

POLITECNICO DI TORINO

Master's Degree
in Mechatronic Engineering

Modelling and trajectory tracking of a quadrotor using MATLAB Simulink, Simscape and ROS-Gazebo.



Supervisors

Prof. Alessandro Rizzo
Prof. Umberto Montanaro
Prof. Aldo Sorniotti
Simone Martini

Candidate

Mattia Miscali

a.a. 2021-2022

Abstract

The goal of this Master Thesis consists on defining and building a multi platform simulation environment for quadrotors, this environment can be used to collect data from different platforms and compare the results of position and attitude controllers in different simulators.

The main focus of this work regards the analysis and replication of different models in several simulation platforms. In particular, two mathematical models will be analysed and simulated in MATLAB Simulink, a Simscape model will be simulated in the Simscape Multibody environment and finally the quadrotor is replicated and simulated in Gazebo 7 with ROS Kinetic.

Several controllers are designed in Simulink and deployed on ROS Gazebo. This allows to simulate the deployment of such controllers in real hardware and to evaluate their performances thanks to the data collected in ROS. The resulting data is then analysed thanks to some Key Performance Indicator. The testing experiment consists on the tracking of a reference trajectory starting from the resting position on the ground. The controller with the best performance in this experiment is the smooth Sliding Mode Controller.

Contents

Abstract	i
List of Figures	IV
List of Tables	VI
1 Introduction	1
1.1 Drones and Quadrotors	1
1.2 Project's Objective and Thesis Structure	1
1.3 Environment	2
1.3.1 ROS and Gazebo	2
1.3.2 MATLAB Simulink and Simscape	3
2 Modelling	4
2.1 Mathematical Model	4
2.1.1 Reference Frames and Generalized Coordinates	4
2.1.2 Dynamic Constants and Generalized Forces	6
2.1.3 Newton-Euler Derivation	9
2.1.4 Euler-Lagrangian Derivation	10
2.1.5 Simulink File Of The Mathematical Model	12
2.2 ROS Model	15
2.2.1 Custom Simulator	15
2.2.2 Simulator workspace	15
2.2.3 Custom package	16
2.2.4 Key files for ROS simulations	18
2.3 Simscape Model	21
3 Model Validation	24
3.1 Actuation and Acquisition Node	24
3.1.1 ROS Input	25
3.1.2 ROS Output	26
3.2 Model Validation Experiment	28
3.3 Model Validation Results	30

4	Control	34
4.1	Under Actuated Systems and Control Architectures	35
4.1.1	Outer Loop	36
4.1.2	Conversion Block	36
4.1.3	Feedback Linearization	37
4.2	Inner Loop	38
4.2.1	PD Controller	38
4.2.2	Sliding Mode Control	38
4.2.3	MRAC	39
4.3	Control Implementation and Results	40
4.3.1	Reference	40
4.3.2	Tuning	41
4.3.3	Results	43
5	Conclusion	47
	Bibliography	49

List of Figures

1.1	ROS.	3
1.2	MATLAB.	3
2.1	Reference frames.	5
2.2	Forces and torques for a clockwise (left) and counter clockwise (right) rotor.	7
2.3	Real lift (\mathbf{L}) and equivalent rolling moment (\mathbf{M}_R) for a clockwise (left) and counter clockwise (right) rotor.	8
2.4	Simulink file of the Newton-Euler differential equations	13
2.5	Simulink file of the Euler-Lagrange differential equations	14
2.6	ROS input section from Simulink code	16
2.7	Communication_node python script.	17
2.8	ROS nodes and topics. Nodes are contained into circles and connected by topics.	17
2.9	Representation of the launch file composition. Circles represent launch files, rhombuses represent final nodes.	18
2.10	Representation of the urdf file composition. Rectangles represent xacro files, hexagons represent final components of the drone.	19
2.11	Simscape Model.	21
2.12	Simscape Model Base.	22
2.13	Simscape Model Rotor.	23
2.14	Simscape Model Sensor.	23
3.1	Actuation and Acquisition node.	25
3.2	MATLAB Assign Function.	26
3.3	BUS correction.	27
3.4	Input signal.	28
3.5	Input signal for Gazebo.	29
3.6	Model validation experiment.	29
3.7	Model validation position.	30
3.8	Model validation orientation.	31
3.9	Model validation linear velocity.	31

3.10	Model validation Euler derivatives.	32
4.1	Controller structure.	35
4.2	Controller plus feedback linearization structure.	37
4.3	Simulink set up.	40
4.4	Simulink set up for ROS deployment.	41
4.5	Trajectories.	43
5.1	Gazebo Map.	48

List of Tables

1	Symbols.	vii
3.1	Position error.	32
3.2	Orientation error.	32
3.3	Linear velocity error.	33
3.4	Euler angles derivatives error.	33
4.1	Outer loop parameters.	41
4.2	PD parameters.	42
4.3	PD with Feedback Linearization parameters.	42
4.4	SMC parameters.	42
4.5	MRAC parameters.	42
4.6	PD RMSE results.	44
4.7	PD with Feedback Linearization RMSE results	44
4.8	SMC RMSE results.	45
4.9	MRAC with Feedback Linearization RMSE results.	45
4.10	PD ME results.	45
4.11	PD with Feedback Linearization ME results.	46
4.12	SMC ME results.	46
4.13	MRAC with Feedback Linearization ME results.	46

Parameter	Description	Unit of Measurement	Value
$x - y - z$	Inertial (world) reference frame		
$x_b - y_b - z_b$	Body reference frame		
ξ	Position of the quadrotor in the inertial RF	m	<i>variable</i>
η	Rotation of the quadrotor with respect to the inertial RF	rad	<i>variable</i>
ω_i	rotor angular velocity	rad/s	<i>control variable</i>
ω_Γ	sum with sign of the rotor angular velocities	m	<i>variable</i>
m_UAV	Body Mass	kg	0.69502
m_p	Rotor Mass	kg	0.009
m	Total mass of the drone	kg	0.73102
L_b	Body Length	m	0.34
L	Arm Length	m	0.17
h	Rotor z-offset from CoM	m	0.00951
I_x	Total Moment of inertia along x_b	kg m ²	0.0076974
I_y	Total Moment of inertia along y_b	kg m ²	0.0076974
I_z	Total Moment of inertia along z_b	kg m ²	0.0133784
$I_{x,p}$	Rotor Moment of inertia along x_{rotor}	kg m ²	5.63175×10^{-5}
$I_{y,p}$	Rotor Moment of inertia along y_{rotor}	kg m ²	5.63175×10^{-5}
$I_{z,p}$	Rotor Moment of inertia along z_{rotor}	kg m ²	1.125×10^{-4}
C_T	Thrust coefficient	kg m/rad ²	8.54858×10^{-6}
C_M	Drag moment coefficient	m	0.016
C_R	Rolling moment coefficient	kg m/rad ²	0.1×10^{-5}
C_D	Drag force coefficient	kg/rad	8.06428×10^{-4}

Table 1. Symbols.

Chapter 1

Introduction

1.1 Drones and Quadrotors

Drones are becoming increasingly popular in the modern society shifting from an initial recreational use to a more vast usage in fields like infrastructures inspection, delivery, reconnaissance and more. Thanks to the reduction of costs of electronics and at the same time an always higher computational power of micro controllers, unmanned aerial vehicles such as quadrotors are emerging and will become even more present in our day to day experience in the next decades [1]. In particular, quadrotors with fixed arms are one of the most widely used multi rotors drones, their light weight, high reliability, robustness and ease of design have made them the preferred platform for present and future research. This research field is therefore focused on autonomous flight and on improving the mathematical model of the drone itself and on the complex dynamics that an aerial vehicle is subject to ([2], [3], [4]).

1.2 Project's Objective and Thesis Structure

In this project the main goal is to design a multi platform simulation environment to test autonomous controllers for trajectory tracking problems. The main programs to be used in the analysis will be MATLAB, Simulink, Simscape, ROS and Gazebo. This allows to have several models and different simulators where the controllers can be independently tested.

The second chapter of the thesis is focused on the quadrotors models. For instance, the mathematical models will be introduced. Two mathematical models will be derived with two different methods. Then, the ROS workspace developed by [5] will be deeply studied with particular attention to the quadrotor *'humming-bird'* which resembles the most the study subject. A particular ROS node will be developed in Python in order to allow the communication between the controllers

designed in MATLAB Simulink and the pre existent ROS workspace. In conclusion of this chapter, the Simscape model developed to mirror the ROS model will be analysed. This model is designed to act exactly like the ROS quadrotor in order to work strictly on MATLAB and to obtain similar results with respect to ROS.

The third chapter regards the model validation test. Here, the different models are compared to each other thanks to an open loop signal. The same input signal, the angular velocities of the four rotors, is sent to the different models and the resulting evolution of the state variables is inspected.

The fourth chapter is focused on the control of the drone. Here the control algorithms that have been tested are explained and their application both in Simulink and in ROS is described. Then, the results from the trajectory tracking experiment are presented with some considerations about the different controllers performances.

In the last chapter the conclusions of this work are presented. Some suggestion about future work and possible application of this environment are given.

1.3 Environment

Given the thesis's objective and in order to show the real time feasibility of the several controllers, the implementation of the controllers has to be performed in an advanced robotic simulation environment such as ROS and Gazebo. The other simulations are performed in MATLAB Simulink with the help of the Simscape toolbox, here the controller is designed and then deployed as a C++ node in ROS.

1.3.1 ROS and Gazebo

Robot Operating System (ROS) [6] is one of the most popular open source framework for robot software development and deployment. Despite its name, ROS is a meta-operating system since it runs on Unix-based platform like Ubuntu. Even if it is not a real time operating system, ROS is provided with its own clock and communication system and is widely used for testing real time application thanks to the ability to integrate real time code.

Gazebo is a free complex 3D indoor and outdoor robot simulator [7]. It has a robust physics engine such as Open Dynamic Engine (ODE). Gazebo can be easily integrated with ROS through a set of Gazebo plugin that have the same ROS message interface, this allows to ease the deployment of physical robot and application development.



Figure 1.1. ROS.

1.3.2 MATLAB Simulink and Simscape

MATLAB [8] is a programming platform that can be used for data analysis, algorithm development and model and application elaboration. It uses the MATLAB matrix-based language, useful for computational mathematics. MATLAB is widely used among engineers for numerous applications, including control system design. Thanks to the integration with Simulink, it is possible to incorporate MATLAB algorithms into models and perform multi domain simulations.

Simulink [9] is a block diagram environment that, among its many features, allows for modelling and simulating dynamic systems and for automatic code generation. These two features are very useful for this project' purpose, allowing to design and test the controller in an user friendly environment and, only after, deploy the code in the ROS environment as C++ code.

Simscape [10] is a Simulink library that allows to model and simulate multidomain physical systems. Simscape allows to rapidly create models of physical systems within the Simulink environment.



Figure 1.2. MATLAB.

Chapter 2

Modelling

In this Chapter several models will be derived, analysed and explained. In particular, the derivation of two mathematical models will be the focus of the first section, the analysis of the adopted ROS model will be the focus of the second section and the development of the Simscape model will be the focus of the last section.

All these models will have an important role in the final goal of the thesis. The mathematical model will be used inside of the controller as reference model, the Simscape model will be the real plant to which the controller will be applied and it will be used to tune all the parameters to obtain a fully working controller and, in conclusion, the ROS model will be used inside the ROS and Gazebo environment to test the quadrotor and the final controller.

2.1 Mathematical Model

Two mathematical models will be derived with the Newton-Euler formulation and with the Euler-Lagrangian formulation. During the model validation phase, these two models will be tested with respect to the Simscape and ROS model and the best one will be chosen.

2.1.1 Reference Frames and Generalized Coordinates

A quadrotor is an aerial vehicle based on 4 rotors placed at the end of 4 arms. As shown in fig. 2.1, the system is symmetrical with respect to the $x_b - z_b$ and $y_b - z_b$ planes.

The two reference frames used for this model are the inertial (or world) reference frame $x - y - z$ and the body reference frame $x_b - y_b - z_b$ (Fig. 2.1). The transformation between the two reference frames is defined by the translation $\xi = [x, y, z]^T$ and the rotation $\eta = [\phi, \theta, \psi]^T$. Where ϕ, θ and ψ are the Euler angles corresponding to the ZYX (Roll-Pitch-Yaw) rotation.

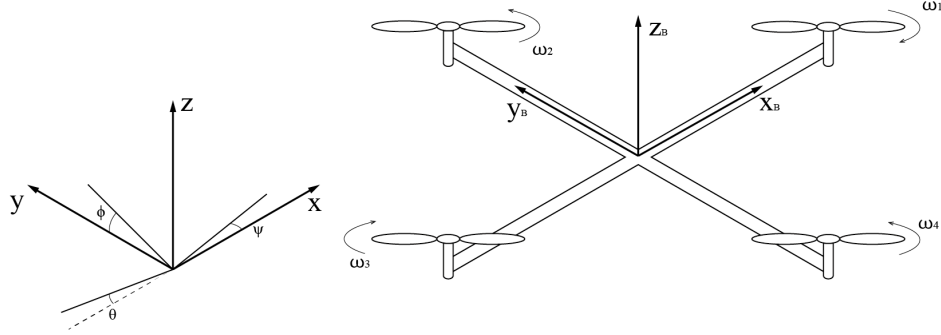


Figure 2.1. Reference frames.

The rotation matrix resulting from the Roll-Pitch-Yaw sequence is obtained by:

- rotation about the x axis with angle ψ for the yaw;

$$\mathbf{R}_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & -\sin(\psi) \\ 0 & \sin(\psi) & \cos(\psi) \end{bmatrix} \quad (2.1)$$

- rotation about the y axis with angle θ for the pitch;

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (2.2)$$

- rotation about the z axis with angle ϕ for the roll.

$$\mathbf{R}_z(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

As shown in [11], the final rotation matrix is $\mathbf{R}_b^w = \mathbf{R}_z(\phi)\mathbf{R}_y(\theta)\mathbf{R}_x(\psi)$.

$$\mathbf{R}_b^w = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\phi & S_\psi S_\theta C_\phi - C_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix} \quad (2.4)$$

since for an orthogonal matrix $\mathbf{R}^{-1} = \mathbf{R}^T$:

$$\mathbf{R}_w^b = \begin{bmatrix} C_\psi C_\theta & S_\psi C_\theta & -S_\theta \\ C_\psi S_\theta S_\phi - S_\psi C_\phi & S_\psi S_\theta S_\phi + C_\psi C_\phi & C_\theta S_\phi \\ C_\psi S_\theta C_\phi + S_\psi S_\phi & S_\psi S_\theta C_\phi - C_\psi S_\phi & C_\theta C_\phi \end{bmatrix} \quad (2.5)$$

where $C_x = \cos(x)$ and $S_x = \sin(x)$.

According to [12], the Kinematic equation which links the angular speed of the body frame $\boldsymbol{\nu} = [p, q, r]^T$ and the derivative of the ZYX euler angles $\dot{\boldsymbol{\eta}} = [\dot{\phi}, \dot{\theta}, \dot{\psi}]^T$ can be described thanks to the matrix \mathbf{W}_η :

$$\boldsymbol{\nu} = \mathbf{W}_\eta \dot{\boldsymbol{\eta}}, \quad \mathbf{W}_\eta = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & C_\theta S_\phi \\ 0 & -S_\phi & C_\theta C_\phi \end{bmatrix} \quad (2.6)$$

$$\dot{\boldsymbol{\eta}} = \mathbf{W}_\eta^{-1} \boldsymbol{\nu}, \quad \mathbf{W}_\eta^{-1} = \begin{bmatrix} 1 & S_\phi T_\theta & C_\phi T_\theta \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi / C_\theta & C_\phi / C_\theta \end{bmatrix} \quad (2.7)$$

In conclusion, the generalized coordinates that are used to define the position and orientation of the drone in the inertial reference frame are $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$. They are collected in a the single vector \mathbf{q} :

$$\mathbf{q} = \begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{\eta} \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix} \quad (2.8)$$

2.1.2 Dynamic Constants and Generalized Forces

Several external forces and torques are present in the system. Two forces and two torques are directly generated by the rotors while the gyroscopic effect will be described and analysed for the different formulation.

According to [5], the generalized forces generated by each rotor are :

$$\mathbf{F}_T = \omega^2 C_T \cdot \mathbf{k}_b \quad (2.9)$$

$$\mathbf{F}_D = -\omega C_D \cdot \mathbf{v}_A^{\perp, w} \quad (2.10)$$

$$\mathbf{M}_R = -\epsilon \omega C_R \cdot \mathbf{v}_A^{\perp, w} \quad (2.11)$$

$$\mathbf{M}_D = -\epsilon C_M \cdot \mathbf{F}_T \quad (2.12)$$

These forces model the real effects of the entire rotor dynamic (Fig.(2.2)). They are respectively vertical thrust, drag force, rolling moment and drag moment. In particular, the rolling moment has been slightly modified with respect to [5] in order to take in account the different effect of clockwise and counter clockwise

rotors. As shown in fig. 2.3, where the real lift distribution (\mathbf{L}) is presented, the direction of the resulting rolling moment (\mathbf{M}_R) is different for the two types of rotors.

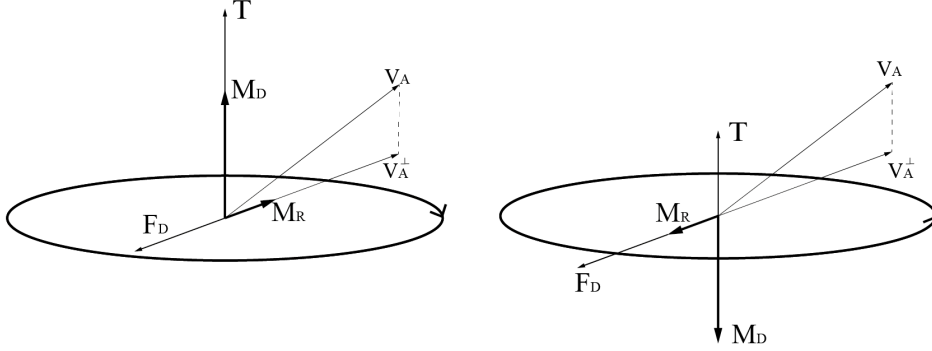


Figure 2.2. Forces and torques for a clockwise (left) and counter clockwise (right) rotor.

C_T , C_D , C_R and C_M are dynamic coefficients that can be empirically measured. ω is the angular velocity of the rotor, \mathbf{k}_b is the unit vector in the body z-direction. ϵ represents the direction of rotation of each rotor ($\epsilon = +1$ for counterclockwise rotors, $\epsilon = -1$ for clockwise rotors).

$\mathbf{v}_A^{\perp,w}$ is the projection of the linear velocity of the rotor \mathbf{v}_A^w in the rotor plane. The velocity \mathbf{v}_A^w is computed as the velocity of the body in the inertial reference frame plus the cross product between the angular speed and the arm represented in the inertial reference frame:

$$\mathbf{v}_A^w = \dot{\boldsymbol{\xi}} + \boldsymbol{\nu} \times (\mathbf{R}_b^w \mathbf{r}^b) \quad (2.13)$$

In particular:

- The Thrust force (2.9) represents the main effect of the rotor. The rotating blades produce an upward lift that is proportional to the square of the angular velocity of the rotor ω_i .
- The Drag moment (2.12) represents the effect of induced drag in an airfoil, this drag produces a resulting torque which is opposite with respect to the direction of rotation of the rotor (example: a clockwise rotor will produce a counter clockwise (upward) drag moment, as shown in fig. 2.2). This moment is proportional to the square of the angular velocity of the rotor ω_i .
- The Drag force (2.10) represents the effect of aerodynamic drag of a rotor that moves with linear velocity in space. It depends on the angular velocity of the rotor, on the drag coefficient and on the linear velocity of the rotor.

- The Rolling moment (2.11) represents the difference of lift produced by the blade that moves against the relative wind and the lift produced by the blade that moves in the same direction of the relative wind (Fig. 2.3). This results in an uneven distribution of lift and in a consequent torque that rolls the rotor with respect to the direction of motion. It depends on the direction of rotation of the rotor, on the angular velocity, on the rolling coefficient and on the linear velocity of the rotor.

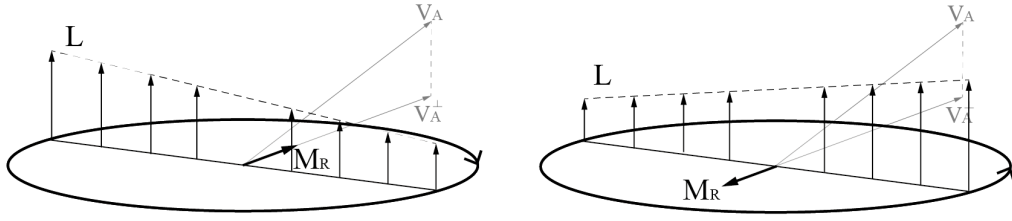


Figure 2.3. Real lift (\mathbf{L}) and equivalent rolling moment (\mathbf{M}_R) for a clockwise (left) and counter clockwise (right) rotor.

The generalized forces applied to the body are therefore Thrust and Drags, both are represented in the inertial reference frame since in both the following derivation the force section is computed in the inertial reference frame:

$$\begin{aligned} \mathbf{f} &= \sum_{i=1}^4 (\mathbf{R}_b^w \mathbf{T}^b + \mathbf{F}_{D,i}) \\ &= \mathbf{R}_b^w \begin{bmatrix} 0 \\ 0 \\ C_T(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \end{bmatrix} + \begin{bmatrix} C_D(-\omega_1 v_{Ax,1}^{\perp,w} - \omega_2 v_{Ax,2}^{\perp,w} - \omega_3 v_{Ax,3}^{\perp,w} - \omega_4 v_{Ax,4}^{\perp,w}) \\ C_D(-\omega_1 v_{Ay,1}^{\perp,w} - \omega_2 v_{Ay,2}^{\perp,w} - \omega_3 v_{Ay,3}^{\perp,w} - \omega_4 v_{Ay,4}^{\perp,w}) \\ C_D(-\omega_1 v_{Az,1}^{\perp,w} - \omega_2 v_{Az,2}^{\perp,w} - \omega_3 v_{Az,3}^{\perp,w} - \omega_4 v_{Az,4}^{\perp,w}) \end{bmatrix} \end{aligned} \quad (2.14)$$

where $v_{Ax,i}^{\perp,w}$, $v_{Ay,i}^{\perp,w}$ and $v_{Az,i}^{\perp,w}$ are the components of the projected linear velocity of each rotor i expressed in the inertial reference frame.

The generalized torques applied to the body are the Drag torques, the Rolling torques and the generalized forces multiplied by their arm. The torques will be expressed in the body reference frame since the two mathematical models that will be derived in the following sections keep the rotational part of the equation in this reference frame.

$$\boldsymbol{\tau} = \sum_{i=1}^4 (\mathbf{M}_{D,i} + \mathbf{M}_{R,i}^b + \mathbf{r}_i^b \times (\mathbf{F}_{T,i} + \mathbf{F}_{D,i}^b)) = \boldsymbol{\tau}_1 + \boldsymbol{\tau}_2 + \boldsymbol{\tau}_3 + \boldsymbol{\tau}_4 \quad (2.15)$$

For the first rotor the resulting torque is:

$$\boldsymbol{\tau}_1 = \begin{bmatrix} 0 \\ 0 \\ C_M C_T \omega_1^2 \end{bmatrix} + \mathbf{R}_w^b(C_R \omega_1 \mathbf{v}_{\mathbf{A},1}^{\perp,w}) + \begin{bmatrix} L \\ 0 \\ h \end{bmatrix} \times \left(\begin{bmatrix} 0 \\ 0 \\ C_T \omega_1^2 \end{bmatrix} + \mathbf{R}_w^b(-C_D \omega_1 \mathbf{v}_{\mathbf{A},1}^{\perp,w}) \right) \quad (2.16)$$

The arm of the rotor is $\mathbf{r}_1 = [L, 0, h]^T$, where L is the arm length of the quadrotor and h is the distance between the center of mass of the body and the rotor center of mass where the forces are applied.

$$\boldsymbol{\tau}_2 = \begin{bmatrix} 0 \\ 0 \\ -C_M C_T \omega_2^2 \end{bmatrix} + \mathbf{R}_w^b(C_R \omega_2 \mathbf{v}_{\mathbf{A},2}^{\perp,w}) + \begin{bmatrix} 0 \\ L \\ h \end{bmatrix} \times \left(\begin{bmatrix} 0 \\ 0 \\ C_T \omega_2^2 \end{bmatrix} + \mathbf{R}_w^b(-C_D \omega_2 \mathbf{v}_{\mathbf{A},2}^{\perp,w}) \right) \quad (2.17)$$

$$\boldsymbol{\tau}_3 = \begin{bmatrix} 0 \\ 0 \\ C_M C_T \omega_3^2 \end{bmatrix} + \mathbf{R}_w^b(C_R \omega_3 \mathbf{v}_{\mathbf{A},3}^{\perp,w}) + \begin{bmatrix} -L \\ 0 \\ h \end{bmatrix} \times \left(\begin{bmatrix} 0 \\ 0 \\ C_T \omega_3^2 \end{bmatrix} + \mathbf{R}_w^b(-C_D \omega_3 \mathbf{v}_{\mathbf{A},3}^{\perp,w}) \right) \quad (2.18)$$

$$\boldsymbol{\tau}_4 = \begin{bmatrix} 0 \\ 0 \\ -C_M C_T \omega_4^2 \end{bmatrix} + \mathbf{R}_w^b(C_R \omega_4 \mathbf{v}_{\mathbf{A},4}^{\perp,w}) + \begin{bmatrix} 0 \\ -L \\ h \end{bmatrix} \times \left(\begin{bmatrix} 0 \\ 0 \\ C_T \omega_4^2 \end{bmatrix} + \mathbf{R}_w^b(-C_D \omega_4 \mathbf{v}_{\mathbf{A},4}^{\perp,w}) \right) \quad (2.19)$$

2.1.3 Newton-Euler Derivation

The Newton-Euler method can be used to derive the dynamic equations of the system.

For the translational equations, expressing every vector in the inertial reference frame, the gravitational force and the generalized forces (2.14) contributes in the acceleration of the quadrotor:

$$\begin{aligned} m\ddot{\boldsymbol{\xi}} &= m\mathbf{g} + \mathbf{f} \\ m\ddot{\boldsymbol{\xi}} &= m \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \mathbf{f} \end{aligned} \quad (2.20)$$

For the rotational equations, expressing every vector in the body reference frame, the angular acceleration of the inertia $\mathbf{I}\dot{\boldsymbol{\nu}}$, the centripetal forces $\boldsymbol{\nu} \times (\mathbf{I}\boldsymbol{\nu})$ and the gyroscopic forces $\boldsymbol{\Gamma}$ are equal to the external generalized forces (2.15):

$$\mathbf{I}\dot{\boldsymbol{\nu}} + \boldsymbol{\nu} \times (\mathbf{I}\boldsymbol{\nu}) + \boldsymbol{\Gamma} = \boldsymbol{\tau} \quad (2.21)$$

where \mathbf{I} is the inertia matrix of the drone and $\mathbf{\Gamma}$ has the form, as shown in [13]:

$$\mathbf{I} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \quad (2.22)$$

$$\mathbf{\Gamma} = I_{z,p} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ \omega_\Gamma \end{bmatrix} \quad (2.23)$$

where $I_{z,p}$ is the rotor moment of inertia with respect to the z rotor axis and ω_Γ is the sum of the angular velocities with the proper sign.

$$\omega_\Gamma = -\omega_1 + \omega_2 - \omega_3 + \omega_4 \quad (2.24)$$

In conclusion, the total set of differential equation derived with the Newton-Euler formulation is:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_x & 0 & 0 \\ 0 & 0 & 0 & 0 & I_y & 0 \\ 0 & 0 & 0 & 0 & 0 & I_z \end{bmatrix}^{-1} \left(m \begin{bmatrix} 0 \\ 0 \\ -g \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ -(I_y - I_z)qr \\ -(I_z - I_x)pr \\ -(I_x - I_y)pq \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ I_{z,p}q\omega_\Gamma \\ -I_{z,p}p\omega_\Gamma \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau} \end{bmatrix} \right) \quad (2.25)$$

2.1.4 Euler-Lagrangian Derivation

The Lagrangian is the sum of the rotational and translational kinematic energy minus the potential energy of the system.

$$\begin{aligned} L &= K_{trans} + K_{rot} + E_{pot} \\ &= \frac{1}{2} m \dot{\boldsymbol{\xi}}^T \dot{\boldsymbol{\xi}} + \frac{1}{2} \boldsymbol{\nu}^T \mathbf{I} \boldsymbol{\nu} - m \mathbf{g} \end{aligned} \quad (2.26)$$

The Euler-Lagrangian equations are:

$$\begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau} \end{bmatrix} = \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{\mathbf{q}}} \right) - \frac{\delta L}{\delta \mathbf{q}} \quad (2.27)$$

The linear equation can be derived in the following way:

$$\mathbf{f} = \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{\boldsymbol{\xi}}} \right) - \frac{\delta L}{\delta \boldsymbol{\xi}} = \frac{d}{dt} (m \dot{\boldsymbol{\xi}}) - m \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} = m \ddot{\boldsymbol{\xi}} - m \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \quad (2.28)$$

In order to derive the rotational Euler-Lagrangian equation the Kinetic energy is written substituting $\boldsymbol{\nu}$ with $\boldsymbol{\eta}$:

$$\mathbf{J}(\boldsymbol{\eta}) = \mathbf{W}_\eta^T \mathbf{I} \mathbf{W}_\eta = \begin{bmatrix} I_x & 0 & -I_x S_\theta \\ 0 & I_y C_\phi^2 + I_z S_\phi^2 & (I_y - I_z) C_\phi S_\phi C_\theta \\ -I_x S_\theta & (I_y - I_z) C_\phi S_\phi C_\theta & I_x S_\theta^2 + I_y S_\phi^2 C_\theta^2 + I_z C_\phi^2 C_\theta^2 \end{bmatrix} \quad (2.29)$$

The rotational Lagrangian can be then rewritten as:

$$K_{rot} = \frac{1}{2} \dot{\boldsymbol{\eta}}^T \mathbf{J} \dot{\boldsymbol{\eta}} \quad (2.30)$$

The rotational equation can be derived in the following way:

$$\boldsymbol{\tau} = \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{\boldsymbol{\eta}}} \right) - \frac{\delta L}{\delta \boldsymbol{\eta}} = \frac{d}{dt} (\mathbf{J} \dot{\boldsymbol{\eta}}) - \frac{1}{2} \frac{\delta}{\delta \boldsymbol{\eta}} (\dot{\boldsymbol{\eta}}^T \mathbf{J} \dot{\boldsymbol{\eta}}) = \mathbf{J} \ddot{\boldsymbol{\eta}} + \frac{d}{dt} (\mathbf{J}) \dot{\boldsymbol{\eta}} - \frac{1}{2} \frac{\delta}{\delta \boldsymbol{\eta}} (\dot{\boldsymbol{\eta}}^T \mathbf{J} \dot{\boldsymbol{\eta}}) \quad (2.31)$$

The final result can be rewritten as:

$$\boldsymbol{\tau} = \mathbf{J} \ddot{\boldsymbol{\eta}} + \mathbf{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) \dot{\boldsymbol{\eta}} \quad (2.32)$$

where the matrix $\mathbf{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})$ is the Coriolis matrix that contains the centripetal terms.

As shown in [14], the matrix $\mathbf{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})$ has the form:

$$\mathbf{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \quad (2.33)$$

$$C_{11} = 0$$

$$C_{12} = (I_y - I_z)(\dot{\theta} C_\phi S_\phi + \dot{\psi} S_\phi^2 C_\theta) + (I_z - I_y) \dot{\psi} C_\phi^2 C_\theta - I_x \dot{\psi} C_\theta$$

$$C_{13} = (I_z - I_y) \dot{\psi} C_\phi S_\phi C_\theta^2$$

$$C_{21} = (I_z - I_y)(\dot{\theta} C_\phi S_\phi + \dot{\psi} S_\phi^2 C_\theta) + (I_y - I_z) \dot{\psi} C_\phi^2 C_\theta + I_x \dot{\psi} C_\theta$$

$$C_{22} = (I_z - I_y) \dot{\phi} C_\phi S_\phi$$

$$C_{23} = -I_x \dot{\psi} S_\theta C_\theta + I_y \dot{\psi} S_\phi^2 C_\theta S_\theta + I_z \dot{\psi} C_\phi^2 C_\theta S_\theta$$

$$C_{31} = (I_y - I_z) \dot{\psi} C_\theta^2 C_\phi S_\phi - I_x \dot{\theta} C_\theta$$

$$C_{32} = (I_z - I_y)(\dot{\theta} C_\phi S_\phi S_\theta + \dot{\phi} S_\phi^2 C_\theta) + (I_y - I_z) \dot{\phi} C_\phi^2 C_\theta + I_x \dot{\psi} S_\theta C_\theta - I_y \dot{\psi} S_\phi^2 C_\theta S_\theta - I_z \dot{\psi} C_\phi^2 C_\theta S_\theta$$

$$C_{33} = (I_y - I_z) \dot{\phi} C_\phi S_\phi C_\theta^2 - I_y \dot{\theta} S_\phi^2 C_\theta S_\theta - I_z \dot{\theta} C_\phi^2 C_\theta S_\theta + I_x \dot{\theta} S_\theta C_\theta$$

This formulation does not take in account the gyroscopic effect due to the rotating propellers. The term $\boldsymbol{\Gamma}$ (2.23) previously defined is added to the final equation:

$$\boldsymbol{\tau} = \mathbf{J} \ddot{\boldsymbol{\eta}} + \mathbf{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) \dot{\boldsymbol{\eta}} + \boldsymbol{\Gamma} \quad (2.34)$$

The total set of differential equation derived with the Euler-Lagrangian method is:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \mathbf{m}^{3 \times 3} & \mathbf{0}^{3 \times 3} \\ \mathbf{0}^{3 \times 3} & \mathbf{J}^{3 \times 3} \end{bmatrix}^{-1} \left(\begin{bmatrix} 0 \\ 0 \\ -g \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{0}^{3 \times 3} & \mathbf{0}^{3 \times 3} \\ \mathbf{0}^{3 \times 3} & \mathbf{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ I_{z,p}(C_\phi \dot{\theta} + C_\theta S_\phi \dot{\psi})\omega_\Gamma \\ -I_{z,p}(\dot{\phi} - S_\theta \dot{\psi})\omega_\Gamma \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau} \end{bmatrix} \right) \quad (2.35)$$

2.1.5 Simulink File Of The Mathematical Model

In order to simulate the mathematical models and given the fact that these models will be used as reference plant for the controller, two different simulink files are developed starting from the two sets of differential equations.

As matter of fact the two sets can be integrated two times given the input $\omega_i(t)$ (for $i = 1, 2, 3, 4$) and the initial conditions $\mathbf{q}(0) = [x(0), y(0), z(0), \phi(0), \theta(0), \psi(0)]^T$ and $\dot{\mathbf{q}}(0) = [\dot{x}(0), \dot{y}(0), \dot{z}(0), \dot{\phi}(0), \dot{\theta}(0), \dot{\psi}(0)]^T$ to obtain a complete evolution of the system in time. For what concerns the Newton-Euler set of differential equation, the variables in this set are $\ddot{\boldsymbol{\xi}}$ and $\dot{\boldsymbol{\nu}}$ instead of $\ddot{\mathbf{q}} = [\ddot{\boldsymbol{\xi}}; \ddot{\boldsymbol{\eta}}]$. To link $\ddot{\boldsymbol{\eta}}$ with $\dot{\boldsymbol{\nu}}$ the following formula can be derived:

$$\begin{aligned} \ddot{\boldsymbol{\eta}} &= \frac{d}{dt} (\mathbf{W}_\eta^{-1} \dot{\boldsymbol{\nu}}) = \frac{d}{dt} (\mathbf{W}_\eta^{-1}) \dot{\boldsymbol{\nu}} + \mathbf{W}_\eta^{-1} \ddot{\boldsymbol{\nu}} \\ &= \begin{bmatrix} 0 & \dot{\phi} C_\phi T_\theta + \dot{\theta} S_\phi / C_\theta^2 & -\dot{\phi} S_\phi T_\theta + \dot{\theta} C_\phi / C_\theta^2 \\ 0 & -\dot{\phi} S_\phi & -\dot{\phi} C_\phi \\ 0 & \dot{\phi} C_\phi / C_\theta + \dot{\theta} S_\phi T_\theta / C_\theta & -\dot{\phi} S_\phi / C_\theta + \dot{\theta} C_\phi T_\theta / C_\theta \end{bmatrix} \dot{\boldsymbol{\nu}} + \mathbf{W}_\eta^{-1} \ddot{\boldsymbol{\nu}} \end{aligned} \quad (2.36)$$

The simulink file related to the Newton-Euler set of differential equations (2.25) is shown in fig.2.4. The simulation gets as input $\boldsymbol{\omega} = [-\omega_1, \omega_2, -\omega_3, \omega_4]$ which are the angular velocities of the rotors with the proper sign and produces as output \mathbf{q} and $\mathbf{d}\mathbf{q}$. Going deeper in the code it is possible to see the first sum between the terms:

- Thrust and Drag moment summed with Rolling moment and Drag Force (last term of (2.25));
- $\boldsymbol{\Gamma}$ (third term of (2.25));
- $\boldsymbol{\nu} \times (\mathbf{I}\boldsymbol{\nu})$ (second term of (2.25));
- gravity effect (first term of (2.25));

This total sum is then left multiplied by \mathbf{B}^{-1} where:

$$\mathbf{B} = \begin{bmatrix} \mathbf{m}^{3 \times 3} & \mathbf{0}^{3 \times 3} \\ \mathbf{0}^{3 \times 3} & \mathbf{I}^{3 \times 3} \end{bmatrix} \quad (2.37)$$

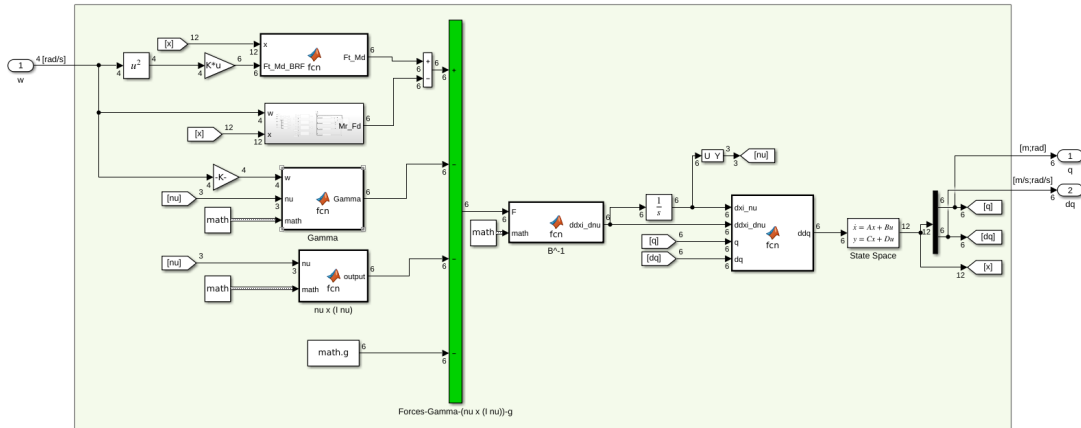


Figure 2.4. Simulink file of the Newton-Euler differential equations

- Thrust and Drag moment summed with Rolling moment and Drag Force (last term of (2.35));
- $\mathbf{C} \cdot d\mathbf{q}$ (second term of (2.35));
- gravity effect (first term of (2.35));
- $\mathbf{\Gamma}$ (third term of (2.35));

The output of this sum is then left multiplied by the matrix \mathbf{B}_1^{-1} , where:

$$\mathbf{B}_1 = \begin{bmatrix} \mathbf{m}^{3 \times 3} & \mathbf{0}^{3 \times 3} \\ \mathbf{0}^{3 \times 3} & \mathbf{J}^{3 \times 3} \end{bmatrix} \quad (2.38)$$

At this point it is possible to double integrate $\mathbf{d}\mathbf{d}\mathbf{q}$ in the *State Space* block to obtain a single vector with $[\mathbf{q}; \mathbf{d}\mathbf{q}]$.

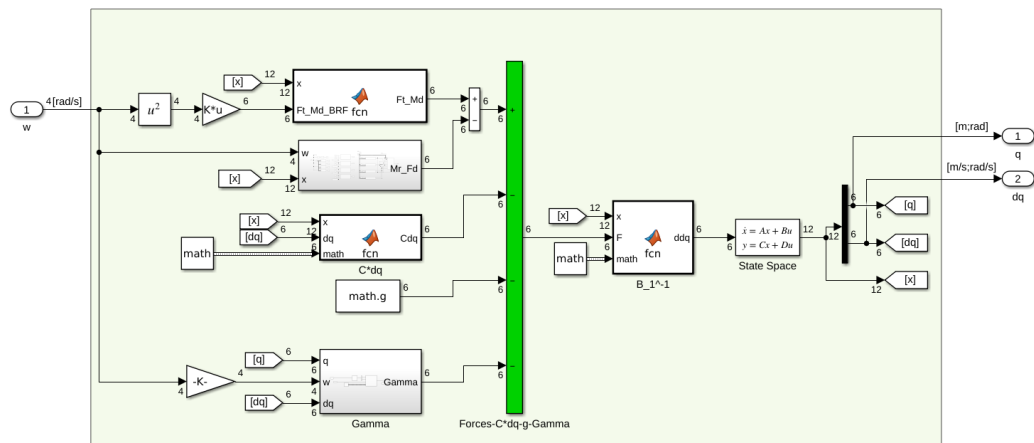


Figure 2.5. Simulink file of the Euler-Lagrange differential equations

2.2 ROS Model

In order to have an external testing environment for the drone simulation and the controller application, ROS and gazebo are chosen. The rotorS simulator [5] is adopted, this group of packages contains all the urdf files, plugins, custom messages and custom worlds that can be exploited in Gazebo.

The simulator provides a high number of different drones, plugins and controllers. For this study just the simple drone urdf, launch files and motor plugin are used. In particular, the hummingbird quadrotor is used for the entire study, this drone corresponds exactly to the structure described in the mathematical model. Despite the fact that the visual mesh can look complicated and not symmetrical, the inertial characteristics describe a simple structure with z-y and z-x symmetry.

ROS is an interesting test environment thanks to its modularity and the simplicity of modifying pre-existing code and nodes. In particular, MATLAB and ROS have a strong crossplatform connection that allow to write code in MATLAB and Simulink and to use it directly in ROS as a node. In addition, it is possible to convert the MATLAB and Simulink code in C++ and deploy it as an independent ROS node. Thanks to the MATLAB coder, the simulink controller can be converted in C++ code and deployed in order to test it in another environment.

2.2.1 Custom Simulator

The basic environment of the rotorS simulator is modified and upgraded. A new node is added to the basic workspace and many files are modified in order to simplify its use in this specific case.

2.2.2 Simulator workspace

The packages needed for the simulation are divided in simulator packages, utility package and custom package.

The simulator packages are contained in the `rotors_simulator` folder:

- **rotors_description**: contains the entire urdf and mesh library of the simulator. Many different drones can be simulated like the classic quadrotor (hummingbird) or a more complex hexarotor (firefly);
- **rotors_gazebo**: contains all the launch files needed to run the simulation. Many different worlds are present beyond the basic empty world;
- **rotors_gazebo_plugin**: contains all the C++ files that can be used as plugins in the Gazebo environment.

The utility package is the `mav_comm` which contains the custom messages used in the simulator. These messages allow to compress important information about

drones such as the angular rotor velocities that can be easily used by plugins or sent over topics.

2.2.3 Custom package

The custom package has been developed at University of Surrey, it consists of one node (`communication_node`) which is implemented in order to have a new layer between the motor speed input provided by the Simulink code and the proper motor speed input needed by Gazebo.

The main issue of using MATLAB Simulink to generate a ROS node is related to the impossibility of implementing custom ROS messages. To overcome this problem, the communication node works as an intermediate ROS layer to convert the classic ROS messages (such as `Float64MultyArray`) from Simulink into custom ROS messages (fig. 2.8).

In particular, the rotorS simulator exploits the *mav_msgs/Actuators* defined in the utility package `mav_comm`. This type of message can carry information such as the position, the angular velocity and the angular acceleration of each rotor. The Simulink code (fig. 2.6) publishes the vector of four `Float64` numbers (message type: `Float64MultyArray`) to the intermediate topic `hummingbird/command/raw/motor_speed`.

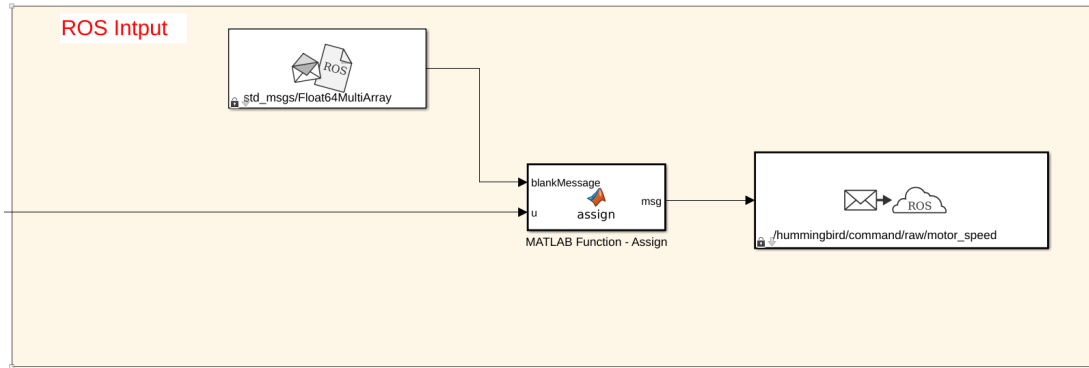


Figure 2.6. ROS input section from Simulink code

At the same time, the `communication_node` (fig. 2.7) subscribes to the intermediate topic `hummingbird/command/raw/motor_speed`, converts the `Float64MultyArray` messages into *mav_msgs/Actuators* and publishes them in the final topic `hummingbird/command/motor_speed`.

```

communication_node.py (~/rotors_sim_ws/src/communication_pkg/src) - gedit
Open Save

1 #!/usr/bin/env python
2
3 import rospy
4 from mav_msgs.msg import Actuators
5 from std_msgs.msg import Float64MultiArray
6 from std_msgs.msg import Float64
7
8 def callback(data):
9     msg = Actuators(); #builds the msg type (imported from mav_msgs.msg)
10
11     msg.angular_velocities = data.data; #copies the raw msg in the Actuator msg
12
13     pub = rospy.Publisher('hummingbird/command/motor_speed', Actuators, queue_size=1000) #subscribes to final topic
14
15     pub.publish(msg) #publishes the msg in the final topic
16
17
18 def listener():
19     #initialization of the node
20     rospy.init_node('communication_node', anonymous=True)
21
22     #subscription to the intermediate topic so that each time a message appear the callback func is called
23     rospy.Subscriber('hummingbird/command/raw/motor_speed', Float64MultiArray, callback)
24
25     # spin() simply keeps python from exiting until this node is stopped
26     rospy.spin()
27
28 if __name__ == '__main__':
29     listener()

```

Python Tab Width: 8 Ln 18, Col 17 INS

Figure 2.7. Communication_node python script.

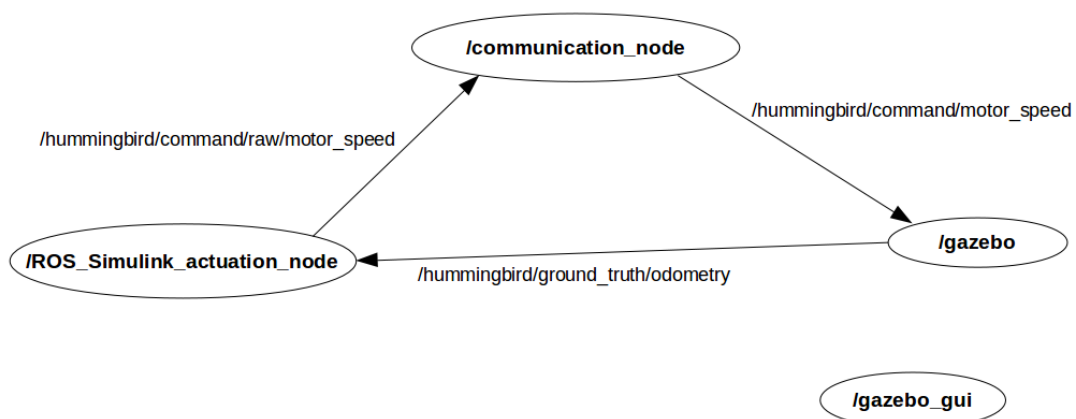


Figure 2.8. ROS nodes and topics. Nodes are contained into circles and connected by topics.

2.2.4 Key files for ROS simulations

To initialize the Gazebo simulation environment the `mav.launch` file present in the `rotor_gazebo` package is executed.

```
$ roslaunch rotors_gazebo mav.launch
```

This launch file is structured as follow (also shown in fig. 2.9):

- definition of main variables (Section 1);
- `empty_world.launch` is included (Section 2), this launch file belongs to the basic `gazebo_ros` package. It starts Gazebo and its GUI with an empty world. Among the arguments of this call the world "basic" is chosen, this is described in the `rotors_gazebo/world` folder;
- `spawn_mav.launch` is included (Section 3), this launch file spawns the robot in the Gazebo environment given the proper parameters;
- custom communication node is executed (Section 4).

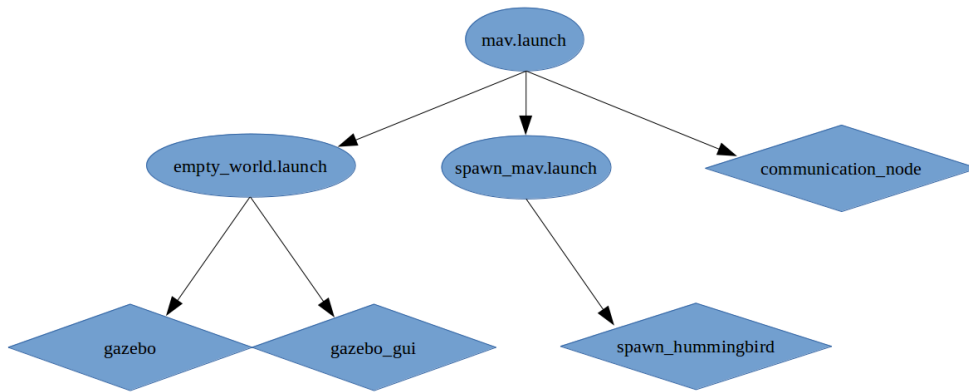


Figure 2.9. Representation of the launch file composition. Circles represent launch files, rhombuses represent final nodes.

The `spawn_mav.launch` file spawns the drone in Gazebo given a certain number of inputs like its urdf, initial position and main parameters. The robot physical description is given to ROS by the urdf (Universal Robot Description Format). This format allows to write the link and joint chain, the visual and the collision characteristics of each link, the plugins applied to each link and many more characteristics that entirely describe the robot and its dynamic properties. This description is written in the `hummingbird_base.xacro` present in the `rotor_description` package and it is composed as follow (also shown in fig. 2.10):

- `components_snippets.xacro` is included (Section 1), this xacro file contains the entire list and description of components that can be mounted on the model (imu sensors, gps sensors, magnetometers etc.);

- hummingbird.xacro is included (Section 2), this xacro file contains the numeric parameters and the physical description of the base and the rotors of the drone.
- the "default_imu" is mounted (Section 3), this sensor is chosen among the ones present in the components_snippets.xacro list;
- the "ground_truth_imu_and_odometry" is mounted (Section 4), this sensor is chosen among the ones present in the components_snippets.xacro list and it is needed to print the state variable value in a given topic.

The hummingbird.xacro file is organized as follow:

- definition of the main properties (Section 1)
- the multirotor_base.xacro is included (Section 2), this xacro file contains the default macros for a drone base and a general vertical rotor;
- the "multirotor_base_macro" is called (Section 3) in order to build the body of the quadrotor;
- the "vertical_rotor" macros are called (Section 4) in order to build the four rotors of the quadrotor.

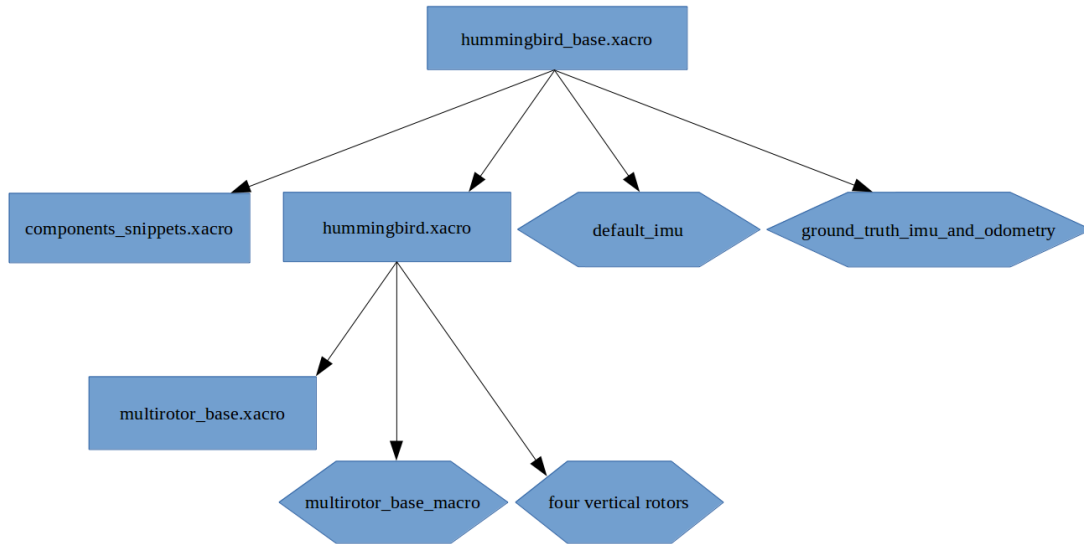


Figure 2.10. Representation of the urdf file composition. Rectangles represent xacro files, hexagons represent final components of the drone.

The plugin present on each rotor is described by the gazebo_motor_model.cpp file present in the `rotor_gazebo_plugin` package. This plugin is needed to activate the rotors rotation, upload their position at each simulation step, compute

the forces and torques etc.

The plugin file is composed by many functions but the most important one is the **GazeboMotorModel::UpdateForcesAndMoments()** function, which computes and apply the several forces and torques to the drone:

- in Section 1 the thrust force (eq. (2.9)) is computed and applied to the rotor;
- in Section 2 the drag force (eq. (2.10)) is computed and applied to the rotor;
- in Section 3 the drag moment (eq. (2.12)) is computed and applied to the body;
- in Section 4 the rolling moment (eq. (2.11)) is computed and applied to the body.

Notes:

- In this plugin the torques are applied to the body of the drone and not to the single rotor. This is needed because the gazebo plugin does not work otherwise. In the simulink model the forces and torques are all applied to the rotors frame in order to follow the mathematical model;
- The drag moment application in Section 3 of the original rotorS simulator script is modified. In the original code the drag moment is rotated according to the difference of orientation between the rotor and the body, this rotation is needed because this plugin can be exploited by many other drones with non fixed rotors. In the case of study, the rotors are fixed and therefore drag moment (which has only the z-component different from 0) is not rotated when transformed in the body reference frame. In particulare, the line:

```
parent_links.at(0)->AddRelativeTorque(drag_torque_parent_frame);
```

is substituted by:

```
parent_links.at(0)->AddRelativeTorque(drag_torque);
```

- The rolling moment computation in Section 3 of the original rotorS simulator script is modified. To distinguish from clockwise and counterclockwise rotors the *turning_direction_* coefficient has been added in the computation of this torque. In particular, the line:

```
rolling_moment = -std::abs(real_motor_velocity) *
                  rolling_moment_coefficient_ *
                  body_velocity_perpendicular;
```

is substituted by:

```
rolling_moment = -std::abs(real_motor_velocity) *
    rolling_moment_coefficient_ *
    body_velocity_perpendicular*turning_direction_;
```

2.3 Simscape Model

Simscape is a Simulink extension that allows to model, to test and to visualize multidomain systems. This environment allows to build a model in Simulink which closely recalls the ROS model, this Simscape model is then used for quick testing and tuning. In this case, the quadrotor physical and visual characteristic are initially imported from the ROS model thanks to the MATLAB command `smimport(.urdf)`. The initial `hummingbird_base.xacro` file is converted in urdf format thanks to an online converter and then fed in the MATLAB command. The first output of this command is a rough representation of the final model. Many modifications are applied to the model, the base and rotor subsystems are defined. In each rotor the several forces and torques are added. A sensor section is defined in order to capture all the important variables that are later sent to MATLAB for analysis. The final Simscape model (fig. 2.11) is composed by a configuration section, a base subsystem (*hummingbird_base*), four rotor subsystem (*hummingbird_rotor_i*, $i=1,2,3,4$) with each its own input rotor velocity and a sensor subsystem.

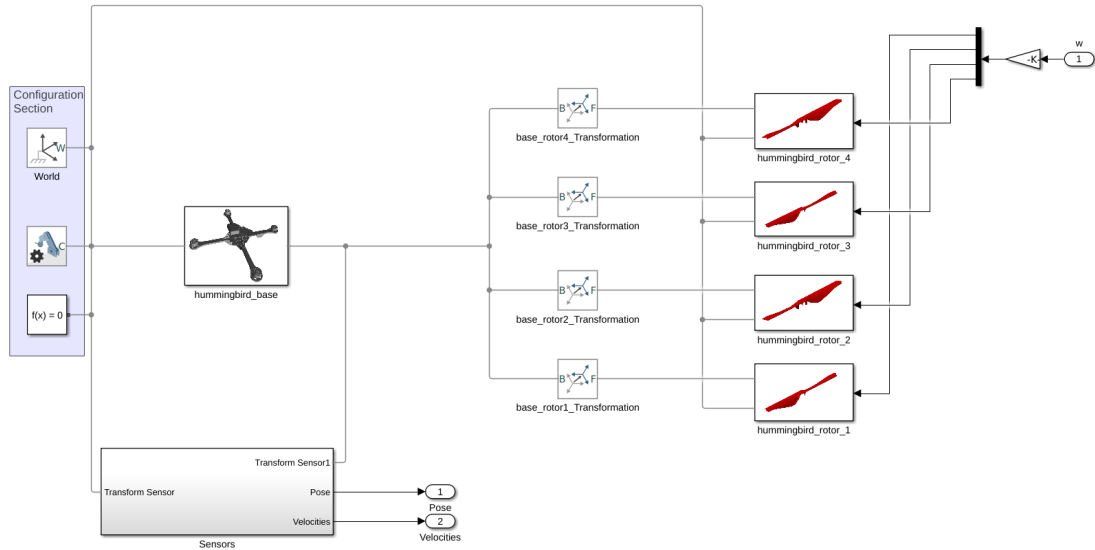


Figure 2.11. Simscape Model.

The configuration section defines the world reference frame, the gravity settings and the solver settings for the entire model.

The hummingbird_base subsystem (fig. 2.12) is composed by a visual and inertia block where the physical characteristic are defined, a reference frame block to initialize the body reference frame (reference frame that is fixed to the drone) and a six-degree of freedom block that allows the free relative motion between the world reference frame and the body reference frame.

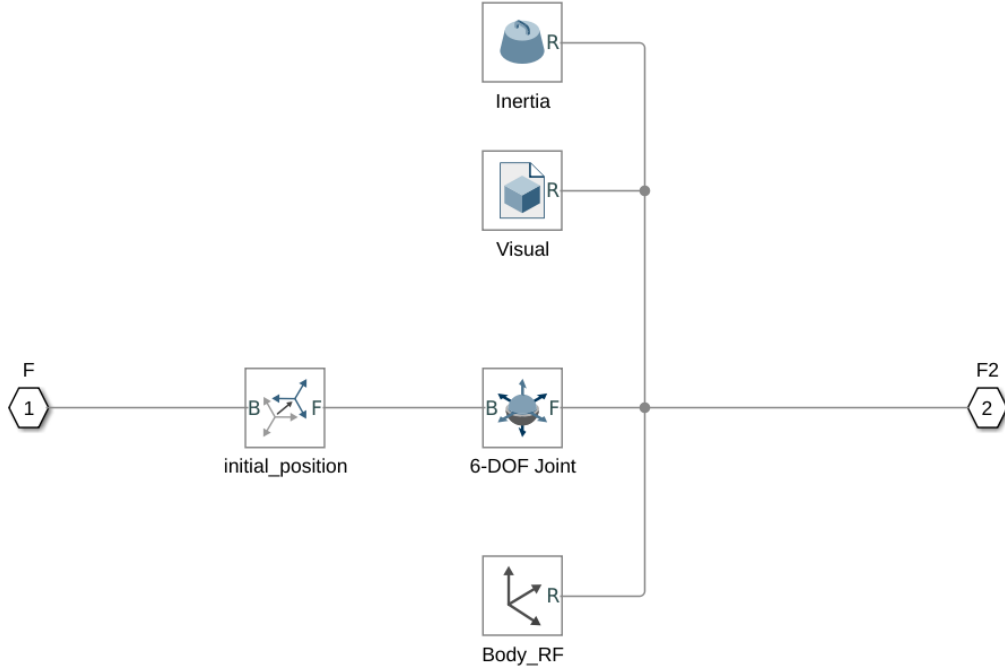


Figure 2.12. Simscape Model Base.

The hummingbird_rotor subsystem (fig. 2.13) revolves around the *hummingbird_joint* block which allows the rotation of the propeller. This block is connected to the rotor velocity input, to the rotor_fixed_rf, to the *inertia* and *visual* block of the propeller and finally has as output the rotor velocity. Then, in the bottom left there is a section for sensing and computing the perpendicular linear velocity of the rotor in the world reference frame. This velocity is used together with the rotor angular velocity to compute forces and torques. In conclusion, the three purple blocks represent the computation and application of forces and torques (eq. (2.9), (2.10), (2.11), (2.12)) in the rotor_fixed_rf which is the reference frame corresponding to the initial reference frame of the rotor.

The sensor subsystem (fig. 2.14) is based on the *Transform Sensor* block which is able to sense the state of a reference frame with respect to another reference frame. In this case, this block is used to calculate the position, the orientation, the linear velocity and the Euler angle derivatives of the body reference frame

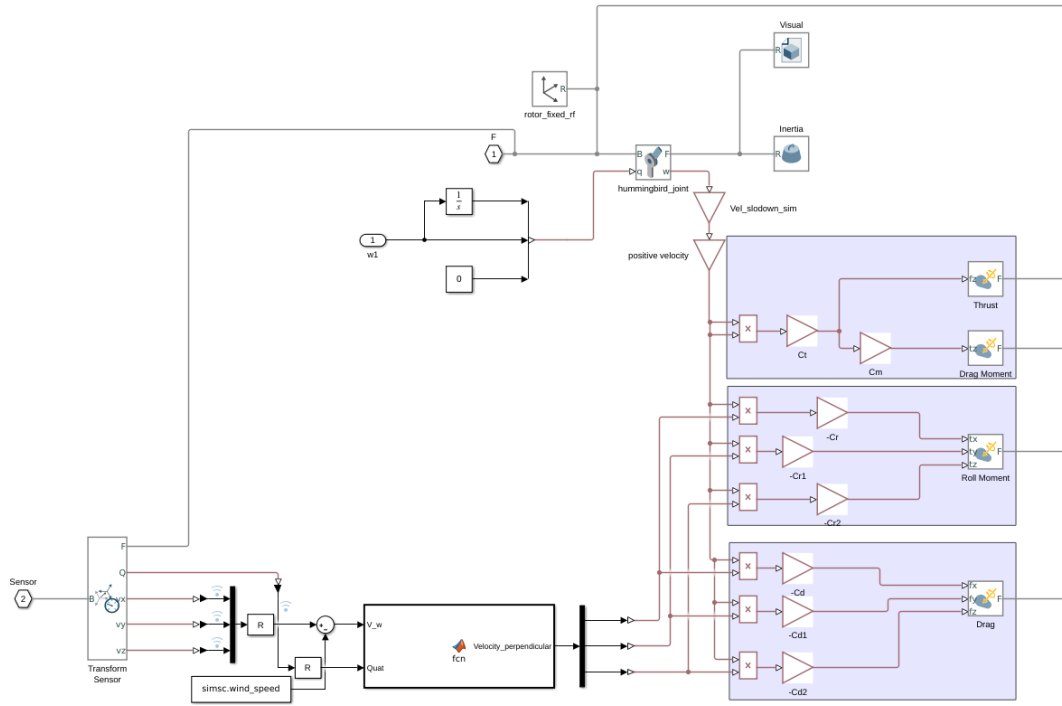


Figure 2.13. Simscape Model Rotor.

with respect to the world (inertial) reference frame.

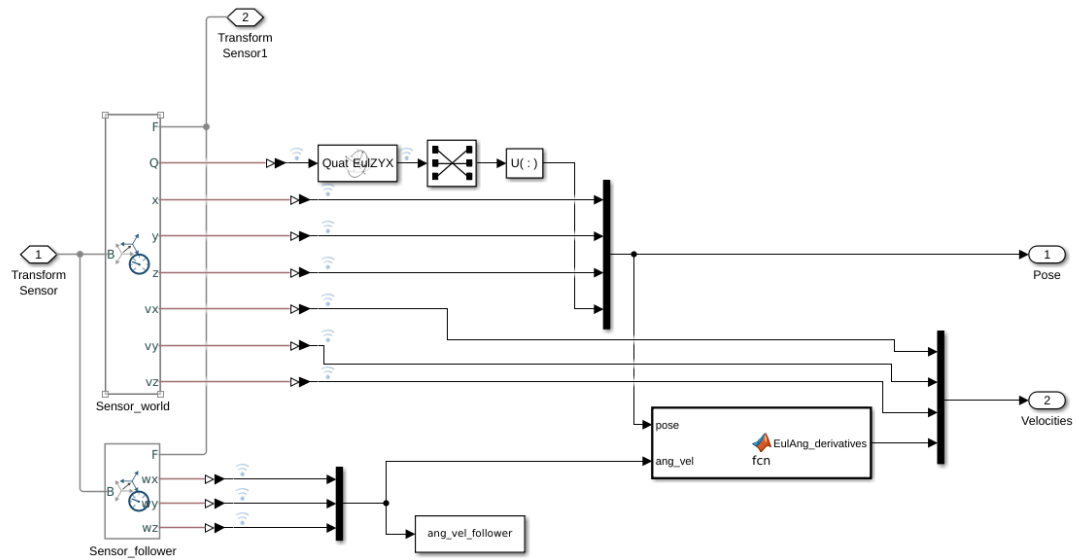


Figure 2.14. Simscape Model Sensor.

Chapter 3

Model Validation

The main focus of this chapter is the comparison of the several models previously analysed. The results of this comparison should reveal a good matching between all the different models. This result is needed in order to reach the final goal of this work. Every model will have a role in the final demonstration, the mathematical model will be needed inside the controller, the Simscape model will be the real controlled plant and the ROS model will be used inside the Gazebo environment with the developed controller to provide a further trial.

3.1 Actuation and Acquisition Node

In order to set up the model validation experiment between the Simscape, the mathematical and the ROS models a simple node able to provide input and read output from Gazebo is needed.

For instance, the Simulink code is created (Fig. 3.1), this code is able to connect to the `hummingbird/command/raw/motor_speed` topic and publish an open loop input signal. At the same time it can read from the `/hummingbird/-ground_truth/odometry` and from the `/hummingbird/motor_speed/` respectively the state variable and the rotational motor speed. In conclusion it can save these values in the `ros_output_complete.mat` file.

The final Simulink code is then converted by the Simulink encoder in a C++ file as shown in [?]. The output of such conversion are three files (a `.sh` file, a `rtw` file and a `.tgz` file). This three files are dragged into the workspace folder and built as a ROS package with the following command line:

```
$ ./build_ros_model.sh ROS_actuation_and_acquisition_node.tgz .
```

The data is then collected in a specific simulation. The classic `mav.launch` file is run to set up the gazebo environment and to spawn the drone; after that, the `ROS_actuation_and_actuation_node` is run with the command:

```
~/rotors_sim_ws$ rosrn ros_actuation_and_acquisition_node
ros_actuation_and_acquisition_node
```

After the simulation the `ros_output_complete.mat` file will appear in the directory from where the node was run. In a second moment these results are compared to the ones from the simulink and mathematical model.

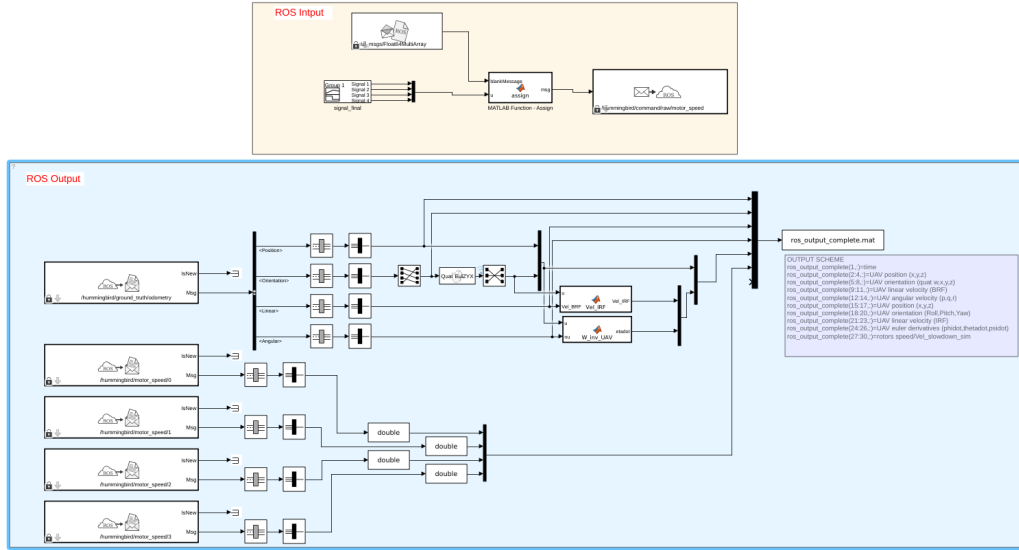


Figure 3.1. Actuation and Acquisition node.

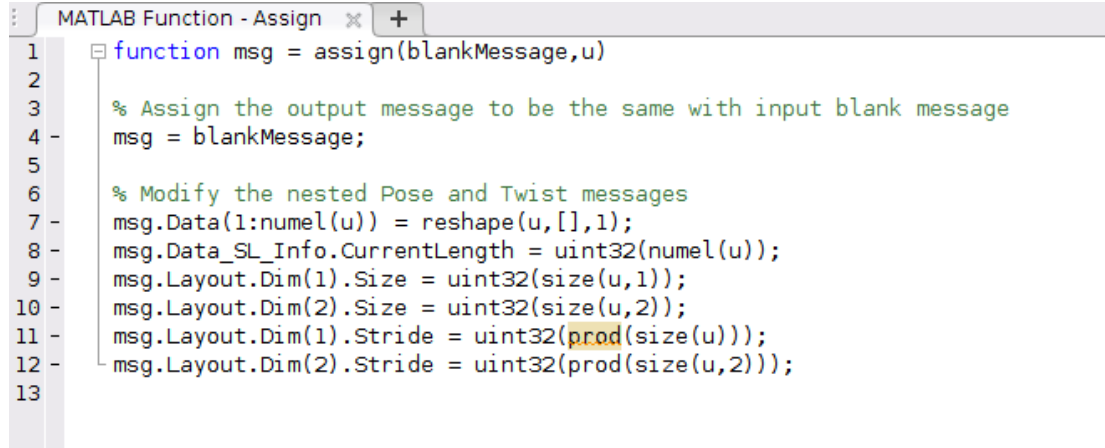
3.1.1 ROS Input

The ROS input section is based on a *BlankMessage* block, a *signal builder* block, a *MATLAB assign function* and a *Publish* block. These components respectively build the message and the signal, assign the signals to the right BUS section and publish the final message in the `/hummingbird/command/raw/motor_speed` topic. The Simulink *signal routing* library contains a *BUS assignment* block which can be usually used to directly assign a scalar value to a Blank Message. In this case this block cannot be used because the signal is a vector of four scalars. To overcome such problem a dedicated MATLAB Function is built (Fig. 3.2).

Note: if the assign block is used in another simulink code the BUS name corresponding to the output must be changed. Steps:

1. the block has to be copied in the new simulink and connected to the *BlankMessage*, to the input and to the *Publish* Block;
2. the simulink can be run. The buses will be built among the MATLAB variables and an error will occur;

3. open the MATLAB Function in the MATLAB workspace. In EDITOR/SIMULINK open the Edit Data section;
4. the output msg DataType must be modified in BUS: <object name>, with <object name> corresponding to one of the BUS variable built in the MATLAB workspace (for example: Bus: SL_<file_name>_Float64MultiArray where <file_name> is the file name). To edit this field click on edit and pick the BUS from the list of active BUSES (fig.3.3).



```

1  function msg = assign(blankMessage,u)
2
3  % Assign the output message to be the same with input blank message
4  msg = blankMessage;
5
6  % Modify the nested Pose and Twist messages
7  msg.Data(1:numel(u)) = reshape(u,[],1);
8  msg.Data_SL_Info.CurrentLength = uint32(numel(u));
9  msg.Layout.Dim(1).Size = uint32(size(u,1));
10 msg.Layout.Dim(2).Size = uint32(size(u,2));
11 msg.Layout.Dim(1).Stride = uint32(prod(size(u)));
12 msg.Layout.Dim(2).Stride = uint32(prod(size(u,2)));
13

```

Figure 3.2. MATLAB Assign Function.

3.1.2 ROS Output

The ROS output section is based on a subscribe block, a signal conversion part and a To File block. This allows to read from a certain topic and save the output in a .mat file.

The output of such section is composed by the raw output variables, by the converted output variables and by the rotor motor speeds.

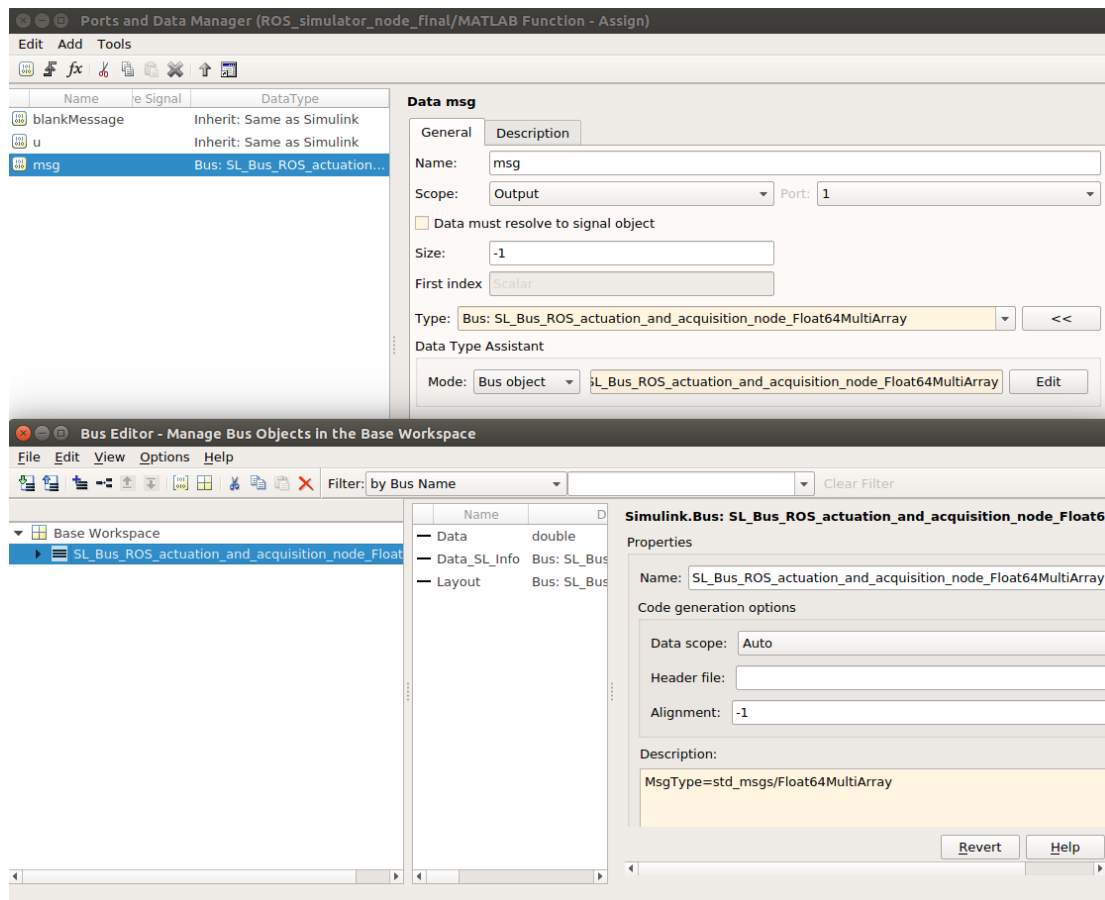


Figure 3.3. BUS correction.

3.2 Model Validation Experiment

The model validation experiment consists on testing the several models with the same input and initial conditions. The evolution of the different models will be recorded and the state variables will be plotted to evaluate the matching.

For instance, a proper input signal has been developed through trial and error techniques to obtain a complete and durable trajectory (Fig. 3.4).

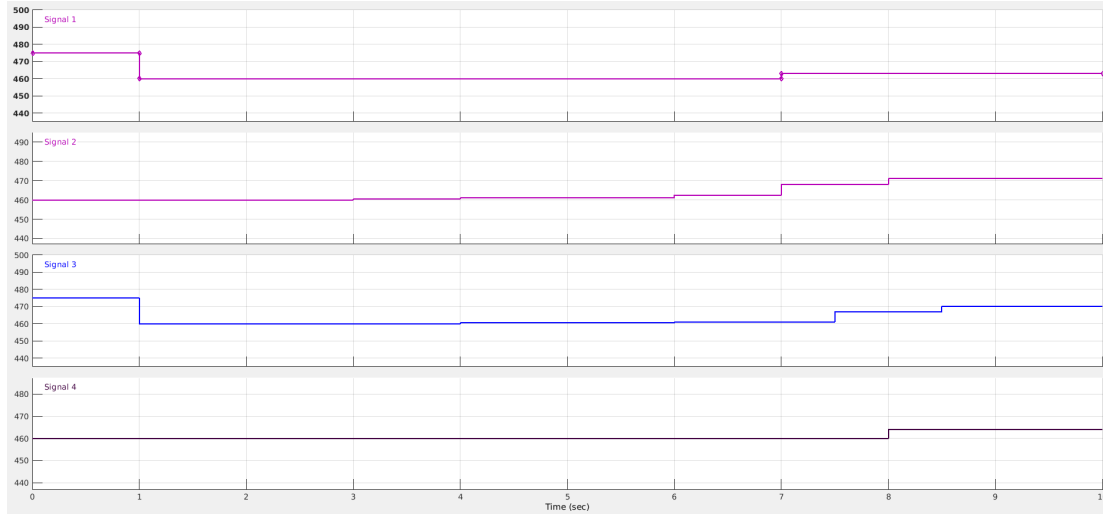


Figure 3.4. Input signal.

This signal is used for the simulink code that contains the mathematical and Simscape models (Fig. 3.6). For the Actuation and Acquisition node (Fig. 3.1) the signal is slightly modified by adding a zero input for the first second of the simulation, this allows the node to start properly and without cutting out any part of the initial input signal.

The simulink code used for this experiments (Fig. 3.6) contains the Euler-Lagrange model (Fig. 2.5) the Newton-Euler model (Fig. 2.4) and the Simscape model (Fig. 2.11) and it is able to collect and save in the MATLAB workspace the state variables of each model.

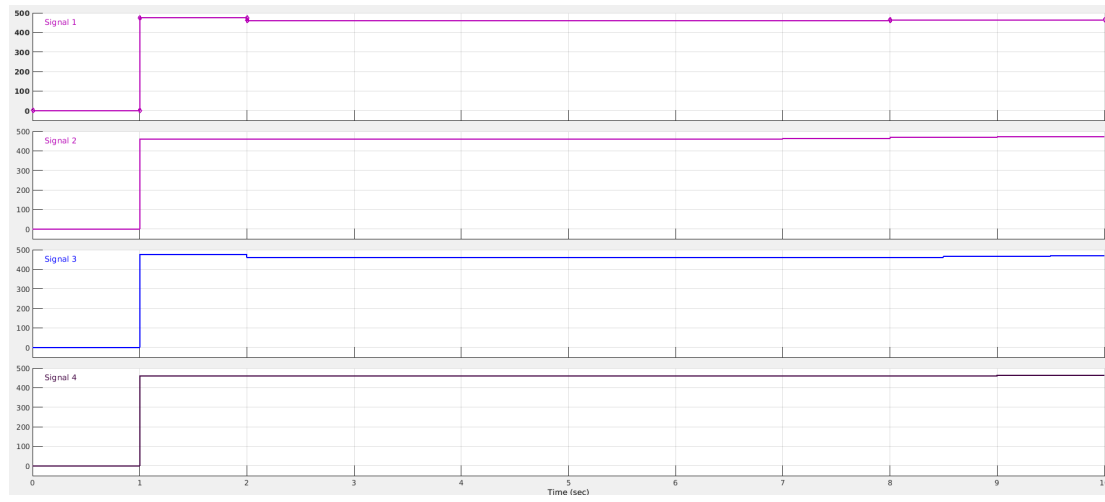


Figure 3.5. Input signal for Gazebo.

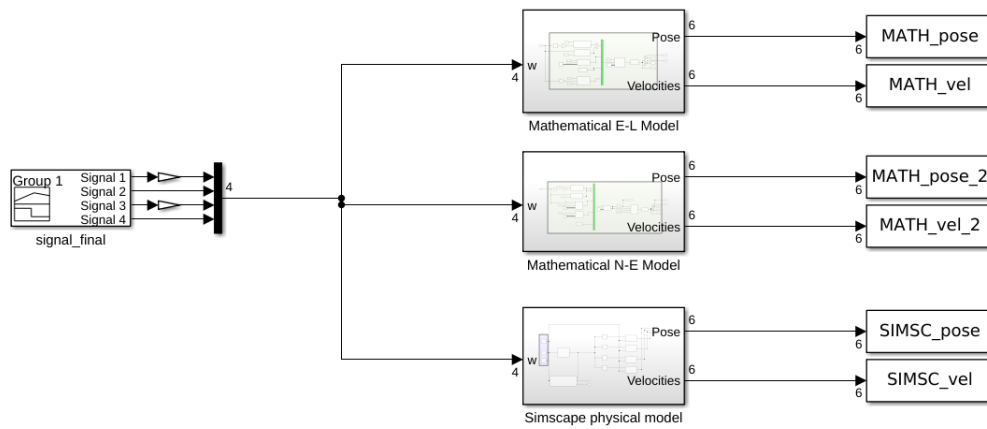


Figure 3.6. Model validation experiment.

3.3 Model Validation Results

The results of the model validation experiment are shown in Fig. 3.7,3.8,3.9,3.10. This images represent the evolution of the state variables ξ , η , $\dot{\xi}$ and $\dot{\eta}$ given the input defined in the previous section and the resting initial conditions: $\mathbf{q}(0) = [0,0,0,0,0,0]^T$ and $\dot{\mathbf{q}}(0) = [0,0,0,0,0,0]^T$.

From a first look it is possible to notice that all the models behave in a similar way, the variables evolve in the same direction and with similar derivatives. From the error section of each figure it is advatagious to check which, among the two mathematical models and the Simscape model, differs more with respect to the ROS model. In particular, the Euler-Lagrangian model shows higher absolute error that increase with time.

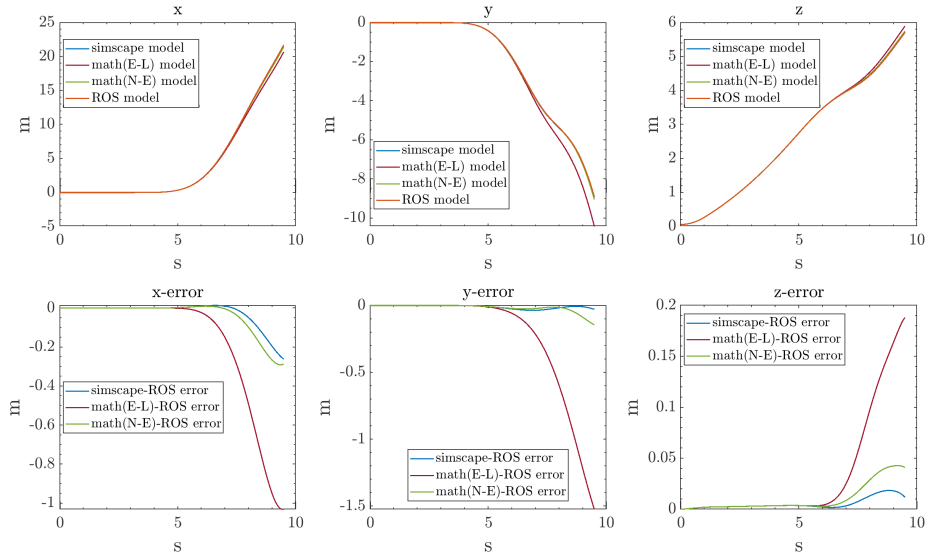


Figure 3.7. Model validation position.

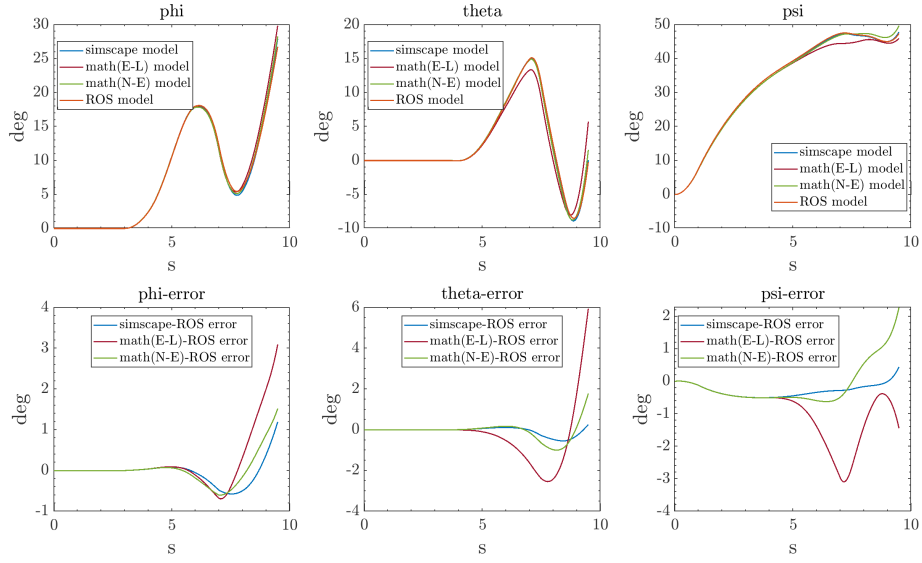


Figure 3.8. Model validation orientation.

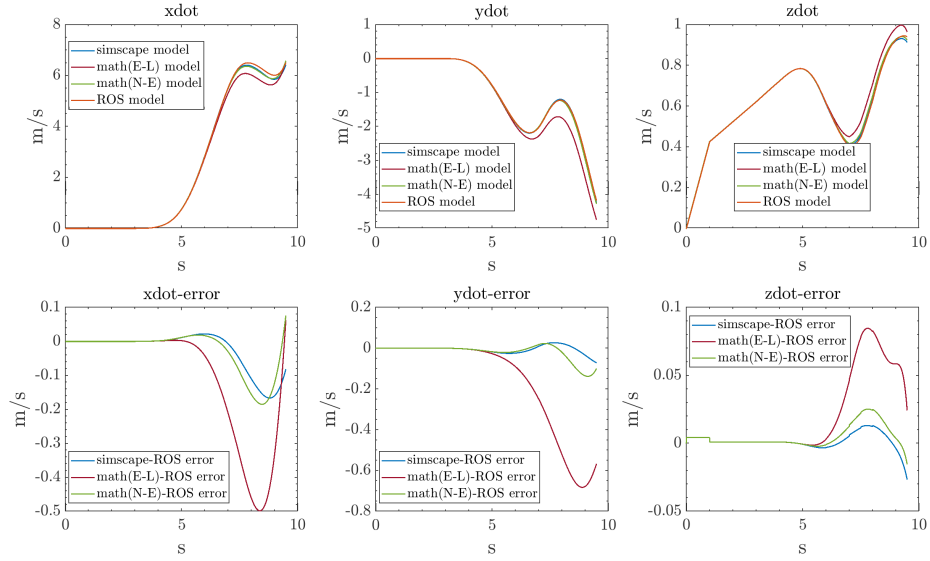


Figure 3.9. Model validation linear velocity.

In tables 3.1, 3.2, 3.3, 3.4, it is possible to check the maximum absolute error and the average percentage error between the Simscape model and the two mathematical models with respect to the ROS model.

From all these results it is possible to see that the Euler-Lagrangian model has higher errors.

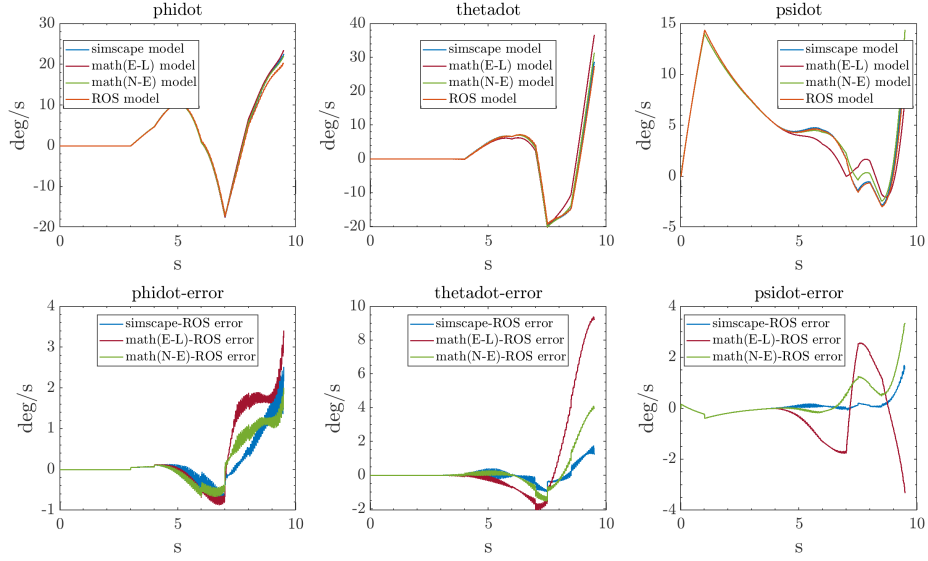


Figure 3.10. Model validation Euler derivatives.

	x error [m]	y error [m]	z error [m]
avg % simscape-ROS error	0.36758	0.72618	0.28902
avg % (E-L)-ROS error	1.6666	3.5448	0.49309
avg % (N-E)-ROS error	0.47002	0.78136	0.39181
max absolute ROS-ROS error	0.26122	0.036197	0.018514
max absolute (E-L)-ROS error	1.5941	2.0344	0.087062
max absolute (N-E)-ROS error	0.29041	0.14408	0.042843

Table 3.1. Position error.

	phi error [deg]	theta error [deg]	psi error [deg]
avg % simscape-ROS error	1.2667	0.70159	0.98156
avg % (E-L)-ROS error	2.0624	8.1665	3.4401
avg % (N-E)-ROS error	1.4002	1.1616	1.4527
max absolute simscape-ROS error	1.1937	0.54734	0.50449
max absolute (E-L)-ROS error	2.4131	5.9439	4.5597
max absolute (N-E)-ROS error	1.5207	1.7777	2.2715

Table 3.2. Orientation error.

	$\dot{\mathbf{x}}$ error [m/s]	$\dot{\mathbf{y}}$ error [m/s]	$\dot{\mathbf{z}}$ error [m/s]
avg % simscape-ROS error	0.98509	1.2199	0.78222
avg % (E-L)-ROS error	3.9217	11.279	3.2932
avg % (N-E)-ROS error	1.0342	1.6055	1.1345
max absolute simscape-ROS error	0.1665	0.071286	0.026641
max absolute (E-L)-ROS error	1.108	1.4376	0.16735
max absolute (N-E)-ROS error	0.18495	0.13779	0.025199

Table 3.3. Linear velocity error.

	$\dot{\phi}$ error [deg/s]	$\dot{\theta}$ error [deg/s]	$\dot{\psi}$ error [deg/s]
avg % simscape-ROS error	1.8832	1.3059	3.447
avg % (E-L)-ROS error	3.1274	8.4773	29.885
avg % (N-E)-ROS error	2.1768	2.8223	17.593
max absolute simscape-ROS error	2.5222	1.7515	1.7024
max absolute (E-L)-ROS error	2.1146	12.477	5.397
max absolute (N-E)-ROS error	2.0068	4.1066	3.3592

Table 3.4. Euler angles derivatives error.

Chapter 4

Control

The main goal of this Chapter is developing on Simulink and deploying on ROS Gazebo different types of controllers and analysing their performances. These controllers will compute the angular velocity of the for rotors in order to impose a certain orientation and therefore position to the quadrotor.

In the first section, the problem of under actuated systems will be discussed and a particular solution for quadrotors is presented. This problem characterizes systems that have a smaller number of inputs with respect to the number of variables to be controlled. To solve this problem the links between orientation and position of quadrotor will be exploited to develop a working controller.

The solution for the underactuated problem consists on building a first block of the controller which just computes the missing reference and feeds the actual controller with the complete set of euler angles to be followed.

This means having a control structure divided in two loops (Fig. 4.1): an outer loop that runs slower which computes the complete reference and in mean time provides the commanded acceleration and an inner loop that runs faster that computes the second derivatives of the euler angles.

Later in the discussion the different controllers that are tested in the inner loop will be presented one by one. Each control law and its application will be described. The tuning of each controller is performed both in Simulink and in ROS Gazebo because the controllers work differently in the two environments. In Simulink the simulation time does not correspond to the real time while, on the other hand, ROS Gazebo tries to have a simulation time that follows the real time.

In conclusion, the results of the tracking experiment for each controllers will be analysed with respect to some Key Performance Indicator. The tracking experiment consists on following a spiral trajectory that rises upwards starting from a resting position on the ground.

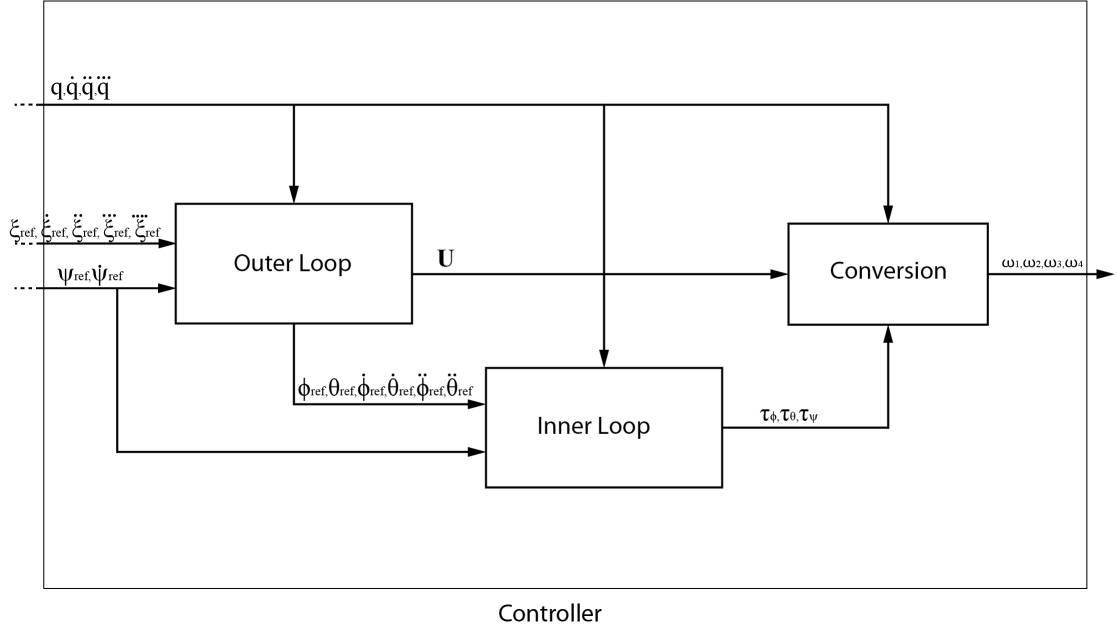


Figure 4.1. Controller structure.

4.1 Under Actuated Systems and Control Architectures

The first step in designing a controller for any system is to verify if the control inputs can steer the entire system's configuration. This means checking whether the mechanical system is fully actuated or under actuated.

As shown in [15] it is possible to check this property by analysing the second order differential equation of the system:

$$\ddot{\mathbf{q}}(t) = \mathbf{f}(t, \mathbf{q}(t), \dot{\mathbf{q}}(t)) + \mathbf{G}(\mathbf{q}(t))\mathbf{u}(t) \quad (4.1)$$

In particular, it is necessary to check the rank of $\mathbf{G}(\mathbf{q}(t))$. If this value is lower than the number of state variables the system is underactuated while, if it is equal to the number of state variable, the system is fully actuated. In the quadrotor case $\text{rank}(\mathbf{G}(\mathbf{q}(t))) = 4$ while the state variables are 6 and therefore the system is underactuated.

Then, another issue in designing the control architecture consists on analysing the controller output and eventually transforming such output in the real plant input. In this project, a position and an attitude controller will be deployed which means that the output will be the acceleration (or virtual control \mathbf{U}) and the torques $\tau_\phi, \tau_\theta, \tau_\psi$ as shown in Fig. 4.1, while the plant input are the angular velocities of the four rotors. the transformation from the controller output to the plant input is performed in the conversion block.

Finally, a partial feedback linearization is performed based on the mathematical

Euler-Lagrangian model. This method is used to improve the performance of the controllers.

4.1.1 Outer Loop

To solve the problem of the underactuated system it is possible to exploit the strong coupled properties of quadrotors as shown in [13] to obtain the reference for the uncontrollable variables. In this project it is decided to have as reference the position ξ , the yaw angle ψ and their derivatives.

The relationship between the attitude and the linear acceleration is built on the position error PID closed-loop simplified equations of the quadrotor. It is necessary to add that, with respect to the formulas obtained in [13], in this project the position closed loop equations contain also the integral of the position error:

$$\ddot{\xi}_e + K_d \dot{\xi}_e + K_p \xi + K_i \int \xi_e dt = 0 \quad (4.2)$$

where ξ_e is the position error between the reference and the actual position:

$$\xi_e = \xi_{ref} - \xi \quad (4.3)$$

The virtual control vector can be written as follow:

$$U = \ddot{\xi} = \ddot{\xi}_{ref} + K_d(\dot{\xi}_{ref} - \dot{\xi}) + K_p(\xi_{ref} - \xi) + K_i(\int \xi_{ref} dt - \int \xi dt) \quad (4.4)$$

from the simplified system equation it is possible to obtain the roll and pitch reference angle as function of the the virtual control variables and the reference yaw angle:

$$\phi_{ref} = \arcsin \left(\frac{U_1 \sin \psi_{ref} + U_2 \cos \psi_{ref}}{\sqrt{U_1^2 + U_2^2 + (U_3 + g)^2}} \right) \quad (4.5)$$

$$\theta_{ref} = \arcsin \left(\frac{U_1 \cos \psi_{ref} + U_2 \sin \psi_{ref}}{U_3 + g} \right) \quad (4.6)$$

The calculation of these reference angles and their derivatives $\dot{\phi}_{ref}, \dot{\theta}_{ref}, \ddot{\phi}_{ref}, \ddot{\theta}_{ref}$ is performed in each controller and represents the outer loop of the controller structure.

4.1.2 Conversion Block

The conversion block is the other part of the control structure that is common for every type of controller. This section transforms the output of the outer and inner loop in the angular velocities of the rotors.

For instance, it is necessary to transform the output of the outer loop $U = \ddot{\xi}$ in thrust. As shown in [13], the thrust force is computed as follow:

$$T = m[U_1(\sin \theta \cos \psi \cos \phi + \sin \psi \sin \phi) + U_2(\sin \theta \sin \psi \cos \phi - \cos \psi \sin \phi) + (U_3 + g) \cos \theta \cos \phi] \quad (4.7)$$

A simplified relationship between the thrust force, the torques and the angular velocities of the rotors can be derived from 2.14 and 2.15:

$$\begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} C_T & C_T & C_T & C_T \\ 0 & C_T & 0 & -C_T \\ -C_T & 0 & C_T & 0 \\ C_TC_M & -C_TC_M & C_TC_M & -C_TC_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \quad (4.8)$$

inverting this equation it is possible to obtain the square and then the final value of the angular velocities.

4.1.3 Feedback Linearization

A feedback linearization algorithm has been added in the PD and MRAC controllers to increase their performances. As shown by [11] in chapter 8.5.2 a new control $\mathbf{u}(t)$ can be designed taking in consideration the dynamic equations of the model.

Starting from eq. 2.34 a new control law can be designed:

$$\mathbf{u} = \mathbf{J}\mathbf{y} + \mathbf{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})\dot{\boldsymbol{\eta}} + \boldsymbol{\Gamma} \quad (4.9)$$

where \mathbf{y} represents a new input vector that is obtained from the inner control loop. The new structure of the controller is shown in fig. 4.2.

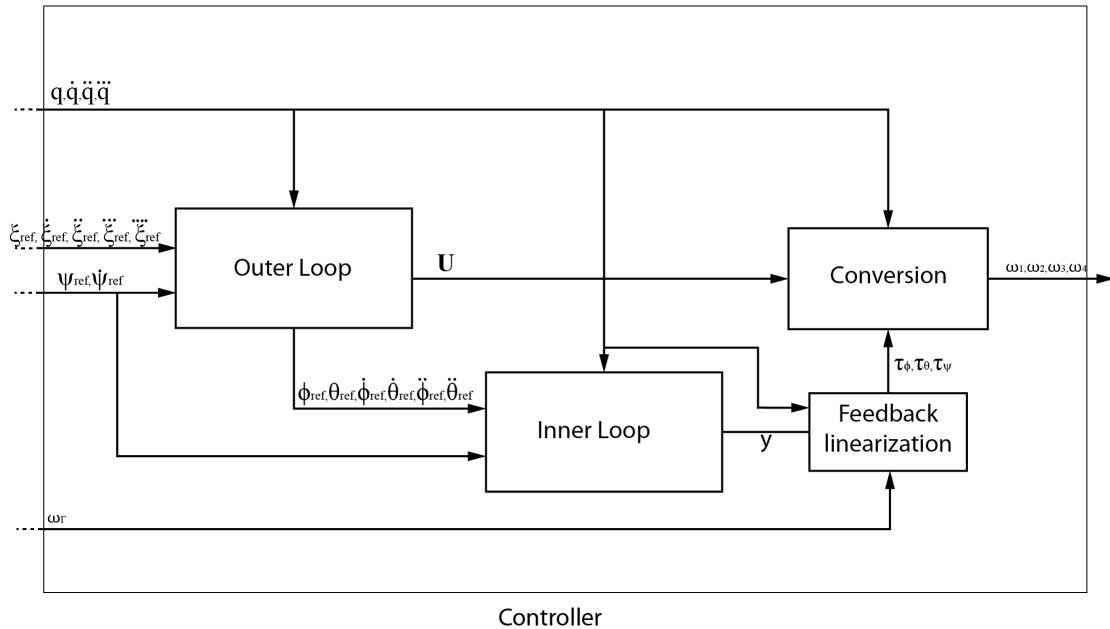


Figure 4.2. Controller plus feedback linearization structure.

4.2 Inner Loop

4.2.1 PD Controller

The first algorithm that is tested inside of the inner loop controller is a simple Proportional Derivative controller (PD). This controller computes the difference between the reference, its derivatives and the current state variables and then multiply it by some gains. This gains correspond to two square matrices, respectively one for the position error and one for the derivative error:

$$\begin{bmatrix} K_{\phi,P} & 0 & 0 \\ 0 & K_{\theta,P} & 0 \\ 0 & 0 & K_{\psi,P} \\ K_{\phi,D} & 0 & 0 \\ 0 & K_{\theta,D} & 0 \\ 0 & 0 & K_{\psi,D} \end{bmatrix} \quad (4.10)$$

The final commanded output is:

$$\begin{aligned} \tau_x &= \left(K_{\phi,P} (\phi_{ref} - \phi) + K_{\phi,D} (\dot{\phi}_{ref} - \dot{\phi}) \right) I_x \\ \tau_y &= \left(K_{\theta,P} (\theta_{ref} - \theta) + K_{\theta,D} (\dot{\theta}_{ref} - \dot{\theta}) \right) I_y \\ \tau_z &= \left(K_{\psi,P} (\psi_{ref} - \psi) + K_{\psi,D} (\dot{\psi}_{ref} - \dot{\psi}) \right) I_z \end{aligned} \quad (4.11)$$

4.2.2 Sliding Mode Control

The second algorithm to be tested inside the inner loop is a Sliding Mode Controller. The control law based on this algorithm is divided in two parts:

- the first one, also called discontinuous controller, brings the error vector toward a decision rules, called sliding surface.
- the second one, also called equivalent controller, once the error vector is restricted in the sliding surface, tracks the dynamics imposed by the equations describing the sliding surface.

The sliding surface for the three variables to be controlled can be written as:

$$\begin{aligned} s_\phi &= (\dot{\phi}_{ref} - \dot{\phi}) + \lambda_\phi (\phi_{ref} - \phi) \\ s_\theta &= (\dot{\theta}_{ref} - \dot{\theta}) + \lambda_\theta (\theta_{ref} - \theta) \\ s_\psi &= (\dot{\psi}_{ref} - \dot{\psi}) + \lambda_\psi (\psi_{ref} - \psi) \end{aligned} \quad (4.12)$$

where $\boldsymbol{\lambda} = [\lambda_\phi, \lambda_\theta, \lambda_\psi]$ is the first parameter to be tuned in the control algorithm. The general form of the control law $u(t)$ can be written as:

$$u(t) = u_{eq}(t) + u_d(t) \quad (4.13)$$

As shown by [16] the final control laws are:

$$\begin{aligned}\tau_\phi &= u_\phi(t) = \left[\lambda_\phi \left(\dot{\phi}_{ref} - \dot{\phi} \right) + \ddot{\phi}_{ref} - \dot{\theta} \dot{\psi} \left(\frac{I_y - I_z}{I_x} \right) + \frac{I_{z,p}}{I_x} \dot{\theta} \omega_\Gamma \right] \frac{I_x}{L} + K_{d,\phi} \frac{s_\phi}{|s_\phi| + \delta_\phi} \\ \tau_\theta &= u_\theta(t) = \left[\lambda_\theta \left(\dot{\theta}_{ref} - \dot{\theta} \right) + \ddot{\theta}_{ref} - \dot{\phi} \dot{\psi} \left(\frac{I_z - I_x}{I_y} \right) - \frac{I_{z,p}}{I_y} \dot{\theta} \omega_\Gamma \right] \frac{I_y}{L} + K_{d,\theta} \frac{s_\theta}{|s_\theta| + \delta_\theta} \\ \tau_\psi &= u_\psi(t) = \left[\lambda_\psi \left(\dot{\psi}_{ref} - \dot{\psi} \right) + \ddot{\psi}_{ref} - \dot{\phi} \dot{\theta} \left(\frac{I_x - I_y}{I_z} \right) \right] I_z + K_{d,\psi} \frac{s_\psi}{|s_\psi| + \delta_\psi}\end{aligned}\quad (4.14)$$

where ω_Γ is the sum of the angular velocities (eq. 2.24) and $\mathbf{K}_d = [K_{d,\phi}, K_{d,\theta}, K_{d,\psi}]$ is the second parameter to be tuned.

4.2.3 MRAC

The last algorithm to be tested as inner loop controller is a model reference adaptive controller. The objective of this controller is to design a MIMO state feedback adaptive control law such that the state system x globally uniformly asymptotically tracks the reference state x_{ref} of the reference model:

$$\dot{x}_{ref} = A_{ref}x_{ref} + B_{ref}r(t) \quad (4.15)$$

then, given any bounded command $r(t)$, the control input u needs to be chosen such that the state tracking error $e = x - x_{ref}$ globally uniformly asymptotically tends to zero:

$$\lim_{t \rightarrow \infty} \|x - x_{ref}\| = 0 \quad (4.16)$$

The control law to be applied has the form:

$$u(t) = \hat{K}_x^T(t)x(t) + \hat{K}_r^T(t)r(t) \quad (4.17)$$

where \hat{K}_x^T and \hat{K}_r^T are the estimates of the ideal unknown gain matrices K_x^T and K_r^T . The adaptive laws to select the gains are:

$$\begin{aligned}\dot{\hat{K}}_x &= \Phi_x + S^T y_e x^T \beta_x \\ \dot{\hat{K}}_r &= \Phi_r + S^T y_e r^T \beta_r\end{aligned}\quad (4.18)$$

and

$$\begin{aligned}\dot{\Phi}_x &= S^T y_e x^T \alpha_x \\ \dot{\Phi}_r &= S^T y_e r^T \alpha_r\end{aligned}\quad (4.19)$$

where S is an invertible matrix such that:

$$P_\Phi = \hat{\Phi}_r S > 0 \quad (4.20)$$

and y_e is:

$$y_e = B_{ref}^T P_e \quad (4.21)$$

and P_e is the solution of the equation:

$$P_e A_{ref} + A_{ref}^T P_e = -Q \quad (4.22)$$

where Q is a strictly positive matrix. The term $\alpha_x, \alpha_r, \beta_x$ and β_r are strictly positive diagonal matrices that define the variation of the adaptive gains in time. This matrices represents the parameters to be tuned in order to prepare the algorithm for the experiment.

4.3 Control Implementation and Results

The different controllers are developed in Simulink and then deployed on the ROS workspace to be tested in Gazebo.

The first part of the experiment consists on building the entire system in Simulink as shown in fig. 4.3. Here it is possible to notice the reference block, the controller block, the plant and the scope section where the results are plotted. The results of such simulation can be also collected in the MATLAB workspace and analysed later.

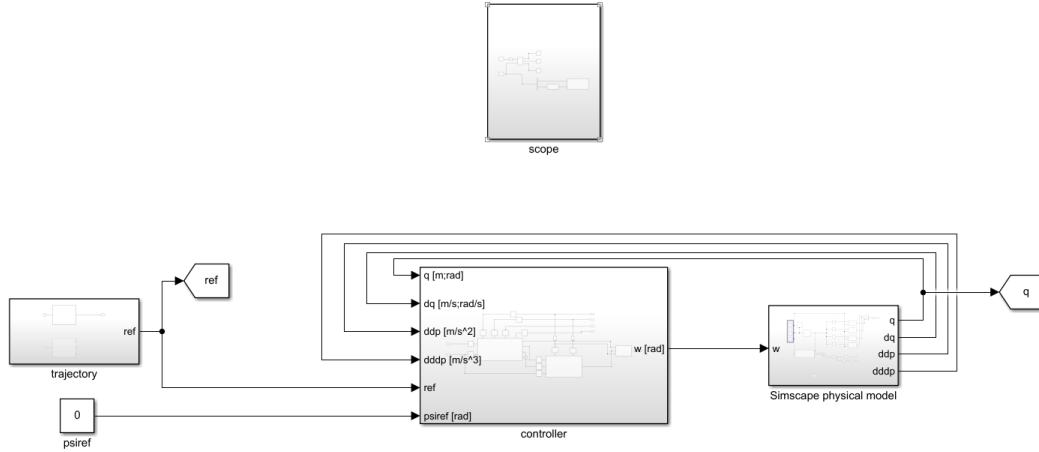


Figure 4.3. Simulink set up.

The second part of the experiment regards the modification of the starting Simulink set up in order to prepare it for the ROS simulation. As shown in fig. 4.4, the only parts left from the initial Simulink are a slightly modified reference block and the controller while the plant disappear because it is already present as independent drone in Gazebo. The new section of this Simulink regards the communication section with all the blocks regarding the message initialization, the topic connection and the data collection where the important variables are saved in a *.mat* file, this method is the same used in the actuation and acquisition block (fig. 3.1).

4.3.1 Reference

The reference signal used for this experiment is composed by the coordinates, the yaw angle and their derivatives up to the fourth time derivative. In particular, a

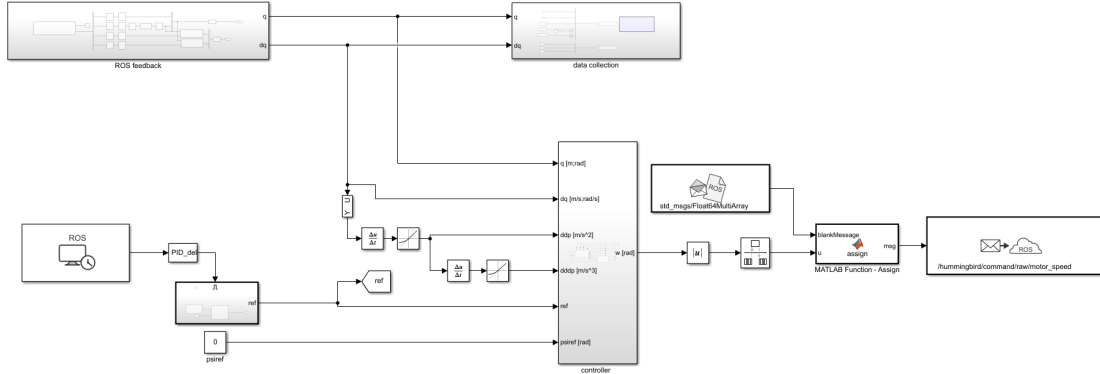


Figure 4.4. Simulink set up for ROS deployment.

spiral rising trajectory is defined with the mathematical law:

$$\begin{aligned} x &= k_{x,1} \cos(k_{x,2}t) - x_0 \\ y &= k_{y,1} \sin(k_{y,2}t) - y_0 \\ z &= k_{z,1}t - z_0 \\ \psi &= 0 \end{aligned} \quad (4.23)$$

where $k_{x,1}, k_{x,2}, k_{y,1}, k_{y,2}$ are constants that define the amplitude and the period of the spiral while $k_{z,1}$ defines the rising velocity of the drone. In this experiment the radius of the spiral ($k_{x,1}, k_{y,1}$) was 2 meters, the frequency ($k_{x,2}, k_{y,2}$) was 0.5 Hz and the rising velocity ($k_{z,1}$) was 1 meter per second. The initial condition x_0, y_0, z_0 are needed to center the starting point of the trajectory in the inertial reference frame coordinates (0,0,0).

4.3.2 Tuning

In order to compare the several inner loop controller performances, the outer loop parameters are the same for all the controllers. With reference to 4.4 the parameters values are reported in tab 4.1. The PD controller is implemented both

Parameter	x	y	z
K_p	5	5	5
K_d	1	1	2
K_i	0.2	0.2	0.2

Table 4.1. Outer loop parameters.

with and without feedback linearization and the parameters referring to eq. 4.10 are reported in tab. 4.2, 4.3.

The Sliding Model Controller parameters (eq. 4.14) are reported in tab. 4.4.

Parameter	ϕ	θ	ψ
K_p	200	200	200
K_d	20	20	20

Table 4.2. PD parameters.

Parameter	ϕ	θ	ψ
K_p	350	350	350
K_d	35	35	35

Table 4.3. PD with Feedback Linearization parameters.

Parameter	ϕ	θ	ψ
K	3	3	3
λ	10	10	10
δ	0.1	0.1	0.1

Table 4.4. SMC parameters.

In conclusion, the MRAC parameters related to eq. 4.18 4.19 are reported in tab. 4.5.

Parameter	ϕ	θ	ψ	$\dot{\phi}$	$\dot{\theta}$	$\dot{\psi}$
α_x	200000	20000	200000	8800	8800	8800
β_x	20000	2000	20000	880	880	880

Table 4.5. MRAC parameters.

And the model reference matrices are:

$$\begin{aligned}
 A_{ref} &= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -k_{ref} & 0 & 0 & -b_{ref} & 0 & 0 \\ 0 & -k_{ref} & 0 & 0 & -b_{ref} & 0 \\ 0 & 0 & -k_{ref} & 0 & 0 & -b_{ref} \end{bmatrix} \\
 B_{ref} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
 \end{aligned} \tag{4.24}$$

Here, k_{ref} and b_{ref} are two parameters related to the settling time of the reference

system. In this case a settling time of 1 second is chosen and therefore:

$$\begin{aligned}
 t_s &= 1 \\
 \lambda_1 &= 4.6/t_s = 4.6 \\
 \lambda_2 &= 5\lambda_1 = 23 \\
 k_{ref} &= \lambda_1\lambda_2 = 105.8 \\
 b_{ref} &= \lambda_1 + \lambda_2 = 27.6
 \end{aligned} \tag{4.25}$$

4.3.3 Results

In this section the results of the trajectory tracking experiment are presented. In fig. 4.5 it is possible to see the the several trajectories of each drone controlled by a different inner loop algorithm.

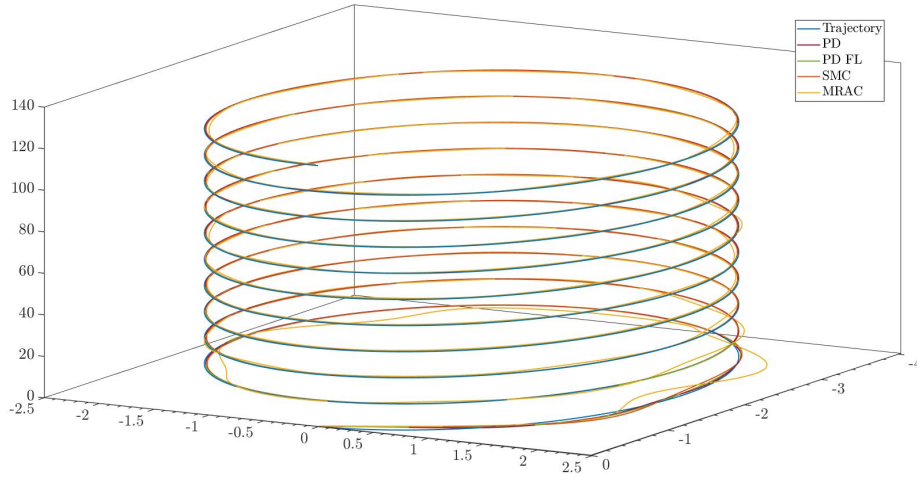


Figure 4.5. Trajectories.

In particular it is possible to notice how the model reference adaptive control algorithm needs some time to adjust the gains in order to have an acceptable tracking; while the other controllers, that have been previously tuned, immediately have good performances. In order to better evaluate the algorithm performances some Key Performance indicator are introduced and then applied to the experimental outputs.

Root Mean Square Error (RMSE) and Maximum Error (ME) are used to analyse the position error, the velocity error, the orientation error and the euler angle derivatives error. The RMSE of a vector \mathbf{x} with respect to the reference vector \mathbf{x}^{ref} (in this case the trajectory to be followed) is computed in the following way:

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (x_i - x_i^{ref})^2}{n}} \tag{4.26}$$

where n is the number of samples in the vector and x_i is the i -th sample of the vector \mathbf{x} .

On the other hand, the ME is:

$$ME = \max(\mathbf{x} - \mathbf{x}^{ref}) \quad (4.27)$$

These KPI's have been computed for the total experiment duration of 120 seconds and for each cycle that the quadrotor performs in order to notice the evolution of the errors in time. The results of each simulation is reported in the tables 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13.

From the error analysis it can be stated that the Sliding Mode Controllers has better performances with lower total and final RMSE and ME. It is also possible to notice how the feedback linearization helps the PD controller improving its performances among all the KPI's.

PD			total	cycle 1	cycle 2	cycle 3	cycle 4	
position RMSE			0.014526	0.040129	0.022608	0.022539	0.022525	
velocity RMSE			0.019332	0.060681	0.016225	0.016235	0.016211	
orientation RMSE			0.0066593	0.020242	0.0076358	0.0074925	0.0078571	
euler angle der RMSE			0.015739	0.046876	0.019615	0.019627	0.020047	
			cycle 5	cycle 6	cycle 7	cycle 8	cycle 9	cycle 10
			0.02245	0.022425	0.022546	0.023183	0.022443	0.022526
			0.016079	0.016063	0.016242	0.016571	0.016195	0.016144
			0.0075539	0.0075456	0.0076772	0.0079626	0.0075342	0.0075852
			0.019468	0.019567	0.019738	0.019949	0.019405	0.019641

Table 4.6. PD RMSE results.

PD with Feedback Lin.	total	cycle 1	cycle 2	cycle 3	cycle 4	
position RMSE	0.014148	0.039872	0.021431	0.021388	0.021416	
velocity RMSE	0.018945	0.059507	0.01535	0.015394	0.015406	
orientation RMSE	0.0050179	0.014776	0.0065389	0.0066208	0.0064414	
euler angle der RMSE	0.014243	0.040852	0.019563	0.0199	0.019567	
	cycle 5	cycle 6	cycle 7	cycle 8	cycle 9	cycle 10
	0.021307	0.021356	0.021408	0.021862	0.021205	0.021391
	0.015252	0.015318	0.015336	0.015758	0.015257	0.015384
	0.0064451	0.0065109	0.0065948	0.0065772	0.0063928	0.0065018
	0.019548	0.01962	0.019993	0.019854	0.019558	0.019599

Table 4.7. PD with Feedback Linearization RMSE results .

SMC			total	cycle 1	cycle 2	cycle 3	cycle 4	
position RMSE			0.01507	0.043949	0.020868	0.020784	0.020768	
velocity RMSE			0.02045	0.064332	0.014902	0.014863	0.01489	
orientation RMSE			0.0053053	0.015636	0.0067994	0.0068058	0.0067898	
euler angle der RMSE			0.011244	0.029296	0.017535	0.017624	0.017821	
			cycle 5	cycle 6	cycle 7	cycle 8	cycle 9	cycle 10
			0.020727	0.020747	0.020754	0.021234	0.0208	0.020747
			0.014851	0.014881	0.014888	0.015242	0.01482	0.01485
			0.0068562	0.0068554	0.0068278	0.0069106	0.006786	0.0068657
			0.017631	0.017981	0.017723	0.017681	0.01758	0.017736

Table 4.8. SMC RMSE results.

MRAC with Feedback Lin.	total	cycle 1	cycle 2	cycle 3	cycle 4	
position RMSE	0.021482	0.064347	0.028369	0.027765	0.026211	
velocity RMSE	0.027785	0.084858	0.043679	0.032201	0.025048	
orientation RMSE	0.013406	0.040904	0.019016	0.015758	0.013174	
euler angle der RMSE	0.017719	0.050412	0.032893	0.028473	0.022771	
	cycle 5	cycle 6	cycle 7	cycle 8	cycle 9	cycle 10
	0.026277	0.026641	0.025803	0.024981	0.025351	0.020747
	0.028432	0.034806	0.029371	0.024599	0.025533	0.01485
	0.014105	0.016309	0.014853	0.012524	0.013223	0.0068657
	0.02297	0.028515	0.02623	0.02133	0.02066	0.017736

Table 4.9. MRAC with Feedback Linearization RMSE results.

PD	total	cycle 1	cycle 2	cycle 3	cycle 4	
position ME	0.4115	0.4115	0.071815	0.070703	0.071621	
velocity ME	1	1	0.037581	0.038871	0.040369	
orientation ME	0.11862	0.11862	0.014859	0.015146	0.015124	
euler angle der ME	0.6073	0.6073	0.063945	0.061388	0.073881	
	cycle 5	cycle 6	cycle 7	cycle 8	cycle 9	cycle 10
	0.069065	0.071284	0.07067	0.070344	0.067516	0.069685
	0.036291	0.037118	0.038231	0.037723	0.041247	0.036582
	0.014749	0.015284	0.014844	0.01535	0.015705	0.014618
	0.062379	0.066042	0.066741	0.067703	0.068629	0.06465

Table 4.10. PD ME results.

PD with Feedback Lin.	total	cycle 1	cycle 2	cycle 3	cycle 4	
position ME	0.37063	0.37063	0.068876	0.066537	0.067032	
velocity ME	1	1	0.03421	0.032785	0.034378	
orientation ME	0.10373	0.10373	0.012556	0.01335	0.012698	
euler angle der ME	0.69817	0.69817	0.072471	0.084467	0.0703	
	cycle 5	cycle 6	cycle 7	cycle 8	cycle 9	cycle 10
	0.064625	0.064569	0.063538	0.067499	0.066649	0.062762
	0.034251	0.032373	0.033311	0.033136	0.036044	0.03352
	0.01235	0.012045	0.01316	0.012224	0.013597	0.012867
	0.069044	0.068861	0.090846	0.06616	0.068216	0.065665

Table 4.11. PD with Feedback Linearization ME results.

SMC	total	cycle 1	cycle 2	cycle 3	cycle 4
position ME	0.47686	0.47686	0.06538	0.063109	0.064352
velocity ME	1	1	0.03326	0.03142	0.031806
orientation ME	0.1149	0.1149	0.0084517	0.0086417	0.0067636
euler angle der ME	0.30159	0.30159	0.060315	0.057727	0.057883
cycle 5	cycle 6	cycle 7	cycle 8	cycle 9	cycle 10
0.063134	0.064264	0.062316	0.06016	0.062351	0.062791
0.031412	0.032474	0.031401	0.031238	0.033048	0.032509
0.0075412	0.0072998	0.0070483	0.0070208	0.00789	0.0081154
0.065214	0.065007	0.059352	0.053838	0.061572	0.059217

Table 4.12. SMC ME results.

MRAC with Feedback Lin.	total	cycle 1	cycle 2	cycle 3	cycle 4	
position ME	0.63163	0.63163	0.12588	0.114	0.0939	
velocity ME	1	1	0.25221	0.16559	0.091056	
orientation ME	0.24559	0.24559	0.059506	0.036555	0.026301	
euler angle der ME	0.63082	0.63082	0.13519	0.11975	0.084093	
	cycle 5	cycle 6	cycle 7	cycle 8	cycle 9	cycle 10
	0.096982	0.095591	0.09083	0.085546	0.094834	0.10368
	0.11226	0.19205	0.12063	0.079317	0.093384	0.11352
	0.026313	0.045091	0.033602	0.020847	0.024212	0.031428
	0.082381	0.14432	0.10074	0.06442	0.063881	0.076721

Table 4.13. MRAC with Feedback Linearization ME results.

Chapter 5

Conclusion

From the results reported in chapter 4, it can be inferred that the simulation environment is adequate for testing algorithms and running several type of simulations. The real strength of this project is the possibility of designing different controllers in a friendly environment such as MATLAB Simulink and then deploying such algorithms in a more realistic simulator like ROS Gazebo. Thanks to this work it is possible to study the performance of already developed controllers and also future algorithms in an application that will become crucial in the day to day life since drones will become always more important in many fields and their control will be on the most important aspect to be studied and improved. In this project a simple experiment has been run to test the environment capability. In the future it will be possible to exploit this work and all the functionality of ROS Gazebo to run more interesting tests. For example it will be possible to run this simulation in more realistic maps like in fig. 5.1 and maybe give more articulated trajectories to the drone to follow. Exploiting the modularity and ROS Gazebo it will be possible to spawn a vehicle in Gazebo that the drone may follow and guard.

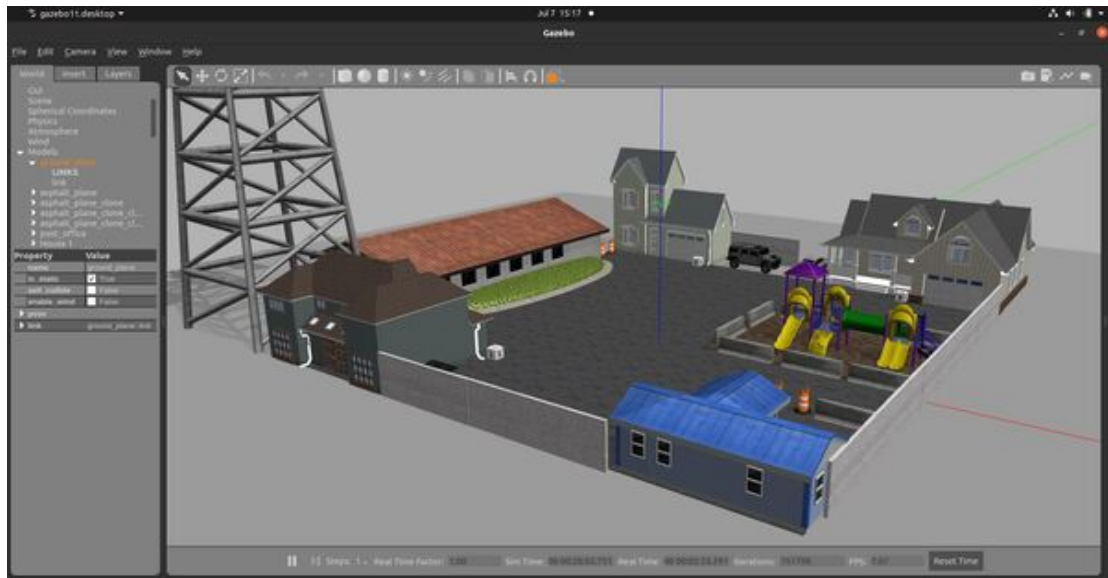


Figure 5.1. Gazebo Map.

Bibliography

- [1] Sullivan and Frost. Commercial drone market to hit 2.44 million units by 2023, says frost and sullivan. <https://www.prnewswire.com/news-releases/commercial-drone-market-to-hit-2-44-million-units-by-2023--says-frost--sullivan.html>, Apr 2020.
- [2] Moses Bangura, Robert Mahony, et al. Nonlinear dynamic modeling for high performance control of a quadrotor. 2012.
- [3] Robert Mahony, Vijay Kumar, and Peter Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics and Automation magazine*, 19(3):20–32, 2012.
- [4] V Solovyev Viktor, I Finaev Valery, A Zargaryan Yuri, O Shapovalov Igor, and A Beloglazov Denis. Simuation of wind effect on quadrotor flight. *ARPJ Journal of Engineering and Applied Sciences*, 10(4):1535–1538, 2006.
- [5] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Rotors—a modular gazebo mav simulator framework. In *Robot operating system (ROS)*, pages 595–625. Springer, 2016.
- [6] Documentation. <http://wiki.ros.org/>.
- [7] Simulate before you build. <https://gazebo-sim.org/home>.
- [8] What is matlab? <https://mathworks.com/discovery/what-is-matlab.html>.
- [9] What is simulink? <https://mathworks.com/products/simulink.html>.
- [10] What is simscape? <https://mathworks.com/products/simscape.html>.
- [11] Bruno Siciliano, Oussama Khatib, and Torsten Kröger. *Springer handbook of robotics*, volume 200. Springer, 2008.
- [12] Thomas S Alderete. Simulator aero model implementation. *NASA Ames Research Center, Moffett Field, California*, page 21, 1995.
- [13] Zongyu Zuo. Trajectory tracking control design with command-filtered compensation for a quadrotor. *IET control theory & applications*, 4(11):2343–2355, 2010.
- [14] Guilherme V Raffo, Manuel G Ortega, and Francisco R Rubio. An integral predictive/nonlinear h ∞ control structure for a quadrotor helicopter. *Automatica*, 46(1):29–39, 2010.

- [15] Andrea Laffitto, Robert B Anderson, and Keyvan Mohammadi. An introduction to nonlinear robust control for unmanned quadrotor aircraft: how to design control algorithms for quadrotors using sliding mode control and adaptive control techniques [focus on education]. *IEEE Control Systems Magazine*, 38(3):102–121, 2018.
- [16] Marco Herrera, William Chamorro, Alejandro P Gómez, and Oscar Camacho. Sliding mode control: An approach to control a quadrotor. In *2015 Asia-Pacific conference on computer aided system engineering*, pages 314–319. IEEE, 2015.