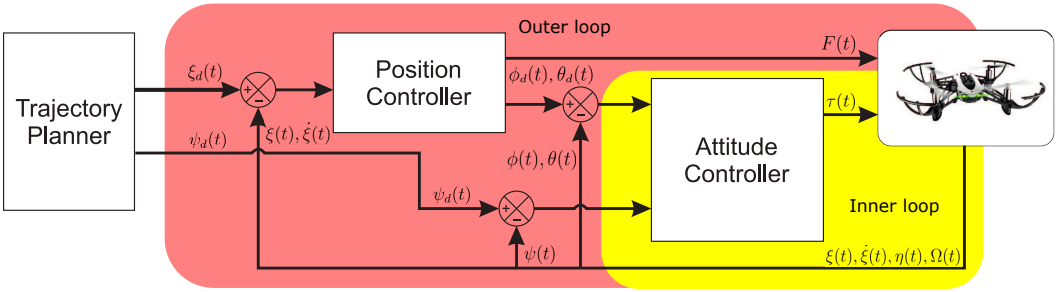

Figure 2: Cascade Control Block Diagram

2.1 Inner Loop (Attitude Controller)

The design of the nested cascade system start with the design of the controller for the inner loop, which must have a faster dynamics. To do so, we can summarize equations (9) and (10) to obtain the attitude dynamics in the state-space framework:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \theta \\ \psi \\ p \\ q \\ r \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{11}$$

Moreover, recalling the fact that previously we performed a linearization around a specific operating point (7), now we must consider the stabilization problem in term of the error between the actual states and the initial conditions. Therefore, let's define $e_\gamma = \gamma_d - \gamma$, $\gamma = \{\dot{\eta}, \eta, \dot{\Omega}, \Omega\}$. So, we can rewrite the error dynamics as:

$$\begin{bmatrix} e_{\dot{\phi}} \\ e_{\dot{\theta}} \\ e_{\dot{\psi}} \\ e_{\dot{p}} \\ e_{\dot{q}} \\ e_{\dot{r}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_\phi \\ e_\theta \\ e_\psi \\ e_p \\ e_q \\ e_r \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{12}$$

Now, to perform the design procedure we can apply any method adapted for the state space framework (i.e. Pole Placement, Linear Quadratic Regulator (LQR), Lyapunov 2$^{\text{nd}}$ method). So e.g., considering a LQR, for a continuous time system, the inner-loop state-feedback control law $\tau = -Ke$, with $\tau = \begin{bmatrix} \tau_\phi & \tau_\theta & \tau_\psi \end{bmatrix}^T$ and $e = \begin{bmatrix} e_\phi & e_\theta & e_\psi & e_p & e_q & e_r \end{bmatrix}^T$, and where K is the gain matrix to be tuned such that the following quadratic cost function is minimized:

$$J(\tau) = \int_0^\infty (e^T Q e + \tau^T R \tau) \tag{13}$$

During the design of such a controller, we must pay attention in the fact that we are dealing with an approximation of the real system which is valid only near to the operating point. Recognizing this aspect, a good strategy can be to penalize the changes of η. Moreover, we need also to deal with the fact that the actuators have limitations, so it should be interesting to penalize the input signal. This can be implemented by tuning the parameters Q and R respectively.

2.2 Outer Loop (Position Controller)

The outer loop is associated to the position control. Here is important to make some remarks. In this system all the actuators apply a force along the z_B axis, thus to have a displacement toward x_B and y_B axis, a force unbalance is required to provoke a roll or a pitch rotation and consequently to have a movement in these directions.

Moreover, in order to obtain the required rotations, let's consider the position controller. From (8) we can obtain the relationship between the second derivative of the position variables and the Euler angles θ and ϕ.

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = -g \underbrace{\begin{bmatrix} \cos\psi & \sin\psi \\ \sin\psi & -\cos\psi \end{bmatrix}}_{R_{xy}} \begin{bmatrix} \theta \\ \phi \end{bmatrix} \tag{14}$$

Pre-multiplying both sides by the inverse of the Rotation Matrix R_{xy} followed by some mathematical arrangements gives us the following equation which provides the required values θ_d and ϕ_d to reach a specific position on the plane $X - Y$:

$$\begin{bmatrix} \theta_d \\ \phi_d \end{bmatrix} = -\frac{1}{g} \begin{bmatrix} \cos\psi & \sin\psi \\ \sin\psi & -\cos\psi \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} \tag{15}$$

Note that the inverse of R_{xy} is itself.

Recall that $\xi = \begin{bmatrix} x & y & z \end{bmatrix}^\top$, i.e. the vector of positions. To exponentially drive all the components of error, we want to command the acceleration vector to satisfy:

$$\ddot{x} = \ddot{x}_d + K_{Dx}(\dot{x}_d - \dot{x}) + K_{Px}(x_d - x) \tag{16}$$

$$\ddot{y} = \ddot{y}_d + K_{Dy}(\dot{y}_d - \dot{y}) + K_{Py}(y_d - y) \tag{17}$$

Also, from (8) for the z component, we can write:

$$F = m(g + \ddot{z}_d + K_{Dz}(\dot{z}_d - \dot{z}) + K_{Pz}(z_d - z)) \tag{18}$$

Now, since the tracking planner provides piecewise constant set points (x_d, y_d and z_d) in the context of the current contest, from (16) and (16), the equation (19) turns to:

$$\begin{bmatrix} \theta_d \\ \phi_d \end{bmatrix} = -\frac{1}{g} \begin{bmatrix} \cos\psi & \sin\psi \\ \sin\psi & -\cos\psi \end{bmatrix} \begin{bmatrix} K_{Px}(x_d - x) - K_{Dx}\dot{x} \\ K_{Py}(y_d - y) - K_{Dy}\dot{x} \end{bmatrix} \tag{19}$$

as well as (18) to:

$$F = m(g + K_{Pz}(z_d - z) - K_{Dz}\dot{z}) \tag{20}$$

Finally, the tuning procedure of the proportional and derivative gains (K_Px, K_Py, K_Pz, K_Dx, K_Dz, K_Dz) of (19-20) can be done in MATLAB® by implementing the nonlinear model and linear controller given above.

Note that by following the procedure presented above, the system inputs are the thrust and torques, and so, any conversion is required regarding to the use of these controllers on Simulink® Quadcopter simulation, we just need to use the "ControlMixer" and the "Thrust to Motor Commands" blocks to derive motor commands.

The proposed strategy is generic and so can be used for other quadrotors. In order to obtain the results for the Parrot the following parameters are used:

- initial location of the body in the flat Earth reference frame:

$$\xi_0 = \begin{bmatrix} 57.0000 & 95.0000 & -0.0460 \end{bmatrix} \tag{21}$$

- mass and inertia of the rigid body:

$$m = 0.0630kg, \qquad I = \begin{bmatrix} 0.5829 & 0 & 0 \\ 0 & 0.7169 & 0 \\ 0 & 0 & 1.0000 \end{bmatrix} \times 10^{-4} \tag{22}$$

References

[1] S. Bouabdallah and R. Siegwart. Backstepping and sliding-mode techniques applied to an indoor micro quadrotor. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2247–2252, 2005.

[2] Luis Rodolfo García Carrillo, Rogelio Lozano, Alejandro Enrique Dzul López, and Claude Pégard. *Quad Rotorcraft Control*. Springer-Verlag, London, 2013.

[3] R. Mahony, V. Kumar, and P. Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics Automation Magazine*, 19(3):20–32, 2012.