



**Politecnico
di Torino**

Politecnico di Torino

Master's Degree in Aerospace Engineering
Academic Year 2021/2022
December 2021

Design of a small quadrotor UAV and modeling of an MPC-based simulator

Supervisors:

Prof. Elisa CAPELLO
Dr. Davide PASSONI

Candidate:

Tommaso ZANATTA

The design is not just what it looks like and feels like.

The design is how it works.

- Steve Jobs

Acknowledgments

Many people have influenced me personally and academically over the years that culminate in this thesis. I cannot express enough thanks to everyone who has accompanied me through this beautiful chapter of my life. Firstly, I would like to express my deep and sincere gratitude to my supervisor Tenure-Track Assistant Professor of Flight Mechanics at Polytechnic University of Turin, Department of Mechanical and Aerospace Engineering, Dr. Elisa Capello for encouraging me during this project and always being there to help me with any problems, without her support I couldn't have succeeded in completing this project. I offer my sincere appreciation for the internship opportunity provided by ALTEN Italia s.p.a. I am deeply obliged to Dr. Renato Sala for giving me an opportunity to work on this stimulating project. My completion of this thesis could not have been accomplished without the support and patience of my co-advisor Dr. Davide Passoni. Also, I would like to mention Dr. Davide Carminati and Dr. Iris David Du Mutel de Pierrepont Franzetti for their useful advice throughout the project and for their kindness.

Any attempt at any level can't be satisfactorily completed without the support and guidance of my family. Special thanks to my mom Michela and my dad Nello for their love and support. Without them, this day would not have been possible. Thanks to my sister Elisa who have been my side throughout these tough months of my life always cheering me up. I could not have done any of this without you. Thank you all.

Abstract

The concept of Autonomous Vehicles (AVs), such as self-driving cars and Unmanned Aerial Vehicles (UAVs), has been widely explored in recent years. UAVs, or commonly known as pilotless aircrafts, are widely used in military and security applications. As an example, in 1946 to 1948, United States Air Force and Navy used unmanned B-17 and F6Fs to fly into nuclear clouds within minute after bomb detonation, to collect radioactive samples. Moreover, availability and accessibility of advanced autonomous technology, has encourage the development of civilian applications.

Quadrotors are Vertical Take-Off and Landing (VTOL) aerial vehicles with a four motor-rotor assembly for generating lift and controllability. In recent years, ease of design and simple dynamics have increased their use in aerial robotics research. In the last decades efforts were made to improve autopilot performance developing advanced Guidance, Navigation and Control (GNC) algorithms. Despite numerous advantages, quadrotors may suffer various problems such as gust disturbance, mechanical vibration, and actuator failures during flight. All these problems reduce the flight performance and bring difficulties in the controller design. An effective controller must ensure smooth and collision free flight in the complex surrounding environment, considering aerodynamic drag and moments. In literatures, the multirotor control problem has been addressed using several control approaches, like Proportional-Integrative-Derivative (PID), Linear-Quadratic-Regulator (LQR) and H-infinity for linear control system. In this work a linear Model-Predictive-Control (MPC) approach is applied.

The aims of this thesis are to develop, implement and test, both hardware (HW) and software (SW) system of a multirotor UAV for indoor applications. There are many quadrotors that are commercially available nowadays, however, off-the-shelf quadrotors usually lack the ability to be reprogrammed and are unsuitable for use as research platforms. Therefore, an HW selection has been carried out in the first part of the project, starting from Commercial-Off-the-Shelf (COTS) components. The motors, the propellers, the battery, the Electronic Speed Control (ESCs), the frame and the avionics systems (such as autopilot, flight computer and sensors) are chosen concerning compatibility constraints, good endurance, and low weight. The quadrotor autopilot is based on 'Pixhawk 4 mini' and it is able to fly indoor using stationary ultrasonic beacons ('Marvelmind Set HW v4.9') as indoor positioning system. Using personalized 3D printed components, all the selected items were integrated and assembled in an efficient, working configuration. The first part of the project ended with the verification and validation of communication between HW's components and flying tests.

The second part of the project concerns the SW development and testing. Due to the complexity and the computational need of the control algorithm, most of the commercial autopilots, are based on Proportional Derivative Integrative (PID) controller. This thesis proposed a Receding Horizon Control (RHC). RHC, also known as Model Predictive Control (MPC), is an optimal control-based strategy that uses a plant model to predict the effect of an input profile on the evolving state of the plant. At each sample time, an optimal control problem is solved, and its optimal input profile is implemented until another plant measurement becomes available. The updated information is used to formulate and solve a new optimization problem, and the process is repeated. A practical disadvantage is its computational cost, so the MPC applications are usually limited to linear process with relatively slow dynamics such as chemical engineering systems.

However, with the raising of more powerful computers it is now being used in system with faster dynamics. Starting from the equations of motions of an aircraft, assuming some simplifications, the nonlinear mathematical model of the plant is derived. Quadrotor dynamics can be split into two categories: slow dynamics, regarding the position; and fast dynamics, regarding the attitude and altitude. Thanks to this classification, an MPC-based cascade controller is developed. The controller is structured into two loops: an outer loop related to UAVs slow dynamics (controlled by a PD controller) and an inner loop related to UAVs fast dynamic (controlled by MPC controller). The controller is tested tracking different reference trajectories ranging from simple ones to complex waypoints-following trajectories with increasing difficulty in terms of changes in the states. In these studies, a linearized model is adopted, and the Receding Horizon method is applied to generate the optimal control sequence. The MPC parameters (prediction horizons, weighting matrices, and discretization parameters) are selected by trial-and-error approach. Several simulations are conducted to examine and evaluate the performance of the proposed control approach using MATLAB and Simulink environment. Simulation results show that this kind of control is highly effective to track different types of given reference trajectory. The performances of the controller have been further tested on a virtual environment, using Unreal Engine as plant to have more realistic results representations.

Future work involves Hardware-in-the-Loop (HIL) and Processor-in-the-Loop testing. Once such processes are concluded, a custom MAVLink-based flight controller (in C language) will be developed, deploying it into the real quadrotor flight computer and testing the algorithm's performance with experimental flight tests.

Contents

Introduction.....	14
1.1 – UAVs Historical Survey and Classification.....	15
1.2 – Motivations of the Thesis	18
1.3 – Thesis Outline	22
Multicopter Mathematical Model	24
2.1 – Quadcopter Working Principle	24
2.1.1 – Reference Frames	25
2.1.2 – Euler Angles and Quaternions	26
2.2 – Quadcopter Mathematical Model	29
2.3 – Quadcopter Dynamics: Forces and Moments	29
2.4 – State-Space Model Representation.....	30
2.5 – Quadcopter Linearized Model	31
2.5.1 – Equilibrium Point and Linearization	32
2.5.2 – Controllability and Observability	33
Quadcopter Design	34
3.1 – Quadcopter Anatomy	34
3.1.1 – Airframe: Holybro X500 Frame.....	35
3.1.2 – Control Module: Pixhawk 4 Mini and PX4	35
3.1.3 – Propulsion Module: Motors and Propellers	38
3.1.4 – Propulsion Module: Electronic Speed Control ESCs	40
3.1.5 – Energy Module: LiPo Battery	41
3.1.6 – Sensors and Indoor Positioning System.....	42
3.2 – Quadcopter Assembly: Hardware Architecture.....	44
3.2.1 – SolidWorks CAD Model and 3D Printed Components	46
3.2.2 – Prototype Quadcopter	48
Guidance, Navigation and Control Algorithm	49
4.1 – Guidance Algorithm.....	49
4.2 – Navigation Algorithm.....	51
4.3 – Control Algorithm	52
4.3.1 – Proportional Integral Derivative – PID Controller	52

4.3.2 – Receding Horizon Controller – RHC Controller	54
Simulation Results	63
5.1 – Simulation in MATLAB and Simulink Environment.....	63
5.1.1 – Model-In the Loop Simulation – Path Following	65
5.1.2 – Model-In-the-Loop Simulation – Obstacle Avoidance.....	68
5.1.3 – Path Following and Obstacle Avoidance: Results Discussion	69
Unreal Engine® Simulations and AirSim Simulator.....	77
6.1 – Simulink 3D Animation and UAV Toolbox	77
6.1.1 – UAV Toolbox Interface for Unreal Engine.....	78
6.1.2 – Unreal Engine 3D Results Visualization	79
6.2 – AirSim Simulator	81
6.2.1 – Introduction to AirSim	81
6.2.2 – AirSim Flight Controller.....	84
Conclusions.....	87

List of figures

Figure 1: flying fixed wing UAV MQ-9A Predator B.	15
Figure 2: flying fixed wing Northrop Grumman RQ-4 Global Hawk.	15
Figure 3: SITARIA UAV fixed-wing unmanned electric aircraft designed by UAVOS.	16
Figure 4: MQ-8 Fire Scout single-rotor UAV developed by Northrop Grumman for USAF.	17
Figure 5: Freefly ALTA 8 Multirotor Camera Drone design by Freefly Systems.	17
Figure 6: Joby S4 fixed-wing hybrid e-VTOL developed by Joby Aviation.	17
Figure 7: octocopter performs power-line inspection.	18
Figure 8: picture provided by new service approach for power-line inspections provided by Siemens SIEAERO.	18
Figure 9: agricultural drones being used to spray pesticide on crops in a village in Poyang, central China's Jiangxi province.	19
Figure 10: false-colour difference vegetation index from near-infrared images collected by an Agribotix drone.	19
Figure 11: photo captured by UAVs high resolution camera during search and rescue operation, after the devastating earthquake and tsunami in Banda Aceh, Indonesia on 26 December 2004.	19
Figure 12: Wing company launches its first drone delivery service in Australia (2019).	20
Figure 13: photo of Wing drone from Google delivers goods during COVID-19 pandemic.	20
Figure 14: quadrotor cross 'X' configuration.	24
Figure 15: quadrotor plus '+' configuration.	24
Figure 16: quadrotor upward movement.	25
Figure 17: quadrotor roll movement.	25
Figure 18: quadrotor pitch movement.	25
Figure 19: quadrotor yaw movement.	25
Figure 20: illustration of used reference frames: the Body reference frame in the top left and the NED inertial reference frame in the bottom right.	26
Figure 21: photo of kit Holybro X500 frame contents.	35
Figure 22: representation of Holybro battery mounting board.	35
Figure 23: representation of Holybro payload platform board.	35
Figure 24: representation of Holybro X500 complete assembly.	35
Figure 25: Ardupilot Mega 2.0.	36
Figure 26: AeroQuad flight controller board.	36
Figure 27: OpenPilot CC3D flight controller.	36
Figure 28: PaparazziUAV Lisa/M v2.0.	36
Figure 29: Pixhawk 4 Mini flight controller.	36
Figure 30: MikroKopter v2.5 flight controller.	36
Figure 31: PX4 software structure from (Meier, 2015).	36
Figure 32: PX4 flight stack schematic representation.	37
Figure 33: illustration of Pixhawk 4 Mini flight controller.	37
Figure 34: illustration of stator width and height of a BLDC motor.	39

Figure 35: photo of Goolsky A2212 Outrunner Motor installed on Holybro X500 frame arm.	40
Figure 36: photo of complete propulsion module: Goolsky 1045 propellers and Goolsky A2212, installed on Holybro X500 frame.	40
Figure 37: wire welding procedure of BLDC motor Goolsky A2212 1000KV and HP 30A ESC.	41
Figure 38: pie chart showing the weights distribution in grams of the realized prototype quadcopter.	41
Figure 39: illustration of 3S LiPo power battery Youme, nominal voltage 11.1[V], capacity of 3300[mAh] with a C-rating of 50[C].....	42
Figure 40: Starter Set HW v4.9-NIA includes four stationary Beacons HW v4.9, one mobile Beacon HW v4.9 and one router modem v5.1.	43
Figure 41: illustration of Marvelmind Set HW v4.9-NIA 4x stationary beacons, 1x mobile Beacon and modem disposition. Reference (Robotics, Indoor "GPS" Autonomous Copter Setting Manual)	43
Figure 42: illustration of QGroundControl application user interface running on a Windows 10 operating system with a PX4-based autopilot.	44
Figure 43: illustration of Pixhawk 4 Mini power wiring scheme. Reference "PX4 User Guide".....	45
Figure 44: photo of designed quadrotor PMB and ESC wiring connection, installed on X500 frame bottom plate.....	45
Figure 45: photo of Raspberry-Pi 3 installed on Holybro X500 frame bottom plate through 3D printed support.	45
Figure 46: photo of LCD, ESCs and mobile ultrasonic sensor installed on Holybro X500 frame through 3D printed component.....	45
Figure 47: quadrotor hardware architecture block diagram and wiring scheme of connections between power module, propulsion module, autopilot, companion computer, local positioning system and external additional hardware.	46
Figure 48: photo of 3D printer during printing of the component Marvelmind mobile Beacon sensor's chase.	46
Figure 49: 3D printed ultrasonic sensor support installed on Marvelmind mobile Beacon.	46
Figure 50: Solidworks® render of designed CAD model of Pixhawk 4 Mini autopilot chase.	47
Figure 51: photo of Pixhawk 4 Mini secure to Holybro X500 bottom plate through 3D printed chase.	47
Figure 52: Solidworks® render of designed CAD model of LCD support for Holybro X500 payload platform.	47
Figure 53: photo of LCD screen installed on Holybro X500 frame payload platform board through 3D printed support.	47
Figure 54: Solidworks® render of complete designed quadrotor CAD model, including Holybro X500 airframe, BLDC motors and propellers. In the top left front view, in the top right top view and in the bottom perspective view.....	48
Figure 55: on the left front view of quadrotor final assembly; on the right perspective view of quadrotor final assembly.	48
Figure 56: illustration of total potential field $U_{tot} = U_{att} + U_{rep}$ computed by APF algorithm for the snail pattern in presence of obstacles. Quadcopter path trajectory has been added, showing the attractive actions of waypoints and the repulsive actions of obstacles on quadcopter behaviour during a path following manoeuvre.....	51
Figure 57: block diagram PD controller to control quadrotor slow dynamics.	53

Figure 58: block diagram PID controller to control quadrotor altitude.	53
Figure 59: snail pattern, path following MPC controller. On the left MPC controller inputs F_z , τ_x , τ_y , τ_z ; on the right 3D plot of snail path.	56
Figure 60: $\Delta\tau_x$ inputs optimization results over prediction horizon $N = 30$. Each coloured circle represents a control input $\Delta\tau_x$ computed by the MPC. The prediction horizon ($N=30$) is on the x-axis while the different colours are referred to predicted inputs at different times of computations.	57
Figure 61: flow chart of MPC algorithm implementation in MATLAB and Simulink 2018b.	60
Figure 62: illustration of main elements of simplified Simulink model for Path Following, block diagram....	63
Figure 63: illustration of inputs and outputs of designed Trajectory Planner block in MATLAB-Simulink 2018b.	64
Figure 64: illustration of inputs and outputs of designed Motor Mixer block in MATLAB-Simulink 2018b. ..	64
Figure 65: illustration of complete Simulink model based on PD (position control) and MPC (attitude and altitude control) controller. Software platform MATLAB-Simulink 2018b.	65
Figure 66: illustration of complete Simulink model based on MPC (position and attitude control) and PID (altitude control) controller. Software platform MATLAB-Simulink 2018b.	67
Figure 67: illustration of main elements of modified Simulink model for Obstacle Avoidance task, block diagram.	68
Figure 68: Artificial Potential Field Simulink subsystem. Software platform MATLAB-Simulink 2018b.	68
Figure 69: comparison of square pattern path following North-East plane; on the left PD controller and on the right MPC controller.	69
Figure 70: comparison of square pattern time response z_{NED} and w_{NED} ; on the left MPC altitude control and on the right PID altitude control.	70
Figure 71: comparison of square pattern time response u_{NED} and v_{NED} ; on the left PD velocity control and on the right MPC velocity control.	70
Figure 72: comparison of butterfly pattern path following North-East plane; on the left PD controller and on the right MPC controller.	71
Figure 73: butterfly pattern time response of Euler angles ϕ , θ , ψ and angular rates p , q , r of a system based on a MPC controller for the attitude and altitude control and a PD controller for the position control.	72
Figure 74: butterfly pattern time response of Euler angles ϕ , θ , ψ and angular rates p , q , r of a system based on a MPC controller for the attitude and position control and a PID controller for the altitude control.	72
Figure 75: comparison of snake pattern path following North-East plane; on the left PD controller and on the right MPC controller.	73
Figure 76: comparison of snake pattern time response z_{NED} and w_{NED} ; on the left MPC altitude control and on the right PID altitude control.	74
Figure 77: comparison of square pattern time response u_{NED} and v_{NED} ; on the left PD velocity control and on the right MPC velocity control.	74
Figure 78: comparison of snail pattern path following North-East plane; on the left PD controller and on the right MPC controller.	75
Figure 79: path following and obstacle avoidance North-East plane MPC controller (time of simulation 60[s], MPC prediction horizon 30 step). On the left square pattern; on the right butterfly pattern.	76

Figure 80: path following and obstacle avoidance North-East plane MPC controller (time of simulation 100[s] snake and 45[s] snail, MPC prediction horizon 30 step). On the left snake pattern; on the right snail pattern.	76
Figure 81: UAV Toolbox blocks used for co-simulation framework that links Simulink and Unreal Engine from Epic Games.	78
Figure 82: work logic diagram Simulink and Unreal Engine communication.	78
Figure 83: On the left Unreal Engine Editor main page configuration; on the right mobile preview of path following and obstacle avoidance task of the simulated quadrotor running in Simulink UAV Toolbox and visualized in a customize scene in Unreal Engine (Microsoft Windows 10).	79
Figure 84: MATLAB results showing North-East plane quadrotor behaviour in the avoidance manoeuvre of 3 obstacles using an MPC based controller. Obstacle position $xobs = 4,0,0; 11,0,0; 17,0,0[m]$ with dimensions $Robs = 0.5[m]$ and a tolerance of $\eta_0 = 0.4,0.4,0.4[m]$	80
Figure 85: Unreal Engine Editor high resolution screenshot of customize obstacle avoidance path. The disposition of obstacles and the area of manoeuvre have been defined in accordance with data provided by MATLAB script.	80
Figure 86: Unreal Engine camera capture of running quadcopter simulation using Python API commands implemented in Visual Studio Code. Environment name: ZhangJiaJie. Ground Unit: Windows 10 laptop, Intel Core i7, graphic processor unit NVIDIA GeForce GTX970M.	81
Figure 87: illustration of AirSim simulator architecture. The scheme shows a block diagram of the 6 main elements and their interactions.	82
Figure 88: overview of AirSim RPC procedure based on MAVLink library.	83
Figure 89: on the left file settings. json compatible with latest AirSim v1.6 configured for 'SimpleFlight' vehicle communication; on the right settings. json compatible with AirSim v1.2 (or earlier) configured for 'PX4' communications protocol based on MAVLink messages.	83
Figure 90: on the left Python function used to establish a connection with the quadrotor, to enable the API commands, to arming the motors and take-off with a time delay of 3 [s] on AirSim; on the right Python function used to show lidar point cloud and detect obstacles.	84
Figure 91: illustration of AirSim environment Africa and customize FC connection procedure using Cygwin64 Terminal for Windows 10. The connection is established, and quadrotors motors are ready to be armed and take-off.	86
Figure 92: illustration of AirSim environment and customize FC connection using Cygwin64 Terminal for Windows 10. Quadrotor is flying and some examples of implemented commands results are printed on Cygwin64 Terminal interface.	86

List of tables

Table 1: UAV categorization, Kimon P. Valavanis, George J. Vachtsevanos, "Handbook of Unmanned Aerial Vehicles", Springer Reference, 2015 & Peter van Blyenburgh, "UAV Systems: Global Review", Avionics'06 Conference Amsterdam, March 9, 2006.	16
Table 2: open-source autopilots of multicopters.	36
Table 3: main characteristics of Pixhawk 4 mini, as taken from 'Pixhawk 4 Mini Technical Data Sheet' (Foundation, 2018).	38
Table 4: guideline table for motors and propellers selection.	40
Table 5: technical details Marvelmind Starter Set HW v4.9-NIA. Reference (Robotics, Marvelmind Starter Set HW v4.9-NIA, 2020)	43
Table 6: Quadcopter physical properties used in controllers tuning.	52
Table 7: definition of vector of states, continuous state space matrices A (states) B (inputs) for implemented MPC controller. On the left system based on MPC for attitude and altitude control; on the right system based on MPC for position and attitude control.	62
Table 8: Position Control - outer-loop quadrotor slow dynamics PD parameters; Attitude & Altitude Control – inner loop quadrotor fast dynamics MPC states weighting matrix $QMPC \in \mathbb{R}^{10,10}$; Inputs Control – inner loop quadrotor fast dynamics MPC inputs weighting matrix $RMPC \in \mathbb{R}^{6,6}$	66
Table 9: Altitude Control - outer-loop quadrotor slow dynamics PID parameters; Position & Attitude Control – inner loop quadrotor fast dynamics MPC states weighting matrix $QMPC \in \mathbb{R}^{10,10}$; Inputs Control – inner loop quadrotor fast dynamics MPC inputs weighting matrix $RMPC \in \mathbb{R}^{6,6}$	67
Table 10: square pattern reference waypoints and time of arrivals.	69
Table 11: butterfly pattern reference waypoints and time of arrivals.	71
Table 12: snake pattern reference waypoints and time of arrivals.	73
Table 13: snail pattern reference waypoints and time of arrivals.	75

Introduction

Unmanned Aerial Vehicles (UAVs), commonly known as *drones*, are aircraft able to fly without any human pilot, crew, or passenger on board. This doesn't necessary imply that the vehicle can fly autonomously. UAVs can be controlled by an autonomous on onboard controller (*autonomous flight*) or can be remotely piloted. Typically, military drones are remotely operated from a Ground Control Station (GCS) with the help of different communication protocols such as MAVLink, UranusLink and UAVCAN. The term *autonomous flight* for UAV systems is used to identify the capability of a system to know the position and, in certain case, also sensing the environment to flight with little or no pilot inputs. Autonomous Vehicles (AVs) must be able to compute the absolute or relative positions in space and then take decision on the path to follow for reaching a target, relying on available data, provided by its onboard electronic system.

Despite numerous advantages, like hovering and manoeuvrability, vertical take-off and landing and simple design, rotorcraft and multi-rotor UAVs may suffer various problems such as gust disturbance, mechanical vibration, and actuator failures during flight. All these problems reduce the flight performance and bring difficulties in the controller design. An effective controller must ensure smooth and collision free flight in the complex surrounding environment, considering aerodynamic drag and moments. These abilities come from the integration of an *autopilot* inside the control loop of the UAV. Most of modern autopilots incorporate control law algorithms to meet demanding requirements of high performances flight manoeuvres and to successfully accomplish the task of autonomous flight. In the last decades, a large number of control theory have been implemented for onboard *Guidance, Navigation and Control* (GNC) applications; however, despite their success, only a small number of that applications have been developed on real platform because of their complexity, nonlinear nature, and computational cost. In literature, the UAVs control problem has been addressed using control approaches, like *Proportional-Integrative-Derivative* (PID), *Linear-Quadratic-Regulator* (LQR) and *H-infinity* for linear control system. Some GNC solutions are off-the-shelf and can be customized by the user, but are usually used for research purposes. So, a low cost and reconfigurable system is the best solution for research. Despite the recent progress in the field of autonomous flight, navigation in GPS-denied environments continues to be a challenging problem that has been tackled in recent researches through sensor-based approaches.

This thesis focuses on a particular application of UAVs for *GPS-denied* environment, using a control technique based on *Receding Horizon Control* (RHC), also known as *Model-Predictive-Control* (MPC). The proposed solution is a feedback control system that first became popular in the 1980s. MPC leverages a plant model to predict the effect of an input profile on the evolving state of the plant. At each sample time, an *optimal control problem* is solved, and its optimal input profile is implemented until another plant measurement becomes available. The updated information is used to formulate and solve a new optimization problem, and the process is repeated. Furthermore, RHC handles input constraints, output constraints and a variety of control objectives, enabling controlled systems to operate near their physical limits, obtaining performances superior to linear control strategies.

1.1 – UAVs Historical Survey and Classification

Current developments in UAVs trace their modern origins back to the development of aerial torpedoes and flying bombs during the *First World War* (1914-1918). First efforts consisted of combining wood airframes with gyroscope and propellers to carry heavy payloads of explosives over a distance of hundreds of kilometres, to be used against U-boat bases. These rudimental UAVs highlighted two operational problems: launch-recover operations and stabilization during flight. During the *Interwar Period*, radio and improved aircraft engineering allowed UAV to enhance their performance. Radio-controlled drones were used by both the *Allied* and *Axis* during the *Second World War* (1939-1945). Soon after the end of the *Second World War* interest in reconnaissance missions increased; moreover, with the start of the *Cold War* (1947-1989), UAVs began to be used as *Intelligence, Surveillance and Reconnaissance* (ISR) systems. The improvement of unmanned vehicle performances continued throughout the *Vietnam War* (1955-1975), but the major studies were conducted during the *Gulf War* (1991). While drones have had a long history in military deployment, their increasingly widespread use in non-military roles is not negligible. Unmanned aerial vehicles are an example of how military technologies, in the past as well as in the present, can affect the civilian business with similar but completely different application.

In literature, different UAV classifications have been proposed to differentiate existing unmanned system, based on their operational characteristics and their capabilities. Different *requirements* and *rules* are imposed on different UAV classes, based on their operational applications. Aspects such as mean take-off weight, wingspan, speed, operative range, maximum altitude, and operating conditions are all specifications used as metrics to distinguish different class of UAVs. As discussed in Weibel and Hensman (R.E. Weibel, 2004), *mean take-off weight* (MTOW) is a good metric to classify aircraft for regulatory purposes since it correlates with the expected impact kinetic energy, which is affecting safety and operations. As an example, largest UAVs such as ‘*Predator*’ (Figure 1) or ‘*Northrop Grumman RQ-4 Global Hawk*’ (Figure 2) can weight several thousand pounds and can have wingspans on the order of 3 to 30 meters. On the other hand, there are UAVs whose maximum dimensions are on the order of centimetres and weights on the order of grams.



Figure 1: flying fixed wing UAV MQ-9A Predator B.



Figure 2: flying fixed wing Northrop Grumman RQ-4 Global Hawk.

Table 1 shows a possible classification based on UAVs mass, range, flight altitude and endurance.

Table 1: UAV categorization, Kimon P. Valavanis, George J. Vachtsevanos, "Handbook of Unmanned Aerial Vehicles", Springer Reference, 2015 & Peter van Blyenburgh, "UAV Systems: Global Review", Avionics'06 Conference Amsterdam, March 9, 2006.

	Mass (kg)	Range (km)	Flight alt. (m)	Endurance (h)
Micro	<5	<10	250	1
Mini	<20/25/30/150 ^a	<10	150/250/300	<2
Tactical				
Close range (CR)	25–150	10–30	3,000	2–4
Short range (SR)	50–250	30–70	3,000	3–6
Medium range (MR)	150–500	70–200	5,000	6–10
MR endurance (MRE)	500–1,500	>500	8,000	10–18
Low altitude deep penetration (LADP)	250–2,500	>250	50–9,000	0.5–1
Low altitude long endurance (LALe)	15–25	>500	3,000	>24
Medium altitude long endurance (MALe)	1,000–1,500	>500	3,000	24–48
Strategic				
High altitude long endurance (HALe)	2,500–5,000	>2,000	20,000	24–48
Stratospheric (Strato)	>2,500	>2,000	>20,000	>48
Exo-stratospheric (EXO)	TBD	TBD	>30,500	TBD
Special task				
Unmanned combat AV (UCAV)	>1,000	1,500	12,000	2
Lethal (LET)	TBD	300	4,000	3–4
Decoys (DEC)	150–250	0–500	50–5,000	<4

^aVaries with national legal restrictions

A variety of UAV system has been developed and is currently in the advancement phase; some of them includes the *fixed-wing aircraft*, *single rotor*, *multirotor* and *fixed-wing hybrid UAV*. A classification based on *aerodynamic configuration* is shown below.

- *Fixed Wing UAVs* are equipped with fixed wings and a rotor positioned on the front or back of the main body. This class of UAVs outperforms multirotors in term of flight autonomy and cruising velocities. However, similar to aeroplanes, they do not have the ability to vertically take-off and land. They require a runway or alternatively can be catapult launched. Concerning their application's field, they are extensively used in various monitoring operations such as meteorological and environmental monitoring or reconnaissance.



Figure 3: SITARIA UAV fixed-wing unmanned electric aircraft designed by UAVOS.

- *Single-Rotor or rotorcraft UAVs* are propelled by a single rotor positioned in the middle of the vehicle's body (*main rotor*) and a rotor on the tail to counteract the torque generated by the main propeller (*tail rotor*). This class of UAVs are able to vertically take-off and land, hover, fly in low altitudes, rotate in the air, and move backwards and sideways. However, they need complex mechanical transmission of torque, so they are not widely used.



Figure 4: MQ-8 Fire Scout single-rotor UAV developed by Northrop Grumman for USAF.

- *Multi-Rotor UAVs* are the most common category. This class of UAVs poses a set of advantages compared to fixed wing class, such as hovering, vertical take-off and landing and accurate manoeuvring. Despite their advantages, multirotors consume significantly more energy because thrust is generated only by propellers, unlike fixed wing UAVs where thrust is generated by propulsors and shape of the spacecraft. There are several types of *chasses* available with four, six, eight or more propellers. One of the most used is the *quadrotor*, which has four rotors and it is typically designed in a cross configuration, with two pairs of opposite rotors rotating clockwise and the other two rotating counter-clockwise to balance the torque.



Figure 5: Freefly ALTA 8 Multirotor Camera Drone design by Freefly Systems.

- *Fixed-Wing Hybrid UAVs* are able to vertical take-off and land but, at the same time, can assume a configuration similar to fixed-wing during flight. These properties allow the UAVs to increase cruising speed like fixed wing UAVs and vertically take-off and land like single-rotor and multi-rotors.



Figure 6: Joby S4 fixed-wing hybrid e-VTOL developed by Joby Aviation.

1.2 – Motivations of the Thesis

Thanks to their adaptability and versatility UAV applications have proliferated vastly in the last few years and their operational experience has proven that UAV technology can have a dramatic influence in the military and civilian areas. The utility of *drones* in military applications is readily apparent. UAVs can potentially carry out the range of tasks normally executed by human operators without placing human pilots in jeopardy. These military purposes are not suitable for civilian usage, as most of these UAVs are large in size, equipped with expensive instruments and needed proper supporting facilities. It's worth mentioning that availability and easily accesses of advanced autonomous technologies have encouraged UAVs civilian applications. According to Merkert and Bushell (Rico Merkert, 2020), drones' field of application can be classified into four main categories: *monitoring and data acquisition, photography, logistics and recreation*. Hereafter some practical examples.

- *Saving life*: according to Zurli and Leiras (Raissa Zurli Bittencourt Bravo, 2015), the used of unmanned aerial vehicles to support humanitarian actions has grown since 2001, after the terrorist attack of 9/11. One of the most challenging difficulties facing United Nations and Non-Governmental Organizations when responding to disasters, like floods, earthquakes and hurricanes is to understand the condition of the affected population accurately and rapidly. Current methods are time consuming, and the captured data are often inaccurate. Drones can perform in disaster environment, providing first-class services without compromising lives of life-saving human operators.
- *Forecast of hurricanes*: a drone can reach areas subject to devastating meteorological events in order to collect data for the study and prevention of similar events.
- *Infrastructure maintenance*: drones can be controlled by specialized operators carrying out dangerous operations like power-line inspection in close proximity to live electrical cables. For the overall surveying industry, UAV usage brings vital time and money saving and decreases exposure of staff to dangerous environments. Moreover, *micro-UAVs* are most appropriate for reconnaissance mission inside buildings because of their compact design and manoeuvrability.

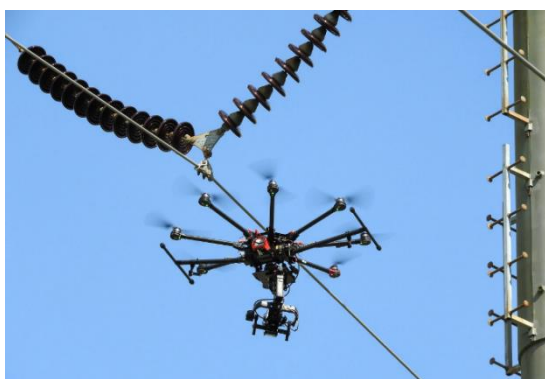


Figure 7: octocopter performs power-line inspection.



Figure 8: picture provided by new service approach for power-line inspections provided by Siemens SIEAERO¹.

¹ Siemens has launched a new service approach for overhead line inspection called 'SIEAERO' at *European Utility Week 2018* in Vienna, Austria. SIEAERO smart analytics software is utilizing artificial intelligence AI and machine learning to store manage and analyse all data in one integrated software system.

- *Agricultural monitoring:* images taken by low altitude remote sensing platforms, such as drones, are useful in *Precision Agriculture*² (PA). According to Ehsani and Mari Maja (Reza Ehsani, 2013), the number of farms in the United States in 2012 was estimated at 2.2 million, with an average size of 471 acres. Most manual activities require operators to perform intensive field collection and are therefore destructive and time-consuming. In contrast, UAVs can be applied in agriculture activities such as crop scouting, irrigation and drainage planning, efficient use of chemicals and pesticide completely autonomously, thus saving resources and time. Moreover, UAVs can be equipped with a variety of imaging sensors, such as hyperspectral camera or thermal camera, collecting field data providing useful information on field status, composition, and production capability.



Figure 9: agricultural drones being used to spray pesticide on crops in a village in Poyang, central China's Jiangxi province.

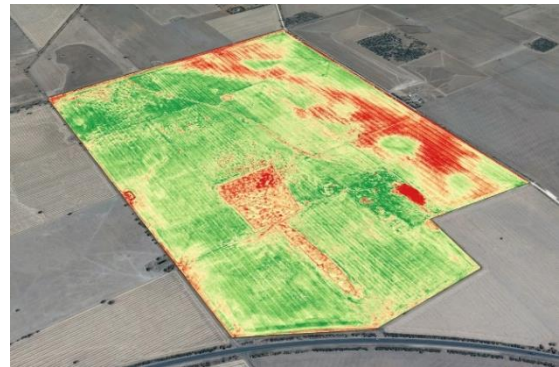


Figure 10: false-colour difference vegetation index from near-infrared images collected by an Agribotix drone.

- *Support law enforcements:* drones can be used to perform surveillance operations, accident investigation and suspect tracking and crowd monitoring.
- *Aerial Photography:* a drone can perform risk-free aerial shooting for harsh and hazardous environments. Earth observation can significantly contribute to improving efforts in developing proper disaster mitigation strategies and providing relevant agencies with essential information for alleviating impacts of a disaster and relief management.



Figure 11: photo captured by UAVs high resolution camera during search and rescue operation, after the devastating earthquake and tsunami in Banda Aceh, Indonesia on 26 December 2004.

² According to the *International Society of Precision Agriculture*, PA is a management strategy that gathers, processes and analyses temporal, spatial and individual data and combines it with other information to support management decisions according to estimated variability for improved resource use efficiency, productivity, quality, profitability, and sustainability of agriculture production.

- *Packages delivery:* UAVs can be customized to delivery small-sized packages. Many multinational technology companies, such as, *Google* has already built and tested autonomous aerial vehicles, and believes they could be used for goods deliveries.



Figure 12: Wing company launches its first drone delivery service in Australia (2019).



Figure 13: photo of Wing drone from Google delivers goods during COVID-19 pandemic.

- *3D mapping:* drones are able to use their sensors to capture images, scans environments and elaborate useful data to process them in term of three-dimensional maps.

Beyond aforementioned, two more motivators for multirotor are *reliability* and *compactness*. These two characteristics are essential for portable systems that operate in close proximity to people and closed unknown environments. Conventional helicopters use complex mechanism to control the attitude of the airframe. This system is called ‘*swashplate*’, an assembly of sophisticated pieces of high-speed machinery operating in a vibrating environment, highly prone to failure without constant maintenance. On the other hand, quadrotors have an almost negligible chance of catastrophic failure with inexpensive maintenance, thanks to the simple design of the direct-drive electric motor head (with only four moving parts, the *propellers*). Moreover, the reduced rotor diameters and the absence of a long tail boom that can collide with obstacles, make multirotor, and particularly *quadrotors*, the ideal solution for precision tasks to be accomplished in indoor and outdoor environments.

Currently, UAVs are mostly remotely piloted by humans, operating from ground stations. This approach requires highly skilled pilots which is expensive and time-consuming. In the field of security and civilian applications, there is a strong need to provide reliable technology, so that some of the decision-making responsibility resides within the vehicle. This approach comes at a price, as controlling a multirotor is not easy because of the coupled dynamics and its under-actuated design configuration. Moreover, the dynamics of a quadrotor is highly non-linear, and several uncertainties must be considered, thereby making its flight control a challenging venture.

The motivation of this thesis is to face the main technological challenge of developing, modeling and controlling an autonomous multi-rotor system for GPS-denied applications, integrating sensors, actuators, and algorithms into a lightweight and volume efficient working solution. Since commercially available quadrotor can be very restrictive for use in research applications, in this thesis a customized quadrotor has been used.

Thus, the main objective of this thesis is to build a customized small prototype quadrotor, using Commercial-Off-the-Shelf (COTS) components. More particularly, the proposed research focuses on the development of

a multirotor UAV for indoor GPS-denied environment, using a set of ultrasonic sensors as *indoor positioning system*. The work focuses on both hardware (HW) and (SW) implementation and testing.

The first part of the project concerns the HW selection and assembly. There are many quadrotors that are commercially available nowadays, however, off-the-shelf quadrotors usually lack the ability to be reprogrammed and are unsuitable for use as research platforms. Therefore, an HW selection has been carried out, starting from COTS components. All the components, for example, the motors, the propellers, and the avionics systems (such as autopilot, flight computer and sensors) are chosen concerning compatibility constraints, good endurance, and low weight. Research and development activities presented herein include selection of frame-compatible COTS avionics elements, design and modeling of 3D printed integration components and their assembly in an efficient working configuration. As concerns SW implementation, links between avionic systems and indoor ultrasonic sensor have been established using algorithms written in C and Python. The first part of the project ended with the verification and validation of communication between HW components and flying tests.

The second part of the project concerns the SW development and testing. Due to the complexity and the computational need of the control algorithm, most of the commercial autopilots are based on Proportional Derivative Integrative (PID) controller. The main objective of this thesis is to develop a structured *interior-point method* for the efficient solution of the MPC-based optimal control problem.

Starting from the equations of motion of an aircraft, assuming some simplifications, the nonlinear mathematical model of the plant is derived. Quadrotor dynamics can be split into two categories: slow dynamics, regarding the position; and fast dynamics, regarding the attitude and altitude. Thanks to this classification, an MPC-based cascade controller is developed. The controller is structured into two loops: an *outer loop* related to UAVs slow dynamics (controlled by a PD controller) and an *inner loop* related to UAVs fast dynamic (controlled by MPC controller). Two solutions are compared: the first one features a MPC in the inner loop driving both the quadrotor attitude and altitude control, whereas in the outer loop a simple PD controller is used to track the North-East position. The second solution uses a PID for altitude control and an MPC for attitude and North-East position control. Both controller's performance are tested tracking different reference trajectories ranging from simple ones to complex waypoints-following trajectories with increasing difficulty in terms of changes in the states. Furthermore, an *Artificial Potential Field* (APF) based guidance algorithm is provided to test the ability of obstacle avoidance.

A broader contribution of this study is the evaluation of the designed controller in a simulated environment. The experimental results and data collected in this research project provide a reference basis to future embedded applications of MPC controller to a real platform. Several simulations are conducted to examine and evaluate the performance of the proposed control approach using MATLAB and Simulink environment. Simulation results show that this kind of control is highly effective to track different types of given reference trajectory. The performances of the controller have been further tested on a virtual environment, using *Unreal Engine* as plant to have more realistic results representations. Furthermore, the controller block is designed such that it can undergo the code generation process without any substantial modifications.

In conclusion the research has ultimately resulted in the following scientific contribution in the area of UAV modeling and controller design:

- A customized small quadrotor UAV for indoor GPS-denied environment has been developed from COTS hardware components. The framework is designed such that it can be equipped with a set of ultrasonic sensors (*'Marvelmind Set HW v4.9' indoor positioning system*) to generate a navigation solution, which, in turn, is used in the guidance and control algorithm.
- A MPC controller written in MATLAB is provided and tested in different simulation environments (Simulink and Unreal Engine). Using the *'Embedded Coder'* tool the MPC Simulink model is converted into C language and tested in a simulated environment based on *Unreal Engine* software framework. Experimental results and data collected in this research provide reference basis to future embedded applications of MPC to real platform.

1.3 – Thesis Outline

This work is the result of eight months activities in the research, modeling, and assembly of a real quadcopter platform, in collaboration with *ALTEN Italia spa*, a French multinational technology consulting and engineering company placed in Milan.

This thesis consists of six chapters, here summarized for a quick understanding of the whole work structure:

- *Chapter 1* provides a brief introduction to the topic of UAV as well as main motivation of the thesis. It presents a brief overview of UAVs history and a possible categorization of unmanned vehicles based on *aerodynamic configuration*. It also outlines the main advantages-drawbacks of UAVs and their military and non-military application. The chapter ends with the outlines of motivation, aims and organization of the work.
- *Chapter 2* focuses on the preliminaries required for comprehensive understanding of concepts presented in this work. Firstly, the operation principle of multirotor is presented, followed by the basic mathematical notions for the derivation of quadrotor dynamic model and the controller design. Starting from reference frames and vehicle's axes configuration, the equation of movement is obtained. Once obtained a *state space* representation of the model, a linearization is performed in order to apply a *Linear Model Predictive Control* MPC.
- *Chapter 3* presents a brief overview of UAV technology and individual components used in multirotor. It describes the development of the various hardware and software components that constitute the proposed customized quadrotor. The design and modeling of the prototype quadrotor are provided, focusing on its avionics COTS components selection and integration in a working configuration. It also presents the integration of ultrasonic sensors as *indoor positioning system* to generate a navigation solution, which, in turn, is used in the guidance and control algorithm. Eventually, details of modelled quadrotor are provided through a CAD model realized in SolidWorks environments. The chapter ends with a photographic collection of quadrotor and flight test.
- *Chapter 4* presents an overview of typical Guidance, Navigation and Control (GNC) system, focusing on the control algorithms applied in this thesis. A brief overview of RHC and interior-point method is provided, followed by the development of an MPC-based cascade controller. The state space linearized model, obtained in *Chapter 2*, is used in the design of two solutions: the first one features a MPC in the inner loop driving both the quadrotor attitude and altitude control, whereas in the

outer loop a simple PD controller is used; the second one uses a PID for altitude control and an MPC for attitude and position control.

- *Chapter 5* treats the results obtained from Simulink model. All the controllers presented in *Chapter 4* are tested. The results obtained from different path tracking are commented and compared for different patterns. This chapter also presents the problem of *obstacles avoidance* including an APF algorithm in the main model. Lastly the performance of the designed MPC in *Unreal Engine* simulated environment are investigated.
- *Chapter 6* summarises the main points and results of this thesis. It also outlines the possible future works of this project together with some improvement suggestions.

Multirotor Mathematical Model

In this chapter a qualitative introduction on the working principles of quadrotor is discussed. Firstly, a brief description of the reference frames used to characterize drone's dynamics is exposed. Before describing the mathematical model, the configuration of the axes is explained. Then, the mathematical model is derived, using Newton's law. Once obtained the mathematical model, a *linearization* is performed, obtaining a *state space* representation of which *controllability* and *observability* are analysed.

2.1 – Quadrotor Working Principle

The simplest fully controllable multi-rotor aircraft is the *quadrotor*, which uses four rotors connected to the fuselage through booms, generally arranged in a *square* pattern. The whole quadcopter setup consists of two clockwise (CW) rotating motors and two counter-clockwise (CCW) rotating motors on the vertex of a square framework. There are two possible configurations: an 'X' (cross) configuration, which maximizes the moments generated by the motor thrust (shown in Figure 14), and a '+' (plus) configuration (shown in Figure 15). According to Niemiec and Gandhi (Robert Niemiec, September 5-8, 2016), an important distinction between the cross and plus configuration is that when producing a pitching or rolling moment, the cross-type uses all four rotors, as opposed to the plus-type that uses only two of them (this will be examined deeply in the next paragraph).

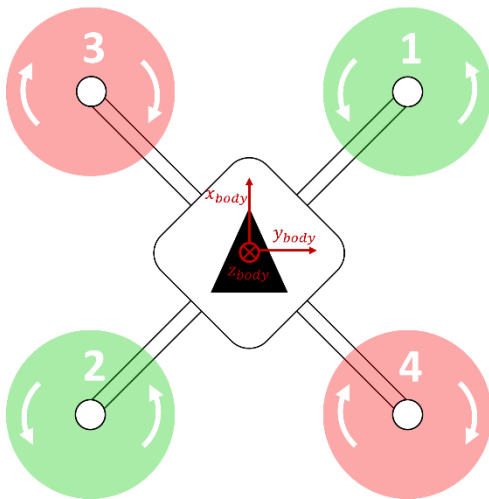


Figure 14: quadrotor cross 'X' configuration.

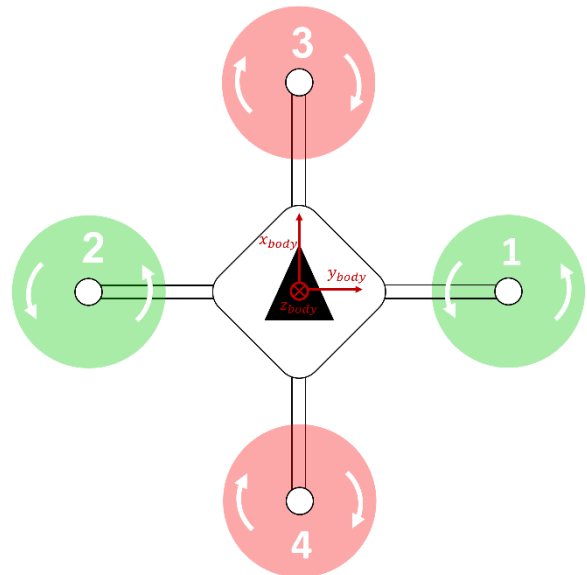


Figure 15: quadrotor plus '+' configuration.

The spin direction of the motors is critical since it counterbalances the torque generated by the spinning propellers, preventing the rotation of the drone around its 'down' axis. The quadrotor is an *under-actuated*

system, this means that the rotational and translational motions in the three-dimensional space in all 6 *Degrees of Freedom* (DoF) have to be coupled, bringing to a highly *non-linear* dynamic.

Four control inputs, corresponding to the rotational speed of the four rotors, are generally used to control the quadcopter dynamic. All the control actions, roll, pitch, yaw, and up-thrust are controlled by changing the thrust of the rotors using *Pulse-Width-Modulation* (PWM) to obtain the desired output. While this is valid to control the speed of each rotor individually, this kind of control produces a highly coupled response. As an example, for both the plus and cross configurations, the *collective mode*, affects only the overall generated thrust and doesn't generate any roll, pitch or yaw torques. Considering the *pitch mode*, the two front rotors speed up and the two rear rotors slow down on the cross-configuration, generating a nose-up pitching moment. Of the two front rotors speeding up, one rotates CW and the other CCW, hence the torques generated cancels out. The same is true of the rear rotors slowing down, hence pitch mode doesn't introduce any yaw moment on the cross-configuration. On the other hand, the pitch mode speeds up the single front rotor and slow down the single rear rotor on the plus configuration. The torque does not vary linearly with RPM, hence the increase in torque of the CCW spinning front rotor does not cancel with the torque reduction of the CCW rear rotor, resulting in a yaw moment of the airframe requiring a compensation with a yaw control input. That having been said, there remains a distinction between the two considered configurations: the *pitch* and *roll* control modes on the cross-type are decoupled from yaw, while they introduce yawing moments on the plus-type, requiring a compensation action.

As said earlier, the attitude and the position of the quadrotor can be controlled to desire values by changing the speed of the four rotors. The *space motion* of the rigid body aircraft, can be divided into two parts: the barycentre movement and the movement around the CoG. Space motion can be described any time with six degree of freedom, three translation and three rotation motions along the three axes. Depending on the speed rotation it is possible to identify the four basic movements of the quadrotor, which are shown below.

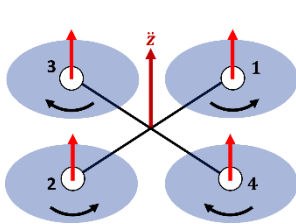


Figure 16: quadrotor upward movement

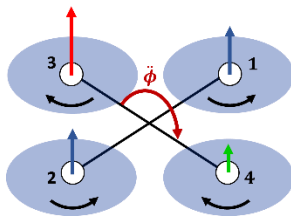


Figure 17: quadrotor roll movement

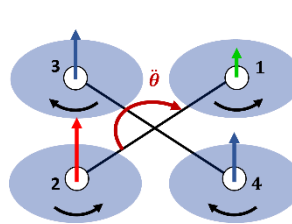


Figure 18: quadrotor pitch movement

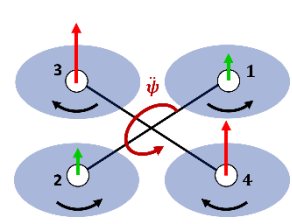


Figure 19: quadrotor yaw movement

2.1.1 – Reference Frames

There are different frames of reference that need to be considered in quadrotor analysis. First of all, it is necessary to understand how the quadcopter moves in space and which reference frames are used. There are two main reference frames: an *inertial* reference frame and a *Body* fixed frame (both of them are presented in Figure 20). In this works the *North-East-Down* (NED) reference frame is used. The NED frame, also called the *Inertial Frame*, is fixed with the 'down' axis points toward the centre of the Earth. It doesn't move or rotate with the drone; on the contrary, the *Body* frame is a *non-inertial* reference frame composed by a set of axes fixed to the body. The body frame is rotated with respect to the inertial frame whenever drone moves since it is centred in the quadrotor *Centre of Gravity* (CoG). The axes of the *non-inertial* frame

are x_b, y_b, z_b . The x_b axis points in the forward direction; the z_b axis points towards the centre of the Earth, and lastly y_b completes the *right-hand rule*.

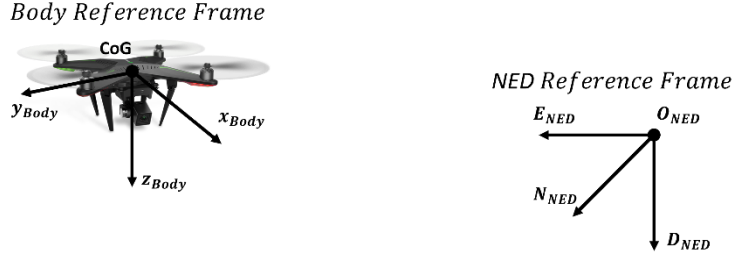


Figure 20: illustration of used reference frames: the *Body* reference frame in the top left and the *NED* inertial reference frame in the bottom right.

2.1.2 – Euler Angles and Quaternions

As introduced in the paragraph above, the *Body* frame is rotated with respect to the inertial frame whenever drone moves since it is centred in the quadrotor *Centre of Gravity* (CoG); thus, a conversion from one reference frame to another is needed to describe the orientation of the airframe in the three-dimensional space. This conversion can be performed making use of two distinct approach *Euler angles* or *Quaternions*.

- *Euler Angles* can describe any arbitrary three-dimensional rotation with a sequence of individual rotation around three axes. Euler angles are also defined as three parameters ϕ , θ and ψ . A three-dimensional rotation matrix is a 3×3 matrix because each point in a frame has three coordinates that must be changed. Considering an arbitrary frame x, y, z , the rotations around each of the frame axes are expressed as three matrices, respectively R_x , R_y and R_z :

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix} \quad R_y = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \quad R_z = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad (1)$$

In order to build the matrix that convert *NED* inertial reference frame to *Body* reference frame, three rotations have to be performed: the first one around z_{NED} of value ψ , the second one around y'_{NED} of value θ (it is used y'_{NED} instead of y_{NED} since the new y axis refers to the rotated reference frame) and the last one around x''_{NED} of ϕ (it is used x''_{NED} instead of x_{NED} since the new x axis refers to the twice rotated reference frame). It is possible to rewrite the three rotations multiplying the three matrices defined in (1), obtaining the transformation from *NED* to *Body*:

$$R_{NED}^{body} = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix} \quad (2)$$

The opposite conversion between the *Body* frame and the *NED* frame is obtained throw the transposition of matrix in (2). The resulting matrix is shown below:

$$R_{body}^{NED} = \begin{bmatrix} \cos \theta \cos \psi & \sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi & \cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi \\ \cos \theta \sin \psi & \sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi & \cos \phi \sin \theta \sin \psi + \cos \phi \cos \psi \\ -\sin \theta & \sin \phi \cos \theta & \cos \phi \cos \theta \end{bmatrix} \quad (3)$$

As an example, assuming $a \in \mathbb{R}^3$ defined in *NED* and the corresponding $a' \in \mathbb{R}^3$ in the *Body* reference frame, the possible transformations are:

$$a' = R_{NED}^{body} \cdot a = R_{body}^{NED^T} \cdot a \quad \text{or} \quad a = R_{body}^{NED} \cdot a' = R_{NED}^{body^T} \cdot a' \quad (4)$$

Even though *Euler* angles grant a better visualization of the attitude of the vehicle, it is worth mentioning that they must be limited due to the singularities that can be found using them as attitude variables within the *kinematics equations*. Moreover, ϕ and ψ become undistinguishable when θ assumes critical values; often referred to as *Gimbal Lock Singularity*. Due to these disadvantages the quadrotor attitude is often described using the *quaternions*.

Before introducing quaternion algebra, the relationship between the quadrotor attitude with reference to *NED* inertial frame (*Euler* angles) and the angular velocities p, q and r is provided:

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

Quaternions are *hyper complex* numbers in \mathbb{R}^4 that allow the representation of orientations and rotations of an object in three dimensions with the advantage over Euler angles method of not suffering from singularity. A quaternion consists of four elements: a *scalar* part, often referred to as q_0 , and a *vectorial part* $\mathbf{q} = [q_1 \ q_2 \ q_3]$. The *4-tuple* can be represented in many ways, while two of the most popular approaches are shown in equations (6):

$$q = q_0 + q_1 \mathbf{i} + q_2 \mathbf{j} + q_3 \mathbf{k} \quad q = [q_0 \ q_1 \ q_2 \ q_3]^T \quad (6)$$

All the quantities q_i are real numbers and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ satisfy the following identities:

$$\begin{aligned} \mathbf{i}^2 &= \mathbf{j}^2 = \mathbf{k}^2 = -1 \\ \mathbf{ij} &= \mathbf{k}, \mathbf{ji} = -\mathbf{k} \\ \mathbf{jk} &= \mathbf{i}, \mathbf{kj} = -\mathbf{i} \\ \mathbf{ki} &= \mathbf{j}, \mathbf{ik} = -\mathbf{j} \end{aligned} \quad (7)$$

Quaternion *conjugate*, quaternion *normalization* and quaternion *multiplication* are the three main operations used in the quaternion algebra for attitude control.

Given the quaternion $q = q_0 + q_1 \vec{\mathbf{i}} + q_2 \vec{\mathbf{j}} + q_3 \vec{\mathbf{k}}$ its *conjugate* is defined as:

$$q^* = q_0 - q_1 \mathbf{i} - q_2 \mathbf{j} - q_3 \mathbf{k} \quad q = [q_0 \ -q_1 \ -q_2 \ -q_3]^T \quad (8)$$

And its *norm* as:

$$|q| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \quad (9)$$

The *normalization* operation is used to convert a quaternion into *unit* quaternion as shown in (10):

$$\hat{q} = \frac{q_0 + q_1 \mathbf{i} + q_2 \mathbf{j} + q_3 \mathbf{k}}{\sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}} \quad (10)$$

The *multiplication* of two quaternions is being performed by the *Kronecker*³ product, also referred to as *Hamilton* product, denoted as \otimes .

³ *Kroner product*: if q represent one rotation and p represents another rotation, the quaternion multiplication $q \otimes p$ represents the combined rotation. The product is equal to: $q \otimes p = q_0 p_0 - \mathbf{q} \cdot \mathbf{p} + q_0 \mathbf{p} + p_0 \mathbf{q} + \mathbf{q} \times \mathbf{p}$

Thanks to the *Hamilton* product, it is possible to express an *error quaternion*, like the difference between a desired attitude and the current one, using the following equation:

$$q_{err} = q^{-1} \ddot{A} q_{des} \quad (11)$$

Where q is the current estimated quaternion and q_{des} is the provided reference. The *inverse* of a quaternion is defined as the normal inverse of a complex number, $q^{-1} = \frac{q^*}{|q|^2}$. Moreover, if the quaternion is unitary, then the inverse is the same as its conjugate.

Thanks to *Euler's Rotation Theorem*, any displacement of a rigid body in 3D space such that a point of the rigid body remains fixed is equivalent to a single rotation about an axis passed through that fixed point. The axis of rotation passing through that fixed point is called *Euler axis* and is represented by $\vec{a} = (a_x, a_y, a_z)$ and α is a simple rotation angle. Using these two elements, the corresponding rotation is translated into the equivalent quaternion:

$$q = \cos \frac{\alpha}{2} + \vec{a} \sin \frac{\alpha}{2} \quad (12)$$

Assuming a vector $\vec{v} \in R^3$ from reference frame l , and $\vec{v}' \in R^3$ as the same vector in a different reference frame m , then the rotation can be expressed as:

$$\begin{bmatrix} 0 \\ \vec{v}' \end{bmatrix} = q \cdot \begin{bmatrix} 0 \\ \vec{v} \end{bmatrix} \cdot q^{-1} = \begin{bmatrix} 1 & 0^T \\ 0 & R_q(q) \end{bmatrix} \begin{bmatrix} 0 \\ \vec{v} \end{bmatrix} \quad (13)$$

Where the transformation from l to m is given by matrix:

$$R_q(q) = \begin{bmatrix} q_0^2 + q_1^2 + q_2^2 + q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & q_0^2 - q_1^2 + q_2^2 + q_3^2 \end{bmatrix} \quad (14)$$

Moreover, the composition of rotations can be expressed just as with rotation matrices. Assuming two consecutive rotations, q_1 and q_2 , the composition of such rotations is provided by the matrix:

$$R_q(q_1 \cdot q_2) = R_q(q_1)R_q(q_2) \quad (15)$$

Lastly the *quaternion time derivative* can be defined using the following relation:

$$\dot{q} = \frac{1}{2} \mathbf{\Omega} \mathbf{q} = \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ \mathbf{q} \end{bmatrix} \quad (16)$$

Where $\mathbf{\Omega} = [p \ q \ r]^T$ represents the vehicle body rates and the value assumed by the quaternion can be estimated by integrating equation (16).

Despite quaternions are simpler than Euler angles, are numerically stable and more efficient in terms of computing implementations, in this thesis the Euler angles method have been adopted to describe the attitude of the vehicle because of its easy visualization properties.

2.2 – Quadrotor Mathematical Model

Mathematical model describes quadrotor movement and behaviour with respect to the inputs values of the model and external influences on the system. It can be seen as a function that traces the inputs and the outputs of the system. The idea behind the definition of a *mathematical model* is to make possible the development of a control system. If the model is too complex, the design of the controller will be arduous. Depending on the requirements, it is necessary to find a compromise between model complexity and accuracy. Because of quadrotor nonlinear nature, some assumptions are made in the modelling process. This project results are based on some simplifications concerning quadrotor structure and other factors without losing precision in the description of motion. The simplifications adopted are listed below:

- The CoG of the quadrotor is assumed to be at the centre of the body, coincident with the origin of the *body* reference frame;
- The effects of Earth rotation are assumed negligible;
- The vehicle and its components are rigid bodies fixed to each other;
- The structure is symmetrical;
- Aerodynamic forces are assumed to be negligible;
- No gust disturbance, air fluctuations and ground effect are considered;

The introduction of these simplification hypothesis scales down problem complexity, however difficulties are still present due to the non-linearities of the model and the control challenges of an under-actuated system.

2.3 – Quadrotor Dynamics: Forces and Moments

Normally two approaches are used to develop the mathematical model of an aerial vehicle: the *Newton-Euler Equations* or the *Lagrangian Equations*. In this thesis a procedure similar to (Brian L Stevens, 2015) is adopted. The forces and moments that act on a quadcopter are mainly due to the gravity and the four propellers. Each propeller is responsible of the generation of an upward force F and a torque τ . Thus, the total force acting on a quadrotor is given by the sum of all the forces and the same principle applies for the torques.

$$F = \sum_{i=1}^4 F_i \quad \tau = \sum_{i=1}^4 \tau_i \quad (17)$$

Considering the *forces*, the model of the quadrotor is obtained making use of *Newton's Second Law*:

$$F_{Body} + R_{NED}^{Body} mg = \frac{d}{dt} m v_{Body} \quad (18)$$

Where the total forces acting on quadcopter's body, expressed in *Body* frame are included in F_{Body} , m is the mass of the vehicle, g is the gravitational acceleration and v_b is the vector of linear velocities of quadcopter's CoG expressed in *Body* frame. According to the *Coriolis Theorem*, the derivative of a vector with reference to a different frame can be defined as:

$$\frac{d}{dt} v_{Body} = \dot{v}_{Body} + \omega_{Body} \times v_{Body} \quad (19)$$

Thus, applying this definition to Newton's Second Law it is obtained:

$$F_{Body} + R_{NED}^{Body} m g = \dot{m} v_{Body} + m \frac{d}{dt} v_{Body} = 0 + \dot{v}_{Body} + \omega_{Body} \times v_{Body} \quad (20)$$

Where the term $\dot{m} v_{Body} = 0$ since the variation of mass is significant only when dealing with varying-mass systems along the mission. The vectors v_{Body} and ω_{Body} are defined as:

$$v_{Body} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad \omega_{Body} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad (21)$$

On the other hand, *moments* are obtained as:

$$T_{Body} = \frac{d}{dt} H_{Body} = \frac{d}{dt} (J \omega_{Body}) \quad (22)$$

Where T_{Body} is the total torque acting about the *CoG*, and $H_{Body} = J \omega_{Body}$ is the *angular momentum* vector of the vehicle, both expressed in the *Body* frame. Introducing the *Inertia matrix*⁴ of a rigid body J , it is possible to rewrite the moments equation leaving alone ω_{Body} , results in the following equation:

$$\dot{\omega}_{Body} = -J^{-1} (\omega_{Body} \times (J \omega_{Body})) + J^{-1} T_{Body} \quad (23)$$

Once obtained the dynamics of the quadcopter, the *state space* representation is provided.

2.4 – State-Space Model Representation

As introduced in *Paragraph 2.1* the quadrotor is an under-actuated nonlinear system, its model can be described using a *differential equation* of the type:

$$\dot{x} = f(x) + g \cdot u \quad (24)$$

Where the vector of states is $x \in \mathbb{R}^{12}$ and the vector of control inputs is $u \in \mathbb{R}^6$. The states and inputs components used in this thesis are provided hereafter:

$$x = [p_N \ p_E \ h \ \phi \ \theta \ \psi \ u \ v \ w \ p \ q \ r]^T \quad u = [F_x \ F_y \ F_z \ \tau_x \ \tau_y \ \tau_z]^T \quad (25)$$

Using the rotation matrix R_{Body}^{NED} , the derivative of quadrotor position in the *inertial* frame is determined:

$$\begin{bmatrix} \dot{p}_N \\ \dot{p}_E \\ \dot{h} \end{bmatrix} = R_{Body}^{NED} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (26)$$

The derivatives of the Euler angles can be expressed as:

⁴ The *Inertia* matrix contains the moments of inertia and the cross-products of inertia. Due to the assumption of a symmetrical body with respect to the x-z plane, the only terms to be considered J_{xx}, J_{yy}, J_{zz} and J_{xz} .

$$J = \begin{bmatrix} J_{xx} & -J_{xy} & -J_{xz} \\ -J_{xy} & J_{yy} & -J_{yz} \\ -J_{xz} & -J_{yz} & J_{zz} \end{bmatrix} = \begin{bmatrix} J_{xx} & 0 & -J_{xz} \\ 0 & J_{yy} & 0 \\ -J_{xz} & 0 & J_{zz} \end{bmatrix}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} p + \tan \theta (q \sin \phi + r \cos \phi) \\ q \cos \phi - r \sin \phi \\ 1 \\ \cos \theta (q \sin \phi + r \cos \phi) \end{bmatrix} \quad (27)$$

Using the aforementioned Newton's Second Law, the derivative of vector v_{Body} is defined as:

$$\dot{v}_{Body} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} -qw + rv - g \sin \theta \\ pw - ru + g \sin \phi \cos \theta \\ -pv + qu + g \cos \phi \cos \theta \end{bmatrix} + \frac{1}{m} \begin{bmatrix} F_N \\ F_E \\ F_D \end{bmatrix} \quad (28)$$

In conclusion, using the equation derived in (23) the angular acceleration in *Body* frame is written as:

$$\dot{\omega}_{Body} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} q(rc_1 + pc_2) \\ prc_5 - c_6(p^2 - r^2) \\ q(pc_8 - rc_2) \end{bmatrix} + \begin{bmatrix} c_3\tau_x + c_4\tau_z \\ c_7\tau_y \\ c_4\tau_x + c_9\tau_z \end{bmatrix} \quad (29)$$

Where, according with the hypothesis of symmetry, assuming $\Gamma = J_x J_y - J_{xz}^2$ the coefficients c_i are defined as:

$$\begin{aligned} c_1 &= \frac{J_{xz}(J_x - J_y + J_z)}{\Gamma} & c_2 &= \frac{J_{xz}(J_x - J_y + J_z)}{\Gamma} & c_3 &= \frac{J_z}{\Gamma} \\ c_4 &= \frac{J_{xz}}{\Gamma} & c_5 &= \frac{J_z - J_y}{\Gamma} & c_6 &= \frac{J_{xz}}{\Gamma} \\ c_7 &= \frac{1}{J_y} & c_8 &= \frac{J_{xz}^2 + J_x(J_x - J_y)}{\Gamma} & c_9 &= \frac{J_x}{\Gamma} \end{aligned} \quad (30)$$

Lastly, the matrices of the *state space representation* are provided:

$$f(x) = \begin{bmatrix} R_{Body}^{NED} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \\ p + \tan \theta (q \sin \phi + r \cos \phi) \\ q \cos \phi - r \sin \phi \\ \frac{1}{\cos \theta} (q \sin \phi + r \cos \phi) \\ -qw + rv - g \sin \theta \\ pw - ru + g \sin \phi \cos \theta \\ -pv + qu + g \cos \phi \cos \theta \\ q(rc_1 + pc_2) \\ prc_5 - c_6(p^2 - r^2) \\ q(pc_8 - rc_2) \end{bmatrix} \quad g = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1/m & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/m & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/m & 0 & 0 & 0 \\ 0 & 0 & 0 & c_3 & 0 & c_4 \\ 0 & 0 & 0 & 0 & c_7 & 0 \\ 0 & 0 & 0 & c_4 & 0 & c_9 \end{bmatrix} \quad (31)$$

2.5 – Quadrotor Linearized Model

Once obtained quadrotor's mathematical model, a linearization is performed for subsequent controller design. From the state space representation, the linearization is obtained around an equilibrium point of the system, \bar{x} . In order to define a linear model, the following equation must be fulfilled:

$$f(\dot{x}, x, u) = 0 \quad \text{with} \quad \dot{x} \equiv 0; u \equiv 0 \text{ or constant} \quad (32)$$

Considering the *nonlinear* nature of the equations defined in $f(x)$, an approximation is introduced, assuming small variations of the angles it is possible to use: $\sin \alpha \approx \alpha$ and $\cos \alpha \approx 1$. Thus, the equations become:

$$\begin{aligned}
\dot{p}_N &\approx u + v(\phi\theta - \psi) + w(\theta + \phi\psi) \\
\dot{p}_E &\approx w\psi + v(\phi\theta\psi + 1) + w(\theta\psi - \phi) \\
\dot{h} &\approx -h\theta + v\phi + w \\
\dot{\phi} &\approx p + \theta(q\phi + r) \\
\dot{\theta} &\approx p + \theta(q\phi + r) \\
\dot{\psi} &\approx (q\phi + r) \\
\dot{u} &\approx -qw + rv - g\theta + 1/m F_N \\
\dot{v} &\approx pw - ru + g\phi + 1/m F_E \\
\dot{w} &\approx -pv + qu + g + 1/m F_D \\
\dot{p} &\approx q(rc_1 + pc_2) + c_3\tau_x + c_4\tau_z \\
\dot{q} &\approx prc_5 - c_6(p^2 - r^2) + c_7\tau_y \\
\dot{r} &\approx q(pc_8 - rc_2) + c_4\tau_x + c_9\tau_z
\end{aligned} \tag{33}$$

2.5.1 – Equilibrium Point and Linearization

By setting $\dot{x} = 0$, providing the appropriate substitutions in (33), the *equilibrium point* \bar{x} , and the corresponding set of *control inputs* \bar{u} are obtained:

$$\begin{aligned}
\bar{x} &= [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T \in \mathbb{R}^{12} \\
\bar{u} &= [0 \ 0 \ mg \ 0 \ 0 \ 0]^T \in \mathbb{R}^6
\end{aligned} \tag{34}$$

That having been said, the *linearized model* can be written using matrices A, B, C and D :

$$\begin{aligned}
\dot{x} &= Ax + Bu \\
y &= Cx + Du
\end{aligned} \tag{35}$$

Where the matrices A, B, C and D are respectively the state matrix, the input matrix, the output matrix, and the feedthrough matrix, defined as follow:

$$A = \frac{\partial f(\bar{x}, \bar{u})}{\partial x} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{36}$$

$$B = \frac{\partial f(\bar{x}, \bar{u})}{\partial u} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1/m & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/m & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/m & 0 & 0 & 0 \\ 0 & 0 & 0 & c_3 & 0 & c_4 \\ 0 & 0 & 0 & 0 & c_7 & 0 \\ 0 & 0 & 0 & c_4 & 0 & c_9 \end{bmatrix} \tag{37}$$

$$C = eye(12,12) = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \dots & 0 & 1 \end{bmatrix} \quad (38)$$

$$D = zeros(12,6) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (39)$$

The last two matrices have been chosen considering the absence of feedthrough signals and the output as the states of the system. Both of them have been defined according to the built Simulink model.

2.5.2 – Controllability and Observability

The dual concepts of *controllability* and *observability* are fundamental in control system theory and consequently need to be investigated in more details. According to Aguirre (Aguirre, 2016) a system is *state controllable* if there exists a control u that can drive the system from an arbitrary initial state $x(t_i = 0)$ to an arbitrary final state $x(t_f)$ in a finite time. This property measures the effect of the control input over the state variables and can be determined using the rank of matrix C , defined as follow:

$$C = [B \quad AB \quad A^2B \quad \dots \quad A^{n-1}B] \quad (40)$$

On the contrary a system is *state observable* at a time t_f if the initial state $x(t_i = 0)$ can be uniquely determined from the knowledge of a finite time history of the input $u(\tau)$ and output $y(\tau)$. This property points out the capability of determining the states of a system by measuring its outputs over a limited time interval and can be determined using the rank of matrix O , defined as:

$$O = [C^T \quad A^T C^T \quad (A^T)^2 C^T \quad \dots \quad (A^T)^{n-1} C^T] \quad (41)$$

In this project, both matrix C and O have been computed and their ranks have been evaluated using MATLAB 2018b. The results show that both the matrices are full rank; thus, the quadrotor system is completely *controllable* and *observable*.

Quadrotor Design

3.1 – Quadrotor Anatomy

The majority of commercial quadrotor are designed with the aim of doing aerial photography. Despite their advantages, this kind of quadcopters have closed-source avionics, thus they cannot be used in research applications. There are many research activities based on open-source projects for quadrotors development; the aims of these studies are tailored for the development and testing of flight control, state estimation and navigation algorithms. According to Bangura (Moses, February 2017), the idea of open-source UAVs was introduced thanks to the ‘*AscTec Hummingbird Project*’, a research activity conducted by the *General Robotics, Automation Sensing and Perception* (GRASP) laboratory and the *Flying Machine Arena*. This work has led to the development of a unified communication protocol called *MAVLink*⁵ Meier, in 2009.

According to Khosiawan and Nielsen (Yohanes Khosiawan, 2016), there are three elements to be considered in employing UAVs for a certain application domain: *tasks*, *environment*, and *UAV Operation System* (UAV OS). Tasks consist of a series of action that UAVs must fulfil in an efficient way without compromising its structure or surrounding environments; in this work the main task is indoor navigation. Environment is the surroundings and infrastructures where the UAV will be employed; in this project an indoor environment is considered. UAV OS includes different components such as imaging devices, sensors, recharge centres and other resources which are fundamental for the UAV operations; the UAV OS of this project is provided in the following paragraphs.

This chapter outlines the different hardware (HW) and software (SW) components that have been used in the development of the real-world quadrotor platform of this work. The quadrotor is built using entirely Commercial-Off-the-Shelf (COTS) components which ensures the replacement of any of the HW components for any of the dynamics levels. A typical quadrotor configuration consists of an airframe (*cross* or *plus* configuration), four brushless direct-current motors (BLDC), four propellers, one or more batteries, a flight control board (which is part of the avionic system), a companion computer and four Electronic Speed Controllers (ESCs) that are used to regulate the thrust produced by each of the four BLDC. In addition to the previous elements, quadrotors can be equipped with a transmitter-receiver system and additional sensors as *local positioning system* for indoor application or *global positioning system* (GPS) for outdoor applications.

⁵ MAVLink ‘*Micro Air Vehicle Communication Protocol*’ is a messaging protocol for communicating with small UAVs and between their onboard components. It follows a modern *hybrid publisher-subscriber* and *point-to-point* design pattern. All data streams are published as *topic* while configuration subprotocols are point-to-point with retransmission. Messages are defined within XML files; each file defines the message set supported by a peculiar MAVLink system.

3.1.1 – Airframe: Holybro X500 Frame

The airframe is an essential component of a quadcopter (and more in general of any UAVs). The torques generated by the motor systems, the landing impacts and other external factors make the frame vital in terms of design and maintenance. Furthermore, the frame must be as light as possible, to increase the possible payload, and as robust as possible, for facing high vibrations. Most available materials for frame are carbon fiber, wood, aluminium, plastic & PVC, and fiberglass.

In accordance with the purpose of indoor autonomous flight, it has been chosen the commercially available *Holybro X500* airframe kit. The *X500* frame is a full carbon fiber airframe produced by *Holybro*. As shown in the following figures, the kit (shown in Figure 21) consists of two carbon fiber landing gear with plastic connectors, a battery mounting board (shown in Figure 22), a payload platform board (shown in Figure 23) and two carbon fiber plates with a thickness of 2 [mm] (shown in Figure 24). It has been chosen a *cross* chassis with a wheelbase⁶ of 500 [mm] and a weight of about 470 [g]. The choice of that airframe was driven by its perfectly compatibility with *Pixhawk 4* and *Pixhawk 4 mini* autopilot.



Figure 21: photo of kit Holybro X500 frame contents.



Figure 22: representation of Holybro battery mounting board.



Figure 23: representation of Holybro payload platform board.



Figure 24: representation of Holybro X500 complete assembly.

3.1.2 – Control Module: Pixhawk 4 Mini and PX4

Avionics systems applied into UAVs were adapted from standard aviation avionics (the majority of them are similar or exactly the same). Most of commercially available quadrotors are characterized by their own avionics with closed-source firmware and software. One of the main avionics systems is the *autopilot* which is used to control the attitude, position, and trajectory of UAVs. Autopilots can be semi-automatically, which means that commands from remote pilot are needed, or fully automatically. Some of the most popular

⁶ The multirotor frame size is defined as the distance from opposite corner motors.

avionics boards currently in use are shown in Table 2: *Ardupilot*, *AeroQuad*, *Openpilot*, *Paparazzi*, *Pixhawk* and *Mikrokopter*. An avionic board can be divided into the software part and the hardware part. The SW is the brain of the vehicle, and it is used to process and send information, while the HW part includes components such as the Inertial Measurement Unit (IMU), barometers, microcomputers, GPS receiver and interfaces elements which are used as bridge connection between on-board items.

Table 2: open-source autopilots of multicopters.



Figure 25: Ardupilot Mega 2.0



Figure 26: AeroQuad flight controller board

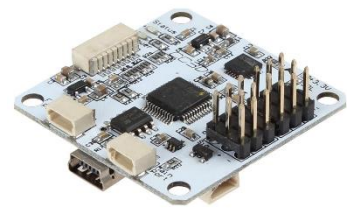


Figure 27: OpenPilot CC3D flight controller



Figure 28: PaparazziUAV Lisa/M v2.0



Figure 29: Pixhawk 4 Mini flight controller

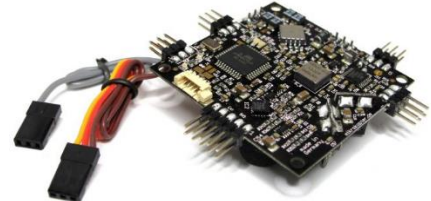


Figure 30: Mikrokopter v2.5 flight controller

All the avionics boards are developed with a *Ground Control Station* (GCS) software in order to visualize telemetry data and to control the vehicle. There are a lot of free open-source GCS software such as: *Mission Planner* (Ardupilot), *Openpilot*, *Paparazzi* and *QGroundControl* (PX4). Most of the avionics boards come with micro-electro-mechanical-system (MEMS) gyroscopes, accelerometers, and magnetometers to measure the attitude as well as determine the position. The autopilot, also referred to as flight controller (FC), processes measured data for either stabilizing the UAV and hold the position in hover state or flying predefined paths. In this work a *Pixhawk 4 Mini* autopilot running the PX4 open-source software (*Dronecode Foundation*, San Francisco, CA, USA) has been used. The PX4 has been selected because of its *modular* architecture which is built on the *NuttX* real-time operating system, simplifying the difficulties of adding new sensors, peripherals, expansion modules and the development of new state control algorithms. Due to its properties, PX4 is a great testing environment for research purposes.

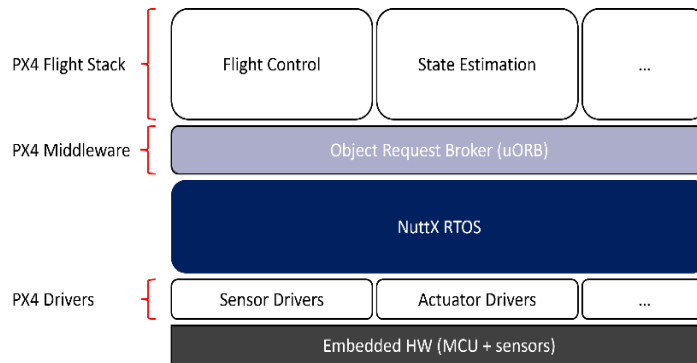


Figure 31: PX4 software structure from (Meier, 2015).

As shown in Figure 31, the PX4 flight control framework consists of three main layers: PX4 *flight stack* (containing individual applications such as the FC, state estimation and others), PX4 *middleware* (containing communications between the applications and the driver) and PX4 *drivers* (architect specific). The lower half of the scheme manages device *drivers* required for a particular microcontroller or bus type. Moreover, this half includes a simulation layer, that enables the PX4 flight code to run on a desktop operating system and control a modelled vehicle in a simulated environment, providing Software-In-the-Loop (SITL) simulations. The PX4 *flight stack* is a collection of guidance, navigation, and control algorithm for autonomous flight; it includes controller for VTOL and fixed-wing vehicles in addition to attitude and position estimators.

That having been said, the core part of the PX4 is the control block, which represents a control algorithm that calculates and sends control signals to the propulsion module, relying on sensors data and states estimation. A schematic representation of PX4 controller is provided in Figure 32. The inputs to the control system are the reference states, referred to as η_{ref} , RC signals and sensors readings η . The *telemetry module* is used for two-way communication between the aircraft and the base station while the *system monitoring and logging block* is used to store flight data like aircraft status, battery status and propulsion status.

Figure 32: PX4 flight stack schematic representation.

Figure 33: illustration of Pixhawk 4 Mini flight controller.

The *position and attitude estimator* receive sensors readings and combines them using an algorithm, the *Extended Kalman Filter* (EKF) to estimate the vehicle states. These states are used by the *navigator* and the *controller module*. Both position and attitude controllers, receive the estimated states and the RC input desired by the user, via the radio receiver connected to the Pixhawk. The controller output commands which are then converted into motor commands for each motor. Lastly the obtained commands are sent to the ESCs of each motor to regulate the speed of each propeller as desired.

In conclusion a brief overview of *Pixhawk 4 Mini* autopilot characteristics (shown in Table 3) is provided. *Pixhawk 4 Mini* have been developed in order to provide the same performance of *Pixhawk 4* while eliminating interfaces that are usually unused; this allows smaller drones applications (it can fit in a 250 [mm] racing drone). It is based on the *FMUv5* design standard and is optimized to run PX4 flight control software. Table 3 outlines autopilot main characteristics:

Table 3: main characteristics of *Pixhawk 4 mini*, as taken from '*Pixhawk 4 Mini Technical Data Sheet*' (Foundation, 2018).

Processor	32 Bit Arm Cortex-M7, 216 MHz, 512 KB RAM
On-board Sensors	Two redundant IMUs (accels and gyros) One magnetometer IST8310 One barometer MS5611
Interfaces	Eight PWM outputs Four dedicated PWM/capture inputs on FMU One dedicated R/C input for CPPM, Spektrum/DSM and S. Bus Three general purposes serial port Two I2c ports Three SPI buses One CANBuses for CAN ESC Analog inputs for voltage / current of battery Two additional analog input
Weight and Dimensions	Weight of 37.2 [g] Dimensions: 38 x 55 x 15. [mm]

3.1.3 – Propulsion Module: Motors and Propellers

As introduced in *Chapter 2*, the control of a multirotor UAVs is achieved using the differential thrust produced by individual's rotors. The use of a compact, lightweight, and efficient propulsion system is one of the main technological requirements for a long endurance electric UAV. Currently, the most popular trend is to use battery sources which provide direct current to operate the electric power module. Electric motors operate on *Faraday's Law* of electromagnetic induction. Calling *emf* the electromotive force induced in an inductor coil, the module of the force is directly proportional to the rate of change of the flux linkage, thus, to create *emf* a magnetic field and a moving coil are needed. The currently available types of DC motors include *stepper*, *brushed* and *brushless*. Brushless DC motor (BLDC) is a permanent magnet electric motor with an electronic commutation system; electromagnets (armature) are located on the stator while permanent magnets are located on the rotor. Compared to conventional direct current DC motors, BLDC have higher efficiency, increased reliability, and higher torque by weight. Because of their better characteristics and superior performances, they have rapidly gained popularity; moreover, they can provide information about motor poles, thus, the speed of the motor-rotor system, which is essential in control algorithms design. Despite their advantages, DC motors have a drawback due to the conversion of the DC voltage into three phase voltages, which is accomplished by expensive electronics, known as *Electronic Speed Control* ESCs.

As shown in Figure 34, the size of BLDC motors is normally given by a four-digit number, where the first two digits stands for the *stator width*, while the last two digits for the *stator height*.



Figure 34: illustration of stator width and height of a BLDC motor.

Taller stators have larger surface; therefore, they can cut through more magnetic field, increasing overall thrust generation; for this reason, taller stators are preferred for high rpm and fast drones. On the other hand, a wider stator allows a larger bearing, improving efficiency and smoothness of the motor.

The *rotational speed*, defined as *revolutions-per-minutes* (rpm), and the *torque* are other two characteristics to take into consideration in the selection of a BLDC motor. Typically, the parameter specified by the manufacturer is the *KV*, which is defined as the increase of brushless motor's rpm when the voltage goes up by one volt $1[V]$ without load. This parameter is evaluated without the propellers on; once mounted the propellers, the rpm decreases due to air resistance. Generally speaking, heavier quadrotors pair with medium to low *KV*, while lighter quadrotor usually use high *KV* motors⁸. By pairing a high *KV* motor with an excessively large propeller, the motor tries to produce the required torque drawing more current, subsequently generating too much heat which could lead to motor overheats and, in the worst cases, electrical shorts inside the motor.

Aircraft *propellers* are characterized by three main parameters: *diameter*, *pitch*, and the *number of blades*. By rotation of propellers, the aerodynamic forces and moment generated, directly affect the dynamics of the multirotor. Assuming fixed pitch propellers, the required aerodynamic forces and torques are achieved by changing the rpm of the rotor. Generally speaking, increasing the number of blades, the pitch angle or the diameter, results in a larger amount of induced airflow, hence greater thrust force. On the other hand, greater resistance to rotation is exerted, hence motor torque and power consumption increases. That having been said, the thrust force of the single rotor is defined as:

$$F_i = k_F \omega_i^2 \quad \text{with} \quad k_F = C_T \rho A r^2 \quad (42)$$

Where ω_i is the angular velocity of the i -th rotor and k_F is the thrust force factor expressed in Ns^2 . The thrust force factor depends on the propeller's geometry (A propeller disk surface and r radius), air density ρ and propeller thrust coefficient C_T . Both *multi-bladed* and *two-bladed* propellers are good solutions, however two-bladed propellers are more common due to their higher aerodynamic efficiency. Assuming the air

⁸ $KV > 2000$ is mainly used to propel micro and small aircraft, $2000 \geq KV \geq 200$ is used to propel aircraft intended for photography or similar tasks where cargo masses are below $10 [Kg]$ and $KV < 200$ are intended for heavier payload.

density to be constant and the absence of turbulence and wind gust, the thrust, torque, and power depend only on propeller speed.

Table 4 presents a general guideline for selection of BLDC motors and propellers based on components characteristics which are often provided by manufacturer. However, it must be emphasised that this is not a rule; depending on applications fields different couplings are also possible.

Table 4: guideline table for motors and propellers selection.

Frame Size	Propellers Size	Motor Dimensions	KV
150 [mm]	≤ 3 "	$\leq 1105 - 1306$	≥ 3000 [KV]
180 [mm]	4 "	1806	2600 – 3000 [KV]
210 [mm]	5 "	2204 – 2208, 2306	2300 – 2600 [KV]
250 [mm]	6 "	2204 – 2208, 2306	2000 – 2300 [KV]
350 [mm]	7 "	2208	1600 [KV]
450 [mm]	≥ 8 ", 9", 10 "	≥ 2212	≤ 1000 [KV]

Once introduced motors and propeller's properties, the *thrust-to-weight ratio* is defined. The higher the thrust, the easier is to control your drone in elaborate acrobatics. Racing drones, also referred to as *First Person View* (FPV), can be equipped with motors of 10: 1 or 13: 1 thrust-to-weight ratio, because of their application in competitive race where manoeuvrability and speed are essential. However, high thrust-to-weight ratio required expert pilot due to the reduced stability of the system; therefore, quadrotors are commonly equipped with motors between 3: 1 to 4: 1 thrust-to-weight ratio.

In accordance with the guidelines of Table 4 and the weight of the modelled quadrotor 1.4 [Kg], the following components have been chosen: *Goolsky A2212 1000KV Outrunner Motors* and *Goolsky 1045 propellers*.

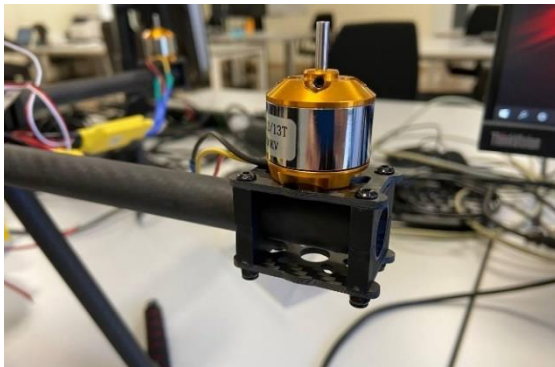


Figure 35: photo of Goolsky A2212 Outrunner Motor installed on Holybro X500 frame arm.



Figure 36: photo of complete propulsion module: Goolsky 1045 propellers and Goolsky A2212, installed on Holybro X500 frame.

3.1.4 – Propulsion Module: Electronic Speed Control ESCs

Electronic Speed Controllers (ESCs) are electronic circuit provided to control the rotational speed and the direction of rotation of BLDC motors. The ESC consists of a *microcontroller* that processes the input PWM signal and *switching transistors*. The ESC implements the proper sequence of switches to energize particular phases of a motor, achieving continuous rotation, thus desired rpm set by the input from the flight controller.

The main parameter for selecting the ESC is the maximum allowed *current*, which is defined as the *current rating*, measured in [A]. Motors draw current when spinning, hence if they draw more current than ESC can

handle it will start to overheat and eventually fail. Generally, a maximum allowable current must be from 20% to 50% higher than the maximum motor current, in order to avoid motors overheating or failure.

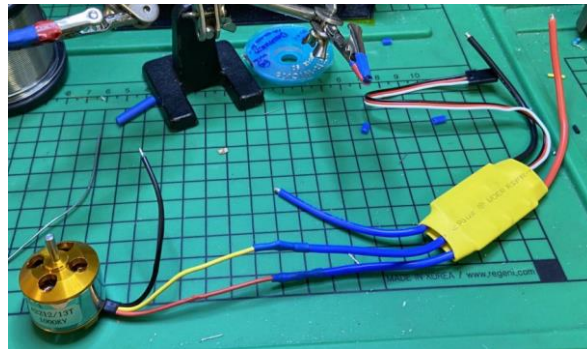


Figure 37: wire welding procedure of BLDC motor Goolsky A2212 1000KV and HP 30A ESC.

3.1.5 – Energy Module: LiPo Battery

It is worth mentioning that the main limitation of UAVs is their flight time, which is bound by battery capacity, which is in turn limited by the size and weight of the UAVs. According to Mandel, Milford, and Gonzalez (Nicolas Mandel, A Method for Evaluating and Selecting Suitable Hardware for Deployment of Embedded System on UAVs, 2020), over 90% of power is consumed by the motors supplying thrust; however, the influence of computational capabilities on power is also to be considered. According to Boroujerdian and Genc (Behzad Boroujerdian, 2018), faster computation capabilities can greatly reduce total power consumption for the same task due to reduced time spent at hover and lower accelerations; studies confirm that an efficient system design can improve battery endurance. By increasing processing speed to 5 times, drone's energy consumption can be reduced by close to 4 times. As an example, assuming a small quadrotor, a 4-cell lithium-polymer (4S LiPo) battery with a capacity of 2200 [mAh] gives an average flight time of 10 minutes which can be improved to 30 minutes by increasing processing speed.

Battery is also a demanding element concerning UAVs weight. In accordance with the work of Kumar and Michael (V. Kumar, 2012), the battery unit accounts for between 25% to 50% of the gross weight of a quadcopter. Figure 38 shows the weight distribution of the realized quadcopter; as can be seen, the battery source, *Youme 3S LiPo*, is the second heaviest only to the main *Holybro X500* carbon-fiber frame.

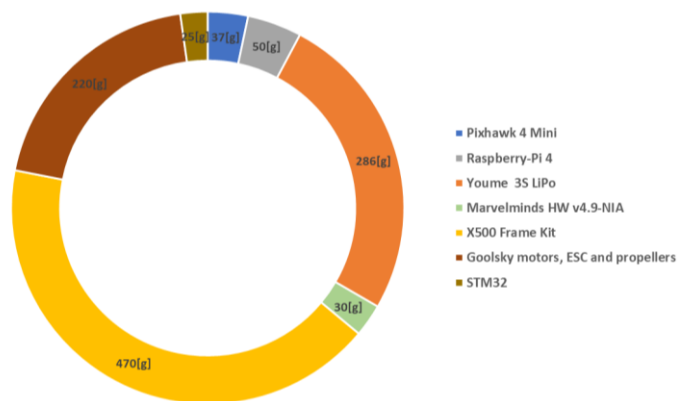


Figure 38: pie chart showing the weights distribution in grams of the realized prototype quadcopter.

In accordance with (Marcin Biczyski, 2020), *Lithium Polymer (LiPo)* batteries currently dominate the market of lightweight aerial vehicles thanks to their high energy density and high current discharge capabilities. These batteries consist of several cells, connected in series with a voltage that changes according to the state of charge of the battery. In nominal condition cell voltage is about 3.7 [V]. The cells can be connected in series or in parallel, denoted respectively by *S* or *P*. Furthermore, batteries are characterized by *capacity*, expressed in [mAh] and a *C-rating*. Battery capacity is defined as the maximum amount of energy that can be extracted from the battery under certain specified conditions. The most common measure of battery capacity is [Ah], which is the number of hours for which a battery can provide a current equal to the discharge rate at the nominal voltage of the battery. On the other hand, the *C-rating* is the measurement of the current in which a battery is charged and discharged; as an example, a battery with 50 [C] and 1300 [mAh] has a maximum discharge current of $50 \cdot 1.3 = 65$ [A].

In this project a 3S *LiPo* battery with a nominal voltage of 11.1 [V], a capacity of 3300 [mAh] and a *C-rating* of 50 [C] has been chosen. Recalling what said in *Paragraph 3.1.3*, the maximum motors speed in no-load conditions can be calculated from the *KV* parameter multiplied by the applied voltage:

$$\omega_{No\ Load} = KV \cdot V \quad (43)$$

This means that the higher the voltage, the faster the motor spin. In this work a *LiPo* battery with a nominal voltage of 11.1 [V] and four motors with 1000 *KV* have been considered, thus, a theoretical value of 11100 [rpm], in no-load conditions, is obtained for each motor.



Figure 39: illustration of 3S *LiPo* power battery Youme, nominal voltage 11.1[V], capacity of 3300[mAh] with a *C-rating* of 50[C].

3.1.6 – Sensors and Indoor Positioning System

A suite of *sensor system* is required to provide quadrotor with position and attitude information that are vital in autonomous flight. In order to design the control system, it is needed to collect certain data linked to the flying behaviour of the quadcopter. These data include the position, the altitude, the linear accelerations, and angular rates that can be provided by the Inertial Measurement Unit (IMU). IMU consists of gyroscopes and accelerometers that could send data directly either on Euler angles or quaternions. In this thesis the *Euler angles* output is chosen over the *quaternions*, due to the convenience of the data based on the mathematical model of the system. The data coming from the sensors are received by the microcontroller (μC), then they are processed in order to control the behaviour of the quadcopter and the output signal is sent directly to the motor drivers. The inertial measurements packets are also used as a '*heartbeat*' for the avionics and base station, with received packets signifying an active and healthy communication link.

The attitude is sensed by three gyroscopes along with a 3D magnetometer. Moreover, the system is equipped with three accelerometers to measure the acceleration in body fixed coordinates. The height, position and velocity information are provided by an indoor positioning system, which is described hereafter.

Getting position information in an outdoor environment can be done easily using *Global Positioning System* (GPS) as has been done in many previous works, such as (Kayton, 1997). Unfortunately, there is no way to obtain position information of an UAV in an indoor or in a satellite occluded area environment. For this reason, many research have been conducted on the development of a solution to the accurate and reliable location of UAVs in indoor environments such as using a *laser ranger*, *ultrasonic sensors*, *infrared sensors*, and *visual sensors*. For all the solutions proposed above, several experiments have been conducted. As an example, in (Ruijie He, 2008) it was demonstrated the limited range of field of view of *laser range-finder* which can cause vehicle to lose track of its own position in certain configurations and some parts of environment. According to Mustafah, Azman and Akbar (Yasir Mohd Mustafah, 2012), a localization system based on *visual sensors* has a reduced accuracy due to the limited number of cameras that can equipped the UAV and the possibility of data losses during transmission.



Figure 40: Starter Set HW v4.9-NIA includes four stationary Beacons HW v4.9, one mobile Beacon HW v4.9 and one router modem v5.1.

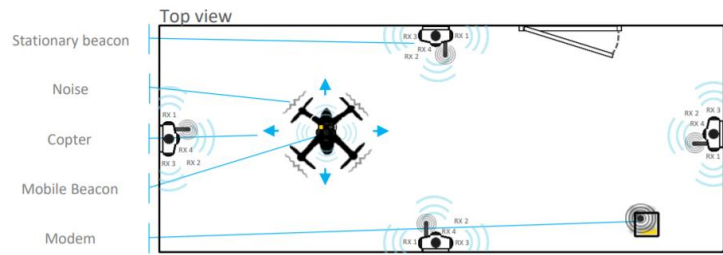


Figure 41: illustration of Marvelmind Set HW v4.9-NIA 4x stationary beacons, 1x mobile Beacon and modem disposition. Reference (Robotics, Indoor "GPS" Autonomous Copter Setting Manual)

In this project, UAV localization has been performed using a set of *ultrasonic sensors*, the *Marvelmind Starter Set HW v4.9-NIA*⁹. The advantage of using a set of ultrasonic sensors is that it can provide accurate information, within a tolerance of $\pm 2[cm]$, without any additional distance measurement sensor. The navigation system is based on *stationary beacons* that are linked through radio interfaces running on a frequency of 915 [MHz]. A central router, connected to the ground station (an octa core *Intel i7* ground laptop, running Windows 10 as operating system), provides the location of the *mobile beacon* using the time delay of ultrasonic signals, also known as *Time-Of-Flight*, and triangulation algorithms.

Table 5: technical details Marvelmind Starter Set HW v4.9-NIA. Reference (Robotics, Marvelmind Starter Set HW v4.9-NIA, 2020)

Distance Between Beacons	Up to 50 [m] in indoor environment
Coverage Area	Coverage similar to cellular networks up to 1000 [m ²]
Location Precision	Absolute 1%-3% of the distance between beacons (differential $\pm 2[cm]$)
Location Update Rate	Can be set manually – nominal 0.05 to 25 [Hz]
Power Supply	Internal LiPo battery 1000 [mAh] – stationary up to 72 [hr], mobile up to 12 [hr]
Weight	59 [g] including battery and housing and antenna
Beacon Size	55 x 55 x 65 [mm] (considering antenna)

⁹ *Marvelmind Indoor Navigation System* offers two SW possibilities, *Non-Inverse-Architecture* (NIA) and *Inverse-Architecture* (IA). The difference is that NIA consists of two or more stationary Beacons and a mobile Beacon transmitting on the same frequency, while the IA the mobile Beacon receives different ultrasonic frequencies at the same time.

3.2 – Quadrotor Assembly: Hardware Architecture

Based on the previous investigations and observations, the architecture of the designed quadrotor system is presented. In accordance with the classification provided in (Yuntian Li, 2018) UAV system can be divided into four parts: *ground station*, *on-board avionics*, *on-board actuators*, and *accessories*.

A *ground station* consists of a ground station HW and SW. The software is typically designed as an application, running on a ground-based computer that communicates with UAV via *wireless telemetry*. It displays real-time data on the UAVs performance and position and can be used to control UAV in flight, uploading new mission's commands and settings parameters. In this project, the *QGroundControl* software has been used. This GCS offers many advantages, it works with MAVLink based autopilot (both *PX4* and *ArduPilot* are supported) and it runs on all platform desktop (Windows, Mac OS X, Linux) and mobile (Android and iOS). The ground station software has been tested on octa core *Intel i7* ground laptop, running Windows 10 as operating system, which communicates with onboard avionics.

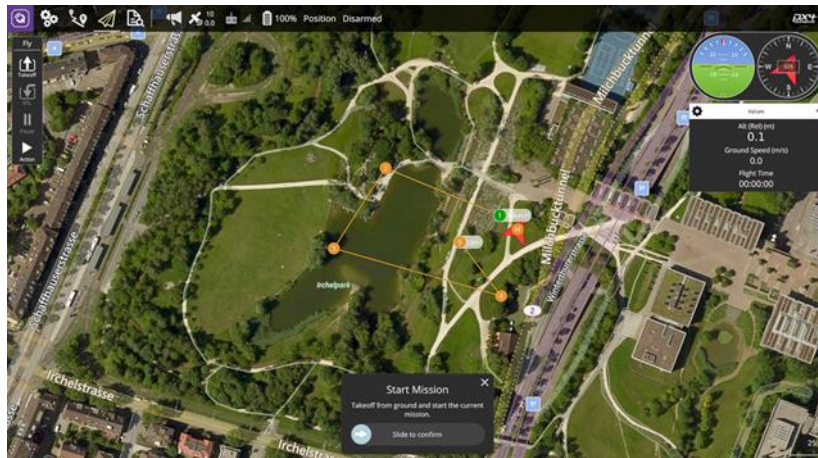


Figure 42: illustration of QGroundControl application user interface running on a Windows 10 operating system with a PX4-based autopilot.

The quadrotor *on-board avionics* and *actuators* consist of the aforementioned (Paragraph 3.1) *propulsion module*, *control module* and *energy module*. The control module includes the autopilot *Pixhawk 4 Mini* and the companion computers, respectively the *Raspberry-Pi 3* and the *Nucleo STM32*. On the other hand, the 3S *LiPo* battery and the power module are classified as part of energy module. The designed quadrotor hardware architecture is shown in Figure 47. Starting from the *energy module*, the 3S *LiPo* battery is directly connected to the power module which consists of the *Power Control Board (PCB)* and the *Battery Eliminator Circuit (BEC)*. The *Power Module Board (PMB)* (shown in Figure 43 and Figure 44) serves the purpose of a power module as well as a power distribution board. It provides regulated power to the autopilot and at the same time it sends information about battery's voltage and current draw to the autopilot. Initially, PDB were very simple, they were made of a thick copper with an input from the battery and multiple outputs. As the need for regulated voltages for various components has become more common, manufacturers have begun including voltage regulators on the PMB so that sensitive components can be reliably powered. The BEC is a voltage regulator which is designed to drop big voltage to smaller ones, allowing the power of on-board devices such as autopilot with a lower voltage without using a separate battery. As can be seen, the propellers are directly connected through the ESC to the PCB (shown in Figure 44) while the autopilot and the *Raspberry-Pi* are powered by a voltage of 5 [V] provided by the BEC.

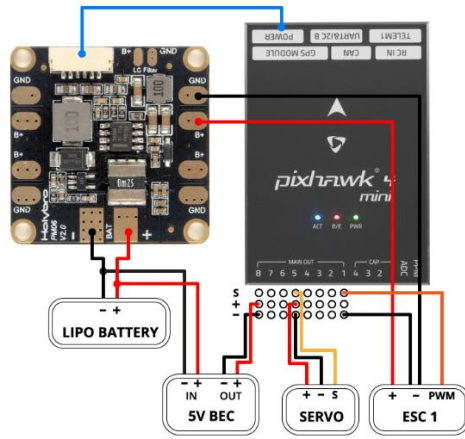


Figure 43: illustration of Pixhawk 4 Mini power wiring scheme. Reference "PX4 User Guide".

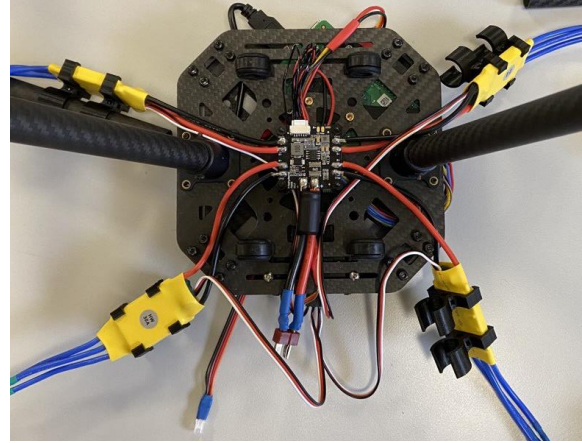


Figure 44: photo of designed quadrotor PMB and ESC wiring connection, installed on X500 frame bottom plate.

The flight controller is the main element of the *control module*. It is used to deal with all the management and control actions like arming, disarming, responding to GCS inputs, adjusting motor speed and so on. Since most data processing and fusing are time-consuming and cannot be handled by normal flight control board, the most suitable solution is to use a companion computer. As can be seen in Figure 47 the *Raspberry-Pi 3* (RPi3) communicates with *Pixhawk* autopilot using *Pulse-Position-Modulation PPM* signals; while the *STM32* connects to autopilot's *GPS1* port, providing data received from a serial connection with *Marvelmind* mobile Beacon. Two algorithms run on the *Nucleo STM32*: the first one is used to read NMEA packets provided by the ultrasonic indoor positioning sensor and convert them in ASHTECH; while the second one displays warnings or failure messages on LCD. The autopilot's *TELEM1* port is used to communicate with the FC using the *MAVLink* protocol. Hence, additional tasks, such as obstacle avoidance, can be performed using algorithms running on *Raspberry-Pi*, sending MAVLink commands directly to *Pixhawk* *TELEM1* port.

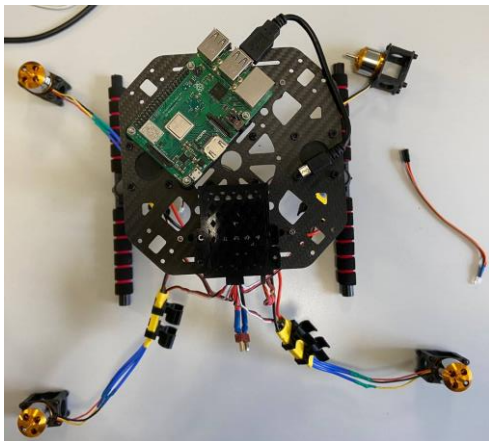


Figure 45: photo of Raspberry-Pi 3 installed on Holybro X500 frame bottom plate through 3D printed support.

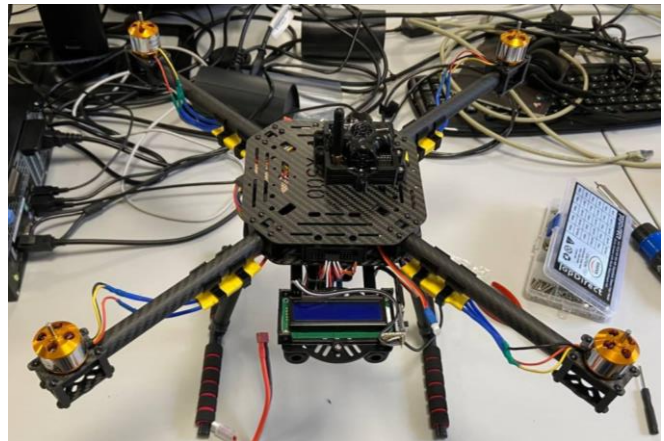


Figure 46: photo of LCD, ESCs and mobile ultrasonic sensor installed on Holybro X500 frame through 3D printed component.

The on-board accessories are part of the *equipment module*, which includes external HW items such as the *Liquid Crystal Display* (LCD) and the remote *joystick* controller. As stated before, LCD screen has been introduced to display warning or error messages that can occur during flight. On the other side, the joystick has been developed as a remote control that communicates with RPi, using USB port, in emergency situation. Joystick's inputs are provided to an algorithm that runs on RPi. The remote controller has a security purpose;

it is designed to send predefined MAVLink messages to autopilot, performing emergency manoeuvres (such as engine shut down or rapid landing), that the user requests, in presence of dangerous situations.

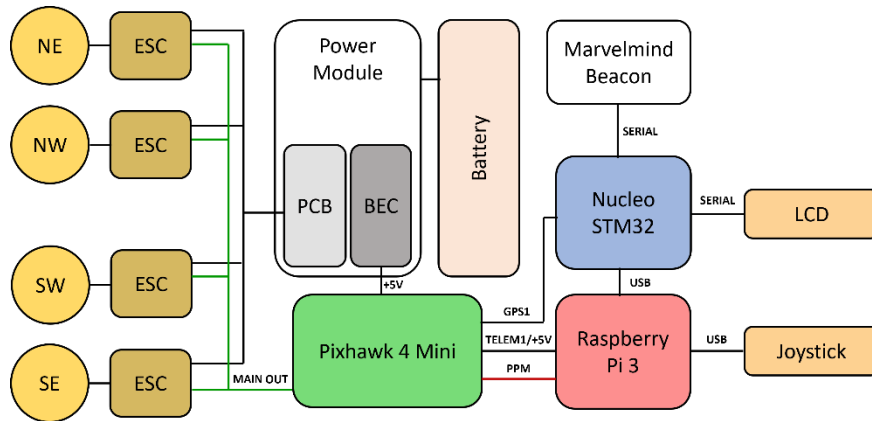


Figure 47: quadrotor hardware architecture block diagram and wiring scheme of connections between power module, propulsion module, autopilot, companion computer, local positioning system and external additional hardware.

3.2.1 – SolidWorks CAD Model and 3D Printed Components

Despite their simple working principle, quadcopters are highly complex vehicles with numerous hardware components and wirings connections that must be integrated in a volume efficient, lightweight working configuration. Moreover, all on-boards avionics must withstand the mechanical loads and the strong vibrations, produced by propellers rotation during flight. These factors turn into harsh assembling requirements. In this project, different commercially available items have been chosen, hence, their installation on the chassis and their wiring connection call for the design of custom chasing and supports that allow stable and reliable but at the same time easily removable connections.

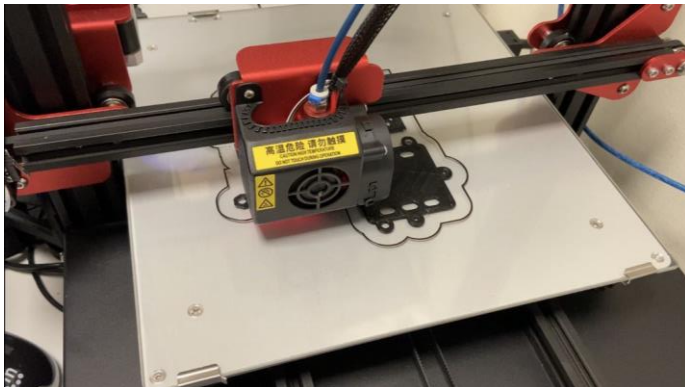


Figure 48: photo of 3D printer during printing of the component Marvelmind mobile Beacon sensor's chase.



Figure 49: 3D printed ultrasonic sensor support installed on Marvelmind mobile Beacon.

Considering the aforementioned problems, *3D printing* seems to be the optimal solution. *3D printing*, also known as *additive manufacturing*, is defined as the process of converting a digital design into a tangible model. The 3D printer creates specified object by depositing thin layers of material on top of the other, all under computer control (shown in Figure 48). According to (R A Navrotsky, 2021), the application of this technology affords ample opportunities for different field of application; as an example, the printing of units

and parts of small UAVs makes it possible to obtain new configurations and to improve existing devices. In this project, 3D printed components have been used to:

- Secure the autopilot *Pixhawk 4 Mini* in place during quadcopter manoeuvres. Two openings along the two opposing sides of the perimeter of the chase enable the wiring connection of the autopilot with the *Power Module Board* and both the companion computer *RPi* and *Nucleo STM32*;

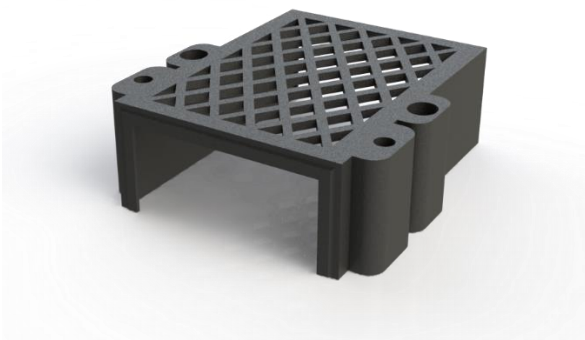


Figure 50: Solidworks® render of designed CAD model of Pixhawk 4 Mini autopilot chase.



Figure 51: photo of Pixhawk 4 Mini secure to Holybro X500 bottom plate through 3D printed chase.

- Secure the ultrasonic local positioning sensor *Marvelmind* mobile Beacon to the upper plate of *Holybro X500* frame. The sensor's chase has been designed in order to damp vibrations during flight and to be easily removable when recharging is needed (shown in Figure 49);
- Secure the *Liquid Crystal Display* LCD to the *Holybro X500* payloads platform board and to allow the connection to the companion computer *Nucleo STM32*;



Figure 52: Solidworks® render of designed CAD model of LCD support for Holybro X500 payload platform.

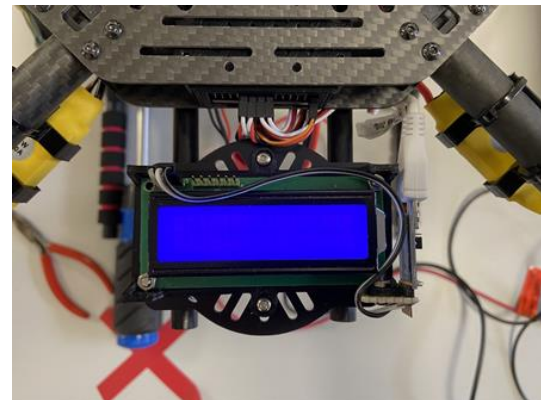


Figure 53: photo of LCD screen installed on Holybro X500 frame payload platform board through 3D printed support.

- Secure the ESCs to the arm of the *Holybro X500* frame, reducing the propagation of vibration.

All the 3D printed components have been manufactured using 3D printable *Poly-Lactic Acid* (PLA) filament. All the chases and supports have been implemented consistent with the *Holybro X500* airframe; hence a complete CAD of the quadrotor frame has been designed in *Solidworks®* (shown in Figure 54).



Figure 54: Solidworks® render of complete designed quadrotor CAD model, including Holybro X500 airframe, BLDC motors and propellers. In the top left front view, in the top right top view and in the bottom perspective view.

3.2.2 – Prototype Quadrotor

Lastly, some photos of the quadrotor final assembly are provided.



Figure 55: on the left front view of quadrotor final assembly; on the right perspective view of quadrotor final assembly.

Guidance, Navigation and Control Algorithm

Regardless UAVs size and mission application, all UAVs share the need for *navigation sensors* and avionics systems which provide an estimate of the vehicle's full state vector. The states normally include three position coordinates, three components of velocity and six components to describe vehicle attitude (Euler angles and angular rates). In addition to states estimation, drones need control and guidance systems to manoeuvre in a way consistent with their mission. The intrinsically unstable nature of UAV necessitates a rigorous approach to the analysis and design of their *Guidance, Navigation and Control* (GNC) techniques. The *guidance system* generates instructions on what state trajectory should be followed, while the *control system* operates the aircraft controls, based on *navigation sensors* estimations, to follow the desired trajectory. According to (Kimon P. Valavanis, 2015), there are four main flight control techniques for UAVs: *Linear Flight Control*, *Nonlinear Flight Control*, *Adaptive Control* and *Robust Control*. This chapter focuses on *linear* control techniques, which are often used to improve UAV performance and reliability. Such techniques are actively being studied to handle nonlinear aerodynamic-kinematic effects, actuator saturations and rate limitations, modeling uncertainties and time varying dynamics. The chapter is organised as follow: firstly, a brief overview of general guidance and navigation algorithm is provided, focusing on the guidance algorithm (*Artificial Potential Field*, APF) used in the obstacle avoidance task; then the control techniques (PD controller, PID controller and MPC controller) developed to follow the designed path are analysed.

4.1 – Guidance Algorithm

In order to achieve autonomous flight in complex indoor environments with multiple obstacles, there is the need of robust and reliable *path planning* algorithm. The term *path planning* refers to get from present location to the desired location with minimum control effort and time. On the other hand, *path following* is the concept of following an already present path with minimum error. In accordance with (U. Orozco-Rosas, 2019) and (T. T. Mac, 2016) many path planning methods have been proposed in literature, such as *iteration method* or *heuristic method*, however both of them need more time for computing to find good path planning. Meanwhile, the *real-time method* does not need iterations and has simple mathematical equations, thus it needs less time for processing.

One of the main *real-time methods* is the *Artificial Potential Field* (APF). The APF algorithm uses the position-based potential functions to modify the quadrotor's velocity as it moves closer to the goal and obstacles. It receives as inputs the current position of the UAV, the obstacle positions and the target position and computes the desired velocities as outputs. As mentioned before, the APF has simple equations, so it needs low computational effort, and it is suitable for real time applications. However, one drawback of APF is the local minima problem which occurs when the sum of the overall forces (attractive and repulsive) acting on the system is null. Most of solutions proposed in literature have faced those problems by modifying the algorithm. An example is provided in (Y. B. Chen, 2016). This section presents the analytical implementation

of a modified APF algorithm for obstacle avoidance applications. Commonly APF consists of two forces, the first one is the *attractive force* F_{att} , while the second is the *repulsive force* F_{rep} . The attractive force is used to navigate the UAV to the desired reference.

Assuming a state vector of two elements $x = [x_q, y_q]^T$, the attractive force is expressed as:

$$\begin{aligned} U_{att} &= \frac{1}{2} k_a (x - x_d)^2 \\ F_{att}(x) &= -\nabla U_{att}(x) \\ &= -K_a (x - x_d) \end{aligned} \quad (44)$$

Where k_a is the attractive gain and x_d is the reference states vector. On the other hand, the repulsive force, which is responsible of the avoidance of obstacles, is defined as:

$$\begin{aligned} U_{rep} &= \begin{cases} \frac{1}{2} k_r \left(\frac{1}{\rho} - \frac{1}{\rho_0} \right)^2, & \text{if } \rho < \rho_0 \\ 0, & \text{if } \rho > \rho_0 \end{cases} \\ F_{rep}(x) &= -\nabla U_{rep}(x) \\ &= \begin{cases} -k_r \left(\frac{1}{\rho} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2} \frac{\partial \rho}{\partial x}, & \text{if } \rho < \rho_0 \\ 0, & \text{if } \rho > \rho_0 \end{cases} \end{aligned} \quad (45)$$

Where k_r is the repulsive gain, $\rho = \sqrt{x - x_0}$ is the distance between the quadrotor and the obstacle (with $x_0 = [x_o, y_o]^T$ and ρ_0 is the minimal distance between the UAV and the obstacle that has a repulsive effect. Moreover, the term $\partial \rho / \partial x$ can be substituted with the following expression:

$$\frac{\partial \rho}{\partial x} = \left[\frac{\partial \rho}{\partial x_q} \quad \frac{\partial \rho}{\partial y_q} \right] = \frac{x_0 - x}{\rho} \quad (46)$$

In accordance with the modified APF algorithm proposed in (Iswanto, 2019), the avoidance of the local minima can be performed adding a virtual force $F_{vir}(x)$ to the repulsive force, hence it is obtained:

$$\begin{aligned} F_{rep}(x) &= \begin{cases} F_{rep1}(x) + F_{vir}(x), & \text{if } \rho < \rho_0 \\ 0, & \text{if } \rho > \rho_0 \end{cases} \\ F_{rep1}(x) &= -k_r \left(1 - \frac{\rho}{\rho_0} \right) \frac{x_0 - x}{\rho^3} \\ F_{vir}(x) &= -k_v \frac{1}{\rho_0} \end{aligned} \quad (47)$$

Lastly, the total force of the APF can be computed as the sum of the attractive term and repulsive term:

$$F_{total}(x) = F_{att}(x) + \sum_{i=1}^n F_{rep_i}(x) \quad n = \text{number of obstacles} \quad (48)$$

In this thesis, the aforementioned APF formulation has been adapted in order to run simultaneously with a trajectory planner algorithm. As will be deeper discussed in *Chapter 5*, the *trajectory planner* is aimed to create a path that the quadrotor is able to follow in a given amount of time. It imposes a trapezoidal linear velocity profile on the segment linking two consecutive waypoints, hence the attractive force provided by the APF algorithm has been discarded, providing the system with only the repulsive term desired velocities.

Lastly, for the sake of clarity, an example of APF potential field is provided. Figure 56 shows the APF potential field in presence of obstacles considering a *snail pattern*. The aim of this graph is to show the attractive action of waypoints and the repulsive actions of obstacles, through the overlapping of APF attractive, repulsive potential field and quadcopter's trajectory.

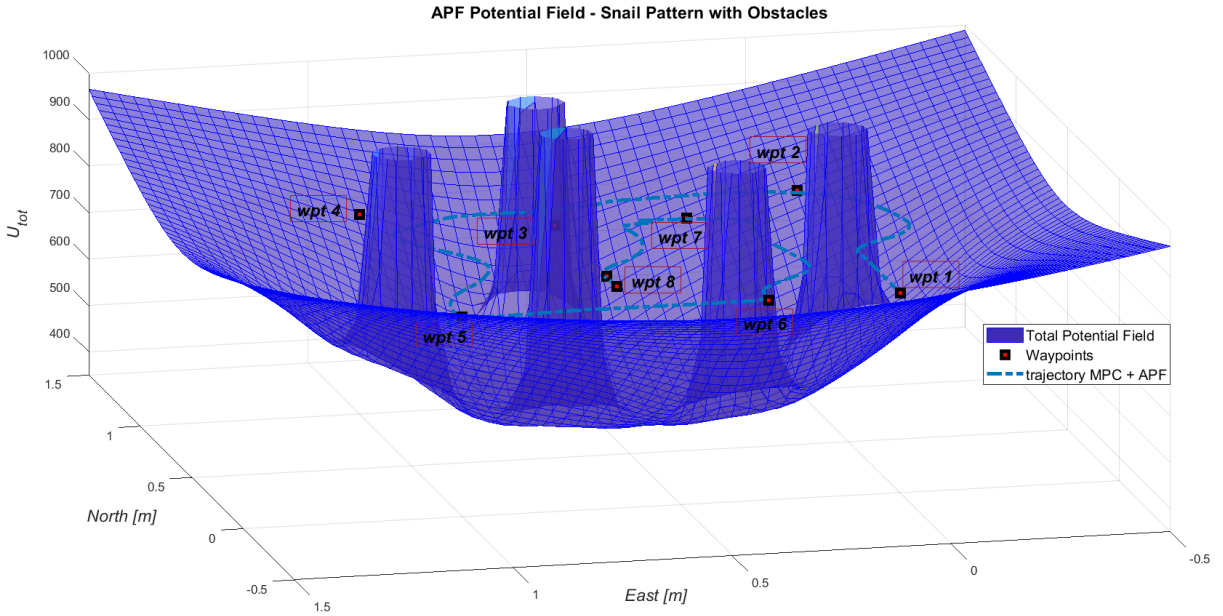


Figure 56: illustration of total potential field $U_{tot} = U_{att} + U_{rep}$ computed by APF algorithm for the snail pattern in presence of obstacles. Quadcopter path trajectory has been added, showing the attractive actions of waypoints and the repulsive actions of obstacles on quadcopter behaviour during a path following manoeuvre.

4.2 – Navigation Algorithm

The *navigation* module in a GNC system implements aircraft state estimation. As introduced in *Chapter 1*, in order to be able to guide and control the UAV, the current states of the vehicle must be provided to the controller at high fidelity and high bandwidth. The advantages provided by powerful computer and lower computational costs have allowed *sensor fusion* for navigation. Thanks to multi-sensor navigation, it is possible to obtain high fidelity information about the aircraft using minimum combination of sensors. This is a considerable advantage to UAV, where size, weight, power, and endurance are all critical.

In this project the controller algorithms have been tuned and tested assuming the states provided by the *plant* model as the estimated states, hence the *estimator* and the *filtering stage* have been bypassed.

4.3 – Control Algorithm

Control system design represents a considerable area of research application. Multirotor UAVs are generally a very interesting platform for evaluating existing and testing new control approaches. It is worth mentioning that in real system, such as multirotor, the field of control theory is closely related to signals processing, hence data from sensors need to be processed. As mentioned before, in this project it has been assumed the states provided by the *plant* as estimated states, bypassing the states estimator block and filtering stage.

Generally, UAV dynamics can be classified into two categories: fast dynamics, which relates to the attitude and angular rates, and slow dynamics, which relates to the position (referred to a fixed reference frame). This classification allows the development of a cascade controller composed by two loops: an *inner loop* related to fast dynamics (which often includes the altitude) and an *outer loop* related to position.

As introduced in *Chapter 1*, this thesis focuses on two controllers' architectures: the first one based on an MPC for quadrotor attitude and altitude control (inner loop) and a PD controller for the position control; the second one based on an MPC for quadrotor position and attitude control and a PID controller for the altitude control (inner loop). In designing MPC, particular properties of the quadcopter are required. The physical properties used in this project are based on (Carminati, 2019) and are listed in Table 6.

Table 6: Quadcopter physical properties used in controllers tuning.

Quadcopter Parameters	
Mass, m	1.5 [Kg]
Moment Arm, l	0.2 [m]
Moment of Inertia about x-axis, I_{xx}	0.0170 [Kg · m ²]
Moment of Inertia about y-axis, I_{yy}	0.0308 [Kg · m ²]
Moment of Inertia about z-axis, I_{zz}	0.0173 [Kg · m ²]

4.3.1 – Proportional Integral Derivative – PID Controller

A *Proportional Integral Derivative* PID controller is a well-known feedback controller most widely used in various type of plant industries utilizing robotics all over the world. The widespread use of PID controllers in industries has affected efforts in the designing of conventional PID controller achieving good performance.

A complete PID controller generates control actions using information about the difference between the measured outputs of the system and a reference signal or setpoint; this value is called error e . The generic PID generates a control action taking into account the current error estimation (measured at a certain time), the derivative and the integral of the error. This controller takes many structures, however, the most important one is as in the following form:

$$u(t) = K_p e(t) + K_I \int_0^t e(t) dt + K_D \frac{de(t)}{dt} \quad (49)$$

Where $u(t)$ is the input signal to the plant and the error is expressed as $e(t) = r(t) - y(t)$ where $y(t)$ is the vector of systems outputs and $r(t)$ is the vector containing the reference signals. The arbitrary selected parameters K_p , K_I and K_D are respectively the *proportional* gain, the *integral* gain, and the *derivative* gain.

As can be seen a PID controller is simple and easy to design which often results in satisfactory performances; however, it has various limitations since the fixed gains limits the performance of the algorithm over a wide range of operating points. As a PID controller is based on a linear model, nonlinearities in the system brings uncertainty and degraded performance. The overall performances can be significantly penalized by the current operating point and cannot achieve optimal control.

One interesting aspect of PID controller is the selection of the method of control; it can be *mode-based* or *non-mode based*. The *mode-based* is based on independent controllers for each controlled state and a higher level controller decide how these interact; while the *non-mode based* consists of a single controller for all the states together. In this thesis the adopted *mode-based* approach is tested considering two different approaches: the first one a *Proportional Derivative* PD which relates to quadrotor position control and the second one a *Proportional Integral Derivative* PID which relates to altitude control. A schematic representation is provided in Figure 57 and Figure 58.

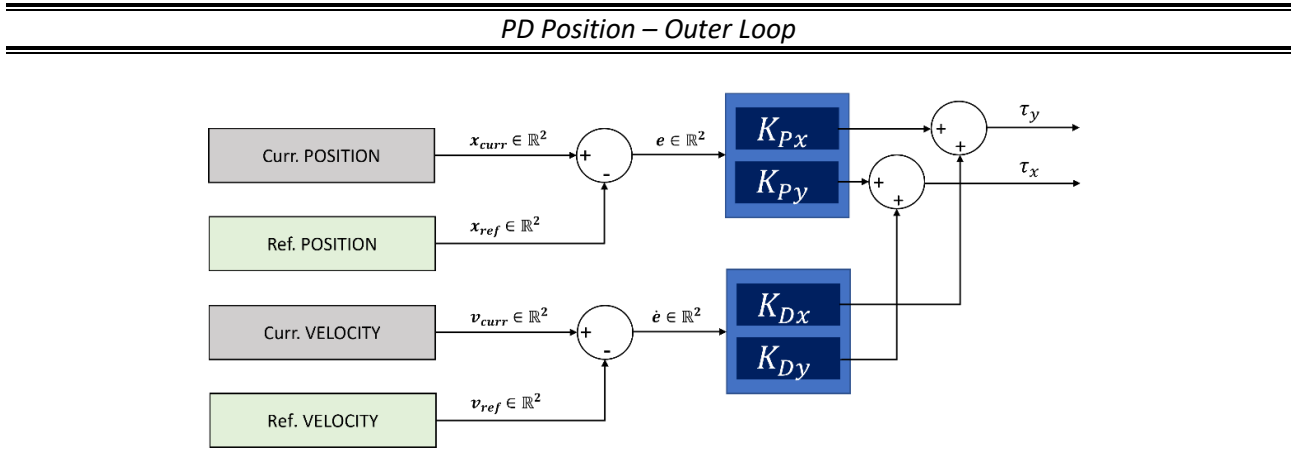


Figure 57: block diagram PD controller to control quadrotor slow dynamics.

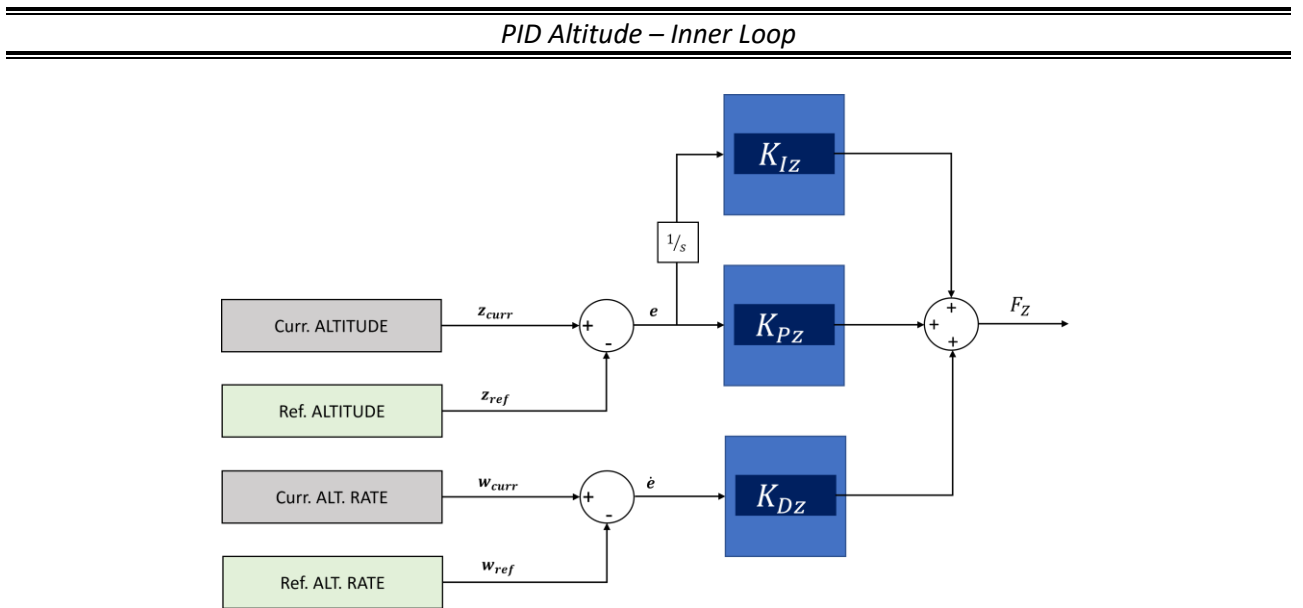


Figure 58: block diagram PID controller to control quadrotor altitude.

4.3.2 – Receding Horizon Controller – RHC Controller

The *Receding Horizon Controller* RHC, also referred to as *Model Predictive Controller* MPC, is a widely spread technology in industry for the control of highly complex multivariable systems. The approach is said *predictive*, because the optimal control action is obtained as the result of an optimal control problem formulated over a time-interval that starts at the current time up to a certain time in the future. In accordance with (Bemporad, Model Predictive Control Design: New Trends and Tools, 2006) the basic idea behind MPC is to start with a *model* of the open-loop process (in order to explain the relations between system's variables), then, add constraints specifications on system variables (such as inputs constraints and outputs constraints) and define desired performance specifications through different weights on tracking errors and actuators efforts (like classical linear quadratic regulator). Firstly, an optimal control problem based on the given model, constraints and weights is constructed and translated into an equivalent optimization problem. Then, at each sampling time, the optimization problem, which is based on the initial state and the reference, is solved by taking the current state as the initial condition of the problem. The result of the optimization problem is an *optimal sequence* of control inputs and only the first sample of such sequence is applied to the process. At the next time step the aforementioned procedure is repeated over a shifted prediction horizon (this is the reason of the approach's name *receding-horizon*).

Once defined an accurate model of the controlled system and the performance index and constraints are determined in accordance with the desired system behaviour, MPC provides near-optimal control. It is worth mentioning that the simpler the model the easier is solving the optimization problem. While *simulation models* looks for the most accurate model to numerically reproduce the behaviour of the system, *prediction model* uses very simple model, yet representative enough to capture the main dynamics relations. The RHC controller can be conceptually divided into two parts: an *estimator*, which makes predictions about the future states based on the current available information and an *optimizer*, which calculates an optimal plan of actions assuming the estimates are correct. A typical RHC working policy is based on the following steps:

- At instant t it is considered a time interval extending T time step into the future $t, t + 1, \dots, t + T$;
- By using available data at time t a *predictive model* is built;
- Objective function and estimations are based on data available at time t . The problem of constrained objective function minimization is addressed using a solver. The optimal values obtained as the solution of the RHC optimization problem are defined as a *plan of action* for the next T steps;
- The first sequence of control inputs is applied to the system, the remaining are discarded. At the next time step the process is repeated with the updated estimates of the current states.

This section is organised as follow: firstly, a brief overview of MPC based on linear models and quadratic programming are provided; then the implemented MPC algorithm in MATLAB and Simulink is presented.

4.3.2.1 – Problem Formulation: Linear Model Predictive Control

In this section the fundamental formulation of the *Linear-Model-Predictive-Controller* LMPC is presented. Several formulations of MPC algorithms have been proposed in literature; however, the simplest MPC algorithm is based on the *linear discrete-time* prediction model. The discrete dynamical system model used by the controller is the state-system formulation reported below:

$$\begin{aligned} x_{k+1} &= A x_k + B u_k & k = 0, 1, 2, \dots \\ y_k &= C x_k \end{aligned} \quad (50)$$

Where $y \in \mathbb{R}^p$ is the vector of outputs, $u \in \mathbb{R}^m$ is the vector of inputs control actions and $x \in \mathbb{R}^n$ is the vector of states. The receding horizon regulator, also referred to as RHC or MPC, is based on the minimization of the following *infinite* horizon open-loop *quadratic objective function*:

$$\min_{u^N} \sum_{j=0}^{\infty} (x_{k+j}^T Q x_{k+j} + u_{k+j}^T R u_{k+j} + \Delta u_{k+j}^T S \Delta u_{k+j}) \quad (51)$$

Where Q is a symmetric positive semidefinite penalty matrix on the outputs, R is a symmetric positive definite penalty matrix on the inputs and u_{k+j} is the vector of the input at time j in the open-loop objective function. Lastly, S is a symmetric positive matrix on the rate of change of the input and $\Delta u_{k+j} = u_{k+j} - u_{k+j-1}$ is the rate of change of the input vector at instant j . On the other hand, the minimization shown in equation (51) returns a vector u^N defined as the vector of the N future open-loop control moves:

$$u^N = [u_k \ u_{k+1} \ u_{k+2} \ \dots \ u_{k+N-1}] \quad (52)$$

As can be seen, the last element of vector u^N is u_{k+N-1} since the value of the input vector at time $k + N$ is set to zero for all $j \geq N$ in the open-loop objective function. The RHC computes the vector u^N that optimizes the open-loop objective function, defined in equation (51), and applies the first input command u_k to the plant. The procedure is repeat at each subsequent control interval, considering a new state vector at time k provided by the plant measurements. The infinite horizon open-loop quadratic objective function reported in (51) can be adapted to a *finite* horizon of prediction; the result is a *finite* time optimal control problem:

$$\begin{aligned} \min_{u^N} \sum_{j=0}^{N-1} (x_{k+j}^T Q x_{k+j} + u_{k+j}^T R u_{k+j} + \Delta u_{k+j}^T S \Delta u_{k+j}) &+ x_{k+N}^T \bar{Q} x_{k+N} + \Delta u_{k+N}^T S \Delta u_{k+N} \\ \text{s.t.} \quad x_{k+1} &= A x_k + B u_k & k = 0, 1, \dots, N-1 \\ x_0 &= x(t) \\ u_{min} &\leq u_k \leq u_{max} & k = 0, 1, \dots, N-1 \\ y_{min} &\leq C x_k \leq y_{max} & k = 0, 1, \dots, N-1 \end{aligned} \quad (53)$$

Where N is the prediction horizon, $u^N \in \mathbb{R}^{Nm}$ is the sequence of the manipulated variables to be optimized, the two matrices \bar{Q} and S are weighting matrix (like Q and R) of appropriate dimensions defining the performance index, $u_{min}, u_{max} \in \mathbb{R}^m$ and $y_{min}, y_{max} \in \mathbb{R}^p$ define constraints on inputs and state variables respectively. Assuming $k = 0$ in equation (53), by substituting $x_j = A^j x(t) + \sum_{i=0}^{j-1} A^i B u_{k-1-i}$ the optimal control problem can be recast as the *Quadratic Programming* QP problem:

$$\begin{aligned} U^*(x(t)) &\triangleq \arg \min_U \frac{1}{2} U' H U + x'(t) C' U + \frac{1}{2} x'(t) Y x(t) \\ \text{s.t.} \quad G U &\leq W + S x(t) \end{aligned} \quad (54)$$

Where $U^*(x(t))$ is the sequence of optimal control inputs, $H = H' > 0$ and C, Y, G, W, S are matrices of appropriate dimensions. Hence, the MPC operates the following actions: at time t it measures or estimates the current state $x(t)$, solves the QP problem and applies only the first sequence to the system.

$$u(t) = u_0^*(x(t)) \quad (55)$$

The procedure is repeated at each time step of simulation, hence at time $t + 1, t + 2, \dots, t_{end}$. It is worth mentioning that when there are no constraints, $N \rightarrow \infty$ and P is the solution of the *Riccati Equation* associated with the matrices A, B and weights Q, R , the MPC coincides with the classical *Linear Quadratic Regulator* LQR controller. Hence the MPC can be seen as a constrained LQR controller.

4.3.2.2 – Reference Tracking Formulation

Once designed the basic MPC controller law (53) other possible applications of MPC can be further analysed. Usually there is interest to make a certain output vector $y(t) = Cx(t)$ track a reference signal $r(t)$ under specified constraints. This task can be addressed by the following cost function:

$$\sum_{j=0}^{N-1} (y_j - r(t)) Q_y (y_j - r(t)) + \Delta u_j' R \Delta u_j \quad (56)$$

Where the matrix $Q_y \in \mathbb{R}^{p,p}$ is the matrix of output weights and the increment of commands values, defined as in (51), is the new optimization variable. Hence the tracking task is addressed using the following setup:

$$U^*(x(t)) \triangleq \arg \min_U \frac{1}{2} U' H U + [x'(t) \ r'(t) u'(t-1)] C' U + \frac{1}{2} x'(t) Y x(t) \quad (57)$$

$$s. t. \quad G U \leq W + S \begin{bmatrix} x(t) \\ r(t) \\ u(t-1) \end{bmatrix}$$

Where the new control law is defined as $u(t) = u(t-1) + \Delta u_0^*(x(t), r(t), u(t-1))$. For the sake of clarity Figure 59 shows the control actions provided by the MPC controller in a simple path following manoeuvre. The results show only the actions applied to the plant, while Figure 60 show one component (in this case $\Delta \tau_x$) of the entire optimized vector $U^*(x(t))$ computed at different time of simulations, classified by colour.

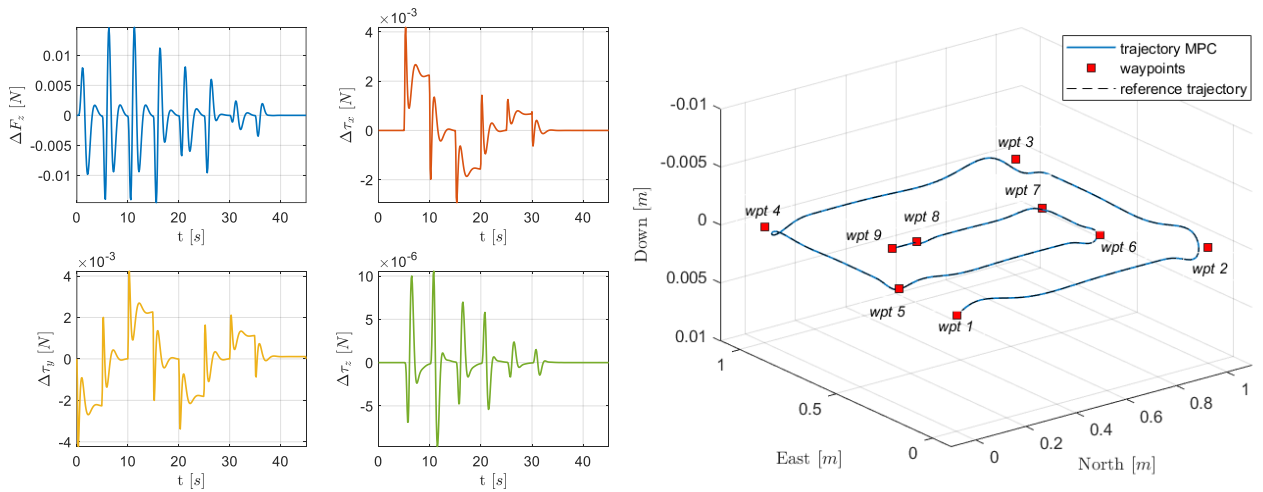


Figure 59: snail pattern, path following MPC controller. On the left MPC controller inputs $F_z, \tau_x, \tau_y, \tau_z$; on the right 3D plot of snail path.

As can be seen, the values of the pink marker's series are near zeros, since the time in which the optimization has been computed ($t \cong 4.9[s]$) is not close enough to consider the next control action. As expected, torque values increasingly change (from orange to green) moving toward the next reference waypoint on the path.

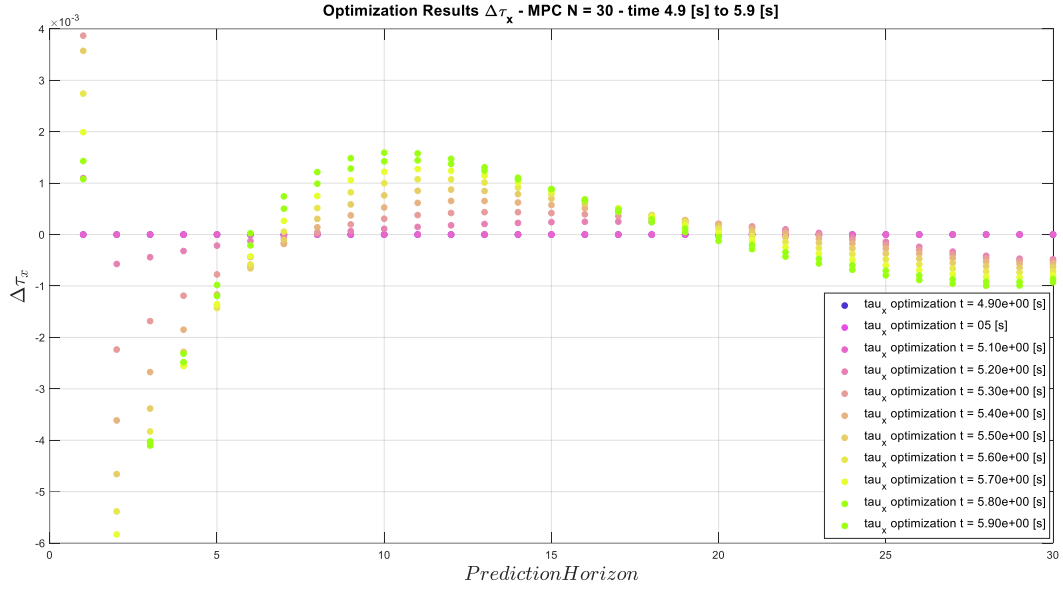


Figure 60: $\Delta\tau_x$ inputs optimization results over prediction horizon $N = 30$. Each coloured circle represents a control input $\Delta\tau_x$ computed by the MPC. The prediction horizon ($N=30$) is on the x-axis while the different colours are referred to predicted inputs at different times of computations.

4.3.2.3 – Problem Solution: Interior Point Method

According to (Wright, 1997) developments in MPC have created a demand for fast, reliable solution of problems in which nonlinearities, noise and constraints on the states and controls may all be present. *Interior-point methods* and *stochastic optimization* are both powerful tools that can improve significantly MPC performances. The problems proposed in (53) and (54) are both examples of convex quadratic program. Two successful methods for addressing this class of problems are the *active set method* (Fletcher, 1987) and the *interior-point method* (Wright, 1997). In this section an *interior-point-based approach* for solving the MPC optimization problem is proposed. The description of the *interior-point method* can be addressed defining the *mixed monotone linear complementarity problem* (mLCP), which is a powerful paradigm that generalizes the optimality conditions for linear quadratic programs.

The mLCP is defined in terms of a square, positive semidefinite matrix $M \in \mathbb{R}^{n,n}$ and a vector $q \in \mathbb{R}^n$; the problem is to find the vectors z , x , and s such that:

$$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} + \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} 0 \\ s \end{bmatrix} \quad (58)$$

$$x \geq 0, \quad s \geq 0, \quad x^T s = 0$$

The matrices M_{11} and M_{22} are square submatrices of dimensions n_1 and n_2 . The infeasible-interior point algorithm starts at point (z^0, x^0, s^0) with $x^0 > 0$ and $s^0 > 0$. All iterations (z^k, x^k, s^k) maintain the positivity properties but the *complementarity gap* defined as:

$$\mu_k = (x^k)^T s^k / n_2 \quad (59)$$

Gradually reduced to zero as the number of iterations tends to $k \rightarrow \infty$. Each step of the algorithm is a modified Newton step for the system of nonlinear equations defined in (58) and the complementarity condition $x_i s_i = 0$ with $i = 1, 2, \dots, n_2$. Hence the system can be written as:

$$F(z, x, s) \stackrel{\text{def}}{=} \begin{bmatrix} M_{11}z + M_{12}x \\ M_{21}z + M_{22}x - s \\ XSe \end{bmatrix} = 0 \quad (60)$$

Where it has been adopted the following notations: $X = \text{diag}(x_1, x_2, \dots, x_{n_2})$ and $S = \text{diag}(s_1, s_2, \dots, s_{n_2})$. Once introduced the mLCP the algorithm can be expressed using the following form:

$$\begin{aligned} &\text{Given:} \quad (z^0, x^0, s^0) \quad \text{with} \quad (x^0, s^0) > 0 \\ &\text{for:} \quad k = 0, 1, 2, \dots \\ &\text{for some } \sigma_k \in (0, 1), \quad \text{solve:} \end{aligned} \quad (61)$$

$$\begin{bmatrix} M_{11} & M_{12} & 0 \\ M_{21} & M_{22} & -I \\ 0 & S^k & X^k \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta x \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_1^k \\ -r_2^k \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix}$$

Where $r_1^k = M_{11}z^k + M_{12}x^k$ while $r_2^k = M_{21}z^k + M_{22}x^k - s^k$. Lastly e is $e = (1, 1, \dots, 1)^T$.

Moreover, introducing a parameter α_k , the values of $(z^{k+1}, x^{k+1}, s^{k+1})$ are defined as follow:

$$\begin{aligned} (z^{k+1}, x^{k+1}, s^{k+1}) &= (z^k, x^k, s^k) + \alpha_k (\Delta z^k, \Delta x^k, \Delta s^k) \\ &\text{with } \alpha_k \in (0, 1] \quad \text{that satisfy } (x^{k+1}, s^{k+1}) > 0 \end{aligned} \quad (62)$$

The only two parameters to choose in implementing the algorithm are the values of σ_k and α_k . The choice of σ_k is arbitrary (generally it is confined to the range $[10^{-3}, 0.8]$) while α_k is required to satisfy some conditions detailed in (Wright, 1997). When the two parameters satisfy the aforementioned conditions, global convergence to a solution of system (58) is attained (whenever such solution exist).

In practical implementations of *interior-point methods*, α_k is always chosen via the following simple heuristic:

$$\begin{aligned} \alpha_k^{\max} &= \{\alpha \in (0, 1] \mid (z^k, x^k, s^k) + \alpha(\Delta z^k, \Delta x^k, \Delta s^k) > 0\} \\ \alpha_k &= \min(1, 0.995 \cdot \alpha_k^{\max}) \end{aligned} \quad (63)$$

The major computational effort to be performed is the solution of the system (61), the matrix in this system has a lot of structure due to the zeroes blocks and the diagonal I, S^k and X^k . Moreover, the matrix M is sparse so sparse matrix factorizations are called for; however, problems like MPC optimization problem require simpler factorization code. The solution of (61) is obtained eliminating the Δs component:

$$\Delta s = (x^k)^{-1} (-X^k S^k e + \sigma_k \mu_k e - S^k \Delta x^k) \quad (64)$$

By substituting into the first two rows of (61) it is obtained the final system:

$$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} + (X^k)^{-1} S^k \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta x \end{bmatrix} = \begin{bmatrix} -r_1^k \\ -r_2^k - s^k + \sigma_k \mu_k (X^k)^{-1} e \end{bmatrix} \quad (65)$$

In most cases, the partitions M_{11} , M_{12} , M_{21} and M_{22} are zero diagonal or have some simple structure so further reduction of the system is possible. The solution presented for the problem formulated in (58) can be easily adapted to a convex linear quadratic programming problem. Considering a general convex QP and the *Karush-Kuhn-Tucker*¹⁰ (KKT) conditions the QP problem can be expressed in the form of mLCP.

$$\begin{aligned} \min_z \frac{1}{2} z^T Q z + c^T z \quad s. t. \quad & H z = h, G z \leq g \\ M_{11} = \begin{bmatrix} Q & H^T \\ -H & 0 \end{bmatrix} & \quad M_{12} = \begin{bmatrix} G^T \\ 0 \end{bmatrix} \\ M_{21} = [-G \quad 0] & \quad M_{22} = 0 \\ q_1 = \begin{bmatrix} c \\ h \end{bmatrix} & \quad q_2 = g \quad s \leftarrow t \end{aligned} \quad (66)$$

¹⁰ The *Karush-Kuhn-Tucker* KKT conditions, in mathematical optimization, are defined as the first derivative tests for an optimal solution in nonlinear programming, assuming regularity conditions to be satisfied. From a mathematical point of view, allowing inequality constraints, the KKT conditions can be thought as a generalization of the method of *Lagrange Multipliers* which allows only equality constraints.

4.3.2.4 – Implementation of MPC Algorithm in MATLAB and Simulink

The standard way of computing the *Linear* MPC is to solve the QP problem online at each sample time t . Commercially available tools are classified into two main categories: tools with a proprietary real-time industrial control system, such as *DMCplus* (by Aspen Technology Inc.); and tools intended for analysis and prototyping, such as the *MPC Toolbox* for MATLAB (MathWorks, Inc.).

In this project the MPC algorithm has been design and implemented from scratch (no Simulink pre-built toolbox has been used). The algorithm has been designed working on both MATLAB and Simulink environments. It consists of a first script written in MATLAB and a Simulink model, which is run directly from MATLAB. All the parameters of the continuous state space model and controller are initialized from the script. These values include the quadrotor's physical properties (mass and moments of inertia), quadrotor thrust and drag coefficient, maximum and minimum allowed thrust, and torque, continuous state space matrices, MPC prediction horizon, MPC weighting matrices, PD (and PID) tuning parameters, model's initial states and inputs, time of simulation and reference trajectories (and obstacles) to be tracked (and avoid). For the sake of clarity, a flow chart of the implemented algorithm in MATLAB and Simulink is provided in Figure 61:

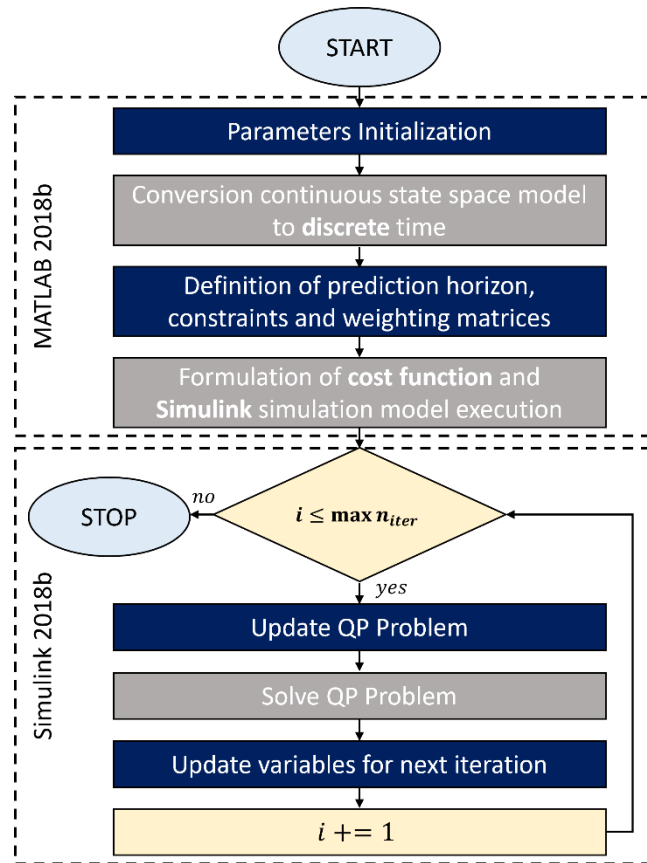


Figure 61: flow chart of MPC algorithm implementation in MATLAB and Simulink 2018b.

Once defined matrices A, B, C, D , the states space model is obtained using the MATLAB function:

$$sys = ss(A, B, C, D) \quad (67)$$

By using the MATLAB function *c2dm* the continuous state space model is converted into a discrete one.

$$[A_d, B_d, C_d, D_d] = c2dm(A, B, C, D, Dts) \quad (68)$$

Where the matrices A, B, C, D are the continuous states matrices and Dts is the user-defined discretization time. In this project the D matrix is assumed null since there is no feedthrough signals. The next step is the definition of MPC prediction horizon H_p and the initialization of inputs constraints u_{min}, u_{max} and weighting matrices Q and R which are used in the optimization problem formulation. Once defined the matrices A_d, B_d, C_d, D_d and the MPC weighting matrices, the *cost* function is assembled in the following form:

$$J(z, x_0) = x'_0 Q x_0 + \begin{bmatrix} x_1 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}^T \begin{bmatrix} Q & 0 & \dots & 0 \\ 0 & Q & \vdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \dots & 0 & P \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix} + \begin{bmatrix} u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}^T \begin{bmatrix} R & 0 & \dots & 0 \\ 0 & R & \vdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \dots & 0 & R \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \quad (69)$$

$$\begin{bmatrix} x_1 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix} = \begin{bmatrix} A_d & 0 & \dots & 0 \\ A_d B_d & B_d & \vdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ A_d^{N-1} B_d & A_d^{N-2} B_d & \dots & B_d \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} + \begin{bmatrix} A_d \\ A_d^2 \\ \vdots \\ A_d^N \end{bmatrix} x_0$$

These equations can be replaced by the compact form:

$$J(z, x_0) = x'_0 Q x_0 + \begin{bmatrix} x_1 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}^T \bar{Q} \begin{bmatrix} x_1 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix} + \begin{bmatrix} u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}^T \bar{R} \begin{bmatrix} u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \quad (70)$$

$$\begin{bmatrix} x_1 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix} = \bar{S} z + \bar{T} x_0$$

By replacing the vector of states $[x_1 \dots x_{N-1} x_N]$ in the cost function $J(z, x_0)$, it is found:

$$\begin{aligned} J(z, x_0) &= x'_0 Q x_0 + (\bar{S} z + \bar{T} x_0)' \bar{Q} (\bar{S} z + \bar{T} x_0) + z' \bar{R} z \\ &= \frac{1}{2} x'_0 2(Q + \bar{T}' \bar{Q} \bar{T}) x_0 + x'_0 2 \bar{T}' \bar{Q} \bar{S} z + \frac{1}{2} z' 2(\bar{R} + \bar{S}' \bar{Q} \bar{S}) z \\ &= \frac{1}{2} x'_0 Y x_0 + x'_0 F' z + \frac{1}{2} z' H z \end{aligned} \quad (71)$$

The QP have been implemented; then, by using the MATLAB function:

$$SimOut = sim('ModelName.slx', Ts) \quad (72)$$

The *Simulink.SimulationOutput* object is returned, containing all the output computed by the Simulink model. The QP problem is solved using the policy presented in Section 4.3.2.3, the solver is implemented through a *MATLAB Function* Simulink block as function *quad_wright*, which is defined as:

$$[x, iter] = quad_wright(H, f, A, b, maxiter, tolerance, x0) \quad (73)$$

Where the matrices H, f are the matrices of the QP problem and the b vector define the upper and lower inputs constraints u_{min}, u_{max} . In compact form the *convex QP* is expressed as, reference (Wright, 1997):

$$J(x) = \frac{1}{2} x' H x + f' x$$

$$Ax \leq b \quad (74)$$

In accordance with notation used in *Section 4.3.2.1*, at each time step the *quad_wright* function returns a vector $U^*(x(t)) \in \mathbb{R}^{pH_p}$ containing the control inputs predicted over the prediction horizon H_p .

Lastly, Table 7 shows the vector of the controlled variables $x \in \mathbb{R}^{n_{states}}$, the continuous state space matrices $A \in \mathbb{R}^{n_{states} \times n_{states}}$ and $B \in \mathbb{R}^{n_{states} \times n_{inputs}}$ (matrices $C = eye(n_{states})$ and $D = zeros(n_{states}, n_{inputs})$ are not shown) and the number of receding prediction horizon H_p .

Table 7: definition of vector of states, continuous state space matrices A (states) B (inputs) for implemented MPC controller. On the left system based on MPC for attitude and altitude control; on the right system based on MPC for position and attitude control.

MPC Attitude and Altitude	MPC Position and Attitude
$x = [\phi, \theta, \psi, p, q, r, z, w]$	$x = [x, y, u, v, \phi, \theta, \psi, p, q, r]$
$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$
$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & c_3 & 0 & c_4 \\ 0 & 0 & 0 & 0 & c_7 & 0 \\ 0 & 0 & 0 & c_4 & 0 & c_9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/m & 0 & 0 & 0 \end{bmatrix}$	$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1/m & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/m & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & c_3 & 0 & c_4 \\ 0 & 0 & 0 & 0 & c_7 & 0 \\ 0 & 0 & 0 & c_4 & 0 & c_9 \end{bmatrix}$
$H_p = 30$	$H_p = 30$

Simulation Results

This section provides a brief overview of the implemented Simulink models and simulation results for the designed control algorithms, showing their performance in both path planning and obstacle avoidance tasks. The use of simulated environments allows designed algorithms to be analysed and tested without the need to act directly on a real platform. All system blocks, used in Simulink model, are discussed, and characterized with their impact on the results. Lastly, the results of modeling and its effects on the control strategies are presented. The simulations have been performed using MATLAB®-Simulink® 2018b and MATLAB®-Simulink® 2021b as software platforms.

5.1 – Simulation in MATLAB and Simulink Environment

This paragraph describes the main elements of the designed Simulink model. To make the simulation as real as possible the Simulink model has been designed taking into consideration a non-linear quadrotor model with fast and the slow dynamics. Moreover, a trajectory planning is implemented to generate complex trajectory path and to perform waypoint following tasks. The paragraph ends with the analysis of obstacle avoidance tasks accomplished by an *Artificial Potential Field* APF algorithm.

Figure 62 shows a simplified block diagram of studied model. Four main blocks are considered:

- the *trajectory planner* computes the reference path, which is provided as input to the controller;
- the control algorithms, defined in the *controller* block, evaluate the corrective actions to be performed in order to follow the desired path;
- the *state estimator* block, measures the current states, computed by the *plant*, and feeds the trajectory planner and the control block with the measured values.

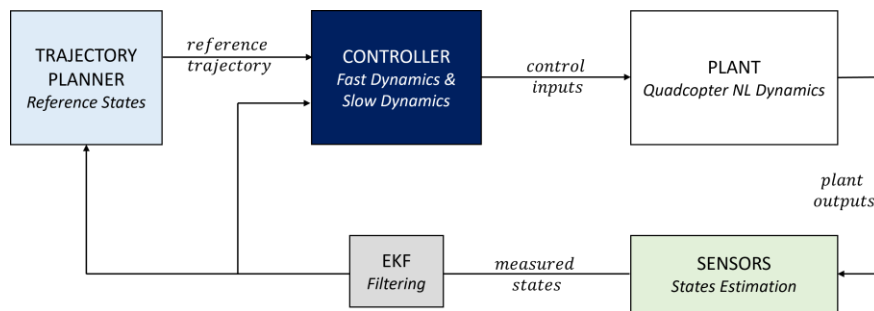


Figure 62: illustration of main elements of simplified Simulink model for Path Following, block diagram.

The *trajectory generator* consists of two algorithms: the first one is implemented in MATLAB and is used to generate time-dependent reference path given a set of waypoints with specified *Time-Of-Arrival* (ToA). The second one is a Simulink block that is used to synchronize the desired ToA with time of simulation. Each segment, linking two consecutive waypoints, is characterized by a trapezoidal linear velocity profile. The

algorithm integrates the quadrotor reference NED position as a function of time providing a maximum acceleration, which cannot exceed half gravitational acceleration $\frac{g}{2}$, and a maximum velocity that allows the successful accomplishment of the manoeuvre (respecting ToA). Algorithm's inputs are a vector $ToA \in \mathbb{R}^n$ and a matrix $WyP \in \mathbb{R}^{n,3}$ where n is the number of specified waypoints. The vector ToA consists of the time of arrivals of all the waypoints, while the matrix WyP provides the NED positions $x_{NED}, y_{NED}, z_{NED}$ of each waypoint. Desired time-dependent positions and velocities are provided to the Simulink block *Trajectory Planner*, shown in Figure 63. Additionally, a Simulink block called *Digital Clock*, provides the time of simulation, allowing the synchronization of the desired reference with the running simulation.

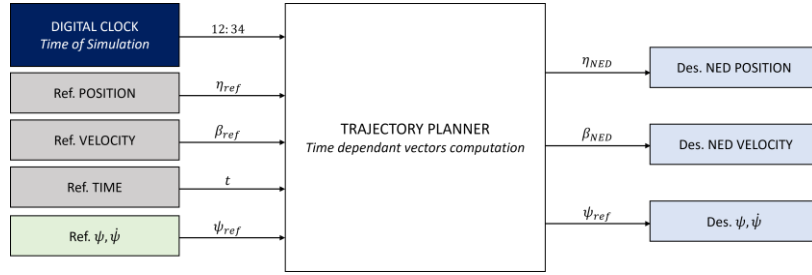


Figure 63: illustration of inputs and outputs of designed Trajectory Planner block in MATLAB-Simulink 2018b.

The *plant* block, shown in Figure 62, consists of the set of differential equations derived in Chapter 2. The non-linear model updates quadrotor states using the control forces and torques provided by the *controller*. This block is essential since it will help testing algorithms performance. In accordance with experimental tests performed in (Carminati, 2019) using a thrust stand *RCBenchmark Series 1520*, it has been used an actuator model to convert the force and moments provided by the control block into *Pulse-Width-Modulation* PWM signal that are directly used to control motors ESCs. The phase of testing and validation of the designed controllers in simulated environments, like *Unreal Engine*[®], and flight simulators, such as *AirSim*[®], required the conversion of torques and thrust inputs into PWM signals. Firstly, these signals are processed through a saturation block, then they are packed into MAVLink messages, using predefined functions such as *mavlink_msg_hil_actuator_controls_pack*¹¹, and send to the autopilot in order to update the motors states.

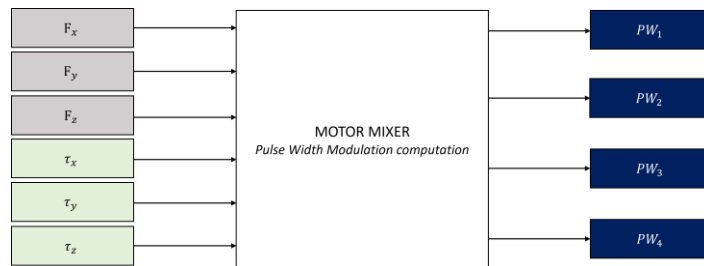


Figure 64: illustration of inputs and outputs of designed Motor Mixer block in MATLAB-Simulink 2018b.

¹¹ There are different MAVLink libraries compatible with different programming languages. In this project both the *Mavgen* library for C and Python have been used. The MAVLink C library generated by *Mavgen* is a header-only implementation that is highly optimized for resource-constrained system with limited RAM and flash memory.

It must be mentioned that for the tune and testing of the controllers all the states have been used directly as estimated states. Hence, sensors and filtering stage (like *Extended Kalman Filter*) have been bypassed.

The following section is addressed to the description of the *control* techniques considered in this project and to test their performance on path following and obstacle avoidance tasks. As mentioned in *Chapter 1*, the controller block consists of two loops: an inner loop responsible of quadrotor fast dynamics control and an outer loop applied to quadrotor slow dynamics. Since two different controllers' architecture have been tested, the discussion of both controller's structure and performance are carried out individually.

5.1.1 – Model-In the Loop Simulation – Path Following

The aim of this paragraph is to present two simulation models that have been designed to test path following algorithms. The rest of the paragraph is organized as follow: *Section 5.1.1.1* presents a simulation model based on a PD controller to control the outer loop slow dynamics and an MPC controller to control the inner loop fast dynamics. Lastly, in *Section 5.1.1.2* a Simulink model based on a MPC controller to control quadrotor position and attitude dynamics and a PID controller to control the altitude z and velocity w_{NED} is proposed.

5.1.1.1 – PD and MPC controller

The proposed Simulink model, shown in Figure 65, is composed by a total of four subsystems. The *Trajectory Planner* and the *NL_Dynamic* (previously referred to as *plant*) blocks have been described in *Paragraph 5.1*; the 'PD' and the 'MPC' blocks, which are related respectively to the control of quadrotor slow and fast dynamics, are further scrutinized hereafter.

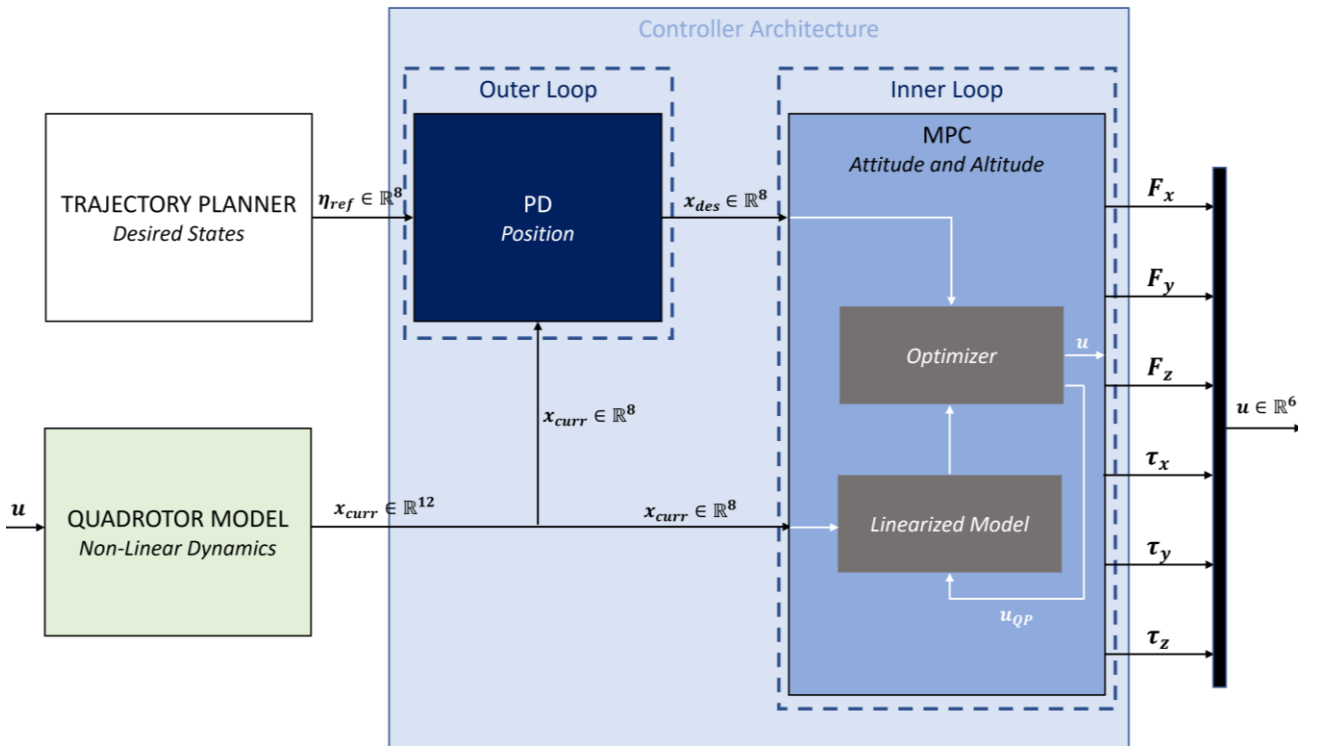


Figure 65: illustration of complete Simulink model based on PD (position control) and MPC (attitude and altitude control) controller. Software platform MATLAB-Simulink 2018b.

For the sake of clarity, a brief overview of the simulating process is outlined. Firstly, the outputs from the *Trajectory Planner* (time-dependant values of position x_{des}, y_{des} and velocity u_{des}, v_{des}) are provided as input to the outer-loop controller, the PD block. The latter also receives as inputs the current states, provided by the *plant* block, and computes the desired *Euler angles* and *angular rates*, which are directly provided to the inner-loop controller MPC. Similarly, the MPC block receives as inputs two vectors, respectively the vector of desired states $x_{des} \in \mathbb{R}^8$ and the vector of current states $x \in \mathbb{R}^8$. At each time step it computes the control actions that need to be performed to track the reference trajectory. Lastly, the forces and moments are provided to the *plant* which update the states and feeds the controllers blocks with the updated states.

Table 8: Position Control - outer-loop quadrotor slow dynamics PD parameters; Attitude & Altitude Control – inner loop quadrotor fast dynamics MPC states weighting matrix $Q_{MPC} \in \mathbb{R}^{10,10}$; Inputs Control – inner loop quadrotor fast dynamics MPC inputs weighting matrix $R_{MPC} \in \mathbb{R}^{6,6}$.

Position Control - PD		
Axis	Parameter	Value
x	K_{Px}	0.02
	K_{Dx}	0.39
y	K_{Py}	0.034
	K_{Dy}	0.29

Attitude & Altitude Control - MPC		
Axis	Parameter	Value
ϕ	$Q_{MPC}(1,1)$	3×10^4
θ	$Q_{MPC}(2,2)$	10.5
ψ	$Q_{MPC}(3,3)$	18
p	$Q_{MPC}(4,4)$	9×10^4
q	$Q_{MPC}(5,5)$	10.3
r	$Q_{MPC}(6,6)$	18
z	$Q_{MPC}(7,7)$	9.5×10^2
w	$Q_{MPC}(8,8)$	7.5×10^1

Inputs Control - MPC		
Axis	Parameter	Value
F_x	$R_{MPC}(1,1)$	1
F_y	$R_{MPC}(2,2)$	1
F_z	$R_{MPC}(3,3)$	5×10^1
τ_x	$R_{MPC}(4,4)$	3×10^{-3}
τ_y	$R_{MPC}(5,5)$	3×10^{-3}
τ_z	$R_{MPC}(6,6)$	3×10^{-3}

The evaluation of controllers' parameters have been obtained through trial and error. The obtained values, used in the simulations results (Section 5.1.3), are reported in Table 8.

5.1.1.2 – MPC and PID controller

In this section, the second proposed solution is considered. The simulation works the same as the model presented in Section 5.1.1.1; however, in this configuration the quadrotor dynamics is completely controlled by the MPC controller, except for the altitude, z_{des}, w_{des} , which is based on a separate PID controller block. As shown in Figure 66 the MPC block receives two inputs vectors. The desired position, velocities (provided by the *Trajectory Planner* x_{des}, y_{des} and u_{des}, v_{des}) and the desired attitude are combined in a vector of dimension $x_{des} \in \mathbb{R}^{10}$. The second input is provided by the *plant* block and has dimension $x \in \mathbb{R}^{10}$. Once found the solution of the optimization problem, the MPC returns as outputs the vector of control inputs, which is a vector of dimension $u \in \mathbb{R}^6$.

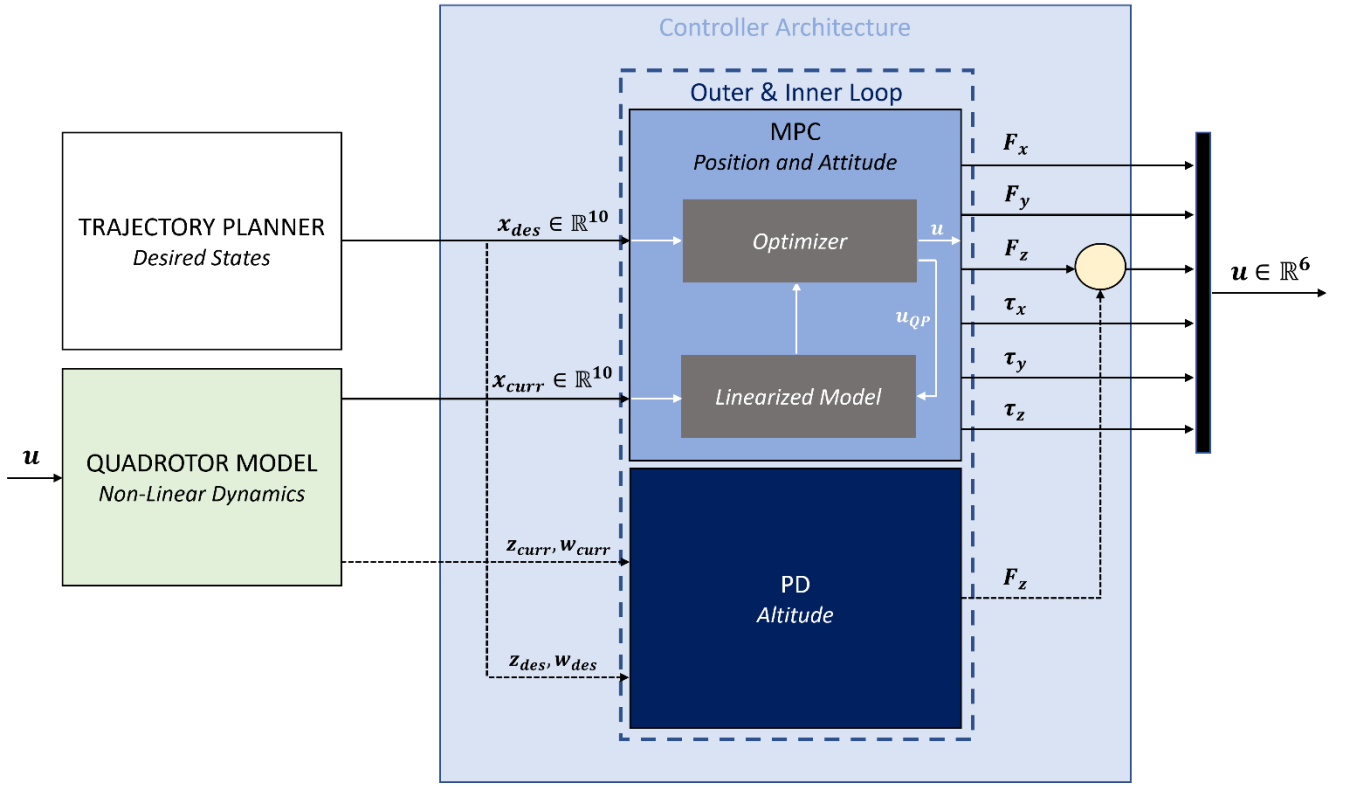


Figure 66: illustration of complete Simulink model based on MPC (position and attitude control) and PID (altitude control) controller. Software platform MATLAB-Simulink 2018b.

Table 9: Altitude Control - outer-loop quadrotor slow dynamics PID parameters; Position & Attitude Control – inner loop quadrotor fast dynamics MPC states weighting matrix $Q_{MPC} \in \mathbb{R}^{10,10}$; Inputs Control – inner loop quadrotor fast dynamics MPC inputs weighting matrix $R_{MPC} \in \mathbb{R}^{6,6}$.

Altitude Control - PID		
Axis	Parameter	Value
z	K_{Pz}	35.6
	K_{Iz}	12.18
	K_{Dz}	0.005

Position & Attitude Control - MPC		
Axis	Parameter	Value
x	$Q_{MPC}(1,1)$	30
y	$Q_{MPC}(2,2)$	30
u	$Q_{MPC}(3,3)$	16.4
v	$Q_{MPC}(4,4)$	16.7
ϕ	$Q_{MPC}(5,5)$	3
θ	$Q_{MPC}(6,6)$	10
ψ	$Q_{MPC}(7,7)$	18
p	$Q_{MPC}(8,8)$	9
q	$Q_{MPC}(9,9)$	10
r	$Q_{MPC}(10,10)$	18

Inputs Control - MPC		
Axis	Parameter	Value
F_x	$R_{MPC}(1,1)$	1
F_y	$R_{MPC}(2,2)$	1
F_z	$R_{MPC}(3,3)$	2×10^{-1}
τ_x	$R_{MPC}(4,4)$	2×10^{-1}
τ_y	$R_{MPC}(5,5)$	2×10^{-1}
τ_z	$R_{MPC}(6,6)$	2×10^{-1}

In accordance with the previous analysis, the evaluation of the controllers' parameters have been obtained through trial and error. The chosen values of both MPC and PID controllers are defined in Table 9.

5.1.2 – Model-In-the-Loop Simulation – Obstacle Avoidance

Parallel to the design of path following algorithms, the study of a collision avoidance solution has been carried out. In this section the problem of obstacle avoidance is addressed using an *Artificial Potential Field* APF algorithm. As shown in Figure 67 the simulation model is similar to the previous one, presented in Section 5.1.1. However, the inputs of the controller's block are changed. The reference velocities and positions are provided by the *Trajectory Planner* until an obstacle is detected. This means that, as long as the difference between quadcopter position and obstacle position is within a predefined tolerance: $\Delta x_{obs} = x_{curr} - x_{obs} < toll$, the reference velocities are provided only by the *Trajectory Planner*; otherwise, they are the result of the *Trajectory Planner* and APF algorithm combination. In other words, the APF comes with a switching function, to provide the reference velocities only in close proximity to an obstacle, if not its output is null.

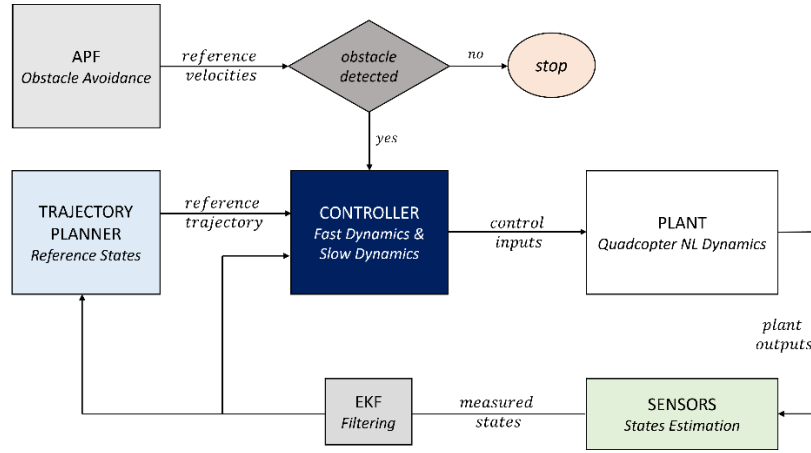


Figure 67: illustration of main elements of modified Simulink model for Obstacle Avoidance task, block diagram.

To implement the APF algorithm, different algorithms found in literature have been evaluated and it has been chosen the most appropriate for the purpose of this project. The algorithm has been subsequently implemented in a *MATLAB Function* to fit the existing working model. As shown in Figure 68, the APF receives as inputs two vectors of three elements, $pos \in \mathbb{R}^3$ and $pos_{des} \in \mathbb{R}^3$. The first one is provided by the *plant* block while the second one is the desired vector of positions computed by the *Trajectory Planner* algorithm.

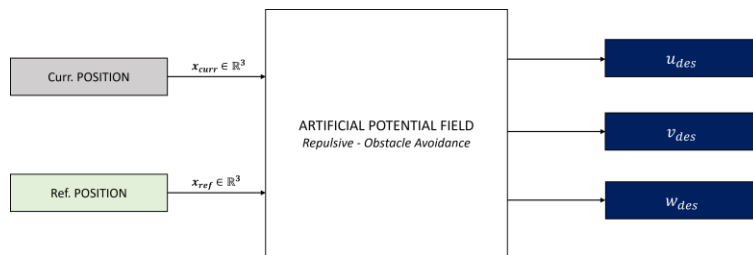


Figure 68: Artificial Potential Field Simulink subsystem. Software platform MATLAB-Simulink 2018b.

5.1.3 – Path Following and Obstacle Avoidance: Results Discussion

The aim of this section is to illustrate the ability of the designed controllers in stabilizing quadrotor dynamics throughout Simulink simulations. Some flight patterns are proposed, and the produced references are analysed. The complexity of the flight patterns increases progressively intending to test the controller capabilities in path following and in obstacle avoidance manoeuvres. Since quadrotor is intended for indoor applications, the paths have been sized in order to withstand typical room dimensions.

5.1.3.1 – Square Pattern

Table 10: square pattern reference waypoints and time of arrivals.

Waypoint	Time of Arrival	Waypoint Position
1	0	[0,0,0]
2	10	[0,0,-1]
3	20	[0,3,-1]
4	30	[3,0,-1]
5	40	[0,-3,-1]
6	50	[-3,0,-1]
7	60	[0,3,-1]

In accordance with the description provided in Section 5.1, the *Trajectory Planner* receives as inputs the vectors of waypoints positions and ToA and computes the desired time-dependant vectors of positions and velocities that are used as inputs for the controller block. Table 10 shows the elements of the vector $ToA \in \mathbb{R}^n$ (listed under Time of Arrival column) and the components of the matrix of waypoints $WyP \in \mathbb{R}^{n,3}$ (listed under Waypoint Position column) where $n = 7$ is the number of waypoints that determine the path.

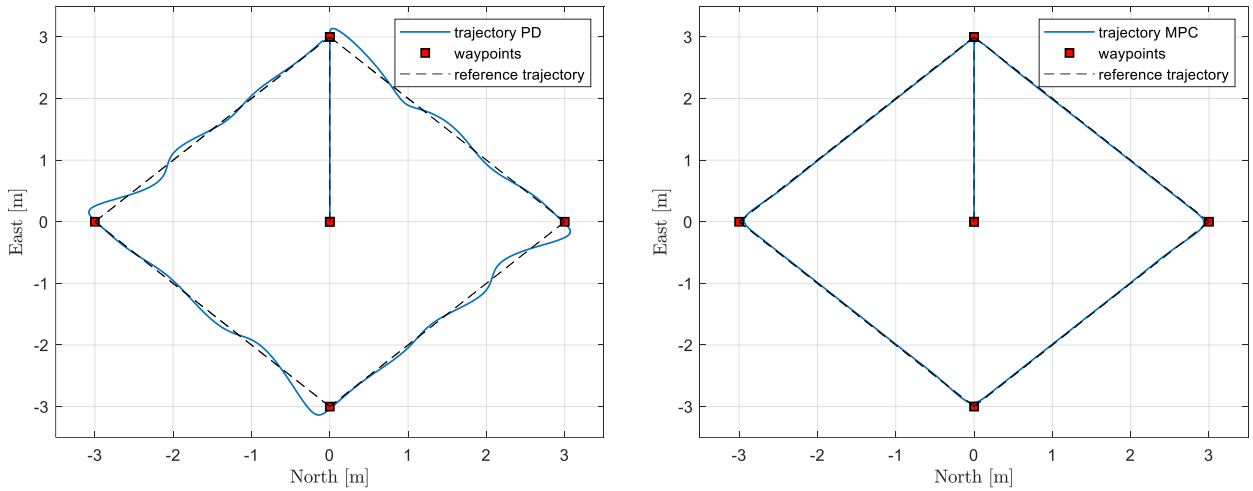


Figure 69: comparison of square pattern path following North-East plane; on the left PD controller and on the right MPC controller.

The results show that the performance of the PD controller in controlling the slow dynamics are less accurate compared to MPC. The PD outer loop needs to be improved since the obtained path is not as good as the results obtained with the MPC control applied to quadrotor slow dynamics. Both proposed solutions complete the path in the given time, however, the MPC results are close to the reference during all the simulation, while the PD presents a meaningful error. Moreover, looking at the PD results, at each change of direction the system requires more time to stabilize compared to MPC behaviour.

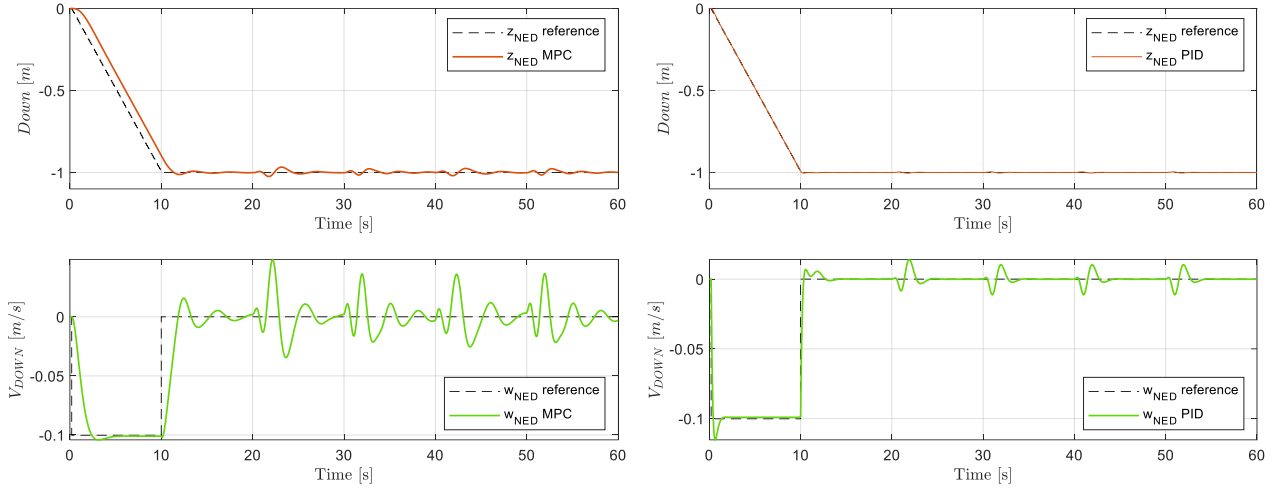


Figure 70: comparison of square pattern time response z_{NED} and w_{NED} ; on the left MPC altitude control and on the right PID altitude control.

Figure 70 shows the response of both proposed solutions in the tracking of a reference altitude. While the trajectory in the xy plane, shown in Figure 69, shows significant discrepancies, the reference altitude tracking of the PD and the MPC is comparable. The solution based on MPC (shown on the left) has a slower z_{NED} response, showing slight fluctuations at the changes of direction. Moreover, looking at the w_{NED} trend, despite the smaller overshoot, the behaviour is much more unstable compared to the PID solution (shown on the right). In conclusion, the MPC presents more fluctuations with smaller velocity overshoot, compared to PID controller, however, it is able to follow the reference z_{NED} even though some slight fluctuations are present at changes of direction. The results of the linear velocity u_{NED} and v_{NED} , shown in Figure 71, outlines the unstable behaviour of the PD controller in the velocity tracking. As can be seen, both the u_{NED} and v_{NED} required time to stabilize around the reference. The change of velocity is faster in the PD controller; however, the MPC proves better performances in reference following.

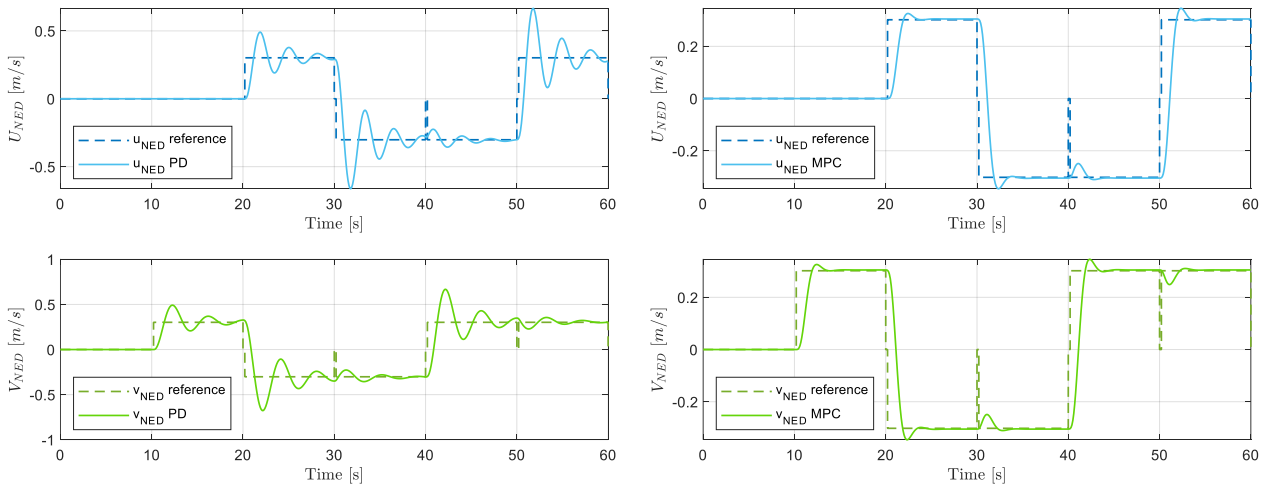


Figure 71: comparison of square pattern time response u_{NED} and v_{NED} ; on the left PD velocity control and on the right MPC velocity control.

5.1.3.2 – Butterfly Pattern

Table 11: butterfly pattern reference waypoints and time of arrivals.

Waypoint	Time of Arrival	Waypoint Position
1	0	[0,0,0]
2	10	[0,0,-2.5]
3	20	[3,-3,-2.5]
4	30	[3,3,-2.5]
5	40	[-3,-3,-2.5]
6	50	[-3,3,-2.5]
7	60	[3,-3,-2.5]

Table 11 lists the vectors of waypoints and time of arrivals provided to the MATLAB function *trajectory planner* for the butterfly pattern following. As described in the previous Section 5.1.3.1, two controllers are considered, respectively the first one based on a PD position control, while the second one is based on an MPC position control. Figure 72 shows an overall view of the *butterfly* pattern execution, comparing the two studied solution, PD on the left and MPC on the right.

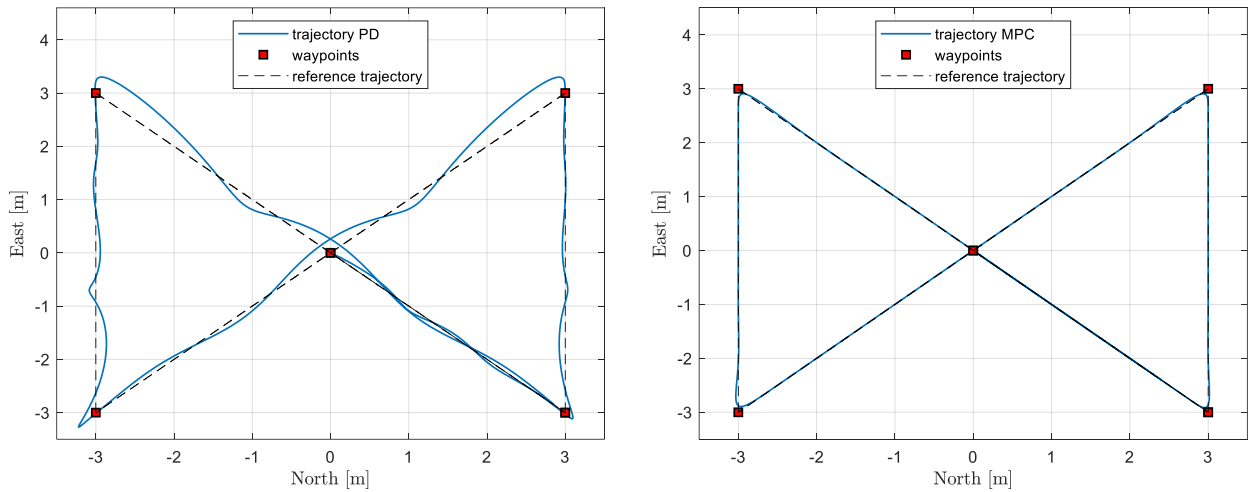


Figure 72: comparison of butterfly pattern path following North-East plane; on the left PD controller and on the right MPC controller.

As can be noticed in Figure 72, the PD controller position response is not as accurate as the MPC. Both proposed solutions complete the path in the given time, however, the MPC results are closer to the reference during all the simulation, while the PD presents larger overshoots at changes of direction. Moreover, the MPC has a better behaviour, not only in terms of position, but also in the attitude parameters with an accurate and stable behaviour. The behaviour of the Euler angles and angular rates is shown in Figure 73 (first proposed solution) and Figure 74 (second analysed solution). Both controllers display similar behaviour. The attitude based on a MPC controller is significantly more stable with overshoot only in correspondence of change directions, while the attitude control based on a PD algorithm is not as stable as the MPC, presenting higher peaks values and oscillations around the zero reference. If the peaks are analysed, the largest pitch and yaw variation are in proximity of the end of the simulation, this depends on how fast the quadrotor is going during the specific phase of the manoeuvre.

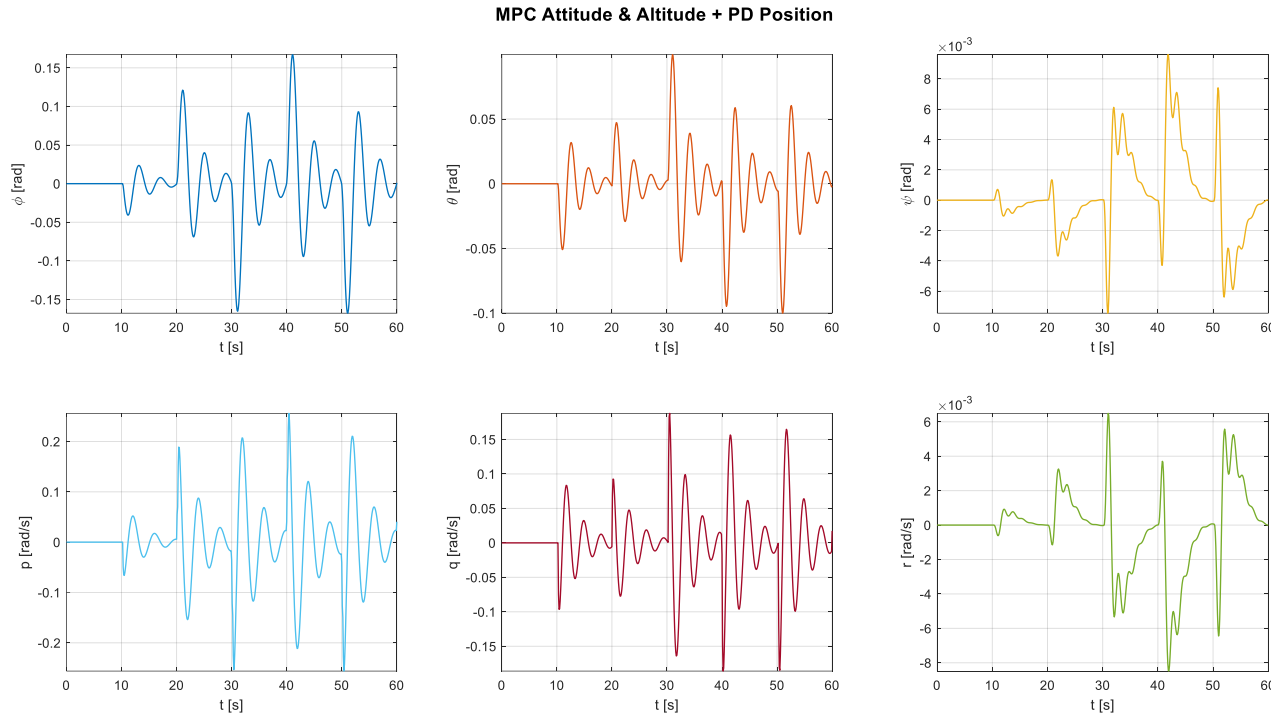


Figure 73: butterfly pattern time response of Euler angles ϕ, θ, ψ and angular rates p, q, r of a system based on a MPC controller for the attitude and altitude control and a PD controller for the position control.

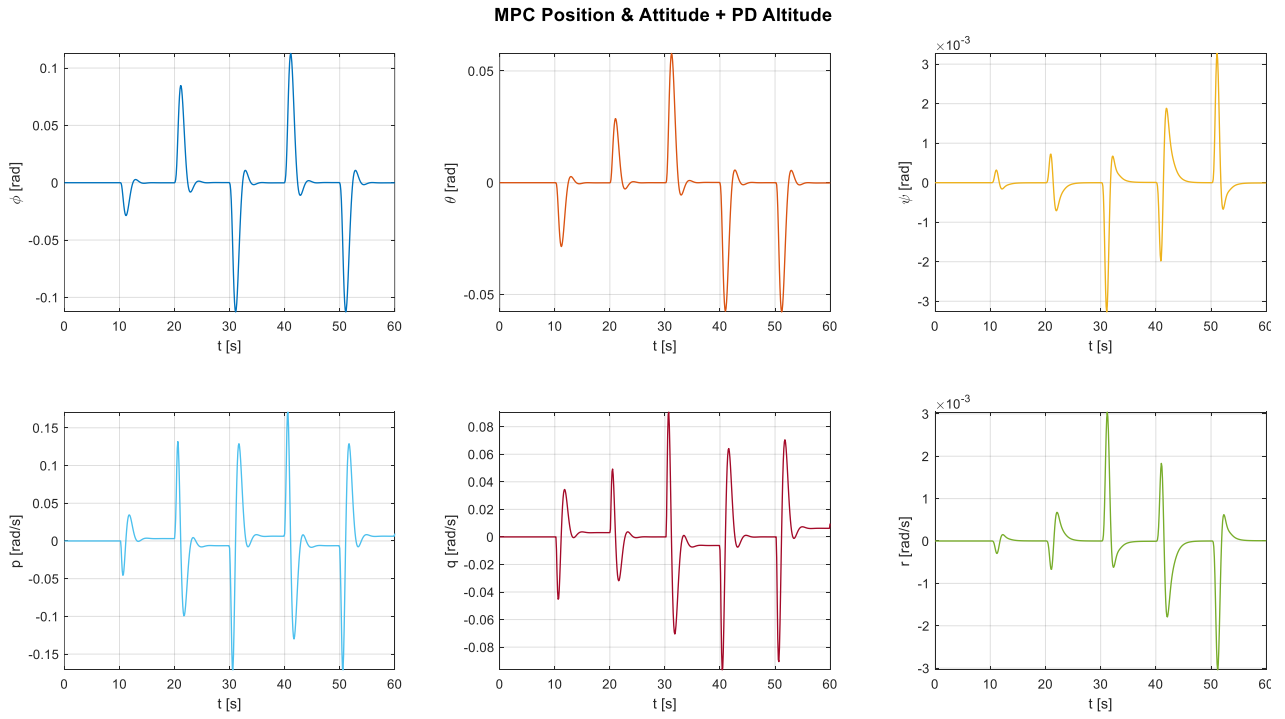


Figure 74: butterfly pattern time response of Euler angles ϕ, θ, ψ and angular rates p, q, r of a system based on a MPC controller for the attitude and position control and a PID controller for the altitude control.

5.1.3.3 – Snake Pattern

Table 12: snake pattern reference waypoints and time of arrivals.

Waypoint	Time of Arrival	Waypoint Position
1	0	[0,0,0]
2	10	[0,0,-2.5]
3	20	[3,-3,-2.5]
4	30	[3,0,-2.5]
5	40	[-3,0,-2.5]
6	50	[-3,3,-2.5]
7	60	[3,3,-2.5]
8	70	[3,6,-2.5]
9	80	[-3,6,-2.5]
10	90	[-3,-3,-2.5]
11	100	[3,-3,-2.5]

Table 12 lists the vectors of waypoints and time of arrivals provided to the MATLAB function *trajectory planner* for the snake pattern. In this section the complexity of the flight pattern is increased intending to test the proposed controller capabilities in path following. Considering the ToA vector and the number of waypoints, it can be noticed that the system is subject to a different reference each 10[s], hence it must be fast, responsive, and precise to track all waypoints in the predefined time of simulation.

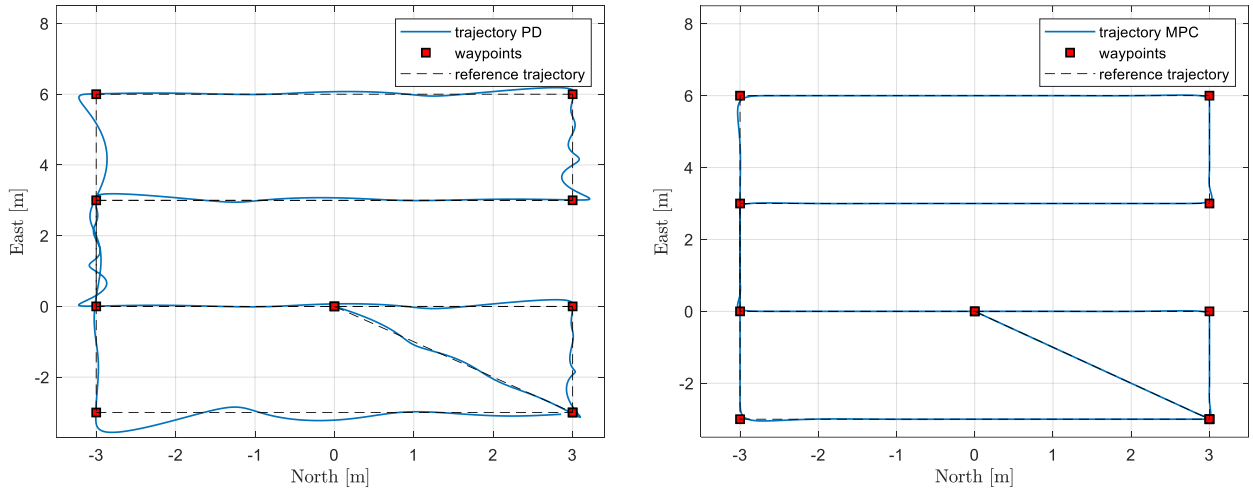


Figure 75: comparison of snake pattern path following North-East plane; on the left PD controller and on the right MPC controller.

In accordance with the results presented in Figure 75, the trajectory is complete by both controllers in the predefined given time. As expected, the response of the PD controller is not as good as the MPC. This behaviour depends on the rise in the number of reference waypoints and the time given to track each of them. The MPC presents less overshoot and oscillations than the PD which requires longer time to stabilize around the reference. However, at the changes of direction (such as waypoint [3,3,-2.5] or [-3,6,-2.5]) even the MPC presents slight overshoots which depend on how fast the quadrotor is going. In conclusion, in accordance with the aforementioned properties, the snake pattern is better performed with a system based on an MPC controller than a PD controller.

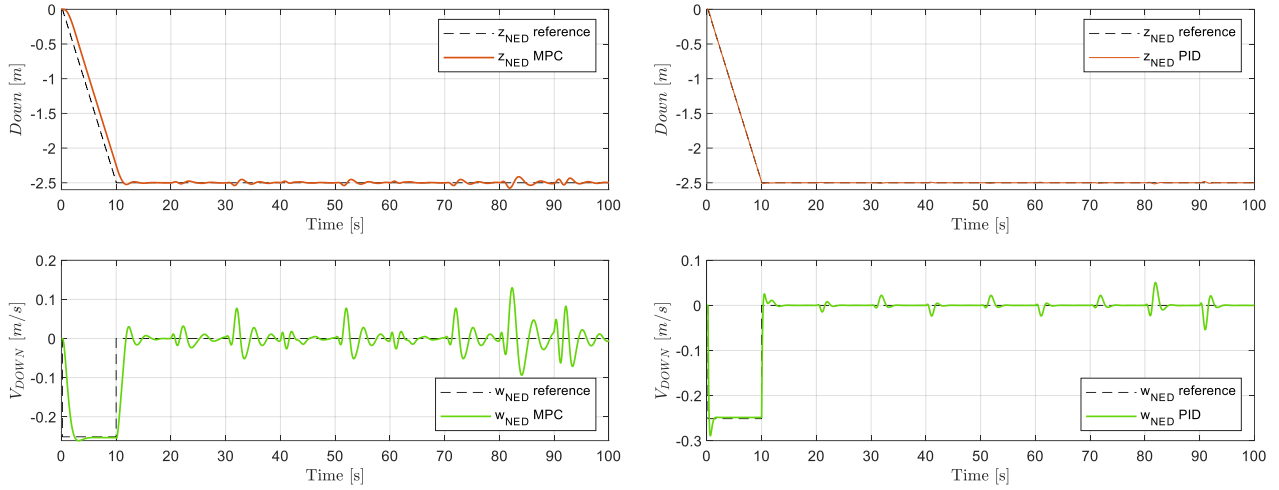


Figure 76: comparison of snake pattern time response z_{NED} and w_{NED} ; on the left MPC altitude control and on the right PID altitude control.

Figure 76 shows the comparison of the proposed controller results in the tracking of a reference altitude. The figures on the left present the results obtained for the MPC-based attitude control; while figures on the left an altitude control based on a PD algorithm. The aim of these figures is to outlines the behaviour of the simulated system in response to an out-of-plane manoeuvre. As can be noticed, the z_{NED} is close to the reference in both controllers; however, the w_{NED} needs to be further investigated. The PID controller has faster behaviour and higher overshoot than the MPC. On the other hand, the MPC response is slower and fluctuates around the reference provided by the trajectory planner. Figure 77 inspects the velocity reference signals and the time response of both proposed systems during the snake pattern manoeuvre. It shows that the PD solution is faster during velocity changes but not as good as the MPC in the tracking of the reference. On the other hand, the MPC presents better performance in terms of path following but is less responsive in the tracking of reference velocities.

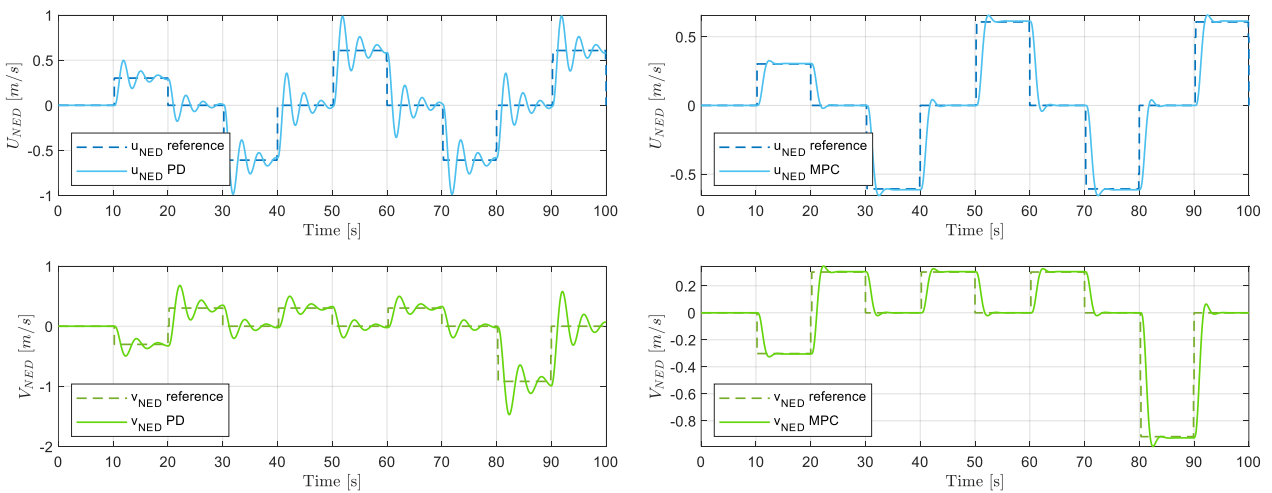


Figure 77: comparison of square pattern time response u_{NED} and v_{NED} ; on the left PD velocity control and on the right MPC velocity control.

5.1.3.4 – Snail Pattern

Table 13: snail pattern reference waypoints and time of arrivals.

Waypoint	Time of Arrival	Waypoint Position
1	0	[0,0,0]
2	5	[1,0,0]
3	10	[1,1,0]
4	15	[0,1,0]
5	20	[0,0.3,0]
6	25	[0.8,0.3,0]
7	30	[0.8,0.6,0]
8	35	[0.3,0.6,0]
9	45	[0.2,0.6,0]

Table 13 shows the elements of the vector $ToA \in \mathbb{R}^n$ (listed under Time of Arrival column) and the components of the matrix of waypoints $WyP \in \mathbb{R}^{n,3}$ (listed under Waypoint Position column), where $n = 9$ is the number of waypoints that determine the path. In this section, the complexity of the path following task is further improved by reducing the dimensions of the reference path. Moreover, the time between consecutive waypoints is halved to 5[s]. As shown in Figure 78 the overall simulation is performed within a one-by-one-meter area. Lastly, the behaviour of the controller in presence of obstacles is presented.

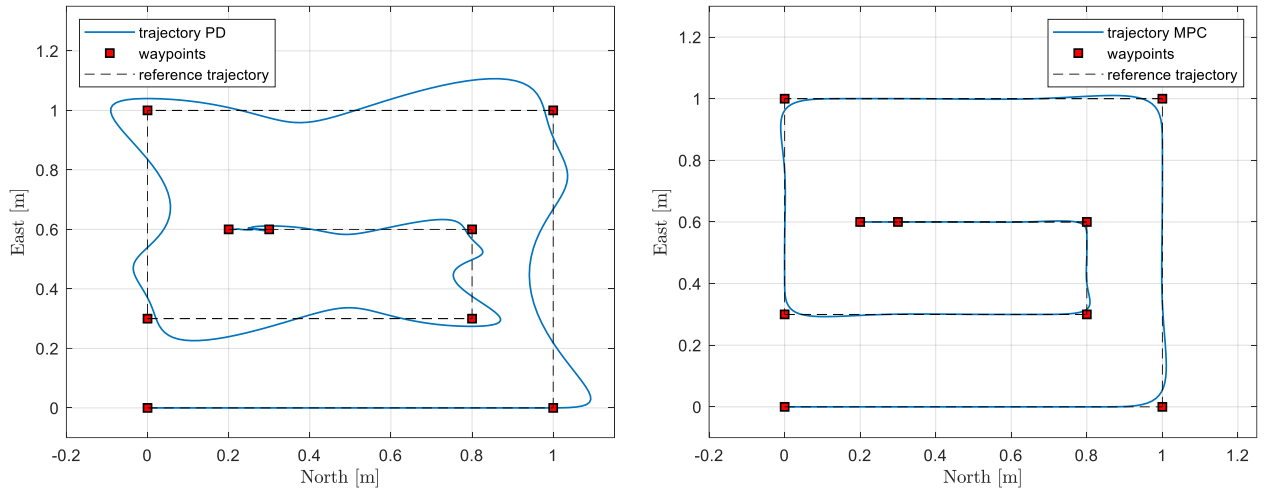


Figure 78: comparison of snail pattern path following North-East plane; on the left PD controller and on the right MPC controller.

As expected, the overall controller's performance have been affected by the reduction of path dimensions. The PD controller presents meaningful overshoots and significant position errors; hence, the quadrotor slow dynamics PD controller needs to be further improved. On the other hand, despite the shrink of path dimensions and the imposed hard timing constraints, the MPC presents a responsive behaviour. However, the closeness of waypoints and the reduced time of arrival generate fast changes of directions, thus not all waypoints are touched during the simulation (such as waypoint [1,0,0] or [0,1,0]). In conclusion in presence of path following tasks which require fast manoeuvres within restricted area, such as indoor applications, the combination of MPC controller for the translational and rotational dynamics and the PID controller for the altitude dynamics is the best solution.

5.1.3.5 – Obstacle Avoidance Results: MPC and APF based controller

Once studied the performance of the designed controllers, the problem of obstacle avoidance is addressed. The following results are based on the same system used in the above paragraphs (MPC for position and attitude control and PID for altitude control) by the addition of the *Artificial Potential Field* APF algorithm. The results shown in Figure 79 and Figure 80 have been obtained considering spherical obstacles, dimensioned in accordance with path dimensions. In all the graphs the same representations scheme is used: obstacles are blue circles, while the green dashed lines outline obstacles tolerances. Different situations have been considered (closest obstacles, obstacle in critical position, obstacle near waypoints); however, the designed algorithm provide reliable results. Depending on the quadrotor's positions and velocities, the obstacles are always avoided correctly. Moreover, the *square* and *snake* patterns (respectively Figure 79 left and Figure 80 left) don't complete the path in the given time; this is a consequence of states and path adaptation to obstacles.

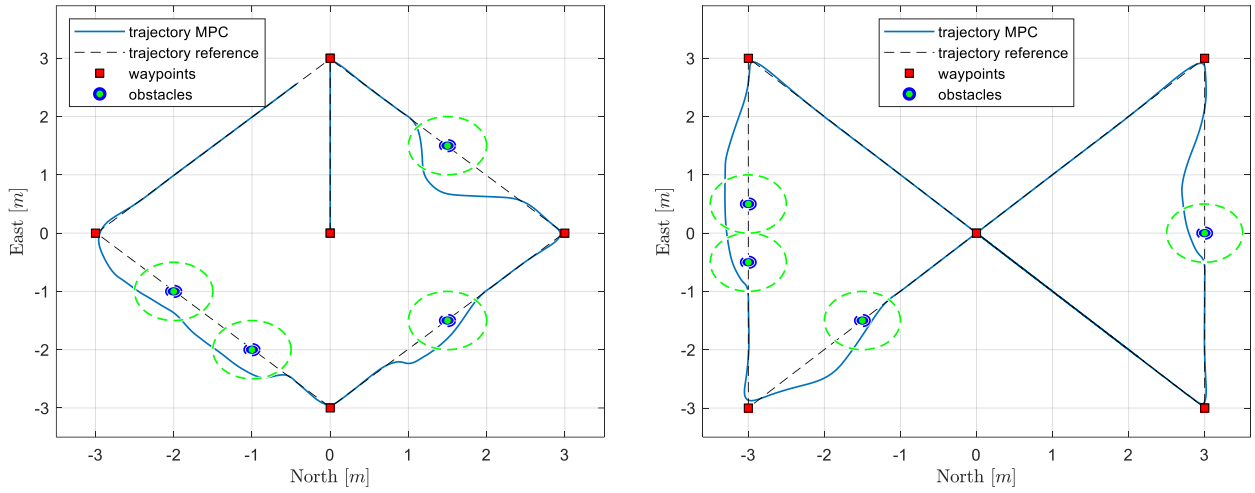


Figure 79: path following and obstacle avoidance North-East plane MPC controller (time of simulation 60[s], MPC prediction horizon 30 step). On the left square pattern; on the right butterfly pattern.

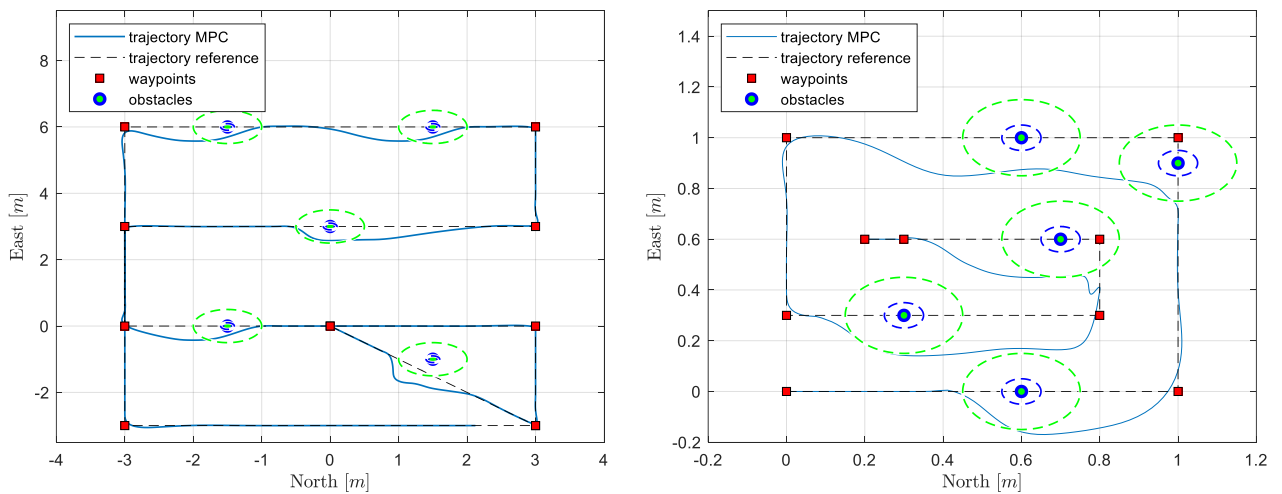


Figure 80: path following and obstacle avoidance North-East plane MPC controller (time of simulation 100[s] snake and 45[s] snail, MPC prediction horizon 30 step). On the left snake pattern; on the right snail pattern.

Unreal Engine® Simulations and AirSim Simulator

As mentioned in *Chapter 1*, the development of software-based system has a high cost of verification and validation. The use of simulated environment to test and verify system performance can reduce these costs and optimize engineer's workflow without resorting to prototyping. According to (Dekker, 1988) a simulation environment is defined as a programming environment of a computer, that is dedicated to system simulations and that takes care for flexible and intelligent interfacing between a user and the system to be studied. It is made clear that a simulator has to support different tasks, such as modelling description, experimentation, knowledge handling and reporting, which ask for high performance computing. That having been said, there are many simulators commercially available; in this thesis *AirSim* based on *Unreal Engine*® software framework is presented. *Unreal Engine* is a software framework primarily design for the development of video games, created in 1998 by *Epic Games*®. Thanks to its graphic capabilities, it is used in numerous applications, from video games design to film making and 3D photorealistic objects animation. On the other hand, *AirSim* is an open-source cross-platform simulator for drones and ground vehicles, built on *Epic Games Unreal Engine*, developed by Microsoft. It is a cross-platform that supports *Software-In-the-Loop* and *Hardware-In-the-Loop* simulations with the most popular flight controllers, such as *PX4* and *Ardupilot*. It is worth mentioning that *AirSim* is developed as an *Unreal* plugin; hence, it can be simply dropped into any *Unreal* environment for physically and visually realistic experiment with deep learning, computer vision and reinforcement learning algorithms for autonomous vehicles. This section is structure as follow: firstly, a brief overview of *Simulink UAV* toolbox for *Unreal Engine* is provided; then the development and testing of a custom flight controller, based on a MPC attitude controller in *AirSim* is presented.

6.1 – Simulink 3D Animation and UAV Toolbox

Simulink 3D Animation® package provides apps for linking Simulink model and MATLAB scripts to 3D graphic objects. This package can be used to visualize and verify dynamic system behaviour in a virtual reality environment. As an example, the *Simulink 3D Animation* toolbox enables the animation of 3D graphics objects such as car, vehicles, quadrotor, multicopter and fixed wing vehicles in virtual reality environments, such as *Unreal Engine*. The links between Simulink model and MATLAB algorithms allows the use of sensors to sense collisions and other events in the virtual world and feed them back into MATLAB and Simulink to be processed and analysed.

Another useful Simulink library for UAV behaviours visualization and flight controller testing is *UAV Toolbox*. This is a Simulink 2021b toolbox that provides tools and applications for the design, simulation and testing of UAVs. This toolbox is used to design customized autonomous flight algorithms and flight controllers that can be tested and analyse in photorealistic 3D environments. Thanks to its connections it is possible to generate personalized flight controller for desktop simulation and *Hardware-In-the-Loop* (HIL) testing. Moreover, it is possible to simulate on-board avionics and receive as feedback sensors information such as camera images,

lidar's point-cloud, *Inertial-Measurement-Unit* (IMU) measurements and GPS data. The toolbox is developed in order to support both *C* and *C++* code generation for rapid prototyping in *Hardware-In-the-Loop* testing and standalone hardware, like *Pixhawk* autopilot.

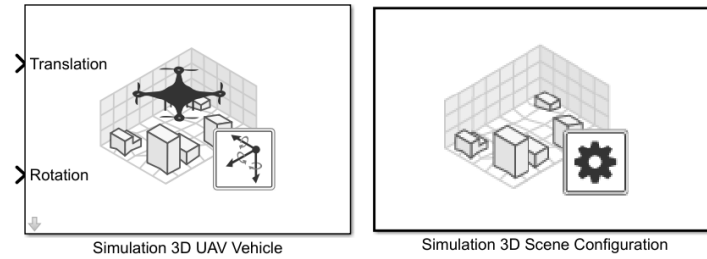


Figure 81: UAV Toolbox blocks used for co-simulation framework that links Simulink and Unreal Engine from Epic Games

UAV Toolbox provides a co-simulation framework that models driving algorithms in Simulink and visualizes their performance in a virtual simulation environment. This environment is the aforementioned *Unreal Engine* by *Epic Games*. Figure 81 shows two blocks that are used and connected with UAV non-linear model to perform realistic closed loop simulations. These blocks are *Simulation 3D UAV Vehicle* and *Simulation 3D Scene Configuration*. During each simulation, the *Unreal Engine* simulation block *Simulation 3D UAV Vehicle* initializes the vehicle information and sends its translation and rotation signals to the simulation 3D Scene Configuration block. The latter receives vehicle data and implements a 3D simulation environment that is rendered by using the *Unreal Engine* platform. This block is built such as it can simulate a series of different prebuilt scenes, or it is possible to create new one using *UAV Toolbox Interface for Unreal Engine Projects*.

6.1.1 – UAV Toolbox Interface for Unreal Engine

Once designed the flight controller algorithms, Simulink co-simulates the algorithms in the visualization engine. The workflow of communications between Simulink and *Unreal Engine* can be summarized as follow. First, *UAV Toolbox* configures the visualization environments, specifically the scene capture from cameras and initial object position, then it determines the next position of the objects using the simulated environment feedback. This logic can be summarized with the following scheme:

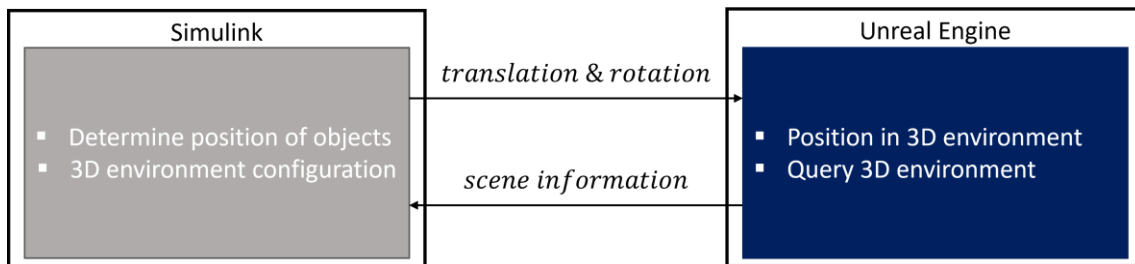


Figure 82: work logic diagram Simulink and Unreal Engine communication.

As mentioned before, during each simulation, the *UAV* toolbox in Simulink sends information about UAV translation and rotation to the *Simulation 3D Scene Configuration* block, which receives vehicle data and sends it to the sensor block. The latter processes the information in order to correctly visualize the quadcopter. It is worth mentioning that there is a *priority* property which controls the execution order of this

actions. By default, Simulink has the following order of priority: *Simulation 3D UAV Vehicle*, *Simulation 3D Scene Configuration* and lastly *Simulation 3D Camera*, *Fisheye Camera* and *Lidar*.

6.1.2 – Unreal Engine 3D Results Visualization

This section is organised as follow: first, a brief overview of Simulink-*Unreal Editor* configuration procedure is provided, then an example of path following and obstacle avoidance task, running on a customize scene, is presented. Once installed the *UAV Toolbox Interface for Unreal Engine Projects* it is possible to modify or create customize scenes using the *Unreal Editor* from *Epic Games*. Custom scenes allow the co-simulation in both Simulink and *Unreal Editor* (Figure 83) so that the user can directly modify the object positions or the environment settings between simulations runs. The *UAV toolbox for Unreal Engine* consists of two files: the first one is an *Unreal Engine* project called '*AutoVrtlEnv.uproject*' which includes the editable versions of many prebuilt scenes that the user can select from the *scene source* settings parameter of the block; the other file contains two plugins, which are '*MathWorkSimulation.uplugin*' and '*MathWorksUAVContent.uplugin*'. Both of them are needed to establish the connection between Simulink and the virtual environment's editor, hence they must be installed in the plugin folder of the local installation folder of *Unreal Engine*. It must be mentioned that, since the latest released version of *Unreal Engine* v4.27 is not compatible with *UAV Toolbox Interface for Unreal Engine Projects*, it has been necessary to install an older version of the software v4.25.

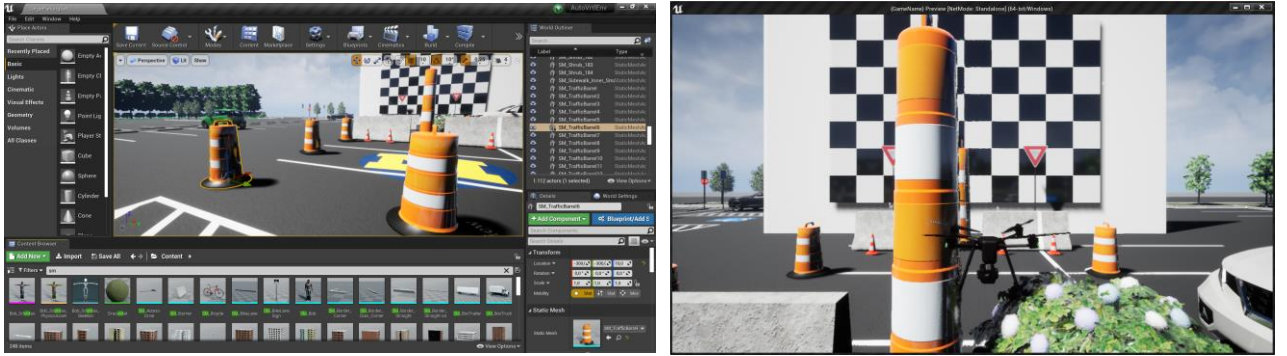


Figure 83: On the left *Unreal Engine Editor* main page configuration; on the right mobile preview of path following and obstacle avoidance task of the simulated quadrotor running in Simulink *UAV Toolbox* and visualized in a customize scene in *Unreal Engine* (Microsoft Windows 10).

Once installed the aforementioned plugins, the MPC-based Simulink model presented in Section 5.1.1.2 has been adapted to co-simulate with *Unreal Engine* introducing the *Simulation 3D UAV Vehicle* and *Simulation 3D Scene Configuration* blocks. By setting the parameter *scene source* to '*Unreal Editor*' and the project parameter to the path containing the file '*AutoVrtlEnv.uproject*' the connection can be established. Lastly, the *Sim3dLevelScriptActor* level blueprint used by *UAV Toolbox*, which is the algorithm that controls how objects interacts with the environment, must be associated with the current project.

After correctly done the steps, the customized scene has been created. Figure 85 shows a perspective view of the customized environment in *Unreal Engine Editor*; the area of manoeuvre has been delimited using traffic cones while wire road-centre barriers and bushes have been used as obstacles to avoid. The scene has been designed in accordance with the obstacles position provided in the MATLAB script. Moreover, the MPC parameters (prediction horizon H_p , weighting matrices Q_{MPC} , R_{MPC} , inputs constraints u_{min} , u_{max} and initial states x_0), the obstacles positions $x_{obs} \in \mathbb{R}^3$, the APF parameters and the path waypoints have been defined in the MATLAB script (the parameters values are the same of the model presented in Chapter 5). The

results of the Simulink simulation, shown in Figure 84, have been visualized using *Unreal Editor* as interface (Figure 83). As expected, the quadrotor successfully completes the path in the given time avoiding all the obstacles in North-East, such as North-Down planes.

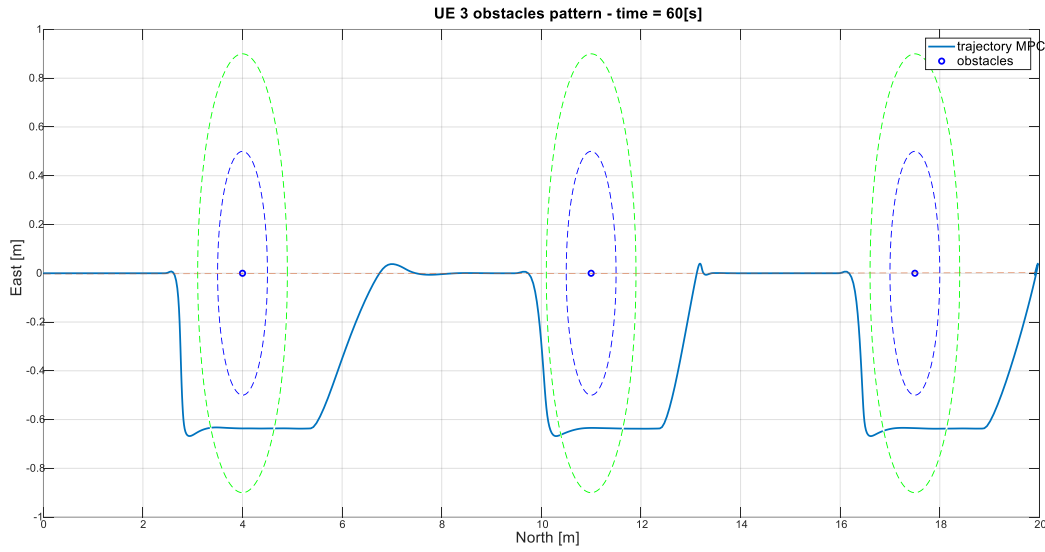


Figure 84: MATLAB results showing North-East plane quadrotor behaviour in the avoidance manoeuvre of 3 obstacles using an MPC based controller. Obstacle position $x_{obs} = [4,0,0; 11,0,0; 17,0,0][m]$ with dimensions $R_{obs} = 0.5[m]$ and a tolerance of $\eta_0 = [0.4,0.4,0.4][m]$



Figure 85: Unreal Engine Editor high resolution screenshot of customize obstacle avoidance path. The disposition of obstacles and the area of manoeuvre have been defined in accordance with data provided by MATLAB script.

6.2 – AirSim Simulator

As mentioned before, AirSim is an open-source simulator developed by *Microsoft AI & Research* as a plugin for *Unreal Engine* framework. It was publicly made available in February 2017. AirSim aims to be a useful tool in the development of autonomous vehicles and the gathering of training data for *Artificial Intelligence* (AI) studies, trying to narrow the gap between simulations and reality. The plugin-based architecture of AirSim, allows its use in any environment developed by *Unreal Engine*, such as those available in the marketplace or customize ones. The *Unreal Engine* platform provides open source code access, making easier the interaction with visually realistic graphics by offering modern graphics features such as photometric lights, planar reflections, and lit translucency. An example of *Unreal Engine* realistic graphic is shown in Figure 86. However, it must be noticed that, due to the young age of AirSim, there is little to none-research based on simulator, hence the studies of its architecture, protocols of communications and settings configurations have been carried out studying AirSim source code and open-source flight controllers (FC) algorithm.



Figure 86: Unreal Engine camera capture of running quadcopter simulation using Python API commands implemented in Visual Studio Code. Environment name: ZhangJialie. Ground Unit: Windows 10 laptop, Intel Core i7, graphic processor unit NVIDIA GeForce GTX970M.

This section presents a brief overview of the main features of AirSim, such as APIs commands and ways of communicating with AirSim relevant to FC implementations; moreover, it is analysed a modified open-source PID-based autopilot written in C language, which has been tested and modified, replacing the pre-existent PID attitude controller with the MPC algorithm implemented in *Chapter 5.1.1.2*. Lastly, the results of the custom made PID and MPC based flight controller are provided.

6.2.1 – Introduction to AirSim

Doing some research on the features of AirSim and its different ways of communicating, determine the possibility of implementing a custom flight controller in C, able to control the quadrotor in the simulator. Before proceeding with the main AirSim commands and settings, a schematic illustration of simulator architecture is provided. The architecture consists of six core elements: the *vehicle model*, the *environment model*, the *sensors models*, *physical engine*, *rendering engine* and a public *Application Programming Interface*

(API) layer. A high level overview of AirSim architecture is shown in Figure 87. In accordance with the information provided by the developers of AirSim, running a simulation, results in:

- the *flight controller* block being continuously fed with sensors data from the *sensor model* block;
- by using these data, the FC can compute the actuator signal, which is handled as input by the *vehicle model*. The *vehicle model* contains all the physical information of the vehicle, such as mass, inertia, thrust coefficient, drag coefficient, and computes the forces F and torques τ generated by the virtual rotors in accordance with the vehicle information specified by the user.

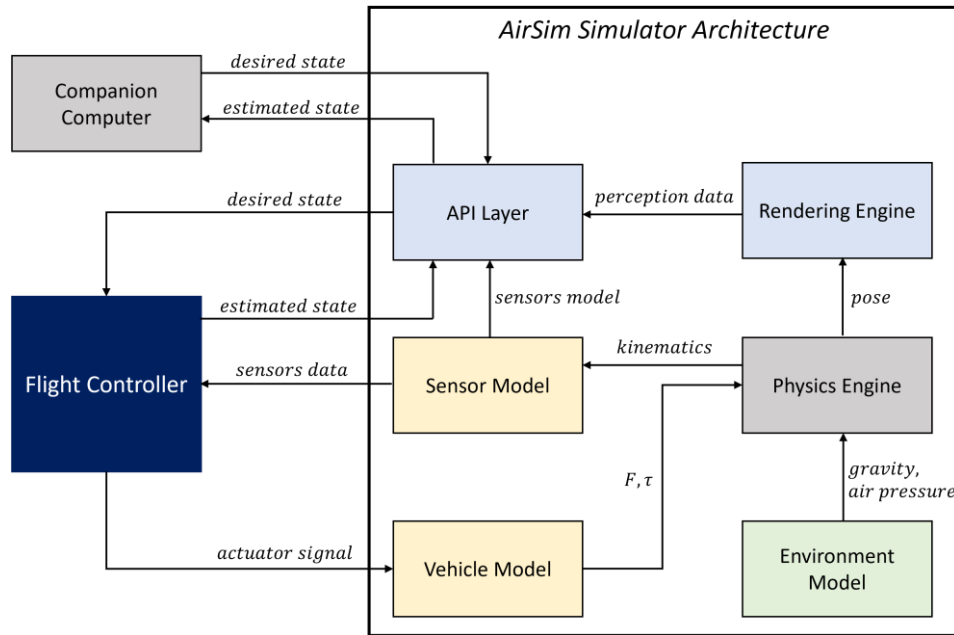


Figure 87: illustration of AirSim simulator architecture. The scheme shows a block diagram of the 6 main elements and their interactions.

- The forces and moment signals are passed to the *physics engine* which combines this information with the data provided by the *environment model* (such as gravity force, air pressure and magnetic field) providing as outputs the kinematic states of the vehicle and all the bodies that are simulated in the virtual environment;
- lastly, these states information are handled by the *sensor model*, which processes the kinematics into sensors data, feeding back the flight controller (the loop is closed).

On the other hand, the desired states can be provided in two ways: using a *Remote Controller* (RC) which sends desired states directly to the FC; or a *Companion Computer* (CC) which communicates via the API layer with the use of *Remote Procedure Call* (RPC). Moreover, the CC allows more complex tasks such as mission planning or path planning determining the desired sequence of waypoints.

6.2.1.1 – Remote Procedure Call and API Commands

The term *Remote Procedure Call* (RPC) is used to describe a computer programming invoking a procedure to execute in a different program (not necessarily limited to the current computer). The idea behind RPC consists of a *client-server* based communication procedure. The *clients* creates a procedure which is encoded and sent over the ethernet (or serial) to a *server* that decodes the message and processes the requested

procedure. Once the task is completed, the server returns a signal as a response to the client. AirSim uses an API accessible through RPC on localhost port 41451 by default (it is possible to change the port values in the simulator settings file 'settings.json'). An example of AirSim RPC procedure based on MAVLink library is shown in Figure 88. The flight controller creates the messages which are encoded to binary form and transferred over the ethernet or serial to be received by the *Ground Control Station* GCS which decodes and parses the messages. The GCS, in turn, sends messages to the flight controller with a similar procedure.

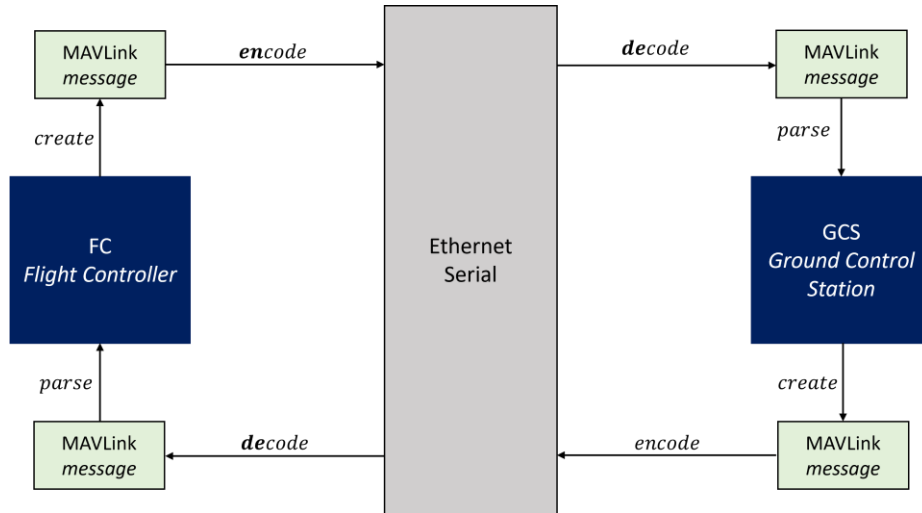


Figure 88: overview of AirSim RPC procedure based on MAVLink library.

As mentioned before, AirSim allows the customization of different settings through the changing of its settings file managed in a *Java-Script-Object-Notation* (JSON) called 'settings.json', which is located on the path of installation of AirSim. This file allows the definition of the model vehicle, referred to as *SimMode* (*multirotor* or *car*), the sensors to simulate (such as *Lidar*, *GPS*, *Camera*) and the communication settings, which is defined as *DefaultVehicleConfig*. The latter tells AirSim to use the default settings based on *SimpleFlight* communication or to switch the way of communication to *PX4* autopilot, allowing the communication by sending and receiving MAVLink messages. In other words, AirSim exposes two APIs: the first one implements its functionalities by using the built-in FC *SimpleFlight*; while the second one, *PX4*, communicates through the usage of MAVLink messages allowing individual motors control. It must be noticed that the 'settings.json' schema has been changed with AirSim v1.2; for the sake of clarity, in this thesis it has been used the older v1.1 schema to allow PX4 autopilot and MAVLink based communication.

```

{
  "SettingsVersion": 1.2,
  "SimMode": "Multirotor",
  "ClockSpeed": 1.0,
  "Vehicles": {
    "SimpleFlight": {
      "VehicleType": "SimpleFlight",
      "DefaultVehicleState": "Armed",
      "EnableCollisionPassthrogh": false,
      "EnableCollisions": true,
      "AllowAPIAlways": true,
    }
  }
}

```

```

{
  "SettingsVersion": 1.2,
  "SimMode": "Multirotor",
  "ClockSpeed": 1.0,
  "DefaultVehicleConfig": "PX4",
  "PX4": {
    "FirmwareName": "PX4",
    "UseSerial": false,
    "UdpIp": "127.0.0.1",
    "UdpPort": 14560,
    "SitiIp": "127.0.0.1",
    "SitiPort": 14556,
    "LocalHostIp": "127.0.0.1",
  }
}

```

Figure 89: on the left file settings.json compatible with latest AirSim v1.6 configured for 'SimpleFlight' vehicle communication; on the right settings.json compatible with AirSim v1.2 (or earlier) configured for 'PX4' communications protocol based on MAVLink messages.

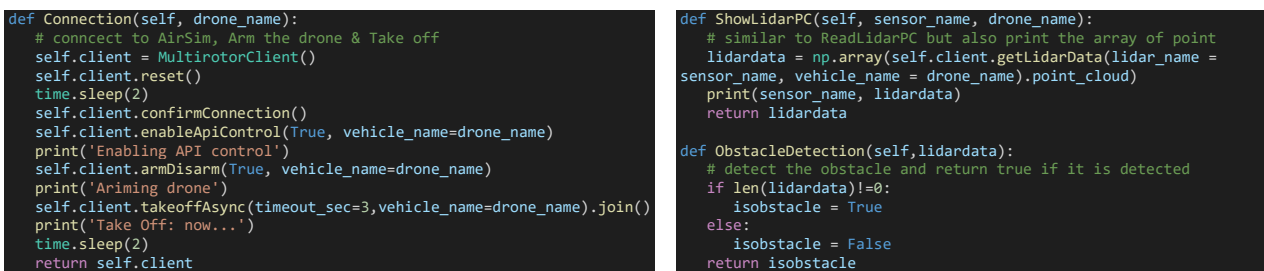
Figure 89 shows two examples of settings files; on the left it is presented a *SimpleFlight* vehicle configuration compatible with the latest version of AirSim, on the right a *PX4* vehicle configuration based on MAVLink messages, compatible with AirSim v1.2 or earlier. By setting *DefaultVehicleConfig* to *SimpleFlight* (left case) the flight controller performs default implemented procedures such as arm, disarm, take-off, land, hover and move to a specified position. Changing the *DefaultVehicleConfig* to *PX4* results in AirSim sending MAVLink messages over *User Datagram Protocol* (UDP) instead of serial. As shown in Figure 89 (right) the UDP port used by AirSim is specified at lines 8-9, while the SITL communication is defined at lines 10-11. The latter provides a version of PX4 running in Linux. Since the ground station used in this project is an octa core *Intel i7* ground laptop, running Windows 10 as operating system, the *Cygwin Toolchain Terminal* has been used to allow the PX4 Linux toolchain setup.

6.2.2 – AirSim Flight Controller

AirSim supports various flight controllers that can be either *Hardware-In-the-Loop* (HIL) or completely *Software-In-the-Loop* (SIL). Moreover, third party flight controllers can be used if they support communication through MAVLink-based messages. The default AirSim flight controller is called *SimpleFlight*. In this project both the default software-simulated flight controller provided by AirSim, and a customize flight controller, hereinafter referred to as *simple_flight_controller_c*, have been tested.

6.2.2.1 – Simple Flight Controller

The default API exposes multiple procedures provided by the default flight controller. The library used by AirSim is called *rpclib*. In this project the flight controller, *SimpleFlight*, capabilities have been tested developing a personalized *class* in Python, using *Visual Studio Code v1.61.2*. Different aspects have been tested from the simpler tasks, like arm, disarm and take-off, to the development of obstacles avoidance manoeuvre using lidar data. Some of the implemented class functions are shown in Figure 90.



```
def Connection(self, drone_name):
    # connect to AirSim, Arm the drone & Take off
    self.client = MultirotorClient()
    self.client.reset()
    time.sleep(2)
    self.client.confirmConnection()
    self.client.enableApiControl(True, vehicle_name=drone_name)
    print('Enabling API control')
    self.client.armDisarm(True, vehicle_name=drone_name)
    print('Arming drone')
    self.client.takeoffAsync(timeout_sec=3, vehicle_name=drone_name).join()
    print('Take Off: now...')
    time.sleep(2)
    return self.client

def ShowLidarPC(self, sensor_name, drone_name):
    # similar to ReadLidarPC but also print the array of point
    lidar_data = np.array(self.client.getLidarData(lidar_name =
    sensor_name, vehicle_name = drone_name).point_cloud)
    print(sensor_name, lidar_data)
    return lidar_data

def ObstacleDetection(self, lidar_data):
    # detect the obstacle and return true if it is detected
    if len(lidar_data)!=0:
        isobstacle = True
    else:
        isobstacle = False
    return isobstacle
```

Figure 90: on the left Python function used to establish a connection with the quadrotor, to enable the API commands, to arming the motors and take-off with a time delay of 3 [s] on AirSim; on the right Python function used to show lidar point cloud and detect obstacles.

6.2.2.2 – Custom Flight Controller

By using the settings presented in Figure 89 (right) AirSim will send MAVLink messages over UDP port 14560. In order to receive these messages, it is necessary to listen for them on the specify socket address, store them in a buffer and parse a MAVLink message for the buffer. All these functions are performed through a MAVLink function called *mavlink_parse_char()*, defined in the MAVLink library. This function receives as inputs the ID of the channel to be parsed and parse one byte at a time until a complete packet could be successfully decoded. When a complete message is received, it is copied into a variable. Once received, the

message must be classified depending on what type of message it is. This can be performed looking at the ID of the message and decoding its information using a specific function defined in the MAVLink library.

As an example, the *mavlink_msh_hil_gps_decode()* or *mavlink_msg_local_position_ned_decode()* are functions used to decode respectively the messages provided by the GPS and the positions provided by inertial reference frame *North-East-Down*. Like the reception of messages, in order to be able to send MAVLink messages, they must be created and encoded into a suitable format for transmission. These messages are sent using a function called *send_mavlink_message()*. The message library used to control the vehicle in AirSim is the *actuator controls pack* which packs the PWM signals of each motor into a message struct.

The first phase of this project consists of testing a PID-based open-source¹² flight controller written in C (Wachsler, 2018), called *simple_flight_controller_c*. The code is based on a socket communication on localhost UDP port 14560 and four threads, respectively:

- the first one is used to create a MAVLink message listener;
- the second one to send heartbeat messages;
- the third one to update motor states;
- the fourth to compute PID controls actions.

By starting a simulation using *Cygwin64* terminal, all the threads run simultaneously allowing the user to control the quadcopter. All the implemented command arm (starting motors), disarm (stopping motors), takeoff (PID computes commands to reach the desired altitude), flyto (PID computes commands to reach the desired latitude and longitude using GPS status), land (PID computes commands to land), GPS (print the current GPS values latitude-longitude-altitude), altitude (print current altitude) have been tested and they work all correctly.

The second part of the project aims to adapt the aforementioned PID controller in order to control the attitude using the MPC attitude controller, presented in *Chapter 5.1.1.2*. This task has been performed using the tool *Embedded Coder* from Simulink. Firstly, the Simulink model has been modified since not all Simulink blocks can be imported in C. The flight controller algorithm, *simple_flight_controller_c*, has been modified adding a fifth thread for the attitude control, based on MPC. Therefore, the final FC consists of a PID position controller and an MPC attitude controller. The implemented algorithm shows good behaviours in term of position however it has not been possible to stabilize quadrotor attitude. This behaviour can be a consequence of fluctuations in IMU measurements of Euler angles and not optimal tuning of MPC weightings matrices. As the overall thesis target was to test the implemented MPC-based flight controller in *AirSim* simulator, it is possible to say that the main target of this part of the thesis has been achieved; however, the results obtained aimed to be further studied in future advancements of this thesis.

¹² The flight controller is part of a repository *GitHub* by Joel Wachsler and Daniel Aros Banda. This repository contains code examples of how to communicate with AirSim by using the default API accessible through RPC and other message oriented API called MAVLink. Reference: (Wachsler, 2018)



Figure 91: illustration of AirSim environment Africa and customize FC connection procedure using Cygwin64 Terminal for Windows 10. The connection is established, and quadrotors motors are ready to be armed and take-off.

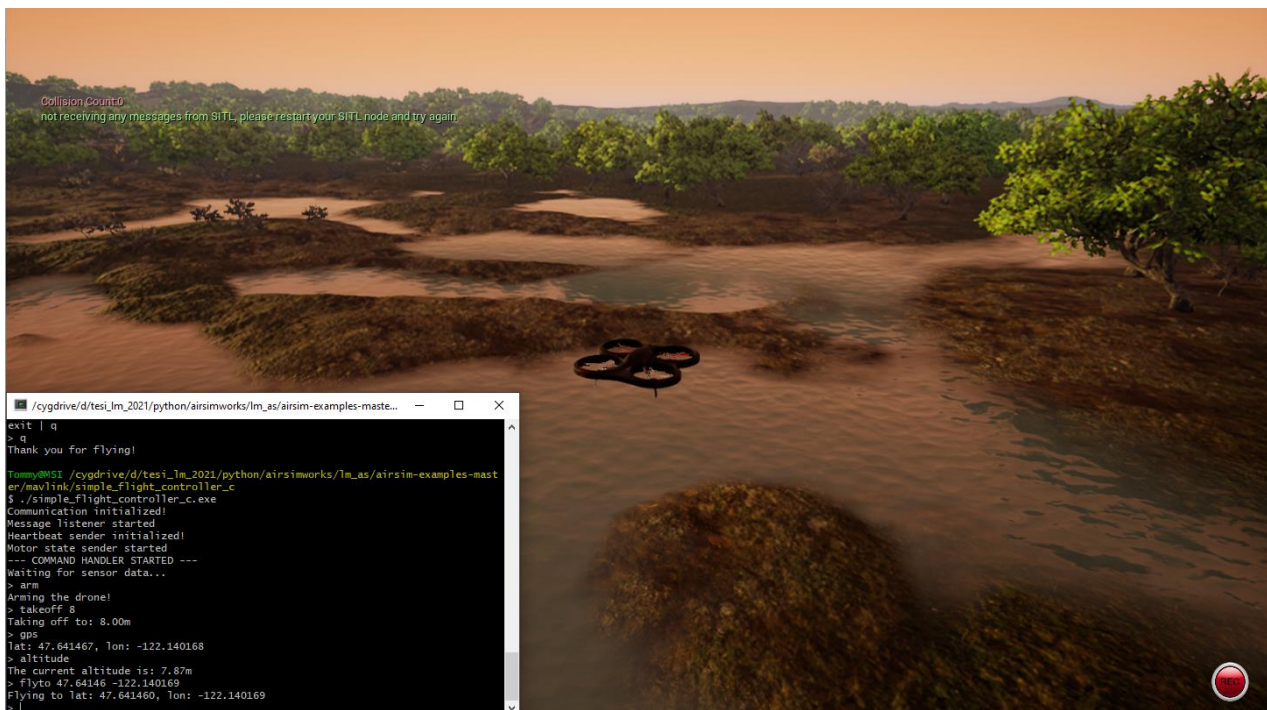


Figure 92: illustration of AirSim environment and customize FC connection using Cygwin64 Terminal for Windows 10. Quadrotor is flying and some examples of implemented commands results are printed on Cygwin64 Terminal interface.

Conclusions

This thesis aimed to identify an effective solution for the modeling and development of a customized quadrotor for indoor applications and to design an MPC based simulator. This work is the result of the integration of 6 months internship experience and research activities conducted in the fields of GNC algorithm and their embedded implementation on hardware components. As established, during the internship, carried out at the *ALTEN Italia s.p.a.* company (Milan), a small quadrotor UAV for indoor applications made of COTS hardware and software have been built. The cheap, fast, and accessible COTS technologies offer a rapid and effective solution for UAV development. The design process started with the definition of the customer drone operational needs, in this case the autonomous flight in indoor or GPS-denied environments. This work is meant to be a study and modeling of a customized quadrotor working architecture based on ultrasonic sensors, *Marvelmind Set HW v4.9-NIA*, as indoor positioning system. Thanks to the efficient integration of open source software and hardware, the main objective of indoor flight and autonomous flight has been accomplished. The objective has been nevertheless achieved thanks to the choice of the appropriate components, in particular the *Pixhawk 4 Mini* autopilot and the companion computers *Raspberry Pi 3* and *Nucleo STM32*, that allowed high computational power while keeping the overall weight of quadcopter relatively low. Many problems, related to hardware interfaces and software communications, have been faced during the assembly of the final configuration. Most of the software challenges have been overcome implementing purpose-oriented code (written in both *Python*, *C* and *C++* code languages).

The second part of the thesis focuses on the design of an MPC based simulator and its *Model-In-the-Loop* and *Software-In-the-Loop* simulations testing. In this work, two novel control architectures are presented. The first one is an MPC cascade controller based on a PD position controller and an MPC attitude and altitude controller. The second one is a full MPC based controller with a PID altitude controller. Both simulations are implemented in Simulink considering a non-linear dynamical model of quadrotor. Different controller's behaviours in path following tasks are compared. Moreover, MPC have been further improved introducing a guidance *Artificial Potential Field* algorithm. This configuration has been analysed in obstacle avoidance tasks, providing excellent results. The overall final results show that the MPC for position control has higher nominal performance in terms of path following, executing all the patterns in the given time of arrival ToA with high accuracy. On the other hand, the PD controller is less accurate and presents more fluctuations in the tracking of desired references. *Model-In-the-Loop* testing has been performed for all the pairings of control algorithm in order to achieve the best outcome. It has been concluded that the PD presents much rough response including higher overshoot, while the MPC shows a near optimal behaviour due to both smooth reference path following and precise variable control actions. Summing up, when an underactuated system such a quadrotor, is called for precise and accurate manoeuvres, the MPC provides the best performance. It embodies technical specifications into the control algorithm, and in particular no a-posteriori patches are

required to consider limitations on system's variable. So, an MPC-based system is a systematic design flow, being independent of the chosen model and performance constraint specifications.

Using *Simulink 3D Animation™*, *Simulink UAV* toolbox and *Unreal Engine Editor* (developed by *Epic Games*), the performance of the MPC based simulator have been visualized and monitored. This approach allows the user to have better understanding of quadrotor position and attitude response in proximity of reference waypoints or obstacles. Furthermore, the MATLAB and Simulink MPC controller have been converted in C language, using Simulink *Embedded Coder*. The latter allows MPC testing on a virtual autopilot in the open-source cross platform for drones *AirSim Simulator*. All MATLAB and Simulink models have been designed such that they can be adapted for code generation without heavy modifications.

As mentioned before, during the development of this project, great efforts have been put into the purpose of performing *Software-In-the-Loop* simulations in *Unreal Engine* environment using *AirSim*. This kind of testing is strongly desired previous to experimental flight tests with real platform. The main difficulties were faced in establishing a communication between the MPC algorithm, running in *Cygwin64 Terminal* for Windows, and *AirSim* simulator, which requires a *User Datagram Protocol* UDP-based link. Considering the absence of documents and research of *AirSim* architecture and settings, an open-source PID-based autopilot written in C has been modified, replacing the pre-existent PID attitude controller with the implemented MPC algorithm. The result is a custom flight controller in C, that controls a quadrotor in a virtual environment using a communication based on a MAVLink protocol. The controller shows good behaviours in terms of position; however, the quadrotor attitude cannot be stabilized. This behaviour can be a consequence of fluctuations in IMU measurements of Euler angles and not optimal tuning of MPC weightings matrices.

Future works are aimed to improve the simulation environment and test the controllers' performance in path following and obstacle avoidance, introducing noises and disturbances. Moreover, a more complex trajectory planner can be designed; such as a *Dubins* curves-based planner or a *Bézier* curves-based planner. Moreover, in future work, the MPC algorithm will be implemented on a real prototype in real time environment including the development of avoidance algorithm once obtained the output data from detection sensors. It should be also interesting to investigate the detection and avoidance decision mechanism when multiple obstacles are introduced. Lastly, as the overall target of this thesis is the autonomous flight in indoor environments, the equipment on-board must be as independent as possible from ground. This implies operating with sensors that do not require infrastructure deployment previous to flight.

Bibliography

- [1.] Aguirre, L. A. (2016, July 13). Controllability and Observability of Linear Systems: some noninvariant aspects. *IEEE Transactions on Education*, p. 33-39.
- [2.] Akhil M., M. K. (2012). Simulation of the Mathematical Model of a Quad Rotor Control System using Matlab Simulink. *Applied Mechanics and Materials*, 110-116.
- [3.] Akhil M., M. K. (2012). Simulation of the Mathematical Model of a Quad Rotor Control System using Matlab Simulink. *Applied Mechanics and Materials*, 2577-2584.
- [4.] Akkas Uddin Haque, A. E. (2020, January). UAV Autonomous Localization using Macro-Features Matching with a CAD Model.
- [5.] Alfian Ma'arif, A. A. (2021). Artificial Potential Field Algorithm for Obstacle Avoidance in UAV Quadrotor for Dynamic Environment. *IEEE International Conference on Communication, Networks and Satellite (Comnetsat)*.
- [6.] Arjomandi, M. (2007). Classification of Unmanned Aerial Vehicles. *Academia*.
- [7.] Atheer L. Salih, M. M. (2010). Flight PID Controller Design for a UAV Quadrotor. *Scientific Research and Essays Vol. 5(23)*, 3660-3667.
- [8.] B. Kada, Y. G. (2011). Robust PID Controller Design for an UAV Flight Control System. *Proceedings of the World Congress on Engineering and Computer Science* (p. Vol). San Francisco, USA: WCECS.
- [9.] Behzad Boroujerdian, H. G. (2018). MAVBench: Micro Aerial Vehicle Benchmarking. *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Fukuoka, Japan: IEEE.
- [10.] Bemporad, A. (2006). Model Predictive Control Design: New Trends and Tools. *Proceedings of the 45th IEEE Conference on Decision & Control*, San Diego, CA, USA.
- [11.] Bemporad, A. (2019). *Model Predictive Control*. Lucca, Italy: School For Advanced Studies Lucca.
- [12.] Bhushan, N. (2019). *UAV: Trajectory Generation and Simulation*. Arlington, Texas, USA: UTA Research Institute.
- [13.] Brian L Stevens, F. L. (2015). *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons.
- [14.] C. V. Rao, S. J. (1998). Application of Interior-Point Methods to Model Predictive Control. *Journal of Optimization Theory and Applications*, 723-757.
- [15.] Carminati, D. (2019). *Design and Testing of Indoor UAS Control Techniques*. Turin, Italy: Politecnico di Torino.
- [16.] Cezary, S. (2015). UAVs and Their Avionic Systems: Development Trends and Their Influence On Polish Research and Market. *Aviation*, 49-57.

- [17.] Chinedu Amata Amadi, W. S. (2018). *Design and Implementation of Model Predictive Control on Pixhawk Flight Controller*. Stellenbosh, South Africa: Faculty of Engineering, Stellenbosch University.
- [18.] Christiana Honsberg, S. B. (2020). *Battery Capacity*. Tratto da PV Education PVCDROM: <https://www.pveducation.org>
- [19.] Daniel Aros Banda, J. W. (2018). *Exploration of AirSim using C and Rust in the Context of SafetyCritical Systems*. Stockholm, Sweden: KTH Royal Institutu of Technology School of Electrical Engineering and Computer Science.
- [20.] Dekker, L. (1988). Simulation Environments. *Systems Analysis and Simulation II*, 344-350.
- [21.] Denis Kotarski, P. P. (2021). A Modular Multirotor Unmanned Aerial Vehicle Design Approach for Development of an Engineering Education Platform. *Sensors*.
- [22.] *Drone Inspections are Improving Efficiency in the Utilities Industry*. (202, September 23). Tratto da Drone-powered Solutions: <https://www.feds.ae>
- [23.] Emil Fresk, G. N. (2013). Full Quaternion Based Attitude Control for a Quadrotor. *2013 European Control Conference (ECC)*. Zürich, Switzerland.
- [24.] Fletcher, R. (1987). *Practical Methods of Optimization*. New York: John Wiley and Sons.
- [25.] Foundation, D. (2018). *Pixhawk® 4 Mini Technical Data Sheet*.
- [26.] Fum, W. Z. (2015). *Implementation of Simulink Controller Design on Iris+ Quadrotor*. Monterey, California: Naval Postgraduate School.
- [27.] G.T. Poyi, M. W. (2013). Validation of a quad-rotor helicopter Matlab/Simulink and Solidworks models. *IET Conference on Control and Automation 2013: Uniting Problems and Solutions*. Birmingham, UK: IET .
- [28.] Gatti, M. (2015). *Design and Prototyping High Endurance Multi-Rotor*. Bologna, Italy: Università di Bologna.
- [29.] II, R. G. (2020). *COTS DRONE DESIGN: A RAPID EQUIPAGE ALTERNATIVE FOR FORCE RECON COMPANIES*. Monterey, California: Naval Postgraduate School.
- [30.] Iswanto, A. M. (2019). Artificial potential field algorithm implementation for quadrotor path planning. *International Journal of Advanced Computer Science and Applications*, vol. 10, 575–585.
- [31.] John F. Keane, S. S. (2013). A Brief History of Early Unmanned Aircraft. *Johns Hopkins APL Technical Digest*, Volume 32, Number 3.
- [32.] José A. Paredes, F. J. (2017). 3D Indoor Positioning of UAVs with Spread Spectrum Ultrasound and Time-of-Flight Cameras. *Sensors*.
- [33.] Kayton, M. a. (1997). *Avionics Navigation Systems*. John Wiley and Sons.

- [34.] Kenneth R. Muske, J. B. (1993). Model Predictive Control with Linear Models. *AIChE Journal*, Vol. 39, No. 2.
- [35.] Kimon P. Valavanis, G. J. (2015). *Handbook of Unmanned Aerial Vehicles*. Dordrecht: Springer.
- [36.] Lei Deng, Z. M. (2018). UAV-based multispectral remote sensing for precision agriculture: A comparison between different cameras. *ISPRS Journal of Photogrammetry and Remote Sensing*.
- [37.] Liang, O. (2021, June 12). *How to Choose FPV Drone Motors*. Tratto da OscarLiang.com: <https://oscarliang.com>
- [38.] Luke S. Dai, M. A. (2016). *3D Printed Quadcopters*. New Jersey, USA: New Jersey Governor's School of Engineering and Technology.
- [39.] Marcin Biczyski, R. S. (2020). Multirotor Sizing Methodology with Flight Time Estimation. *Journal of Advanced Transportation*.
- [40.] Matija, K. (2020). *Modelling and control of hybrid propulsion systems for multirotor unmanned aerial vehicles*. Zagreb, Croatia: University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture.
- [41.] Meier, L. H. (2015). PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. *Robotics and Automation (ICRA), 2015 IEEE International Conference*.
- [42.] Moses, B. (February 2017). *Aerodynamics and Control of Quadrotors*. Canberra, Australia: A thesis submitted for the degree of Doctor of Philosophy.
- [43.] Nicolas Mandel, M. M. (2020). A Method for Evaluating and Selecting Suitable Hardware for Deployment of Embedded System on UAVs. *Sensors*.
- [44.] Nicolas Mandel, M. M. (2020). A Method for Evaluating and Selecting Suitable Hardware for Deployment of Embedded System on UAVs. *Sensors*.
- [45.] Petar Piljek, D. K. (2020). Method for Characterization of a Multirotor UAV Electric Propulsion System. *Applied Sciences*.
- [46.] Peter Burggräf, A. R. (2019). Quadrotors in factory applications: design and implementation of the quadrotor's P-PID cascade control system. *SN Applied Sciences*.
- [47.] Pounds, P. E. (2007). *Design, Construction and Control of a Large Quadrotor Micro Air Vehicle*. Australian National University.
- [48.] *Powerline inspection with UgCS*. (s.d.). Tratto da UgCS: <https://www.ugcs.com>
- [49.] Protocol, M. A. (s.d.). *MAVLink Developer Guide*. Tratto da MAVLINK Micro Air Vehicle Communication Protocol: <https://mavlink.io/en/>
- [50.] Quan, Q. (2017). *Introduction to Multicopter Design and Control*. Singapore: Springer.

- [51.] R A Navrotsky, G. V. (2021). Exploration of strength characteristics quadrocopter frame structure obtained using 3d printing technology. *IOP Conf. Series: Materials Science and Engineering*. Moscow, Russia: IOP Publishing.
- [52.] R.E. Weibel, R. H. (2004). Safety considerations for operation of different classes of UAVs in the NAS. *Proceedings of the AIAA 4th Aviation Technology, Integration and Operations Forum and AIAA 3rd Unmanned Unlimited Technical Conference, Workshop and Exhibit*. Chicago.
- [53.] Rahul, B. (June 9, 2019). *Quaternion and Euler Angles*. Arizona, USA.
- [54.] Raissa Zurli Bittencourt Bravo, A. L. (2015). Literature review of the applications of UAVs in humanitarian relief. *XXXV Encontro Nacional De Engenharia De Producao*. Fortaleza, CE, Brasil.
- [55.] Reid, J. (2017, January 27). *Understanding KV Ratings*. Tratto da Rotor Drone PRO: <https://www.rotordronepro.com>
- [56.] Research, M. (2021). *AirSim*. Tratto da Microsoft GitHub AirSim: <https://microsoft.github.io/AirSim/>
- [57.] Reza Ehsani, J. M. (2013). *The Rise of Small UAVs in Precision Agriculture*. Lake Alfred, USA: American Society of Agricultural and Biological Engineer.
- [58.] Rico Merkert, J. B. (2020). Managing the drone revolution: A systematic literature review into the current use of airborne drones and future strategic directions for their effective control. *Journal of Air Transport Management*.
- [59.] Robert Niemiec, F. G. (September 5-8, 2016). A Comparison Between Quadrotor Flight Configurations. *42nd European Rotorcraft Forum*. Lille, France.
- [60.] Robotics, M. (2020). *Marvelmind Starter Set HW v4.9-NIA*. Tratto da Marvelmind Robotics: <https://marvelmind.com/product>
- [61.] Robotics, M. (s.d.). *How to Build Autonomous Drones Indoor*. Tratto da Marvelmind Robotics: <https://marvelmind.com/drones>
- [62.] Robotics, M. (s.d.). *Indoor "GPS" Autonomous Copter Setting Manual*. Tratto da Marvelmind Robotics: https://marvelmind.com/pics/indoor_navigation_system_ENG_copter_help_manual.pdf
- [63.] Ruijie He, P. S. (2008). Planning in information space for a quad rotor helicopter in a GPS-denied environment. *2008 IEEE International Conference on Robotics and Automation* (p. 1814-1820). Pasadena, CA, USA: IEEE.
- [64.] Sabatino, F. (2015). *Quadrotor Control: Modeling, Nonlinear Control Design and Simulation*. Stockholm, Sweden: KTH Electrical Engineering.
- [65.] Sheng-Lung Peng, L. H. (2019). *Intelligent Computing and Innovation on Data Science*. Warsaw, Poland: Springer.

- [66.] Siemens launches SIEAERO – the next generation of overhead line inspection. (2018, November 14). Tratto da Geospatial World. Advancing Knowledge for Sustainability: <https://www.geospatialworld.net>
- [67.] Slawomir Grzonka, G. G. (2012). A Fully Autonomous Indoor Quadrotor. *IEEE Transactions on Robotics*, 90-100.
- [68.] Sravan Kumar N., R. K. (2016). Design and Control Implementation of Quadcopter. *International Journal of Mechanical And Production Engineering*, 2320-2092.
- [69.] Szyk, B. (2021, August 15). *Drone Motor Calculator*. Tratto da Omni Calculator: <https://www.omnicalculator.com>
- [70.] T. T. Mac, C. C. (2016). Heuristic Approaches in Robot Path Planning: A Survey. *Robotics and Autonomous Systems*, 13-28.
- [71.] Tamino Wetz, N. W. (2021). Distributed wind measurements with multiple quadrotor unmanned aerial vehicles in the atmospheric boundary layer. *Atmospheric Measurement Techniques*, 3795-3814.
- [72.] The MathWorks, I. (2021). *Simulink 3D Animation*. Tratto da MathWorks: <https://it.mathworks.com>
- [73.] U. Orozco-Rosas, O. M. (2019). Mobile Robot Path Planning Using Membrane Evolutionary Artificial Potential Field. *Applied Soft Computing Journal*, 236–251.
- [74.] V. Kumar, N. M. (2012). Opportunities and challenges with autonomous. *The International Journal of Robotics Research*, 1279-1291.
- [75.] Wachsler, J. (2018, June 20). *airsim-examples*. Tratto da GitHub: <https://github.com/JoelWachsler/airsim-examples>
- [76.] Wang, L. (2009). *Model Predictive Control System Design and Implementation Using MATLAB*. Melbourne, Australia: Springer.
- [77.] Wei Dong, G.-Y. G. (2013). Modeling and Control of a Quadrotor UAV with Aerodynamic Concepts. *International Journal of Aerospace and Mechanical Engineering*, Vol:7, No:5.
- [78.] Wei Dong, G.-Y. G. (2013). Modeling and Control of a Quadrotor UAV with Aerodynamic Concepts. *International Journal of Aerospace and Mechanical Engineering*, Vol:7, No:5.
- [79.] Wright, S. J. (1997). Applying New Optimization Algorithms to Model Predictive Control. *Chemical Process Control-V, CACHE, AIChE* , 147-155.
- [80.] Y. B. Chen, G. C. (2016). UAV path planning using artificial potential field method updated by optimal control theory. *International Journal of Systems Science*, vol. 47, 1407-1420.
- [81.] Yasir Mohd Mustafah, A. W. (2012). Indoor UAV Positioning Using Stereo Vision Sensor. *International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012)* (p. 575-579). Kuala Lumpur, Malaysia: Elsevier.

- [82.] Yohanes Khosiawan, I. N. (2016). A system of UAV application in indoor environment. *Production & Manufacturing Research*, 2-22.
- [83.] Yuntian Li, M. S. (2018). A Novel Distributed Architecture for UAV Indoor Navigation. *International Conference on Air Transport - INAIR 2018* (p. 13-22). ELSEVIER.
- [84.] Zoran Benić, P. P. (2016). *MATHEMATICAL MODELLING OF UNMANNED AERIAL VEHICLES WITH FOUR ROTORS*. Zagreb, Croatia: Interdisciplinary Description of Complex Systems.