Traffic efficient control of a water irrigation system

Master thesis

Ivaylo Ganchev

TU Delft

Traffic efficient control of a water irrigation system

Master thesis

by

Ivaylo Ganchev

Student Name	Student Number
Ivaylo Ganchev	5135001

Instructor: Manuel Mazo Espinosa
Faculty: Faculty of Mechanical, Maritime and Materials Engineering (3mE), Delft

Faculty of Mechanical, Maritime and Materials Engineering (3mE)
Delft University of Technology

TU Delft

Abstract

Water scarcity is a persistent global issue that requires effective solutions. One increasingly popular approach to address the distribution of freshwater in different regions is the utilization of Water Irrigation Systems (WIS). Due to their vital role in ensuring people's well-being, it is crucial to design a reliable infrastructure that can maintain water levels in channels as close as possible to predefined reference points.

Extensive research papers have been dedicated to discussing various aspects of WIS control, modeling, and identification. Although simulations can yield valuable insights, it is essential to validate the results in real-life systems due to certain factors that cannot be precisely estimated, such as leakages along the channels. To address this need, a scaled-down version of a WIS at TU Delft has been developed, which effectively emulates the behavior of a real water irrigation system. Through continuous refinement and design enhancements, the simulator has proven to be a valuable tool.

Working with the scaled-down WIS has presented significant challenges, but a dedicated controller has been developed, yielding satisfactory results. The approach to controlling the WIS is simple yet effective, employing a decentralized controller that is applied to both the nonlinear simulator and the real setup. Remarkably, this approach successfully meets the predefined reference points, even in the face of applied disturbances to the system.

Contents

List of Figures

List of Tables

Preface

For the past year and a half, I have been working with the water testbed unit in the laboratory of TU Delft. Despite all the difficulties, both personal and otherwise, I can say that I have really enjoyed working on this project. The group meetings organized by Dr. Manuel Mazo Espinosa have been both exciting and enlightening. Seeing the progress of others really motivated me to continue with my work. Being able to share ideas with the whole group also makes you feel like you are in a real working environment.

I would like to thank Dr. Manuel Mazo Espinosa for his efforts, help, and patience during the whole period. He was always there with the right tips to solve a problem, both in my thesis and in my courses. Also, I would like to thank all my friends and family who supported me throughout my journey at TU Delft and cheered me up when needed.

My stay in Delft was not easy, but I learned a lot of stuff that will help me grow as a person and as an engineer, and I believe it will be worthy.

Ivaylo Ganchev
Delft, June 2023

Nomenclature

Abbreviations

Abbreviation	Definition
WIS	Water irrigation system
GUI	Graphical User Interface
WSL	Windows Subsystem for Linux
$PECT$	Periodic Event-Trigger Control
STC	Self-Trigger Control
ECT	Event-Trigger Control

Symbols

Symbol	Definition	Unit
Q	Flow	[m^3/s]
A	Cross-sectional area	[m^2]
R	Resistance	
C_d	Coefficient of discharge	
L	Gate width	[m]
g_o	Gate opening	[m]
g	Gravity constant	[m/s^2]

1

Introduction

This report represents my final work as a student at Delft University of Technology, where I am pursuing a master's degree in Systems and Control. Within this document, I provide a comprehensive explanation of the main theory, supported by various graphs, tables, figures, and examples. Its primary purpose is to familiarize the reader with my work and demonstrate the knowledge I have acquired during my studies here. To attain the desired results, I built upon the advancements made by Jacob [15] and Bas [4], who previously worked on the same project. Their contributions include the development of the water testbed unit in the TU Delft laboratory, which serves as the central focus of my research.

1.1. Significance of Water Irrigation Systems

The topic I have chosen to address holds immense significance on a global scale—the issue of water supply. As the world's population continues to grow [19], the demand for food intensifies, exerting additional pressure on agriculture. Moreover, the adverse effects of global warming are further impacting natural water reservoirs. Consequently, there is an urgent need to devise intelligent solutions, such as water irrigation systems, to effectively combat these challenges. Extensive research indicates that irrigation accounts for a substantial 70% of global water usage, particularly within the agricultural sector [5], [6]. Such water irrigation systems can be designed to cater to remote farms devoid of accessible drinking water sources. The scarcity of freshwater in certain regions also contributes to the proliferation of diseases, posing significant threats not only to the local populations but also potentially to individuals worldwide.

1.2. Key components of WIS

Normally, those systems Fig. 1.1 consist of several key components. There is one main dam or river which acts as a main supplier for the system. There are also the canals which are designed to distribute

the water to the areas where it is needed and finally, there are the gates which are used to regulate the level of water in the canals along the system.

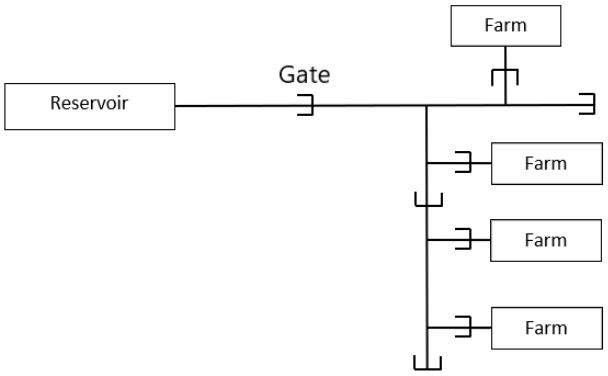


Figure 1.1: A top view of a water irrigation system designed to supply several farms with water for the crops and animals

1.3. My goal

My task in this master thesis will be to design a controller that will send signals to the gates according to a change in the water level in a pool in order to meet a reference point. The designed controller needs to handle disturbances that can occur in the system. The two major disturbances that act on a water irrigation system are rain and water draining from farmers. Both can have a severe impact on the water levels, especially if the canal in this area is not big enough. Other smaller disturbances that can influence the system are dry seasons and spillages, whose effects are not going to be considered very much in detail.

Working on this problem can be hard and time-consuming if it needs to be applied directly to a real system with hundreds of kilometers of canals. For that reason, a scaled-down version of WIS is being designed at TU Delft, along with a whole wireless network around it. It consists of a set of Fireflies boards, each providing local information for one gate and two sensors around it.

1.4. Document outline

The goal of this project is to design a reliable controller that can track references and reject disturbances acting on the system. To achieve this, both Python and Matlab will be used. First of all, a detailed explanation of the setup in the basement of the university will be provided. There will be provided an explanation of the type of gates that are used together with the sensors and actuators for measuring the water level and regulating it. Furthermore, this section will get the reader familiar with the flow-restricting panels that contribute to simulating a long channel while the length of one pool is no more than a meter and some information about the wireless architecture. Those panels were added after Bas Boot's experiments to provide more accurate dynamics.
The subsequent chapter will acquaint readers with the approach taken to design a non-linear simulator capable of accurately mimicking the behavior of the physical system. Equations governing water flow dynamics will prove instrumental in this regard. Furthermore, a comparative analysis between the actual system and the simulator will be presented in this chapter.
The next chapter will be dedicated to the real setup. An identification of the pools will be provided, together with graphs showing that the extracted models indeed match the dynamics of the real system. Afterward, one chapter will be dedicated to the controller design, together with simulations in Matlab and experiments on the real setup.
Finally, the thesis will conclude with a wrap-up of the results and future work propositions.

2

Testbed unit

Simulating a water irrigation system can be very hard due to the scale of the real-life one. It can go up to hundreds of kilometers of canals, and designing a control scheme for it from scratch can be time-consuming. For that reason, in the basement of TU Delft, there is a small-scale, three-pool testbed that can emulate the dynamics of the big one Fig 2.1. In this section, will be given an overview of the setup, which will include all the components and repairs done throughout the years.

Figure 2.1: Front and back view of the water irrigation system setup in TU Delft basement. It consists of the plant itself, a computer operation on Linux to control it, and the wireless network around it.

2.1. Testbed overview

As mentioned, the water testbed Fig. 2.1 consists of three pools that simulate irrigation canals in a real system. In Fig. 2.3 a top overview of the setup is given. Pools 1, 2, and 3 are denoted with A1, A2, and A3, respectively, and A0 denotes the pool acting as a supply one. There is the pump, which is the main source of water for the system Fig. 2.2. Green circles denote the pressure sensors; red bars are the gates; and black circles are the water inlets. Finally, there is an overflow reservoir where the water drains from the system. This is in order to preserve the system and not flood the basement. In each of the three pools, there is one other valve Fig. 2.4 that simulates rain, which will be later used to add disturbances.

Figure 2.2: Picture of pool 0 with the main source of water for the whole setup.

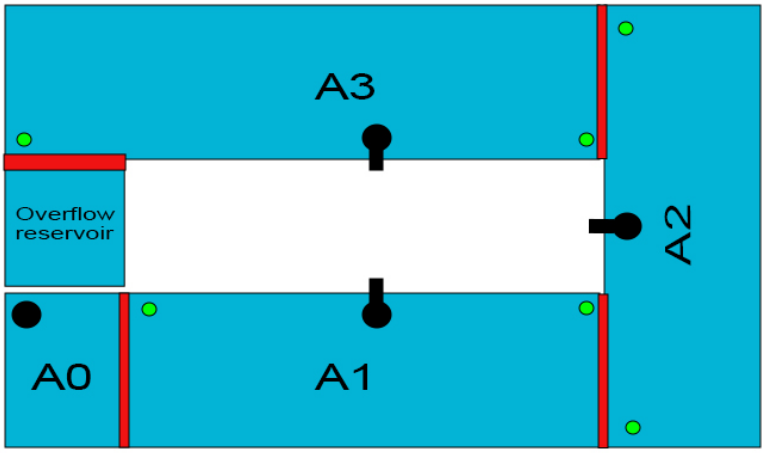


Figure 2.3: Scheme of the 3 pool system setup

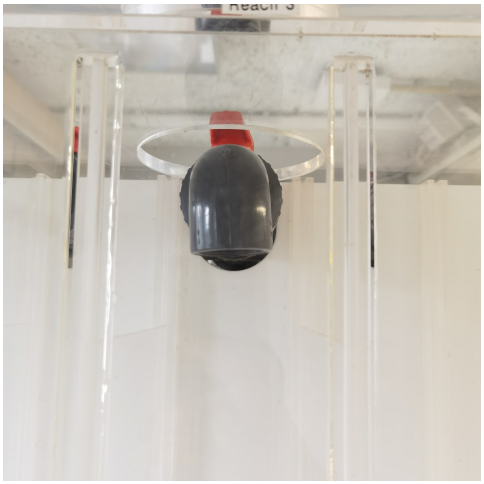


Figure 2.4: Valves simulating rain

2.2. Flow restricting panels

Because the pools are relatively small, panels with orifices are installed, which serve the purpose of slowing down the dynamic processes. In terms of fluid dynamics, they can be described as fluid resistors. Each panel has 35 orifices, which are situated at the bottom end in five lines. They are all of the same diameter and the same distance apart from each other, which helps in the modeling part when applying Bernulli's equations. The first and third pools are divided into 7 compartments, and the second pool is divided into 3.

2.3. Gates

As seen in Fig. 2.3 there are 4 gates in the whole setup. The first three are undershoot Fig. 2.5a gates, and the last connecting pool 3 with the overflow reservoir is an overshoot one Fig. 2.5b. They are controlled by servos. The one used for the current setup is the Hitec HS785HB Fig. 2.6 which has a rotational range of 1260 degrees and pulse cycle of 20ms [11]. In order to control a gate, the servo is connected to a pole via a gear that moves the gate up and down according to the commands. Because

the last gate is an overshoot one, the pole needs to be shortened, and thus the servo range needs to be limited in order to prevent it from being damaged [4].

(a) Undershoot gate **(b)** Overshoot gate

Figure 2.5: Different types of gates in the WIS

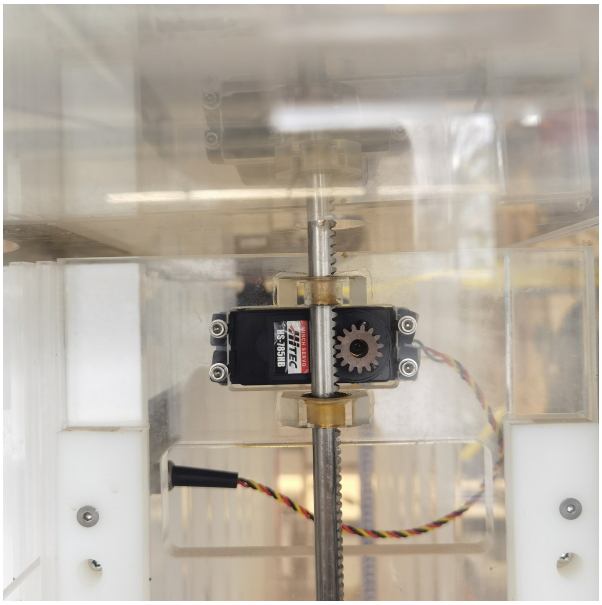


Figure 2.6: Servo Hitec HS-785HB mounted to control the gates positions according to the commands given manually or by controller

2.4. Sensors

In order to measure the water height, pressure sensors from the brand Keller Fig. 2.7 are used [12] which are placed under the pools as indicated in Fig. 2.3. For achieving faster readings from the sensors, an external AD converter (ADSS15) is used, made on a custom PCB called Fireboard [15]. The data from the sensor gives pressure as output, so it needs to be converted. In order to achieve the

most accurate results, each sensor needs to be calibrated separately. According to [4] the calibration can be done via two points of water height using the formula:

$$h_i = a_i * value_i + b_i \tag{2.1}$$

As the values for a and b in [4] were last calculated in 2021, they need to be updated now in order to produce more accurate results. The calibration points chosen are 0cm and 29.5 cm, and the calibration data are presented in Tab. 2.1. In order to measure the actual water level in the pools, measuring tapes were used, which can be seen in Fig. 2.5b. They are situated at the beginning and end of each pool for better precision in readings.

Sensor	a	b
1	0.0188	-188.6845
2	0.0054	-68.5464
3	0.0181	-180.7098
4	0.0186	-184.7429
5	0.0189	-185.5793
6	0.0053	-57.6036
7	0.0054	-59.4792

Table 2.1: Calibration parameters

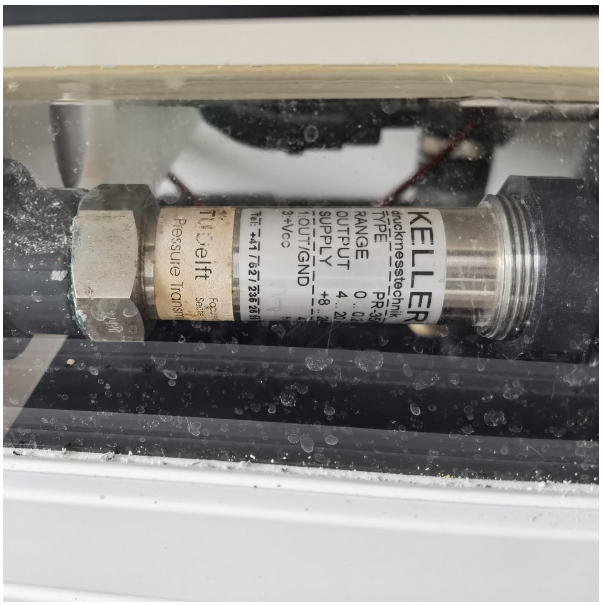


Figure 2.7: Keller pressure sensor series 35X

While collecting data from the sensors, some high-frequency peaks occur in the measurements, as seen in 2.8. These peaks can be annoying when reading the data, so a moving median filter is applied to all the readouts from the sensors. The window length that provides the smoothest results is 6 samples.

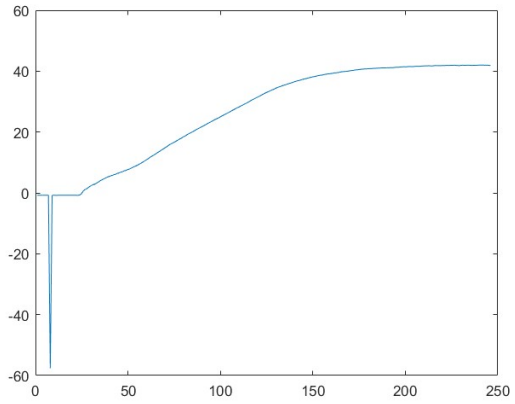


Figure 2.8: Sensor measurement with high-frequency peak

2.5. Wireless comunication

In order to be able to control the system wirelessly, a communication network is designed, which is explained in depth in [4] together with the developments in [15]. Here, just the fireflies will be discussed, which are responsible for reading data from the sensors and sending commands to the gates.

2.5.1. Network infrastructure

For reading the measurements from the sensors and sending commands to the gates, a set of eight fireflies is used. They were designed and implemented in the testbed setup by Jacob Jan Lont and the TU Delft technicians a couple of years ago, and all the information can be found in his thesis, [15]. Further, the communication between all the devices was developed by Bas Boot. Each firefly is considered a node and serves its own purpose, which is explained in Table 2.2.

Firefly name	Node ID	IEEE MAC Address	Role
FF1	201	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x51, 0x04	Gate 1, sensors 1 and 2
FF2	202	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x51, 0x6c	Gate 2, sensors 3 and 4
FF3	203	0x00, 0x12, 0x4B, 0x00, 0x19, 0x32, 0xe6, 0x91	Gate 3, sensors 5 and 6
FF4	204	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x52, 0x05	Gate 4, sensor 7
FF5	205	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x51, 0xa8	Controller
FF6	206	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x51, 0xd7	Relay
FF7	207	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x51, 0xcd	Relay
FF8	208	0x00, 0x12, 0x4B, 0x00, 0x19, 0x4A, 0x51, 0xb5	Relay

Table 2.2: Firefly configuration in the water tesbed setup

When the controller is in manual mode, simple commands can be sent through the serial port that will open or close the gates depending on what experiments are needed. For example, we can refer to gates 1 to 4 with nodes 201 to 204, and the servo commands range from 0 to 255, which means fully closed and open, respectively. So the command 201 255 will fully open the first gate, which is connecting the supply pool to the first one (see Fig 2.3). Node 205 is also responsible for switching between manual and automatic mode with the commands 205 0 or 205 1, respectively. The automatic mode considers the controller implementation of Bas with the old models of the pools.
The communication between the fireflies is done via UART protocol which is configured to its maximum speed - 460800 Bd. This is done in order to minimize delays in communication.

2.6. Upgrades throughout the period

The original setup was built in 2001[18]. Throughout the years, it has been revised and upgraded according to the needs of the experiments. Furthermore, due to the good maintenance over the 22-year period, the testbed is still in perfect condition. According to [15] new water pumps and servo motors were needed, and all the connections were inspected and some of them re-glued. In addition, Firefly microcontrollers were added to establish wireless connection with the servos. The latest updates are the automated valves Fig. 2.9. This allows for repetitive scenarios in experiments. Before that, all of them were controlled manually, which means that one experiment is very hard to be repeated multiple times. Finally, some more water is added in the supply tank since, with time some of it has evaporated.

Figure 2.9: Automated valves

2.7. Accessibility and safety

Primarily, in order to work with the setup, there is a computer in the basement that is connected directly to the model and is using Linus OS, which is preferred for serial communications. A Graphical User Interface (GUI) Fig. 2.10 is developed by the technicians in the lab. It can control the vales and the water pumps, which are enabled by turning on the ”Main” button, and also the fireflies, which can be turned on and off. As said in the previous section, the valves are now automatic, and their position can range from 0 to 90 degrees and can be set from the slider. Bas and Jacob have developed the wireless communication throughout their thesis work, which is used to control the gates through Python. [4], [15].

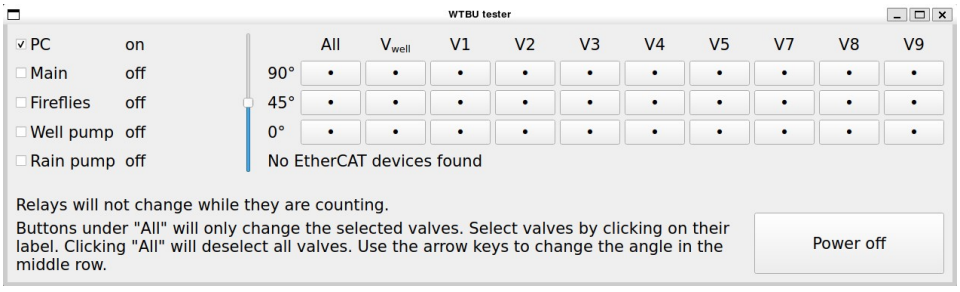


Figure 2.10: GUI for controlling the valves and the pumps

Further to that, there is a server that provides an opportunity to work remotely. This is very beneficial since one can have access to the whole setup without being physically there. As a drawback,

sometimes a hard reset of the network needs to be done, which is possible only in the lab, and a Linux emulator is needed for non-Linux OS. In terms of safety, the whole system has a dedicated place with an elevated fence that prevents spillages in the lab. Furthermore, there is a limited supply of water for the system, and is programmed to stop after some amount of time of constant work or if there is not enough water in the tank, which also protects the pumps. No matter all the safety concerns, when the WIS is accessed remotely, there are cameras with which all the work can be observed.

3

Simulator

This chapter presents a comprehensive simulator designed to accurately replicate the behavior of the real system. The simulator incorporates equations that enable the calculation of flow in various scenarios, including overflow, undershoot gates, overshoot gates, and flow-restrictive lintels with orifices. Additionally, the simulator accounts for external factors such as rain and water drainage, which are inherent to the real system. However, it is important to note that minor disturbances such as some leakages have not been included in the current simulation. Strategies to address these issues will be explored in subsequent sections of this chapter.

3.1. Mathematical implementation of the flows

Building such simulators can be crucial for real systems of the scale of the water irrigation ones. In order to represent long channels spreading hundreds of kilometers, the St. Venant equations can be used which are proven by several surveys to give decent results in terms of modeling [17], [16]. Apart from those sources, these equations are explained extensively in my literature survey supporting this thesis. In essence, they are based on the conservation of mass and momentum equations. These are non-linear equations used to describe non-uniform flow. However, solving these equations can be quite challenging, and for that reason, a different approach is chosen when we build the simulator for the testbed at the university. This method relies only on the mathematical representation of the flows in different scenarios.

3.1.1. Lintels

As mentioned in 2 after the issues with the fast dynamics in Bas Boot's thesis [4] panels to restrict the flow in each pool are implemented. They significantly slow down the dynamics and manage to give a better representation of what the water levels will be in a real system with a length of several kilometers.

In simple words, those panels can be looked as flow resistors [7]. Each panel has 35 orifices situated at the bottom in 5 lines of 7. All of them are of equal radius and equally spread apart in order to satisfy some criteria of Bernoulli's law. In order to model the resistance of one lintel, the following formula is used:

$$R = \frac{1}{C_d \cdot (35 \cdot A) \cdot \sqrt{\frac{2}{\rho}}} \tag{3.1}$$

In Eq. 3.1 C_d is a coefficient of discharge, which represents the ratio between the actual and ideal discharge and is an experimentally found value. It should be at most 1 since otherwise, this will mean that the actual flow is bigger than the ideal. A represents the area of a single orifice and needs to be multiplied by 35 because of the number of holes. Finally, ρ is the water density which is 1000 $[kg/m^3]$. Having this in mind, the flow through one resistant panel is given by the following equation:

$$Q = \frac{g \cdot \rho}{R} \cdot (y_{us} - y_{ds}) \tag{3.2}$$

3.1.2. Gates

Due to the fact that there are two different types of gates, the equation for each one will slightly vary. As said, the first three gates of the setup are undershoot, which means that the water will flow under them when they are open, as shown in Fig. 3.1.

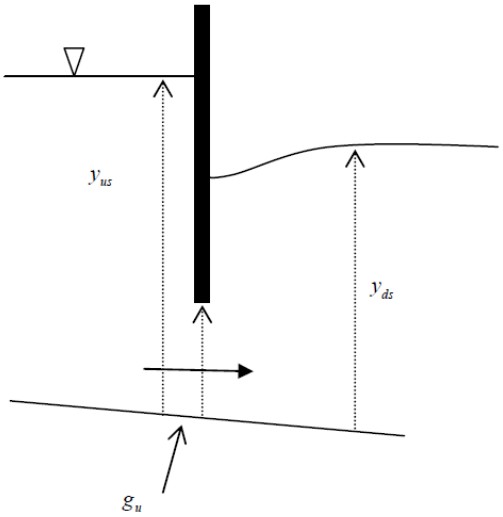


Figure 3.1: Undershoot gate flow

The flow for this will depend on the difference between the upstream and downstream water levels and can be expressed by the following equation:

$$Q = C_d \cdot L \cdot go \cdot (2 \cdot g \cdot (y_{us} - y_{ds})) \tag{3.3}$$

In Eq. 3.3 C_d is again the discharge coefficient, L is the gate width, go is the gate opening, and g is the gravity constant.

The second type of gate in the system is the overshoot Fig. 3.2. It can be explored as a sill with variable height, which will depend on the position of the gate.

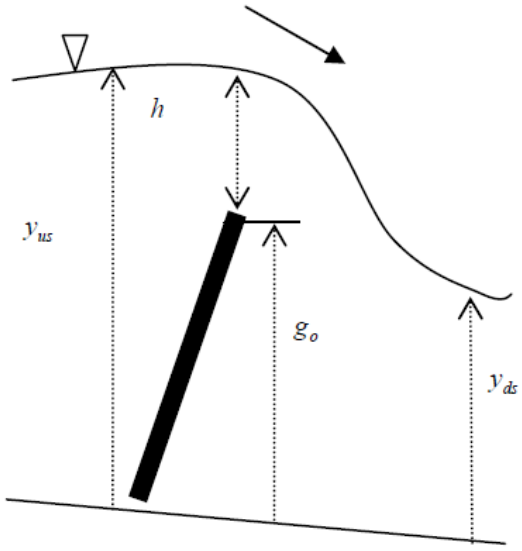


Figure 3.2: Overshoot gate flow

In this type, it is assumed that the flow is traveling from upstream to downstream, and for that reason, it depends only on the upstream water level. Its mathematical representation is given by the following equation:

$$Q = C_d \cdot L \cdot \sqrt{2 \cdot g} \cdot (y_{us} - sill)^{3/2}, \tag{3.4}$$

where $sill$ represents the sill height when the gate is fully open plus the gate position.

3.1.3. Overflows

In the real system, there are several overflows that prevent the water from spilling out of the setup. There is one in the supply pool, which is connected directly to the overflow reservoir, and two in each of the pools in the first and last compartments, apart from the last pool since there the overshoot gate acts as overflow. Those need to be implemented in the simulator as well since they can play a significant role in getting the same results when doing experiments. The most important of them is the one in the supply pool since the overflow level is always reached in the real model experiments. Here, the same approach as the one used for the overshoot gate can be applied and it can be modeled as a sill of a certain height, which in our case is 0.60m. Several different approaches are used, and all of them yield very close results. One of the formulations is the following:

$$Q = C \cdot L \cdot H^{3/2}, \tag{3.5}$$

where C is the coefficient of discharge, L is the lenght of the weir, and $H = h - 0.6$ is the water height above the sill.

3.2. Simulations

In this section will be presented results from the simulator with fully open gates. References on the implementation in Simulink can be found in Appendix A together with the codes in Appendix B used to simulate it.

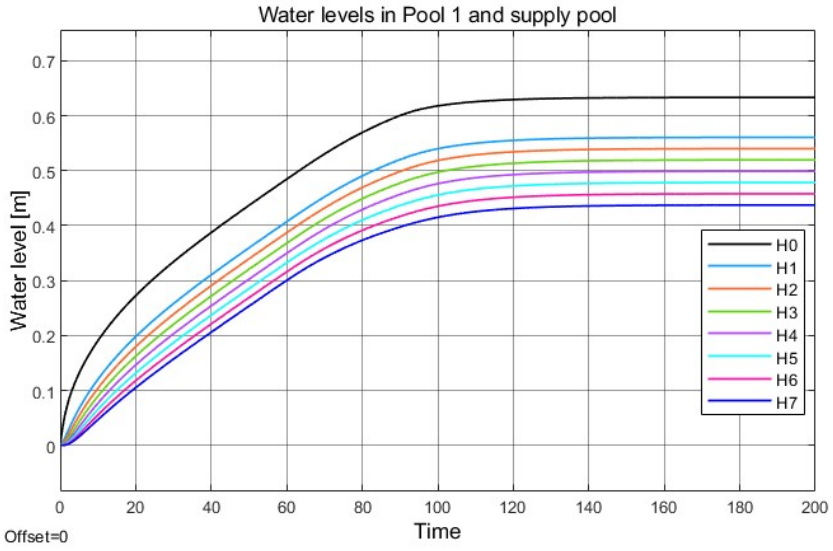


Figure 3.3: Pool 1 and pool 0 water levels for fully open gates

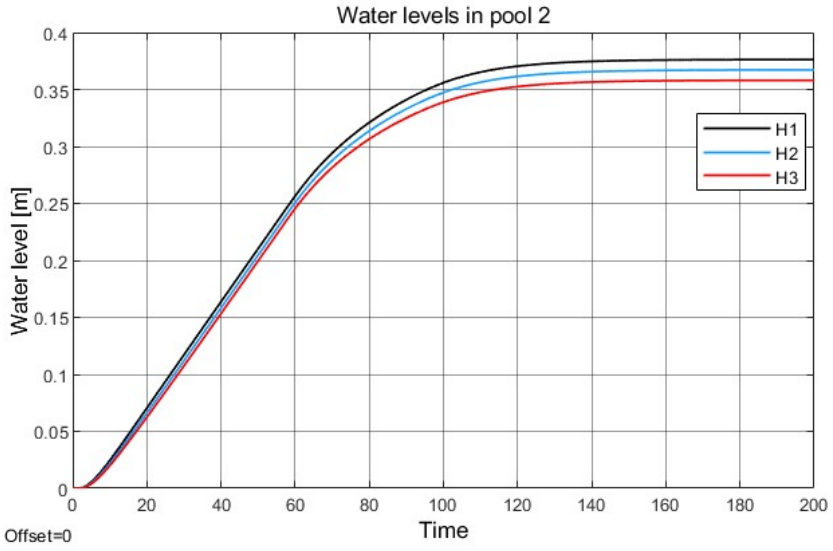


Figure 3.4: Pool 2 water levels for fully open gates

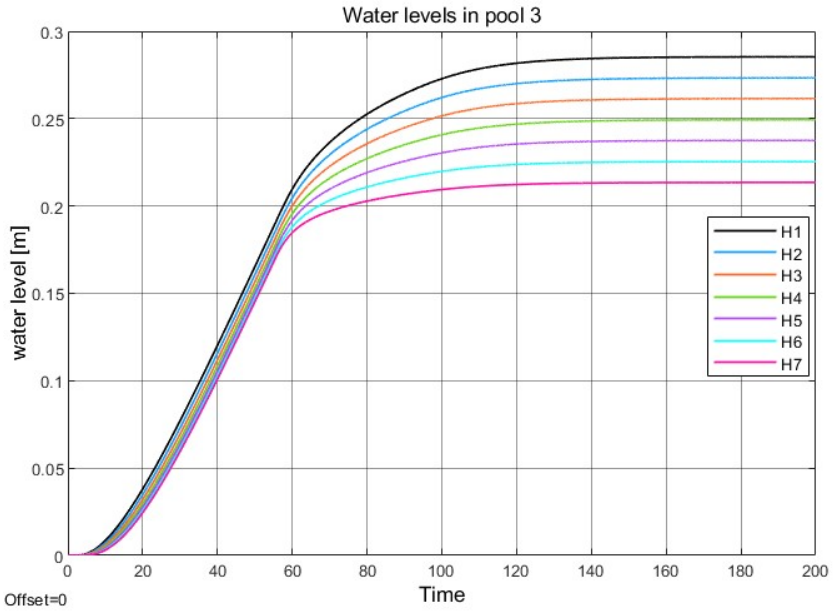


Figure 3.5: Pool 3 water level for fully open gates

Since we are trying to represent a real water irrigation system, which is supposed to have very long channels, it cannot be expected that the water level at the beginning and at the end will be the same. As seen from Figs. 3.3, 3.4 and 3.5 this behavior is captured by the simulator thanks to the flow-restrictive panels in each pool. Later, it will be shown that those levels indeed match those from the real setup in the university within 1-2 centimeters. From all the graphs, it can be seen that only pool 0 reaches the overflow, which is also the case in the real system that has been represented.

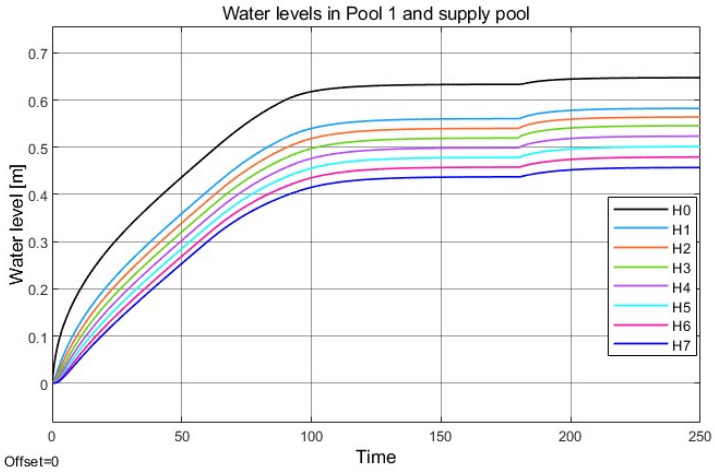


Figure 3.6: Pool 1 and pool 0 water levels for fully open gates with rain disturbance in the first pool

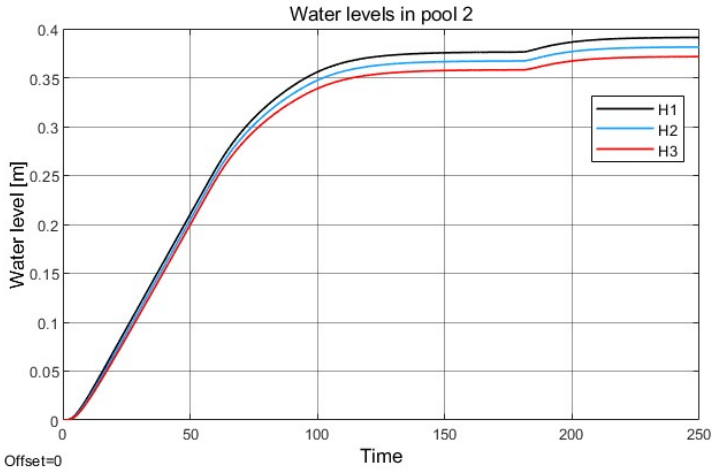


Figure 3.7: Pool 2 water levels for fully open gates with rain disturbance in the first pool

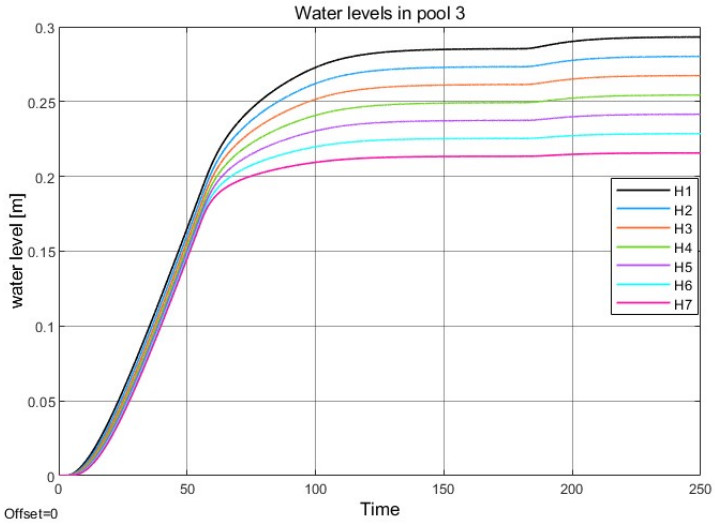


Figure 3.8: Pool 3 water level for fully open gates with rain disturbance in the first pool

In this set of graphs, the effect of a rain disturbance applied in the first pool at 180 s is shown. It can be seen that its effect slowly decreases throughout the length of the channels, which is an expected behavior.

3.2.1. Comparison with the real model

In this subsection, we will present some experiments done with the real setup and the simulator to compare how representative it can be to study the behavior of a real system.

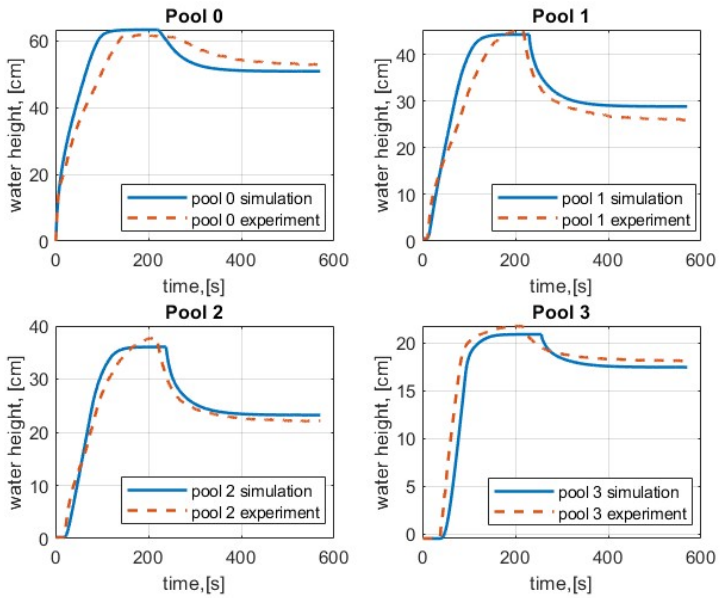


Figure 3.9: Comparison of the real system and the simulator with draining disturbance applied after the water levels settle, solid blue line - simulator, dashed red line real system

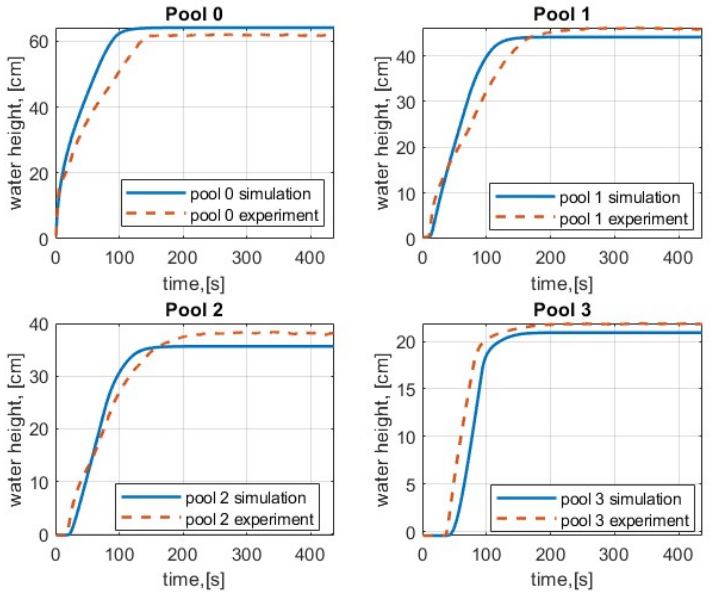


Figure 3.10: Comparison of the real system and the simulator with fully open gates and no disturbances, solid blue line - simulator, dashed red line real system

From both graphs, it can be seen that the dynamics of the simulator do not quite match the ones from the real system and turn out to bea bit faster, but it gives a good enough representation of the behavior of the system as well as the final value of the water height. One possibility for this mismatch is certainly unmodeled dynamics, which may be caused by leakages in the real system. Another reason can be inaccuracies in the mathematical representation of the flows that are included in the model.

4

Identification

This chapter delves into the essential aspect of identification in control design. Identification plays a pivotal role in simplifying the model extraction process, thereby facilitating the tuning of a controller. Here, we explore the methods employed for identification, including the techniques utilized and the specific settings employed in the real setup to yield meaningful and reliable results. By thoroughly examining the identification process, this chapter aims to provide valuable insights into the foundation upon which effective control strategies can be built.

4.1. Identification procedure

The main purpose of this master thesis is to design a controller that has to maintain a certain water level in each of the tanks. Having a complicated nonlinear model with all the flow dynamics and the real system can end up being very challenging to tune a controller. For that reason, an identification procedure needs to be done. As the simulator is tuned to match the real system as much as possible, both can be used to collect data. In our case, the identification is done using the real system. With the sensors calibrated correctly, each pool is identified using the same procedure.

First of all, the upstream gate is opened to a certain position, and the downstream gate is fully closed. As the water level in the identified pool settles, the delay and the time constant are extracted in Matlab using the graph of the step response. Since our system is a string of pools, the delay for the second and third pools needs to be measured from the time the first compartment of each pool starts filling up. When all the measurements are taken, a first-order linear model is extracted for each pool, and it has the following form:

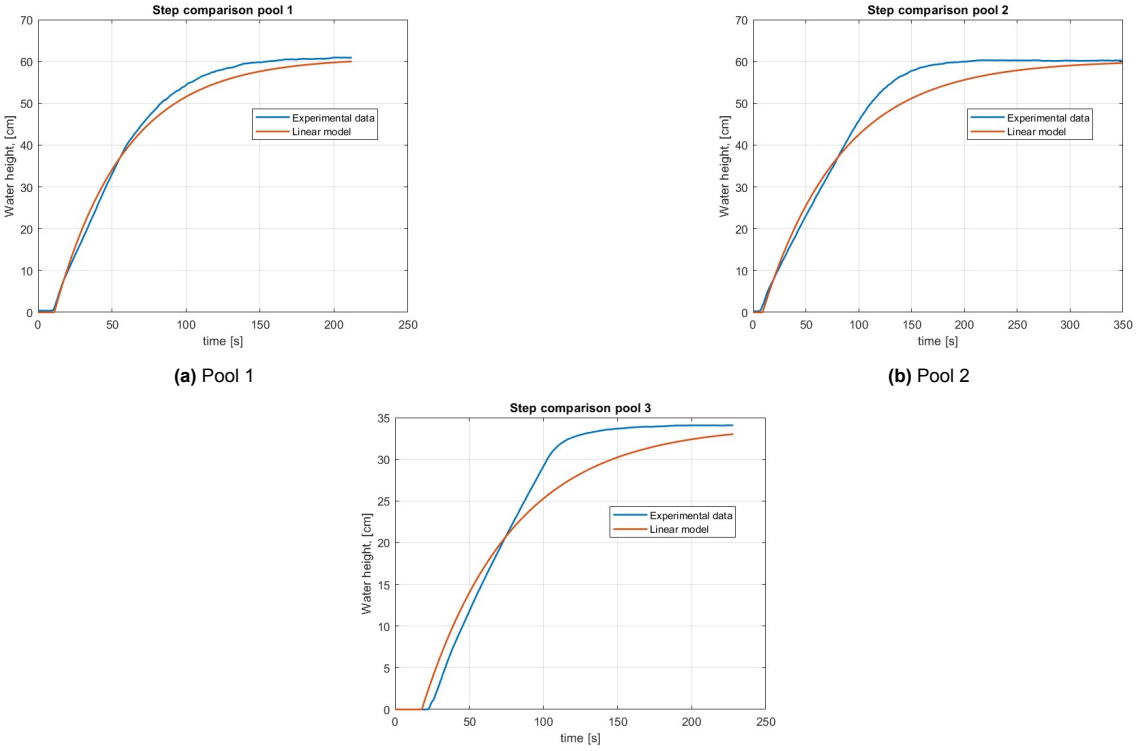
$$G(s) = e^{-ds}\frac{K}{Ts+1}, \tag{4.1}$$

where d is the delay, K is the gain, and T is the time constant, which shows the time needed for the system to reach 60% of its steady state. The parameters for each pool are given in the following table:

Pool	delay, [s]	Gain	Time constant, [s]
0	0	66.58	9
1	11	60.9	47
2	9	60.24	74
3	16	34.08	62

Table 4.1: Identified model parameters

A comparison of the linear models and the real system experiments is shown in the graphs below.



(a) Pool 1

(b) Pool 2

(c) Pool 3

Figure 4.1: Comparison of the real pools simulation and the linear models from the Identification

As it can be seen, although the models used are very simple, of first order, they can still represent the real system behavior quite well.

5

Controller design

This chapter unveils the results obtained from the controller tuning procedure for the system. The control approach employed is designed to be straightforward yet effective, with the primary goal of achieving satisfactory outcomes. It is important to note that this is the initial exploration of the model incorporating water-restrictive panels, which adds a greater level of realism to the system. Subsequent sections will delve into more advanced methods that can be considered for future work and design enhancements.

5.1. Controller type

In the literature on this topic, there are many types of controllers that have been used. From PID-type controllers in [17] with or without a feedforward term, distributed controllers in [14] and centralized and decentralized controllers in [5]. Also in [4] is provided an explanation about an event-triggered controller. In our case, a simple PID-type controller for each gate is designed to control its position with feedback from the downstream sensor in the current pool.

5.2. Anti-windup compensation

In any real system, there are certain physical limitations that need to be taken into account when it is modeled and controlled. In our case, that is the position of the gates. They cannot be less closed than their fully closed position or more open than their fully open position. Although in simulations this can be neglected because we can basically apply big signals randomly to the plant, this saturation effect needs to be taken seriously if we want to mimic the real system. If we apply just a normal structuration block, when the control action hits a limit, the close loop is basically ”broken” and the system starts performing as an open loop [2],[3]. This will lead to the integrator integrating an error that is not correct anymore.

This can lead to slower transients or even cause instability. One way to tackle this phenomenon is to apply an anti-windup compensator. The schematic solution to this issue is given in Fig. 5.1.

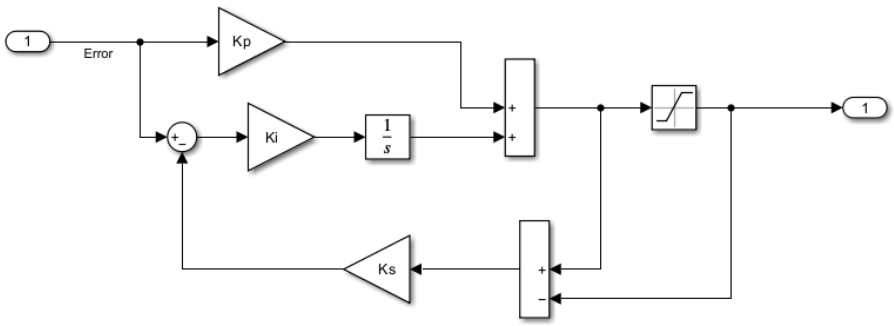


Figure 5.1: anti-windup schematic solution

The idea of this method is to form an inner feedback loop in which, while the controller output is saturating, an error signal, that is, the difference between the actual output and the saturated one, has been integrated. As soon as the control signal goes below the point of saturation, the normal feedback loop starts working again. In Fig. 5.1 $K_s = 1/T_s$ determines how quickly the controller output resets. As a rule of thumb, T_s can be used the same as the integrating constant.

5.3. Controller tuning

In order to tune the controller, the Zigler-Nichols method is used. It is an approach based on putting the system into a close-to-unstable state [8]. First of all, only a P controller is applied, and it is increased until the system starts oscillating around a desired setpoint. As soon as constant oscillations are reached Fig. 5.2 the value of the P controller is considered as K_u critical gain. Then the period of the oscillations T_u is measured, and Tab.5.1 is used to calculate the gains for each part of the PID controller.

Controller	K_p	K_i	K_d
P	$0.5K_u$	0	0
PI	$0.45K_u$	$1.2/T_u$	0
PID	$0.6K_u$	$2/T_u$	$0.125T_u$

Table 5.1: Gain values from the Zigler-Nichols procedure

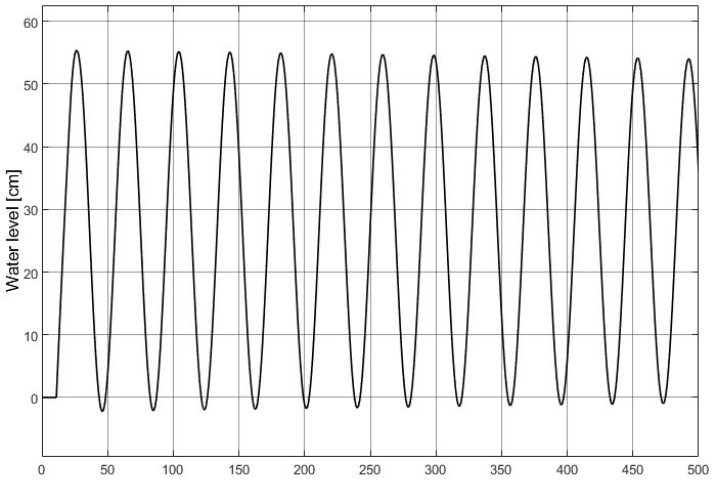


Figure 5.2: Constant oscillations from the Zigler-Nichols tuning

Since the stability margins of the Zigler-Nichols tuning procedure are small [8] a bit of fine-tuning of the gains is required in order to achieve smoother processes, especially with the proportional part. Here it can be seen why simulations are crucial when designing the controller. If this behavior needed to be applied to the real system, it could damage the servos, which will need to constantly adjust their position. Also, since the gates cannot immediately go from one position to another, the process would have taken a lot of time.

5.4. Simulation results

When tuning the controllers for each gate, the Zigler-Nichols procedure is done for each pool separately to extract the correct settings. Then the controller settings are put together into a simpler system consisting only of the model of the gates and the linear models of the pools in order to check if the gains need adjustments. Finally, when the results are satisfying, the controller is transferred to the model with nonlinear dynamics.

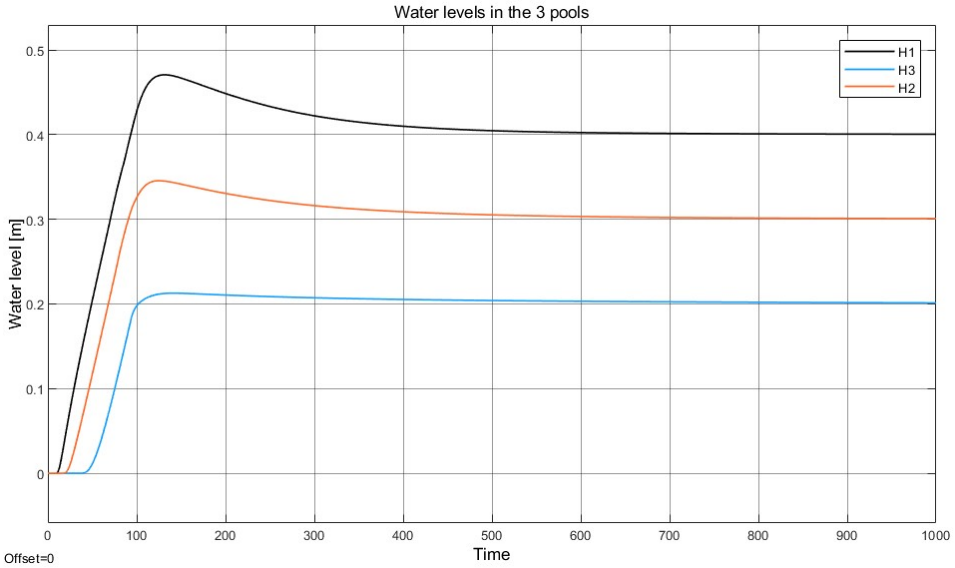
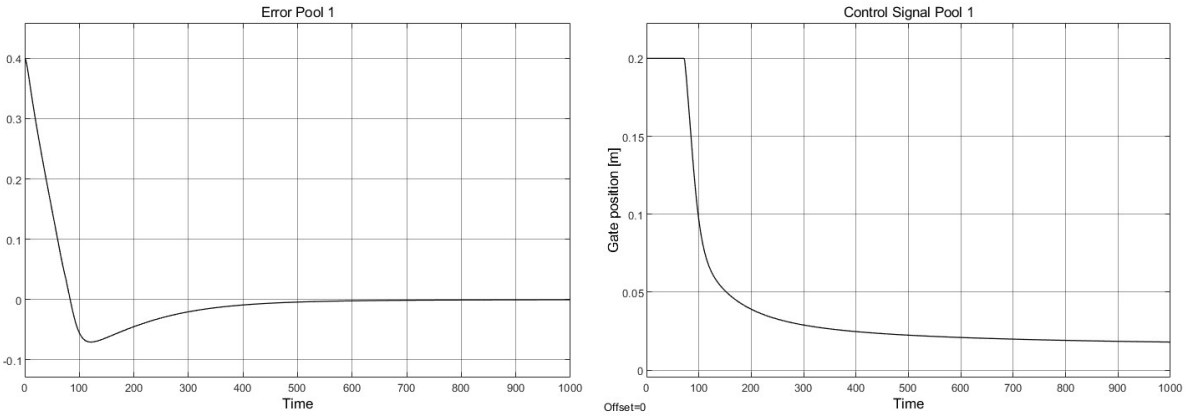
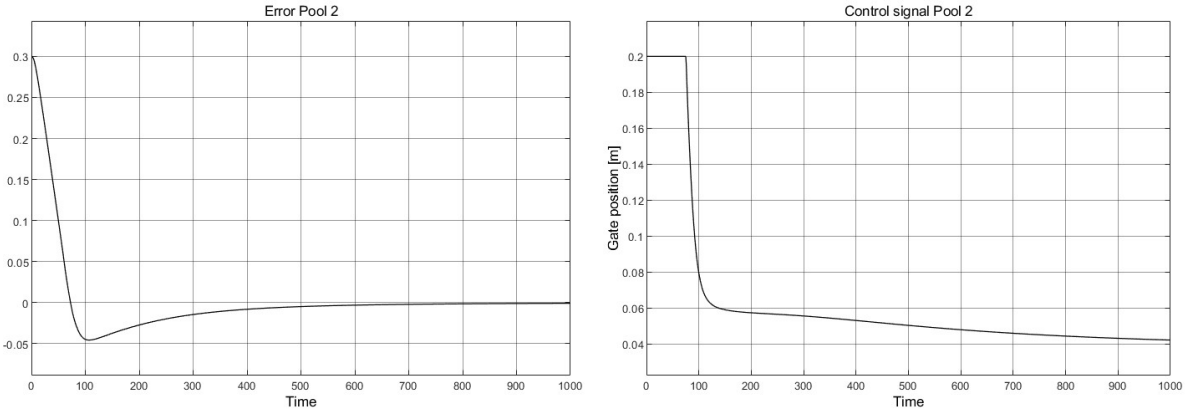


Figure 5.3: Water levels of the pools with set points of 0.4, 0.3, and 0.2 meters respectively for pools 1 2, and 3



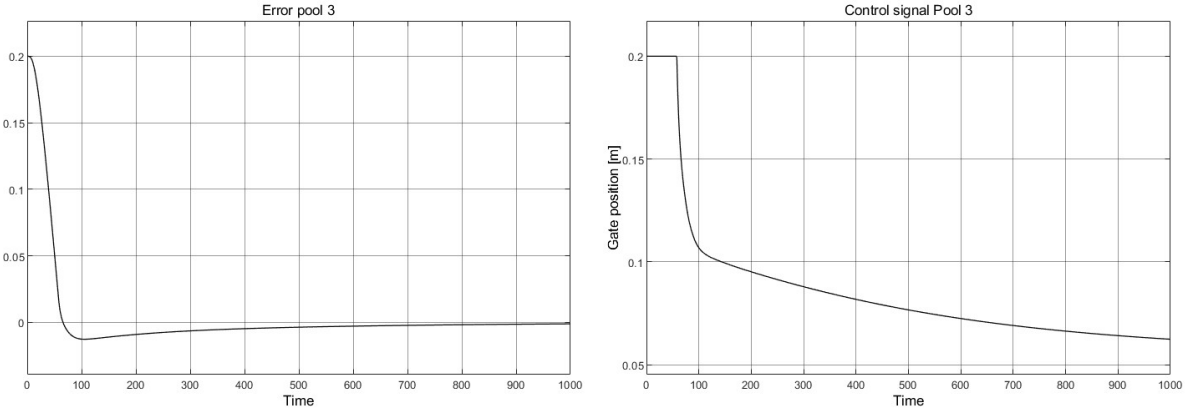
(a) Error signal **(b)** Control signal

Figure 5.4: Pool 1 error and control signals for a set point of 0.4 meters water level



(a) Error signal **(b)** Control signal

Figure 5.5: Pool 2 error and control signals for a set point of 0.3 meters water level



(a) Error signal **(b)** Control signal

Figure 5.6: Pool 3 error and control signals for a set point of 0.2 meters water level

As it can be seen from Fig.5.3 although the results are a bit slow, around 350 seconds, controllers manage to track the given setpoints. We can observe that in Fig.5.3 there is a 12% overshoot and less than 10% in the other two pools. The controllers are tuned for having smaller overshoots which

compromises a bit the speed of the process. Furthermore, in the control signal graphs can be observed that the curves are smooth with no sudden fluctuations which can harm the servos in the real setup.

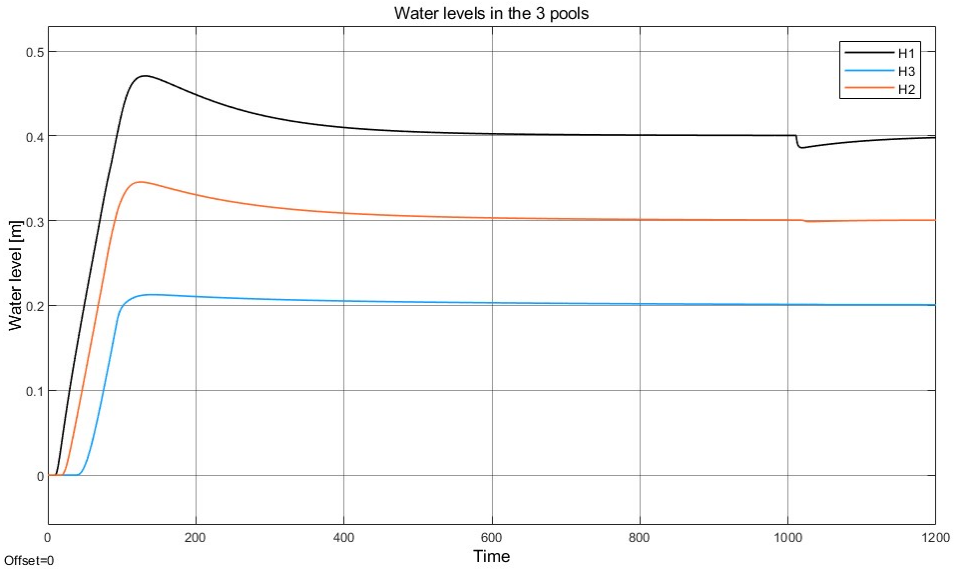


Figure 5.7: Water levels of the pools with set points of 0.4, 0.3, and 0.2 meters respectively for pools 1 2, and 3 and an off-take disturbance at 1000th second

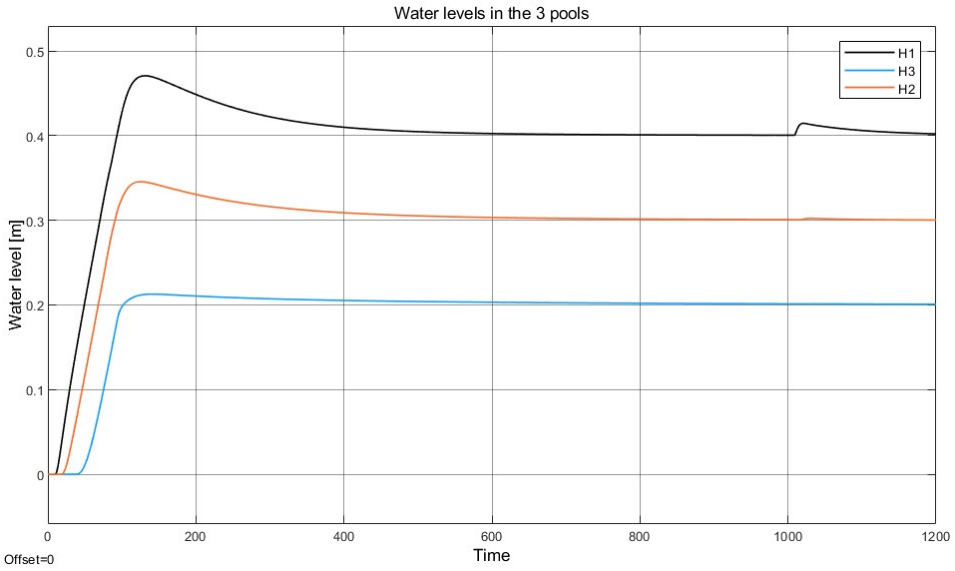


Figure 5.8: Water levels of the pools with set points of 0.4, 0.3, and 0.2 meters respectively for pools 1 2, and 3 and an rain disturbance at 1000th second

Another experiment that is performed in simulations is the off-take and rain disturbances, which are the most common disturbances in those kinds of systems. It can be seen from Fig.5.7 which is the off-take disturbance, and Fig.5.8- the rain disturbance, that the controller manages to handle them quite well. Both simulations are performed when the disturbance is acting in the first pool, and we can observe that the disturbances are hardly visible in the second pool and basically not affecting the third one.

5.5. Experiments

When designing the controller for the real system, Python was used. The settings from the nonlinear simulator have been used, and only the proportional part has been tuned to have faster transients. The controller is set to provide gate adjustments to the gates every two seconds. Again, as in the simulation, each gate is given as feedback the downstream sensor measurement in order to have the same setup as the one simulated in the previous section. The controller is tested with the same setpoints as the ones in the simulator, and the results are given in Fig 5.8

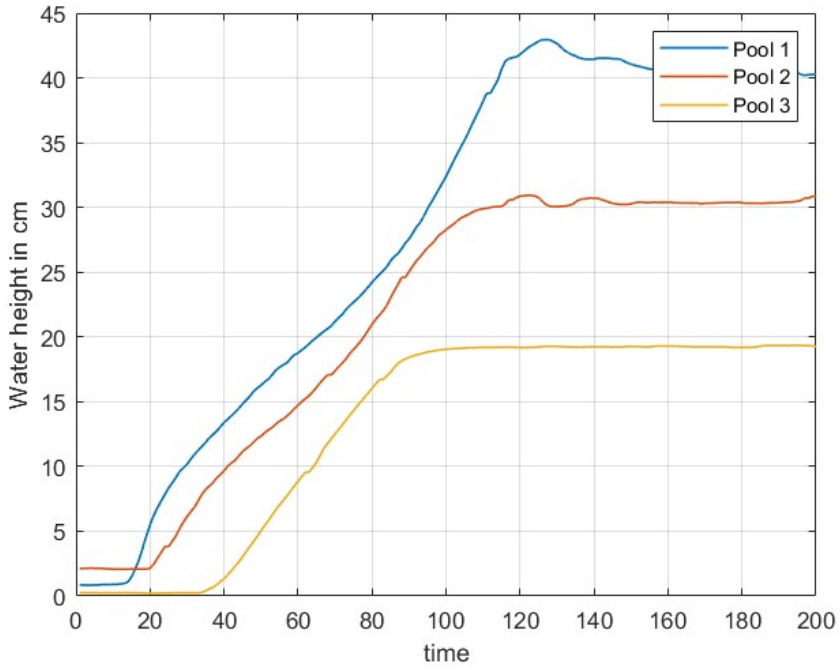


Figure 5.9: Water levels of the pools with set points of 40, 30, and 20 centimeters respectively for pools 1, 2, and 3

All the results end up within the 5% of the setpoint value after the 150th second, which can be considered a steady state. There is a slight overshoot in pool 1, and the response of the real system is faster than the one from the simulations. This can again be due to missing dynamics in the simulator.

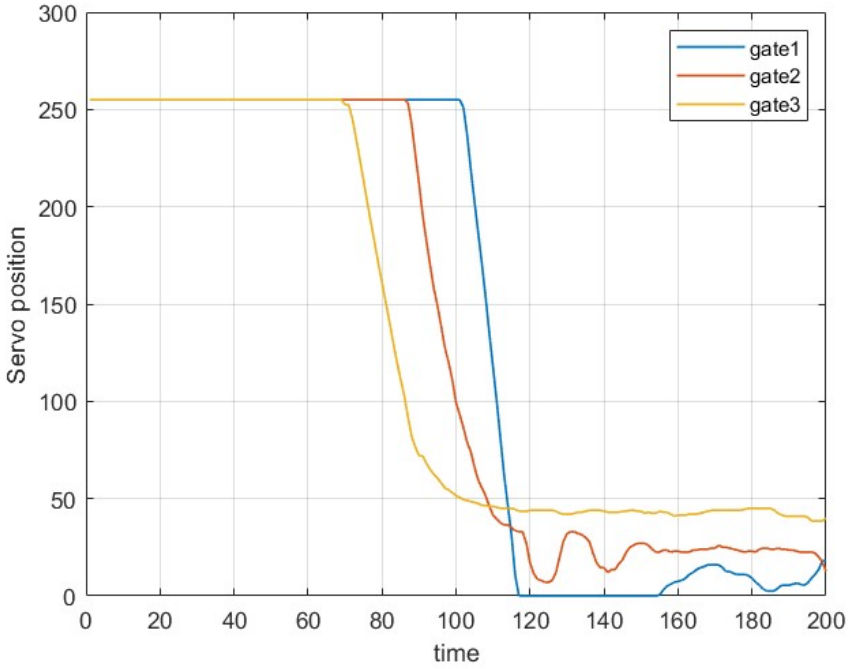


Figure 5.10: Control actions given to the gates

In Fig.5.10 is presented the control signal which is the commands sent to the servo in order to achieve the desired set points. The controller is also tested when off-take disturbances are present

which emulate farmers draining water from the canals. The results are presented in Fig.5.11 together with the control signals from the gates in Fig.5.12.

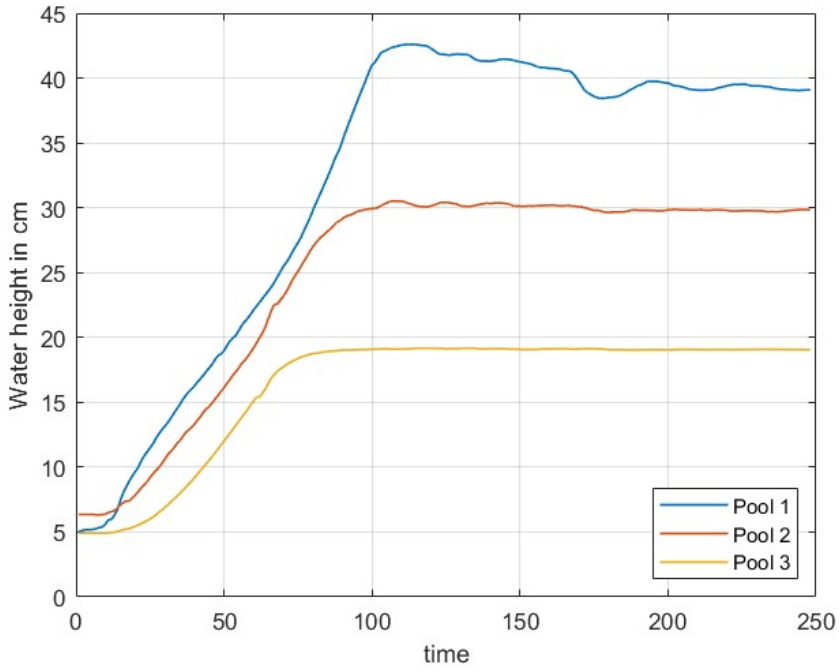


Figure 5.11: Experiments under an off-take disturbance

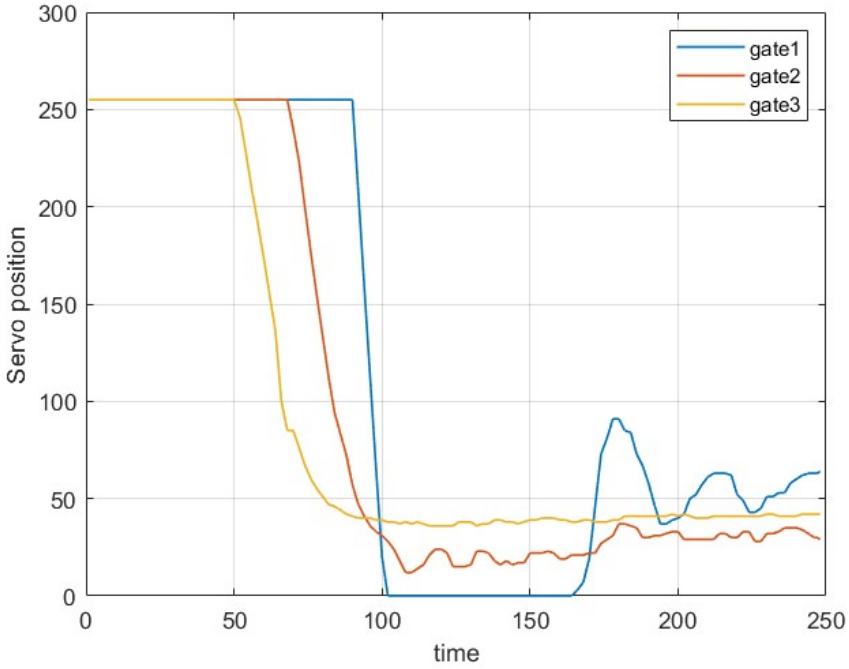


Figure 5.12: Control signal

6

Conclusion and future work

The aim of this thesis was to come up with a strategy to control the water irrigation system at the technical university of Delft. To achieve this, my work is hardly dependent on the progress of all the previous people working on the development of the setup. Although an event-triggered controller was designed by Bas Boot, it was quite slow and was tuned for the previous plant. Placing the water-restrictive panels changed the dynamics of the system completely, and a new controller needed to be tuned. Along with it, a nonlinear simulator was built in Matlab that aims to be a correct representation of the real plant.

6.1. Conclusions

In Ch.3 the full mathematical formulation of the nonlinear simulator is presented together with the results from its simulations. Before building it, based on the understanding of the nature of a real-life WIS, it is expected that the levels of the first and last compartments in a pool will differ. This behavior is confirmed by the testbed and the simulator Fig. 3.3, Fig.3.4, and Fig. 3.5. In terms of comparison between the real system and the simulated one, both Fig.3.9 and Fig. 3.10 can be explored. As stated in the chapter, the simulator manages to represent the testbed that has been explored. Of course, it can be tuned for a much closer match, but this means that all the leakages from the gates, internal to the pools, and from the valves need to be measured precisely. Further to that, the dynamics of the servos can be included in the model. In my literature review, I came across another formulation for simulating a water irrigation system. It is via the so-called St.Venant equations, which have the following form:

$$\begin{aligned} &\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = q_L \quad \text{Mass equation} \\ &\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x}\left[\frac{\beta Q^2}{A}\right] + gA\frac{\partial h}{\partial x} - gA\overline{S} + gA\frac{Q^2}{K^2} = 0 \quad \text{Momentum equation} \end{aligned} \tag{6.1}$$

They were the best option, but in the literature, they were mostly used to describe extremely lengthy channels with variable shapes. This more straightforward technique was utilized since our model could very easily be represented by a flow equation.

It can be seen that the controller manages to provide decent results and get the water heights very close to the desired set points. The system manages to tackle the disturbances while adjusting the gate position accordingly as seen in Fig.5.11 and Fig.5.12. Having those results together with Fig.5.9 and all the simulations, it can be said that the aim of this report has been achieved.

Being a decentralized controller has some drawbacks. One of them is the fact that each controller has information from one sensor. In that case, as soon as one pool of the string reaches its desired value, the downstream gate closes. That affects the water level the most in the first pool since there the leakage is the biggest and it can take too much time for the water to settle. Having this in mind, in the further sections, some other propositions for controllers will be discussed that could be applied in the future when this system is studied.

6.2. Future work

6.2.1. Plant in TU Delft

First of all, I would like to point out one drawback to the real system. All the undershoot gates have leakages which cause overshoots that slow down drastically the time needed for the experiments to settle.

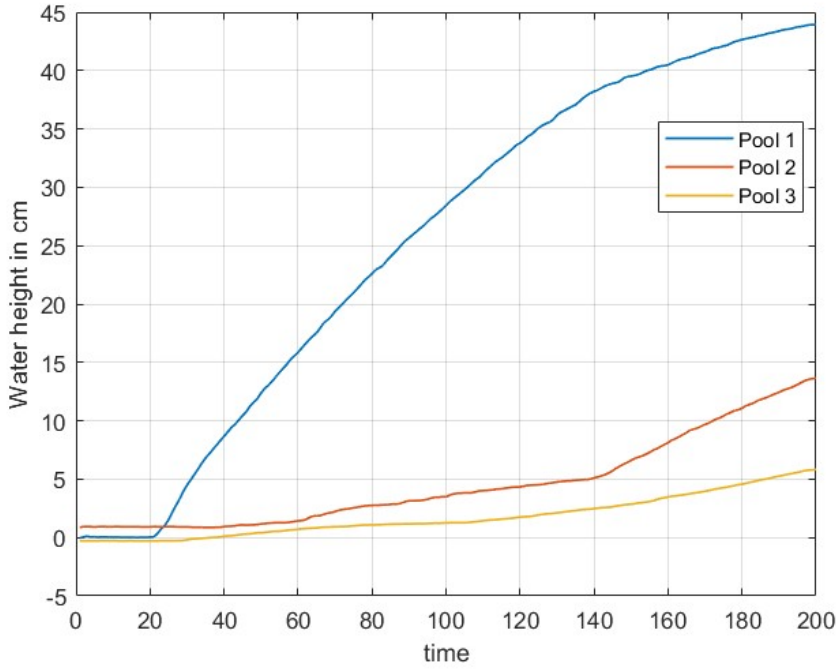


Figure 6.1: Water levels of the three pools with fully closed gates

In Fig 6.1 can be seen that the first pool stars fill quite fast even though the gate connecting it to the supply pool is fully closed. The same can be observed with the other pools, but at slower rates. For future experiments, the first pool can be used as a second storage pool, and only the water levels of the last two tanks can be controlled. Further to that, new gates or additional sealing of all the gates can be applied. This will significantly ease any experiments and contribute to building a more reliable simulator.

6.2.2. Control design

The PI controller designed in this master's thesis certainly serves its purpose very well. In Fig.5.9 can be seen that the water levels are really close to the desired setpoints. Due to the high sensitivity of the pressure sensors, the controller always needs to adjust the gate positions, which leads to results that are within half a centimeter of the desired value, which is still in the 2% - 3% of the steady state. One drawback of the current controller is that it works best if the desired upstream water level is higher than the downstream. One solution for that problem can be to design a global controller that will have all the information from all the pools and will adjust the gates accordingly.

Distributed controller
Inspiration about building such kind of controller can be found in [13] ,[14] and [4]. The controllers built in those reports are based on the $\mathcal{H}\infty$ loop-shaping approach. A localized portion of the controller for a single pool can be seen in Fig.6.2.

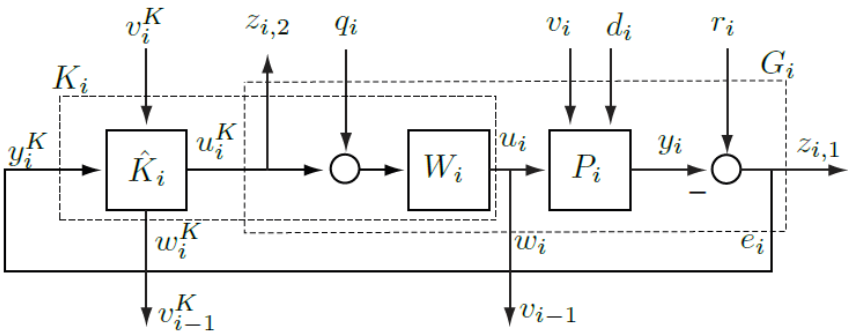


Figure 6.2: Localized portion of controlled channel, [13]

In this approach, the controller K_i is split into a loop-shaping weight W_i and a compensator $\hat{K}_i$ and P_i denotes the model of the plant. The local portion of the regulator aims to shape the plant dynamics to obtain a local loop gain that is consistent with the local performance. The resulting weight can be used as an input for the $\mathcal{H}_\infty$ loop-shaping of the distributed controller which aims for global performance. In Fig.6.2 v_i is the outflow from the downstream gate, v_{i-1} is the inflow from the upstream gate, r_i is the reference set point, d_i is the disturbance acting to the plant, y_i^K is the water level for i-th pool, u_i is the control signal, q_i is used to model additional uncertainties [13]. The local part, according to [4] can be a simple PI-compensator of the form:

$$W_i(s) = \frac{k_i(1 + s\phi_i}{s(1 + s\rho_i)} \tag{6.2}$$

The I-part of the controller needs to be present in order to achieve zero steady-state error. In order to achieve robustness k_i needs to be tuned so that the local cross-over frequency is less the $1/\tau_i$. Also, the loop gain of the shaped plant must be low in the frequencies surrounding the ones of the local wave dynamics to ensure that no excitement of the dominant waves. In order to synthesize $\hat{K}_i$ the

local weights and the plant are put together into a large matrix G which is the so-called generalized plant an $H(G, \hat{K})$ denotes the closed loop transfer function [13]. The aim is to select a $\hat{K}$ the stabilize and minimise:

$$||H(G, \hat{K})||_\infty = \sup_{\omega \in \mathbb{R}} \overline{\sigma}(H(G, \hat{K})(j\omega)) \tag{6.3}$$

Event-trigger control
Looking at Fig.5.10, it can be observed that the servos are changing their positions and adjusting the gates a bit too much to address the sensitivity of the sensors. This solution might not be very healthy for the servos, and also, constantly adjusting the control signals means constantly using the network. For

that reason, the controller can be expanded to an event-based control with a certain threshold Fig. 6.3. This will reduce the bandwidth and power used without affecting the system's productivity negatively.

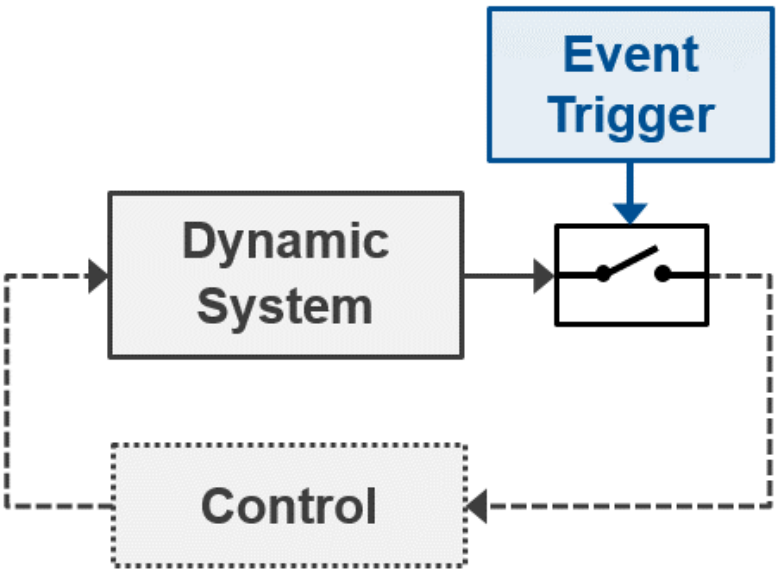


Figure 6.3: Event-based controller schematic solution

This approach can significantly reduce the communication in the network and thus reduce the cost of building it. In this approach, there is a mechanism that is constantly monitoring the triggering condition Eq.6.4 and adjusting the control actions only if the condition is met. There are some variations of the event-trigger control, which will be briefly discussed here.

$$\|\hat{x}(t_k) - x(t_k)\| > \sigma \|x(t_k)\| \tag{6.4}$$

Periodic Event-Trigger Control (PECT) In the ECT, the triggering condition is monitored continuously, and control action is taken if needed (violation of the condition). In PECT, a sampling time is chosen, and the condition is checked only for the times multiple of the sampling on $t_k = kh, k \in \mathbb{N}$. The control law for this type is:

$$\hat{u}(t) = K\hat{x} \text{ for } \quad t \in R_+ \tag{6.5}$$

with $\hat{x}$ left-continuous for $t \in (t_k, t_{k+1}]$ and is given by:

$$\hat{x}(t) = \begin{cases} x(t_k), & \text{when } \mathcal{C}(x(t_k), \hat{x}(t_k)) > 0 \\ \hat{x}(t_k), & \text{when } \mathcal{C}(x(t_k), \hat{x}(t_k)) \leqslant 0 \end{cases} \tag{6.6}$$

where $\mathcal{C}$ is the triggering condition. If $\mathcal{C}(x(t_k), \hat{x}(t_k)) > 0$ happen $x(t_k)$ is sent to the network and $\hat{x}$ and $\hat{(u)}$ are updated accordingly. If the second case from Eq.6.6 is met $\hat{u}$ is kept the same for at least one more sapling time period[10].

Self-Triggered Control (SCT) In order to reduce even further the calculation and transmission over the network, the time for a control action can be calculated based on a prediction. These are the so-called STC [9]. However, this controller does not take disturbances into account and is not applicable for the WIS because of all the disturbances that are present in the system. For that reason, a more suitable regulator can be the Preventive Event-Triggered Controller (PECT) [1]. It takes unknown but bounded disturbances into consideration and tries to estimate at what time an event will happen based on them.

References

[1] Gabriel de Albuquerque Gleizer and Manuel Mazo Jr. “Self-triggered output-feedback control of LTI systems subject to disturbances and noise”. In: *Automatica* 120 (2020), p. 109129.

[2] Karl J Astrom. “Automatic tuning of PID controllers”. In: *Insturment Society of America* (1988).

[3] Petra Bernhoff. *System identification and control of an irrigation channel with a tunnel*. 2008.

[4] Bas Boot. “Reducing wireless control communication for a water irrigation system”. In: (2021).

[5] Michael Cantoni et al. “Control of large-scale irrigation networks”. In: *Proceedings of the IEEE* 95.1 (2007), pp. 75–91.

[6] Stefano Casadei et al. “Application of smart irrigation systems for water conservation in Italian farms”. In: *Environmental Science and Pollution Research* 28 (2021), pp. 26488–26499.

[7] Ernest Doebelin. *System dynamics: modeling, analysis, simulation, design*. CRC Press, 1998.

[8] George Ellis. *Control system design guide: using your computer to understand and diagnose feedback controllers*. Butterworth-Heinemann, 2012.

[9] Wilhelmus PMH Heemels, Karl Henrik Johansson, and Paulo Tabuada. “An introduction to event-triggered and self-triggered control”. In: *2012 ieee 51st ieee conference on decision and control (cdc)*. IEEE. 2012, pp. 3270–3285.

[10] WPM Heemels Heemels, MCF Donkers, and Andrew R Teel. “Periodic event-triggered control for linear systems”. In: *IEEE Transactions on automatic control* 58.4 (2012), pp. 847–861.

[11] *Hitec HS785HB servo*. URL: `https://servodatabase.com/servo/hitec/hs-785hb`.

[12] *Keller sensor 35 series*. URL: `https://keller-druck.com/en/products/pressure-transmitters/front-flush-pressure-transmitters/series-35x`.

[13] Yuping Li and Michael Cantoni. “Distributed controller design for open water channels”. In: *IFAC Proceedings Volumes* 41.2 (2008), pp. 10033–10038.

[14] Yuping Li and Bart De Schutter. “Stability and performance analysis of an irrigation channel with distributed control”. In: *Control Engineering Practice* 19.10 (2011), pp. 1147–1156.

[15] JJ Lont. “Wireless Event-Triggered Control for Water Irrigation Systems”. In: (2020).

[16] Su Ki Ooi, MPM Krutzen, and Erik Weyer. “On physical and data driven modelling of irrigation channels”. In: *Control Engineering Practice* 13.4 (2005), pp. 461–471.

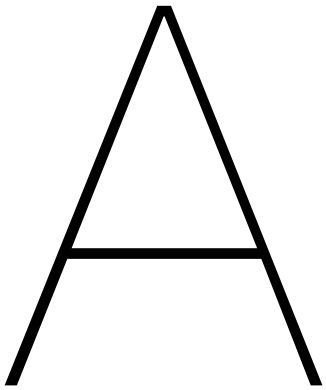
[17] Su Ki Ooi and Erik Weyer. “Control design for an irrigation channel from physical data”. In: *Control Engineering Practice* 16.9 (2008), pp. 1132–1150.

[18] J.M. Schram. “Handleiding watermodel”. In: (2001).

[19] *World population*. Accessed on 23.04.2023. URL: `https://data.worldbank.org/indicator/SP.POP.TOTL?end=2021&start=1960&view=chart`.

[20] *WSL installation*. URL: `https://pureinfotech.com/install-wsl-windows-11/`.

[21] *X410 extension*. URL: `https://x410.dev/`.



Non-linear simulator

This appendix will contain the Simulink implementation of the non-linear simulator used to represent the results of the real system. Together with the general overview will be shown the implementation of the flow equations for each pool.

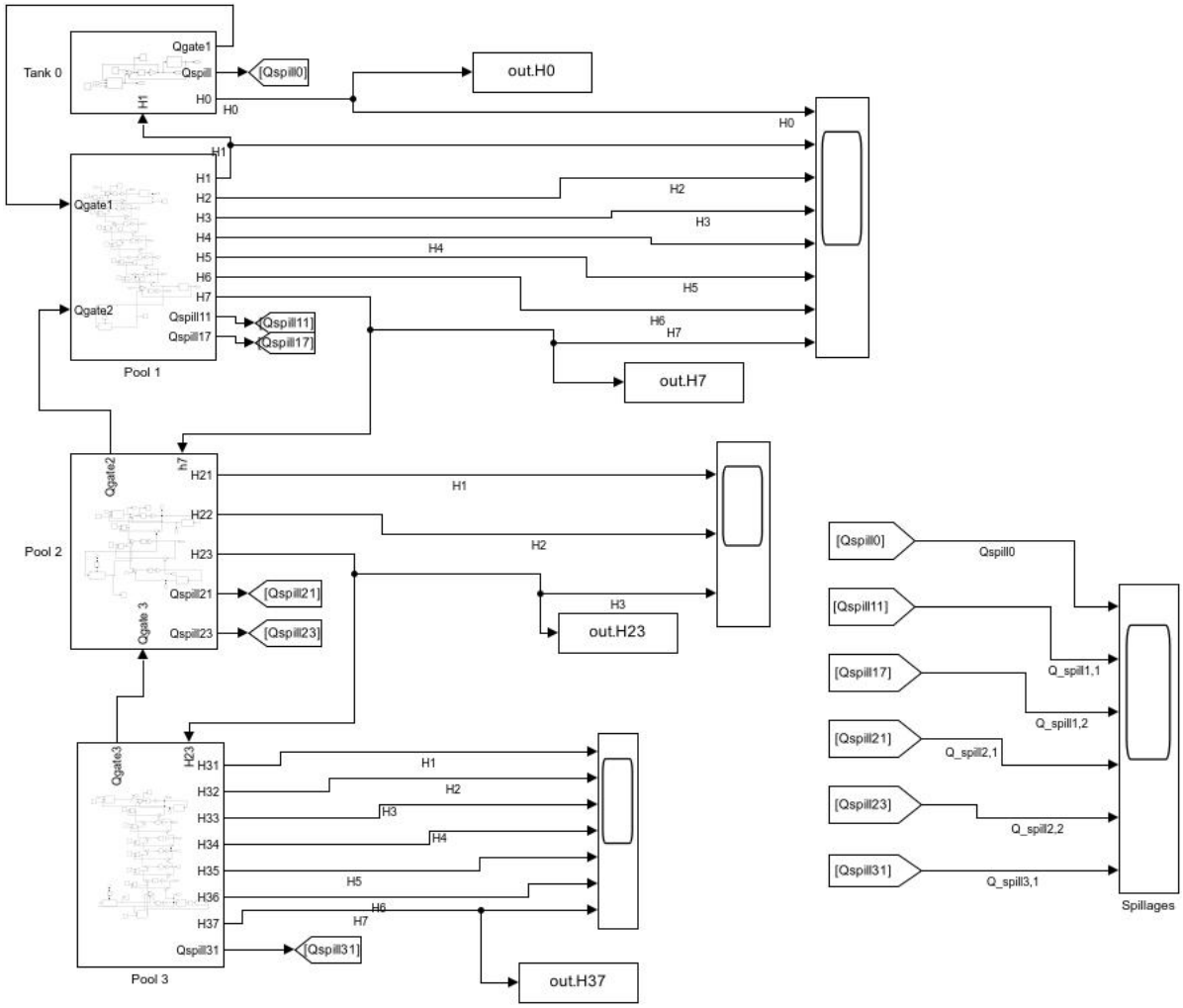


Figure A.1: General overview of the Simulink model of the non-linear simulator

For cleanness of the model, each pool is modeled in a different subsystem block where only the water levels and the spillage flows are given as outputs. In each subsystem the equations from Ch. 3 are applied in order to represent the flows that appear in the real system.

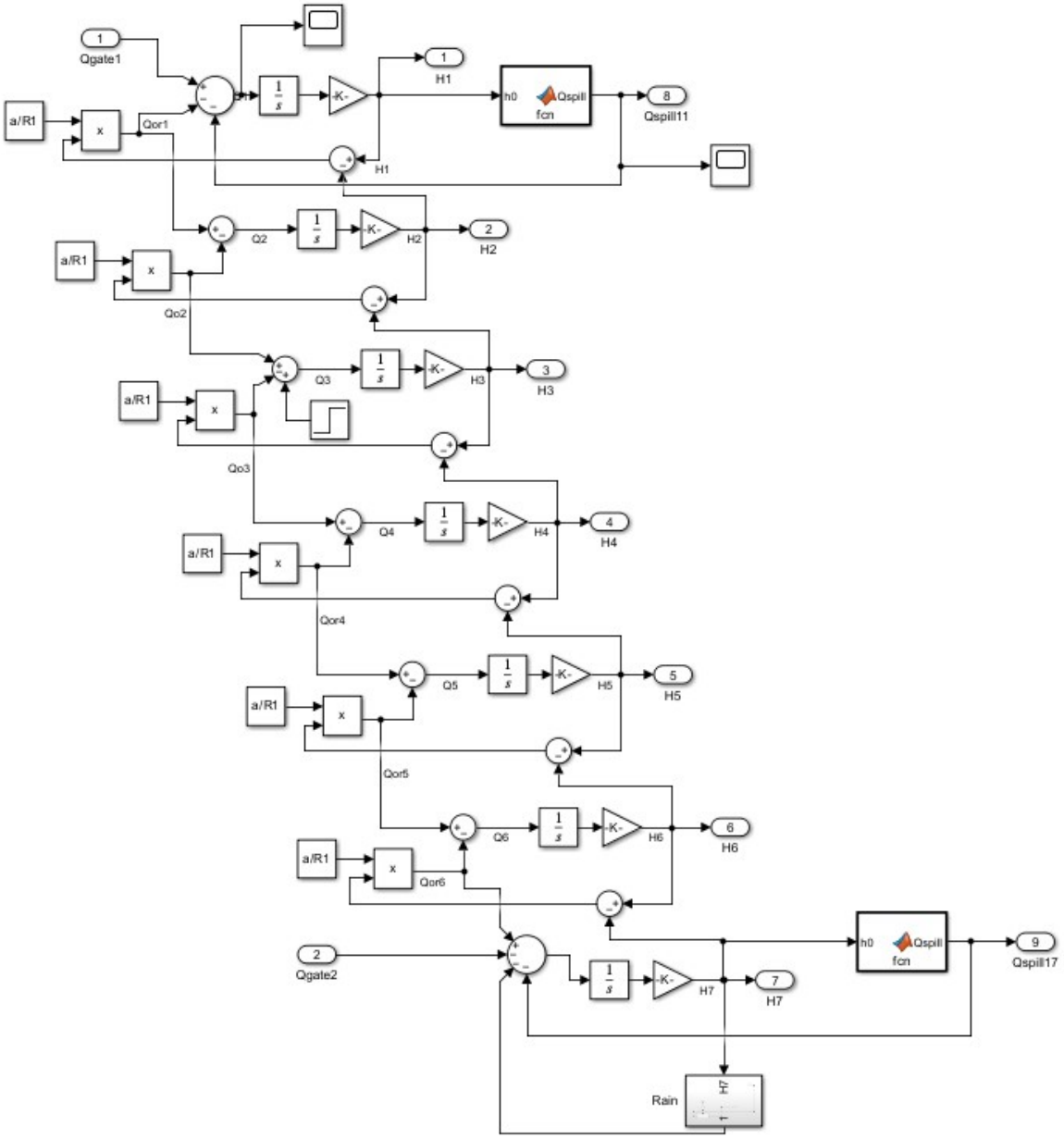


Figure A.2: An overview of how a pool is modeled

This is a representation of how a single pool is modeled. The spillage flows are represented in Matlab Function blocks and all the flows from the orifice lintels are implemented by simple blocks for easiness. Further to that, the disturbance from the rain implemented with a step block and the draining one are added.

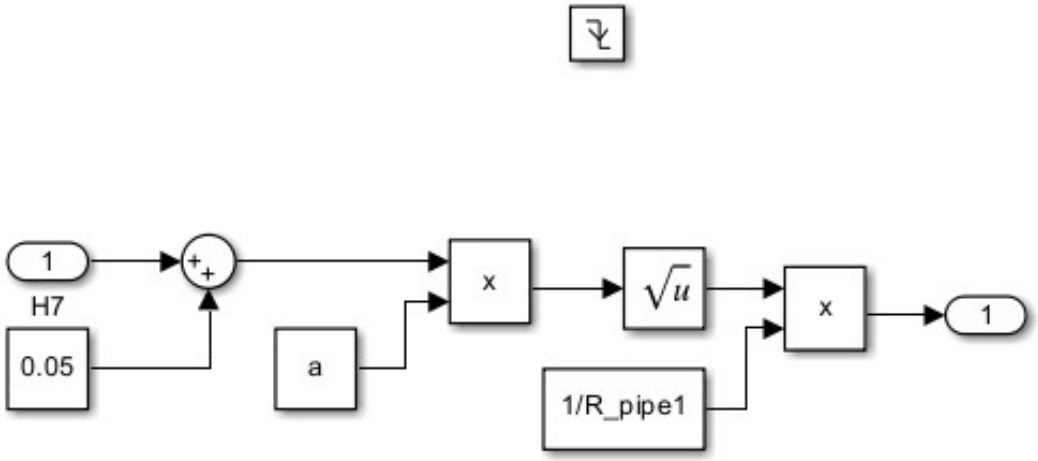


Figure A.3: Drain diturbance

To model the drain disturbance is used an equation that represents the flow from a pipe mounted at the bottom of the pool. Also, a timer trigger is used in order to trigger the disturbance at the same time as in the real setup.

B

Codebase

Here are collected all the codes, Simulink files, and .csv files used throughout this project. All of them are tuned with data from the real system in the TU Delft basement. For ease of sharing, they are uploaded to a GitHub repository. The versions used are Matlab 2022b and Python 3. Since half of the time while performing the experiment I was using the remote server to access it I suggest the use of Visual Code software, installing the ”Remote-SSH” extension together with the Python extension. Also, I was working on a Windows computer, so some additional installations are needed in order to work with the server. First of all, a Windows Subsystem for Linux (WSL) is needed since the computer in the lab is working on Linux [20]. Furthermore, the GUI Fig.2.10 and the cameras will not be visible if the X410 extension is not installed [21]. Having those, one can work effortlessly with the setup remotely.

```
https://github.com/IvayloGan/WIS.git
```