Politecnico di Torino

Corso di Laurea Magistrale

in Mechatronics Engineering

Tesi di Laurea Magistrale

Embedded Model Control for Networked Control Systems

Supervisor

Prof. Carlo Novara

Assistant supervisor

Prof. Carlos Norberto Pérez Montenegro

Author

José María Pardo Álvarez

A.A 2018/2019

Summary

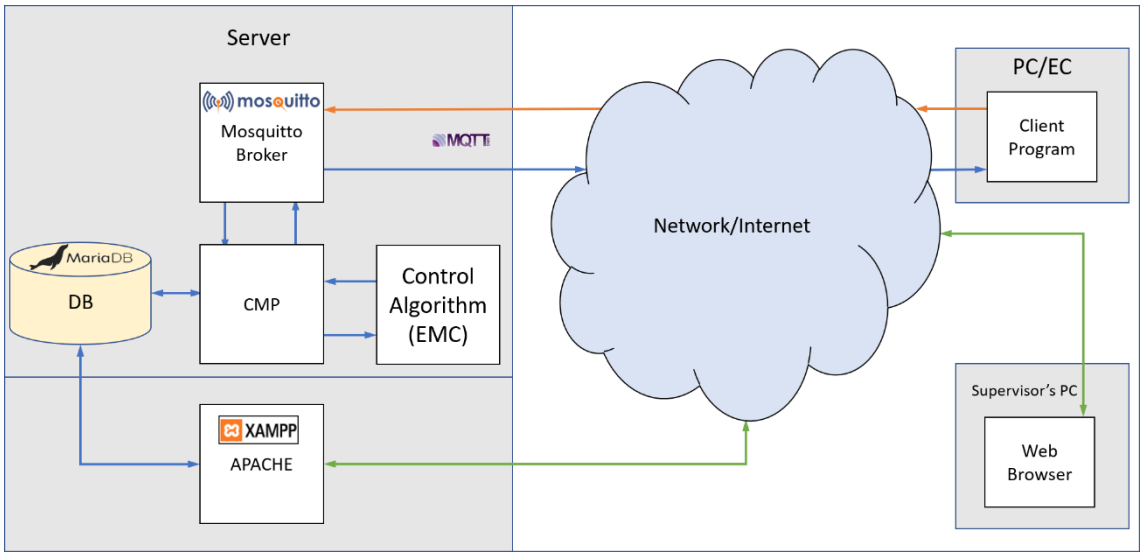
This thesis covers the design and implementation of a Networked Control System (NCS) based on the application of the Embedded Model Control (EMC) technique, developed in Politecnico di Torino university. It covers an explanation of the method, as well as the tools and technical components used for this purpose.

It also contains practical applications of the developed system and simulations to demonstrate the results.

It starts with a background and context explanation as well as a justification of the project, followed by the objectives to achieve.

Chapter 2 develops an explanation of the theoretical concepts used along this document, namely the EMC technique and NCS definition and characteristics. Enough detail is provided to understand the concepts used in the implementation, and additional material can be found in the references provided for further study.

In Chapter 3, the technical aspects and tools used in the implementation are described in order to understand the implementation described in Chapter 4. There, a V Model methodology is followed to develop the architecture (see figure below) and implementation of the desired control system with particular interest in the MQTT protocol for the communication, the implementation of the dynamic models used for obtaining results for Chapter 7, and the implementation of a web interface in order to hide complexity for the control engineer thought as the end-user.



Chapter 5 focuses in a modification made on the original EMC technique in order to deal with the variability of the time interval between the controller's sampling time due to the network influence on the system. This modification ensures stability over a very wide range of variation of this time interval, causing this technique to be aimed in applications which need stability as the main requirement over performance (understood as quickness of response).

Chapter 6 is dedicated to troubles found during the realisation of the project, in order to warn and advice possible replications and future modifications.

Finally, chapters 7 and 8 show results gathered from the developed system controlling simulated plants and the main conclusions extracted from them. Consequences derived are that the EMC technique succeeds in low frequency disturbs rejection (remarking bias rejection) tested in 100 executions with random parameter uncertainties and a remarkable tolerance to sampling time increases (up to a 180% increase in the tested example), while the modification exposed in Chapter 5 succeeds in keeping the system stable in all cases from a sampling time value of 0.5 seconds up to 10 seconds (a 1900% increment).

These results are tested implementing the control unit in localhost, local area through ethernet and in a multipurpose server located in Politecnico di Torino university while the plant is simulated in Spain using the Internet.

Table of contents

List of figures

List of tables

Chapter 1 Introduction and objectives

This project represents the Tesi di Laurea Magistrale (from now on just thesis) of the author, and aims to implement and test a control system involving a network, the Internet, by using the Embedded Model Control approach.

This method, developed in the Politecnico di Torino, is based on the control of a simulated model and actively reject disturbances and noises on measurements. Due to its own characteristics, it is suitable for being deployed in a networked environment where, because of the very nature of the network, several challenges arise to the control action.

The document is divided in several chapters that deal with the several aspects of this project.

It starts with the current context of Networked Control Systems (NCS), including definition of the concept and some of the common elements and characteristics inherent to these systems.

As the main objective of this project is to check the feasibility and adequateness of the Embedded Model Control over a networked environment, a deep enough description of the technique is provided in Chapter 2.

Chapter 3 deals with the required technical background used. It covers the diverse technologies, programs and languages used to implement the desired system.

In Chapter 4, the methodology used to create the system is discussed and the implementation details are explained. It represents the application of the technologies described in Chapter 3.

Several troubles, both expected and expected, were found during realisation of this project, and they are exposed in Chapter 5 in order to provide knowledge acquired and lessons learned for future developments based on this work.

Finally, Chapters 7 and 8 cover the obtained results and show the conclusions derived from them. The latter also provides some future modifications and lines proposed as a continuation of this project in order to improve it.

1.1 Current context

Nowadays, Internet's spreading through the world enables many professionals and companies to develop services which can be applied everywhere and anytime, and allows engineers to keep under constant innovation the techniques applied to many fields.

In the context of Control Theory and Automation, the Internet has been used for years to share and compile data from controlled systems, but mainly this has been used for remote high-level data gathering for further analysis or references and setpoint's manipulation such as SCADA systems, while the whole control loop is carried by microcontrollers and embedded controllers, PLCs, computers or any other elements, beside the controlled system (plant).

This is, mainly, due to the additional complexity that a communication network adds in a control loop (because of the delays imposed by the said network or the probability of messages' losses, for example), usually incompatible with requirements of control systems handling physical or fast plants, so traditionally the approach was avoided except in very well justified cases which had characteristics leading necessarily to a networked schema.

This usual vicinity between the controller and the plant is even more critical as the time constant of the plant is reduced because the delay introduced by the network would be too big if further distance is considered.

However, as Internet's connections are being improved and new control techniques arise, the impact of these disadvantages can be reduced and dealt with.

Recent developments as, for example, optical fibre, 5G and faster computing elements, make the interest in distributed control networks.

A strong prove of this is the rising interest in IoT[1] and sensor networks, where a huge number of computing elements can be geographically distributed and interconnected by a wide enough network being this, in many occasions, the Internet.

1.2 Reasons of the project

The traditional approaches for automatic control rely on several well-stablished architectures and strategies and mostly all of them require the plant and controller to be within a small range one from another. This is particularly challenging, as environments surrounding the plant tend to be quite hostile towards electronic equipment, so traditional controllers are usually robust and reliable by requirement.

In contrast, networked architectures allow to overcome some problems as sizing of the elements, as it becomes relatively easy to use big computing elements (from PC's to racks of servers and, more recently, cloud computing).

Because of these reasons, to implement a distant controller which can be sized as needed (scalable) and runs in a friendly environment becomes temptating. Unluckily, as distance between controller and plant is increased, so the noise, delays and other perturbances do.

This becomes specially important when dealing with Internet, which is a network of networks. Actors involved in the control loop have little or no margin to choose or modify the path followed by the packets, so delay times induced by the transport delays of the network are unknown (and quite random). This, added to the nonnegligible probability of packet's loss evidences the need for a robust control strategy, capable of overcoming these issues or, at least, avoid instabilities and degradation of the control loop.

Another challenging aspect is the modelling of the controlled system (the plant), as it is impossible to fully represent all the relevant dynamics and model complexity incurs in bigger development times and higher computation costs, making the need of balancing complexity and resources a crucial need.

This is where the Embedded Model Control (EMC) approach is pertinent, as it relies on the idea of a control command based on a simulation rather than on real measurements. This technique provides effective disturb rejection and as is model-based, it can be

[1] Internet of Things

effectively used in environments where measurement gathering is not as ideal as discrete systems usually require.

As for the model, it is designed to deal with non-considered dynamics, so the EMC has the ability of accounting those dynamics into what is called disturbed states, in order to provide a better control over the desired states.

In this project, the implementation of a networked version of the EMC algorithm is proposed and done, as well as an improvement over the original EMC technique that allows to deal with non-invariant sampling times in a way that is able to keep dynamics of the controlled system as designed.

1.3 Objectives

The goals of this project are, thus, to test the EMC technique (already proven by previous research to be theoretically stable and effective) in a networked environment, running the control unit in a remote server which adds computational power. This is due to the possibility of arbitrarily size the server according to the needs of the particular applications.

As for the detailed objectives of the project, they are:

- Confirm EMC's feasibility in a networked implementation
- Development of a fully functional system that implements a control loop between a controller in a server and a plant, using the previous EMC technique
- Development of a web interface to facilitate interactions with the system
- Test the asynchronous EMC technique in the previously implemented system

1.3.1 EMC suitability

The main objective (and reason) for this project is to implement a networked control system using the EMC approach to evaluate the adequateness of this method when Internet or another network is involved.

Mathematical development of the EMC technique proves its feasibility [1] and it has successfully tested and implemented where several real variations and disturbances are present. However, the imposition of a network makes it necessary to further test the capabilities in such a scenario.

1.3.2 Implementation of the control system

For keeping the results as close to reality, not only simulations should be done but also a working implementation of the control system.

The only part that is not implemented in this thesis is the controlled plant, which will be a real time simulation in Simulink.

In order to validate the system, several levels of architecture are used (see Chapter 4), to validate not only the EMC algorithm, but the communication loop involved and the computing elements used.

1.3.3 Development of a web interface

The main user of the system would be a control engineer, who should be able to manipulate the control parameters without too much complexity in those aspects more related to a computer engineer.

This is proposed to be done by creating a simple web to provide the possibility of monitorisation of the relevant variables and to act on the control strategy itself.

Chapter 2 Theoretical background

This chapter deals with the details of the theoretical aspects of this project regarding mainly characteristics and definition of networked control systems and the Embedded Model Control method. Mathematics behind it will be introduced deeply enough to allow the reader to understand the applications of it, and references for further study are provided.

2.1 Networked Control Systems

As opposed to the traditional scheme used in control applications, a networked control system involves several elements interconnected by a network, which is considered as another element as important as the controller or plant.

With the increasing connectivity around the world and the advantages that these systems have over traditional schemes leads to a remarkable increase in their use in a wide variety of scenarios, such as remote surgery [2], IoT sensor's management, etcetera.

Despite the variety of architectures that can be implemented, there are some characteristics (both advantages and potential problems) that are in some degree common for all of them.

All these aspects can be dealt with by imposing an abstraction layer, decoupling the complexity of the communications layer and the control schema [3].

2.1.1 Communication channel's imperfections

As the network is not ideal, the presence of delays, packets' losses, interferences (specially in wireless networks [3] [4]) adds complexity to the overall system. Thus, is necessary to take these imperfections into account as they can lead to instability.

Delays

As a simple approximation, we can consider 3 sources of delays in a network:

- Computation's delay
- Network's access' delay
- Network's transmission's delay

The computation delay refers to the time added in signal's computation in the various digital components involved. They are usually very small compared with the rest of the delays, so it's usually neglected (as it's done in this thesis).

Once the signal or packet is ready for transmission, it can be delayed before being sent through the network due to congestion or management of the messages' queues. This problem is tightly related to the maximum throughput reached in this thesis, as in the implementation (see MQTT tests, page 44).

Finally, after the message it's dispatched by the messages' enqueuer, it takes some time for the packet to arrive at destination, which is denoted as the network's transmission delay.

Packets' dropouts

In any network, packet's dropouts are likely to occur as a consequence of a network congestion or failure between the links involved [4], and this is more likely to happen if the Internet is involved, as data paths are assigned dynamically and the end user (i.e. sender or receiver) doesn't have any control over the design of these paths.

Order of arrival

As the information's path is not controlled when using the Internet, there's a not negligible probability of packet's not arriving in order. One of the possible approaches to deal with this phenomenon is to timestamp each packet, and discard those that were supposed to arrive earlier than the most current and successful reception. However, this approach is not always convenient as sometimes valuable information can be still obtained from those discarded packets.

2.1.2 Network's interfaces

Another aspect to be considered and specified during the implementation of an NCS is how the controller and plant interact with the proposed network. This is, for example, what kind of protocol (see MQTT, page 21) is used or what kind of signals and their format or codification.

A good interface allows to hide the network complexity to the control engineer and provides means for an easy handling of information without losing too many capabilities and flexibility.

2.1.3 Differences in time's bases

As one of the most important characteristics of a control system is time, differences in the different element's clocks can lead to problems. Synchronisation, clock analysis and timestamping become critical in a networked environment, especially when "hard" Real Time systems are considered.

Several approaches exist, such as using an NTP server or handling timestamps on a single machine in the way that always the same source of time is used.

2.2 Embedded Model Control

From now EMC, the Embedded Model Control approach for control is a methodology developed in the Politecnico di Torino university and works on the principles of modelling up to a certain degree the target plant, and then update the model with information about non-modelled elements.

The controller itself contains a discrete model of the plant with different states: controllable and not controllable states, $\mathbf{x_c}$ and $\mathbf{x_d}$.

While a more elaborated explanation is presented below, the main idea behind EMC is the simulation of the plant in a model embedded into the control loop from where the control output (i.e. the command to be sent to the actuator) is derived. As the model does not represent the complete reality, deviations of the signals measured on the plant from their simulated counterparts are introduced into the model through a noise estimator, which tries to estimate some of them in order to later cancel them through the command law.

It is worth to mention that a thorough study of the EMC, its stability and several variants can be found in [1], [5], [6] and [7].

2.2.1 Embedded Model

The EMC technique relies on the simulation of the so-called Embedded Model, which is executed in the controller.

This discrete model represents the plant within a certain level of detail and the command law uses the states of the model in order to compute the output (command). These states can be grouped into 2 major sets:

- Controllable states, $\boldsymbol{x_c}$
- Disturbance states, $\boldsymbol{x_d}$

The first one, $\boldsymbol{x_c}$, is composed of those states which are affected by the command input and is the set of states which have to reach the states reference, while the second one, $\boldsymbol{x_d}$, contains those states which represent deviations not controllable by the command law.

The main goal of this approach is, thus, to guide the controllable states towards the desired objective and to counterbalance/cancel the disturbance states' effect over $\boldsymbol{x_c}$.

The generic equations for an EM are, taken from [5]:

$$\boldsymbol{x}(i+1) = A\boldsymbol{x}(i) + Bu(i) + G\boldsymbol{w}(i) \tag{2.1}$$

$$\boldsymbol{y}_m = C\boldsymbol{x}(i)$$

$$z_m = F\boldsymbol{x}(i)$$

where

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x_c} \\ \boldsymbol{x_d} \end{pmatrix}, \qquad A = \begin{pmatrix} A_c & H_c \\ 0 & A_d \end{pmatrix}, \qquad B = \begin{pmatrix} B_c \\ 0 \end{pmatrix}, \qquad G = \begin{pmatrix} G_c \\ G_d \end{pmatrix}$$

$$C = (C_c \quad C_d), \qquad F = (F_c \quad 0)$$

As for the notation used, bold lowercase is used for vectors ($\boldsymbol{x}, \boldsymbol{w}$), normal lowercase for scalars (z_m, w_0) and uppercase for matrices (A, G, A_c).

It can be noted that the next value for $\boldsymbol{x_c}$ is affected by its current state through A_c, by $\boldsymbol{x_d}$ through H_c, and by u through B_c, while $\boldsymbol{x_d}$ is only affected by $\boldsymbol{x_d}$.

The non-considered dynamics and disturbances rejection comes from the $\boldsymbol{w}$ vector, the output of the Noise Estimator. This disturbance vector affects through the matrix G.

The matrix C acts as expected from classical state space representation, and z_m, obtained through F matrix can be seen as a performance parameter to be minimised or maximised, depending on the control strategy goal. For the sake of simplicity, $z_m = y_m$ is used in this thesis.

All the elements described before constitute the Embedded Model itself, but there are still modules remaining in order to achieve control.

2.2.2 Noise Estimator

One of the characteristics of the EMC is the existence of a Noise Estimator, which is in charge of measuring deviations of the simulated output from the measured output in order to estimate part of the disturbances and feed them back to the EM to update the disturbance states $\boldsymbol{x_d}$.

The generic equation is as follows:

$$\boldsymbol{w} = L\boldsymbol{e_m} \tag{2.2}$$

With

$$\boldsymbol{e_m} = \boldsymbol{x} - \boldsymbol{x_m} \tag{2.3}$$

Being $\boldsymbol{x_m}$ the states obtained from the embedded model and $\boldsymbol{x}$ the states measured from the plant (affected by disturbances and noise). In this sense, $\boldsymbol{e_m}$ (model error) represents deviation of the model's states from the plant's states.

From this unit, the estimation $\boldsymbol{w}$ is obtained and can be fed to the embedded model.

As a result, the embedded model and the noise estimator act as a one-step predictor of the controlled system, and from this prediction, a one-step prediction of the command sent to the actuator is computed.

This is a remarkable characteristic of the EMC, as it allows to compute the command law one step ahead so it can arrive to the actuator and be executed in its right time (the

network induces a delay, of this delay is smaller than the expected control sampling time, the effect of this delay can be neglected).

2.2.3 Command Law generator

This is the part of the controller in charge of computing the command to be sent to the plant by using the information provided by the EM.

It is the sum of three components, in order to achieve a better response in presence of disturbances and non-linearities.

The first term refers to the ideal and linear command needed to control the system's dynamics corresponding to the controllable states.

The second term is an error feedback to bring the states to the desired reference due to the presence of neglected dynamics.

The last term performs a disturbance rejection to minimize the effect of disturbs on the states of the system.

The equation is as follows:

$$u_m(i+1) = \underline{u}(i) + K\,\underline{\boldsymbol{e_c}}(i) - M\boldsymbol{x_d}(i) \tag{2.4}$$

Where

$$\underline{\boldsymbol{e_c}}(i) = \underline{\boldsymbol{x}}(i) - (\boldsymbol{x_c}(i) + Q\boldsymbol{x_d}(i)) \tag{2.5}$$

From this point, only matrix K has to be tuned, as M and Q are matrices defined by the equality:

$$\begin{pmatrix} H_c + QA_d \\ 0 \end{pmatrix} = \begin{pmatrix} A_c & B_c \\ F_c & 0 \end{pmatrix} \begin{pmatrix} Q \\ M \end{pmatrix} \tag{2.6}$$

As the reference dynamics are:

$$\underline{\boldsymbol{x}}(i+1) = A_c\underline{\boldsymbol{x}}(i) + B_c\underline{u}(i) \tag{2.7}$$

$$\underline{z}(i) = F_c\underline{\boldsymbol{x}}(i)$$

an ideal command law ($\underline{u}$) can be computed by any control technique. In thesis, pole placement by a static feedback is used.

For the tuning of K, pole placement is done by assigning the desired eigenvalues to the matrix $A_t = A_c - B_c K$.

As an example of the reference dynamics[1] and the pole placement procedure followed, by using a simple punctual mass model with the terms computed assuming a constant control timestep T:

$$\underline{\boldsymbol{x}}(i+1) = A_c \underline{x}(i) + B_c \underline{u}(i)$$

$$\underline{u}(i) = -K\underline{x}(i)$$

$$A_c = \begin{pmatrix} 1 & T \\ 0 & 1 \end{pmatrix}, \qquad B_c = \begin{pmatrix} 0 \\ T \end{pmatrix}, \qquad K = (k_1 \quad k_2)$$

$$A_t = \begin{pmatrix} 1 & T \\ -Tk_1 & 1 - Tk_2 \end{pmatrix}, \qquad A_t - I\lambda = \begin{pmatrix} 1-\lambda & T \\ -Tk_1 & 1 - Tk_2 - \lambda \end{pmatrix}$$

$$|A_t - I\lambda| = \lambda^2 + (Tk_2 - 2)\lambda + T^2 k_1 - Tk_2 + 1$$
$$(\lambda - p_1)(\lambda - p_2) = \lambda^2 - (p_1 + p_2)\lambda + p_1 p_2$$

Identifying terms in the last equation:

$$Tk_2 - 2 = -(p_1 + p_2)$$
$$T^2 k_1 - Tk_2 + 1 = p_1 p_2$$

So

$$k_2 = \frac{-(p_1 + p_2) + 2}{T}$$
$$k_1 = \frac{p_1 p_2 + Tk_2 - 1}{T^2}$$

[1] Note that as the example refers to the reference dynamics, there isn't any disturbance state and $\underline{\boldsymbol{x}}$ refers to the controllable dynamics.

2.2.4 Stability

One of the most important aspects of any control system is stability, as an unbounded response usually leads to destruction of the controlled system at least. As a more serious consequence, damage to persons or even casualties can occur.

Several techniques and criteria exist to analyse and ensure stability, and in this thesis the BIBO stability is considered. This means that the system must react boundedly whenever a bounded input is provided.

When dealing with systems in their state space representation, a very extended and useful technique for stability analysis is the determination of the eigenvalues.

This is, if we have a discrete state space representation:

$$\boldsymbol{x}(n+1) = A\boldsymbol{x}(n) + B\boldsymbol{u}(n)$$

$$\boldsymbol{y}(n) = C\boldsymbol{x}(n) + D\boldsymbol{u}(n)$$

the system will be asymptotically stable if all the eigenvalues of A, designated by λ_i satisfy the condition

$$|\lambda_i| < 1, \ \forall i \qquad 2.8$$

$$\lambda_i \in \mathbb{C}$$

In the continuous case, an analogue condition holds:

$$\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + B\boldsymbol{u}(t)$$

$$\boldsymbol{y}(t) = C\boldsymbol{x}(t) + D\boldsymbol{u}(t)$$

the system will be stable if the eigenvalues of A satisfy

$$Re(\lambda_i) < 0, \ \forall i \qquad 2.9$$

$$\lambda_i \in \mathbb{C}$$

For these reasons, one of the main purposes of control systems is to modify the system dynamics to force all the eigenvalues to satisfy inequality 2.8 in discrete systems or 2.9 in continuous scenarios.

As in the EMC technique, eigenvalues assignment is done by feedbacking the states with through a properly computed gain K, stability is ensured as long as the sampling

time keeps within certain limits, as the eigenvalues of a discretised system strongly depends on T.

Demonstrations and stability analysis on the EMC method can be found in [1] and [5]

In section 4.5, the values used in the implementation are presented and, in Chapter 5, a method for dealing with a variable sampling time is presented and explained, as a major challenge for sampled systems is to guarantee a stable sampling rate, something very difficult in an NCS.

Chapter 3 Technical background

This chapter deals in detail with the diverse technical elements involved in the project. Some of them are involved with the networked nature of the system, like MQTT and Apache. Others apply to the implementation of the programs used, as PHP and threads, and the RTC and threads are oriented to the timing issues.

Additionally, Wordpress and SQL are used to provide a Control-Engineer-friendly interface and data storage.

3.1 MQTT

MQTT is a TCP-based protocol for exchanging messages in a MQ message queuing system. In short, a client can implement an MQ system (in this particular case it will be called BROKER) and can exchange messages with other MQ systems using the MQTT protocol.

Originally developed by Arlen Nipper (Eurotech) and Andy Stanford-Clark (IBM) [8], it was designed for managing data communication across several clients, in opposition to the traditional point-to-point. The main requirements proposed during development of the protocol were:

- Simple to implement
- To have a quality of service data delivery
- Lightweight and bandwidth efficient
- Data agnostic
- Continuous session awareness

As a result, an implementation of the MQTT protocol leads to a low-power consumption and bidirectional communication between clients in a publish/subscribe paradigm.

The advantages of this particular method over others which also allow communication between clients (such as HTTP POST) derive from the fact that MQTT was developed by IBM to handle small size messages between IoT devices, so battery consumption as

well as bandwidth usage by this protocol are reduced, increasing the throughput as a consequence [9].

Publication/subscription is different from request/answer in that:

A client can send (publish) a message to a channel (from now TOPIC), and that message is redirected by the broker to those clients who subscribed to that topic.

One particularity of this schema is that, by default, clients connected to the broker (either for publishing or subscribed) can't see other clients, so a message can't be sent to a specific client (it can, though, by publishing to a certain topic which only a specific client or group of clients can subscribe to).

Another main advantage derived form MQTT is the decoupling between data generation and data consumption due to the publication/subscription schema. Data sources (e.g. sensors and instrumentation) can communicate their measurements by publishing them into the network, and different clients can subscribe to topics of their interest, without having to know what kind of client sends the data and without having to change the communication method (i.e. they don't have to behave differently whether they are communication with a server, a PC or an embedded device).

This means that, apart from making things easier for the clients to communicate without having to deal with the technical issues of it, new opportunities for data analysis arise as new clients can subscribe to new data sources.

As IBM released MQTT's source codes for the general public in 2011, several implementations of MQTT services are available. In this thesis, MOSQUITTO is used as implementation, due to the fact that it's open source, lightweight and can be customised to the particular needs of the system, but other implementations could have been used (in particular, IBM offers its own implementation and, in a future, it could be used). Mosquitto can be downloaded as binaries for Windows and Linux or as source code[1] to be built (compiled).

In this particular project, binaries were used for Windows platforms (in this case thread support and websockets compatibility is not available as they were precompiled) and custom compilation from source was used for Linux platforms with thread support and websockets compatibility.

[1] https://mosquitto.org/

Another capability of the broker is to have multiple listeners. Each listener is linked to a particular port and can be configured independently to have different settings. This allows, for example, to define different interfaces with TLS encryption, Access Control List or other parameters.

In this thesis, MQTT v3.1.1 specification was used, as it is widely used and accepted as a standard by the OASIS Consortium and by ISO [10].

3.2 Wordpress

For implementing the webpage, the author opted for Wordpress. This is a package that allows to create a page based on several templates, allowing the designer to focus mainly on the appearance and functionality without losing the capabilities of a full customization by modification of the Wordpress' PHP source files.

It can be installed directly from the official webpage, and configuration takes at maximum 5 minutes to set a first and simple Web. It uses a working installation of MySQL or MariaDB database to store all the Web's content, and provides a very friendly User Interface (UI) for the site's management (namely blog-related actions as management of posts and content, management of the existing pages, users and members administration, etcetera…).

3.3 Apache

The Apache Foundation is an organization that aims to provide a full suite of open source tools to implement Web-related services. In particular, the Apache HTTP Server provides a full, cross-platform web server, commonly used as an element in a XAMP[2] stack (in this project both LAMP and WAMP stacks were used).

[2] XAMP is an acronym for the OS (X), Apache (A), MySQL (M) and PHP (P), where the OS can be Windows (WAMP), Linux (LAMP) or, generically, X.

3.4 PHP

PHP is an interpreted scripting language (with many similarities with C) specially created for being executed on Web servers. It allows to implement logic and functionalities to a webpage, and in this particular project is used to provide functionalities when accessing through the Web portal.

3.5 SQL

Structured Query Language (SQL) is, informally, a language for managing data in a relational database. It became a standard for the ANSI in 1986 and of the ISO [11] in 1987.

In this project, there is a database in which values of the signals involved are recorded, and they are accessed through SQL queries using a PHP module, for access from the Web, and ConnectorC[3], the C library for access from the device holding the database itself.

Several databases support and can be queried by this language, and MariaDB is used in this thesis as is trivial to install from the Linux repository and is open source. As

3.6 Threads

As one of the characteristics of the system, the ability to manage several plants with different control strategies, a fundamental problem appears as there can be races and collisions between control programs if two clients demand commands within a minimum difference of time.

The solution implemented makes use of threads. They are the minimum non-preemptable set of instructions that a processor must handle without changing its context.

[3] For installing it in Linux, the package *default-libmysqlclient-dev* installs ConnectorC or the corresponding equivalent depending on the specific database server installed (ConnectorC for Oracle's MySQL server or other for MariaDB, for example).

As one plant sends its measurement, the CMP starts an independent thread which calls the control program. This allows a pseudo-concurrent execution of the started threads, each one of them handling both the execution of the demanded control program and the communication of the result to the client (plant).

In Ubuntu or Linux, POSIX threads are supported and implemented through the *pthread* library, whereas in Windows (a non-POSIX environment) …

3.7 Real-Time Clock

A critical aspect when measuring times in general is the source of the base time units.

Common PCs have an RTC source running even if the power is not available, through a dedicated battery. However, this clock usually has seconds precision, so additional time sources exist and are implemented via the OS when the system is booted. These sources use the RTC to synchronise and rely on other hardware sources to keep track of the time elapsed, as the CPU's clock (that causes the Time Stamp Counter (TSC) of the CPU to increment).

This is not always a desirable source when computing in a time-sensitive scenario (as controlling a physical system), as the frequency of the CPU's clock can be variable depending on the platform used. In these situations, it's imperative to use an external base of time such as an oscillator with a constant frequency and its own power source.

In this thesis this last option was considered but abandoned, as the precision that can be achieved in a common OS is enough (with some precautions, see threads) to make the implementation of such an RTC unjustifiable in the context of this thesis. In a commercial system, though, would be a very good improvement to implement it, at least, server-side.

Other systems have some external and invariant sources of time, such as the High Precision Event Timer present in the Southbridge, but in many cases the use of this timer is not enabled by default or its use is not advised as the overhead to read the time's value is not convenient.

3.8 Git

Even it is not a functional part of the system, it is worth of mention as it is involved in the development process. Git is a version control system for handling variations and changes implemented in the files involved in the project.

It allows developers to keep track of the development of the project, as one can revert, save and recover different versions of the files, as well as handling of collisions and conflicts between changes relating to the same file by different collaborators, provides an easy way to implement features not tested yet and easily turning back to an stable state in case things don't go as expected.

Several online services can be used for both hosting (what is called in the jargon as repository) and managing the project, such as Gitlab, Github and Bitbucket.

For this project, a local repository was used in a Linux environment (Ubuntu) and changes were periodically "pushed" to the online version of the same repository hosted in Gitlab.

3.9 Encryption

Encryption can be used to provide security in MQTT messages by making their content not understandable for anyone not involved in the process. Two different schemes can be used for providing such encryption:

- Symmetric encryption, where both actors in the communication flow use the same key for encryption and decryption (the Pre-Shared Key or PSK).
- Asymmetric encryption uses different keys for the client and the server through the use of certificates.

The former provides a faster encryption/decryption process, but requires both server and client to have access to the same key and, thus, a careful way of sharing this key.

In contrast, the latter is slower but allows the server and client to have different keys. This is desirable as a compromised element cannot reveal the key stored in the other element. It has the disadvantage of the need to generate the certificates, something that is required to the system administrator.

To see further details, see Security in page 57.

Chapter 4 Methodology and implementation

This chapter explains the methods used in the project, as well as the implementation carried out and the technical solutions adopted.

It is divided in subsections covering the several aspects of the project:

- Development methodology for the system
- Development of the algorithms for the EMC in Simulink

4.1 Development methodology – V Model

This section covers the system's life cycle model used as a baseline for planning the project stages. It's englobed into the pre-specified and sequential methods, and it covers the stages from the system's requirements' elicitation to implementation and verification.

The stages of this model can vary depending on the domain of the project (usually software engineering or system engineering).

The main philosophy behind it is the planning of the project into different levels, from a broader and more abstract level to a narrower and more technical implementations, while planning and executing testing and verification stages for each of these levels (see Figure 1).

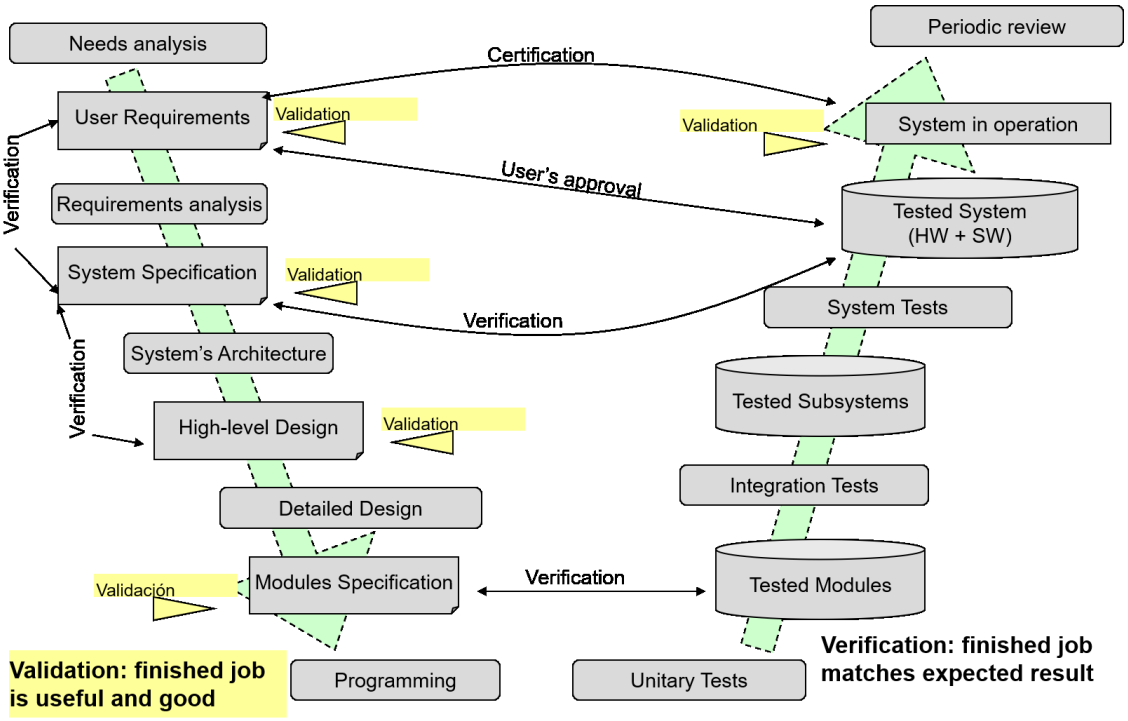


Figure 1 - V Model for System Engineering

This is the model for generic projects. However, a simplified version of the V Model is used in this thesis, as the model above is more well-suited for bigger software or system projects (see Figure 2).

The stages of the V Model used here are:

- System's specification and requirements' elicitation
- Architectural design
- Unitary design
- Unitary test
- Integration and architectural test
- Requirements verification

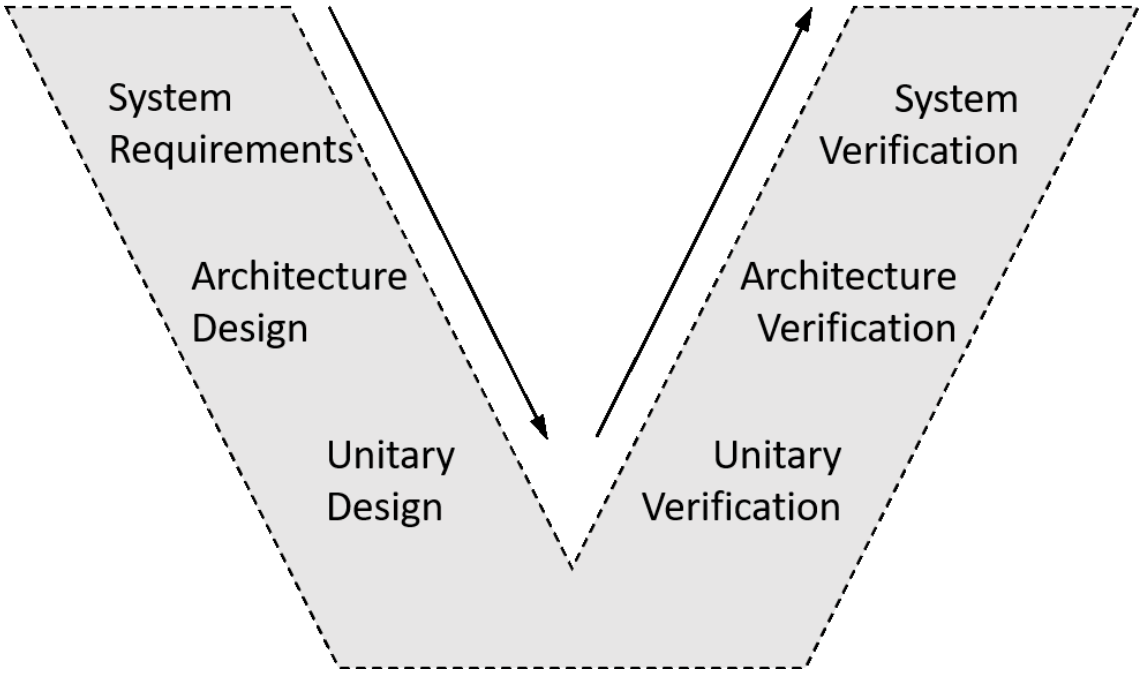


Figure 2 - V Model used in this thesis

As seen from Figure 2, the V Model used starts by gathering the requirements from the stakeholders. This is a crucial stage as changes in requirements in a later stage of the project involve a huge impact in time, costs and the amount of non-reusable elements.

After stablishing all needs and requirements, the next step is to design the high-level architecture of the solution, where components and their interrelations are modelled as high-level elements, focusing on the functions rather than on the implementation.

Once the architecture is defined, it becomes time to design each of the elements (units) in enough detail to allow implementation.

As code of the units is generated, it becomes necessary to verify compliance of each one of the units with the requirements associated to them.

If the unit tests performed return successful results, integration of the several units can be made and, again, the integration has to be tested and evaluated, as functional units can lead to a non-functional or misbehaved system when interrelations between elements happen.

As the last step, once all stages of verification have been successfully finished, the system needs to be verified in order to check that the system not only performs as designed, but also as desired.

To sum things up, the V Model stablishes a hierarchical subdivision of the development of the product, each subdivision being associated with a test and verification stage.

4.2 System's specifications and requirements

These are the requirements to be met by the final developed system, more specific than those enunciated in section 1.3 but aiming to achieve them as a goal.

The architecture, unit and test's design must be done accordingly in order to considerate the system successful.

It should be noted that the choice of words when enunciating the requirements between "must" and "may" is not arbitrary, and have a categorising connotation.

As a summary, the main requirements are:

- Must perform effective control over the Internet of a remote plant
- Must use of the EMC approach for this kind of control scheme
- Must include the implementation of a User Interface, preferably a Web interface
- Must be easy to use for a control engineer, who doesn't have to know anything about the system.
- Must be secure against unauthorised access
- May be scalable to different setups
- May be portable to multiple kind of clients and control strategies

4.3 Architectural design

The chosen architecture comprises a central MQTT broker and a control management program, both on a central server, to where each client[1] can connect. The control management program (CMP) receives the pertinent messages from the broker (and

[1] Users of the control system. This is, for example, a raspberry collecting data from sensors in the plant to be controlled and sending them to the server via MQTT.

previously sent by clients) and redirects them to the particular control program the client needs.

This means that this system can handle several and different plants, and each one of them be controlled using a particular algorithm for that client.

MQTT messages can travel from client to server (and vice versa) through the Internet or other network. As an example, in this thesis a LAN using ethernet and a localhost network were also used.

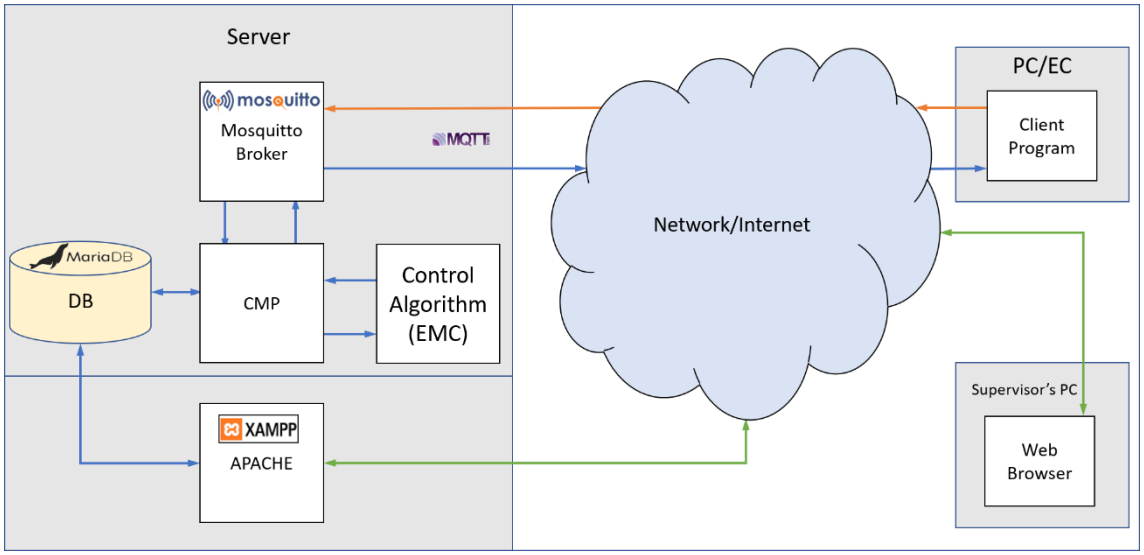


Figure 3 - Architecture of the system

The reason behind understanding both the CMP and Control Program as separate units is to allow execution of several control strategies (Control Programs) implemented as independent units.

4.4 Units

As seen in Figure 3, the most basic elements in this control system are the client program, the Mosquitto broker and the control program. Each one of them was tested in isolation and then tested together, both in localhost and through local and Internet networks. They work together coordinated and managed by the Control Management Program and can be monitored through a webpage interface thanks to the XAMP combo (see section 3.3).

Control Program

Performs the control action based on the information supplied by the rest of the system by computing the command law.

Control Management Program

This unit is in charge of reading the MQTT messages coming from the clients and to redirect them into the Control Program in order to fetch the results and send them to the correct plant.

Server

The Control Management Program, the Control Program and the MQTT Broker exist in the server, which is the frame in which these programs can run. Because of that, the server is thought as a unit itself.

It should be secure, reliable enough (a temporal disconnection of the server can be potentially harmful for systems being controlled) and have enough computation power to handle all the elements hosted inside, as well as communications from/to clients.

Client Program

Receives commands from the server (sent by the Control Management Program) and produces the actuator's signals. It also measures data from sensors and sends them to the server (as inputs for the Control Program).

Client[2]

The unit that contains the Client Program and the environment in order to run it. It must have Internet access.

[2] Many times in this thesis, the term client is used to refer to both the client and the client program (running inside the client) when using it in a context of opposition to the server. In cases which differences between them are significant, both terms are used.

Web Interface

As one of the requirements states that an easy-access interface must be implemented for a control engineer without deep knowledge of background behind this project, a web interface was thought as it doesn't require a specific setup and is easily accessed everywhere.

It must provide access to the different control programs used to allow modifications on the algorithms used, as well as to a database containing the logs of the relevant variables of the control loop (e.g. reference, timestamp, command signal, output of the system…).

4.5 Implementation of the unit elements

The CMP and the client programs are written in C, to exploit the existing libraries for MQTT using Mosquitto as well as being easily compiled in many environments. It is also used due to the easy implementation in the Raspberry, as a C compiler (GCC) is already present by default and the whole process is fully supported.

The control programs, called by the CMP for each particular client, can be written in any language as long as they accept data from the CMP and successfully answer in the format required. That is, a series of arguments passed with a call to the control program, and results returned by the control program to the standard output. This way, the CMP can read the resulting commands from the control program and return them to the plant.

For the Web interface, a webpage was made using Wordpress, and it contains several PHP scripts which can access the control database for data collection and show the results in the web browser, as well as a basic editor for editing and compiling control programs written in C and hosted in the central server.

4.5.1 Control Program – PID

As the EMC was a new technique for the author, a standard discrete PID (Proportional, Integral and Derivative) was implemented in C to ensure the correct communications between the MQTT client and the program itself in the testing phase.

This simple PID was modelled to control a liquid deposit (see Figure 4) with a hole on the wall near its base, which can be filled by a commanded input caudal.

The equations that model this plant are:

$$A\frac{\partial h}{\partial t} = Q_{in} - Q_{out}$$

$$Q_{out} = A_s\sqrt{2gh}$$

With h as the level of liquid (altitude), A the area of the surface of the liquid, A_s the cross section of the exit hole and g the gravity (the second equation derives from Bernoulli's principle).

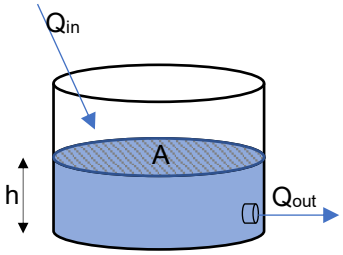


Figure 4 - Deposit model

As for the controller, a discrete PID was implemented both in Simulink and in C:

$$e(t) = h_{ref}(t) - h(t)$$

$$e_d(t) = \frac{e(t) - e(t - \Delta T)}{\Delta T}$$

$$e_i(t) = e_i(t - \Delta T) + e(t) \cdot \Delta T$$

$$u(t) = k \cdot [e(t) + k_d e_d(t) + k_i e_i(t)]$$

With $\Delta T = t - t_{prev}$ being the time difference between 2 samples.

4.5.2 Control Program – EMC

In this thesis, 2 different Embedded Models were used for simulation of 2 different plants:

1. A motorised shaft moving a decentred tool (lathe, Figure 5)
2. A simple punctual mass (Figure 8)

The two EMC controllers (for the lathe and punctual mass) were implemented by Simulink modelling and later generating C code.

The first model, taken directly from [5], represents a shaft driven by an electric motor through a gearbox with a decentred punctual mass modelling a tool as shown in Figure 5.

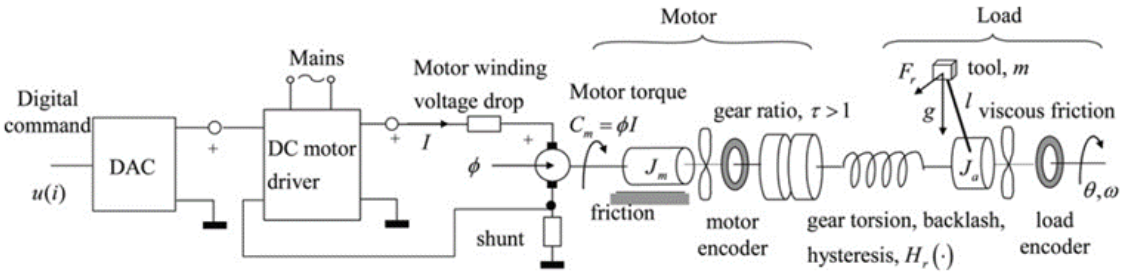


Figure 5 - Model used for EMC simulations with fixed sampling time (source: [5])

This model was used to test in first place that the EMC works over the Internet.

The Embedded Model used is (taken directly from [5]):

$$A_c = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \qquad H_c = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \qquad A_d = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \Longrightarrow A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \qquad G_c = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \qquad G_d = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \Longrightarrow G = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$C = (1 \quad 0 \quad 0 \quad 0)$$

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_c \\ \boldsymbol{x}_d \end{pmatrix}, \qquad \boldsymbol{x}_c = \begin{pmatrix} \theta \\ \nu \end{pmatrix}, \qquad \boldsymbol{x}_d = \begin{pmatrix} a \\ s \end{pmatrix}$$

$$J(i) = J_m \tau^2 + J_a + lm^2, \qquad a_g(i) = \frac{mgl\cos\big(\theta(i)\big)T^2}{J(i)}$$

$$b(i) = \frac{\phi \tau T^2}{J(i)}$$

$$u(i) = b(i)I(i) - a_g(i) - a_r(i)$$

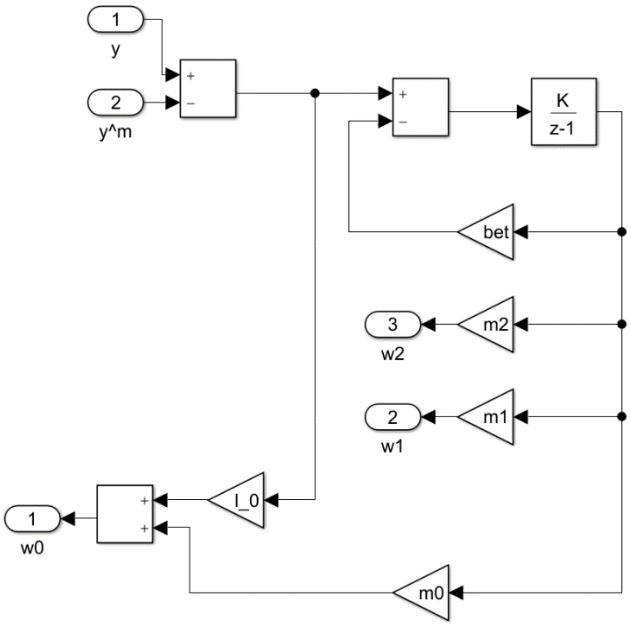


Figure 6 - Simulink's block diagram of the noise estimator for the lather model

As seen in Figure 6, the noise estimator takes the difference between the real plant's output and the output estimated in the EM and produces the $\boldsymbol{w}$ vector[3].

The equations of the noise estimator implemented, in z-transform and matrix form are:

$$\boldsymbol{w}(z) = (y(z) - \hat{y}_m(z))\left[\frac{1}{z-1+\beta}\begin{pmatrix} m_0 \\ m_1 \\ m_2 \end{pmatrix} + \begin{pmatrix} l_0 \\ 0 \\ 0 \end{pmatrix}\right]$$

It should be noted that, while the next model uses an static observer as noise estimator, in the lathe example taken from [5] a dynamic observer is needed as there are 2 controllable states but only direct measurement is taken from the plant (position).

The second model used is a simple punctual mass moving unidirectionally. As shown in the Simulink model (Figure 8), simple integrators simulate the movement of the particle, and a simple EM simulates this behaviour. The reasons to keep this model simple is because an improvement over the previously used EMC is applied and a variable sampling time is considered to take into account network delays, therefore, the need to avoid multiple sources of trouble.

The model used for the punctual mass is as follows (see section Embedded Model Control in page 14):

[3] $\boldsymbol{w} = [w0 \quad w1 \quad w2]^T$

$$A = \begin{pmatrix} 1 & T & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix},\ B = \begin{pmatrix} 0 \\ T \\ 0 \end{pmatrix},\ C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix},\ G = \begin{pmatrix} 0 & 0 \\ T & 0 \\ 0 & T \end{pmatrix} \quad 4.1$$

And the states used are:

$$\boldsymbol{x_c} = \begin{pmatrix} position \\ velocity \end{pmatrix}, \qquad \boldsymbol{x} = \begin{pmatrix} \boldsymbol{x_c} \\ x_d \end{pmatrix}$$

Being p_1, p_2 and p_3 the 3 poles of the Noise Estimator, its equation is:

$$\boldsymbol{w} = L\boldsymbol{e_m}, \qquad L = \begin{pmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \end{pmatrix}, \qquad \boldsymbol{e_m} = \boldsymbol{x} - \boldsymbol{x_m}, \qquad \boldsymbol{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$$

With:

4.2

$$l_{11} = \frac{a_3 + 2Tl_{12} - 3}{T^2}$$

$$l_{12} = \frac{a_2 + 3}{T},$$

$$l_{21} = \frac{a_3 + a_4 + Tl_{12} - 2}{T^2}$$

$$l_{22} = 0$$

$$\lambda^3 + \lambda^2 a_2 + \lambda a_3 + a_4 = (\lambda - p_1)(\lambda - p_2)(\lambda - p_3)$$

Regarding the command law generator, the matrix K is computed with the pole placement procedure as in the noise estimator (see Command Law generator in page 17)

$$k_1 = \frac{a_3 + Tk_2 - 1}{T^2},\ k_2 = \frac{a_2 + 2}{T},\ K = (k_1 \quad k_2) \quad 4.3$$

$$\lambda^2 + \lambda a_2 + a_3 = (\lambda - p_1)(\lambda - p_2)$$

$$M = \frac{1}{T}, \qquad Q = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

In the punctual mass case, the noise estimator is much a simpler static state observer. This is because two states are measured (position and velocity) instead of just the position (first example), eliminating the need of a dynamic component.

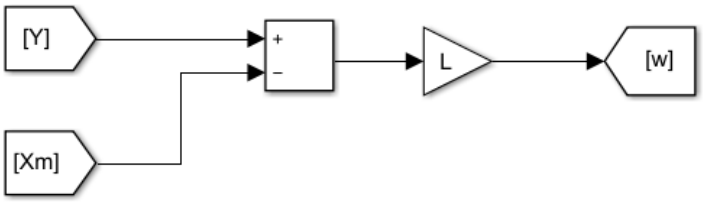


Figure 7 - Simulink's block diagram of the noise estimator for the punctual mass

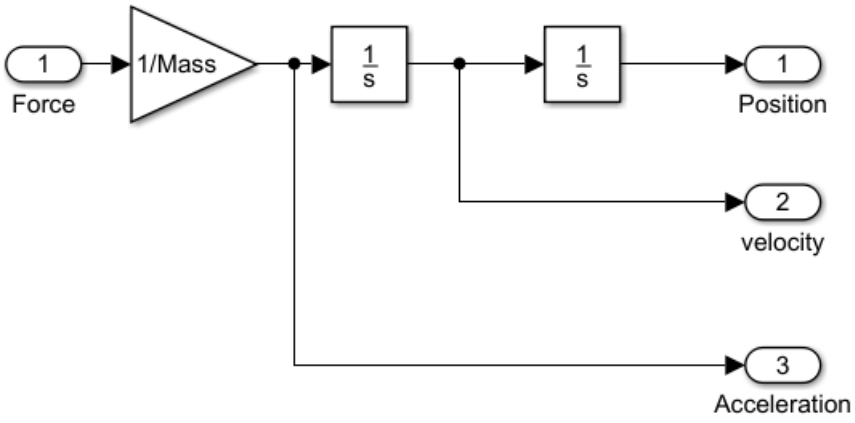


Figure 8 - Simulated plant

In order to generate the code, Simulink Coder and Matlab Embedded Coder were used. After that, the output files can then be easily modified and integrated into the system by adding the MQTT and MySQL functionalities.

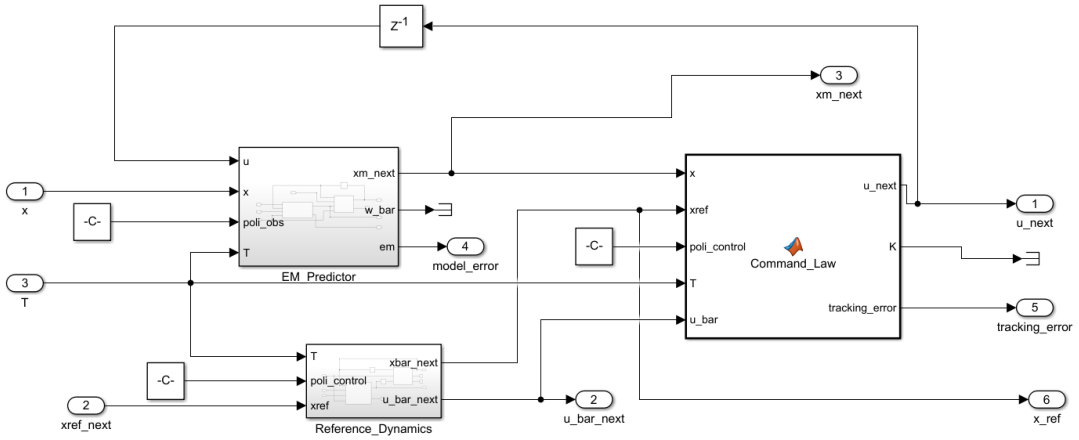


Figure 9 - EMC Controller for the punctual mass

The values chosen for the poles in the implementation were:

Element	Discrete eigenvalues
L (Noise Estimator)	0.9, 0.9, 0.9
K (Command law)	0.5, 0.3
K (Reference dynamics)	0.1, 0.2

Table 1 - Control parameters for the punctual mass model

The basic steps followed by the EMC program are:

1. Initialisation of the MQTT interface
2. Initialisation of the Embedded Model variables
3. Control loop
    a. After measurements reception, execution of one simulation's step
    b. Sending of results back to the client
4. Libraries clean-up and termination of execution

4.5.3 Server

One of the elements used for development was a Raspberry Pi Model 3B+. This is a low-cost ARM-based computer, which supports Linux, Windows IoT Core and several other

non-demanding OSs. In this project, Raspbian, a Debian version for the Raspberry Pi, was used in a headless[4] configuration.

The main purposes of the Raspberry were:

- Act as server when connection to Politecnico di Torino's server is down and for local tests, during development stage.
- Act as one possible client, measuring the real plant and retrieving commands from the server to apply them via the actuators, during ¿production?

As the development of the project went forward, it was discovered that the capabilities of the Raspberry used (model 3B+) allows a complete simulation of an EMC algorithm so, in addition to the signals received from the central server, a whole control algorithm can be also loaded into the client. This is particularly useful in case of a disconnection or service interruption, as allows the plant to remain stable and controlled in a particular setpoint or even to perform a controlled shutdown of the whole system if the particular plant and situation demands it.

The other solution for the server, and the final choice due to the distance, was a PC running Windows 10 in the Politecnico di Torino (WAMP stack). For communication, FTP[5] was used to upload files while Web interaction was used for starting the programs by using several PHP scripts.

The MQTT broker chosen is the Mosquitto implementation, as it provides C libraries and an easy way to implement without previous expertise. It can be installed both on Windows and Linux, and is easily configured through several files hosted in the server.

The main configuration file is called "mosquitto.conf", and it contains all the configuration options used by the broker. If it doesn't exist, default parameters apply. However, fine tuning of the several options is strongly recommended as it can avoid security breaches, loss of performance.

[4] Without UI, the computer is handled via terminal. While having a less appealing look, resources usage and consumption are drastically reduced.

[5] File Transfer Protocol

Parameters used in this project are collected in the table below:

Option	Value	Description
sys_interval	10	Interval in seconds between broker $SYS publications.
pid_file	/home/pi/mosquitto.pid	In Linux, file to store the broker's process identificatory when mosquitto is called as a daemon.
max_inflight_messages	1	Maximum number of messages being processed by the broker simultaneously. A value of 1 guarantees in-order delivery.
message_size_limit	1024	Maximum size in bytes for each message.
set_tcp_nodelay	true	Disable's Nagle's algorithm
per_listener_settings	true	Allows to define settings of each listener separately.
log_dest	stderr	Destination of log messages (if enabled).

Table 2 - Broker's parameters

The other files used are:

Access Control List (ACL), defines access policies to clients connected to the broker, to allow or deny publication/subscription to topics. The implementation used is to assign a different name to each client, allowing it to subscribe and publish only in its relevant topics to avoid interference.

PSK file, contains the PSK and identification for using a ciphered connection. Each client can have its own key, or different keys according to the kind of user can be defined.

Password file, contains usernames and passwords[6], which are then used to grant access to each client.

As a summary, each client has its own username with a password associated, and there are different keys to provide TLS encryption and connect to the broker.

4.5.4 Control Management Program

Written in C, the CMP consists in an endless loop. When MQTT messages containing measurements from the plant arrive, a callback[7] function that makes the call to the Control Program (compiled apart or not) is executed, results are collected and information is sent to the client.

Two versions of the program were made: one for Windows and another one for Linux, this last one having also the possibility of executing the callback function as an independent thread.

This threaded functionality allows the CMP to delegate the task of calling the control program to independent threads, so the CMP can continue listening for MQTT packets and allows multiple control loops (this have to be carefully designed, as real-time requirements for all the control loops involved have to be reached).

[6] PSK file and Password file are not equal. The first one refers to the keys used for encryption/decryption while the second one contains passwords to allow clients to use certain usernames (required by the ACL for granting privileges)

[7] A function executed when a certain trigger event happens (such as receiving a message)

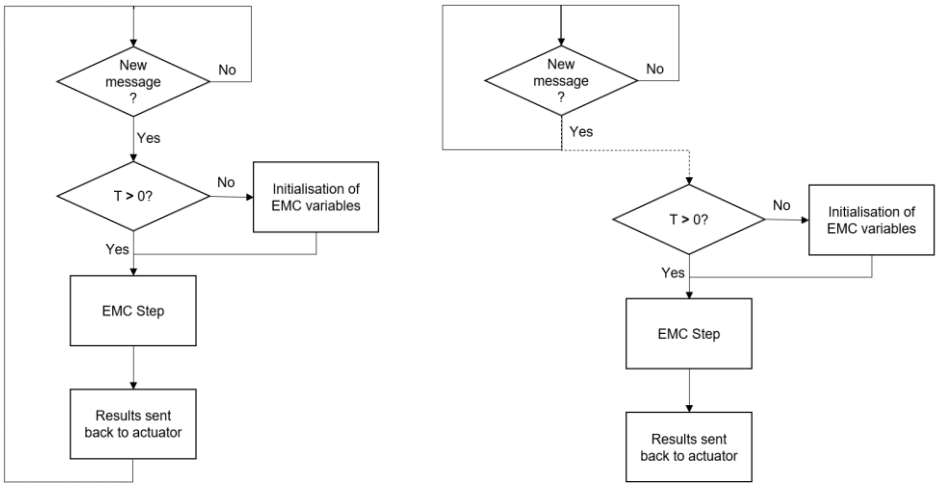


Figure 10 - CMP's flow diagrams: Blocking (left) and threaded (right) versions

4.5.5 Client

In all simulations, the client program is implemented as a Simulink subsystem and interacts with the model of the plant.

In further development, a C implementation of a client program would require MQTT support and access to signals in concordance with the particular needs of the case.

4.5.6 Web Interface

The Webpage created and hosted in the server (thanks to Apache) was created with Wordpress, which provides a quick and easy way to implement a functional and visually appealing web.

The starting point is an already existing theme[8] which was tuned to serve the purposes of this project by adding custom "shortcodes[9]" to handle database access through Wordpress' PHP/MariaDB platform.

It must be noted that this web interface is the major security weakness of the system, as access to the web is handled over HTTP and not HTTPS. Possible future solutions to this is proposed in Security, page 57.

[8] "Twentyseventeen"

[9] PHP code snippets called executed within a wordpress page when loading

From the resulting test, the control engineer can login to see the control parameters and modify the control programs via an online editor. Recompilation is executed in the server by just clicking on the corresponding button.

4.6 Tests

Simulations of Control Programs

As a first testing stage, correct function of the control programs (PID and EMC) had to be tested by simulation in Simulink.

The objective is to test correct mathematical behaviour by carrying a Model-In-the-Loop simulation.

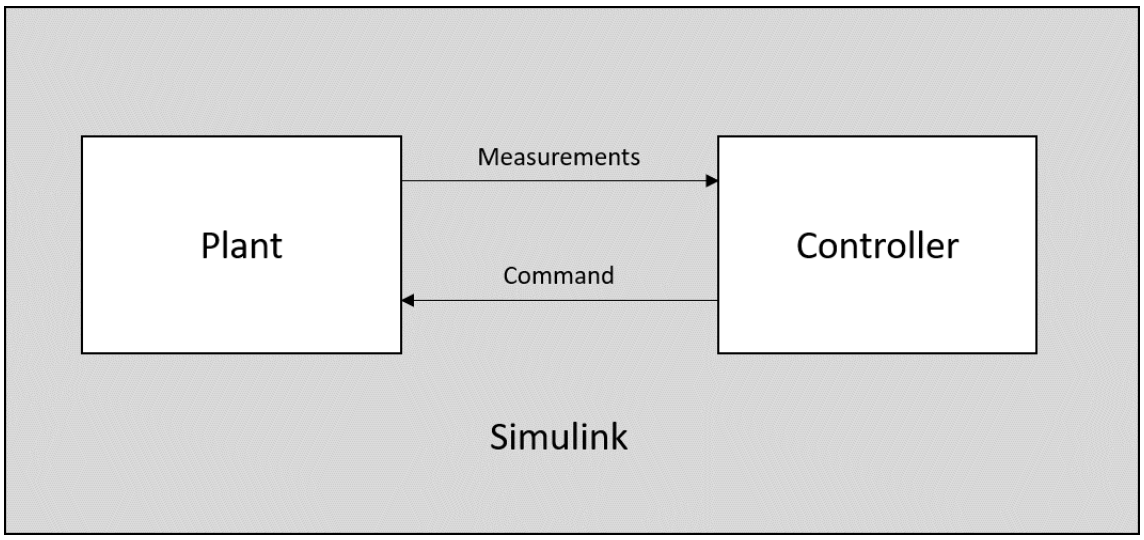


Figure 11 - MIL Tests architecture

MQTT

MQTT communications were tested unitarily by accessing the broker through the command line tools included in the Mosquitto installation. Message receptions were checked and later a stress test was also conducted. In this situation, the communications are deliberatively saturated to see where the performance limits are (which are dependent on the hardware used and should be done if a different setup is used).

The stress test was performed by a simple C program which acts as client and have a loop sending as many packets as possible:

Results on the broker's throughput (as a moving average of number publications received per minute) was obtained through the broker's system publications, in particular by subscribing to the topics "$SYS/broker/load/publish/sent/1min" and "$SYS/broker/load/publish/received/1min":

Setup	Maximum throughput (Msgs/min)
Localhost	7.8×10^6
Server in Raspberry	1.3×10^6
Server in Politecnico di Torino	2.7×10^6

Table 3 - Maximum throughput imposed by MQTT broker

Other test carried on the network is the measurement of the round-trip time of a message going to the broker and returning to the client. The message carries its generation's timestamp, and its difference with the arrival timestamp (client side) is computed.

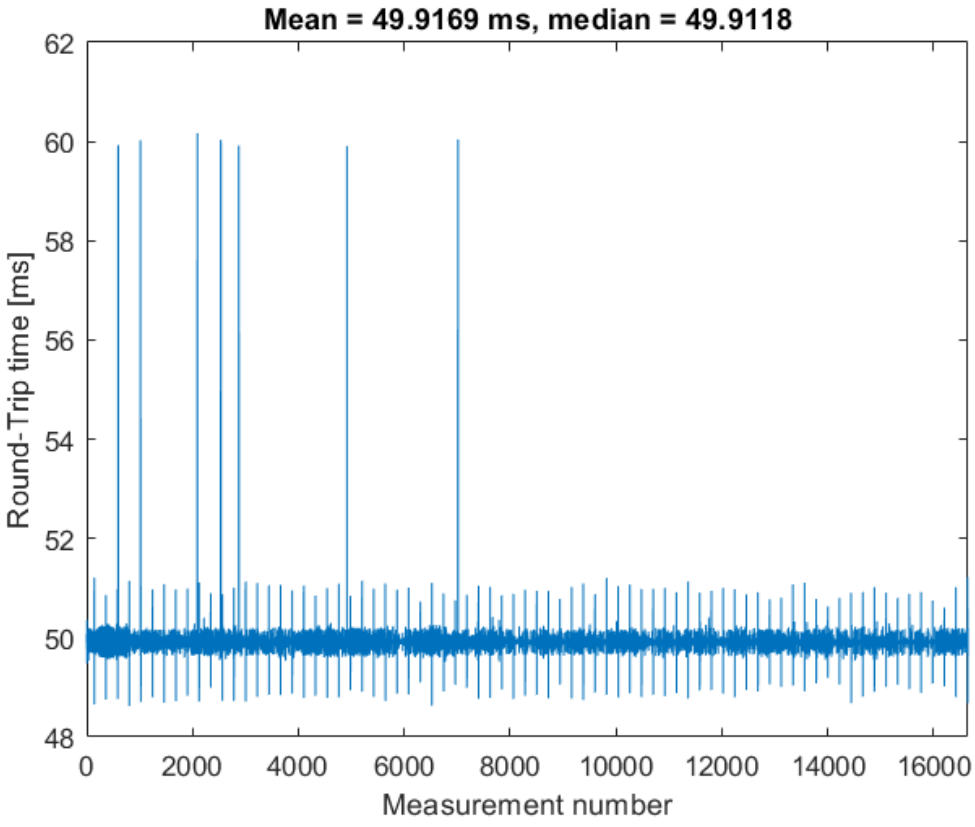


Figure 12 - Round-Trip delays with Raspberry Pi

As it can be seen from figure above, delays with the Raspberry acting as server are very likely to be around 50ms. After plotting a histogram of the measurements, it can be seen

that the cumulative density function and the probability density function are well approximated by a student's t distribution[10] with parameters (see figures 13 and 14):

$$\mu = 49.9113, \qquad \sigma = 0.0666285, \qquad \nu = 2.06408$$

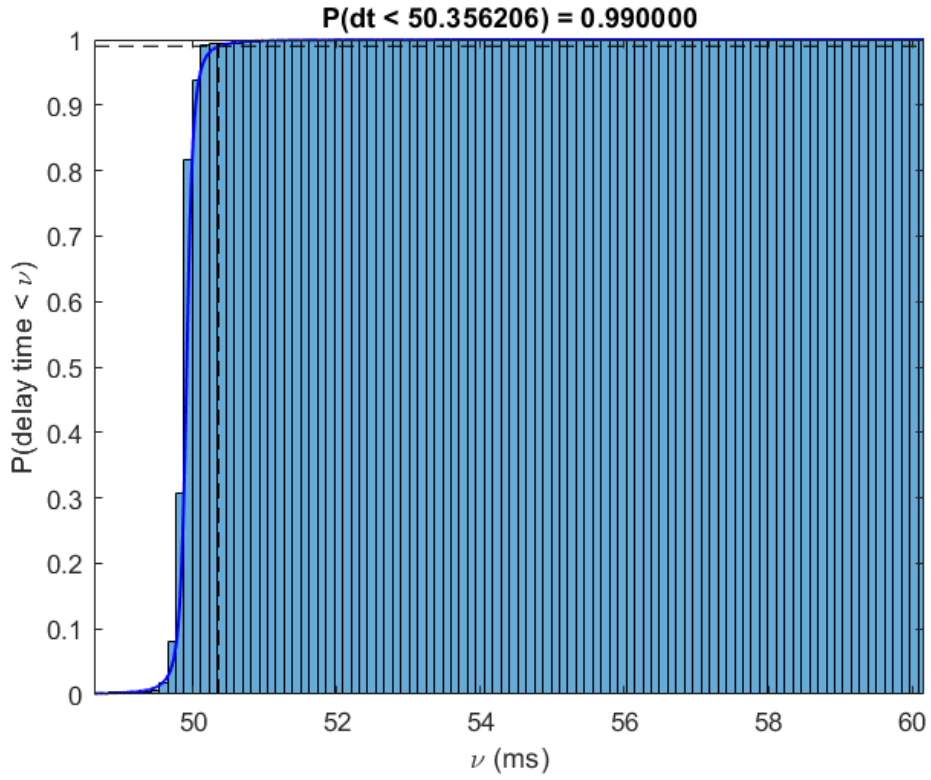


Figure 13 - Probability Density Functions for Raspberry (Student's t approximation in continuous blue line)

[10] Distribution approximated by Matlab's `fitdist()` function

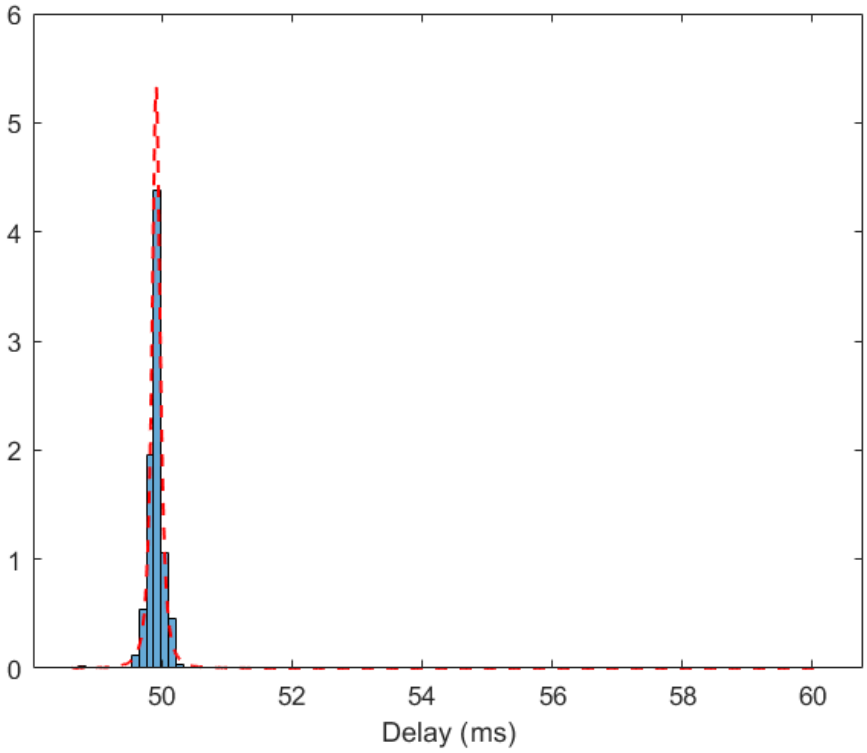


Figure 14 - Probability Density Functions for Raspberry (Student's t approximation in red dashed line)

The situation when the server at Politecnico di Torino is used is quite different, as values are more distributed but no bilaterally, as seen in Figure 15.

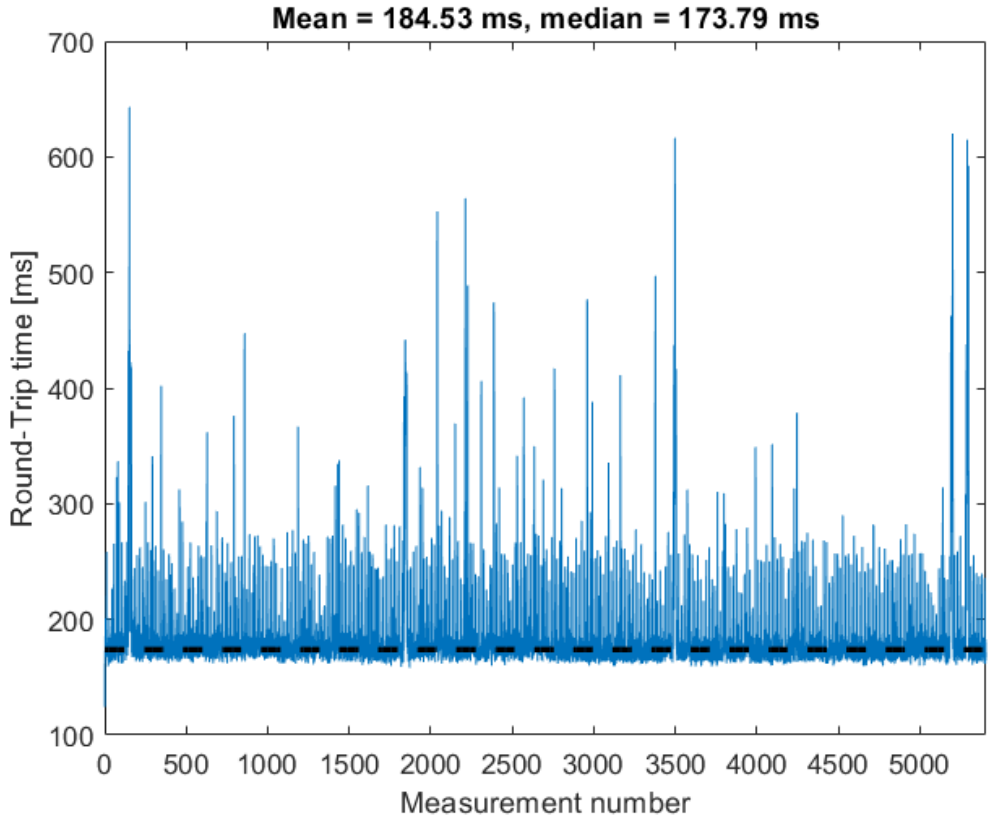


Figure 15 - Round-Trip delays with Politecnico di Torino's server

In this case, the Student's t distribution is not as good as with the Raspberry, as it can be seen in figures below, where a Student's t distribution with parameters below is the best approximation obtained:

$$\mu = 172.614, \qquad \sigma = 6.32223, \qquad \nu = 1.24533$$

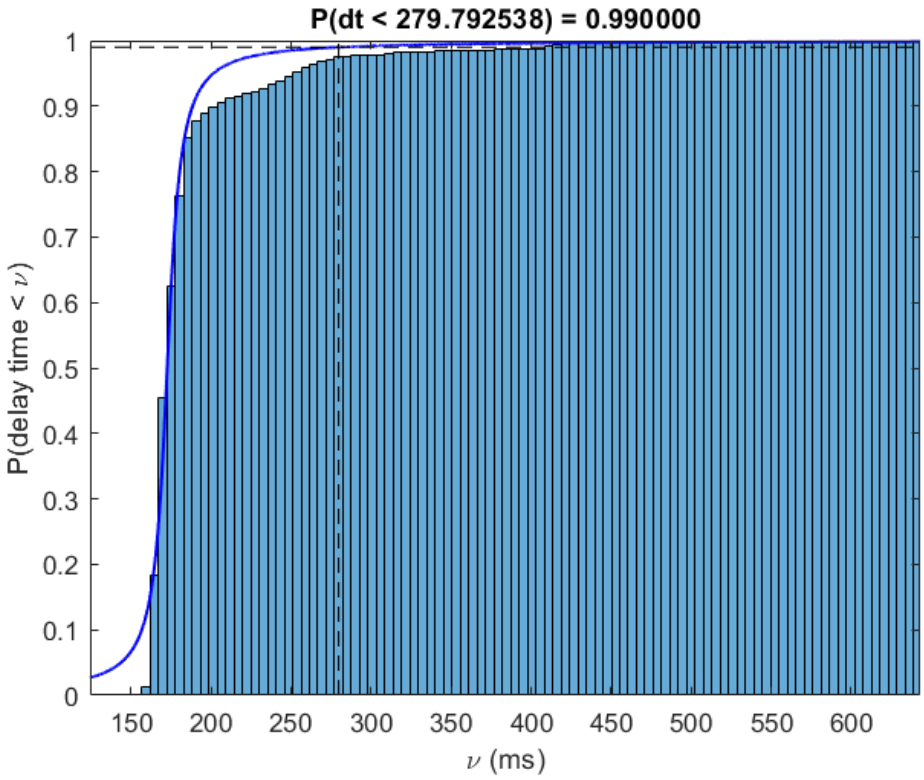


Figure 16 - Cumulative Probability Functions for Politecnico di Torino's server (Student's t approximation in continuous blue line)

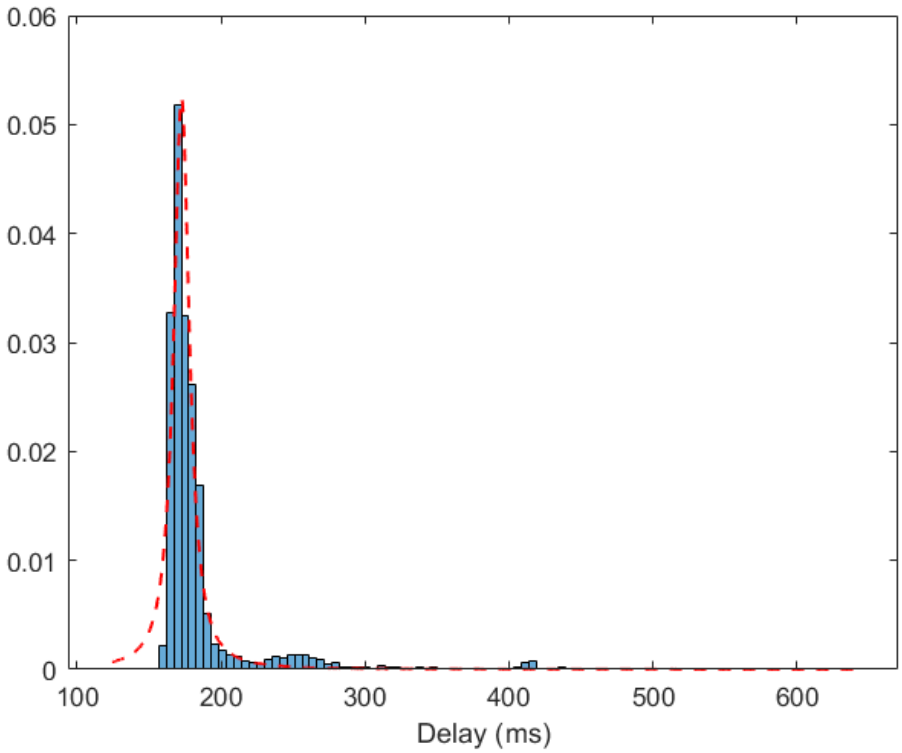


Figure 17 - Probability Density Functions for Politecnico di Torino's server (Student's t approximation in red dashed line)

Integration tests

For testing the MQTT's integration with Simulink and the controller, a simple[11] test model was used as a first approach. This was done to avoid failures due to a badly tuned controller or simulated plant and narrow down the source of possible errors to the MQTT communication.

In early tests, big values are taken in order to guarantee that ΔT remains constant and then, after successful results as shown in Figure 18, decreasing control times were gradually used to check the performance limits.

Derived from the delay times measurements, connection with Politecnico di Torino's server has a round trip time with a median of 173 ms. Adding a margin due to computations of the model and other processes involved, nominal control times of 500 ms are being considered. This can be further improved by using future better connections such as 5G, reducing distance between plant and controller and using dedicated networks (on the ethernet network delays are around 50 ms)

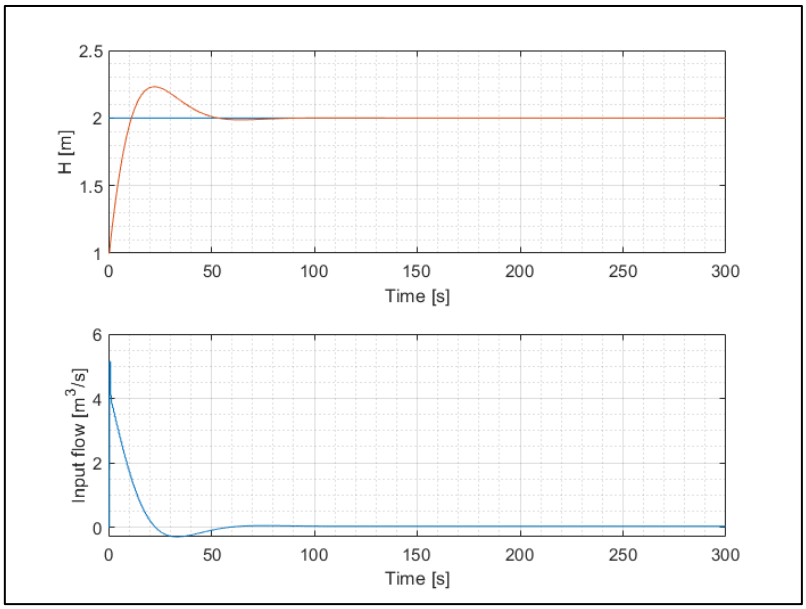


Figure 18 - PID-controlled response

[11] Althoug simple, it is still a non-linear system

Plant and external[12] EMC assembly

With MQTT communications and both plant and controller working successfully, the next testing phase is the one related to the system integration.

This was done by sequentially deploying the EMC controller, first in localhost (Software-In-the-Loop, Figure 19), then involving a “simple” network as ethernet to involve 2 separated machines (Figure 20)

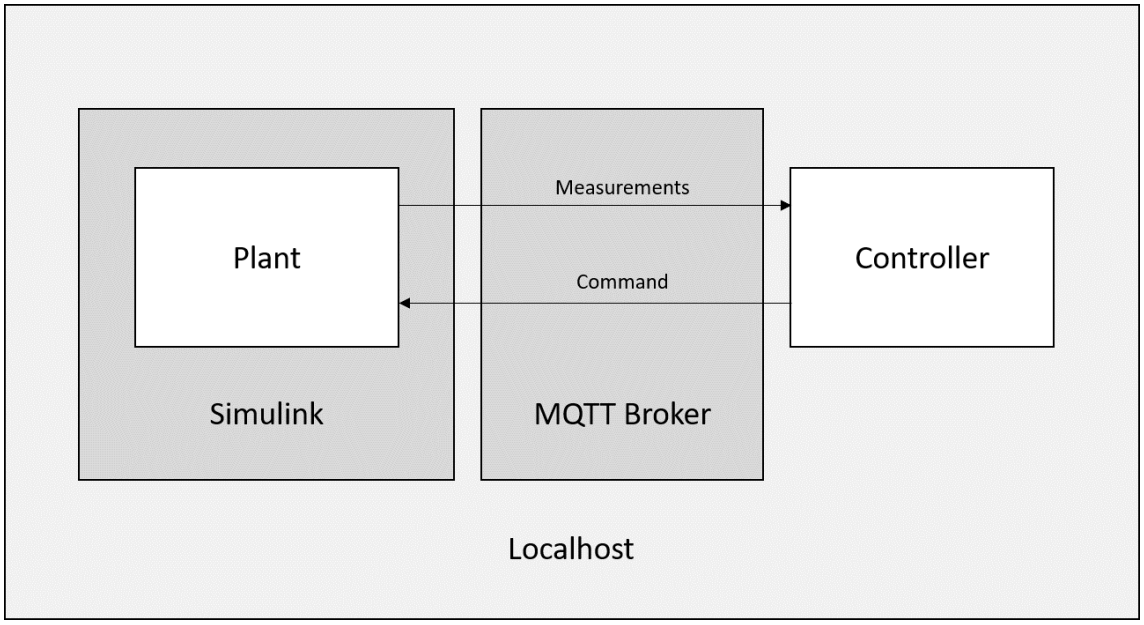


Figure 19 – SIL Tests in localhost architecture

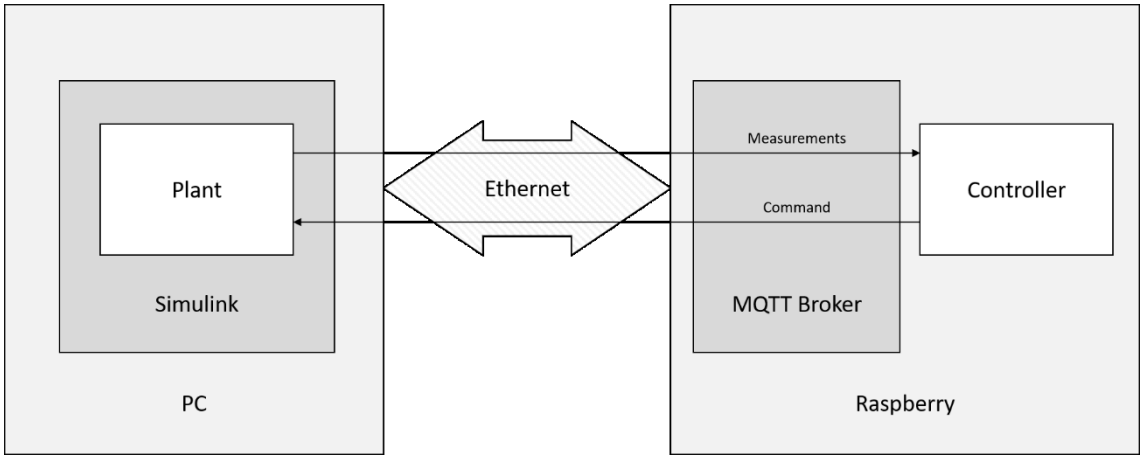


Figure 20 - Tests over Ethernet

[12] Please note that external refers to the controller not being simulated in Simulink along the plant (i.e. using MQTT)

Chapter 5 EMC with variable sampling time

This chapter deals with the final objective of this project: a functional implementation and feasibility verification of the asynchronous version of the EMC technique.

The system developed as shown in Chapter 4 was used, after test were passed, to implement the idea of the asynchronous EMC. Starting from the point mass model, a variable sampling time is applied, as in an NCS time delays are usually not possible to keep at a constant value (particularly true if the Internet is involved).

The main idea behind this modification is to compute the values of the elements of the EMC method at each iteration step. This allows to follow different strategies:

- Keep the discrete eigenvalues (of the model, noise estimator and reference dynamics) inside the stability circle and fixed to the desired values, thus allowing to always ensure stability of the system in the disturbing conditions provoked by the Networked nature of itself.
- Keep the continuous eigenvalues to keep dominant frequencies of the system fixed and, therefore, the frequencies range that the system will be able to follow.
- A mixed strategy combining the two mentioned before. This means fixing the continuous eigenvalues to hold the system's characteristic frequency up to a certain value of the sampling time. After this critical sampling time, discrete poles are fixed and so does the stability, in detriment of the system performance.

5.1 Control sampling time

In the previous chapter, discretisation of the plant is done assuming a constant sampling rate, so the controller receives measurements from the plant at regular intervals. This is usually very difficult to guarantee, as seen in section 2.1.1, and this chapter deals with the case in which is not constant at all. Therefore, instead of speaking about a control sampling time T, from now it will be T_i.

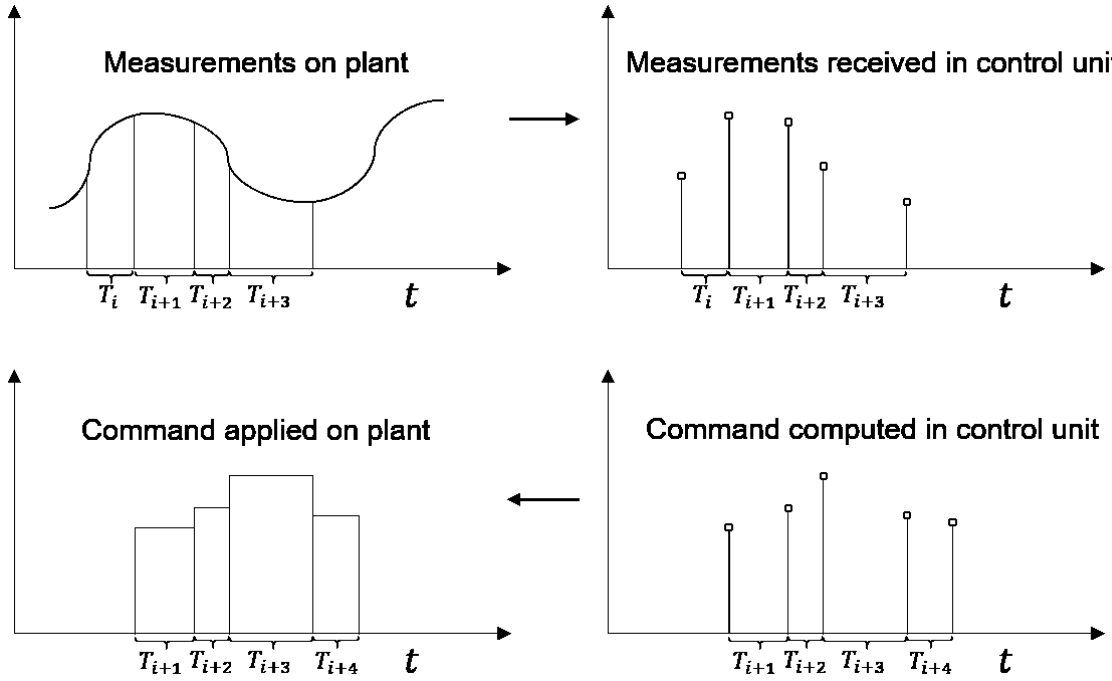


Figure 21 - Variable sampling and hold concept

5.2 Tracking and frequency

When dealing with a discrete system, where sampling is being performed at a certain rate, this rate has to be fast enough to allow the controller to follow the measured signal and generate a sufficiently fast command law. If sampling is done at a lower frequency, information is lost and the signal cannot be successfully discretised and controlled.

The rate is called the Nyquist rate, and is equal to twice the highest frequency of the signal (and in ultimate case, dynamics) to be estimated or worked with.

This introduces a stability condition that restricts the sampling or control time in the form:

$$T_i < \frac{1}{2f}, \ \forall i \qquad 5.1$$

This can be a challenging issue due to the variable delays in an NCS, but as the approach being considered updates the model with the measured T_i, the EMC is able to keep the value of the frequency limits by adjusting the frequencies of the system (as they rely on the continuous eigenvalues [12]) accordingly and ensuring they fulfil the Nyquist criterion 5.1.

As the eigenvalues are being placed taking into account the variable T, it must be noted that the frequencies of the system, as they change, limit the frequency of the reference that the system will be able to follow.

In this sense, the EMC controller act as a low-pass filter, so a maximum rate of variation for the reference has to be considered when running this control strategy, anything over that frequency being dumped by the controlled system itself.

This problem can be solved by fixing the continuous eigenvalues and, thus, fixing the frequencies of the system. However, as it can be seen from equation 5.2, this has the effect of modify the discrete eigenvalues and, by doing so, the risk of entering instability needs to be considered (see section 2.2.4)

5.3 Implementation

To build this modification, a small variation has to be done over the designed system (see section 4.5, “Implementation of the unit elements”).

On the Simulink blocks from the punctual mass model, each element that depends on T_i for its design needs to be fed with the measured T_i as input. This input is computed as each measurement received from the plant contains the timestamp at the moment of sensor reading, so the difference with the previous measurement is easily computed.

This affects the following blocks:

- Embedded Model
- Noise Estimator
- Command Law generator

Recalling the model’s equations (expressions 4.1 for the matrices of the EM, equations 4.2 for the NE and 4.3 for the K computation), they need to be recomputed at each control step, so the mathematical expressions for the elements involved are:

$$A_i = \begin{pmatrix} 1 & T_i & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \qquad B_i = \begin{pmatrix} 0 \\ T_i \\ 0 \end{pmatrix}, \qquad C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \qquad G_i = \begin{pmatrix} 0 & 0 \\ T_i & 0 \\ 0 & T_i \end{pmatrix}$$

With

$$T_i = t_i - t_{i-1}$$

With t_i being the timestamp of the last received measurement and t_{i-1} the timestamp of the previous measurement.

The feedback gain becomes

$$k_{i,1} = \frac{a_3 + Tk_2 - 1}{T^2}, \qquad k_{i,2} = \frac{a_2 + 2}{T}, \qquad K_i = (k_{i,1} \quad k_{i,2})$$

$$\lambda^2 + \lambda a_2 + a_3 = (\lambda - p_1)(\lambda - p_2)$$

$$M_i = \frac{1}{T_i}, \qquad Q_i = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

And the Noise Estimator changes as well:

$$\boldsymbol{w} = L\boldsymbol{e_m}, \qquad L = \begin{pmatrix} l_{i,11} & l_{i,12} \\ l_{i,21} & l_{i,22} \end{pmatrix}, \qquad \boldsymbol{e_m} = \boldsymbol{x} - \boldsymbol{x_m}, \qquad \boldsymbol{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$$

With:

$$l_{i,11} = \frac{a_3 + 2T_i l_{i,12} - 3}{T_i^2}$$

$$l_{i,12} = \frac{a_2 + 3}{T_i},$$

$$l_{i,21} = \frac{a_3 + a_4 + T_i l_{i,12} - 2}{T_i^2}$$

$$l_{i,22} = 0$$

$$\lambda^3 + \lambda^2 a_2 + \lambda a_3 + a_4 = (\lambda - p_1)(\lambda - p_2)(\lambda - p_3)$$

As for the strategies exposed at the beginning of the chapter, it can be chosen to define fixed discrete eigenvalues λ_i and to use them in expressions above.

Otherwise, fixed continuous eigenvalues μ_i lead to variable discrete eigenvalues λ_i, both related by the expression

$$\lambda_i = e^{\mu_i T_i} \qquad 5.2$$

and computations using those discrete eigenvalues are still the same.

Chapter 6 Troubles confronted

As several aspects and disciplines are combined in this thesis, a wide variety and nature of troubles was found, these are organized by the topic they belong to.

It's necessary to remark that while many of these problems were addressed and solved, some of them weren't. This is mainly because doing so would have required a significatively deeper understanding of the subject out of the scope of the thesis.

6.1 Control Theory issues

Despite including a variation of the EMC to handle different control times, it can be noted the difference in performance of the same controller depending on how the communications are done.

6.2 MQTT issues

Despite the huge number of packets that the Broker can send, there's a brief delay between the moment a message arrives from a publisher until the publisher resend it to the subscribers. This delay has not been measured due to the fact that doing so would require to modify the Broker's source code way beyond the scope of this thesis, but it has been included as part of the measured delays from the Internet.

Another difficulty found was the fact that Mosquitto is, for the moment, a very interesting tool with many capabilities as long as you stay in Linux. In Windows, although almost all functionalities are available, some characteristics as the threaded interface or websockets compatibility are not. In the end, two different versions can be built depending on the application's target (cross-platforming becomes more difficult to implement), as opposed to the more desirable objective of achieving a single code which leads to different implementations by using different compilers.

Another difficulty came up during the stress test (see MQTT tests on page 44), as when a huge number of messages is being sent, the queue associated to the client reaches

its maximum size and the client gets silently[1] disconnected. This is improved in MQTT v5.0, which was accepted as an OASIS standard during the final development of this project and was not implemented to avoid possible incompatibilities with the rest of the system.

6.3 Security

Depending on the particular system to be controlled, several aspects as who is allowed to access the system, to listen to the communication traffic or even to act as a client may arise and gain importance.

For this reason, the standard functionality of the developed system uses encrypted MQTT messages using TLS built-in support from the Mosquitto package, as well as an Access Control List which defines which users are allowed to read or write in each topic.

This means: when a client connects to server via MQTT, it must encrypt the traffic using a Pre-Shared Key (symmetric encryption[2]) and send the username and password as identification for topic access.

Definition of access policies as well as new users can be easily performed within the configuration files (ACLd and msqttpsswrd respectively).

This behaviour can be disabled in certain systems where it's not a problem if a third element intercepts or interacts with communications, but this is not recommended by default.

The elements that a new client needs in order to connect and use the control service are chosen to be distributed as binaries already compiled for the target platform for the sake of simplicity and to avoid the end-user to modify system's security, so a system administrator figure is needed to modify client-dependant data in the source codes and perform the custom build. This process has been simplified by constraining the elements to be modified to just the CMP's header file and also by using Makefiles for the building process.

---

[1] No message error is received. Only when a new publication is tried from the client, a MOSQ_ERR_NO_CONN code is returned by the C function.

[2] Mosquitto supports TLS by using either a symmetric encryption/decryption with a key known by both client and broker (PSK) or an asymmetric scheme with certificates.

As for the Web service provided for monitoring and controller tuning, restricted areas protected with password are provided. However, due to the fact that connection to the webpage is done via HTTP (which nowadays is considered unsecure due to well documented security issues), this web service is the weakest point from the security point of view. Because of this, if the system is going to be used in the current state, the controller editor should not be implemented to avoid non-authorised accesses (as credentials for login are sent in plain text, they are very vulnerable to be read by unauthorised third-parties).

In a future, HTTPS should be implemented for accessing the webpage, and the editor could be securely used. In order to do this, a domain must be obtained instead of using plain IP of the server.

With the domain, a certificate[3] can be linked to it and, after proper configuration of the Apache server, HTTPS can be implemented.

6.4 Compilation and dependencies

As the programs created for this project rely on some non-standard C libraries, several dependencies and libraries have to be previously installed. It is important to pay attention to what kind of device is going to be the target of the compilation process as, depending on that, those dependencies may or not be compatible.

A solution is to statically link those libraries (and cross-compile the programs) not present in the target machine, but this has to be handled carefully because those libraries can be using other dependencies as well, so tools for handling dependencies are strongly recommended in those cases to avoid complexity on dependency-tracking labours.

In particular, external dependencies are:

- Libmosquitto (provides MQTT functions)
    - OpenSSL (in case TLS support is needed)
- Connector C (or MariaDB equivalent, for handling data insertion into DB)

[3] A certificate can be issued by a Certificate Authority (CA), or it can even be generated by the administrator of this system (this is a les recommended option as access from a not configured browser could not be granted).

6.5 Network Issues

The main drawback when using Internet is the lack of certainty not only around the delay in the network, but also around the arrival or not of the message, a very dangerous characteristic when dealing with automatic control.

Because of this uncertainty, a conservative approach has been taken and, while under some circumstances the controller's sampling time could be smaller, a safety factor is used to reduce the probability of messages' losses or extremely big delays (see MQTT stress test section on page 44).

Tests were made for measuring packets' delays while going through the network by sending a timestamped message to the broker and back and recording the time at the packet arrival (round-trip time delay).

Also, as the server is a normal PC running Windows 10 and other programs non-related to the project. This provoked several reboots and, therefore, impossibility to connect to the server as the programs needed to be manually started.

Because of this, the major number of tests were carried on the Raspberry, only using the Politecnico's server for final tests.

6.6 Time measurement and Real Time aspects

One of the main aspects when dealing with time measurement in a computer is how does a certain method actually gets the measurement and how reliable is the source of that measurement. Specific measures had to be taken and, despite those measures, it's not always possible to get an accurate reading.

OS like Ubuntu and Windows hide access to hardware by redirecting all desired interactions through the kernel. This is convenient and useful provided that abstraction helps the programmer making the whole process easier and hiding the complexity associated to hardware handling, and it also provides a security layer as the user cannot manipulate the hardware directly, avoiding wrong procedures and, thus, potential malfunctioning of the system or even its destruction.

However, the additional "steps" to be followed when accessing hardware through the kernel makes the process of measurement's acquisition slower (causing deviations from the real measurement) and while hiding the real hardware handling is a desirable

characteristic for non-RT projects, it also means that finding out the primary source of the measurement becomes considerably harder.

In addition, while common microcontrollers have built-in timer/counters with a fixed clock source and a relatively easy access, normal CPUs found on PCs (like x86/64) have more complex structures and timer/counters are implemented in several ways, relying on hardware elements like the HPET or the TSC based on the CPU's clock, which can be variable[4] [13].

In particular, when measuring times with PCs in this thesis, the Query Performance Counter is used under Windows and function `clock_gettime` using the `CLOCK_MONOTONIC` source under Ubuntu. It's not trivial to know which is the ultimate source of time.

Another problem related to time management is pre-emption. If a kernel task is running, it cannot be stopped to allow execution of the RT task in default Linux unless the RT task is made in the kernel space. For dealing with this, the Linux's kernel can be modified to allow pre-emption for RT tasks and then recompiled.

These considerations must be taken into account depending on the application to be controlled but, in this thesis, it was unnecessary as the times involved for the control loop are around tens or hundreds of milliseconds, usually much higher than the delays induced by the motives above.

In Linux, a threaded version of the MQTT client is available.

For the Raspberry, however, fine tuning of the server was possible as seen in [14], but also unnecessary in this context. Also, the Raspberry was running a headless OS.

[4] In this thesis, an Intel I7 6700HQ is used, and it implements an invariant TSC as the recommended time source. However, the HPET can also be used and programs may use it if it is desired.

Chapter 7 Results

This chapter details all the obtained results of the system's tests.

The first section involves the EMC technique using fixed values of T_i in the model's design, so recalculation is not performed.

Second section presents the results of the AEMC modification, to check robustness over an increasing value of T_i.

7.1 EMC with fixed T

In order to see the impact of the network in the proposed system, all tests were carried out in 4 variants and, hence, the chapter is divided in 4 sections:

1. Plant and controller both on Simulink
2. Plant and controller/server on the same machine (localhost)
3. Controller running in an external server over LAN (ethernet)
4. Controller running in an external server over Internet

Each setting is tested 100 times, having each iteration different random initial conditions, friction's values and model deviations.

In particular, for these simulations the following were used:

Real mass deviations

$$m_{real} = m_{ideal}(1 + \nu)$$

$$\nu = N(0,0.05)$$

Viscous friction

$$F_{friction} = \mu\dot{x}$$

$$\mu \in [0,40]\,\frac{kg}{m \cdot s}$$

Initial conditions

$$\boldsymbol{x}(0) = \begin{pmatrix} x_0 \\ \dot{x}_0 \end{pmatrix}$$

$$x_0 \in [-15, 15], \qquad \dot{x}_0 \in [-1, 1]$$

7.1.1 Plant and controller on Simulink

As the EMC was initially implemented in Simulink, Model-In-the-Loop (MIL) simulations were made first (see Figure 11 - MIL Tests architecture).

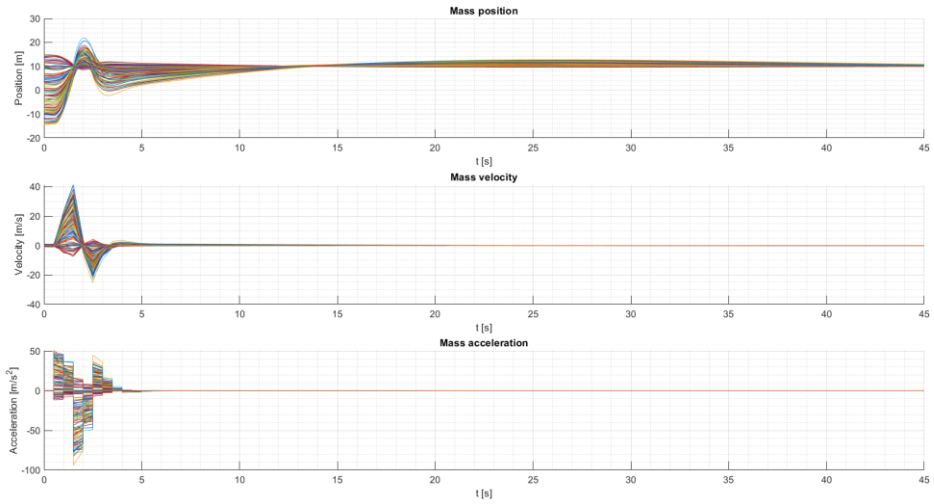


Figure 22 - Plant's states in MIL simulations

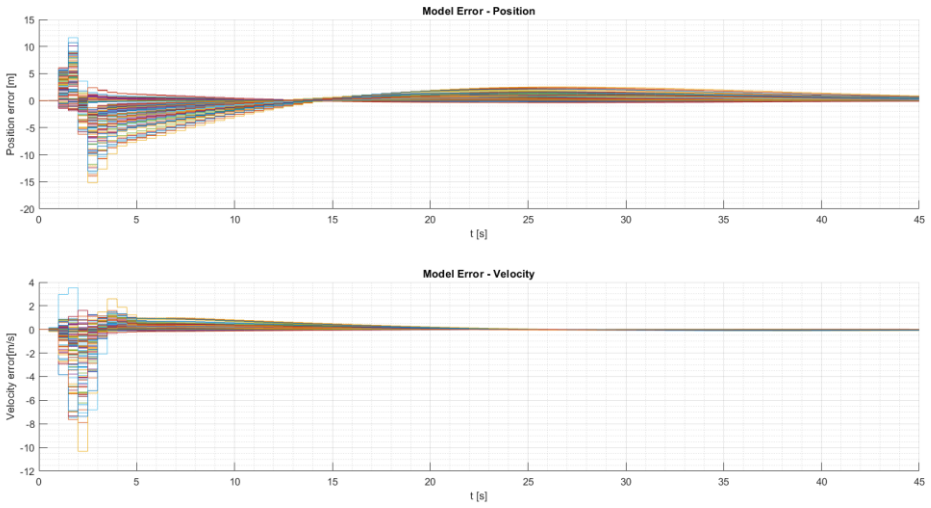


Figure 23 - Model errors in MIL simulations

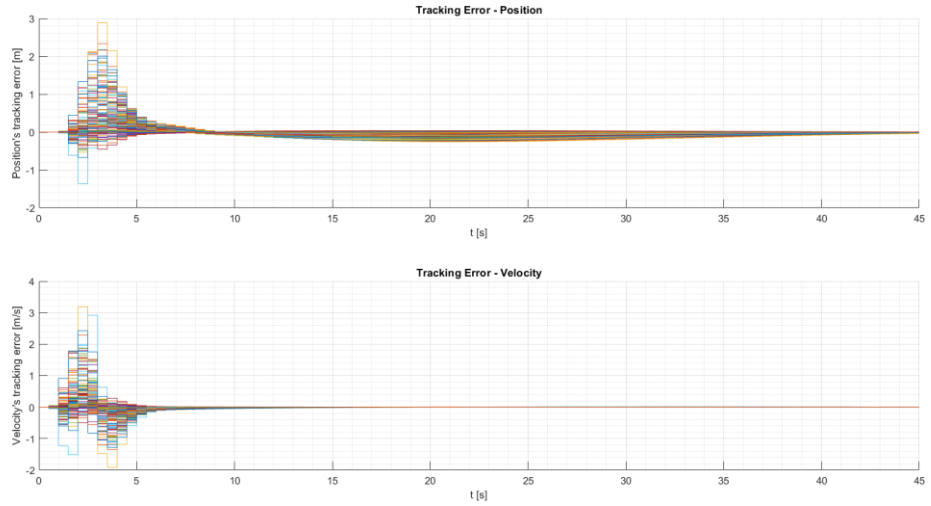


Figure 24 - Tracking errors in MIL simulations

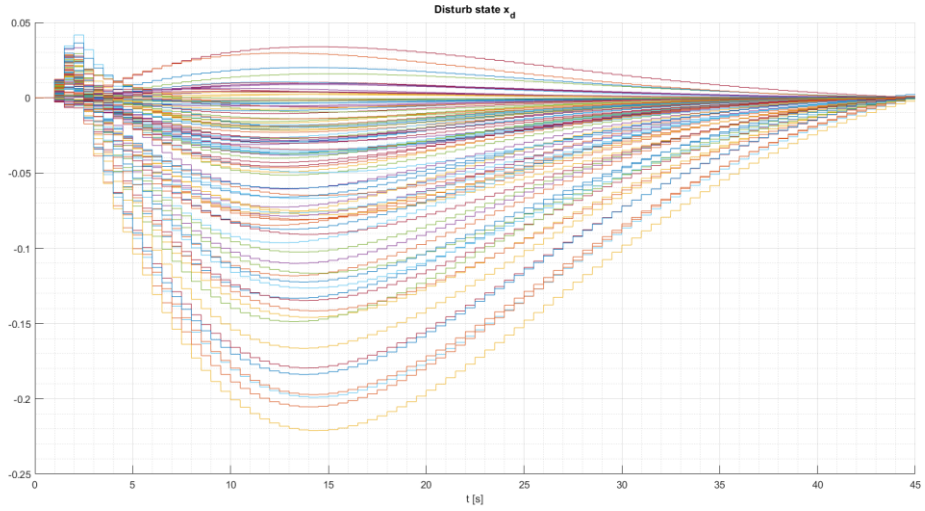


Figure 25 - Disturb state in MIL simulations

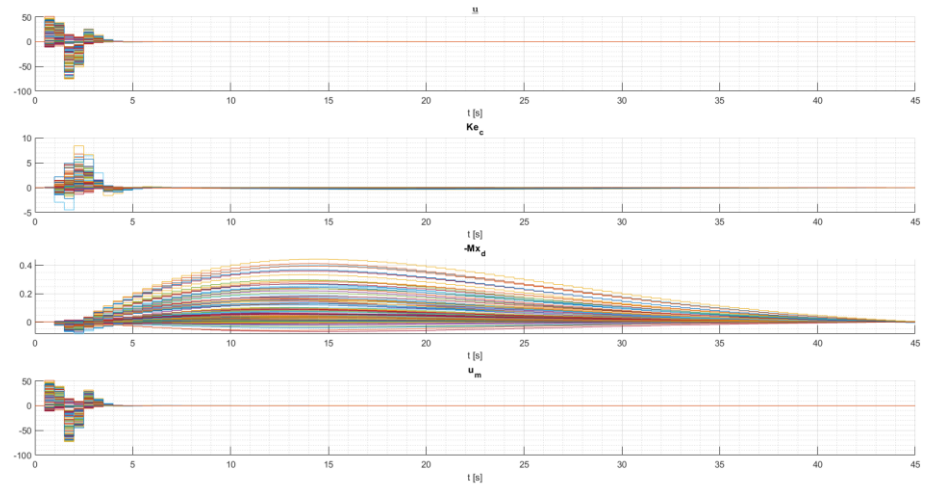


Figure 26 - Actuator's commands in MIL simulations

As seen from figures above, the EMC method effectively keeps the errors bounded (Figures 23 and 24) and convergent to zero, ensuring that the system will eventually reach the steady state of the reference set-point (Figure 22).

This is due to the action of the components of the command law (see section 2.2.3 and equation 2.4)

$$u_m(i+1) = \underline{u}(i) + K\,\underline{\boldsymbol{e_c}}(i) - M\boldsymbol{x_d}(i)$$

Where the black term represents and ideal reference command for a linear and undisturbed plant, the red term implements a feedback action to bring the model's states

closer to the reference states and a purple term to reject disturbances accounted through the disturb state x_d.

These command components are clearly shown in Figure 26 as the first 3 plots, and the fourth one represents the command (addition of the components).

7.1.2 Plant and controller on localhost

After successful runs of the models, code was generated and tested with the whole system in localhost (see Figure 19 – SIL Tests in localhost architecture).

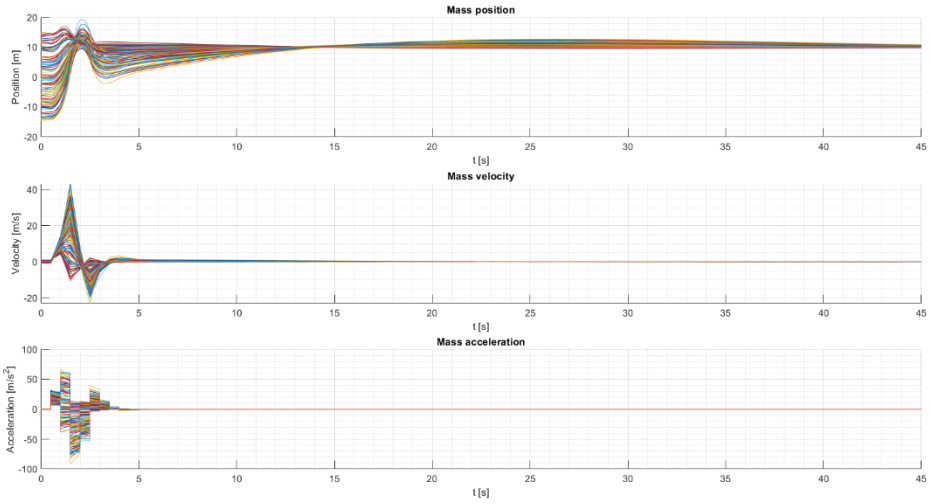


Figure 27 - Plant's states in SIL simulations

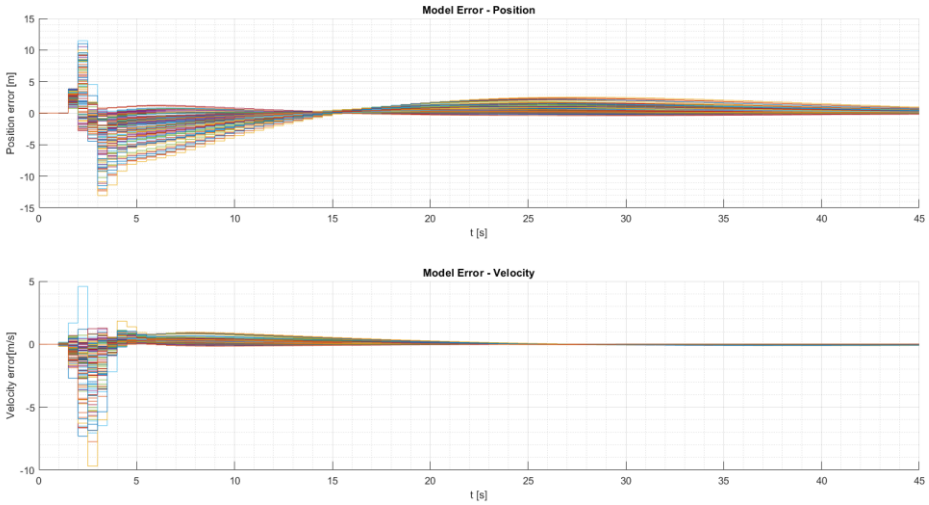


Figure 28 - Model errors in SIL simulations

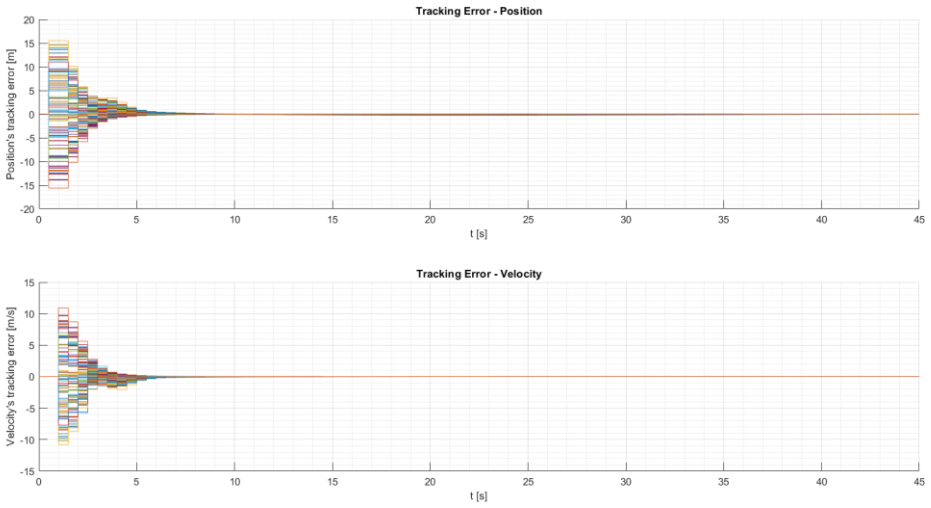


Figure 29 - Tracking errors in SIL simulations

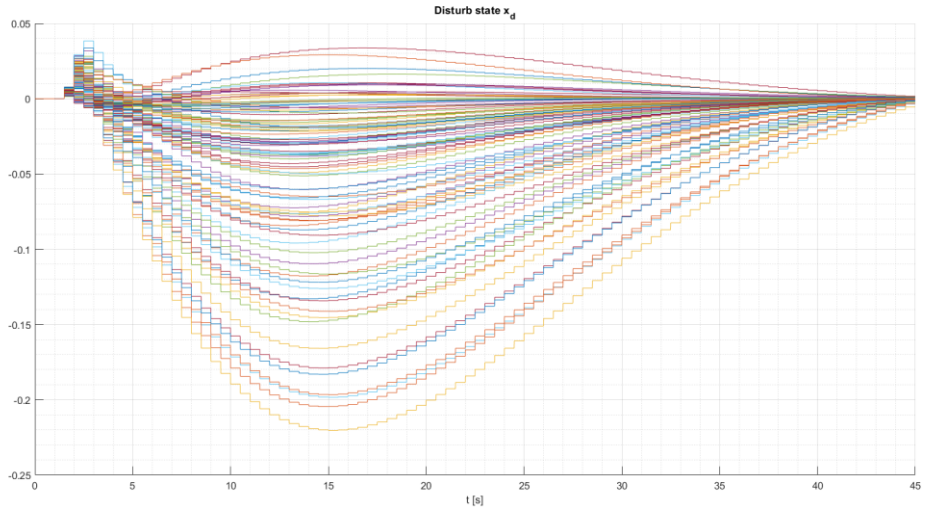


Figure 30 - Disturb state in SIL simulations

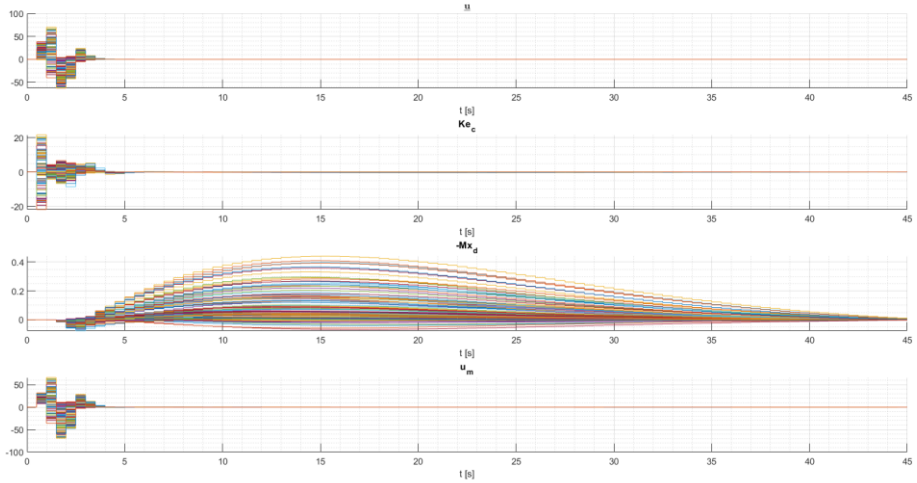


Figure 31 - Actuator's commands in SIL simulations

Successfully, no big differences are found between simulations in this section and the ones in previous section (7.1.1 - Plant and controller on Simulink).

Some variations are due to the change in precision, as MQTT communications transmit values with lower number of significant digits than the internal representation of Simulink.

As expected, the test validates the architecture in a localhost environment, being the control action performed in a satisfactory way and, as errors are bounded, the components of the actuator's command still work as they are designed to and disturb states don't become unstable (Figures 27 to 31)

7.1.3 Controller in external server through ethernet

To check if messages' flow works properly when 2 different machines are involved. For this stage, the Raspberry Pi was used. The Internet was not yet involved and connection was made in local area through an ethernet cable. This particular setup is used to check, firstly, compatibility with other platforms than Windows and, secondly, that network interfaces from both OSs don't interfere with the function of the system.

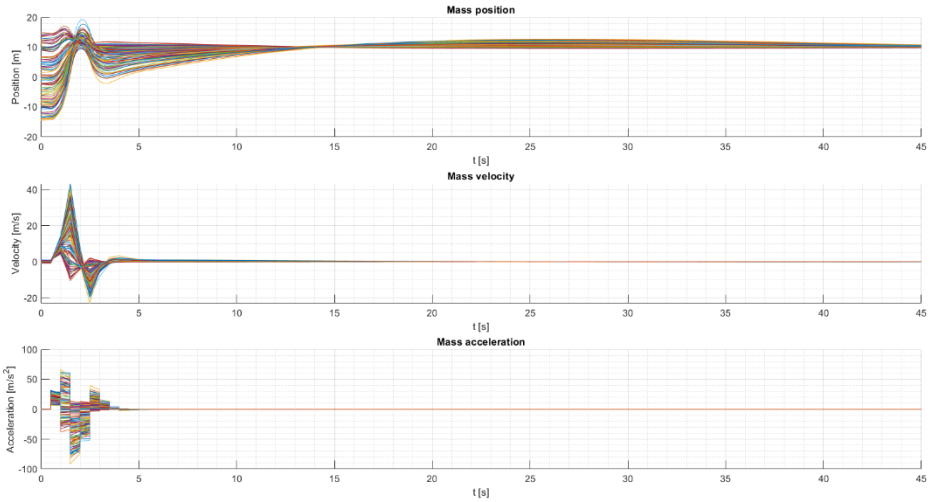


Figure 32 - Plant's states using the Raspberry as server

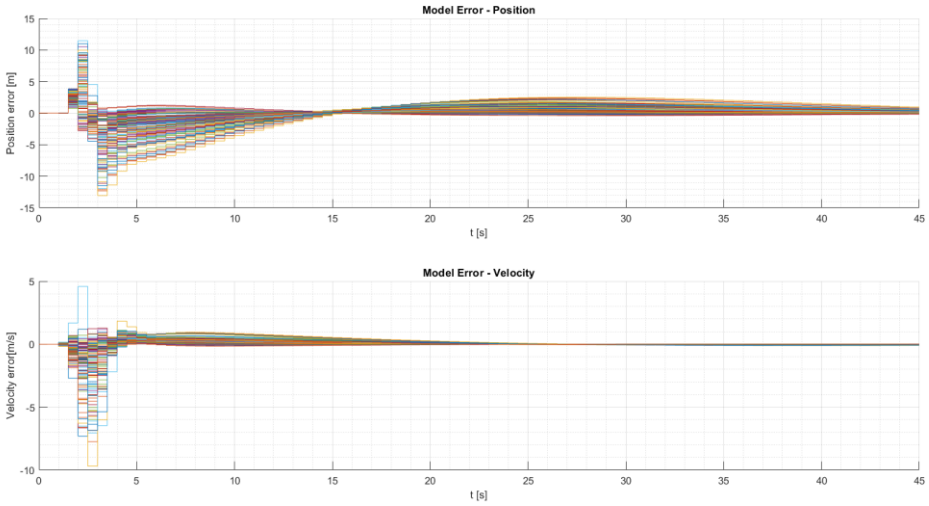


Figure 33 - Model errors using the Raspberry as server

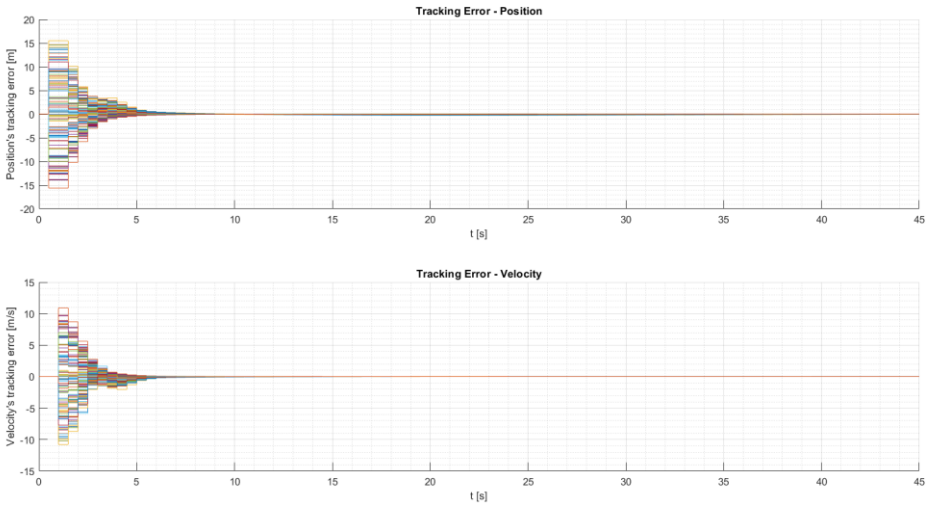


Figure 34 - Tracking errors using the Raspberry as server

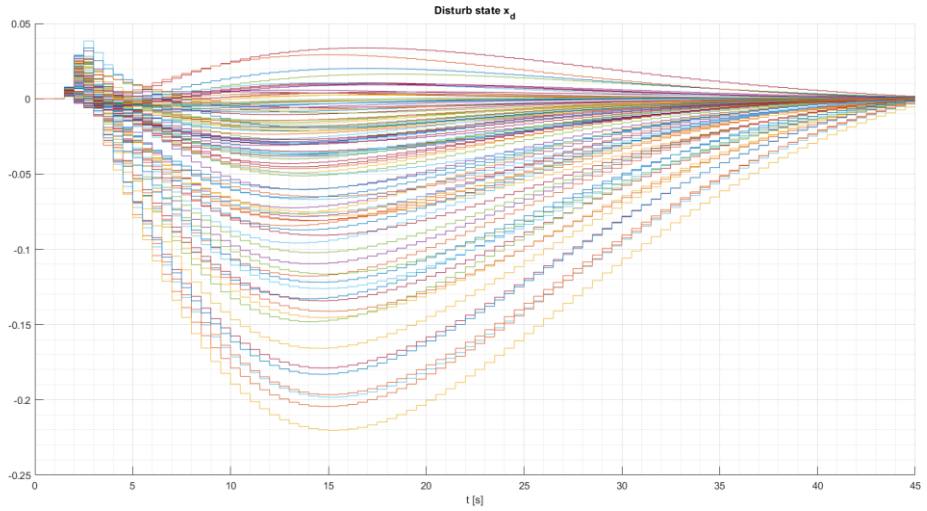


Figure 35 - Disturb state using the Raspberry as server

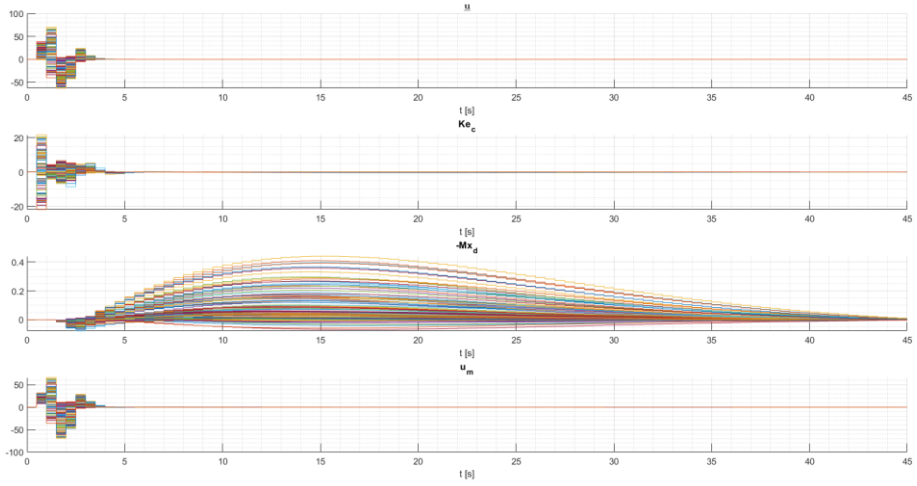


Figure 36 - Actuator's commands using the Raspberry as server

Comparing figures from 32 to 36 above with results from previous sections (7.1.1 and 7.1.2), system performance is validated as no significant differences are found.

7.1.4 Controller in external server through Internet (CIL)

The main results of this thesis are those derived from the complete implementation of the control system running the EMC through the Internet, connecting the plant in Madrid, Spain, and the control server in Torino, Italy[1].

[1] In practice, the path followed by the packets sent in the control loop are unknown to the system, derived from the way Internet works, so endpoints are considered.

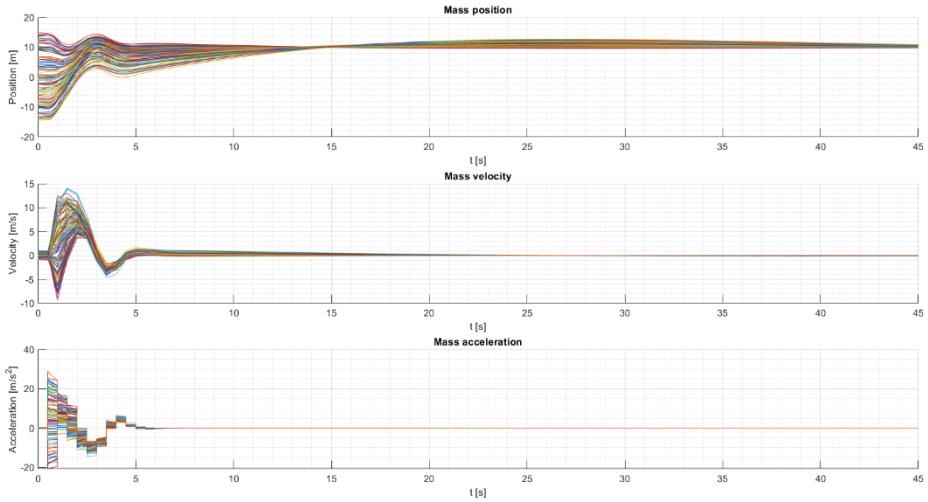


Figure 37 - Plant's states in CIL simulations

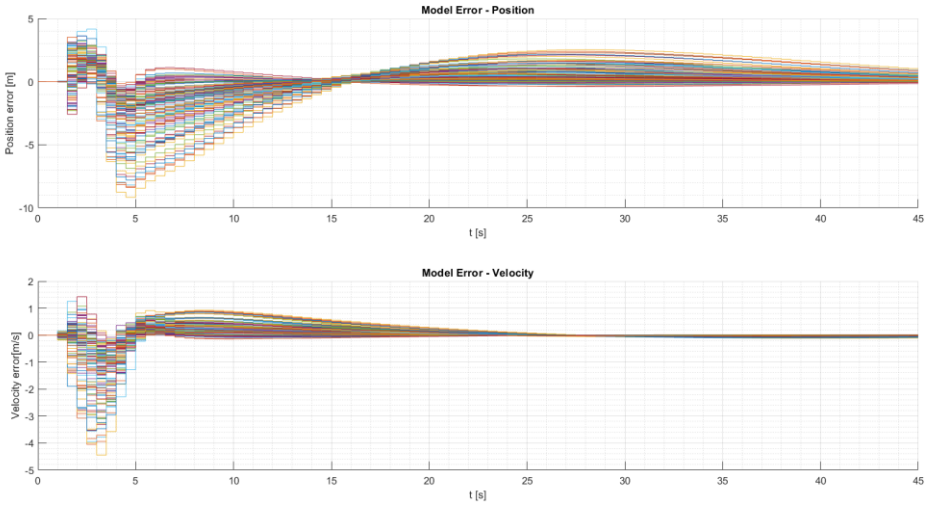


Figure 38 - Model errors in CIL simulations

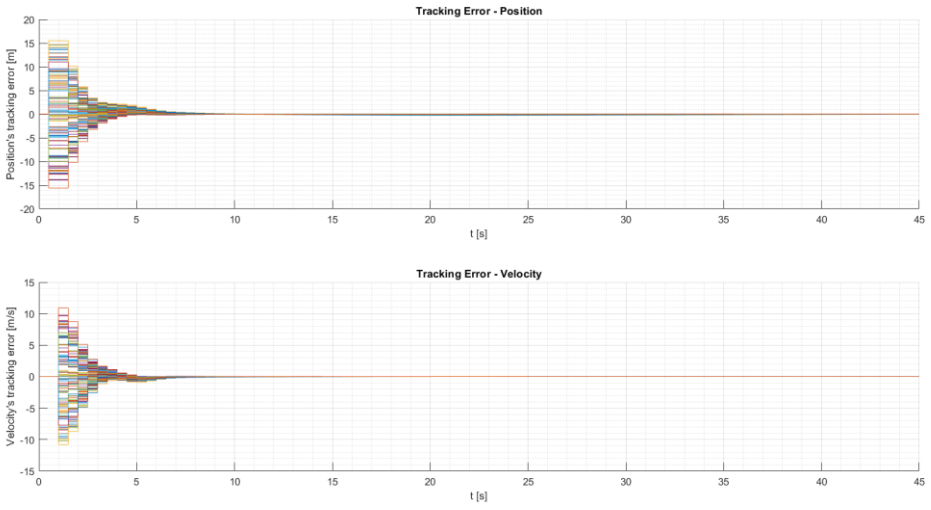


Figure 39 - Tracking errors in CIL simulations

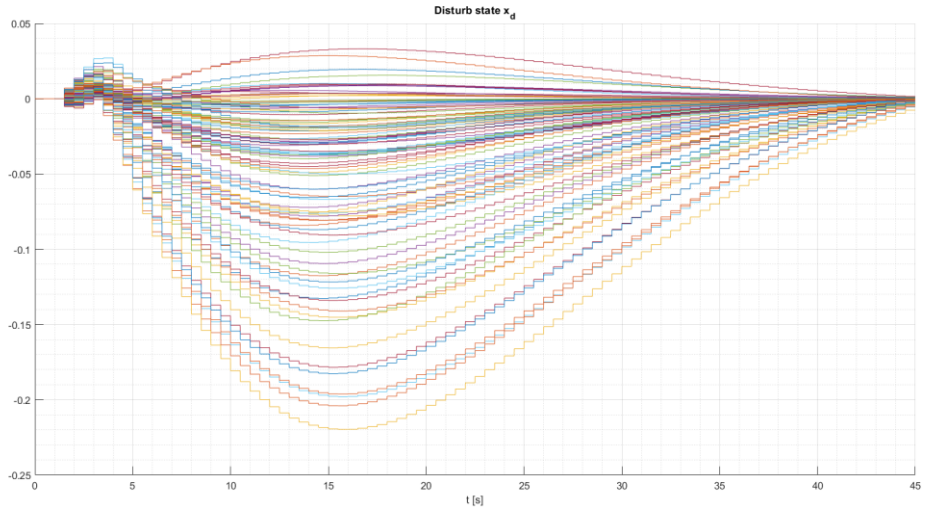


Figure 40 - Disturb state in CIL simulations

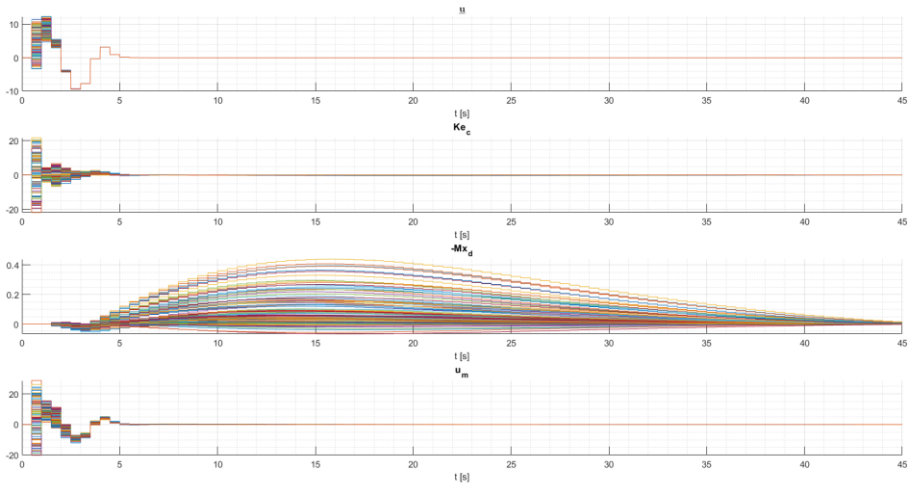


Figure 41 - Actuator's commands in CIL simulations

As seen in figures 37, 38, 39, 40 and 41, performance of the system working over the objective network doesn't present significative changes. The desired system works as expected, and shows its capabilities successfully, so validation of the Cloud-In-the-Loop setup is done.

Even the geographic distance and being the Internet involved, both tracking and model errors remain bounded and, if a constant reference is inputted, convergent to 0.

As exposed in section 7.1.1 above, this behaviour derives from the command law structure and the model's updating using the noise estimator (see section 2.2.3, equation 2.4).

7.2 Simulations with variable T (AEMC)

This set of results serve as a comparation between the fixed control time system implementing the EMC and the variation exposed in Chapter 5 using the EMC with recalculations on each timestep AEMC.

For comparison, a set of parameters is fixed and on each iteration the values of T_i are gradually increased from a starting value of 0.5s to a final value of 10.5s in increments of 0.1s.

In this section, only MIL and CIL test are performed, as in previous section (7.1), SIL and CIL using the Raspberry were successfully performed and no additional need of the

system's function existed as this section present results for validating the AEMC algorithm.

7.2.1 Plant and controller on Simulink

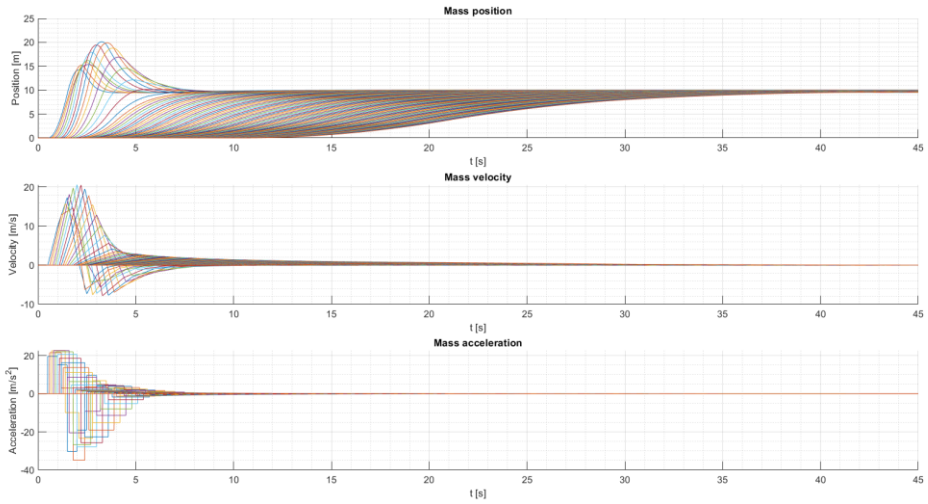


Figure 42 – Plant's states in Asynchronous EMC MIL simulations

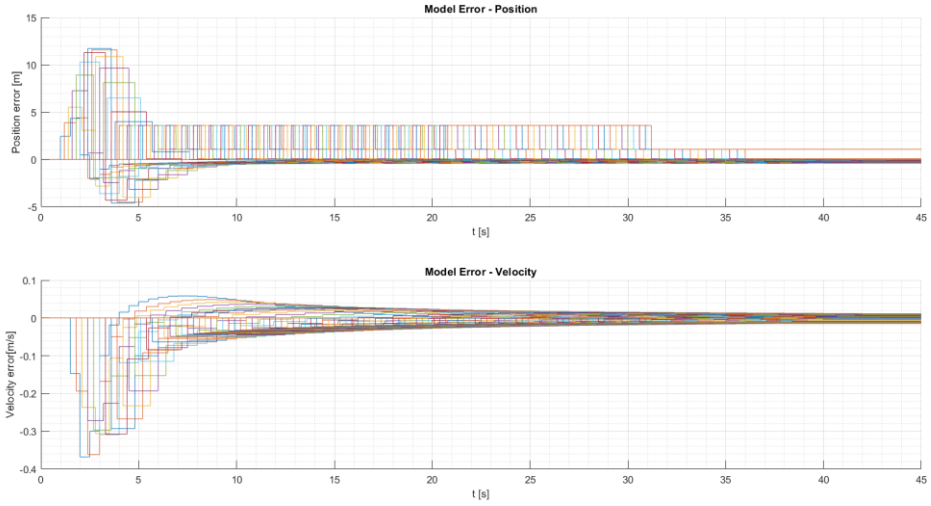


Figure 43 - Model errors in Asynchronous EMC MIL simulations

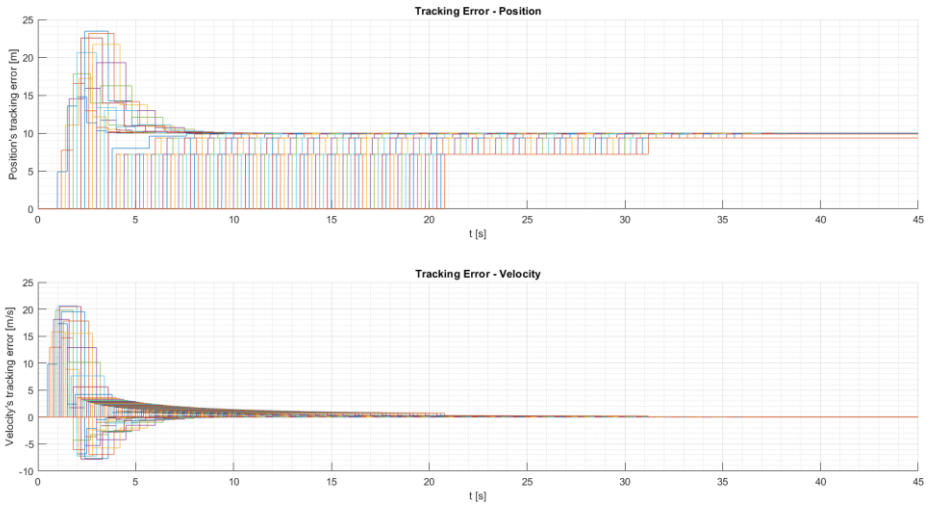


Figure 44 - Tracking erros in Asynchronous EMC MIL simulations

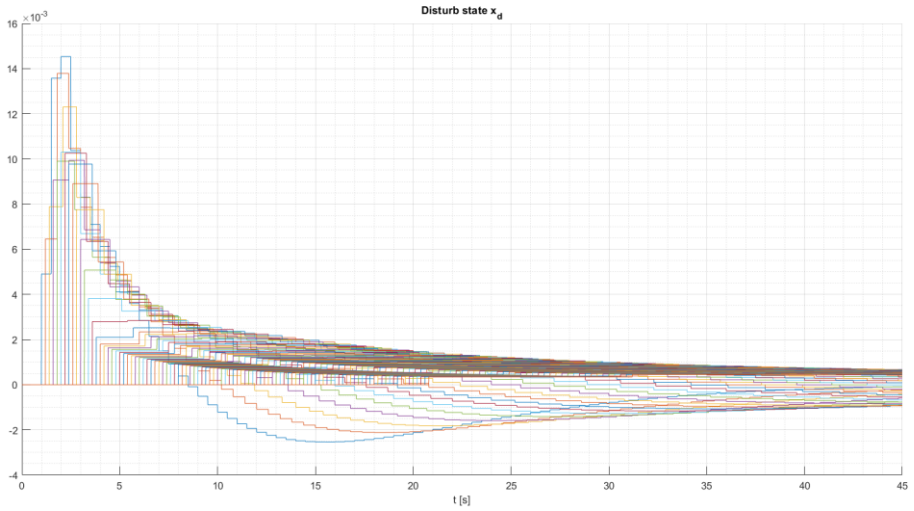


Figure 45 - Disturb state in Asynchronous EMC MIL simulations

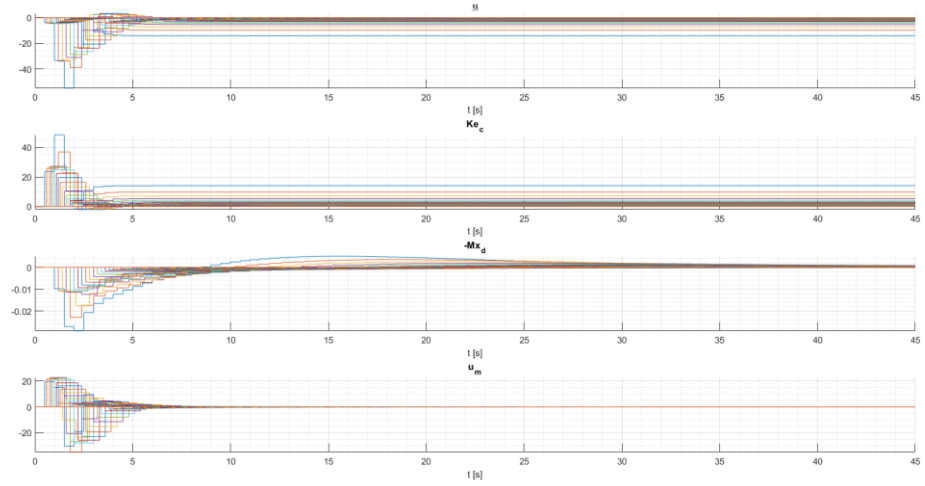


Figure 46 - Actuator's commands in Asynchronous EMC MIL simulations

From figures above, it can be checked that the algorithm proposed works as expected.

It avoids instabilities with the increasing values of T_i by fixing the discrete eigenvalues of the system making, thus, the continuous dynamics of the controlled plant slower.

Next step is to validate and test the utility of the method in the real networked environment.

7.2.2 Controller in external server through internet (CIL)

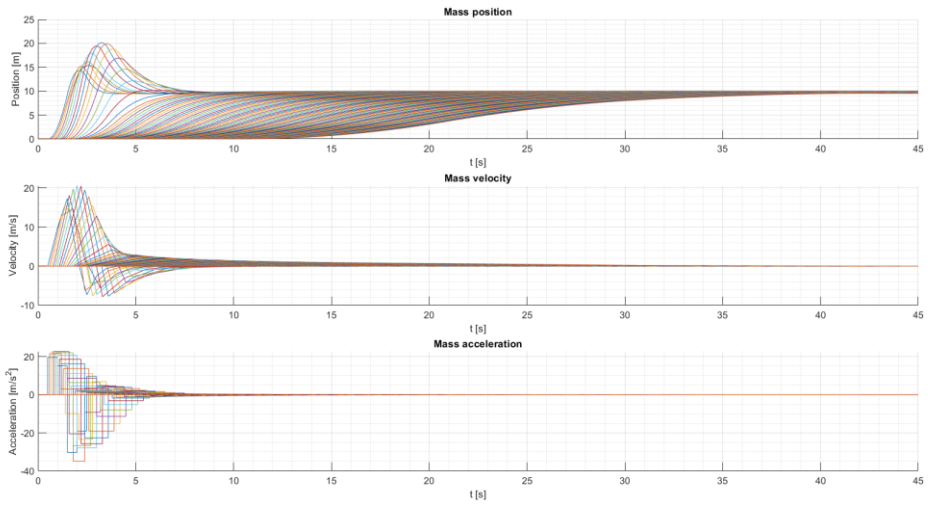


Figure 47 - Plant's states in Asynchronous EMC CIL simulations

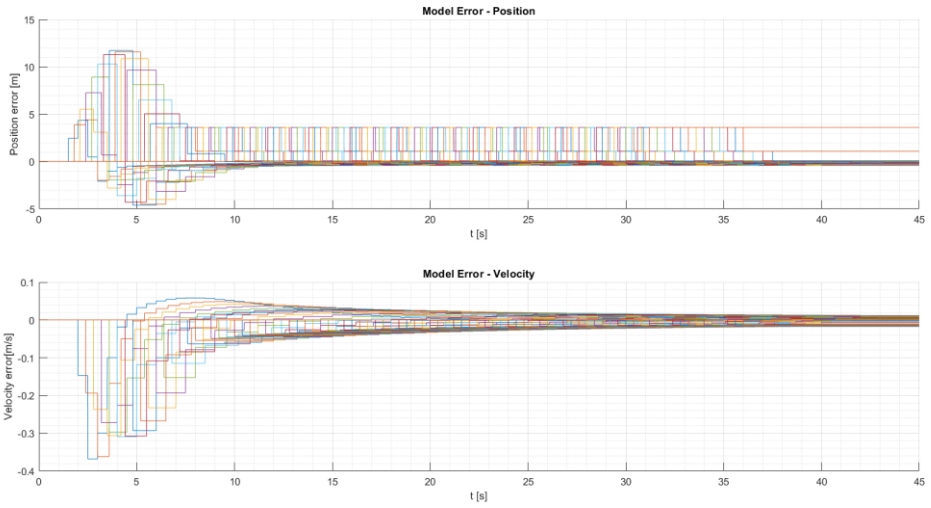


Figure 48 - Model errors in Asynchronous EMC CIL simulations

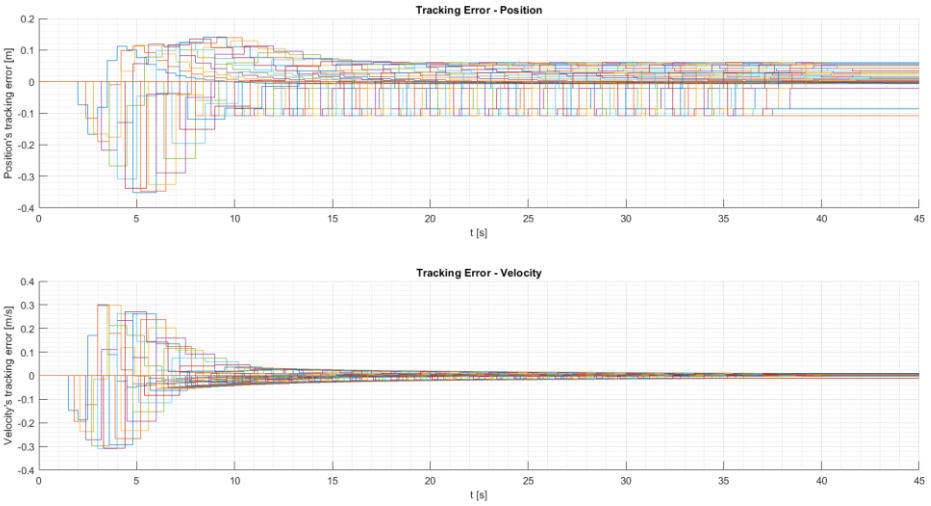


Figure 49 - Tracking errors in Asynchronous EMC CIL simulations

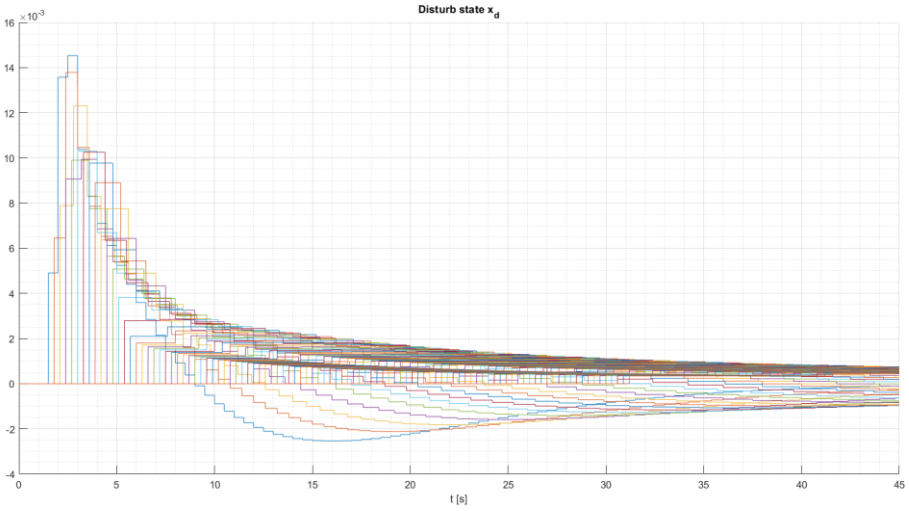


Figure 50 - Disturb state in Asynchronous EMC CIL simulations

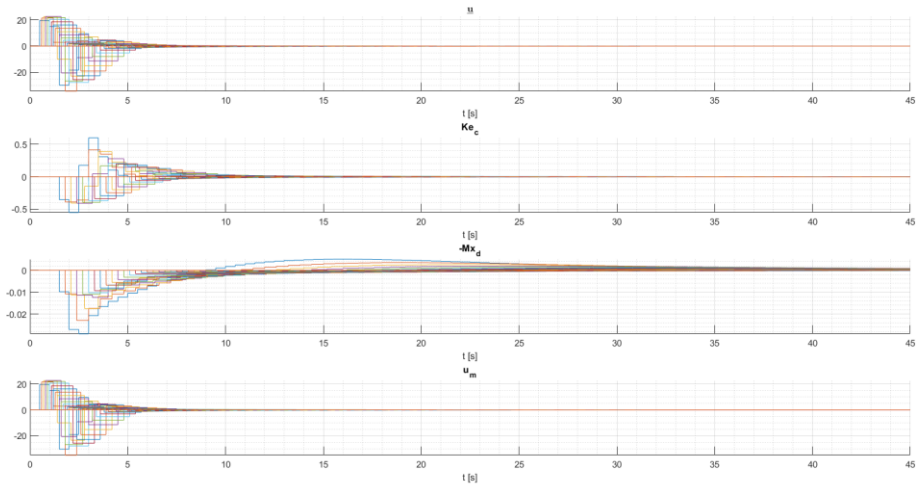


Figure 51 - Actuator's commands in Asynchronous EMC CIL simulations

It can be seen from plots above (figures 42 to 51), that the Asynchronous EMC modification is fully functional and works as expected when implemented, without observed differences between the Simulink simulations of the controller or its implementation as C code in the server located Politecnico di Torino.

As the approach chosen for demonstration is to fix discrete eigenvalues of the system (see Table 1 in section 4.5.2), it can be seen that stability is maintained along the whole range of T_i values used.

Chapter 8 Conclusions

As seen in the previous chapter, in all scenarios the control strategy works: it keeps the error bounded, avoids instability and is able to reach the steady state sooner or later.

The values used for changing each iteration of the Monte Carlo's simulations are chosen to test robustness of the system (in real applications modelled mass' deviations are not likely to be greater than the 15%, viscous friction should be modelled and parametrised, etc.), and the system success in reaching the final objective.

Due to technical restrictions, the sampling time used for control commanding had to be greater than 0.5 s (further details can be found in page 59). Using this approach allow us to minimize the impact of the network's delays by imposing a superior limit in round trip times. However, bigger delays occur as it is impossible to guarantee transmission times over the Internet.

Bias in sensors and unmodelled dynamics are effectively counteracted due to the ability of the EMC to move that deviations from the real states and take them into the disturbance states. This is one of the most important features of the EMC technique.

It can be observed in the command's plots (Figures 26, 31, 36, 41, 46 and 51) the different components of the actuator's signal, each one of them having a different mission as explained in section 2.2.3, and the function of the disturb state x_d accounting deviations of the model from the plant.

High frequency noise present in measurements is also attenuated, being the threshold dependant on the poles chosen for the observer. In that sense, the EM plus the NE act both as a low-pass filter.

Although the static feedback provides a better response in a constant control sampling time environment without many disturbances, this performance becomes very degraded when this sampling time increases or in the presence of certain deviations such as sensor biases. With an increase of a 40% of the nominal time, the static feedback controller is unable to maintain stability and the system becomes unstable.

On the contrary side, the EMC technique stands a 180% increase without entering into the unstable regimen, and this is further improved by adding support for the variation of the time into the model, allowing the controller to successfully stabilise de system.

The last result, using the update of the T_i value, affects the maximum supported input frequency of the reference. This is because, as stated in section 5.2, when the discrete eigenvalues are placed to the desired values, the derived continuous eigenvalues of the plant change accordingly and so does the frequency of the system (see equation 5.1). This implies that the system becomes slower and unable to follow, for example, a sinusoidal reference with a higher frequency.

The important result derived from the asynchronous modification of the EMC technique is the ability to keep dynamics of the system under control on each control step by measuring the last T_i value. This is true both in theory and in practice as shown in the implementation of this thesis.

The overshoot, which also varies as the T_i values do, has also to be considered depending on the application of the control loop (i.e. the plant to be controlled).

In what security concerns, a very recommended continuation of this project (specially when considering a commercial or professional application) is a security auditory by a specialist who can further improve and certify its compliance with current state-of-the-art techniques. The author has implemented TLS encryption, but the choice of restraining the system to certain ciphers and TLS version must be performed by an expert, as the sole analysis would comprise an entire thesis on its own.

This is because a production implementation of a system involving the Internet is going to be sooner or later under attack attempts.

Another recommended future modification is to implement the MQTT communications using the new standard MQTT v5.0 published by OASIS. It provides further capabilities such as enhanced error messages when clients are force disconnected by the server, improved request/response mechanisms and additional parameters allowed to be sent by a client to configurate its own relationship with the server.

To sum up, both the EMC and Asynchronous EMC techniques work as expected over a networked environment and its use is recommended as allows the implementation of a low-cost and power computing element near the plant, delegating the computational requirements to a more powerful element (from a standard PC, a rack of servers, to rented services of Cloud Computing) via the Internet.

Bibliography

[1] E. Canuto, C. Novara, L. Massotti, D. Carlucci y C. Perez Montenegro, Spacecraft Dynamics and Control: The Embedded Model Control Approach, Butterworth-Heinemann, 2018.

[2] Y. Hyuk Kim, L. Dinh Phong, W. M. Park, K. Kim y K. Rha, «Laboratory-level telesurgery with industrial robots and haptic devices communicating via the internet,» *International Journal of Precision Engineering and Manufacturing,* vol. 10, pp. 25-29, April 2009.

[3] A. Bemporad, M. Heemels y M. Johansson, Networked Control Systems, M. Morari, M. Thoma y A. Bemporad, Edits., Springer London, 2010.

[4] X. Ge, F. Yang y Q.-L. Han, «Distributed networked control systems: A brief overview,» *Information Sciences,* vol. 380, pp. 117-131, 2017.

[5] E. Canuto, “Embedded Model Control: Outline of the theory,” *ISA transactions,* vol. 46, no. 3, pp. 363-77, 2007.

[6] E. Canuto, W. Acuna-Bravo, A. Molano-Jimenez and C. Perez Montenegro, “Embedded model control calls for disturbance modeling and rejection,” *ISA Transactions,* vol. 51, no. 5, pp. 584-595, 2012.

[7] W. Acuña-Bravo, A. Molano-Jiménez and E. Canuto, “Embedded mode control, performance limits: A case study,” *DYNA,* vol. 84, no. 201, pp. 267-277, 2017.

[8] A. Diaz and A. Nipper, Interviewees, *Episode 9.* [Interview]. January 2012.

[9] R. A. Atmoko, R. Riantini y M. K. Hasin, «IoT real time data acquisition using MQTT protocol,» *Journal of Physics: Conference Series,* vol. 853, May 2017.

[10] International Organization for Standardization, Information Technologies - Message Queuing Telemetry Transport (MQTT) v3.1.1, ISO Standard 20922, 2016.

[11] International Organization for Standardization, Information Technologies - Database languages - SQL, ISO Standard 9075, 2016.

[12] F. Matía Espada, A. Jiménez Avello y R. Aracil Santonja, Teoría de sistemas, Madrid: Sección de Publicaciones de la ETS de Ingenieros Industriales, Universidad Politécnica de Madrid, 2006.

[13] Intel Corporation, "17.17 TIME-STAMP COUNTER," in *Intel 64 and IA-32 Architectures Software Developer's Manual*, 2019, pp. 3327-3328.

[14] M. Prudek, Enhancing Raspberry Pi Target for Simulink to Meet Real-Time Latencies, Czech Technical University in Prague, 2017.

[16] Y.-B. Zhao, X.-M. Sun, J. Zhang y P. Shi, «Networked Control Systems: The Communication Basics and Control Methodologies,» *Mathematical Problems in Engineering,* vol. 2015, p. Article ID 639793, 2015.

[17] D. Hristu-Varsakelis and W. S. Levine, Eds., Handbook of Networked and Embedded Control Systems, Birkhäuser, 2005.

[18] Y.-Q. Xia, Y.-L. Gao, L.-P. Yan and M.-Y. Fu, "Recent progress in networked control systems - A survey," *International Journal of Automation and Computing,* vol. 12, no. 4, pp. 343-367, 01 Aug 2015.

[19] T. L. Floyd, Digital fundamentals, Boston: Pearson, 2015.

[20] SEBoK Authors, *System Life Cycle Process Models: Vee, SEBoK,* R. Cloutier, Ed., San Diego: International Council on Systems Engineering (INCOSE), 2019.

Acronyms

Along this thesis report several terms and expressions are used, and it's convenient to define them in order to have a complete comprehension of the topics dealt.

Acronyms

AEMC: Asynchronous Embedded Model Control

ANSI: American National Standards Institute

CA: Certificate (or Certification) Authority

CMP: Control Management Program

DB: Database

EM: Embedded Model

EMC: Embedded Model Control

HPET: High Precision Event Timer

HTTP: HyperText Transfer Protocol

LAN: Local Area Network

MQ: Message Queueing

MQTT: MQ Telemetry Transport

NCS: Networked Control System

NTP: Network Time Protocol

NE: Noise Estimator

OS: Operating System

PID (when referring to a controller): Proportional, Integral and Derivative

PoliTO: Politecnico di Torino

PSK: Pre-Shared Key

SQL: Structured Query Language

TCP: Transmission Control Protocol

TSC: TimeStamp Counter

TLS: Transport Layer Security

XML: eXtensible Markup Language

Appendix A Codes

A.1 WordPress shortcode

PHP code for display results on the webpage

```
function showResults_func( $atts ){
	$servername = "localhost";
	$username = "jmp";
	$password = "xxxxxxx";
	$dbname = "jmp";

	// Create connection
	$conn = new mysqli($servername, $username, $password, $dbname);

	if ($conn->connect_error) {
		die("Connection failed: " . $conn->connect_error);
	}
	$sql = "SELECT * FROM sim_control";
	$conn->query($sql);
	$result = $conn->query($sql);
$html = '<div><table style="width:100%">
  <tr>
	<th>Time</th>
	<th>Command</th>
	<th>Reference</th>
```

```
    <th>Output</th>
    <th>Error</th>
    <th>Integral of error</th>
</tr>';
    if ($result->num_rows > 0) {
          while ($row = $result->fetch_assoc())
          {
          $html=$html."<tr>";
          $html=$html."<td>";
          $html=$html.$row["time"];
          $html=$html."</td><td>";
          $html=$html.$row["u"];
          $html=$html."</td><td>";
          $html=$html.$row["yr"];
          $html=$html."</td><td>";
          $html=$html.$row["y"];
          $html=$html."</td><td>";
          $html=$html.$row["error"];
          $html=$html."</td><td>";
          $html=$html.$row["errorInt"];
          $html=$html."</td></tr>";
          }
    }
    mysqli_close($conn);
    $html = $html . "</table></div>";
    return $html;
```

```
}


add_shortcode('showResults','showResults_func');
```