University of Naples Federico II

DEPARTMENT OF ELECTRICAL ENGINEERING
AND INFORMATION TECHNOLOGIES

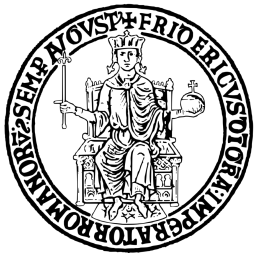
Master Degree - Automation Engineering

Water Tank Network Control System

Candidates
Alessandro Di Nola
P38000098
Lorenzo D' Oriano
P38000224
Silvia Leo
P38000225
Giulia Gelsomina Romano
P38000223

Supervisor
Prof. Sabato Manfredi

Academic Year: 2023/2024



Contents

List of Figures

Chapter 1

Introduction

This thesis presents a comprehensive project involving the management of three water tanks, treated first with a centralized control system and subsequently with a decentralized control system. The primary aim of the project is to explore and compare, based on our findings, the practical efficiency and functionality of these two different control approaches in managing water distribution and levels in a multi-tank system.
Starting the project, the initial steps involved the assembly of various components, as explained in Chapter 2, followed by a detailed overview of all the components used in the project. The roles of each component, for this specific case, are discussed to give a clear understanding of the project's technical foundation.
In Chapter 3, the focus is on the specific set-ups employed in the experiments, in particular for the communication protocols and the water sensors. The configurations and parameters for each test scenario are outlined, explaining how the components were integrated and the rationale behind the chosen setups.
Advancing to Chapter 4, attention is directed to the centralized control.
The design and implementation of the central controller, which was chosen to be the upper tank, its interaction with the other water tanks, and the performance metrics observed during testing are explained with the aid of *MATLAB*. The advantages and limitations of centralized control are analyzed in detail.
In Chapter 5, distributed control is explored. For this purpose, various approaches such as the Dynamic Consensus algorithm, the Weighted Consensus algorithm and the Robust Control were implemented reporting also the outcomes of different experiments on the real net and on *MATLAB* environment. The benefits and challenges of a decentralized approach are thoroughly examined.
The concluding chapter, Chapter 6, contains our reflections and impressions gathered throughout the developing of the project. This discussion covers not only the technical aspects but also the experiential and qualitative ones of the project.

Chapter 2

Physical model and its Components

In the following chapter, the physical model consisting of three interconnected water tanks and the used electronic and mechanical components will be described.

2.1 Physical model

The physical model is described as follow.

Figure 2.1: Physical Model.

As shown in Fig. 2.1 the water tank system is made by three square trays, each one has capacity of 3 liters. The tray in the center is raised 14 *cm* above the others thanks to a specially built support. The former is connected to each of the others by a mechanical component made by a syringe of 11 *mm* diameter.

Figure 2.2: Focus on the syringe.

This syringe, shown in Fig. 2.2, has two openings: the first one is made to connect the upper tank with the syringe itself and it's located 2 *cm* from the bottom of the tray, the second one is on the lateral surface of the barrel and it is used to spill water to the bottom tanks. Moreover it is used to simulate a manual valve thanks to the piston that can regulate the largeness of this last opening with the possibility to close it up completely.
Each tank on the bottom, instead, is linked to the raised one by a pump which has a motor that drains water on request and will be analyzed in detail later on this chapter.

2.2 Electronic Components

In this section, a brief summary will be offered relating to all the used electronic components that are disposed as illustrated in 2.3.

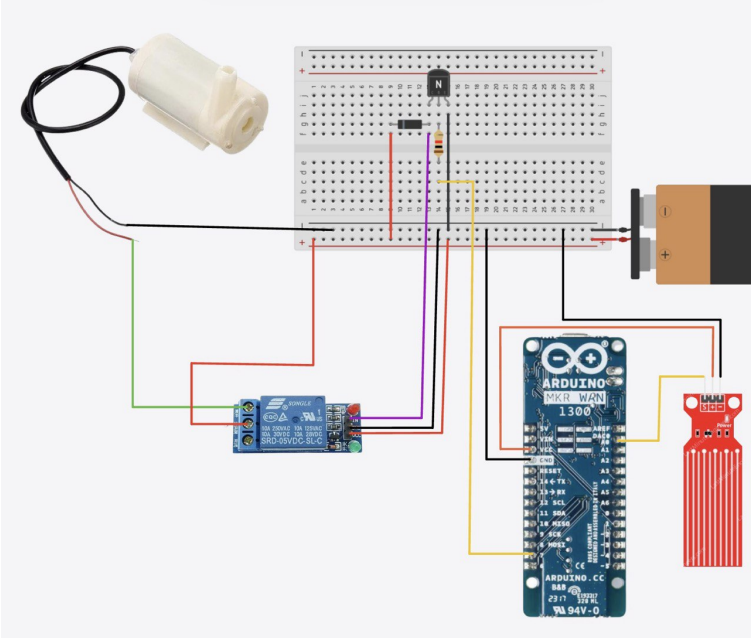


Figure 2.3: Components connection

Figure 2.4: Overall circuit

2.2.1 MKR 1000 WiFi

Over the years, Arduino has developed a variety of boards, each one with different capabilities and functionalities. The Arduino MKR1000 is a versatile board that includes a LiPo charging circuit that allows it to run on batteries, has a WiFi module with a crypto-chip to ensure

secure communications, and can be configured to constrain its power consumption to keep it running for long periods.

Figure 2.5: Board MKR1000

The MKR1000 board has 28 pins, some of which have fixed functionalities and some of which have configurable functionalities. Fixed functionality pins include GND and 5V, which provide access to the board's ground and 5-volt reference voltage lines respectively. The pins labeled A0 to A6 are analog pins. These pins can transmit or receive voltage values between 0 and 3.3 volts, relative to GND. The pins labeled 0 to 14 are digital pins. When these pins transmit data, they can only transmit voltage values of 0 or 3.3 volts, relative to GND. When they receive data, the voltage is interpreted as HIGH or LOW, relative to some threshold between 0 and 3.3 volts. The pins on the Arduino boards can be used to control and communicate with external components, such as LEDs, motors, or sensors. Pins can be configured either as INPUT or OUTPUT. In this project, pin $A0$ and pin $D7$, respectively, are used to control the water sensor and the water pump.

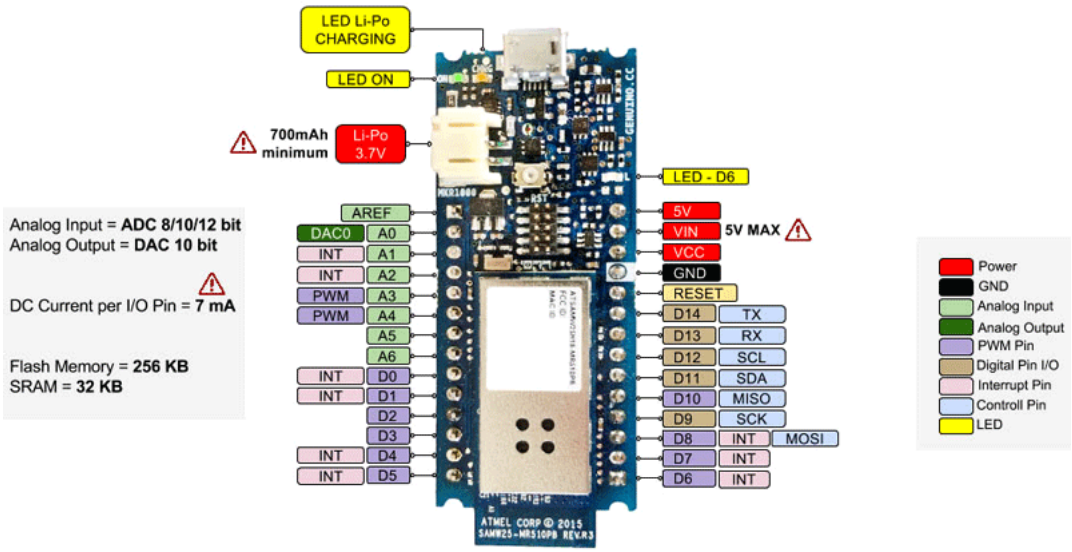


Figure 2.6: Arduino MKR1000: Pinout.

2.2.2 Water Sensor

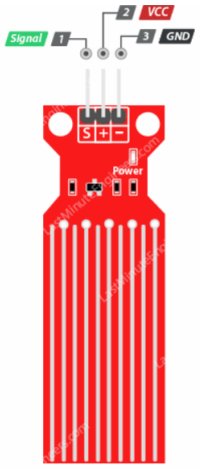


Figure 2.7: Water Sensor.

The Water Sensor is a proportional sensor used to detect the level of water in a tank. The Water Sensor is compatible with any Arduino board, making its integration into projects extremely simple and immediate. This sensor consists of a printed circuit board with conductive traces that, when in contact with water, change the electrical resistance. This voltage variation is read by the Arduino microcontroller and converted into precise digital readings, allowing the determination of the water level present.

The S pin is connected to the $A0$ Arduino's pin, the *plus* pin is linked to the VCC Arduino's pin and, lastly, the *minus* pin is connected to the ground.

2.2.3 Pump System

The pump system is composed by the pump itself, the relay shield and the amplifier and it's alimented by a $5V$ battery.

Pump

The water pump chosen is a 5-volt pump. It's a small, low-voltage device commonly used in electronics projects for tasks such as fluid delivery, cooling, or circulation. It has two pins: the red one is connected to the NC relay's pin while the black one pin is linked to the ground. The pump with the upper nozzle, to which a plastic tube has been attached, transfers water from the lower tank to the upper one.

Figure 2.8: Pump.

Relay Shield

The relay shield acts as an electrically controlled switch which allows to turn on or off another device with a low-power control signal. It consists of three primary pins.
The VCC pin and the GND pin are respectively connected to the red and black terminal of the battery. The IN pin is connected, through the amplifier circuit, to the $D7$ Arduino's pin. This setup is necessary because the Signal Input of the Relay operates at 5V, whereas the Arduino's digital pins operate at 3.3V, as previously discussed.
The relay contacts play a crucial role in electrical switching applications, it has three contacts. The COM (Common) terminal is the common connection point that links either the NC or NO contact, depending on the relay's state. The NO (Normally Open) terminal is connected to the COM terminal when the relay coil is not energized. When the relay coil receives a

control signal (energized state), the NO contact closes, allowing current to flow between the COM and NO terminals. While the *NC* (Normally Closed) is connected to the COM terminal when the relay coil is in its default state (energized state). In this state, the NC contact is closed, allowing current to flow between the COM and NC terminals. When the relay coil is energized (de-nergized state), the NC contact opens, interrupting the current flow between the COM and NC terminals. In this case, the *COM* terminal is connected to the positive voltage of the battery and, since it has been preferred a Normally Closed behaviour, the *NC* terminal is connected to the red pin of the pump, as mentioned before.

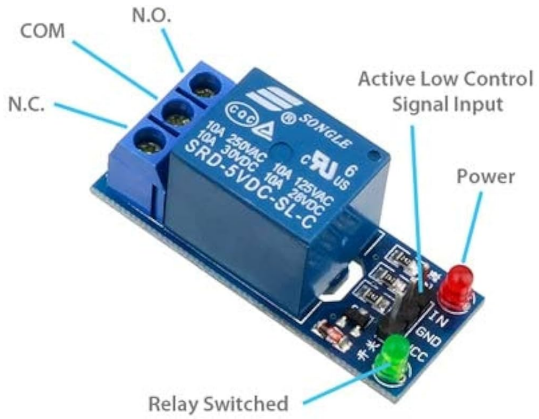


Figure 2.9: Relay shield.

Amplifier Circuit

The amplifier circuit is made up by a NPN PN2222 transistor, a Resistance of 1 kΩ and a Diode Rectifier. As mentioned before, the transistor allows a small current from the Arduino MKR1000 to control a larger current needed to activate the relay. In this case, the NPN transistor operates by turning on when a small current flows from the base to the emitter, allowing a much larger current to flow from the collector to the emitter. The transistor has 3 pins as shown in Fig. 2.10. The pin *1* is connected to the ground, the pin *2* is connected, through the resistance, to *D*7 Arduino's pin and the pin *3* is connected to relay's *IN* pin.

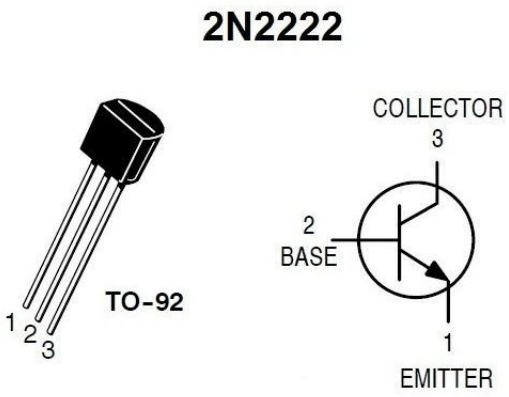


Figure 2.10: Transistor.

The aim of the resistance is to limit the current flowing into the base of the transistor to protect the Arduino board and ensure proper operation of the transistor.
The diode provides protection against voltage spikes generated when the relay coil is de-energized. This phenomenon is known as back electromotive force (EMF) and can damage the transistor or the Arduino board. The anode of the diode is connected to the transistor's pin *3* while the cathode is connected to the positive voltage of the battery.

Figure 2.11: Resistance and Diode

Other components

In this project also a Bread Board and different types of Jumper wires are used. The aim of the Bread Board is to expand the number of pins for connecting electronic components while the Jumper wires simplify the connection of various components.

Chapter 3

System and Network Set-up

This chapter will show the steps to prepare the system for different types of network structures. This set-up is due to the fact that both sensors and IP addresses change at every simulation.

3.1 Sensors calibration

In this case, the Sensor is divided into three areas *High*, *Medium* and *Low*, as shown in Fig. 3.1. The value of each level is obtained from a calibration of each sensor since the areas can be slightly different for every simulation and for every sensor. Moreover, each level has been assigned a value from 0 to 3, in ascending order where 0 corresponds to *Empty* and 3 to *High*, to be able to graph the results on *MATLAB*.

Figure 3.1: Low, Medium and High level of the Water Sensor

3.2 WiFi connection

In order to create a connection between the nodes, it is necessary to connect them all to the same WiFi. A group of nodes can communicate with each others through the IP addresses and, for this reason, these need to be acquired first.

Since the IP address is linked to the WiFi net, whenever the node is connected to a different WiFi rather than the previous one, it is necessary to reacquire it for every node.
In this project, it was used the IP address method of communication because it offers significant benefits in terms of ease of integration and data management.
When every node is connected to the same WiFi, to be sure that the process was made successfully, only the last triplet of digits should change as shown in Fig. 3.2.

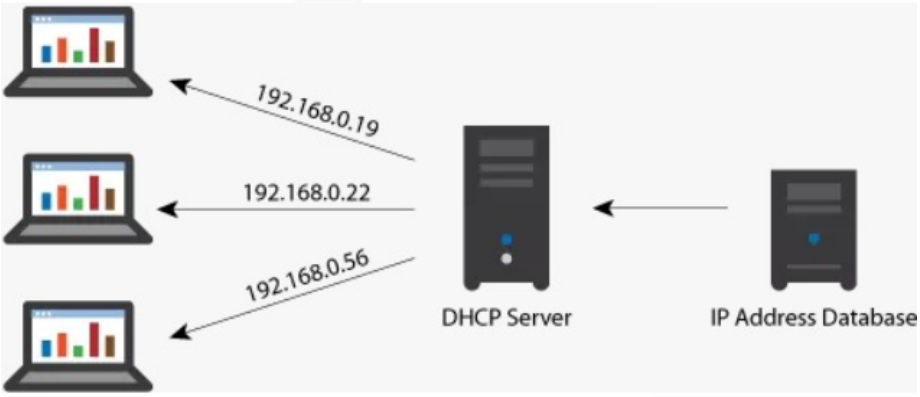


Figure 3.2: IP address communication

In this case, each node is characterized by an Arduino MKR1000 board connected to a different computer. Each computer collects the measurements acquired from the water sensor linked to its tank.

Chapter 4

Centralized Algorithm

In this chapter it will be shown the Centralized Algorithm, where a master node is chosen to collect all measurements from the other nodes. Based on the knowledge of the current environment, the master node will decide the best control action. The overall measurements will be collected offline through a *.txt* file and then analyzed using *MATLAB*.

4.1 Centralized Network Structure

Centralized systems are a type of computing architecture where all or most of the processing and data storage is done on a single central server or a group of closely connected servers. This central server manages all operations, resources, and data, acting as the hub through which all client requests are processed. The clients connected to the central server, typically, have minimal processing power and rely on the server for most computational tasks.
Specifically, the upper tank was chosen as master node while the bottom ones were selected as slaves. The master node received the water sensor's measurements from the slaves that can be *High*, *Medium* or *Low* according to the previous set-up. Based on these information, it decided whether or not to activate the pumps located in the lower tanks. This happens whenever the sensor of the master reads a different value rather than *High* and one of the lower tank's sensor had a higher value than the master level and, in another simulation, to ensure that the master level remained at a reference value.

4.2 Simulations and Results

This section will illustrate the methodologies used, present the data obtained and discuss the implications of the results emerging from the simulations.

4.2.1 Low Delay Test

In the initial test, a delay of 0.5 seconds to acquire and send data was applied to all three devices. This resulted in the master device receiving updated values from both slave devices with significant delay, suggesting a queuing effect as shown in the Fig. 4.1 and Fig. 4.2.

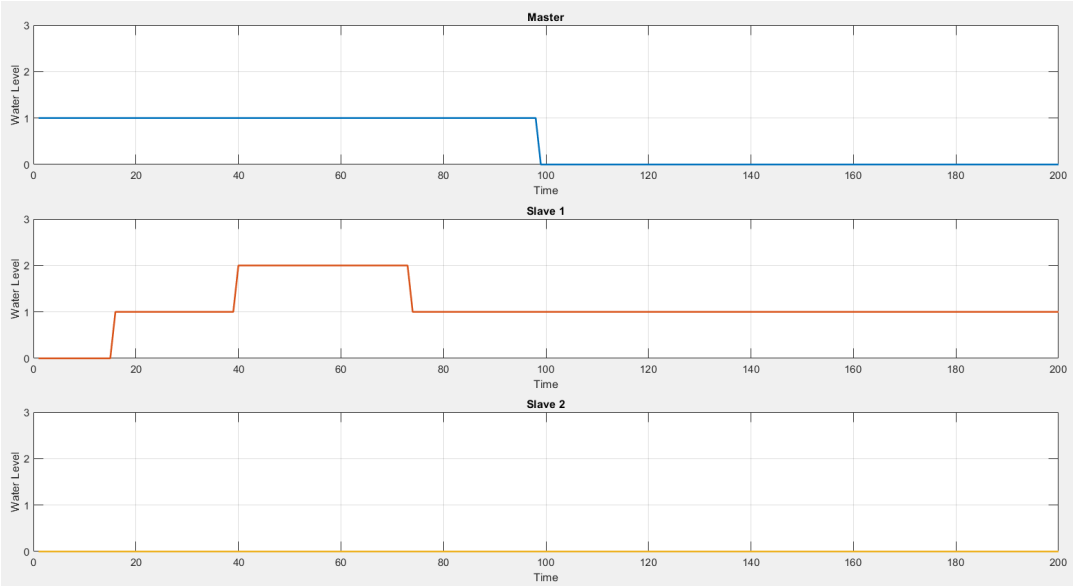


Figure 4.1: Water Levels

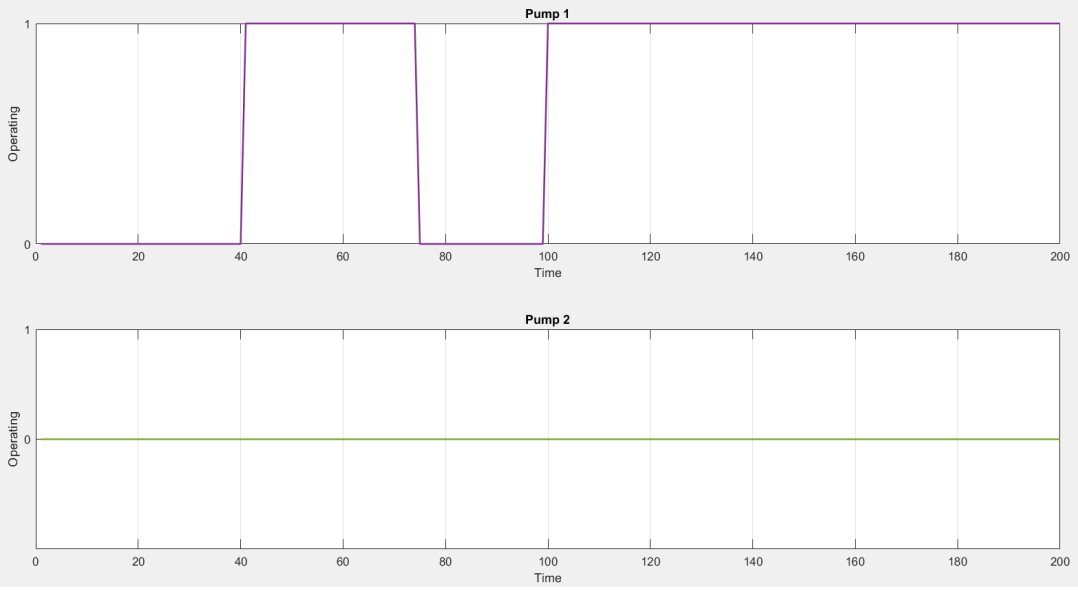


Figure 4.2: Pump Functioning

Moreover, it can be noticed that the master device attempted to activate the pumps and initially succeeded. Although, due to network congestion, it soon became unable to activate the pumps because the flags to signal the slaves to activate their pump were no longer sent. This is noticeable from the fact that slave's water levels stopped decreasing while master's one stopped increasing. In addition to this, it can be noted that one of the slave device lost priority in the queue and was unable to communicate at all with the master device, resulting in a fixed readout of its water level even though it was actually rising.
Eventually, the master device's water was completely drained to further fill the slave devices' water.

4.2.2 Accurate Delay Test

To address this, the period for the first slave was increased to 1 second and for the second slave to 2 seconds, while maintaining a 0.5 second delay for the master. Under these conditions, values were updated regularly and the system worked properly.

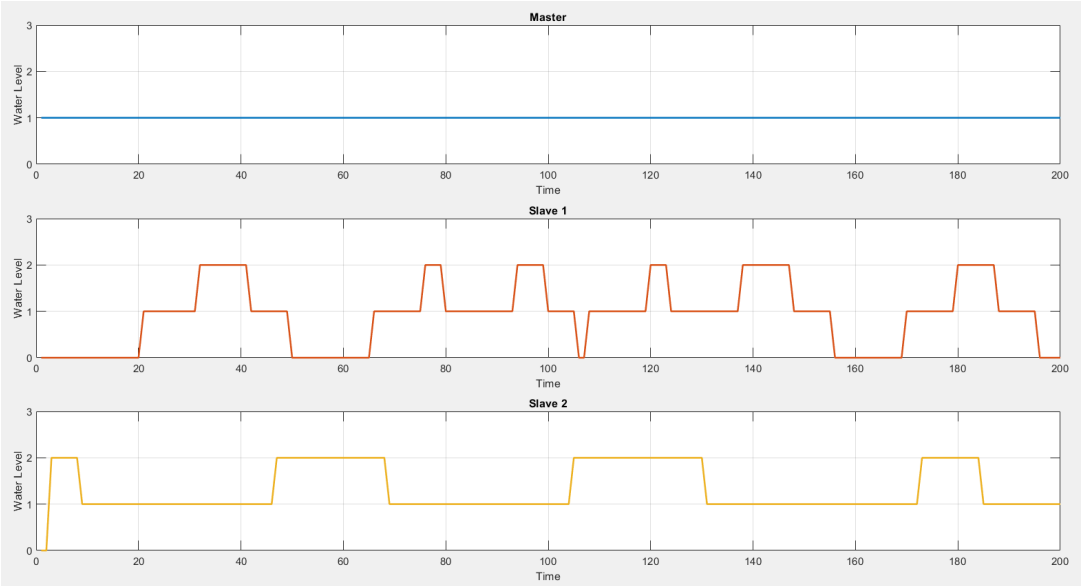


Figure 4.3: Water Levels

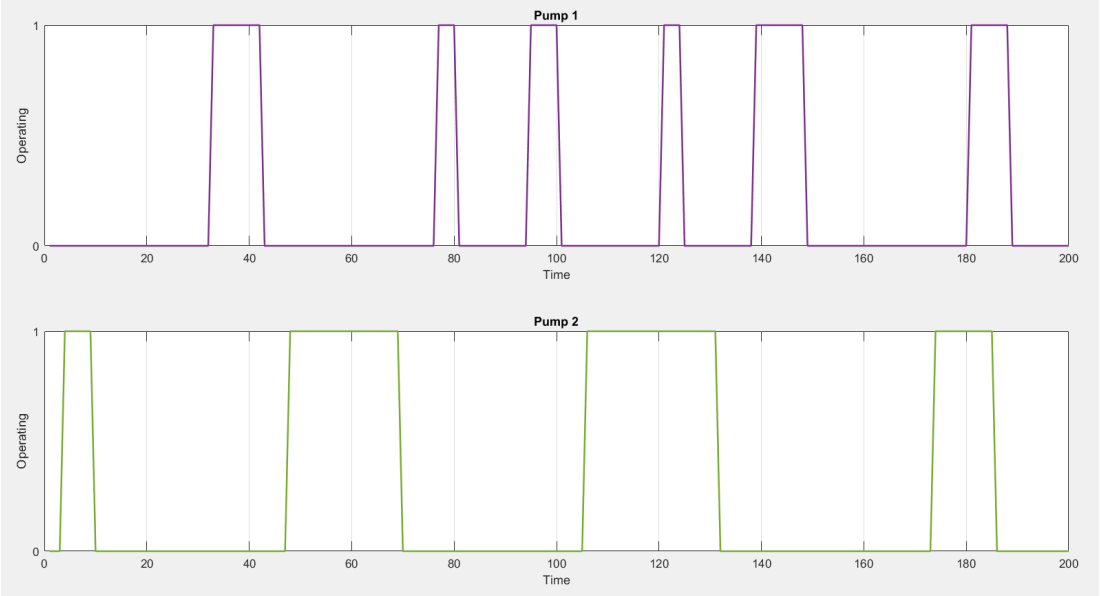


Figure 4.4: Pump Functioning

As shown in Fig. 4.3 Master's level of water never drops below 1 because the slaves always replenish it having an higher value and never raise above 1 since it's constantly spilling water to the bottom tanks. It should be noted that refueling occurs with a small, almost negligible, delay which can be seen by superimposing the two images in Fig. 4.3 and Fig. 4.4.
Testing the communication range, it was observed that moving the router away did not cause a gradual deterioration in performance, rather, the devices ceased to function entirely once the connection was lost. In fact, it was found that the devices maintained efficient communication up to a distance of 30 meters, despite the presence of a wall and some reduction in WiFi signal strength while, at approximately 35 meters with two walls in between, all devices lost connection and the system stopped working.
As an experimental variation, the master device was moved away from both the slaves and the router, followed by moving each slave individually. This resulted in a similar situation to the router being moved too far, with a slightly shorter but still effective communication range.

4.2.3 Constant Reference Control

This new simulation was done for a network where the master node had to remain at a constant reference value with the following results.

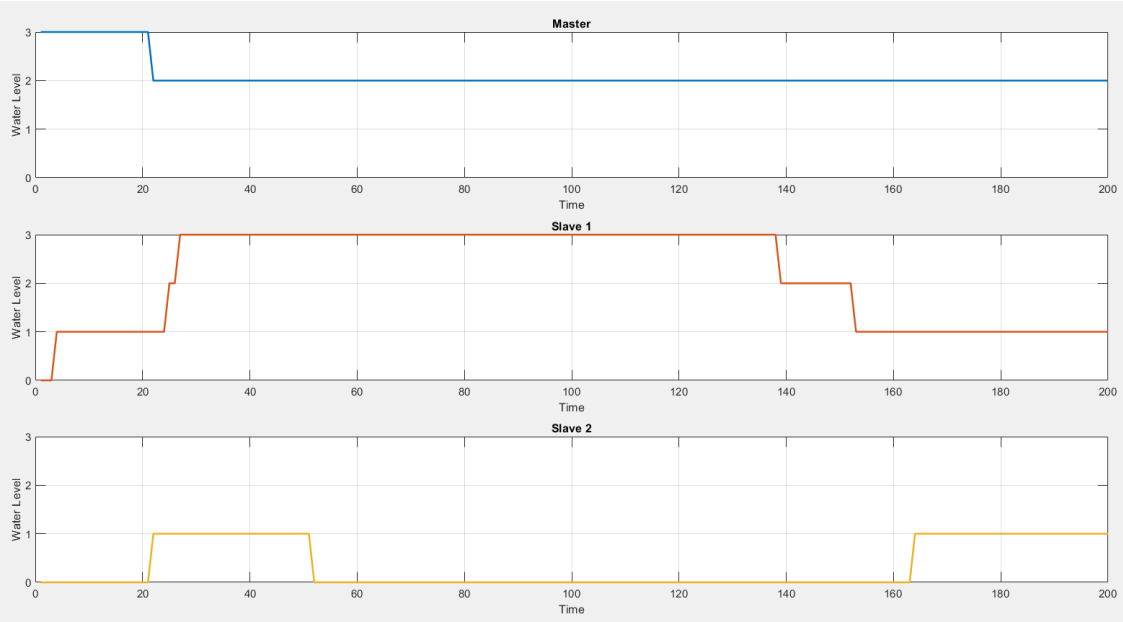


Figure 4.5: Water Levels

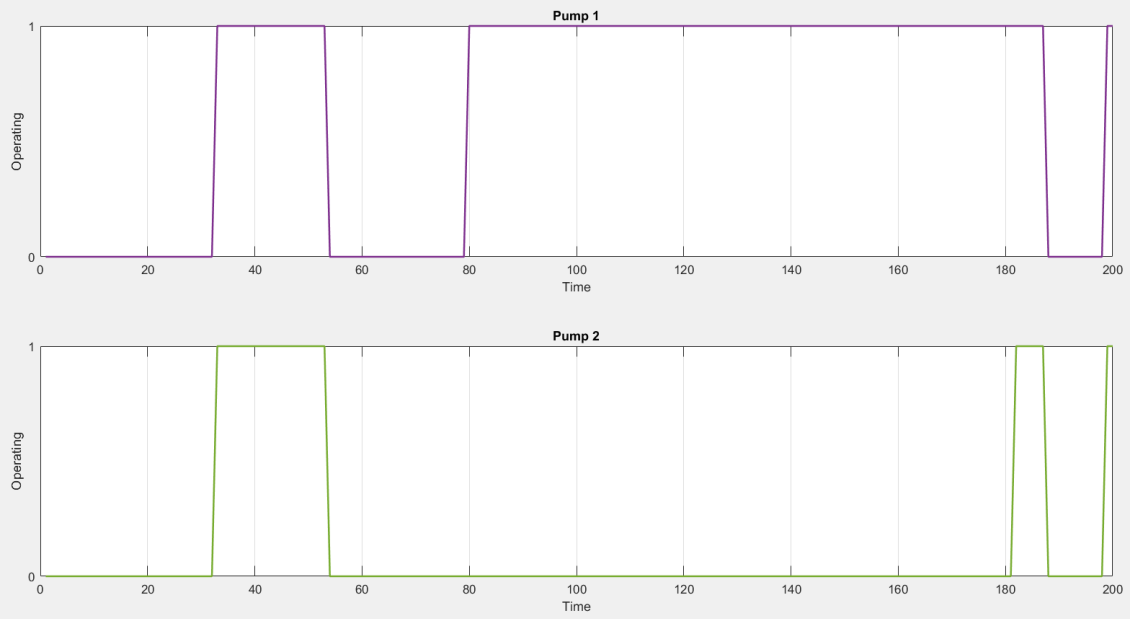


Figure 4.6: Pump Functioning

As illustrated in Fig. 4.5 and Fig. 4.6, the master maintained the reference level of *Medium* but it must be noted that it could have gone down to *Low* or *High* based on the calibration of the sensors or for any delays in the master sensor. To ensure better maintenance of the reference, an *Upper* and a *Lower Medium* bands have been added so that there was a greater margin for error caused by calibration or delays.

4.2.4 Hacker Attack Test

Moving on, the next and last test for centralized control, consisted in a simulation of a hacker attack in which one of the two slave devices, the second one in this case, sent packets with an excessive frequency to the Master, blocking the queue and not allowing access neither to the other slave nor to the master itself.

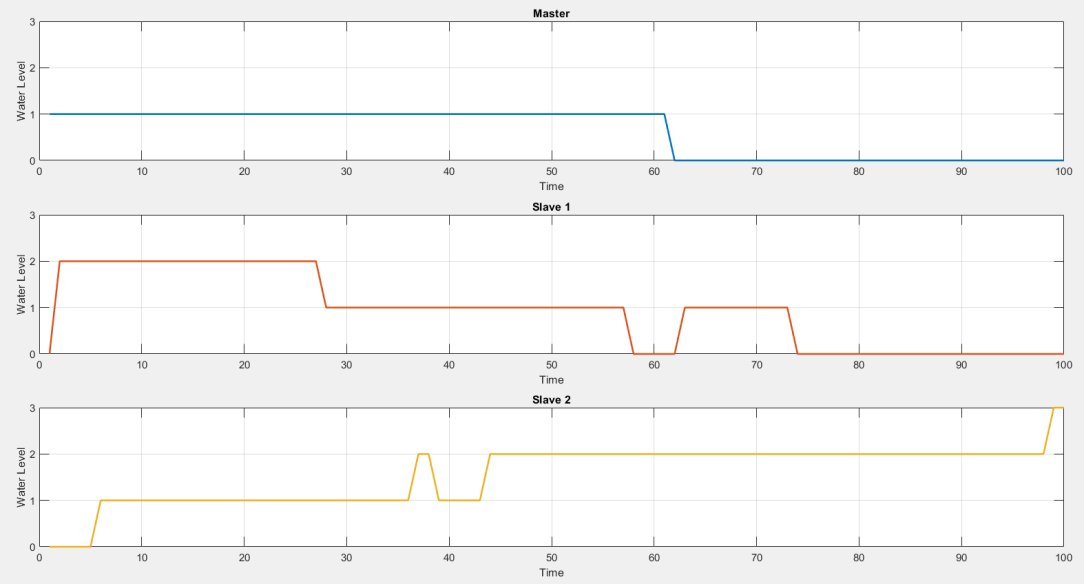


Figure 4.7: Water Levels

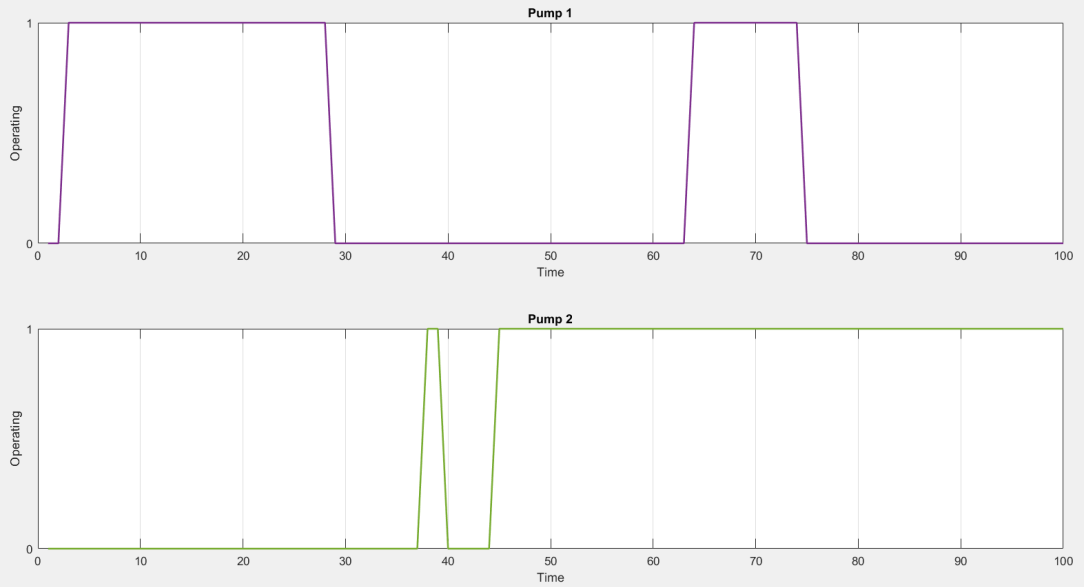


Figure 4.8: Pump Functioning

In fact, as can be seen from the fig. 4.8, the Master actually sent a signal to turn on the pump but, in practice, this signal never managed to arrive and the pump remained turned off leading Master's level to be *Empty* and second Slave's level to become *High* as can be seen in Fig. 4.7.

4.3 Centralized Algorithm: Advantages and Limitations

The use of a centralized control algorithm in network structures offers several notable advantages.
Firstly, it simplifies management and monitoring, as all operations are controlled by a single master node. This centralization streamlines the decision-making process and ensures better coordination among various components. Additionally, resource optimization is another significant benefit. By concentrating computational tasks on a central master node, the system can utilize resources more efficiently, allowing slave nodes, which typically have lower computational capabilities, to focus on specific tasks such as data collection.
Another advantage of centralized systems is improved data integrity. Since all data processing and storage occur in a central location, the risk of data discrepancies is reduced, ensuring more consistent and reliable data management. Lastly, centralized systems offer scalability. In fact, upgrading the master node or adding more resources to it can enhance the system's capabilities without significantly altering the existing infrastructure.
On the other hand, the centralized control algorithm also has several disadvantages. One of the primary drawbacks is the risk that if the master node encounters a problem or fails, the entire system can become inoperative, leading to significant downtime and potential data loss. For instance, if the master node is overwhelmed by too many requests or computational tasks, this might lead to delays and worse performance of the whole system. Moreover, after a hardware failure, data recovery and system restoration can be more complicated compared to distributed systems.
Finally, security vulnerabilities are another concern. If an attacker gains access to the master node, they could potentially control or disrupt the entire system.
In conclusion, while centralized control provides benefits such as simplified management, resource optimization, and improved data integrity, having a single master node also brings disadvantages since, in case of failure or attack, the system is more vulnerable and this can make the restoration of settings and data more arduous.

Chapter 5

Distributed Algorithm

In this chapter there will be shown Distributed Algorithms, where the three nodes involved are in a configuration where a node, called A, communicates with the other two nodes, B and C, while the other two just communicate with the first one as shown in Fig. 5.1.
The overall measurements will be collected offline through a *.txt* file and then analyzed using *MATLAB* after that the algorithms were firstly tested on *MATLAB* to show their theoretical validity.

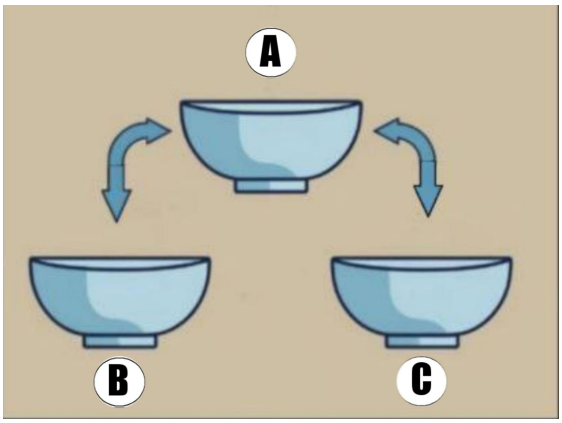


Figure 5.1: Basins Scheme

5.1 Distributed Network Structure

Distributed systems are a type of computing architecture where, when the algorithm starts, each node has its own initial state that specifies the level of water in the tank. Nodes do not have a complete view of the global state of the system, but they can communicate with neighboring nodes to exchange information during the execution of the algorithm.
In this project, it was firstly exposed a net in which there was an unstable node that fell after few cycles and, to address this, three different solving algorithms were proposed: a Dynamic

Consensus where the node was cut off and a Robust Control that removed it once it felt. In parallel, the same net without the fallen node was implemented using the algorithms seen before with the addition of a Weighted Dynamic Consensus which weighted less the contribution of unreliable nodes.

5.2 Simulation and Results

This section will illustrate the methodologies used, present the data obtained and discuss the implications of the results emerging from the simulations.

5.2.1 Dynamic Consensus Algorithm

The Dynamic Consensus algorithm is a protocol used to achieve agreement on a single data value among distributed processes or systems. Its primary purpose is to ensure that all nodes in a network converge to a common state or value, even in the presence of failures or unreliable communication.
For this project, the goal was to achieve Consensus on the total average of the tank's water level through providing, for every node, the estimated value to the neighbor nodes using the following formula:

$$x_i(k+1) = x_i(k) - \varepsilon \sum_{j \in J} a_{ij}(x_i(k) - x_j(k)) + z(k+1) - z(k)$$

Where:

- $x_i(k+1)$ and $x_i(k)$, are the current and previous estimated values that the node itself updates and sends, in each cycle, to its neighbor nodes.
- $x_j(k)$, is the previous estimated value of a neighbor node of i.
- ε, constitutes the step with which the algorithm is executed and was chosen equal to 0.4 in accord to its upper limit of $1/d_m$, where d_m is the maximum degree of the nodes, to ensure the stability of the algorithm.
- a_{ij}, are the elements of the symmetric adjacency matrix A and are equal to 1, if there is a connection between node i and node j, otherwise the value is 0.
- J, consists of the set of neighbors of the considered node i.
- $z(k+1)$ and $z(k)$, represents respectively the current and the previous level of the water of the considered tank.

Once the Consensus value was achieved for every node, the current level of water z was updated to reach a new equilibrium whose value should be, in theory, the average of the initial conditions.

In the following graphics are reported the results simulated on *MATLAB* for a functioning net with 3 nodes and for the same net where one of the node was detached after a while.

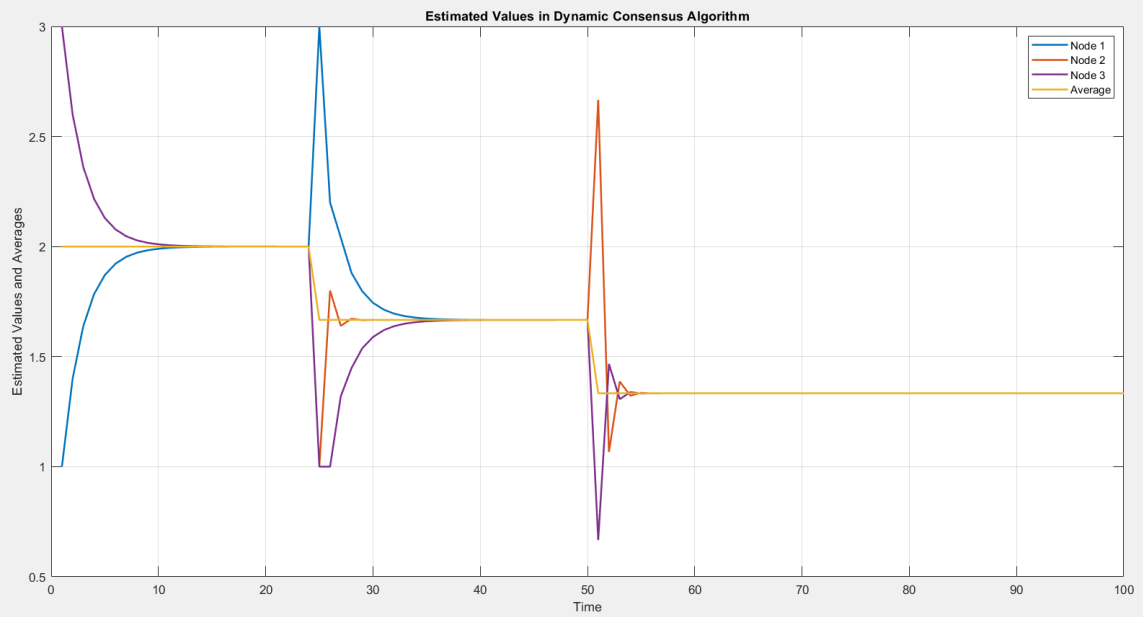


Figure 5.2: Dynamic Consensus on *MATLAB*

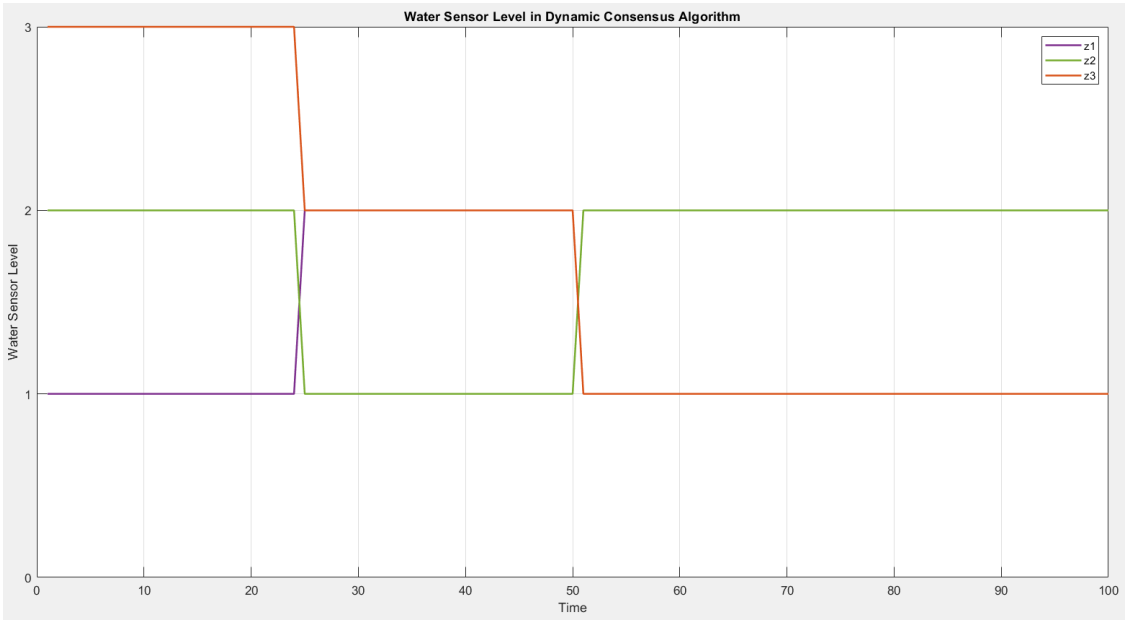


Figure 5.3: Water Levels on *MATLAB*

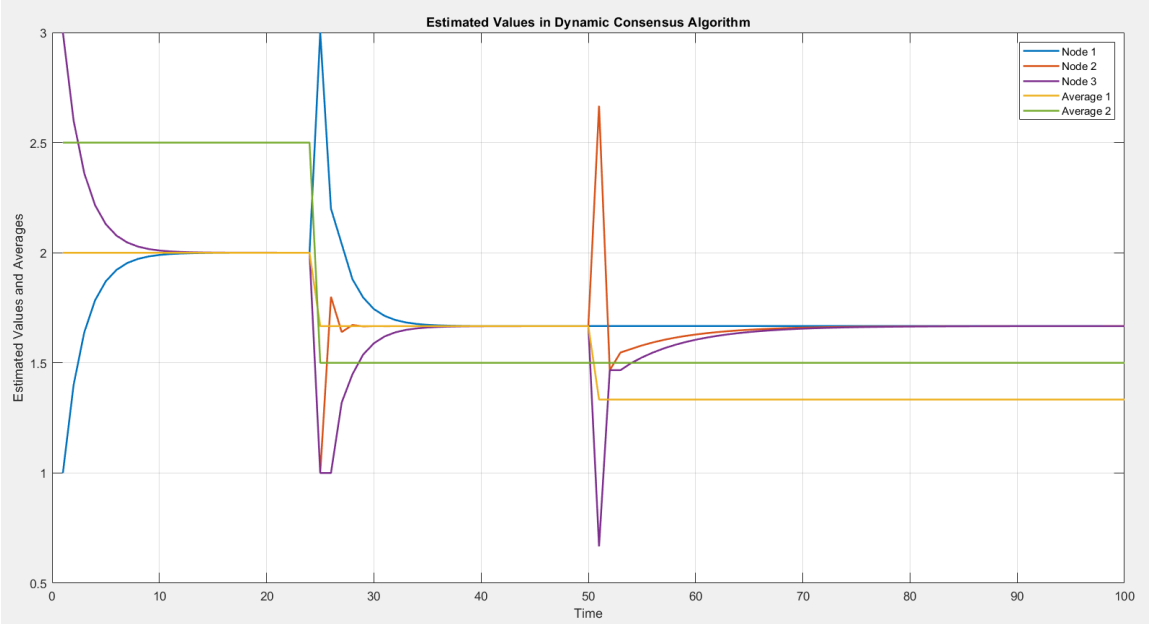


Figure 5.4: Dynamic Consensus on *MATLAB* with fallen node

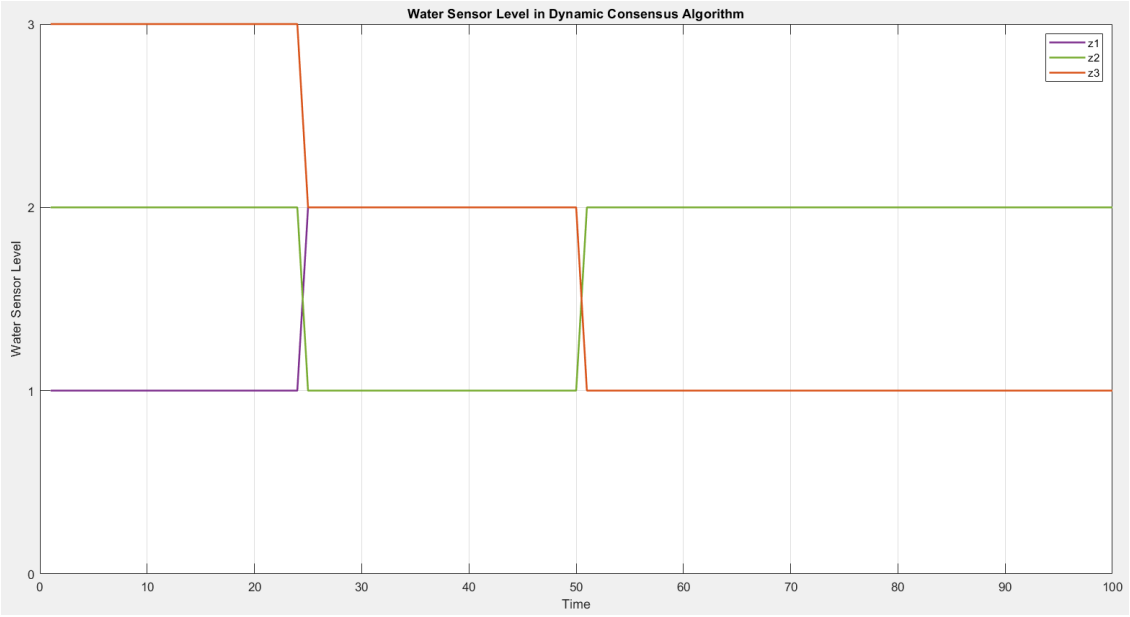


Figure 5.5: Water Levels on *MATLAB* with fallen node

It can be noticed in Fig. 5.2 and Fig. 5.3 that Consensus for the 3 nodes is always reached on their overall average. This average changed because, to represent a simulation as close to the real one as possible, the sensors were not all calibrated perfectly in the same way and, therefore, the overall average did not remain constant and, consequently, the nodes reached

3 Consensus values in 3 different moments.
The following graphics in Fig. 5.4 and 5.5 show that, since at time instant 40 node 1 was detached from the net, the nodes initially reached Consensus on the total average but then node 1 stabilized at its own water level value, since it was detached from the net, bringing the other two nodes to not reach Consensus on their average but on the same value of the detached node.
The same algorithm was tested at first on the faulty network without any solution algorithm for the unstable node problem.

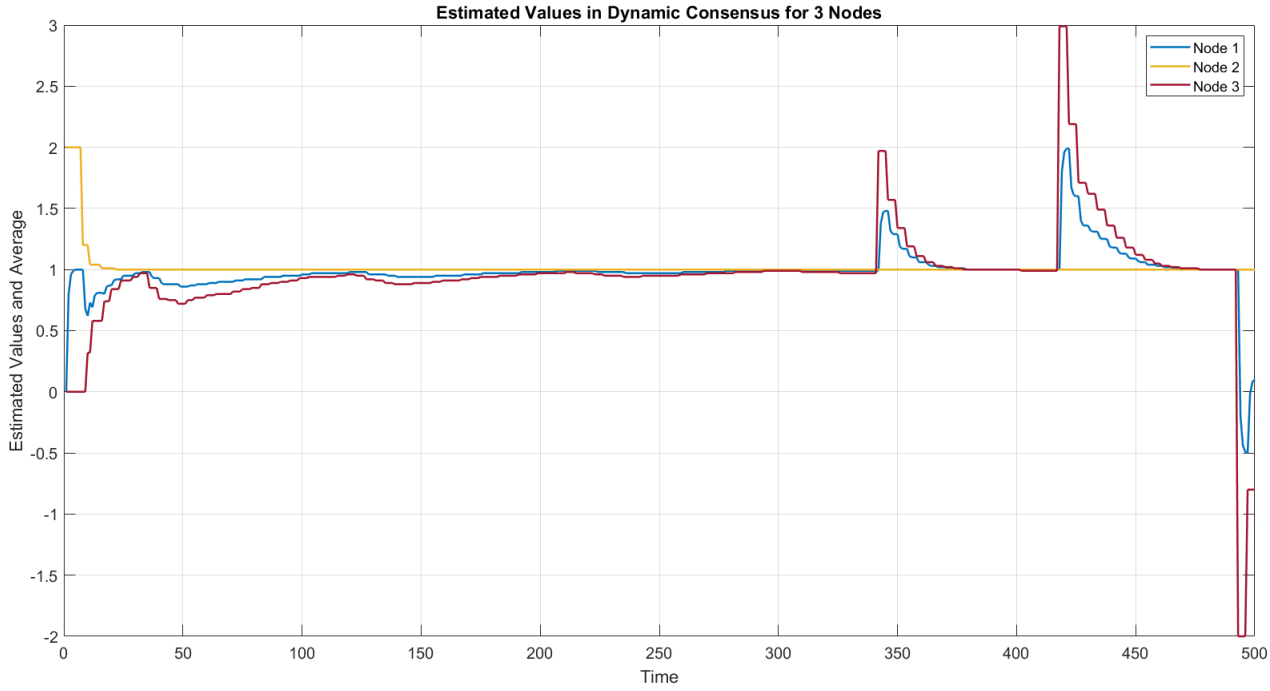


Figure 5.6: Dynamic Consensus on faulty real net

As can be seen from Fig. 5.6, the presence of the unstable node led the network to stabilize on an incorrect Consensus value equal to the value of x_2 since, after the fall of the node, its last value continued to influence the estimate of the central node x_1 and, consequently, also that of the other node x_3. For this simulation it was needless to report the average and the water levels because the net stabilized almost instantly on another value.
As anticipated, the first and simplest solution for the fallen net was to eliminate the unstable node and reach Consensus for the distributed network composed of the other two nodes.

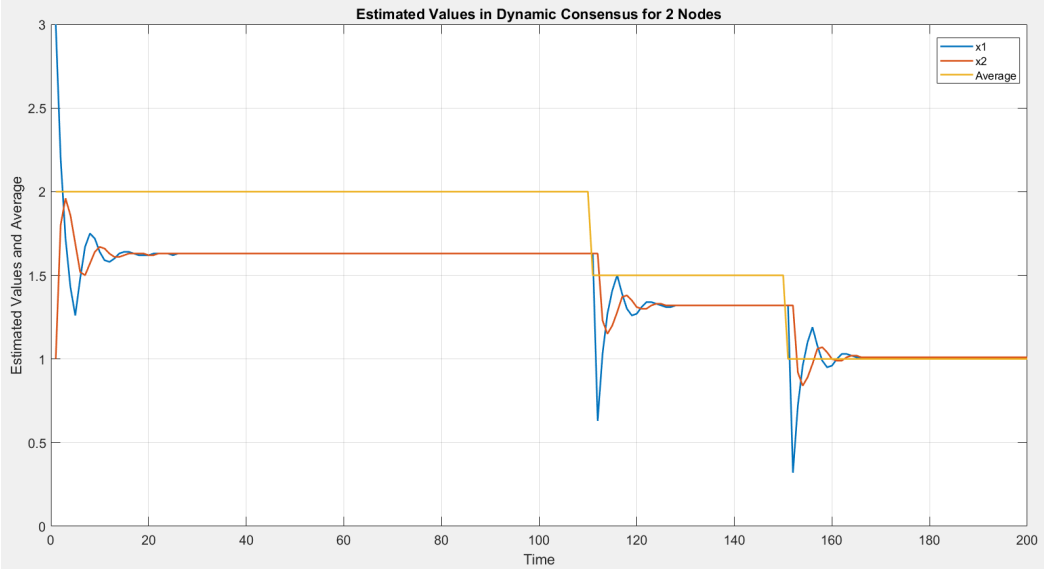


Figure 5.7: Dynamic Consensus of the reduced faulty real net

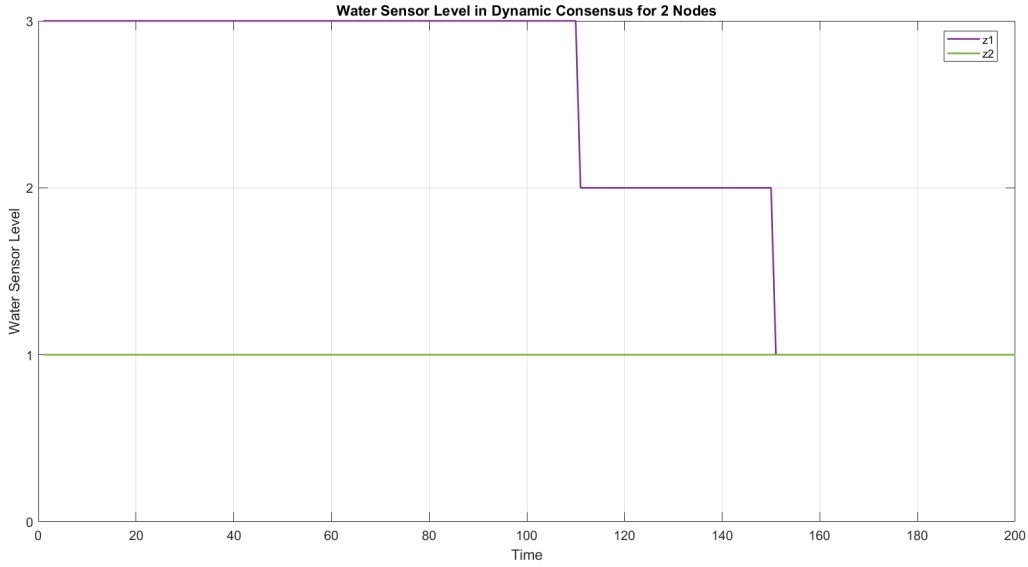


Figure 5.8: Water Levels of the reduced faulty real net

The results reported in Fig. 5.7 and Fig. 5.8 evidence the situation that was previously predicted where, due to the different calibration of the sensors, there is a variation in the average even if the total amount of water did not change during the simulation. Despite that, the Consensus for the reduced net was reached, after only three adjustments, as expected.

The same algorithm was tested for the functioning net with the following results.

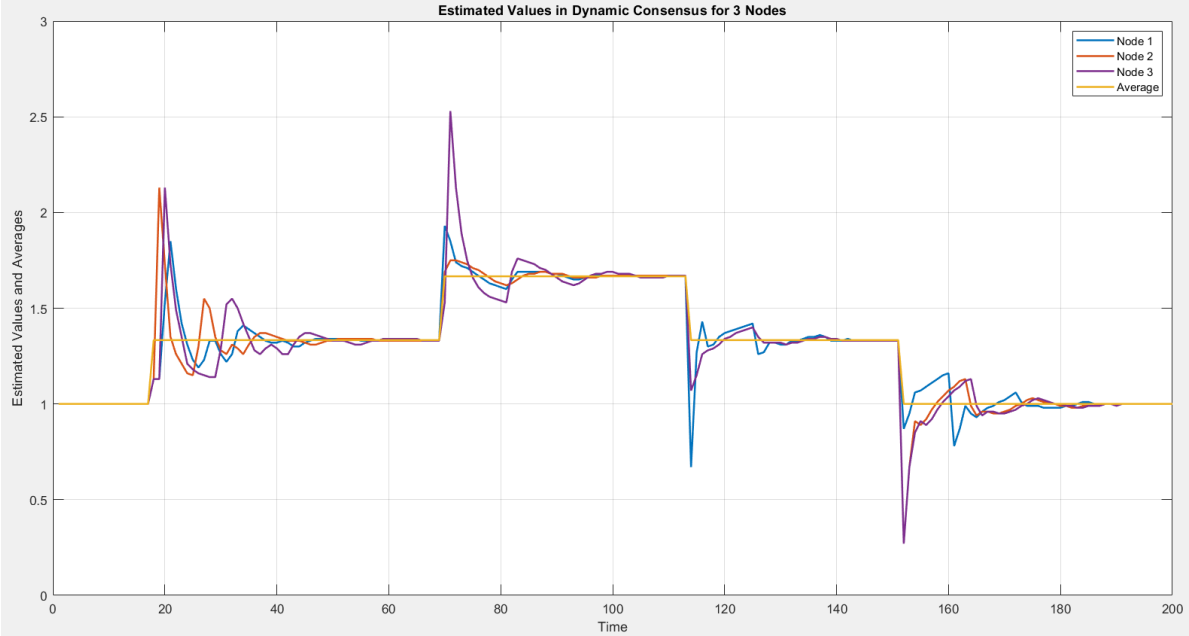


Figure 5.9: Dynamic Consensus on healthy real net

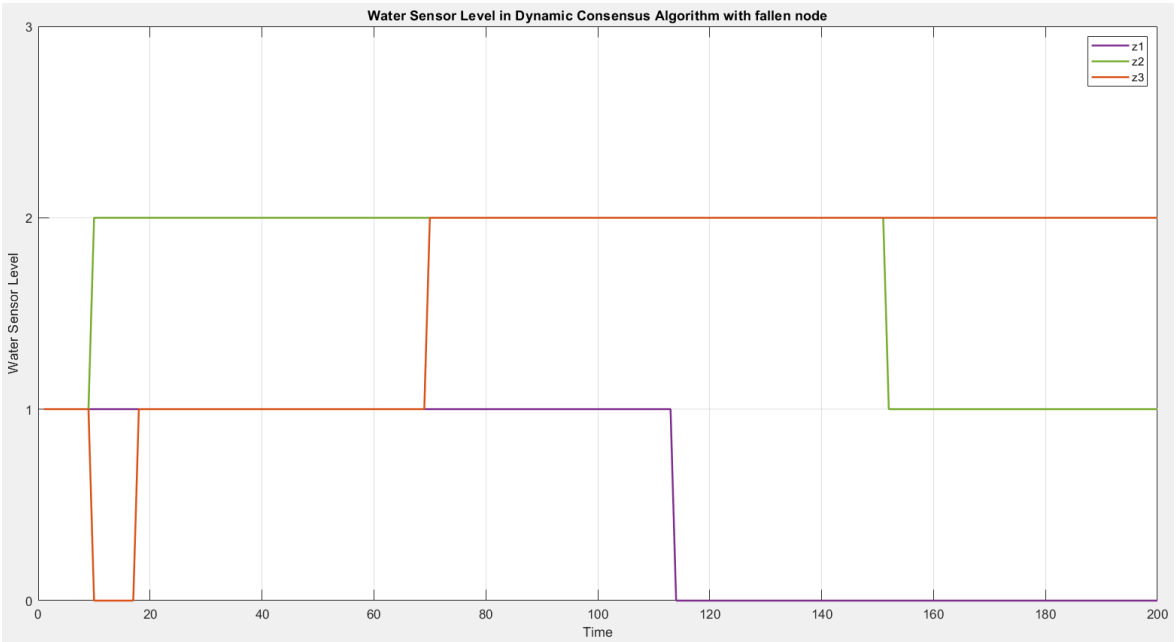


Figure 5.10: Water Levels on healthy real net

5.9 and Fig. 5.10 show the same behavior expected from the theoretical tests ran on *MATLAB* except for some small negligible imperfections.

After this, it was simulated a test where a node was dropped mid-execution with the following results.

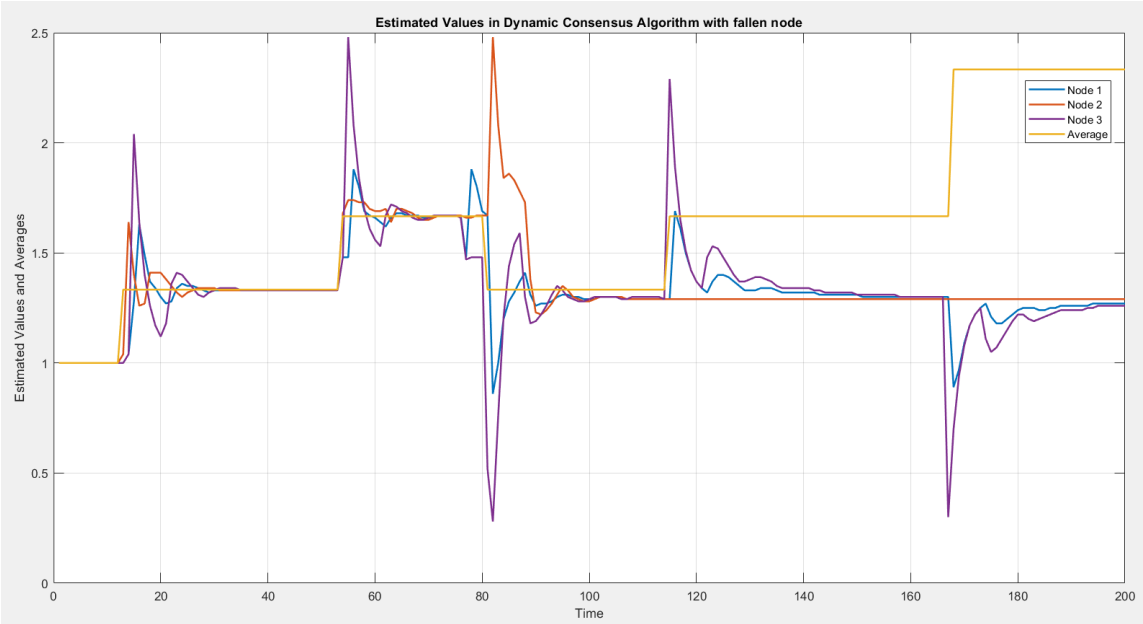


Figure 5.11: Dynamic Consensus on healthy real net with fallen node

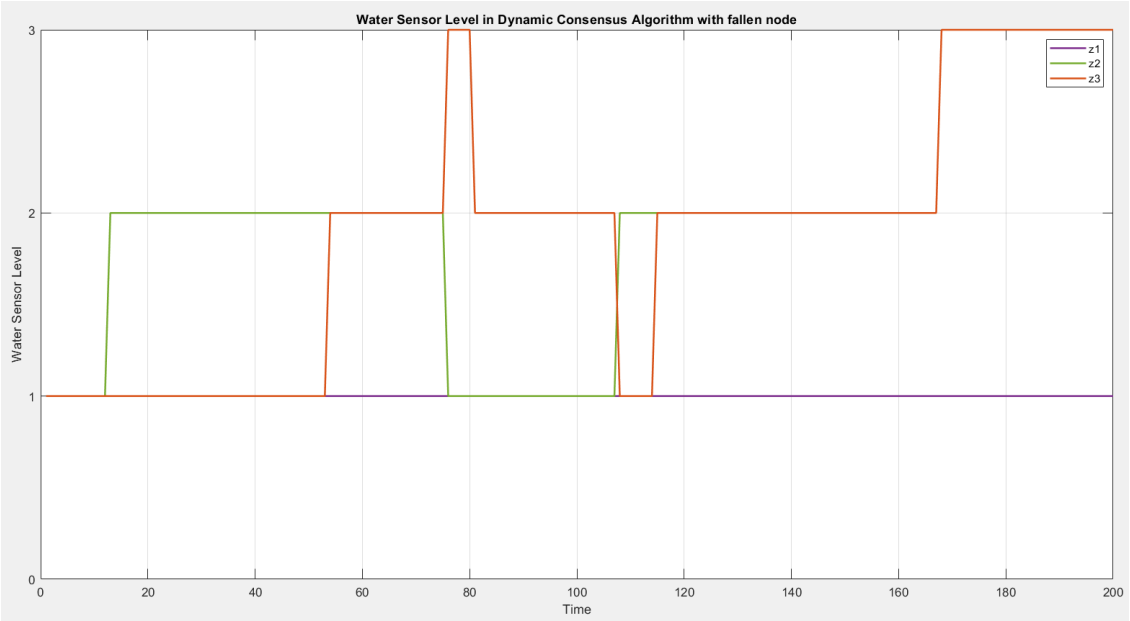


Figure 5.12: Water Levels on healthy real net with fallen node

This simulation, shown in Fig. 5.11 and Fig. 5.12, had results that are coherent with the ones saw in the theoretical tests done on *MATLAB* with the nodes reaching Consensus on

the total average until one of them felt and then adapting the new Consensus to the value of the fallen node.

5.2.2 Weighted Dynamic Consensus Algorithm

Moving on, building on the principles of the Dynamic Consensus algorithm, the Weighted Dynamic Consensus algorithm was implemented. The algorithm introduces various weights w_i for the nodes to account for differences in their importance or reliability giving a higher weight to the better nodes. In fact, if w_i are all the same then it coincides with the previous algorithm. This modification allows the system to converge to a consensus that better reflects the weighted contributions of each node. The updated formula for the estimated value that each node provides to its neighbors is given by:

$$x_i(k+1) = x_i(k) - \varepsilon \left(\frac{1}{w_i}\right) \sum_{j \in J} a_{ij}(x_i(k) - x_j(k)) + z(k+1) - z(k)$$

Where:

- w_i, represents the weight assigned to a node i based on its quality and reliability. In particular, it was chosen a weight of 4 for the middle node, a weight of 3 for the side stable node and a weight of 1 for the unstable one.
- ε, still constitutes the step with which the algorithm is executed but, for this algorithm, it was chosen equal to 0.3 in accord to its upper limit of $\min_i \left\{\frac{w_i}{d_i}\right\}$, where d_i is the degree of the node i.

Regarding the other parameters ($x_i(k+1)$, $x_i(k)$, $x_j(k)$, a_{ij}, J, $z(k+1)$ and $z(k)$) they remain as defined in the previous algorithm.
This algorithm was firstly implemented on *MATLAB* with the results shown in sequence.

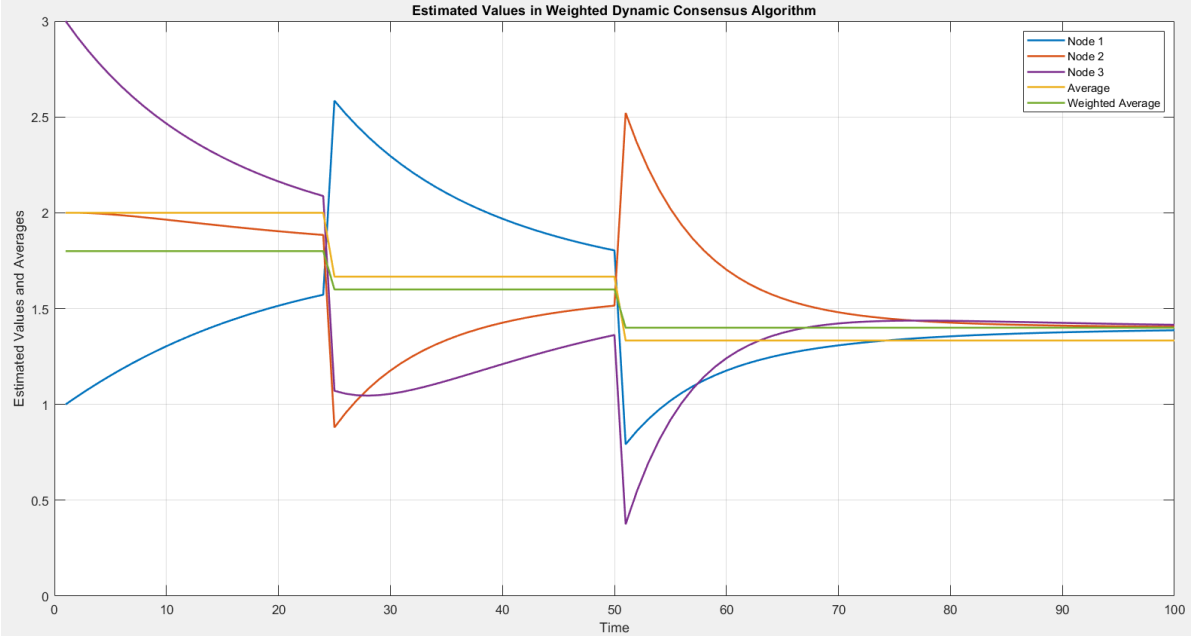


Figure 5.13: Weighted Dynamic Consensus on *MATLAB*

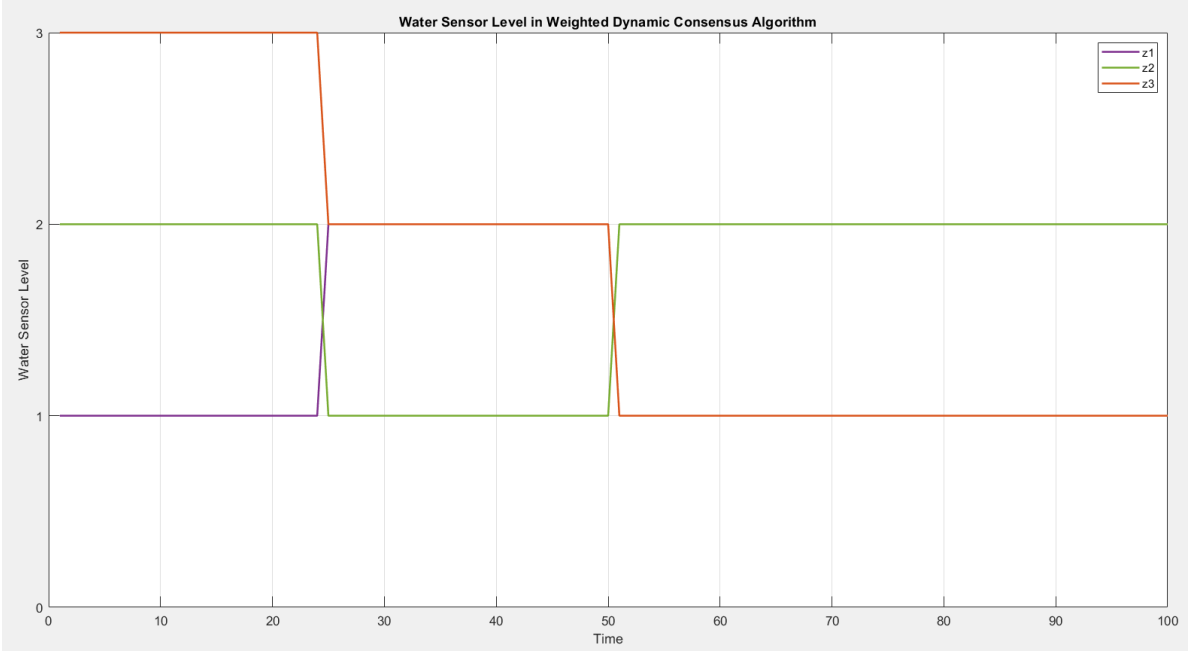


Figure 5.14: Water Levels on *MATLAB*

As expected, the nodes no longer reached Consensus on their average but on the weighted one according to the assigned weights that were given to each node based on their reliability. In particular, it was assigned a higher weight to node 1 and 2 than to node 3 making this

last one less relevant and impacting in the reaching of Consensus.
Afterwards, the same algorithm was tested on the real net with the following results.

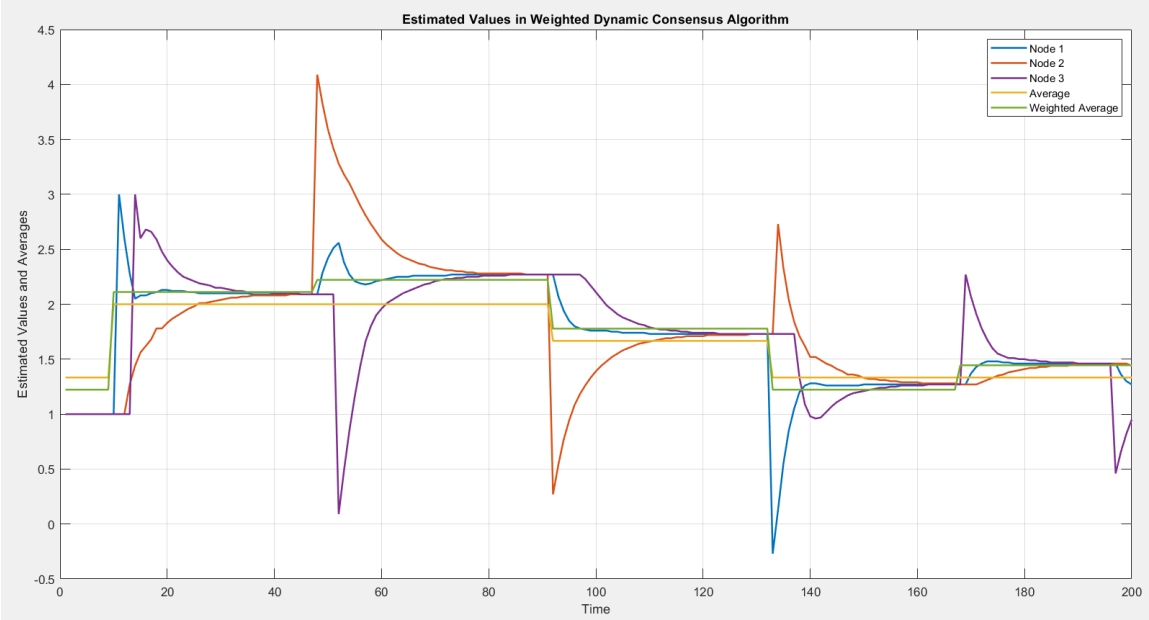


Figure 5.15: Weighted Dynamic Consensus on healthy real net

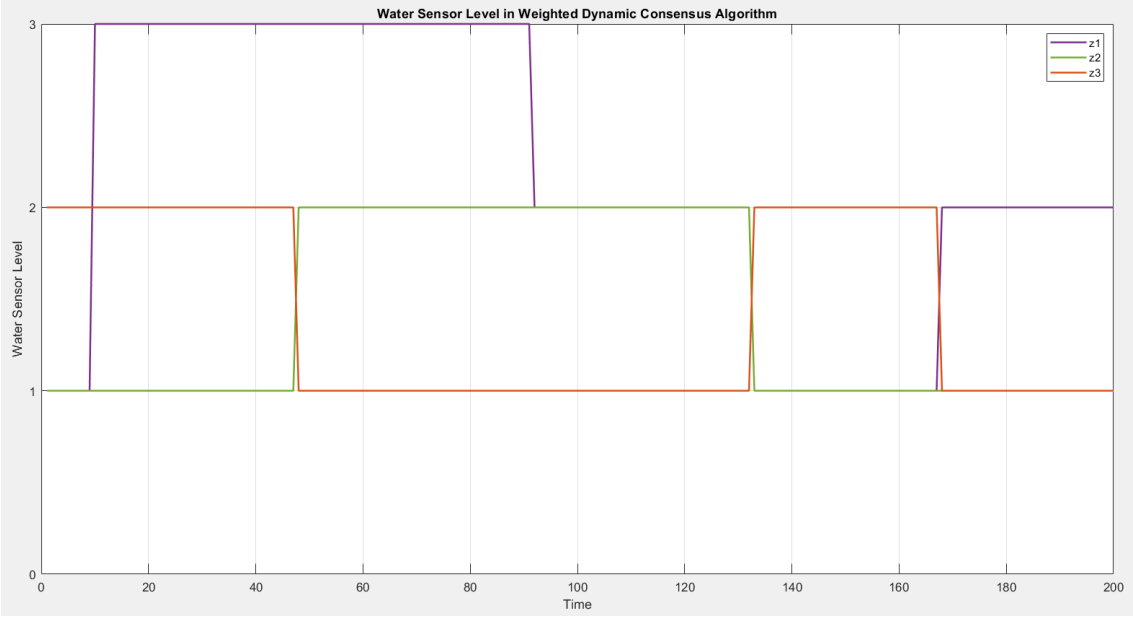


Figure 5.16: Water Levels on healthy real net

In accordance with what was stated and tested on *MATLAB*, the results illustrated in Fig.

5.15 and Fig. 5.16 evidence that, by assigning a lower weight to one of the nodes, the system converged to the weighted average rather than the standard one.
It can be noticed that this particular algorithm was not tested for the net with the faulty node since its application is valid when the values provided by the nodes are differently reliable rather than when a node totally falls as in the previous examples.

5.2.3 Robust Control Algorithm

The Robust Control Algorithm was introduced to address the limitations observed in previous algorithms, providing a more reliable and resilient solution for achieving Consensus in distributed systems in presence of a fallen node.
By implementing this algorithm, the system can dynamically adjust the influence of each node based on the current network conditions, thereby improving the overall robustness. This approach ensures that the consensus process remains stable and accurate, even in the event of node failures or unreliable communication. The key feature of this algorithm is its ability to adapt to changes in the network and provide a fault-tolerant solution according to the following equations:

$$\begin{cases} \delta_{ij}(k+1) = \delta_{ij}(k) - a_{ij}(x_i(k) - x_j(k)) \\ x_i(k+1) = \varepsilon \sum_{j \in J} \delta_{ij}(k+1) + z_i(k+1) \end{cases}$$

Where:

- $\delta_{ij}(k+1)$ and $\delta_{ij}(k)$ represents the rate of change of the control variable between node i and node j.

For what concerne the other parameters (ε, $x_i(k+1)$, $x_i(k)$, $x_j(k)$, a_{ij}, J and $z_i(k+1)$) they remain as defined in the Dynamic Consensus algorithm.
This algorithm was firstly tested on *MATLAB* in a net with two, and then three, nodes with the results reported in the following graphics.

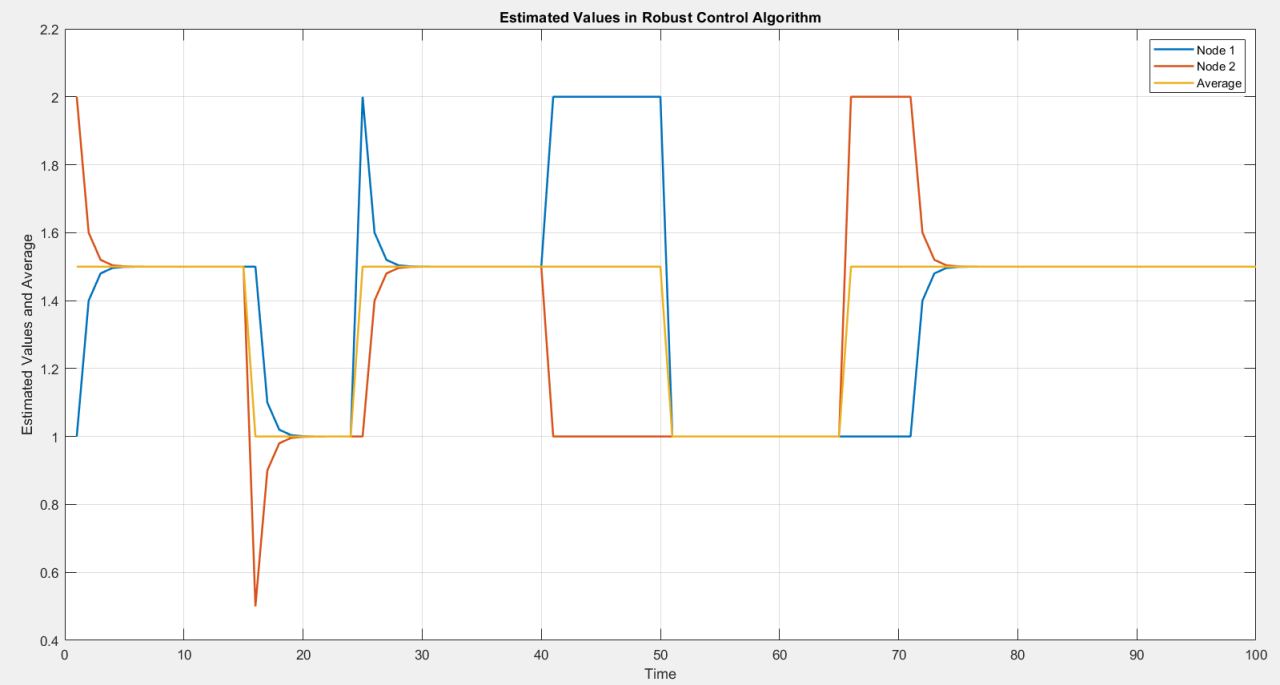


Figure 5.17: Robust Control in *MATLAB* with 2 nodes

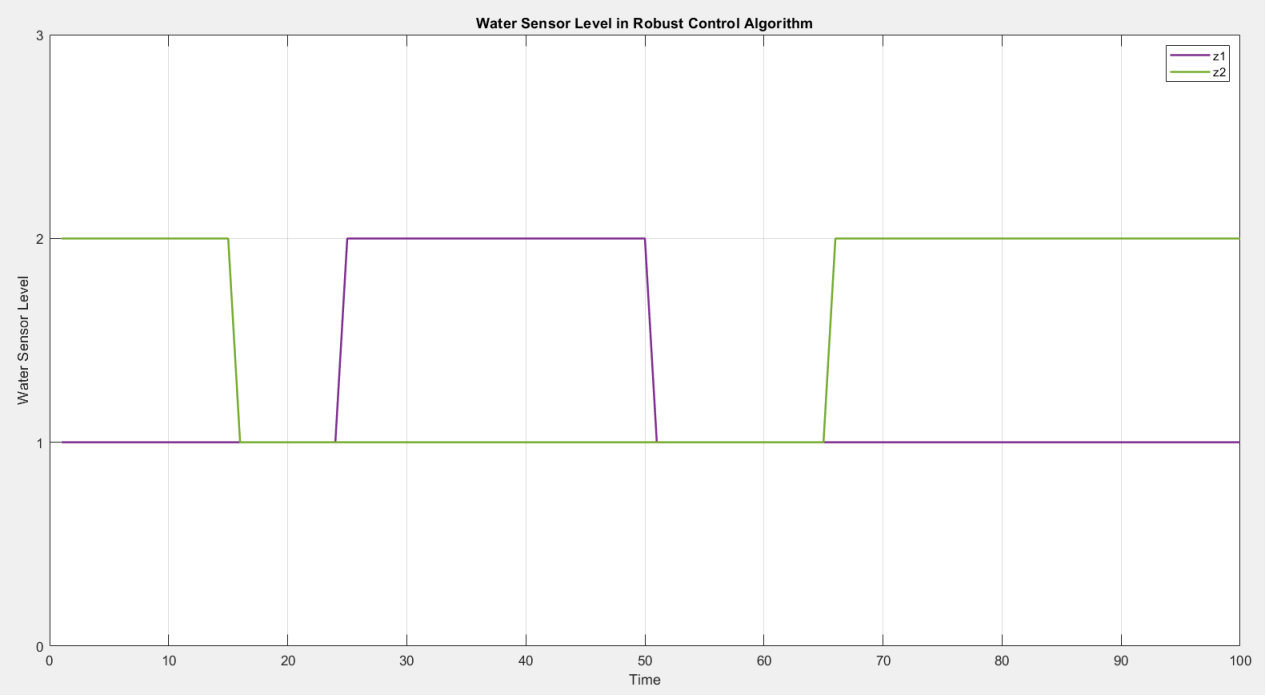


Figure 5.18: Water Levels in *MATLAB* with 2 nodes

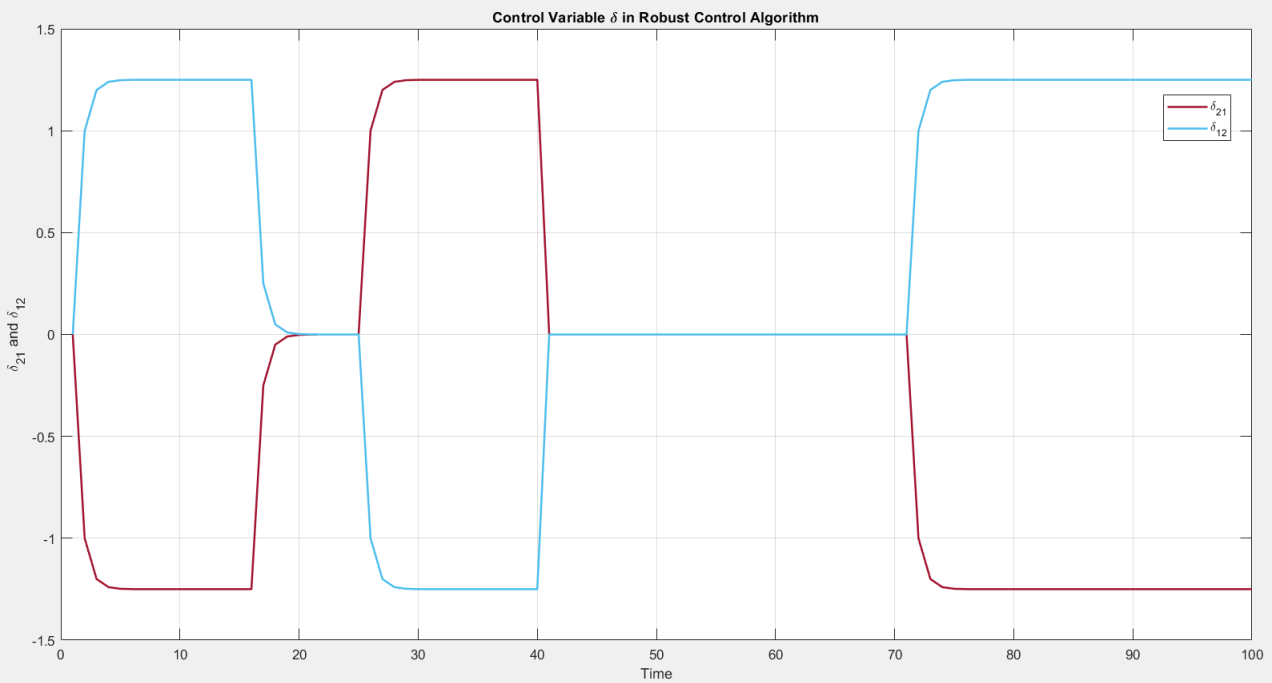


Figure 5.19: δ in *MATLAB* with 2 nodes

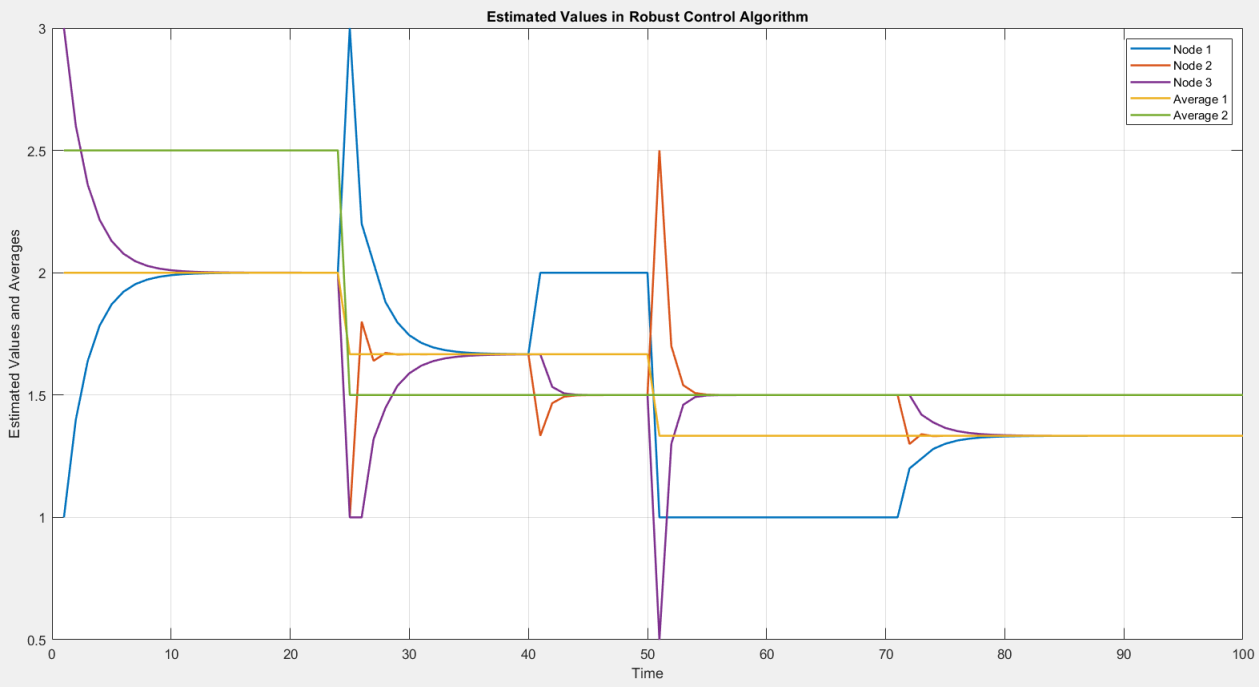


Figure 5.20: Robust Control in *MATLAB* with 3 nodes

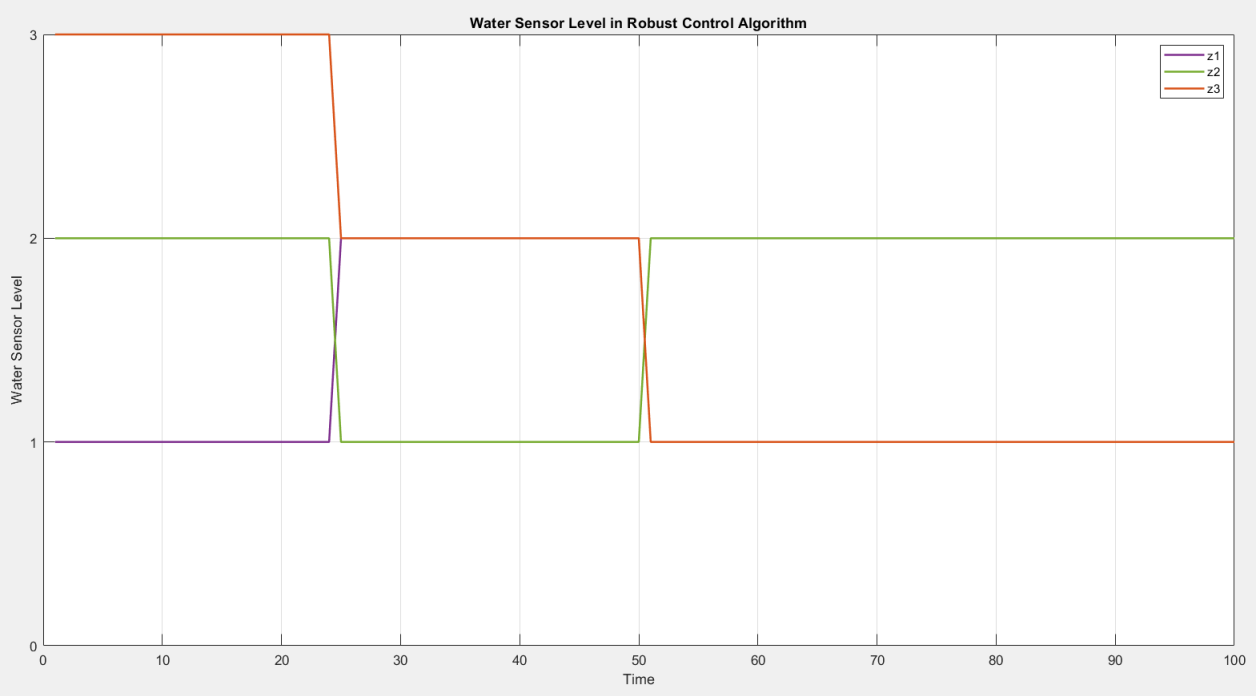


Figure 5.21: Water Levels in *MATLAB* with 3 nodes

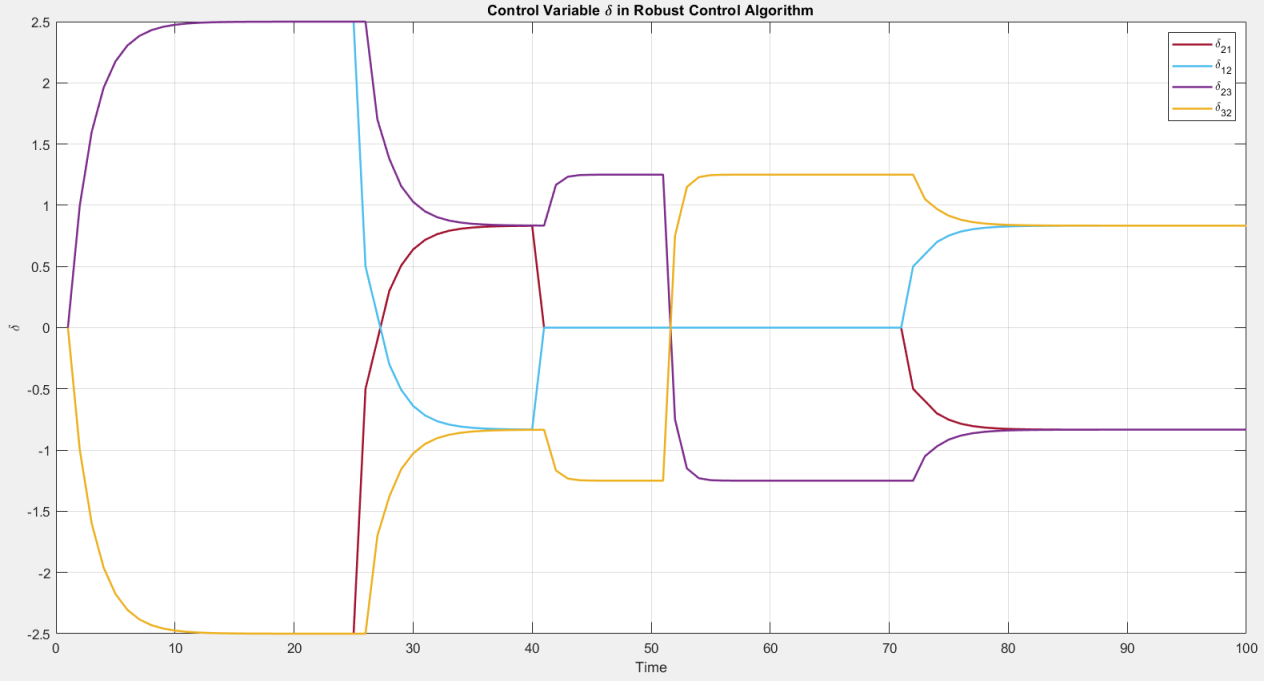


Figure 5.22: δ in *MATLAB* with 3 nodes

In both *MATLAB* simulations it was supposed a malfunction in which node 1 detached at time step 40 and reattached at time step 70. When this occurred, nodes in Fig. 5.17 no longer went to the total *average* but to the value of their own sensors value until the node was reattached. This happens as well in the next simulation with three nodes as shown in Fig. 5.20 the only difference is that this time the other two nodes didn't go to their sensors value but to the new *average* made on these two.
It is also reported the course of water levels in Fig. 5.18 and Fig. 5.21 with values decided offline.
Eventually, it can be seen that both in Fig. 5.19 and Fig. 5.22 the control variable δ are opposite in pairs as expected.
Moving on, this algorithm was applied on the $3-nodes$ faulty real net. The results are reported below.

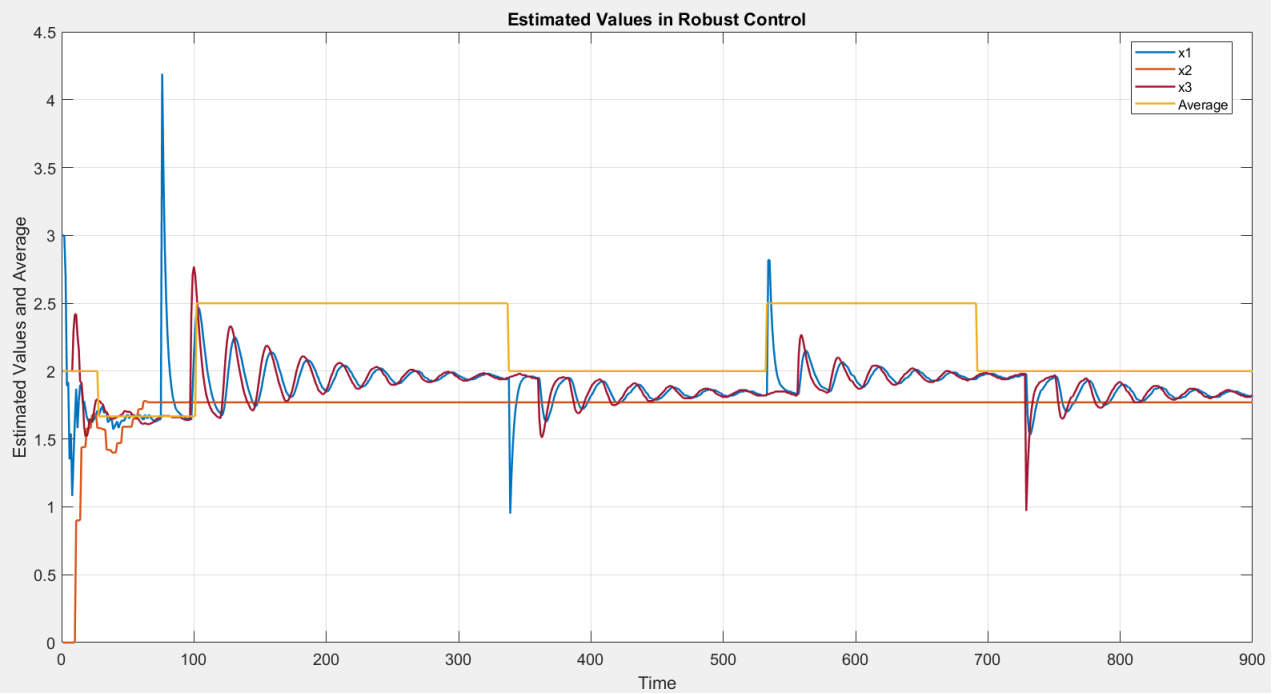


Figure 5.23: Robust Control on faulty real net

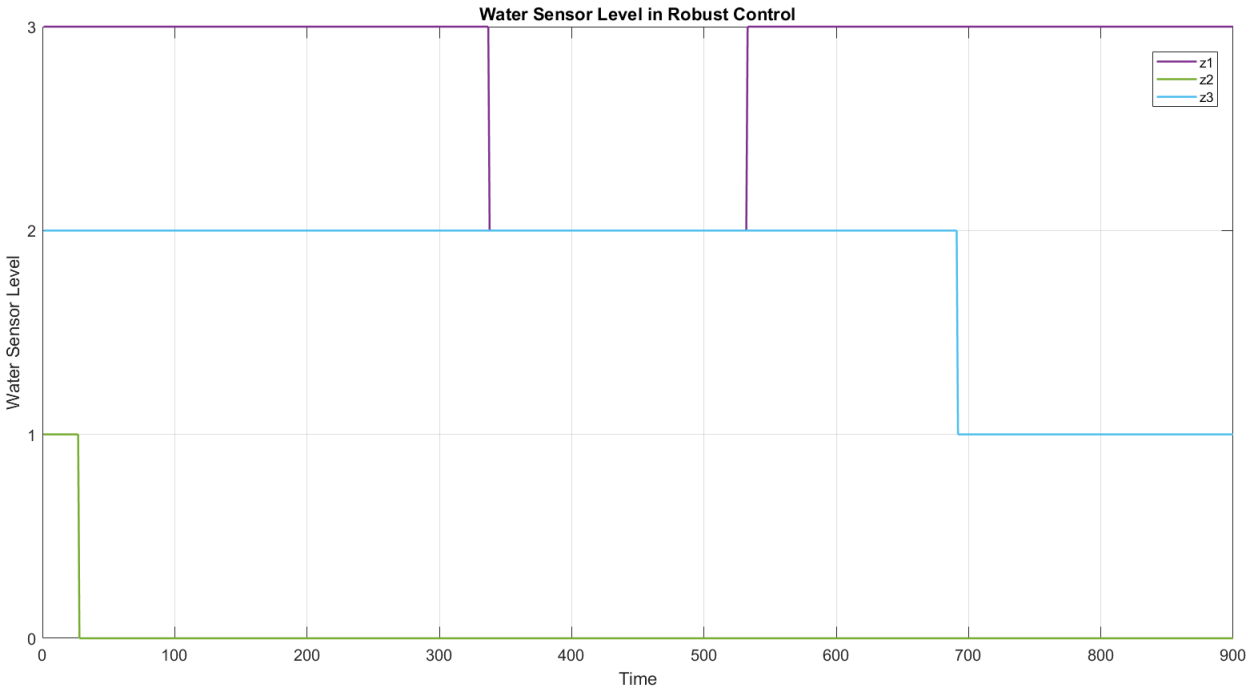


Figure 5.24: Water Levels on faulty real net

As expected, the results illustrated in Fig. 5.23 and Fig. 5.24 obtained with the Robust Control evidence that, as long as the unstable node did not fail, all three nodes tried to reach consensus at the overall average water level. However, when the third node failed, the algorithm "removed" it from the system in order that the remaining two nodes could reach consensus at the average of their water levels.

The same algorithm was tested for the functioning net where one of the nodes was detached and then reinserted to the net with the following results.

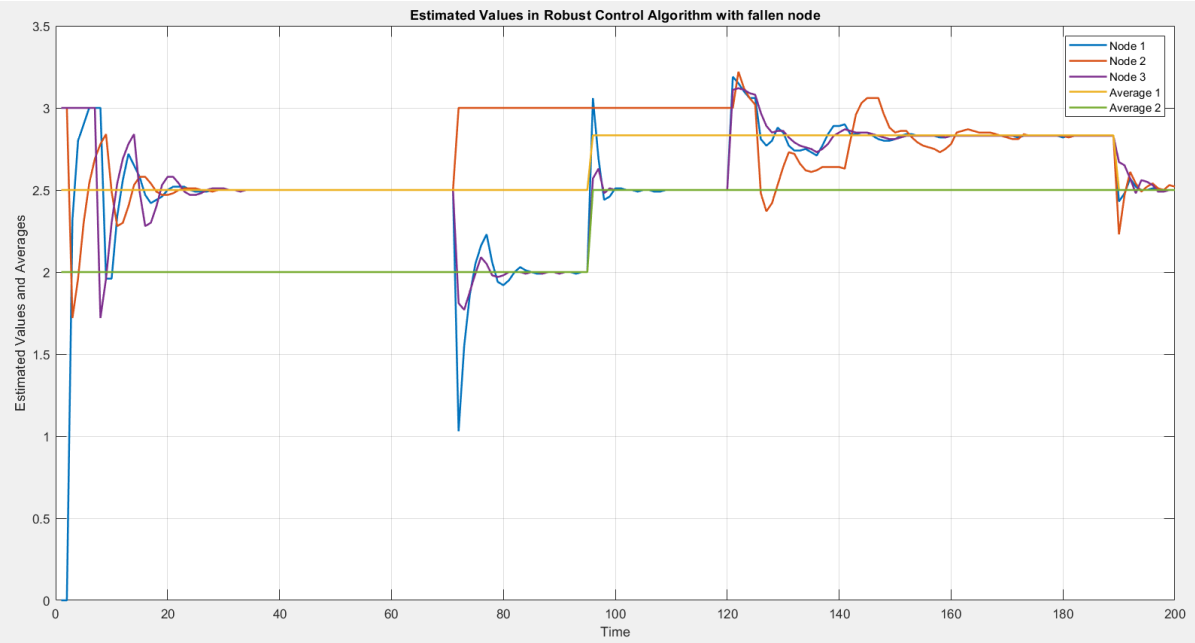


Figure 5.25: Robust Control on healthy real net

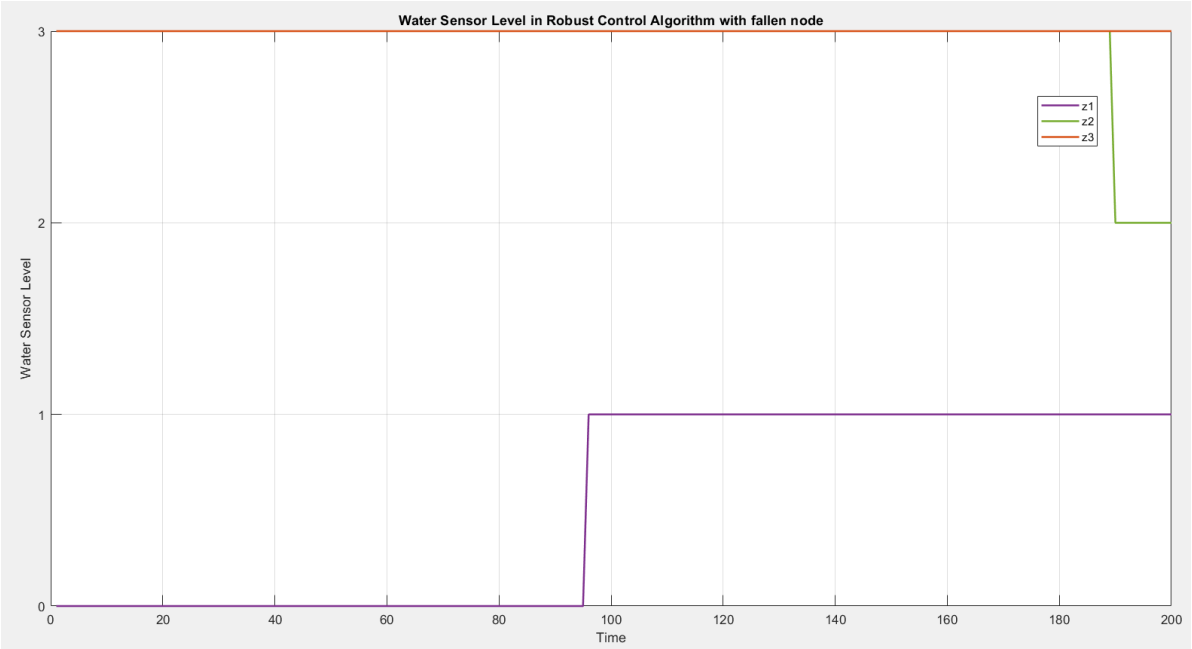


Figure 5.26: Water Levels on healthy real net

Fig. 5.25 and Fig. 5.26 show that, as it happened in the *MATLAB* simulation, beside an offset of 0.5 for both averages, there was a reached Consensus for the three nodes on their total average until the second node was detached from the net. From that time instant, until it was readmitted, the other two nodes went on their average while the third one went on its own value. In the last remaining part of the simulation the nodes reached again Consensus on their total average.

These results demonstrate that the Robust Control Algorithm is the most stable and effective solution among those proposed, providing a significant improvement in maintaining system reliability and consensus accuracy, even in the event of node failures or unreliable communication.

5.3 Distributed Algorithms: Advantages and Limitations

Each of the algorithms proposed in this chapter for handling node failures offers benefits and drawbacks.
The Dynamic Consensus algorithm is straightforward and easy to implement, providing a basic mechanism for nodes to reach Consensus. Its simplicity is its primary advantage, as it requires minimal computational resources and is relatively easy to maintain. However, this approach struggles significantly when nodes fail, as it lacks mechanisms to adapt to changes in the network and the only solution was to cut off the unstable node from the net.
The Weighted Dynamic Consensus algorithm introduces a more sophisticated approach by assigning different weights to nodes based on their reliability. Nonetheless, when a node completely fails, this algorithm still faces challenges, as the failing node, even if in a mitigated way, continues to affect the consensus.
The Robust Control Algorithm represents the most advanced solution, designed to dynamically adjust the influence of each node based on current network conditions. This approach significantly enhances the system's resilience, ensuring stable and accurate Consensus even in the presence of node failures or unreliable communication. However, this increased accountability comes at the cost of higher complexity and potentially greater computational and communication overhead.
In conclusion, these three distributed algorithms, compared to the centralized one, benefit from increased fault tolerance, as they do not rely on a single point of failure and they can better balance the computational load across multiple nodes, potentially improving overall performance and scalability. On the other hand, there is an increased complexity in implementation and maintenance and it may require more sophisticated error-handling mechanisms.

Chapter 6

Conclusion

This thesis explored the management of three water tanks using both centralized and decentralized control systems. The primary goal was to compare the practical efficiency and functionality of these two approaches in managing water distribution and levels.
Starting with components assembly and setup, as detailed in Chapters 2 and 3, we established a solid technical foundation.
Chapter 4 focused on the centralized control system, where the upper tank served as the master node. We analyzed its performance, noting both strengths and limitations.
In Chapter 5, we transitioned to distributed control, implementing the Dynamic Consensus, Weighted Dynamic Consensus, and Robust Control algorithms. Each algorithm was tested, revealing unique advantages and challenges. Compared to centralized control, distributed algorithms offered increased fault tolerance and better load balancing, despite their complexity and potential latency. The Robust Control algorithm was proved to be the most reliable, maintaining stability even with node failures.
Overall, while centralized control is simpler, distributed control provides greater robustness, making it ideal for dynamic environments. This project highlighted the importance of selecting the appropriate control strategy based on system requirements.
Reflecting on our work, we gained valuable insights into control systems, appreciating the complexities of both centralized and decentralized approaches. This project was a challenging yet rewarding experience, deepening our understanding of real-world control system network applications.