POLITECNICO DI TORINO

Master's Degree in MECHATRONIC ENGINEERING

Master's Degree Thesis

Design and Implementation of Enhanced Model Reference Adaptive Controllers (EMRAC) for Quadcopters Trajectory Tracking via HIL on Pixhawk 6x

Supervisors

Prof. UMBERTO MONTANARO

Prof. ALESSANDRO RIZZO

Prof. ALDO SORNIOTTI

SIMONE MARTINI

Candidate

SALVATORE MATTEO MENNEA

April 2024

Summary

The main objective of this Master's thesis is to develop a comprehensive platform for testing and deploying custom controllers for quadrotor drones for trajectory tracking and deploy and test different version of EMRAC controllers on an actual flight control board.The toolchain is capable of evaluating advanced control algorithms on real hardware using Software-in-the-Loop (SITL) and Hardware-in-the-Loop (HIL) simulations. The thesis utilizes the PX4-Autopilot firmware, an open-source software designed for autonomous aerial vehicles. The implemented advanced control algorithm, EMRAC (Enhanced Model Reference Adaptive Controller), builds upon the MRAC (Model Reference Adaptive Controller) to enhance performance. To benchmark EMRAC's effectiveness, PD and MRAC controllers were designed. These control algorithms were developed using MATLAB/Simulink, facilitated by the PX4 toolbox (UAV Toolbox Support Package for PX4 Autopilots). This toolchain allows testing controllers in a simulation environment (e.g. jMAVSim) while also generating deployable code. The generated code can be executed on a generic computer (SITL) or deployed on a real Pixhawk 6x flight controller (HIL). The algorithms were also tested by increasing the mass and inertia of the quadrotor to demonstrate the EMRAC's potential to operate effectively under non-nominal conditions.

Table of Contents

List of Tables

List of Figures

Acronyms

BRF	Body reference frame
IRF	Inertial reference frame
MIL	Model-in-the-loop
SITL	Software-in-the-loop
HIL	Hardware-in-the-loop
KPI	Key Performance Indicator
QGC	Qgroundcontrol
MRAC	Model Reference Adaptive Controller
EMRAC	Enhanced Model Reference Adaptive Controller

Parameter	**Symbol**	**Value**	**Unit**
Drone mass	m	0.8	kg
Gravity acceleration	g	9.81	$m \cdot s^{-2}$
Distance from propeller to drone center	l	0.165	m
Full thrust of one propeller	T_{max}	4	N
Anti-torque at full thrust of one propeller	Q_{max}	0.05	$N \cdot m$
Moment of inertia of rotation around the z-axis	J_z	0.005	$kg \cdot m^2$
Moment of inertia of rotation around the x-axis	J_x	0.005	$kg \cdot m^2$
Moment of inertia of rotation around the y-axis	J_y	0.009	$kg \cdot m^2$
Air drag force coefficient	K_1	0.01	$N \cdot s^2 \cdot m^{-1}$
Air drag torque coefficient	K_2	0	$N \cdot m \cdot s^2 \cdot rad^{-2}$
Position vector	ξ		m
Attitude vector	η		rad
Angular velocity vector in BRF	ν		rad/s
Inertial reference frame	$x - y - z$		
Body reference frame	$x' - y' - z'$		

Chapter 1

Introduction

1.1 Drones and Quadrotors

In recent years, the remarkable advancements in robotics and unmanned aerial vehicle (UAV) technology have given rise to a diverse range of applications, with quadrotor drones taking centre stage. These agile and versatile aerial machines have captured the imagination of researchers, hobbyists, and industries alike due to their ability to perform complex manoeuvres and access areas that are typically inaccessible or hazardous to humans. Quadrotor drones, also known as quadcopters, belong to the family of multirotor UAVs and are characterized by their four rotors arranged in a square or X configuration.
Quadrotor drones' popularity stems from their exceptional features like hovering, VTOL, and flight stability. This versatility makes them perfect for diverse applications: aerial photography, surveillance, search and rescue, agriculture monitoring, package delivery, and beyond. As technology advances, quadrotor drones continue to open new possibilities, pushing the boundaries of what they can achieve.

1.2 Project's Objective and Thesis Structure

The objective of this thesis is to create a platform able to simulate and deploy on real hardware, some control algorithms focusing on adaptive controllers for trajectory tracking problems.The target controller that has been implemented is the EMRAC algorithm that has been compared with the benchmark controllers as PD and MRAC and also with different architectures of the EMRAC controller. The main software used are MATLAB, Simulink and Jmavsim. The second chapter is an overview about PX4 autopilot, the firmware used to perform SITL and HIL on Jmavsim through UAV Toolbox Support Package for PX4 Autopilots.The third chapter is about the configuration of the toolchain environment, explain how to set the MIL, SITL and HIL and the logic behind those different kind of simulations.Both the software and hardware settings were explained. The fourth chapter of this thesis introduces the mathematical model of the quadrotor essential to understand how to control it. On the fifth chapter, different controllers are introduced:PD, MRAC.Those controllers have been used as benchmark for the main objective of this thesis: the design of the enhanced model reference adaptive controller(EMRAC).

Chapter 2

PX4 overview

2.1 Autopilot hardware and software

2.1.1 Pixhawk autopilot board

Pixhawk, a highly advanced autopilot board, is widely recognized for its performance and versatility in the field of unmanned aerial systems (UAS) and robotics. Its origins can be traced back to a master thesis project at ETH Zurich in 2008. Over time, it has transformed into a popular open-source solution, often combined with the PX4 flight stack. Serving as a functional and cost-effective alternative for implementing flight control algorithms professionally, Pixhawk is renowned for its user-friendly nature. It allows individuals to engage in advanced tasks without requiring an extensive understanding of autopilot design. Instead, a basic knowledge of control theory and high-level programming languages such as C++, Java, or Python is sufficient for implementing automatic control solutions with Pixhawk.

The Pixhawk project also provides open hardware designs that serve as blueprints for assembling various components like the CPU and sensors. These designs offer guidance on how to connect and map the pins, enabling people to easily build and customize their own hardware.

It is important to note that there are multiple versions of Pixhawk. The Pixhawk project utilizes a naming system called FMUvX (Flight Management Unit Version X) to designate each version. Higher FMU numbers indicate newer versions, although they do not necessarily imply increased capabilities. The most recent version, FMUv6C, is utilized in Pixhawk 6c.

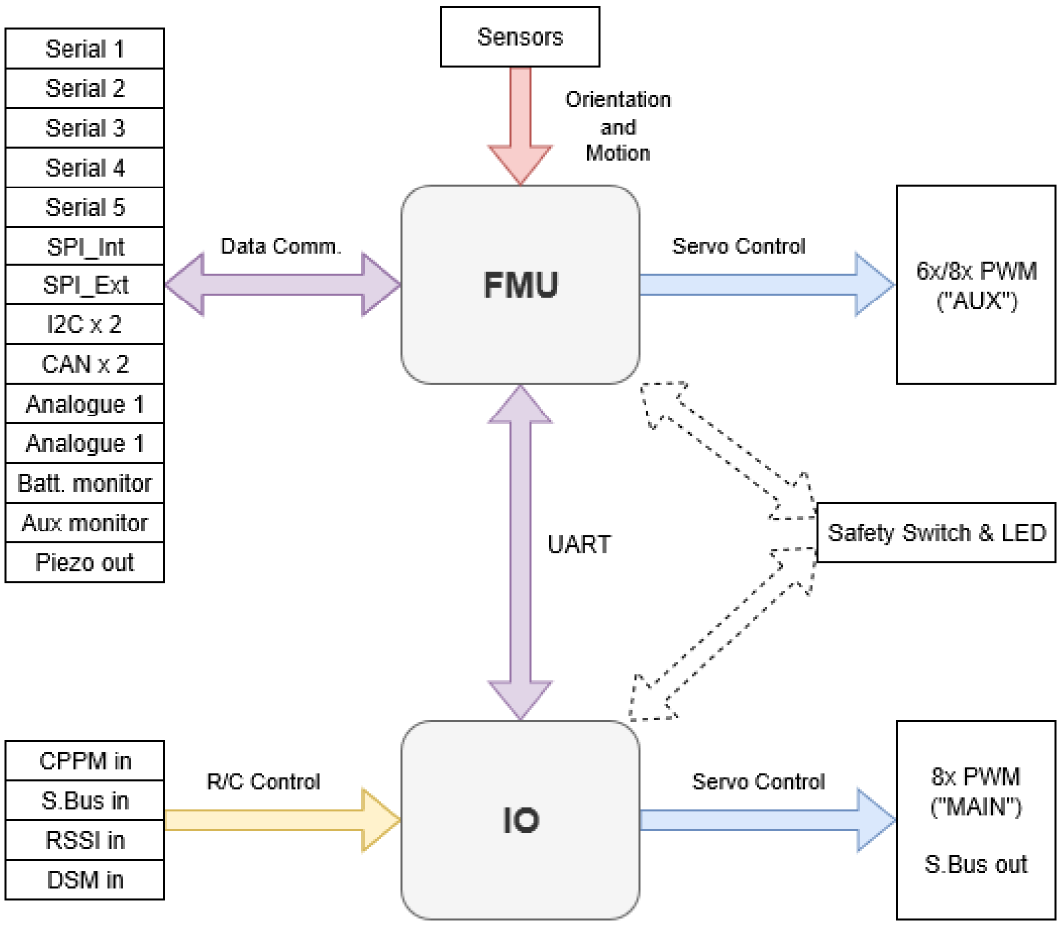


Figure 2.1: Pixhawk FMU and I/O architecture [1]

2.1.2 PX4 Architecture

PX4 is diveded into two main layers [2]:

- the flight stack
- the middleware

In the figure 2.2 an overview of the essential blocks of PX4 is shown. The top part of the figure shown the middleware blocks and the lower section shows the flight stack components.

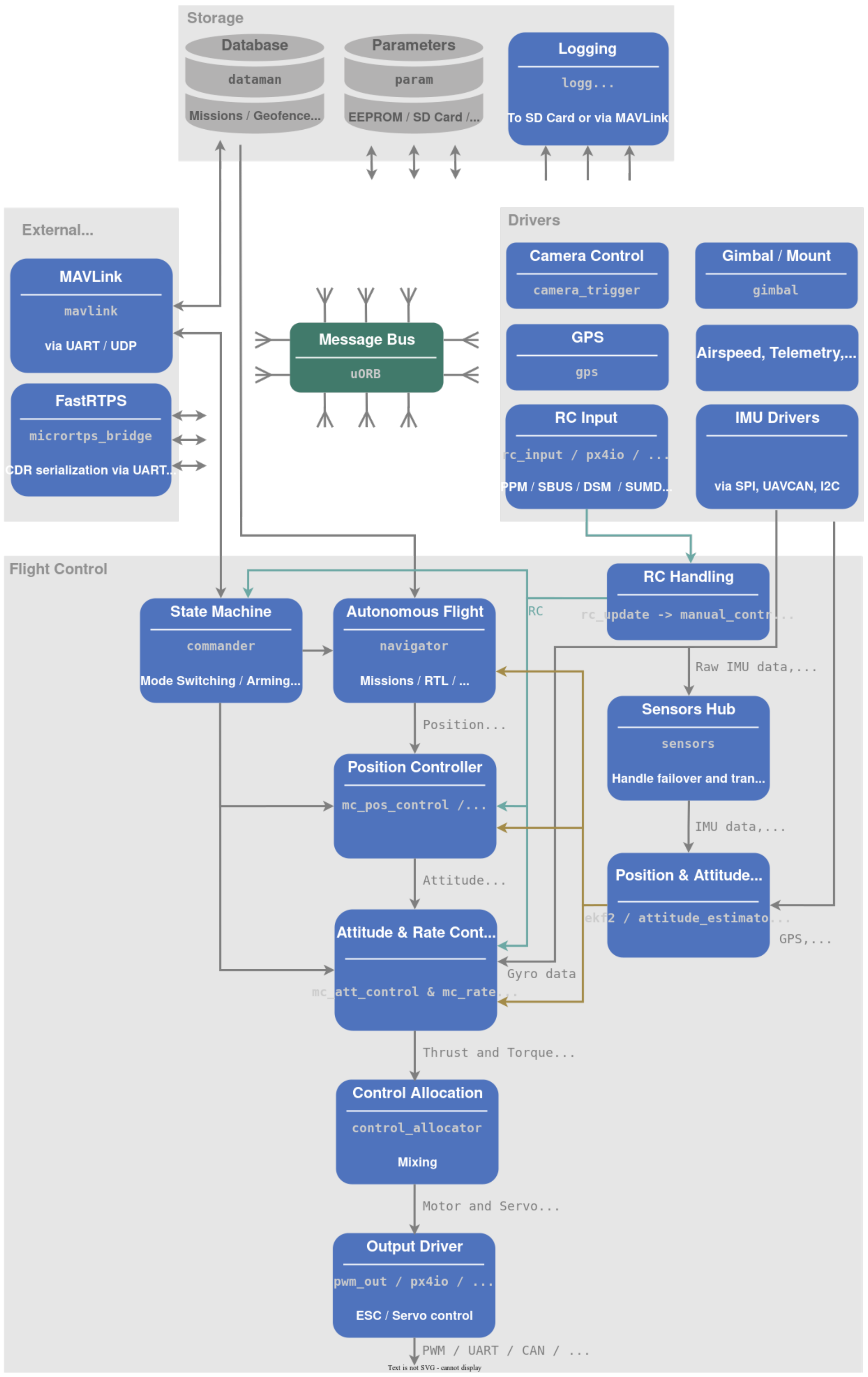


Figure 2.2: PX4 architecture [2]

The structure is organized in modules.Every block of the PX4 structure correspond to one or more modules.As we can see, there's the message bus uORB in the middle of the architecture that are responsible to publish and subscribe data, letting modules communicate between each other.

uORB middleware

uORB (micro Object Request Broker) is a publish-subscribe messaging middleware that plays a crucial role within the PX4 autopilot ecosystem. Developed specifically for real-time embedded systems, uORB facilitates efficient and reliable communication between various modules and components within the PX4 flight stack.The uORB is an asynchronous publish() / subscribe() messaging API used for inter-thread/inter-process communication. A component sends a message by publishing it to a particular topic, such as vehicle_local_position. Other components receive the message by subscribing to that topic.

This decoupled communication mechanism ensures a modular and scalable architecture, allowing PX4 to easily handle a wide array of sensors, actuators, and high-level control algorithms.Its efficient publish-subscribe model allows data to flow between different modules without direct dependencies, promoting code modularity and maintainability. Each module can publish data on specific topics, and other modules can subscribe to those topics, receiving updates as soon as new information becomes available.

Flight stack

The flight stack is a set of guidance, navigation and control algorithms for drones. In the figure 2.3 the flight stack architecture is shown.

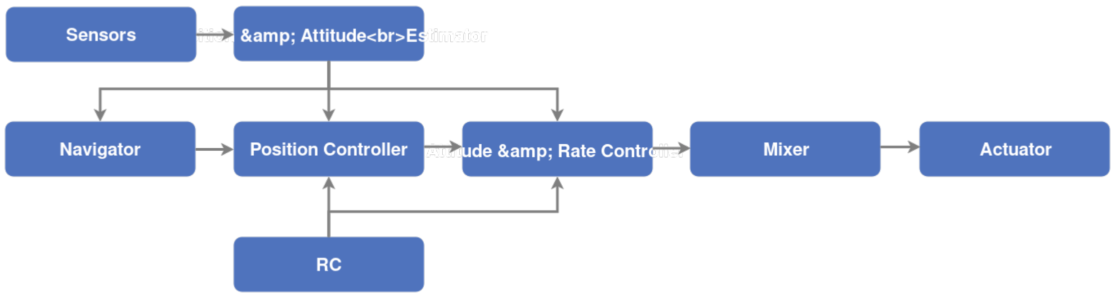


Figure 2.3: The flight stack [2]

As it's shown in the figure above, the flight stack is composed by several blocks that are:

- **Commander**:contains information on the status of the drone and information on the modes of flight, including failsafe;
- **Navigator**: It contains flight set points, such as launch points, takeoff and waypoints, which are the main points for constructing the flight path;
- **Mc_controller**: It has two principal modules (position and attitude_and_rate controller)
- **Mixer**: its task is to map the controllers signals generated by mc_controller into virtual control signals sent to the drone actuators;
- **Sensor hub**:it is the core of the signals measured by the sensors: the low-level output data are made available through the drivers and are made available to the controller;

- **EKF2(Extended Kalman Filter)**: it estimates the position and attitude;
- **Drivers**: They are the interface among the physical sensor and the firmware architecture;
- **Mavlink**: it implements the "MAVLINK" which is a communication protocol which sends specific signals through uORB;

NuttX operating system

NuttX is a real-time operating system (RTOS) that serves as a key component in the PX4 autopilot ecosystem. NuttX is designed for embedded systems, making it an ideal choice for PX4's flight control applications. It is open source(BSD licence), highly portable, efficient and stable. It is the base framework on which all the PX4 architecture works.

2.2 Development environment

2.2.1 Software

The operating system used was Windows 10.The software used were:

- **QgroundControl**: this software is the interface to the unmanned vehicle configuration with PX4, and it's the station for the planning and execution of the flight path that the drone has to follow.

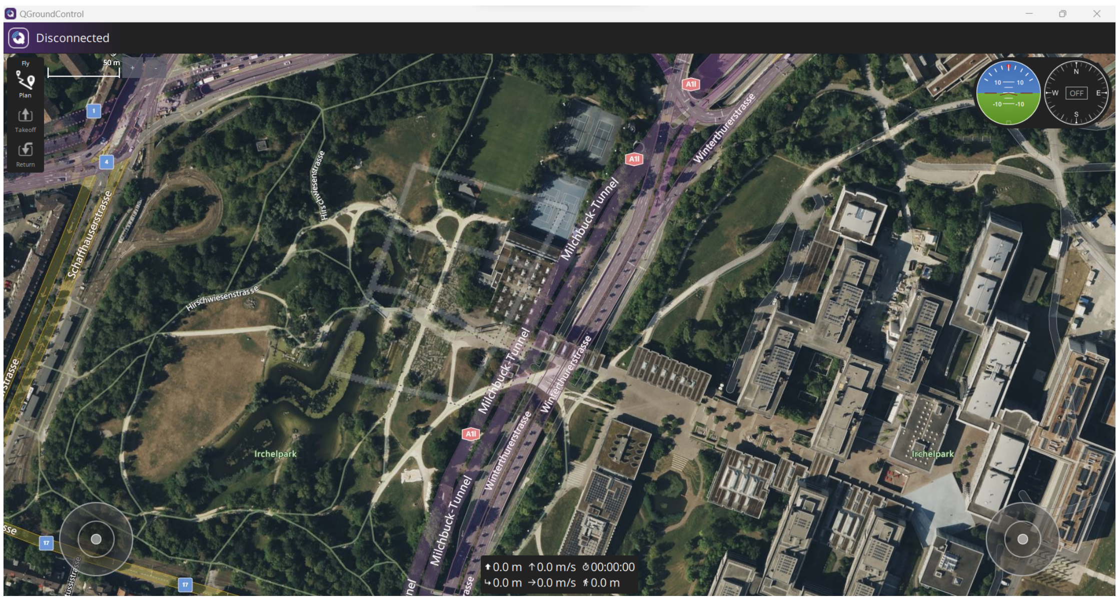
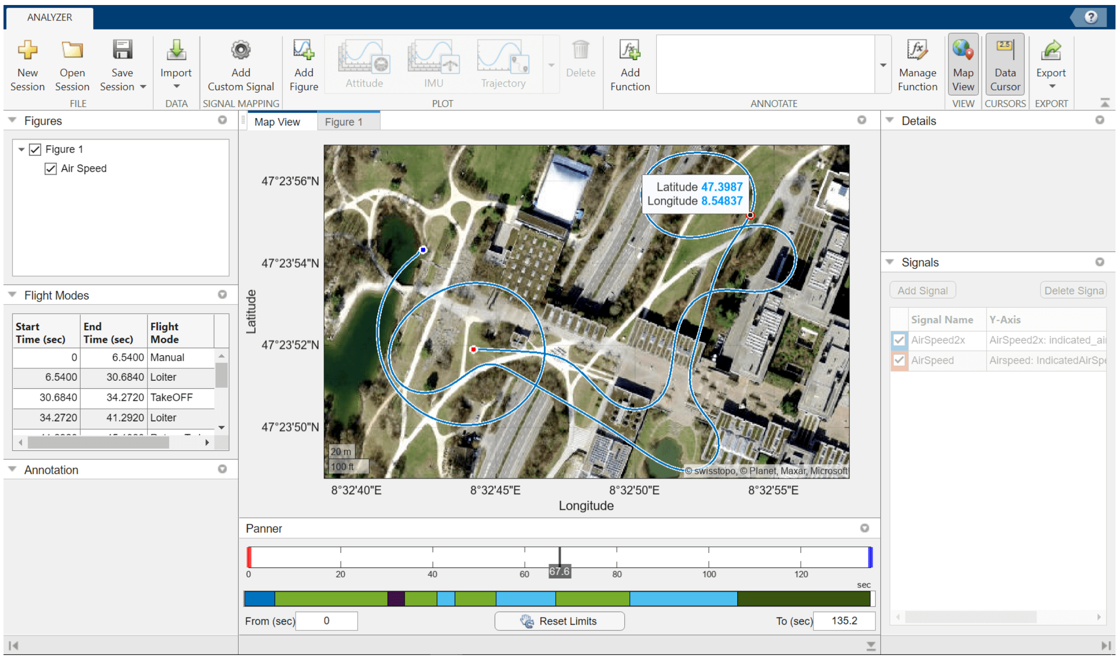


Figure 2.4: QGC home

From the homepage, it's possible to access to four section:

- **Fly Plan**:it allows planning the mission of the UAV. Some different points have to be defined. The launch point, that is the point where the UAV goes up before starting the flight, takeoff point, that is the point where the drone has to start the flight mission and the waypoints, that are the points that define the trajectory during the flight mission;

- **Vehicle Setup**:In this section is possible to upload the firmware, to set some parameters for the mission and to define the airframe;
- **Analyse Tools**:it allows downloading the mission file .ulog after the mission. These files in order to be read needs "Flight Log analyser" a specific app available in MATLAB.



Flight Modes

Start Time (sec)	End Time (sec)	Flight Mode
0	6.5400	Manual
6.5400	30.6840	Loiter
30.6840	34.2720	TakeOFF
34.2720	41.2920	Loiter

Signal Name	Y-Axis
AirSpeed2x	AirSpeed2x: indicated_ai
AirSpeed	Airspeed: IndicatedAirSp

From (sec) 0   To (sec) 135.2

Figure 2.5: Flight Log Analyzer

- **Application settings**: this sections is used to set communication setting with the flight control board, to set which measurement unit to use for the flight data. It is also possible to set the flight file in CSV format.

mario.mihalkov@surrey.ac.uk

- **PX4.Windows.Cygwin.Toolchain.0.8.msi** This is the only software officially supported for the PX4 firmware download on Windows
- **MATLAB(R2023b)**: this is the main software used, provided my MathWorks.All the control algorithm and data analysis has been built with this software, alongside the following toolbox:
  - **Simulink**: This is the tool for the modelling and dynamic simulations.Here, the control laws were built and then were deployed.
  - **UAV Toolbox Support Package for PX4 Autopilots**
    UAV Toolbox Support Package for PX4 Autopilots enables you to create Simulink models that work with the uORB middleware.[3]. This toolbox allows you to create controllers, estimators, and navigators using Simulink. Once you design these in

Simulink, you can deploy them directly to PX4 Autopilot boards. By integrating the Simulink-generated code with the PX4 flight stack, you can deploy your designs to the PX4 Autopilots easily. This package has several blocks that provide interfaces with the PX4 flight stack as shown in figure 2.6 [4].

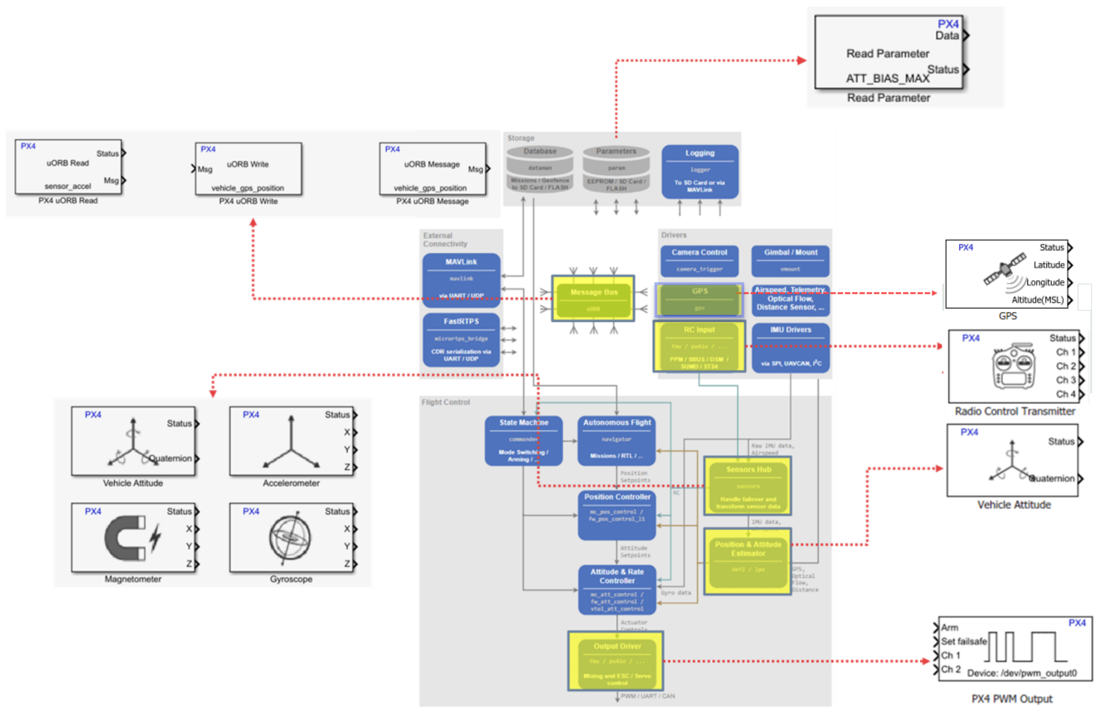


Figure 2.6: Simulink Package PX4 blocks

Thanks to this package, it's possible to design a controller from scratch using the blocks that are able to write and read data through the uORB messages.

- **jMAVSim**:This is the simulator provided my the toolbox. It's only available for quad rotors. It provides a 3D visualization of the UAV in a simulating environment.

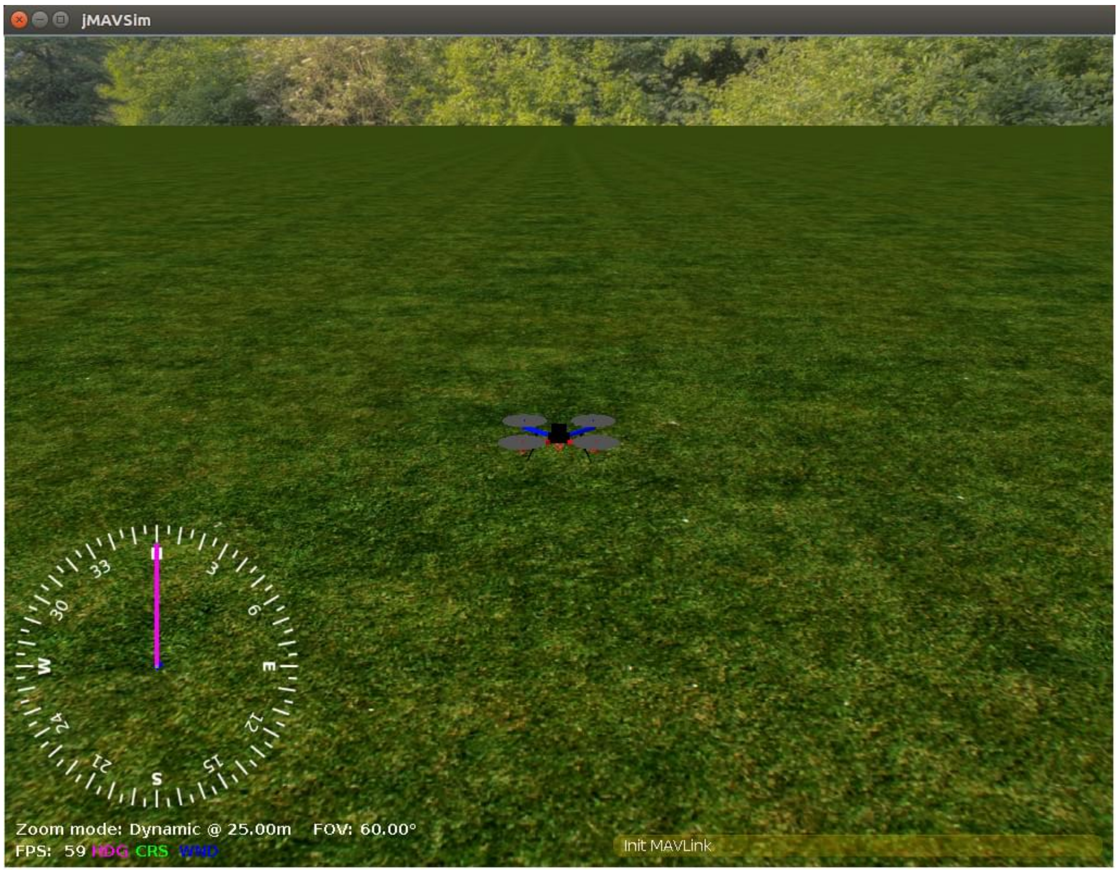


Figure 2.7: jMAVsim

2.2.2 Hardware

For what concerning the hardware side, the following item was used:

- **Pixhawk 6x**: this is the controller on which the control laws have been implemented in HIL mode.In figure 2.8 its design can be seen. All the hardware details can be seen in the table :

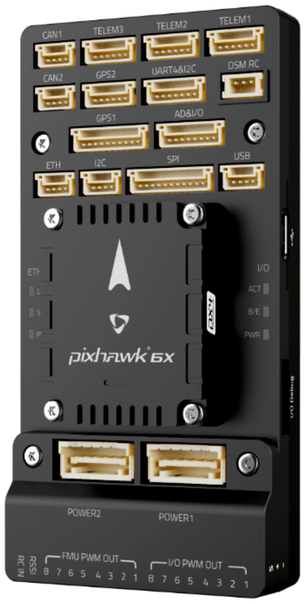


Figure 2.8: Pixhawk 6x

Processors	FMU Processor: STM32H753 32-Bit Arm Cortex-M7,480MHz
	IO Processor: STM32F103 32-Bit Arm Cortex-M3,72Mhz
RAM	2 MB flash memory,1MB RAM
	64KB SRAM
Sensors	Accel/Gyro: ICM-20649
	Accel/Gyro: ICM-42688-P
	Accel/Gyro: ICM-42670-P
	Mag: BMM150
	Barometer:2xBMP388
Interfaces	16-PWM servo inputs
	R/C input for Spektrum / DSM
	Dedicated analog/PWM RSSI input and S.Bus output
	Dedicated R/C input for PPM ans S.Bus input
	4 general purpose serial ports
	2 GPS ports
	1 I2C port
	1 Ethernet port
	1 SPI bus
	2 CAN Buses for CAN peripheral
	2 Power inputs ports with SMBus

Table 2.1: Pixhawk 6x Hardware details

- **Serial convertor CP2102/9 (from USB to TTL)**: It's needed to configure the interface between Pixhawk and the host PC.Through this converted is it possible to deploy the control algorithm and to analyse the executing Simulink model in the target hardware.

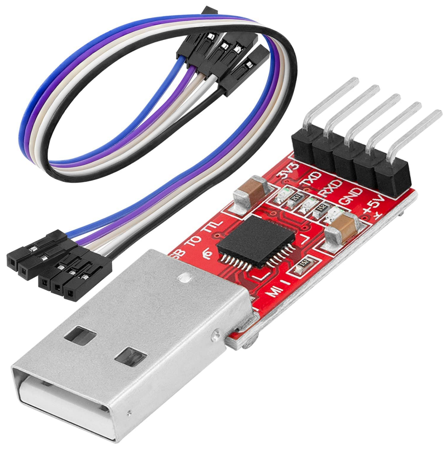


Figure 2.9: Serial converter CP2102/9

The main characteristics are:

- The analog input and output works at 3.3V. It can also be supplied at 5V or at 3.3V.
- It needs a driver that can be found to the official documentation in order to be recognized with the host PC.After the installation, the converter can be seen in the "Device Management" section in the host PC.

- **JST connector**:It is a 6 pin connector with two female terminals. It allows connecting the Pixhawk to the host PC.One terminal goes to the TELEM1 port on the flight control board, and another terminal goes to the CP2102/9 converter. It has 6 wires, 4 are needed to connect with the CP2102/9 converter: the red one goes to the Vcc(power supply), the blue one goes to ground(GND), the yellow ones go to RXD (receiving) and TXD(transmitting).

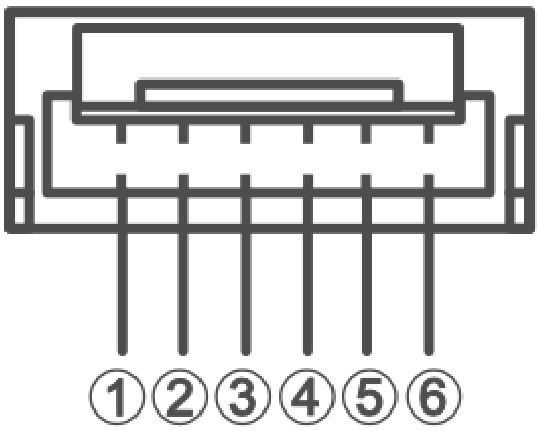


Figure 2.10: JST GH 1.25 mm connectors diagram [5]

Pin	Signal	Volt
1(red)	VCC	+5V
2(black)	TX7/5/2 (out)	+3.3V
3(black)	RX7/5/2 (in)	+3.3V
4(black)	CTS7/5/2(in)	+3.3V
5(black)	RTS7/5/2 (out)	+3.3V
6(black)	GND	GND

Table 2.2: Telem1, Telem2, Telem3 ports [5]

UART	Device	Port
USART1	/dev/ttyS0	GPS
USART2	/dev/ttyS1	TELEM3
USART3	/dev/ttyS2	Debug Console
UART4	/dev/ttyS3	UART4 & I2C
UART5	/dev/ttyS4	TELEM2
USART6	/dev/ttyS5	PX4IO/RC
UART7	/dev/ttyS6	TELEM1
UART/	/dev/ttyS7	GPS2

Table 2.3: Serial Port Mapping [5]

Chapter 3

MIL,SITL and HIL

In order to design the new controllers implemented in the PX4 Autopilot firmware, a V-shaped approach has been followed.The MIL,SITL and HIL were performed for each control law.In the figure 3.1 the PX4 toolchain is shown [6]:

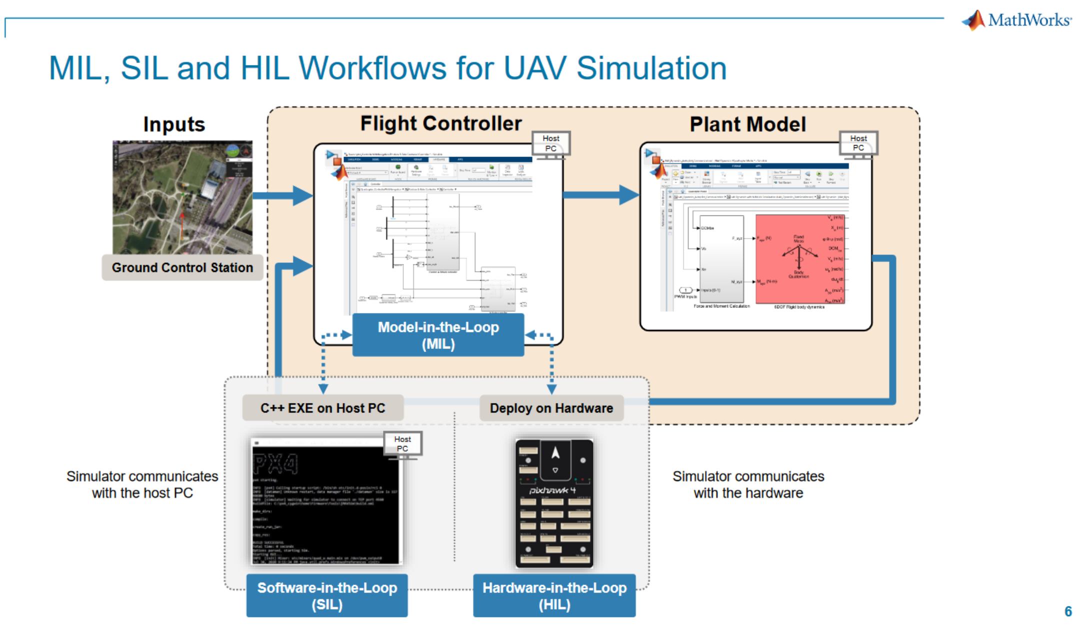

MathWorks
MIL, SIL and HIL Workflows for UAV Simulation
Inputs
Ground Control Station
Flight Controller
Host PC
Plant Model
Host PC
Model-in-the-Loop (MIL)
C++ EXE on Host PC
Deploy on Hardware
Simulator communicates with the host PC
Simulator communicates with the hardware
Software-in-the-Loop (SIL)
Hardware-in-the-Loop (HIL)
6

Figure 3.1: SITL and HIL workflows [6]

3.1 Setup and configuration

In order to use this toolbox, you need to install the PX4.Windows.Cygwin.Toolchain.0.8.msi available on GitHub.Python38 is also needed to compile everything. All the links are available in the setup setting on the UAV Toolbox support for PX4 autopilots is downloaded.
The following steps have to be followed [7] :

- **Installing Python 3.8.2 on Windows for Firmware Upload**:when setting up the toolbox,you are redirected to link where you can download Python 3.8.2.
- **Setting Up Cygwin Toolchain and Downloading PX4 Source Code**: To set up Cygwin toolchain and download the PX4 source code that is used in UAV Toolbox Support Package for PX4 Autopilots, follow these steps:
  - Download version 0.8 of PX4 Cygwin Toolchain MSI Installer, which is compatible with PX4 Firmware v1.12.3;
  - Run the MSI installer and start the installation of the toolchain.
  - Change the installation folder for Cygwin to any local folder, and then click OK.
  - If you do not have the PX4 Source Code (PX4 Autopilot Firmware v1.12.3) downloaded in the host computer, select the option Clone PX4 repository and Start Simulation, and then click Finish. This option clones the current PX4 main Firmware. When you click Verify Installation in Step 6 below, the firmware is checked out automatically to v1.12.3.
- **Selecting PX4 Autopilot Application**[8]:In the Select Application in Simulink screen of the Hardware Setup process, select the required option for the PX4 Autopilot algorithm that you are going to design using the support package:
  - **Design Flight Controller Algorithm** in Simulink — Select this option to design a flight controller algorithm in Simulink. This selection results in an additional step (Select System Startup Script in PX4) as you proceed further with the Hardware Setup screens. In that step, you will get the option to select the preference for the startup script for the Autopilot. If you select the PX4 default startup script rCS as the startup script for the Autopilot, to avoid any interference with the default PX4 multi-copter controller modules. Then mc_pos_control and mc_att_control modules are disabled in the rCS startup script [8];
  - **Design Path Follower Algorithm** in Simulink — Select this option to design a path follower algorithm in Simulink. This option can be used in general to design any algorithms in Simulink apart from controllers or in scenarios where user would like to keep the default PX4 controllers enabled and available during Autopilot boot-up. This selection results in an additional step (Select Airframe for QGround Control) as you proceed further with the Hardware Setup screens. The default PX4 startup script, rCS, will be used for starting the modules. All the default modules existing in the rCS including the multi-copter controller modules are kept enabled and no modules are disabled with this selection.[8]

    For our use, the Design Flight Controller Algorithm is chosen.
- **Selecting Startup Script for PX4 Autopilot**:In the Select System Startup Script in PX4 screen of the Hardware Setup process, select the desired option for the PX4 Autopilot startup script.
  - **Use default startup script (rcS)** — Select this option to use the PX4 default startup script rCS as the startup script for the Autopilot. If you select Design Flight Controller Algorithm in Simulink as the algorithm choice in the Select Application in Simulink Hardware setup screen, the default PX4 multi-copter controller modules, mc_pos_control and mc_att_control modules are disabled in the rCS startup script.

If you select Design Path Follower Algorithm in Simulink, no modules are disabled in the rCS. This selection results in two additional steps, Download QGroundControl and Airframe in QGroundControl as you proceed further with the Hardware Setup screens.

- **Use custom startup script (rc.txt)**:Select this option to use a custom startup script called rc.txt which will be placed in the SD card on the Autopilot as the startup script.

According to what you want to perform(SITL or HIL), the target hardware has to be change, as it will be explained later on in this chapter.

3.2 Model-in-the-loop

In MIL, no particular setting is requested.Ones installed the UAV Toolbox Support Package for PX4 Autopilots, some examples will be available on your PC. In MIL both the controller and the plant are simulated inside Simulink and there's no interface with the PX4 firmware, but all the processes that takes place in the PX4 firmware are simulated in the Simulink environment.In order to visualize this example in Simulink, type:
$openExample('px4/MonitorTunePX4HostTargetFlightPlantModelExample')$
in the prompt command windows.

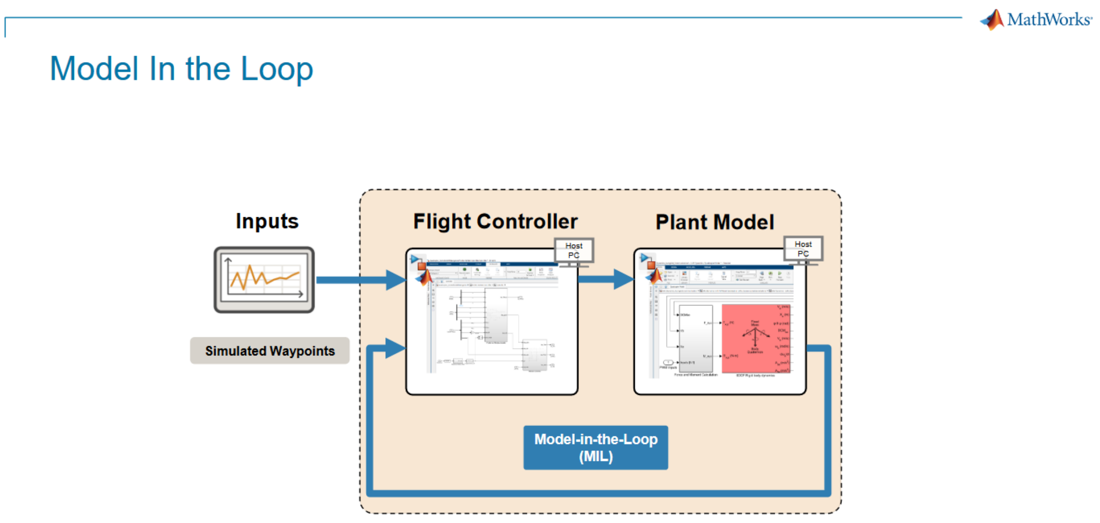
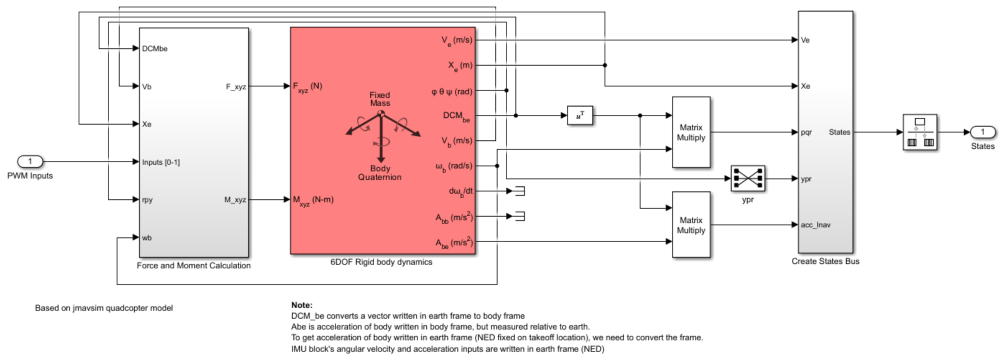


Figure 3.2: MIL workflow [6]

In the figure 3.2 the MIL workflow is shown.Three main blocks are shown:

- **Simulated Waypoints**:in this block, the trajectory is defined.It can be defined both through QGC and through a MATLAB function;
- **Flight Controller**: this is the block where the customized controllers have to be inserted;

- **Plant Model**: this is the plant shown in figure 3.3 provided by Simulink that is based on the jMAVsim simulator provided with this toolbox;



Note:
DCM_be converts a vector written in earth frame to body frame
Abe is acceleration of body written in body frame, but measured relative to earth.
To get acceleration of body written in earth frame (NED fixed on takeoff location), we need to convert the frame.
IMU block's angular velocity and acceleration inputs are written in earth frame (NED)

Figure 3.3: UAV Dynamics

In MIL there's no code generation, both the plant and the flight controller are simulated in the Simulink environment on the host PC.

3.3 Software-in-the-loop

In SITL the controller is code generated and the controller output is fed to the plant model.Two simulators are available: Simulink and jMAVsim.

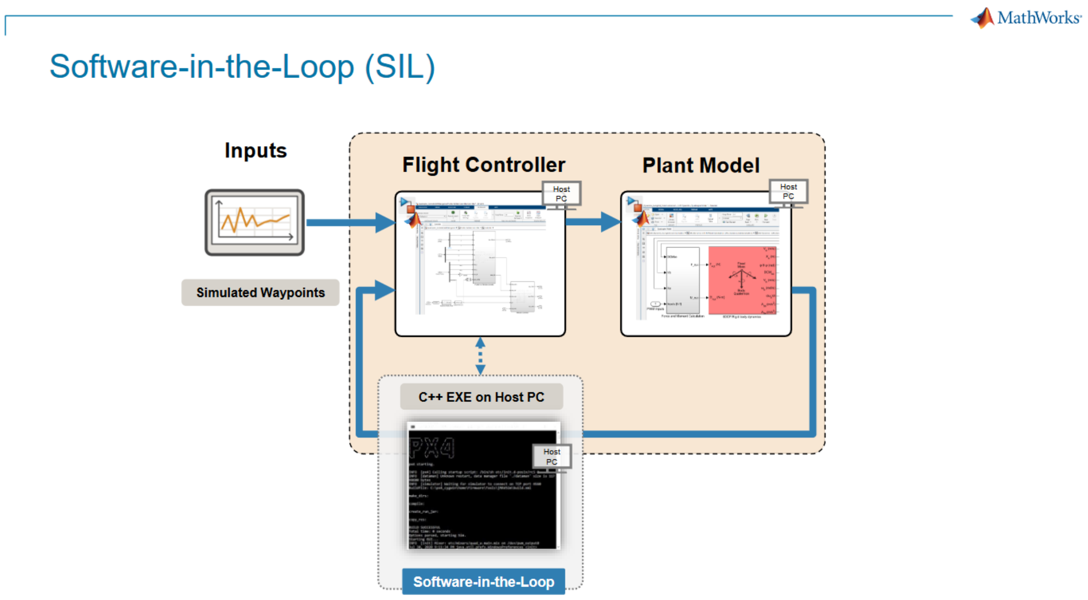


Figure 3.4: SITL workflow [6]

Both SITL and HIL can be run in two ways:

- **Monitor and Tune**:Monitor and Tune enables you to tune model parameters and evaluate the effects of different parameter values on model results in real-time. When you change parameter values in a model, the modified parameter values are communicated to the target hardware immediately. You can monitor the effects of different parameter values by viewing the output signals on Sink blocks or in Simulation Data Inspector (SDI). Doing so helps you find the optimal values for performance. This process is called parameter tuning.Monitor and Tune accelerates parameter tuning. You do not have to rerun the model each time you change parameters.[9]
- **Build, Deploy & Start**:this option enables you to build and deploy the code directly on the target, but you can't tune the models parameters online you need to restart every time the simulation. This is the best option because it runs at its own speed, rather than Monitor and Tune that needs to adjust the execution time to MATLAB.

3.4 Hardware-in-the-loop

When the HIL mode is running, both QGC and the UAV model has to communicate with the flight controller board.Nevertheless, just one of those application can run on the serial port.In order to fix this problem, a MAV link bridge is used as is shown in figure 3.5

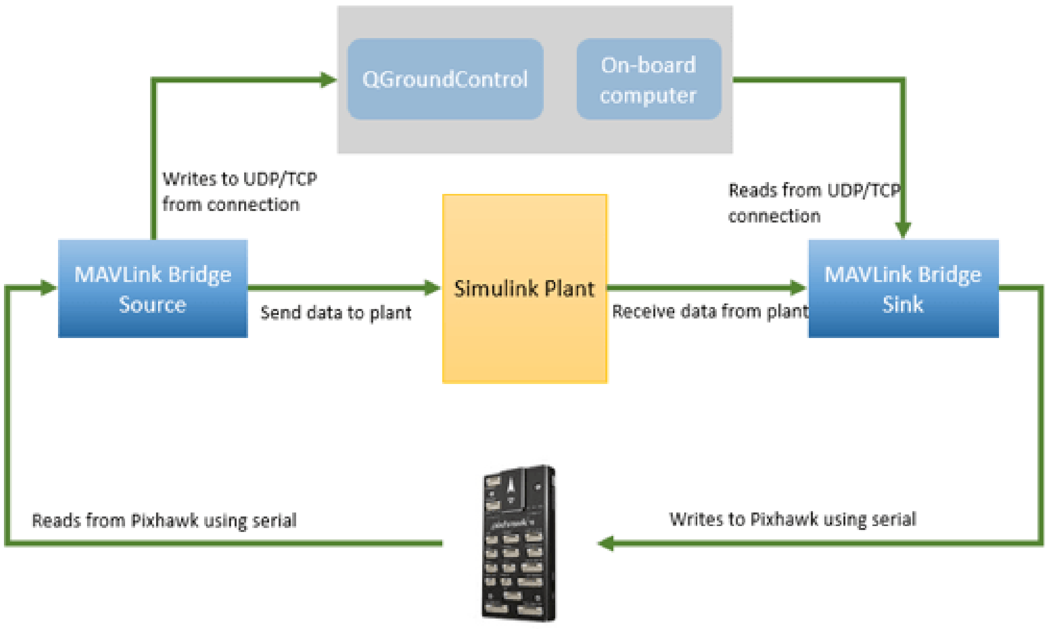


Figure 3.5: PX4 HIL workflow [10]

3.4.1 Pixhawk configuration via QGC

Firstly,the firmware upload is needed via QGC: MATLAB 2023b is the compatible with the firmware version 1.12.3. For this project, the stable release v1.12.3 called px4_fmu-v6x_default.px4 has been loaded through the QGC:

- Open QGC and through **Vehicle setup**,select **Firmware**;
- Connect the Pixhawk board to PC via USB type-C;
- From the right column upload the firmware,selecting the version and wait till the download is completed;

Now, the board configuration is needed:

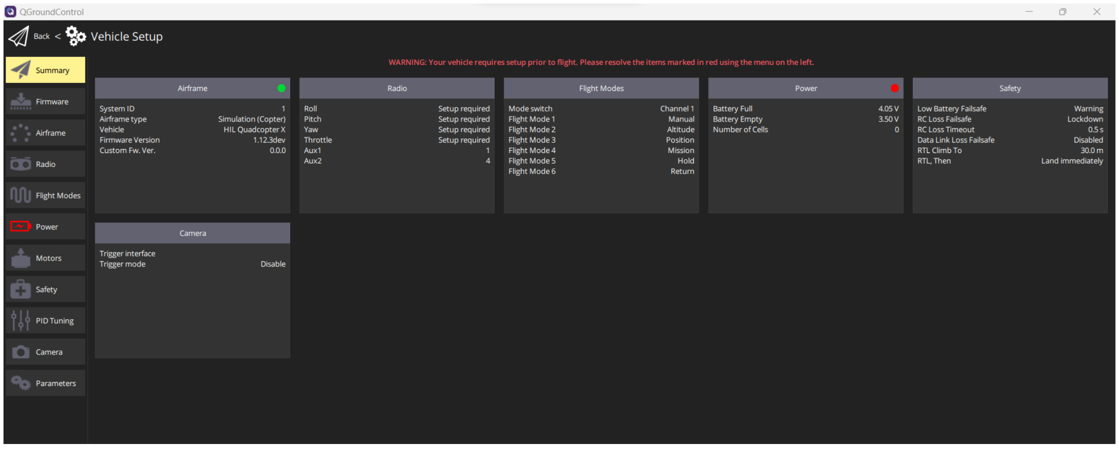
- In the **Safety** section enable **HIL mode**;
- In the **Airframe** section, select **HIL Quadcopter X**;
- in the **Flight Modes** section, set up the 6 channel for the flight mode(the user is free to choose);

Then go to the **Application setting** area. In **Autoconnect to the following devices,** clear all the option except for **UDP**; This setting stops the communication among QGC and Pixhawk through USB, that it's possible just through the simulator bridge. If you want to start communicating via USB, just click on **Pixhawk** option. Now you need to go in **Vehicle setup** section and in the **Parameters** area the following parameters have to be set up:

- **COM_RC_IN_MODE** to Joystick/No RC Checks. This allows joystick input and disables RC input checks.

- **COM_DISARM_LAND** to -1. This disables the timeout for auto-disarm when QGC detects a landing. This helps in avoiding a potential PX4 failsafe when drone motors stops at the highest take-off point momentarily.
- **NAV_ACC_RAD** to 10;
- **NAV_RCL_ACT** to Disabled. This ensures that no RC failsafe actions interfere when not running HITL with a radio control.

Now the sidebar of the **Vehicle setup** does not have any red elements, except for the power sidebar as shown in figure 3.6:this will not be a problem for the simulation. All these settings have been implemented following the MathWorks HIL documention.



WARNING: Your vehicle requires setup prior to flight. Please resolve the items marked in red using the menu on the left.

Airframe	
System ID	1
Airframe type	Simulation (Copter)
Vehicle	HIL Quadcopter X
Firmware Version	1.12.3dev
Custom Fw. Ver.	0.0.0

Radio	
Roll	Setup required
Pitch	Setup required
Yaw	Setup required
Throttle	Setup required
Aux1	1
Aux2	4

Flight Modes	
Mode switch	Channel 1
Flight Mode 1	Manual
Flight Mode 2	Altitude
Flight Mode 3	Position
Flight Mode 4	Mission
Flight Mode 5	Hold
Flight Mode 6	Return

Power	
Battery Full	4.05 V
Battery Empty	3.50 V
Number of Cells	0

Safety	
Low Battery Failsafe	Warning
RC Loss Failsafe	Lockdown
RC Loss Timeout	0.5 s
Data Link Loss Failsafe	Disabled
RTL Climb To	30.0 m
RTL, Then	Land immediately

Camera	
Trigger interface	
Trigger mode	Disable

Figure 3.6: Final QGC setting

3.4.2 Hardware setup via UAV Toolbox Package for PX4 Autopilots

After having set all the parameters in QGC, MATLAB has to be configured.From the main page, go to **Manage Add-ons**, select the UAV package for PX4 autopilots settings:

- Go to **Hardware setup** and install the toolchain, following the instruction:it's important to select the option **Clone PX4 repository and start simulation**.This option will create the "Firmware" folder that has all the tools we need for the project;
- After having selected the target, firmware building is required;
- once firmware building is done, before select Next, the identification of the COM port is required for the firmware upload.So go to **Device Management** and plug the Pixhawk via USB. The COM port will be displayed, use it for the firmware upload.
- After having upload the firmware, select "Get Accelerometer data". A warning will be displayed, do not worry, this is because the firmware is HIL mode.

Ones, everything has been done, typing **open_system('px4demo_QGCWaypointFollower_hitl')** a Simulink project with the default controller linked with QGC is opened. This is the starting point for the design of the implementation of the custom controllers.

There are two ways for the HIL mode, as said before:

- **Monitor and tune**: This option allows you to deploy your controller and tuning the parameter at the same time. As shown in figure 3.7, the Pixhawk communicate via FTDI converter with the Simulink controller and via MAVlink with the simulator.

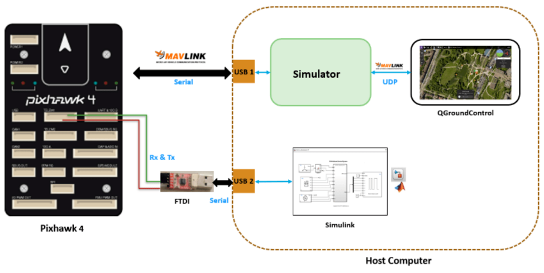


Figure 3.7: HIL with Monitor and Tune

- **Build Deploy & Run**:This option allows you to upload the code without Simulink running. There's just the communication between the simulator and the Pixhawk via MAVlink (through USB). In this way, the simulator is free to run the uploaded code.

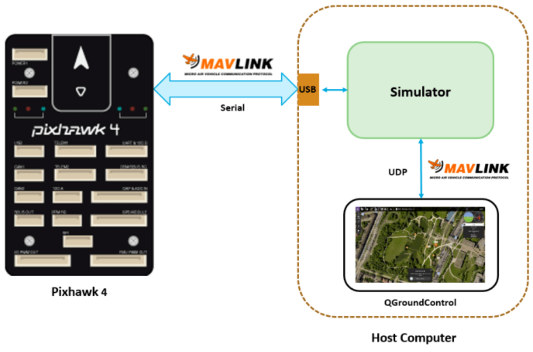


Figure 3.8: HIL with Build,Deploy & run

Chapter 4

Modelling

4.1 Mathematical Model

4.1.1 Euler angles and Coordinate Systems

The mathematical model used, was for a "x" mode quadcopter[11].The two reference frame used for this model are the inertial NED (North,East,Down) reference frame $x - y - z$ and the body reference frame $x' - y' - z'$. In the figure 4.1 the coordinate system used is shown.

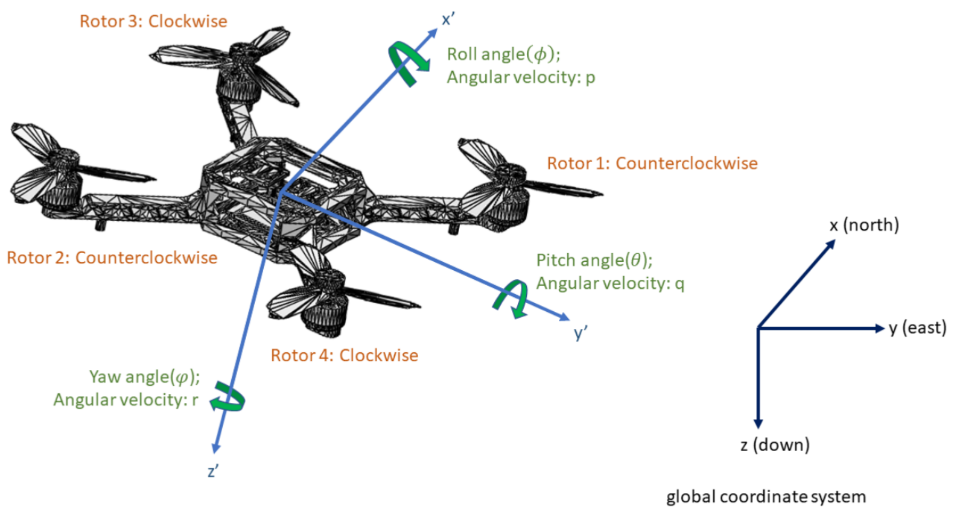
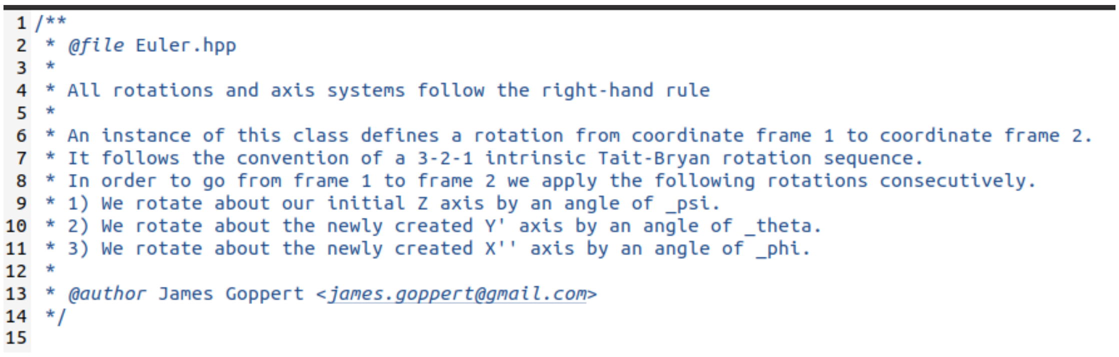


Figure 4.1: Euler Angles and the coordinate system [12]

Here, $\xi = [x, y, z]^T$ are the positions in global coordinate system; ν=[p, q, r]^T are angular velocity in the drone body frame;η=[ϕ,θ,ψ]^T are Euler angles.Also, we show in the figure the motors and the direction of each motor rotation in the drone body reference frame.Regarding Euler angles, we have the following discussions. From global coordinate system to body reference frame, we can first rotate ψ angle around z axis firstly; then rotate θ angle around y axis; finally rotate ϕ angle around x axis. Through this series of rotations of Euler angles, we could obtain

the complete coordinate system conversion.
This model is based on the model jMAVsim. In fact, in order to check this, we can go in the simulator source code. In the Firmware/Matrix folder theres's Euler.hpp where the PX4 rotation order is shown as in figure 4.2



```
/**
 * @file Euler.hpp
 *
 * All rotations and axis systems follow the right-hand rule
 *
 * An instance of this class defines a rotation from coordinate frame 1 to coordinate frame 2.
 * It follows the convention of a 3-2-1 intrinsic Tait-Bryan rotation sequence.
 * In order to go from frame 1 to frame 2 we apply the following rotations consecutively.
 * 1) We rotate about our initial Z axis by an angle of _psi.
 * 2) We rotate about the newly created Y' axis by an angle of _theta.
 * 3) We rotate about the newly created X'' axis by an angle of _phi.
 *
 * @author James Goppert <james.goppert@gmail.com>
 */

```

Figure 4.2: PX4 rotation order: Euler.hpp file

4.1.2 Rotation Matrix

First of all, the reletionship between Euler angles changes rates and body angular velocities is introduced. ν represents the body angular velocity: $\nu = [p, q, r]^T$. From global coordinate system to drone's body frame, we have [13]:

$$\nu = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \cos(\theta)\sin(\phi) \\ 0 & -\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix} \dot{\eta} \tag{4.1}$$

Denoting W as [14]:

$$\begin{aligned} W &= \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \cos(\theta)\sin(\phi) \\ 0 & -\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix}, \\ \nu &= W\dot{\eta} \end{aligned} \tag{4.2}$$

W is invertible if $\theta = (2k-1)\phi/2, (k \in Z)$. (i.e. W is not singular).
From equation (4.1), we have:

$$\dot{\eta} = W^{-1}\nu \tag{4.3}$$

In order to find the rotation matrix between global coordinate reference frame and drone's body frame, the concept of some basic rotation matrices is introduced:

- Rotate ψ angle around z axis (in NED reference):

$$R_z = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

- Rotate θ angle around y axis (in NED reference):

$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{4.5}$$

- Rotate ϕ angle around x axis (in NED reference):

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{4.6}$$

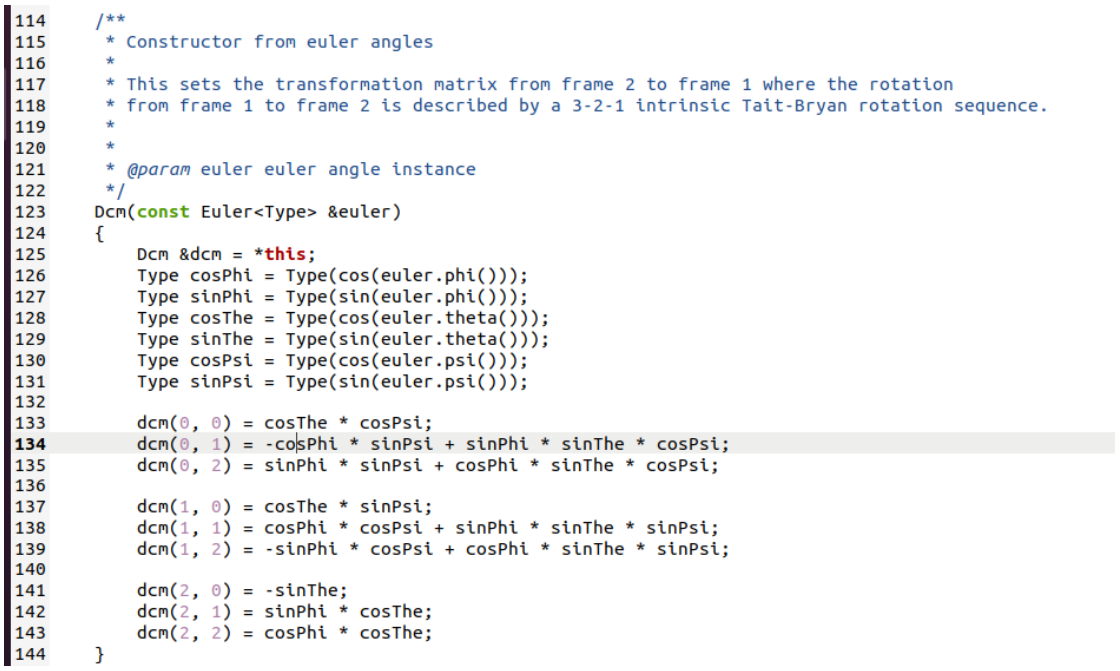
So the rotation matrix from body to inertial frame R_b^e can be derived as follow following the rotation order mentioned above [13]:

$$R_b^e = R_z(\psi)R_y(\theta)R_x(\phi) = \begin{bmatrix} c_\theta c_\psi & -c_\theta s_\psi + c_\psi s_\theta s_\varphi & s_\psi s_\varphi + c_\psi c_\varphi s_\theta \\ c_\theta s_\psi & c_\psi c_\varphi + s_\theta s_\varphi s_\psi & -s_\psi c_\psi + c_\varphi s_\theta s_\psi \\ -s_\theta & c_\theta s_\varphi & c_\theta c_\varphi \end{bmatrix} \tag{4.7}$$

with $c = cos$ and $s = sin$
With the following constraints:

- -$\pi < \phi < \pi$ and $-\pi < \psi < \pi$: the angles ϕ (roll) and ψ (yaw) cover 2πradians. The interval so defined is to indicate that for values greater than or equal to 0 the angle rotates in a counterclockwise direction, following the counterclockwise, following the right hand rule, vice versa for values less than 0;
- $-\pi/2 < \theta < \pi/2$: the angle θ (pitch) covers π radians, and the interval is thus defined for the same reasons as for the other two angles;

It is possible to check that the rotation matrix is the same of the simulator looking for Dcm.hpp file as shown in the figure 4.3



```
    /**
     * Constructor from euler angles
     *
     * This sets the transformation matrix from frame 2 to frame 1 where the rotation
     * from frame 1 to frame 2 is described by a 3-2-1 intrinsic Tait-Bryan rotation sequence.
     *
     *
     * @param euler euler angle instance
     */
    Dcm(const Euler<Type> &euler)
    {
        Dcm &dcm = *this;
        Type cosPhi = Type(cos(euler.phi()));
        Type sinPhi = Type(sin(euler.phi()));
        Type cosThe = Type(cos(euler.theta()));
        Type sinThe = Type(sin(euler.theta()));
        Type cosPsi = Type(cos(euler.psi()));
        Type sinPsi = Type(sin(euler.psi()));

        dcm(0, 0) = cosThe * cosPsi;
        dcm(0, 1) = -cosPhi * sinPsi + sinPhi * sinThe * cosPsi;
        dcm(0, 2) = sinPhi * sinPsi + cosPhi * sinThe * cosPsi;

        dcm(1, 0) = cosThe * sinPsi;
        dcm(1, 1) = cosPhi * cosPsi + sinPhi * sinThe * sinPsi;
        dcm(1, 2) = -sinPhi * cosPsi + cosPhi * sinThe * sinPsi;

        dcm(2, 0) = -sinThe;
        dcm(2, 1) = sinPhi * cosThe;
        dcm(2, 2) = cosPhi * cosThe;
    }
```

Figure 4.3: PX4 Rotation matrix: Dcm.hpp file

4.1.3 Dynamic Equations

Using the rotation matrix in 4.7,the translational dynamics for the quadcopter can be computed as follow:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \frac{1}{m} \cdot R_b^e \cdot \begin{bmatrix} 0 \\ 0 \\ -(T_1 + T_2 + T_3 + T_4) \end{bmatrix} - \frac{K_1}{m} \cdot \begin{bmatrix} \dot{x} \cdot |\dot{x}| \\ \dot{y} \cdot |\dot{y}| \\ \dot{z} \cdot |\dot{z}| \end{bmatrix} \tag{4.8}$$

where T_{1-4} are thrust of each propeller; and the aerodynamic forces are modeled as a second order drag with respect to the velocity in order to stop the vehicle in absence of horizontal thrust, where K_1 is the air drag force coefficient.
Regarding the rotational dynamics, it is given by:

$$\Sigma M = J \cdot \dot{\nu} + \nu \times (J \cdot \nu) \tag{4.9}$$

where J is the mass moment of inertia of the drone:

$$J = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix} \tag{4.10}$$

and J_x, J_y, J_z are inertial moments respectively around x, y, z axis.
From (4.9) the total torques applied to the drone around x,y,z axis can be easily derived:

$$\Sigma M = \begin{bmatrix} (T_2 + T_3 - T_1 - T_4) \cdot l \\ (T_2 + T_4 - T_1 - T_3) \cdot l \\ (Q_1 + Q_2 - Q_3 - Q_4) \end{bmatrix} - K_2 \cdot \nu \cdot |\nu| \tag{4.11}$$

where Q_{1-4} are anti-torques of each motor given by the propellers; and the aerodynamic moments are modeled as a second order drag moments with respect to the angular velocity of the drone to stop the vehicle rotation in the absence of thrust moments, where K_2 is the air drag torque coefficient. Therefore, using equations(4.9)(4.10)(4.11),the following equations are derived:

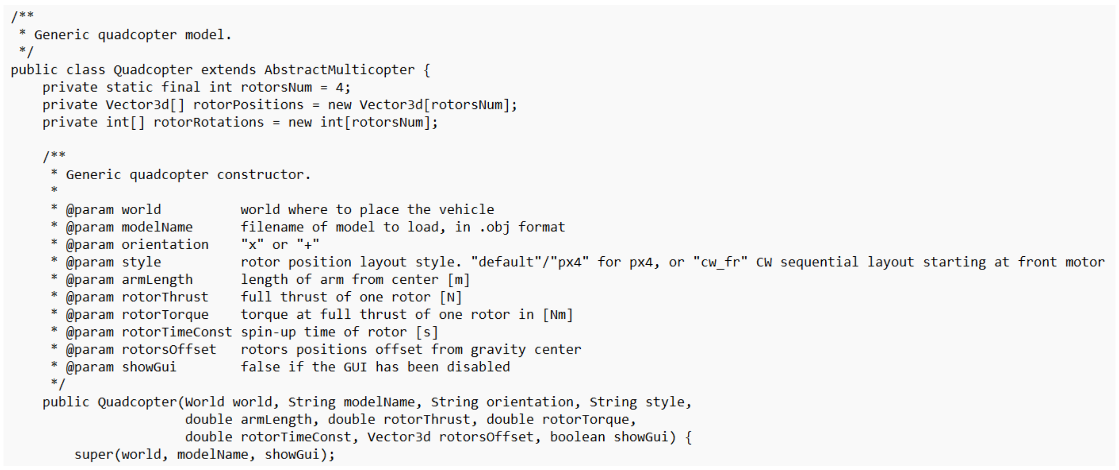
$$\begin{cases} \dot{p} = \frac{1}{J_x}[\frac{\sqrt{2}}{2}l(T_2+T_3-T_1-T_4) - q\cdot r(J_z-J_y) - K_2\cdot p\cdot|p|] \\ \dot{q} = \frac{1}{J_y}[\frac{\sqrt{2}}{2}l(T_1+T_3-T_2-T_4) - p\cdot r(J_x-J_z) - K_2\cdot q\cdot|q|] \\ \dot{r} = \frac{1}{J_z}[\frac{\sqrt{2}}{2}l(Q_1+Q_2-Q_3-Q_4) - p\cdot r(J_y-J_x) - K_2\cdot r\cdot|r|] \end{cases} \tag{4.12}$$

Based on the equations (4.3)(4.7)(4.8)(4.12),the dynamic model of the quadcopter can be derived:

$$\begin{cases} \ddot{x} = -\frac{1}{m}[(\sin\phi\sin\psi+\cos\phi\sin\theta\cos\psi)\cdot(T_1+T_2+T_3+T_4) + K_1\cdot\dot{x}\cdot|\dot{x}|] \\ \ddot{y} = -\frac{1}{m}[(-\sin\phi\cos\psi+\cos\phi\sin\theta\sin\psi)\cdot(T_1+T_2+T_3+T_4) + K_1\cdot\dot{y}\cdot|\dot{y}|] \\ \ddot{z} = -\frac{1}{m}[(\cos\phi\cos\theta)\cdot(T_1+T_2+T_3+T_4) + K_1\cdot\dot{z}\cdot|\dot{z}|] + g \\ \dot{p} = \frac{1}{J_x}[\frac{\sqrt{2}}{2}l(T_2+T_3-T_1-T_4) - q\cdot r(J_z-J_y) - K_2\cdot p\cdot|p|] \\ \dot{q} = \frac{1}{J_y}[\frac{\sqrt{2}}{2}l(T_1+T_3-T_2-T_4) - p\cdot r(J_x-J_z) - K_2\cdot q\cdot|q|] \\ \dot{r} = \frac{1}{J_z}[\frac{\sqrt{2}}{2}l(Q_1+Q_2-Q_3-Q_4) - p\cdot r(J_y-J_x) - K_2\cdot r\cdot|r|] \\ \dot{\phi} = p + q\tan\theta\sin\phi + r\tan\theta\cos\phi \\ \dot{\theta} = q\cos\phi - r\sin\phi \\ \dot{\psi} = q\frac{\sin\phi}{\cos\theta} + r\frac{\cos\phi}{\cos\theta} \end{cases} \tag{4.13}$$

The loaded drone in jMAVsim is characterized by specific parameters that can be modified going into the java code located into the firmware folder.The Simulink plant implements the same default parameters of the virtual environment. If you want to see or modify those data you need to go into these two files in the Firmware folder:

- **Quadcopter.java** In this section the structure of the quadcopter is defined through quadcopter class.Within it, the Quadcopter attribute is passed the parameters shown in the figure 4.4

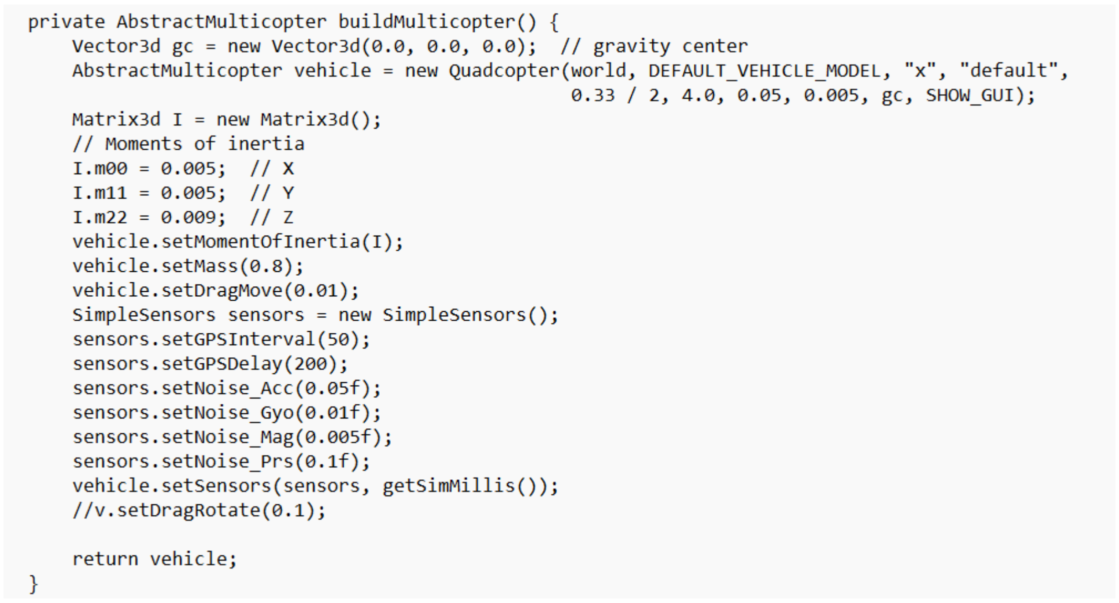


```
/**
 * Generic quadcopter model.
 */
public class Quadcopter extends AbstractMulticopter {
    private static final int rotorsNum = 4;
    private Vector3d[] rotorPositions = new Vector3d[rotorsNum];
    private int[] rotorRotations = new int[rotorsNum];

    /**
     * Generic quadcopter constructor.
     *
     * @param world          world where to place the vehicle
     * @param modelName      filename of model to load, in .obj format
     * @param orientation    "x" or "+"
     * @param style          rotor position layout style. "default"/"px4" for px4, or "cw_fr" CW sequential layout starting at front motor
     * @param armLength      length of arm from center [m]
     * @param rotorThrust    full thrust of one rotor [N]
     * @param rotorTorque    torque at full thrust of one rotor in [Nm]
     * @param rotorTimeConst spin-up time of rotor [s]
     * @param rotorsOffset   rotors positions offset from gravity center
     * @param showGui        false if the GUI has been disabled
     */
    public Quadcopter(World world, String modelName, String orientation, String style,
                      double armLength, double rotorThrust, double rotorTorque,
                      double rotorTimeConst, Vector3d rotorsOffset, boolean showGui) {
        super(world, modelName, showGui);
```

Figure 4.4: Parameters given to the quadcopter class

- **Simulator.java** In this section there is the parameter initialization of the Quadcopter object that builds the model as shown in the figure 4.5



```java
private AbstractMulticopter buildMulticopter() {
    Vector3d gc = new Vector3d(0.0, 0.0, 0.0);  // gravity center
    AbstractMulticopter vehicle = new Quadcopter(world, DEFAULT_VEHICLE_MODEL, "x", "default",
                                             0.33 / 2, 4.0, 0.05, 0.005, gc, SHOW_GUI);
    Matrix3d I = new Matrix3d();
    // Moments of inertia
    I.m00 = 0.005;  // X
    I.m11 = 0.005;  // Y
    I.m22 = 0.009;  // Z
    vehicle.setMomentOfInertia(I);
    vehicle.setMass(0.8);
    vehicle.setDragMove(0.01);
    SimpleSensors sensors = new SimpleSensors();
    sensors.setGPSInterval(50);
    sensors.setGPSDelay(200);
    sensors.setNoise_Acc(0.05f);
    sensors.setNoise_Gyo(0.01f);
    sensors.setNoise_Mag(0.005f);
    sensors.setNoise_Prs(0.1f);
    vehicle.setSensors(sensors, getSimMillis());
    //v.setDragRotate(0.1);

    return vehicle;
}
```

Figure 4.5: Parameters definition passed to the quadcopter object

Chapter 5

Control

The main goal of this chapter is to design different types of controllers and deploying them on PX4 firmware and analyse their performances.The controllers implemented will compute the amount of thrust on each propeller in order to force the UAV to have a certain orientation,consequently a desired position. Firstly in this chapter, under actuated system are analysed and the applied solution will be discussed.Later in this chapter the controllers implemented into the inner loop are described. For each controller HIL is performed.In conclution, the results of the different controllers will be described whith respect to some Key Persormance Indicator.

5.1 Control Architecture

A quadcopter is an under-actuated system.When a system is under-actuated, it means that has less input then the variable to be controlled.A quadrotors has 6 degrees of freedom and just 4 control variables.In order to fix this problem, two control loops are designed:the outer and the inner control loop. The outer control loop is an altitude controller and it creates the missing euler angles references for the inner loop.The inner loop is an attitude controller that takes as input the two missing references created by the outer plus the known one (in our case the desired yaw).In 5.1 the quadrotor control architecture is shown.

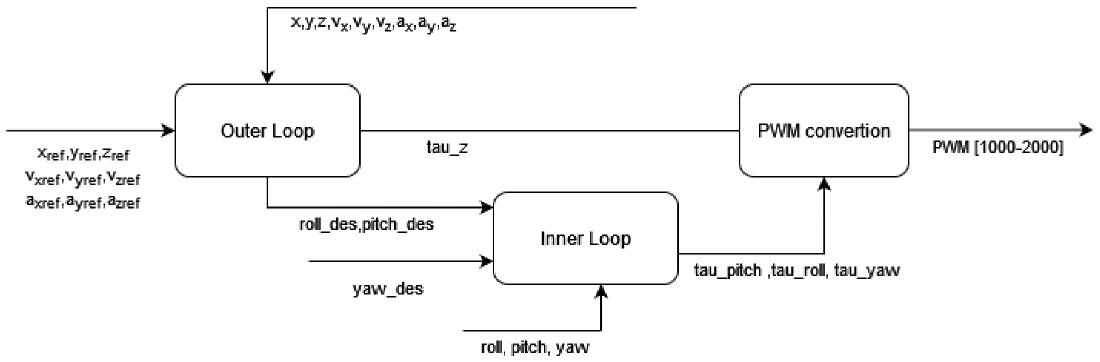


Figure 5.1: Quadrotor control architecture

5.1.1 Outer Loop and Convertion Block

In order to solve the problem of under actuated system, the outer loop is designed. It creates the missing references for the inner loop.In this project, the reference is the position and the yaw angle ψ and their derivatives. The outer loop refers to the position control loop.It is always implemented as a PID controller,while the inner loop has different controllers implemented.The relationship that links the attitude and the linear acceleration is built on the position error PID closed-loop equations of the drone.

Position control design

The outer loop design has been taken from [13]. Taking into account the matrix form of traslational equation with respect to the Earth Frame:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = m \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ -u_1 \end{bmatrix} - k_1 I \begin{bmatrix} \dot{x}|\dot{x}| \\ \dot{y}|\dot{y}| \\ \dot{z}|\dot{z}| \end{bmatrix} \tag{5.1}$$

kwnowing that R is the rotation matrix defined as follow:

$$R = \begin{bmatrix} c_\theta c_\psi & -c_\theta s_\psi + c_\psi s_\theta s_\varphi & s_\psi s_\varphi + c_\psi c_\varphi s_\theta \\ c_\theta s_\psi & c_\psi c_\varphi + s_\theta s_\varphi s_\psi & -s_\psi c_\psi + c_\varphi s_\theta s_\psi \\ -s_\theta & c_\theta s_\varphi & c_\theta c_\varphi \end{bmatrix} \tag{5.2}$$

with $c = cos$ and $s = sin$

Control along z

Deriving the traslation equation along z axis [13]:

$$\ddot{z} = g - \frac{1}{m}(cos(\theta)cos(\varphi))u_1 - \frac{k_1}{m}\dot{z}|\dot{z}| \tag{5.3}$$

with $u_1 = T_1 + T_2 + T_3 + T_4 = 4T_{max}(\tau_T - 1)$. Defing z_r as the reference for z, the control law is the following:

$$\tau_T = 1 - \frac{m}{4T_{max}(\cos\varphi)(\cos\theta)}(-g + k_1\dot{z}|\dot{z}| + v_z) \tag{5.4}$$

where $v_z = \dot{z}_r - k_{z_1}(\dot{z} - \dot{z}_r) - k_{z_0}(z - z_r) + w$.In fact, replacing τ_T into the model:

$$\ddot{z} = v_z \tag{5.5}$$

defing the error as $e_z = z - z_r$,we have:

$$\ddot{e_z} + k_{z_1}\dot{e_z} + k_{z_0}e_z = 0. \tag{5.6}$$

This is a LTI error system: the choice of k_{z_1} and k_{z_0} is such that:

$$P(s) = s^2 + k_{z_1}s + k_{z_0} \tag{5.7}$$

have all the real poles strictly less than 0.Being a second order system, two poles have to be chosen such that : $k_{z_1} = -(p_{z_1} + p_{z_2})$ and $k_{z_0} = p_{z_1}p_{z_2}$. The same has been done to the other control laws

Robust control action w

w is the robust control action defined as:

$$w = \frac{\rho}{\|\zeta\|}\zeta \qquad\qquad \rho > 0 \tag{5.8}$$

where ζ is defined as:

$$\zeta = D^T Q\xi \tag{5.9}$$

being $D = \begin{bmatrix} O \\ I \end{bmatrix}$ and Q the solution to the the following equation:

$$\widetilde{\mathcal{H}}^T Q + Q\widetilde{\mathcal{H}} = -P \tag{5.10}$$

where $\widetilde{\mathcal{H}}$ is:

$$\widetilde{\mathcal{H}} = (\mathcal{H} - \mathcal{D}\mathcal{K}) = \begin{bmatrix} \mathcal{O} & I \\ -\mathcal{K}_{\mathcal{P}} & -\mathcal{K}_D \end{bmatrix} \tag{5.11}$$

is a matrix whose eigenvalues all have negative real parts — K_P and K_D being positive definite — which allows the desired error system dynamics to be prescribed.

Control along x and y

Defining the translation equations for small angles variations on the x and y axis [13]:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = -\frac{4T_{\max}(\tau_T - 1)}{m} \begin{bmatrix} \sin\psi & \cos\psi \\ -\cos\psi & \sin\psi \end{bmatrix} \begin{bmatrix} \varphi \\ \theta \end{bmatrix} - \frac{1}{m} \begin{bmatrix} k_1 & 0 \\ 0 & k_1 \end{bmatrix} \begin{bmatrix} \dot{x}|\dot{x}| \\ \dot{y}|\dot{y}| \end{bmatrix} \tag{5.12}$$

τ_T is already known from the previous calculations,so all the data needed to compute ϕ and θ are known. Rewriting the system in a simpler way:

$$F = -\frac{4T_{\max}(\tau_T - 1)}{m} \begin{bmatrix} \sin\psi & \cos\psi \\ -\cos\psi & \sin\psi \end{bmatrix} \tag{5.13}$$

$$G = \frac{1}{m} \begin{bmatrix} k_1 & 0 \\ 0 & k_1 \end{bmatrix} \begin{bmatrix} \dot{x}|\dot{x}| \\ \dot{y}|\dot{y}| \end{bmatrix} \tag{5.14}$$

So it becomes:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = F \begin{bmatrix} \varphi \\ \theta \end{bmatrix} - G \tag{5.15}$$

the chosen control law is:

$$\begin{bmatrix} \varphi \\ \theta \end{bmatrix} = F^{-1}(V_{xy} + G) \tag{5.16}$$

The control action is defined as follows [13]:

$$V_{xy} = -k_{xy1} \begin{bmatrix} (\dot{x} - \dot{x}_r) \\ (\dot{y} - \dot{y}_r) \end{bmatrix} - k_{xy0} \begin{bmatrix} x - x_r \\ y - y_r \end{bmatrix} + \begin{bmatrix} \ddot{x}_r \\ \ddot{y}_r \end{bmatrix} + w \tag{5.17}$$

whew w has been preavious defined. Replacing V_{xy} into the translation system equation:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = FF^{-1}(V_{xy} + G) - G = V_{xy} \tag{5.18}$$

Now, the outer loop outputs are set τ_T ,ϕ_r, θ_r. τ_T goes directly into the PWM convertion block and ϕ_r,θ_r go to the inner loop.

5.1.2 PWM convertion

This block is responsable to create the virtual control input to the propellers.The thrust and torques are inputs that are mappend in four control variables.The control variable in PX4 firmaware is the PWM that ranges from 1000 to 2000. Taking into account the mathematical model preaviusly described [13]:

$$\begin{cases} m\dot{V} = P + RF_B - F_A \\ J\dot{\omega} + \omega \times J\omega = M_B - M_A \\ \dot{\eta} = W\omega \end{cases} \tag{5.19}$$

where:

$$F_B = \begin{bmatrix} 0 \\ 0 \\ -u_1 \end{bmatrix} \qquad M_B = \begin{bmatrix} \frac{\sqrt{2}}{2} l u_2 \\ \frac{\sqrt{2}}{2} l u_3 \\ u_4 \end{bmatrix}$$

where u_i(for $i = 1, ...,4$) are the virtual control inputs. The model has to be rewritten in function of the τ_k signals that comes from the controllers. Making the following assumtion:

- The force that every propellers can generate is $T_i \in [0, T_{max}]$, with $T_{max} = 4$;
- The torque that every propellers can generate is $Q_i = \frac{c_D}{c_L} T_i \in [0, Q_{max}]$ with $Q_{max} = 0.05$ and there c_D is the linear friction coefficient that is opposite to the propellers thrust and c_L is the angular friction coefficient opposing the rotation of rotor blades;

The normalized force generated by the i-th motor is $v_i = \frac{T_i}{T_{max}} \in$ [0,1]. The relationship between the virtual inputs u_i, the control signals $\tau_T, \tau_R, \tau_P, \tau_Y$ and the motor forces is:

$$\begin{cases} u_1 = 4T_{\max}(\tau_T - 1) = T_1 + T_2 + T_3 + T_4 \\ u_2 = -4T_{\max}\tau_R = -T_1 + T_2 + T_3 - T_4 \\ u_3 = -4T_{\max}\tau_P = T_1 - T_2 + T_3 - T_4 \\ \frac{c_L}{c_D} u_4 = 4T_{\max}\tau_Y = T_1 + T_2 - T_3 - T_4 \end{cases} \tag{5.20}$$

Dividing by T_{max}:

$$\begin{cases} 4(\tau_T - 1) = v_1 + v_2 + v_3 + v_4 \\ -4\tau_R = -v_1 + v_2 + v_3 - v_4 \\ -4\tau_P = v_1 - v_2 + v_3 - v_4 \\ 4\tau_Y = v_1 + v_2 - v_3 - v_4 \end{cases} \tag{5.21}$$

Rewriting the system in matrix form as follows:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 4(\tau_T - 1) \\ -4\tau_R \\ -4\tau_P \\ 4\tau_Y \end{bmatrix} \tag{5.22}$$

that:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \tau_T \\ \tau_R \\ \tau_P \\ \tau_Y \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{5.23}$$

where the matrix:

$$M = \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \tag{5.24}$$

is the **Mixer Matrix** of the **Mixer and Send to actuator** block. Here PWM scaling also takes place, so that the vector is generated:

$$v' = \begin{bmatrix} v'_1 \\ v'_2 \\ v'_3 \\ v'_4 \end{bmatrix} \tag{5.25}$$

which will go directly into the input to the simulated ESCs to generate the force and thrust torques to control the drone. The scaling is defined as :

$$v'_i = v_i(P_{max} - P_{min}) + P_{min} \in [P_{min}, P_{max}] \tag{5.26}$$

with $P_{max} = 2000$ and $P_{min} = 1000$.So we have:

$$v'_i = 1000v_i + 1000 \tag{5.27}$$

and as a result:

$$v_i = \frac{v'_i}{1000} - 1 \tag{5.28}$$

In 5.2, there's the complete PWM allocation scheme,starting from the control signals τ_i to the actuation forces, going through PWM scaling and and their respective ESCs.

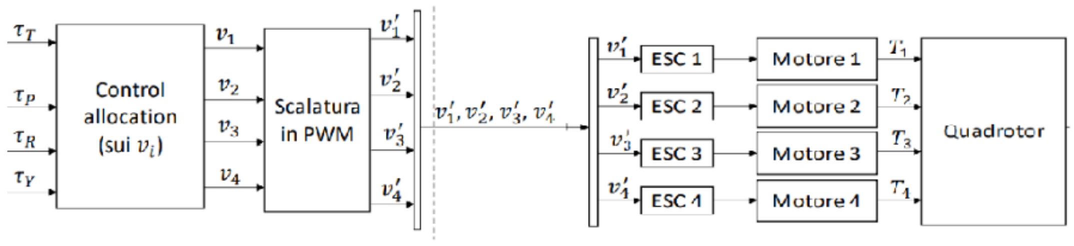


Figure 5.2: PWM allocation [13]

5.2 Inner Loop

In this section, all the inner loop controllers are analysed.

5.2.1 PD Controller

The first controller implemented is the Proporsional-Derivative controller. This controller multiplies by a proportional and a derivative gain the state error and its derivative. The gains are square matrices one for the state vector, the other one for its derivative:

$$K_P = \begin{bmatrix} K_{P_\varphi} & 0 & 0 \\ 0 & K_{P_\theta} & 0 \\ 0 & 0 & K_{P_\psi} \end{bmatrix} \tag{5.29}$$

$$K_D = \begin{bmatrix} K_{D_\varphi} & 0 & 0 \\ 0 & K_{D_\theta} & 0 \\ 0 & 0 & K_{D_\psi} \end{bmatrix} \tag{5.30}$$

The final outputs are:

$$\begin{aligned} \tau_{\varphi} &= K_{P_{\varphi}}(\varphi_{ref} - \varphi) + K_{D_{\varphi}}(\dot{\varphi}_{ref} - \dot{\varphi}) \\ \tau_{\theta} &= K_{P_{\theta}}(\theta_{ref} - \theta) + K_{D_{\theta}}(\dot{\theta}_{ref} - \dot{\theta}) \\ \tau_{\psi} &= K_{P_{\psi}}(\psi_{ref} - \psi) + K_{D_{\psi}}(\dot{\psi}_{ref} - \dot{\psi}) \end{aligned} \tag{5.31}$$

5.2.2 MRAC

Model reference adaptive control (MRAC) is a control methodology that dynamically adjusts control gains based on errors computed relative to a reference model. It is particularly beneficial in scenarios involving uncertainties in the system model, variations in system parameters, and changes in environmental conditions (such as wind). By continuously updating control gains, MRAC ensures robust performance even in the presence of these uncertainties and variations. The reference model is:

$$\dot{x}_{ref} = A_{ref}x_{ref} + B_{ref}r(t) \tag{5.32}$$

then,giving any bounded signals $r(t)$, the control input $u(t)$ has to be chosen such that:

$$\lim_{t\to\infty} \|x - x_{ref}\| = 0$$

The control law implemented is the follwing:

$$u_{MRAC}(t) = K_X(t)x(t) + K_R(t)r(t),$$

where:

$$\begin{aligned} K_X &= \Phi_X + S^T y_e x^T \beta_X, \quad \text{and} \quad \dot{\Phi}_X = S^T y_e x^T \alpha_X, \\ K_R &= \Phi_R + S^I y_e r^T \beta_R, \quad \text{and} \quad \dot{\Phi}_R = S^T y_e r^T \alpha_R, \end{aligned} \tag{5.33}$$

where y_e is computed as:

$$y_e = B_m^T P_e x_e, \quad \text{with } P_e \text{ being the solution of } P_e A_m + A_m^T P_e = -Q, \tag{5.34}$$

where Q is a stricly positive matrix.The terms α_x,α_r,β_x,β_r are stricly positive diagonal matrices that define the evolution of the gains over the time.These terms are the one that needs to be tuned.

5.2.3 EMRAC

The EMRAC implementation enhances the MRAC algorithm by integrating adaptive integral and adaptive switching control mechanisms. These additions fortify the control system's capabilities, enabling it to effectively adjust to changing dynamics and conditions [15]. The controller actions are as follows:

$$u(t) = u_{MRAC} + u_D + u_I + u_N + u_{LQR}; \tag{5.35}$$

where:

$$u_{MRAC}(t) = K_X(t)x(t) + K_R(t)r(t), \tag{5.36}$$

$$u_D(t) = K_D(t)d(t), \tag{5.37}$$

$$u_I(t) = K_I(t)x_I(t) \tag{5.38}$$

where x_I is the integral of the tracking error computed as:

$$\dot{x}_I = x_e - \sigma_I\left(||x_I||\right)\rho_e x_I, \quad \text{and} \quad x_e = x_m - x, \tag{5.39}$$

where x_e is the state tracking error, ρ is a positive diagonal matrix and σ_I is the σ-modification strategy to prevent the drift of the integral of the tracking error.
The adaptive integral action u_I is designed to improve the tracking of the reference. The σ-modification strategy and the adaptive switching control action $u_N(t)$ increases the robustness of the closed-loop tracking performance.
The EMRAC controller architecture is shown in 5.3

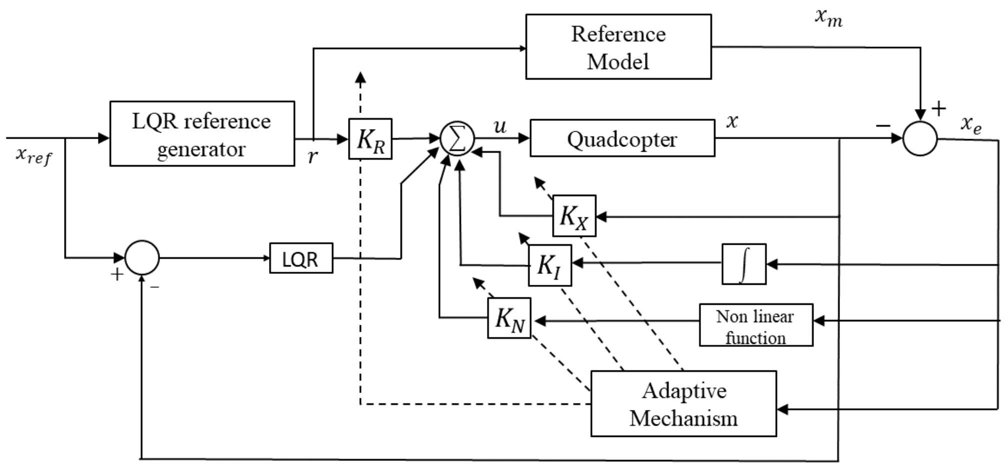


Figure 5.3: EMRAC architecture

Reference system

EMRAC is based on a reference system that describes the behaviour expected from the original system. The reference system is defined as [15]:

$$\dot{x}_m = A_m x_m + B_m r + E_m d \tag{5.40}$$

where:

- $x_m \in \mathbb{R}^{n_x}$ the reference model state, where n_x is the dimensions of the state space;
- $r \in \mathbb{R}^{n_u}$ is the reference input assumed to be bounded, with n_u the dimension of the control input;
- $d \in \mathbb{R}^{n_d}$ the measurable disturbance and n_d its dimension;
- $A_m \in \mathbb{R}^{n_x \times n_x}$, $B_m \in \mathbb{R}^{n_x \times n_u}$, $E_m \in \mathbb{R}^{n_x \times n_d}$ are the dynamics matrix, the input matrix and the disturbance matrix of the reference model respectively;

The plant state space equation is the following:

$$\dot{x} = Ax + Bu + Ed + G, \quad x(t_0) \in \mathbb{R}^{n_x} \tag{5.41}$$

where:

- $x \in \mathbb{R}^{n_x}$ is the state vector of the plant,n_x is the dimensions of the state space;
- $u \in \mathbb{R}^{n_u}$ is the plant input vector,n_u is the dimensions of the control input;
- $t_0 \in \mathbb{R}$ is the initial time instant
- $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $E \in \mathbb{R}^{n_x \times n_d}$ are the dynamics matrix,the input matrix and the matrix of the measurable disturbance, respectively, which are assumed constant with unknown entries.
- $G \in \mathbb{R}^{n_x}$ is the nonmeasurable disturbance;

Both the measurable and the non-measurable disturbances are considered to be bounded, meaning that there exists $G_\infty > 0$ and $d_\infty > 0$ such that

$$\|G(t)\| \leq G_\infty, \quad \|d(t)\| \leq d_\infty \ \forall t \geq t_0 \tag{5.42}$$

It is assumed that there exist some constant matrices $\hat{\Phi}_R \in \mathbb{R}^{n_u \times n_u}$, $\hat{\Phi}_X \in \mathbb{R}^{n_u \times n_x}$, $\hat{\Phi}_D \in \mathbb{R}^{n_u \times n_d}$, and an invertible matrix $S \in \mathbb{R}^{n_u \times n_u}$ such that following matching conditions are satisfied

$$\begin{aligned} B_m &= B\hat{\Phi}_R, \\ A_m &= A + B\hat{\Phi}_X = A + B_m\hat{\Phi}_R^{-1}\hat{\Phi}_X, \\ E_m &= E + B\hat{\Phi}_D = A + B_m\hat{\Phi}_R^{-1}\hat{\Phi}_D, \\ P_\phi &= \hat{\Phi}_R S = S^T \hat{\Phi}_R^T > 0 \end{aligned} \tag{5.43}$$

The ideal gains $\hat{\Phi}_R, \hat{\Phi}_X, \hat{\Phi}_D$ can be collected in the matrix $\hat{\Phi} \in \mathbb{R}^{n_u \times n_w}$,with $n_w = 2n_x + n_u + n_d$ and in the vector $\hat{\phi} \in \mathbb{R}^{n_u n_w}$ defined as

$$\hat{\Phi} = \begin{bmatrix} \hat{\Phi}_X \ \hat{\Phi}_R \ \widehat{\Phi}_D \ \widehat{\Phi}_I \end{bmatrix} = \begin{bmatrix} \hat{\phi}_1 & \hat{\phi}_2 & \cdots & \widehat{\phi}_{nw-1} & \widehat{\phi}_{n_w} \end{bmatrix} \tag{5.44}$$

$$\hat{\Phi}_I = \mathcal{O}_{n_u, n_x} \tag{5.45}$$

$$\hat{\phi} = \begin{bmatrix} \phi_1{}^T \phi_2{}^T \cdots \phi_{n_w-1}{}^T \phi_{n_w}{}^T \end{bmatrix}^T, and \, ||\hat{\phi}|| \leq M_\phi \tag{5.46}$$

Adaptive gains

The adaptive gains are defined as follows [15] :

$$K_X = \Phi_X + S^T y_e x^T \beta_X, \quad \text{and} \quad \dot{\Phi}_X = S^T y_e x^T \alpha_X + F_X, \tag{5.47}$$

$$K_R = \Phi_R + S^T y_e r^T \beta_R, \quad \text{and} \quad \dot{\Phi}_R = S^T y_e r^T \alpha_R + F_R, \tag{5.48}$$

$$K_D = \Phi_D + S^T y_e d^T \beta_D, \quad \text{and} \quad \dot{\Phi}_D = S^T y_e d^T \alpha_D + F_D, \tag{5.49}$$

$$K_I = \Phi_I + S^T y_e x_I^T \beta_I, \quad \text{and} \quad \dot{\Phi}_I = S^T y_e x_I^T \alpha_I + F_I, \tag{5.50}$$

where $\alpha_X, \beta_X, \alpha_I, \beta_I \in \mathbb{R}^{n_x \times n_x}$,$\alpha_R, \beta_R \in \mathbb{R}^{n_u \times n_u}$,and $\alpha_D, \beta_D \in \mathbb{R}^{n_d \times n_d}$ are strictly positive diagonal matrices and $F_x, F_I \in \mathbb{R}^{n_u \times n_x}$,$F_R \in \mathbb{R}^{n_u \times n_u}$ and $F_D \in \mathbb{R}^{n_u \times n_d}$ are the locking strategies for preventing the unbounded evolution of the gains in the presence of disturbances and unmodeled dynamics defined as:

$$F_X = -\sigma_\phi(||\phi||)\Phi_X \rho_X, \tag{5.51}$$

$$F_I = -\sigma_\phi(||\phi||)\Phi_I\rho_I, \tag{5.52}$$

$$F_R = -\sigma_\phi(||\phi||)\Phi_R\rho_R, \tag{5.53}$$

$$F_D = -\sigma_\phi(||\phi||)\Phi_D\rho_D, \tag{5.54}$$

where ρ_X, $\rho_I \in \mathbb{R}^{n_x \times n_x}$, $\rho_R \in \mathbb{R}^{n_u \times n_u}$ and $\rho_D \in \mathbb{R}^{n_d \times n_d}$ are strictly positive diagonal matrices. Moreover,y_e is computed as

$$y_e = B_m^T P_e x_e, \quad \text{with } P_e \text{ being the solution of } P_e A_m + A_m^T P_e = -Q, \tag{5.55}$$

where $Q \in \mathbb{R}^{n_x \times n_x}$ is a strictly positive matrix.

σ-modification

The σ modification strategy is implemented to prevent the unbounded evolutions of the gains. In order to prevent the drift of the integral error, in 5.39 the following σ modification strategy is implemented [15]:

$$\sigma_I(||x_I||) = \begin{cases} 0 \quad if \quad ||\phi|| \leq \hat{M}_I \\ \eta_I \left(\frac{||x_I||}{\widehat{M}_I} - 1 \right) \quad if \quad \hat{M}_I \leq ||x_I|| \leq 2\hat{M}_I \\ \eta_I \quad if \quad ||x_I|| \geq 2\hat{M}_I \end{cases} \tag{5.56}$$

where η_I and $\widehat{M_I}$ are strictly positive constants.
In 5.51,5.53,5.52 the σ modification is implemented as:

$$\sigma_\phi(||\phi||) = \begin{cases} 0 \quad if \quad ||\phi|| \leq \hat{M}_\phi \\ \eta_\phi \left(\frac{||x_\phi||}{\widehat{M}_\phi} - 1 \right) \quad if \quad \hat{M}_\phi \leq ||x_\phi|| \leq 2\hat{M}_\phi \\ \eta_\phi \quad if \quad ||x_\phi|| \geq 2\hat{M}_\phi \end{cases} \tag{5.57}$$

The constraints are computed as:

$$\widehat{M_\phi} \geq \sqrt{\frac{\lambda_{\max}(\Gamma_\rho \Gamma_a^{-1} \otimes P_\phi^{-1})}{\lambda_{\min}(\Gamma_\rho \Gamma_a^{-1} \otimes P_\phi^{-1})}} M_\phi, \quad and \quad \eta_\phi \lambda_{\min}\left(\Gamma_\rho \Gamma_a^{-1} \otimes P_\phi^{-1}\right) > \frac{3}{4}\lambda_{\min}(Q) \tag{5.58}$$

with $\otimes$ being the Kronecker product and the strictly positive matrices Γ_ρ, $\Gamma_a \in \mathbb{R}^{n_w \times n_w}$ defined as:

$$\Gamma_a = \Delta(\alpha_X, \alpha_R, \alpha_D, \alpha_I) = diag(\alpha_1, \alpha_2, \ldots, \alpha_{n_w}) \tag{5.59}$$

$$\Gamma_\rho = \Delta(\rho_X, \rho_R, \rho_D, \rho_I) = diag(\rho_1, \rho_2, ..., \rho_{n_w}) \tag{5.60}$$

Adaptive switching control action

The adaptive switching control action $u_N(t)$ has two different formulations $u_N^{ew}(t)$ and $u_N^{uv}(t)$ [15]:

$$u_N^{(uv)}(t) = K_N^{(uv)}(t)\frac{y_e}{||y_e||}, \; K_N^{(uv)} = S^T \Phi_{N0} \tag{5.61}$$

$$\dot{\Phi}_{N0} = \alpha_{N0} h_0(||y_e||_\Omega) - \sigma_{N0}(||\Phi_{N0}||)\rho_{N0}\Phi_{N0} \tag{5.62}$$

$$u_N^{(ew)}(t) = K_N^{(ew)}(t)\psi(y_e), \qquad K_N^{(ew)} = S^T\Phi_N \tag{5.63}$$

$$\psi(y_e) = [\mathrm{sgn}(y_{e_1})\ \mathrm{sgn}(y_{e_2})\ \cdots \mathrm{sgn}(y_{e_{nu}})]^T \tag{5.64}$$

$$\dot{\Phi}_{Nj} = \alpha_{Nj} h_j\left(|y_e|\right) - \sigma_{Nj}\left(||\Phi_{Nj}||\right)\rho_{Nj}\Phi_{Nj}, \quad j = 1, \cdots, n_u \tag{5.65}$$

where $\Phi_{N0} \in \mathbb{R}$, $\Phi_N = diag\left(\Phi_{N1},\ \Phi_{N2}, \ldots,\ \Phi_{N_{nu}}\right) \in \mathbb{R}^{n_u}$ and the σ modification is defined as:

$$\sigma_{N_j}(||\Phi_{N_j}||) = \begin{cases} 0 \quad if \quad ||\Phi_{N_j}|| \leq \widehat{M}_{N_j} \\ \eta_{N_j}\left(\frac{||\Phi_{N_j}||}{\widehat{M}_{N_j}} - 1\right) \quad if \quad \widehat{M}_{N_j} \leq ||\Phi_{N_j}|| \leq 2\widehat{M}_{N_j} \\ \eta_{N_j} \quad if \quad ||\Phi_{N_j}|| \geq 2\widehat{M}_{N_j} \end{cases} \tag{5.66}$$

where $||y_e||_\Omega$ with $\Omega \in \mathbb{R}^{n_u \times n_u}$ is a strictly positive matrix and $\alpha_{N_j}, \rho_{N_j}, \eta_{N_j}, \widehat{M}_{N_j}, j = 1, \cdots, n_u$ strictly positive constants. $\widehat{M}_{N_j}$ and η_{N_j} are defined as:

$$\widehat{M}_{N0} > \frac{\delta_\infty}{\lambda_{\min}\left(SP^{-1}S^T\right)}, \quad and \quad \widehat{M}_{Nj} > \frac{\delta_{j\infty}}{\widehat{c}_j}, \quad j = 1, \cdots n_u \tag{5.67}$$

The h-function are defined as:

$$h_0 = (||y_e||_\Omega) = \frac{||y_e||_\Omega{}^{\varsigma_0}}{\xi_0 + \gamma_0 ||y_e||_\Omega{}^{\varsigma_0}} \tag{5.68}$$

$$h_j = (\|y_{ej}\|_\Omega) = \frac{|y_e|^{\zeta_j}}{\xi_j + \gamma_j\,|y_e|^j}, \quad j = \ 1, \ \cdot\cdot, n_u \tag{5.69}$$

considering ξ_j, ζ_j, γ_j with $j = 0, \cdots, n_u$ strictly positive constants.

LQR control action

An LQR controller has been used to generate the $r(t)$ reference signal that track the reference euler angles. It has also been used as a direct control action for the quadrotor attitude control, but it is smaller than the adaptive control actions.
LQR is an optimal control techniques that aims to steer the dynamics of the quadrotor while minimizing the energy used in the control [16]. The goal of the LQR control is to steer the state variable from an initial value x_0 to the origin $x_f \ \longrightarrow\ 0$ while minimizing the following cost function:

$$J = \frac{1}{2}\int_{ta}^{tf} (x^T Q_{lqr} x + u^T R_{lqr} u)\, dt + \frac{1}{2} x(t_f)^T H x(t_f) \tag{5.70}$$

where Q,H,R are design weight symmetric matrices such that Q, H are positive semi definite and R is positive definite.
The solution of the optimization problem leads to a control input of the form:

$$v(t) = -R_{lqr}^{-1} B^T K(t) x(t) \tag{5.71}$$

where $K(t)$ is the symmetric matrix solution of the Riccatti differential equation(DRE)

$$\dot{K} = -KA - A^T K - Q_{lqr} + KBR_{lqr}^{-1}B^T K, \quad K(tf) = H \tag{5.72}$$

With the following condition:

- System is linear time invariant (LTI) and controllable
- Cost function is defined over an infinite time interval ($t_f \longrightarrow \infty$)
- Weight matrices are constant with terminal cost $H = 0_{6\times 6}$

$K(t) \rightarrow K_{ss}$ as $t_f \rightarrow \infty$. Where K_{ss} is the solution of the algebraic Riccati equation (ARE)

$$0 = -KA - A^T K - Q_{lqr} + KBR_{lqr}^{-1}B^T K \tag{5.73}$$

derived from the DRE by setting $\dot{K}(t) = 0$.This form of LQR is known as Infinite Horizon LQR problem and can be solved with the **lqr** command in Matlab using the command: "[F,K]=**lqr**(A,B,Q_{lqr},R_{lqr})" where $F = R_{lqr}^{-1}B^T H$. Therefore,5.71 can be rewritten as

$$v(t) = -Fx(t) \tag{5.74}$$

The aim of the LQR algorithm is to achieve the tracking of the desired Euler angles state x_d.Taking into account the following system dynamics:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{5.75}$$

where:

$$\text{A=}\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad and \quad B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Recalling the reference system equation 5.40,where:

$$r(t) = Fx_d \tag{5.76}$$

Defining the following terms:

$$\begin{aligned} A_m &= A - BF \\ B_m &= B \\ E_m &= E \end{aligned} \tag{5.77}$$

and setting:

$$v_{LQR} = F(x_d - x) \tag{5.78}$$

the reference system tracks the euler angles vector state.

Chapter 6

Simulation and Analysis

In this chapter, the trajectory implemented for each simulation is going to be explained. The tuning of all the parameters of all controllers is also presented.

6.1 Simulation Objective

The objective of this project is to test different control algorithm performing the same trajectory, comparing their performances.The control algorithm tested are the following (Outer+Inner):

- PD+PD
- PD+MRAC
- PD+C-EMRAC-UV
- PD+C-EMRAC-EW
- PD+D-EMRAC-UV

The trajectory chosen is a helix with 12 cycles with radius of 8 meters with a climbing rate of 1 meter per cycle.This choice allows us to assess key performance indicators (KPIs) in relation to the number of helix cycles. Since the gains are initially set to zero at the beginning of the simulation, time is required to achieve gain convergence.The simulation starts at 100 seconds and runs for 120 seconds. Before the helix trajectory starts, the drone will hover at 1 meter for the first 100 seconds. The pause will allow the simulation environment to finish its setup and start recording all signals sent to the workspace within Simulink.
All the simulation has been performed with the nominal values of mass and inertia($m = 0.8kg, I_{xx} = I_{yy} = 0.05$ and $I_{zz} = 0.09$) for each control algorithm.
In order to test the potentiality of the adaptive algorithms and to compare their results with the benchmark controllers(PD and MRAC), all controllers previously mentioned have been tested increasing the mass and inertia percentage of the quadrotor as follows:

- +10%
- +20%
- +30%
- +50%

6.2 Simulation parameter

6.2.1 Outer loop tuning

All the controllers have been tested, keeping the same tuning of the outer loop for consistency during comparison.

p_{xy1}	p_{xy2}	p_{z1}	p_{z2}
-1.1	1	-9	-8.8

Table 6.1: Outer loop parameters

where $k_{z_1} = -(p_{z_1} + p_{z_2})$, $k_{z_0} = p_{z_1}p_{z_2}$,$k_{xy_1} = -(p_{xy_1} + p_{xy_2})$ and $k_{xy_0} = p_{xy_1}p_{xy_2}$.

6.2.2 PD tuning

In the equations 6.1 the tuned parameter of the PD controller are shown.

$$K_P = \begin{bmatrix} 0.08 & 0 & 0 \\ 0 & 0.08 & 0 \\ 0 & 0 & 0.8 \end{bmatrix}$$
$$K_D = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \tag{6.1}$$

It can be notice that the gains are the same of x and y-axis while on the z-axis is one order of magnitude greater. This is because the quadcopter is symmetric along x and y so it has the same inertia along those axes $I_{xx} = I_{yy} = 0.05$ while along z-axis $I_{zz} = 0.09$.

6.2.3 MRAC tuning

	MRAC
α_x	[0.1,0.1,0.1,0.1,2,2]I_{nl}
α_r	[0.01,0.01,0.2]I_n
β_x	0.1*α_x
β_r	0.1*α_r

Table 6.2: MRAC α and β parameters

Considering $n = 3$ and $l = 2$, I_{nl} and I_n are defined as a (6×6) and a (3×3) identity matrix, respectively.

6.3 EMRAC tuning

6.3.1 Centralized EMRAC controller

In the centralized EMRAC controller, there's just one EMRAC algorithm that control the orientation of the drone.

	C-EMRAC-UV	C-EMRAC-EW
α_X	[0.1,0.1,0.1,0.1,2,2]I_{nl}	[0.1,0.1,0.1,0.1,0.1,2,2]I_{nl}
α_R	[0.01,0.01,0.2]I_n	[0.01,0.01,0.2]I_n
α_I	[0.1,0.1,0.2,0.2,2,2]I_{nl}	[0.1,0.1,0.2,0.2,2,2]I_{nl}
α_N	100000	[2,5,5]
β_X	$0.1\alpha_X$	$0.1\alpha_X$
β_R	$0.1\alpha_R$	$0.1\alpha_R$
β_I	$0.1\alpha_I$	$0.1\alpha_I$

Table 6.3: Centralized EMRAC α and β parameters

	C-EMRAC-UV	C-EMRAC-EW
ρ_X	$10^{-5}I_{nl}$	$10^{-5}I_{nl}$
ρ_R	$10^{-5}I_n$	$10^{-5}I_n$
ρ_I	$10^{-5}I_n$	$10^{-5}I_n$
ρ_N	$10^{-5}I_{nl}$	$10^{-5}I_{nl}$
M_ϕ	$8 \cdot 10^{-4}$	$8 \cdot 10^{-4}$
$\widehat{M_I}$	4	4
$\widehat{M_N}$	0.15	0.15

Table 6.4: Centralized EMRAC σ-modification parameters

6.3.2 Decentralized EMRAC algorithm

In the decentralized version of this algorithm, a controller for each euler angle is designed.So it means there are three EMRAC controllers to control pitch(ϕ),roll(θ) and yaw(ψ).

	Pitch D-EMRAC	Roll D-EMRAC	Yaw D-EMRAC
α_X	[0.015,0.015]I_n	[0.02,0.02]I_n	[2,2]I_n
α_R	0.015	0.015	0.1
α_I	[3,3]I_n	[3,3]I_n	[8,8]I_n
α_N	10^6	10^6	10^5
β_X	$0.1\alpha_X$	$0.1\alpha_X$	$0.1\alpha_X$
β_R	$0.1\alpha_R$	$0.1\alpha_R$	$0.1\alpha_R$
β_I	$0.1\alpha_I$	$0.1\alpha_I$	$0.1\alpha_I$

Table 6.5: Decentralized EMRAC α and β parameters

considering $n = 2$, I_n is defined as a (2x2) identity matrix.

	Pitch D-EMRAC	Roll D-EMRAC	Yaw D-EMRAC
ρ_X	$10^{-5}I_{nl}$	$10^{-5}I_{nl}$	$10^{-5}\ I_{nl}$
ρ_R	$10^{-5}I_n$	$10^{-5}I_n$	$10^{-5}I_n$
ρ_I	$10^{-5}I_{nl}$	$10^{-5}I_{nl}$	$10^{-5}I_{nl}$
ρ_N	$10^{-5}I_n$	$10^{-5}I_n$	$10^{-5}I_n$
M_ϕ	0.0054	$2 \cdot 10^{-}4$	0.0050
$\widehat{M_I}$	1	1	0.2
$\widehat{M_N}$	0.45	0.4	20
η_I	1	1	1
η_N	1	1	1
η_ϕ	1	1	1

Table 6.6: Decentralized EMRAC σ-modification paramaters

Considering $n = 3$ and $l = 2$, I_{nl} and I_n are defined as a (6×6) and a (3×3) identity matrix, respectively.
M-terms define the threshold values for the activation of the σ-modified strategy. When $||\phi||$,$||\Phi_{Nj}||$ and $||x_I||$ passes or exceeds these pre-defined threshold bands, the σ modification strategy is triggered.

6.4 KPI Results

As it has already been said at the beginning of this chapter, several simulations have been performed testing the drone at nominal and no-nominal conditions with all the controllers algorithms previously described in order to underline the capabilities of the EMRAC. The KPIs used to compare each of the controllers are the Root Mean Square Error (RMSE) and the maximum error(ME).
The RMSE of a vector x_{ref} with respect to the state vector x is computed as follows:

$$RMSE = \sqrt{\frac{\sum\limits_{i=1}^{n} (x_i^{ref} - x_i)}{n}} \tag{6.2}$$

where n is the number of samples in the vector and x_i is the $i - th$ sample of the vector x. Furthermore, ME is computed in the following way:

$$ME = max(x_{ref} - x) \tag{6.3}$$

Having designed different inner controllers with the same outer algorithms, the attitude error has been taken into account. The most important KPI used to compare the attitude tracking performance has been the RMSE attitude error.
The histograms below show the progression of RMSE as a function of number of cycles in helix. The importance of this presentation is that it shows how EMRAC algorithms need time for gains to converge. It is important to note that errors are higher in the early stages of the simulation and decreases as the trajectory cycles increase.

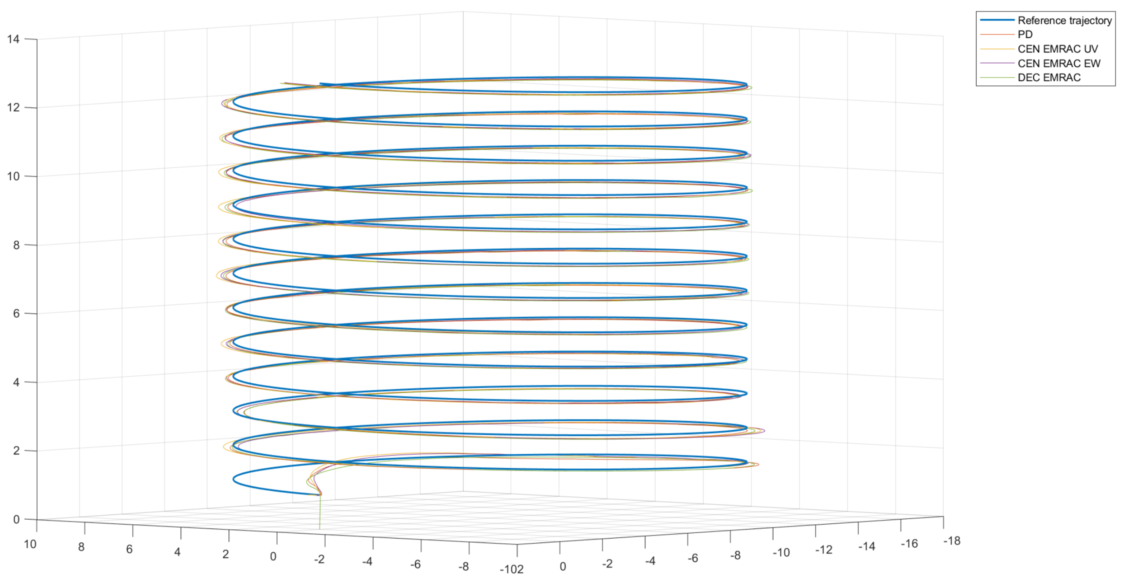


Figure 6.1: Trajectory

In the figure 6.1 a trajectory comparison among the reference and the trajectory performed by each controller is shown.

6.4.1 Nominal mass and inertia conditi.on

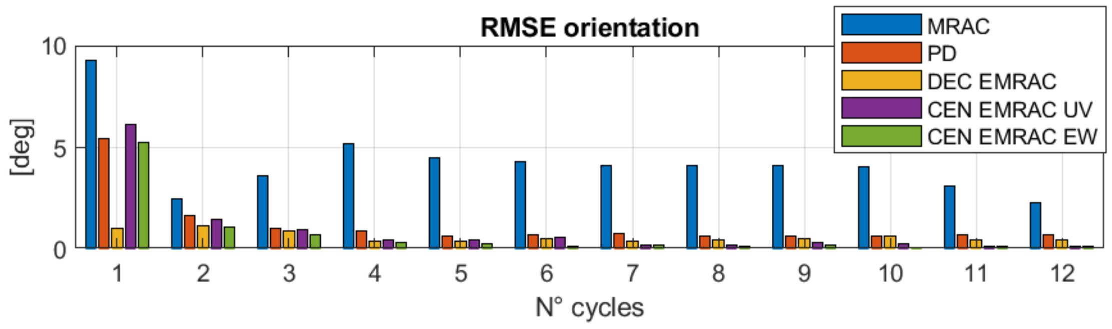


Figure 6.2: Nominal:RMSE orientation

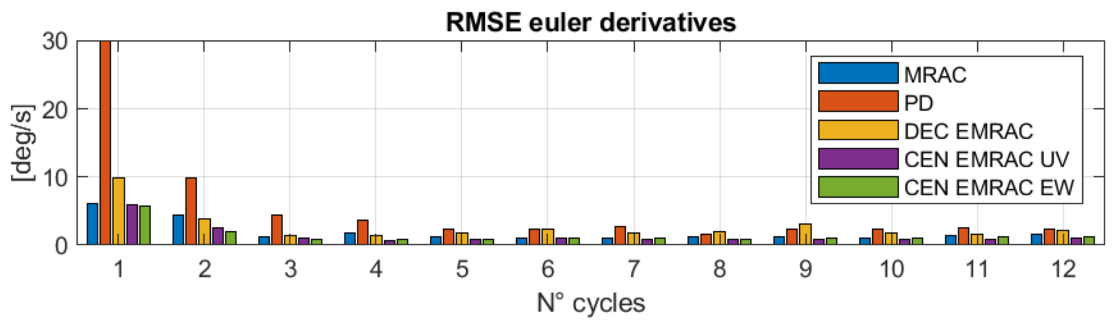


Figure 6.3: Nominal:RMSE euler derivatives

6.4.2 +10 % mass and inertia

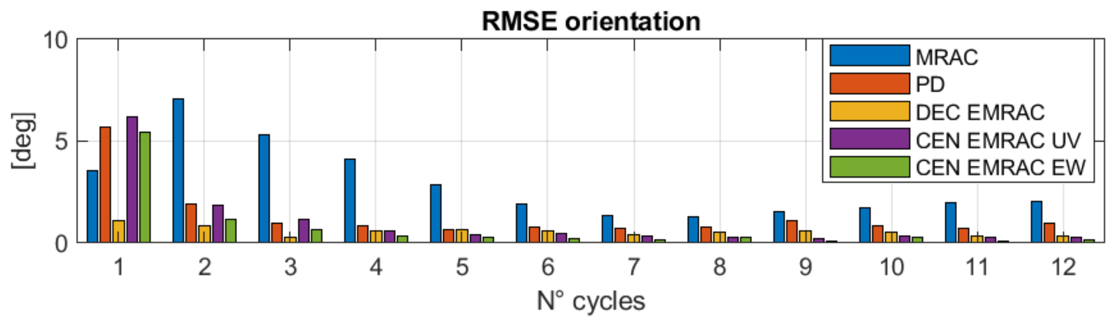


Figure 6.4: 10%:RMSE orientation

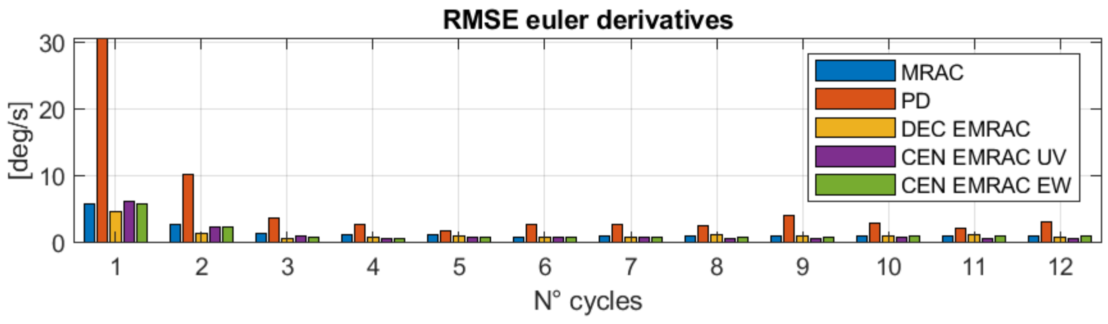


Figure 6.5: 10%:RMSE euler derivatives

6.4.3 +20 % mass and inertia

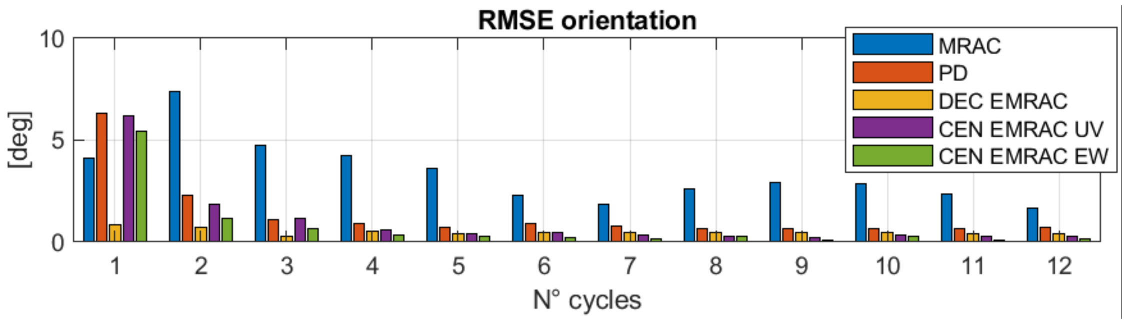


Figure 6.6: 20%:RMSE orientation

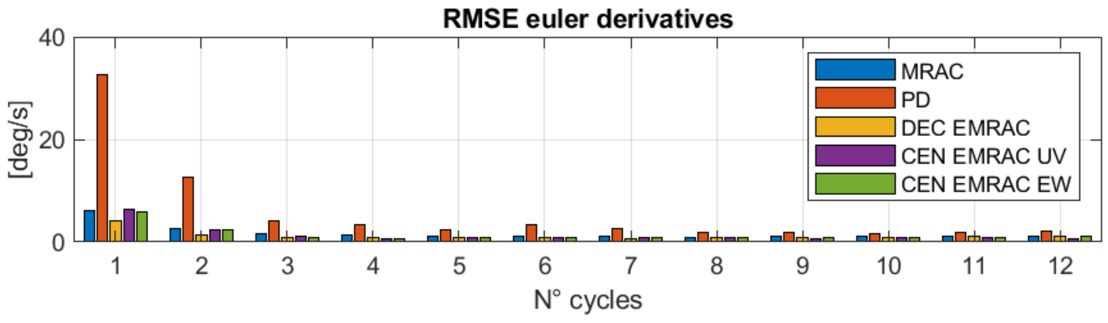


Figure 6.7: 20%:RMSE euler derivatives

6.4.4 +30 % mass and inertia

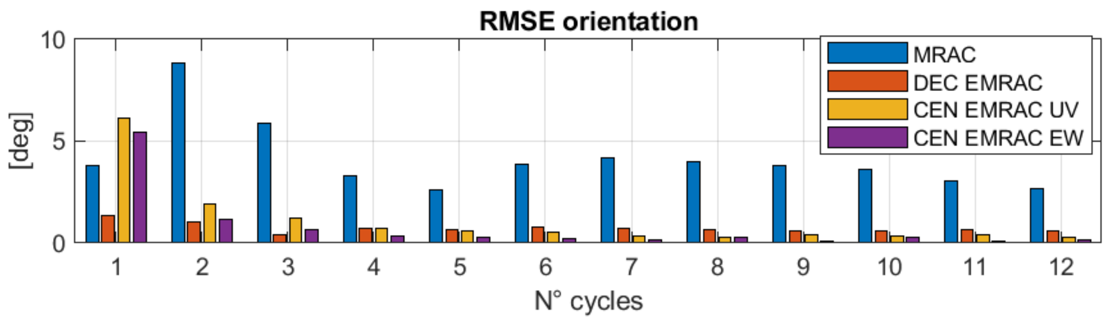


Figure 6.8: 30%:RMSE orientation

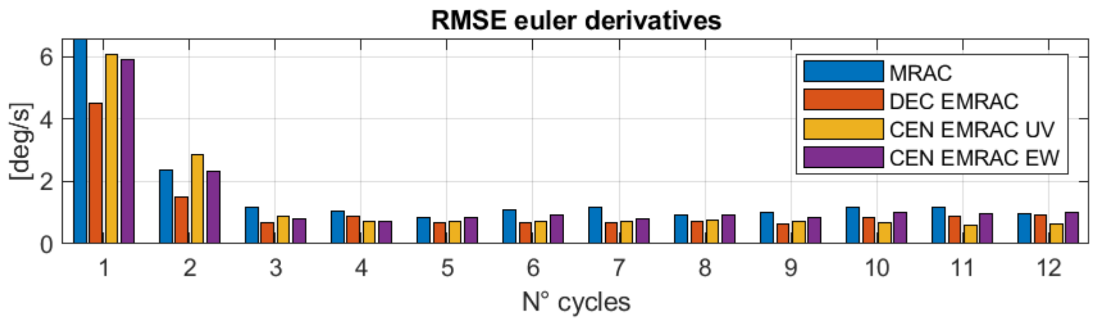


Figure 6.9: 30%:RMSE euler derivatives

6.4.5 +50 % mass and inertia

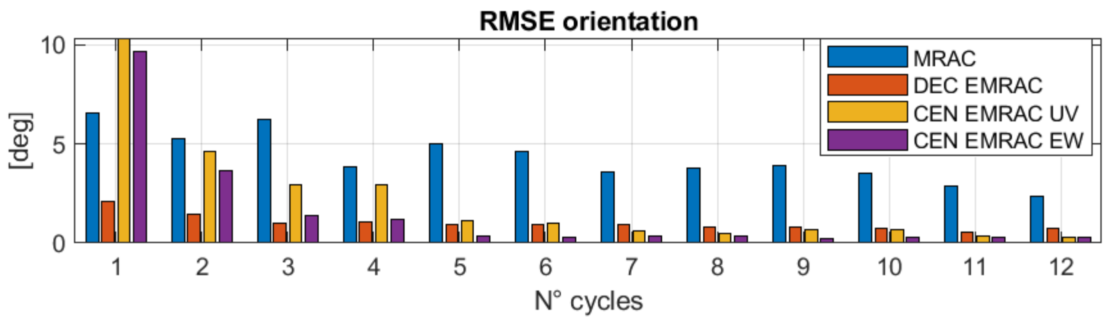


Figure 6.10: 50%:RMSE orientation

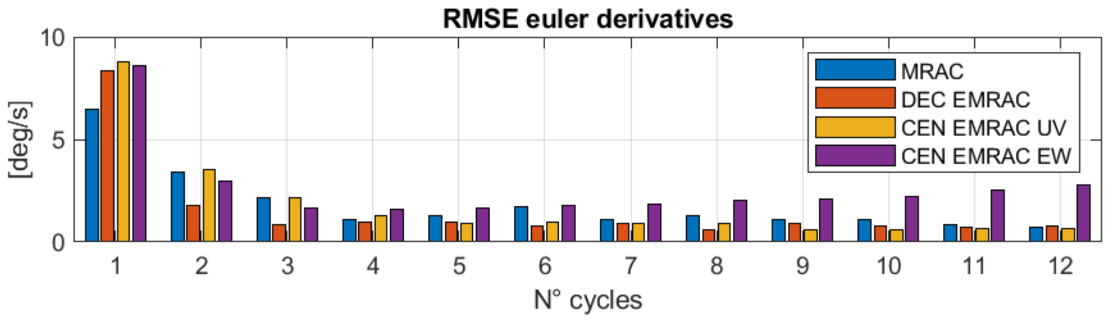


Figure 6.11: 50%:RMSE euler derivatives

6.5 Analysis

All the controllers have been tested keeping the same tuning of the outer loop for consistency during comparison. Analysing the previous histograms, it can be stated that:

- In nominal mass condition, EMRAC controller achives smaller orientation error with respect to PD controller;
- Increasing mass and inertia values, EMRAC controllers are able to adapt and achieve even smaller attitude errors with respect to the PD controller;
- EMRAC and MRAC sustain a larger mass increment with respect to the PD (50% vs 20% mass increment limit, respectively);
- EMRAC controllers achieve significantly smaller attitude error compared to MRAC using the same tuning for α and β, showing the beneficial effect of the enhanced control actions (integrative and switching);

Chapter 7

Conclusion and Future work

From the results obtained, it can be stated that EMRAC has a smaller orientation error with respect to the benchmark controllers. The adaptive controllers have better performance in not nominal condition.The PD stops working at 20% mass and inertia increment, while the adaptive algorithms perform with a small attitude error till 50% mass and inertia increment.As can be seen from the histograms, the algorithm that performs better is the C-EMRAC-EW.The MRAC has the same tuning of the EMRAC algorithms and it has the worst KPIs.This has been done to better compare the base algorithm with the augmented one. After the 50% increment, the EMRAC stops working due to physical limitation of the rotors.
The code from Simulink has been generated and deployed on an actual flight control board, and the tests were performed on a realistic simulator(jMAVsim).
The following can be stated:

- Successful Hardware-in-the-Loop implementation and deployment of EMRAC on Pixhawk 6x;
- All quadrotor simulations are carried on jMAVsim simulator considering sensor feedback noise;

This thesis project can be continued with further improvements as:

- Design adaptive solutions for the outer-loop controller;
- Augment the inner loop with a Neural Network control actions;
- Deploy the code on an actual drone

Appendix A

MATLAB Code

A.1 Trajectory function

```
function ref= fcn(t,T_start)
radius=8;
freq=2*pi*0.1;

if t<T_start
    x=0;
    y=0;
    z=-1;
    dx=0;
    dy=0;
    dz=0;
    ddx=0;
    ddy=0;
    ddz=0;
else
t=t-T_start;

% Position
x=radius*sin(freq*t);
y=radius*cos(freq*t)-radius;
z=-0.1*t-1;

% Velocities
dx=freq*radius*cos(freq*t);
dy=-freq*radius*sin(freq*t);
dz=-0.1;

%Acceleration
ddx=-freq^2*radius*sin(freq*t);
ddy=-freq^2*radius*cos(freq*t);
ddz=0;
end

ref=[x,y,z,dx,dy,dz,ddx,ddy,ddz];
end
```

A.2 Position control along z

[13]

```
\label{}
function tau_Thrust = fcn(z,vz,angle,ref,Q,kz_p,kz_d)

desz=ref(3);
vzdes=ref(6);
zdotdotdes=ref(9);
% Drone parameters
m=0.8;
g=9.81;
k1=0.01;
Tmax=4;
T=4*Tmax;

roll=angle(3);
pitch=angle(2);

%robust action
rho_z=0.01;
xi=[z-desz,vz-vzdes]';%error position and velocity vector
D=[0;1];
zeta=D'*Q*xi;
eps=0.001;
if abs(zeta)>=eps
w=(rho_z/abs(zeta))*zeta;
else
w=(rho_z/eps)*zeta;
end

% Control on Z axis
Vz=-kz_d*(vz-vzdes)-kz_p*(z-desz)+zdotdotdes+w;


u1=(m*(g-Vz)-k1*vz*abs(vz))/(cos(roll)*cos(pitch));
tau_Thrust=(u1/T)+1;
end
```

A.3 Position control on x and y

[13]

```
function [des_roll,des_pitch] = fcn(x,y,vx,vy,yaw,ref,Q,kxy_p,kxy_d,tau_Thrust)

% Reference position
desx=ref(1);
desy=ref(2);
% Reference velocities
desvx=ref(4);
desvy=ref(5);
% Reference Acceleration
desxdotdot=ref(7);
desydotdot=ref(8);
% Drone parameters
```

```
m=0.8; %kg
g=9.81; %m/s^2
k1=0.01;
Tmax=4; %N
T=4*Tmax;
u1=T*(tau_Thrust-1);

% x and y control
F=(-u1/m)*[sin(yaw),cos(yaw);-cos(yaw),sin(yaw)];
G=(k1/m)*[vx*abs(vx);vy*abs(vy)];

% Robust control action
rho_xy=0.01;
xi=[x-desy,y-desy,vx-desvx,vy-desvy]';%error position and velocity vector
D=[zeros(2);eye(2)];
zeta=D'*Q*xi;
w=(rho_xy/abs(zeta))*zeta;
% eps=0.001;
% if abs(zeta)>=eps
%
% else
% w=(rho_xy/eps)*zeta;
% end


%control law
Vxy=-kxy_d*[vx-desvx;vy-desvy]-kxy_p*[x-desx;y-desy]+[desxdotdot;desydotdot]+w;
out=F\(Vxy+G);  % inv(F)*(Vxy+G) is less accurate
des_roll=out(1);
des_pitch=out(2);
end
```

A.4 Setup controllers script

```
    close all
clear all
clc

%
SampleTime=0.01;
T_start=100;
Tf=60+T_start;
% Helix parameters
radius=8;
freq=2*pi*0.1;

% OUTER
%% Z control

%PD action
%eigenvalues
pz1=-9;
pz2=-8.8;
kz_p=pz1*pz2;
kz_d=-(pz1+pz2);
% Robust control action
P_z=eye(2);
```

```
H_hat_z=[0,1;-kz_p,-kz_d];
Q_z=lyap(H_hat_z,P_z); %solution of the lyapunov eq.

%% X and Y control
% PD eigenvalues
p1xy=-1.1;
p2xy=-1;
kxy_p=p1xy*p2xy;
kxy_d=-(p1xy+p2xy);
%robust control xy
P_xy=eye(4);
H_hat_xy=[zeros(2),eye(2);-kxy_p*eye(2),-kxy_d*eye(2)];
Q_xy=lyap(H_hat_xy,P_xy); %solution of the lyapunov eq.

%% CONTROLLERS
k=input('Controller to be run, type: \n 1->PD \n 2->CEN_EMRAC \n 3->DEC_EMRAC \n
    4->MRAC BASE \n');
if(k~=1 && k~=2 && k~=3 && k~=4 && k~=5)
disp('ERROR: Controller not valid')
return
end

switch k
    case 1  % PD
    Kp=0.08*diag([1 1 10]);
    Kd=0.01*diag([1 1 10]);
    disp('PD load')

    case 2 %CEN_EMRAC
    EMRAC_CEN_setup_lqr
    disp('CEN_EMRAC loaded')

    case 3 %DEC_EMRAC
    DEC_EMRAC_PHI_setup_LQR
    DEC_EMRAC_THETA_setup_LQR
    DEC_EMRAC_PSI_setup_LQR
    disp('DEC_EMRAC loaded')

    case 4 % MRAC base
    MRAC_base_lqr
    disp('MRACbase loaded')

end
```

A.5 Setup EMRAC script

A.5.1 Centralized EMRAC

```
n_x=6;
n_u=3;
n_d=3;

EMRAC.P_phi=1;
EMRAC.S=eye(3);
%% LQR reference model
n=3;
np = 3;
```

```
lp = 2;
A = [zeros(3*(2-1),3*(2-1)), eye(3*(2-1)); zeros(3*(2-1), 3*2)];
B = [zeros(3*(2-1),3*(2-1)); eye(3*(2-1))];

%LQR reference model

Q_lqr = 1000*diag([20 20 20 0.01 0.01 0.01]);
R_lqr = 1*eye(n);
Kbl = lqr(A,B,Q_lqr,R_lqr);
EMRAC.A_m = A-B*Kbl;
EMRAC.B_m = B;
EMRAC.E_m = zeros(np*lp,n);

%% LQR feedforward
Q_fw=1e-3*diag([1 1 1 1 1 1]);
R_fw=1000000*diag([1 1 1]);
F = lqr(A,B,Q_fw,R_fw);
disp('loaded')

%% adaptive weights

EMRAC.alpha_X     = 0.1*diag([1 1 2 2 10 10]); %nx*nx dimension positive diagonal
EMRAC.alpha_R     = 0.01*diag([1 1 10]); %nu*nu dimension positive diagonal1

EMRAC.alpha_D     = 10*diag([1 1 1]); %nd*nd dimension positive diagonal
EMRAC.alpha_I     = 0.1*diag([1 1 2 2 20 20]); %nx*nx dimension positive diagonal
EMRAC.alpha_N     = 100000; %positive constant
EMRAC.alpha_N_ew    = [2 5 5]; %positive constant

%beta
EMRAC.beta_X      = 0.1*EMRAC.alpha_X; %positive
EMRAC.beta_R      = 0.1*EMRAC.alpha_R; %positive

EMRAC.beta_D      = 0.1*EMRAC.alpha_D; %positive
EMRAC.beta_I      = 0.1*EMRAC.alpha_I; %positive
disp('loaded')
%% sigma modification parameters
EMRAC.sigma_active=1; %activate sigma modification parameters

if EMRAC.sigma_active == 0
    disp('Sigma Modification OFF')
else
    disp('Sigma Modification ON')
end

EMRAC.rho_e       = 10e-6*diag([ones(n_x,1)]); %nx*nx dimension positive diagonal
EMRAC.rho_X       = 10e-6*diag([ones(n_x,1)]); %nx*nx dimension positive diagonal
EMRAC.rho_R       = 10e-6*diag([ones(n_u,1)]); %nu*nu dimension positive diagonal
EMRAC.rho_D       = 10e-6*diag([ones(n_d,1)]); %nu*nu dimension positive diagonal
EMRAC.rho_I       = 10e-6*diag([ones(n_x,1)]); %nx*nx dimension positive diagonal
EMRAC.rho_N       = 0.1; %positive constant
EMRAC.rho_N_ew      =1e-3*[1 1 1]; %positive constant

EMRAC.GAMMA_alpha = diag([diag(EMRAC.alpha_X); diag(EMRAC.alpha_R); ...
    diag(EMRAC.alpha_D); diag(EMRAC.alpha_I)]);

EMRAC.GAMMA_rho    = diag([diag(EMRAC.rho_X); diag(EMRAC.rho_R); ...
    diag(EMRAC.rho_D); diag(EMRAC.rho_I)]);



%%
```

```
%y_e gain
EMRAC.Q             = diag([ones(n_x,1)]);        %nx*nx dimension positive
EMRAC.P_e           = lyap(EMRAC.A_m',EMRAC.Q);   %gain of y_e

EMRAC.K_x_0=[0.1*eye(3) 0.01*eye(3)];
EMRAC.K_r_0=0.1*eye(3);


EMRAC.M_phi             = 0.0008;
EMRAC.M_phi_hat    = sqrt(max(eig(kron(EMRAC.GAMMA_rho*inv(EMRAC.GAMMA_alpha),inv(
    EMRAC.P_phi)))...
     /min(eig(kron(EMRAC.GAMMA_rho*inv(EMRAC.GAMMA_alpha),inv(EMRAC.P_phi)))))*
    EMRAC.M_phi;
EMRAC.M_I_hat     = 4;                       %positive constant
EMRAC.M_N_hat     = 0.15;                       %positive constant

EMRAC.M_N_hat_ew=6e-4*[1 1 1];

EMRAC.eta_I         = 1;                          %positive constant
EMRAC.eta_phi       = 1;
EMRAC.eta_N         = 1;                          %positive constant
EMRAC.eta_N_ew=[1e-3 1e-3 1e-3];

EMRAC.Omega         = diag([1 1 1]); %nu*nu dimension positive diagonal

%h(y_e) calculation uv
EMRAC.sigma_0       =  2;
EMRAC.gamma_0         = 1e-5;
EMRAC.xi_0          = 1;
%hi(y_e) calculation ew
EMRAC.sigma_ew      =  2*[1 1 1];
EMRAC.gamma_ew         = 1e-5*[1 1 1];
EMRAC.xi_ew           = 1*[1 1 1];

%sign y_e
EMRAC.epsilon      = 5;
```

A.5.2 Decentralized EMRAC

PHI EMRAC

```
n_x=2;
n_u=1;
n_d=2;

PHI_EMRAC.P_phi=1;
PHI_EMRAC.S=1;
Ts=SampleTime;

n=1;
np =1;
lp = 2;
PHI_EMRAC.A = [zeros(np*(lp-1),np*(lp-1)), eye(np*(lp-1)); zeros(np*(lp-1), np*lp)
    ];
PHI_EMRAC.B = [zeros(np*(lp-1),np*(lp-1)); eye(np*(lp-1))];

%LQR

```

```matlab
PHI_EMRAC.Q_lqr =1e0*diag([2e3 1e1]);
PHI_EMRAC.R_lqr = 1;

PHI_EMRAC.Kbl = lqr(PHI_EMRAC.A,PHI_EMRAC.B,PHI_EMRAC.Q_lqr,PHI_EMRAC.R_lqr);

PHI_EMRAC.A_m = PHI_EMRAC.A-PHI_EMRAC.B*PHI_EMRAC.Kbl;
PHI_EMRAC.B_m = PHI_EMRAC.B;
PHI_EMRAC.E_m = zeros(np*lp,n+1);

%% adaptive weights

PHI_EMRAC.alpha_X     = 0.01*diag([ones(n_x,1)]); %nx*nx dimension positive diagonal
PHI_EMRAC.alpha_R     = 0.01*diag([ones(n_u,1)]); %nu*nu dimension positive diagonal
PHI_EMRAC.alpha_D     = 11*diag([ones(2,1)]); %nd*nd dimension positive diagonal
PHI_EMRAC.alpha_I     = 2*diag([ones(n_x,1)]); %nx*nx dimension positive diagonal
PHI_EMRAC.alpha_N     = 10000000; %positive constant

PHI_EMRAC.beta_X      = 0.1*PHI_EMRAC.alpha_X; %positive
PHI_EMRAC.beta_R      = 0.1*PHI_EMRAC.alpha_R; %positive
PHI_EMRAC.beta_D      = 0.1*PHI_EMRAC.alpha_D; %positive
PHI_EMRAC.beta_I      = 0.1*PHI_EMRAC.alpha_I; %positive

%% sigma modification parameters
EMRAC.sigma_active=1; %activate sigma modification parameters

if EMRAC.sigma_active == 0
    disp('Sigma Modification OFF PHI')
else
    disp('Sigma Modification ON PHI')
end

PHI_EMRAC.rho_e       = 1e-6*diag([ones(n_x,1)]); %nx*nx dimension positive diagonal
PHI_EMRAC.rho_X       = 1e-6*diag([ones(n_x,1)]); %nx*nx dimension positive diagonal
PHI_EMRAC.rho_R       = 1e-6*diag([ones(n_u,1)]); %nu*nu dimension positive diagonal
PHI_EMRAC.rho_D       = 1e-6*diag([ones(2,1)]); %nu*nu dimension positive diagonal
PHI_EMRAC.rho_I       = 1e-6*diag([ones(n_x,1)]); %nx*nx dimension positive diagonal
PHI_EMRAC.rho_N      = 1e-6; %positive constant

%y_e gain
PHI_EMRAC.Q           = diag([1 1]);        %nx*nx dimension positive
PHI_EMRAC.P_e         = lyap(PHI_EMRAC.A_m',PHI_EMRAC.Q);  %gain of y_e



PHI_EMRAC.GAMMA_alpha = diag([diag(PHI_EMRAC.alpha_X); diag(PHI_EMRAC.alpha_R); ...
    diag(PHI_EMRAC.alpha_D); diag(PHI_EMRAC.alpha_I)]);
PHI_EMRAC.GAMMA_rho   = diag([diag(PHI_EMRAC.rho_X); diag(PHI_EMRAC.rho_R); ...
    diag(PHI_EMRAC.rho_D); diag(PHI_EMRAC.rho_I)]);

PHI_EMRAC.M           = 0.0002;
PHI_EMRAC.M_phi_hat   = sqrt(max(eig(kron(PHI_EMRAC.GAMMA_rho*inv(PHI_EMRAC.GAMMA_alpha),inv(PHI_EMRAC.P_phi))))...
     /min(eig(kron(PHI_EMRAC.GAMMA_rho*inv(PHI_EMRAC.GAMMA_alpha),inv(PHI_EMRAC.P_phi)))))*PHI_EMRAC.M;

```

```


PHI_EMRAC.M_I_hat    = 1;  %positive constant
PHI_EMRAC.M_N_hat    = 0.45; %positive constant

PHI_EMRAC.eta_I        = 1; %positive constant sigma mod for integral action xIdot
PHI_EMRAC.eta_phi      = 1;
PHI_EMRAC.eta_N      = 1;     %positive constant

PHI_EMRAC.Omega        = diag([ones(n_u,1)]); %nu*nu dimension positive diagonal

%h(y_e) calculation
PHI_EMRAC.sigma_0      =  2;
PHI_EMRAC.gamma_0        = 1e-5;
PHI_EMRAC.xi_0          = 1;

%sign y_e
PHI_EMRAC.epsilon      = 5;
```

THETA EMRAC

```
n_x=2;
n_u=1;
n_d=2;

THETA_EMRAC.P_phi=1;
THETA_EMRAC.S=1;

n=1;
np =1;
lp = 2;
THETA_EMRAC.A = [zeros(np*(lp-1),np*(lp-1)), eye(np*(lp-1)); zeros(np*(lp-1), np*
      lp)];
THETA_EMRAC.B = [zeros(np*(lp-1),np*(lp-1)); eye(np*(lp-1))];

%LQR

THETA_EMRAC.Q_lqr = 1e0*diag([2e3 1e1]);
THETA_EMRAC.R_lqr = 1*eye(n);

THETA_EMRAC.Kbl = lqr(THETA_EMRAC.A,THETA_EMRAC.B,THETA_EMRAC.Q_lqr,THETA_EMRAC.
      R_lqr);

THETA_EMRAC.A_m = THETA_EMRAC.A-THETA_EMRAC.B*THETA_EMRAC.Kbl;
THETA_EMRAC.B_m = THETA_EMRAC.B;
THETA_EMRAC.E_m = zeros(np*lp,n+1);


%% adaptive weights

THETA_EMRAC.alpha_X      = 0.01*diag([ones(n_x,1)]); %nx*nx dimension positive
      diagonal
THETA_EMRAC.alpha_R      = 0.01*diag([ones(n_u,1)]); %nu*nu dimension positive
      diagonal
THETA_EMRAC.alpha_D      = 11*diag([ones(n_d,1)]); %nd*nd dimension positive
      diagonal
THETA_EMRAC.alpha_I      = 2*diag([ones(n_x,1)]); %nx*nx dimension positive diagonal
THETA_EMRAC.alpha_N      = 10000000; %positive constant

THETA_EMRAC.beta_X       = 0.1*THETA_EMRAC.alpha_X; %positive
THETA_EMRAC.beta_R       = 0.1*THETA_EMRAC.alpha_R; %positive
THETA_EMRAC.beta_D       = 0.1*THETA_EMRAC.alpha_D; %positive
THETA_EMRAC.beta_I       = 0.1*THETA_EMRAC.alpha_I; %positive
%%
%sigma modification parameters
EMRAC.sigma_active=1; %activate sigma modification parameters

if EMRAC.sigma_active == 0
    disp('Sigma Modification OFF THETA')
else
    disp('Sigma Modification ON THETA')
end

THETA_EMRAC.rho_e        = 1e-6*diag([ones(n_x,1)]); %nx*nx dimension positive
    diagonal
THETA_EMRAC.rho_X        = 1e-6*diag([ones(n_x,1)]); %nx*nx dimension positive
    diagonal
THETA_EMRAC.rho_R        = 1e-6*diag([ones(n_u,1)]); %nu*nu dimension positive
    diagonal
THETA_EMRAC.rho_D        = 1e-6*diag([ones(n_d,1)]); %nu*nu dimension positive
    diagonal
THETA_EMRAC.rho_I        = 1e-6*diag([ones(n_x,1)]); %nx*nx dimension positive
    diagonal
THETA_EMRAC.rho_N       = 1e-6; %positive constant

%y_e gain
THETA_EMRAC.Q            = 1*diag([ones(n_x,1)]);        %nx*nx dimension positive
THETA_EMRAC.P_e          = lyap(THETA_EMRAC.A_m',THETA_EMRAC.Q);  %gain of y_e



THETA_EMRAC.GAMMA_alpha = diag([diag(THETA_EMRAC.alpha_X); diag(THETA_EMRAC.
    alpha_R);  ...
    diag(THETA_EMRAC.alpha_D); diag(THETA_EMRAC.alpha_I)]);
THETA_EMRAC.GAMMA_rho    = diag([diag(THETA_EMRAC.rho_X); diag(THETA_EMRAC.rho_R);
    ...
    diag(THETA_EMRAC.rho_D); diag(THETA_EMRAC.rho_I)]);

THETA_EMRAC.M            = 0.0002;
THETA_EMRAC.M_phi_hat    = sqrt(max(eig(kron(THETA_EMRAC.GAMMA_rho*inv(THETA_EMRAC.
    GAMMA_alpha),inv(THETA_EMRAC.P_phi))))...
    /min(eig(kron(THETA_EMRAC.GAMMA_rho*inv(THETA_EMRAC.GAMMA_alpha),inv(
    THETA_EMRAC.P_phi)))))*THETA_EMRAC.M;


THETA_EMRAC.M_I_hat     = 1;                        %positive constant
THETA_EMRAC.M_N_hat     = 0.4;                        %positive constant

THETA_EMRAC.eta_I        = 1;                            %positive constant
THETA_EMRAC.eta_phi      = 1;
THETA_EMRAC.eta_N       = 1;                            %positive constant

THETA_EMRAC.Omega        = diag([ones(n_u,1)]); %nu*nu dimension positive diagonal

%h(y_e) calculation
THETA_EMRAC.sigma_0      =  2;
THETA_EMRAC.gamma_0        = 1e-5;
THETA_EMRAC.xi_0          = 1;

%sign y_e
```

```
THETA_EMRAC.epsilon     = 5;
```

PSI EMRAC

```
n_x=2;
n_u=1;
n_d=1;

PSI_EMRAC.P_phi=1;
PSI_EMRAC.S=1;


n=1;
np =1;
lp = 2;
PSI_EMRAC.A = [zeros(np*(lp-1),np*(lp-1)), eye(np*(lp-1));zeros(np*(lp-1), np*lp)
    ];
PSI_EMRAC.B = [zeros(np*(lp-1),np*(lp-1)); eye(np*(lp-1))];

%LQR

PSI_EMRAC.Q_lqr = 1e0*diag([2e3 1e1]);
PSI_EMRAC.R_lqr = 1;

PSI_EMRAC.Kbl = lqr(PSI_EMRAC.A,PSI_EMRAC.B,PSI_EMRAC.Q_lqr,PSI_EMRAC.R_lqr);

PSI_EMRAC.A_m = PSI_EMRAC.A-PSI_EMRAC.B*PSI_EMRAC.Kbl;
PSI_EMRAC.B_m = PSI_EMRAC.B;
PSI_EMRAC.E_m = zeros(np*lp,n);

%% adaptive weights

PSI_EMRAC.alpha_X    = 1*diag([ones(n_x,1)]); %nx*nx dimension positive diagonal
PSI_EMRAC.alpha_R    = 0.1*diag([ones(n_u,1)]); %nu*nu dimension positive diagonal
PSI_EMRAC.alpha_D    = 11*diag([ones(n_d,1)]); %nd*nd dimension positive diagonal
PSI_EMRAC.alpha_I    = 3*diag([ones(n_x,1)]); %nx*nx dimension positive diagonal
PSI_EMRAC.alpha_N    = 100000; %positive constant

PSI_EMRAC.beta_X     = 0.1*PSI_EMRAC.alpha_X; %positive
PSI_EMRAC.beta_R     = 0.1*PSI_EMRAC.alpha_R; %positive
PSI_EMRAC.beta_D     = 0.1*PSI_EMRAC.alpha_D; %positive
PSI_EMRAC.beta_I     = 0.1*PSI_EMRAC.alpha_I; %positive

%% sigma modification parameters
EMRAC.sigma_active=1; %activate sigma modification parameters

if EMRAC.sigma_active == 0
    disp('Sigma Modification OFF PSI')
else
    disp('Sigma Modification ON PSI')
end

PSI_EMRAC.rho_e      = 1e-6*diag([ones(n_x,1)]); %nx*nx dimension positive
    diagonal
PSI_EMRAC.rho_X      = 1e-6*diag([ones(n_x,1)]); %nx*nx dimension positive
    diagonal
PSI_EMRAC.rho_R      = 1e-6*diag([ones(n_u,1)]); %nu*nu dimension positive
    diagonal
```

```
PSI_EMRAC.rho_D       = 1e-6*diag([ones(n_d,1)]); %nu*nu dimension positive
     diagonal
PSI_EMRAC.rho_I       = 1e-3*diag([ones(n_x,1)]); %nx*nx dimension positive
     diagonal
PSI_EMRAC.rho_N       = 1e-6; %positive constant

%y_e gain
PSI_EMRAC.Q           = 5*diag([ones(n_x,1)]);        %nx*nx dimension positive
PSI_EMRAC.P_e         = lyap(PSI_EMRAC.A_m',PSI_EMRAC.Q);  %gain of y_e



PSI_EMRAC.GAMMA_alpha = diag([diag(PSI_EMRAC.alpha_X); diag(PSI_EMRAC.alpha_R);
      ...
     diag(PSI_EMRAC.alpha_D); diag(PSI_EMRAC.alpha_I)]);
PSI_EMRAC.GAMMA_rho    = diag([diag(PSI_EMRAC.rho_X); diag(PSI_EMRAC.rho_R); ...
     diag(PSI_EMRAC.rho_D); diag(PSI_EMRAC.rho_I)]);

PSI_EMRAC.M             = 0.005;
PSI_EMRAC.M_phi_hat    = sqrt(max(eig(kron(PSI_EMRAC.GAMMA_rho*inv(PSI_EMRAC.
     GAMMA_alpha),inv(PSI_EMRAC.P_phi))))/min(eig(kron(PSI_EMRAC.GAMMA_rho*inv(
     PSI_EMRAC.GAMMA_alpha),inv(PSI_EMRAC.P_phi)))))*PSI_EMRAC.M;


PSI_EMRAC.M_I_hat    = 0.2;                          %positive constant
PSI_EMRAC.M_N_hat    = 20;                         %positive constant

PSI_EMRAC.eta_I        = 1;                          %positive constant
PSI_EMRAC.eta_phi      = 1;
PSI_EMRAC.eta_N      = 1;                          %positive constant

PSI_EMRAC.Omega        = diag([ones(n_u,1)]); %nu*nu dimension positive diagonal

%h(y_e) calculation
PSI_EMRAC.sigma_0      =  2;
PSI_EMRAC.gamma_0      = 1e-5;
PSI_EMRAC.xi_0         = 1;

%sign y_e
PSI_EMRAC.epsilon      = 5;

```

Appendix B

KPIs results

B.1 Nominal conditions

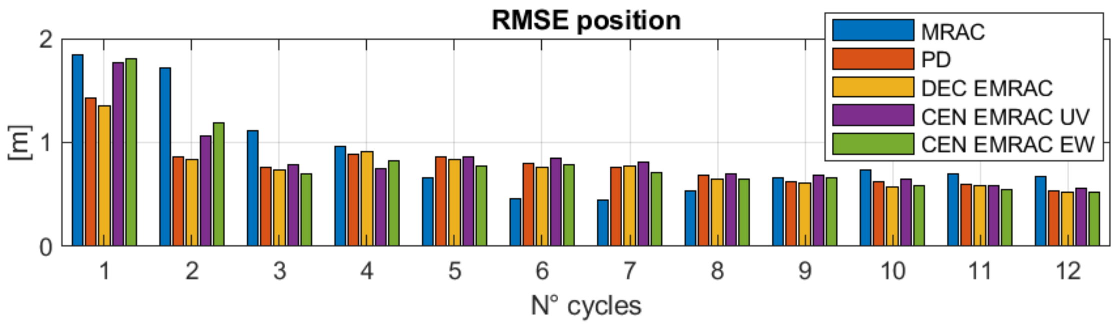


Figure B.1: RMSE position nominal condition

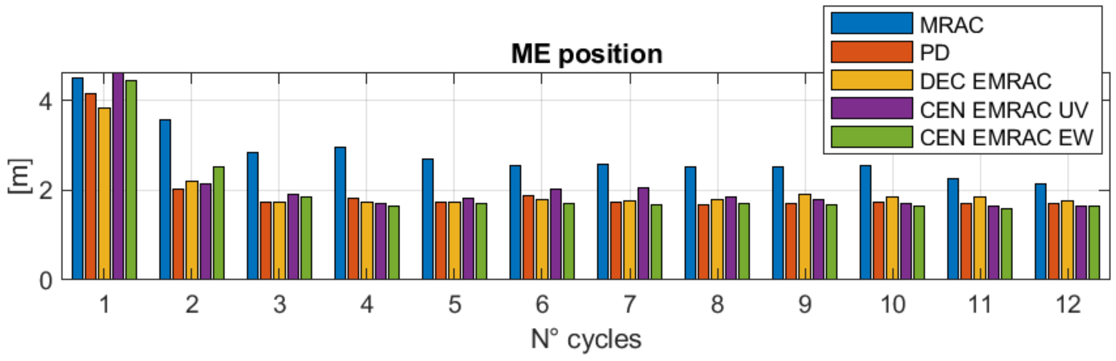


Figure B.2: ME position nominal condition

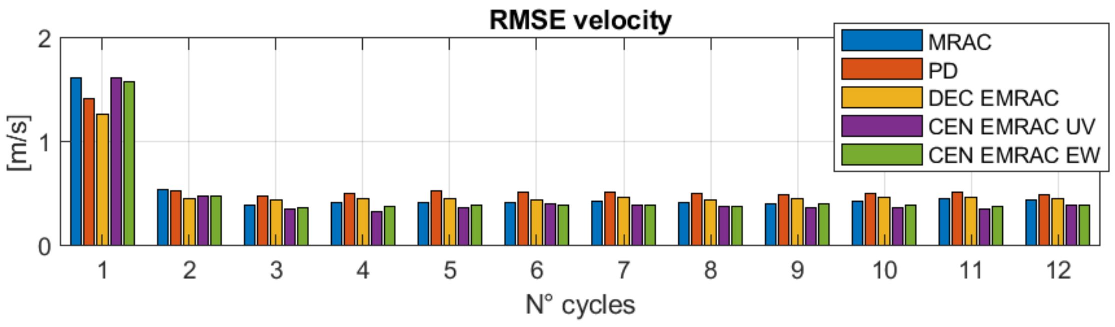


Figure B.3: RMSE velocities nominal condition

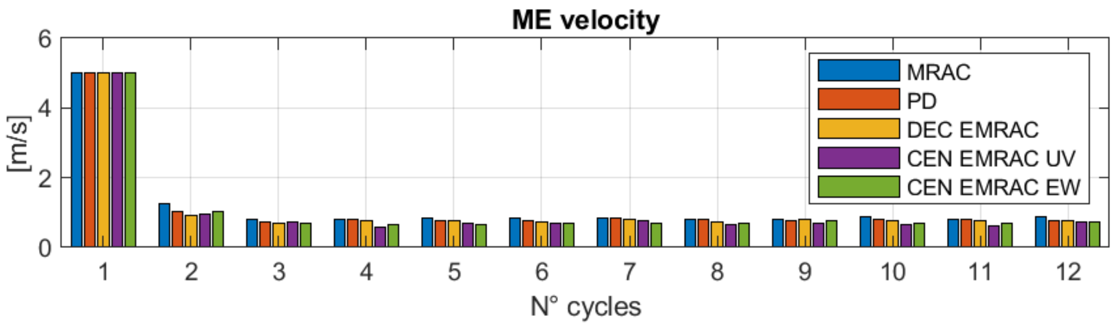


Figure B.4: ME velocities nominal condition

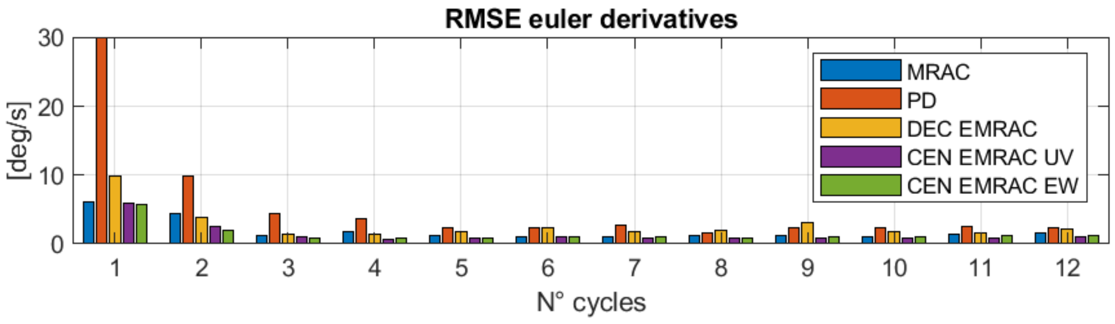


Figure B.5: RMSE euler derivatives nominal condition

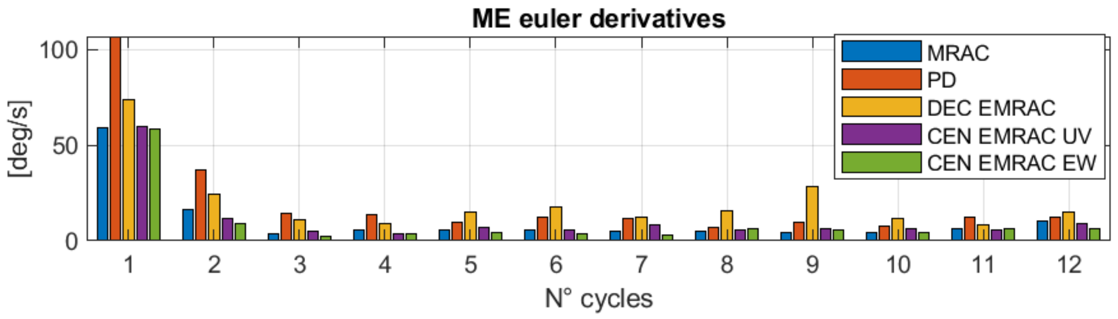


Figure B.6: ME euler derivatives nominal condition

B.2 10% mass and inertia increament

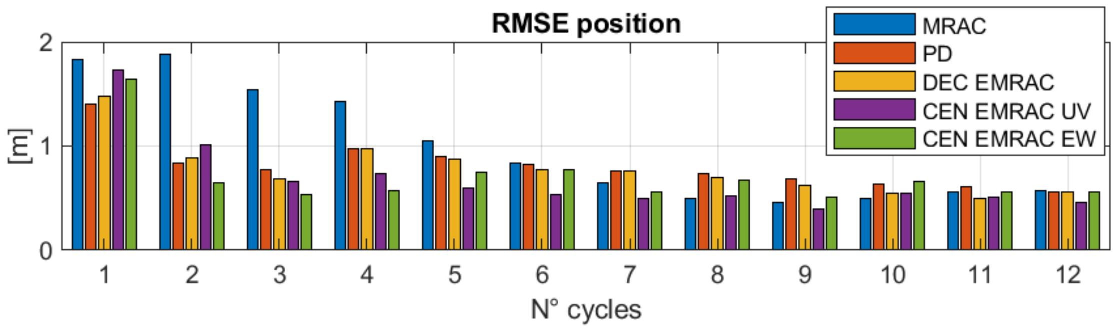


Figure B.7: RMSE position +10%

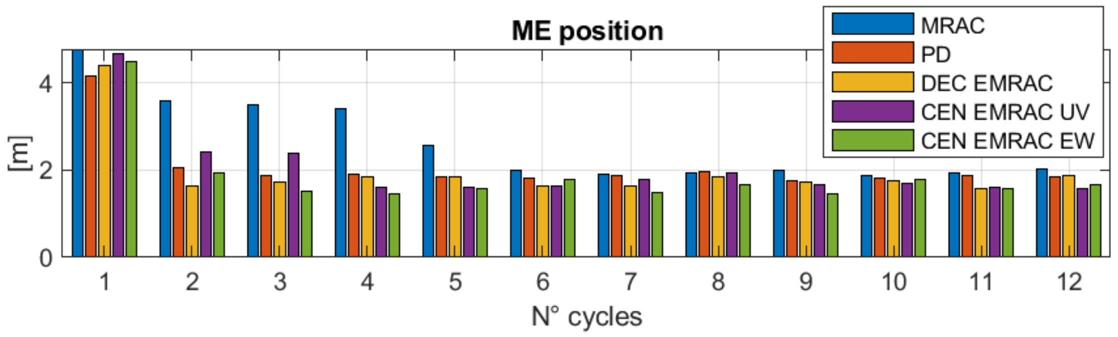


Figure B.8: ME position +10%

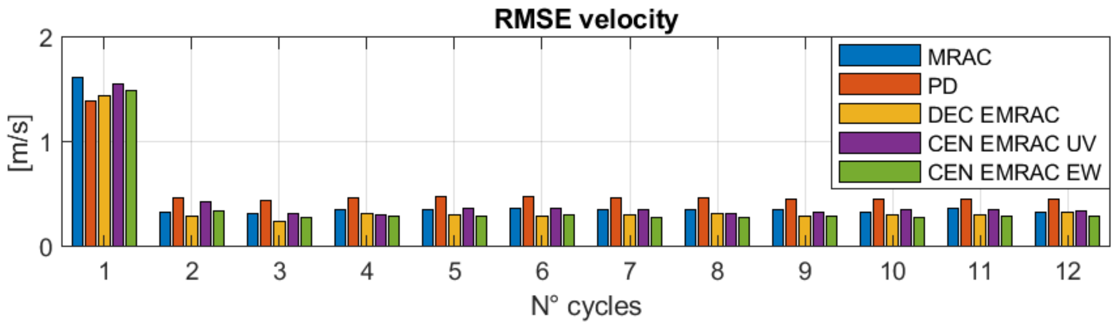


Figure B.9: RMSE velocities +10%

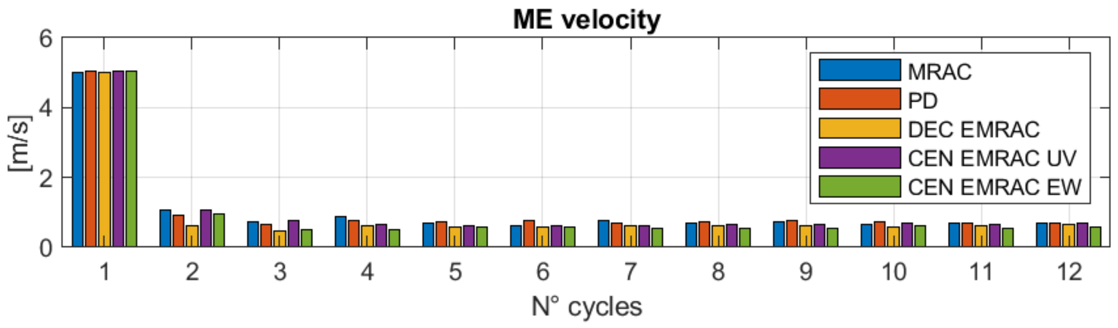


Figure B.10: ME velocities +10%

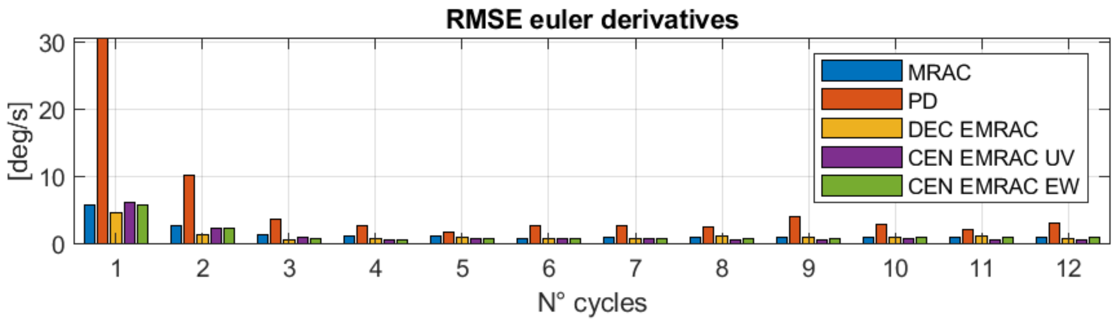


Figure B.11: RMSE euler derivatives +10%

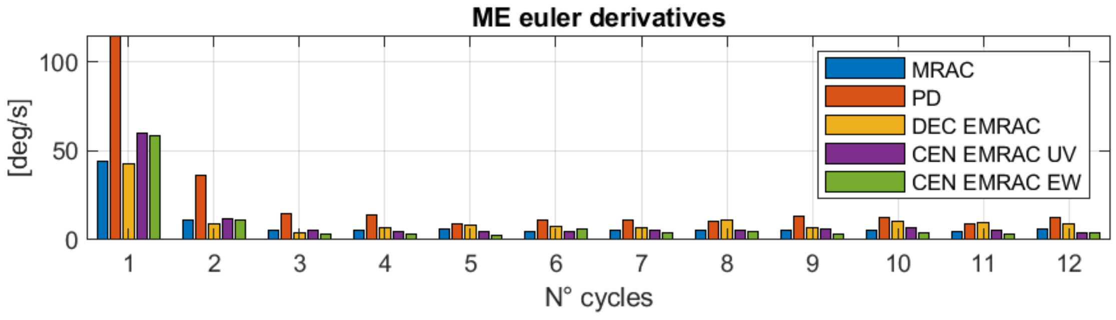


Figure B.12: ME euler derivatives +10%

B.3 20% mass and inertia increment

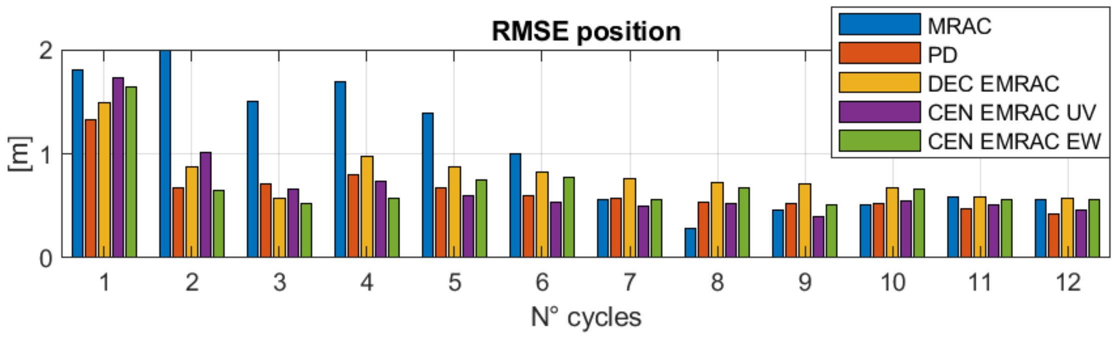


Figure B.13: RMSE position +20%

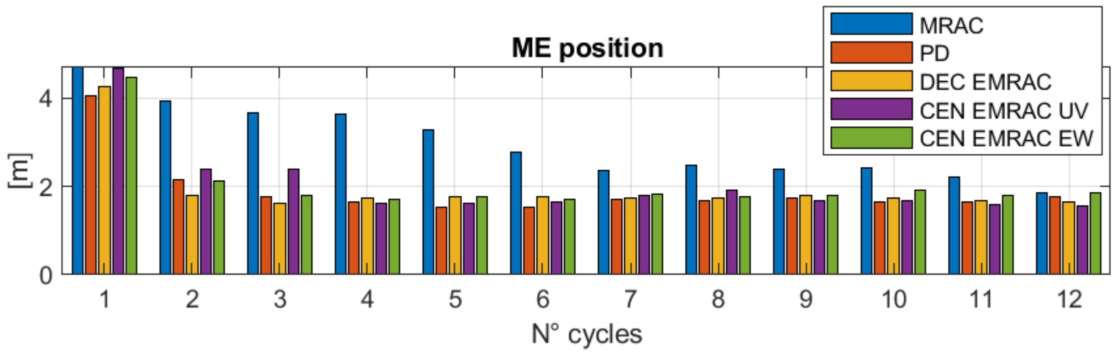


Figure B.14: ME position +20%

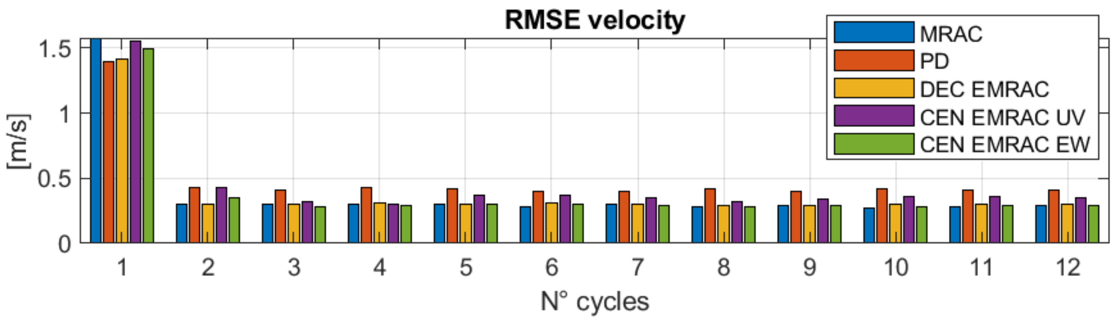


Figure B.15: RMSE velocities +20%

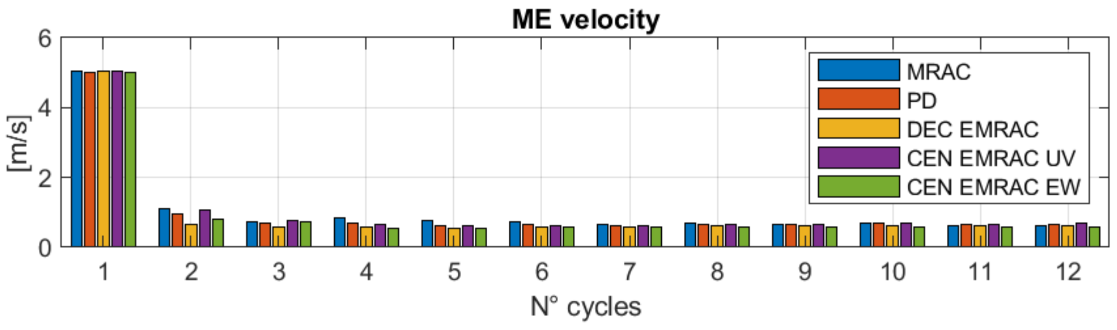


Figure B.16: ME velocities +20%

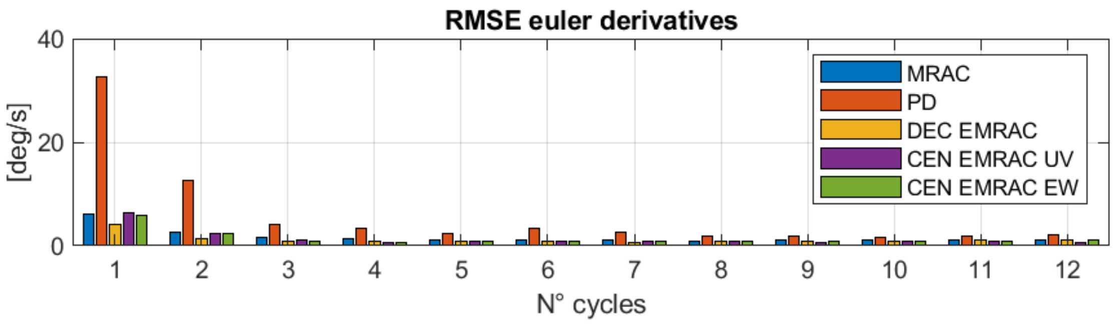


Figure B.17: RMSE euler derivatives +20%

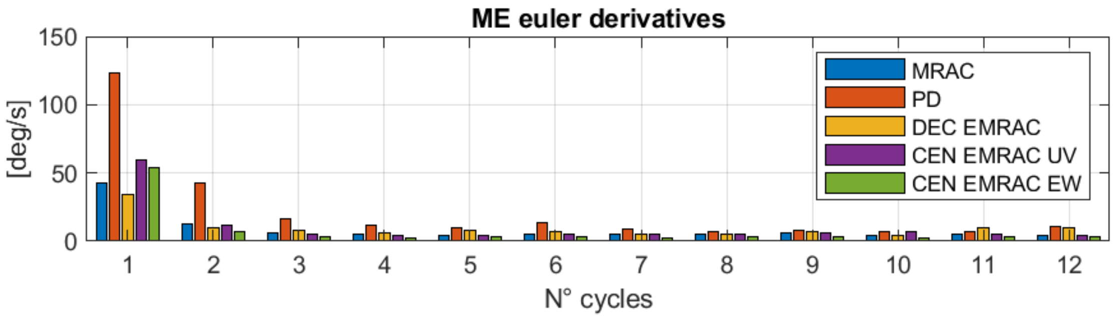


Figure B.18: ME euler derivatives +20%

B.4 30% mass and inertia increment

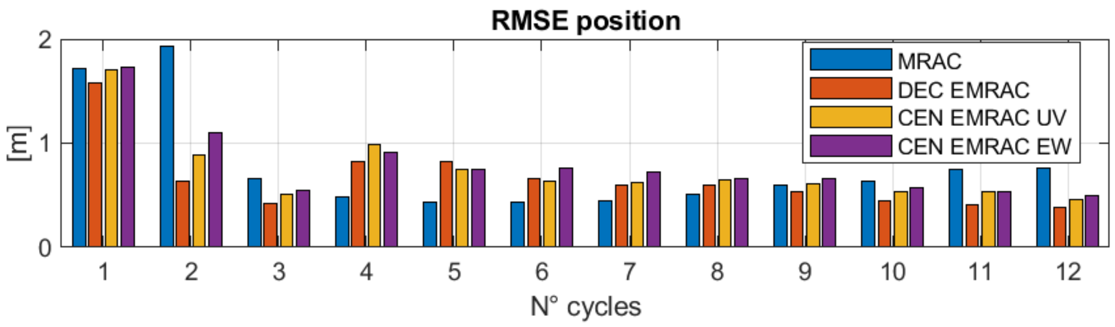


Figure B.19: RMSE position

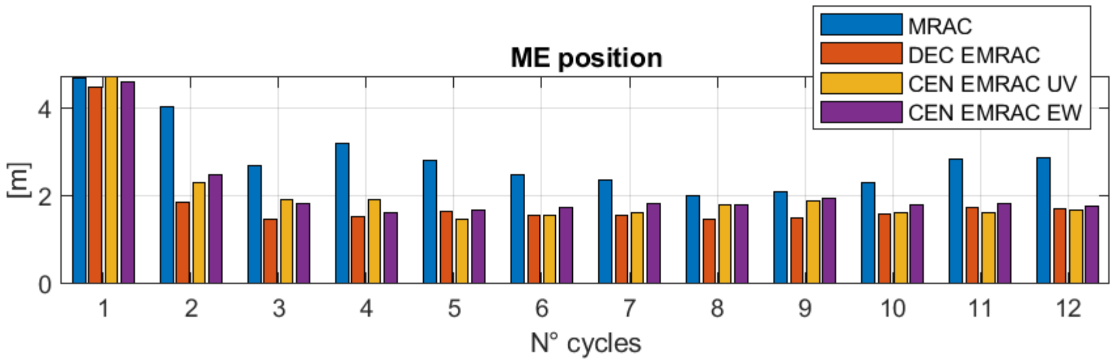


Figure B.20: ME position 30%

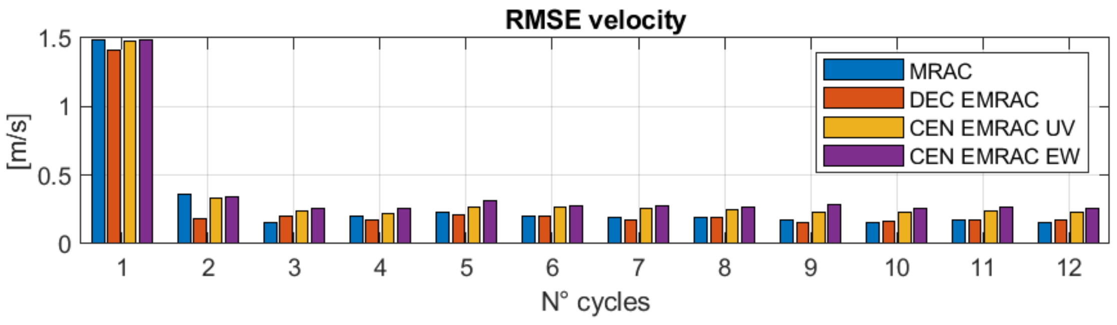


Figure B.21: RMSE velocities 30%

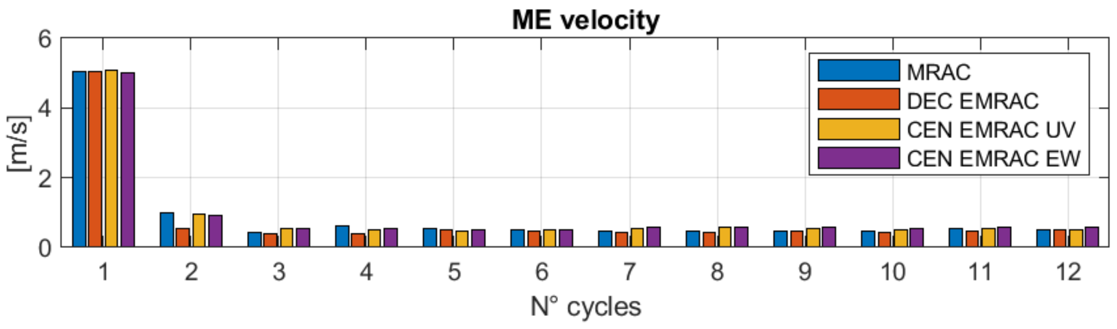


Figure B.22: ME velocities 30%

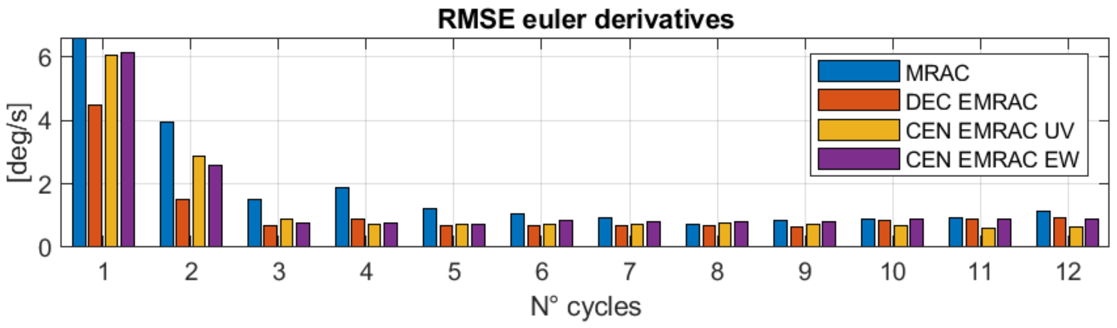


Figure B.23: RMSE euler derivatives 30%

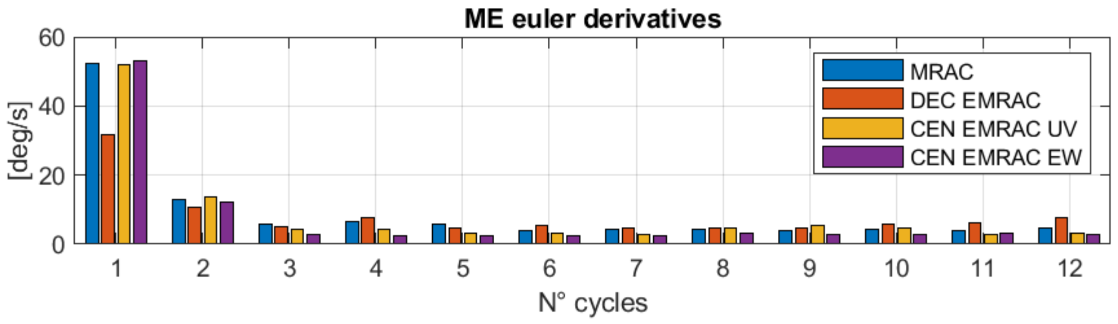


Figure B.24: ME euler derivatives 30%

B.5 50% mass and inertia increment

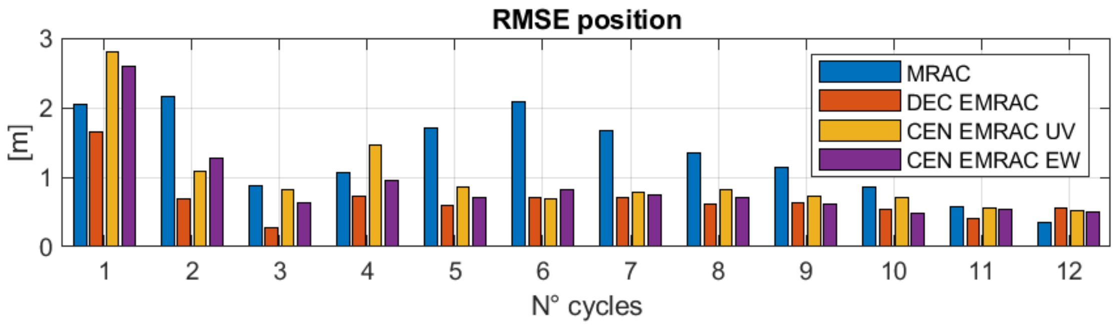


Figure B.25: RMSE position 50%

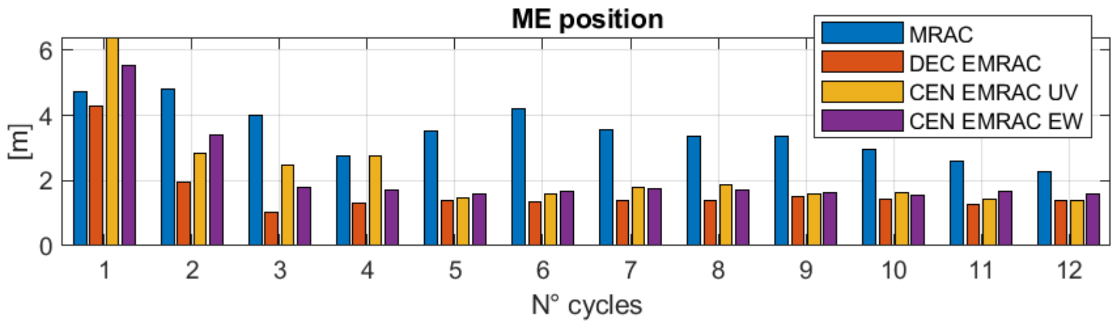


Figure B.26: ME position 50%

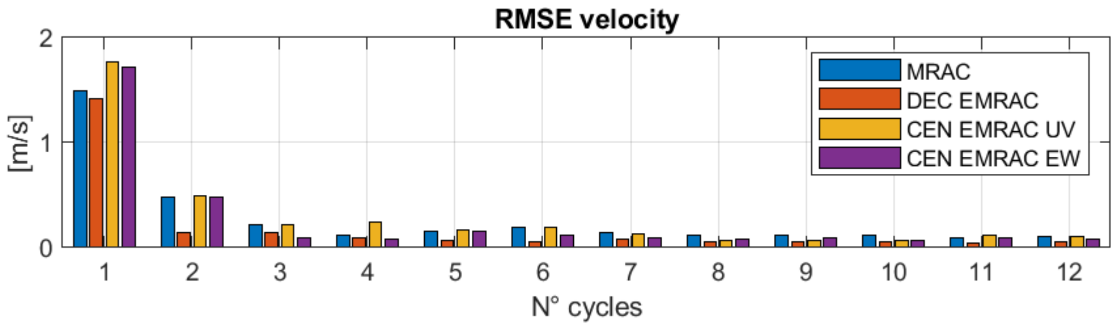


Figure B.27: RMSE velocities 50%

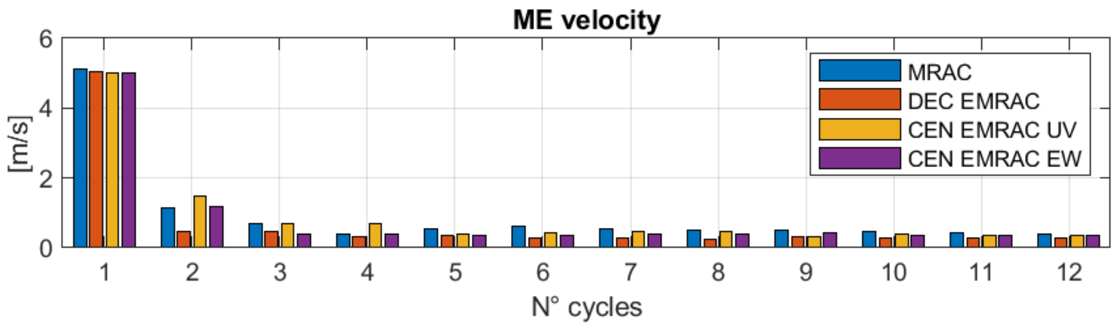


Figure B.28: ME velocities 50%

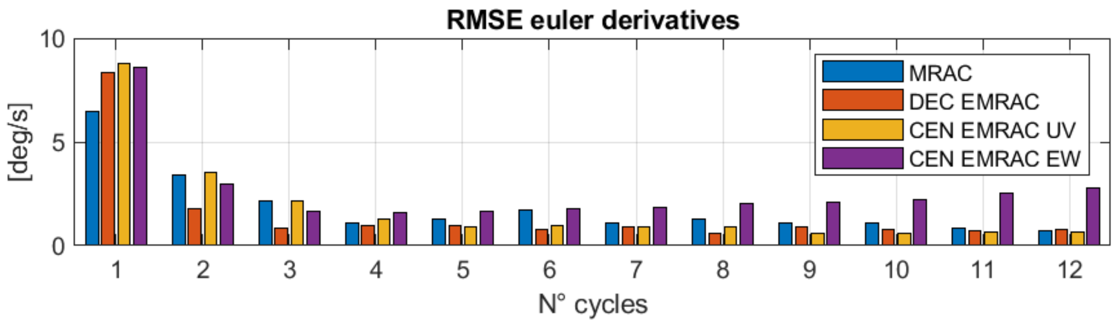


Figure B.29: RMSE euler derivatives 50%

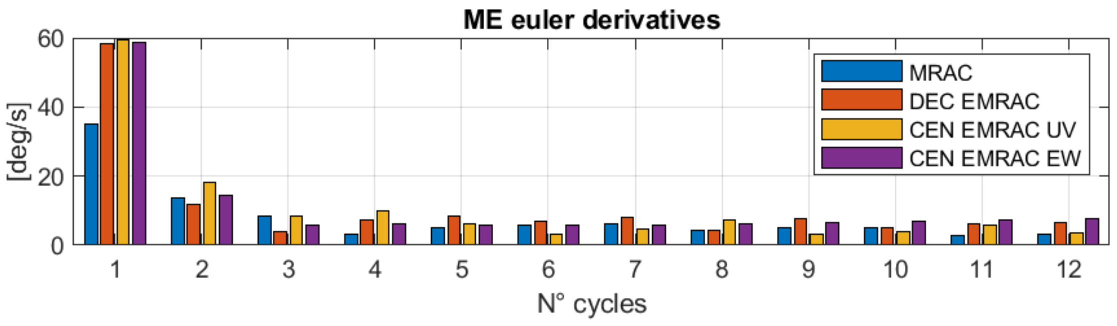


Figure B.30: ME euler derivatives 50%

Appendix C

Graphs

C.1 C-EMRAC-UV control action

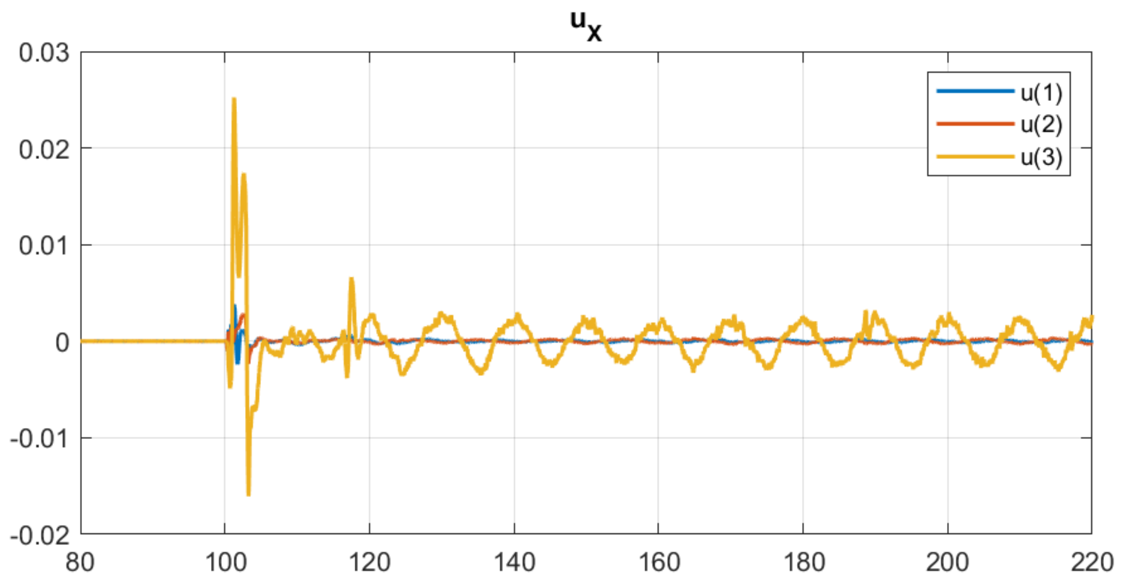


Figure C.1: C-EMRAC-UV u_X

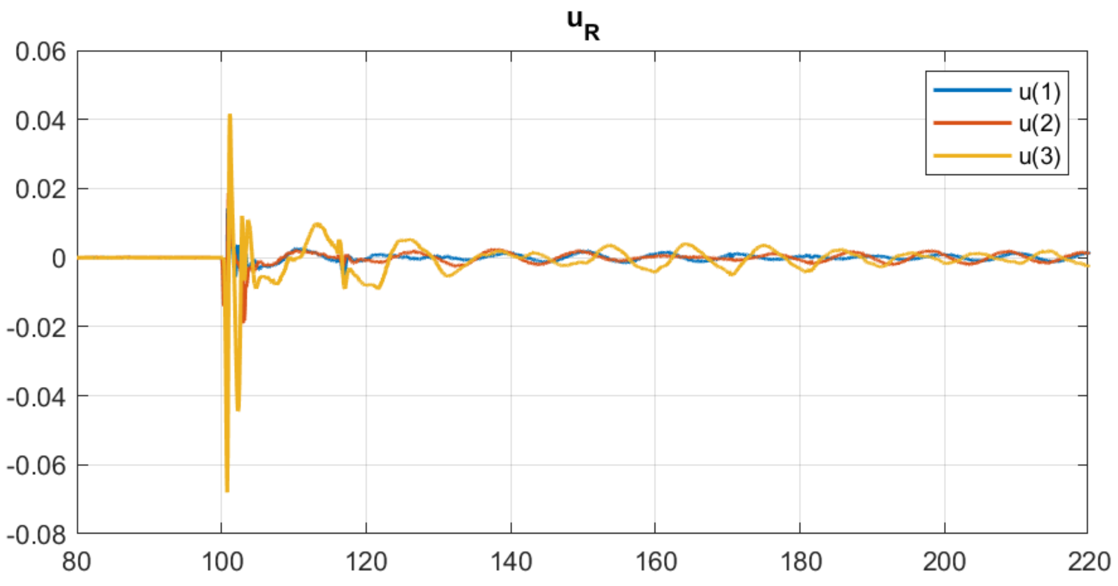


Figure C.2: C-EMRAC-UV u_R

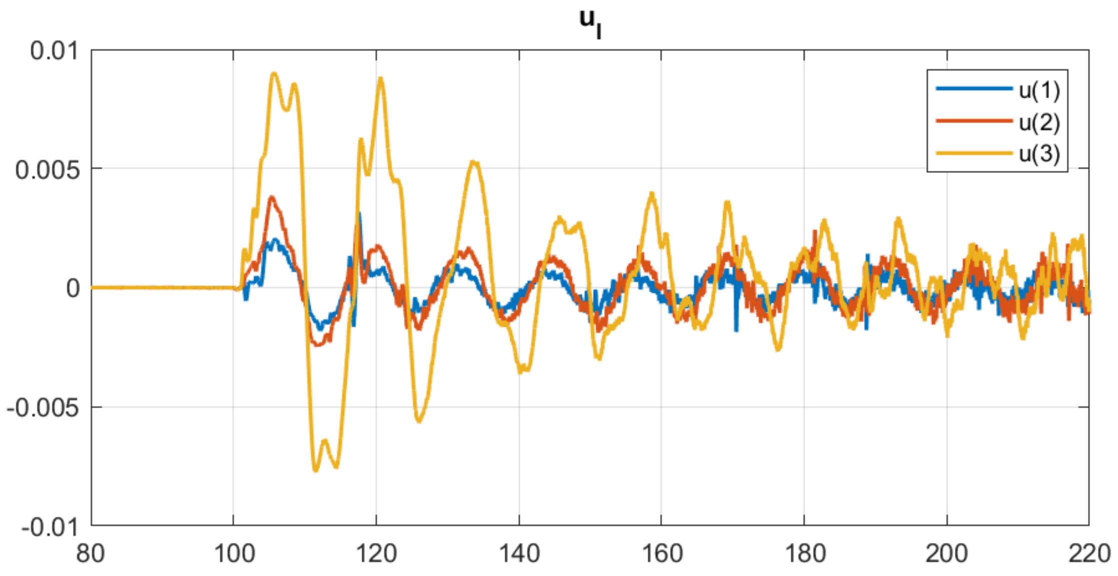


Figure C.3: C-EMRAC-UV u_I

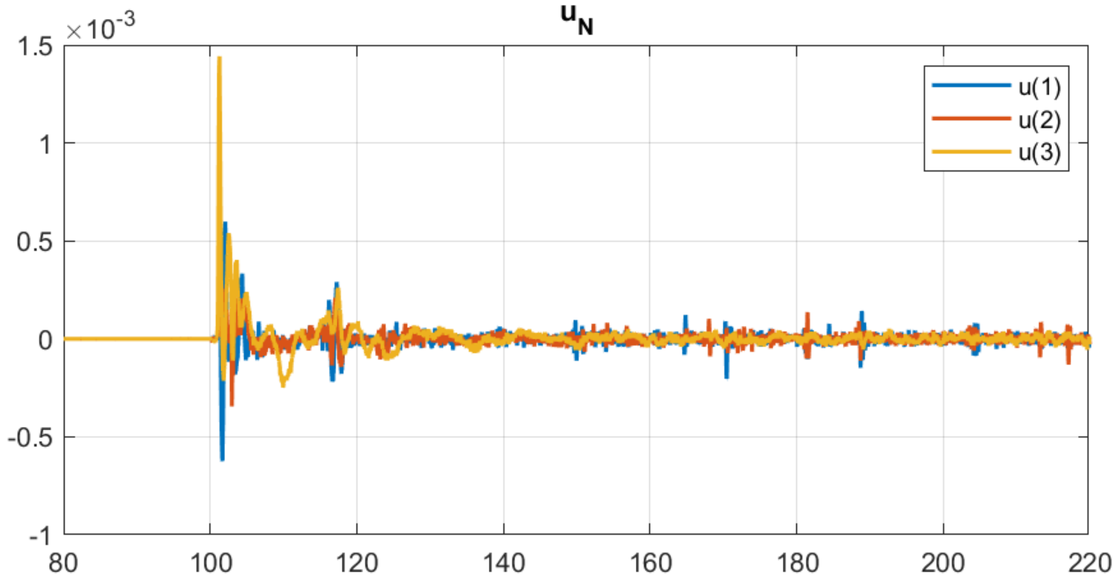


Figure C.4: C-EMRAC-UV u_N

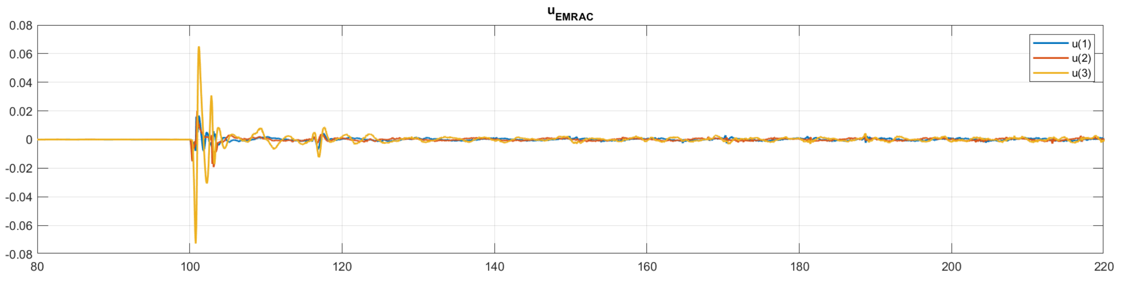


Figure C.5: C-EMRAC-UV u_{emrac}

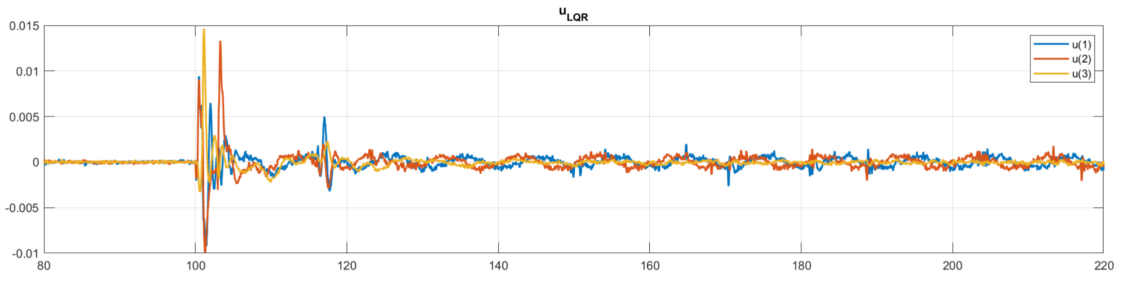


Figure C.6: C-EMRAC-UV u_{lqr}

C.2 C-EMRAC-UV gains

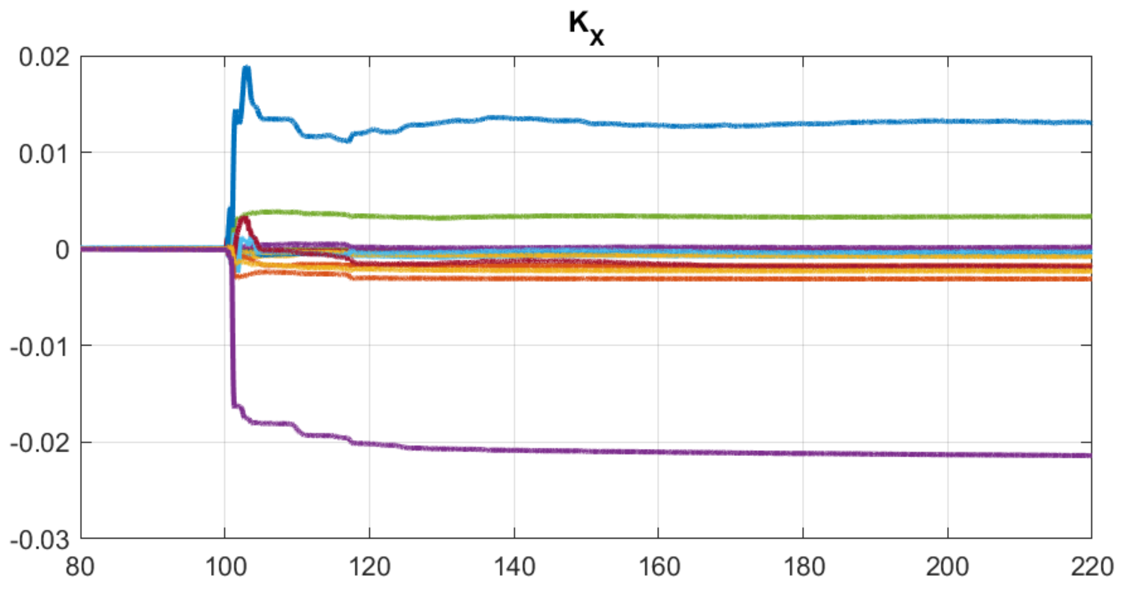


Figure C.7: C-EMRAC-UV K_X

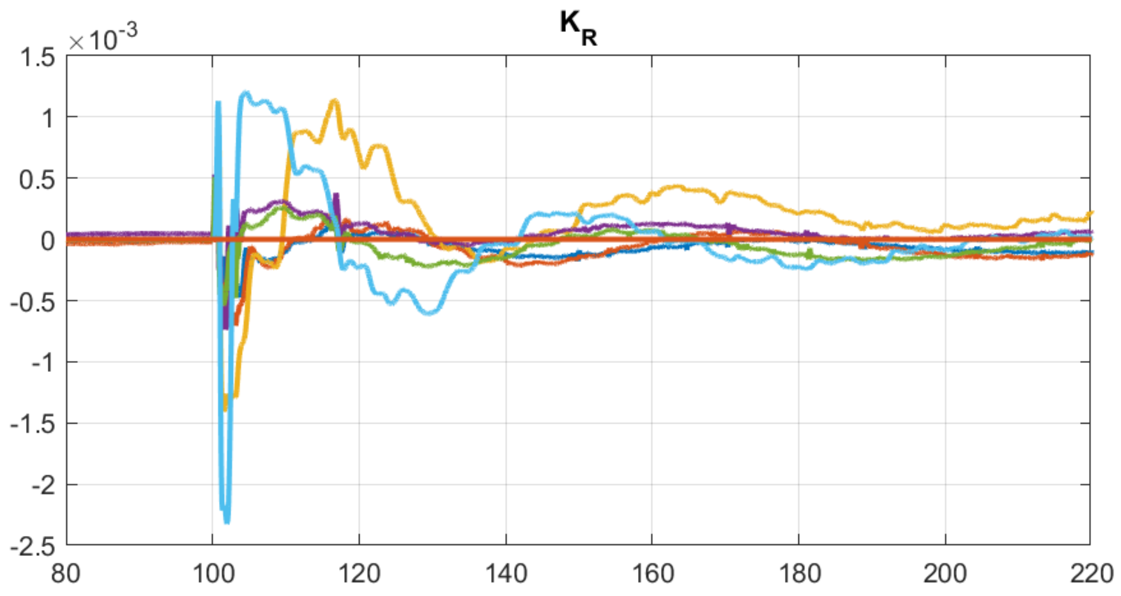


Figure C.8: C-EMRAC-UV K_R

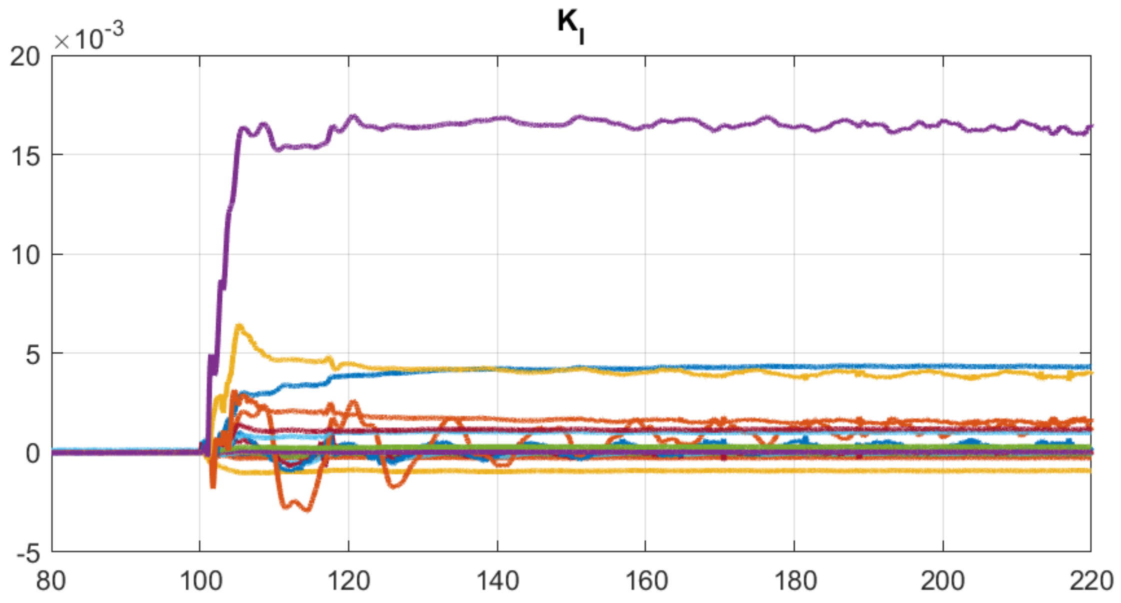


Figure C.9: C-EMRAC-UV K_I

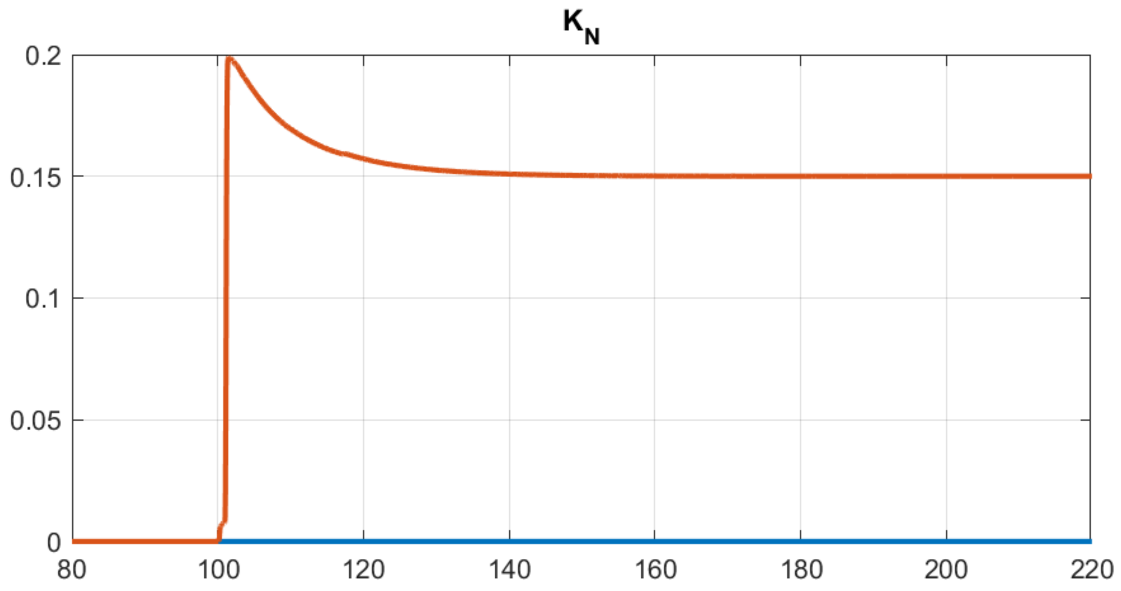


Figure C.10: C-EMRAC-UV K_N

C.3 C-EMRAC-UV σ-modification

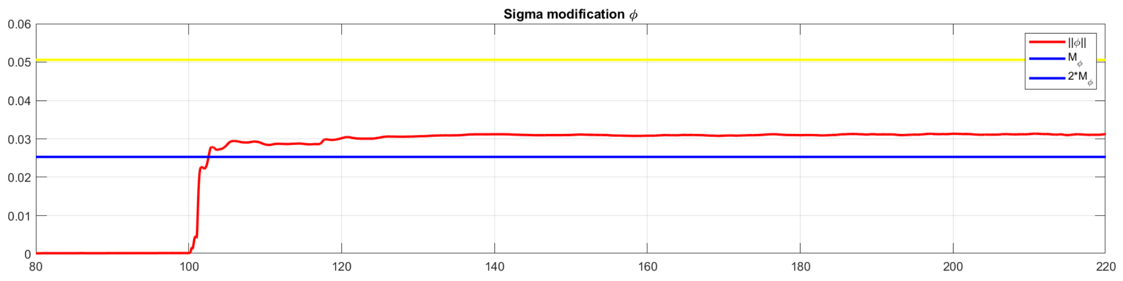


Figure C.11: C-EMRAC-UV σ-modification

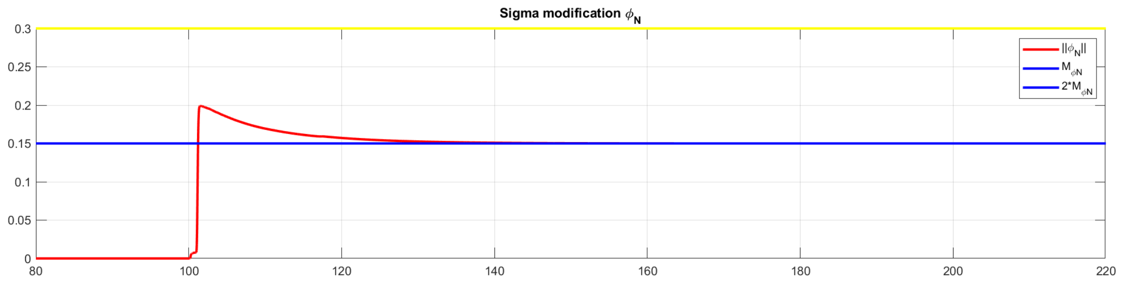


Figure C.12: C-EMRAC-UV σ_N-modification

C.4 C-EMRAC-EW control action

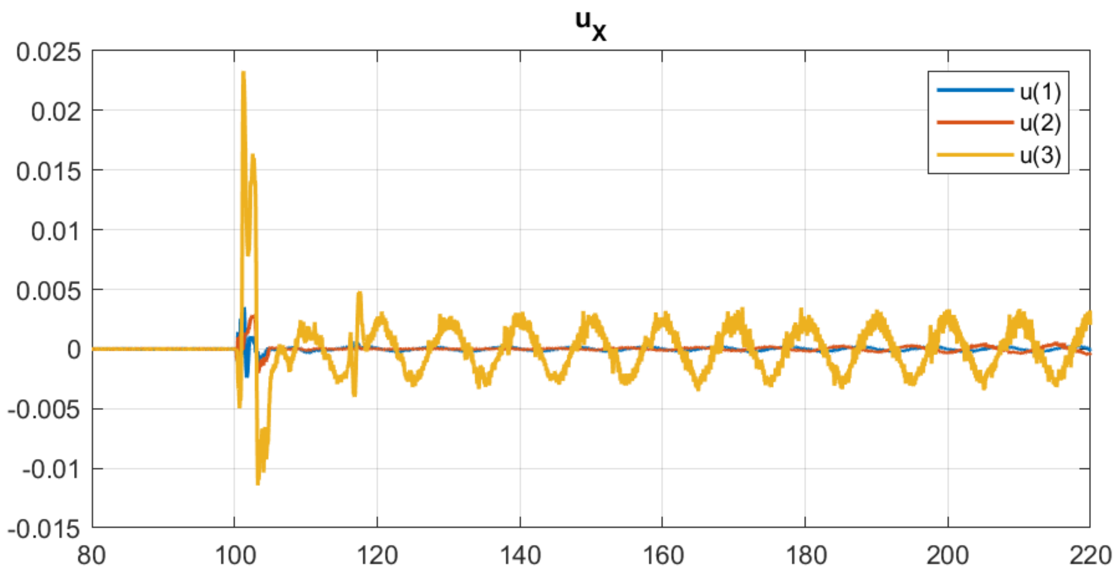


Figure C.13: C-EMRAC-EW u_X

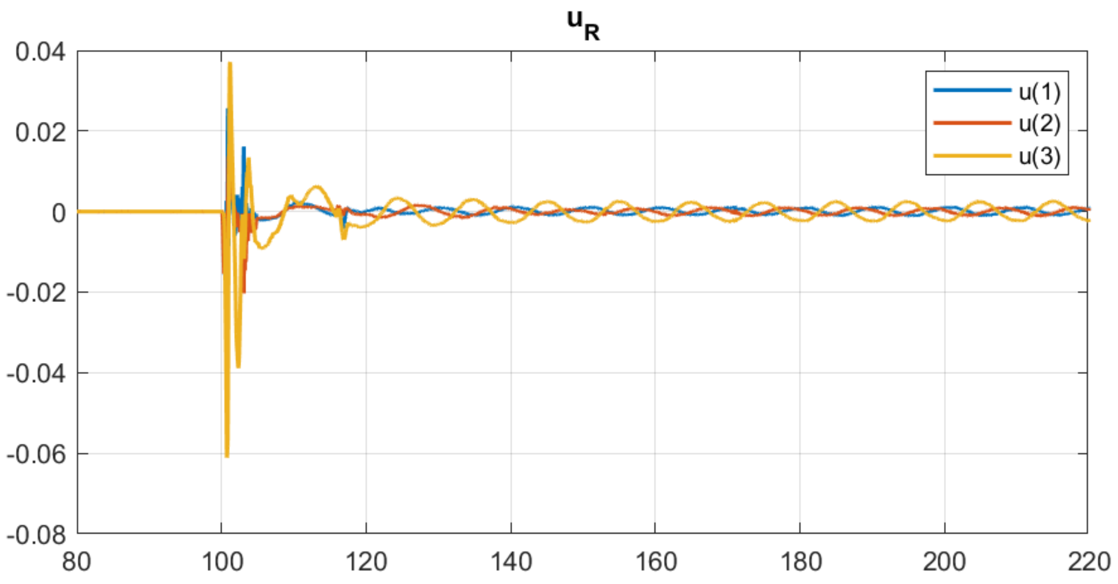


Figure C.14: C-EMRAC-EW u_R

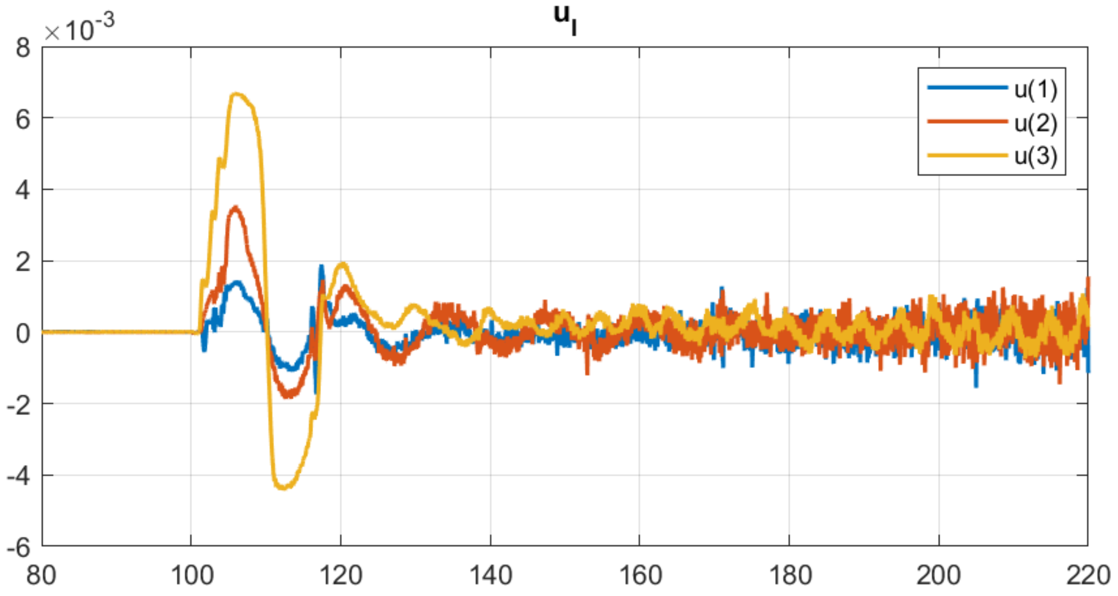


Figure C.15: C-EMRAC-EW u_I

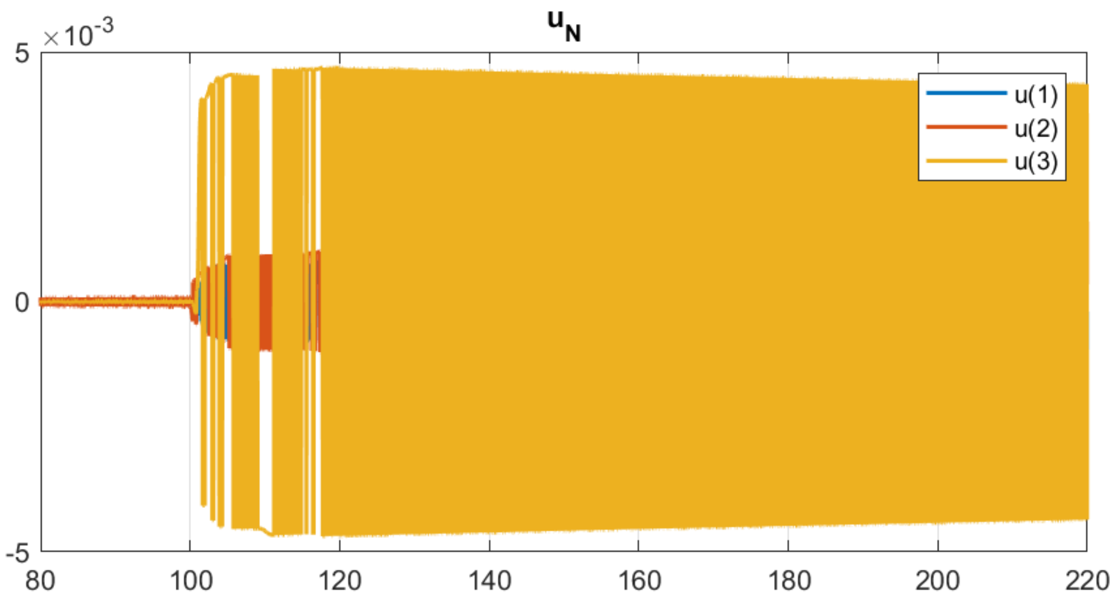


Figure C.16: C-EMRAC-EW u_N

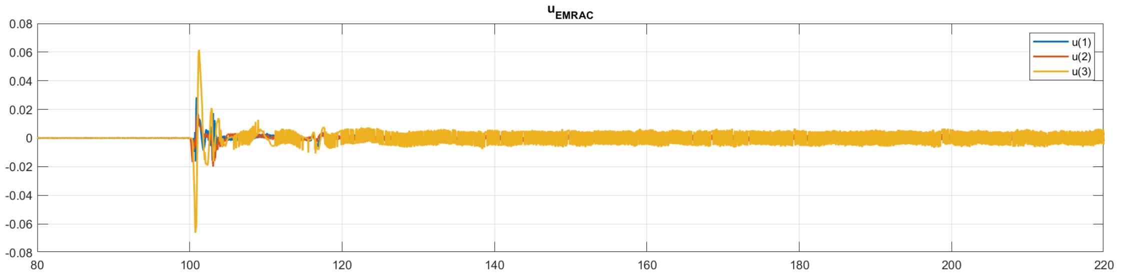


Figure C.17: C-EMRAC-EW u_{emrac}

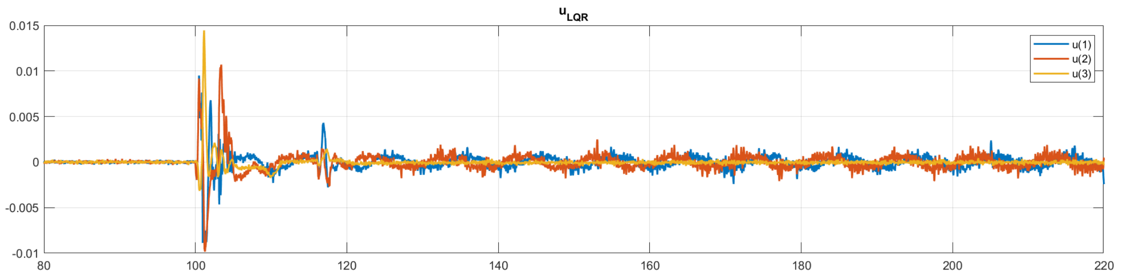


Figure C.18: C-EMRAC-EW u_{lqr}

C.5 C-EMRAC-UV gains

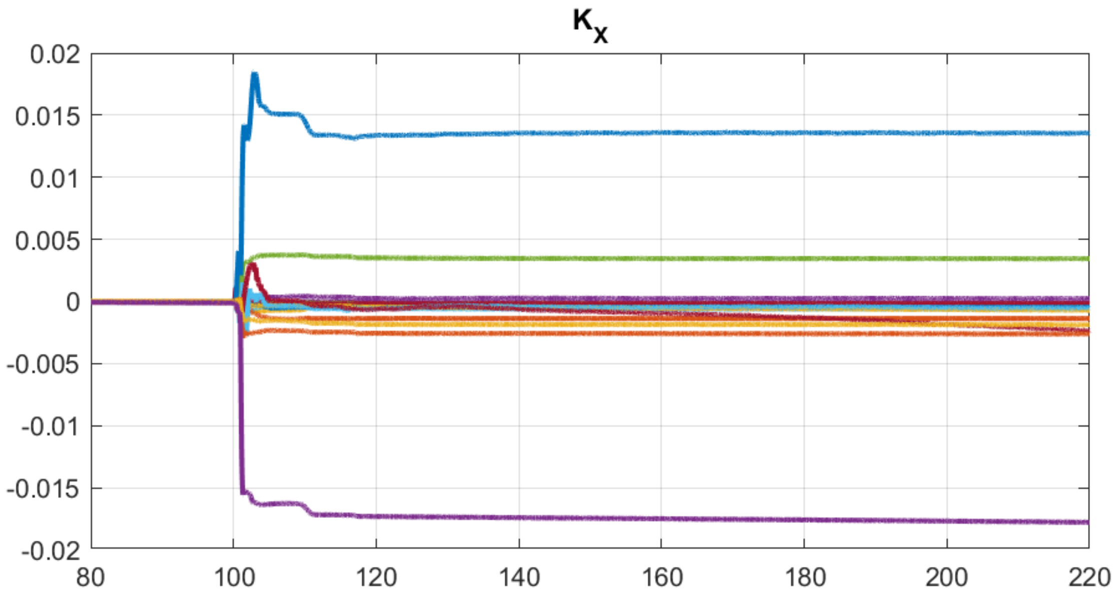


Figure C.19: C-EMRAC-EW K_X

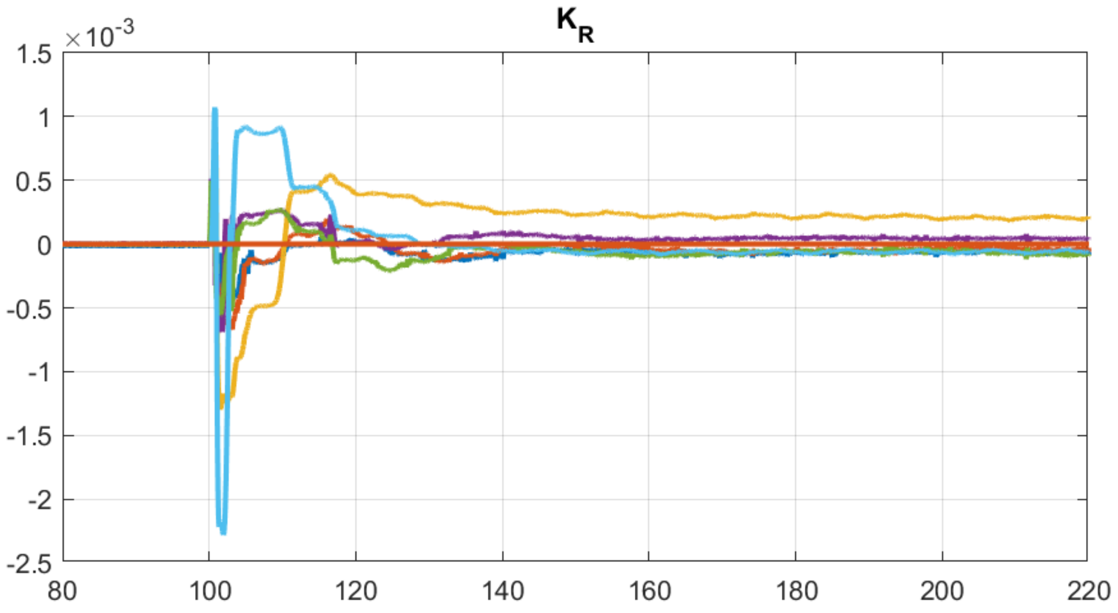


Figure C.20: C-EMRAC-EW K_R

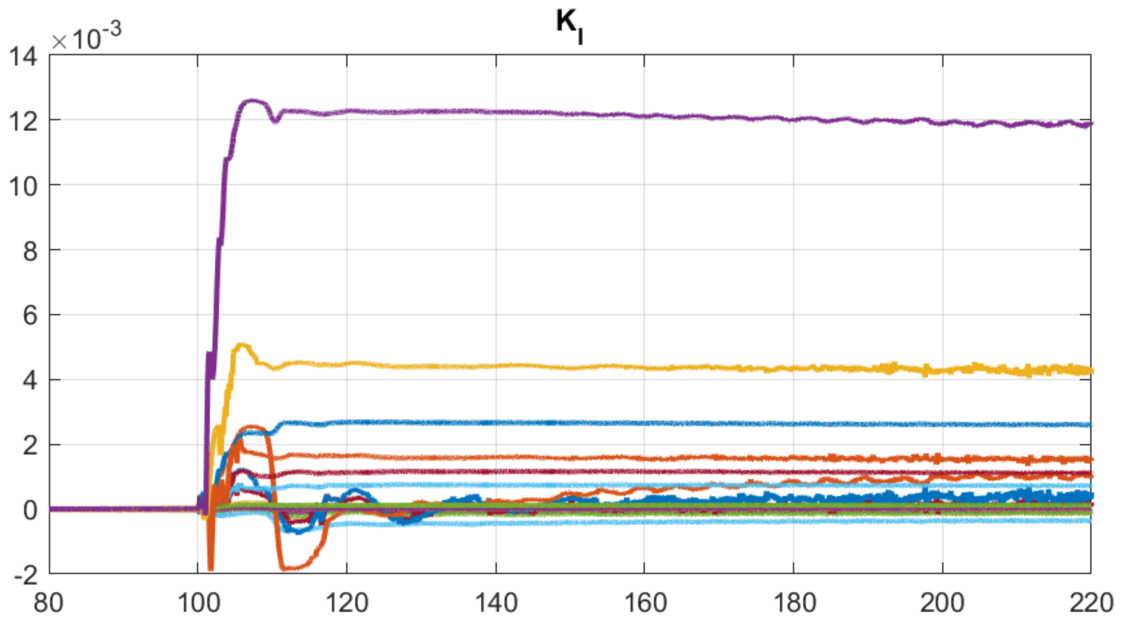


Figure C.21: C-EMRAC-EW K_I

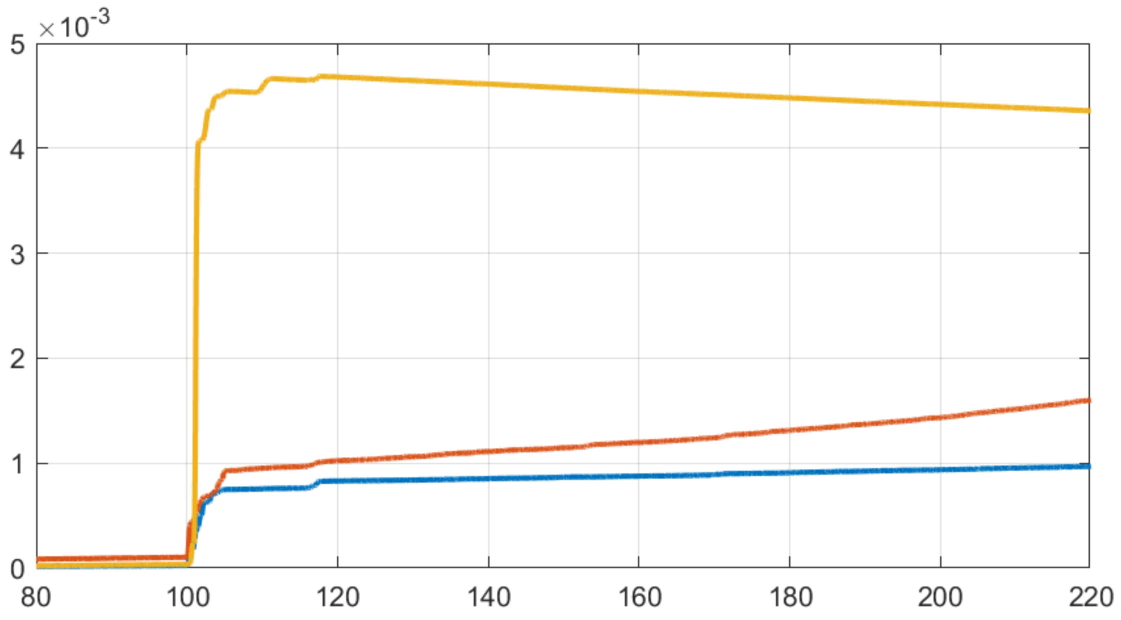


Figure C.22: C-EMRAC-EW K_N

C.6 C-EMRAC-UV σ-modification

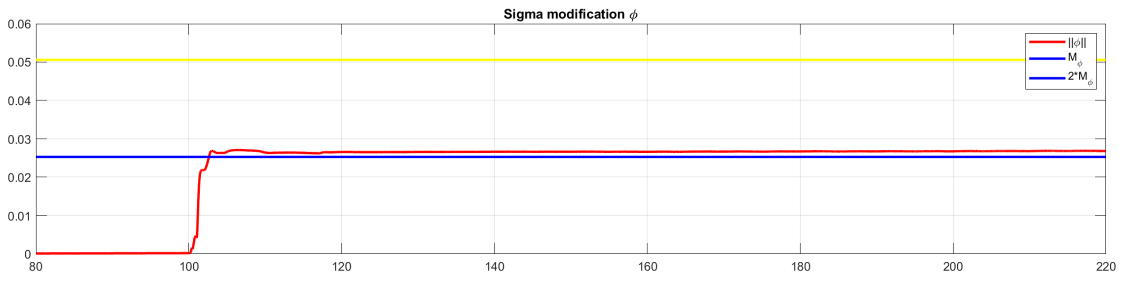


Figure C.23: C-EMRAC-EW σ-modification

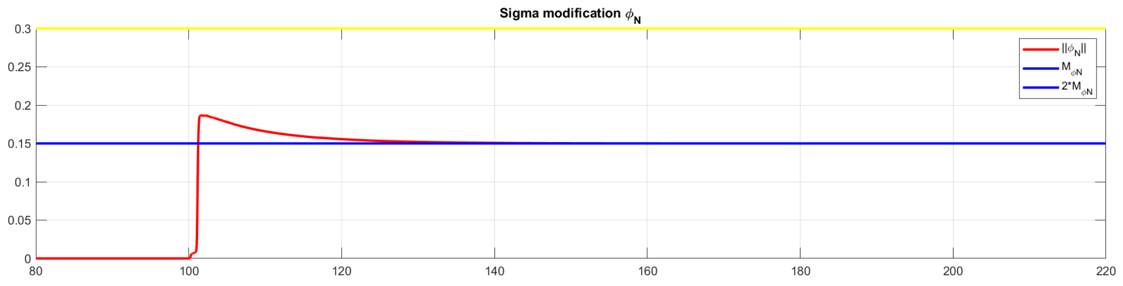


Figure C.24: C-EMRAC-EW σ_N-modification

Bibliography

[1] PX4 docs. *PX4 Reference Flight Controller Design.* URL: `https://docs.px4.io/v1.12/en/hardware/reference_design.html` (cit. on p. 15).

[2] PX4 user guide. *PX4 Architectural Overview.* URL: `https://docs.px4.io/main/en/concept/architecture.html` (cit. on pp. 15–17).

[3] Asper Garrett D. and Simmons Benjamin M. «Rapid Flight Control Law Deployment and Testing Framework for Subscale VTOL Aircraft». In: *NASA/TM–20220011570* (2022), pp. 7–8. URL: `https://ntrs.nasa.gov/api/citations/20220011570/downloads/NASA-TM-20220011570.pdf` (cit. on p. 19).

[4] Mathworks. *Integration with General PX4 Architecture.* URL: `https://uk.mathworks.com/help/supportpkg/px4/ug/px4-capabilities-integration.html` (cit. on p. 20).

[5] Holybro Docs. *Pixhawk 6X.* URL: `https://docs.holybro.com/autopilot/pixhawk-6x` (cit. on pp. 23, 24).

[6] Mathworks expo. *Simulate and Deploy UAV Applications with SIL and HIL Workflows.* URL: `https://uk.mathworks.com/content/dam/mathworks/mathworks-dot-com/images/events/matlabexpo/online/2022/simulate-and-deploy-uav-applications-with-sil-and-hil-workflows.pdf` (cit. on pp. 25, 27, 29).

[7] Mathworks. *Setup and Configuration for PX4 firmware in Simulink.* URL: `https://uk.mathworks.com/help/supportpkg/px4/setup-and-configuration_buiyb9j-1.html?s_tid=CRUX_lftnav` (cit. on p. 25).

[8] Mathworks. *Selecting PX4 Autopilot Application.* URL: `https://uk.mathworks.com/help/supportpkg/px4/ug/selecting-px4-autopilot-application.html` (cit. on p. 26).

[9] Mathworks. *UAV Toolbox Support Package for PX4® Autopilots.* URL: `https://de.mathworks.com/help/pdf_doc/supportpkg/px4/px4_ug.pdf` (cit. on p. 29).

[10] Mathworks. *MAVLink Connectivity for QGC, On-board Computer and Simulink Plant.* URL: `https://uk.mathworks.com/help/supportpkg/px4/ug/mavlink-bridge-hitl.html` (cit. on p. 30).

[11] Wang Xianghe. «Sliding Mode Control Design with Disturbance Observer for a Quadrotor». In: *Master's Thesis,Northeastern University,Boston, Massachusetts* (2021), pp. 6–10. URL: `https://repository.library.northeastern.edu/files/neu:bz617b344/fulltext.pdf` (cit. on p. 33).

[12] Yutao Jing, Xianghe Wang, Juan Heredia-Juesas, Charles Fortner, Christopher Giacomo, Rifat Sipahi, and Jose Martinez-Lorenzo. «PX4 Simulation Results of a Quadcopter with a Disturbance-Observer-Based and PSO-Optimized Sliding Mode Surface Controller». In: *Drones* 6.9 (2022). ISSN: 2504-446X. DOI: `10.3390/drones6090261`. URL: `https://www.mdpi.com/2504-446X/6/9/261` (cit. on p. 33).

[13] Davide Di Antonio. *CONTROLLO DI DRONI MULTIROTORE TRAMITE APPROCCIO HARDWARE-IN-THE-LOOP*. URL: `https://hdl.handle.net/20.500.12075/13065` (cit. on pp. 34, 35, 40–43, 62).

[14] Simone Martini, Serhat Sönmez, Alessandro Rizzo, Margareta Stefanovic, Matt J. Rutherford, and Kimon P. Valavanis. «Euler-Lagrange Modeling and Control of Quadrotor UAV with Aerodynamic Compensation». In: *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2022, pp. 369–377. DOI: `10.1109/ICUAS54217.2022.9836215` (cit. on p. 34).

[15] Umberto Montanaro, Simone Martini, Zhou Hao, Yang Gao, and Aldo Sorniotti. «Multi-input enhanced model reference adaptive control strategies and their application to space robotic manipulators». In: *International Journal of Robust and Nonlinear Control* 33.10 (2023), pp. 5246–5272. DOI: `https://doi.org/10.1002/rnc.6639`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/rnc.6639`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.6639` (cit. on pp. 44–47).

[16] Donald E. Kirk. *Optimal Control Theory: An Introduction*. Dover Publications, 2004 (cit. on p. 48).

[17] Zongyu Zuo. «Trajectory tracking control design with command-filtered compensation for quadrotor». In: *Control Theory and Applications, IET* 19 (Dec. 2010), pp. 2343–2355. DOI: `10.1049/iet-cta.2009.0336`.