POLITECNICO DI TORINO

Master's degree in Mechatronic Engineering

Master's Thesis

Cyber-Physical Security on Distributed Control Systems: an observer-based approach

Supervisor
Prof. Alessandro Rizzo

Candidate
Lorenzo Biroli

Supervisor
Brain Technologies
Ing. Giovanni Guida

Academic Year 2020-2021

Abstract

This academic work was born as an evolution of the pre-existing BAT-MAN project developed in the last two years by the company *Brain Technologies*. From it, we took the basic concepts in order to apply them into the *Cyber-physical security* field. The goal in fact is to use an observer-based approach to be able to successfully detect an attack and, above all, identify the kind of attack the system is receiving.

Different analysis and tests have been performed, so that only the most significant are presented in this work. The first part presents a brief dissertation of the cyber-physical security field in general. Part II aims to the introduction of the formal theory that explains the system developed, while in Part III an application to a DC-motor is presented, as well as the reasoning that took us to the final concept. Finally, Part IV concludes the work and presents a final discussion on the method.

Contents

List of Figures

Part I

Introduction

Chapter 1

Introduction

1.1 Why it is important to study Cyber-Physical Security?

The concept behind this thesis work is inspired by the growing technological development that involves an ever increasing number of devices remotely connected, expanding an approach already implemented by the company *Brain Technologies* in other applications, with the goal of applying it to the *Fault and Attack detection*, with particular emphasis on *distributed cyber-physical security*.

With the advancement of technology, we observe more and more an increasing fusion between informatic and physical systems, with huge impact on life and security of human beings. This makes of primary importance the development of defense systems able to detect extern attacks to the system, such as the one direct to the Iranian nuclear power plants, one of the most famous cyber-attack of this kind. In that case, a virus called *Stuxnet* was able to modify the speed of the nuclear power plants centrifuges, sending in the meantime wrong data to the operators, so to avoid its detection [1].

The cyber-physical security theme has become of great relevance in the last decades, with a notable increase of the literature on the subject.

1.2 Brief history of cyber-physical security

Resuming the contents of [2] and [3], on which this section is based, the term *cyber-physical system* was conied in 2006 to describe all this kind of system that integrates cyber and physical worlds, but the researches and the interest in this interaction was real strong yet many years before, starting from the '90s. However, the origins of this field have their basis on the computer invention. The first computer *ENIAC* was built in 1946, but we have to wait until 1973 for the first real-time computations, which laid the foundation for the cyber-physical system to born. Meanwhile, also

the Internet started its developmente, with the first network *ARPANET* developed in 1969, so that around the end of the '90s the convergenge of communication and computation was at complete. An important milestone in the building of complete and useful cyber-physical system was posed around 1998 with the development of sensors capable of sensing, communication and computation, increasing even more the physical integration of the cyber-systems. In recent years, all this has evolved even more, with the concept of *Internet of Things* and an increasing demand in for better and more efficient energy, transportation, healtcare and water systems.

In parallel to the evolution of the cyber-physical system technology, the security problem were studied, to make us able to detect and interrupt an intruder that could modify the behaviour of a system. The history of cyber-physical system presents differents cases of importante and noticeable attacks, like the most famous *Stuxnet* one, and are presented graphically in figure 1.1, directly taked from [3]. A complete report of all the different attack occurred is impossible to compile, due to the lack of information in many cases. In fact, in the figures, as the authors claimed, only the publicly reported attacks are presented.

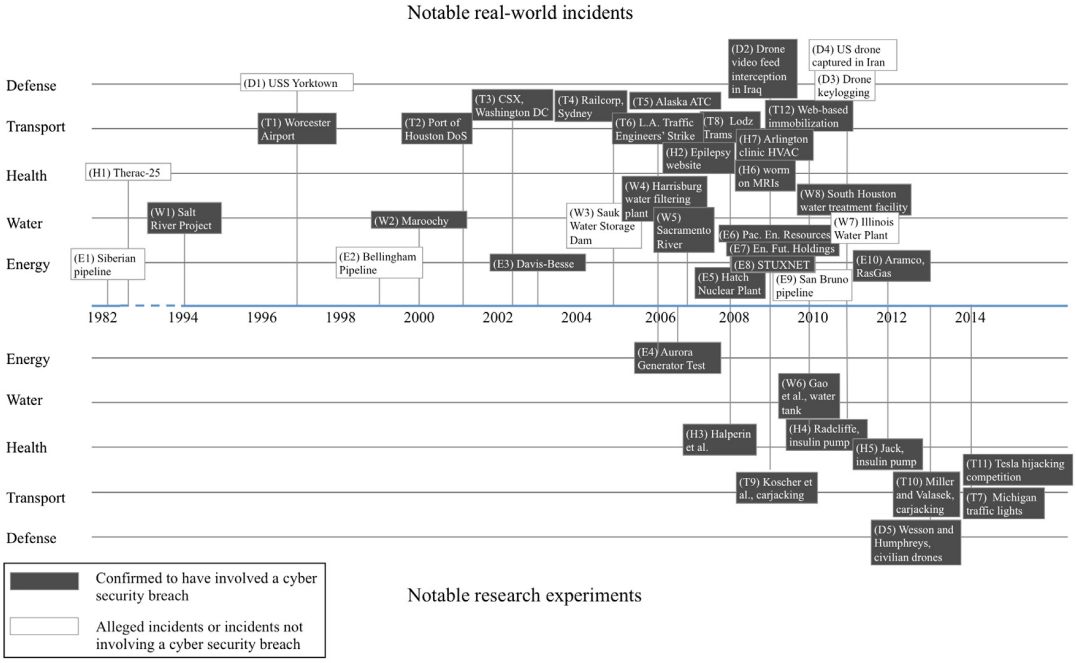


Figure 1.1: Historical timeline of known CPS attacks [3]

1.3 State of the art of cyber-pysical security

As of now, the *CPS* security field is still not totally examined, and the researches and studies are increasing as long as the technology advances and the intruders achieve more and more refined methods of attacking. In tables 1.1 and 1.2, directly taken from [4], a taxonomy of the different possible attacks that a CPS can receive is presented. In table 1.1, the second column represents if noise has been considered or not in the approach. Appendix **A** better explains some of the terms present in the table.

According to [5] the defense of a CPS has to be performed through three different aspects:

- **Prevention**
  Ability to prevent attacks by providing authentication, access controls, security policies, and network segmentation.
- **Detection**
  Ability to detect that the CPS is receiving an attack.
- **Response**
  The ability of the CPS to automatically reduce the impact of the attack without human intervention.

The attacks, as well as the faults that the CPS could faces, are different and numerous, and need a specific approach to mitigate them. Looking at the different faults, [6] presents a good list of the different possibilities that a designer can try in order to avoid and/or counterattacks the effects of faults:

- **Data Methods and Signal Models**
  - Limit checking and trend checking
  - Data analysis (PCA)
  - Spectrum analysis and parametric models
  - Pattern recognition (neural nets)
- **Process Model Based Methods**
  - Parity equations
  - State observers
  - Parameter estimation
  - Nonlinear models (neural nets)
- **Knowledge Based Methods**

- Expert systems
- Fuzzy logic

On the other hand, looking at the attacks induced by an external intruder, we can have different possibilities too. According to [7], the taxonomy of the attack detection methods is divided depending on the type of controller, *centralized* or *distributed*:

- **Centralized Controllers**
  - *Attack Detection Design*
    - State Estimation
    - χ^2 detector
    - Fault Detection and Identification method
    - Binary Hypothesis-Based Method
    - Model-Free Detection Scheme
  - *Attack Detection Against Actuator and Sensor Attack*
    - Attack Space Search Method
    - Convex Relaxation Method
    - Attack Estimation Method
    - Watermarking Method
  - *Attack Detection for Nonlinear Systems*
    - Iterative State Estimation
    - Kalman Filter-Based Method
    - Observer-Based Method
  - *Attack Detection in the Presence of Noise*
    - Modified Kalman filter
- **Distributed Controllers**
  - *Attack Detection Design*
    - Centralized Method
    - Singular Value Decomposition
    - Fault Detection and Identification Method

More informations on the state of art of CPS security can be found in [8], an interesting paper presenting all the different threats that various CPS have to counterattacks, that can't be resumed here in few pages.

Type of System	**Noise**	**Attack Models**	**Defense Mechanism**
Power Grid	Yes	False data injection on sensors	Residue detector
Control System	No	Attacks on Sensor and Actuators	Detection Filters
Control System	No	Attacks on Sensor and Actuators	Optimization Decoders
Control System	Yes	Replay Attacks	χ^2 detector
Wireless Network	No	State Attacks	Output Estimator
Distributed Network	No	State Attacks	Combinatorial estimator
Sensor Network	Yes	Dynamic False Data Injection	Residue Detector

Table 1.1: Taxonomy of CPS Security Approaches from a Control-theoretic Perspective [4]

Analysis Perspective	**Highlights**
Protocol Vulnerabilities	Modeling CPS protocols to detect anomalies
PLC Software	Verifying PLC code and memory to prevent violations
Process Variables	Predicting CPS process behavior to detect anomalies
Network Measurements	Data-driven approaches to infer CPS cyber attacks

Table 1.2: Taxonomy of CPS Security Approaches from a Cyber Security Perspective [4]

Chapter 2

Problem Setup

In this chapter is introduced the problem we have to deal with. Before showing the analysis and the development of the attack identification system, which is performed in *Part III*, here there is a brief introduction of the general setup to which our system aims.

2.1 Goal of the work

The system studied in this thesis is a *distributed control system* (*DCS*) composed by different *embedded system* remotely connected. Because of this the communications between different systems can be attacked and modified by an external intruder. The objective is then be able not only to detect the attack, but also to identify which of the communication has been attacked, or, in other words, which of the signals the embedded systems share has been modified.

2.2 System under attack

Here we introduce the formal presentation of the problem. Our *DCS* system is composed by different systems here presented:

- *Plant*: the real physical system;
- *Load*: the real physical system to which the *plant* is connected;
- *Controller*: the system able to control the plant;
- *Reference generator*: the system that generates the reference for the controller;
- *Observers*: the system that contains the observers used for the detection and identification of the attacks;

- *Detection and Identification system* (*DaIS*): system which performs the attack detection and identification algorithm after having received the outputs of the plant and of the observers;

These systems communicate between them sharing information, thus every signal could be modified by an intruder. Let's formalize the problem. We can model the plant as a MIMO system of the form:

$$\begin{cases} \dot{x}_p(t) = A_p x_p(t) + B_p u_p(t) + E_p d_p(t) \\ y_p(t) = C_p x_p(t) \end{cases} \tag{2.1}$$

where:

- $x_p \in \mathbb{R}^{n_p}$ is the state of the plant;
- $u_p \in \mathbb{R}^{r_p}$ is the known plant input;
- $d_p \in \mathbb{R}^{q_p}$ is the unknown plant input, given by the *load*;
- $y_p \in \mathbb{R}^{m_p}$ is the plant output;
- A_p, B_p, E_p, C_p are matrices respectively $\in \mathbb{R}^{n_p \times n_p}$ $\mathbb{R}^{n_p \times r_p}$, $\mathbb{R}^{n_p \times q_p}$, $\mathbb{R}^{m_p \times n_p}$;

The system (2.1) is controlled by a controller able to achieve the tracking of a specified *reference signal* $r(t)$:

$$y_{p,i}(t) = r(t), \text{ where } i \text{ represents the output signal to be controlled}$$

The controller can be represented in state-space as:

$$\begin{cases} \dot{x}_c(t) = A_c x_c(t) + B_c u_c(t), \\ y_c(t) = C_c x_c(t) + D_c u_c(t), \end{cases} \tag{2.2}$$

where:

- $x_c \in \mathbb{R}^{n_c}$ is the state of the controller;
- $u_c = \begin{bmatrix} r(t) \\ y_p(t) \end{bmatrix} \in \mathbb{R}^{r_c}$ is the controller input;
- $y_c = u_p \in \mathbb{R}^{m_c}$ is the controller output;
- A_c, B_c, C_c, D_c are matrices respectively $\in \mathbb{R}^{n_c \times n_c}$, $\mathbb{R}^{n_c \times r_c}$, $\mathbb{R}^{m_c \times n_c}$, $\mathbb{R}^{m_c \times r_c}$;

As already stated, systems (2.1) and (2.2) represent a *DCS* and an external attack could involve any different signal they receive and/or send, i.e.: $y_p(t), r(t), y_c(t)$.

To be able to detect and identify the kind of attack the system is receiving, an observer has to be built. So, it is necessary to build the state-space representation of the total system composed by (2.1) and (2.2) [9]:

$$\begin{cases} \dot{x}_{tot}(t) = A_{tot}x_{tot}(t) + B_{tot}u_{tot}(t) + E_{tot}d_{tot}(t), \\ y_{tot}(t) = C_{tot}x_{tot}(t), \end{cases} \tag{2.3}$$

where:

- $x_{tot} = \begin{bmatrix} x_p(t) \\ x_c(t) \end{bmatrix} \in \mathbb{R}^{n_{tot}}$, with $n_{tot} = n_p + n_c$, is the state of the total system composed by the plant and the controller;
- $u_{tot} = r(t) \in \mathbb{R}^{r_{tot}}$ is the total system input;
- $d_{tot} \in \mathbb{R}^{q_{tot}}$, with $q_{tot} = q_p$, as well as $d_{tot} = d_p$ is the total system unknown input;
- $y_{tot} = \begin{bmatrix} y_p(t) \\ y_c(t) \end{bmatrix} \in \mathbb{R}^{m_{tot}}$, with $m_{tot} = m_p + m_c$ is the total system output;
- A_{tot}, B_{tot}, C_{tot}, D_{tot} are matrices respectively $\in \mathbb{R}^{n_{tot} \times n_{tot}}$, $\mathbb{R}^{n_{tot} \times r_{tot}}$, $\mathbb{R}^{m_{tot} \times n_{tot}}$, $\mathbb{R}^{m_{tot} \times r_{tot}}$;

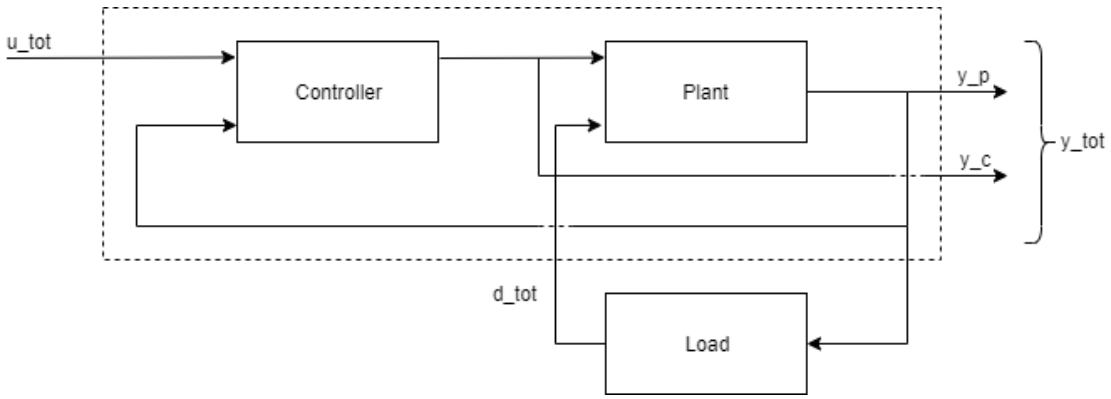


Figure 2.1: Model of system (2.3)

2.3 Concept of the attack identification system

Here a brief explanation of the concept of the project is presented. The methodology takes its basis from the *Fault Detection* literature, and evolves them in order to achieve the correct attack identification. Therefore, the general approach is still

the usage of observers in order to reconstruct the *nominal* correct state of the system under attack, so to check if there are differences between the observers and system outputs. In this thesis approach, the next step is to build a bank of different observers, each one able to reconstruct the state of the system under different attacks. In other words, every different observer contains into its parameters the information of one of the possible attacks, so that when that specific attack occurs, its output produce no residual errors with respect to the system output.

Once the bank of observers is built, an algorithm able to identify the correct attack occurring based on the different residuals each observer produce, is needed. The design of the bank od observers and of the algorithm, as well as the reasonings and preliminar analysis that led to the final results are explained in *Part III*.

Part II

Theory

Chapter 3

Residual Computation

3.1 Observer Design

As seen in *Part I*, in order to detect and identify an attack, an *observer* is needed, able to estimate the state x_{tot} of system (2.3). Since (2.3) presents an unknown input, is not possible to use a *Luenberger observer*, but a more advanced *Unknown Input observer* has to be designed [10]:

$$\begin{cases} \dot{z}_{obs} = F_{obs} z_{obs} + T_{obs} B_{tot} u_{tot} + K_{obs} y_{tot}, \\ \hat{x}_{tot} = z_{obs} + H_{obs} y_{tot}, \\ y_{obs} = C_{obs} \hat{x}_{tot} \end{cases} \tag{3.1}$$

where:

- $z_{obs} \in \mathbb{R}^{n_{obs}}$, with $n_{obs} = n_{tot}$ is the internal state of the observer;
- F_{obs}, T_{obs}, K_{obs}, H_{obs} are matrices to be built in order to effectively estimate x_{tot}. They respectively $\in \mathbb{R}^{n_{obs} \times n_{obs}}$, $\mathbb{R}^{n_{tot} \times n_{tot}}$, $\mathbb{R}^{n_{obs} \times m_{tot}}$, $\mathbb{R}^{n_{tot} \times m_{tot}}$;
- $C_{obs} \in \mathbb{R}^{m_{tot} \times n_{tot}}$ is needed in order to choose which states output;

A good design of the observer matrices make it able to achieve the condition

$$\lim_{t \to \infty} y_{obs}(t) = y_{tot}(t)$$

in a finite time. Of course, the less is t, the better, as long as other unwanted effect arises, such as, for examples, peaks during the tranistional phases, and so on. In order to detect an attack, the residual

$$e(t) = y_{obs}(t) - y_{tot}(t)$$

has to be checked according to the following relation:

$$\begin{cases} |e(t)| < threshold, \text{if system (2.3) is not under attack} \\ |e(t)| > threshold, \text{if system (2.3) is under attack} \end{cases}$$

The *threshold* variable has to be chosen *a-priori* according to the system in order to maximize the ratio $\frac{attacks\ identified}{attacks\ to\ (2.3)}$. The decision of the value is up to the designer.

This setup, however, can only detect the presence or not of an attack, but it is not able to distinguish between the different kind of attacks, or the different communications that have been attacked. To achieve this goal, is necessary to build differents *UIOs* in order to reconstruct the correct x_{tot} in the system, under different attacks.

3.2 UIO for system (2.3) under attack

In the case of an attack, the *UIO* (3.1) has to be modified in order to achieve the condition

$$\lim_{t\to\infty} y_{obs} = y_{tot,\ under\ attack} \tag{3.2}$$

To achieve (3.2), an additional input u_{obs} is given to the observer, representing the model of the attack, defined *a-priori*:

$$\begin{cases} \dot{z}_{obs} = F_{obs,att} z_{obs} + T_{obs,att} \begin{bmatrix} B_{tot} & B_{obs,att} \end{bmatrix} \begin{bmatrix} u_{tot} \\ u_{obs} \end{bmatrix} + K_{obs,att} y_{tot}, \\ \hat{x}_{tot} = z_{obs} + H_{obs,att} y_{tot}, \\ y_{obs} = C_{obs} \hat{x}_{tot} \end{cases} \tag{3.3}$$

where:

- $B_{obs,att} \in \mathbb{R}^{n_{obs}}$, $F_{obs,att}$, $T_{obs,att}$, $K_{obs,att}$, $H_{obs,att}$ have to be constructed in order to reflects the behaviour of the system (2.3) under attack;

3.2.1 Model of the attack

As seen, the *UIO* (3.3) need as input the model of the attack. This means that we have to provide to the observer an *a-priori* defined signal that reproduce the possible attack that (2.3) can receive, in terms of amplitude, kind of signal (step, ramp, sine wave, ...) and frequency. Depending on the *a-priori* knowledge of the designer on its system and its possible attacks, is up to him the decision of the number and the kind of model of attack to build. Of course, a tradeoff is needed. While a greater number of different attack model is better from an accuracy of the *attack identification system* point of view, on the other hand it increases

the computational time and the complexity of the decisional algorithm. Different analysis and simulation can help to decide the correct number of attacks.

In this thesis work the model of attack has been set just to one kind. The reason of that and the results this decision led to are better showed in *Part III*.

3.3 Bank of UIOs

In section 3.2 is shown how to construct an *UIO* for the system (2.3) under attack. But to successfully identify and detect all the kinds of different attacks, a bank of $n+1$ observers is needed, where n represents the maximum number of different communications that can be attacked, plus one additional *UIO* for the nominal case. It has to be considered that also the *bank of UIOs* is an embedded system that communicates remotely with (2.3), so that also its communications can be attacked. Take further notes that the computation of the residuals and of the decisional algorithm also takes place in another embedded system, with its own communications, and thus with its own possible sources of attacks.

After having defined the number of different possible communication attacked n, it is necessary to build the bank of $n+1$ *UIOs*, each one able to estimate correctly the state $x_{tot,i}$, where $i = 1,2,\ldots,n+1$ represents the different attack that (2.3) is receiving (when $i = 1$ the system is in nominal case).
The bank is here represented:

$$
\begin{aligned}
i = 1 \quad & \begin{cases} \dot{z}_{obs,i} = F_{obs,i} z_{obs,i} + T_{obs,i} B_{tot} u_{tot} + K_{obs,i} y_{tot}, \\ \hat{x}_{tot,i} = z_{obs,i} + H_{obs,i} y_{tot}, \\ y_{obs,i} = C_{obs} \hat{x}_{tot,i} \end{cases} \\
i = 2 \quad & \begin{cases} \dot{z}_{obs,i} = F_{obs,i} z_{obs} + T_{obs,i} \begin{bmatrix} B_{tot} & B_{obs,i} \end{bmatrix} \begin{bmatrix} u_{tot} \\ u_{obs} \end{bmatrix} + K_{obs,i} y_{tot}, \\ \hat{x}_{tot,i} = z_{obs,i} + H_{obs,i} y_{tot}, \\ y_{obs,i} = C_{obs} \hat{x}_{tot,i} \end{cases} \\
& \vdots \\
i = n+1 \quad & \begin{cases} \dot{z}_{obs,n+1} = F_{obs,n+1} z_{obs} + T_{obs,n+1} \begin{bmatrix} B_{tot} & B_{obs,n+1} \end{bmatrix} \begin{bmatrix} u_{tot} \\ u_{obs} \end{bmatrix} + K_{obs,n+1} y_{tot}, \\ \hat{x}_{tot,n+1} = z_{obs,n+1} + H_{obs,n+1} y_{tot}, \\ y_{obs,n+1} = C_{obs} \hat{x}_{tot,n+1} \end{cases}
\end{aligned}
\tag{3.4}
$$

3.4 Residuals computation

With the bank of observer (3.4) connected to the system (2.3), a total of $m_{tot}(n+1)$ residuals has to be computed. In fact, for every $n+1$ observer (i.e., different possible attack) , a total of m_{tot} residual can be computed. The next step is to choose, between the $(n+1)$ sets of m_{tot} residuals, which represents the attack in progress. At every instant t, a matrix

$$E_{res}(t) = \begin{bmatrix} e(t)_{1,1} & \dots & e(t)_{1,j} & \dots & e(t)_{1,m_{tot}} \\ \vdots & & \vdots & & \vdots \\ e(t)_{i,1} & \dots & e(t)_{i,j} & \dots & e(t)_{i,m_{tot}} \\ \vdots & & \vdots & & \vdots \\ e(t)_{n+1,1} & \dots & e(t)_{n+1,j} & \dots & e(t)_{n+1,m_{tot}} \end{bmatrix} \in \mathbb{R}^{(n+1)\times m_{tot}}$$

can be built, where $i = 1, \dots, (n+1)$ represents the kind of attacks the system can receive plus the nominal case, and $j = 1, \dots, m_{tot}$ represents the different residual we can compute for every i^{th} attack, since we have m_{tot} outputs.

Then, in order to take note of the history of the error, the integral of the square of the residual is performed. The square operation is needed to avoid sign discordance. This way, we transform the residuals into an increasing monotone function. The new matrix E_{res} becomes:

$$E_{res}(t) = \begin{bmatrix} \int_0^t e(t)_{1,1}^2\, dt & \dots & \int_0^t e(t)_{1,j}^2\, dt & \dots & \int_0^t e(t)_{1,m_{tot}}^2\, dt \\ \vdots & & \vdots & & \vdots \\ \int_0^t e(t)_{i,1}^2\, dt & \dots & \int_0^t e(t)_{i,j}^2\, dt & \dots & \int_0^t e(t)_{i,m_{tot}}^2\, dt \\ \vdots & & \vdots & & \vdots \\ \int_0^t e(t)_{n+1,1}^2\, dt & \dots & \int_0^t e(t)_{n+1,j}^2\, dt & \dots & \int_0^t e(t)_{n+1,m_{tot}}^2\, dt \end{bmatrix} \in \mathbb{R}^{(n+1)\times m_{tot}} \tag{3.5}$$

The final step consists in finding an algorithm able to detect the current attack based on $E_{res}(t)$.

3.4.1 Synthetic index

Before moving to the detection and identification algorithm, a preliminary step on the residuals has to be done. Up to now, the residuals computation give us a matrix with different column, each one representing the residual of one of the output on which we compute them. With this situation, the attack identification may be more difficult since we have m_{tot} different index to analyze, that can give us different information on what is going on in the system.

In order to simplify the identification algorithm, without losing the information each j^{th} residual gives, a syntetic index a_i able to resume all that is computed. It is

simply defined as the norm of the m_{tot} residual for each i^{th} observer:

$$a_i = \sqrt{\left(\int_0^t e(t)_{i,1}^2\, dt\right)^2 + \ldots + \left(\int_0^t e(t)_{i,j}^2\, dt\right)^2 + \ldots + + \left(\int_0^t e(t)_{i,m_{tot}}^2\, dt\right)^2}$$

This way, we can rewrite another time the matrix E_{res} used in the attack identification as:

$$E_{res}(t) = \begin{bmatrix} \sqrt{(\int_0^t e(t)_{1,1}^2\, dt)^2 + \ldots + (\int_0^t e(t)_{1,j}^2\, dt)^2 + \ldots + (\int_0^t e(t)_{1,m_{tot}}^2\, dt)^2} \\ \vdots \\ \sqrt{(\int_0^t e(t)_{i,1}^2\, dt)^2 + \ldots + (\int_0^t e(t)_{i,j}^2\, dt)^2 + \ldots + (\int_0^t e(t)_{i,m_{tot}}^2\, dt)^2} \\ \vdots \\ \sqrt{(\int_0^t e(t)_{n+1,1}^2\, dt)^2 + \ldots + (\int_0^t e(t)_{n+1,j}^2\, dt)^2 + \ldots + (\int_0^t e(t)_{n+1,m_{tot}}^2\, dt)^2} \end{bmatrix} \tag{3.6}$$

where $E_{res} \in \mathbb{R}^{(n+1)}$.
The next step consists in defining an algorithm able to detect and succesfully identify the correct communication attacked. This will be explained in the next chapter.

Chapter 4

Attack Identification

The *Attack Identification* process is made up of two different phases, a *Detection phase* and an *Identification phase*.

4.1 Detection phase

The detection phase is directly taken from the basis of the *Fault Detection* literature, and consists in a check on $E_{res,1}$, since when $i = 1$, the residuals concern the *nominal observer*, i.e.: the observer designed to reconstruct the correct x_{tot} state when no attack is present on (2.3), integrated with the output of a *system copy*,i.e: a state-space representation of (2.3). As already seen in section 3.1, the idea is to simply check if the computed index overcome an *a-priori* defined threshold. The index used for the detection, defined as E_{det} is computed as:

$$E_{det} = \|[E_{res,1}(t),\ E_{res,copy}(t)]\|$$

where $E_{res,copy}(t)$ is the vector that contains the integrals of the errors of the system copy with respect to the real system:

$$E_{res,copy}(t) = \begin{bmatrix} \int_0^t (y_{tot,1}(t) - y_{copy,1}(t))^2\, dt \\ \vdots \\ \int_0^t (y_{tot,j}(t) - y_{copy,j}(t))^2\, dt \\ \vdots \\ \int_0^t (y_{tot,m_{tot}}(t) - y_{copy,m_{tot}}(t))^2\, dt \end{bmatrix}^\intercal$$

$$\begin{cases} E_{det}(t) < threshold, \text{if system (2.3) is not under attack} \\ E_{det}(t) > threshold, \text{if system (2.3) is under attack} \end{cases}$$

The definition of the threshold is up to the designer, that has to take into account noise, model uncertainties, false alarm rates, and so on.

If this first phase detect an attack, the second phase is triggered, which is the core of the project. In fact, the second phase consists in the identification algorithm, that has to correctly identify the attacked communication.

4.2 Identification phase

The identification phase is much more complex with respect to the detection one, because we have to discern between the different possible attack that the system can receive.

After various test and analysis on the different possible method to use, which will be better explained in *Part III*, the algorithm implemented consisted in a *decision tree*. Before entering in the explanation of the decision tree algorithm used for the project, a brief introduction to what is a decision tree is presented.

4.2.1 What is a decision tree?

A *decision tree* is a computational intelligence model able to predict responses to data [11], easily expressed in the form of a set of logical rules describing the decision functions [12].

It has, as its name suggests, a tree-like structure, where the branches continues to split up depending on the data, until we reach a *leaf*, i.e.: the prediction that the decision tree is making (the output). Figure 4.1, taken from [13] shows an example of a decision tree.

The classification process is made up of two phases: a *training* one, where the decision tree model is built up on a training set, and a *prediction* phase, in which the decision tree is used to predict data classes. The training set should be selected randomly, and each data has to be labeled into a specific class. During the prediction phase, the labels are the output of the decision tree, while the observation data are the inputs. The second phase must be done on a different set with respect to the training one, in order to verify the efficiency in prediction. Another important aspect to keep in mind when designing a decision tree is the *overfitting* problem. When the decision tree predicts also the noise (it takes into consideration also non important data), it is said that it is overfitting, i.e.: the decision tree is so efficient on the training set that it can not predict correctly others data that are not part of that set, making it useless in a real application. To avoid overfitting it is possible to use a χ^2 test to understand which leaves and/or branches can be cut off. [14]

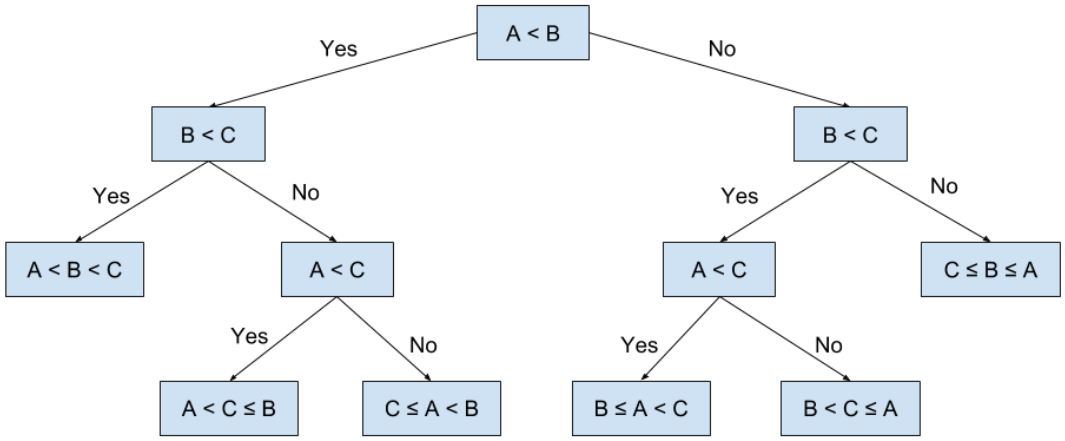


Figure 4.1: Example of a decision tree [13]

4.2.2 Residuals reset and time dependency

Before moving to the decision tree design explanation, it is really important to notice the time dependency of E_{res}. In fact, since the residuals are then integrated, their value is directly dependent on the time window used to compute them. This can nullify the usage of a decision tree to make the identification, since the data would never be consistent between them as the values of E_{res} would grow constantly.

In order to solve this, a reset of the initial condition of the integral is performed every *a-priori* defined time period. This way the computation of the index used for the identification will ever be done in the same time window, thus avoiding different values depending on the observation time. So, E_{res} (3.6) becomes:

$$E_{res}(t) = \begin{bmatrix} \sqrt{(\int_{t_i}^{t_f} e(t)_{1,1}^2\, dt)^2 + \ldots + (\int_{t_i}^{t_f} e(t)_{1,j}^2\, dt)^2 + \ldots + (\int_{t_i}^{t_f} e(t)_{1,m_{tot}}^2\, dt)^2} \\ \vdots \\ \sqrt{(\int_{t_i}^{t_f} e(t)_{i,1}^2\, dt)^2 + \ldots + (\int_{t_i}^{t_f} e(t)_{i,j}^2\, dt)^2 + \ldots + (\int_{t_i}^{t_f} e(t)_{i,m_{tot}}^2\, dt)^2} \\ \vdots \\ \sqrt{(\int_{t_i}^{t_f} e(t)_{n,1}^2\, dt)^2 + \ldots + (\int_{t_i}^{t_f} e(t)_{n,j}^2\, dt)^2 + \ldots + (\int_{t_i}^{t_f} e(t)_{n,m_{tot}}^2\, dt)^2} \end{bmatrix} \tag{4.1}$$

where $t_i < t < t_f$; t_i represents the time at which the computation of the integral starts, while t_f represents the end of the computation. This means, that in order to make possible the utilization of the decision tree in the real world, the E_{res} computation must be done every T seconds, where $T = t_f - t_i$. This way, a the end

of the period, we end up having $E_{res}(t_f)$, that will be used by the decision tree for the identification of the attack.

All this means that is possible to identify the attack only at the end of each period. It goes without saying, then, that the definition of the time period is really important, and must be done after having analized the tradeoff between decision tree accuracy, and speed of the identification system.

4.2.3 Decision tree design

As seen in 4.2.1, to design an effective decision tree able to identify the attack, first of all we need to build a suitable training set. The training set must carefully computed, to reach a tradeoff between an effective decision tree, and the risk of *overfitting*.

The *training set* is computed running the system under different situations and saving the results given by it. Before that, however, the designer of the system has to decide what to use as data for training. In fact, in section 4.2.2 we have analized the time dependency of E_{res} and its reset after T seconds. This bring us to the question of what data utilize as training. The simplest possibility could be to just use the last value computed, i.e.: $E_{res}(t_f)$. This situatuon, however, make the system strongly dependent on the time period decided for the residual computation. So, other possibilities can be chosen, such as the mean of the values, the median, or even all these grouped. In *PART III*, where an application to a DC-motor is studied, the difference between these possibilities is better analyzed.

So, when the decision of the index to be utilized is made, after having saved a sufficient number of data, the *training set* will be (when for example, the index utilized is the last value of the integral at time t_f):

$$training\ set = \begin{bmatrix} E_{res,1}^{\intercal}(t_f) \\ E_{res,2}^{\intercal}(t_f) \\ \vdots \\ E_{res,n_{data}}^{\intercal}(t_f) \end{bmatrix} \in \mathbb{R}^{n_{data} \times n} \tag{4.2}$$

The matrix is so composed:

- every *row* consists in a different simulation, thus in a different attack that the system (2.3) is receiving;
- n_{data} is the number of different simulation performed to build the training set;
- every *column* $i = 1,2,\dots,n$ represents the different result given by the differents n *UIOs*; note that the totale number is n and not $n+1$, since the observer for the nominal case is used for the *detection phase*.

4.3 Transient initial phase

This last section is to brief remember that at $t = 0$, the observers will produce residuals greater than zero due to the different initial condition, but that has not to be seen as an indication of the presence of an attack. To overcome this, is sufficient to start the *Detection and Identification Algorithm* at the end of this transient phases. Is worth notice, however, that this phase should last very few seconds, so this should not make any problem in a real application. Figure 4.2 shows an example of this. The data are referred to three different *UIOs*, and the reset occurs at $t = 6s$. It can be seen how the transient phase affects the behaviour of the errors.

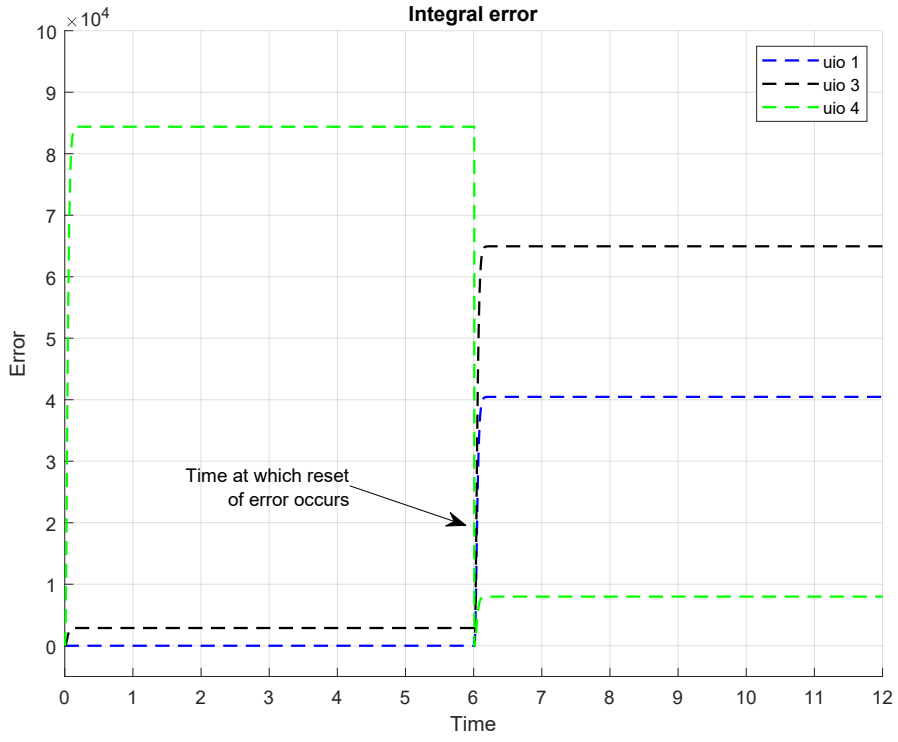


Figure 4.2: Errors behaviour example before and after the reset

Part III

Case Study

Chapter 5

DC motor

In this chapter will be presented the case study of a DC-motor to which the *Attack identification* system is applied. In the next chapter all the analisys and reasoning that led to the final results are showed. As said, the case studied is represented by a DC motor controlled by a cascade of two PI controller. Here we present the state-space representations of the different systems. In Appendix **B** the data used are presented.

5.1 Plant - DC motor

The DC motor is represented by the model (2.1). We can explicit the matrices of the system as:[15]

$$A_p = \begin{bmatrix} -\frac{R}{L} & -\frac{K_e}{L} \\ \frac{K_t}{J_m} & 0 \end{bmatrix}, B_p = \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix}, E_p = \begin{bmatrix} 0 \\ -\frac{1}{J_m} \end{bmatrix}, C_p = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.1}$$

where:

- R is the resistance in the armature circuit;
- L is the inductance in the armature circuit;
- K_e is the inductive voltage constant;
- K_t is the torque constant;
- J_m is the motor inertia;

For what concerns state, input and output we have:

$$x_p = \begin{bmatrix} i_a \\ \omega \end{bmatrix}, u_p = \begin{bmatrix} u_a \end{bmatrix}, d_p = \begin{bmatrix} T_r \end{bmatrix}, y_p = \begin{bmatrix} i_a \\ \omega \end{bmatrix} \tag{5.2}$$

where:

- i_a is the armature current of the DC-motor;
- ω is the angular speed of the DC-motor;
- u_a is the input voltage of the DC-motor;
- T_r is the load torque;

This way, (2.1) can be explictly rewritten as:

$$\begin{cases} \begin{bmatrix} \dot{i_a}(t) \\ \dot{\omega}(t) \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{K_e}{L} \\ \frac{K_t}{J_m} & 0 \end{bmatrix} \begin{bmatrix} i_a(t) \\ \omega(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} u_a(t) + \begin{bmatrix} 0 \\ -\frac{1}{J_m} \end{bmatrix} T_r(t) \\ \begin{bmatrix} i_a(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i_a(t) \\ \omega(t) \end{bmatrix} \end{cases} \tag{5.3}$$

5.1.1 Load model

The load is modeled as a system with his own inertia and friction. The equation that describe it is:

$$T_r = J_l \dot{\omega} + \beta_l \omega \tag{5.4}$$

where:

- J_l is the load inertia;
- β_l is the load friction coefficient;

5.2 Reference

The refence has been modeled as a square wave with amplitude of 75 $\frac{rad}{s}$, offset of 75 $\frac{rad}{s}$, frequency of $\frac{1}{5}$ Hz, and duty cycle of 50%, and represent the angular speed profile we want the DC-motor to follow. To avoid overshoot it is filtered by a low-pass filter with transfer function $\frac{1}{1+\frac{s}{6}}$. From now on the reference angular speed will be designed as ω_r.

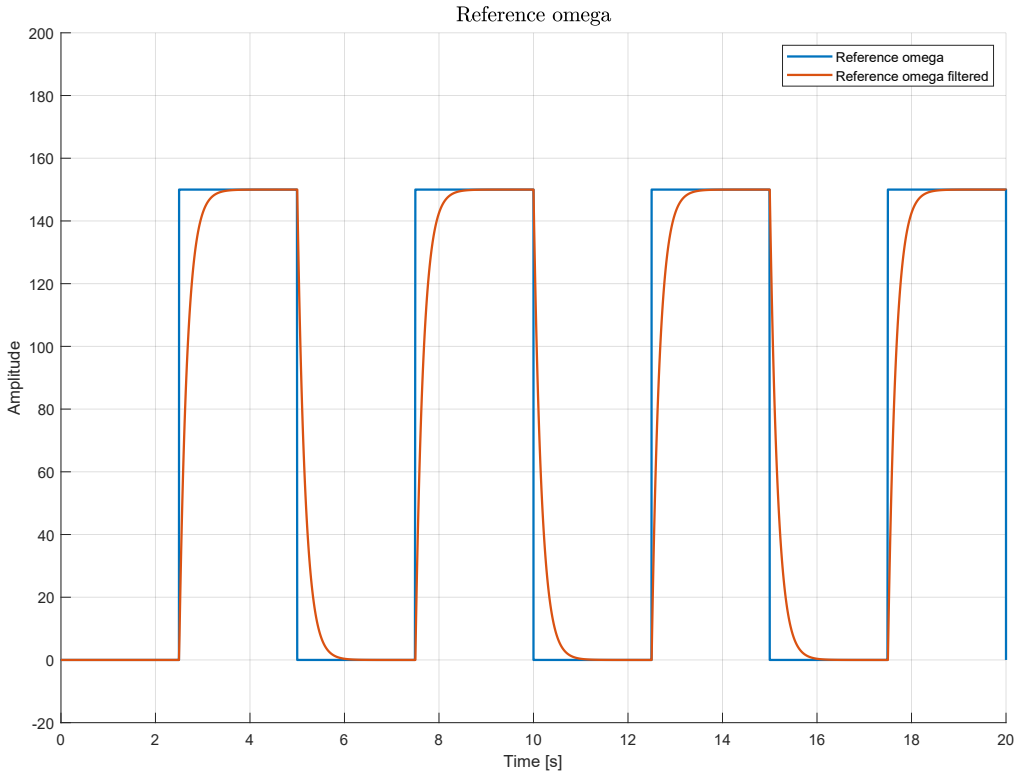


Figure 5.1: Angular speed reference for the DC-motor

5.3 PI controllers

The DC-motor is controlloed by two PI controllers in cascade. The first one operates the angular speed reference tracking, outputting the armature current control input, while the second one operates the armature current control, outputting the input voltage of the DC-motor. It is important to notice that the current has to be saturated into a specific range to avoid high temperature ranges in the motor. To do so, the output of the first PI is saturated within a range of $\pm\ 2i_{a,n}$, where $i_{a,n}$ is the nominal armature current. The scheme is represented by figure 5.2. The two state space representations of the single *PIs* are showed below. Since the system has a saturation, when the total controller state space has to be written, we need to output the armature current reference, so to saturates it externally, and then feedback it to the controller once saturated.

- **PI 1** - produces the current armature reference

$$\begin{cases} \dot{x}_{c,1} = B_{c,1}u_{c,1} \\ y_{c,1} = C_{c,1}x_{c,1} + D_{c,1}u_{c,1} \end{cases} \tag{5.5}$$

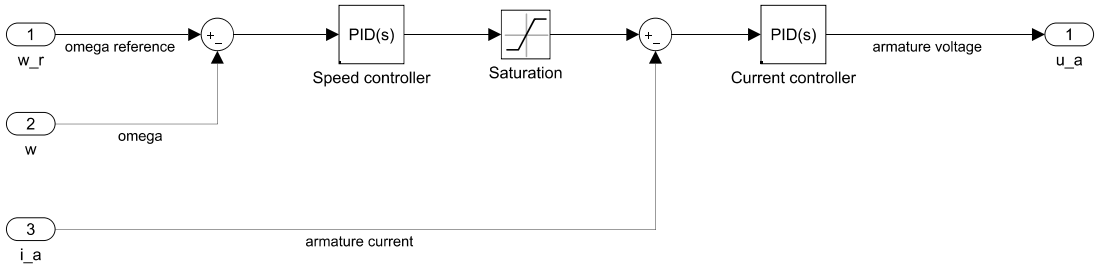


Figure 5.2: Simulink scheme of the controller

- $x_{c,1} = \int (\omega_r - \omega)$;
- $u_{c,1} = \dot{x}_{c,1} = (\omega_r - \omega)$;
- $B_{c,1} = 1; C_{c,1} = K_{i,1}; D_{c,1} = K_{p,1}$;
- $K_{p,1}$ = proportional coefficient of PI 1;
- $K_{i,1}$ = integral coefficient of PI 1;
- $y_{c,1} = i_r$, i.e: armature current that the DC-motor should have to obtain the tracking of ω_r;

- **PI 2** - produce the input voltage of the DC-motor, i.e: the control input

$$\begin{cases} \dot{x}_{c,2} = B_{c,2} u_{c,2} \\ y_{c,2} = C_{c,2} x_{c,2} + D_{c,2} u_{c,2} \end{cases} \tag{5.6}$$

  - $x_{c,2} = \int (i_{r,sat} - i_a)$;
  - $u_{c,2} = \dot{x}_{c,2} = (i_{r,sat} - i_a)$;
  - $B_{c,2} = 1; C_{c,2} = K_{i,2}; D_{c,2} = K_{p,2}$;
  - $K_{p,2}$ = proportional coefficient of PI 2;
  - $K_{i,2}$ = integral coefficient of PI 2;
  - $y_{c,2} = u_a$;

5.4 Total system

After having defined the state-space of the systems considered individually, it is possible to move to the overall state-space representation, needed for the design of

the *UIOs*[1].

$$\begin{cases} \dot{x}_{tot}(t) = A_{tot}x_{tot}(t) + B_{tot}u_{tot}(t) + E_{tot}d_{tot}(t) \\ y_{tot}(t) = C_{tot}x_{tot}(t) \end{cases} \tag{5.7}$$

$$x_{tot} = \begin{bmatrix} \int(\omega_r - \omega_m) \\ \int(i_{r,sat} - i_a) \\ i_a \\ \omega_m \\ \int u_a \\ \int i_r \end{bmatrix}, u_{tot} = \begin{bmatrix} \omega_r \\ i_{r.sat} \end{bmatrix}, d_{tot} = \begin{bmatrix} T_r \end{bmatrix}, y_{tot} = \begin{bmatrix} i_a \\ \omega_m \\ \int u_a \\ \int i_r \end{bmatrix} \tag{5.8}$$

So, we end up with 15 different possible communication attacked, that are so defined:

- 1 - output u_a of the controller; both the plant and the observer receives the wrong value;
- 2 - input u_a of the plant; it receives the wrong value, while the observer receives the correct one.
- 3 - input $\int u_a$ of the observer; it receives the wrong value, while the plant receives the correct one.
- 4 - output i_a of the plant; the controller, the observer and the *DaIS* receive the wrong value;
- 5 - input i_a of the controller; it receives the wrong value, while the observer and the *DaIS* receive the correct one;
- 6 - input i_a of the observer; it receives the wrong value, while the controller and the *DaIS* receive the correct one;
- 7 - input i_a of the *DaIS*; both the controller and the receiver receive the correct value.
- 8 - output ω of the plant; the controller, the observer and the *DaIS* receive the wrong value;
- 9 - input ω of the controller; it receives the wrong value, while the observer and the *DaIS* receive the correct one;.
- 10 - input ω of the observer; it receives the wrong value, while the controller and the *DaIS* receive the correct one;
- 11 - input ω of the *DaIS*; both the controller and the receiver receive the correct value.

[1] see Appendix **B** for the demonstration of the matrices computation.

- 12 - input $\int u_a$ of the *DaIS*; both the controller and the receiver receive the correct value.
- 13 - input ω_r of the controller; the observer receives the correct value;
- 14 - input ω_r of the observer; the controller receives the correct value;
- 15 - input i_r of the observer;

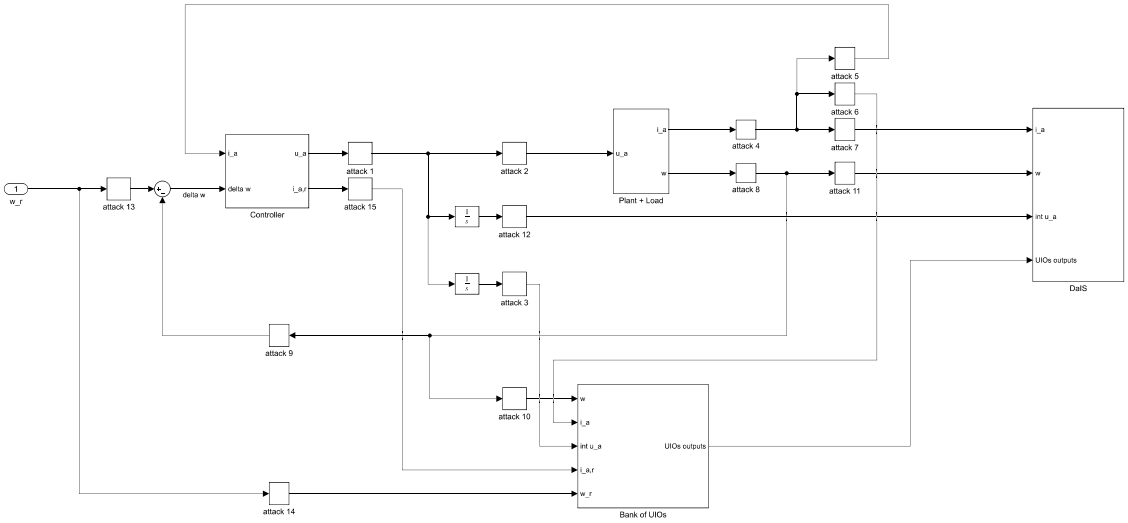


Figure 5.3: Simulink scheme of the system with the rapresentation of all the communication attacks

5.5 UIOs

Defined the overall state-space representation it is possible to design the *Unknown Input Observer*[2]

5.5.1 Nominal case observer

To design the observer for the nominal case it is not needed the model of the attack, so that its state-representation is:

$$\begin{cases} \dot{z}_{obs,1} = F_{obs,1} z_{obs,1} + T_{obs,i} B_{tot} u_{tot} + K_{obs,1} y_{tot}, \\ \hat{x}_{tot,1} = z_{obs,1} + H_{obs,1} y_{tot}, \\ y_{obs,1} = C_{obs} \hat{x}_{tot,1} \end{cases}$$

[2]see Appendix B for the matrices computation.

where:

$$y_{obs,1} = \begin{bmatrix} i_{a,obs,1} \\ \omega_{obs,1} \\ \int u_{a,obs,1} \end{bmatrix}$$

5.5.2 Attack observer

In case of the observer designed to model the attacks, it needed to input them also the model of the attack, that in our case was a continuous signal with amplitude 30.

$$\begin{cases} \dot{z}_{obs,i} = F_{obs,i} z_{obs} + T_{obs,i} \begin{bmatrix} B_{tot} & B_{obs,i} \end{bmatrix} \begin{bmatrix} u_{tot} \\ u_{obs} \end{bmatrix} + K_{obs,i} y_{tot}, \\ \hat{x}_{tot,i} = z_{obs,i} + H_{obs,i} y_{tot}, \\ y_{obs,i} = C_{obs} \hat{x}_{tot,i} \end{cases}$$

where:

- $i = \forall i \in \{2,3,\ldots,16\}$
- $u_{obs} = 30$
- $y_{obs,i} = \begin{bmatrix} i_{a,obs,i} \\ \omega_{obs,i} \\ \int u_{a,obs,i} \end{bmatrix}$

For the observer matrices computation it is possible to see [10]. For the computation of F, the eigenvalues chosen are:

$$\lambda = 3 \begin{bmatrix} -10 & -10 & -11 & -11 & -11 & -12 \end{bmatrix}$$

Chapter 6

Tests and analysis

After having defined the framework used for the analysis, in this chapter is analized the tests and reasoning that led to the final setup of the identification algorithm.

6.1 Synthetic index definition

In this section the reasoning that led to the definition of a synthetic index are showed.

It is important to notice that in these previous analysis the setup was not yet the final one, so some of residuals and the attacks modeled are different with respect to the definitive ones. Anyway, the reasonings that led to the final configuration are still valid and significative.

6.1.1 Winning Model Method

In this initial setup the attacks modeled are just the first 12, and the index used in the identification algorithm are the square and the integral of the residuals related to current, angular speed and voltage.

The first identification concept, before moving to the more complicated decision tree, was to check the *best model* at each sample time. So, each sample time the computation of the residuals is done, and the observer related to the minimum residual is considered the best. By doing this for a certain time sequence, we end up having a graph that can tell us for how much time every observer has benn the best. The winning observer for the most time is considered the observer related to the attack that the system is receiving.

$$\sum_{t=1}^{N} a_i(t),\ where\ \begin{cases} a_i(t) = 1,\ if\ e_i(t) = min\left([e_1(t), \ldots, e_n(t)]\right) \\ a_i(t) = 0,\ if\ e_i(t) \neq min\left([e_1(t), \ldots, e_n(t)]\right) \end{cases},\ \forall i = 1,2,\ldots,n$$

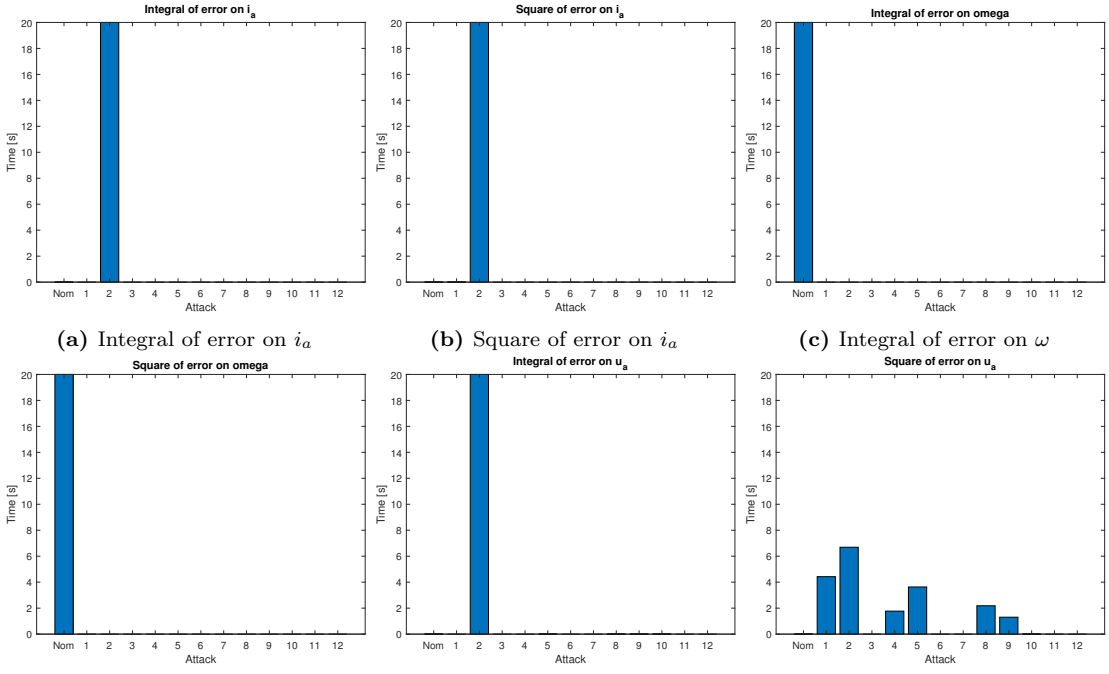
where:

- N = total number of time samples;
- n = total number of observers;
- e_i = residual related to the i^{th} observer;

The observer with the max a_i is the observer related to the attack to identify.

However, since we have different indices (six in this case), we can have different results for each one of that, so that we are unable to identify the correct attack. In figure 6.1 is showed an example of this. In that case, the system is receiving a continuous *attack 2* with amplitude 100, the same modeled in the observers. Anyway, according to the different indices, we can not tell for sure that the attack is the *attack 2*.

Another problem arises when the attack the system is receiving differs from the one modeled in the observers. In this case we may unable to identify the correct attack in any of indices, as showed in figure 6.2, where the system receives an *attack 2* with amplitude of 40, while the observers still are modeled with an attack of amplitude 100.

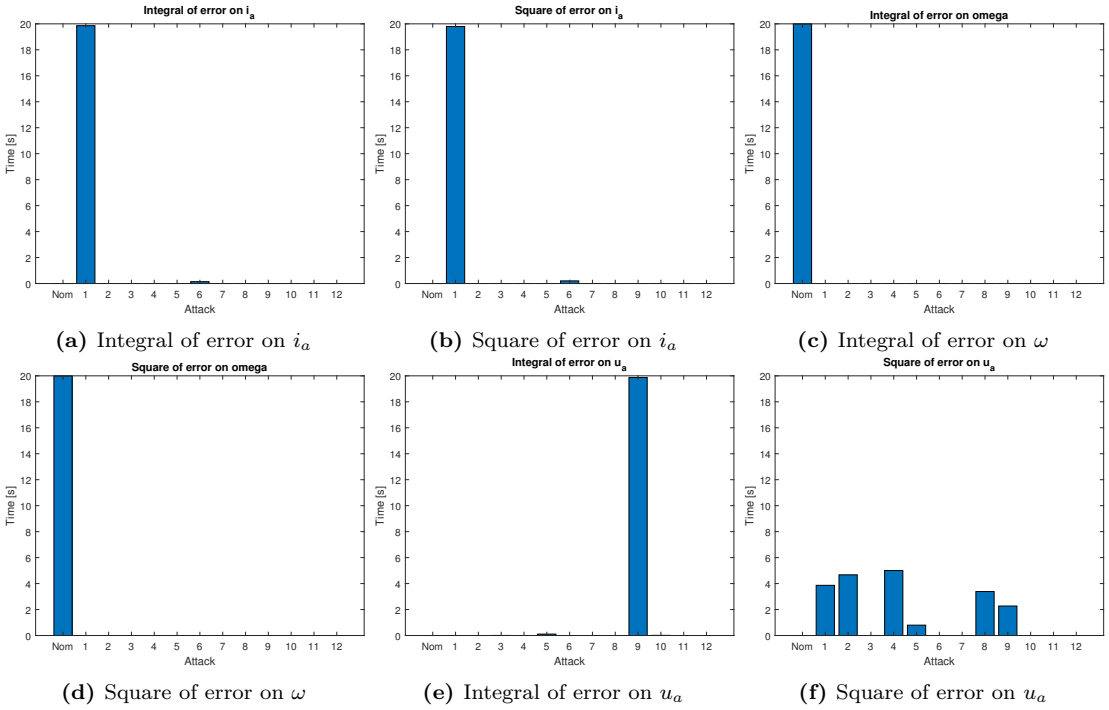


(a) Integral of error on i_a **(b)** Square of error on i_a **(c)** Integral of error on ω

(d) Square of error on ω **(e)** Integral of error on u_a **(f)** Square of error on u_a

Figure 6.1: Results of the winning model method, with attack 2 with amplitude 100, as in the attack modeled in the UIOs

In order to resume all the different possible results that this method gives, figures 6.3 and 6.4 are showed. They presents the ability to identify correctly the attack



(a) Integral of error on i_a **(b)** Square of error on i_a **(c)** Integral of error on ω

(d) Square of error on ω **(e)** Integral of error on u_a **(f)** Square of error on u_a

Figure 6.2: Results of the winning model method, with attack 2 with amplitude 40, while in the UIOs is modeled with amplitude 100

depending on the attack amplitude (*y-axis*) and on the index we are using (*different colors*). On the *x-axis* are presented the different communication attacked.

The first graph is related to the case in which the amplitude of the attack modeled in the observer is 100, i.e.: the max value simualted in the attacks, while the second one presents the case in which in the observers the attack is modeled with amplitude 50. The second one gives better overall result, so that is preferable to use a model of the attacks with the lowest possible amplitude.

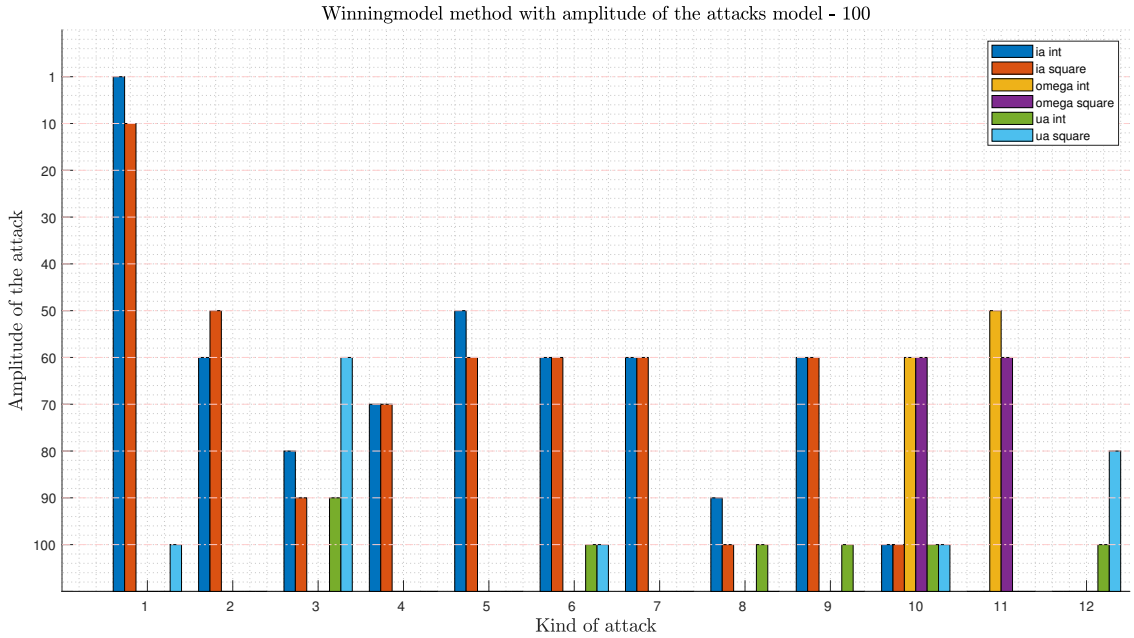


Figure 6.3: Efficiency of the winning model method with varying attack amplitude and amplitude of the model of the attack in the observers equal to 100

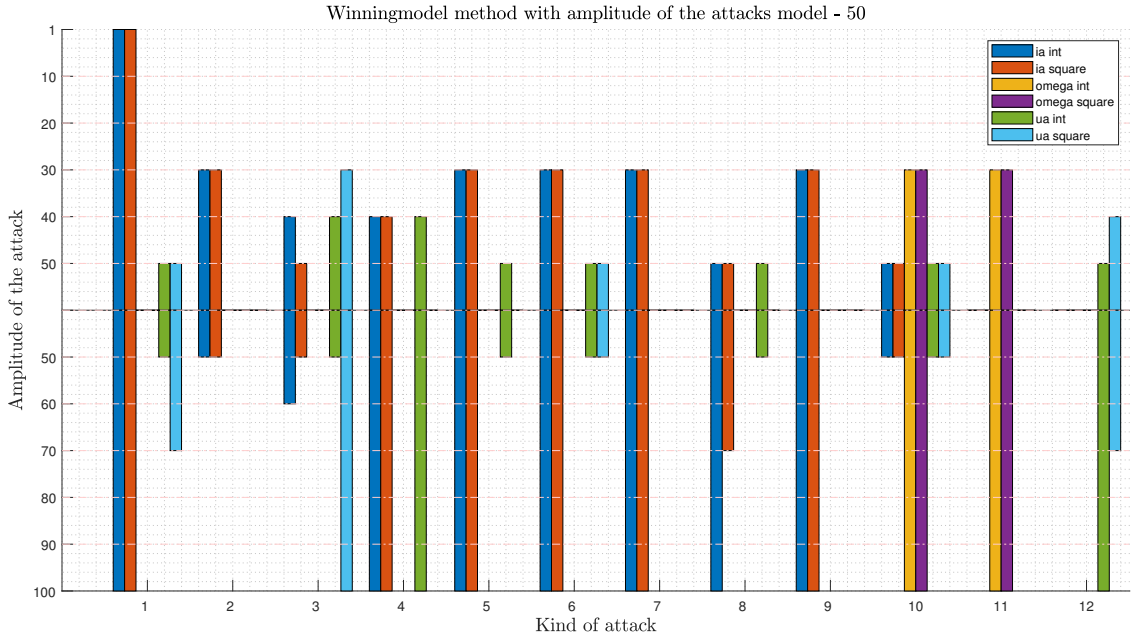


Figure 6.4: Efficiency of the winning model method with varying attack amplitude and amplitude of the model of the attack in the observers equal to 50

6.1.2 Probability Density Function method

Since the *Winning Model* method did not give satisfactory results, another method has been tried, attempting to avoid the dependency on the residuals, that give us only indications about the *amplitudes* of the errors, and trying to focus also on other aspects. To do this, a method based on the probability density function (*PDF*) of the outputs has been tried. The concept is to check the difference between the *PDFs* of the outputs of the system attacked and of the observers, so that the observer with the most similar *PDFs* with respect to the system ones, is the one corresponding to the attack.
The difference is computed through the *Bhattacharyya distance* D_B, so that[16]:

$$D_B(p,q) = \frac{1}{4}\ln\left(\frac{1}{4}\left(\frac{\sigma_p^2}{\sigma_q^2}+\frac{\sigma_q^2}{\sigma_p^2}+2\right)\right)+\frac{1}{4}\left(\frac{(\mu_p-\mu_q)^2}{\sigma_p^2+\sigma_q^2}\right)$$

where:

- μ = mean of the *PDF*;
- σ^2 = variance of the *PDF*;

So, the observer with the min value of D_B is considered the one related to the attack. Varying the amplitude of the attacks, the amplitude of the models, and looking for the different indices selected, we can produce two graphs similar to the ones of section 6.1.1.

As before, we can not identify correctly the attack in every situations, considering also that we have different results basing on which index we are checking. All this, led to the definition of a *synthetic index* able to resume the informations of the different indices we have.

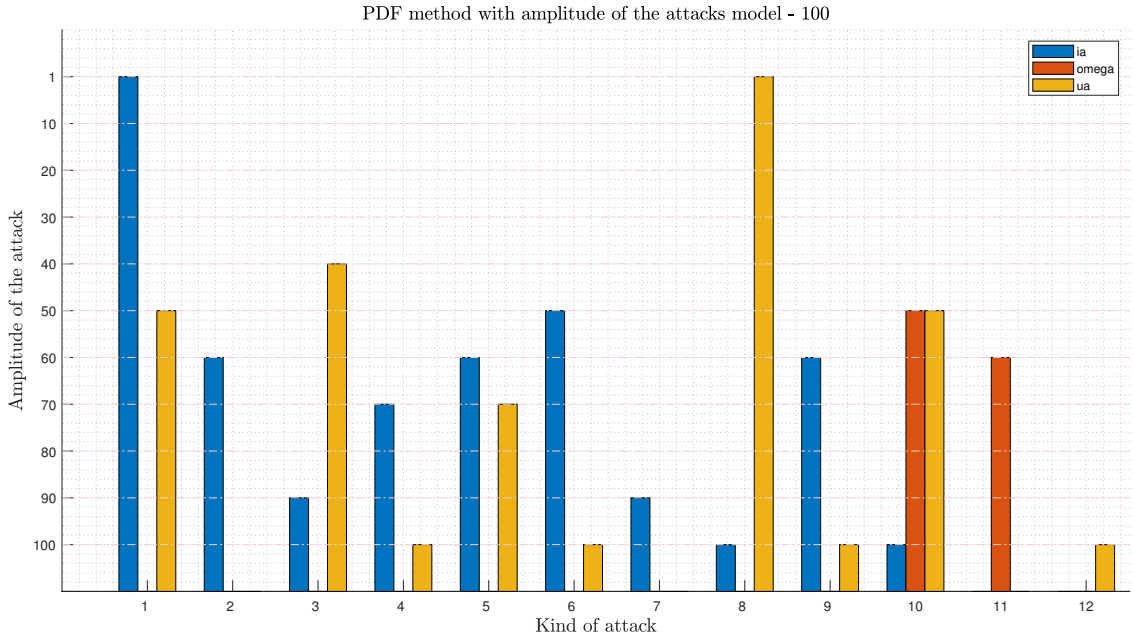


Figure 6.5: Efficiency of the PDFs method with varying attack amplitude and amplitude of the model of the attack in the observers equal to 100

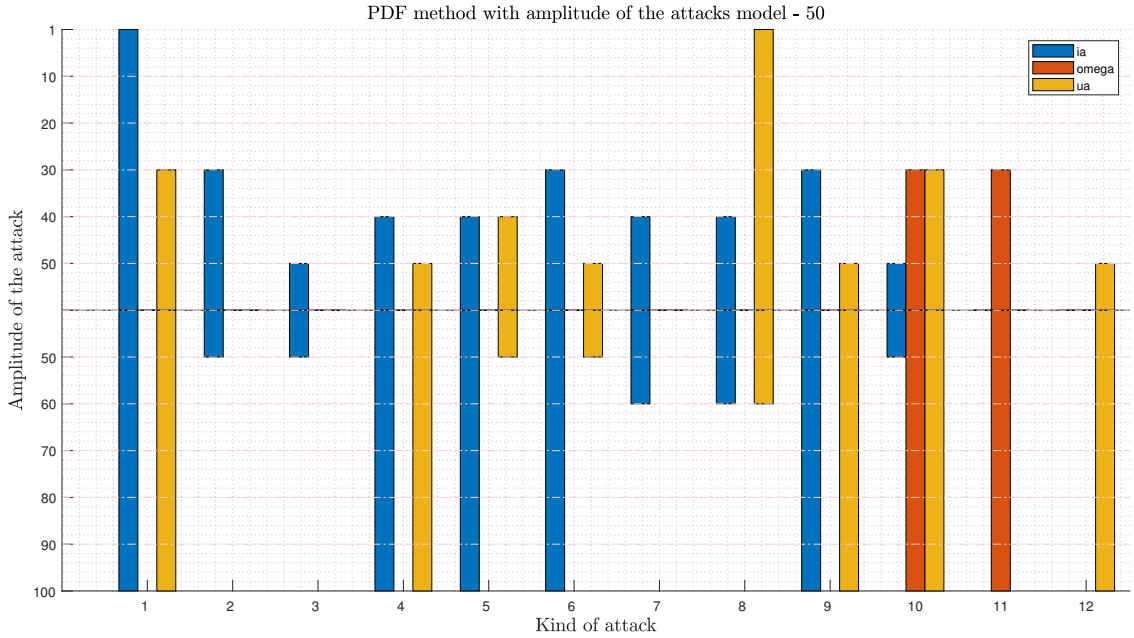


Figure 6.6: Efficiency of the PDFs method with varying attack amplitude and amplitude of the model of the attack in the observers equal to 50

6.1.3 Norm of the residuals

As seen in the previous sections, the indices alone are not able to tell us which attack the system is receiving, since they give us different indications. In order to not lose their informations but making us able to have one indication of the attack from every observer, a synthetic index is defined. Instead of looking to the residuals of the three outputs (i_a, ω, u_a) as separate entities, we compute the norm of the three residuals, as already seen in section 3.4.1:

$$E_{res,i}(t) = \sqrt{\left(\int_0^t (i_a(t) - i_{a,i}(t))^2\, dt\right)^2 + \left(\int_0^t (\omega(t) - \omega_i(t))^2\, dt\right)^2 + \left(\int_0^t (u_a(t) - u_{a,i}(t))^2\, dt\right)^2}$$

With this new definition of the index used for the identification, we sticked with the *Winning Model* method. The analysis of the robustness of this method with the new index has been done, just like the previous sections, so that figure 6.7 and 6.8 shows us the results. In this new simulations, *attack 13* and *attack 14* have been added to the possible attacks set.

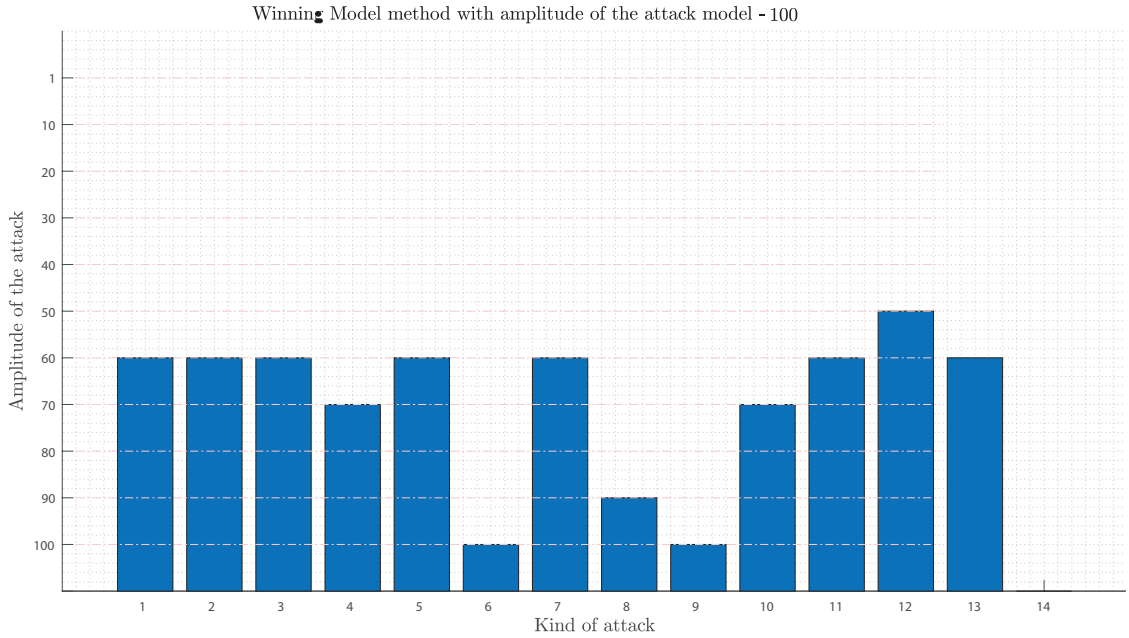


Figure 6.7: Efficiency of the Winning model method with varying attack amplitude and amplitude of the model of the attack in the observers equal to 100

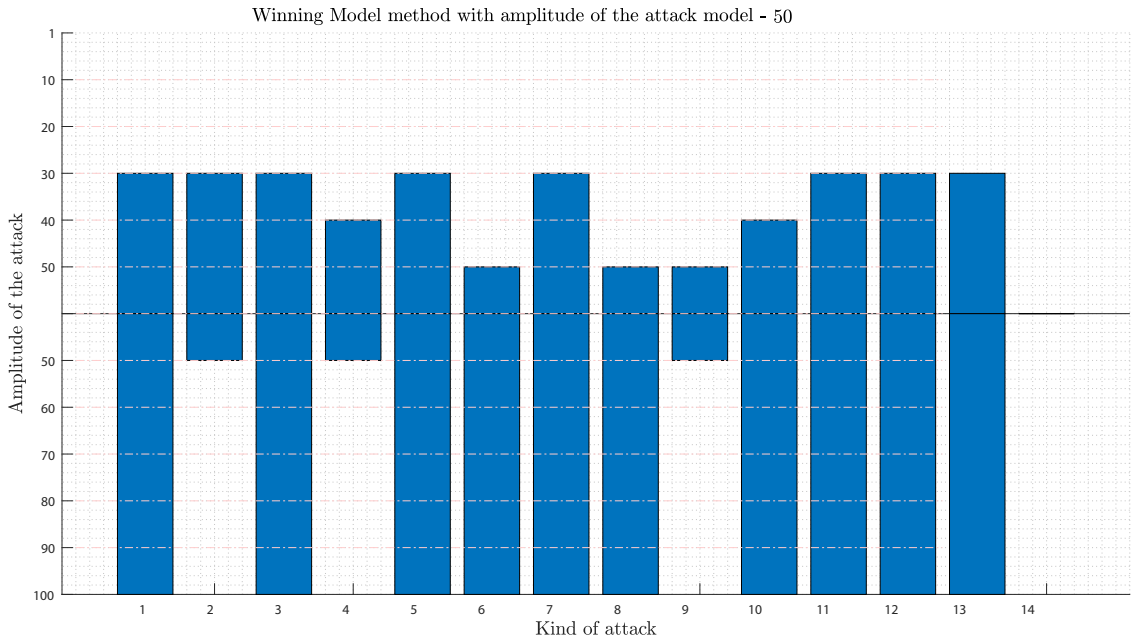


Figure 6.8: Efficiency of the Winning Model method with varying attack amplitude and amplitude of the model of the attack in the observers equal to 50

6.2 Residuals statistical analysis

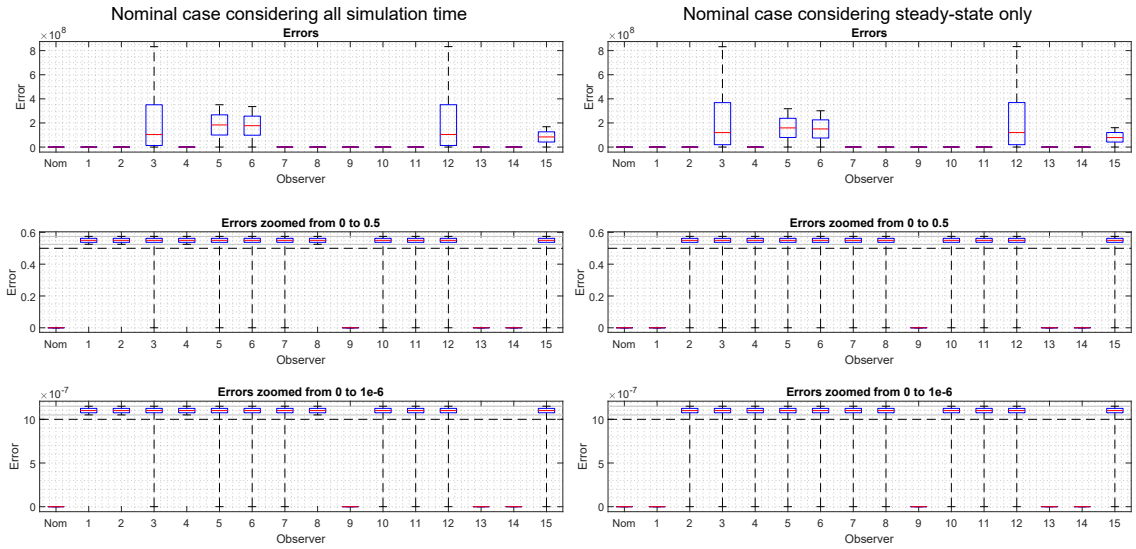
Up to this point, the methods tested are not still robust enough to make them useful in a variety of situations that could occurr in real life. So, before moving to a better algorithm able to identify the attacks, an analysis of the residuals has been performed, so to verify if it can be a good indicator for the identification, or it was preferable to move to sometingh else (like for examples, *PDFs*).

Figure 6.9 shows the boxplot analysis of the residuals. They were produced simulating each of the possible attacks (this time finally all the 15 attacks are modeled) plus the nominal case, in order to check if the residuals of the corresponding observerd did goes to zero as espected, while the others not. For each attack two different graph were produced, the first one considering all te simulation time (100 s), while the second one considering only the observation period from the end of the transient phase (from 5 to 100 s).

The results were satisfactory since showed that the residuals is a good indicator of the correctness of the observer. Only in some cases there is a certain grade of uncertainty, due to the fact that the observers produce the same outputs in these cases. This is the situation of the nominal case and the attacks 1, 9, 13 and 14. We will see how to deal with them in the next sections.

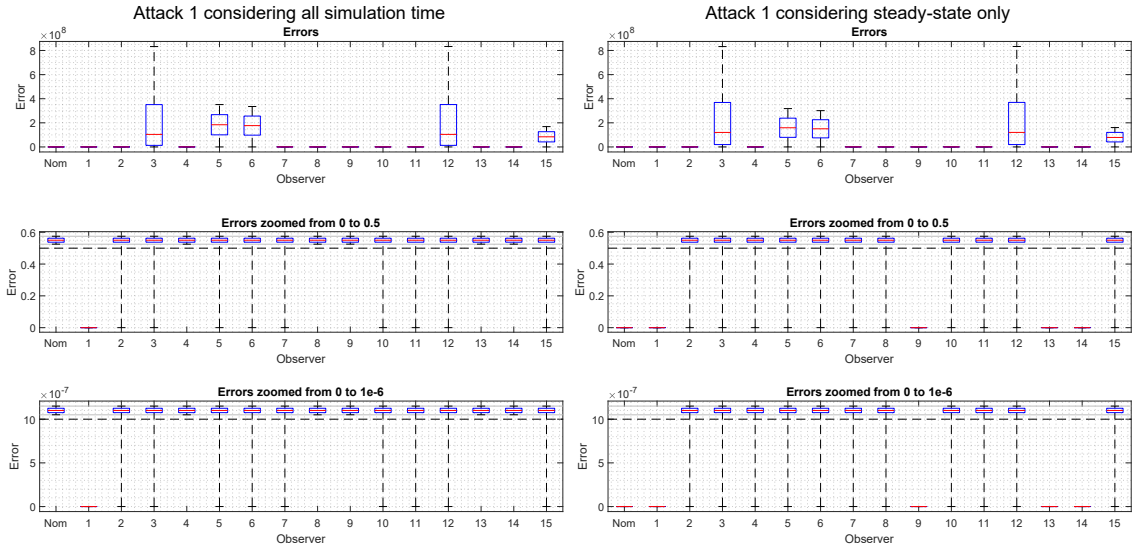
Another important thing that came up is the need to check the residual only in the steady-state situations. This is due to the fact that we need some time for the residuals to go to zero due to the different initial condition, and this is visible

because of the integral of the residuals. That is the experimental proof of the need of the integral reset, as showed in *Part II*.



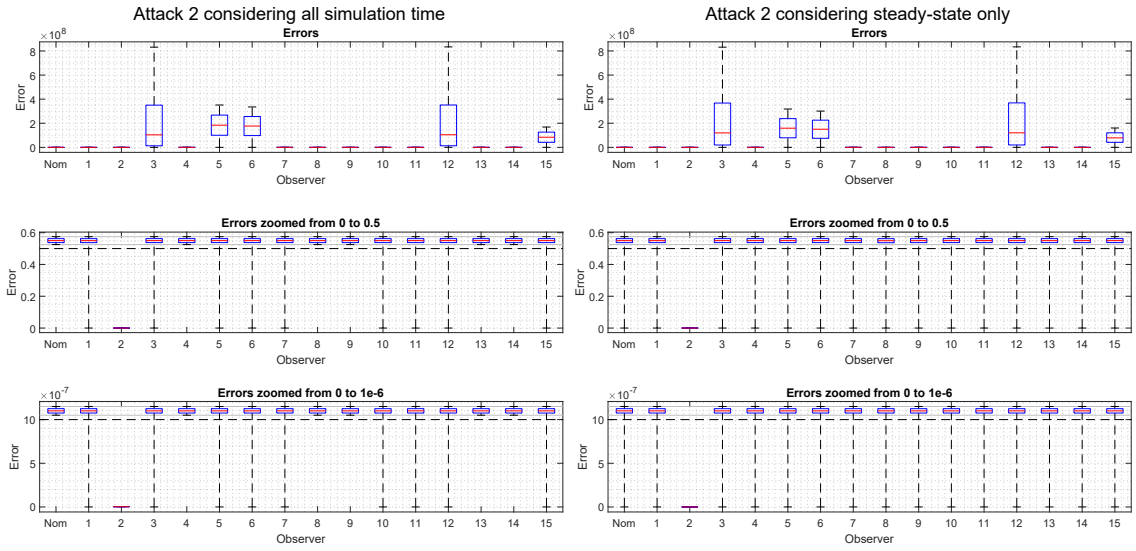
(a1) **(a2)**

(a) Nominal case



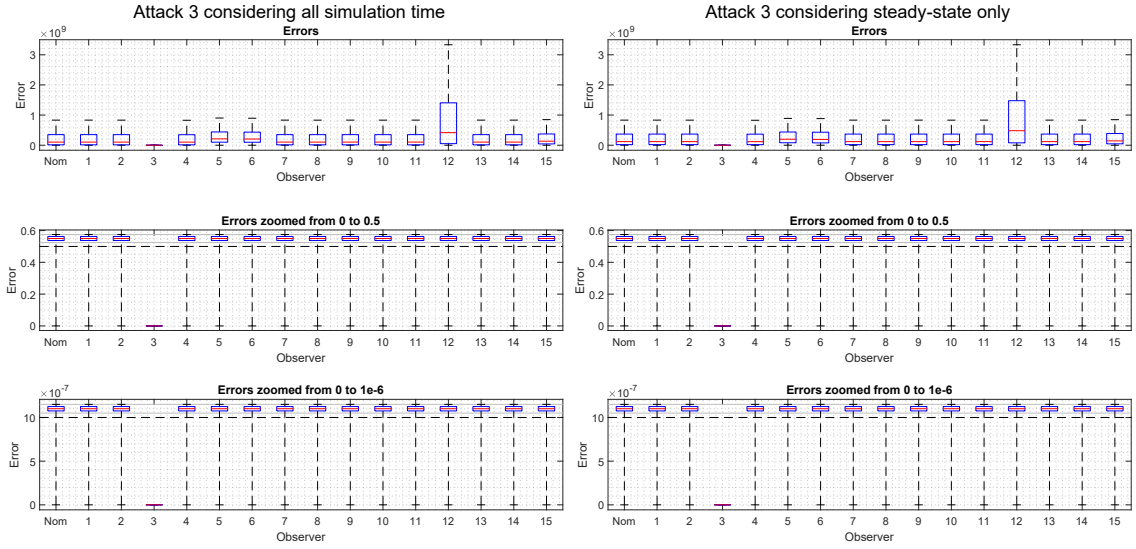
(b1) **(b2)**

(b) Attack 1



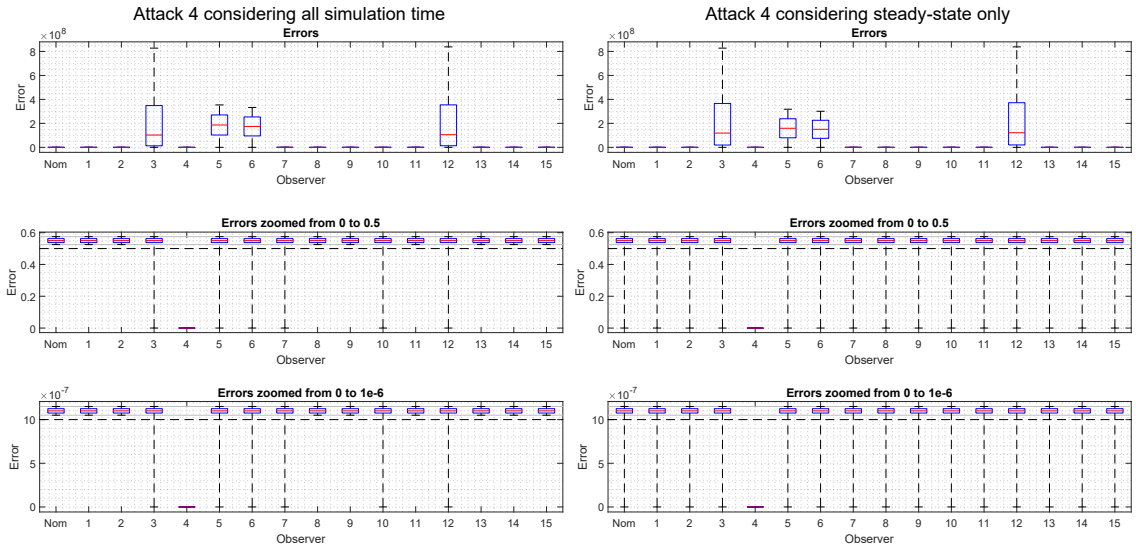
(c1) **(c2)**

(c) Attack 2



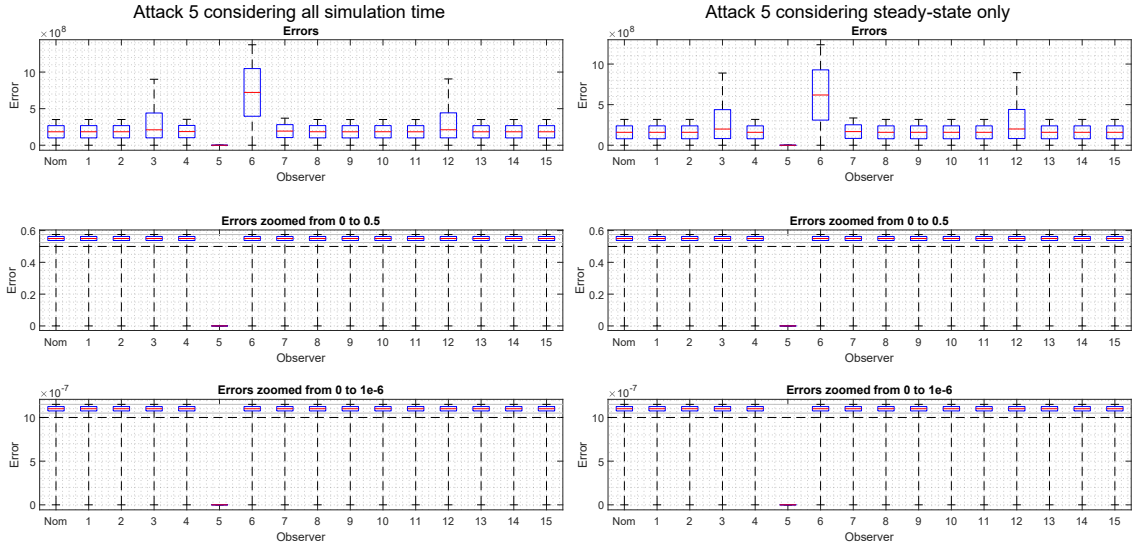
(d1) **(d2)**

(d) Attack 3



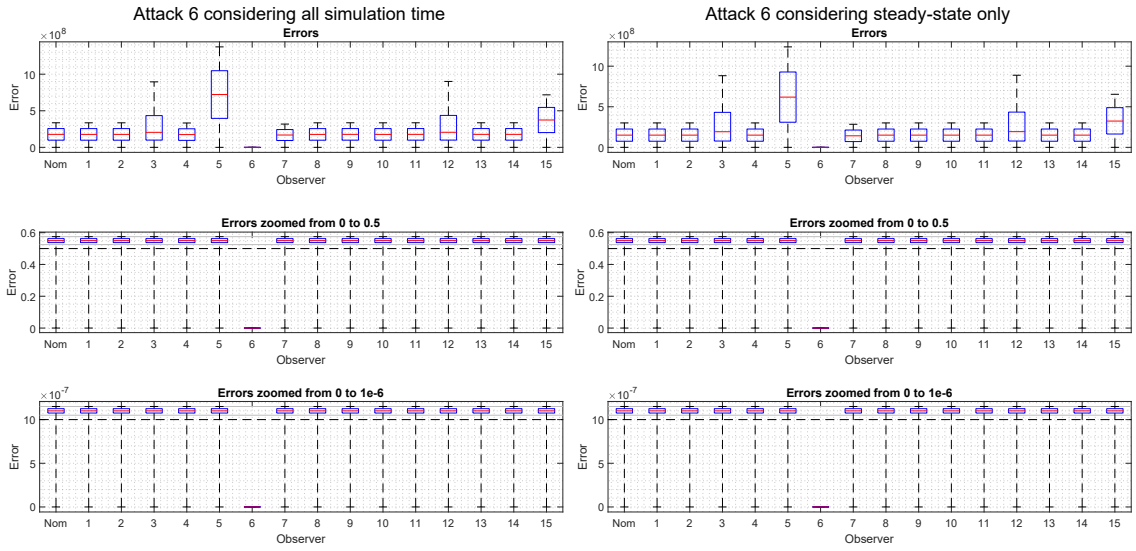
(e1) **(e2)**

(e) Attack 4



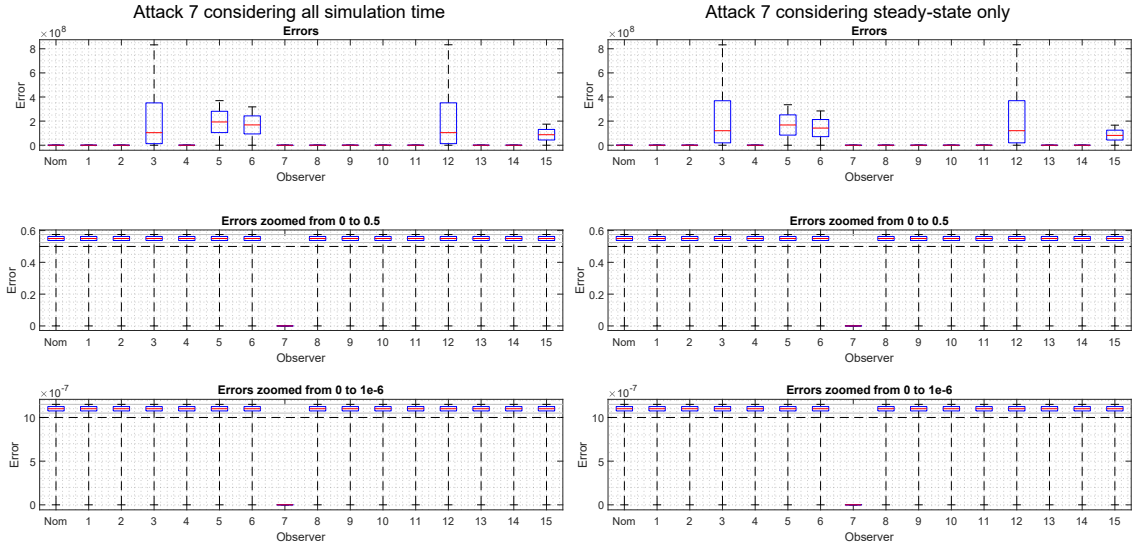
(f1) **(f2)**

(f) Attack 5



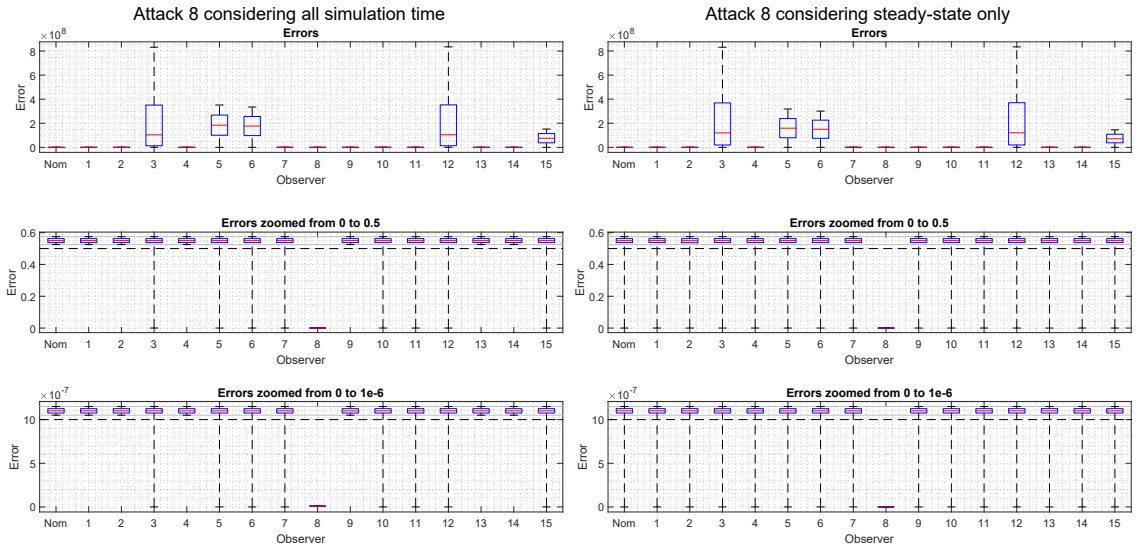
(g1) **(g2)**

(g) Attack 6



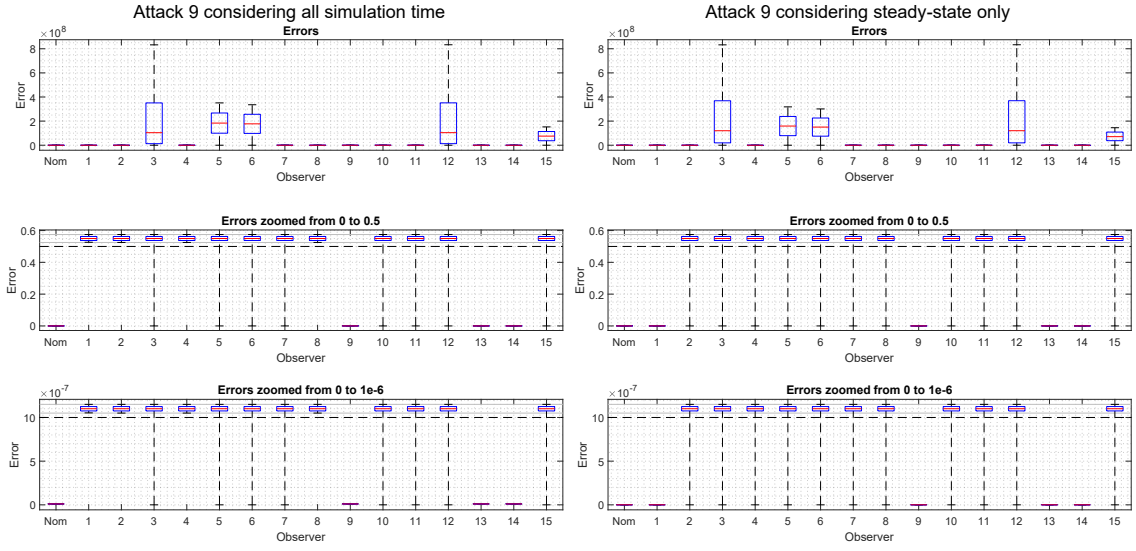
(h1) **(h2)**

(h) Attack 7



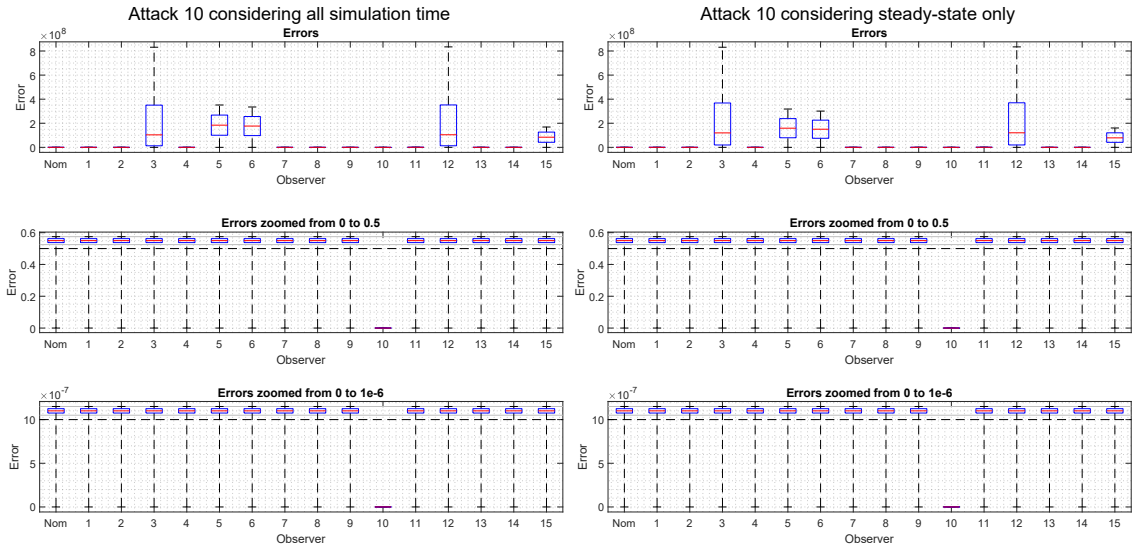
(i1) **(i2)**

(i) Attack 8



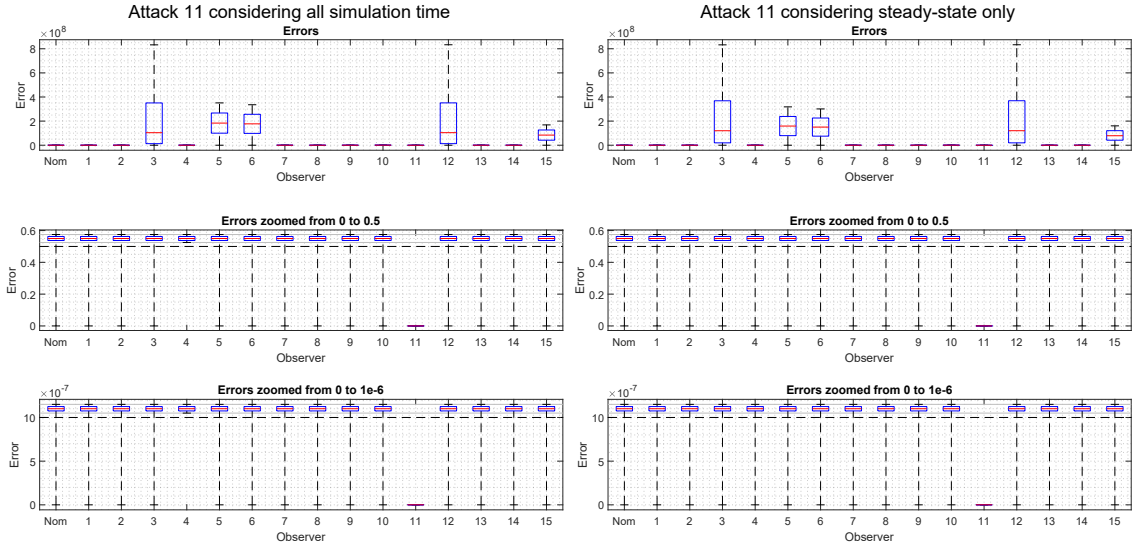
(j1) **(j2)**

(j) Attack 9



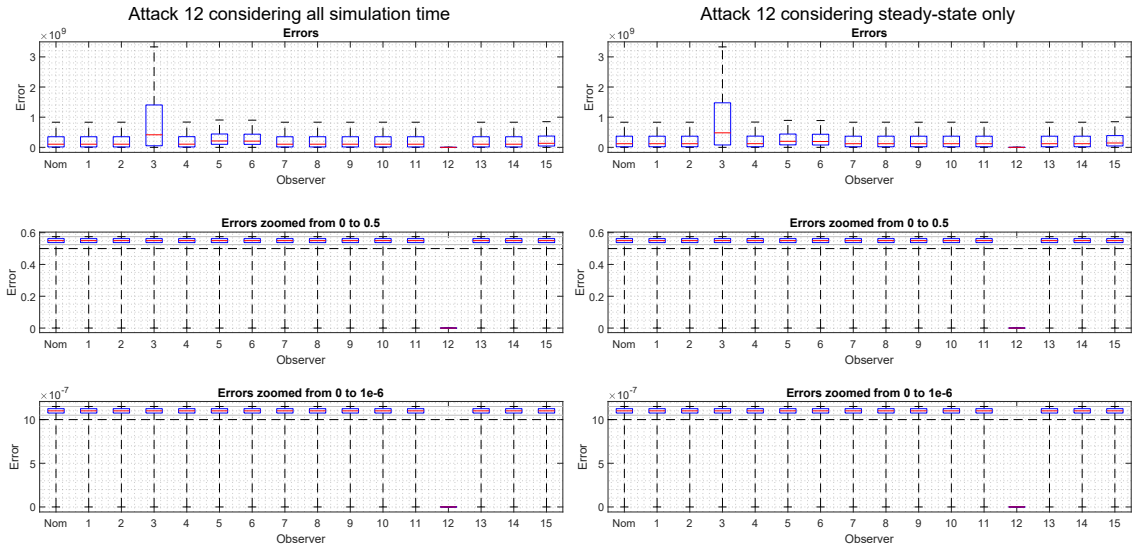
(k1) **(k2)**

(k) Attack 10



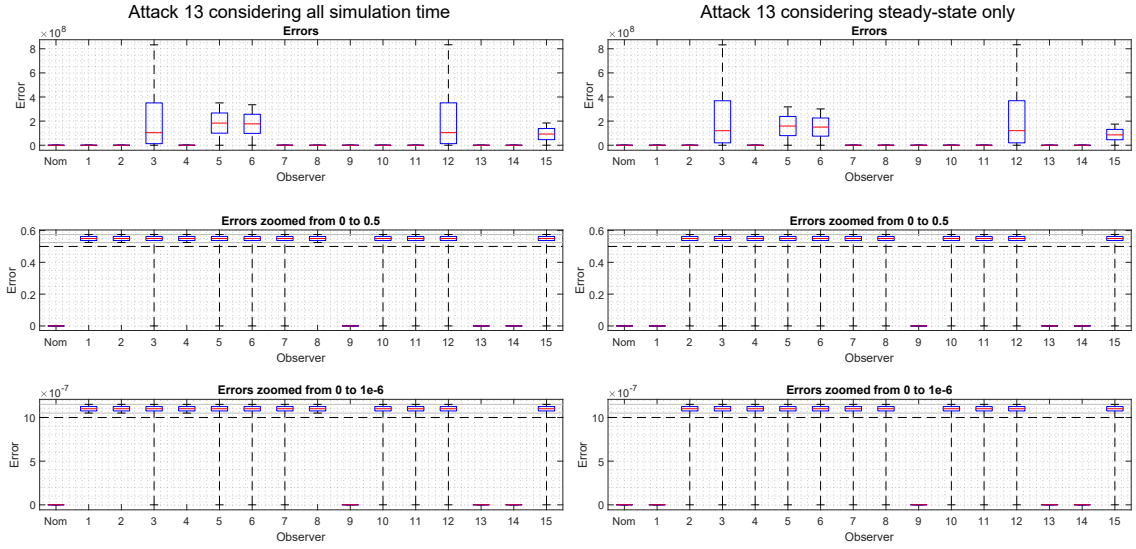
(l1) **(l2)**

(l) Attack 11



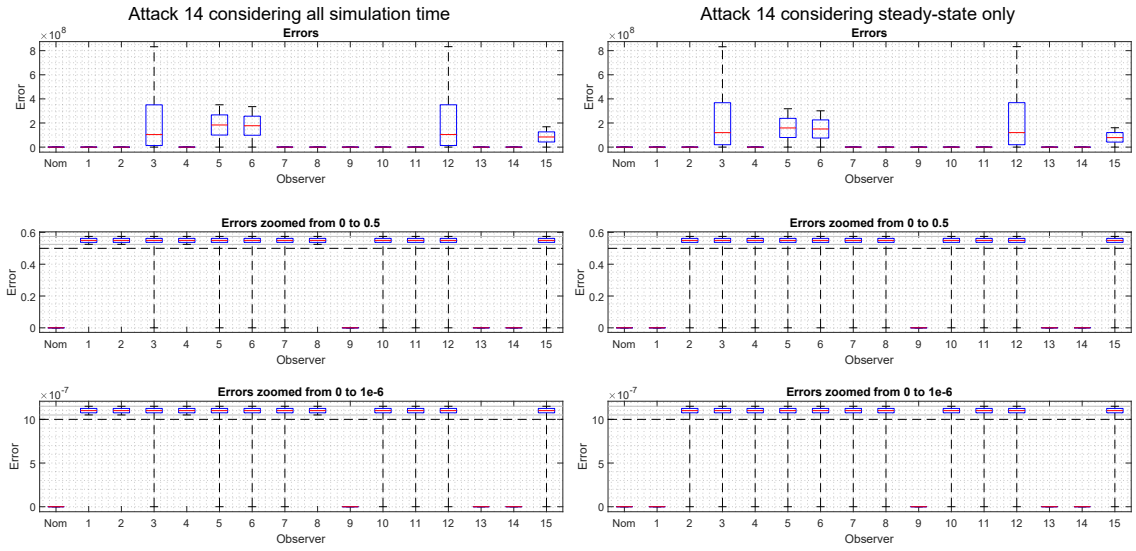
(m1) **(m2)**

(m) Attack 12



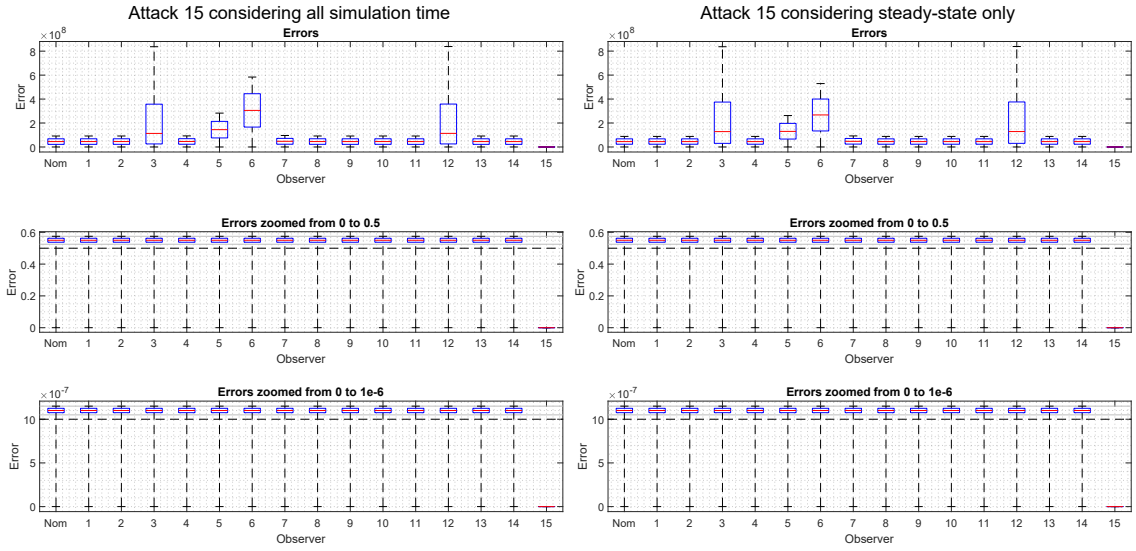
(n1) **(n2)**

(n) Attack 13



(o1) **(o2)**

(o) Attack 14



(p1) **(p2)**

(p) Attack 15

Figure 6.9: Boxplots analysis of the residuals for each observers with different attacks

6.3 Detection phase

As seen in *Part II*, while the identification phase is done with the decision tree, the detection phase is performed in a different way, using the classic approach of the overcoming of an *a-priori* defined threshold. However, figure 6.9 showed us that some of the attacks produce interchangeable results with respect to the nominal case, and so a direct check of the residual of the *"nominal" UIO* would have resulted in a difficulty of the detection of attacks 1,9,13 and 14. To overcome this, besides the *"nominal" UIO* a *copy* of the system has been designed. For *system copy* is intended a state-space system with the same state matrices already computed at the beginning, i.e. $A_{tot}, B_{tot}, C_{tot}, D_{tot}$.

The system copy receives as input only the reference ω_r, while the "nominal" UIO receives $\omega_r,\ i_a,\ \omega,\ u_a$. This way, the first one is able to detect the attacks that the observer can't detect, and viceversa.
Finally, the detection algorithm takes place as:

$$\begin{cases} \|[E_{res,1}(t),\ E_{res,copy}(t)]\| > \text{treshold, if system is under attack} \\ \|[E_{res,1}(t),\ E_{res,copy}(t)]\| < treshold, \text{if system is not under attack} \end{cases} \tag{6.1}$$

where:

- $E_{res,1}$ is the vector containing the residuals related to the *"nominal" UIO*;
- $E_{res,copy}$ is the vector that contains the errors related to the *system copy*; it is so defined:

$$E_{res,copy}(t) = \begin{bmatrix} \int_0^t (i_a(t) - i_{a,copy}(t))^2\, dt \\ \int_0^t (\omega(t) - \omega_{copy}(t))^2\, dt \\ \int_0^t (u_a(t) - u_{a,copy}(t))^2\, dt \end{bmatrix}^\intercal$$

Since in our simulation the uncertainties in the model and the noise is not modeled, the system is totally deterministic so the definition of the threshold is pretty straightforward, since it just has to be greater than the value reached by the end of the simulation time by the nominal case. However, in a real situation, it has to been chosen carefully, in order to avoid both false positives and false negatives. In our case, by the way, it has been set to 1000, to better show it in the graph.

Figure 6.10 shows an example of the detection algorithm, where the attacks simulated have amplitude 50, while in the models is 30. Simulation time is set to 15 s, with a reset at 5 s in order to cancel out all the errors due to the transient phase. It shows all the $\|[E_{res,1}(t),\ E_{res,copy}(t)]\|$ for all the 15 attacks plus the nominal case. From the graph is possible to see that all the attacks trigger the detection algorithm, except for the *attack 1*, but this is not a big problem in a real situation, since it does not affect the output of the plant, except for a transient phase at the beginning due to the different initial condition.

Others simulation not showed here, demonstrated that this approach gives similar results with every other possible attack. The cons of this method is due to the fact that we use a system copy. In fact, while the observers are able to reconstruct the internal state of the system, the system copy can not, taking as input only ω_r, so that in any case the detection system would be interrupted, such, for examples, in a loss in current, it would have an internal state different from the real one, thus making it unreliable in the detection.

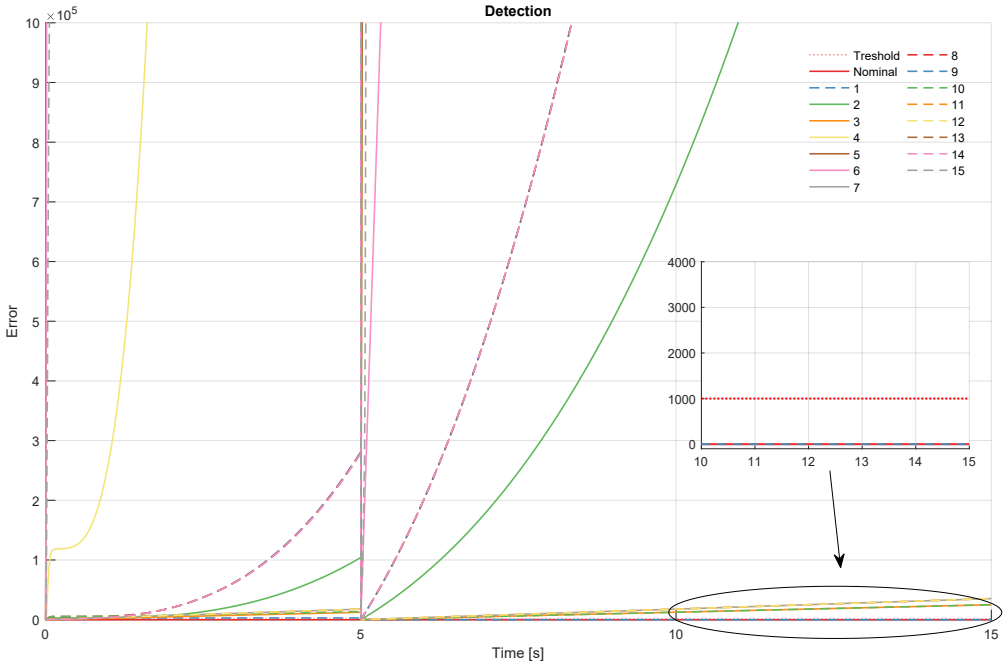


Figure 6.10: Resulting graph of $\|[E_{res,1}(t),\ E_{res,copy}(t)]\|$ for all the attacks

6.4 Identification phase

Now we move to the most important concept of this work, the identification algorithm. As seen in the previous sections, a method based only on the difference between the residuals wasn't robust enough to make it reliable in a real case. It was necessary to switch to a new approach, and the choice fell on a decision tree method, due to its semplicity with respect to a more complex neural network.

6.4.1 Training sets

The first operation to do is to train the decision tree and to do so, a suitable dataset has to be built. Since our goal is to make the identification system robust, we need

a dataset large enough to include all the possible situations that the real system can encounter. In a real application this would mean to do a huge number of simulation, but for obvious computational and time reasons, we considered only a restricted dataset, in order to understand the efficiency of this method.

Thus, the dataset was built in order to check the efficinecy of the system with these conditions:

- Three different references;
- Two different kind of attacks, continuous and intermittent;
- Different amplitude for the attacks;

Then, the simulation data for the datasets are:

- **Attack model**
  - *Amplitude*: 30;
  - *Kind of attack*: continuous;
- **3 References**
  - *Amplitudes*: 75 - 50 - 25;
  - *Frequency*: $\frac{1}{5}$ Hz;
  - τ *of LPF*: $\frac{1}{6}$;
- **Simulation time**
  - *Total time*: 25 s;
  - *Reset*: at 5 s;
  - *Observation time*: 5 – 25 s;
- **Attacks simulated**
  - *Continuous*
    - *Amplitude*: 50, 40, 30, 20, 10;
  - *Intermittent*
    - *Amplitude*: 25, 20, 15, 10, 5;
    - *Frequency*: $\frac{1}{2}Hz$;
    - *Duty Cycle*: 50%;
    - *Offset*: 25, 20, 15, 10, 5;

To resume this, we have a total of

$$5 \text{ amplitude} * 3 \text{ references} * 2 \text{ kind of attacks} * 15 \text{ positions} = 450 \text{ simulations}$$

However, the fact that we compute the integral of the residuals, and that we have to reset the computation every tot seconds, presents a problem regarding which data use for the identification. So, to better analyze the most meaningful index, four different dataset had been built, starting from the same simulations:

- *Dataset 1*: it consists in the *mean* of the residuals along all the oservation time;
- *Dataset 2*: it consists in the *median* of the residuals along all the oservation time;
- *Dataset 3*: it consists in the last value of the residuals; in our case it means that we took the residuals value at time t = 25 s;
- *Dataset 4*: it consists in a combination of the above three dataset, taking as observations all the three indeces;

So, every dataset is a matrix $\in \mathbb{R}^{450\times15}$, except for the last one, which $\in \mathbb{R}^{450\times45}$. We have 15 column because of the 15 observers that model the attacks. It is not necessary to add also the "nominal" UIO because it is already used in the detection phase. From these datasets it is possible to train four differents decision tree to test in the next phase, in order to define the most efficient in the identification process.

6.4.2 Decision tree efficiency

The next and last step is to test the decision tree just trained. To do so a certain number of simulation with random attacks has been performed. The frameworks is the subsequent:

- *Total simulations*: 400;
- *Reference*: random between the three possible references with amplitude 75, 50, 25;
- *Kind of attack*: random between continuous and intermittent;
- *Amplitude*: random between 10 and 50;
- *Position of attack*: random between the 15 possibilities;
- *Simulation time*: 45 *s*;
- *Reset time*: 5 *s* (to eliminate the transitorial phase) and 25 *s* (to start a new observation phase different from the one used for the decision trees training);

- *Time at which attack begins*: random between 5 and 20 s;

For each simulation the residuals matrix has been saved. Then, the identification process has been completed in a *Matlab script*, computing mean, median and last value of the residuals. The results are presented in the next figures and tables.

Figure 6.11 shows the occurences of the different position of the attack simulated among the 400 simulations computed, while table 6.1 points out the mean and the standard deviation of the amplitude of the attacks simulated, to higlights the variability of it.

Figure 6.12 show the *identification rate*, computed as the ratio between the attacks identified correctly and the total number of attacks, making the comparison between the four decision trees. From the simulations it can be noticed that the best results are given by the decision tree trained with the *median* of the residuals, while also the *means* give quite satisfactory results. From the graph it is possible also to distinguish the identification rate between the two *observation periods*. As expected, the identification rate is not sufficient in the first period, where the attack begins in the middle of it, causing different results with respect to the training data. By the way, it is however not so low to make it totally useless, meaning that in some specific not ideal cases could be used as an indicator of the probability of the presence of an attack. Their uses, in any case, should be restricted to the steady-state situation, where the identification rate rises.

Figure 6.13, instead, compare the identification rate with respect to the kind of attack, continuous or intermittent. It is evident that in any case the better results are given in the case of a continuous attack. This could be either seen as a good or bad result. In fact, this means that in the case of a continuous attack the identification rate is greater than what we have seen before, so in a situation where we can now *a-priori* the kind of attack that the system can receive, we can build more powerful decision tree. On the other hand, in a more geneal setup, this causes a less efficiency with some kind of attacks, meaning that a *smart* intruder could focus on this kind of attacks to reduce their identification. However, it is important to notice that the observers are modeled only with continuous attacks. Unfortunately, due to computational and time reasons, weren't made test with others model added to the previous ones, but this could be a solution to this, with the drawback of the need of a more powerful system to handle more observers.

Figure 6.14 compares the different identification rate with respect to the position of the attack. It is possible to note that some attack are more identifiable than others. In this case too, thus, the reasonings made before can be applied.

Finally, figure 6.15 compares the results with respect to the reference applied to the system. It is possible to see that there are no significant differences, meaning that the identification rate is not a dependant on the reference, as it is with the position and the kind of attack.

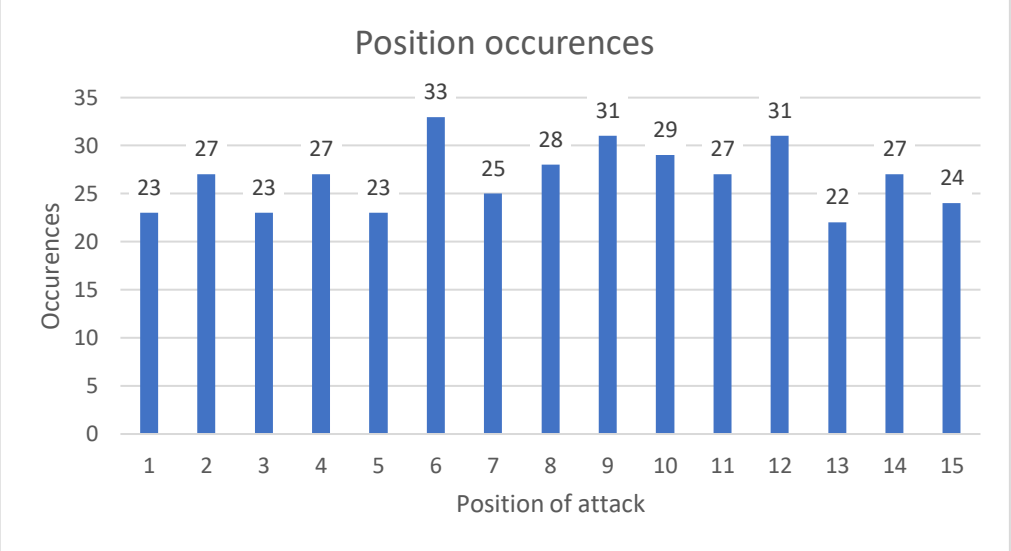


Figure 6.11: Position occurences in the 400 simulations done for the test

Attacks amplitudes	
Mean	29.6
Standard Deviation	11.4
Kind of attack	
Continuous	199
Intermittent	201
References amplitudes	
75	152
50	128
25	120

Table 6.1: Resume of the random simulations characteristics

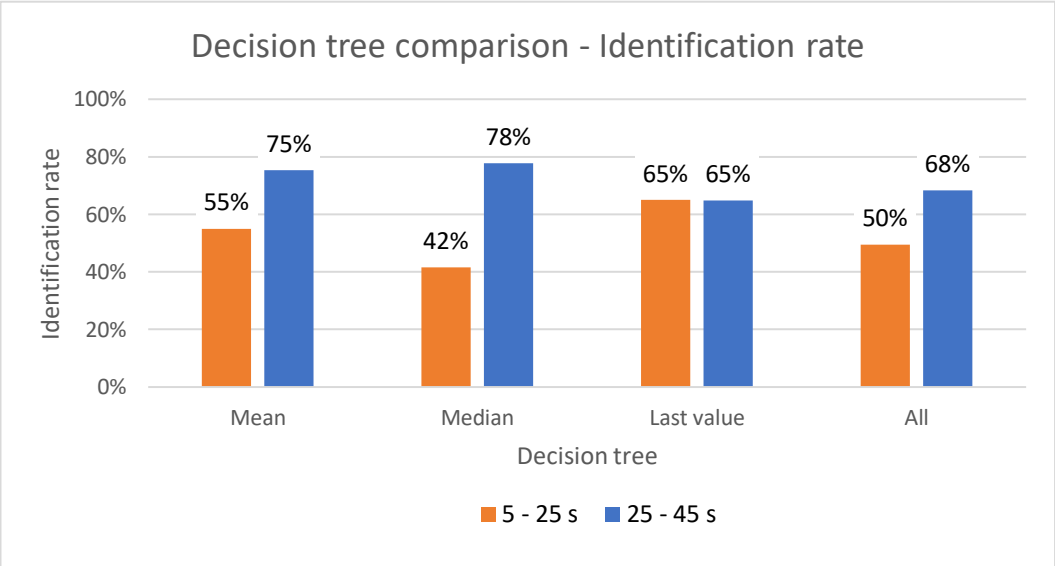


Figure 6.12: Identification rate comparison between the four different decision trees

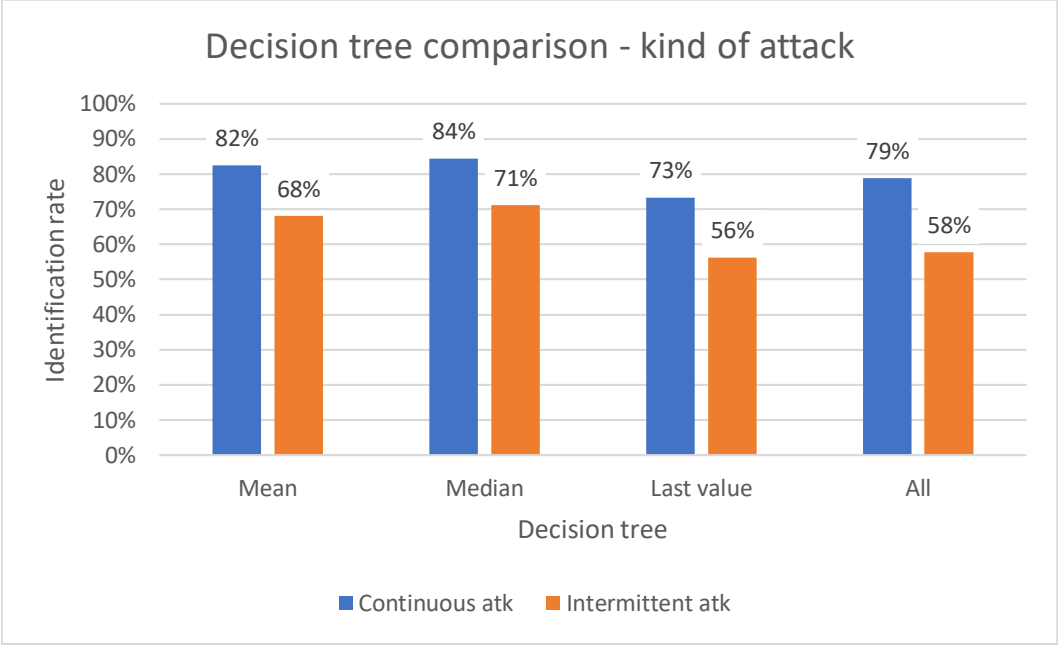


Figure 6.13: Identification rate comparison between the four different decision trees with respect to the kind of attack

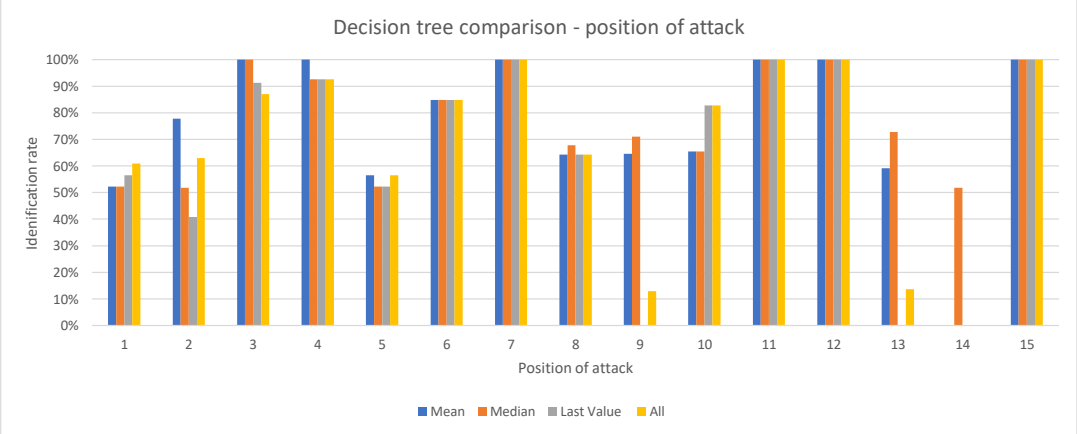


Figure 6.14: Identification rate comparison between the four different decision trees with respect to the position of the attack

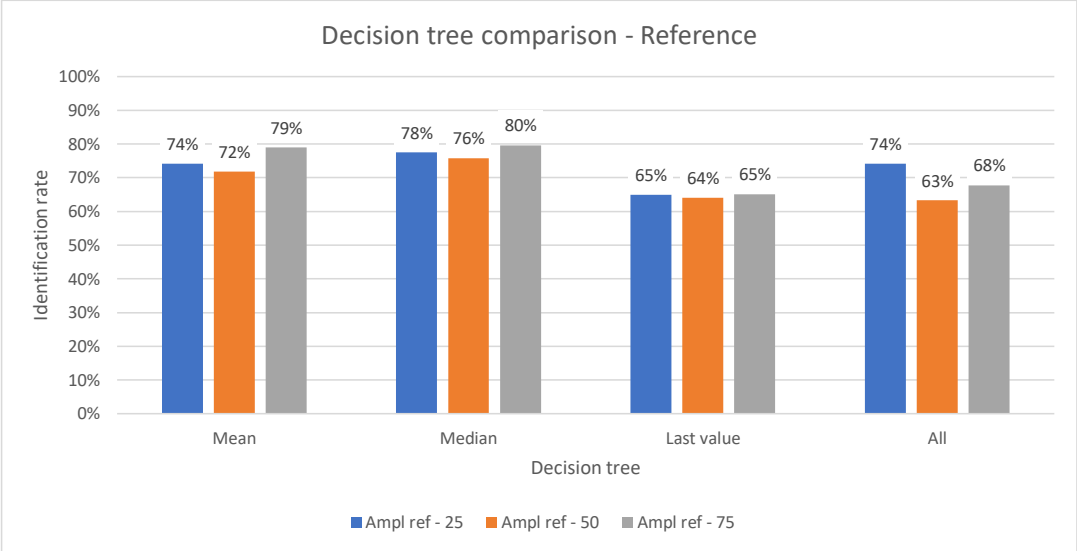


Figure 6.15: Identification rate comparison between the four different decision trees with respect to the reference

Part IV

Conclusions

Chapter 7

Conclusions

During this six months the problem of the identification of the communication attacked in a distributed control system has been addressed. The solution proposed is an evolution of an already tested algorithm by *Brain Technologies* based on a banks of observers to model the different possible attacks. However, as seen, this alone was not a solution suitable for our purpose, due to the huge number of different possible attakcs that would require more powerful computational system. In order to solve this, we slightly changed the "look-at-the-residual-only" approach, to improve it with a decision tree able to make the system more robust without the need of more observers. The final results are quite satisfactory, but still some improvement are needed so that the system can face a variety of different situations.

7.1 Possible improvements

To enhance the identification rate of this method, different possibilities can be tested.

- **Increase the number of observers**.
  With this approach it is possible to model more different kind of attacks than just the continuous one with a fixed amplitude as our case; this would provide more informations about the possible source of attack, and a better chance on the individuation not only of the altered communication, but also of the kind of attack the system is receiving. On the other hand, however, this would require more computation power in order to process the outputs of an increasing number of observers. A possible solution could be the design of a less complex estimation system [17].

- **Improve the decision tree design**.
  In our project the decision tree was simply trained with the computed dataset, without a focus on its design. Different possibilities are presented when

improving a decision tree. In fact, it could be possibile to modify the number of *leafs* and *branches* [18] [19], for example.

- **Move to a more complex neural network**.
  Another possibilities could be to use a more complex neural network instead of the decision tree. This could help to move to a more sophisticated algorithm able to recognize also some sort of patterns between the residuals in different attacks.[20]

7.2 Pro and Cons

Here we resume the most important aspects of this identification system.

- **Pro**
  - Ability to identify not only the presence of the attack but also the position on which it occurred;
  - Ample room for improvement in the identification algorithm;
  - With the use of a decision tree, the computational complexity of the system is reduced, and it is dependant on the number of communnications presents in the distributed control system;
  - The method can be applied to every distributed control system;
  - The design of the concept is quite simple and just need the building of the observers and of the decision tree;
  - The cost of the system is relatively low, but is dependant on the number of possible source of attacks;
- **Cons**
  - The identification rate is dependant on the kind of attack received;
  - There is the need of a training phase of the decision tree;
  - The detection of the attack can be difficult in a certain subset of attacks that doesn't modify the outputs;
  - The detection of the attack can be difficult in case of a loss of current, making the *system copy* internal state different from the real one;

Appendix A

Terminology

Here there is an explanation of some of the term used, as well as the explanation of some attacks that a CPS can receive.

- *CPS* - It stands for *Cyber-physical system*;
- *DCS* - It stands for *Distributed control system*;
- *UIO* - It stands for *Unknown input observer*;
- *DaIS* - It stands for *Detection and Identification system*
- *PDF* - It stands for *Probability Density Function*
- *LPF* - It stands for *Low-Pass Filter*
- *DoS attack* - it stands for *Denial of Service* and represent an attack where the intruder makes unavailable the resources of an informatic system, sending lot of requests to the server; it could be thinked as a lack of information from the sensors;
- *False Data Injection Attack* - an attack that can manipulate measurements without modifying the residual; the intruder has to know the system model;
- *Replay Attacks* - the intruder hijacks the sensors and records their measurements, so that he can replay them afterward to hide his attack; it is necessary that the system is in steady-state;

Appendix B

Case study data

In this chapter the data used in the simulations performed are presented.

B.1 DC-motor and controllers data

- **DC-motor**
  - Nominal armature voltage: $u_n = 280\ V$
  - Nominal angular speed: $\omega_n = 56\pi\ \frac{rad}{s}$
  - Nominal armature current: $i_n = 7.2\ A$
  - Winding equivalent resistance: $R = 6.41\ \Omega$
  - Winding equivalent inductance: $L = 23 \times 10^{-3}\ H$
  - Inductive voltage constant: $K_e = 1.36\ \frac{V \times s}{rad}$
  - Torque constant: $K_t = 1.36\ \frac{N \times m}{A}$
  - Motor inertia: $J_m = 0.026\ kg \times m^2$
- **Load**
  - Friction: $\beta_l = 0.051$
  - Load inertia: $J_l = 0.005\ kg \times m^2$
- **PIs**
  - $K_{p,1} = 80$
  - $K_{i,1} = 100$
  - $K_{p,2} = 70$
  - $K_{i,2} = 100$

B.2 Matrices computation

In this section the external inputs are written as q to avoid confusion between the input of the single systems, that keep the u notation.
To indicate a specific entry of a matrix we use the following notation:

$$A^{(2,1)} = a_{2,1}, \text{ where } A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}$$

B.2.1 Overall state-space system

1. **Controller**

   (a) State space representation of the two PIs

   - **PI 1**

$$\begin{cases} \dot{x}_{c,1} = B_{c,1} u_{c,1} \\ y_{c,1} = C_{c,1} x_{c,1} + D_{c,1} u_{c,1} \end{cases}$$

   - **PI 2**

$$\begin{cases} \dot{x}_{c,2} = B_{c,2} u_{c,2} \\ y_{c,2} = C_{c,2} x_{c,2} + D_{c,2} u_{c,2} \end{cases}$$

$$\begin{aligned} x_{c,1} &= \left[\int (\omega_r - \omega) \right]; & u_{c,1} &= \left[\omega_r - \omega \right]; & y_{c,1} &= \left[i_r \right] \\ x_{c,2} &= \left[\int (i_{r,sat} - i_a) \right]; & u_{c,2} &= \left[i_{r,sat} - i_a \right]; & y_{c,2} &= \left[u_a \right] \end{aligned}$$

   (b) Adding saturation.
   To add the saturation we output $y_{c,1}$ so that it can be saturated externally and then we input the new $i_{r,sat}$ to the first PI. The scheme is:

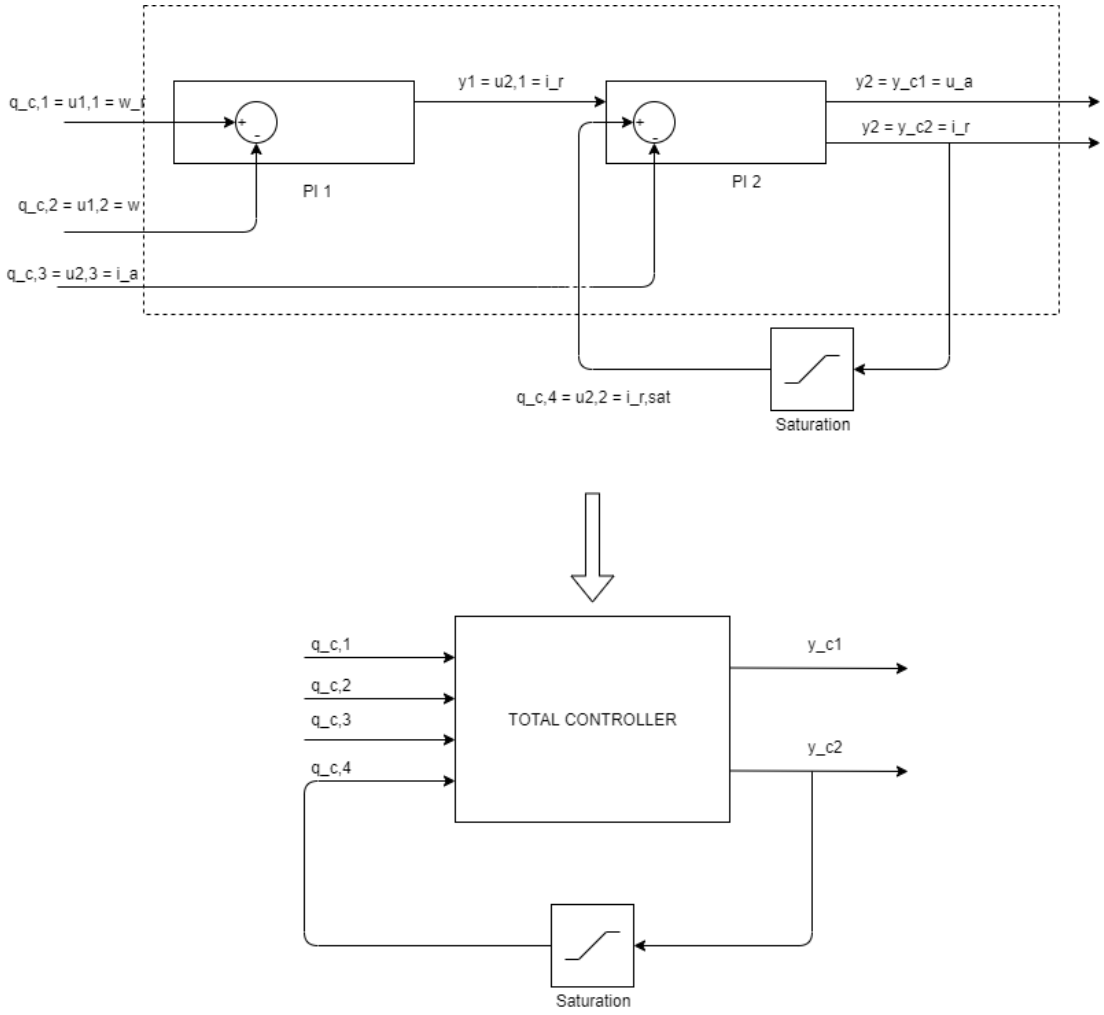


Figure B.1: PIs general scheme; q_c represents the external inputs

So that the new states, input and outputs are:

$$x_{c,1} = \left[\int (\omega_r - \omega)\right]; \qquad u_{c,1} = \begin{bmatrix}\omega_r \\ \omega\end{bmatrix}; \qquad y_{c,1} = \begin{bmatrix} i_r \end{bmatrix}$$

$$x_{c,2} = \left[\int (i_{r,sat} - i_a)\right]; \qquad u_{c,2} = \begin{bmatrix} i_r \\ i_{r,sat} \\ i_a \end{bmatrix}; \qquad y_{c,2} = \begin{bmatrix} u_a \\ i_r \end{bmatrix}$$

$$x_c = \begin{bmatrix} \int (\omega_r - \omega) \\ \int (i_{r,sat} - i_a) \end{bmatrix}; \qquad q_c = \begin{bmatrix} \omega_r \\ \omega \\ i_a \\ i_{r,sat} \end{bmatrix}; \qquad y_c = \begin{bmatrix} u_a \\ i_r \end{bmatrix}$$

(c) We write the equations for the total controller state space-representation.

- **PI 1**

$$\begin{cases} \dot{x}_{c,1} & = \omega_r - \omega = q_{c_1} - q_{c_2} \\ y_{c,1} & = i_r = K_{i,1} \int (\omega_r - \omega) + K_{p,1}\omega_r - K_{p,1}\omega = \\ & = K_{i,1}x_{c,1} + \begin{bmatrix} K_{p,1} & -K_{p,1} \end{bmatrix} \begin{bmatrix} \omega_r \\ \omega \end{bmatrix} = \\ & = K_{i,1}x_{c,1} + \begin{bmatrix} K_{p,1} & -K_{p,1} \end{bmatrix} \begin{bmatrix} q_{c_1} \\ q_{c_2} \end{bmatrix} \end{cases}$$

- **PI 2**

$$\begin{cases} \dot{x}_{c,2} & = i_{r,sat} - i_a = q_{c_4} - q_{c,3} \\ y_{c,2} & = \begin{bmatrix} u_a \\ i_r \end{bmatrix} = \begin{bmatrix} K_{i,2} \int (i_{r,sat} - i_a) - K_{p,2}i_a + K_{p,2}i_{r,sat} \\ y_{c,1} \end{bmatrix} \\ & = \begin{bmatrix} K_{i,2}x_{c,2} - \begin{bmatrix} -K_{p,2} & K_{p,2} \end{bmatrix} \begin{bmatrix} i_a \\ i_{r,sat} \end{bmatrix} \\ y_{c,1} \end{bmatrix} \\ & = \begin{bmatrix} K_{i,2}x_{c,2} - \begin{bmatrix} -K_{p,2} & K_{p,2} \end{bmatrix} \begin{bmatrix} q_{c_3} \\ q_{c_4} \end{bmatrix} \\ y_{c,1} \end{bmatrix} \end{cases}$$

Thus we can write:

$$B_{c,1} = \begin{bmatrix} 1 & -1 \end{bmatrix}; \qquad C_{c,1} = \begin{bmatrix} K_{i,1} \end{bmatrix}; \qquad D_{c,1} = \begin{bmatrix} K_{p,1} & -K_{p,1} \end{bmatrix}$$
$$B_{c,2} = \begin{bmatrix} 1 & -1 \end{bmatrix}; \qquad C_{c,2} = \begin{bmatrix} K_{i,2} \end{bmatrix}; \qquad D_{c,2} = \begin{bmatrix} K_{p,2} & -K_{p,2} \end{bmatrix}$$

(d) Considering that $y_c = y_{c,2}$ and $x_c = \begin{bmatrix} x_{c,1} \\ x_{c,2} \end{bmatrix}$, the total controller state-space representation is:

$$\begin{cases} \dot{x}_c = B_c q_c \\ y_c = C_c x_c + D_c q_c \end{cases} \tag{B.1}$$

where:

- $B_c = \begin{bmatrix} B_{c,1}^{(1)} & B_{c,1}^{(2)} & 0 & 0 \\ 0 & 0 & B_{c,2}^{(2)} & B_{c,2}^{(1)} \end{bmatrix}$
- $C_c = \begin{bmatrix} 0 & C_{c,2} \\ C_{c,1} & 0 \end{bmatrix}$

- $D_c = \begin{bmatrix} 0 & 0 & D_{c,2}^{(2)} & D_{c,2}^{(1)} \\ D_{c,1}^{(1)} & D_{c,2}^{(2)} & 0 & 0 \end{bmatrix}$

2. **Total system**

(a) Plant state space representation:

$$\begin{cases} \dot{x}_p = A_p x_p + B_p u_p + E_p d_p \\ y_p = C_p x_p \end{cases}$$

$$x_p = \begin{bmatrix} i_a \\ \omega \end{bmatrix}, u_p = \begin{bmatrix} u_a \end{bmatrix}, d_p = \begin{bmatrix} T_r \end{bmatrix}, y_p = \begin{bmatrix} i_a \\ \omega \end{bmatrix}$$

(b) Scheme

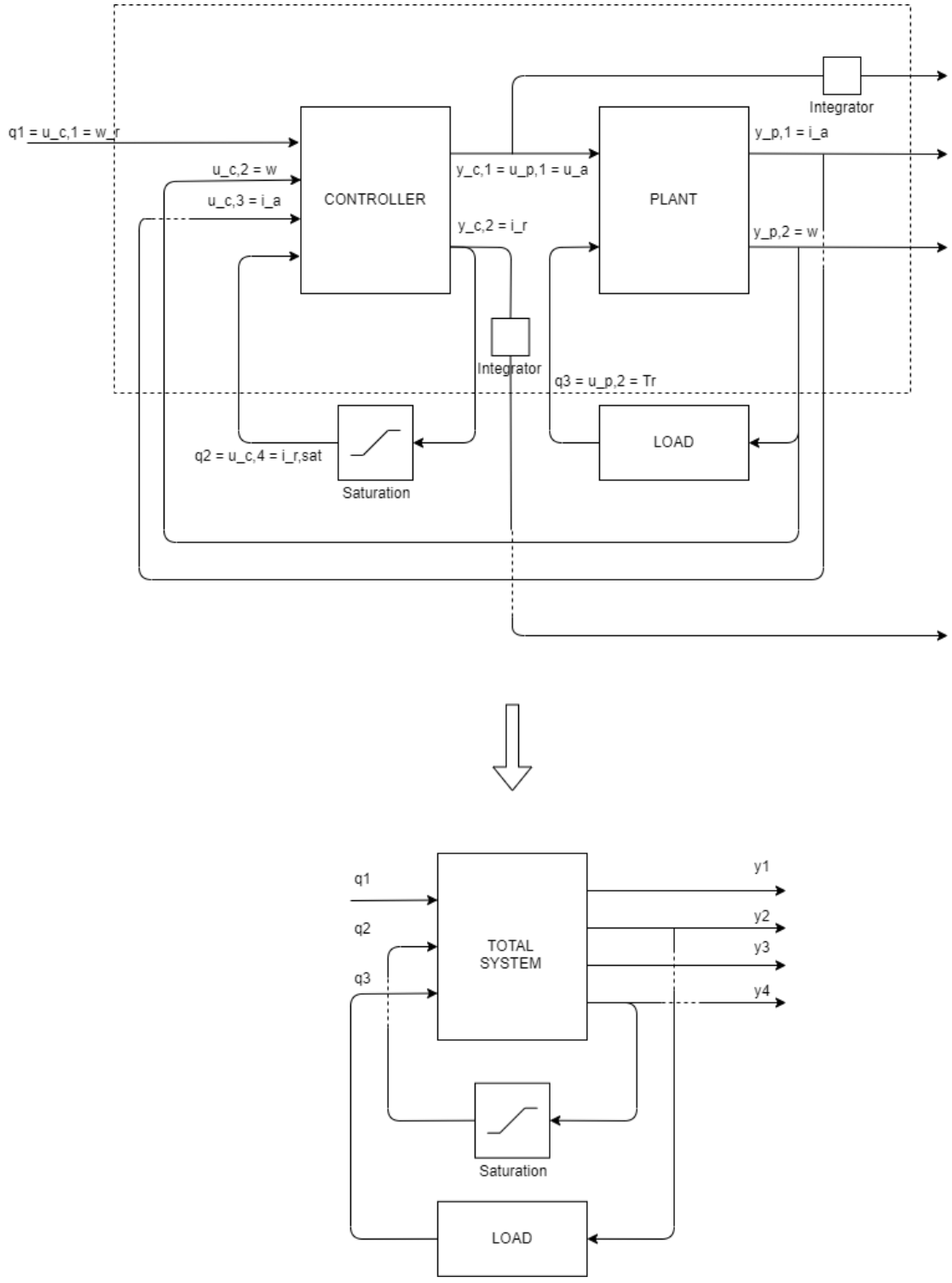


Figure B.2: Total system general scheme; q represents the external inputs

So that the new states, input and outputs are:

$$x = \begin{bmatrix} \int (\omega_r - \omega) \\ \int (i_{r,sat} - i_a) \\ i_a \\ \omega \\ \int u_a \\ \int i_r \end{bmatrix}; \quad q = \begin{bmatrix} \omega_r \\ i_{r,sat} \\ T_r \end{bmatrix}; \quad y = \begin{bmatrix} i_a \\ \omega \\ \int u_a \\ \int i_r \end{bmatrix}$$

(c) Equations.
From

$$\begin{cases} u_c = \begin{bmatrix} \omega_r \\ \omega \\ i_a \\ i_{r,sat} \end{bmatrix} = \begin{bmatrix} q^{(1)} \\ y_p^{(2)} \\ y_p^{(1)} \\ q^{(2)} \end{bmatrix} = \begin{bmatrix} q^{(1)} \\ x_p^{(2)} \\ x_p^{(1)} \\ q^{(2)} \end{bmatrix} \\ y_c = \begin{bmatrix} u_a \\ i_r \end{bmatrix} = \begin{bmatrix} u_a \\ y_c^{(2)} \end{bmatrix} = \begin{bmatrix} u_a \\ C_c^{(2,1)} x_c^{(1)} + D_c^{(2,1)} q^{(1)} + D_c^{(2,2)} x_p^{(2)} \end{bmatrix} \\ u_p = \begin{bmatrix} u_a \\ T_r \end{bmatrix} = \begin{bmatrix} y_c^{(1)} \\ q^{(3)} \end{bmatrix} = \begin{bmatrix} C_c^{(1,2)} x_c^{(2)} + D_c^{(1,3)} x_p^{(1)} + D_c^{(1,4)} q^{(2)} \\ q^{(3)} \end{bmatrix} \end{cases}$$

we can write

$$\begin{cases} \dot{x}^{(1)} & = \dot{x}_c^{(2)} = \omega_r - \omega = B_c^{(1,1)} q^{(1)} + B_c^{(1,2)} x_p^{(2)} \\ \dot{x}^{(2)} & = \dot{x}_c^{(2)} = i_{r,sat} - i_a = B_c^{(2,3)} x_p^{(1)} + B_c^{(2,4)} q^{(2)} \\ \dot{x}^{(3)} & = \dot{x}_p^{(1)} = \dot{i}_a = A_p^{(1,1)} x_p^{(1)} + A_p^{(1,2)} x_p^{(2)} + B_p^{(1)} (C_c^{(1,2)} x_c^{(2)} + D_c^{(1,3)} x_p^{(1)} + D_c^{(1,4)} q^{(2)}) \\ \dot{x}^{(4)} & = \dot{x}_p^{(2)} = \dot{\omega} = A_p^{(2,1)} x_p^{1)} + A_p^{(2,2)} x_p^{(2)} + E_p^{(2)} q^{(3)} \\ \dot{x}^{(5)} & = y_c^{(1)} = u_a = C_c^{(1,2)} x_c^{(2)} + D_c^{(1,3)} x_p^{(1)} + D_c^{(1,4)} q^{(2)} \\ \dot{x}^{(6)} & = y_c^{(2)} = i_r = C_c(2,1) x_c^{(1)} + D_c^{(2,1)} q^{(1)} + D_c^{(2,2)} x_p^{(2)} \end{cases}$$

(d) The total state-space representation of the system composed by the controller and the plant is:

$$\begin{cases} \dot{x} = Ax + Bq + Ed \\ y = Cx \end{cases} \tag{B.2}$$

where:

- $A = \begin{bmatrix} 0 & 0 & 0 & B_c^{(1,2)} & 0 & 0 \\ 0 & 0 & B_c^{(2,3)} & 0 & 0 & 0 \\ 0 & B_p^{(1)} C_c^{(1,2)} & A_p^{(1,1)} + B_p^{(1)} D_c^{(1,3)} & A_p^{(1,2)} & 0 & 0 \\ 0 & 0 & A_p^{(2,1)} & A_p^{(2,2)} & 0 & 0 \\ 0 & C_c^{(1,2)} & D_c^{(1,3)} & 0 & 0 & 0 \\ C_c^{(2,1)} & 0 & 0 & D_c^{(2,2)} & 0 & 0 \end{bmatrix}$
- $B = \begin{bmatrix} B_c^{(1,1)} & 0 \\ 0 & B_c^{(2,4)} \\ 0 & B_p^{(1)} D_c^{(1,4)} \\ 0 & 0 \\ 0 & D_c^{(1,4)} \\ D_c^{(2,1)} & 0 \end{bmatrix}$
- $E = \begin{bmatrix} 0 \\ 0 \\ 0 \\ E_p^{(2)} \\ 0 \\ 0 \end{bmatrix}$
- $C = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

Appendix C

Simulink schemes

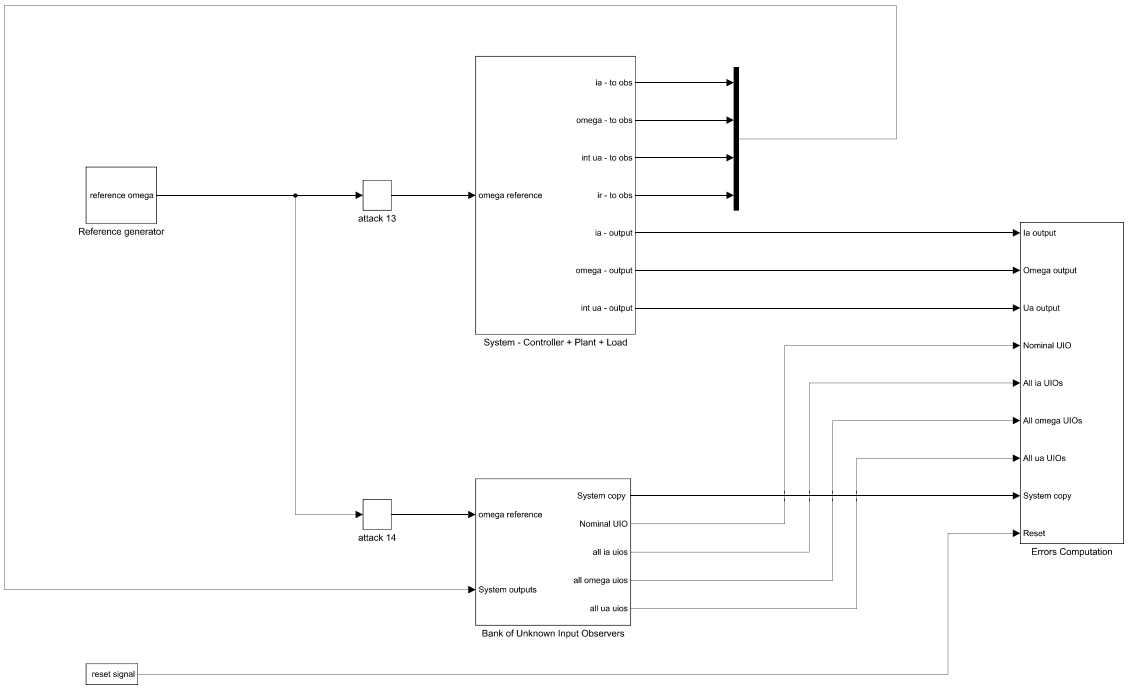


Figure C.1: Overall system

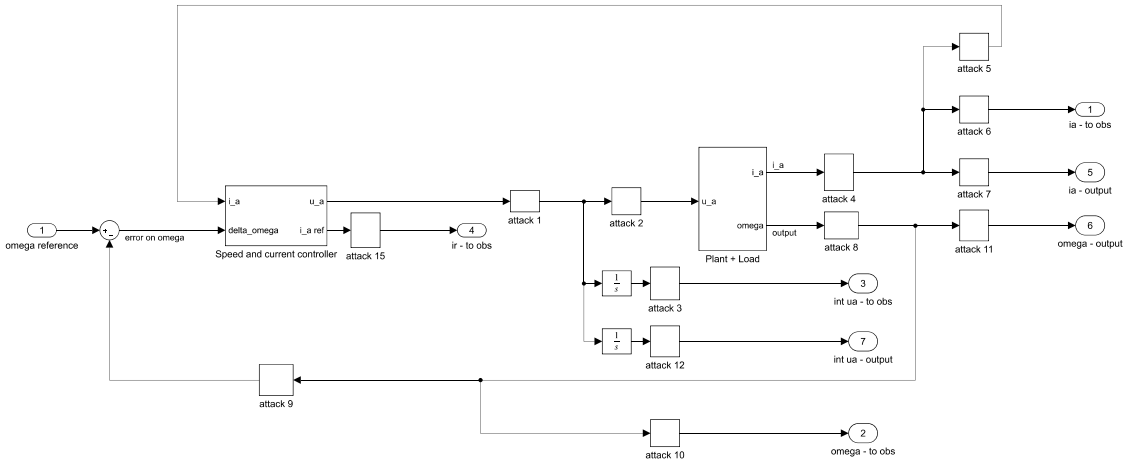


Figure C.2: System under attack

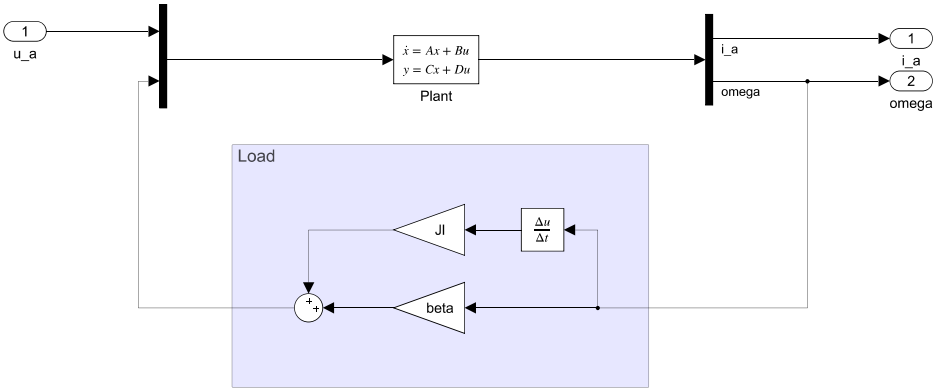


Figure C.3: DC-motor and load

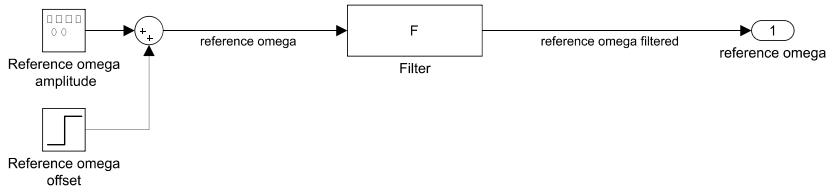


Figure C.4: Reference

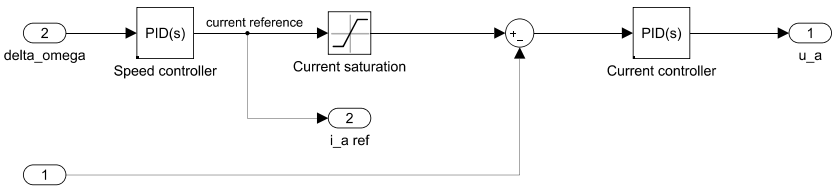


Figure C.5: Controller

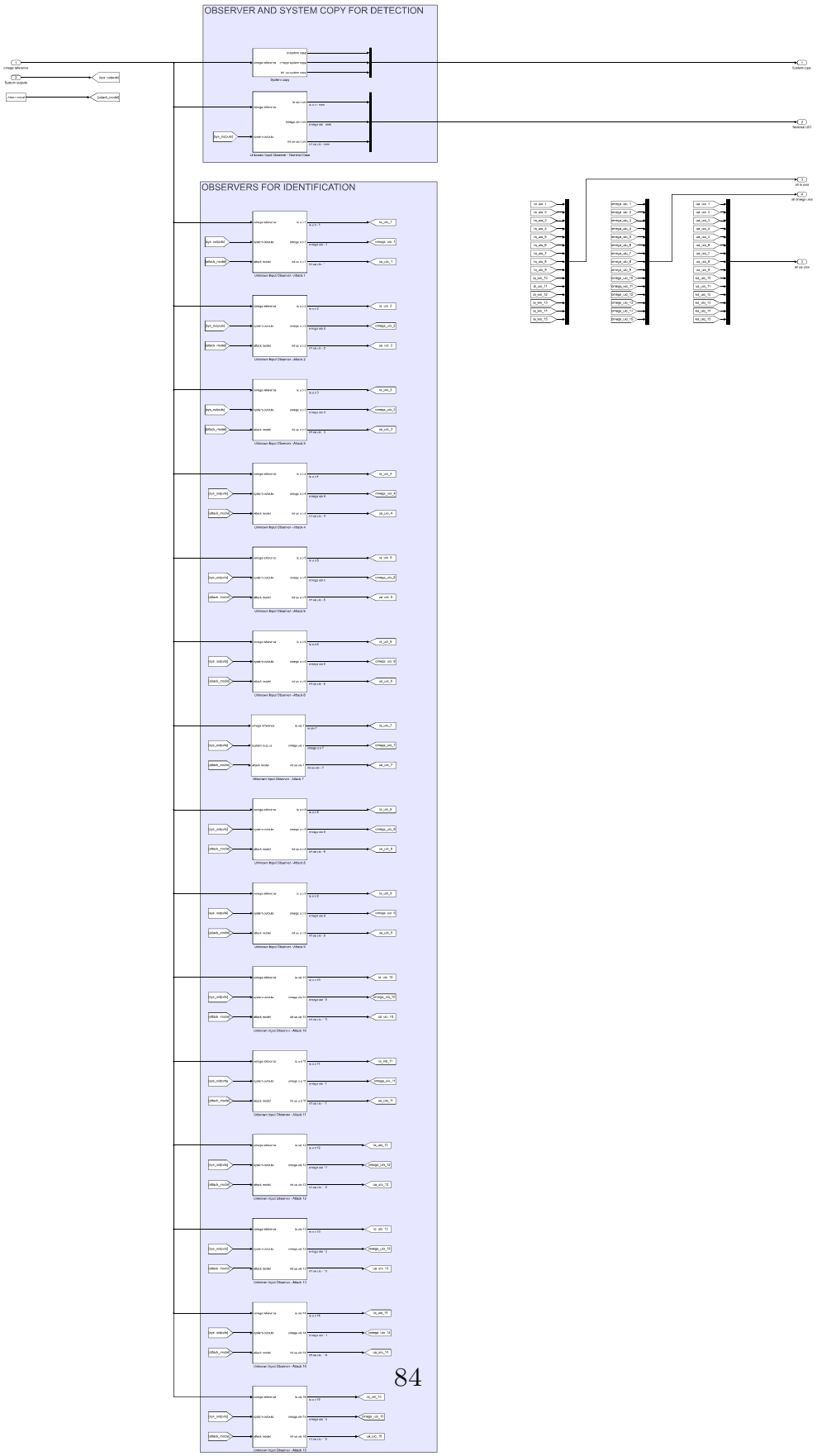


Figure C.6: Bank of UIOs

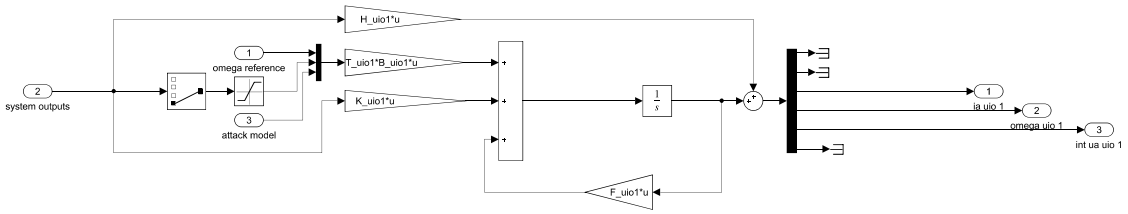


Figure C.7: Example of an UIO - UIO designed for attack 1

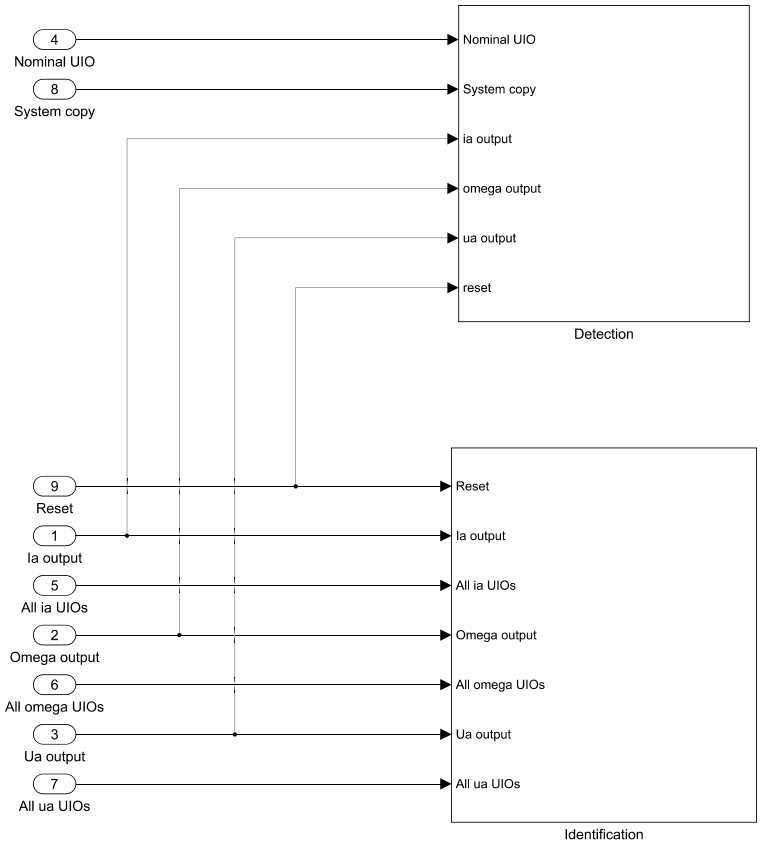


Figure C.8: Detection and Identification embedded system

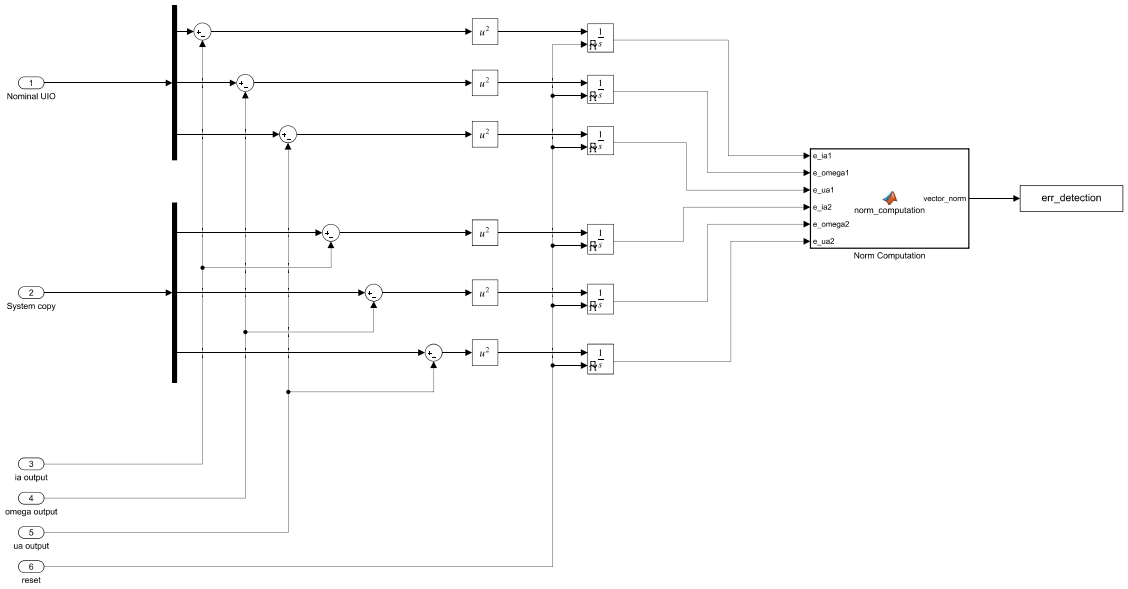


Figure C.9: Index computation for detection

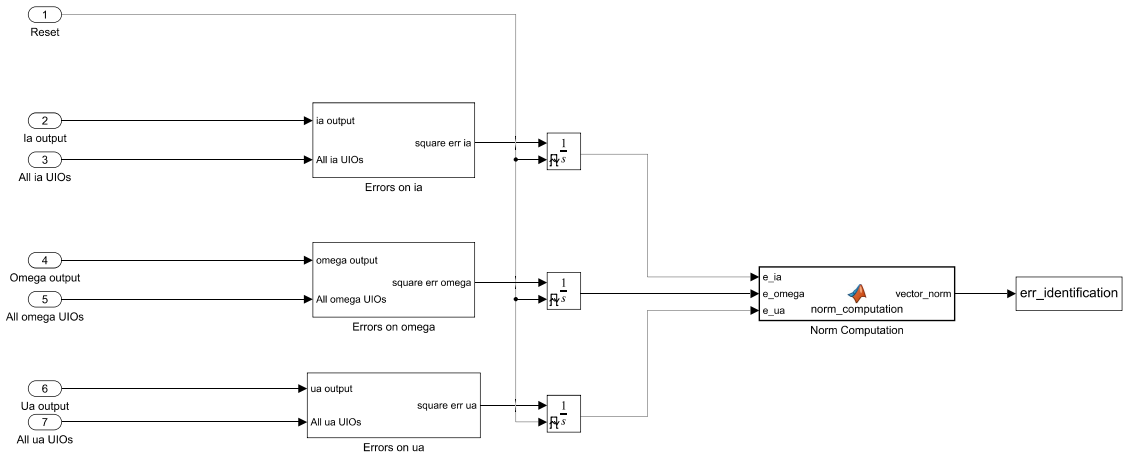


Figure C.10: Indeces computation for identification

Bibliography

[1] R. Langner. "Stuxnet: Dissecting a Cyberwarfare Weapon". In: *IEEE Security Privacy* 9.3 (2011), pp. 49–51. DOI: 10.1109/MSP.2011.67.

[2] K. Kim and P. R. Kumar. "Cyber–Physical Systems: A Perspective at the Centennial". In: *Proceedings of the IEEE* 100.Special Centennial Issue (2012), pp. 1287–1308. DOI: 10.1109/JPROC.2012.2189792.

[3] George Loukas. "A History of Cyber-Physical Security Incidents". In: Dec. 2015, pp. 21–57. ISBN: 9780128012901. DOI: 10.1016/B978-0-12-801290-1.00002-3.

[4] E. Bou-Harb. "A Brief Survey of Security Approaches for Cyber-Physical Systems". In: *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. 2016, pp. 1–5. DOI: 10.1109/NTMS.2016.7792424.

[5] J. Giraldo et al. "Security and Privacy in Cyber-Physical Systems: A Survey of Surveys". In: *IEEE Design Test* 34.4 (2017), pp. 7–17. DOI: 10.1109/MDAT.2017.2709310.

[6] Dubravko Miljković. "Fault detection methods: A literature survey." In: May 2011, pp. 750–755.

[7] S. Tan et al. "Brief Survey on Attack Detection Methods for Cyber-Physical Systems". In: *IEEE Systems Journal* 14.4 (2020), pp. 5329–5339. DOI: 10.1109/JSYST.2020.2991258.

[8] A. Humayed et al. "Cyber-Physical Systems Security—A Survey". In: *IEEE Internet of Things Journal* 4.6 (2017), pp. 1802–1831. DOI: 10.1109/JIOT.2017.2703172.

[9] Eugene L. Duke and Dryden Flight Research Facility. *Combining and connecting linear, multi-input, multi-output subsystem models [microform] / Eugene L. Duke*. English. National Aeronautics and Space Administration, Ames Research Center, Dryden Flight Research Center Edwards, Calif, 1986, p. 1 v.

[10] Jie Chen, Ron Patton, and HONG-YUE ZHANG. "Design of unknown input observers and robust fault detection filters". In: *International Journal of Control* 63 (Feb. 2007), pp. 85–105. DOI: 10.1080/00207179608921833.

[11] Inc. The MathWorks. *Decision Trees*. URL: https://it.mathworks.com/help/stats/decision-trees.html.

[12] Krzysztof Grąbczewski. *Meta-Learning in Decision Tree Induction*. 1st ed. Studies in Computational Intelligence 498. Springer International Publishing, 2014.

[13] *Classifying data with decision trees*. URL: https://elf11.github.io/2018/07/01/python-decision-trees-acm.html.

[14] Andrew W. Moore. *Decision Trees - Sistemi informativi per le Decisioni*. URL: http://www-db.deis.unibo.it/courses/SID/old/Lezioni/04%20-%20Decision%20trees.pdf.

[15] Zs Horváth and G. Molnárka. "Design Luenberger Observer for an Electromechanical Actuator". In: *Acta Technica Jaurinensis* 7 (Oct. 2014). DOI: 10.14513/actatechjaur.v7.n4.313.

[16] A. Bhattacharyya. "On a measure of divergence between two statistical populations defined by their probability distributions". In: *Bull. Calcutta Math. Soc.* 35 (1943), pp. 99–109. ISSN: 0008-0659.

[17] Y. Shoukry et al. "SMT-Based Observer Design for Cyber-Physical Systems under Sensor Attacks". In: *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*. 2016, pp. 1–10. DOI: 10.1109/ICCPS.2016.7479119.

[18] Xiaodan Wang et al. "An Improved Algorithm for Decision-Tree-Based SVM". In: *2006 6th World Congress on Intelligent Control and Automation*. Vol. 1. 2006, pp. 4234–4238. DOI: 10.1109/WCICA.2006.1713173.

[19] Wei-Yin Loh and Yu-Shan Shih. "Split Selection Methods for Classification Trees". In: *Statistica Sinica* Vol. 7 (1997), pp. 815–840.

[20] Abdullahi Uwaisu Muhammad et al. "Survey on Training Neural Networks". In: *International Journal of Software Engineering and Knowledge Engineering* Vol. 5 (Mar. 2015), pp. 169–173.