University of Stavanger

Faculty of Science and Technology

MASTER`S THESIS

Study program/Specialization: Master`s Degree Programme / Robot Technology and Signal Processing	Spring semester, 2023 Open
Writers: Gent Luta	Gent Luta (Writer's signature)
Faculty supervisor: Dr. Damiano Rotondo, Didrik Efjestad Fjereide	
Thesis title: *Design and Implementation of Model Predictive Control for a Coupled Tank System*	
Credits (ECTS): 30	
Key words: Model predictive control, Linear quadratic regulator, Quadratic programming, System modeling, Optimization	Pages: 182 + enclosure: 186 Stavanger, 15.07/2023 Date/year

University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Design and Implementation of Model Predictive Control for a Coupled Tank System

Master's Thesis in Robot Technology and Signal Processing
by

Gent Luta

Internal Supervisors

Damiano Rotondo

Didrik Efjestad Fjereide

July 15, 2023

“Running water never grows stale. So you just have to ***keep on flowing****.”*

Bruce Lee, 1996

Abstract

Model predictive control (MPC) is a control technique that optimizes, over the manipulated inputs, future trajectories of a dynamic system.

Given an objective function, the MPC uses a prediction model (which mathematically tries to encapsulate the dynamic behaviour of a system as accurately as possible) and an optimization algorithm in order to find the optimal manipulated inputs, which, when applied to the prediction model, results in a predicted trajectory that minimizes the objective function.

Model predictive control follows the *receding horizon* control policy, which implies that the optimization procedure is computed over a finite horizon window. When the optimization is completed, only the first calculated optimal manipulated input is applied to the system, and the rest are discarded. The finite horizon window then shifts by one sample, and the procedure is repeated *ad infinitum.*

This project investigates four types of model predictive controllers: (i) linear-, (ii) explicit-, (iii) adaptive-, (iv) and nonlinear-MPC. It is shown that, for a multi-input, coupled, fast-sampled tank system, the four types of model predictive controllers provide adequate control in regards to set-point tracking *and* disturbance rejection. Furthermore, it is shown that a model predictive controller can provide signal previewing functions as well as decoupling functions, while also being subjected to constraints on the controlled outputs, manipulated inputs, and the manipulated input rates. The aforementioned observations are shown using simulation results and experimental results from the tank system.

Additionally, more traditional control techniques such as LQR-, and PID-based controllers are evaluated in a similar manner as the four model predictive controllers. It is shown that the performance of the controllers, with regards to the integral of absolute error (IAE) performance index, can be ranked in the subsequent order (from best to worst): (i) nonlinear-MPC, (ii) linear-MPC, (iii) explicit-MPC, (iv) LQR-based controller, (v) adaptive-MPC, (vi) PID feedback controller with feedforward control action and a linear decoupler, (vii) PID feedback controller with feedforward control action, and lastly, (viii) PID feedback controller. These observations are shown using experimental results from the tank system.

Lastly, it is shown that the linear-, explicit-, and adaptive-MPC are able to provide feasible control policies with a sampling interval of $0.1\,s$. A feasible control policy implies that the model predictive controller is able to calculate the optimization procedure within

the time frame of one sample. For the nonlinear MPC, it is shown that this controller provides a feasible control policy with a sampling interval of $0.5\,s$. These observations are shown using experimental results from the tank system, in addition to simulation results.

Acknowledgements

I would like to thank the lovely people at the University of Stavanger for providing me with fives years of great experiences and memories.

Thank you to all of the amazing professors who, just through their pure dedication and brilliance, motivate hundreds of students daily. In particular, I would like to give a special thanks to Dr. Damiano Rotondo, Dr. Kristian Thorsen, Dr. Tormod Drengstig, and Dr. John Håkon Husøy for giving me endless wisdom, knowledge, and inspiration.

As special thanks goes to the doctors, surgeons, and nurses at Hillevåg - *Dagkirurgisk enhet.* You are all real-life superheros.

Lastly, I would like to express my deepest appreciation to my dear mother and father. The sacrifices you have made for me and my siblings are too many to count, and I will forever be grateful to you. *Ju dua shumë.*

Declaration of Authorship

I, Gent Luta, declare that this thesis titled, 'Design and Implementation of Model Predictive Control for a Coupled Tank System' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master's degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed: Gent Luta

Date: 15.07.2023

List of Figures

List of Tables

List of Algorithms

Contents

Chapter 1

Introduction

1.1 History of Automatic Control

In the 21st century, the advent of the Fourth Industrial Revolution has brought a rapid change in our technology and industries, especially in the fields of automatic control and artificial intelligence. While today's technological advancements happen at a rapid rate, there is a long history of scientists, engineers, mathematicians and other brilliant minds that built the foundation of fundamental theory.

Historians divide the history of automatic control in four appropriate periods: early control (to 1900), the pre-classical period (1900-1935), the classical period (1935-1960), and modern control (post-1955). The three former periods will be the focus of this section.

Remark 1.1. This section only discusses the history of automatic control from the early control (to 1900) and up to the classical period (1935-1960). As model predictive control was not introduced until 1978, this section does not cover the history of this control technique. While this section gives a brief account on the history of automatic control, some of the most significant inventions, and some of the most important discoveries within this field, it is not strictly required in order to understand the contents of this project. Therefore, the reader is free to skip to Section 1.2.

Early Control: To 1900

Some of the first automatic systems can be dated more than 2,000 years ago. The water clock is one of the oldest time-measuring instruments, and remnants of such devices have been found all over the world. The Greek inventor Ctesibius (285–222 B.C.) designed what is considered to be the first self-govern and self-regulating water clock.

Ctesibius' water clock consisted of two chambers: an initial chamber to ensure a constant inflow rate, and a second chamber to measure time. To ensure a constant inflow rate, Ctesibius fitted the initial chamber with an overflow pipe. The water from the overflow pipe was fed into the second chamber, which filled up at a precise rate. This allowed for an accurate measurement of time.

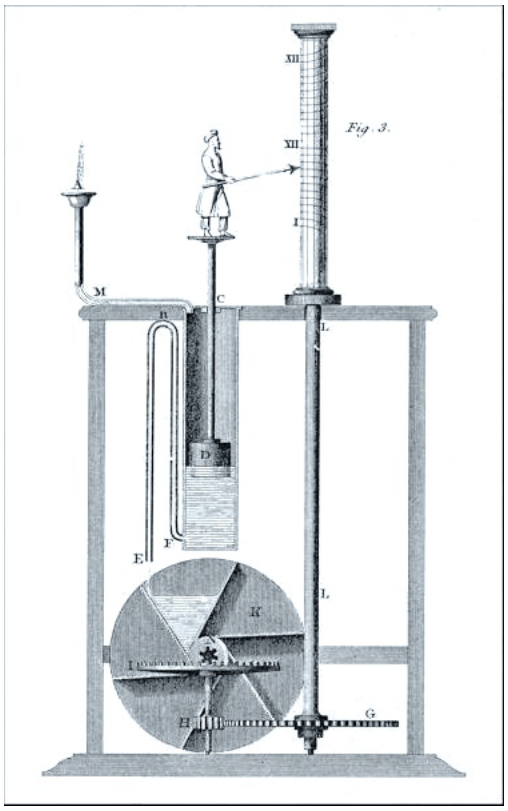


Figure 1.1: Illustration of Ctesibius' water clock. Image from Wikipedia / Public Domain Mark 1.0.

For his design to work continually, Ctesibius added a siphon to the second chamber. This allowed the water clock to be emptied and refilled automatically. While this was a great improvement from earlier water clocks, there was still one problem: the Greeks divided the daylight hours into twelve. This meant that their hours were shorter in the winter than in the summer.

To solve this, Ctesibius included a water wheel, cogs and a cylinder. The water that exited the second chamber through the siphon filled up the water wheel, which made it rotate. Through a mechanical linkage, the rotation of the water wheel caused the cylinder to rotate a tiny amount every day. The cylinder was etched with hour lines that was nearer or farther apart depending on the time of the year. A floating pointer was installed in the second chamber, which projected the current water level on to the cylinder, thus, indicating the time of day.

Ctesibius' water clock became the most accurate time piece ever constructed for over 1,800 years, until the pendulum clock was invented by Christiaan Huygens (1629-1695) in 1656.

* * *

One of the first recorded automatic feedback control systems was invented by the Dutch engineer Cornelis Drebbel (1572-1633). Cornelis invented a self-regulating furnace that maintained a desired temperature automatically.

The furnace itself consisted of three nested metal boxes, much like a Russian matryoshka doll, which was placed over an enclosed fire. The outer box was fitted with vents at the bottom and an opening at the top to ensure sufficient air circulation. The middle box was filled with water, and acted as a buffer between the intense heat in the outer box and the controlled heat in the center box. The center box is where Cornelis placed chicken eggs or different metals, and the furnace acted as an incubator [1].

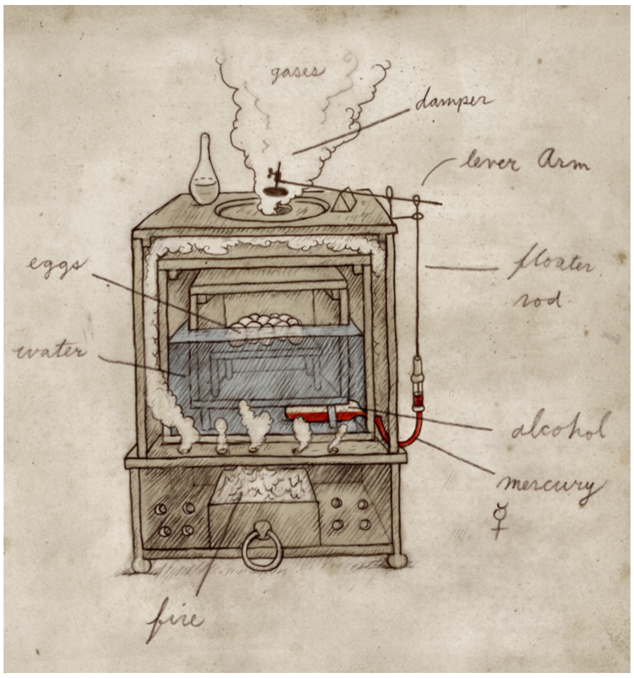


Figure 1.2: Illustration of Cornelis' oven. Image by Jonathon Rosen / All rights reserved.

To control the temperature in the center box, Cornelis devised what is considered to be one of the first thermostats. The thermostat was an L-shaped glass tube filled with alcohol and mercury. The tube was placed in the water of the middle box, just touching the center box. Towards the end of the tube was a metal rod, floating in the mercury. The metal rod was connected to a damper, which hovered over the top opening of the outer box.

Due to thermal expansion, the alcohol would expand and push the mercury towards the end of the tube. This caused the metal rod to rise upwards. When the metal rod elevated high enough, the damper would close down on the opening of the outer box, effectively cutting of the air that fed the fire. The fire would then slowly die out, and the temperature in the center box would drop to the desired point.

If the temperature was too low, then the alcohol would contract, and the metal rod would be lowered. This would lift up the damper, effectively increasing the air flow to the fire and, thus, increasing the temperature of the center box.

As many consider it the first of its kind, Cornelis had developed crucial principles for automatic temperature control and incubators. These principles began to be applied in the 18th century. E.g., some of the work of the French entomologist René Antoine Ferchault de Réaumur (1683-1757), who proposed several automatic temperature controlled devices, were based on Cornelis' furnace [2].

* * *

The steam engine governor is considered to be the most significant control development during the 18th century[2].

In 1776, the Scottish inventor James Watt (1736-1819) made significant improvements on Thomas Newcomen's steam engine (1712). Named after its creator, the *Watt steam engine* was more energy efficient than previous steam engines, and it is considered as a decisive breakthrough in the Industrial Revolution [3].

Since 1776, James made several improvements to the Watt steam engine. One desired feature for factory machinery was that they would operate at a constant speed. This was a feature that would be added to the Watt steam engine in 1789 [2].

In order for the Watt steam engine to operate at a constant speed, James included a throttle valve and a centrifugal governor to his design. Through a mechanical linkage, such as a drive shaft and a bevel gear construction, the center spindle of the governor would start to rotate when the steam engine was operating.

Attached to the center spindle were two masses on lever arms. The rotation of the center spindle transferred kinetic energy to the masses, causing them to experience centrifugal force. This force, if large enough, lifted the masses upwards against gravity. The lever arms, attached to the masses, would then change the position of a sleeve surrounding the center spindle. The position of this sleeve would, through another mechanical linkage, adjust the aperture of the throttle valve. The throttle valve controlled the flow of steam to the steam engine.

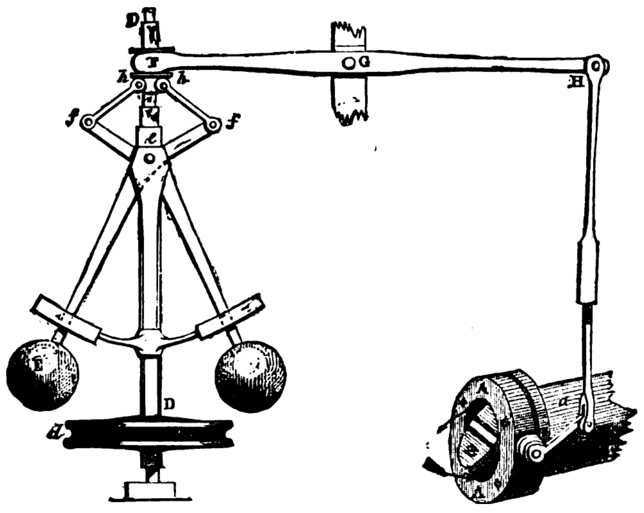


Figure 1.3: Illustration of a centrifugal governor. Image from Wikipedia / Public Domain Mark 1.0.

If the steam engine was over-speeding, then the center spindle would rotate rapidly, causing the masses to experience a large centrifugal force. The masses, together with the sleeve, would be lifted upwards against gravity. This motion caused the aperture of the throttle valve to close, cutting off the amount of steam supplied to the engine and, thus, lowering its speed.

By using the principles of proportional control, James created an automatic feedback system that could self-regulate the speed of the Watt steam engine. Being self-regulated and more energy efficient, relative to the then-existing steam engines, made the Watt steam engine the universal prime mover in many branches of the economy [3].

* * *

Although the Watt steam engine was a key technological advancement for the Industrial Revolution, it was not perfect. During the 19th century, even after James Watt's death, many efforts were made to improve on the original Watt governor. This lead to numerous governor patents being permitted all over the world.

One common problem with the centrifugal governors was the concept of *hunting*. Hunting occurred mostly due to the governors being over sensitive, meaning that any small change in the engine speed led to a substantial change in the throttle valve. This caused the engine speed to continuously oscillate below and above the desired speed.

Motivated by the concept of hunting, an extensive search for stability started in the mid-19th century. In 1868, the Scottish mathematician James Clerk Maxwell (1831-1879) published his paper 'On Governors'. In the paper, Maxwell derived differential equations to describe the dynamics for various governor mechanisms. These differential equations were used to provide necessary and sufficient *conditions* to determine the stability of the systems under consideration [4].

Although it was little recognised at the time, today, Maxwell's 'On Governors' is considered as a significant contribution and a central paper in control theory.

The Pre-Classical Period: 1900-1935

The application of feedback controllers grew rapidly in the early years of the 20th century. As systems and devices grew more complex, so did the quantity of variables that needed to be controlled. Voltage, current, and frequency were key control variables for the electric motor, while temperature, pressure, and flow were key control variables for the industrial processes.

Although control devices became more applicable, most of them were designed without any clear understanding of the control laws implemented or the dynamics of the system that was to be controlled [2]. There was a lack of theoretical understanding, as well as an absent technical language regarding control design. Furthermore, there was no simple design methods for control systems that could be easily applied. The design methods at the time were limited to the differential equations of the system and the Routh-Hurwitz stability criterion, which is a laborious process.

Despite the above-mentioned circumstances, complex control systems were being developed in the early years of the 20th century. One of the first systems that incorporated proportional-integral-derivative (PID) control and automatic gain adjustment was made in 1911, by the American inventor Elmer Sperry (1860-1930). Elmer created an automatic ship steering device that compensated for disturbances, which occurred when the sea conditions changed. Elmer's PID controller was based on an intuitive approach, rather than mathematical equations [5].

In 1922, the Russian American mathematician Nicolas Minorsky (1885-1970) published his paper titled 'Directional stability of automatically steered bodies'. During this period, Nicolas worked on automatic ship steering for the US Navy. Through observations of how a helmsman steered a ship, Nicolas provided theoretical analysis of the control laws that we today associate with PID control. Alongside Maxwell's 'On Governors', Nicolas' paper stands out as one of the significant contributions in control theory.

The Classical Period: 1935-1950

Between 1935 and 1942, there were many advancements in control theory being made across many different institutes and countries. Some of the most significant advancements are briefly mentioned below:

- In 1940, the American engineer Hendrik W. Bode introduced the concept of gain and phase margins. By plotting these margins, also known as Bode plots, one was able to display the frequency response of a system clearly. This resulted in a new approach where the system stability could be studied in the frequency domain. This method was considered as significantly simpler and easier to implement than traditional time-domain-based methods. Although Hendrik introduced this concept in 1940, the full details of his work was not known until he published the book titled 'Network Analysis and Feedback Amplifier Design' in 1945.
- In 1942, the two American control engineers John G. Ziegler and Nathaniel B. Nichols published their paper titled 'Optimum Settings for Automatic Controllers'. In the paper, they present a simple method for adjusting control parameters, by making use of the *ultimate sensitivity* and *period* of the system. The method eventually became known as the Ziegler-Nichols tuning rules, which is a prominent tuning method in control engineering today.
- During this period, the Electrical Engineering Department of MIT worked on several projects that would greatly impact the field of control engineering. Most notably was the projects led by Harold L. Hazen and Gordon S. Brown. These projects introduced, among other things, the use of block diagrams to represent and manipulate both electrical and mechanical systems, and differential analyzers to simulate control systems.

* * *

The race for technological superiority between the Allies and the Axis was an important aspect of the Second World War, and it played a critical role in its outcome. Research was highly motivated by military use, and technologies such as guided missiles, remote sensing, aircraft, and even atomic weapons were being developed at a rapid rate. The discipline of control systems was no exception, and research in this field also revolved around military applications.

The most prominent control task during the Second World War was aiming anti-aircraft weaponry. Properly aligning an anti-aircraft gun with its target was a cumbersome ordeal that involved many complex tasks: detecting the enemy target, calculating its future trajectory, and precisely control the movement of the heavy gun.

In 1943, the MIT Radiation Laboratory developed the SCR-584, which was an *automatic-tracking microwave radar*. The SCR-584 was the most advanced radar of its era, and it became the primary fire-control radar (FCR) for anti-aircraft weaponry.

Relaying the information from the SCR-584 to an anti-aircraft gun manually proved to be inefficient, as the enemy targets moved with such high velocity. Therefore, a system that directly linked the SCR-584 radar with the gun controller was needed. To accomplish this, a group[1] at Bell Telephone Laboratories created the M9 director.

The M9 director was a computer that continuously calculated fire solutions, based on the information that it automatically received from the SCR-584 radar. After calculating a fire solution, the M9 director would then automatically control the servomechanisms in the gun, effectively adjusting its position accordingly. This system proved to be very successful, especially against V-1 rockets.

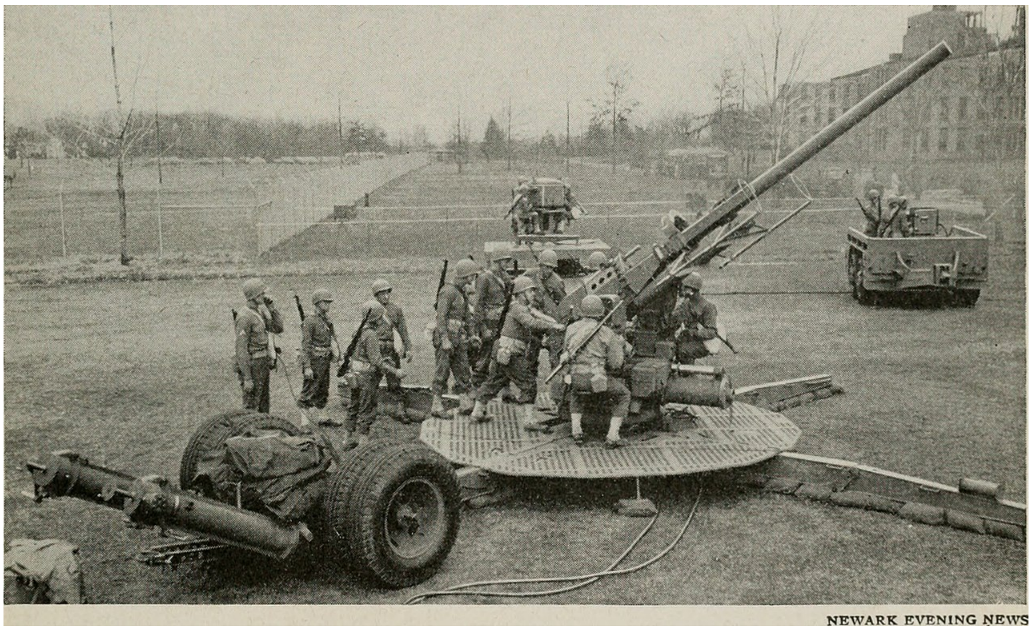


Figure 1.4: A demonstration of the M9 Director (center background) at Bell Laboratories in 1943. Image from Wikipedia / No known copyright restrictions.

The research related to the anti-aircraft controller system led to a broader understanding of bandwidth, noise and non-linearities in systems [2]. It also was a source of inspiration for the American mathematician Norbert Weiner (1894-1964), who made major developments in the study of stochastic systems with his book titled 'The Extrapolation, Interpolation and Smoothing of Stationary Time Series with Engineering Applications', which was published in 1942.

[1]Lead by Bode and including Blackman, C.A. Lovell, and Claude Shannon.

The majority of classical control techniques had been established by the end of the Second World War, and much of the research done on control systems began to be published in the post-war period[2].

1.2 Background and Motivation

Since the dawn of the Digital Revolution, the computer has become mankind's most important tool. In the 21st century, the frontier of technological advancement has been expanding vast and rapidly, including the complexity and capabilities of computers.

Today, the average smartphone is exponentially more powerful than the guidance computer NASA used for the famous Apollo 11 mission in 1969 [6]. While this astronomical advancement in technology is great for innovation, it also gives a rebirth to old ideas, one of which being Model Predictive Control (MPC).

MPC refers to a class of algorithms that utilizes an explicit process model to compute a sequence of manipulated variable adjustments in order to optimize the future behaviour of the plant [7],[8]. Originally presented as Model Predictive Heuristic Control (MPHC), Richalet *et al.* [9] offered the first description of MPC applications in 1978.

MPHC technology was originally developed for industrial processes such as petroleum refineries and chemical plants. These industrial processes are often highly nonlinear, multivariate systems subjected to multiple constraints, both economical and physical. Additionally, these systems are typically slow, which eased the real-time implementation of MPHC.

Due to the complex nature of these systems, the implementation of modern control techniques to industrial control had not been so successful [9]. In addition to that, Moore's law began to prove true during this time period, which resulted in much more powerful computers. These two statements became the core motivation behind the use of digital computers in industrial process control and the development of MPC.

Surveys such as [7] and [8] show how MPC technology quickly became popular for its intended market, namely areas like refining, petrochemicals and chemicals.

In the original description of MPHC, namely [9], the following question is raised:

> 'With the availability of much more powerful computers, should not the basic approaches to control systems application be reconsidered?'

[2]For a more detailed description on the history of automatic control, the article by Bennett [2] is strongly recommended.

A common metric for gauging our technological advancement is by comparing the scaling of metal–oxide–semiconductor field-effect transistors (MOSFET) over the years. When the aforementioned question first was raised in 1978, the average MOSFET scale was approximately 3-microns [10]. Today, the scaling is approximately $5nm$ (or 0.005-microns) [11].

This drastic change in computational power has considerably decreased the cost and increased the capacity of computers, sensors, communications and other hardware used in the industrial process. With this fact in mind, it is a fair assessment to raise a similar question as Richalet *et al.* [9] did almost 45 years ago:

With the availability of much more powerful computers, should not the basic approaches to control systems applications, in particular MPC technology, be reconsidered?

This leads us to the main motivation behind this project, which is to explore state-of-the-art MPC technology, and to investigate how it has evolved with the availability of much more advanced and sophisticated computers since its inception as MPHC.

1.3 Objectives

This project aims at designing and implementing an MPC-based control system for a two-tank configuration. The two-tank system is available in KE E-458, which is a laboratory facility at the University of Stavanger (UiS). The performance of the MPC-based control system will be compared against other more conventional control strategies.

At first, a simple linear MPC strategy will be used. In subsequent phases of the projects, other types of MPC strategies will be applied, such as explicit-, adaptive-, and nonlinear-MPC.

It is expected that the (dis)advantages of applying this technique are evaluated both in a simulation environment and with experimental data obtained from the two-tank system.

The activities and objectives for this project are presented in a condensed list as the following:

- Literature study and analysis of state-of-the-art MPC technology.
- Implementation and evaluation of linear MPC in a simulation environment.
- Implementation and evaluation of advanced MPC in a simulation environment.
- Experimental validation of the designed MPC-based control.

- Comparison of MPC against PID and LQR control strategies.
- Combining state estimation, in the form of Moving Horizon Estimation (MHE), with MPC.

1.4 Approach and Contributions

The overall approach to this project follows the same order as the objectives presented in Section 1.3. Firstly, literature study and analysis of the theory behind MPC is conducted. For this purpose, the literature available at the Stavanger University Library is made use of, in addition to published articles available on the internet.

When the literature study is concluded, the different MPC-based controllers are designed and implemented in a simulation environment. This is done in the programming software MATLAB, as well as in the toolbox Simulink. Note that, in order to properly design an MPC, a prediction model is required. Therefore, some time is spent on system modeling of the two-tank system.

After the MPCs are successfully implemented in a simulation environment, the next step is to experimentally validate the simulations, which is done by implementing the MPCs on the real two-tank system. As a part of the project, more traditional control techniques such as LQR- and PID-based controllers are also implemented on the two-tank system.

When all of the experimental trials are conducted for the MPC-, LQR-, and PID-based controllers, the results are evaluated, and their performances are compared against each other.

As a final part of the project, reflections are made on what worked as intended, what did not work, and what could have been done differently.

This project provides a descriptive work on the process of designing, implementing, testing, and evaluating MPC-based controllers.

The project covers the theoretical background of the MPC approach, while also providing a detailed discussion on practical matters such as MPC specification selection and code syntax for implementation.

Additionally, thorough experiments are conducted on the MPCs (both in simulations and on the real two-tank system), and these results are discussed and evaluated in detail.

This project provides further insight regarding what separates the MPC approach from more traditional control techniques such as LQR-, and PID-based controllers. This discussion is supplemented by experimental results from MPC-, LQR-, and PID-based controllers, respectively, where it is shown that the MPC-based controllers provided superior closed-loop systems, in terms of the performance index *integral of absolute error* (IAE).

1.5 Outline

This project is divided into the seven subsequent chapters:

- Chapter 1 introduces:
  - the background and motivation for the project,
  - what the objectives of the project are,
  - a brief summary of the overall approach, and
  - the outline of the project.
- Chapter 2 discusses:
  - the theoretical aspects of a linear-quadratic-regulator (LQR),
  - the theory behind a linear model predictive controller (MPC), and
  - the theory behind a nonlinear model predictive controller (NMPC).
- Chapter 3 gives:
  - a thorough description of the two-tank system,
  - a complete discussion on how to obtain a nonlinear model that describes the dynamics of the two-tank system,
  - a complete discussion on how to obtain a linear model from the nonlinear mode,
  - an analysis on the properties of the linear model, and
  - a brief introduction to numerical solvers and discretization methods.
- Chapter 4:
  - defines the different MPC specifications that need to be selected when designing an MPC,
  - it also provides detailed justifications as to why the different MPC specifications were selected as they were in this project, and lastly

  - it shows how to design a linear-, explicit-, adaptive-, and nonlinear-MPC using MATLAB syntax.

- Chapter 5:
  - looks to implement and evaluate the linear-, explicit-, adaptive-, and nonlinear-MPC in a simulation environment, and
  - it provides comparisons of the simulation results between the different MPC-based controllers.

- Chapter 6 looks to:
  - provide experimental validation of the linear-, explicit-, adaptive-, and nonlinear-MPC, by implementing them on the real two-tank system,
  - provide experimental validation of an LQR-based controller, by implementing it on the real two-tank system,
  - provide experimental validation of multiple PID-based controllers, by implementing them on the real two-tank system, and finally
  - it looks to compare the performance of the different controllers.

- Chapter 7 gives:
  - a brief summary of the different theoretical and practical aspects covered in this project,
  - a brief summary on the advantages of the MPC approach,
  - a brief summary on the disadvantages of the MPC approach, and
  - a brief discussion on possible future directions that are of interest to further understand, and to improve on, the MPC designs presented in this project.

Additionally, this project provides supplemental information and insight in the form of appendices. The appendices cover the following:

- Appendix A derives the infinite-horizon steady state optimum gain matrix K_∞ for the LQR-based controllers used during the experimental validation in Chapter 6.
- Appendix B derives the PID feedback controller, the feedforward control action, and the linear decoupler, which are used during the experimental validation in Chapter 6.
- Appendix C provides a brief description of the performance index *integral of absolute error* (IAE), which is a performance index used to compare the quality of different controllers.

- Appendix D provides a brief introduction to box plots, which are tools used by statisticians in order to present important statistical data graphically.
- Appendix E provides specifications (in the form of images) of the different components that the two-tank system consists of.
- Appendix F shows the original project description provided by the supervisor of this project.
- Appendix G provides the original project plan, including objectives and planned dates.
- Appendix H provides the master theses poster presentation, which is a mandatory part of the project to create.
- Appendix I provides all of the MATLAB code used during this project.
- Appendix J provides all of the Simulink schemes used throughout this project.

Finally, all of the references used and cited in this project are listed in the Bibliography.

Chapter 2

Related Work

This chapter is intended to familiarize the reader with the theoretical foundation of different MPC-based control systems. The two MPC strategies discussed in this chapter are *linear* and *nonlinear* MPC.

2.1 Preliminaries

2.1.1 Notation

The notation used in this project is fairly standard and in accordance with other notation from the control theory literature. This project uses an extensive amount of state-space representation to describe systems, both in continuous-time (CT) and discrete-time (DT). The following equations show a generic state-space representation for CT and DT systems:

$$CT \begin{cases} \dot{x}(t) = f(x(t), u(t), w(t), t) \\ y(t) = g(x(t), u(t), w(t), t) \end{cases} \tag{2.1}$$

$$DT \begin{cases} x(k+1) = f(x(k), u(k), w(k), k) \\ y(k) = g(x(k), u(k), w(k), k) \end{cases} \tag{2.2}$$

where:

- $t \in \mathbb{R}$ denotes the current time.
- $k \in \mathbb{Z}$ denotes the current time step.
- $x \in \mathbb{R}^{\mathsf{n}}$ denotes the state vector.

- $u \in \mathbb{R}^{\mathfrak{m}}$ denotes the input vector.
- $w \in \mathbb{R}^{\mathfrak{q}}$ denotes the disturbance vector.
- $y \in \mathbb{R}^{\mathfrak{p}}$ denotes the output vector.
- $f(\cdot) : \mathbb{R}^{\mathfrak{n}} \times \mathbb{R}^{\mathfrak{m}} \times \mathbb{R}^{\mathfrak{q}} \times \mathbb{R} \to \mathbb{R}^{\mathfrak{n}}$.
- $g(\cdot) : \mathbb{R}^{\mathfrak{n}} \times \mathbb{R}^{\mathfrak{m}} \times \mathbb{R}^{\mathfrak{q}} \times \mathbb{R} \to \mathbb{R}^{\mathfrak{p}}$.

The values $\mathfrak{n}$, $\mathfrak{m}$, $\mathfrak{q}$ and $\mathfrak{p}$ are the number of states, inputs, disturbances, and outputs, respectively.

In the special case where $f(\cdot)$ and $g(\cdot)$ are linear functions of x and u, then the system is called *linear*. The following equations show a generic state-space representation for linear CT and DT systems:

$$CT \begin{cases} \dot{x}(t) = A(t)x(t) + B(t)u(t) \\ y(t) = C(t)x(t) + D(t)u(t) \end{cases} \tag{2.3}$$

$$DT \begin{cases} x(k+1) = A(k)x(k) + B(k)u(k) \\ y(k) = C(k)x(k) + D(k)u(k) \end{cases} \tag{2.4}$$

where:

- $A(\cdot) \in \mathbb{R}^{\mathfrak{n}\times\mathfrak{n}}$ denotes the state matrix.
- $B(\cdot) \in \mathbb{R}^{\mathfrak{n}\times\mathfrak{m}}$ denotes the input matrix.
- $C(\cdot) \in \mathbb{R}^{\mathfrak{p}\times\mathfrak{n}}$ denotes the output matrix.
- $D(\cdot) \in \mathbb{R}^{\mathfrak{p}\times\mathfrak{m}}$ denotes the feedthrough matrix.

2.1.2 LQR control

The linear quadratic regulator (LQR), which serves a crucial role in the formulation of the linear quadratic Gaussian (LQG) problem, is considered to be one of the most important and influential results in optimal control theory to date [12]. LQR is a feedback controller developed to control a dynamic system at a minimum cost.

The theory of LQR is a fundamental cornerstone to understanding MPC, therefore, it is deemed necessary to provide a short summary. Note that this section is a highly abbreviated summary of LQR, intended only to provide the main results. For the finer details and proofs related to LQR, the reader is referred to sources such as [13], [14], [15], [16] and [17].

Continuous time LQR control

To start, consider a linear time-invariant (LTI) system given by the following vector-matrix differential equation:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.5}$$

We seek a linear control law to command the input vector $u(t)$ in a desirable manner. The control law is proportional to the state vector, given by the following form:

$$u(t) = -K(t)x(t) \tag{2.6}$$

where $K(t) = [k_1(t), k_2(t), \dots, k_n(t)]$ is a suitable gain matrix. Equation (2.6) is similar to the control law used in full state feedback (FSF). However, FSF seeks a gain matrix $K(t)$ to place the closed-loop poles of the system in a *pre-determined* location. In the case of LQR, we seek a gain matrix $K(t)$ to minimize a specific performance criterion J, often referred to as a *cost function*.

The cost function J is expressed as an integral of the sum between the quadratic form in the state $x(t)$ and control $u(t)$, respectively. I.e.:

$$J = \int_{\tau}^{\mathcal{T}} \left[x^T(t)Qx(t) + u^T(t)Ru(t)\right] dt \tag{2.7}$$

where $Q \succeq 0$ and $R \succ 0$ are symmetric matrices, and often called the *state weighting* matrix and *control weighting* matrix, respectively. The lower limit τ and the upper limit $\mathcal{T}$ are identified as *present time* and *final time*, respectively. The final time $\mathcal{T}$ can either be finite or infinite, and these two cases are referred to as the CT *finite-horizon* or *infinite-horizon* LQR problems, respectively.

Remark 2.1. Minimizing J implies minimizing the terms $x^T(t)Qx(t)$ and $u^T(t)Ru(t)$. This means that the desired state is the origin.

The matrices Q and R specify how significant each element in $x(t)$ and $u(t)$ is, respectively, to the overall cost function J, relative to each other. Consider the following example:

Example 2.1. *A system is described by its position $x_1(t)$, velocity $x_2(t)$, and acceleration $x_3(t)$, yielding the state vector $x(t) = [x_1(t), x_2(t), x_3(t)]^T$. If only the position of the system is of concern, with no regards to its velocity or acceleration, then a state weighting matrix Q can be selected as:*

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

which yields the quadratic form:

$$x^T Q x = x_1^2$$

Alternatively, the position is still of importance, but we want to slightly limit its velocity. A possible choice for Q in this instance can be:

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

which yields the quadratic form:

$$x^T Q x = x_1^2 + 0.2x_2^2$$

The same logic can be applied to the input weighting matrix R.

Substituting $u(t)$ in (2.5) with the control law (2.6), results in the following closed-loop dynamic system:

$$\dot{x}(t) = Ax(t) - BK(t)x(t) = A_c(t)x(t) \tag{2.8}$$

where

$$A_c(t) = A - BK(t) \tag{2.9}$$

The closed-loop dynamic system (2.8) is expressed as a "homogeneous", unforced equation, with $A_c(t)$ making the closed-loop state matrix. From the control theory (see e.g. [14]), that the solution to (2.8) is given by:

$$x(t) = \Phi(t, \tau)x(\tau) \tag{2.10}$$

where $\Phi(t, \tau)$ is the state-transition matrix, which relates the state at time t to the state at time τ, given that $t - \tau > 0$.

With the control law (2.6), the cost function J can be rewritten as:

$$J = \int_{\tau}^{\mathcal{T}} \left[x^T(t)Qx(t) + x^T(t)K^T(t)RK(t)x(t)\right] dt$$

Furthermore, $x(t)$ can be substituted by (2.10):

$$\begin{aligned} J &= \int_{\tau}^{\mathcal{T}} \left[x^T(\tau)\Phi^T(t,\tau)Q\Phi(t,\tau)x(\tau) + x^T(\tau)\Phi^T(t,\tau)K^T(t)RK(t)\Phi(t,\tau)x(\tau)\right] dt \\ &= \int_{\tau}^{\mathcal{T}} \left[x^T(\tau)\Phi^T(t,\tau)\{Q + K^T(t)RK(t)\}\Phi(t,\tau)x(\tau)\right] dt \\ &= x^T(\tau)P(\tau,\mathcal{T})x(\tau) \end{aligned} \tag{2.11}$$

where

$$P(\tau, \mathcal{T}) = \int_{\tau}^{\mathcal{T}} \Phi^T(t,\tau)\{Q + K^T(t)RK(t)\}\Phi(t,\tau)\,dt$$

Clearly, any choice of the gain matrix $K(t)$ directly affects the cost function J given by (2.11). The question then arises: how does one choose an optimum gain matrix $K(t)$, such that J is minimized? The answer is given by the following theorem:

Theorem 2.2. *The optimum gain matrix $K(t)$ that minimizes the CT finite-horizon LQR problem is:*

$$K(t) = R^{-1}B^T P(t, \mathcal{T})$$

where $P(t, \mathcal{T})$ is a symmetric matrix found by solving the Riccati differential equation:

$$-\dot{P}(t,\mathcal{T}) = P(t,\mathcal{T})A + A^T P(t,\mathcal{T}) - P(t,\mathcal{T})BR^{-1}B^T P(t,\mathcal{T}) + Q \tag{2.12}$$

with the boundary condition

$$P(\mathcal{T},\mathcal{T}) = 0$$

Proof. See [14], Section 9.4 of Chapter 9. □

Note that the only condition that must be satisfied is for the boundary $P(\mathcal{T},\mathcal{T}) = 0$, which means that the Riccati differential equation (2.12) is solved by integrating *backward* in time.

As Theorem 2.2 suggests, the solution to the CT finite-horizon LQR problem yields a gain matrix $K(t)$ which is time-varying. However, a steady state gain matrix can be obtained by solving the CT infinite-horizon LQR problem.

If $\mathcal{T} \to \infty$, then $P(t,\infty)$ will either: "blow up" (i.e., $P(t,\infty) \to \infty$) or converge to some constant matrix (i.e., $P(t,\infty) \to P_\infty$). In the case of the latter, as $P(t,\infty)$ converges, its derivative will also converge to zero (i.e., $\dot{P}(t,\infty) \to 0$).

This allows for a simple, yet elegant, solution to the CT infinite-horizon LQR problem given by the following theorem:

Theorem 2.3. *The optimum steady state gain matrix K_∞ that minimizes the CT infinite-horizon LQR problem is:*

$$K_\infty = R^{-1}B^T P_\infty$$

where P_∞ is a symmetric matrix found by solving the algebraic Riccati equation:

$$0 = P_\infty A + A^T P_\infty - P_\infty BR^{-1}B^T P_\infty + Q \tag{2.13}$$

Proof. See [14] Section 9.5 of Chapter 9, or, [13] Section 6 *Solution of the linear regulator problem.* □

Remark 2.4. Solving the Riccati equation (2.12) and the algebraic Riccati equation (2.13) yields multiple solutions to $P(t, \mathcal{T})$ and P_∞, respectively. However, if the system is:

(i) Asymptotically stable, or

(ii) Controllable and observable,

then there exists a *unique*, positive definite solution for $P(t, \mathcal{T})$ and P_∞. For a complete discussion regarding this, the reader is referred to [13] Section 5: *Controllability.*

Discrete time LQR control

Analog to the discussion above, the discrete time LQR counterpart follows the same reasoning with minor deviations. Consider the following DT LTI system:

$$x(k+1) = Ax(k) + Bu(k) \tag{2.14}$$

The optimal control law is given by:

$$u(k) = -K(k)x(k) \tag{2.15}$$

where $K(k) = [k_1(k), k_2(k), \ldots, k_n(k)]$. As before, the goal is to find an optimum gain matrix $K(k)$ that minimizes a certain quadratic cost function J. For the DT case, J is given by:

$$J = \sum_{k=0}^{N-1} [x^T(k)Qx(k) + u^T(k)Ru(k)] \tag{2.16}$$

where Q, R and N are the state weighting matrix, control weighting matrix and the final time step, respectively. Assuming that the control horizon is finite, then the solution is given by the following theorem:

Theorem 2.5. *The optimum gain matrix $K(k)$ that minimizes the DT finite-horizon LQR problem is:*

$$K(k) = (R + B^T P(k+1)B)^{-1} B^T P(k+1)A \tag{2.17}$$

where $P(k)$ is a symmetric matrix found by solving the Riccati difference equation:

$$P(k-1) = A^T P(k)A - A^T P(k)B(R + B^T P(k)B)^{-1} B^T P(k)A + Q \tag{2.18}$$

with the terminal condition:

$$P(N) = 0 \tag{2.19}$$

Proof. See e.g. [15] or [16]. □

As discussed for the CT case, solving the finite-horizon LQR problem yields a time-varying optimum gain matrix $K(k)$. Note also that (2.18) solves for *previous* values of $P(k)$, i.e., $P(k)$ is found iteratively backwards in time with (2.19) as the "starting point".

Alternatively, solving the DT infinite-horizon LQR problem yields a steady state optimum gain matrix K_∞, which can be obtained as follows:

Theorem 2.6. *The optimum gain matrix K_∞ that minimizes the DT infinite-horizon LQR problem is:*

$$K_\infty = (R + B^T P_\infty B)^{-1} B^T P_\infty A \tag{2.20}$$

where P_∞ is a symmetric matrix found by solving the algebraic Riccati equation:

$$P_\infty = A^T P_\infty A - A^T P_\infty B (R + B^T P_\infty B)^{-1} B^T P_\infty A + Q \tag{2.21}$$

Proof. See e.g. [15] or [16]. □

Remark 2.4 also applies for the DT finite/infinite-horizon LQR problems, with respect to the equations (2.18) and (2.21).

The control theory literature regarding LQR is both extensive and comprehensive, especially considering its properties such as stability and optimality. The sources cited in this section are highly recommended to the interested reader who wants to learn more on this topic.

2.2 Linear MPC

The essence of MPC is to optimize, over the manipulated inputs, forecasts of process behavior [18]. The forecasts (or predictions) are made based on a process model, hence the name *model predictive*. To do this optimization, the MPC is dependent on current measurements and the prediction model. While there is some research done on CT MPC using analog circuitry (see e.g. [19]), in an overwhelming majority of cases, the implementation is done as digital control. Therefore, the subsequent discussion will only regard DT MPC.

A typical feedback loop for an MPC-based control system is shown in Figure 2.1. The MPC-based controller consists of a prediction model and an optimizer. The prediction model is a mathematical model that tries to capture the dynamics of the plant as accurately as possible. The optimizer is a computer algorithm that tries to find optimal control moves that drive the plant to some desired state. MPC-based controllers are dependent on the feedback loop, which provides it with the measured outputs of the plant. The references are necessary if the goal of the control system is target tracking. The disturbance rejection of the control system can also be improved if the MPC is provided with measured disturbances. A more concrete example illustrating the MPC

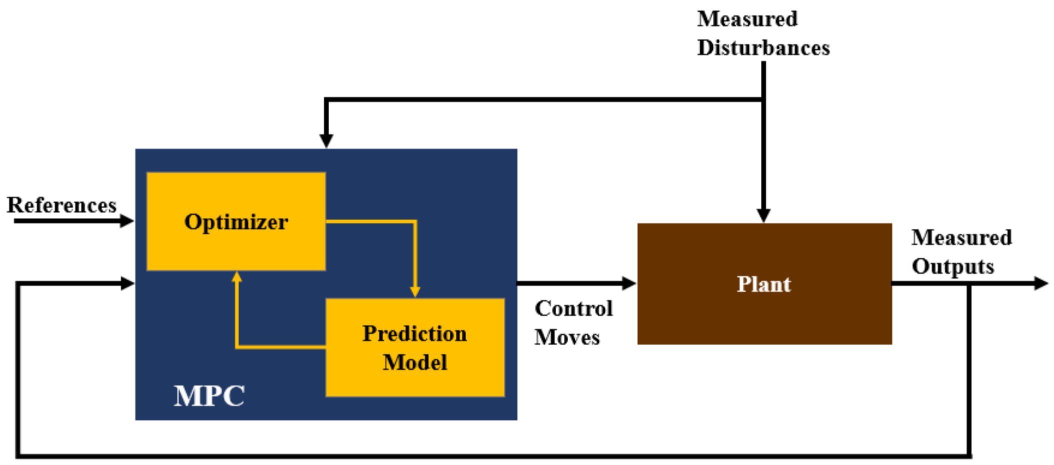


Figure 2.1: A simple MPC conceptual block diagram.

algorithm is shown in Figure 2.2. At the current sampling time k, a new measured

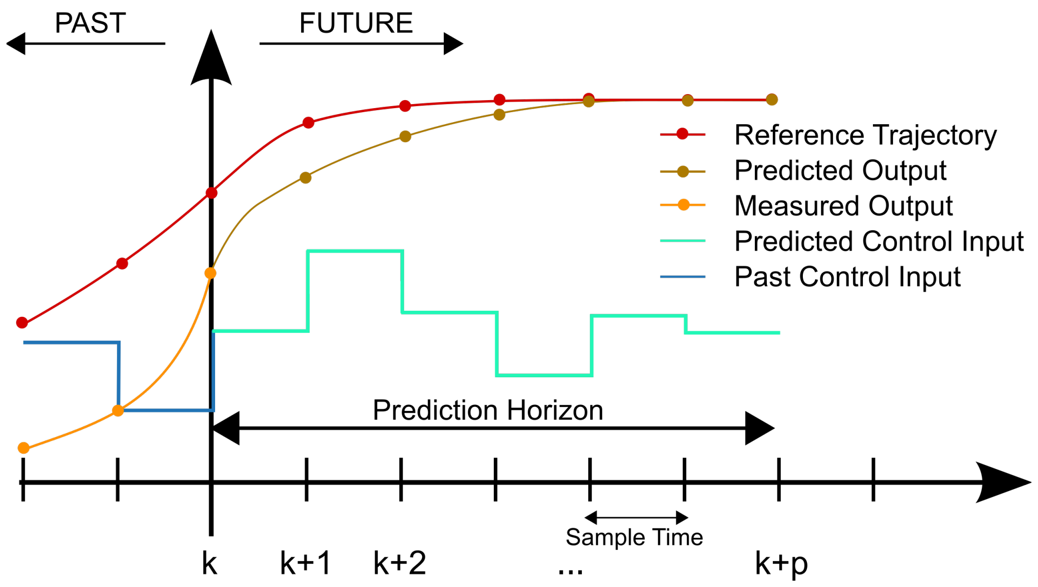


Figure 2.2: A discrete MPC scheme. Figure by Martin Behrendt/CC BY-SA 3.0

.

output (yellow) becomes available. The goal of the MPC calculations is to determine a sequence of predicted control inputs (cyan), such that the predicted output (gold) follows a reference trajectory (red), over a prediction horizon p, in an optimal manner.

In mathematical terms, this simple MPC scheme can be described subsequently: At sampling time k, the output measurement $y_m(k)$ becomes available. Linear MPC uses an LTI prediction model, such as (2.14), to calculate a set of M values of the predicted control input $\{u(k+i-1),\, i=1,2,\ldots,M\}$. I.e.:

$$\mathfrak{U}_{opt}(k) = \{u_{opt}(k), u_{opt}(k+1), \ldots, u_{opt}(k+M-1)\}$$

Note that:

(i) The set of M values containing the predicted control input is denoted by $\mathfrak{U}$.

(ii) Each element in the aforementioned set is the input vector, i.e., $u_{opt}(\cdot) \in \mathbb{R}^{\mathfrak{m}}$.

(iii) The subscript $_{opt}$ denotes the optimal calculated set.

The set $\mathfrak{U}_{opt}(k)$ is calculated in such a way, that the set of P values of the predicted states $\{x(k+i),\, i=1,2,\ldots,P\}$ follows the reference trajectory in an optimal manner. I.e.:

$$\mathfrak{X}_{opt}(k) = \{x_{opt}(k+1), x_{opt}(k+2), \ldots, x_{opt}(k+P)\}$$

Note that the set $\mathfrak{X}_{opt}(k)$ does not contain a prediction for $x_{opt}(k)$. This state can be estimated based on the output measurement $y_m(k)$.

The number of predictions P is regarded as the *prediction horizon*, while the number of calculated control moves M is referred to as the *control horizon* [20]. The MPC strategy requires that $P \geq M \geq 1$. If $P > M$, then the predicted outputs at sampling time k, where: $\{x_{opt}(\kappa),\, \kappa > k+M-1\}$, are calculated based on the last predicted control input $u_{opt}(k+M-1)$. I.e., the input is held constant after the M predicted control moves [20].

After the optimal set $\mathfrak{U}_{opt}(k)$ of M values is calculated, only the first control move $u_{opt}(k)$ is actually sent to the actuators. The other control moves are discarded. On the next sampling instant $k+1$, the MPC will receive the output measurement $y_m(k+1)$, and calculate the following:

$$\mathfrak{U}_{opt}(k+1) = \{u_{opt}(k+1), u_{opt}(k+2), \ldots, u_{opt}(k+M)\}$$

which gives the optimal predicted state:

$$\mathfrak{X}_{opt}(k+1) = \{x_{opt}(k+2), x_{opt}(k+3), \ldots, x_{opt}(k+P+1)\}$$

Again, only the first control move $u_{opt}(k+1)$ is implemented, and the output measurement $y_m(k+2)$ becomes available on the next sampling instant. This procedure continues ad infinitum and is often referred to as a *moving horizon* or *receding horizon*, which is a characteristic trait of the MPC strategy.

A reasonable question posed by the critical reader might be *why calculate an M-step control law, when only the first control action is implemented and the rest are discarded.* This strategy can be justified for two reasons:

(i) Any abstract mathematical model that tries to describe a physical system will never be able to capture its full dynamics. There will always be phenomenons that cannot be modeled, thus, model error and/or model uncertainty will always be present [21].

(ii) Unmeasured disturbances may affect the system at any given time.

The aforementioned reasons may cause the predicted state to diverge from the real process behaviour, which may result in the predicted control input no longer being suitable. This explains why the entire set of predicted control inputs is not applied blindly to the system, but rather, only the first predicted control move is applied.

Remark 2.7. One important detail not mentioned in the discussion above, and that does not appear in Figure 2.2, is the fact that it takes time to perform the MPC calculations. When a new output measurement $y_m(k)$ becomes available, the set $\mathfrak{U}_{opt}(k)$ is calculated after a certain time $\gamma \in \mathbb{R}$. In reality, the control move $u_{opt}(k)$ is implemented no earlier than at time $kT_s + \gamma$, where $T_s \in \mathbb{R}$ is the sampling time. However, it is often assumed that $\gamma \to 0$.

A very desirable feature of MPC is its ability to handle constraints. E.g., when solving for the optimal set of control moves $\mathfrak{U}_{opt}(k)$, one can impose the following constraints:

$$U_{min} \leq \mathfrak{U}_{opt}(k) \leq U_{max},\ \forall k = 0, 1, 2, \ldots, \infty$$
$$X_{min} \leq \mathfrak{X}_{opt}(k) \leq X_{max},\ \forall k = 0, 1, 2, \ldots, \infty$$

Subjecting the controller to such constraints is reasonable in several (if not all) practical situations. Consider a simple example where the water level in a tank is controlled by adjusting the aperture of an outlet valve. The water level in the tank cannot be less than zero, additionally, it cannot be higher than the height of the tank itself. On the other hand, the aperture of the valve cannot be more closed than fully closed, as well as more open than fully open (also referred to as *valve saturation*). Physical limitations such as these create upper and lower bounds, which are constraints that the MPC can take into consideration when calculating the predicted control moves and predicted states, respectively.

To calculate the optimal prediction set $\mathfrak{U}_{opt}(k)$ and $\mathfrak{X}_{opt}(k)$, a performance measuring index is required. Similar to LQR, the linear MPC problem also uses a quadratic objective function. E.g., consider the following cost function for reference tracking:

$$\begin{aligned} J_k = &\sum_{\ell=1}^{P}(x(k+\ell)-\bar{x}(k+\ell))^T Q(x(k+\ell)-\bar{x}(k+\ell)) \\ &+\sum_{\ell=1}^{M}(u(k+\ell-1)-\bar{u}(k+\ell-1))^T R(u(k+\ell-1)-\bar{u}(k+\ell-1)) \end{aligned}$$

where $\bar{x}(\cdot)$ and $\bar{u}(\cdot)$ are the reference trajectory for the state and control input, respectively. In practical situations, too severe changes in the control inputs are unwanted, as it can wear out actuators faster and/or cause disturbances. To account for this, the cost function can be modified accordingly:

$$\begin{aligned} J_k = &\sum_{\ell=1}^{P}(x(k+\ell)-\bar{x}(k+\ell))^T Q(x(k+\ell)-\bar{x}(k+\ell)) \\ &+\sum_{\ell=1}^{M}(u(k+\ell-1)-\bar{u}(k+\ell-1))^T R(u(k+\ell-1)-\bar{u}(k+\ell-1)) \\ &+\sum_{\ell=1}^{M}(\delta u(k+\ell-1))^T S(\delta u(k+\ell-1)) \end{aligned}$$

where $\delta u(k) \triangleq u(k)-u(k-1)$ is the *control rate*, and S is the *control rate weighting* matrix, which is similarly defined as Q and R from Section 2.1.2. Naturally, there are multiple ways of defining the performance measuring index J, all dependent on what objectives are to be maximized/minimized. Regardless of the exact definition of J, this formulation yields an optimization problem that can be solved with effective solution techniques.

The key attributes of the linear MPC discussed so far, are the linear prediction model, linear equality/inequality constraints, and the quadratic objective function. This optimization problem, with these specific attributes, can be solved through quadratic programming (QP).

In QP, the objective function is quadratic and the constraints are linear [20]. The general form of a QP problem is as follows:

$$\underset{x}{\text{Minimize}} \quad \frac{1}{2}x^T H x + f^T x \tag{2.22}$$

$$\text{Subject to} \quad Ax \preceq b \tag{2.23}$$

where:

- x is the solution vector.
- H is the Hessian matrix.
- f is the gradient vector.
- A is a matrix of linear constraint coefficients.
- b is a vector.

For a detailed discussion on *how* a linear MPC optimization problem is converted into the QP problem described by (2.22) and (2.23), the reader is referred to [22], Chapter 28 *Linear Model Predictive Control in the Process Industries* or [23], Chapter 1 *Model Predictive Control Basics*, Section 3 *QP Matrices*. There are several ways of solving a QP problem, among which are the algorithms: *Interior Point*, *Active Set*, *Augmented Lagrangian*, and *Conjugate Gradient*. The details about these algorithms are left to the reader, as it exceeds the scope of this project.

2.3 Nonlinear MPC

Nonlinear MPC (NMPC) follows the same principles as linear MPC, but there are some key differences that [23] points out:

(i) The prediction model can be nonlinear and include time-varying parameters.

(ii) The equality/inequality constraints can be nonlinear.

(iii) The objective/cost function to be minimized can be a nonquadratic (linear or nonlinear) function of the decision variables.

In its most general form, the NMPC problem statement can be described subsequently:

Solve the optimal control problem:

$$\begin{aligned}
&\underset{u(\cdot)}{\text{Minimize}} \quad J_k(x_0, u(\cdot)) = \sum_{\ell=k}^{k+P-1} \Upsilon(x(\ell, x_0), u(\ell)) \\
&\text{Subject to} \\
&\qquad x(k, x_0) = x_0 \\
&\qquad x(\ell+1, x_0) = f(x(\ell, x_0), u(\ell)) \\
&\qquad x(\ell) \in \mathcal{X} \\
&\qquad u(\ell) \in \mathcal{U} \\
&\qquad \delta u(\ell) \in \delta\mathcal{U}
\end{aligned}$$

Where:

- x_0 is the measured output at time step k.
- $f(\cdot,\cdot)$ is the prediction model.
- $\Upsilon(\cdot,\cdot)$ is the objective function.
- $J_k(\cdot,\cdot)$ is the total cost at time step k, given the current measured output x_0 and the P predicted control moves $u(\cdot)$.
- $\mathcal{X} \subseteq X = \mathbb{R}^{\mathrm{n}}$ is the *state constraint set.*
- $\mathcal{U} \subseteq U = \mathbb{R}^{\mathrm{m}}$ is the *control constraint set.*
- $\delta\mathcal{U} \subseteq \delta U = \mathbb{R}^{\mathrm{m}}$ is the *control rate constraint set.*

To solve the aforementioned NMPC problem, sequential quadratic programming (SQP) is used, which, in essence, is a parent problem of the QP problem for linear MPC. Solution algorithms to the NMPC SQP problem are, among others: the *Trust Region Reflective Algorithm*, the *Active Set Algorithm* and the *Interior Point Algorithm.* A detailed discussion on these algorithms can be found in e.g. [24], Chapter 10 *Numerical Optimal Control of Nonlinear Systems*, or in the MATLAB documentation, or in [25].

Chapter 3

System Modeling

As mentioned in Chapter 1, the main objective of this project is to implement MPC to the two-tank system at the UiS laboratory facility. The goal of this chapter is to provide a detailed description of the two-tank system, as well as deriving multiple mathematical models that describe its dynamics. Similar descriptions of the two-tank system can be found in [26] and [27].

3.1 System Description

The two-tank system can be illustrated as in Figure 3.1. The entire two-tank system consists of multiple components that allow for numerous functions, such as:

(i) Temperature control through a *mixer tap* or *heating element.*

(ii) Artificial delay through a *hose coil.*

(iii) Level control through *valves* or a *pump.*

This project will focus solely on level control, meaning that the mixer tap, heating element, and hose coil are not used. Excluding these redundant components yields a "sub-system" of the original two-tank system, which is shown in Figure 3.2. As see in Figure 3.2, the sub-system consists of two tanks: one rectangular shaped (Tank 1) and one square frustum shaped (Tank 2). The two tanks are connected, with the outlet of Tank 1 ($q_{LV001}(t)$) serving as the inlet for Tank 2. At the bottom, there is a collection vessel, which collects the outlet of Tank 2 ($q_{LV002}(t)$).

The two tanks are equipped with one sensor each (LT001 and LT002), which measure the fluid levels of their respective tanks. The fluid levels $h_1(t)$ and $h_2(t)$ correspond to the

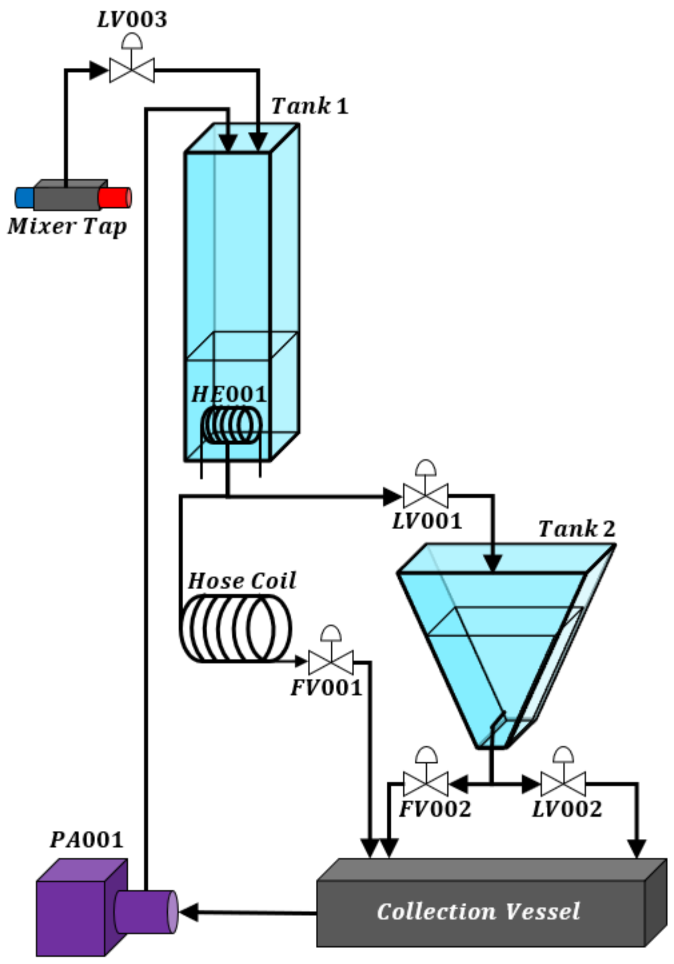


Figure 3.1: Schematic sketch of the two-tank system.

states of the system, which, in this case, will also be regarded as the *controlled variables* (CV's).

The valve actuators (LV001 and LV002) are used to control the states. This is done by manipulating the signals $u_{LV001}(t)$ and $u_{LV002}(t)$, which adjust the aperture of their respective valves. These signals are regarded as the *manipulated variables* (MV's), or, *inputs*.

The pump (PA001) moves the fluid from the collection vessel and back into Tank 1 ($q_{PA001}(t)$). The signal $u_{PA001}(t)$ controls the flow rate of the pump, and this flow rate is measured by the flow meter FT001. In this project, the inlet from the pump is regarded as a *measured disturbance* (MD) to the system.

Note also the following:

(i) Both tanks are open, meaning that the pressure on the fluid surface area is equal to the atmospheric pressure.

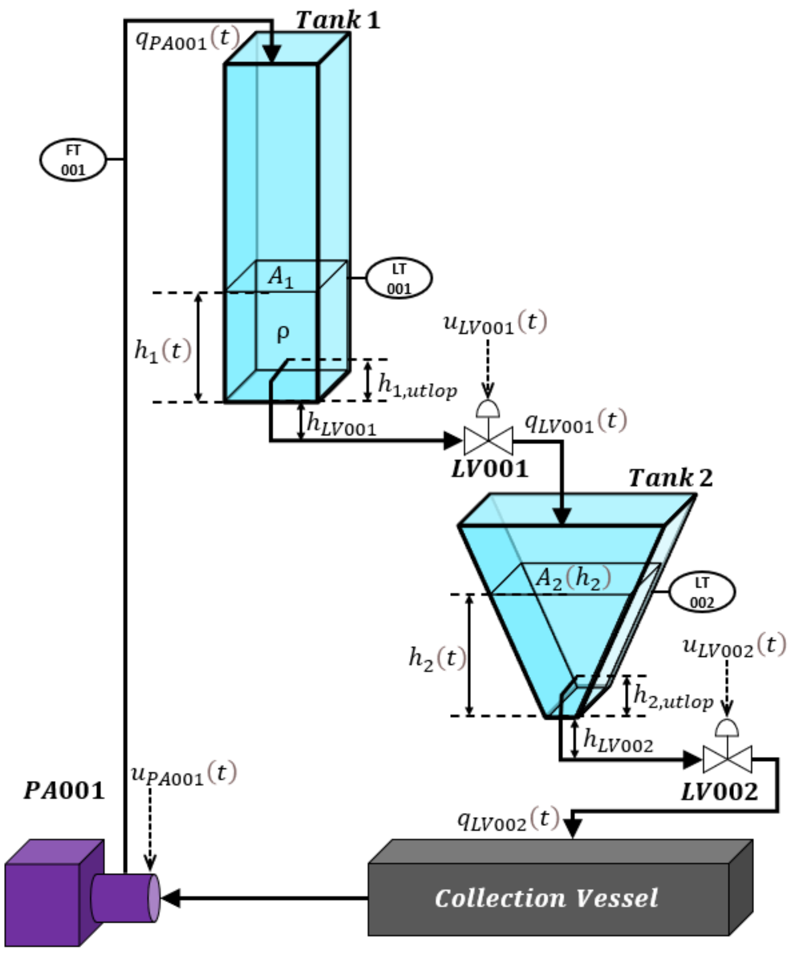


Figure 3.2: Detailed schematic sketch of the sub-system under consideration.

(ii) The two tanks cannot be fully drained due to the plumbing. Therefore, the minimum fluid levels of Tank 1 and Tank 2 are $h_{1,utlop}$ and $h_{2,utlop}$, respectively.

(iii) There is a vertical distance (h_{LV001} and h_{LV002}) between the bottom of the two tanks and their respective valves.

Prior to deriving the dynamic model for the system, additional information and assumptions regarding the pump and valves are provided below. For an easy overview of the system variables and parameters, see Table 3.1 and 3.2.

3.1.1 The Valves

A simple orifice valve is illustrated by Figure 3.3: where x, q and Δp are the flow rate percentage, volumetric outflow, and pressure drop across the valve, respectively. The volumetric outflow can be expressed mathematically as (see e.g. [21] page 48, or [20]

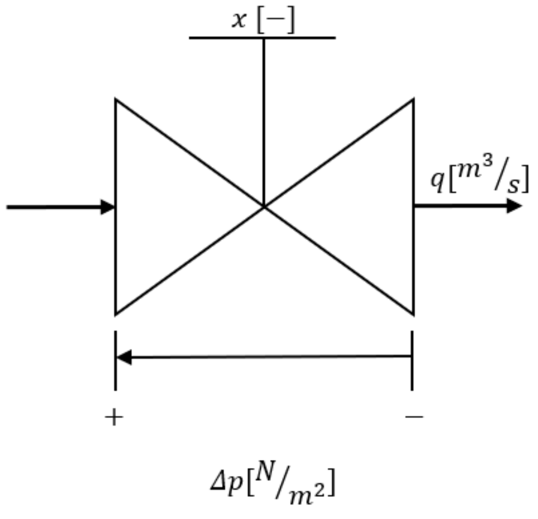


Figure 3.3: Simple sketch of an orifice valve. This sketch uses SI-units.

page 25):

$$
\begin{aligned}
q(t) &= K_v x(t)\sqrt{\Delta p(t)} \\
&= \frac{K_v x(t)\sqrt{\frac{\Delta p(t)}{100000}}}{3600} \quad [^{m^3}/_s]
\end{aligned}
\tag{3.1}
$$

where $K_v\,[\frac{m^3/h}{\sqrt{bar}}]$ is the valve constant. Note that K_v does not use SI-units, hence the conversion $Pa \rightarrow bar$ and $s \rightarrow h$ is necessary, which is done by dividing $\Delta p(t)$ with 100000 and the entire expression with 3600, respectively.

The pressure drop across the valve is simply the difference between the pressure upstream $p_+\,[^N/_{m^2}]$ and the pressure downstream $p_-\,[^N/_{m^2}]$. The pressure downstream is equivalent to one atmospheric pressure $p_a\,[^N/_{m^2}]$, since the fluid flows directly onto an open tank. The pressure upstream is equivalent to the sum of one atmospheric pressure p_a and the pressure exerted by the fluid $p_f\,[^N/_{m^2}]$. The latter can be found using Pascal's law, which states:

$$
p_f(t) = \rho g h(t) \quad [^N/_{m^2}]
\tag{3.2}
$$

with $\rho\,[^{kg}/_{m^3}]$, $g\,[^m/_{s^2}]$ and $h(t)\,[m]$ being the fluid density, gravitational acceleration, and the fluid's vertical height above the valve orifice, respectively. Following this, the pressure drop across the valve can be expressed subsequently:

$$
\begin{aligned}
\Delta p(t) &= p_+(t) - p_- \\
&= p_a + p_f(t) - p_a \\
&= p_f(t) \\
&= \rho g h(t)
\end{aligned}
\tag{3.3}
$$

Recall that, due to the plumbing, there is a small vertical height difference between the floor of the tanks and their respective valves. Therefore, $h(t)$ in (3.3) does in fact represent the sum of the fluid level in the tank and the vertical height difference between the tank and the valve. I.e.:

$$\Delta p_i(t) = \rho g(h_i(t) + h_{LV00i}), \quad i \in \{1, 2\} \tag{3.4}$$

where i denotes either valve LV001 or LV002. This will be the case for the subsequent discussion.

Next, the *flow characteristics* of the valves are considered. Flow characteristics define the relationship between the valve opening and the flow rate under constant pressure conditions [28]. Note that the term *valve opening* refers to the relative position of the valve plunge to its closed position, *not* the opening of the orifice pass area itself. The orifice pass area will always be directly proportional to the flow rate. However, the valve plunge gives different flow rates depending on its position and shape.

There are three main types of valve plunge shapes: *fast opening*, *linear*, and *equal percentage* (see [28] for a thorough review of these). The valves in the two-tank system use equal percentage plunges, which can be described mathematically as:

$$f(z(t)) = R^{z(t)-1} \tag{3.5}$$

where $f(\cdot)$, R, and $z(\cdot)$ are the flow rate percentage, rangeability, and valve opening, respectively. The rangeability is typically defined as:

$$R = \frac{\text{Flow at 95\% opening}}{\text{Flow at 5\% opening}}$$

Figure 3.4 illustrates the flow characteristics of equal percentage valves, with different rangeability: The valves in the two-tank system use $R = 10$. Note that $f(z(t)) \neq 0\%$ when $z(t) = 0\%$, meaning that the valve is slightly open when the valve plunge is completely closed[1]. As a result of this, an approximated valve characteristic is used, which takes the following form (see [26]):

$$f(z(t)) = \frac{e^{z(t)^{1.2}} - 1}{e^1 - 1} \tag{3.6}$$

such that:

$$\begin{aligned} f(0) &= \frac{e^{0^{1.2}} - 1}{e^1 - 1} = \frac{1 - 1}{e^1 - 1} = 0 \\ f(1) &= \frac{e^{1^{1.2}} - 1}{e^1 - 1} = \frac{e^1 - 1}{e^1 - 1} = 1 \end{aligned}$$

[1]Again, recall that $z(t)$ refers to the *relative position* of the valve plunge to its closed position.

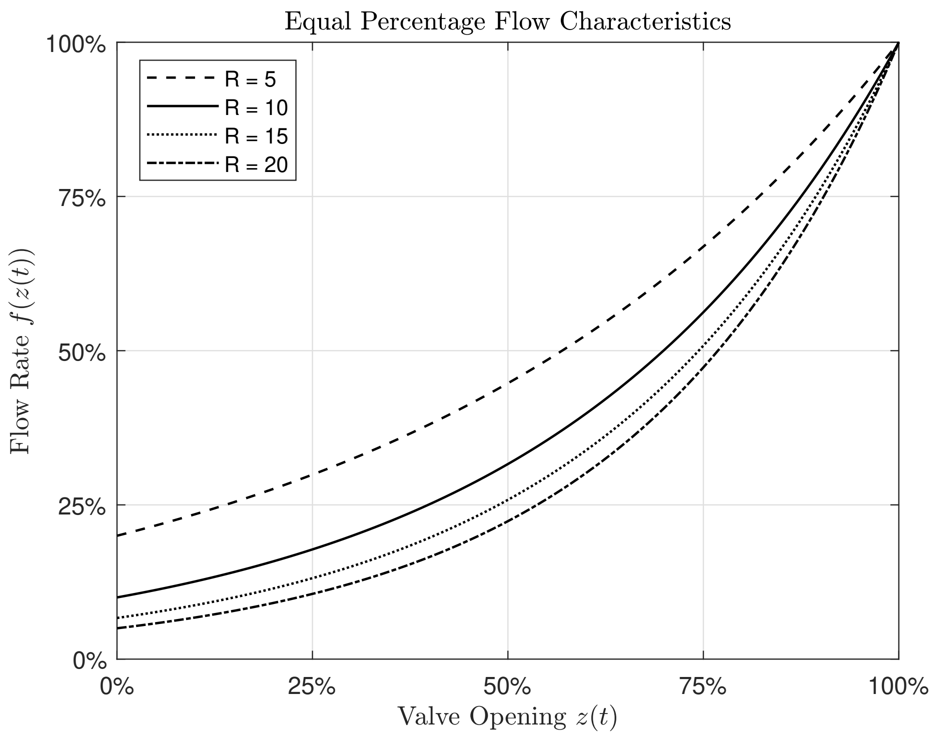


Figure 3.4: Equal percentage flow characteristics for different rangeability.

Figure 3.5 shows a direct comparison between the approximated flow characteristic (3.6), and the equal percentage characteristic (3.5) with $R = 10$: The valves in the two-tank

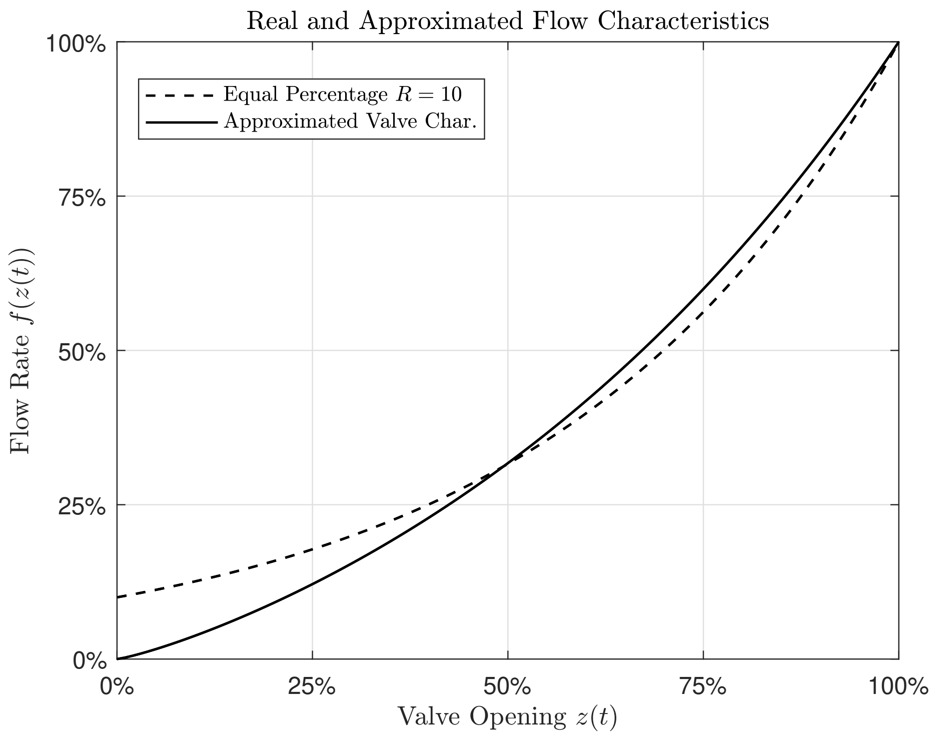


Figure 3.5: Comparison between (3.5) with $R = 10$ and (3.6).

system use pneumatic actuators to adjust the position of the valve plunges $z(t)$. These valves are *normally open* (NO), meaning that the valves will automatically open if the compressed air from the actuators disappear. The pneumatic actuators are controlled by the input signals $u_{LV00i}(t)$, $i \in \{1, 2\}$.

There is a dynamic relationship between the control signals and the plunge positioning of the two valves. However, this dynamic is considered negligible compared to the overall dynamics of the system. Therefore, it is assumed that:

$$z_i(t) = u_{LV00i}(t), \quad i \in \{1, 2\} \tag{3.7}$$

Using (3.7), the approximated flow characteristics (3.6) can be rewritten as:

$$f_i(u_{LV00i}(t)) = \frac{e^{u_{LV00i}(t)^{1.2}} - 1}{e^1 - 1}, \quad i \in \{1, 2\} \tag{3.8}$$

Finally, the volumetric outflow for each of the valves can be expressed by substituting $\Delta p(t)$ and $x(t)$ in (3.1) with (3.4) and (3.8), respectively:

$$q_i(t) = \frac{K_{v,i} f_i(u_{LV00i}(t))}{3600} \cdot \sqrt{\frac{\rho g(h_i(t) + h_{LV00i})}{100000}}, \quad i \in \{1, 2\} \tag{3.9}$$

3.1.2 The Pump

Similarly to the discussion above, the pump characteristics must also be taken into account when modeling the two-tank system. The pump characteristics define the relationship between the control signal $u_{PA001}(t)$ and the flow rate $q_{PA001}(t)$. I.e.:

$$q_{PA001}(t) = f_3(u_{PA001}(t)) \tag{3.10}$$

Normally, the pump characteristics are provided by the pump manufacturer. However, (3.10) may change depending on the environment of the pump during operations (e.g.: pipe resistance, viscosity of the fluid, temperature, etc.).

As explained by Drengstig [26], the pump characteristics can be found by making small increments of 0.05 in the control signal $u_{PA001}(t)$, and measuring the corresponding flow rate $q_{PA001}(t)$. This is done over the entire range of $u_{PA001}(t)$, which is $[0 - 1]$. The resulting pump characteristics are presented in Figure 3.6.

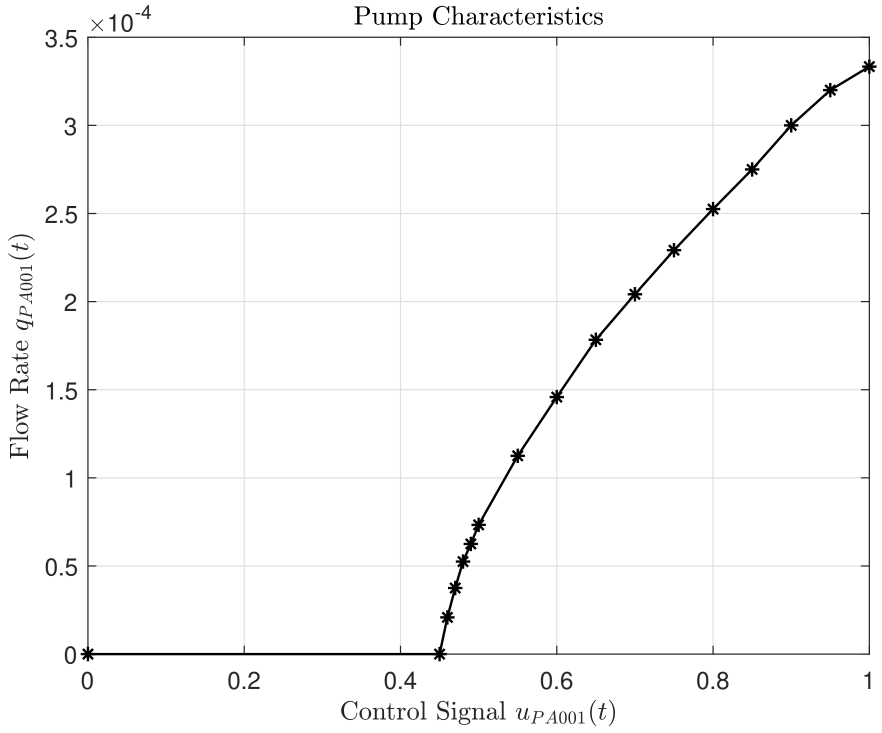


Figure 3.6: Pump characteristics (3.10) when operating in the two-tank system. The data is retrieved from [26].

Name	Description	Unit	Value
$h_1(t)$	Fluid levels of Tank 1, measure by LT001.	m	$[0-1]\,m$
A_1	Fluid surface area of Tank 1.	m^2	$0.0096\,m^2$
ρ	Fluid density (water).	kg/m^3	$1000\,kg/m^3$
$q_{PA001}(t)$	PA001 pump flow rate.	m^3/s	$[0-12]\,l/min$
$u_{PA001}(t)$	PA001 pump control signal.	$-$	$[0-1]$
$q_{LV001}(t)$	LV001 valve volumetric outflow.	m^3/s	$[0-17]\,l/min$
$u_{LV001}(t)$	LV001 valve control signal.	$-$	$[0-1]$
$K_{v,LV001}$	LV001 valve constant.	$\frac{m^3}{time\cdot\sqrt{bar}}$	$11.23\,\frac{m^3}{time\cdot\sqrt{bar}}$
h_{LV001}	Vertical distance from LV001 to Tank 1.	m	$0.05\,m$
$h_{1,utlop}$	Level of drainage (relative to $h_1(t)$).	m	$0.13\,m$
g	Gravitational acceleration (Earth).	m/s^2	$9.81\,m/s^2$

Table 3.1: Parameters and variables related to Tank 1.

System Parameters

3.2 Nonlinear Model

The dynamic behaviour of the two-tank system can be expressed by mathematical models. To achieve such models, balance laws are used. In general terms, the balance laws state that (see e.g. [21] page 14):

Name	Description	Unit	Value
$h_2(t)$	Fluid levels of Tank 2, measure by LT002.	m	$[0 - 0.4]\,m$
$A_2(h_2(t))$	Fluid surface area of Tank 2.	m^2	$[0.025 - 0.07]\,m^2$
$q_{LV002}(t)$	LV002 valve volumetric outflow.	${}^{m^3}\!/_{s}$	$[0 - 17]\,{}^{l}\!/_{min}$
$u_{LV002}(t$	LV002 valve control signal.	$-$	$[0 - 1]$
$K_{v,LV002}$	LV002 valve constant.	$\frac{m^3}{time\cdot\sqrt{bar}}$	$11.25\,\frac{m^3}{time\cdot\sqrt{bar}}$
h_{LV002}	Vertical distance from LV002 to Tank 2.	m	$0.25\,m$
$h_{2,utlop}$	Level of drainage (relative to $h_2(t)$)	m	$0.02\,m$

Table 3.2: Parameters and variables related to Tank 2.

The change of an "amount" (per time unit) in any system is equal to the net "amount" flow to the system.

where *amount* refers to a particular measurable property, such as mass, energy, momentum, entropy, charge, etc. *Net amount flow* refers to the sum of all amount inflows, minus all amount outflows, plus all amount generated in the system. Mathematically, this can be expressed as:

$$\frac{d\upsilon}{dt} = \sum \upsilon_{in} - \sum \upsilon_{out} + \sum \upsilon_{generated} \tag{3.11}$$

where υ is the measurable property of interest. In the case of the two-tank system, υ is chosen to be mass. Thus, the equation (3.11) can be rewritten as the following *mass balance law*:

$$\frac{dm(t)}{dt} = \sum w_{in}(t) - \sum w_{out}(t) \tag{3.12}$$

where $m(t)\,[kg]$, $w_{in}(t)\,[{}^{kg}\!/_{s}]$ and $w_{out}(t)\,[{}^{kg}\!/_{s}]$ are the mass, rate of mass in, and rate of mass out, respectively. Note that the two-tank system does not produce any mass, which is why (3.12) does not include a term containing $w_{generated}(t)\,[{}^{kg}\!/_{s}]$.

The equation (3.12) will be the starting point of the following discussion where the dynamic models for the two tanks are derived.

3.2.1 Tank 1 Model

Firstly, consider the physical relationship:

$$m = \rho V \tag{3.13}$$

where $m\,[kg]$, $\rho\,[{}^{kg}\!/_{m^3}]$, $V\,[m^3]$ are mass, density, and volume, respectively. Due to its rectangular shape, the volume of Tank 1 can be expressed in terms of its base area A_1 and the fluid levels $h_1(t)$. I.e.:

$$V_1(t) = A_1 h_1(t) \tag{3.14}$$

Combining (3.13) and (3.14) yields the following equation for the mass in Tank 1:

$$m_1(t) = \rho A_1 h_1(t) \tag{3.15}$$

Secondly, consider the total rate of mass into Tank 1. From Figure 3.2, it is clear that the only volumetric inflow is $q_{PA001}(t)$, which can be expressed as (3.10). Therefore:

$$\sum w_{1,in}(t) = \rho f_3(u_{PA001}(t)) \tag{3.16}$$

Note the conversion from volumetric inflow to rate of mass by use of (3.13).

Lastly, consider the total rate of mass out of Tank 1. Again, it is clear from Figure 3.2 that the total volumetric outflow is $q_{LV001}(t)$, which can be expressed as (3.9). Thus:

$$\sum w_{1,out}(t) = \rho \frac{K_{v,1} f_1(u_{LV001}(t))}{3600} \cdot \sqrt{\frac{\rho g(h_1(t) + h_{LV001})}{100000}} \tag{3.17}$$

Inserting the equations (3.15), (3.16), and (3.17) into (3.12), and rearranging for $h_1(t)$, yields the following differential equation:

$$\frac{dh_1(t)}{dt} = \frac{1}{A_1}\left(f_3(u_{PA001}(t)) - \frac{K_{v,1} f_1(u_{LV001}(t))}{3600} \cdot \sqrt{\frac{\rho g(h_1(t) + h_{LV001})}{100000}} \right) \tag{3.18}$$

3.2.2 Tank 2 Model

Now the theoretical model for Tank 2 is derived. Again, the equation (3.12) serves as the starting point of the modeling procedure.

Firstly, assuming that the system is isolated, and no fluid leaves the system during operation, then, clearly, the total rate of mass into Tank 2 is equal to the total rate of mass out of Tank 1 (3.17). I.e.:

$$\sum w_{2,in}(t) = \sum w_{1,out}(t) = \rho \frac{K_{v,1} f_1(u_{LV001}(t))}{3600} \cdot \sqrt{\frac{\rho g(h_1(t) + h_{LV001})}{100000}} \tag{3.19}$$

Secondly, the total volumetric outflow of Tank 2 is $q_{LV002}(t)$ (see Figure 3.2), which can be expressed as (3.9). Using the relationship (3.13) with (3.9) gives the following total rate of mass out from Tank 2:

$$\sum w_{2,out}(t) = \rho \frac{K_{v,2} f_2(u_{LV002}(t))}{3600} \cdot \sqrt{\frac{\rho g(h_2(t) + h_{LV002})}{100000}} \tag{3.20}$$

Lastly, consider the total mass in Tank 2, which can be expressed as (3.13). The volume is given by:

$$V_2(t) = A_2(h_2(t))h_2(t) \tag{3.21}$$

Inserting (3.21) in (3.13) gives:

$$m_2(t) = \rho A_2(h_2(t))h_2(t) \tag{3.22}$$

Inserting the equations (3.19), (3.20), and (3.22) into (3.12); and rearranging for $h_2(t)$, results in the following differential equation:

$$\begin{aligned}\frac{dh_2(t)}{dt} = \frac{1}{A_2(h_2(t))} \cdot \Bigg(&\frac{K_{v,1} f_1(u_{LV001}(t))}{3600} \cdot \sqrt{\frac{\rho g(h_1(t) + h_{LV001})}{100000}} \\ &- \frac{K_{v,2} f_2(u_{LV002}(t))}{3600} \cdot \sqrt{\frac{\rho g(h_2(t) + h_{LV002})}{100000}} \Bigg)\end{aligned} \tag{3.23}$$

Finding an expression for $A_2(\cdot)$ proves to be not that trivial, so a detailed description of this is provided below. Figure 3.7 and Table 3.3 are used as a basis for the following calculations.

As a start, Tank 2 is divided into three segments: one cuboid, and two equal triangular prisms (see Figure 3.7).

The fluid surface area can be expressed as:

$$A_2(h_2(t)) = A_{2,0} + 2A_{tp}(h_2(t)) \tag{3.24}$$

where $A_{2,0}$ and $A_{tp}(\cdot)$ are the fluid surface area of the cuboid and the triangular prism, respectively. $A_{2,0}$ is a constant with a value of $0.004\, m^2$ (see Table 3.3).

The fluid surface area of the triangular prism can be expressed as:

$$A_{tp}(h_2(t)) = d_2 \cdot \nu_2(h_2(t)) \tag{3.25}$$

where d_2 and $\nu_2(h_2(t))$ are described in Table 3.3. By use of triangular similarity (TS), a mathematical expression for $\nu_2(h_2(t))$ can be found:

$$\begin{gathered}\frac{\nu_2(h_2(t))}{h_2(t)} \overset{\text{TS}}{=} \frac{\nu_{2,max}}{h_{2,max}} \\ \therefore \\ \nu_2(h_2(t)) = h_2(t) \cdot \frac{\nu_{2,max}}{h_{2,max}}\end{gathered} \tag{3.26}$$

Combining (3.26) with (3.25); and inserting the resulting expression in (3.24), yields:

$$\begin{aligned} A_2(h_2(t)) &= A_{2,0} + 2 \cdot d_2 \cdot \frac{\nu_{2,max}}{h_{2,max}} \cdot h_2(t) \\ &= 0.004 + 2 \cdot 0.08 \cdot \frac{0.175}{0.4} \cdot h_2(t) \\ &= 0.004 + 0.07 \cdot h_2(t) \end{aligned} \tag{3.27}$$

Inserting (3.27) in (3.23) results in the differential equation:

$$\begin{aligned} \frac{dh_2(t)}{dt} = \frac{1}{0.004 + 0.07 \cdot h_2(t)} \cdot \Bigg(& \frac{K_{v,1} f_1(u_{LV001}(t))}{3600} \cdot \sqrt{\frac{\rho g(h_1(t) + h_{LV001})}{100000}} \\ & - \frac{K_{v,2} f_2(u_{LV002}(t))}{3600} \cdot \sqrt{\frac{\rho g(h_2(t) + h_{LV002})}{100000}} \Bigg) \end{aligned} \tag{3.28}$$

Finally, the two-tank system can be expressed by the following nonlinear state-space representation:

$$\begin{bmatrix} \dot{h}_1(t) \\ \dot{h}_2(t) \end{bmatrix} = \begin{bmatrix} \mathfrak{f}_1(h_1(t),\, u_{LV001}(t),\, u_{PA001}(t)) \\ \mathfrak{f}_2(h_1(t),\, h_2(t),\, u_{LV001}(t),\, u_{LV002}(t)) \end{bmatrix} \tag{3.29}$$

where $\mathfrak{f}_1(\cdot,\cdot,\cdot)$ and $\mathfrak{f}_2(\cdot,\cdot,\cdot,\cdot)$ are the functions in the state equations (3.18) and (3.28), respectively.

In this project, it is assumed that the level transmitters LT001 and LT002 measure the states perfectly, with no feedthrough. Thus, the outputs can be expressed as:

$$\begin{bmatrix} \hat{h}_1(t) \\ \hat{h}_2(t) \end{bmatrix} = \begin{bmatrix} \mathfrak{g}_1(h_1(t),\, u_{LV001}(t),\, u_{PA001}(t)) \\ \mathfrak{g}_2(h_1(t),\, h_2(t),\, u_{LV001}(t),\, u_{LV002}(t)) \end{bmatrix} \tag{3.30}$$

where $\mathfrak{g}_1(\cdot,\ \cdot,\ \cdot)$ and $\mathfrak{g}_2(\cdot,\ \cdot,\ \cdot,\ \cdot)$ are the output equations, given by:

$$\mathfrak{g}_1(h_1(t),\, u_{LV001}(t),\, u_{PA001}(t)) = h_1(t) \tag{3.31}$$

$$\mathfrak{g}_2(h_1(t),\, h_2(t),\, u_{LV001}(t),\, u_{LV002}(t)) = h_2(t) \tag{3.32}$$

3.3 Linear Model

Theoretical models for most real physical systems include some degree of nonlinearity. Nonlinear models often describe physical systems with high fidelity, but they are also difficult to handle. On the other hand, linear models are easy to solve, and there are

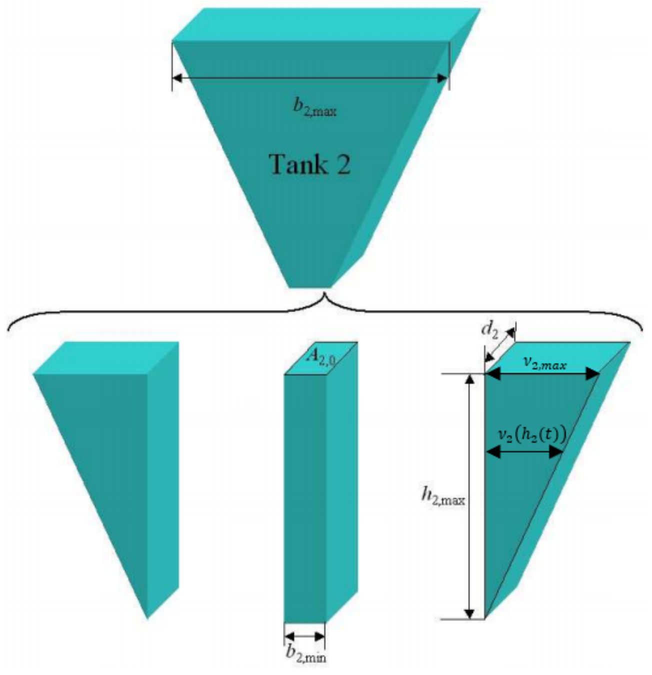


Figure 3.7: Cross section of Tank 2. [26]

Name	**Description**	**Value**
$A_{2,0}$	Area of the cuboid.	0.004 $[m^2]$
d_2	Depth of Tank 2.	0.08 $[m]$
$b_{2,max}$	Upper width of Tank 2.	0.4 $[m]$
$b_{2,min}$	Lower width of Tank 2.	0.05 $[m]$
$h_{2,max}$	Maximum height of Tank 2.	0.4 $[m]$
$\nu_{2,max}$	Maximum width of the triangular prism.	0.175 $[m]$
$\nu_2(h_2(t))$	Width of the triangular prism at height $h_2(t)$.	N/A

Table 3.3: Dimensions of Tank 2.

multiple analysis and design methods developed specifically for linear systems, such as control design, stability analysis, and frequency response [21].

A common engineering practise is to obtain a linear model from the nonlinear model by means of *linearization*. The term linearization is neatly defined by Levine [22] (page 955) as:

> 'Approximation of the nonlinear state equation in the vicinity of a nominal solution by a linear state equation, obtained by dropping second- and higher-order terms of the Taylor expansion (about the nominal solution) of the right-hand side function.'

To understand the definition above, firstly, consider the following nonlinear equation:

$$y = f(x) \tag{3.33}$$

Assuming that $f(x)$ has derivatives of all orders at $x = a$, then, $f(x)$ can be represented by its corresponding Taylor series[2] (see [29] page 543):

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k \tag{3.34}$$

To linearize $f(x)$, the second- and higher- order terms in (3.34) (i.e., terms where $k = 2, 3, 4, \dots$) are dropped, which gives:

$$L(x) = f(a) + f'(a)(x-a) \tag{3.35}$$

where $L(x)$ provides linear approximations for values of f near a [29].

The equation (3.34) can be generalized to functions of more than one variable. For instance, consider the function:

$$y = f(x_1, x_2, \dots, x_d) \tag{3.36}$$

Again, assuming that $f(x_1, x_2, \dots, x_d)$ is continuously differentiable at $x_1 = a_1$, $x_2 = a_2$, ..., $x_d = a_d$, then, $f(x_1, x_2, \dots, x_d)$ can be represented by its corresponding Taylor series (see e.g. [30] Chapter 6 or [27] page 16):

$$\begin{aligned} f(x_1, x_2, \dots, x_d) = f(a_1, \dots, a_d) &+ \sum_{j=1}^{d} \frac{\partial f(a_1, \dots, a_d)}{\partial x_j}(x_j - a_j) \\ &+ \frac{1}{2!}\sum_{j=1}^{d}\sum_{k=1}^{d} \frac{\partial^2 f(a_1, \dots, a_d)}{\partial x_j \partial x_k}(x_j - a_j)(x_k - a_k) \\ &+ \frac{1}{3!}\sum_{j=1}^{d}\sum_{k=1}^{d}\sum_{l=1}^{d} \frac{\partial^3 f(a_1, \dots, a_d)}{\partial x_j \partial x_k \partial x_l}(x_j - a_j)(x_k - a_k)(x_l - a_l) \\ &+ \dots \end{aligned} \tag{3.37}$$

Linearizing (3.36), i.e., dropping the second- and higher- order terms in (3.37), yields the following:

$$L(x_1, \dots, x_d) = f(a_1, \dots, a_d) + \sum_{j=1}^{d} \frac{\partial f(a_1, \dots, a_d)}{\partial x_j}(x_j - a_j) \tag{3.38}$$

where $L(x_1, \dots, x_d)$ provides linear approximations for values of f near $a_1, \dots, a_d$.

Now consider the nonlinear state-space representation of the two-tank system expressed by (3.29)[3]. A linear approximation of the state equations $\mathfrak{f}_1(\cdot,\cdot,\cdot)$ and $\mathfrak{f}_2(\cdot,\cdot,\cdot,\cdot)$ (in the vicinity of a nominal solution denoted by a tilde) is obtained by applying the general

[2]Taylor *series* and Taylor *expansion* are used interchangeably in this project.

[3]Note that the output equations (3.30) already are linear, which is why they are not considered in this section.

formula (3.38):

$$\begin{aligned}\dot{h}_1(t) \approx \quad & \mathfrak{f}_1(\tilde{h}_1(t), \tilde{u}_{LV001}(t), \tilde{u}_{PA001}(t)) \\ + \quad & \left.\frac{\partial \mathfrak{f}_1}{\partial h_1}\right|_{\substack{\tilde{h}_1 \\ \tilde{u}_{LV001} \\ \tilde{u}_{PA001}}} (h_1(t) - \tilde{h}_1(t)) \\ + \quad & \left.\frac{\partial \mathfrak{f}_1}{\partial u_{LV001}}\right|_{\substack{\tilde{h}_1 \\ \tilde{u}_{LV001} \\ \tilde{u}_{PA001}}} (u_{LV001}(t) - \tilde{u}_{LV001}(t)) \\ + \quad & \left.\frac{\partial \mathfrak{f}_1}{\partial u_{PA001}}\right|_{\substack{\tilde{h}_1 \\ \tilde{u}_{LV001} \\ \tilde{u}_{PA001}}} (u_{PA001}(t) - \tilde{u}_{PA001}(t))\end{aligned} \tag{3.39}$$

$$\begin{aligned}\dot{h}_2(t) \approx \quad & \mathfrak{f}_2(\tilde{h}_1(t), \tilde{h}_2(t), \tilde{u}_{LV001}(t), \tilde{u}_{LV002}(t)) \\ + \quad & \left.\frac{\partial \mathfrak{f}_2}{\partial h_1}\right|_{\substack{\tilde{h}_1 \\ \tilde{h}_2 \\ \tilde{u}_{LV001} \\ \tilde{u}_{LV002}}} (h_1(t) - \tilde{h}_1(t)) \\ + \quad & \left.\frac{\partial \mathfrak{f}_2}{\partial h_2}\right|_{\substack{\tilde{h}_1 \\ \tilde{h}_2 \\ \tilde{u}_{LV001} \\ \tilde{u}_{LV002}}} (h_2(t) - \tilde{h}_2(t)) \\ + \quad & \left.\frac{\partial \mathfrak{f}_2}{\partial u_{LV001}}\right|_{\substack{\tilde{h}_1 \\ \tilde{h}_2 \\ \tilde{u}_{LV001} \\ \tilde{u}_{LV002}}} (u_{LV001}(t) - \tilde{u}_{LV001}(t)) \\ + \quad & \left.\frac{\partial \mathfrak{f}_2}{\partial u_{LV002}}\right|_{\substack{\tilde{h}_1 \\ \tilde{h}_2 \\ \tilde{u}_{LV001} \\ \tilde{u}_{LV002}}} (u_{LV002}(t) - \tilde{u}_{LV002}(t))\end{aligned} \tag{3.40}$$

The states and inputs can be redefined with respect to their nominal values. I.e.:

$$h_1(t) = \tilde{h}_1(t) + \Delta h_1(t) \tag{3.41}$$
$$h_2(t) = \tilde{h}_2(t) + \Delta h_2(t) \tag{3.42}$$
$$u_{LV001}(t) = \tilde{u}_{LV001}(t) + \Delta u_{LV001}(t) \tag{3.43}$$
$$u_{LV002}(t) = \tilde{u}_{LV002}(t) + \Delta u_{LV002}(t) \tag{3.44}$$
$$u_{PA001}(t) = \tilde{u}_{PA001}(t) + \Delta u_{PA001}(t) \tag{3.45}$$
$$\hat{h}_1(t) = \tilde{\hat{h}}_1(t) + \Delta \hat{h}_1(t) \tag{3.46}$$
$$\hat{h}_2(t) = \tilde{\hat{h}}_2(t) + \Delta \hat{h}_2(t) \tag{3.47}$$

where the delta-variables (denoted with a Δ in front) represent small deviations from their respective nominal value (denoted with a tilde). Consider the following:

(i) Equation (3.41) and (3.42) are to be inserted in the left-hand-side of (3.39) and (3.40), respectively. As a consequence, the state equations $\mathfrak{f}_1$ and $\mathfrak{f}_2$ (evaluated at the nominal solution) cancels out with the derivative of the nominal state $\dot{\tilde{h}}_1(t)$ and $\dot{\tilde{h}}_2(t)$, respectively.

(ii) Following every partial derivative in (3.39) and (3.40) is the difference between the real state/input and the nominal state/input. These differences can be expressed in terms of the delta-variables, as defined by (3.41) - (3.45).

Applying the observations mentioned above to (3.39) and (3.40) results in the following linear approximations in a neighborhood of the nominal solution:

$$\begin{aligned}\Delta\dot{h}_1(t) = {} & \left.\frac{\partial\mathfrak{f}_1}{\partial h_1}\right|_{\substack{\tilde{h}_1\\ \tilde{u}_{LV001}\\ \tilde{u}_{PA001}}}\Delta h_1(t)\\ & + \left.\frac{\partial\mathfrak{f}_1}{\partial u_{LV001}}\right|_{\substack{\tilde{h}_1\\ \tilde{u}_{LV001}\\ \tilde{u}_{PA001}}}\Delta u_{LV001}(t)\\ & + \left.\frac{\partial\mathfrak{f}_1}{\partial u_{PA001}}\right|_{\substack{\tilde{h}_1\\ \tilde{u}_{LV001}\\ \tilde{u}_{PA001}}}\Delta u_{PA001}(t)\end{aligned} \tag{3.48}$$

$$\begin{aligned}\Delta\dot{h}_2(t) = {} & \left.\frac{\partial\mathfrak{f}_2}{\partial h_1}\right|_{\substack{\tilde{h}_1\\ \tilde{h}_2\\ \tilde{u}_{LV001}\\ \tilde{u}_{LV002}}}\Delta h_1(t)\\ & + \left.\frac{\partial\mathfrak{f}_2}{\partial h_2}\right|_{\substack{\tilde{h}_1\\ \tilde{h}_2\\ \tilde{u}_{LV001}\\ \tilde{u}_{LV002}}}\Delta h_2(t)\\ & + \left.\frac{\partial\mathfrak{f}_2}{\partial u_{LV001}}\right|_{\substack{\tilde{h}_1\\ \tilde{h}_2\\ \tilde{u}_{LV001}\\ \tilde{u}_{LV002}}}\Delta u_{LV001}(t)\\ & + \left.\frac{\partial\mathfrak{f}_2}{\partial u_{LV002}}\right|_{\substack{\tilde{h}_1\\ \tilde{h}_2\\ \tilde{u}_{LV001}\\ \tilde{u}_{LV002}}}\Delta u_{LV002}(t)\end{aligned} \tag{3.49}$$

which can be transformed into the compact linear state-space representation:

$$\begin{aligned}\begin{bmatrix}\Delta\dot{h}_1(t)\\ \Delta\dot{h}_2(t)\end{bmatrix} = {} & \begin{bmatrix}\frac{\partial\mathfrak{f}_1}{\partial h_1} & \frac{\partial\mathfrak{f}_1}{\partial h_2}\\ \frac{\partial\mathfrak{f}_2}{\partial h_1} & \frac{\partial\mathfrak{f}_2}{\partial h_2}\end{bmatrix}_{\mathfrak{NS}} \cdot \begin{bmatrix}\Delta h_1(t)\\ \Delta h_2(t)\end{bmatrix} + \begin{bmatrix}\frac{\partial\mathfrak{f}_1}{\partial u_{LV001}} & \frac{\partial\mathfrak{f}_1}{\partial u_{LV002}}\\ \frac{\partial\mathfrak{f}_2}{\partial u_{LV001}} & \frac{\partial\mathfrak{f}_2}{\partial u_{LV002}}\end{bmatrix}_{\mathfrak{NS}} \cdot \begin{bmatrix}\Delta u_{LV001}(t)\\ \Delta u_{LV002}(t)\end{bmatrix}\\ & + \begin{bmatrix}\frac{\partial\mathfrak{f}_1}{\partial u_{PA001}}\\ \frac{\partial\mathfrak{f}_2}{\partial u_{PA001}}\end{bmatrix}_{\mathfrak{NS}} \cdot \Delta u_{PA001}(t)\end{aligned} \tag{3.50}$$

where the subscript notation $\mathfrak{NS}$[4] implies that the partial derivatives in (3.50) are evaluated at the arbitrary nominal solution. I.e.: $h_1(t) = \tilde{h}_1$, $h_2(t) = \tilde{h}_2$, $u_{LV001}(t) = \tilde{u}_{LV001}$, $u_{LV002}(t) = \tilde{u}_{LV002}$, and $u_{PA001}(t) = \tilde{u}_{PA001}$.

The corresponding output equations (using (3.46) - (3.47)) are:

$$\begin{bmatrix}\Delta\hat{h}_1(t)\\ \Delta\hat{h}_2(t)\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \cdot \begin{bmatrix}\Delta h_1(t)\\ \Delta h_2(t)\end{bmatrix} \tag{3.51}$$

[4]Short for *Nominal solution* (NS) in fraktur font.

The partial derivatives in (3.50) (evaluated at $\mathfrak{NS}$) are calculated to be:

$$\frac{\partial \mathfrak{f}_1}{\partial h_1} = -\frac{\sqrt{100000}}{7.2\cdot 10^8}\cdot\frac{K_{v,1}f_1(\tilde{u}_{LV001})\rho g}{A_1\sqrt{\rho g(\tilde{h}_1+h_{LV001})}} \tag{3.52}$$

$$\frac{\partial \mathfrak{f}_1}{\partial h_2} = 0 \tag{3.53}$$

$$\frac{\partial \mathfrak{f}_2}{\partial h_1} = \frac{1}{0.004+0.07\cdot\tilde{h}_2}\cdot\frac{\sqrt{100000}}{7.2\cdot 10^8}\cdot\frac{K_{v,1}f_1(\tilde{u}_{LV001})\rho g}{\sqrt{\rho g(\tilde{h}_1+h_{LV001})}} \tag{3.54}$$

$$\frac{\partial \mathfrak{f}_2}{\partial h_2} = -\frac{1}{0.004+0.07\cdot\tilde{h}_2}\cdot\frac{\sqrt{100000}}{7.2\cdot 10^8}\cdot\frac{K_{v,2}f_2(\tilde{u}_{LV002})\rho g}{\sqrt{\rho g(\tilde{h}_2+h_{LV002})}} \tag{3.55}$$

$$\frac{\partial \mathfrak{f}_1}{\partial u_{LV001}} = -\frac{K_{v,1}}{3600A_1}\cdot\sqrt{\frac{\rho g(\tilde{h}_1+h_{LV001})}{100000}}\cdot\frac{df_1}{du_{LV001}} \tag{3.56}$$

$$\frac{\partial \mathfrak{f}_1}{\partial u_{LV002}} = 0 \tag{3.57}$$

$$\frac{\partial \mathfrak{f}_2}{\partial u_{LV001}} = \frac{1}{0.004+0.07\cdot\tilde{h}_2}\cdot\frac{K_{v,1}}{3600}\cdot\sqrt{\frac{\rho g(\tilde{h}_1+h_{LV001})}{100000}}\cdot\frac{df_1}{du_{LV001}} \tag{3.58}$$

$$\frac{\partial \mathfrak{f}_2}{\partial u_{LV002}} = -\frac{1}{0.004+0.07\cdot\tilde{h}_2}\cdot\frac{K_{v,2}}{3600}\cdot\sqrt{\frac{\rho g(\tilde{h}_2+h_{LV002})}{100000}}\cdot\frac{df_2}{du_{LV002}} \tag{3.59}$$

$$\frac{\partial \mathfrak{f}_1}{\partial u_{PA001}} = \frac{1}{A_1}\cdot\frac{df_3}{du_{PA001}} \tag{3.60}$$

$$\frac{\partial \mathfrak{f}_2}{\partial u_{PA001}} = 0 \tag{3.61}$$

Note that the variables u_{LV00i}, $i\in\{1,2\}$ and u_{PA001} do not appear directly in the state equations $\mathfrak{f}_1$ and $\mathfrak{f}_2$. In fact, $\mathfrak{f}_1$ and $\mathfrak{f}_2$ depend on the intermediate functions $f_1(u_{LV001})$, $f_2(u_{LV002})$, and $f_3(u_{PA001})$. Therefore, the chain rule is applied to find (3.56), (3.58), (3.59), and (3.60), respectively.

Finding the derivative of the intermediate functions f_1, f_2, and f_3, with respect to their independent variables, can be done numerically by use of the forward difference. The forward difference is defined as (see e.g. [31] page 41-44):

$$f'(x) = \frac{f(x+h)-f(x)}{h} \tag{3.62}$$

where h is the step size.

A graphical interpretation of (3.62) (with respect to f_1, f_2, and f_3) can be seen in Figure 3.8 and Figure 3.9. In Figure 3.8, the slope of $\mathcal{L}_v$ represents $f'_i(\tilde{u}_{LV00i})$, $i\in\{1,2\}$ evaluated at $\{\tilde{u}_{LV00i}=0.5\}\in\mathfrak{NS}$. Similarly, the slope of $\mathcal{L}_p$ in Figure 3.9 depicts $f'_3(\tilde{u}_{PA001})$ evaluated at $\{\tilde{u}_{PA001}=0.65\}\in\mathfrak{NS}$. The step size $h=0.1$ is used in Figure 3.8 and 3.9 for illustration purposes, however, $h=0.01$ is used for the remainder of this project.

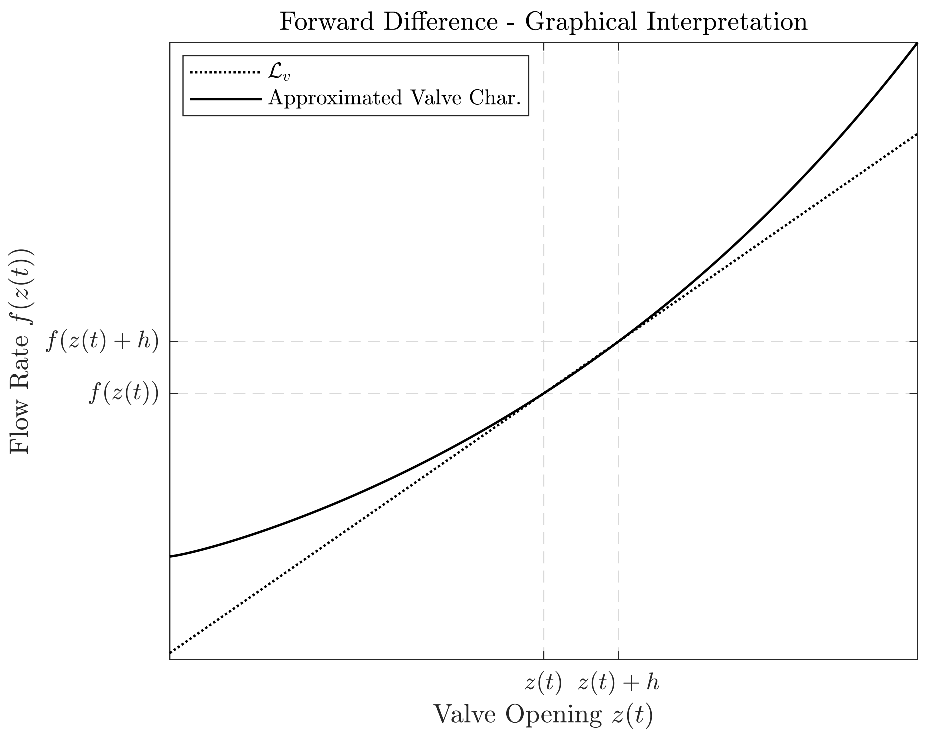


Figure 3.8: Graphical interpretation of (3.62) for f_1 and f_2. Recall that $z_i(t)$ and $u_{LV00i}(t)$, $i \in \{1, 2\}$ can be used interchangeably due to assumption (3.7).

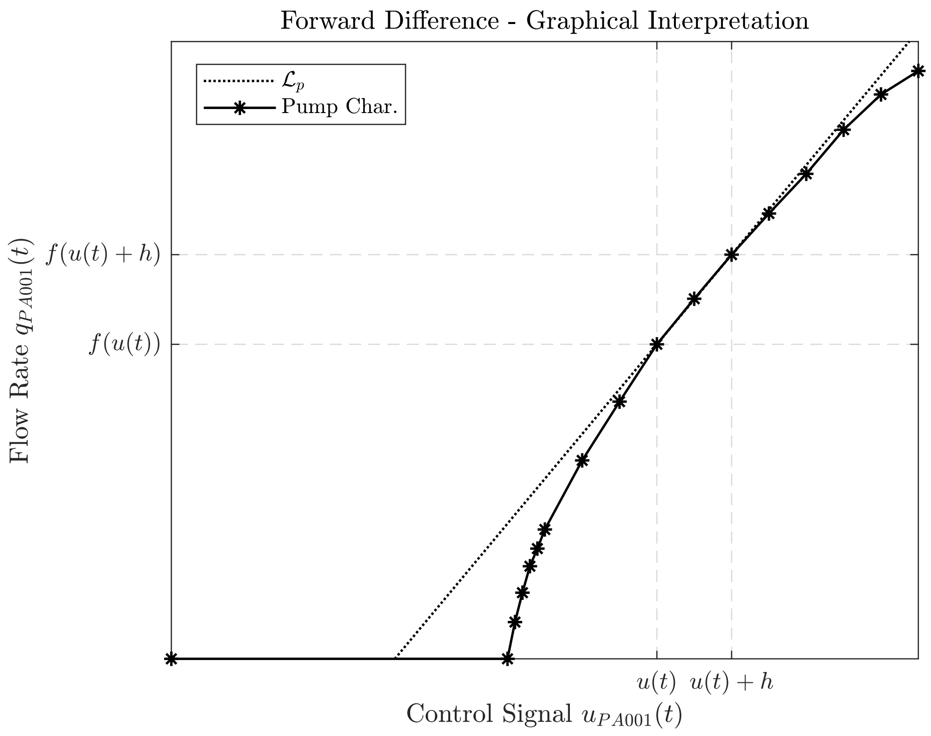


Figure 3.9: Graphical interpretation of (3.62) for f_3. To increase readability, $u_{PA001}(t)$ is abbreviated to just $u(t)$ for this particular figure.

3.4 Analysis of the Linear Model

As an example, assume that the nominal solution is:

$$\underbrace{\tilde{h}_1 = 0.5, \quad \tilde{h}_2 = 0.3, \quad \tilde{u}_{PA001} = 0.8, \quad \tilde{u}_{LV001} = 0.5317, \quad \tilde{u}_{LV002} = 0.5317}_{\mathfrak{NS}}$$

which, in fact, represents an equilibrium point of the state equations $\mathfrak{f}_1$ and $\mathfrak{f}_2$. This can be verified by inserting $\mathfrak{NS}$ in the state equations, resulting in ≈ 0 for both cases.

A linear model (about $\mathfrak{NS}$) can be found by, firstly, calculating the partial derivatives (3.52) - (3.61) (inserted the nominal solution), and, secondly, arranging the calculated partial derivatives as in the state-space representation model (3.50). I.e.:

$$\begin{cases} \begin{bmatrix} \Delta\dot{h}_1(t) \\ \Delta\dot{h}_2(t) \end{bmatrix} = \begin{bmatrix} -0.0230 & 0 \\ 0.0092 & -0.0092 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(t) \\ \Delta h_2(t) \end{bmatrix} + \begin{bmatrix} -0.0719 & 0 \\ 0.0288 & -0.0288 \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(t) \\ \Delta u_{LV002}(t) \end{bmatrix} \\ \qquad + \begin{bmatrix} 0.0450 \\ 0 \end{bmatrix} \cdot \Delta u_{PA001}(t) \\ \begin{bmatrix} \Delta\hat{h}_1(t) \\ \Delta\hat{h}_2(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(t) \\ \Delta h_2(t) \end{bmatrix} \end{cases} \tag{3.63}$$

A direct comparison between the linear approximation (3.63) and the nonlinear model (3.29) can be done in Simulink. To compare the two models, small perturbations are exerted on the pump and valve input signals, respectively. The starting point for the simulation is the nominal solution $\mathfrak{NS}$. Figure 3.10 shows the results of this simulation, while Appendix J.1 shows the Simulink block diagram. Note that, as the models drift farther away from the nominal solution $\mathfrak{NS}$, the linear model deviates increasingly more from the nonlinear model. However, the linear model strongly captures the dynamic behavior of the nonlinear model when operating close to the nominal solution $\mathfrak{NS}$.

Remark 3.1. Note that Tank 2, in Figure 3.10, exceeds its possible fluid levels of $0.4m$ at approximately $200 \leq Time \leq 400$. Although this would not be physically possible on the real system, it is purposely allowed for this particular simulation/comparison.

3.4.1 Stability

The stability of the *unforced part* in (3.63) can be determined by analysing the eigenvalues of the state matrix. By unforced part, it is implied that $\Delta u_{LV001}(t) = \Delta u_{LV002}(t) = \Delta u_{PA001}(t) = 0$. The eigenvalues can be found by solving the following characteristic equation (see [32] Chapter 6):

$$|A - \lambda I| = 0 \tag{3.64}$$

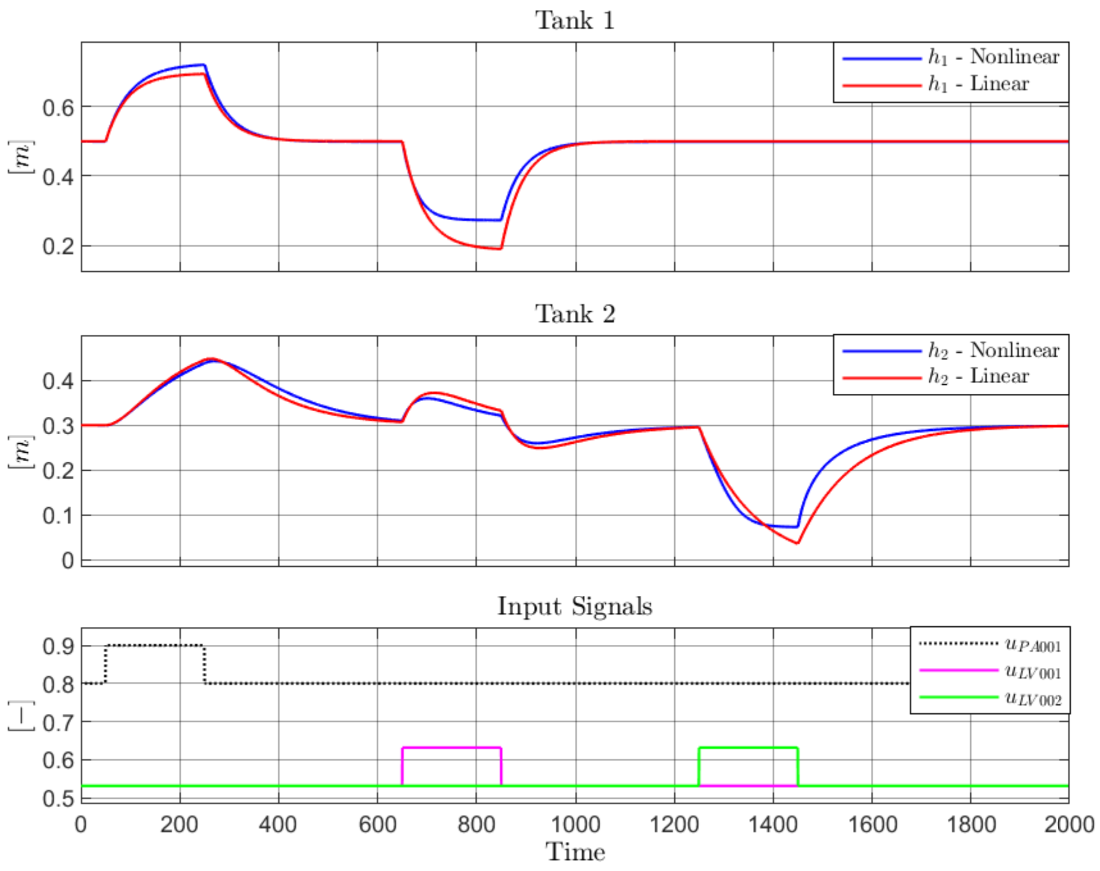


Figure 3.10: Simulation of the nonlinear- and linear- model with perturbations in the input signals.

where A, λ, and I are the state matrix, eigenvalue, and identity matrix, respectively.

Using (3.64) with the state matrix in (3.63) results in the following:

$$\begin{aligned}
\left| \begin{bmatrix} -0.0230 & 0 \\ 0.0092 & -0.0092 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right| &= 0 \\
\left| \begin{bmatrix} -0.0230 - \lambda & 0 \\ 0.0092 & -0.0092 - \lambda \end{bmatrix} \right| &= 0 \\
(-0.0230 - \lambda) \cdot (-0.0092 - \lambda) &= 0 \\
\lambda^2 + 0.00322\lambda + 2.116 \cdot 10^{-4} &= 0 \qquad (3.65)
\end{aligned}$$

Solving (3.65) with the quadratic formula (see e.g. [29] page 42) yields the corresponding eigenvalues:

$$\lambda_1 = -0.0092 \quad \lambda_2 = -0.023 \qquad (3.66)$$

Both of the eigenvalues λ_1 and λ_2 appear on the left-hand side of the complex plane. From the control theory (see e.g. [14] Section 4.4) this means that the unforced part of system (3.63) is *asymptotically stable*, i.e., $\Delta h(t) \rightarrow 0$ as $t \rightarrow \infty$.

Additionally, the eigenvalues are both on the real axis ($Im(\lambda_i) = 0,\ i \in \{1, 2\}$), effectively classifying the nominal solution $\mathfrak{NS}$ as an asymptotically stable *node*, which can be graphically illustrated in the phase plane as in Figure 3.11.

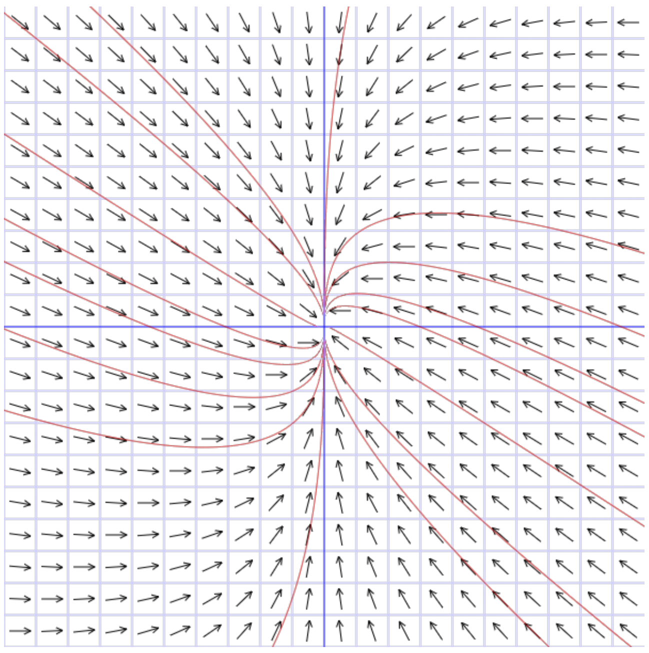


Figure 3.11: Phase plane of the unforced part in (3.63). The horizontal and vertical axes represent $\Delta h_1(t)$ and $\Delta h_2(t)$, respectively. The origin (with respect to the real system variables $h_i(t),\ i \in \{1, 2\}$) represents the nominal solution $\mathfrak{NS}$. John C. Polking of Rice University is acknowledged for providing the MATLAB code which produced this image.

3.4.2 Laplace Transform

The linear model (3.63) can be further analysed by studying its Laplace Transform. The Laplace Transform is a linear operator which is defined as (see e.g. [20] Chapter 3 or [33] Chapter 6):

$$F(s) = \mathscr{L}\{f(t)\} = \int_0^\infty f(t)e^{-st}\,dt \tag{3.67}$$

where

- $f(t)$ is an arbitrary function,
- $F(s)$ is the Laplace Transform of $f(t)$,
- s is a complex independent variable,
- and $\mathscr{L}\{\cdot\}$ is the Laplace Transform operator.

First, consider the Laplace Transform of the linear model for Tank 1:

$$\mathscr{L}\left\{\Delta\dot{h}_1(t)\right\} = \mathscr{L}\left\{-0.0230\Delta h_1(t) - 0.0719\Delta u_{LV001}(t) + 0.0450\Delta u_{PA001}(t)\right\} \tag{3.68}$$

As mentioned, $\mathscr{L}\{\cdot\}$ is a linear operator, meaning that it satisfies the *superposition principle* (see e.g. [34] page 123 - 127). Applying the superposition principle to (3.68) results in:

$$\mathscr{L}\left\{\Delta\dot{h}_1(t)\right\} = -0.0230\mathscr{L}\left\{\Delta h_1(t)\right\} - 0.0719\mathscr{L}\left\{\Delta u_{LV001}(t)\right\} + 0.0450\mathscr{L}\left\{\Delta u_{PA001}(t)\right\} \tag{3.69}$$

Notice how the multiplicative constants are factored out of the $\mathscr{L}\{\cdot\}$ operator. Notice also that the left-hand side of (3.69) contains the Laplace Transform of *the derivative* of a function. Using integration by parts, this equates to (see e.g. [20] page 39, equation 3-9):

$$\mathscr{L}\left\{\Delta\dot{h}_1(t)\right\} = s\Delta H_1(s) + \Delta h_1(0) \tag{3.70}$$

Inserting (3.70) in the left-hand side, and applying (3.67) to the right-hand side of equation (3.69), respectively, yields the following:

$$s\Delta H_1(s) + \Delta h_1(0) = -0.0230\Delta H_1(s) - 0.0719\Delta U_{LV001}(s) + 0.0450\Delta U_{PA001}(s) \tag{3.71}$$

Let the initial condition be at the nominal solution $\mathfrak{NS}$. Furthermore, let the pump operates at the nominal solution $\mathfrak{NS}$. I.e.:

$$\Delta h_1(0) = 0, \quad \Delta U_{PA001}(s) = 0 \tag{3.72}$$

Then, after some algebraic manipulation, (3.71) can be expressed as:

$$\begin{aligned}\frac{\Delta H_1(s)}{\Delta U_{LV001}(s)} &= \frac{-0.0719}{s + 0.0230}\\ &= \frac{-3.1261}{43.4783s + 1}\end{aligned} \tag{3.73}$$

which is the *transfer function* (or, relationship) between the Laplace Transform of the state $\Delta h_1(t)$ and the input $\Delta u_{LV001}(t)$, respectively.

A similar procedure can be done for the linear model of Tank 2, which is shown subsequently:

$$
\begin{aligned}
\mathscr{L}\left\{\Delta\dot{h}_2(t)\right\} &= 0.0092\mathscr{L}\left\{\Delta h_1(t)\right\} - 0.0092\mathscr{L}\left\{\Delta h_2(t)\right\} \\
&\quad + 0.0288\mathscr{L}\left\{\Delta u_{LV001}(t)\right\} - 0.0288\mathscr{L}\left\{\Delta u_{LV002}(t)\right\} \\
&\Longrightarrow \\
s\Delta H_2(s) + \Delta h_2(0) &= 0.0092\Delta H_1(s) - 0.0092\Delta H_2(s) \\
&\quad + 0.0288\Delta U_{LV001}(s) - 0.0288\Delta U_{LV002}(s) \\
&\Longrightarrow \\
\frac{\Delta H_2(s)}{\Delta U_{LV002}(s)} &= \frac{-0.0288}{s + 0.0092} \\
&\Longleftrightarrow \\
&\quad \Delta h_2(0) = 0, \quad \wedge \quad \Delta H_1(s) = 0, \quad \wedge \quad \Delta U_{LV001}(s) = 0 \\
&\therefore \\
\frac{\Delta H_2(s)}{\Delta U_{LV002}(s)} &= \frac{-3.13}{108.6957s + 1}
\end{aligned}
\tag{3.74}
$$

A transfer function provides useful information regarding the dynamic behaviour of the linear model, such as time constant and steady state gain. The steady state gain of a transfer function tells how much a change in the independent variable affects the dependent variable, while the time constant represents how fast the dependent variable reacts to that said change.

The gains (denoted by κ_i, $i \in \{1, 2\}$) and time constants (denoted by τ_i, $i \in \{1, 2\}$) of the two tanks will be useful information for future control design (see Appendix B). From (3.73), it is clear that:

$$\tau_1 \approx 43.5\,[s] \quad , \quad \kappa_1 = -3.1261 \tag{3.75}$$

Similarly, based on (3.74), one can conclude that:

$$\tau_2 \approx 109\,[s] \quad , \quad \kappa_2 = -3.13 \tag{3.76}$$

For a detailed discussion on transfer functions and their properties, see e.g. [20] Chapter 4.

Alternatively, the time constants can be found numerically by analysing the unit step response of the linear model. This can be done in MATLAB using the `step` function, which results in the following:

```

  Pole          Magnitude      Damping        Frequency        Time Constant
                                            (rad/seconds)        (seconds)
```

```

 9.95e-01      9.95e-01      1.00e+00       9.18e-03          1.09e+02
 9.89e-01      9.89e-01      1.00e+00       2.30e-02          4.36e+01
```

Additionally, Figure 3.12 illustrates the step response of the two tanks with respect to each input signal. Comparing the subplots $(1, 1)$ and $(2, 2)$ in Figure 3.12, it is clear

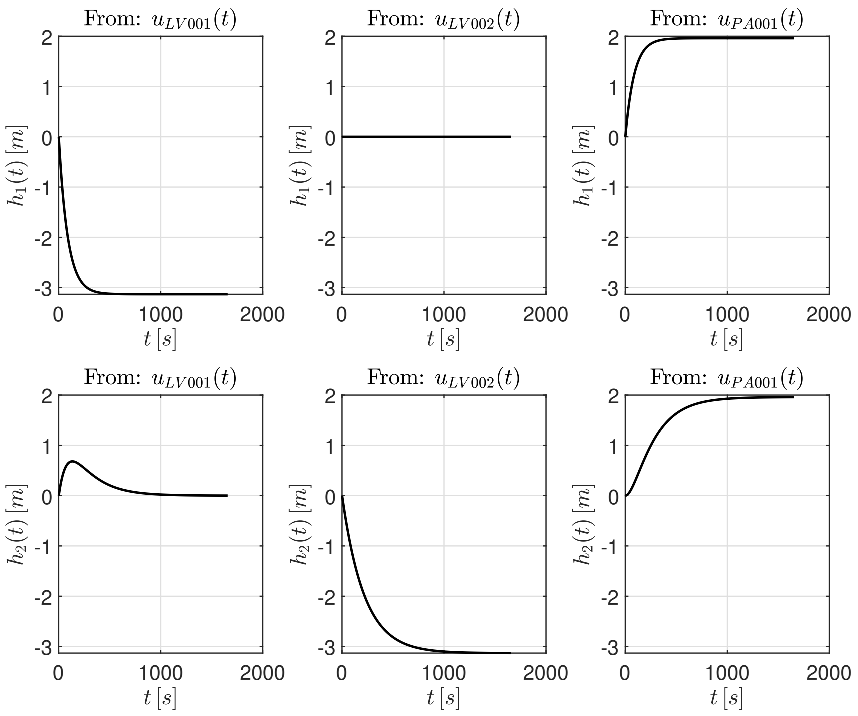


Figure 3.12: Simulation of the step response of each tank with respect to every input signal.

that Tank 1 responds faster than Tank 2 after a unit step in their respective valve signal. This, as well as the printed MATLAB output, validate the time constants in (3.75) and (3.76) that were found analytically through transfer functions.

Remark 3.2. For the sake of visualisation (as previously mentioned), the linear model in Figure 3.12 is allowed to move outside of physical boundaries.

Obviously, the transfer functions (3.73) and (3.74) (and their respective properties) will change depending on the chosen nominal solution $\mathfrak{NS}$ prior to performing the linearization.

3.5 Discretization

So far, both the nonlinear and linear models have been discussed in continuous time. However, as mentioned in Chapter 1, the MPC design for this project will be in discrete time only. Therefore, it is necessary to discretize the models.

3.5.1 Ordinary Differential Equation Solver

Ordinary Differential Equations (ODEs) are usually solved/integrated using numerical methods. These numerical methods are a necessity if the ODEs are to be analysed by a computer.

To simulate the behaviour of the nonlinear model (3.29) (which is an ODE) on a computer, it has to be done via numerical integration. To this end, the MATLAB function `ode45` is used for the duration of this project. `ode45` uses an embedded method named the 'Dormand-Prince pair', which is a member of the explicit Runge-Kutta family of ODE solvers. See [35] for an extensive discussion regarding ODE solvers of the Runge-Kutta family. Additionally, see [36] Section 13.6 for a detailed discussion on the `ode45` MATLAB function.

Remark 3.3. The MATLAB function `ode45` will be the most advanced ODE solver used during this project. Therefore, while the `ode45` function solves an ODE via numerical methods, this numerical solution will be regarded as the continuous time solution of said ODE. So, as emphasized by this remark, be aware of the distinction between the solution of a continuous time ODE (which cannot be expressed on a computer), and the numerical solution obtained via the function `ode45`.

3.5.2 Zero-Order Hold Method

The method *Zero-Order Hold* (ZOH) is used to discretize the linear models. Given the continuous time linear state-space representation:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \tag{3.77}$$

an equivalent discrete time system can be obtained such that:

$$\begin{cases} x(k+1) = A_{zoh}x(k) + B_{zoh}u(k) \\ y(k) = C_{zoh}x(k) + D_{zoh}u(k) \end{cases} \tag{3.78}$$

The matrices in (3.78) are given by (see [17] Section 4.3.3 pp. 101-110):

$$A_{zoh} = e^{At_s} \tag{3.79}$$

$$B_{zoh} = \int_0^{t_s} e^{A\eta} B \, d\eta \tag{3.80}$$

$$C_{zoh} = C \tag{3.81}$$

$$D_{zoh} = D \tag{3.82}$$

where t_s denotes the sampling time.

Clearly, the linear state-space model (3.63) does not have the same structure as the general case given by (3.77). Therefore, an augmented state-space model is used, which combines the manipulated variables with the measured disturbances:

$$\begin{cases} \underbrace{\begin{bmatrix} \Delta\dot{h}_1(t) \\ \Delta\dot{h}_2(t) \end{bmatrix}}_{\Delta\dot{h}(t)} = \underbrace{\begin{bmatrix} -0.0230 & 0 \\ 0.0092 & -0.0092 \end{bmatrix}}_{A_{\mathfrak{NS}}} \cdot \underbrace{\begin{bmatrix} \Delta h_1(t) \\ \Delta h_2(t) \end{bmatrix}}_{\Delta h(t)} + \underbrace{\begin{bmatrix} -0.0719 & 0 & 0.0450 \\ 0.0288 & -0.0288 & 0 \end{bmatrix}}_{B_{\mathfrak{NS}}} \cdot \underbrace{\begin{bmatrix} \Delta u_{LV001}(t) \\ \Delta u_{LV002}(t) \\ \Delta u_{PA001}(t) \end{bmatrix}}_{\Delta u(t)} \\ \underbrace{\begin{bmatrix} \Delta\hat{h}_1(t) \\ \Delta\hat{h}_2(t) \end{bmatrix}}_{\Delta\hat{h}(t)} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{C_{\mathfrak{NS}}} \cdot \underbrace{\begin{bmatrix} \Delta h_1(t) \\ \Delta h_2(t) \end{bmatrix}}_{\Delta h(t)} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{D_{\mathfrak{NS}}} \cdot \underbrace{\begin{bmatrix} \Delta u_{LV001}(t) \\ \Delta u_{LV002}(t) \\ \Delta u_{PA001}(t) \end{bmatrix}}_{\Delta u(t)} \end{cases} \tag{3.83}$$

The matrices in (3.83) can now be discretized by means of the ZOH method, using the equations (3.79) - (3.82). Note that the ZOH discretization method preserves both stability *and* instability [17]. As shown in Section 3.4.1, the unforced part of (3.83) is asymptotically stable. Thus, the corresponding discrete time system (obtained via the ZOH method) will also have an unforced part which is asymptotically stable.

Figure 3.13 and 3.14 show a comparison between the continuous time linear model (see Remark 3.3 in Section 3.5.1), and its corresponding discrete time model obtained via the ZOH method.

For the remainder of this project, when discretizing a linear system, the MATLAB function `c2d` is used, which applies this exact ZOH method described in this section (unless some other method is explicitly stated).

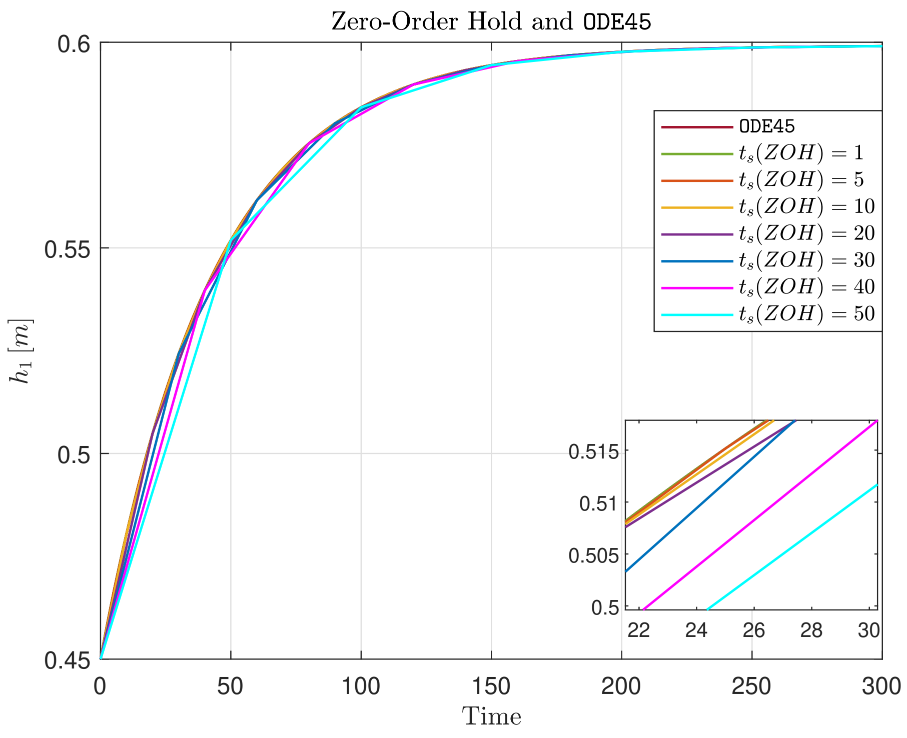


Figure 3.13: Simulation of Tank 1 using `ode45` with intervals $t_s = 0.1$, and the ZOH method with different sampling times $t_s(ZOH)$.

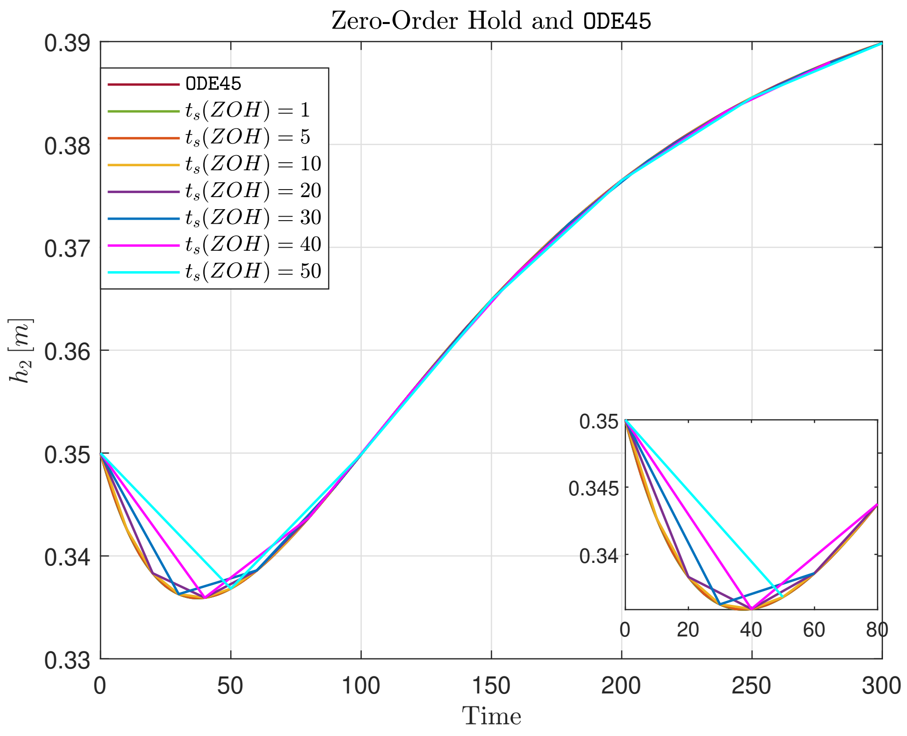


Figure 3.14: Simulation of Tank 2 using `ode45` with intervals $t_s = 0.1$, and the ZOH method with different sampling times $t_s(ZOH)$.

3.5.3 Euler method

Recall that a nonlinear MPC allows for a nonlinear prediction model to be used when solving the SQP optimization problem. However, the nonlinear state equations of the two-tank system discussed so far (see (3.29)) are in continuous time. Therefore, it is necessary to discretize said equations prior to constructing the nonlinear MPC. To this end, the *Euler method* (or sometimes called *forward Euler*) is used.

Given the continuous time ODE:

$$\frac{dy}{dx} = f(x, y) \tag{3.84}$$

and the initial value:

$$y_0 = y(x_0) \tag{3.85}$$

then, there exists a unique solution (also called an *initial-value problem*, see [29] Section 18.3):

$$y = \phi(x) = y_0 + \int_{x_0}^{x} f(t, \phi(t))\, dt \tag{3.86}$$

The Euler method approximates the solution curve (3.86) by a polygonal line [29]. Each segment in the polygonal line are equally spaced horizontally, and the slope of segment n is determined by (3.84) evaluated at the tail of the previous segment, i.e., the end of segment $n - 1$. The horizontal partitioning between each segment is referred to as the *step size*, and is denoted by h.

The polygonal line is found by iteratively evaluating the starting point of the successive segments, beginning from the initial value (3.85). Mathematically, this procedure can be expressed by the general formula (see [29] page 1009 - 1012, or [31] page 131 - 137):

$$\begin{cases} x_{n+1} = x_n + h \\ y_{n+1} = y_n + h \cdot f(x_n,\, y_n) \end{cases} \tag{3.87}$$

which is referred to as an *Euler step*. Equation (3.87), when applied to the nonlinear state equations (3.29), will act as the discrete time prediction model for the nonlinear MPC. Figure 3.15 and 3.16 show a comparison between the simulated nonlinear two-tank model, and multiple discretized models obtained via the Euler method with different step sizes h.

Remark 3.4. The reason as to why the Euler method is preferred as the prediction model for the nonlinear MPC, and not the `ode45` function, is due to its simplicity and fast computational solution, which, after comparing the two methods in Figure 3.15 and 3.16, does not significantly affect the integrity of the prediction model with short step sizes h.

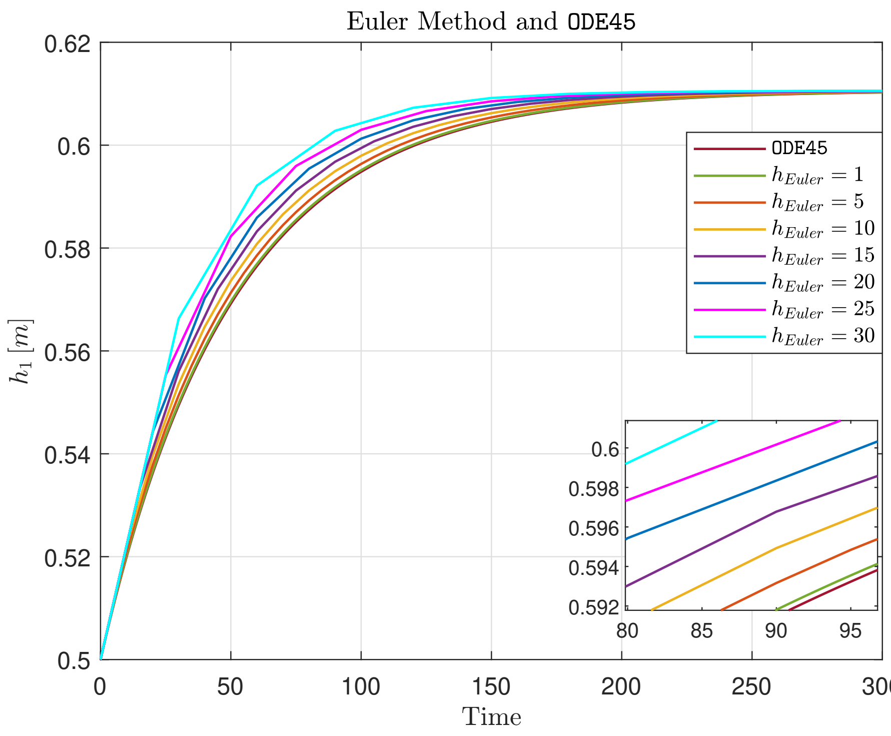


Figure 3.15: Simulation of Tank 1 using `ode45` with intervals $t_s = 0.1$, and the Euler method with different step sizes h_{Euler}.

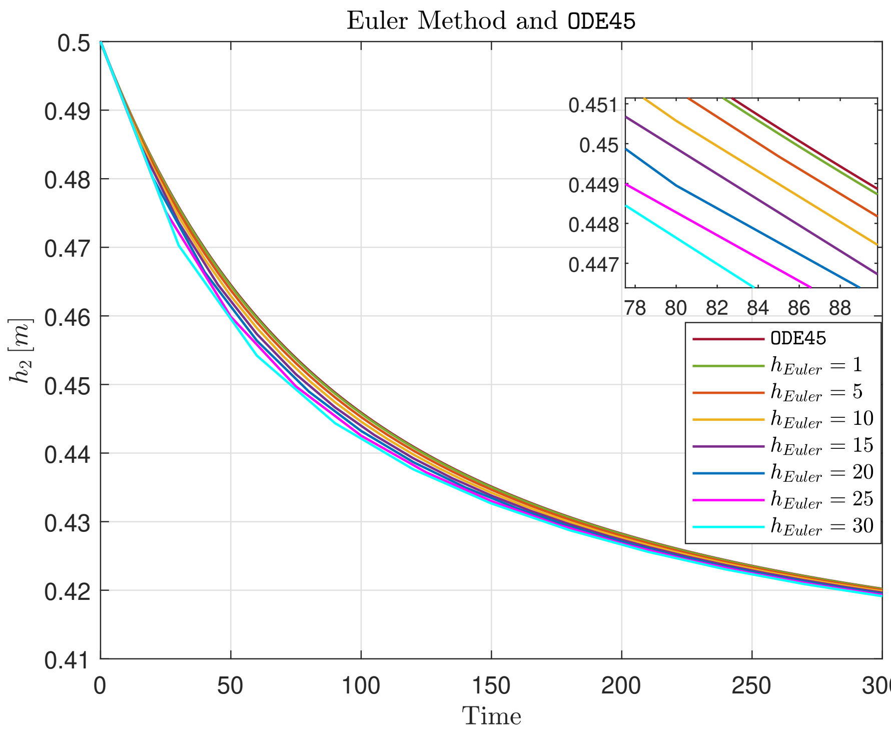


Figure 3.16: Simulation of Tank 2 using `ode45` with intervals $t_s = 0.1$, and the Euler method with different step sizes h_{Euler}.

Remark 3.5. Note that only the state equations (3.29) are addressed in this section, and that the output equations (3.30) are disregarded. This is because, while (3.30) is in continuous time, it is also linear. Since (3.30) is a linear equation, the ZOH method (see Section 3.5.2) can be used to discretize it, which will *not* affect the equations (see (3.81) and (3.82)). Thus, the output equations (3.30) can be used in the prediction model as is.

Chapter 4

MPC Design

In this chapter, multiple MPC-based controllers (such as linear-, explicit-, adaptive-, and nonlinear-MPC) are designed for the two-tank system. The goal of this chapter is to provide a clear distinction between the different MPC-based controllers, what the different MPC specifications imply, and how the MPC-based controllers can be created using MATLAB syntax. Additionally, supplemental justifications will be provided regarding the design choices and parameter selection for the MPC-based controllers when applied to the two-tank system.

4.1 Overview

4.1.1 Software

This project relies extensively on MATLAB software [37], and some of the available add-ons. Most notably are the add-ons: `Model Predictive Control Toolbox`, `Optimization Toolbox`, `Control System Toolbox`, and `Simulink`.

Using the `ver` function in the command window makes MATLAB display the current version of the software and toolboxes. Below are listed the toolboxes (and corresponding versions) used to obtain the results in this project:

```
MATLAB Version: 9.11.0.1837725 (R2021b) Update 2
Operating System: Microsoft Windows 10 Home Version 10.0 (Build 19045)
Java Version: Java 1.8.0_202-b08 with Oracle Corporation Java HotSpot(TM)
    64-Bit Server VM mixed mode

MATLAB                                                Version 9.11
    (R2021b)
```

```
Simulink                                              Version 10.4
    (R2021b)
Control System Toolbox                                Version 10.11
    (R2021b)
Econometrics Toolbox                                  Version 5.7
    (R2021b)
Model Predictive Control Toolbox                      Version 7.2
    (R2021b)
Optimization Toolbox                                  Version 9.2
    (R2021b)
Signal Processing Toolbox                             Version 8.7
    (R2021b)
Statistics and Machine Learning Toolbox               Version 12.2
    (R2021b)
Symbolic Math Toolbox                                 Version 9.0
    (R2021b)
```

4.1.2 Hardware

Performance measures such as code execution time can be greatly affected depending on the hardware running said code. Additionally, background programs can limit the available RAM used by MATLAB, further slowing the code down. Therefore, for replication purposes, it is important to provide the hardware specifications of the computer that obtained the results presented in this project. Further details regarding the computer used in this project are presented below:

- **Unit:** ideapad 710S-13IKB Signature Edition
- **Processor:** Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
- **RAM:** 8,00 GB (7,89 GB usable)
- **System Type:** 64-bit operating system, x64-based processor.

4.2 Linear MPC

4.2.1 Model

Implied by its name, the linear MPC uses a linear prediction model when optimizing for future control inputs. As mentioned in Chapter 3, a linear model of the two-tank system (about a nominal solution $\mathfrak{NS}$) can be obtained by means of linearization. For the remainder of this project, $\mathfrak{NS}$ is chosen to be:

$$\underbrace{\tilde{h}_1 = 0.5, \quad \tilde{h}_2 = 0.3, \quad \tilde{u}_{PA001} = 0.8, \quad \tilde{u}_{LV001} = 0.5317, \quad \tilde{u}_{LV002} = 0.5317}_{\mathfrak{NS}} \tag{4.1}$$

Linearizing the nonlinear model {(3.29), (3.30)} about $\mathfrak{NS}$ yields the following continuous time linear state-space model:

$$\begin{cases} \begin{bmatrix} \Delta\dot{h}_1(t) \\ \Delta\dot{h}_2(t) \end{bmatrix} = \begin{bmatrix} -0.0230 & 0 \\ 0.0092 & -0.0092 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(t) \\ \Delta h_2(t) \end{bmatrix} + \begin{bmatrix} -0.0719 & 0 & 0.0450 \\ 0.0288 & -0.0288 & 0 \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(t) \\ \Delta u_{LV002}(t) \\ \Delta u_{PA001}(t) \end{bmatrix} \\ \begin{bmatrix} \Delta\hat{h}_1(t) \\ \Delta\hat{h}_2(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(t) \\ \Delta h_2(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(t) \\ \Delta u_{LV002}(t) \\ \Delta u_{PA001}(t) \end{bmatrix} \end{cases} \tag{4.2}$$

The linear model (4.2) can be discretized by means of the ZOH method (see Section 3.5.2). Suppose that the sampling time is $t_s = 0.5\,[s]$. Performing the ZOH method with this sampling time yields the following discrete time linear state-space model:

$$\begin{cases} \begin{bmatrix} \Delta h_1(k+1) \\ \Delta h_2(k+1) \end{bmatrix} = \begin{bmatrix} 0.9886 & 0 \\ 0.004554 & 0.9954 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(k) \\ \Delta h_2(k) \end{bmatrix} \\ \qquad\qquad + \begin{bmatrix} -0.03574 & 0 & 0.02237 \\ 0.01426 & -0.01435 & 5.137 \cdot 10^{-5} \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(k) \\ \Delta u_{LV002}(k) \\ \Delta u_{PA001}(k) \end{bmatrix} \\ \begin{bmatrix} \Delta\hat{h}_1(k) \\ \Delta\hat{h}_2(k) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(k) \\ \Delta h_2(k) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(k) \\ \Delta u_{LV002}(k) \\ \Delta u_{PA001}(k) \end{bmatrix} \end{cases} \tag{4.3}$$

In MATLAB, this transformation is done using the `c2d` function, as such:

```
ts = 0.5;                     % Sampling time
sys = ss(A, B_a, C, D_a);     % CT State-Space Model
sys = c2d(sys,ts);            % DT State-Space Model (ts sampling)
```

where `A`, `B_a`, `C`, and `D_a` are the matrices appearing in the continuous time linear state-space model (4.2).

Recall that, in order to perform the ZOH transformation, an augmented state-space system is used. The augmented state-space system contains the manipulated variables ($\Delta u_{LV00i}(k)$, $i \in \{1, 2\}$) *as well as* the measured disturbance ($\Delta u_{PA001}(k)$) in the input vector. This information has to be explicitly stated to the MPC, which can be done by editing the `sys` object with the `setmpcsignals` function:

```
% Signal Types
sys = setmpcsignals(sys, 'MV', [1 2], 'MD', 3, 'MO', [1 2]);
```

where:

- 'MV' are the 'Manipulated variables', i.e., the first and second element in the input vector.
- 'MD' is the 'Measured Disturbance', i.e., the third element in the input vector.
- 'MO' are the 'Measured Outputs', i.e., the first and second element in the output vector.

The discretized `sys` object, together with the sampling time t_s, are used to create a linear MPC object in MATLAB:

```
mpcobj = mpc(sys, ts);    % Linear MPC object
```

It is important to recognize that the linear model, which the MPC uses as a prediction model, works with Δ-variables. Meaning that every control move that the MPC calculates is regarded as the deviation from the nominal solution $\mathfrak{NS}$. Therefore, the nominal solution $\mathfrak{NS}$ has to be *added* to the calculated control moves $\Delta u_{LV00i}(k)$, $i \in \{1, 2\}$ prior to them being applied to the valves.

Conversely, prior to 'feeding' the MPC with measured outputs, the nominal solution $\mathfrak{NS}$ has to be *subtracted*, such that only the Δ-variables remain (see the conversions (3.41) - (3.47)). These conversions can be done automatically by defining the nominal state of the MPC object in MATLAB:

```
mpcobj.Model.Nominal = struct('X',x0,'U',u0,'Y',y0);
```

where `x0`, `u0`, and `y0` are the state vector, input vector, and output vector, respectively, inserted the nominal solution $\mathfrak{NS}$:

$$\texttt{x0} = \begin{bmatrix} 0.5 \\ 0.3 \end{bmatrix}, \quad \texttt{u0} = \begin{bmatrix} 0.5317 \\ 0.5317 \\ 0.8 \end{bmatrix}, \quad \texttt{y0} = \begin{bmatrix} 0.5 \\ 0.3 \end{bmatrix} \tag{4.4}$$

As a final test, one can confirm if the linear discrete time prediction model is indeed controllable. The MATLAB function `ctrb` calculates the controllability matrix $\mathcal{C}$, which is defined as:

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & \dots & A^{\mathfrak{n}-1}B \end{bmatrix} \tag{4.5}$$

In MATLAB, (4.5) is calculated subsequently:

```
ctrb(sys);
```

which gives:

$$\mathcal{C} = \begin{bmatrix} -0.0357 & 0 & 0.0224 & -0.0353 & 0 & 0.0221 \\ 0.0143 & -0.0143 & 0.0001 & 0.0140 & -0.0143 & 0.0002 \end{bmatrix} \tag{4.6}$$

The rank of $\mathcal{C}$ determines if the state-space model pair $(A,\, B)$ is controllable or not. If $\mathcal{C}$ has full row rank, i.e., $rank(\mathcal{C}) = \mathfrak{n}$ (where $\mathfrak{n}$ is the number of states), then, the matrix pair $(A,\, B)$ is said to be controllable. In MATLAB, this can be checked with the following code:

```
rank(ctrb(sys));
```

which yields:

$$rank(\mathcal{C}) = 2 = \mathfrak{n} \tag{4.7}$$

Thus, confirming that the prediction model (4.3) is indeed controllable. See [13] Section 5 for a complete discussion on controllability.

4.2.2 Quadratic Objective Function

In its most general form, the short-hand notation of the quadratic objective function for the linear MPC can be written subsequently (see [23]):

$$J(z_k) = J_y(z_k) + J_u(z_k) + J_{\delta u}(z_k) + J_\varepsilon(z_k) \tag{4.8}$$

where z_k is the QP decision variable. Each term in (4.8) represents the cost related to different aspects of optimal control performance, which, together, make up the total cost.

Controllers are commonly used for output reference tracking. A controller's ability to perform this task can be quantified by the quadratic cost term $J_y(z_k)$, which can be expressed mathematically as:

$$J_y(z_k) = \sum_{i=1}^{P} \left\{ (r(k+i|k) - y(k+i|k))^T \cdot Q^2 \cdot (r(k+i|k) - y(k+i|k)) \right\} \tag{4.9}$$

or as (see [23] page 1-7):

$$J_y(z_k) = \sum_{i=1}^{\mathfrak{p}} \sum_{j=1}^{P} \left\{ Q_{i,i}^2 \cdot [r_i(k+j|k) - y_i(k+j|k)]^2 \right\} \tag{4.10}$$

where:

- $\mathfrak{p}$ is the number of outputs.
- $r(k+i|k) \in \mathbb{R}^{\mathfrak{p}}$ is the output reference vector.
- $r_i(k+j|k) \in \mathbb{R}$ is the reference for output number i.
- $y(k+i|k) \in \mathbb{R}^{\mathfrak{p}}$ is the output vector.
- $y_i(k+j|k) \in \mathbb{R}$ is output number i.
- Q is the diagonal output weighting matrix.
- $Q_{i,i}$ is the i^{th} diagonal element in the output weighting matrix Q.

Commonly for all the cost functions presented in this section is that:

- k is the current control interval.
- P is the prediction horizon.
- The QP decision variable z_k is defined as (see [23] page 1-8):

$$z_k{}^T = \left[u(k|k)^T,\, u(k+1|k)^T,\, \ldots,\, u(k+P-1|k)^T,\, \varepsilon_k\right] \tag{4.11}$$

  where ε_k is a dimensionless slack variable.

Remark 4.1. Note the deviation from the previously discussed quadratic cost functions in Chapter 2, where, in this case, the weighting matrix Q is squared. This also applies for the subsequent cost functions and their respective weighting matrices.

Similarly, on some occasions, it is desirable for the manipulated variables to track some reference (or commonly referred to as a *target*). The controller's ability to follow a target for the manipulated variables is quantified by $J_u(z_k)$, which can be written as:

$$J_u(z_k) = \sum_{i=0}^{P-1} \left\{ \left(u_{target}(k+i|k) - u(k+i|k)\right)^T \cdot R^2 \cdot \left(u_{target}(k+i|k) - u(k+i|k)\right) \right\} \tag{4.12}$$

or as (see [23] page 1-8):

$$J_u(z_k) = \sum_{i=1}^{\mathfrak{m}} \sum_{j=0}^{P-1} \left\{ R_{i,i}^2 \cdot \left[u_{target,\,i}(k+j|k) - u_i(k+j|k)\right]^2 \right\} \tag{4.13}$$

Here,

- $\mathfrak{m}$ is the number of manipulated variables.
- $u_{target}(k+i|k) \in \mathbb{R}^{\mathfrak{m}}$ is the manipulated variable target vector.
- $u_{target,i}(k+j|k) \in \mathbb{R}$ is the target for manipulated variable number i.
- $u(k+i|k) \in \mathbb{R}^{\mathfrak{m}}$ is the manipulated variable vector.
- $u_i(k+j|k) \in \mathbb{R}$ is manipulated variable number i.
- R is the diagonal control weighting matrix.
- $R_{i,i}$ is the i^{th} diagonal element in the control weighting matrix R.

As mentioned in Chapter 2, the rate at which manipulated variables change from one control interval to the next can cause disturbances to the system, or (if the change is too rapid) damage the actuators. The term $J_{\delta u}(z_k)$ determines the cost related to the rate of the manipulated variables, and is expressed subsequently:

$$J_{\delta u}(z_k) = \sum_{i=0}^{P-1} \left\{ (u(k+i|k) - u(k+i-1|k))^T \cdot S^2 \cdot (u(k+i|k) - u(k+i-1|k)) \right\} \tag{4.14}$$

or as (see [23] page 1-8):

$$J_{\delta u}(z_k) = \sum_{i=1}^{\mathfrak{m}} \sum_{j=0}^{P-1} \left\{ S_{i,i}^2 \cdot [u_i(k+j|k) - u_i(k+j-1|k)]^2 \right\} \tag{4.15}$$

with the defined variables:

- $\mathfrak{m}$ is the number of manipulated variables.
- S is the diagonal control rate weighting matrix.
- $S_{i,i}$ is the i^{th} diagonal element in the control rate weighting matrix S.

Finally, the term $J_\varepsilon(z_k)$ quantifies the constraint violations of the controller. In some situations, e.g. when a system is highly prone to disturbances, the system might move from the constraint set. For soft constraints (see Section 4.2.5), this violation can be temporarily allowed, but it will be heavily penalized by the quadratic objective function $J_\varepsilon(z_k)$. This is not a relevant case for the two-tank system, therefore, it will not be discussed any further. See [23] page 1-9 for a detailed description on this term and its implications.

Remark 4.2. Notice that all of the terms in (4.8) are all dependent on the free variable z_k. This is because the output references, manipulated variable targets, previous inputs, and measured disturbances are all known at control interval k. The MPC's prediction model is dependent on the current state of the system, and the current/future inputs, in order to predict the optimal outputs for a horizon of P steps. The current state can either be measured directly (as is the case in the two-tank system), or, it can be estimated based on output measurements through a state observer (more on this in Section 6.10). This leaves the only independent variable z_k, which is defined by (4.11).

4.2.3 Scaling

Prior to tuning the weighting matrices Q, R, and S (defined in Section 4.2.2), it is important to determine the scaling factors of the outputs and inputs properly. Why scaling factors are important is easily described through a simple example:

Example 4.1. *Consider a chemical plant where some liquid is stirred in a tank. This tank is equipped with a control system that performs level control and temperature control. Thus, the state vector can be written as:*

$$x = \begin{bmatrix} h(t) \\ T(t) \end{bmatrix}$$

where $h(t)\,[m]$ and $T(t)\,[°C]$ are the fluid level and fluid temperature, respectively.

The tank is $3m$ tall with a lower boundary on the fluid level of $0.5m$, meaning that the effective range of the height is $0.5m \leq h(t) \leq 3m$. The temperature of the tank ranges between $50°C \leq T(t) \leq 300°C$.

Now, assume that both of the states are weighted equally. I.e., $h(t)$ and $T(t)$ are equally significant (relative to each other) to the overall quadratic objective function. To achieve this, an initial guess for the state weighting matrix is chosen to be:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Furthermore, the reference vector is set to:

$$\bar{x} = \begin{bmatrix} 2m \\ 200°C \end{bmatrix}$$

Assume that both of the states deviate from their respective reference by $+10\%$, i.e., $h(t) = 2m \cdot 1.1 = 2.2m$ and $T(t) = 200°C \cdot 1.1 = 220°C$. The corresponding quadratic *cost for this particular case is shown subsequently:*

$$\begin{aligned}
J &= (\bar{x} - x)^T \cdot Q^2 \cdot (\bar{x} - x) \\
&= \left\{ \begin{bmatrix} 2 \\ 200 \end{bmatrix} - \begin{bmatrix} 2.2 \\ 220 \end{bmatrix} \right\}^T \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^2 \cdot \left\{ \begin{bmatrix} 2 \\ 200 \end{bmatrix} - \begin{bmatrix} 2.2 \\ 220 \end{bmatrix} \right\} \\
&= \begin{bmatrix} -0.2 & -20 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^2 \cdot \begin{bmatrix} -0.2 \\ -20 \end{bmatrix} \\
&= 1 \cdot 0.2^2 + 1 \cdot 20^2 \\
&= \underbrace{0.04}_{J_h} + \underbrace{400}_{J_T}
\end{aligned}$$

Clearly, $J_T \gg J_h$.

Since the relative cost of J_T *is much higher than* J_h*, the MPC will focus (almost exclusively) on controlling the temperature of the tank while neglecting the fluid level. However, this is not the desired control according to the chosen weighting matrix* Q*, which was initially chosen such that both of the states are to be weighted equally. The issue lies in the fact that the states are poorly scaled.*

Scaling factors (denoted by λ*) can be used to properly re-scale the states, and they are defined as:*

$$\lambda_{x_i} = x_i^{max} - x_i^{min}$$

where x_i *is some arbitrary state in the state vector* $x \in \mathbb{R}^{\mathrm{n}}$*. These scaling factors are used to adjust the weighting matrix* Q*, as such:*

$$Q = \begin{bmatrix} \frac{Q_{1,1}}{\lambda x_1} & 0 & \dots & 0 \\ 0 & \frac{Q_{2,2}}{\lambda x_2} & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & \frac{Q_{\mathrm{n},\mathrm{n}}}{\lambda x_{\mathrm{n}}} \end{bmatrix}$$

Back to the example, the scaling factors for the tank are:

$$\begin{aligned}
\lambda_h &= h^{max} - h^{min} = 3 - 0.5 = 2.5 \\
\lambda_T &= T^{max} - T^{min} = 300 - 50 = 250
\end{aligned}$$

which results in the following weighting matrix:

$$\begin{aligned} Q &= \begin{bmatrix} \frac{1}{\lambda_h} & 0 \\ 0 & \frac{1}{\lambda_T} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{2.5} & 0 \\ 0 & \frac{1}{250} \end{bmatrix} \\ &= \begin{bmatrix} 0.4 & 0 \\ 0 & 0.004 \end{bmatrix} \end{aligned}$$

Re-calculating the quadratic cost yields:

$$\begin{aligned} J &= \begin{bmatrix} -0.2 & -20 \end{bmatrix} \cdot \begin{bmatrix} 0.4 & 0 \\ 0 & 0.004 \end{bmatrix}^2 \cdot \begin{bmatrix} -0.2 \\ -20 \end{bmatrix} \\ &= 0.4^2 \cdot 0.2^2 + 0.004^2 \cdot 20^2 \\ &= \underbrace{0.0064}_{J_h} + \underbrace{0.0064}_{J_T} \end{aligned}$$

Now the quadratic cost of the two states are of the same order (here equal) and the MPC will prioritise both states equally (which was initially intended), albeit, the relative value of the total cost has decreased with respect to the non-scaled total cost.

For the two-tank system, the scaling factors are determined to be (see Table 3.1 and 3.2):

$$\lambda_{h_1} = 1 - 0.13 = 0.87 \tag{4.16}$$

$$\lambda_{h_2} = 0.4 - 0.02 = 0.38 \tag{4.17}$$

$$\lambda_{u_{LV001}} = 1 - 0 = 1 \tag{4.18}$$

$$\lambda_{u_{LV002}} = 1 - 0 = 1 \tag{4.19}$$

$$\lambda_{u_{PA001}} = 1 - 0 = 1 \tag{4.20}$$

In MATLAB, the weighting matrices Q, R, and S can be automatically scaled, with respect to the scaling factors (4.16) - (4.20), by adjusting the `ScaleFactor` property of each signal in the MPC object. By default, the `ScaleFactor` property is set to unity. The scale factors for the inputs (4.18) - (4.20) already correspond to the default value, thus, no adjustments are required for these signals. The following code shows how the scale factors (4.16) and (4.17) are applied to the output signals in MATLAB:

```matlab
% Signal Scaling
mpcobj.OV(1).ScaleFactor = 1 - 0.13;     % Range of h1
mpcobj.OV(2).ScaleFactor = 0.4 - 0.02;   % Range of h2
```

4.2.4 Weighting Matrices

The weighting matrices Q, R, and S are MPC controller specifications that determine the significance of the different objective functions described in Section 4.2.2. These tuning variables can be adjusted freely, depending on the priority of the system operating engineer.

In the case of the two-tank system, the MPC controller is concerned with level control, which, by extension, prioritizes output reference tracking. Therefore, the diagonal elements in the output weighting matrix Q should be selected relatively larger than the the diagonal elements in R and S.

Output Weighting Matrix

The general output weighting matrix Q for the two-tank system can be expressed as:

$$Q = \begin{bmatrix} Q_{1,1} & 0 \\ 0 & Q_{2,2} \end{bmatrix} \tag{4.21}$$

Given (4.21), there are three main cases that the controller can operate under:

(i) $Q_{1,1} > Q_{2,2}$, i.e., the controller prioritizes output reference tracking of Tank 1 more than Tank 2.

(ii) $Q_{2,2} > Q_{1,1}$, i.e., the controller prioritizes output reference tracking of Tank 2 more than Tank 1.

(iii) $Q_{1,1} = Q_{2,2}$, i.e., the controller prioritizes output reference tracking of Tank 1 and Tank 2 equally.

In this project, the predominant engineering priority is case (iii), however, case (i) and case (ii) will also be briefly studied (see Chapter 5). The weighting matrices for the three aforementioned cases are selected to be:

$$Q^{i} = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}, \quad Q^{ii} = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}, \quad Q^{iii} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.22}$$

Input Weighting Matrix

For the level control of the two-tank system, there is no penalty that accounts for the usage of the valves. Thus, the input weighting matrix R is selected to be:

$$R = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{4.23}$$

I.e., the inputs $u_{LV00i}(k)$, $i \in \{1, 2\}$ can move freely from their respective targets $u_{target,i}(k)$, $i \in \{1, 2\}$ without affecting the overall quadratic objective function. Due to this chosen control property, the input targets are not applicable, and, by extension, not defined.

Remark 4.3. For the sake of completion, the reader should be aware that there are multiple cases where the inputs are heavily penalized for deviating from their targets. This is often the case when trying to optimize the operational cost[1] of a system. For instance, consider a quadcopter where the inputs are the voltage to their respective rotor. Choosing a large input weighting matrix R forces the MPC to find an optimal solution which uses the rotors conservatively. This can reduce the overall power consumption, effectively increasing the battery life of the quadcopter.

Input Rate Weighting Matrix

The input rate weighting matrix S determines how much changes in the inputs should be penalized, and, therefore, contribute to the quadratic objective function. To reduce disturbance, and to promote smooth control of the two-tank, S is chosen to be:

$$S = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{4.24}$$

Recall that the weighting matrices Q, R, and S all define the *relative* significance of the outputs, inputs, and input rates, with respect to each other. In other words, the MPC with the following control specifications:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad S = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{4.25}$$

will have the exact same control priorities as the MPC with:

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}, \quad R = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad S = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \tag{4.26}$$

[1]Cost, as in physical resources such as fule or electricity.

In MATLAB, the weighting matrices can be adjusted with the following code:

```
% Weighting Matrices
mpcobj.Weights.OutputVariables = [q11 q22];                % Q
mpcobj.Weights.ManipulatedVariablesRate = [s11 s22];       % S
mpcobj.Weights.ManipulatedVariables = [r11 r22];           % R
```

where `q11` and `q22` are $Q_{1,1}$ and $Q_{2,2}$ in (4.21), respectively. The same applies for S and R.

4.2.5 Constraints

As mentioned in Chapter 2, a desirable feature of the MPC is its ability to satisfy multiple constraints on the inputs, input rates, and outputs, respectively. These constraints (or *bounds*) are classified as either *soft*- or *hard*- constraints.

Hard Constraints

Hard constraints are constraints that *must* not be violated by the MPC during operation. If the system is pushed to a state that violates hard constraints (e.g. by some disturbance), then, the QP solver will yield an infeasible solution.

For the two-tank system, hard constraints are imposed *only* on the input variables $u_{LV00i}(k)$, $i \in \{1,2\}$, as such:

$$u_{LV00i,\,min} \leq u_{LV00i}(k) \leq u_{LV00i,\,max} \quad , \quad i \in \{1,2\} \tag{4.27}$$

where (see Table 3.1 and 3.2):

$$\begin{cases} u_{LV00i,\,min} = 0 \\ u_{LV00i,\,max} = 1 \end{cases} \quad , \quad i \in \{1,2\} \tag{4.28}$$

The bounds in (4.28) yields an input constraint set $\mathcal{U} \in \mathbb{R}^{\mathrm{m}}$, which is illustrated in Figure 4.1. In MATLAB, these hard constraints are set subsequently:

```
% Input (hard) Constraints
mpcobj.MV(1).Max = 0.9999;
mpcobj.MV(1).Min = 0.0001;
mpcobj.MV(2).Max = 0.9999;
mpcobj.MV(2).Min = 0.0001;
```

Note that, in a practical setting, the constraints applied in the code are slightly *within* the set $\mathcal{U}$, and not on the exact boundary defined by (4.28). This is to prevent numerical issues

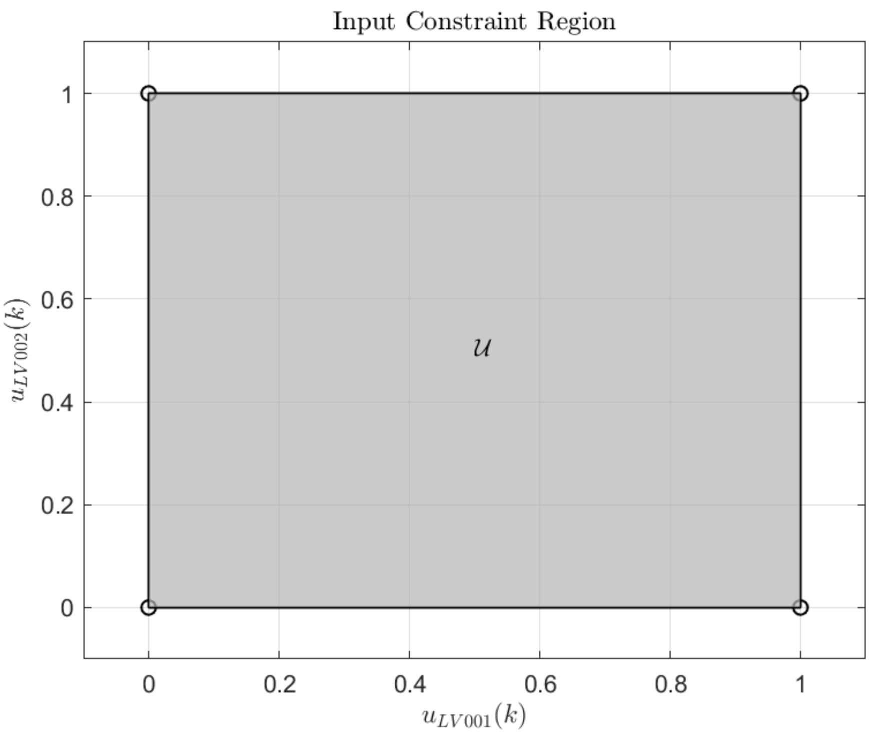


Figure 4.1: Input constraint set $\mathcal{U} \in \mathbb{R}^m$ illustrated by the gray polytope.

due to rounding error and constraint tolerances in the QP solver. For instance, consider this simple example which highlights one possible numerical issue if the constraints are set exactly on the bounds given by (4.28):

Example 4.2. *Suppose that Tank 1 is below the desired output, as may be the case after a step in the reference. The logical action is to close the valve LV001 until $h_1(k)$ reaches the desired output. I.e., $u_{LV001}(k) \to 0$.*

The QP solver, when optimizing for the set of P inputs, is required to satisfy the constraints (4.28) at all times. However, there is a set tolerance factor ς, which determines how strict the QP solver is when deciding if a constraint is satisfied or not. I.e., the QP solver will allow values (outside of the constraint set $\mathcal{U}$) within the ranges:

$$\begin{cases} u_{LV00i,\,min} - \varsigma & \leq & u_{LV00i}(k) & \leq & u_{LV00i,\,min} \\ u_{LV00i,\,max} & \leq & u_{LV00i}(k) & \leq & u_{LV00i,\,max} + \varsigma \end{cases}, \quad i \in \{1, 2\} \tag{4.29}$$

When $u_{LV001}(k) \to 0$, $u_{LV001,\,min} = 0$, and $\varsigma > 0$, then, the QP solver may select $u_{LV001}(k)$ to be some value ζ, which is within:

$$0 - \varsigma \leq \zeta < 0 \tag{4.30}$$

Inserting ζ in the approximated valve characteristic equation (3.8) yields:

$$f_1(\zeta) = \frac{e^{\zeta^{1.2}} - 1}{e^1 - 1} \in \mathbb{C} \tag{4.31}$$

which is a complex value (!), and has no physical interpretation.

The default constraint tolerance for the MPC object in MATLAB is $\varsigma = 1 \cdot 10^{-6}$. Thus, the absolute range for the inputs are:

$$0.0001 - 1 \cdot 10^{-6} \leq u_{LV00i}(k) \leq 0.9999 + 1 \cdot 10^{-6} \quad , \quad i \in \{1, 2\} \tag{4.32}$$

which do not exceed the physical boundaries of (4.28), and numerical issues (as shown in the example above) are avoided.

No constraints are set on the input rates explicitly in MATLAB. However, since the inputs themselves are restricted to approximately:

$$0 \leq u_{LV00i}(k) \leq 1 \quad , \quad i \in \{1, 2\} \tag{4.33}$$

then, the biggest possible rates occur when the inputs either move from $0 \rightarrow 1$ or from $1 \rightarrow 0$. I.e.:

$$-1 \leq \delta u_{LV00i}(k) \leq 1 \quad , \quad i \in \{1, 2\} \tag{4.34}$$

The constraint set $\delta\mathcal{U} \in \mathbb{R}^{\mathfrak{m}}$ is illustrated by Figure 4.2

Soft Constraints

Contrary to hard constraints, the QP solver will yield a feasible solution if soft constraints are violated. During the optimization process, the QP solver may deem it necessary to temporarily violate certain soft constraints, in order to achieve the minimal quadratic objective cost. In practice, soft constraints are imposed on output variables, as it can lead to infeasibility in the QP solver if hard constraints are imposed on both manipulated- and output- variables simultaneously. In MATLAB, soft constraints are determined by the parameter *equal concern for relaxation* (ECR) which will be denoted by β. Given β, the active range for the soft constraints (imposed on output variables) will be:

$$y_{min} - \beta \leq y(k+1|k) \leq y_{max} + \beta \tag{4.35}$$

where:

- $y_{min} \in \mathbb{R}^{\mathfrak{p}}$ is the lower bound on the outputs.

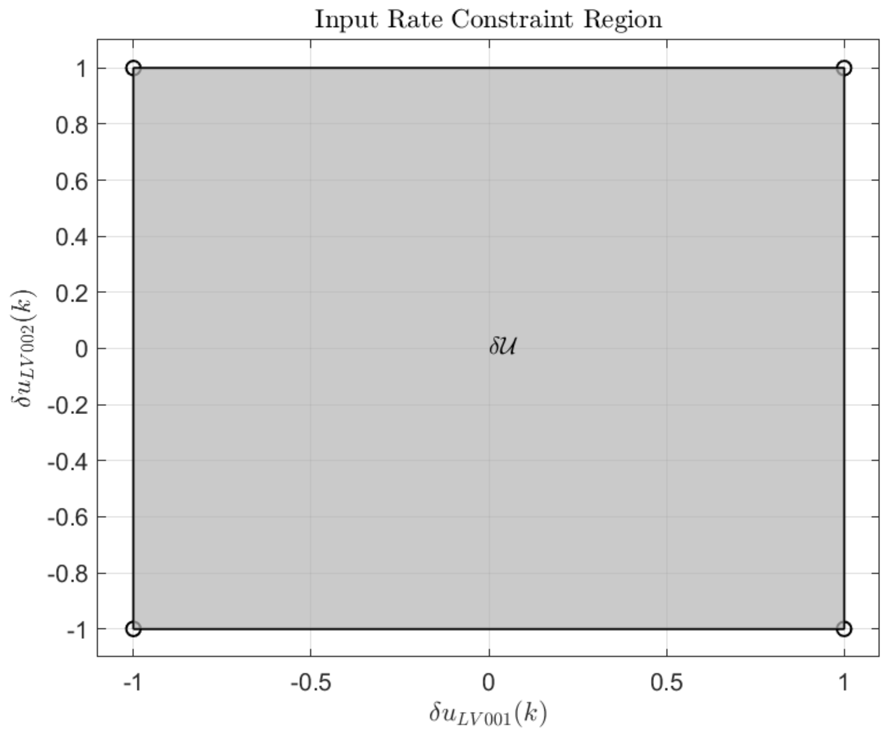


Figure 4.2: Input rate constraint set $\delta\mathcal{U} \in \mathbb{R}^{\mathfrak{m}}$ illustrated by the gray polytope.

- $y_{max} \in \mathbb{R}^{\mathfrak{p}}$ is the upper bound on the outputs.
- $y(k+1|k) \in \mathbb{R}^{\mathfrak{p}}$ is the output vector.

For a detailed discussion on soft constraints, see [23] page 2-7 - 2-9.

As mentioned in Section 4.2.2, no soft constraints are imposed on the linear MPC during control of the two-tank system. Both of the tanks have lower bounds of $h_{1,utlop}$ and $h_{2,utlop}$, respectively (see Table 3.1 and 3.2). Additionally, both of the tanks are equipped with overflow pipes, which prevent the fluid levels of exceeding certain upper bounds. Therefore, there is no possible scenario where a disturbance (or prediction error) can 'push' the system outside of its respective (physical) constraint set $\mathcal{X} \in \mathbb{R}^{\mathfrak{m}}$ (see Figure 4.3). As a result of the discussion above, the soft constraints imposed on the output variables can be set in MATLAB accordingly:

```
% Output (soft) Constraints
mpcobj.OV(1).Max = inf;
mpcobj.OV(1).Min = -inf;
mpcobj.OV(2).Max = inf;
mpcobj.OV(2).Min = -inf;
% Output ECR (slack)
mpcobj.OV(1).MinECR = 1;
mpcobj.OV(1).MaxECR = 1;
mpcobj.OV(2).MinECR = 1;
```

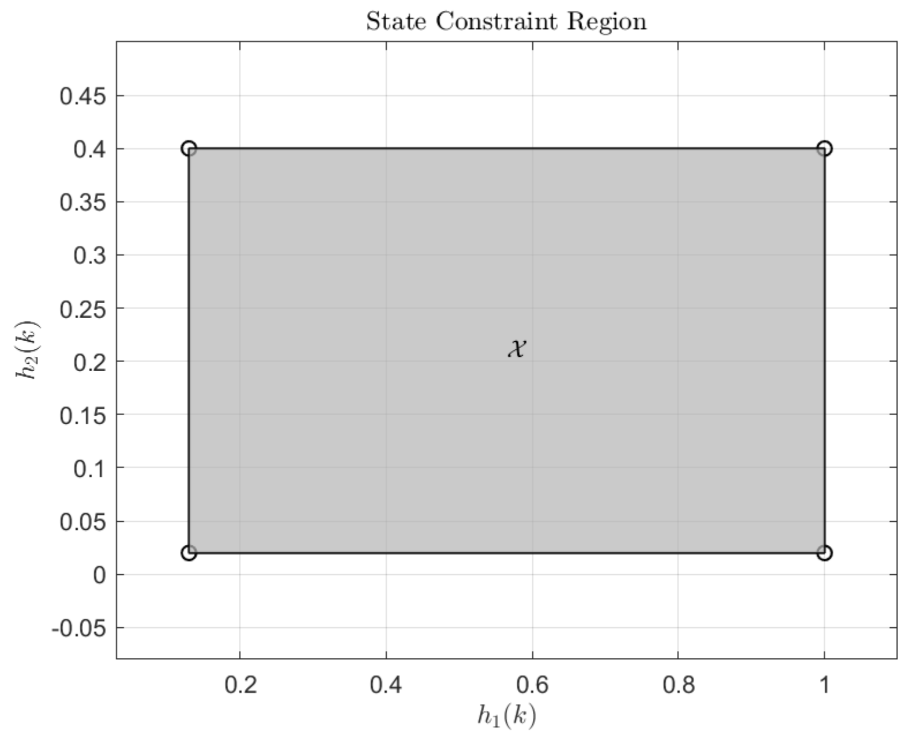


Figure 4.3: State constraint set $\mathcal{X} \in \mathbb{R}^n$ illustrated by the gray polytope.

```
mpcobj.OV(2).MaxECR = 1;
```

which equates to:

$$\begin{bmatrix} -\infty - \beta \\ -\infty - \beta \end{bmatrix} \leq \begin{bmatrix} \hat{h}_1(t) \\ \hat{h}_2(t) \end{bmatrix} \leq \begin{bmatrix} +\infty + \beta \\ +\infty + \beta \end{bmatrix} \tag{4.36}$$

with the ECR paramter $\beta = 1$ (default value).

4.2.6 Prediction- and Control-Horizon

The prediction horizon $P \in \mathbb{N}$ determines how many control intervals the MPC will predict into the future at current control interval k. The total time (t_p) which the MPC predicts into the future is given by:

$$t_p = t_s \cdot P \tag{4.37}$$

where t_s is the sampling time.

The control horizon $M \in \mathbb{N}$ is the number of free manipulated variables in the QP problem that are to be optimized at control interval k [23]. See Section 2.2 for a more detailed discussion on the prediction horizon and control horizon.

In MATLAB, the prediction- and control-horizons can be selected subsequently:

```
% Moving Horizon Length
mpcobj.PredictionHorizon = P  % Prediction Horizon
mpcobj.ControlHorizon = M     % Control Horizon
```

As an example, consider the discrete time system (4.3). Suppose that the system is operating at the nominal solution $\mathfrak{NS}$ (see 4.1), and that the MPC has the specifications $P = M = 40$. I.e., the prediction- and control-horizons are 40 control intervals long.

At time step k, the reference for Tank 1 changes from $0.5m \rightarrow 0.7m$. As mentioned in Chapter 2, the MPC calculates the optimal set of control moves $\mathfrak{U}_{opt}(k)$, such that the prediction model state sequence $\mathfrak{X}_{opt}(k)$ follows the reference in an optimal manner. Figure 4.4 shows these predictions for this particular example. As previously mentioned

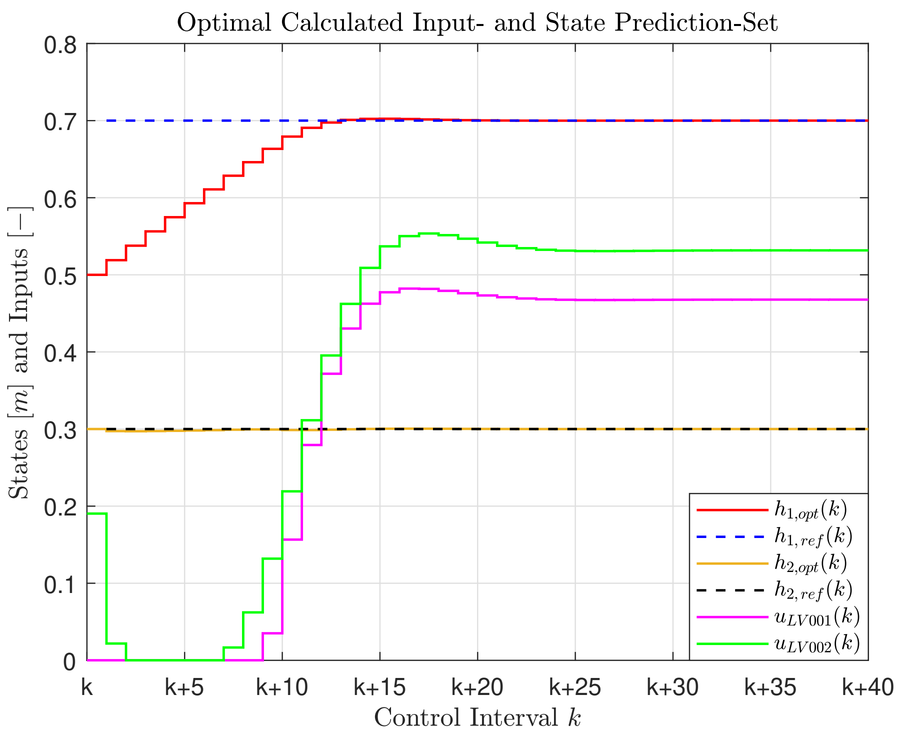


Figure 4.4: Optimal calculated state prediction set $\mathfrak{X}_{opt}(k)$ and control moves $\mathfrak{U}_{opt}(k)$ at control interval k. Prediction- and control- horizons are 40 control intervals long.

(see Section 2.2), one requirement for the moving horizon is that $P \geq M \geq 1$. If $P > M$, then, the last $P - M$ calculated optimal control moves are forced to be equal constants for the remaining prediction horizon. Figure 4.5 shows this case, where $P = 40$ and $M = 12$. Recall that the moving horizon policy only implements the first optimal calculated control move $u_{opt}(k)$, while discarding the remaining $P - 1$ calculated control moves. In Figure

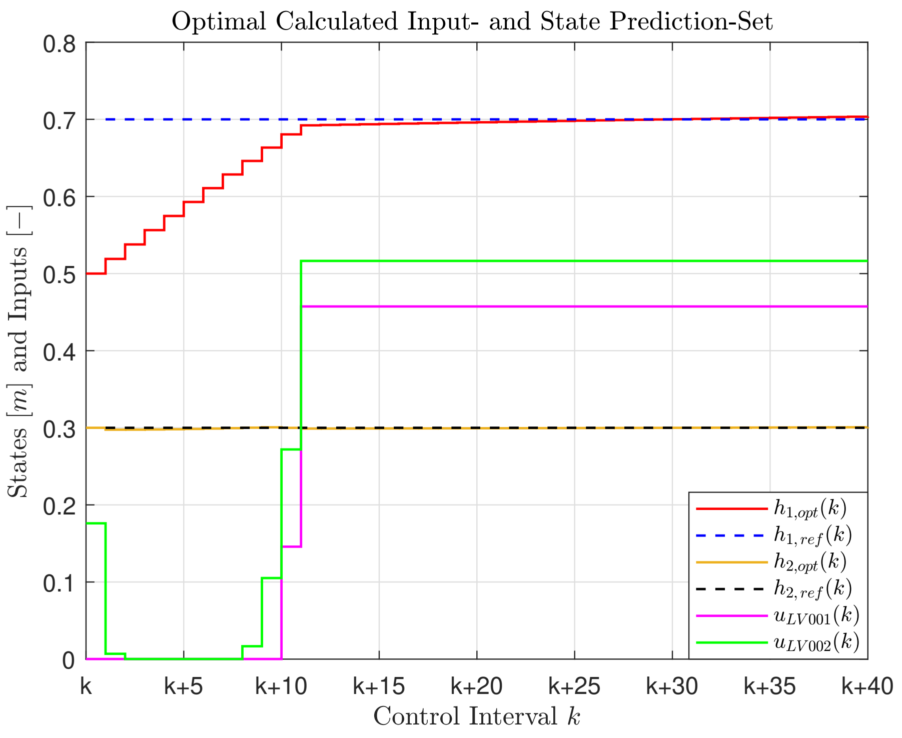


Figure 4.5: Optimal calculated state prediction set $\mathfrak{X}_{opt}(k)$ and control moves $\mathfrak{U}_{opt}(k)$ at control interval k. Prediction- and control- horizons are 40 and 12 control intervals long, respectively.

4.4 and 4.5, the first optimal control move is calculated to be:

$$u^{40}_{opt}(k) = \begin{bmatrix} 0.1903 \\ 0.0001 \end{bmatrix} \quad \text{and} \quad u^{12}_{opt}(k) = \begin{bmatrix} 0.1761 \\ 0.0001 \end{bmatrix} \tag{4.38}$$

where the superscript denotes the respective control horizon length.

Evidently from (4.38), the first optimal calculated control move with $M = 40$ and $M = 12$ do not differ significantly. Additionally, this difference does not pose a significant change in the next predicted state (see $h_{1,opt}(k+1)$ in Figure 4.4 and 4.5). For this reason, choosing horizons where $P \gg M$ is recommended, as it forces the remaining $P - M$ control moves to be equal constants, effectively reducing the QP problem and the computational effort (see [22] Section 28.3.4 or [23] Section 2.1.3).

Input Blocking

An effort to reduce the computational complexity, while maintaining the solution quality, can be done through input blocking.

Input blocking is a strategy where the QP solver optimizes for *one* optimal control input over a predefined *control interval* (or *block*). The optimal control input is then held constant for the duration of said block, until the interval is completed. The control horizon M, with input blocking, can be expressed subsequently:

$$M = \begin{bmatrix} m_1 & m_2 & \dots \end{bmatrix}$$

where m_1, m_2, etc., represent the respective block lengths. Note the necessary condition $P \geq m_1 + m_2 + \dots$, similarly to $P \geq M$ for the case without input blocking.

For example, consider the same simulation scenario as shown in Figure 4.4 and 4.5, with $P = 40$. However, now, the control horizon is divided into the following blocks:

$$M = \begin{bmatrix} 1 & 3 & 6 & 15 & 15 \end{bmatrix} \tag{4.39}$$

which is the equivalent of the following control intervals:

$$\begin{aligned} &k = k, && k+1 \leq k \leq k+3, \\ &k+4 \leq k \leq k+9, && k+10 \leq k \leq k+24, \quad k+25 \leq k \leq k+39 \end{aligned}$$

The MPC will solve *one* QP optimization problem for each of the control intervals, and this solution (for its respective interval) will be held constant for the duration of the control block. Figure 4.6 shows the simulation with M given by (4.39). Comparing the simulations between Figure 4.4 and 4.6, it is clear that the solution quality is worsened by the use of input blocking (especially at $k + 10 \leq \text{Control Interval} \leq k + 25$). However, the QP problem in Figure 4.6 has reduced complexity, and is only solving for 5 unique optimal control inputs, while the QP problem in Figure 4.4 needs to solve for 40 unique optimal control inputs.

Additionally, as discussed for the case in Figure 4.5, the first optimal control input with input blocking does not differ significantly from the first optimal control input with $M = 40$ (see $u_{LV001}(k)$ in Figure 4.4 and 4.6).

As explained by Levine [22] (Section 28.3.4), and by Bemporad *et al.* [23] (Section 2.1.3), a common practise for input blocking is to use small partitioning (i.e., short input blocks) at the beginning of the prediction horizon, and steadily increase the block sizes towards the later stages of the prediction horizon.

In MATLAB, the input blocking shown in Figure 4.6 can be done with the following syntax:

```matlab
% Moving Horizon Length
mpcobj.PredictionHorizon = 40            % Prediction Horizon
```

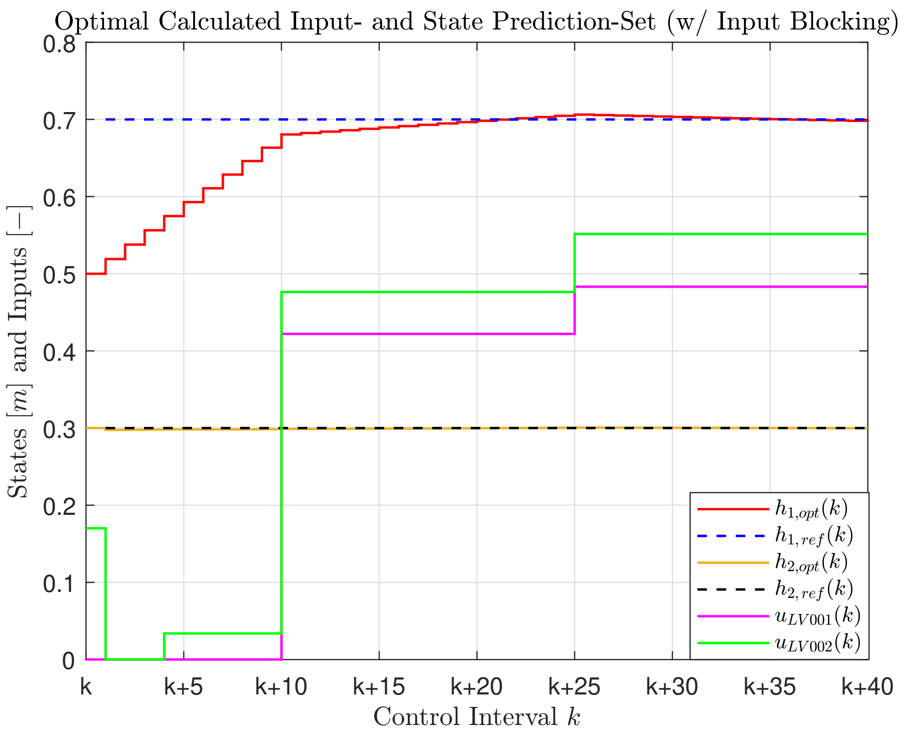


Figure 4.6: Optimal calculated state prediction set $\mathfrak{X}_{opt}(k)$ and control moves $\mathfrak{U}_{opt}(k)$ at control interval k. Prediction horizon is $P = 40$, while the control horizon M is given by (4.39).

```
mpcobj.ControlHorizon = [1 3 6 15 15]      % Control Horizon (Blocking)
```

The choice of the prediction- and control-horizons (and the use of input blocking) for the two-tank system is further discussed in Chapter 5.

4.2.7 QP Solver

The QP solver for the MPC object `mpcobj` uses one of the two main optimization algorithms: Interior-point, and Active-set. The performance of these algorithms will be evaluated in Chapter 5.

The following code shows how to select the two different algorithms in MATLAB:

```
% Active-Set Solver
mpcobj.Optimizer.Algorithm = 'active-set';
-------------------------------------------------
% Interior-Point Solver
mpcobj.Optimizer.Algorithm = 'interior-point';
```

Options such as constraint tolerance, maximum number of iterations when optimizing, and step tolerance are all set to default values, which is shown in Table 4.1. For a detailed discussing on the different solver options, see the MATLAB documentation [23] Section 1.3. To execute the QP optimization in MATLAB, the following syntax is used:

	Interior-point	**Active-set**
`MaxIterations`	50	$4 \cdot (n_c + n_v)$
`ConstraintTolerance`	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$
`UseWarmStart`	N/A	`True`
`OptimalityTolerance`	$1 \cdot 10^{-6}$	N/A
`ComplementarityTolerance`	$1 \cdot 10^{-8}$	N/A
`StepTolerance`	$1 \cdot 10^{-8}$	N/A

Table 4.1: Default options for the MATLAB QP solver. Here, n_c, and n_v are the total number of constraints across the prediction horizon, and the total number of optimization variables across the control horizon, respectively [23].

```
MV = mpcmove(mpcobj , xc , [] , r , v);
```

where:

- `MV` is the optimal calculated manipulated variable ($u_{opt}(k)$).
- `mpcobj` is the MPC object constructed in Section 4.2.1.
- `xc` is a pointer to the current controller state, created using the function `mpcstate` (see [23]).
- `r` is the output reference ($h_{1,ref}$, $h_{2,ref}$).
- `v` is the measured input disturbance (u_{PA001}).

Remark 4.4. The element `[]` in the `mpvmove` function is a placeholder for the measured outputs ($\hat{h}_1$, $\hat{h}_2$). Since there is no need for state estimation (because the states are measured directly), there is no need to provide the measured outputs to the `mpcmove` function.

4.3 Explicit MPC

While the implicit linear MPC (see Section 4.2) performs the QP optimization problem (on line) iteratively at each control interval k, the explicit MPC looks to parameterize the optimization problem, through multiparametric quadratic programming (mpQP), into a piecewise affine control law that can be evaluated, equivalently to a look-up table. The parameterization of the QP problem can be done off line, where the control law is precomputed. Storing the precomputed control law on the system hardware reduces the on line computational efforts, which makes the application of explicit MPC-based controllers more viable for fast sampling systems [38].

When computing the piecewise affine control law, the explicit MPC uses polyhedral computations based on nonnegative least squares (see [38]) to divide the state-space into n_r polyhedral regions, in which, an affine control law is asserted to its respective region. The explicit MPC uses the following piecewise affine control law to evaluate the optimal control move (see [23] Chapter 6):

$$u(k) = F_i \chi(k) + G_i \quad , \quad i = 1, \dots n_r \tag{4.40}$$

where $F_i \in \mathbb{R}^{\mathtt{m} \times n_\chi}$ and $G_i \in \mathbb{R}^{\mathtt{m}}$ are predetermined constants asserted to the polyhedral region i, and $\chi(k) \in \mathbb{R}^{n_\chi}$ is a vector containing the current states and other independent variables that affect the QP problem. In the case of the two-tank system, $\chi(k)$ consists of the variables:

- $h_1(k)$ - Current state of Tank 1.
- $h_2(k)$ - Current state of Tank 2.
- $u_{LV001}(k-1)$ - Previous control move for LV001.
- $u_{LV002}(k-1)$ - Previous control move for LV002.
- $u_{PA001}(k)$ - Current measured disturbance.
- $h_{1,ref}(k)$ - Current output reference for Tank 1.
- $h_{2,ref}(k)$ - Current output reference for Tank 2.
- $\hat{h}_{1,dist}(k)$ - Current disturbance on the measurement of Tank 1.
- $\hat{h}_{2,dist}(k)$ - Current disturbance on the measurement of Tank 2.

As discussed in Chapter 3, it is assumed that the level transmitters LT001 and LT002 measure the fluid level of their respective tank perfectly. Therefore, $\hat{h}_{1,dist}(k) = \hat{h}_{2,dist}(k) = 0, \forall k$.

Since the constants F_i and G_i for $i \in \{1, \dots, n_r\}$ are predetermined, the main on line operation of the explicit MPC is to determine in which polyhedral region $\chi(k)$ resides. This algorithm is described in [23] page 6-5.

Prior to creating an explicit MPC object in MATLAB, the range of each variable in $\chi(k)$ has to be determined. Since the explicit MPC precalculates the piecewise affine control laws off line, it is crucial that no variable in $\chi(k)$ exceed their respective range during operation, as no polyhedral region (and by extension, no affine control law) has been calculated for said case.

To determine the range of $\chi(k)$ in MATLAB, a `range` structure is created, as such:

```
range = generateExplicitRange(mpcobj);
```

which consists of the fields:

```
range =
  a struct with fields:

                State: [1x1 struct]
            Reference: [1x1 struct]
  MeasuredDisturbance: [1x1 struct]
  ManipulatedVariable: [1x1 struct]

range.State
  a struct with fields:

    Min: [4x1 double]
    Max: [4x1 double]

range.Reference
  a struct with fields:

    Min: [2x1 double]
    Max: [2x1 double]

range.MeasuredDisturbance
  a struct with fields:

    Min: [1x1 double]
    Max: [1x1 double]

range.ManipulatedVariable
  a struct with fields:

    Min: [2x1 double]
    Max: [2x1 double]
```

The `range.State` structure contains the bounds on the states from the plant model, and disturbance model, in the following order:

$$\begin{bmatrix} h_1 & h_2 & \hat{h}_{1,dist} & \hat{h}_{2,dist} \end{bmatrix} \tag{4.41}$$

The `range.Reference` structure contains the bounds on the references for the outputs in the following order:

$$\begin{bmatrix} h_{1,ref} & h_{2,ref} \end{bmatrix} \tag{4.42}$$

The `range.MeasuredDisturbance` structure contains the bounds on the measured disturbance in the following order:

$$\begin{bmatrix} u_{PA001} \end{bmatrix} \tag{4.43}$$

The `range.ManipulatedVariable` structure contains the bounds on the manipulated variables in the following order:

$$\begin{bmatrix} u_{LV001} & u_{LV002} \end{bmatrix} \tag{4.44}$$

The aforementioned bounds are set in MATLAB subsequently (see Table 3.1 and 3.2):

```
% Range of the States
range.State.Min(:) = [0.13 0.02 -2 -2];       % Lower Bound
range.State.Max(:) = [1    0.4   2  2];       % Upper Bound
% Range of the References
range.Reference.Min(:) = [0.13 0.02];         % Lower Bound
range.Reference.Max(:) = [1    0.4];          % Upper Bound
% Range of the Measured Disturbance
range.MeasuredDisturbance.Min = 0;            % Lower Bound
range.MeasuredDisturbance.Max = 1;            % Upper Bound
% Range of the Manipulated Variables
range.ManipulatedVariable.Min(:) = [0 0];     % Lower Bound
range.ManipulatedVariable.Max(:) = [1 1];     % Upper Bound
```

Remark 4.5. The lower and upper bounds on the measurement disturbances $\hat{h}_{i,dist}$, $i \in \{1, 2\}$ are determined on an *ad hoc* basis. As previously mentioned, it is assumed perfect measurements when using the level transmitters LT001 and LT002.

With the appropriate ranges defined for all elements in $\chi(k)$, the explicit MPC object can be created in MATLAB:

```
empcobj = generateExplicitMPC(mpcobj,range);
```

where `mpcobj` is the implicit linear MPC object constructed in Section 4.2.

In this project, only default optimization options for explicit MPC generation are used. These default options can be found in the MATLAB documentation.

The piecewise affine control law, given by the explicit MPC, can be evaluated in MATLAB subsequently:

```
MV = mpcmoveExplicit(empcobj, xc, [], r, v);
```

where the variables `MV`, `xc`, `r`, and `v` are defined in Section 4.2.7.

Example 4.3. *As an illustrative example, consider an* `mpcobj` *object (as designed in Section 4.2) with the output weighting matrix Q^{iii} from (4.22), and the horizons $P = 20$ and $M = 2$. Furthermore, consider the vector $\chi(k)$ (with the exception of $h_1(k)$ and $h_2(k)$) to be in the following scenario:*

$$\begin{cases} h_{1,ref}(k) = 0.5, & \hat{h}_{1,dist}(k) = 0, & u_{LV001}(k-1) = 0.5317 \\ h_{2,ref}(k) = 0.3, & \hat{h}_{2,dist}(k) = 0, & u_{LV002}(k-1) = 0.5317, \quad u_{PA001}(k) = 0.8 \end{cases}$$

Given this scenario, the polyhedral partitioning can be graphically represented as a 2-D plot, where $h_1(k)$ and $h_2(k)$ form the two axes (see Figure 4.7). Each region in Figure

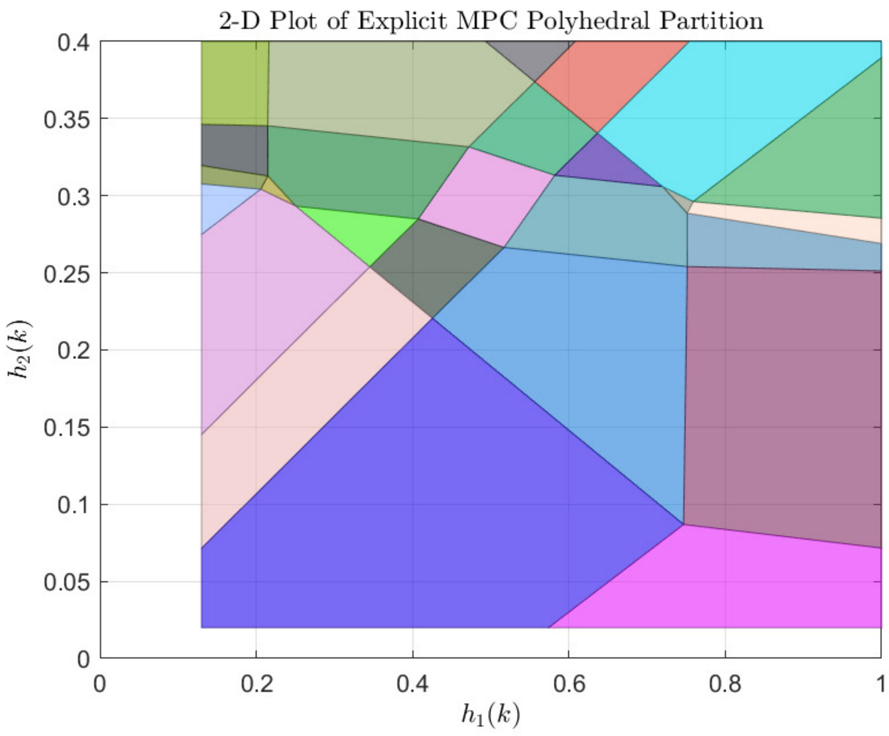


Figure 4.7: Example of the explicit MPC polyhedral partitioning.

4.7 is asserted corresponding constants F_i and G_i, where i denotes the partition number. For this particular example, the number of polyhedral regions is 47. Note that all 47 regions are not visible in Figure 4.7, given that seven out of the nine variables affecting the QP problem are set to constants for graphical purposes. Note also the white spaces in Figure 4.7, which show how no polyhedral regions exist outside of the bounds on $\chi(k)$.

4.4 Adaptive MPC

Adaptive MPC uses the same MPC-based controller as the linear MPC designed in Section 4.2, with one key difference: the prediction model and nominal values update on every control interval.

It was shown in Chapter 3 that a linear approximation (in the vicinity of a nominal solution $\mathfrak{NS}$) of the nonlinear two-tank model can be obtained by means of linearization (see the equations $\{(3.50), (3.51)\}$). It was also shown that the validity of this linear approximation degrades, as the system is driven farther away from the nominal solution $\mathfrak{NS}$ (see Figure 3.10).

For the linear MPC, this means that the prediction model (see Section 4.2.1) will make less accurate predictions over the prediction horizon when the system is operating at a point which is not in the neighborhood of the nominal solution $\mathfrak{NS}$, effectively increasing the modeling error.

Hence, the following compromise arises between two competing factors:

(i) Use a linear MPC with an accurate prediction model, but, the control of the system is limited to the vicinity of the nominal solution $\mathfrak{NS}$.

Or

(ii) Use a linear MPC where the control is *not* limited to the vicinity of the nominal solution $\mathfrak{NS}$, but, there will be increasing modeling error as the system operates farther away from the nominal solution $\mathfrak{NS}$.

One way to circumvent the issue presented above, is by using adaptive MPC. The idea behind adaptive MPC is to (at every control interval) update the prediction model and the nominal solution $\mathfrak{NS}$ *after* solving the QP optimization problem. When the QP optimization problem is solved, the current optimal control input $u_{opt}(k)$ is calculated. Furthermore, the states $h_i(k)$, $i \in \{1, 2\}$, and the measured disturbance $u_{PA001}(k)$ are available at control interval k. Thus, a nominal solution $\mathfrak{NS}_k$, $k = 0, 1, \ldots$ can be constructed, as such:

$$\underbrace{\tilde{h}_1 = h_1(k), \quad \tilde{h}_2 = h_2(k), \quad \tilde{u}_{PA001} = u_{PA001}(k), \quad \tilde{u}_{LV001} = u_{opt,1}(k), \quad \tilde{u}_{LV002} = u_{opt,2}(k)}_{\mathfrak{NS}_k} \tag{4.45}$$

Given the nominal solution $\mathfrak{NS}_k$ (at control interval k), the prediction model is updated by, firstly, calculating the following continuous time state-space representation (see

Section 3.3):

$$\begin{cases} \begin{bmatrix} \Delta\dot{h}_1(t) \\ \Delta\dot{h}_2(t) \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathfrak{f}_1}{\partial h_1} & \frac{\partial \mathfrak{f}_1}{\partial h_2} \\ \frac{\partial \mathfrak{f}_2}{\partial h_1} & \frac{\partial \mathfrak{f}_2}{\partial h_2} \end{bmatrix}_{\mathfrak{NS}_k} \cdot \begin{bmatrix} \Delta h_1(t) \\ \Delta h_2(t) \end{bmatrix} + \begin{bmatrix} \frac{\partial \mathfrak{f}_1}{\partial u_{LV001}} & \frac{\partial \mathfrak{f}_1}{\partial u_{LV002}} & \frac{\partial \mathfrak{f}_1}{\partial u_{PA001}} \\ \frac{\partial \mathfrak{f}_2}{\partial u_{LV001}} & \frac{\partial \mathfrak{f}_2}{\partial u_{LV002}} & \frac{\partial \mathfrak{f}_2}{\partial u_{PA001}} \end{bmatrix}_{\mathfrak{NS}_k} \cdot \begin{bmatrix} \Delta u_{LV001}(t) \\ \Delta u_{LV002}(t) \\ \Delta u_{PA001}(t) \end{bmatrix} \\ \begin{bmatrix} \Delta\hat{h}_1(t) \\ \Delta\hat{h}_2(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(t) \\ \Delta h_2(t) \end{bmatrix} \end{cases} \tag{4.46}$$

and, secondly, discretizing the model (4.46) using the ZOH method with sampling time t_s (see Section 3.5.2).

This way, it is ensured that the system will operate within the neighborhood of the nominal solution $\mathfrak{NS}_k$ at control interval k, thus, minimizing the introduced modeling error.

The following pseudocode illustrates the workflow of an adaptive MPC, as it is implemented in MATLAB:

Algorithm 4.1 Workflow of adaptive MPC

```
1: $k \leftarrow k$    ▷ Current control interval
2: $\hat{h}_i(k) \leftarrow LT00i(k)$, $i \in \{1,2\}$    ▷ Measure outputs
3: $h_i(k) \leftarrow \hat{h}_i(k)$, $i \in \{1,2\}$    ▷ Estimate states
4: $u_{PA001}(k) \leftarrow FT001(k)$    ▷ Measure input disturbance
5: Retrieve previous control moves $u_{LV00i}(k-1)$, $i \in \{1,2\}$ from memory
6: $u_{LV00i}(k) \leftarrow u_{opt,i}(k)$, $i \in \{1,2\}$    ▷ Solve QP optimization problem
7: Update current nominal solution $\mathfrak{NS}_k$
8: Linearize nonlinear model about $\mathfrak{NS}_k$
9: Discretize linear continuous time model
10: Insert $\mathfrak{NS}_k$ and updated prediction model in the MPC object
11: $k \leftarrow k+1$    ▷ Next control interval
```

The QP optimization problem, for the adaptive MPC, can be solved in MATLAB subsequently:

```
MV = mpcmoveAdaptive(mpcobj, xc, sys, NS_k, [], r, v)
```

where the variables `MV`, `mpcobj`, `xc`, `r`, and `v` are defined in Section 4.2.7. The `sys` object is the updated prediction model, which is calculated using the `AdaptiveSys` function (see Appendix I.1). The variable `NS_k` is the current nominal solution $\mathfrak{NS}_k$ plus the term $DX \triangleq h(k) - h(k-1)$, which is defined as the rate of change in the states from one control interval to the next.

4.5 Nonlinear MPC

Contrary to the MPC-based controllers discussed thus far, the nonlinear MPC is not restricted to a linear prediction model, nor linear constraints, and nor a standard quadratic objective function (see Section 2.3).

For this project, the nonlinear MPC will use the same linear constraints, and the same quadratic objective function, as the other MPC-based controllers discussed so far in this chapter. However, the prediction model will be based on the nonlinear two-tank model $\{(3.29), (3.30)\}$, which will be solved by means of the Euler method (see Section 3.5.3).

The following code shows how to construct a nonlinear MPC in MATLAB:

```
nlobj = nlmpc(nx,ny,'MV',mvIndex,'MD',mdIndex); % Nonlinear MPC object
```

where `nx`, `ny`, `mvIndex`, and `mdIndex` are the number of states $\mathfrak{n}$, the number of outputs $\mathfrak{p}$, the indices of the manipulated variables in the input vector, and the index of the measured disturbance in the input vector, respectively.

The nonlinear MPC specifications such as weighting matrices (Q, R, and S), scaling factors (λ_{h_1} and λ_{h_2}), prediction horizon (P), control horizon (M) and constraint sets ($\mathcal{X}$, $\mathcal{U}$, and $\delta\mathcal{U}$) are all designed equivalently to the MPC specifications used in the previously discussed MPC-based controllers (see Section 4.2 in particular, for a review on these specifications and how to program them in MATLAB).

The sampling time of the MPC, and the prediction model, are specified subsequently:

```
% Sampling time
nlobj.Ts = ts;

% Nonlinear DT State Function
nlobj.Model.StateFcn = 'tankDT';
nlobj.Model.IsContinuousTime = false;
nlobj.Model.NumberOfParameters = 1;

% DT Output Function
nlobj.Model.OutputFcn = 'tankOutputFcn';

% Output Jacobian
nlobj.Jacobian.OutputFcn = @(x,u,Ts) [1 0; 0 1];
```

where `ts`, `'tankDT'`, and `'tankOutputFcn'` are the sampling period t_s, function call to the Euler method, and function call to the output equations, respectively. The functions `'tankDT'` and `'tankOutputFcn'` can be found in Appendix I.1.

Since the Euler method is used as a prediction model, the flag `IsContinuousTime` is set to `false` (code line 6). Additionally, the Euler method requires a step size h, which is why the variable `NumberOfParameters` is set to 1 (code line 7). In this case, the Euler step size is set to the sampling time of the nonlinear MPC, i.e., $h = t_s$.

Recall that the output equations are given by:

$$\underbrace{\begin{bmatrix} \hat{h}_1(t) \\ \hat{h}_2(t) \end{bmatrix}}_{\hat{h}(t)} = \underbrace{\begin{bmatrix} \mathfrak{g}_1(h_1(t),\, u_{LV001}(t),\, u_{PA001}(t)) \\ \mathfrak{g}_2(h_1(t),\, h_2(t),\, u_{LV001}(t),\, u_{LV002}(t)) \end{bmatrix}}_{\mathfrak{g}(h_1(t), h_2(t), u_{LV001}(t), u_{LV002}(t), u_{PA001}(t))} = \underbrace{\begin{bmatrix} h_1(t) \\ h_2(t) \end{bmatrix}}_{h(t)} \tag{4.47}$$

Given (4.47), the Jacobian matrix:

$$\left.\frac{D\mathfrak{g}}{Dh}\right|_{\mathfrak{NS}} = C \tag{4.48}$$

evaluated at some arbitrary nominal solution:

$$\underbrace{\tilde{h}_1, \quad \tilde{h}_2, \quad \tilde{u}_{LV001}, \quad \tilde{u}_{LV002}, \quad \tilde{u}_{PA001}}_{\mathfrak{NS}} \tag{4.49}$$

results in:

$$\left.\frac{D\mathfrak{g}}{Dh}\right|_{\mathfrak{NS}} = C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.50}$$

regardless of the arbitrary nominal solution $\mathfrak{NS}$. Therefore, to simplify computational efforts, the Jacobian of the output function (code line 13) can be set to the identity matrix $I \in \mathbb{R}^{\mathfrak{p}}$.

Chapter 5

Simulations

The ensuing chapter will look towards implementing and evaluating the different MPC-based controllers (designed in Chapter 4) in a simulation environment. Multiple simulations will be carried out, where the effects of different MPC specifications will be studied. Additionally, this chapter will look to compare the performance of the controllers against each other, with performance measures such as the total quadratic objective function and code execution time.

5.1 Simulation Environment

The goal of the following experiments is to analyse two main attributes of the controllers: (i) set-point tracking, and (ii) disturbance rejection. To this end, every simulation will be subjected to a rectangular pulse (of length $70\,s$) in the references, and the measured disturbance, respectively. The following pseudocode highlights the procedure of the simulations: The simulations will last a total of $400\,s$, which, with a sampling period of

Algorithm 5.1 Simulation procedure

```
1: $\texttt{TIME} = 0\,s \implies \texttt{SIM} \leftarrow \mathfrak{NG}$    ▷ Start of simulation at $\mathfrak{NG}$
2: $\texttt{TIME} \geq 50\,s \implies h_{1,ref} \uparrow$    ▷ Start of positive rectangular pulse
3: $\texttt{TIME} \geq 120\,s \implies h_{1,ref} \downarrow$    ▷ End of positive rectangular pulse
4: $\texttt{TIME} \geq 150\,s \implies h_{2,ref} \downarrow$    ▷ Start of negative rectangular pulse
5: $\texttt{TIME} \geq 220\,s \implies h_{2,ref} \uparrow$    ▷ End of negative rectangular pulse
6: $\texttt{TIME} \geq 250\,s \implies u_{PA001} \downarrow$    ▷ Start of negative rectangular pulse
7: $\texttt{TIME} \geq 320\,s \implies u_{PA001} \uparrow$    ▷ End of negative rectangular pulse
8: $\texttt{TIME} = 400\,s \implies \texttt{SIM} \leftarrow \texttt{end}$    ▷ End of simulation
```

$t_s = 0.5\,s$, equates to 800 iterations. Every simulation will start at the nominal solution

$\mathfrak{NS}$, which was defined in Section 4.2.1, and is restated below:

$$\underbrace{\tilde{h}_1 = 0.5, \quad \tilde{h}_2 = 0.3, \quad \tilde{u}_{PA001} = 0.8, \quad \tilde{u}_{LV001} = 0.5317, \quad \tilde{u}_{LV002} = 0.5317}_{\mathfrak{NS}} \tag{5.1}$$

5.1.1 Performance measures

The two performance measures of high significance, when comparing the controllers, are the total quadratic objective cost, and the code execution time.

The total quadratic objective function is calculated similarly to the quadratic objective function used by the QP solver (see Section 4.2.2). However, rather than calculating the quadratic cost over the prediction horizon P, the total quadratic cost is calculated over the entire span of the simulation. I.e.:

$$J_{total} = J_{y,total} + J_{u,total} + J_{\delta u,total} \tag{5.2}$$

where:

$$J_{y,total} = \sum_{i=1}^{800} \Big\{ (r(i) - y(i))^T \cdot Q^2 \cdot (r(i) - y(i)) \Big\} \tag{5.3}$$

$$J_{u,total} = \sum_{i=1}^{800} \Big\{ (u_{target}(i) - u(i))^T \cdot R^2 \cdot (u_{target}(i) - u(i)) \Big\} \tag{5.4}$$

$$J_{\delta u,total} = \sum_{i=2}^{800} \Big\{ (u(i) - u(i-1))^T \cdot S^2 \cdot (u(i) - u(i-1)) \Big\} \tag{5.5}$$

As mentioned, one simulation consists of 800 iterations total. Consequently, the QP solver is called upon a total of 800 times. The standard MATLAB stopwatch timer is used to log the elapsed time for all of the 800 QP solver calls, respectively. From this data, additional information can be retired, such as: mean time, max/min time, upper/lower quantile, etc.

The following code shows how the SQP/QP solver execution times for the different MPCs are logged in MATLAB:

```matlab
% Linear MPC
tic                         % Start stopwatch timer
MV = mpcmove(mpcobj,...
      xc, [], r, v);        % Call QP solver
time_elapsed = toc;         % Read elapsed time from stopwatch

% Explicit MPC
tic                                  % Start stopwatch timer
MV = mpcmoveExplicit(empcobj,...
      xc, [], r, v);            % Call QP solver
time_elapsed = toc;             % Read elapsed time from stopwatch

% Adaptive MPC
tic                            % Start stopwatch timer
MV = mpcmoveAdaptive(mpcobj,...
      xc, sys, NS_k, [], r, v)  % Call QP solver
time_elapsed = toc;            % Read elapsed time from stopwatch

% Nonlinear MPC
tic                           % Start stopwatch timer
MV = nlmpcmove(nlobj,...
      x, MV, r, v, nloptions); % Call SQP solver
time_elapsed = toc;           % Read elapsed time from stopwatch
```

Recall that the adaptive MPC, in addition to calling the QP solver, also has to calculate the updated prediction model. The execution time of this process is logged in a similar fashion.

In order to achieve a representative value of these performance measures, the simulation (i.e., Algorithm 5.1) is conducted 10 separate times, for each controller, so that a performance measure mean can be used when comparing the controllers. E.g., when finding a representative value on the total quadratic cost for the linear MPC, the following equation is used:

$$J_{total,\mu} = \frac{1}{10}\sum_{i=1}^{10} J_{total,i} \tag{5.6}$$

where $J_{total,\mu}$ and $J_{total,i}$ are the mean total quadratic cost and the total quadratic cost for simulation i, respectively. The same logic applies for the representative mean value of the total code execution time.

For an overview on the software and hardware that the following experiments are conducted on, see Section 4.1.

5.2 Optimal Prediction and Control Horizons

While all the necessary MPC specifications were discussed in Chapter 4, no explicit choice was made for the prediction horizon P and control horizon M. The goal of this section is to determine a pair (P, M) that provides satisfactory control of the two-tank system, within a satisfactory limit on the code execution time. The pair (P, M) that is deemed ideal in this section will be used for the subsequent parts of the project.

Since the explicit- and adaptive-MPC utilize the linear MPC object `mpcobj` in their creation, respectively, the search for an ideal pair $(P,\ M)$ will be done exclusively on the linear MPC. Additionally, the output weighting matrix Q^{iii} (see (4.22)) will be used for the subsequent parts of this chapter, unless otherwise is explicitly stated, as this is the predominant engineering priority in this project.

Now, to find the ideal pair $(P,\ M)$, one simulation (see Algorithm 5.1) is conducted for a variety of pairs $(P,\ M)$ that all satisfy the necessary condition:

$$P \geq M \geq 1, \quad \{P,\ M\} \in \mathbb{N} \tag{5.7}$$

Each simulation is then compared against each other, based on the performance measures described in Section 5.1.1.

Given an upper limit on the prediction horizon P_U, then, the total number of pairs $(P,\ M)$ (denoted by Ξ) that satisfy (5.7), and $P \leq P_U$, is given by:

$$\Xi = \sum_{n=1}^{P_U} n = \frac{P_U(P_U+1)}{2} \tag{5.8}$$

which is a *triangular number.*

Consider the following example:

Example 5.1. *Suppose that the larges possible prediction horizon is $P_U = 3$. Then, the total number of possible pairs $(P,\ M)$ that satisfy (5.7), and $P \leq 3$, is:*

$$\Xi = \sum_{n=1}^{3} n = \frac{3(3+1)}{2} = 6$$

The pairs $(P,\ M)$ that do not violate the aforementioned conditions are:

$$\begin{cases} (1,\ 1), & (2,\ 1), & (2,\ 2) \\ (3,\ 1), & (3,\ 2), & (3,\ 3) \end{cases}$$

In the case of this experiment $P_U = 20$, which, by using (5.8), results in $\Xi = 210$ unique $(P,\ M)$ combinations. The performance measures, i.e., the code execution time and the quadratic cost, of the 210 simulations are presented as heat maps in Figure 5.1 and Figure 5.2, respectively. Firstly, consider Figure 5.1. As expected, the total execution time generally[1] tends to increase, as the QP optimization problem becomes more complex, i.e., as P and M become larger in value. The $(P,\ M)$ pair with the fastest simulation time is $(3,\ 2)$, which completed one simulation in $0.636\,s$. On the other hand, the slowest

[1]There are some outliers to this statement, e.g., the pair $(P,\ M) = (1,\ 1)$ and $(P,\ M) = (12,\ 4)$.

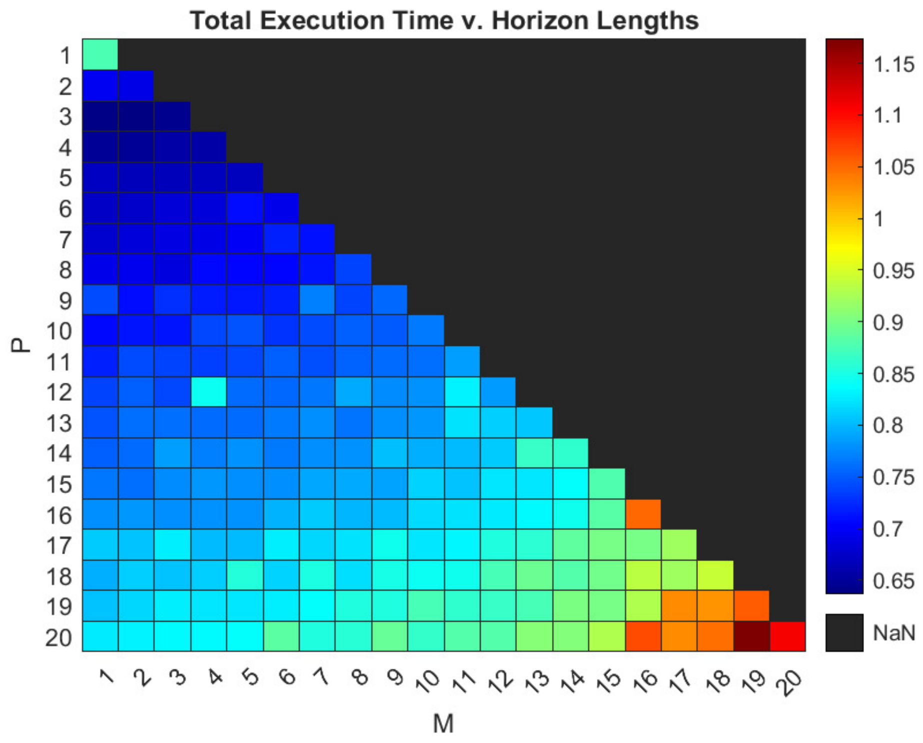


Figure 5.1: Heat map of the total code execution time for the 210 simulations with unique (P, M) combinations.

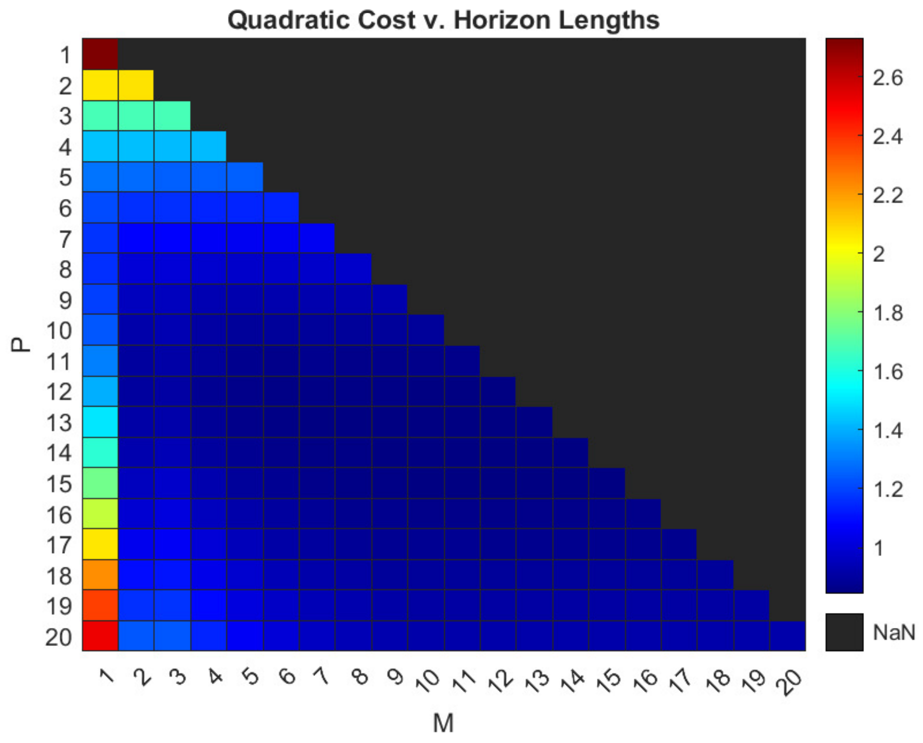


Figure 5.2: Heat map of the total quadratic cost for the 210 simulations with unique (P, M) combinations.

$(P,\ M)$ pair is (20, 19), with a total simulation time of $1.17\,s$. Recall that one simulation consists of 800 iterations, meaning that the average time spent on one QP solver call, for the slowest $(P,\ M)$ pair, is:

$$\frac{1.17\,s}{800} = 1.46\,ms$$

which is a satisfactory code execution time, considering the fact that the sampling period is $t_s = 0.5\,s$.

Secondly, consider Figure 5.2. Clearly, the worst $(P,\ M)$ pair that resulted in the largest total quadratic cost is (1, 1), which is the equivalent of predicting $0.5\,s$ into the future (see (4.37)). This results in a short-sighted MPC policy that does not fully capture the dynamics of the system when predicting the optimal trajectory. As P and M become larger, the total quadratic cost diminishes, however, this only applies up to a certain point. To illustrate this, the rows $1 \le P \le 9$, and columns $1 \le M \le 7$, are removed (see Figure 5.3). Note how, in Figure 5.3, the dominant factor affecting the total quadratic

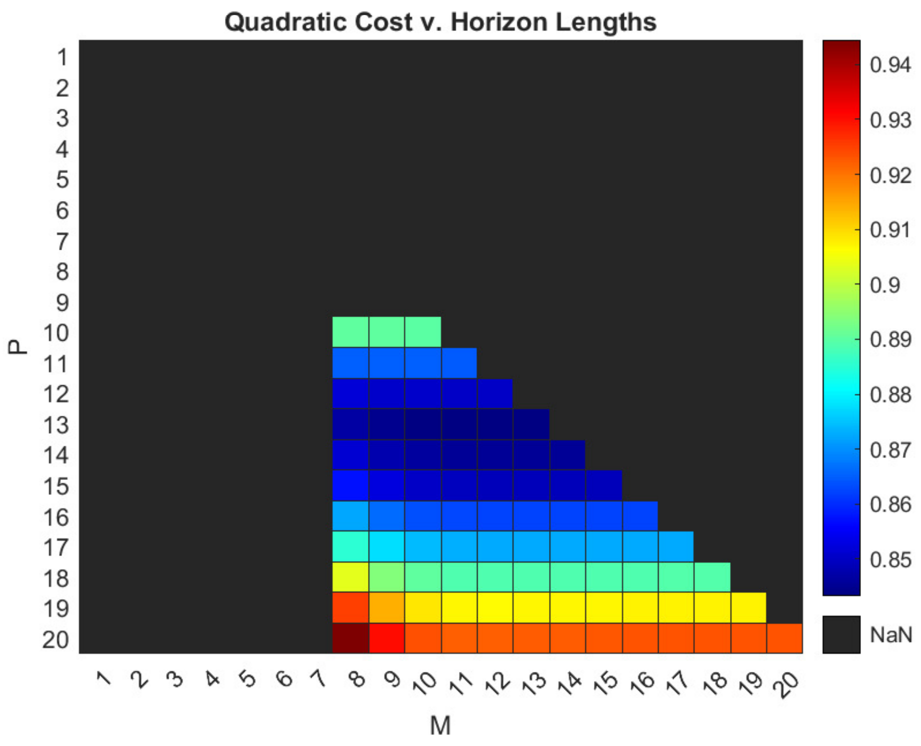


Figure 5.3: Heat map of the total quadratic cost from a highlighted segment of Figure 5.2.

cost is the prediction horizon P. For example, at row $P = 18$, the difference in the total quadratic cost, between the elements in this row, remain relatively insignificant after column $M = 10$. I.e., increasing M any further makes the QP problem more complex, yet, the total quadratic cost remain relatively unaffected. This concept was also illustrated in Section 4.2.6.

In fact, the (P, M) pair that performed the best, with respect to the total quadratic cost, is (13, 13), which resulted in $J_{total} = 0.843$. Further increasing P and M only results in the quadratic cost to grow (as shown in Figure 5.3), as well as the total code execution time (as shown in Figure 5.1).

Following the discussion above, the MPC specifications (P, M) are determined to be (13, 13), as this resulted in satisfactory control of the system, within a satisfactory time limit on the code execution time.

5.3 Linear MPC

The following section looks to conduct 10 separate simulations of Algorithm 5.1, with the linear MPC. The MPC specifications are:

$$P = M = 13,$$

and

$$Q^{iii} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

with the QP solver programmed to use the Active-set method. The other MPC specifications, such as constraints, scaling factors, weighting matrices, etc., are identical to the linear MPC designed in Section 4.2 of Chapter 4.

Firstly, consider one instance of the 10 separate simulations, which is shown in Figure 5.4. Clearly, the linear MPC is capable of tracking the set-points, even when subjected to a square pulse in the measured disturbance. Furthermore, there are several characteristics to the closed-loop response in Figure 5.4 worth mentioning:

(i) Note that the closed-loop system responds *prior* to the rectangular pulse in the references and the measured disturbance, respectively. This characteristic trait is attributed to the predictive nature of the MPC policy. If future set-points and measured disturbances are known, then, the MPC will use this information in the P-step prediction horizon, thus, deciding appropriate control moves ahead of time. This is often referred to as *signal previewing* or *look-ahead* control.

(ii) Recall that the two-tank system is a coupled system, where the outlet of Tank 1 serves as the inlet for Tank 2. In particular, recall that a step in u_{LV001} causes a dynamic response in h_2 (see Figure 3.12, subplot (2,1)). However, this coupling is not present in Figure 5.4.

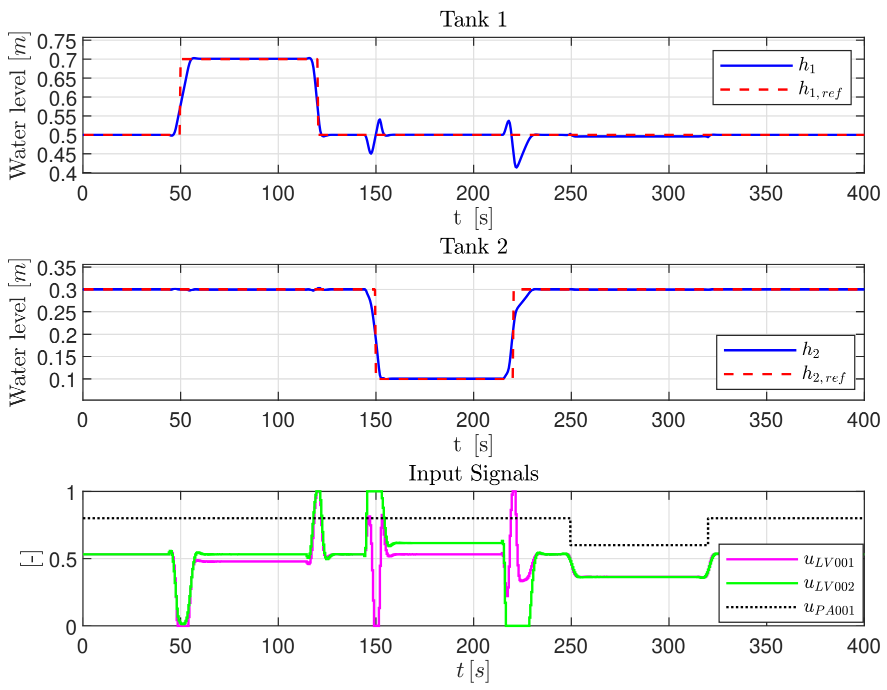


Figure 5.4: Closed-loop response w/ linear MPC.

Note how, at times $t \approx 50\,s$ and $t \approx 125\,s$, the rapid changes in u_{LV001} does not significantly affect h_2. This is due to the decoupling properties of MPC-based controllers.

(iii) As the rectangular pulse in $h_{2,ref}$ transitions (i.e., at $t \approx 150\,s$ and $t \approx 225\,s$), two spikes occur in h_1 at the transitioning instances, respectively. This behaviour is directly linked to the chosen output weighting matrix Q^{iii}, which specifies that the two outputs are to be weighted equally. As the set-point $h_{2,ref}$ transitions, there is a sudden error between the output h_2 and its corresponding set-point $h_{2,ref}$. To reduce this sudden error as quickly as possible, the MPC deliberately allows small spikes in h_1.

For instance, consider the spike at $t \approx 225\,s$. At this point, the output h_2 is far below the new set-point $h_{2,ref}$. The fastest way to increase the fluid level in Tank 2, is by completely closing its outlet valve LV002, and, simultaneously, completely opening its inlet valve LV001. This explains why the fluid level in Tank 1 decreases, which the MPC prepares for, by slightly increasing the fluid level in Tank 1 beforehand. The same concept applies for the spike at $t \approx 150\,s$.

(iv) Lastly, note that all of the constraints are satisfied, as none of the valve signals exceed the input constraint set $\mathcal{U}$ (see Figure 4.1).

The execution time of every 800 QP solver call is logged, for each respective simulation instance. This data is presented in Figure 5.5. For a brief introduction to box plots, see Appendix D.

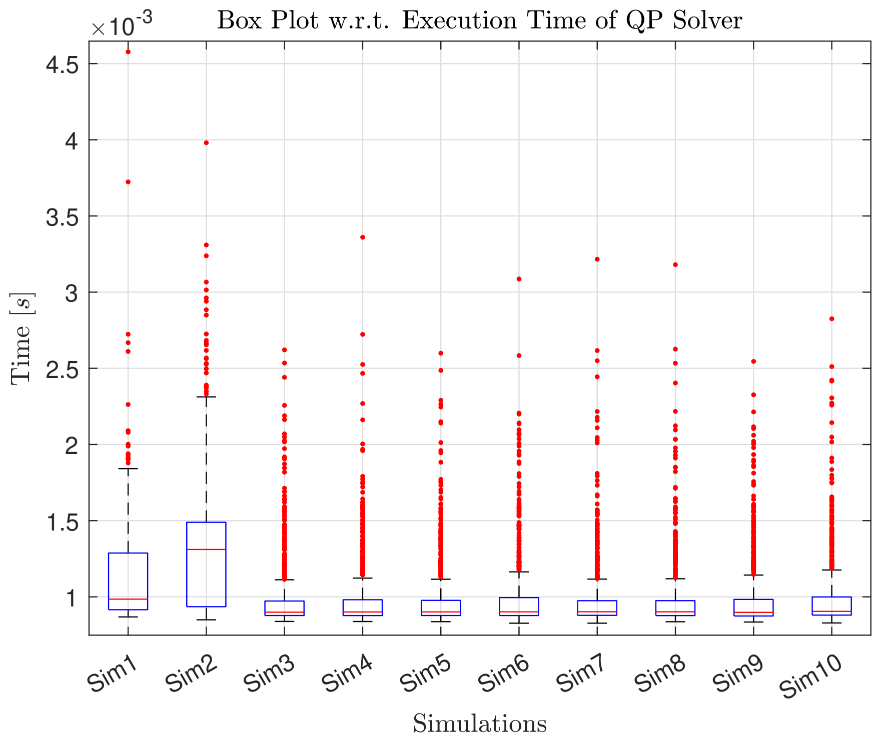


Figure 5.5: Box plot of recorded elapsed time for every 800 QP solver called upon by the linear MPC. Each box represent the data belonging to their respective simulation instance.

Remark 5.1. One outlier (from `Sim1` with a value of 39.41 ms) is **not** included in the data-set presented in Figure 5.5. This is solely due to graphical reasons, as this outlier eclipses all the other data-points, making the scaling of the plot difficult to read.

Evidently from Figure 5.5, the *median* code execution time for *one* QP solver call from the linear MPC is slightly below 1 ms. Additional information that is retrieved from this data-set is presented in Table 5.1.

Observe the similarities, both in the amount and the location, of the outliers between the 10 different simulations in Figure 5.5. Rather than a coincidence, these similarities are more a bi product of the way Algorithm 5.1 is staged. Note how, for the majority of the simulation in Figure 5.4, the closed-loop response is static.

When the system is static, and the outputs follow the set-points, then, the QP solver will quickly converge to the optimal solution, as this is the solution already applied to the valves (i.e., the steady state values). However, when the system is not following the set-points, such as immediately after a square pulse in the set-point or disturbance,

	Median	Max/Min	Total	Outliers	75th/25th Perc.	Upper/Lower Adj.
`Sim1`	$0.98\,ms$	$^{39.41}/_{0.86}\,ms$	$0.93\,s$	18	$^{1.28}/_{0.91}\,ms$	$^{1.84}/_{0.86}\,ms$
`Sim2`	$1.31\,ms$	$^{3.98}/_{0.84}\,ms$	$1.04\,s$	28	$^{1.48}/_{0.93}\,ms$	$^{2.31}/_{0.84}\,ms$
`Sim3`	$0.89\,ms$	$^{2.62}/_{0.83}\,ms$	$0.79\,s$	124	$^{0.97}/_{0.87}\,ms$	$^{1.11}/_{0.83}\,ms$
`Sim4`	$0.90\,ms$	$^{3.36}/_{0.83}\,ms$	$0.79\,s$	115	$^{0.98}/_{0.87}\,ms$	$^{1.12}/_{0.83}\,ms$
`Sim5`	$0.90\,ms$	$^{2.59}/_{0.83}\,ms$	$0.78\,s$	122	$^{0.97}/_{0.87}\,ms$	$^{1.11}/_{0.83}\,ms$
`Sim6`	$0.90\,ms$	$^{3.08}/_{0.82}\,ms$	$0.80\,s$	116	$^{0.99}/_{0.87}\,ms$	$^{1.16}/_{0.82}\,ms$
`Sim7`	$0.90\,ms$	$^{3.21}/_{0.82}\,ms$	$0.79\,s$	121	$^{0.97}/_{0.87}\,ms$	$^{1.11}/_{0.82}\,ms$
`Sim8`	$0.90\,ms$	$^{3.18}/_{0.83}\,ms$	$0.79\,s$	120	$^{0.97}/_{0.87}\,ms$	$^{1.11}/_{0.83}\,ms$
`Sim9`	$0.89\,ms$	$^{2.54}/_{0.83}\,ms$	$0.79\,s$	123	$^{0.98}/_{0.87}\,ms$	$^{1.14}/_{0.83}\,ms$
`Sim10`	$0.90\,ms$	$^{2.82}/_{0.82}\,ms$	$0.80\,s$	115	$^{0.99}/_{0.88}\,ms$	$^{1.17}/_{0.82}\,ms$

Table 5.1: Statistical properties of the data-set from Figure 5.5. Here, **Perc.** and **Adj.** are abbreviations for **Percentile** and **Adjacent**, respectively.

then, the QP solver may take longer to converge to the optimal solution. This concept is illustrated in Figure 5.6, where, at control interval k, the code execution time of the QP solver, *and* the total number of iterations that was necessary to converge to the optimal solution, are plotted. Note how the density of outliers increase at the distinct control

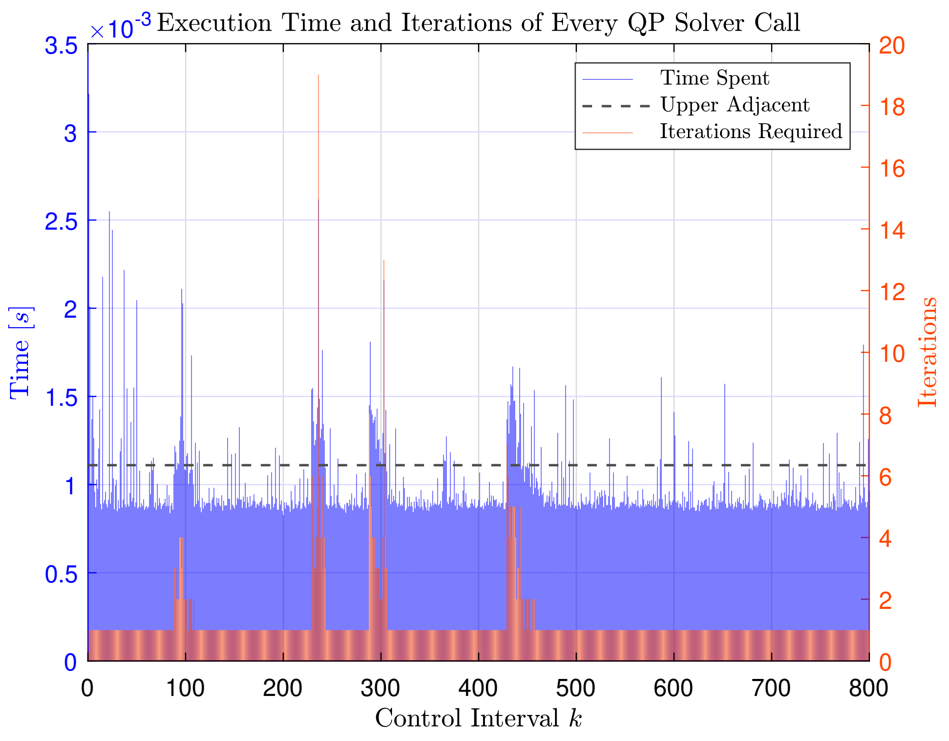


Figure 5.6: Stem plot illustrating the execution time, and the number of necessary iterations, used by the 800 QP solver calls, respectively, from the `Sim7` data-set.

intervals: $90 \leq k \leq 110$, $230 \leq k \leq 240$, $290 \leq k \leq 310$, and $430 \leq k \leq 450$. In the time domain, these control intervals equate to the transitioning periods of the square pulses in the set-points, shown in Figure 5.4. I.e., multiplying the control intervals with the sampling period $t_s = 0.5\,s$, yields the corresponding time intervals: $45\,s \leq t \leq 55\,s$,

$115\,s \leq t \leq 120\,s$, $145\,s \leq t \leq 155\,s$, and $215\,s \leq t \leq 225\,s$. This explains the similarities in the amounts, and the locations, of the outliers between the data-sets shown in Figure 5.5.

Recall that the MPC control policy is based on the moving horizon policy, where one finite horizon optimization problem (here referred to as one QP solver call) has to be solved on every control interval. For this control policy to work, it is required that the MPC is able to compute this finite horizon optimization problem *within* the duration of the current control interval k, $\forall k$. The duration of one control interval is equivalent to the sampling period, which, for this experiment, is set to be $t_s = 0.5\,s$.

During this experiment, the longest computational time (in milliseconds), used by the linear MPC, to complete one QP solver call is $39.41\,ms$ (see Table 5.1, row: `Sim1`, column: **Max/Min**), which is within one control interval of length $t_s = 0.5\,s$. Conclusively, based on the data-set in Table 5.1, this linear MPC has a feasible control policy, where the MPC is able to compute the finite horizon optimization problem within every control interval k of length $t_s = 0.5\,s$.

The representative mean value for the total quadratic cost of the linear MPC is calculated to be:

$$J_{total,\mu} = \underbrace{0.829}_{J_{y,total,\mu}} + \underbrace{0.013}_{J_{\delta u,total,\mu}} + \underbrace{0.000}_{J_{u,total,\mu}} = 0.842 \tag{5.9}$$

The quadratic cost of the 10 respective simulations are identical to each other, and to the mean values given in (5.9).

5.3.1 Alternative Closed-Loop Response

Tank 1 Priority

Suppose that the desired operation of the two-tank system is to prioritize Tank 1. I.e., the paramount objective of the closed-loop system is to keep h_1 at the set-point $h_{1,ref}$ at all times. In the case of the closed-loop system shown in Figure 5.4, albeit aiding h_2 reaching its set-point faster, the spikes occurring in h_1 at times $t \approx 150\,s$ and $t \approx 225\,s$, are detrimental to the paramount objective. To resolve this, the output weighting matrix Q is adjusted according to the desired objective:

$$Q^i = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}$$

Assuming identical MPC specifications to those discussed thus far, with the exception of the updated output weighting matrix Q^i, then, simulating this closed-loop system (Algorithm 5.1) results in the response shown in Figure 5.7.

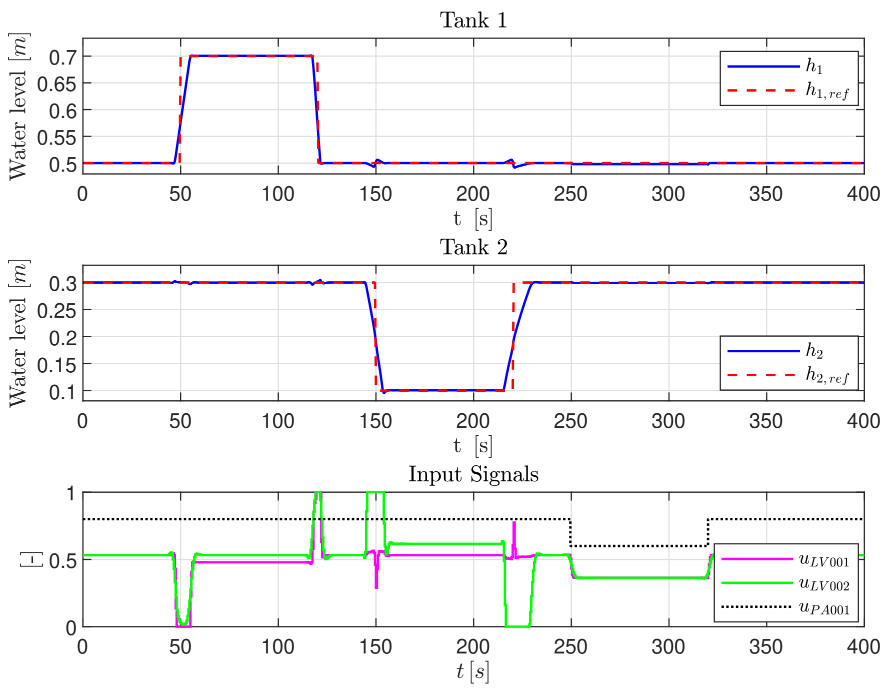


Figure 5.7: Closed-loop response w/ linear MPC. The paramount objective is to prioritize set-point tracking in Tank 1.

While still present, the spikes in h_1 (at times $t \approx 150\,s$ and $t \approx 225\,s$) are significantly smaller in Figure 5.7, compared to the closed-loop response in Figure 5.4. Note also the slower response in h_2, without the aid of the larger spikes in h_1. However, this is expected, as Tank 2 is of low priority for this desired operation.

Tank 2 Priority

Conversely, suppose that the paramount objective is to keep h_2 at the set-point $h_{2,ref}$ at all times. To obtain this desired closed-loop response, the output weighting matrix Q is set to be:

$$Q^{ii} = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}$$

The simulation of this closed-loop system is shown in Figure 5.8. As expected, Figure 5.8 shows that the closed-loop system responds quickly to set-points changes in Tank 2. However, this fast control of h_2 causes h_1 to experience larger spikes.

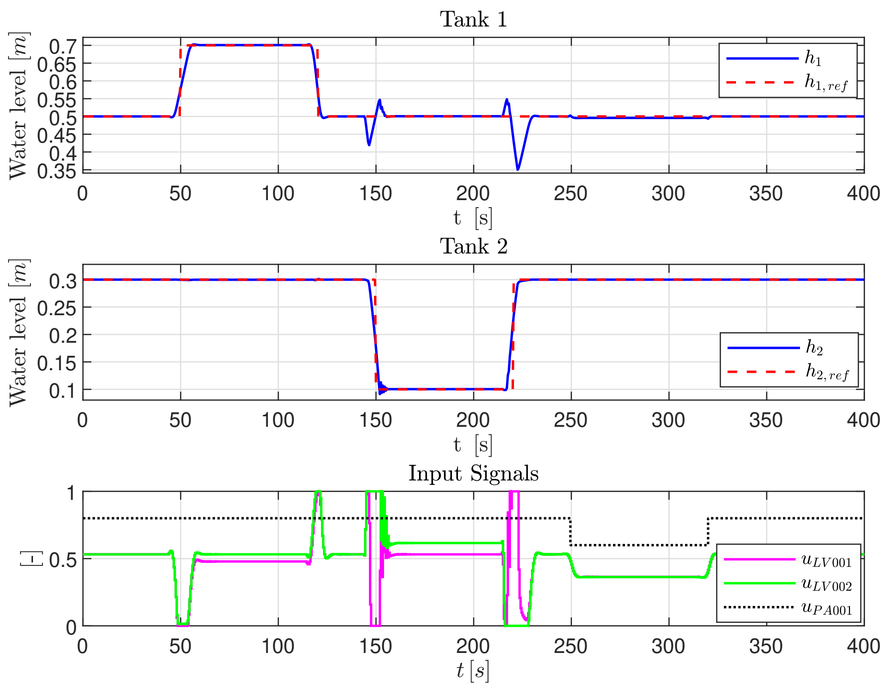


Figure 5.8: Closed-loop response w/ linear MPC. The paramount objective is to prioritize set-point tracking in Tank 2.

5.3.2 Alternative Optimization Solver: The Interior-Point Method

As mentioned in Section 4.2.7 of Chapter 4, the Interior-point optimization algorithm can be used to solve the QP problem. This part looks to perform the same experiments discussed thus far, however, rather than using the active-set method, the linear MPC is programmed to use the interior-point method.

The box plots containing the data-set from 10 separate simulations, where the code execution time for the QP solver calls are logged, are shown in Figure 5.9 (similarly to Figure 5.5).

Remark 5.2. Two outliers (from `Sim1` and `Sim3` with a value of $175.16\,ms$ and $38.81\,ms$, respectively) are **not** included in the data-set presented in Figure 5.9. This is solely due to graphical reasons, as these outliers eclipse all the other data-points, making the scaling of the plot difficult to read.

Additional information regarding the code execution time of the QP solver calls, using the Interior-point method, is presented in Table 5.2.

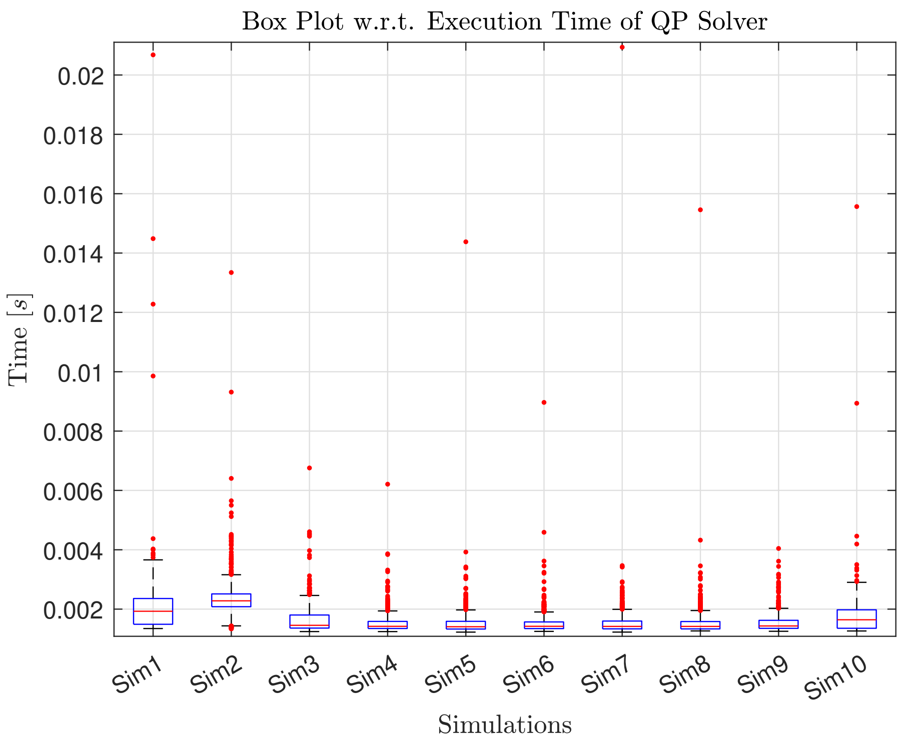


Figure 5.9: Box plot of recorded elapsed time for every 800 QP solver called upon by the linear MPC. Each box represent the data belonging to their respective simulation instance. The Interior-point method is used by the QP solver calls.

	Median	**Max/Min**	**Total**	**Outliers**	**75th/25th Perc.**	**Upper/Lower Adj.**
`Sim1`	$1.92\,ms$	$^{175.16}/_{1.34}\,ms$	$1.80\,s$	11	$^{2.35}/_{1.49}\,ms$	$^{3.65}/_{1.34}\,ms$
`Sim2`	$2.27\,ms$	$^{13.34}/_{1.33}\,ms$	$1.88\,s$	87	$^{2.51}/_{2.08}\,ms$	$^{3.15}/_{1.43}\,ms$
`Sim3`	$1.45\,ms$	$^{38.81}/_{1.24}\,ms$	$1.35\,s$	37	$^{1.80}/_{1.36}\,ms$	$^{2.46}/_{1.24}\,ms$
`Sim4`	$1.42\,ms$	$^{6.21}/_{1.24}\,ms$	$1.22\,s$	76	$^{1.58}/_{1.34}\,ms$	$^{1.93}/_{1.24}\,ms$
`Sim5`	$1.40\,ms$	$^{14.37}/_{1.22}\,ms$	$1.22\,s$	58	$^{1.58}/_{1.32}\,ms$	$^{1.97}/_{1.22}\,ms$
`Sim6`	$1.42\,ms$	$^{8.96}/_{1.24}\,ms$	$1.22\,s$	80	$^{1.56}/_{1.34}\,ms$	$^{1.90}/_{1.24}\,ms$
`Sim7`	$1.41\,ms$	$^{20.93}/_{1.22}\,ms$	$1.24\,s$	67	$^{1.59}/_{1.33}\,ms$	$^{1.99}/_{1.22}\,ms$
`Sim8`	$1.41\,ms$	$^{15.45}/_{1.26}\,ms$	$1.23\,s$	70	$^{1.58}/_{1.33}\,ms$	$^{1.95}/_{1.26}\,ms$
`Sim9`	$1.43\,ms$	$^{4.04}/_{1.25}\,ms$	$1.23\,s$	52	$^{1.62}/_{1.35}\,ms$	$^{2.02}/_{1.25}\,ms$
`Sim10`	$1.64\,ms$	$^{15.56}/_{1.26}\,ms$	$1.39\,s$	10	$^{1.97}/_{1.35}\,ms$	$^{1.97}/_{1.26}\,ms$

Table 5.2: Statistical properties of the data-set from Figure 5.9. Here, **Perc.** and **Adj.** are abbreviations for **Percentile** and **Adjacent**, respectively.

The representative mean value of the total quadratic cost, when using the Interior-point method, is:

$$J_{total,\mu} = \underbrace{0.829}_{J_{y,total,\mu}} + \underbrace{0.013}_{J_{\delta u,total,\mu}} + \underbrace{0.000}_{J_{u,total,\mu}} = 0.842 \tag{5.10}$$

which is the exact same mean quadratic cost as the simulations obtained with the Active-set method (see (5.9)). The closed-loop response of `Sim10` is shown in Figure 5.10. Additionally, Figure 5.11 illustrates the relationship between the execution time

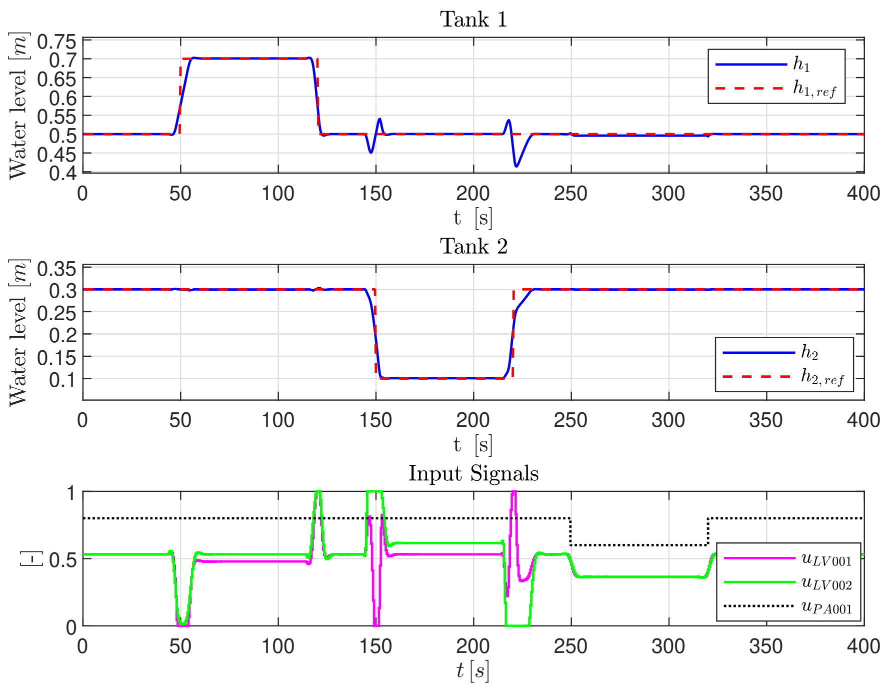


Figure 5.10: Closed-loop response w/ linear MPC, which is programmed to use the Interior-point method.

of the QP solver calls (using the Interior-point method), and the necessary iterations needed in order to converge to an optimal solution. Given the presented data, consider the subsequent observations:

(i) Evidently from Figure 5.9 and Table 5.2, the Interior-point method is a slower optimization algorithm than the Active-set method. This is most obvious when comparing the **Total** column in Table 5.1 and Table 5.2, where one full simulation with the Active-set algorithm spends $\approx 0.80\,s$, while the Interior-point algorithm spends $\approx 1.30\,s$.

(ii) The performance, in terms of the total quadratic cost, is identical between the two methods. This can be seen by comparing $J_{total,\mu}$ in (5.9) with $J_{total,\mu}$ in (5.10). Additionally, the closed-loop response of the two methods are indistinguishable when comparing Figure 5.4 with Figure 5.10.

(iii) Lastly, note that the Interior-point method requires more iterations to reach the optimal solution, than the Active-set method. This can be see by comparing Figure 5.6 with Figure 5.11. As the Active-set method required (mostly) 1 iteration, the Interior-point method requires (mostly) 8 iterations, for the portions of the simulation where the system is static.

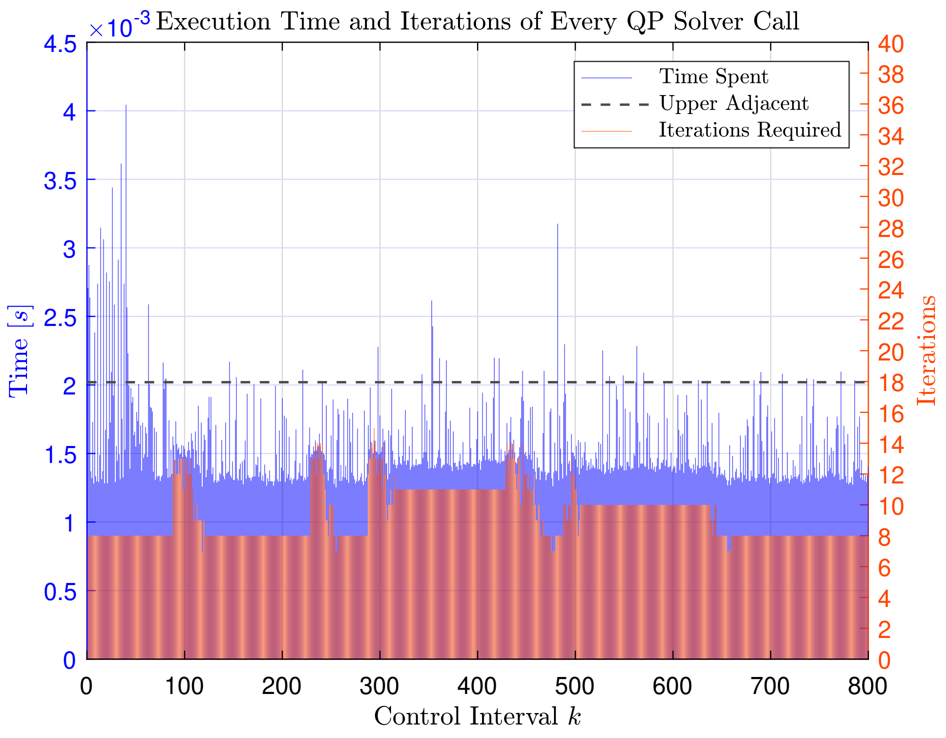


Figure 5.11: Stem plot illustrating the execution time, and the number of necessary iterations, used by the 800 QP solver calls, respectively, from the `Sim9` data-set. The QP solver calls are programmed to use the Interior-point method.

Note that, although the Interior-point method is slower than the Active-set method (based on the presented data in this section), it still provides a feasible control policy, as the MPC is able to compute the QP solver calls within every control interval of length $t_s = 0.5\,s$. However, do to the observations made in this section, the Active-set method will be used (opposed to the Interior-point method) for the subsequent parts of this project.

The simulation results between the MPC-based controllers will be compared against each other at the end of this chapter.

5.4 Explicit MPC

Prior to simulating the explicit MPC, appropriate horizons P and M need to be selected. As briefly discussed in Section 4.2.6, and in Section 5.2, a large control horizon M can greatly increase the complexity of the QP optimization problem, yet, the reduction in the total quadratic cost may be minor (see Figure 5.3).

This is especially the case for explicit MPC, as, once parameterized, it can no longer take advantage of signal previewing. Note how the piecewise affine control law of the explicit

MPC (see 4.40) only relies on the *current* independent variables (with the addition of the previous control moves). Meaning that, even if future set-points are known, the explicit MPC cannot take advantage of this information. For this reason, the need for a long control horizon M becomes superfluous.

To illustrate this, multiple explicit MPCs are design, where the control horizon M varies. The other MPC specifications are selected identical to the linear MPC in Section 5.3. One simulation will be conducted for each explicit MPC, where the total quadratic cost will be recorded, in addition to its complexity. The results of these experiments are shown in Figure 5.12, and in Table 5.3.

Remark 5.3. In this project, the *complexity* of an explicit MPC is determined by: (i) the total number of polyhedral regions, and (ii) the total number of bytes it occupies in the workspace.

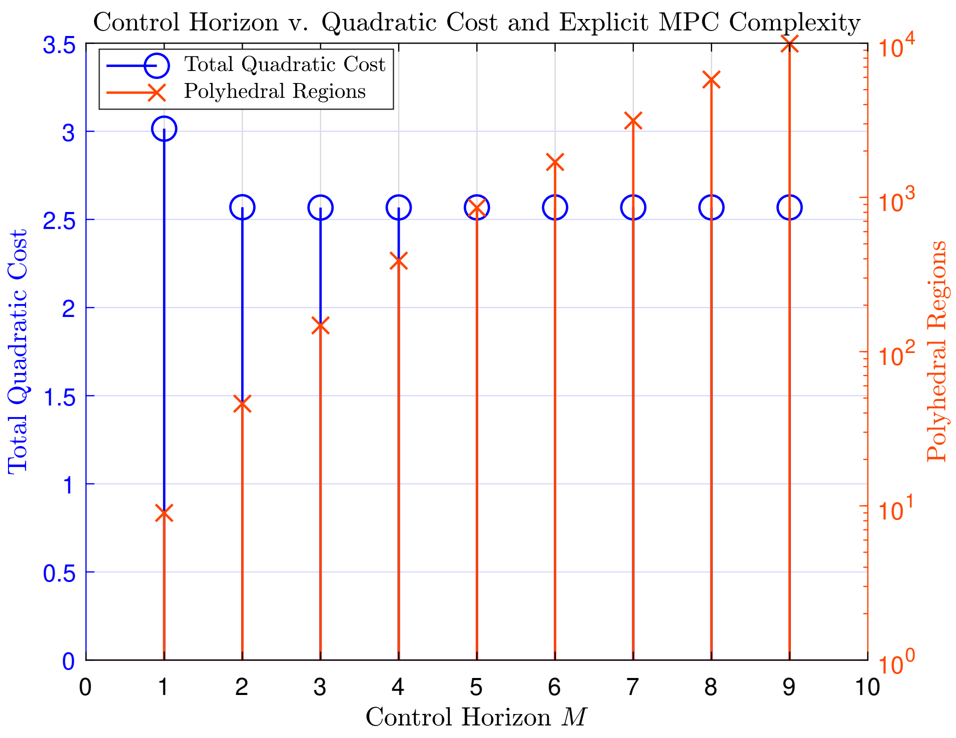


Figure 5.12: Stem plot illustrating the total quadratic cost, and the number of polyhedral regions, for the explicit MPCs designed with different control horizons M. Note the logarithmic scale for the number of *Polyhedral Regions* axis (orange).

Clearly, further increasing the control horizon past $M = 2$ does not result in a noteworthy improvement on the closed-loop response of the system, however, the required data storage on the hardware, in which the explicit MPC is stored, does increase significantly, as M becomes larger. For this reason, the subsequent experiments on the explicit MPC will use a control horizon of length $M = 2$.

Control Horizon	**Total Quadratic Cost**	**Polyhedral Regions**	**Occupied Bytes**
1	3.0150	9	57946
2	2.5694	46	136622
3	2.5673	148	354930
4	2.5681	389	859574
5	2.5684	858	1832458
6	2.5683	1698	3555550
7	2.5684	3152	6481218
8	2.5684	5817	11782454
9	2.5684	9941	19909786

Table 5.3: Supplemental data to Figure 5.12.

With all the MPC specifications selected, similar experiments as those shown in Section 5.3 can be conducted on the explicit MPC. I.e., ten separate instances of Algorithm 5.1 are simulated and recorded. Figure 5.13 shows the box plots of code execution times for these ten simulations, while Table 5.4 contains additional information related to the box plots.

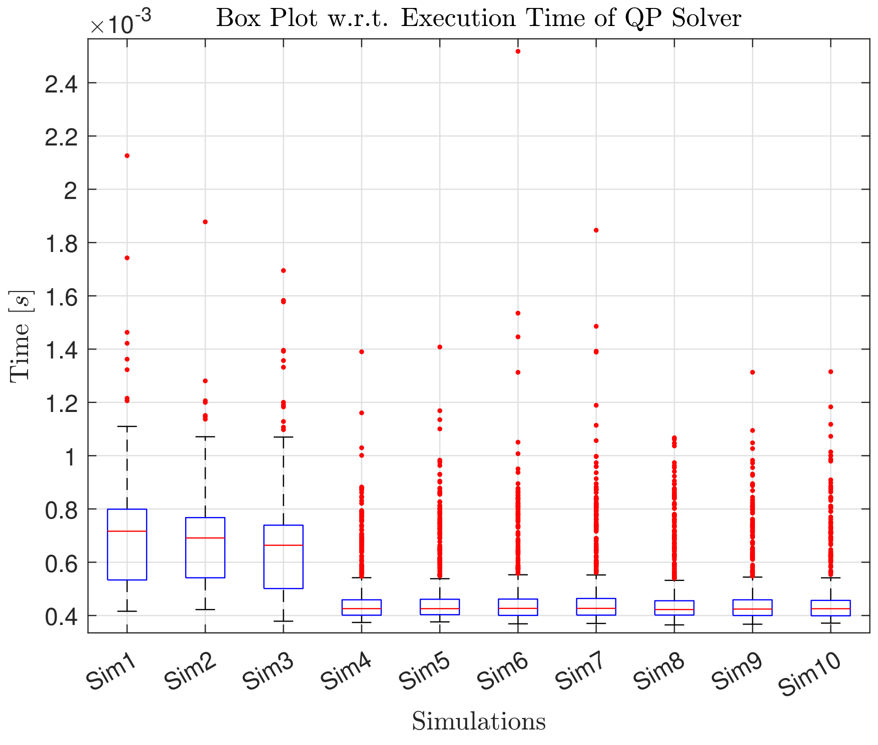


Figure 5.13: Box plot of recorded elapsed time for every 800 QP solver called upon by the explicit MPC. Each box represent the data belonging to their respective simulation instance.

Remark 5.4. One outlier (from `Sim1` with a value of 82.22 ms) is **not** included in the data-set presented in Figure 5.13. This is solely due to graphical reasons, as this outlier eclipses all the other data-points, making the scaling of the plot difficult to read.

	Median	**Max/Min**	**Total**	**Outliers**	**75th/25th Perc.**	**Upper/Lower Adj.**
`Sim1`	$0.71\,ms$	$^{82.22}/_{0.41}\,ms$	$0.62\,s$	9	$^{0.79}/_{0.53}\,ms$	$^{1.10}/_{0.41}\,ms$
`Sim2`	$0.69\,ms$	$^{1.87}/_{0.42}\,ms$	$0.53\,s$	7	$^{0.76}/_{0.54}\,ms$	$^{0.76}/_{0.42}\,ms$
`Sim3`	$0.66\,ms$	$^{1.69}/_{0.37}\,ms$	$0.50\,s$	13	$^{0.45}/_{0.50}\,ms$	$^{1.06}/_{0.37}\,ms$
`Sim4`	$0.42\,ms$	$^{1.39}/_{0.37}\,ms$	$0.36\,s$	94	$^{0.46}/_{0.40}\,ms$	$^{0.54}/_{0.37}\,ms$
`Sim5`	$0.42\,ms$	$^{1.40}/_{0.37}\,ms$	$0.37\,s$	98	$^{0.46}/_{0.40}\,ms$	$^{0.53}/_{0.37}\,ms$
`Sim6`	$0.42\,ms$	$^{2.51}/_{0.36}\,ms$	$0.37\,s$	98	$^{0.46}/_{0.40}\,ms$	$^{0.55}/_{0.36}\,ms$
`Sim7`	$0.42\,ms$	$^{1.84}/_{0.37}\,ms$	$0.37\,s$	99	$^{0.46}/_{0.40}\,ms$	$^{0.55}/_{0.37}\,ms$
`Sim8`	$0.42\,ms$	$^{1.06}/_{0.36}\,ms$	$0.36\,s$	92	$^{0.45}/_{0.40}\,ms$	$^{0.53}/_{0.36}\,ms$
`Sim9`	$0.42\,ms$	$^{1.31}/_{0.36}\,ms$	$0.36\,s$	92	$^{0.45}/_{0.40}\,ms$	$^{0.54}/_{0.36}\,ms$
`Sim10`	$0.42\,ms$	$^{1.31}/_{0.37}\,ms$	$0.36\,s$	85	$^{0.45}/_{0.39}\,ms$	$^{0.54}/_{0.37}\,ms$

Table 5.4: Statistical properties of the data-set from Figure 5.13. Here, **Perc.** and **Adj.** are abbreviations for **Percentile** and **Adjacent**, respectively.

The representative mean value for the total quadratic cost of the explicit MPC is calculated to be:

$$J_{total,\mu} = \underbrace{2.5437}_{J_{y,total,\mu}} + \underbrace{0.0257}_{J_{\delta u,total,\mu}} + \underbrace{0.000}_{J_{u,total,\mu}} = 2.5694 \tag{5.11}$$

The quadratic cost of the 10 respective simulations are identical to each other, and to the mean values given in (5.11).

Figure 5.14 shows the closed-loop response of the explicit MPC, where the data from `Sim10` is used. Note how the closed-loop response in Figure 5.14 does not react prior to the square pulse in the set-points and the disturbance. Note also how, without look-ahead control, the spikes occurring in h_1 (at times $t \approx 150\,s$ and $t \approx 225\,s$) are more aggressive. Yet, the explicit MPC does provide a feasible control policy with respect to the code execution time (see Table 5.4), and it does provide satisfactory control of the two-tank system, with respect to set-point tracking.

5.4.1 Alternative Closed-Loop Response

Figure 5.15 and Figure 5.16 show the closed-loop response of the explicit MPC where the priority of the control is Tank 1 and Tank 2, respectively (similarly to the experiments conducted in Section 5.3.1). Figure 5.16 illustrates a great example where the manipulated variables are being used excessively. This excessive use of the manipulated variables (as mentioned in Section 2.2) can cause unwanted disturbances or even long-term damage to the actuators. To account for this, the manipulated variable rate weighting matrix S is adjusted accordingly:

$$S = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \tag{5.12}$$

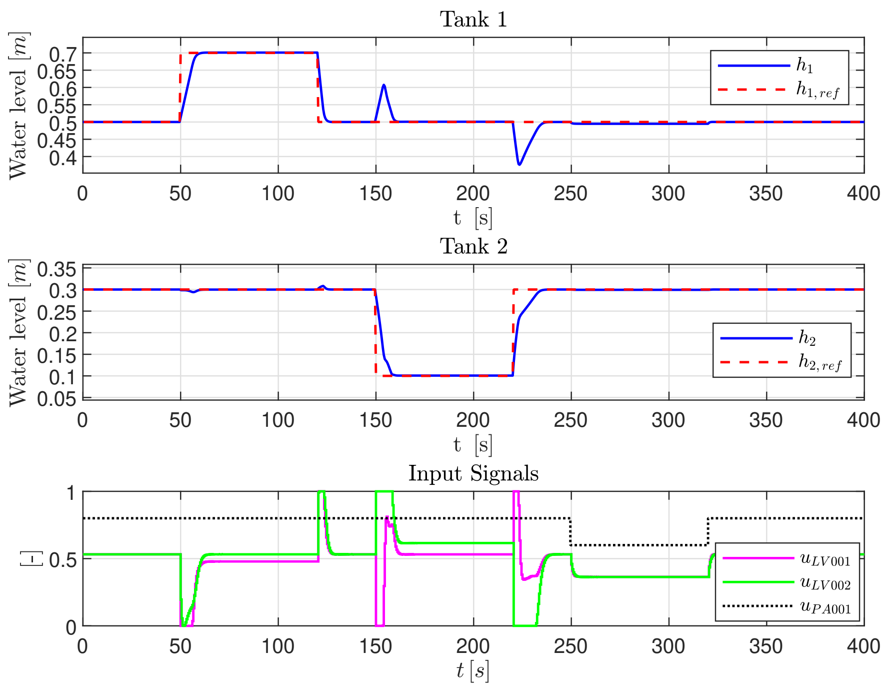


Figure 5.14: Closed-loop response w/ explicit MPC.

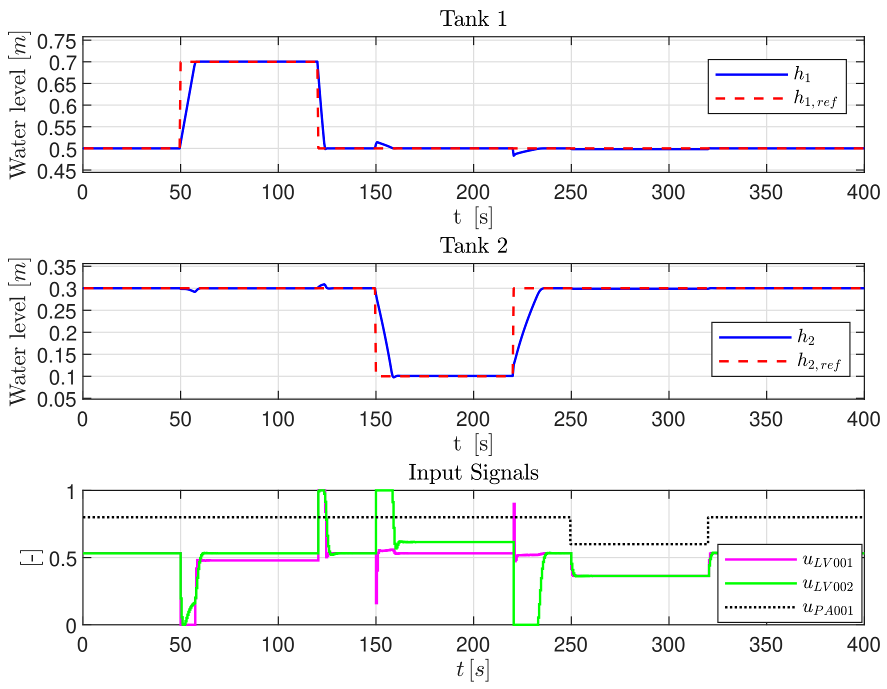


Figure 5.15: Closed-loop response w/ explicit MPC. The paramount objective is to prioritize set-point tracking in Tank 1. I.e., the output weighting matrix is set to be Q^i (see (4.22)).

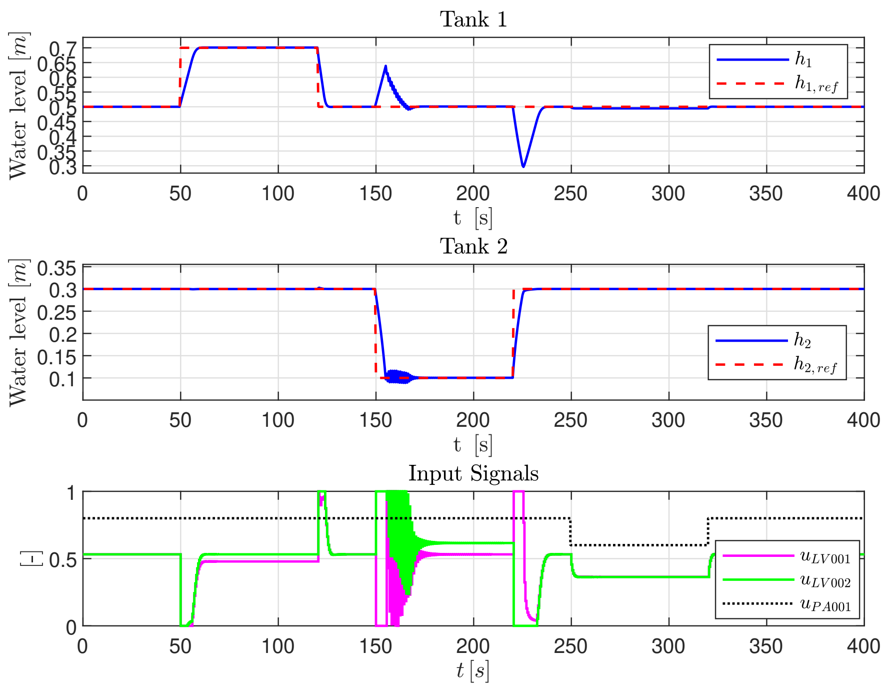


Figure 5.16: Closed-loop response w/ explicit MPC. The paramount objective is to prioritize set-point tracking in Tank 2. I.e., the output weighting matrix is set to be Q^{ii} (see (4.22)).

which results in the closed-loop response shown in Figure 5.17. As previously mentioned, the simulation results between the MPC-based controllers will be compared against each other at the end of this chapter.

5.5 Adaptive MPC

For the simulations of the adaptive MPC, the same MPC specifications to those chosen in Section 5.3 are selected. Similarly to the previous experiments, ten separate simulations are conducted, where the total quadratic cost, and code execution time, are recorded.

In the case of the adaptive MPC, the code execution time of one control interval consists of the elapsed time for one QP solver call to be completed, *plus* the elapsed time for the prediction model to be updated. The recorded data regarding the code execution time for the ten simulations (with the adaptive MPC) are shown in Figure 5.18, and Table 5.5.

Remark 5.5. One outlier (from **`Sim2`** with a value of $42.91\,ms$) is **not** included in the data-set presented in Figure 5.18. This is solely due to graphical reasons, as this outlier eclipses all the other data-points, making the scaling of the plot difficult to read.

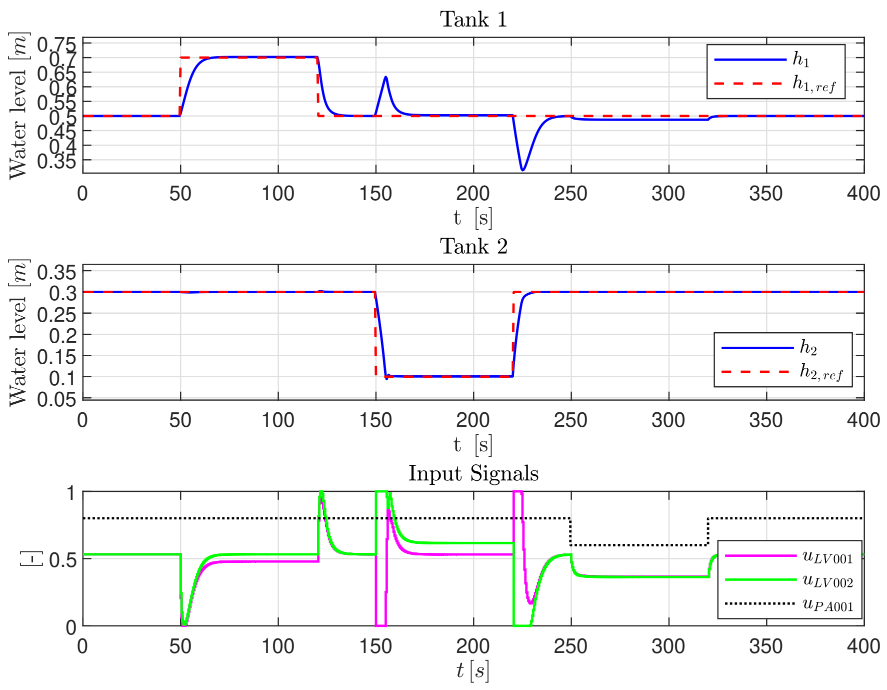


Figure 5.17: Closed-loop response w/ explicit MPC. The paramount objective is to prioritize set-point tracking in Tank 2. I.e., the output weighting matrix is set to be Q^{ii} (see (4.22)). Additionally, the manipulated variable weighting matrix S is adjusted according to (5.12), in order to avoid excessive use of the manipulated variables.

	Median	**Max/Min**	**Total**	**Outliers**	**75th/25th Perc.**	**Upper/Lower Adj.**
`Sim1`	$10.92\,ms$	$32.31/10.35\,ms$	$8.91\,s$	54	$11.24/10.72\,ms$	$11.98/10.35\,ms$
`Sim2`	$11.03\,ms$	$42.91/10.40\,ms$	$9.19\,s$	54	$11.37/10.78\,ms$	$12.20/10.40\,ms$
`Sim3`	$11.07\,ms$	$29.34/10.40\,ms$	$8.98\,s$	36	$11.37/10.80\,ms$	$12.23/10.40\,ms$
`Sim4`	$11.11\,ms$	$28.71/10.36\,ms$	$9.05\,s$	44	$11.48/10.81\,ms$	$12.47/10.36\,ms$
`Sim5`	$11.19\,ms$	$27.71/10.42\,ms$	$9.18\,s$	57	$11.57/10.86\,ms$	$12.61/10.42\,ms$
`Sim6`	$11.15\,ms$	$29.06/10.39\,ms$	$9.04\,s$	33	$11.53/10.80\,ms$	$12.63/10.39\,ms$
`Sim7`	$11.12\,ms$	$28.08/10.24\,ms$	$9.02\,s$	37	$11.46/10.82\,ms$	$12.33/10.24\,ms$
`Sim8`	$11.24\,ms$	$24.77/10.38\,ms$	$9.12\,s$	32	$11.63/10.86\,ms$	$12.76/10.38\,ms$
`Sim9`	$11.16\,ms$	$28.26/10.34\,ms$	$9.08\,s$	36	$11.56/10.81\,ms$	$12.67/10.34\,ms$
`Sim10`	$11.19\,ms$	$28.33/10.36\,ms$	$9.20\,s$	59	$11.60/10.84\,ms$	$12.73/10.36\,ms$

Table 5.5: Statistical properties of the data-set from Figure 5.18. Here, **Perc.** and **Adj.** are abbreviations for **Percentile** and **Adjacent**, respectively.

On average, the total code execution time of one simulation with the adaptive MPC is calculated to be (see column: **Total** in Table 5.5):

$$\begin{aligned}\text{Mean Total Time} &= \frac{8.91+9.19+8.98+9.06+9.18+9.05+9.03+9.13+9.08+9.20}{10}\\ &= 9.09\,s\end{aligned} \tag{5.13}$$

As previously mentioned, the mean total time (5.13) is the sum of two parts: (i) the

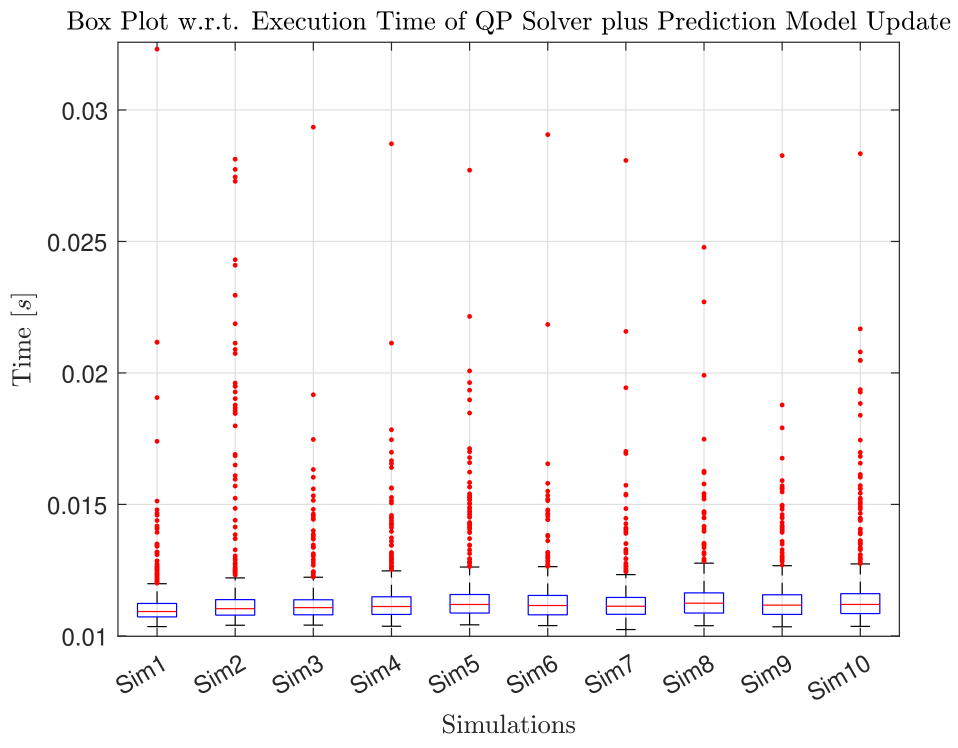


Figure 5.18: Box plot of recorded elapsed time for every 800 control interval (where each control interval consists of one QP solver call, plus one prediction model update) with the adaptive MPC. Each box represent the data belonging to their respective simulation instance.

mean total time of one simulation spent on executing QP solver calls, and (ii) the mean total time of one simulation spent on updating the prediction model. Table 5.6 shows these recorded values for each of the ten simulations. Figure 5.19 illustrates, on

	Code Execution Time of QP Solver Calls	**Code Execution Time of Prediction Model Updates**	**Code Execution Time of One Entire Simulation**
`Sim1`	2.22 s	6.69 s	8.91 s
`Sim2`	2.28 s	6.91 s	9.19 s
`Sim3`	2.22 s	6.76 s	8.98 s
`Sim4`	2.25 s	6.80 s	9.05 s
`Sim5`	2.27 s	6.91 s	9.18 s
`Sim6`	2.23 s	6.81 s	9.04 s
`Sim7`	2.23 s	6.79 s	9.02 s
`Sim8`	2.24 s	6.88 s	9.12 s
`Sim9`	2.24 s	6.84 s	9.08 s
`Sim10`	2.28 s	6.92 s	9.20 s

Table 5.6: Table presenting how long each simulation with the adaptive MPC took, in addition to the time spent on solving QP solver calls and updating the prediction model, respectively.

average, how much time of the adaptive MPC protocol is spent solving the QP solver calls, and how much time is spent on updating the prediction model, respectively (the data from Table 5.6 is used to calculated the averages). Clearly, the majority of the

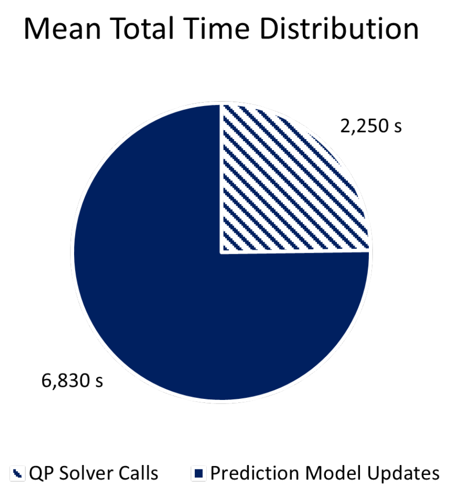


Figure 5.19: Pie chart illustrating the distribution of the mean total code execution time from the ten simulations shown in Figure 5.18.

computational efforts are spent on updating the prediction model, which the adaptive MPC will be using for the next control interval. However, even with this additional task of updating the prediction model, the adaptive MPC does provide a feasible control policy, considering that the protocol of finding an optimal control move, and updating the prediction model, is completed within the length of the control interval (which is $t_s = 0.5\,s$ in this experiment).

The representative mean value for the total quadratic cost of the adaptive MPC is calculated to be:

$$J_{total,\mu} = \underbrace{0.9018}_{J_{y,total,\mu}} + \underbrace{0.0281}_{J_{\delta u,total,\mu}} + \underbrace{0.000}_{J_{u,total,\mu}} = 0.9300 \tag{5.14}$$

The quadratic cost of the ten respective simulations are identical to each other, and to the mean values given in (5.14).

Figure 5.20 shows the closed-loop response of the adaptive MPC, where the data from **`Sim10`** is used.

5.5.1 Alternative Closed-Loop Response

Figure 5.21 and Figure 5.22 show the closed-loop response of the adaptive MPC where the priority of the control is Tank 1 and Tank 2, respectively (similarly to the experiments conducted in Section 5.3.1). Note that the manipulated variable rate weighting matrix S

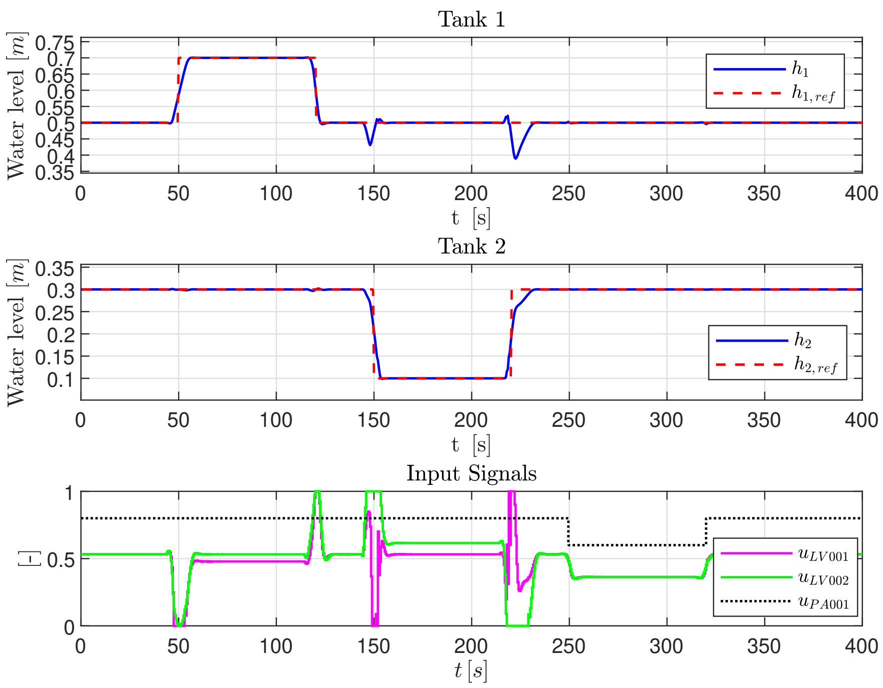


Figure 5.20: Closed-loop response w/ adaptive MPC.

is also adjusted according to (5.12), for both cases, in order to avoid excessive use of the valves.

Note that the presented results for the different MPC-based controllers will be compared against each other at the end of this chapter.

5.6 Nonlinear MPC

Lastly, ten separate simulations of Algorithm 5.1 are conducted using the nonlinear MPC. For these experiments, the code execution time of one control interval is equal to the code execution time of the SQP solver call, during said interval. The MPC specifications are selected similarly to those selected in Section 5.3.

The recorded data regarding the code execution time for the ten simulations (with the nonlinear MPC) are shown in Figure 5.23, and Table 5.7.

Remark 5.6. Note that Figure 5.23 contains all of the data points from the conducted simulations, hence, the most extreme outliers contribute to poor scaling of the box plots, making them difficult to read. For this reason, Figure 5.24 shows a zoomed in portion of the box plots for easier readability, but, be aware that this figure does not include all of the data points.

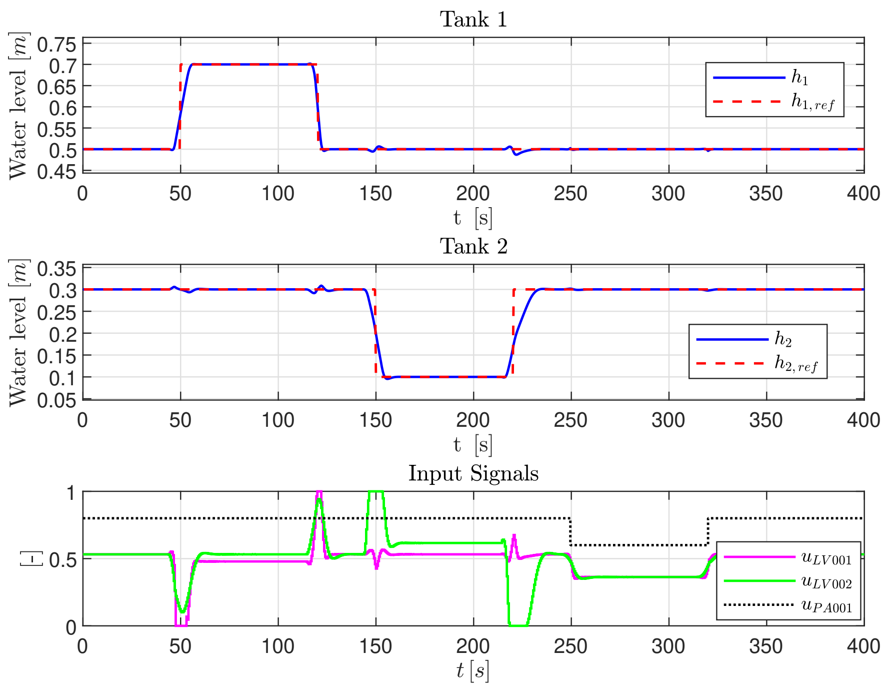


Figure 5.21: Closed-loop response w/ adaptive MPC. The paramount objective is to prioritize set-point tracking in Tank 1. I.e., the output weighting matrix is set to be Q^i (see (4.22)).

	Median	**Max/Min**	**Total**	**Outliers**	**75th/25th Perc.**	**Upper/Lower Adj.**
Sim1	$9.18\,ms$	$385.25/5.57\,ms$	$25.40\,s$	176	$14.76/8.57\,ms$	$20.77/5.57\,ms$
Sim2	$8.97\,ms$	$318.63/5.59\,ms$	$25.27\,s$	173	$14.47/8.35\,ms$	$21.83/5.59\,ms$
Sim3	$8.98\,ms$	$375.51/5.41\,ms$	$24.44\,s$	176	$14.71/8.28\,ms$	$19.95/5.41\,ms$
Sim4	$8.97\,ms$	$217.92/5.39\,ms$	$23.41\,s$	176	$14.81/8.20\,ms$	$23.90/5.39\,ms$
Sim5	$8.90\,ms$	$210.21/5.34\,ms$	$23.18\,s$	174	$14.54/8.23\,ms$	$21.26/5.34\,ms$
Sim6	$9.06\,ms$	$226.74/5.33\,ms$	$23.56\,s$	173	$15.55/8.36\,ms$	$22.44/5.33\,ms$
Sim7	$9.10\,ms$	$215.85/5.36\,ms$	$23.45\,s$	173	$14.59/8.32\,ms$	$21.13/5.36\,ms$
Sim8	$9.11\,ms$	$222.59/5.36\,ms$	$23.53\,s$	173	$14.75/8.32\,ms$	$18.54/5.36\,ms$
Sim9	$9.04\,ms$	$212.95/5.39\,ms$	$23.30\,s$	173	$14.85/8.24\,ms$	$20.28/5.39\,ms$
Sim10	$9.14\,ms$	$238.81/5.38\,ms$	$23.71\,s$	173	$14.68/8.29\,ms$	$19.42/5.58\,ms$

Table 5.7: Statistical properties of the data-set from Figure 5.23. Here, **Perc.** and **Adj.** are abbreviations for **Percentile** and **Adjacent**, respectively.

The representative mean value for the total quadratic cost of the nonlinear MPC is calculated to be:

$$J_{total,\mu} = \underbrace{0.7786}_{J_{y,total,\mu}} + \underbrace{0.0152}_{J_{\delta u,total,\mu}} + \underbrace{0.000}_{J_{u,total,\mu}} = 0.7938 \qquad (5.15)$$

The quadratic cost of the ten respective simulations are identical to each other, and to the mean values given in (5.15).

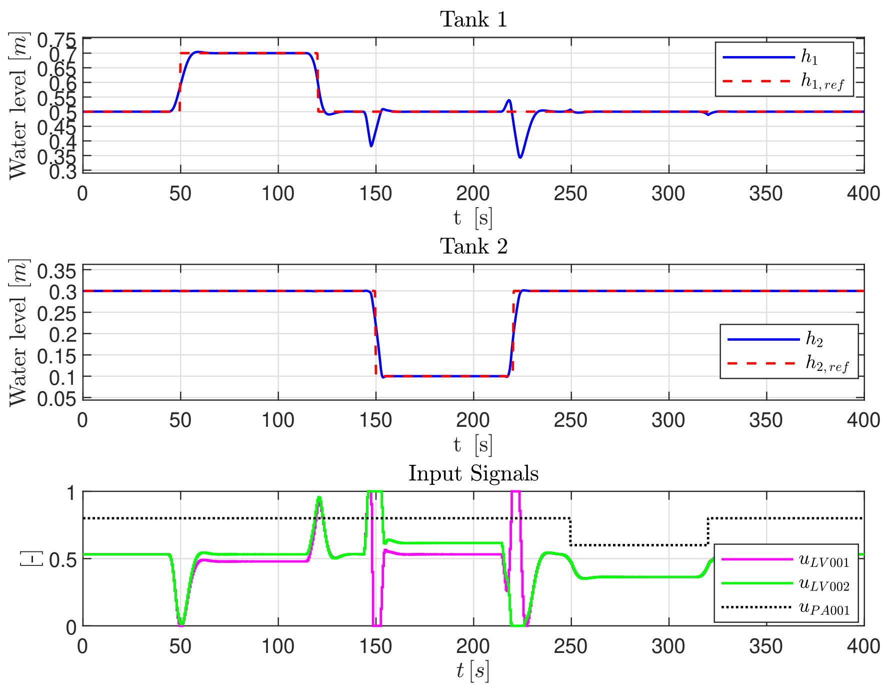


Figure 5.22: Closed-loop response w/ adaptive MPC. The paramount objective is to prioritize set-point tracking in Tank 2. I.e., the output weighting matrix is set to be Q^{ii} (see (4.22)).

Figure 5.25 shows the closed-loop response of the nonlinear MPC, where the data from `Sim10` is used. The relationship between the code executing time, and the number of necessary iterations needed to find an optimal solution, for the SQP solver calls during simulation `Sim1` is shown in Figure 5.26. The red dashed line in Figure 5.26 represents the length of every control interval, which, during this experiment, is set to be $t_s = 0.5\,s$. Evidently from Figure 5.26, and column: Max/Min in Table 5.7, the nonlinear MPC does provide a feasible control policy, as the optimal manipulated variable is calculated within the time frame of every respective control interval. Additionally, the nonlinear MPC does provide a satisfactory control of the two-tank system, as shown in Figure 5.25.

5.6.1 Alternative Closed-Loop Response

Figure 5.27 and Figure 5.28 show the closed-loop response of the nonlinear MPC where the priority of the control is Tank 1 and Tank 2, respectively (similarly to the experiments conducted in Section 5.3.1).

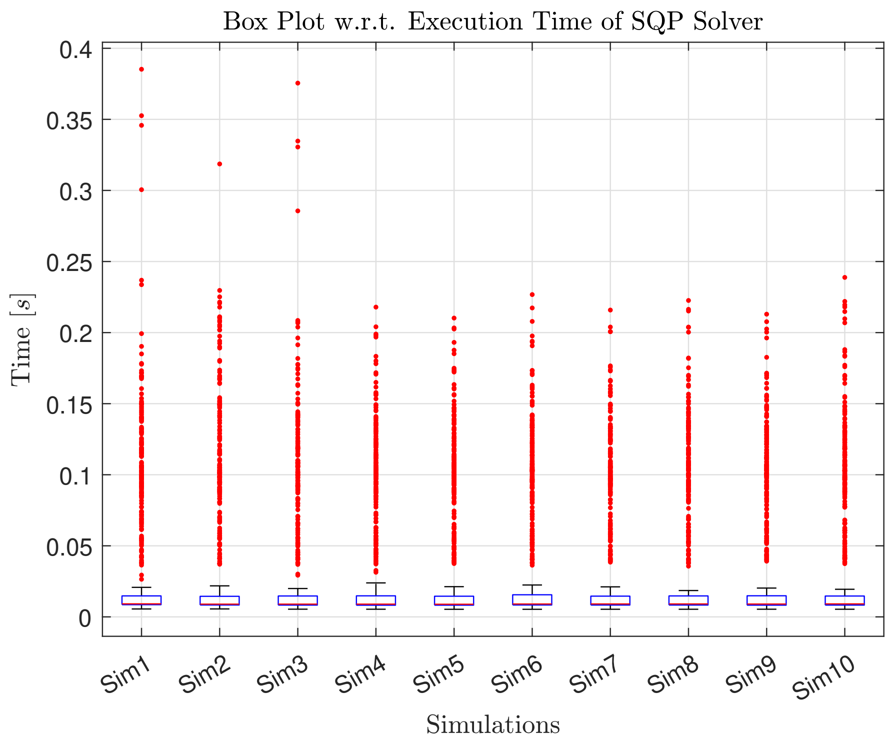


Figure 5.23: Box plot of recorded elapsed time for every 800 SQP solver called upon by the nonlinear MPC. Each box represent the data belonging to their respective simulation instance.

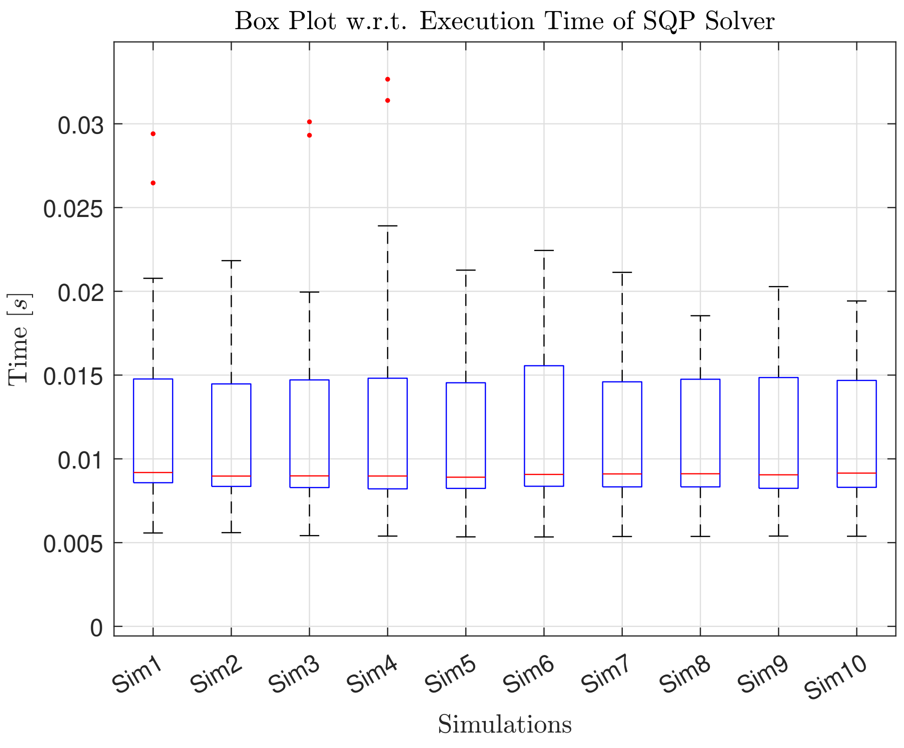


Figure 5.24: Zoomed in version of Figure 5.23, for easier readability of the box plots.

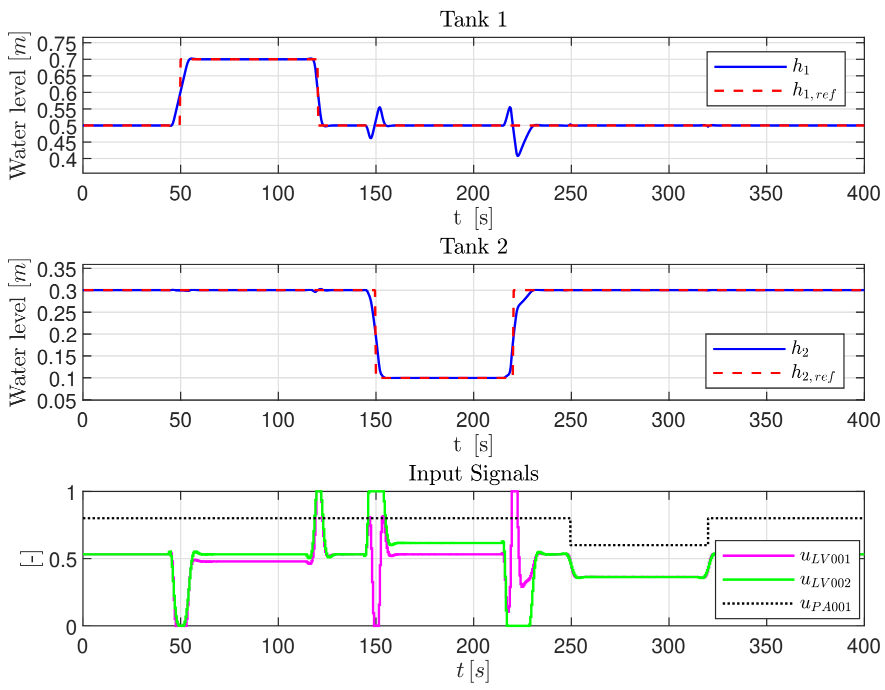


Figure 5.25: Closed-loop response w/ nonlinear MPC.

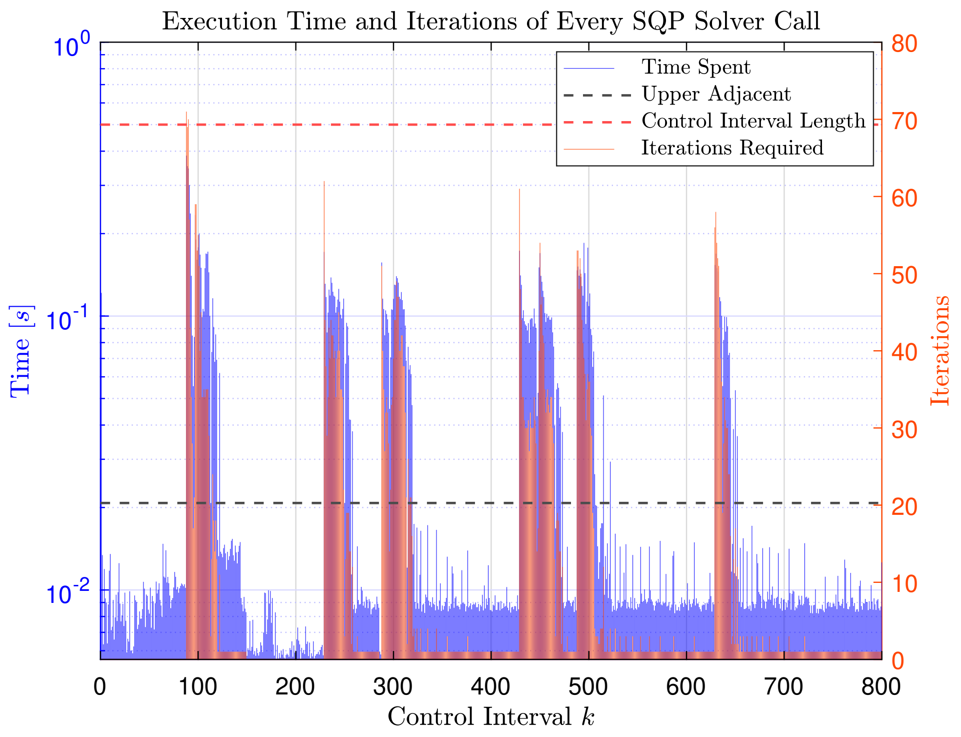


Figure 5.26: Stem plot illustrating the execution time, and the number of necessary iterations, used by the 800 SQP solver calls, respectively, from the `Sim1` data-set. Note the logarithmic scale for the *Time* axis (blue).

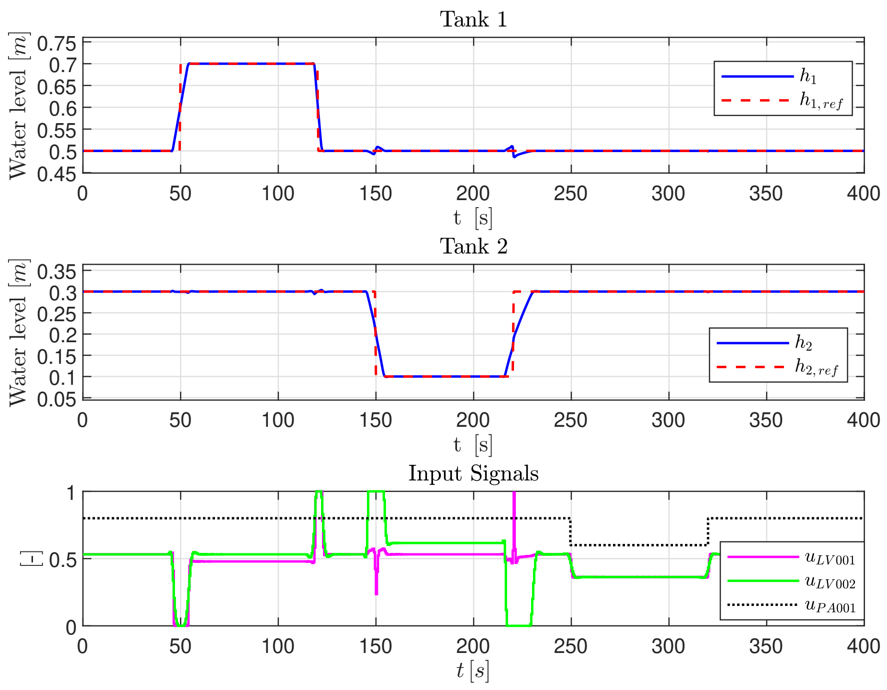


Figure 5.27: Closed-loop response w/ nonlinear MPC. The paramount objective is to prioritize set-point tracking in Tank 1. I.e., the output weighting matrix is set to be Q^i (see (4.22)).

5.6.2 Sub-Optimal Solution

So far, all the different MPC-based controllers have provided feasible control policies during the experiments presented in this chapter, including the nonlinear MPC. However, it may happen that, for some systems, the search for an optimal control move may take longer than the control interval itself. I.e., the MPC may not be able to find a suitable control move $u_{opt}(k)$ before the next control interval $k+1$.

There are several ways of tackling the issue of an infeasible control policy:

(i) If possible, reduce the complexity of the SQP/QP optimization problem. This can be done by making the prediction horizon P, and/or the control horizon M, smaller. Alternatively, one can take advantage of input blocking (as discussed in Section 4.2.6).

(ii) If possible, increase the control interval, such that the SQP/QP optimization problem can be solved within the time-limit. This equates to increasing the sampling period t_s, which heavily relies on how fast the dynamics of the system under consideration are.

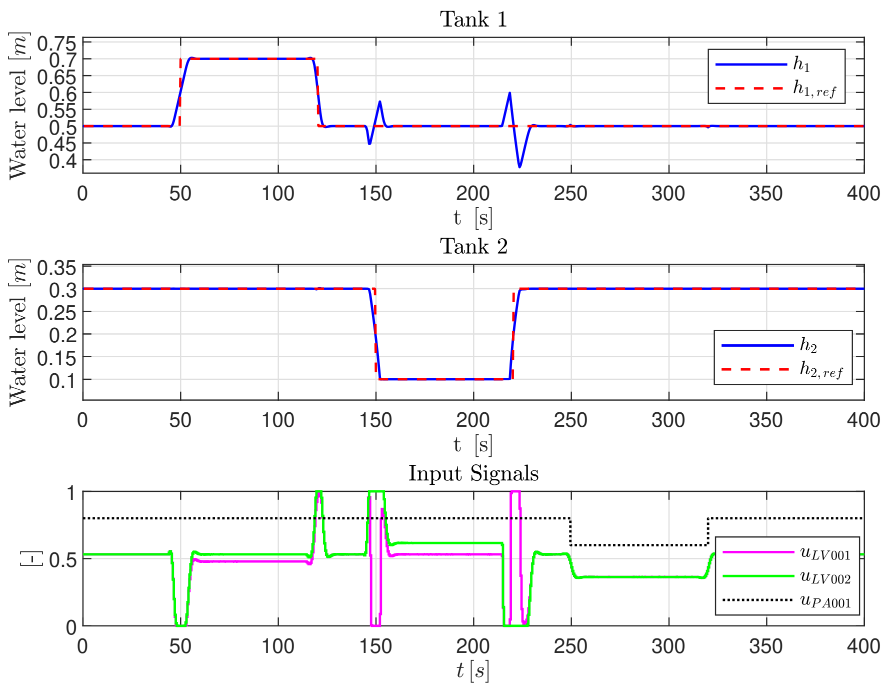


Figure 5.28: Closed-loop response w/ nonlinear MPC. The paramount objective is to prioritize set-point tracking in Tank 2. I.e., the output weighting matrix is set to be Q^{ii} (see (4.22)).

(iii) If possible, use sub-optimal solutions of the SQP/QP optimization problems. I.e., set a maximum limit on the number of iterations that the SQP/QP solver can use during one optimization problem.

The latter option will be briefly explored in this section.

For the nonlinear MPC, it was shown in Figure 5.26 that, at times, the number of required iterations to solve the SQP optimization problem spiked up to approximately 70. While these SQP solver calls were solved within the time-limit of the control interval, supposed that an even faster controller is desired. To achieve this, the maximum number of iterations allowed to the SQP solver is reduced to 30. In MATLAB, this can be done with the following syntax:

```
nlobj.Optimization.SolverOptions.MaxIterations = 30;     % Limit SQP
    Solver to 30 Iterations
nlobj.Optimization.UseSuboptimalSolution = true;         % Toggle Sub-
    Optimal Solution
```

Performing the same simulation of Algorithm 5.1 with this modificed nonlinear MPC results in the closed-loop response shown in Figure 5.29: which yielded the total quadratic

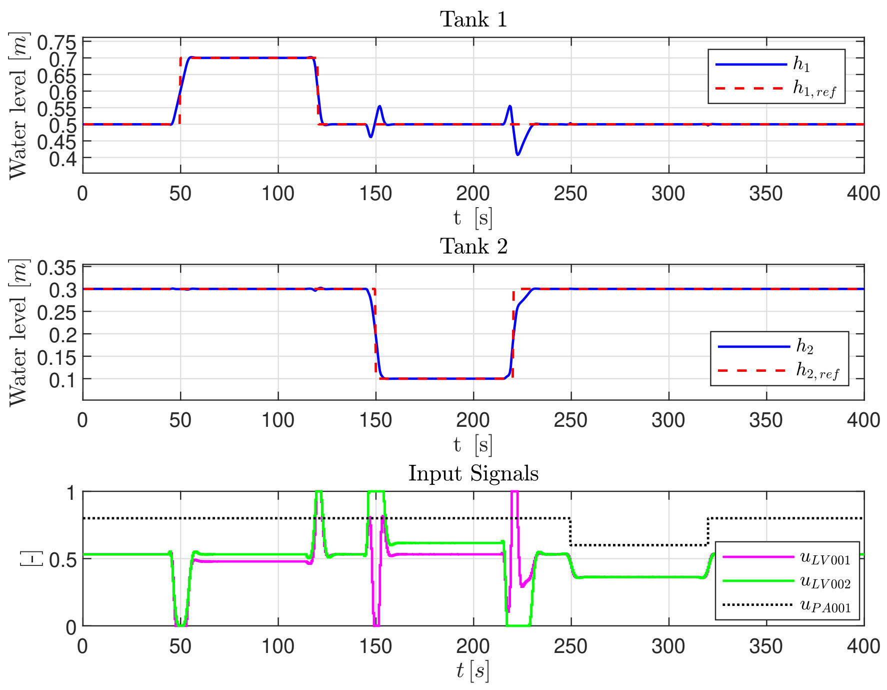


Figure 5.29: Closed-loop response w/ nonlinear MPC, using a sub-optimal solution.

cost:

$$J_{total,\mu} = \underbrace{0.7787}_{J_{y,total,\mu}} + \underbrace{0.0153}_{J_{\delta u,total,\mu}} + \underbrace{0.000}_{J_{u,total,\mu}} = 0.7940 \tag{5.16}$$

By comparing Figure 5.25 with Figure 5.29, and the total quadratic cost (5.15) with (5.16), it is clear that using the sub-optimal solution had no noteworthy impact on the closed-loop response.

The total time spent on the simulation with the sub-optimal solution, however, was $20.92\,s$, compared to the average time of $23.92\,s$, when using the optimal solution (see Table 5.4). Additionally, Figure 5.30 shows the execution time, and the number of necessary iterations, used by the SQP solver calls, when programmed to use a sub-optimal solution. Conclusively, the use of sub-optimal solutions are a viable option when a faster MPC-based controller is desired. In the experiment presented in this section, using the sub-optimal solution resulted in a noteworthy reduction in the computational speed, however, the degradation of the closed-loop response was negligible.

5.7 Analysis of the Simulation Results

Given all the experiments conducted thus far, the following observations are made:

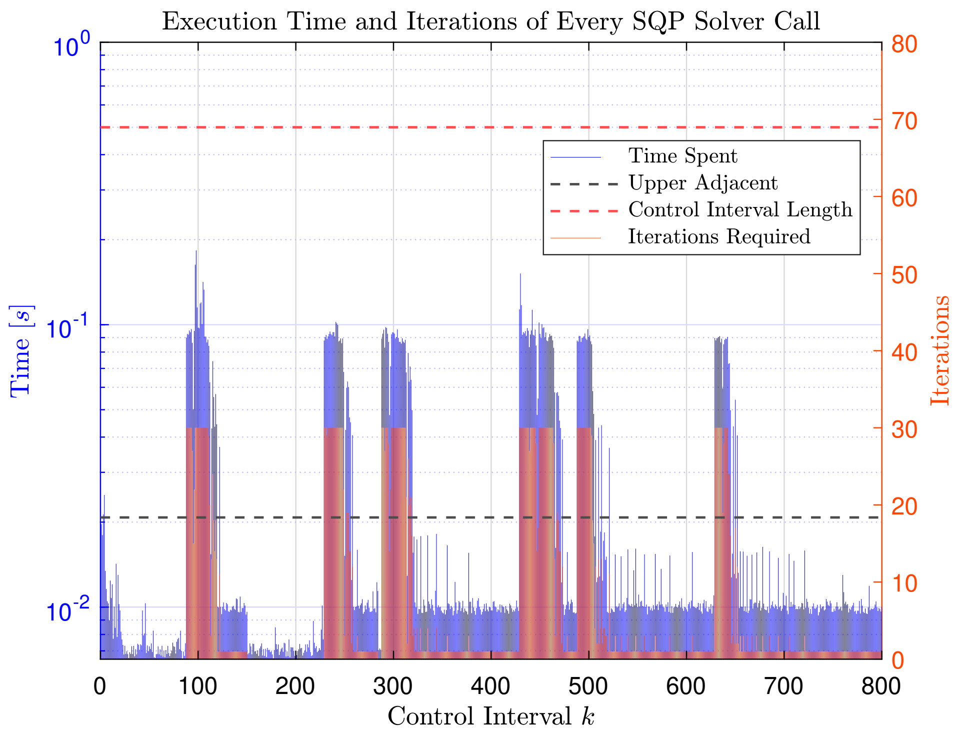


Figure 5.30: Stem plot illustrating the execution time, and the number of necessary iterations, used by the 800 SQP solver calls, respectively. The nonlinear MPC is modified to use sub-optimal solutions, and the SQP solver calls are limited to 30 iterations. Note the logarithmic scale for the *Time* axis (blue).

(i) All of the MPC-based controllers tested in this chapter provided a feasible control policy. I.e., all of the SQP/QP optimization problems were solved within the time-limit of one control interval, which was of length $t_s = 0.5\,s$.

(ii) All of the MPC-based controllers tested in this chapter provided satisfactory control of the two-tank system, even when subjected to square pulses in the set-points and the measured disturbance, respectively.

(iii) All of the MPC-based controllers tested in this chapter satisfied the constraints on the manipulated variables. Meaning that the manipulated variables were contained within the constraint set $\mathcal{U}$ at all times.

(iv) When comparing the **Total** column of Table 5.1, 5.4, 5.5, and 5.7, it is clear that:

- the explicit MPC has the fastest computational execution time of all the MPC-based controllers, with an average time of $0.42\,s$ to complete one simulation.
- the nonlinear MPC has the slowest computational execution time of all the MPC-based controllers, with an average time of $23.92\,s$ to complete one simulation.

(v) When comparing the total quadratic costs (5.9), (5.11), (5.14), and (5.15), it is clear that:

- The explicit MPC has the inferior closed-loop response, with a total quadratic cost of $J_{total,\mu} = 2.5694$.
- The nonlinear MPC has the superior closed-loop response, with a total quadratic cost of $J_{total,\mu} = 0.7938$.

(vi) The weighting matrices Q, R, and S, do provide flexibility to the operating engineer regarding the priority of the closed-loop system, as shown in Section 5.3.1, 5.4.1, 5.5.1, and 5.6.1.

(vii) The use of sub-optimal solutions from the SQP/QP solvers are viable options if a faster MPC-based controller is required, without significant loss of quality in the closed-loop response, as shown in Section 5.6.2.

(viii) The Active-set method, for solving the QP optimization problems, proved to be a faster option that the Interior-point method, which can be seen by comparing the **Total** column in Table 5.1 and 5.2. Additionally, the total quadratic cost between the two methods proved to be indistinguishable, which can be seen by comparing the two costs in (5.9) and (5.10), respectively.

Chapter 6

Experimental Evaluation

Up until now, the different MPC-based controllers have only been implemented and evaluated in a simulation environment. The subsequent chapter, however, will look to implement and evaluate the MPC-based controllers in a practical environment, where they will be performing level control on the real two-tank system. The goal of this chapter is to validate the simulation results in Chapter 5, through experimental tests on the real system.

Additionally, the performance of the MPC-based controllers will be compared, not only against each other, but also against the performance of more traditional control techniques such as LQR- and PID-control. For more information on the design of the LQR- and PID-controllers, see Appendix A and B, respectively.

Lastly, as a collaborative effort with fellow master's student Greta Bekerytė, experiments will be conducted on the two-tank system, where, rather than measuring all of the states directly, some states are estimated by state observers. The goal of these experiments is to analyse the linear MPC's ability to control the system, when a combination of state measurements and state estimates are available.

6.1 Experimental Setup

The real two-tank system can be seen in Figure 6.1, which highlights the different components that are being used. For replication purposes, detailed images of the components, and their specifications, are included in Appendix E.

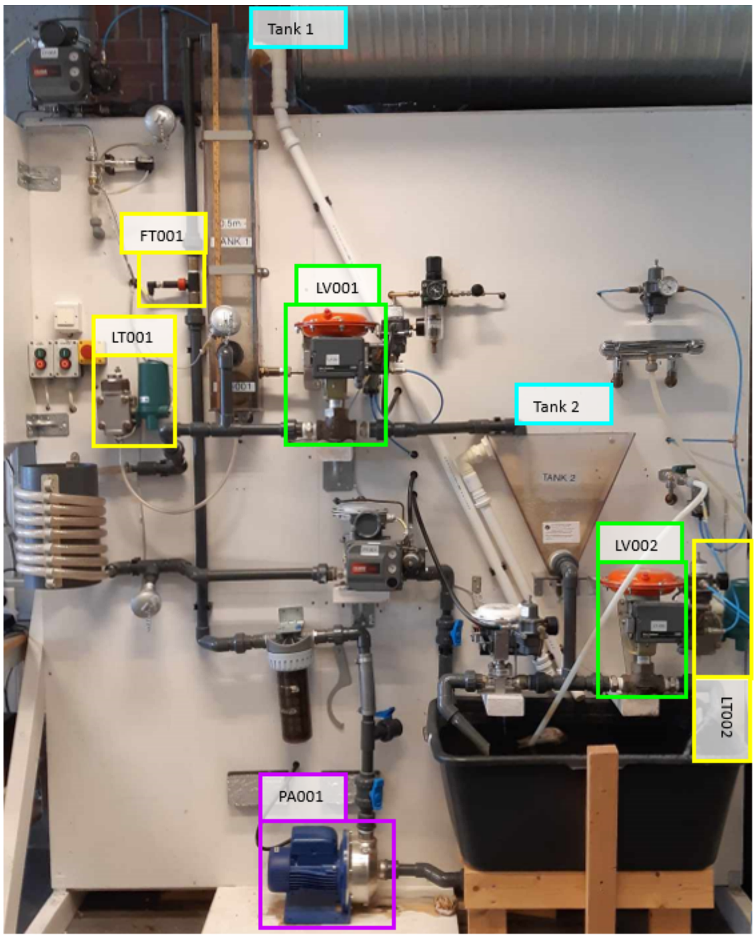


Figure 6.1: Photograph of the two-tank system located at the laboratory facility KE E-458, UiS.

6.2 Experimental Procedure

The experimental procedure is as follows: Like the simulations in Chapter 5, the ex-

Algorithm 6.1 Experimental procedure

```
1: TIME $< 50\,s \implies$ SYS $\leftarrow \mathfrak{NS}$   ▷ Drive system to nominal solution $\mathfrak{NS}$
2: TIME $\geq 50\,s \implies h_{1,ref} \uparrow$   ▷ Start of positive rectangular pulse
3: TIME $\geq 150\,s \implies h_{1,ref} \downarrow$   ▷ End of positive rectangular pulse
4: TIME $\geq 250\,s \implies h_{2,ref} \uparrow$   ▷ Start of positive rectangular pulse
5: TIME $\geq 350\,s \implies h_{2,ref} \downarrow$   ▷ End of positive rectangular pulse
6: TIME $\geq 450\,s \implies u_{PA001} \downarrow$   ▷ Start of negative rectangular pulse
7: TIME $\geq 550\,s \implies u_{PA001} \uparrow$   ▷ End of negative rectangular pulse
8: TIME $= 700\,s \implies$ end   ▷ End of experimental procedure
```

periments conducted on the real two-tank system look to analyse both the set-point tracking and the disturbance rejection abilities of the controllers. Note that there are some differences between the simulations in Chapter 5, and the experimental procedure in Algorithm 6.1:

(i) The total time of the experimental procedure is $700\,s$, rather than $400\,s$.

(ii) Each rectangular pulse, in the set-points and the disturbance, is of length $100\,s$, rather than $50\,s$.

(iii) The rectangular pulse in $h_{2,ref}$ is positive, rather than negative.

(iv) The nominal solution is given by:

$$\underbrace{\tilde{h}_1 = 0.5, \quad \tilde{h}_2 = 0.2, \quad \tilde{u}_{PA001} = 0.8, \quad \tilde{u}_{LV001} = 0.5317, \quad \tilde{u}_{LV002} = 0.5680}_{\mathfrak{NS}} \tag{6.1}$$

rather than (5.1).

6.3 Parameters

Given the speed at which the linear-, explicit-, and adaptive-MPC completed the simulations in Chapter 5 (see Table 5.1, 5.4, and 5.5), the sampling time is chosen to be $t_s = 0.1\,s$ for these controllers. This sampling time is also used for the LQR- and PID-controllers. However, to ensure a feasible control policy during real-time operation of the nonlinear MPC, the sampling time is increased to $t_s = 0.5\,s$ for this controller. Additionally, the nonlinear MPC is programmed to use a sub-optimal solution, where the SQP optimization problem is limited to 30 iterations, as it was shown in Section 5.6.2 that this is a viable option for faster control, without noteworthy loss in performance.

All of the MPCs are programmed to use a prediction horizon of $P = 13$, and a control horizon of $M = 2$. A shorter control horizon is chosen to simplify the SQP/QP optimization problem, and to further promote a feasible control policy during real-time operation of the MPCs. As discussed in Section 5.2, a large control horizon can lead to redundancy, as the complexity of the SQP/QP optimization problem increases, without a noteworthy reduction in the total quadratic cost (see Figure 5.2).

Given the nominal solution (6.1), and a sampling time of $t_s = 0.1\,s$, the discrete time linear prediction model that the linear- and explicit-MPC will be using is given by:

$$\begin{cases} \begin{bmatrix} \Delta h_1(k+1) \\ \Delta h_2(k+1) \end{bmatrix} = \begin{bmatrix} 0.9977 & 0 \\ 0.001273 & 0.9984 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(k) \\ \Delta h_2(k) \end{bmatrix} \\ \qquad + \begin{bmatrix} -0.007181 & 0 & 0.004495 \\ 0.003986 & -0.003802 & 2.866 \cdot 10^{-6} \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(k) \\ \Delta u_{LV002}(k) \\ \Delta u_{PA001}(k) \end{bmatrix} \\ \begin{bmatrix} \Delta \hat{h}_1(k) \\ \Delta \hat{h}_2(k) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(k) \\ \Delta h_2(k) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(k) \\ \Delta u_{LV002}(k) \\ \Delta u_{PA001}(k) \end{bmatrix} \end{cases} \tag{6.2}$$

The weighting matrices Q, R, and S will vary, depending on the priority of the desired closed-loop response, and the MPC-based controller type. These matrices will be clearly defined when experimental results are presented. Like the simulations in Chapter 5, there are three main cases that will be tested for the desired closed-loop responses:

(i) Both tanks are prioritized equally.

(ii) Tank 1 is prioritized more than Tank 2.

(iii) Tank 2 is prioritized more than Tank 1.

6.4 Linear MPC

The first experiment is conducted on the linear MPC, which is programmed to prioritize both tanks equally. For this case, the weighting matrices are given by (6.3), and the experimental results are presented in Figure 6.2. To ensure a more smooth control, and less aggressive use of the control input, the weighting matrices R and S are adjusted accordingly, which also holds true for the subsequent experiments in this chapter.

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}, \quad S = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}, \quad R = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.3}$$

The first $50\,s$ of the experiments are used to drive the system to the nominal solution $\mathfrak{NS}$. Note that the initial conditions of the experimental trials are different. Therefore, to ensure a fair comparison between the controllers, the total quadratic cost is calculated only beyond $t \geq 30\,s$, which is when the system has stabilized on $\mathfrak{NS}$ for all of the experimental trials.

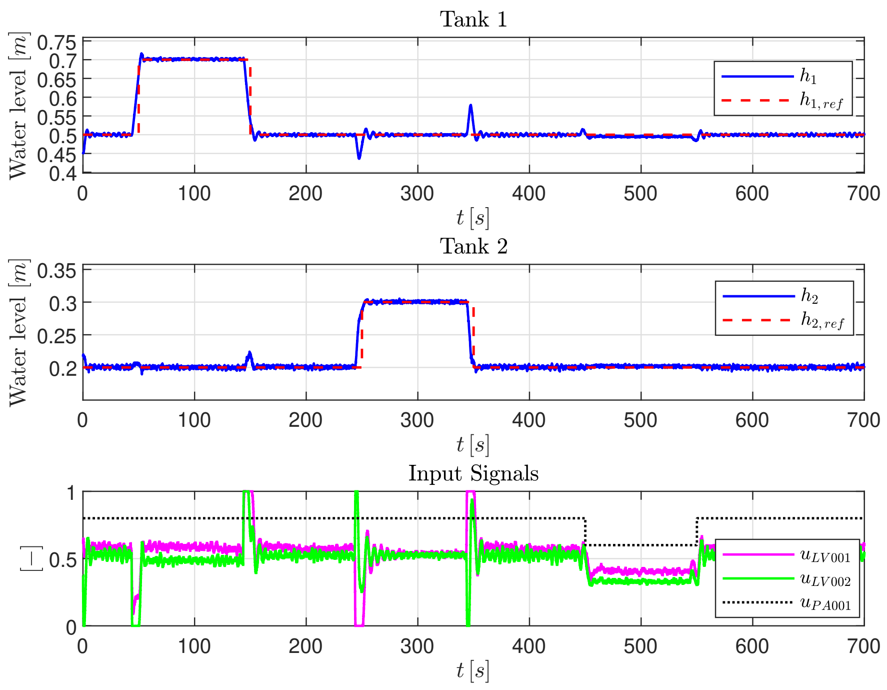


Figure 6.2: Experimental results of the closed-loop response w/ linear MPC. The MPC is programmed to prioritize level control of both tanks equally.

When prioritizing Tank 1 more than Tank 2 in the desired closed-loop response, the following weighting matrices are used:

$$Q = \begin{bmatrix} 500 & 0 \\ 0 & 100 \end{bmatrix}, \quad S = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}, \quad R = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.4}$$

which yield the experimental results shown in Figure 6.3. Lastly, the weighting matrices shown in (6.5) are used when prioritizing Tank 2 more than Tank 1 in the closed-loop response. Figure 6.4 shows the results from this experimental trial.

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 500 \end{bmatrix}, \quad S = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}, \quad R = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.5}$$

The total quadratic costs of the three experiments with the linear MPC are listen in Table 6.1.

Experimental Trial	J_{total}
Figure 6.2	59132
Figure 6.3	421020
Figure 6.4	1196700

Table 6.1: Calculated quadratic costs of the experimental trials with the linear MPC.

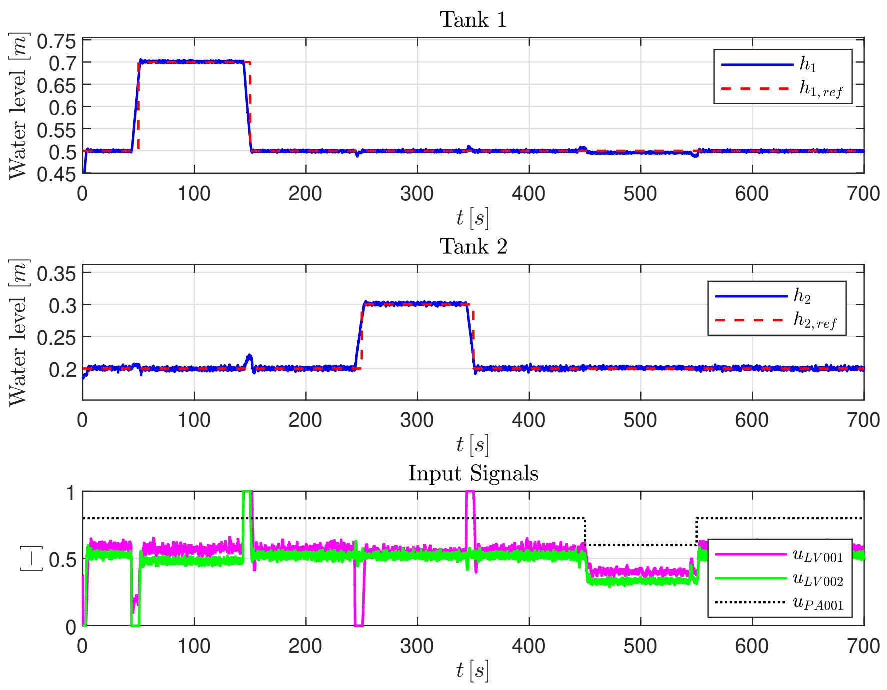


Figure 6.3: Experimental results of the closed-loop response w/ linear MPC. The MPC is programmed to prioritize Tank 1 more than Tank 2.

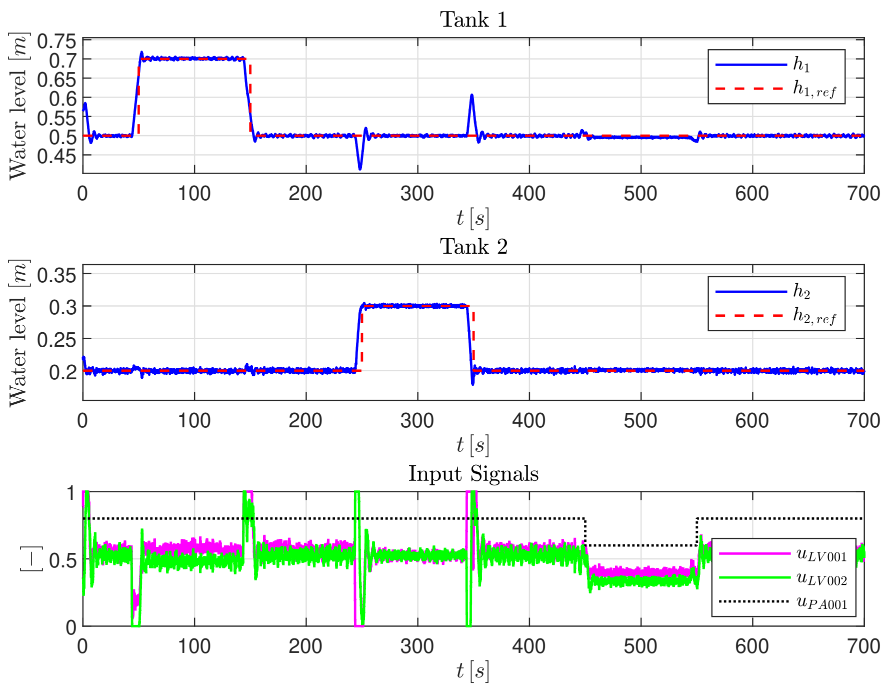


Figure 6.4: Experimental results of the closed-loop response w/ linear MPC. The MPC is programmed to prioritize Tank 2 more than Tank 1.

Remark 6.1. Note that, as the three experimental trials presented thus far all have different objectives, the quadratic costs in Table 6.1 are *not* comparable against each other. Table 6.1 is simply a compact way of presenting the calculated costs. This will be the case for the subsequent experiments in this chapter. In practise, the total quadratic costs of the different controllers are only comparable if their objectives of the closed-loop response are identical. I.e., if the weighting matrices Q, R, and S are the same.

6.5 Explicit MPC

The same weighting matrices used for the linear MPC, are also used for the explicit MPC. I.e., when prioritizing both tanks equally, the explicit MPC is programmed to use the weighting matrices in (6.3). Figure 6.5 shows the results of this experimental trial. Similarly, the explicit MPC uses the weighting matrices in (6.4), when programmed to

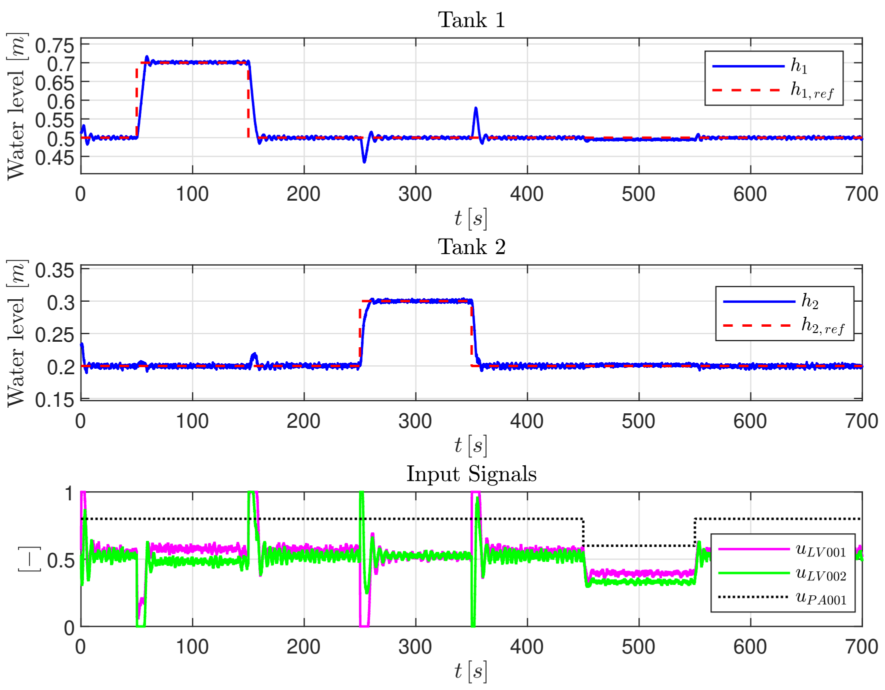


Figure 6.5: Experimental results of the closed-loop response w/ explicit MPC. The MPC is programmed to prioritize level control of both tanks equally.

prioritize Tank 1 more than Tank 2. The results for this experimental trial are shown in Figure 6.6. Finally, when prioritizing Tank 2 more than Tank 1, the explicit MPC uses the weighting matrices in (6.5), and the experimental results for this trial are presented in Figure 6.7. The total quadratic costs of the three experiments with the explicit MPC are listen in Table 6.2.

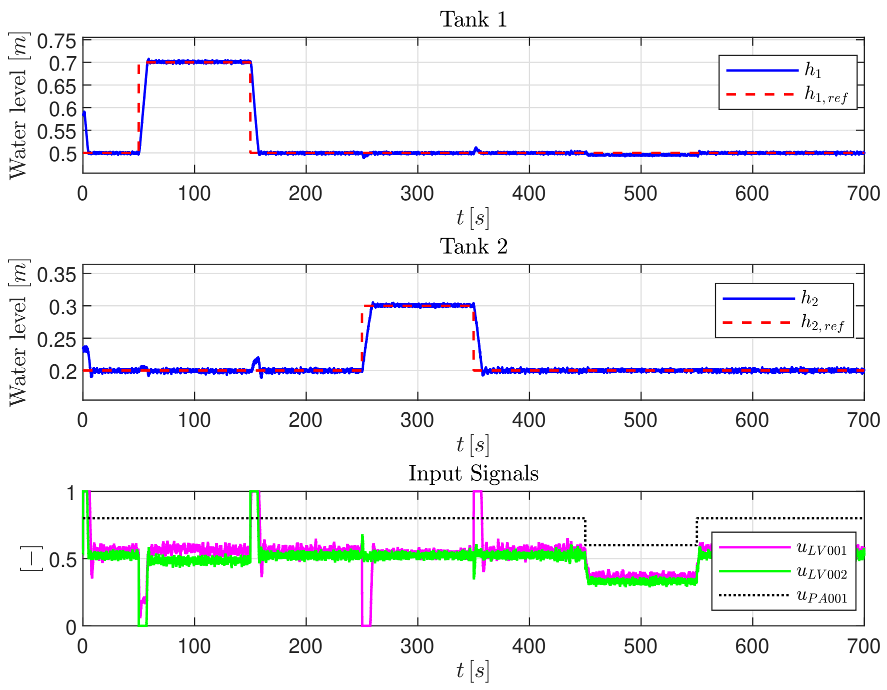


Figure 6.6: Experimental results of the closed-loop response w/ explicit MPC. The MPC is programmed to prioritize Tank 1 more than Tank 2.

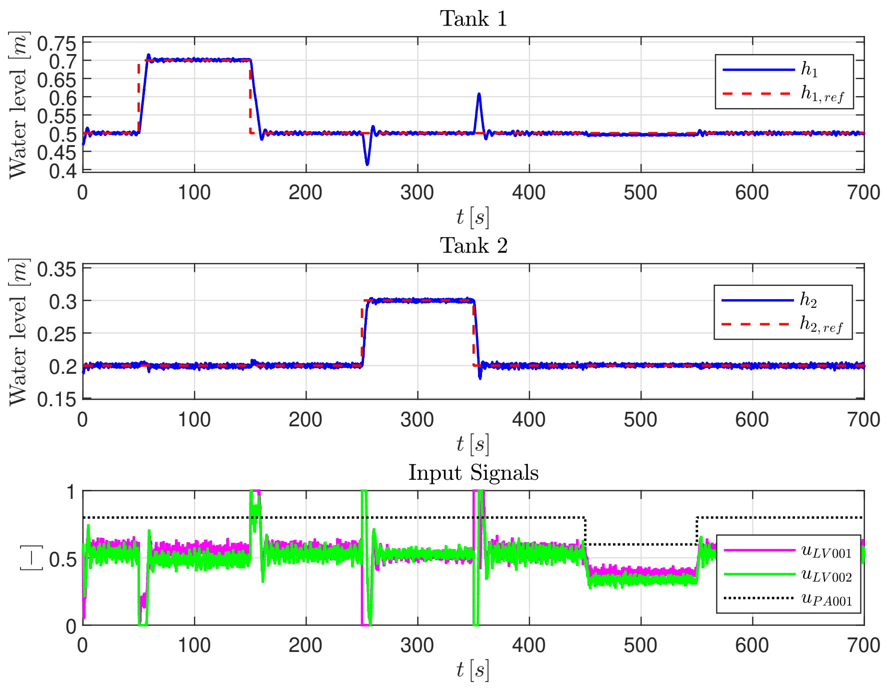


Figure 6.7: Experimental results of the closed-loop response w/ explicit MPC. The MPC is programmed to prioritize Tank 2 more than Tank 1.

Experimental Trial	J_{total}
Figure 6.5	70626
Figure 6.6	823990
Figure 6.7	784880

Table 6.2: Calculated quadratic costs of the experimental trials with the explicit MPC.

6.6 Adaptive MPC

For the adaptive MPC, the weighting matrices S and R are increased slightly more, due to excessive use of the control inputs during operation. For the case where both tanks are weighted equally, the adaptive MPC uses the following weighting matrices:

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}, \quad S = \begin{bmatrix} 60 & 0 \\ 0 & 60 \end{bmatrix}, \quad R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \tag{6.6}$$

which yield the experimental results shown in Figure 6.8. As the adaptive MPC is

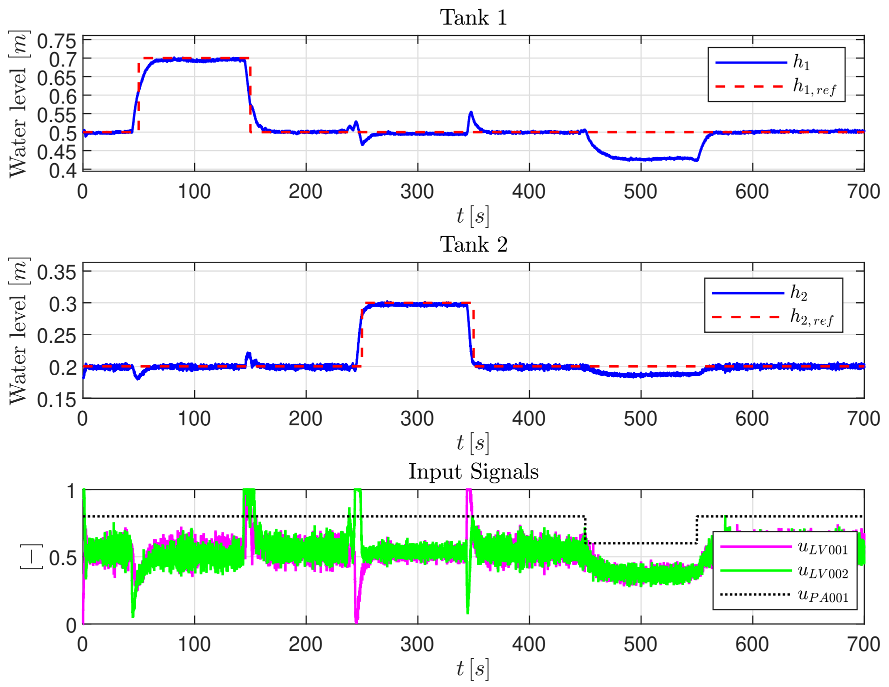


Figure 6.8: Experimental results of the closed-loop response w/ adaptive MPC. The MPC is programmed to prioritize level control of both tanks equally.

programmed to prioritize Tank 1 more than Tank 2 in the closed-loop response, the weighting matrices are changed accordingly:

$$Q = \begin{bmatrix} 200 & 0 \\ 0 & 100 \end{bmatrix}, \quad S = \begin{bmatrix} 60 & 0 \\ 0 & 60 \end{bmatrix}, \quad R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \tag{6.7}$$

The results of this experimental trial can be seen in Figure 6.9. Lastly, for the adaptive

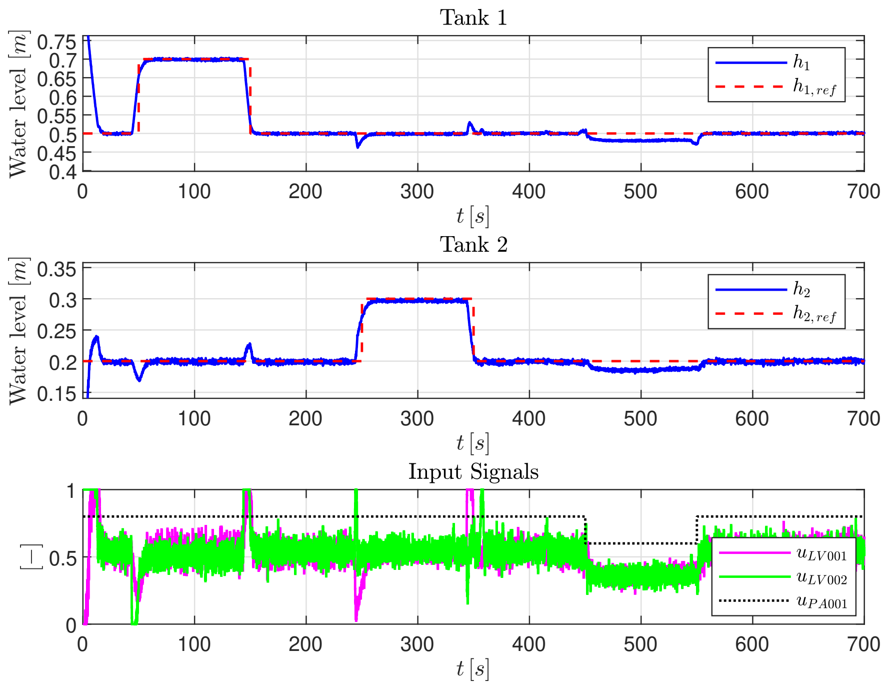


Figure 6.9: Experimental results of the closed-loop response w/ adaptive MPC. The MPC is programmed to prioritize Tank 1 more than Tank 2.

MPC to prioritize Tank 2 more than Tank 1 in the closed-loop response, the weighting matrices are given by:

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 200 \end{bmatrix}, \quad S = \begin{bmatrix} 60 & 0 \\ 0 & 60 \end{bmatrix}, \quad R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \tag{6.8}$$

and the respective experimental results for this case can be seen in Figure 6.10. The total quadratic costs of the three experiments with the adaptive MPC are listen in Table 6.3.

Experimental Trial	J_{total}
Figure 6.8	266580
Figure 6.9	314550
Figure 6.10	512930

Table 6.3: Calculated quadratic costs of the experimental trials with the adaptive MPC.

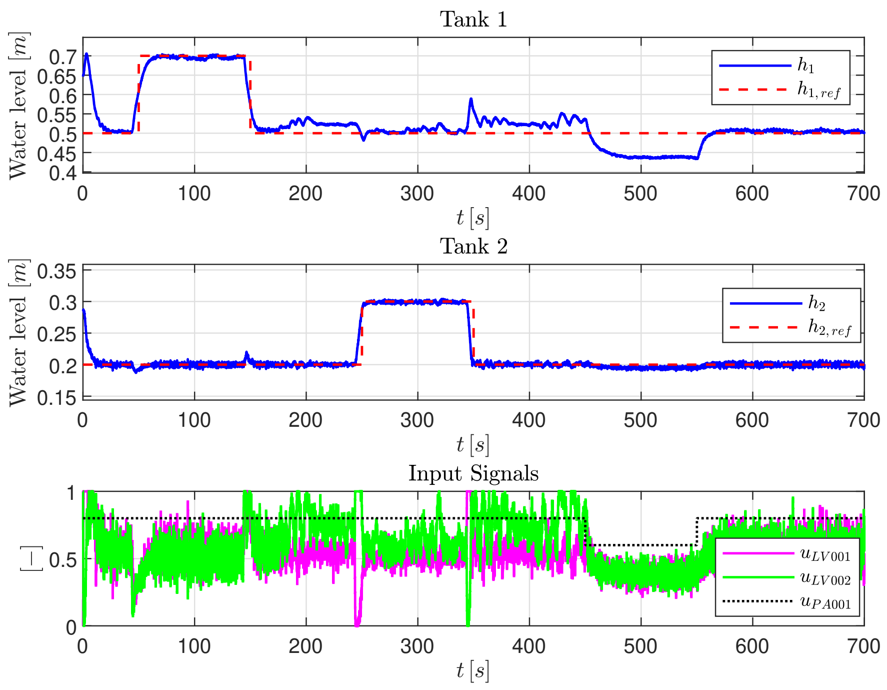


Figure 6.10: Experimental results of the closed-loop response w/ adaptive MPC. The MPC is programmed to prioritize Tank 2 more than Tank 1.

6.7 Nonlinear MPC

The last MPC-based controller to be tested is the nonlinear MPC. The first experiment is where both tanks are weighted equally. To this end, the following weighting matrices are used:

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}, \quad S = \begin{bmatrix} 40 & 0 \\ 0 & 40 \end{bmatrix}, \quad R = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.9}$$

which resulted in the closed-loop response shown in Figure 6.11. Next, the desired closed-loop response is for the nonlinear MPC to prioritize Tank 1 more than Tank 2, which can be achieved by choosing the following weighting matrices:

$$Q = \begin{bmatrix} 200 & 0 \\ 0 & 100 \end{bmatrix}, \quad S = \begin{bmatrix} 40 & 0 \\ 0 & 40 \end{bmatrix}, \quad R = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.10}$$

Figure 6.12 shows the recorded data for this experimental trial. The last experiment conducted, for the MPC-based controllers, is where the nonlinear MPC prioritizes Tank 2 more than Tank 1. The following weighting matrices are used for this case of the desired

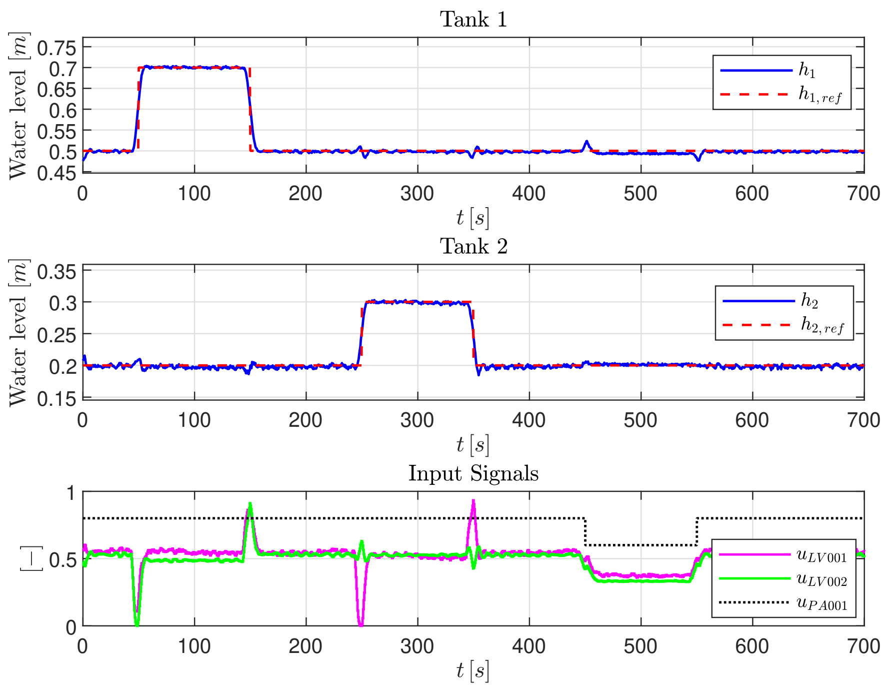


Figure 6.11: Experimental results of the closed-loop response w/ nonlinear MPC. The MPC is programmed to prioritize level control of both tanks equally.

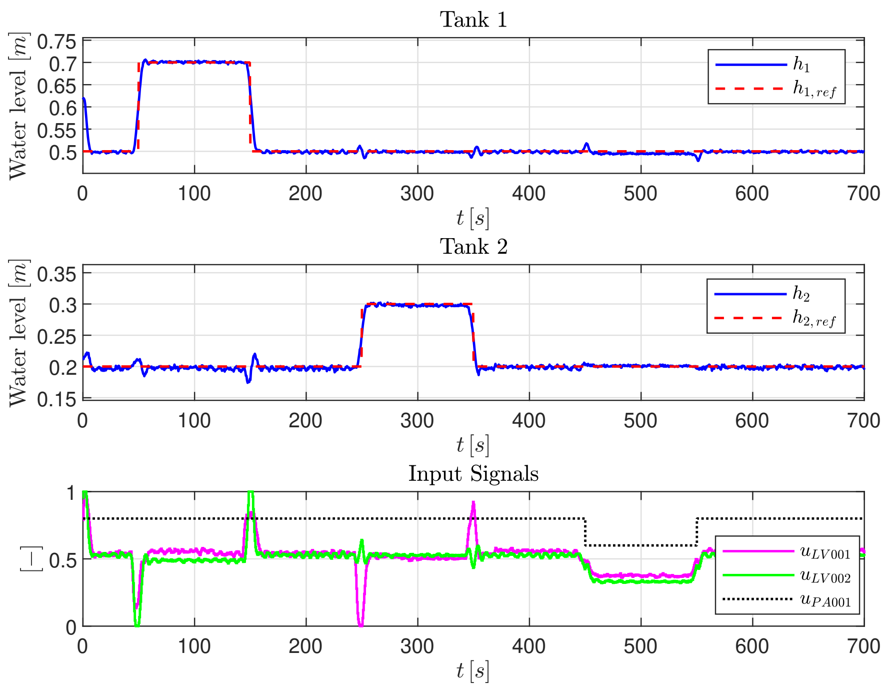


Figure 6.12: Experimental results of the closed-loop response w/ nonlinear MPC. The MPC is programmed to prioritize Tank 1 more than Tank 2.

closed-loop response:

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 200 \end{bmatrix}, \quad S = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}, \quad R = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.11}$$

Figure 6.13 presents the logged data from this experimental trial. The total quadratic

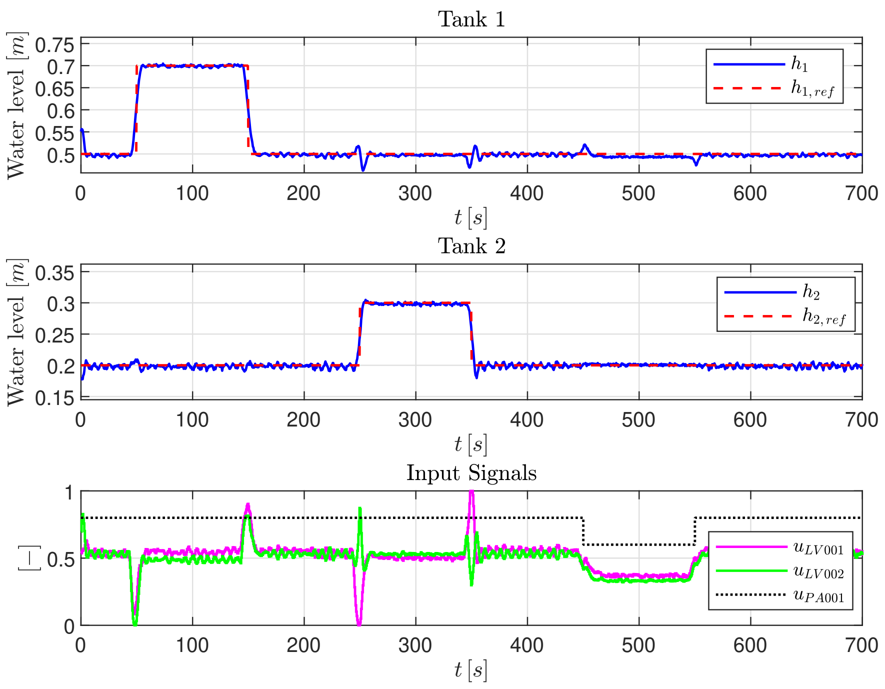


Figure 6.13: Experimental results of the closed-loop response w/ nonlinear MPC. The MPC is programmed to prioritize Tank 2 more than Tank 1.

costs of the three experiments with the nonlinear MPC are listen in Table 6.4.

Experimental Trial	J_{total}
Figure 6.11	4696.8
Figure 6.12	10284
Figure 6.13	12449

Table 6.4: Calculated quadratic costs of the experimental trials with the nonlinear MPC.

6.8 LQR Control

In this section, similar experimental trials to those presented thus far will be conducted, however, an LQR-based feedback controller will be used, rather than an MPC-based controller. As mentioned in the introduction of this chapter, a detailed description on how the LQR controller is designed can be found in Appendix A.

Firstly, consider the case where the desired closed-loop response prioritizes both tanks equally. For this experiment, the weighting matrices are chosen accordingly:

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}, \quad S = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{6.12}$$

The results of this experiment are presented in Figure 6.14. Next, consider the case

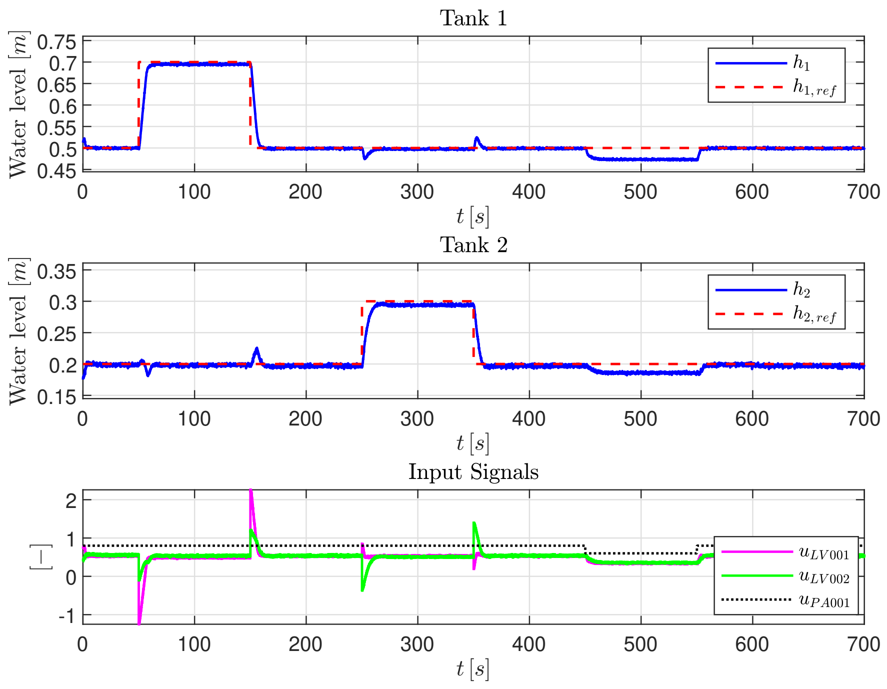


Figure 6.14: Experimental results of the closed-loop response w/ LQR control. The LQR controller is programmed to prioritize level control of both tanks equally.

where Tank 1 is prioritized more than Tank 2, which is done by selecting the following weighting matrices for the LQR controller:

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 10 \end{bmatrix}, \quad S = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{6.13}$$

The results of this experiment can be seen in Figure 6.15. The last experiment conducted on the LQR controller, is the case where the desired closed-loop response prioritizes Tank 2 more than Tank 1. The subsequent weighting matrices are selected for this experiment:

$$Q = \begin{bmatrix} 10 & 0 \\ 0 & 100 \end{bmatrix}, \quad S = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{6.14}$$

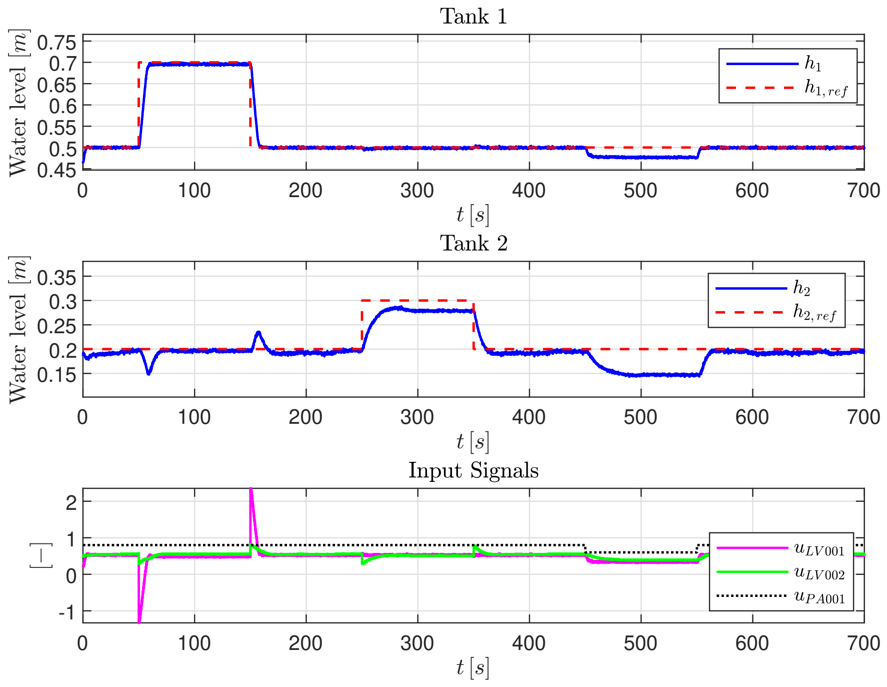


Figure 6.15: Experimental results of the closed-loop response w/ LQR control. The LQR controller is programmed to prioritize Tank 1 more than Tank 2.

which results in the closed-loop response shown in Figure 6.16. The total quadratic costs

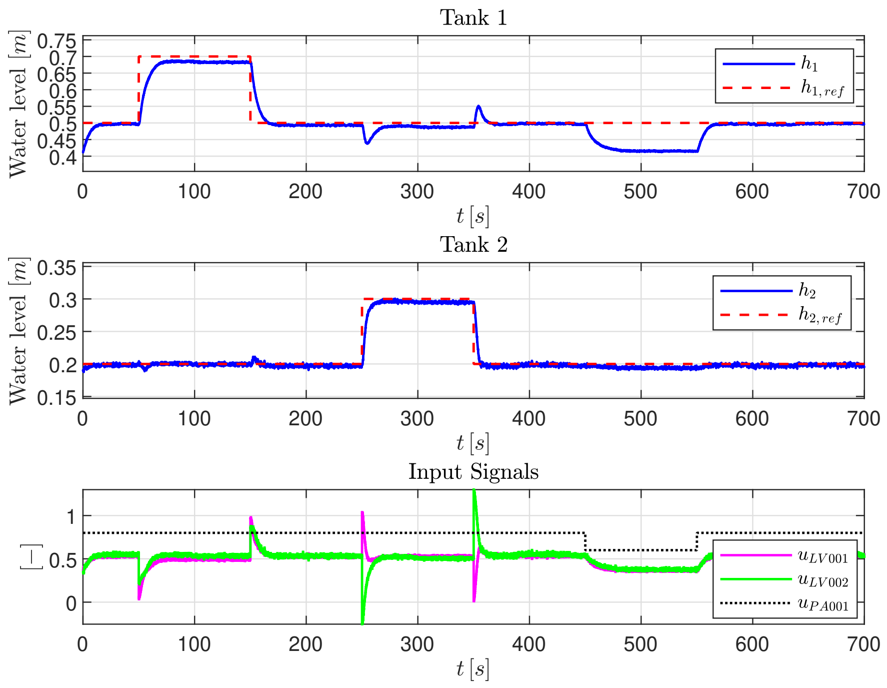


Figure 6.16: Experimental results of the closed-loop response w/ LQR control. The LQR controller is programmed to prioritize Tank 2 more than Tank 1.

of the three experiments with the LQR-based controller are listen in Table 6.5.

Experimental Trial	J_{total}
Figure 6.14	95230
Figure 6.15	40229
Figure 6.16	36652

Table 6.5: Calculated quadratic costs of the experimental trials with the LQR-based controller.

6.9 PID Control

In this section, three different PID controllers will be tested on the two-tank system. As previously mentioned, the details around the design of these PID controllers can be found in Appendix B. The goal of the experiments remain the same, i.e., to determine the set-point tracking and disturbance rejection abilities of the PID controllers. However, as there are no weighting matrices incorporated in the design of the PID controllers (in the sense of Q, R, and S for the MPC and LQR controller design), no total quadratic cost is calculated for the results with PID control. Instead, a more traditional performance index is used in this section, which is the *integral of absolute error* (IAE). For a brief introduction to IAE, see Appendix C.

Firstly, consider a simple PID feedback controller. The results of the experimental trial with this controller is shown in Figure 6.17. For the second experiment, the same PID feedback controller that resulted in the closed-loop response shown in Figure 6.17, is paired up with a feedforward control action, which accounts for changes in the set-points and the disturbance. Figure 6.18 shows the results of this experimental trial. Lastly, in addition to the PID feedback controller and the feedforward control action, a linear decoupler is added. The results of this experiment are shown in Figure 6.19. The IAE performance indices of the three experiments with the PID-based controllers are listen in Table 6.6.

Experimental Trial	IAE
Figure 6.17	1.94610
Figure 6.18	1.19164
Figure 6.19	0.89011

Table 6.6: Calculated IAE performance indices of the experimental trials with the PID-based controllers.

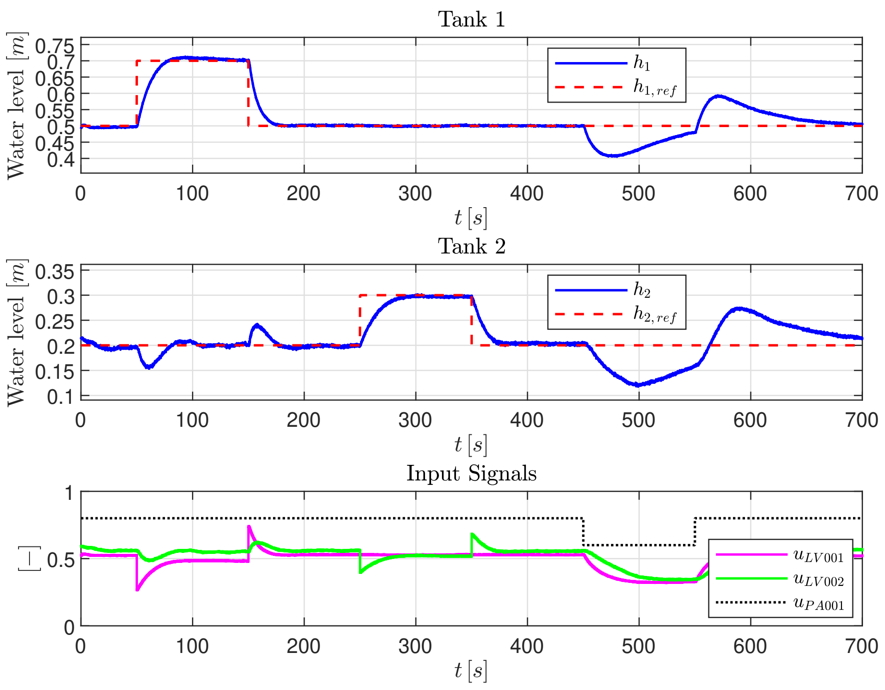


Figure 6.17: Experimental results of the closed-loop response w/ PID control. The PID controller uses a simple feedback loop.

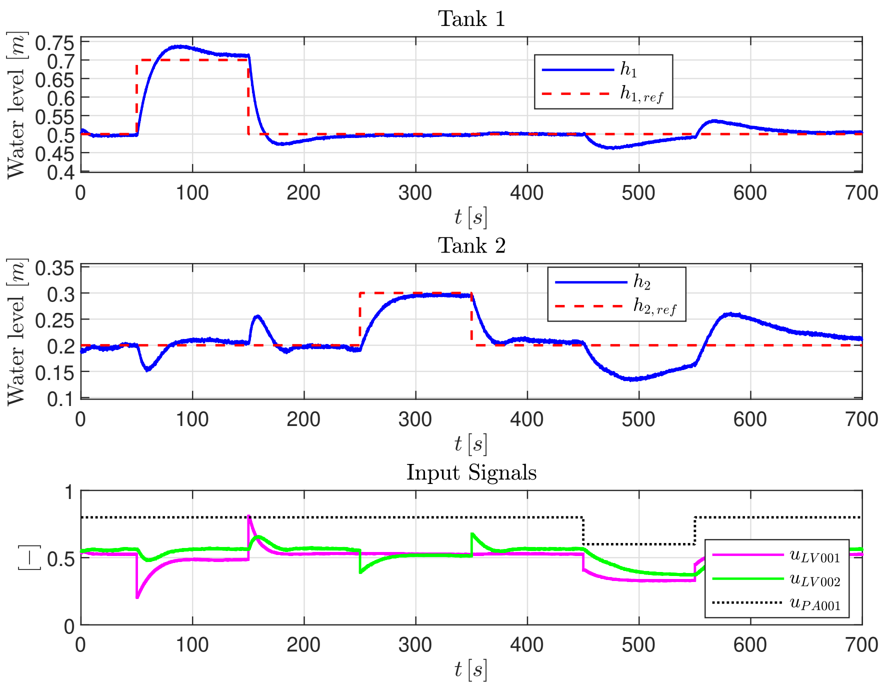


Figure 6.18: Experimental results of the closed-loop response w/ PID control. The PID controller uses a simple feedback loop, paird up with feedforward control on the set-points and the disturbance.

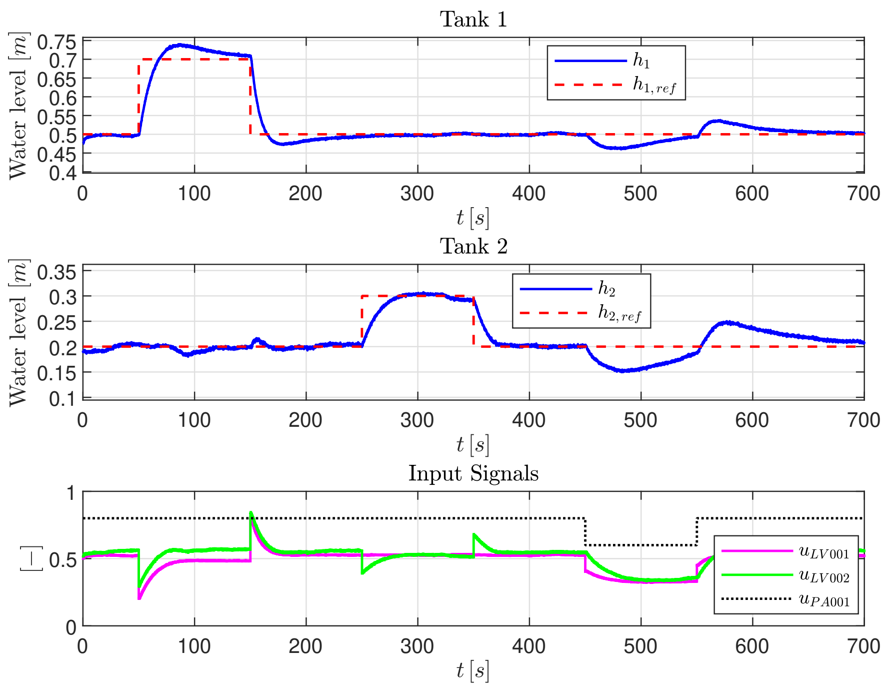


Figure 6.19: Experimental results of the closed-loop response w/ PID control. The PID controller uses a simple feedback loop, paired up with feedforward control on the set-points and the disturbance, in addition to a linear decoupler.

6.10 Linear MPC with State Estimation

The following section is a collaborative effort with fellow master's student Greta Bekerytė. Greta's work explores the *moving horizon estimation* (MHE), which is an optimization approach to state observers.

There are several similarities between the MPC approach to feedback controllers, and the MHE approach to state observers. These similarities can be linked to the duality between the two problems that, together, solve the linear-quadratic-Gaussian (LQG) problem: (i) the linear-quadratic-regulator (LQR) problem, and (ii) the linear-quadratic-estimate (LQE) problem.

In fact, under certain conditions, MPC can be reduced to the LQR problem. Similarly, the MHE can be reduced to the LQE problem. For a detailed discussion on MHE, see [39].

For these experimental trials, the nominal solution is given by:

$$\underbrace{\tilde{h}_1 = 0.3, \quad \tilde{h}_2 = 0.2, \quad \tilde{u}_{PA001} = 0.6, \quad \tilde{u}_{LV001} = 0.4264, \quad \tilde{u}_{LV002} = 0.3902}_{\mathfrak{NS}} \tag{6.15}$$

Given this $\mathfrak{NS}$, and a sampling time of $t_s = 1.5\,s$, the discrete time prediction model used by the linear MPC is given by:

$$\begin{cases} \begin{bmatrix} \Delta h_1(k+1) \\ \Delta h_2(k+1) \end{bmatrix} = \begin{bmatrix} 0.9692 & 0 \\ 0.01698 & 0.9866 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(k) \\ \Delta h_2(k) \end{bmatrix} \\ \qquad + \begin{bmatrix} -0.07269 & 0 & 0.09599 \\ 0.04011 & -0.04377 & 0.0008338 \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(k) \\ \Delta u_{LV002}(k) \\ \Delta u_{PA001}(k) \end{bmatrix} \\ \begin{bmatrix} \Delta \hat{h}_1(k) \\ \Delta \hat{h}_2(k) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(k) \\ \Delta h_2(k) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(k) \\ \Delta u_{LV002}(k) \\ \Delta u_{PA001}(k) \end{bmatrix} \end{cases} \tag{6.16}$$

Furthermore, the linear MPC is programmed to use a prediction horizon of $P = 10$, and a control horizon (with input blocking) of $M = \begin{bmatrix} 2 & 3 & 5 \end{bmatrix}$.

The experimental procedure is as follows: Three experimental trials are conducted, with

Algorithm 6.2 Experimental procedure w/ state estimation

```
1: TIME $< 150\,s \implies$ SYS $\leftarrow \mathfrak{NS}$  ▷ Drive system to nominal solution $\mathfrak{NS}$
2: TIME $\geq 150\,s \implies u_{PA001} \uparrow$  ▷ Positive step in disturbance
3: TIME $\geq 220.5\,s \implies u_{PA001} \uparrow$  ▷ Positive step in disturbance
4: TIME $\geq 291\,s \implies u_{PA001} \downarrow$  ▷ Negative step in disturbance
5: TIME $\geq 360\,s \implies h_{1,ref} \uparrow$  ▷ Start of positive rectangular pulse in reference
6: TIME $\geq 430\,s \implies h_{1,ref} \downarrow$  ▷ End of positive rectangular pulse in reference
7: TIME $\geq 501\,s \implies h_{2,ref} \uparrow$  ▷ Start of positive rectangular pulse in reference
8: TIME $\geq 600\,s \implies h_{2,ref} \downarrow$  ▷ End of positive rectangular pulse in reference
9: TIME $= 650\,s \implies$ end  ▷ End of experimental procedure
```

the following scenarios:

(i) The future evolution of both tanks are estimated based on the measurements of LT001 and LT002.

(ii) The future evolution of both tanks are estimated based on the measurements of LT002.

(iii) The future evolution of both tanks, and the input from the pump, are estimated based on the measurements of LT001 and LT002.

Again, for a more detailed discussion on how the MHE estimates the future evolution of the states, see [39].

The experimental trials for the three aforementioned scenarios are presented in Figure 6.20, 6.21, and 6.22, respectively. As seen in Figure 6.20, the linear MPC rejects

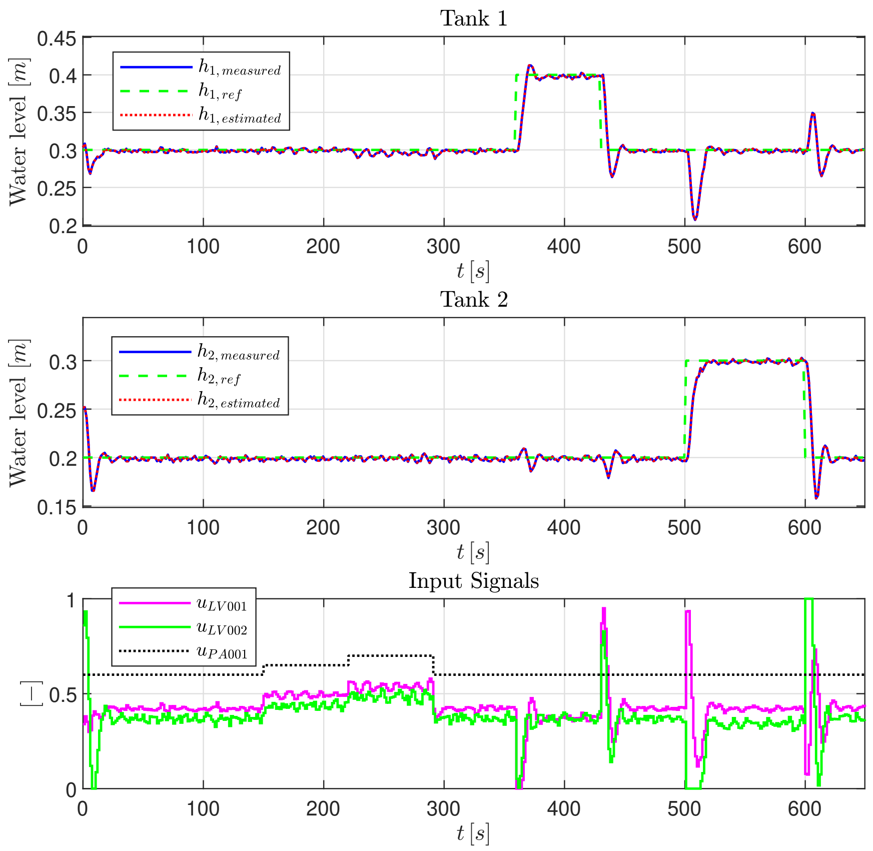


Figure 6.20: Experimental results of the closed-loop system w/ linear MPC. These results are from scenario (i) of the experiments where the linear MPC uses the estimates from the MHE state estimator.

the steps in the disturbance. Additionally, the water levels in the two tanks follow the set-points, however, there is a slight overshoot in Tank 1 when it is subjected to rectangular pulses in the set-point $h_{1,ref}$.

When only the measurements from LT002 are available, the water level in Tank 1 experiances more deviations from its set-point (see Figure 6.21). However, the estimated water level $h_{1,estimated}$, which, in fact, is the variable that the MPC is using when computing future control moves, does follow the set-point without the deviations seen in $h_{1,measured}$.

Lastly, when the MPC uses the estimate for the input from the pump $u_{PA001,estimated}$, the closed-loop response experiences more oscillations, as a result of the steps in the disturbance and the set-points (see Figure 6.22).

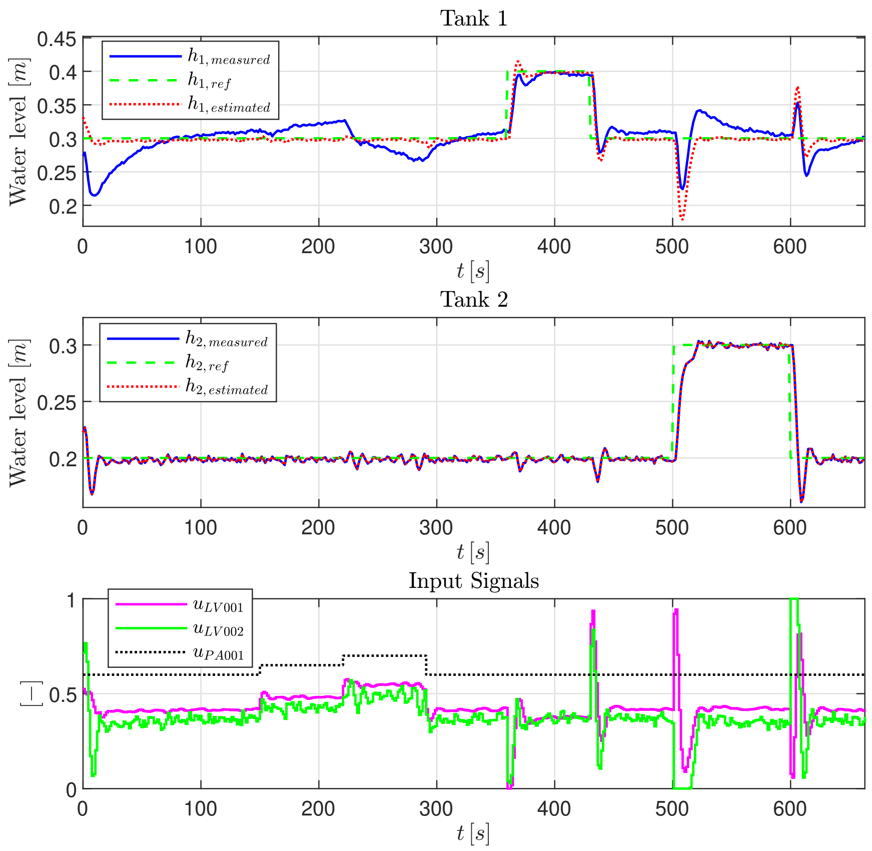


Figure 6.21: Experimental results of the closed-loop system w/ linear MPC. These results are from scenario (ii) of the experiments where the linear MPC uses the estimates from the MHE state estimator.

6.11 Analysis

Given all the experimental trials conducted thus far on the real two-tank system, the following observations are made:

(i) All of the MPC-based controllers provided a feasible control policy. I.e., the MPCs managed to compute the SQP/QP optimization problems within the sampling interval, which was $t_s = 0.1\,s$ for the linear-, explicit-, and adaptive-MPC, and $t_s = 0.5\,s$ for the nonlinear MPC.

Note that, due to its demanding computational requirements, the nonlinear MPC did *not* provide a feasible control policy for a sampling interval of $t_s = 0.1\,s$, which is why this is the only MPC with a higher sampling interval of $t_s = 0.5\,s$.

(ii) All of the MPC-based controllers satisfied the constraints on the manipulated variables during the experimental trials, which is helpful in avoiding saturation on the control inputs.

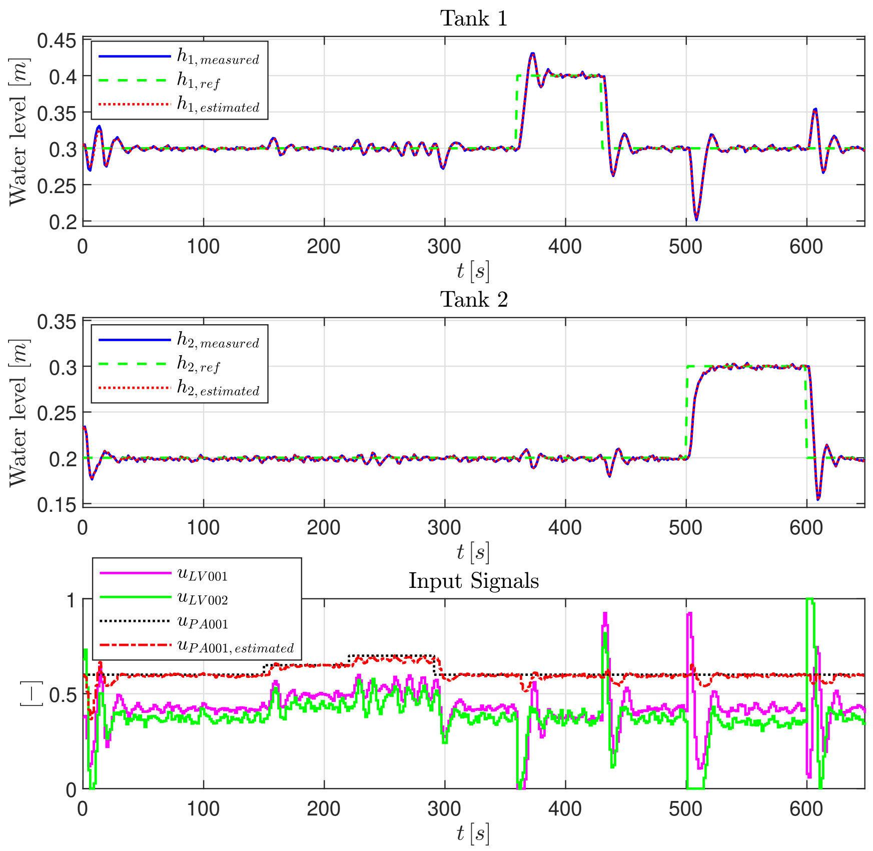


Figure 6.22: Experimental results of the closed-loop system w/ linear MPC. These results are from scenario (iii) of the experiments where the linear MPC uses the estimates from the MHE state estimator.

This is in contrast to the LQR-based controller, which does not look to satisfy any constraint. Since no constraints are imposed on the LQR-controller, note how the control inputs in Figure 6.14, 6.15, and 6.16 all exceed the possible range of $[0 - 1]$ on the control inputs, effectively causing saturation.

(iii) The linear-, explicit-, and nonlinear-MPC provided satisfactory control of the two-tank system. I.e., the closed-loop system, with these controllers, managed to follow the set-points, despite being subjected to changes in the set-points and the measured disturbance.

While the adaptive MPC exhibits satisfactory set-point tracking and disturbance rejection, the excessive use of the manipulated variables do render the adaptive MPC inferior to the other MPC-based controllers. Note how the adaptive MPC had the largest penalty on the manipulated variables and the manipulated variable rates (i.e., S and R), yet, this is the controller with the most aggressive closed-loop response, in terms of control input usage.

(iv) All of the MPC-based controllers exhibited strong decoupling capabilities when Tank 2 was prioritized more than Tank 1. I.e., changes in Tank 1 resulted in negligible disturbances in Tank 2, due to the decoupling properties of MPC (see Figure 6.4, 6.7, 6.10, and 6.13.)

Similarly, the LQR-based controller also provided this decoupling ability (see Figure 6.16).

This is in contrast to the closed-loop response with the PID-based controller, which experiences large disturbances in Tank 2, whenever the level in Tank 1 changes (see e.g. Figure 6.17 and 6.18, at $t = 50\,s$ and $t = 150\,s$).

Incorporating a linear decoupler with the PID feedback controller does result in less disturbances in Tank 2, whenever the level in Tank 1 changes, which can be seen in Figure 6.19.

(v) The MPC based controllers, with the exception of the explicit MPC, are able to provide look-ahead control, if future set-points and measured disturbances are known.

This is in contrast to the explicit MPC, the LQR-based controller, and the PID-based controller, which do not provide this previewing ability.

(vi) While being more prone to changes in the set-points and the disturbance, the linear MPC provides acceptable control of the two-tank system when using the estimates from the MHE state estimator.

Remark 6.2. It is important to emphasise, yet again, that the calculated total quadratic costs $\boldsymbol{J}_{total}$ presented in this chapter (see Table 6.1, 6.2, 6.3, 6.4) cannot be compared against each other directly, as previously done for the simulations in Chapter 5.

Recall that in Chapter 5, all of the MPC-based controllers were simulated using the same weighting matrices Q, R, and S, meaning that the objectives of the closed-loop systems were identical. When this is the case, it is possible to directly compare the total quadratic costs as a performance index, in order to distinguish which controller performed the worst/best at completing their objectives. However, this is not the case for the experimental trials presented in this chapter. I.e., the MPCs were programmed to use different weighting matrices, meaning that their objectives were different, thus, resulting in incomparable quadratic costs.

For the sake of comparison, the IAE performance index is computed for all of the controllers. In this case, only the scenario where the closed-loop system prioritizes both tanks are considered. I.e., the IAE performance index is only calculated for the experimental results from Figure 6.2, 6.5, 6.8, 6.11, and 6.14, respectively. Table 6.7

presents these IAE values, in addition to those calculated for the PID-based controllers (see Table 6.6).

Controller Type	***IAE***
Nonlinear MPC	0.0834
Linear MPC	0.1998
Explicit MPC	0.3158
LQR	0.3808
Adaptive MPC	0.6653
PID Feedback + Feedforward + Linear Decoupler	0.8901
PID Feedback + Feedforward	1.1916
PID Feedback	1.9461

Table 6.7: Calculated IAE performance index for all of the controller types, presented in increasing order. These values are calculated only from the experimental trials where both tanks were weighted equally in the desired closed-loop system.

Remark 6.3. When calculating the IAE performance indices in Table 6.7, it is important to recall that the nonlinear MPC uses a sampling interval of $t_s = 0.5\,s$, while the other controllers use a sampling interval of $t_s = 0.1\,s$. See Appendix C on how this information is included in the calculations.

Given the results in Table 6.7, there are several observations to be made, and parallels/differences to be drawn between the simulations in Chapter 5 and the experimental results presented in this chapter:

(i) In terms of the IAE performance index, the nonlinear MPC scored the lowest. I.e., the nonlinear MPC has the best performance, with respect to the total integral of the absolute error. This coincides with the simulation results, where the nonlinear MPC had the lowest total quadratic cost.

(ii) The adaptive MPC gives conflicting results between the simulations and the experimental trials. In the simulations, the adaptive MPC performed better than the explicit MPC, however, this is not the case for the experimental results.

As mentioned earlier in this chapter, the adaptive MPC exhibited the most aggressive use of the control inputs, even though this MPC was programmed with the biggest (in value) weighting matrices S and R, which, from a theoretical perspective, would imply the adaptive MPC to have the least aggressive use of the control inputs.

(iii) While the regular PID feedback controller performed the worst, i.e., it had the highest IAE index, Table 6.7 shows that the performance can be improved by incorporating a feedforward control action, in addition to a linear decoupler. However, the MPC-based controllers, and the LQR-based controller, all outperformed the PID-based controllers.

6.11.1 Modeling Error

As a final part of the analysis, it is important to recognise key factors as to *why* the experimental results did not match the simulations fully. While the experimental results did provide a close resemblance to the simulations, there are some differences, such as the performance of the adaptive MPC. One possible factor, which consists of multiple components, may be modeling error.

Assumptions

As mentioned in Chapter 3, no mathematical model can describe a physical system perfectly. There will always be physical phenomena that cannot be modeled, which is why there will always be some modeling error.

From a practical point of view, some assumptions were made along the modeling process of the two-tank system:

(i) The valve characteristics were approximated, such that 0% on the valve opening $z(t)$ resulted in 0% flow rate $f(z(t))$ (see Figure 3.5).

(ii) It was assumed that the control signals to the valves $u_{LV00i}(t)$, $i \in [1, 2]$ directly, and more importantly, immediately, corresponded to the valve opening $z_i(t)$ (see (3.7)).

(iii) As the pump characteristics were found experimentally, by measuring the flow $q_{PA001}(t)$ for a *finite* set of control inputs $u_{PA001}(t)$, there is an assumption, in the form of interpolation, as to what the pump characteristics are between the measured points.

Static Noise

Another assumption made during the simulations is perfect measurement of the water level in Tank 1 and Tank 2, through LT001 and LT002, respectively. In reality, there is some sensor noise that is not accounted for.

As an experiment, both valves LV001 and LV002 were shut closed. I.e., $u_{LV00i}(t) = 0$, $i \in [1, 2]$. Furthermore, the pump is turned off, which is done by setting $u_{PA001}(t) = 0.45$ (see Figure 3.6).

When the valves are shut closed, and no water from the pump flows to the system, the water (which is already within the two tanks) become motionless. In this scenario, if the measurements were indeed perfect, then, the signals from LT001 and LT002 would be constants. However, Figure 6.23 shows the real signals from the level transmitters, which clearly show signs of measurement noise.

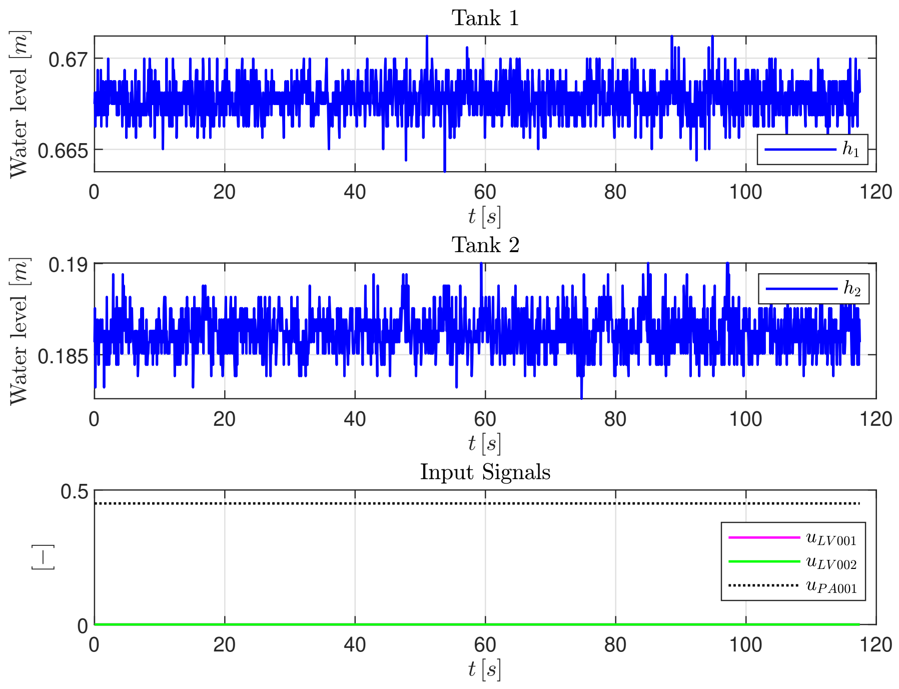


Figure 6.23: Measurements from a static environment in the two-tank system. I.e., the water levels are motionless.

Dynamic Disturbance

One factor that the mathematical model of the two-tank system does not account for, is the disturbance caused by the flowing water. When the water from the pump flows into Tank 1, it causes ripples in the water surface, which affects the measurements. Similarly, ripples in the water surface of Tank 2 are caused by the water that flows into it from Tank 1. Meaning that true steady states, in the mathematical models, do not equate to steady states in the real system.

For instance, consider the case where $u_{LV00i}(t) = 0.6$, $i \in [1, 2]$ and $u_{PA001}(t) = 0.8$, $\forall t$. When the system stabilizes, it is expected that a steady state is achieved, i.e., $\dot{h}(t) = 0$. However, due to the disturbance on the water surfaces, the measured water levels exhibit changes, which are larger than the measurement noise shown in Figure 6.23. This phenomenon is shown in Figure 6.24, where $u_{LV00i}(t)$, $i \in [1, 2]$ and $u_{PA001}(t)$ are held constant for $200\,s$. Most notably, are the changes in Tank 2, where the water ripples can be clearly seen, especially between $150\,s \le t \le 180\,s$.

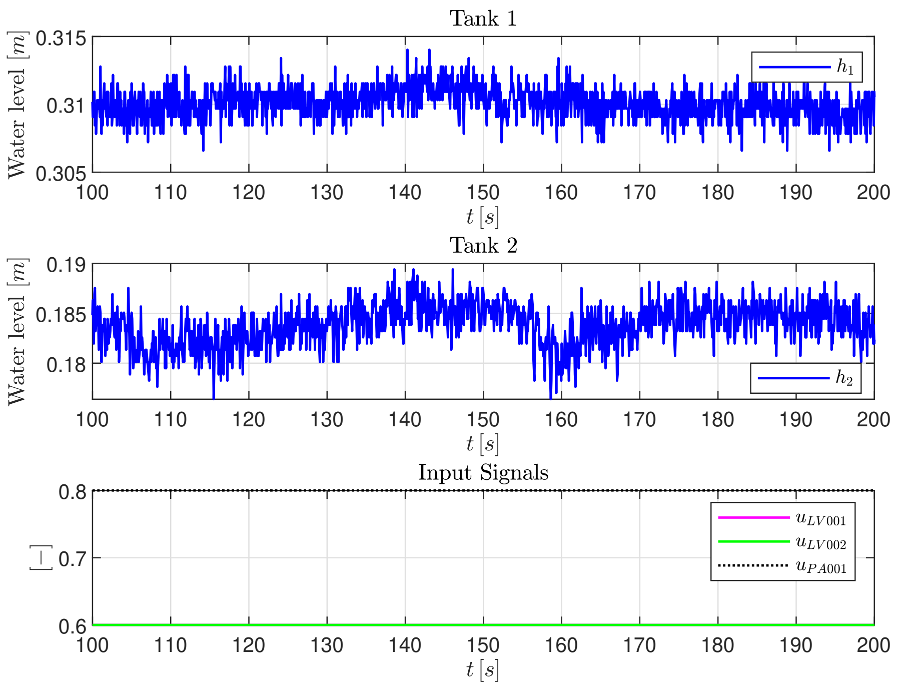


Figure 6.24: Measurements from a steady state environment in the two-tank system. I.e., the pump control input and the valve control inputs are held constant for a prolonged interval. While stabilizing on a certain water level, the system shows signs of disturbances caused by the ripples in the water surfaces, in addition to sensor noise.

Delay

Lastly, when modeling the two-tank system, there is an assumption of zero time delay. Obviously, this is not the case for the real system, as the water has to traverse multiple pipes, which takes time. In the simulations, however, it is assumed that the water enters/exits the tanks immediately.

The time delay can be shown through an experiment where a step is made in $u_{LV001}(t)$. The pump is turned off and valve LV002 is shut closed, as to not interfere with the experiment. The results of this experiment are shown in Figure 6.25, where a delay of

$\approx 2\,s$ can be seen from the time at which valve LV001 opens ($t \approx 35\,s$), to the time at which the water level in Tank 2 starts to rise ($t \approx 37$).

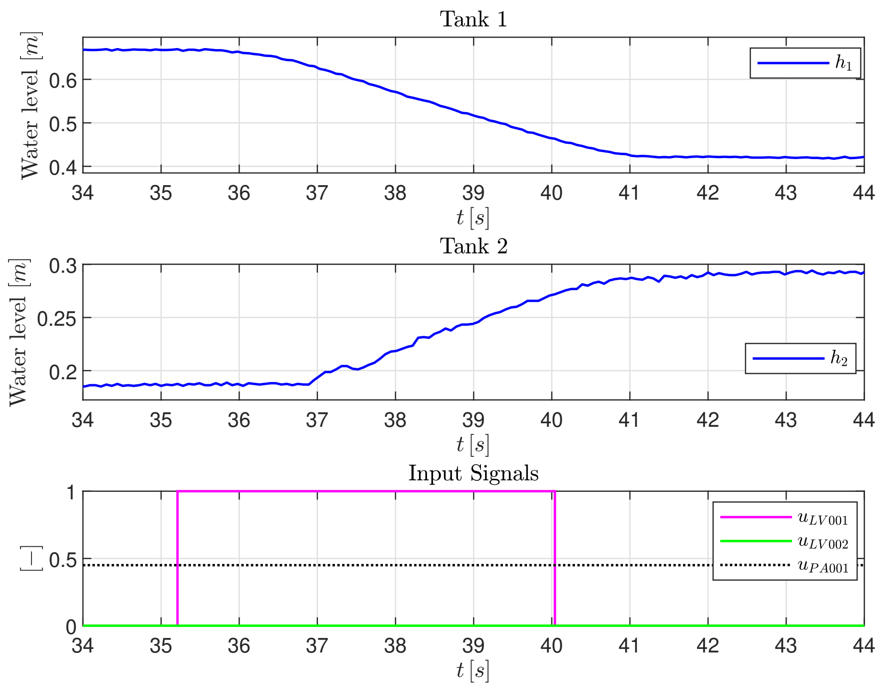


Figure 6.25: Experimental results showing the time delay between valve LV001, and the water level in Tank 2.

The assumptions made during the modeling procedure, the sensor noise, the dynamic disturbance of the water, and the time delay, all add up to the total modeling error between the two-tank model, and the real physical system. These system properties may be contributing factors as to why, for instance, the adaptive MPC exhibited unexpected excessive use of the control inputs, which was not the case in a simulation environment.

Chapter 7

Conclusions

7.1 Summary

This project has covered theoretical aspects of MPC-based controllers, such as the linear-, explicit-, adaptive-, and nonlinear-MPC. Practical design and implementation of the MPC-based controllers have also been covered in detail. MPC specifications such as the prediction horizon, control horizon, sampling time, quadratic objective function, weighting matrices, constraints (hard and soft), scaling factors, and the SQP/QP solver have all been clearly defined, and thorough justifications for the selection of these parameters have been provided.

Given its predictive nature, the prediction model is the crux of any MPC-based controller. Using balance laws, it was shown how a nonlinear mathematical model, which describes the dynamics of the two-tank system, can be obtained.

It was further shown how, through linearization of the nonlinear model, a linear model can be obtained. The linear model was expressed in a state-space form, where system properties such as controllability and stability were shown. Additionally, the linear model was reduced to algebraic equations through the Laplace transform. In the Laplace domain, several transfer functions were derived, which are useful when designing a PID feedback controller, a feedforward control action, and a linear decoupler.

A brief introduction to numerical solvers was provided, which included a description of the MATLAB `ode45` solver and the Euler method. Additionally, the zero-order hold discretization method was used to discretize the linear continuous time state-space model.

Given the prediction models and the MPC specifications, the MPC-based controllers were implemented and evaluated in a simulation environment. The goal of the simulations was to determine the controllers set-point tracking and disturbance rejecting abilities. I.e., rectangular pulses in the set-points and the disturbance were introduced during the simulations, in order to observe how the closed-loop systems would respond to these changes.

The total quadratic costs for the different MPC-based controllers were calculated based on the simulation results. Based on these calculated quadratic costs, the nonlinear MPC proved to have the superior closed-loop response with the lowest total quadratic cost. Conversely, the explicit MPC proved to have the inferior closed-loop response with the highest total quadratic cost.

In addition to calculating the total quadratic costs, the code execution times of the different MPC-based controllers were recorded. Based on these recordings, it was shown that the explicit MPC provided the fastest code execution time, while the nonlinear MPC provided the slowest code execution time.

Experimental validation of the MPC-based controllers was provided, as the controllers were implemented and tested on the real two-tank system. In addition to the MPC-based controllers, an LQR-based controller and three PID-based controllers were also designed, implemented, and tested on the system.

The experimental results showed that the MPC-based controllers do provide satisfactory control of the two-tank system, where the water levels follow the set-points, and the disturbance is rejected by the respective closed-loop systems. This was also the case for the LQR- and PID-based controllers.

When evaluating the performance of the different controllers, the integral of absolute error (IAE) performance index was used. Based on the IAE performance indices, it can be concluded that the nonlinear MPC exhibited the superior closed-loop response. In contrast, the simple PID feedback controller showcased the worst closed-loop performance, with the highest overall IAE score.

Contrary to the simulation results, the adaptive MPC exhibited excessive use of the control inputs during the experimental trials on the two-tank system, effectively degrading its performance. This may be explained by modeling errors such as sensor noise, additional disturbances, and delays in the system, which, due to the continuous updating of the prediction model in an adaptive MPC, may result in unpredicted behaviour.

Lastly, experimental trials were conducted, where the linear MPC used a combination of real measurements and state estimates to control the two-tank system. The state estimates were obtained by the MHE state estimator. These experimental results showed that it is possible to control the system with the use of state estimates, which is useful in practical scenarios where it may not be possible to measure the states directly, or, in the scenarios where the measurements are highly corrupted by noise, in which, an estimate is the preferred option.

7.2 Advantages

There are several advantages to the MPC-based controllers, which may explain the superior IAE performance indices, compared to the IAE performance indices of the LQR- and PID-based controllers:

(i) The MPC-based controllers, with the exception of the explicit MPC, have the ability to account for future changes in the set-points and the disturbance, respectively, if these values are known beforehand. This previewing ability comes from the predictive nature of MPC, where it optimizes the control inputs over a P-step prediction horizon. This previewing ability is not an ability that the infinite-horizon LQR-based controller, nor the PID-based controllers possess.

(ii) Similarly to the LQR-based controller, the MPC-based controllers can be optimized for specific objectives, by adjusting the weighting matrices accordingly. This allows for flexibility for the operating engineer, which has the ability to, e.g., program the closed-loop system such that it prioritizes Tank 2 more than Tank 1.

This is added flexibility that the PID-based controllers do not provide, as it is not possible to, e.g., program the PID controller such that the control input rates are limited, which, in the case of the LQR- and MPC-based controllers, is achieved by simply increasing the S weighting matrix.

(iii) All of the MPC-based controllers, and the LQR-based controller, showcased strong decoupling abilities. Since the two tanks are coupled, any change in Tank 1 acts as a disturbance on Tank 2. This is most evident by studying the experimental results of the simple PID feedback controller (see Figure 6.17, at $50\,s \leq t \leq 200\,s$). Such disturbances on Tank 2, as shown in Figure 6.17, are highly suppressed in the experimental results of the MPC-based controllers and the LQR-based controller. Additionally, if the MPC- and LQR-based controllers are programmed to prioritize Tank 2 more than Tank 1, then, the decoupling ability is reinforced even more.

It was shown that a linear decoupler could be added to aid the PID feedback controller in eliminating this disturbance, however, as seen in Figure 6.19, Tank 2 is still slightly affected by set-point changes in Tank 1, and strongly affected by changes in the disturbance, which is not the case for the MPC- and LQR-based controllers.

(iv) A big advantage of the MPC-based controllers is their ability to satisfy constraints (both hard and soft) on the control inputs, the control input rates, *and* the outputs. The simulations and the experimental results showed that all of the MPC-based controllers satisfied the constraints imposed on the control inputs at all times. This is a big advantage, especially when saturation can be an issue.

This is in contrast to the LQR- and PID-based controllers, which do not account for any constraints in their closed-loop systems, respectively.

Like the weighting matrices, the ability to subject the optimization problem of an MPC to constraints gives added flexibility, which, again, is not available for the LQR- and PID-based controllers.

(v) Contrary to the LQR- and PID-based controllers, the nonlinear MPC has the capabilities of using a nonlinear prediction model, optimized over a nonlinear objective function, subjected to nonlinear constraints.

This is a big advantage, considering the fact that the LQR- and PID-based controllers are restricted to linear models, which are often obtained through linearization around a nominal solution. I.e., the linear model used for the design of the LQR- and PID-based controllers is only applicable in the vicinity of the nominal solution.

For highly nonlinear systems, this means that the linear model quickly becomes inapplicable as the system is driven away from the neighborhood of the point of linearization, effectively rendering the designed LQR- and PID-based controllers non-optimal.

Since the nonlinear MPC utilizes a nonlinear prediction model, this is not an issue, and the closed-loop system is able to provide optimal control over the entire range of the system, and not just in the vicinity of a nominal solution.

(vi) Lastly, as shown in this project, the MPC-based controllers do have the capabilities of controlling a fast sampling system. Recall that the linear-, explicit-, and adaptive-MPC all provided feasible control policies with a sampling interval of $t_s = 0.1\,s$, while the nonlinear MPC provided a feasible control policy with a sampling interval of $t_s = 0.5\,s$.

While the first intended use cases of MPC was slow industrial processes with sampling intervals ranging up to several minutes [9], modern technology provides strong computational capabilities, which allows for the SQP/QP optimization problems to be solved within much shorter time frames. This technological evolution, since the inception of MPHC, makes the MPC-based controllers a viable option for a wide range of systems, not only restricted to slow sampling systems such as industrial processes.

7.3 Disadvantages

While there are several advantages to the MPC approach, there are also some disadvantages that were noted during this project:

(i) As mentioned earlier, the prediction model is the crux of any MPC-based controller. Since the MPC approach is so dependent on the prediction model, the closed-loop system is especially vulnerable to modeling errors.

This can be seen by comparing the control inputs of the LQR- and PID-based controllers, with the control inputs of the MPC-based controllers. While the control inputs from the LQR- and PID-based controllers are less abrupt and smooth, the control inputs from the MPC-based controllers show more signs of being affected by modeling errors such as noise and disturbances. I.e., the control inputs from the MPC-based controllers appear more aggressive and 'jagged'.

(ii) Another disadvantage is that the MPC approach is computationally demanding. The receding horizon policy does require the MPC to solve an SQP/QP optimization problem on every control interval, where, out of the M calculated optimal control moves, only the first optimal control move $u_{opt}(k)$ is applied to the system and the rest are discarded.

While the SQP/QP optimization problems proved to be manageable for fast sampling systems in this project, this might not be the case for larger systems, as the SQP/QP optimization problems become more complex with a larger number of controlled outputs and manipulated variables.

(iii) Although the explicit MPC provided the fastest code execution time (compared to the other MPC-based controllers), it is the controller that requires the most storage on the system hardware. It was shown that the number of polyhedral regions increased drastically, as the control horizon M became larger (see Table 5.3). This sets a physical limit on how complex the explicit MPC can get, before the number of polyhedral regions and predefined piecewise affine control laws exceed the available storage space on the system hardware.

(iv) While the weighting matrices Q, R, and S do provide flexibility to the control engineer, with respect to the desired behavior of the closed-loop system, selecting the right weighting matrices can be a tedious process. I.e., some trial and error is required before the correct weighting matrices, which result in the desired closed-loop system, are found.

(v) Lastly, while not directly a disadvantage from a performance point of view, the MPC specifications are numerous. When designing an MPC-based controller, there are several specifications that need to be accounted for, such as the prediction horizon, control horizon, sampling time, quadratic objective function, weighting matrices, constraints (hard and soft), scaling factors, and the SQP/QP solver. From a practical point of view, the amount of effort needed to correctly tune an MPC-based controller can be considered as a disadvantage, especially when other controller options, such as LQR- and PID-based controllers, have significantly less specifications that need tuning.

7.4 Conclusion

Conclusively, the MPC approach proves to be a viable option for medium sized, fast sampling systems. From the simulations and the experimental trials presented in this project, the MPC approach exhibited satisfactory set-point tracking and disturbance rejecting abilities.

Additionally, the linear-, explicit-, and nonlinear-MPC all exhibited superior closed-loop systems (with respect to the IAE performance index) in comparison to more traditional controller techniques such as LQR- and PID-based controllers.

The MPC's look-ahead control policy provides a closed-loop system which is able to take action *before* changes in the set-points and the disturbances occur, if this information is known beforehand.

The weighting matrices Q, R, and S do provide extended flexibility to the control engineer, where the closed-loop system can be fine-tuned, such that a desired response is achieved.

The MPC can be subjected to constrains on the controlled outputs, manipulated inputs, and manipulated input rates, such that saturations on the outputs and inputs are avoided at all times during operation.

The MPC provides strong decoupling abilities, where the effects of coupled outputs/inputs are minimized. In the case of the two-tank, which is a coupled system, changes in Tank 1 resulted in negligible disturbances in Tank 2 due to the decoupling abilities of the MPC's, especially when Tank 2 was prioritized more than Tank 1 in the closed-loop system.

Due to the MPC's heavily reliance on the prediction model, it is absolutely critical that the modeling error is kept at a minimum. Otherwise, the future predicted trajectory of the system might be inaccurate, effectively rendering the calculated optimal control moves inappropriate.

7.5 Future Directions

An accurate prediction model is the key to a well-functioning MPC. Recent studies in the field of MPC emphasise this notion by incorporating advanced system modeling techniques, such that the modeling error is kept at a minimum.

For instance, [40] discusses the use of data-driven prediction models in combination with model predictive control. [41] incorporates machine learning, in the form of complex neural network architectures, to describe the dynamics of a system, which the MPC uses as a prediction model.

Rather than a dynamic model obtained via balance laws, an interesting avenue is to incorporate more advanced modeling techniques, such as those in the aforementioned research papers, to describe the dynamic behaviour of the two-tank system.

From an engineering perspective, there are some changes that can be incorporated to the system design, such that the modeling error is mitigated. For instance, a low-pass filter can be added to the measured outputs, such that the high-frequency measurement noise from LT001 and LT002 are removed.

Another change in the design of the two-tank system can be to manipulate the pathway of the water from the pipe into the tank. I.e., instead of the water cascading from the pipe, and flowing directly onto the water surface (which gives rise to large ripples in the water surface, effectively acting as a disturbance), a hose can be attached to the end of the pipe, which directs the water to the bottom of the tank. This way, as the water level rises, the hose submerges, meaning that the tank is filled from the bottom. Thus, no water disrupts the water surface, effectively mitigating this dynamic disturbance which was shown in Figure 6.24.

Lastly, another interesting future direction for a possible improvement to the overall closed-loop system with an MPC, is to add an integral action, similarly to the integral action from a PID-controller. With the integral action, it may be possible to completely remove any steady-state error, effectively improving the set-point tracking and the disturbance rejecting abilities of the closed-loop system.

Appendix A

LQR Control

Linearizing the nonlinear two-tank model about the nominal solution:

$$\underbrace{\tilde{h}_1 = 0.5, \quad \tilde{h}_2 = 0.2, \quad \tilde{u}_{PA001} = 0.8, \quad \tilde{u}_{LV001} = 0.5317, \quad \tilde{u}_{LV002} = 0.5680}_{\mathfrak{NS}} \tag{A.1}$$

results in the following continuous time linear state-space model:

$$\begin{cases} \begin{bmatrix} \Delta\dot{h}_1(t) \\ \Delta\dot{h}_2(t) \end{bmatrix} = \begin{bmatrix} -0.02295 & 0 \\ 0.01275 & -0.01559 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(t) \\ \Delta h_2(t) \end{bmatrix} \\ \qquad + \begin{bmatrix} -0.07189 & 0 \\ 0.03994 & -0.03805 \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(t) \\ \Delta u_{LV002}(t) \end{bmatrix} \\ \qquad + \begin{bmatrix} 0.0450 \\ 0 \end{bmatrix} \cdot \Delta u_{PA001}(t) \\ \begin{bmatrix} \Delta\hat{h}_1(t) \\ \Delta\hat{h}_2(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(t) \\ \Delta h_2(t) \end{bmatrix} \end{cases} \tag{A.2}$$

Discretizing (A.2) using the ZOH method with a sampling period of $t_s = 0.1\,s$, results in the following discrete time linear state-space model:

$$\begin{cases} \begin{bmatrix} \Delta h_1(k+1) \\ \Delta h_2(k+1) \end{bmatrix} = \begin{bmatrix} 0.9977 & 0 \\ 0.001273 & 0.9984 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(k) \\ \Delta h_2(k) \end{bmatrix} \\ \qquad + \begin{bmatrix} -0.007181 & 0 \\ 0.003986 & -0.003802 \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(k) \\ \Delta u_{LV002}(k) \end{bmatrix} \\ \qquad + \begin{bmatrix} 0.004495 \\ 2.866 \cdot 10^{-6} \end{bmatrix} \cdot \Delta u_{PA001}(k) \\ \begin{bmatrix} \Delta \hat{h}_1(k) \\ \Delta \hat{h}_2(k) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(k) \\ \Delta h_2(k) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(k) \\ \Delta u_{LV002}(k) \\ \Delta u_{PA001}(k) \end{bmatrix} \end{cases} \tag{A.3}$$

As described in Chapter 2, a steady state optimum gain matrix K_∞ can be found by solving the following discrete time infinite-horizon LQR problem:

$$K_\infty = (R + B^T P_\infty B)^{-1} B^T P_\infty A \tag{A.4}$$

where:

$$P_\infty = A^T P_\infty A - A^T P_\infty B (R + B^T P_\infty B)^{-1} B^T P_\infty A + Q \tag{A.5}$$

The matrices A and B can be found from the state-space model (A.3):

$$A = \begin{bmatrix} 0.9977 & 0 \\ 0.001273 & 0.9984 \end{bmatrix}, \quad B = \begin{bmatrix} -0.007181 & 0 \\ 0.003986 & -0.003802 \end{bmatrix} \tag{A.6}$$

The weighting matrices Q and R are selected depending on the priority of the desired closed-loop response. For the case where both tanks are weighted equally, the LQR uses the following weighting matrices:

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}, \quad S = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{A.7}$$

which, when solving (A.4) and (A.5) for K_∞, yields the following optimum steady state gain matrix:

$$K_\infty = \begin{bmatrix} -8.7864 & 3.1019 \\ -3.2865 & -8.8629 \end{bmatrix} \tag{A.8}$$

When prioritizing Tank 1 more than Tank 2, the LQR-controller uses the weighting matrices:

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 10 \end{bmatrix}, \quad S = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{A.9}$$

which, when solving (A.4) and (A.5) for K_∞, yields the following optimum steady state gain matrix:

$$K_\infty = \begin{bmatrix} -9.2566 & 0.4000 \\ -1.3090 & -2.7349 \end{bmatrix} \tag{A.10}$$

Finally, when prioritizing Tank 2 more than Tank 1, the LQR-controller uses the weighting matrices:

$$Q = \begin{bmatrix} 10 & 0 \\ 0 & 100 \end{bmatrix}, \quad S = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{A.11}$$

which, when solving (A.4) and (A.5) for K_∞, yields the following optimum steady state gain matrix:

$$K_\infty = \begin{bmatrix} -2.3425 & 5.1162 \\ -1.6844 & -7.8983 \end{bmatrix} \tag{A.12}$$

Alternatively, this can be done in MATLAB using the following syntax:

```
[K, P, S] = lqr(sys, Q, R);
```

where `sys`, `Q`, and `R` are the state-space model, output weighting matrix Q, and input weighting matrix R, respectively. The outputs from this function, i.e., `K`, `P`, and `S` are the gain matrix K_∞, the solution of the associated algebraic Riccati equation P_∞, and the poles of the closed-loop system, respectively.

Appendix B

PID Control

Linearizing the nonlinear two-tank model about the nominal solution:

$$\underbrace{\tilde{h}_1 = 0.5, \quad \tilde{h}_2 = 0.2, \quad \tilde{u}_{PA001} = 0.8, \quad \tilde{u}_{LV001} = 0.5317, \quad \tilde{u}_{LV002} = 0.5680}_{\mathfrak{NS}} \tag{B.1}$$

results in the following continuous time linear state-space model:

$$\begin{cases} \begin{bmatrix} \Delta\dot{h}_1(t) \\ \Delta\dot{h}_2(t) \end{bmatrix} = \begin{bmatrix} -0.02295 & 0 \\ 0.01275 & -0.01559 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(t) \\ \Delta h_2(t) \end{bmatrix} \\ \qquad + \begin{bmatrix} -0.07189 & 0 \\ 0.03994 & -0.03805 \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{LV001}(t) \\ \Delta u_{LV002}(t) \end{bmatrix} \\ \qquad + \begin{bmatrix} 0.0450 \\ 0 \end{bmatrix} \cdot \Delta u_{PA001}(t) \\ \begin{bmatrix} \Delta\hat{h}_1(t) \\ \Delta\hat{h}_2(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta h_1(t) \\ \Delta h_2(t) \end{bmatrix} \end{cases} \tag{B.2}$$

Performing the Laplace transform on (B.2) results in the following equations (see Section 3.4.2):

$$s\Delta H_1(s) = -0.02295\Delta H_1(s) - 0.07189\Delta U_{LV001}(s) + 0.045\Delta U_{PA001}(s) \tag{B.3}$$

$$\begin{aligned} s\Delta H_2(s) = &\,0.01275\Delta H_1(s) - 0.01559\Delta H_2(s) \\ &+ 0.03994\Delta U_{LV001}(s) - 0.03805\Delta U_{LV002}(s) \end{aligned} \tag{B.4}$$

Through algebraic manipulation of (B.3) and (B.4), the subsequent transfer functions are found, respectively:

$$G_1(s) = \frac{H_1(s)}{U_{LV001}(s)} = \frac{-0.07189}{s + 0.2295} = \frac{-3.13}{43.57s + 1} \tag{B.5}$$

$$G_2(s) = \frac{H_2(s)}{U_{LV002}(s)} = \frac{-0.03805}{s + 0.01559} = \frac{-2.44}{64.14s + 1} \tag{B.6}$$

The control law for the PID-controller is given by:

$$u(t) = \tilde{u} + K_p e(t) + \frac{K_p}{T_i} \int_0^t e(\tau) d\tau + K_p T_d \frac{de(t)}{dt} \tag{B.7}$$

where

- $\tilde{u}$ is the nominal control action. I.e., the point at which the nonlinear model was linearized, with respect to the control input.
- $e(t)$ is the tracking error between the controlled output and the reference.
- K_p is known as the proportional gain.
- $\frac{K_p}{T_i}$ is known as the integral gain.
- $K_p T_d$ is known as the derivative gain.

To find the PID control parameters K_p, T_i, and T_d, the Internal Mode Control (IMC) method is used. For a first order transfer function, which is the case in (B.5) and (B.6), the IMC method defines the PID control parameters as (see [20] Section 12.2.2, pp. 205-207):

$$K_p = \frac{\tau}{\kappa \tau_c} \tag{B.8}$$

$$T_i = \tau \tag{B.9}$$

$$T_d = 0 \tag{B.10}$$

where:

- κ is gain of the first order transfer function.
- τ is the time constant of the first order transfer function.
- τ_c is the desired time constant of the closed-loop system.

Table B.1 shows the different parameters for the simple PID feedback controller for the two tanks, respectively.

	$G_1(s)$	$G_2(s)$
κ	-3.13	-2.44
τ	43.57	64.14
τ_c	$\tau/4 = 10.9$	$\tau/4 = 16.04$
K_p	-1.27	-1.63
T_i	43.57	64.14
T_d	0	0

Table B.1: Parameters for the PID feedback controller.

B.1 Feedforward Control

The feedforward control action is added only to Tank 1. Firstly, the equation (B.3) is manipulated into an expression for $\Delta H_1(s)$:

$$\Delta H_1(s) = \underbrace{\frac{-0.07189}{s+0.02295}}_{G_1(s)} \Delta U_{LV001}(s) + \underbrace{\frac{0.045}{s+0.02295}}_{G_d(s)} \Delta U_{PA001}(s) \tag{B.11}$$

Now, suppose that the water level in Tank 1 follows the reference perfectly, i.e., $\Delta H_1(s) = \Delta H_{1,ref}(s)$. Furthermore, suppose that the control input in (B.11) is the control input from the feedforward control action, i.e., $\Delta U_{LV001}(s) = \Delta U_{F.Forward}(s)$. Given these definitions, (B.11) can be rewritten subsequently:

$$\Delta H_{1,ref}(s) = \underbrace{\frac{-0.07189}{s+0.02295}}_{G_1(s)} \Delta U_{F.Forward}(s) + \underbrace{\frac{0.045}{s+0.02295}}_{G_d(s)} \Delta U_{PA001}(s) \tag{B.12}$$

Solving (B.13) for $\Delta U_{F.Forward}(s)$ results in:

$$\begin{aligned}\Delta U_{F.Forward}(s) &= \frac{s+0.02295}{-0.07189}\Delta H_{1,ref}(s) - \frac{0.045}{s+0.02295}\cdot\frac{s+0.02295}{-0.07189}\Delta U_{PA001}(s)\\ &= \underbrace{\frac{-0.02295}{0.07189}}_{G_{F.,ref}(s)}\Delta H_{1,ref}(s) + \underbrace{\frac{0.045}{0.07189}}_{G_{F.,d}(s)}\Delta U_{PA001}(s)\end{aligned} \tag{B.13}$$

which is the contribution from the feedforward control action. Note the simplification in $G_{F.,ref}(s)$, which is necessary to achieve a proper[1] transfer function.

B.2 Linear Decoupler

Lastly, a linear decoupler is added to the PID feedback controller and the feedforward control action. A linear decoupler can be constructed through a relative gain array

[1]I.e., the degree of the numerator does not exceed the degree of the denominator.

(RGA). This method will not be covered here, however, a detailed discussion on RGA can be found in [20] Section 18.2.1, pp. 332-334.

In short, the contribution from the linear decoupler is given by:

$$\Delta U_{Decoupler,1}(s) = T_{1,2}(s)\Delta U_{LV002}(s) \tag{B.14}$$

$$\Delta U_{Decoupler,2}(s) = T_{2,1}(s)\Delta U_{LV001}(s) \tag{B.15}$$

where $T_{1,2}(s)$ and $T_{2,1}(s)$ are defined as:

$$T_{1,2}(s) = -\frac{G_{1,2}(s)}{G_1(s)} \tag{B.16}$$

$$T_{2,1}(s) = -\frac{G_{2,1}(s)}{G_2(s)} \tag{B.17}$$

Note that the subscript $G_{i,j}$ denotes the transfer function between the control input j and the output i. I.e.,

$$G_{1,2}(s) = \frac{\Delta H_1(s)}{\Delta U_{LV002}(s)} = 0 \tag{B.18}$$

$$G_{2,1}(s) = \frac{\Delta H_2(s)}{\Delta U_{LV001}(s)} = \frac{-2.56}{64.14s+1} \tag{B.19}$$

Inserting (B.18) and (B.19) into (B.16) and (B.17), respectively, results in the following transfer functions:

$$T_{1,2}(s) = -\frac{0}{G_1(s)} = 0 \tag{B.20}$$

$$T_{2,1}(s) = -\frac{-2.56}{64.14s+1} \cdot \frac{64.14s+1}{-2.44} = -1.049 \tag{B.21}$$

Finally, inserting (B.20) into (B.14), and (B.21) into (B.15), gives the expression for the contribution from the linear decoupler:

$$\Delta U_{Decoupler,1}(s) = 0 \tag{B.22}$$

$$\Delta U_{Decoupler,2}(s) = -1.049\Delta U_{LV001}(s) \tag{B.23}$$

Appendix C

Integral of Absolute Error

One common performance index when comparing controllers is the integral of absolute error (IAE). In the continuous time domain, the IAE performance index is calculated subsequently:

$$IAE = \int_0^\infty |e| dt \tag{C.1}$$

where e is the reference tracking error of the closed-loop system.

In the discrete time domain, the IAE performance index is calculated as:

$$IAE = \sum_{k=0}^{\infty} h|e(k)| \tag{C.2}$$

where h is the time between sample k and $k-1$, i.e., the sampling period t_s.

Note that, in practise, the integral/sum of the IAE performance index is calculated over a finite time interval, and not an infinite integral/sum. The finite time interval is often limited to the length of the simulation/experimental trial under consideration.

Appendix D

Box Plot

A *box and whisker* plot (also called a *box* plot) is a commonly used graphical tool among statisticians, which illustrates key statistical properties of a data set.

Suppose that a data set is generated by selecting 100 uniformly distributed pseudorandom integers ranging from $[40 - 60]$, plus, 10 uniformly distributed pseudorandom integers ranging from $[20 - 30]$, plus, 10 uniformly distributed pseudorandom integers ranging from $[70 - 80]$. Given this data set, the box plot seen in Figure D.1 can be generated.

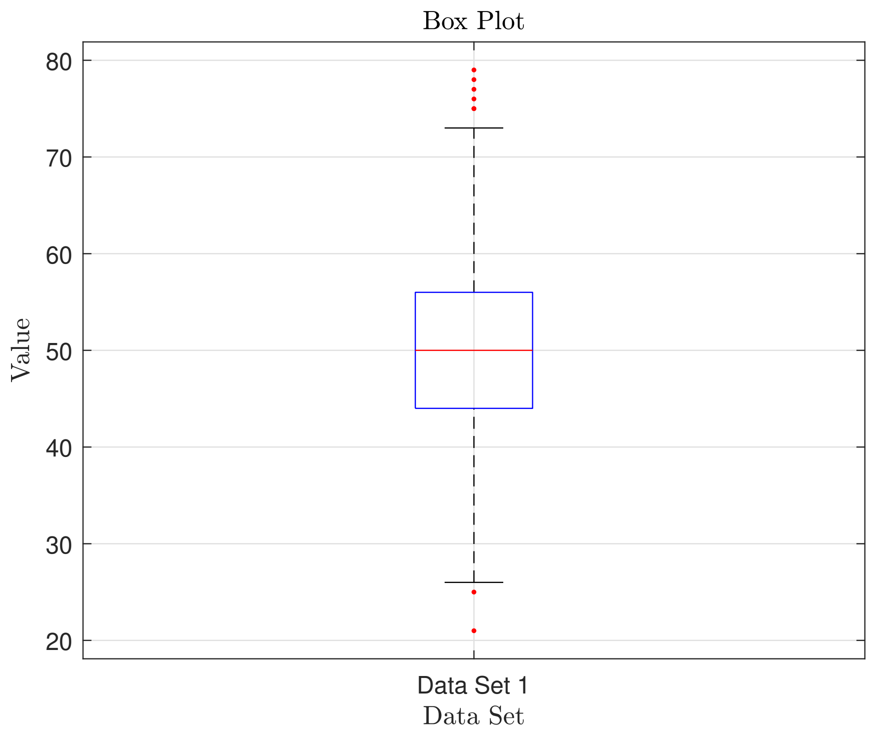


Figure D.1: Box plot generated from a random data set.

There are six key statistical properties that can be observed in Figure D.1, with respect to this data set:

(i) The blue box encloses the *interquartile range* of the data set.

(ii) The blue horizontal line at the bottom of the blue box represents the 25^{th} percentile, i.e., the lower quartile.

(iii) The blue horizontal line at the top of the blue box represents the 75^{th} percentile, i.e., the upper quartile.

(iv) The red line within the blue box represents the *median* value of the data set.

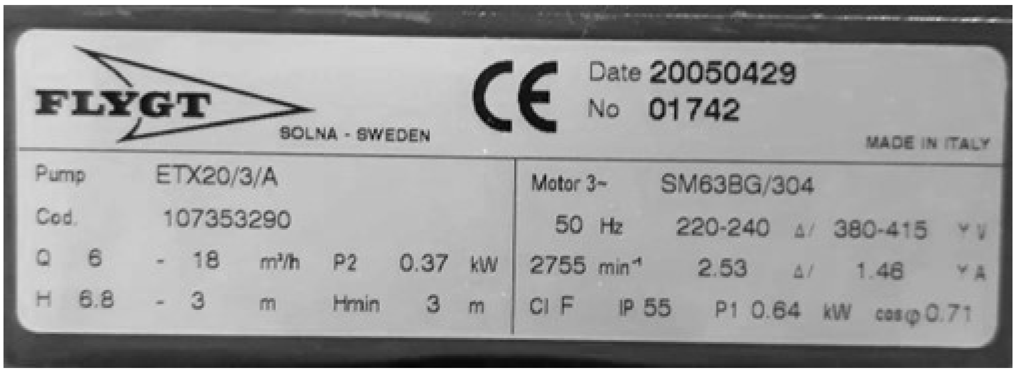
(v) The whiskers are the two lines that are connected to the blue box via dashed lines (above and below). The upper whisker is the largest nonoutlier data point, while the lower whisker is the smallest nonoutlier data point. The upper and lower whiskers are also referred to as the upper and lower *adjacent*, respectively.

(vi) The red dots are outliers. These are defined as data points which are $1.5 \cdot IQR$ away from the top or from the bottom of the box, respectively. IQR is defined as the interquartile range, which is simply the distance between the lower and upper quartiles.

For a more detailed discussion on box plots, see for instance [42], pp. 44-47.

Appendix E

Two-Tank Component Specifications


FLYGT SOLNA - SWEDEN

CE Date 20050429
No 01742

MADE IN ITALY

Pump	ETX20/3/A					Motor 3~	SM63BG/304		
Cod.	107353290					50 Hz	220-240 Δ/	380-415	Y V
Q 6	- 18	m³/h	P2	0.37	kW	2755 min⁻¹	2.53 Δ/	1.46	Y A
H 6.8	- 3	m	Hmin	3	m	Cl F	IP 55	P1 0.64 kW	cosφ 0.71

Figure E.1: Pump specifications.

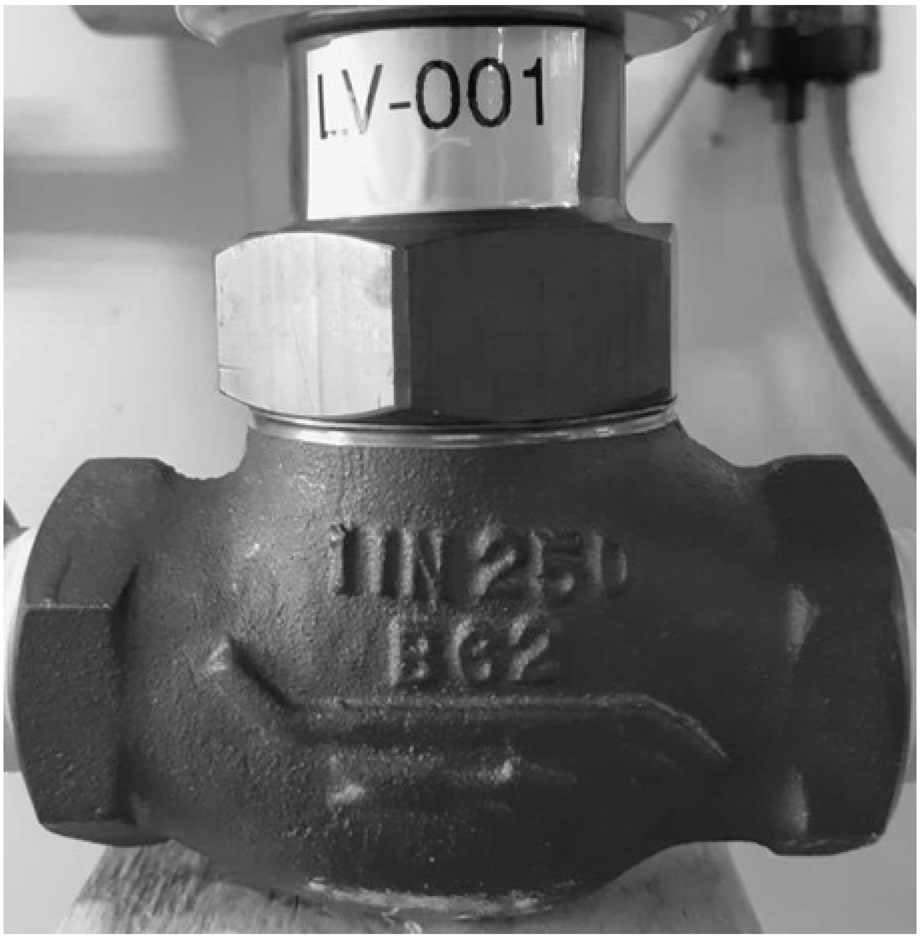
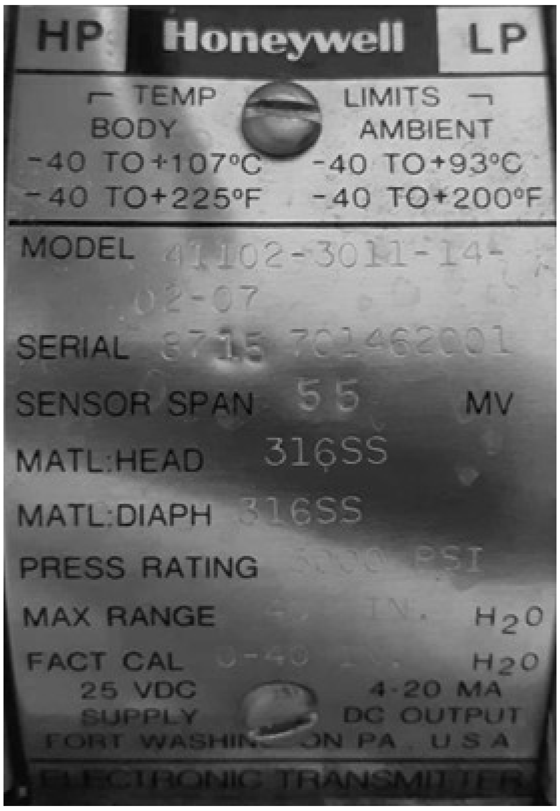


Figure E.2: Valve specifications.



HP **Honeywell** LP

TEMP LIMITS	
BODY	AMBIENT
-40 TO+107°C	-40 TO+93°C
-40 TO+225°F	-40 TO+200°F

MODEL	41102-3011-14-02-07	
SERIAL	8715 701462001	
SENSOR SPAN	55	MV
MATL:HEAD	316SS	
MATL:DIAPH	316SS	
PRESS RATING	3000 PSI	
MAX RANGE	40 IN.	H_2O
FACT CAL	0-40 IN.	H_2O

25 VDC SUPPLY 4-20 MA DC OUTPUT

FORT WASHINGTON PA U S A

ELECTRONIC TRANSMITTER

Figure E.3: Level transmitter specifications.

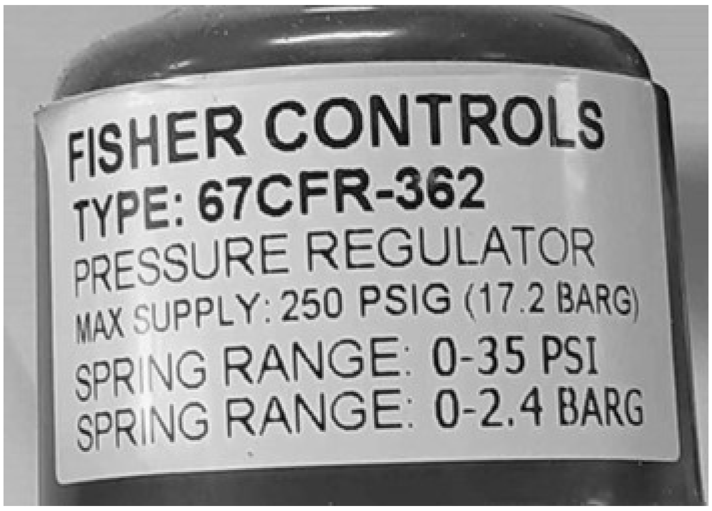

FISHER CONTROLS
TYPE: 67CFR-362
PRESSURE REGULATOR
MAX SUPPLY: 250 PSIG (17.2 BARG)
SPRING RANGE: 0-35 PSI
SPRING RANGE: 0-2.4 BARG

Figure E.4: Positioner specifications.

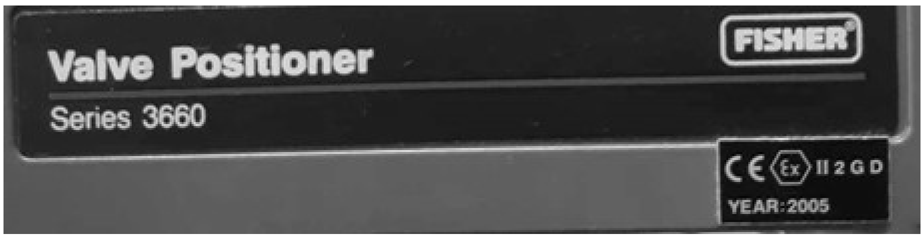


Figure E.5: Positioner model.

Appendix F

Project Description

Masteroppgave – Model predictive control of the two-tank system

Model predictive control (MPC) is a control paradigm that relies on the model of the plant to be controlled in order to predict the effect that a certain input value will have on the plant's state. The prediction is included into an online optimization problem that is solved online to compute the optimal control sequence to be applied. Contrarily to other control strategies, such as the LQR or the PID, MPC-based control systems can anticipate future events. Also, this method is inherently capable of handling highly constrained nonlinear systems. For this reason, MPC has been applied successfully in many industrial settings, such as for example chemical plants and oil refineries, just to name a few.

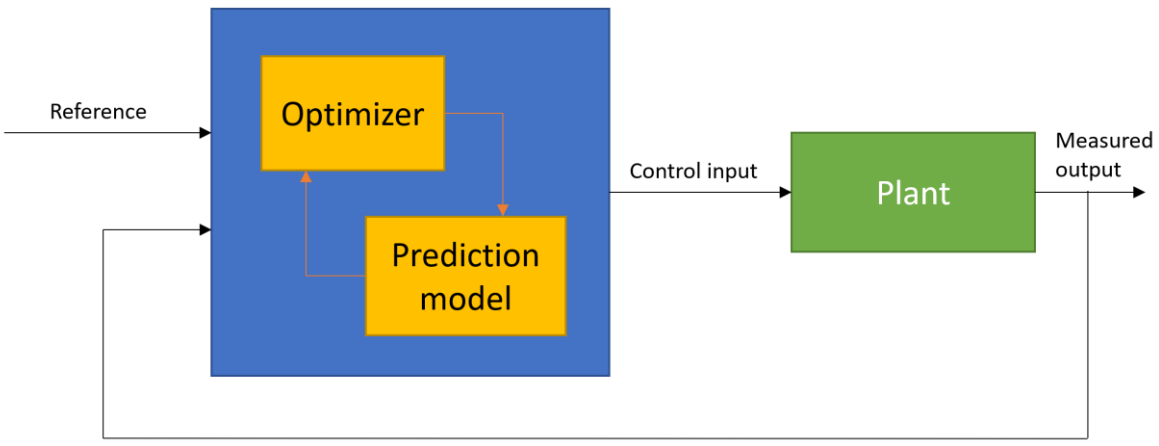


Figure 1 – Model predictive control conceptual scheme

This project aims at designing and implementing an MPC-based control system to control the two-tank system available in the laboratory at KE E-458 and compare its performance against other more conventional control strategies. At first, a simple linear MPC strategy will be used. In subsequent phases of the projects, other types of MPC strategies will be applied, such as nonlinear MPC or robust MPC. It is expected that the advantages/disadvantages of applying this technique are evaluated both in a simulation environment and with experimental data obtained from the two-tank system.

ACTIVITIES AND OBJECTIVES

- Literature study and analysis of the state-of-the-art of MPC
- Implementation and evaluation of linear MPC in simulation environment
- Implementation and evaluation of advanced MPC in simulation environment
- Experimental validation of the designed MPC-based control systems on the two-tank system available in the room KE E-458
- Comparison of MPC against PID and LQR control strategies

STUDENT PROFILE

- Strong skills and interest in control systems (the student should have passed both ELE600 and ELE620)
- Proficiency in MATLAB and Simulink

RELEVANT MATERIAL

https://www.youtube.com/watch?v=YwodGM2eoy4 (Overview of MPC by Steve Brunton)

Rawlings, James B. "Tutorial overview of model predictive control." IEEE control systems magazine 20.3 (2000): 38-52.

Ellis, Matthew, Helen Durand, and Panagiotis D. Christofides. "A tutorial review of economic model predictive control methods." Journal of Process Control 24.8 (2014): 1156-1178.

Appendix G

Project Plan

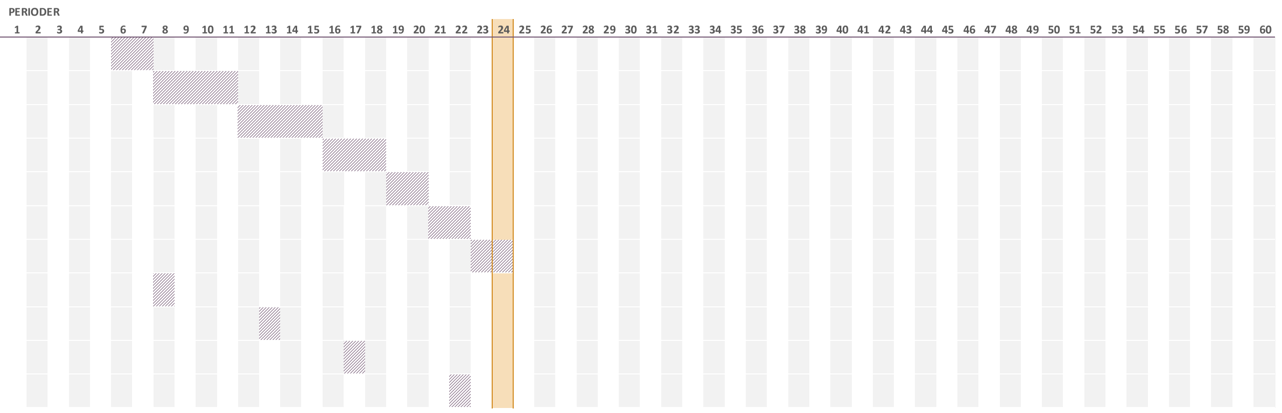
MSc Thesis - Gent Luta

PERIODER = Week Number

Periodeutheving: 24

▨ Planlagt varighet ▨ Faktisk start ■ % fullført ▨ Faktisk (utover planen) ■ % fullført (utover planen)

AKTIVITET	PLANLAGT START	PLANLAGT VARIGHET	FAKTISK START	FAKTISK VARIGHET	PROSENT FULLFØRT
Chapter 1: Introduction	6	2			
Chapter 2: Related Work (Theory)	8	4			
Chapter 3: Approach (Toy examples / Simulations)	12	4			
Chapter 4: Experimental Evaluation	16	3			
Chapter 4.5: Collaboration chapter (MHE)	19	2			
Chapter 5: Conclusions / Discussion	21	2			
Report (Abstract / Acknowledgements / Proof reading)	23	2			
First Draft Report	8	1			
Second Draft Report	13	1			
Third Draft Raport	17	1			
Fourth Draft Report (Masteroppgaver)	22	1			



Appendix H

Master Theses Poster Presentation

Design and Implementation of Model Predictive Control for a Coupled Tank System

Gent Luta

Introduction:

Increased computational power, faster sensors and communications, and bigger storage capacity were the key motivators for the development of Model Predictive Control (MPC). Today, MPC is a wide-spread process control technique within areas such as refining, petrochemicals, and chemicals.

MPC can handle highly nonlinear, multivariate systems that are subjected to multiple constraints. Additionally, MPC has the ability to anticipate future events due to its predictive nature.

Objectives:

This project aims at designing and implementing MPC-based control systems for the two-tank system, such as: Linear MPC, and Explicit MPC

These MPC-based controllers are to be designed, implemented and evaluated on the Two-tank system, as well as in simulation environments.

Conclusion:

Pending. (Extended submission deadline.)

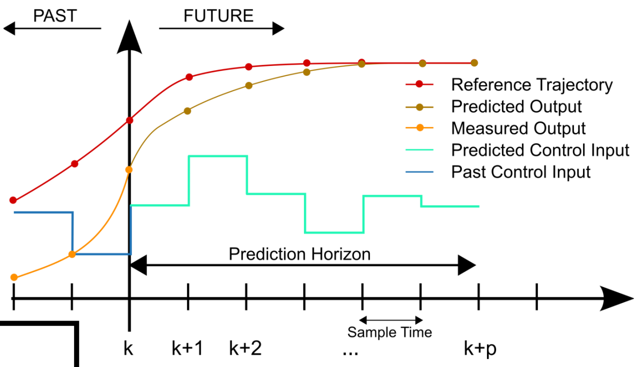
Department of Electrical Engineering
and Computer Science

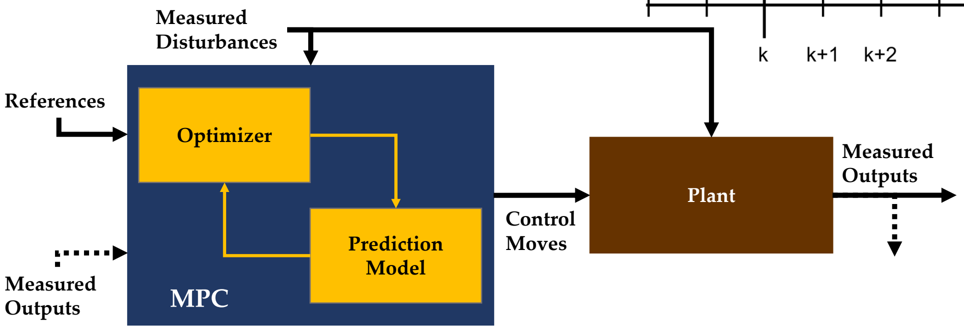
Master's Thesis Poster

Universitetet
i Stavanger

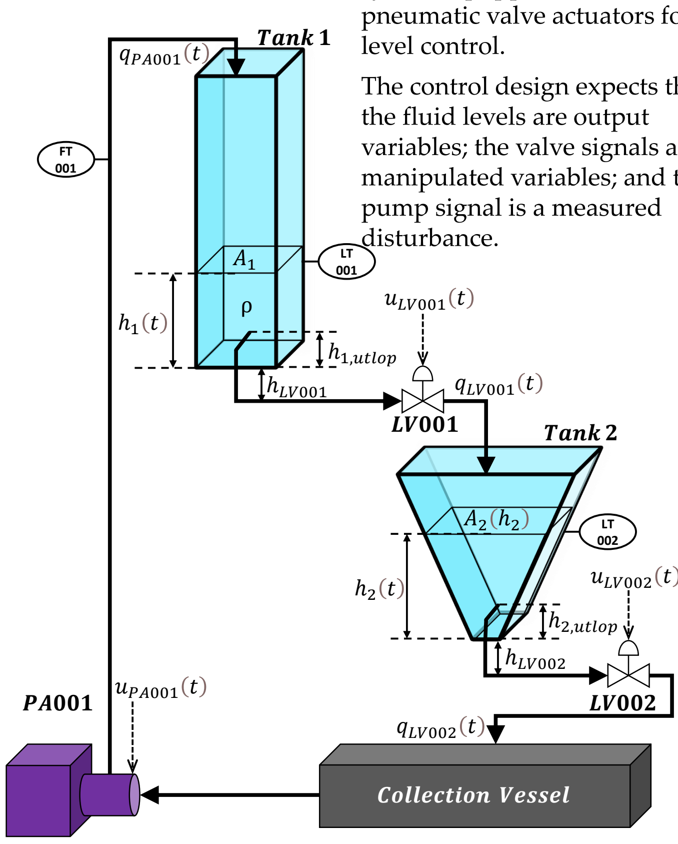
'The essence of MPC is to optimize, over the manipulated inputs, forecasts of process behavior.' - J.B. Rawlings

An MPC consists of a prediction model and an optimizer. The prediction model describes the process dynamics, while the optimizer uses algorithms to minimize / maximize an objective function.





Physical System



Description:

Coupled, pump fed, two-tank system equipped with pneumatic valve actuators for level control.

The control design expects that the fluid levels are output variables; the valve signals are manipulated variables; and the pump signal is a measured disturbance.

Modeling:

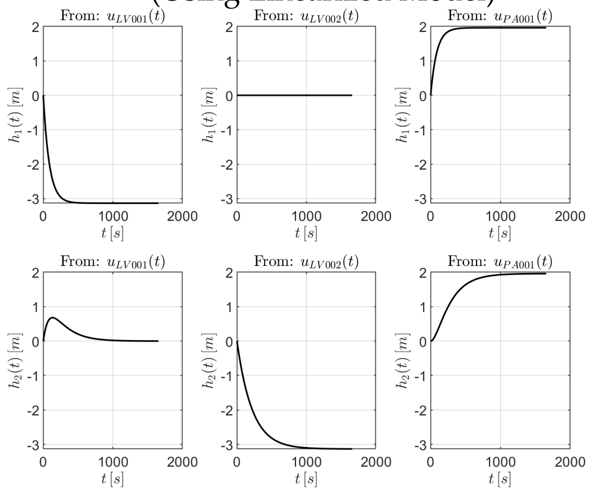
Nonlinear mathematical modeling (i.e., describing the system dynamics) is achieved using conservation / balance laws.

Linear models are obtained by linearization of the nonlinear model.

Step Response:

Step response of the linear model gives us important information for future MPC design such as time constants, which is used when determining sampling time and prediction horizon.

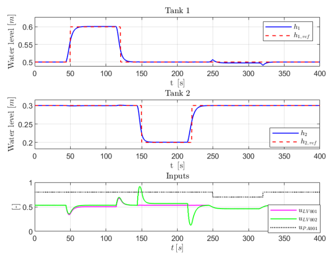
Step Response
(Using Linearized Model)



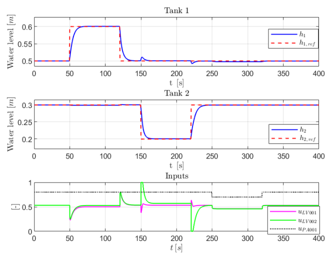
Results:

400 seconds are simulated using a sampling time of 0.5 seconds. I.e., each simulation is 800 time steps. This is done 10 times to calculate the weighted averages and standard deviations of both execution time and total cost of the objective function.
μ : Weighted Average
σ : Standard Deviation

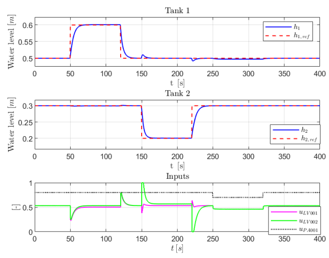
Linear MPC w/ previewing:
Execution Time: μ = 0.985ms , σ = 0.1105ms
Total Objective Function Cost: μ = 462,387 , σ = 0.017e − 3



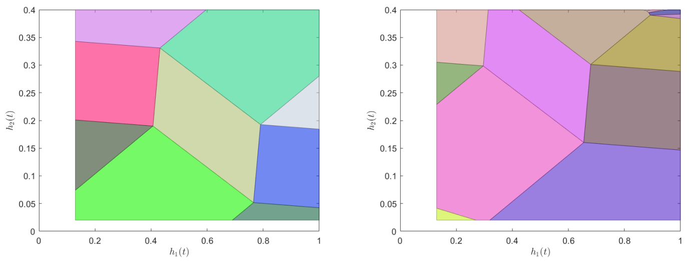
Explicit MPC :
Execution Time: μ = 0.507ms , σ = 7.52e − 5s
Total Objective Function Cost: μ = 466,12 , σ = 2.22e − 5



Linear MPC w/o previewing:
Execution Time: μ = 0.978ms , σ = 0.1012ms
Total Objective Function Cost: μ = 465,695 , σ = 0



Examples of Explicit MPC polyhedral partitioning



Appendix I

MATLAB Code

I.1 Functions

I.1.1 AdaptiveSys.m

```matlab
function sys = AdaptiveSys(x,u, ts)
%% Linearization point
h1_A = x(1);
h2_A = x(2);
u_PA001_A = u(3);
u1_A = u(1);
u2_A = u(2);
f1_A = ValveChar(u1_A);
f2_A = ValveChar(u2_A);

%% System parameters
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

%% Function Handles
f1_handle = @ValveChar;
f2_handle = @ValveChar;
f3_handle = @PumpChar;

delta = 0.01;   %Step size (numerical difference)

%% Linearization
```

```
% A - Matrix
a11 = - (sqrt(100000))/(7.2*10^8) * ((Kv1*f1_A*rho*g) / (A1*sqrt(rho*g*(
    h1_A + h_LV001))));
a12 = 0;
a21 = ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*10^8)) * ((Kv1*
    f1_A*rho*g) / (sqrt(rho*g*(h1_A + h_LV001))));
a22 = - ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*(10^8))) * ((Kv2
    *f2_A*rho*g) / (sqrt(rho*g*(h2_A + h_LV002))));
A = [a11 a12; a21 a22];

% B - Matrix
b11 = - ((Kv1)/(3600*A1)) * sqrt((rho*g*(h1_A + h_LV001))/100000) *
    forward_diff(f1_handle, u1_A, delta);
b12 = 0;
b21 = ((1)/(0.004 + 0.07*h2_A)) * (Kv1/3600) * sqrt((rho*g*(h1_A +
    h_LV001))/100000)*forward_diff(f1_handle, u1_A, delta);
b22 = -((1)/(0.004 + 0.07*h2_A)) * (Kv2/3600) * sqrt((rho*g*(h2_A +
    h_LV002))/100000)*forward_diff(f2_handle, u2_A, delta);
B = [b11 b12 ; b21 b22];

% C - Matrix
C = eye(2);

% D - Matrix
%D = zeros(2);

% G - Matrix (Disturbance)
g11 = forward_diff(f3_handle, u_PA001_A, delta)/A1;
g21 = 0;
G = [g11; g21];

% B_a - Matrix (Augmented)
B_a = [B G];

% D_a - Matrix (Augmented)
D_a = zeros(2, 3);

%% State-Space-Model
sys = ss(A, B_a, C, D_a);   % CT State-Space Model
sys = c2d(sys,ts);          % DT State-Space Model (ts sampling)
% Signal Names
sys.InputName = {'u1', 'u2', 'u_PA001'};
sys.OutputName = {'h1', 'h2'};
sys.StateName = {'h1', 'h2'};
% Signal Units
sys.InputUnit = {'-', '-', '-'};
sys.OutputUnit = {'m', 'm'};
sys.StateUnit = {'m', 'm'};
```

```
% Signal Types
sys = setmpcsignals(sys, 'MV', [1 2], 'MD', 3, 'MO', [1 2]);
```

I.1.2 central_diff.m

```
function derivative = central_diff(f, u, h)
    derivative = (f(u+h) - f(u-h))/(2*h);
end
```

I.1.3 forward_diff.m

```
function derivative = forward_diff(f, u, h)
    derivative = (f(u+h) - f(u))/(h);
end
```

I.1.4 InversValveChar.m

```
function u = InversValveChar(f)
    u = exp(log(log(f*exp(1)-f+1))/1.2);
end
```

I.1.5 PumpChar.m

```
function f = PumpChar(u)
u_PA001 = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
            0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
            8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s
f = interp1(u_PA001,q_PA001,u);
end
```

I.1.6 tankCT_NEW.m

```
function dxdt = tankCT_NEW(x,u)
    %% parameters
    A1 = 0.01;
    Kv1 = 11.25;
    Kv2 = 11.25;
    rho = 1000;
    g = 9.81;
    hLV1 = 0.05;
    hLV2 = 0.25;
    %%
    h1 = x(1);
    h2 = x(2);
    u1 = u(1);
    u2 = u(2);
    uPA = u(3);
    f3 = PumpChar(uPA);
    %% Compute dxdt
    dxdt = [(1/A1)*(f3 - Kv1*ValveChar(u1)*sqrt((rho*g*(h1+hLV1))/100000)
/3600) ; ...
        (1/((0.004 + 0.07*h2)*3600))*(Kv1*ValveChar(u1)*sqrt((rho*g*(h1+
hLV1))/100000)...
        - Kv2*ValveChar(u2)*sqrt((rho*g*(h2+hLV2))/100000))];
```

I.1.7 tankDT_NEW_One_Step.m

```
function xk1 = tankDT_NEW_One_Step(xk, uk, Ts)
    delta = Ts;
    xk1 = xk + delta*tankCT_NEW(xk, uk);
```

I.1.8 tankDT.m

```
function xk1 = tankDT(xk, uk, Ts)
    M = 10;
    delta = Ts/M;
    xk1 = xk;
    for ct=1:M
        xk1 = xk1 + delta*tankCT(xk1, uk);
    end
```

I.1.9 tankOutputFcn.m

```
function y = tankOutputFcn(x, u,params)
    y = [x(1);x(2)];
```

I.1.10 Valve_1_OP_New.m

```
function u_O = Valve_1_OP_New(h1,u_PA001)
    %% Local variables
    f3 = PumpChar(u_PA001);
    Kv1 = 11.25;
    rho = 1000;
    g = 9.81;
    h_LV001 = 0.05;
    %%
    f1 = f3*3600*sqrt(100000/(rho*g*(h1+h_LV001)))/Kv1;
    u_O = InversValveChar(f1);
end
```

I.1.11 Valve_2_OP.m

```
function u_O = Valve_2_OP(u1,h1,h2)
    %% Local variables
    Kv1 = 11.25;
    Kv2 = 11.25;
    rho = 1000;
    g = 9.81;
    h_LV001 = 0.05;
    h_LV002 = 0.25;
    %%
    f2 = (Kv1*ValveChar(u1)*sqrt((rho*g*(h1 + h_LV001))/100000))/(Kv2*
   sqrt((rho*g*(h2+h_LV002))/100000));
    u_O = InversValveChar(f2);
end
```

I.1.12 ValveChar.m

```
function f = ValveChar(u)
    %u = abs(u);
    f = (exp(u^1.2) - 1)/(exp(1) - 1);
end
```

I.2 MPC Design Files

I.2.1 MPC_Adaptive_FINAL.m

```
%% Control of Two-Tank Using Adaptive MPC.
% Initial condition: {h1 = 0.5, h2 = 0.3, u_PA001 = 0.8}

% Author: Gent Luta

% Date: Spring 2023

%% System Parameters (For Simulink Model)
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

h1_max = 1;
h1_min = 0.13;
h2_max = 0.4;
h2_min = 0.02;
Kv_LV001    = 11.25;
Kv_LV002    = 11.25;
z_LV001 = 0:0.05:1;
f_LV001 = (exp(z_LV001.^1.2)-1)/(exp(1)-1);

z_LV002 = 0:0.05:1;
f_LV002 = f_LV001;
u_PA001_data = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
            0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
            8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s


%% Steady State Values
h1_A = 0.5;%0.5
h2_A = 0.3;%0.3
u_PA001_A = 0.8;%0.8

u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);

ts = 0.5;   % Sampling time

sys = AdaptiveSys([h1_A;h2_A], [u1_A;u2_A;u_PA001_A], ts);  % DT Linear
    System

%% Linear MPC
old_status = mpcverbosity('on');

```

```
p = 13;       % Prediction horizon
c = 13;         % Control horizon
%c = [1 4 8];       % Control horizon (blocking)



mpcobj = mpc(sys, ts, p, c);    % Linear MPC object

% Nominal Values
x0 = [h1_A; h2_A];
u0 = [u1_A;u2_A;u_PA001_A];
y0 = x0;
mpcobj.Model.Nominal = struct('X',x0,'U',u0,'Y',y0);

% Set Estimator (None)
setEstimator(mpcobj, 'custom');

% Signal Scaling
mpcobj.OV(1).ScaleFactor = 1 - 0.13;    % Range of h1
mpcobj.OV(2).ScaleFactor = 0.4 - 0.02;  % Range of h2

% Weighting Matrices
mpcobj.Weights.OutputVariables = [1 1];                 % Q
mpcobj.Weights.ManipulatedVariablesRate = [0.1 0.1];    % S
mpcobj.Weights.ManipulatedVariables = [0 0];            % R

% Output (soft) Constraints
% mpcobj.OV(1).Max = 1;
% mpcobj.OV(1).Min = 0.13;
% mpcobj.OV(2).Max = 0.4;
% mpcobj.OV(2).Min = 0.02;

% % Output ECR (slack)
% mpcobj.OV(1).MinECR = 5;
% mpcobj.OV(1).MaxECR = 5;
% mpcobj.OV(2).MinECR = 5;
% mpcobj.OV(2).MaxECR = 5;

% Input (hard) Constraints
mpcobj.MV(1).Max = 0.9999;
mpcobj.MV(1).Min = 0.0001;
mpcobj.MV(2).Max = 0.9999;
mpcobj.MV(2).Min = 0.0001;

% Input Rate (soft) Constraints
% mpcobj.MV(1).RateMin = -0.1;
% mpcobj.MV(1).RateMax = 0.1;
% mpcobj.MV(2).RateMin = -0.1;
% mpcobj.MV(2).RateMax = 0.1;

% %Input Rate ECR (slack)
% mpcobj.MV(1).RateMinECR = 20;
% mpcobj.MV(1).RateMaxECR = 20;
% mpcobj.MV(2).RateMinECR = 20;
% mpcobj.MV(2).RateMaxECR = 20;

% MPC Optimizer Options:
% Interior-Point Solver and Option
% mpcobj.Optimizer.Algorithm = 'interior-point';
% mpcobj.Optimizer.InteriorPointOptions.MaxIterations = 8;
% mpcobj.Optimizer.InteriorPointOptions.ConstraintTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.OptimalityTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.ComplementarityTolerance = 1e-6;
% mpcobj.Optimizer.InteriorPointOptions.StepTolerance = 1e-7;

% Active-Set Solver and Options
% mpcobj.Optimizer.Algorithm = 'active-set';
% mpcobj.Optimizer.ActiveSetOptions.MaxIterations = 2;
% mpcobj.Optimizer.ActiveSetOptions.ConstraintTolerance = 1e-6;

% mpcobj.Optimizer.UseSuboptimalSolution = true;

%% SIM
Duration = 400;         % Simulation time
t = 0:ts:Duration;
N = length(t);

% Signal Previewing
ref = [ ones(1, N-1)*h1_A;
        ones(1, N-1)*h2_A];
u_PA001 = ones(1,N)*u_PA001_A;

% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 150;
t_upa_rise = 250;
t_hold = 70;
idx_ref1 = round(t_h1_rise/ts);
idx_ref2 = round(t_h2_rise/ts);
idx_upa = round(t_upa_rise/ts);
idx_hold = round(t_hold/ts);
%
ref(1,idx_ref1:idx_ref1+idx_hold) = h1_A + 0.2;%0.2
ref(2,idx_ref2:idx_ref2+idx_hold) = h2_A - 0.2;%0.2
u_PA001(1, idx_upa:idx_upa+idx_hold) = u_PA001_A - 0.2;%0.2

```

```

%
x = x0;                          % Current state
xc = mpcstate(mpcobj);           % Controller state pointer
xN = x0;                         % Nominal (updated iteratively)
uN = u0;                         % Nominal (updated iteratively)
yN = y0;                         % Nominal (updated iteratively)
dxN = [0;0];                     % Nominal (updated iteratively)

% History Tracking
UU = zeros(2,N); UU(:,1) = u0(1:2);
MPCXX = zeros(2,N); MPCXX(:,1) = xc.Plant;
II = zeros(1,N-1);
XX_NL = zeros(2,N); XX_NL(:,1) = x0;
TIME_SPENT = zeros(1,N-1);


nsim = 1;    % Number of simulations

TIME_SPENT_AVERAGES = zeros(1,nsim);
COST = zeros(1,nsim);
for j = 1:nsim
hbar = waitbar(0, 'Simulation Progress');
for i = 1:(Duration/ts)
    if i <= N-p-1
        r1 = ref(1,i:i+p-1);
        r2 = ref(2,i:i+p-1);
        r = [r1' r2'];
        v = u_PA001(i:i+p-1)';
    else
        r1 = ref(1,i:end);
        r2 = ref(2,i:end);
        r = [r1' r2'];
        v = u_PA001(i:end)';
    end
    %r = ref(:,i)';     %Without signal preview (reference)
    %v = u_PA001(i);    %Without signal preview (disturbance)

    tic
    [mv, info] = mpcmoveAdaptive(mpcobj, xc, sys,...
        struct('X',xN,'U',uN,'Y',yN, 'DX', dxN), [], r, v); % Solve QP
    optimization problem
    TIME_SPENT(i) = toc;

    % Simulate Nonlinear System
    [~,XNL] = ode45(@(t,x)tankCT_NEW(x, [mv;v(1)]), [0 ts], XX_NL(:,i));
    xnl = [XNL(end,1);XNL(end,2)];

    % Update Controller State Based on Nonlinear System
    xc.Plant = xnl;
    xc.LastMove = mv;
    xc.Disturbance = [0;0];

    % Update Prediction Model and Nominal Solution
     sys = AdaptiveSys(xnl, [mv;v(1)], ts);
     xN = xnl;
     uN = [mv;v(1)];
     yN = xnl;
     dxN = xnl - XX_NL(:,i);

    % History tracking
    UU(:,i+1) = mv;
    MPCXX(:,i+1) = xc.Plant;
    II(i) = info.Iterations;
    XX_NL(:,i+1) = xnl;

    waitbar(i*ts/Duration, hbar);
end
close(hbar)

TIME_SPENT_AVERAGES(j) = mean(TIME_SPENT);

% Find Total Quadratic Cost
cost = 0;
Q = diag(mpcobj.Weights.OutputVariables);
Q(1,1) = Q(1,1)/mpcobj.OV(1).ScaleFactor;   % Scaling
Q(2,2) = Q(2,2)/mpcobj.OV(2).ScaleFactor;   % Scaling
S = diag(mpcobj.Weights.ManipulatedVariablesRate);
R = diag(mpcobj.Weights.ManipulatedVariables);
E = XX_NL(:,2:end) - ref;                   % OV reference tracking error
EU = UU(:,:) - u0(1:2,:);                   % MV target tracking error
mvRate = UU(:,1:end-1) - UU(:,2:end);       % MV rate
for l = 1:N-1
    cost = cost +  E(:,l)'*(Q^2)*E(:,l) + mvRate(:,l)'*(S^2)*mvRate(:,l)
   + EU(:,l)'*(R^2)*EU(:,l);
end
COST(j) = cost;
end
sprintf('Execution Time Weighted Average: %e',mean(TIME_SPENT_AVERAGES))
sprintf('Execution Time Standard Deviation: %e',std(TIME_SPENT_AVERAGES))

sprintf('Cost Weighted Average: %e',mean(COST))
sprintf('Cost Standard Deviation: %e',std(COST))


%% Plot - Simulation
```

```
figure

% Tank 1
subplot(3,1,1)
plot(t, XX_NL(1,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(1,:), 'red', 'LineWidth',1, 'LineStyle','--')
%hold on
%plot(t, MPCXX(1,:), 'cyan', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
legend('$h_{1}$','$h_{1,\,ref}$', 'Interpreter','latex')
ylim([0 1]);
grid on
box on
set(gca,'YTick',0:0.05:1)

% Tank 2
subplot(3,1,2)
plot(t, XX_NL(2,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(2,:), 'red', 'LineWidth',1, 'LineStyle','--')
%hold on
%plot(t, MPCXX(2,:), 'cyan', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
legend('$h_{2}$','$h_{2,\,ref}$', 'Interpreter','latex')
ylim([0 1])
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1)

% Input Signals
subplot(3,1,3)
stairs(t, UU(1,:), 'magenta', 'LineWidth',1)
hold on
stairs(t, UU(2,:), 'green', 'LineWidth',1)
hold on
stairs(t, u_PA001, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('[-]', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    )
ylim([0 1])
```

```
grid on
box on
```

I.2.2 MPC_Explicit_FINAL.m

```
%% Control of Two-Tank Using Adaptive MPC.
% Initial condition: {h1 = 0.5, h2 = 0.3, u_PA001 = 0.8}

% Author: Gent Luta

% Date: Spring 2023

%% System Parameters (For Simulink Model)
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

h1_max = 1;
h1_min = 0.13;
h2_max = 0.4;
h2_min = 0.02;
Kv_LV001    = 11.25;
Kv_LV002    = 11.25;
z_LV001 = 0:0.05:1;
f_LV001 = (exp(z_LV001.^1.2)-1)/(exp(1)-1);

z_LV002 = 0:0.05:1;
f_LV002 = f_LV001;
u_PA001_data = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
              0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
              8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s


%% Steady State Values
h1_A = 0.5;%0.5
h2_A = 0.3;%0.3
u_PA001_A = 0.8;%0.8

u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);
```

```matlab

ts = 0.5;   % Sampling time

sys = AdaptiveSys([h1_A;h2_A], [u1_A;u2_A;u_PA001_A], ts);  % DT Linear
    System

%% Linear MPC
old_status = mpcverbosity('on');

p = 13;       % Prediction horizon
c = 13;        % Control horizon
%c = [1 4 8];      % Control horizon (blocking)



mpcobj = mpc(sys, ts, p, c);    % Linear MPC object

% Nominal Values
x0 = [h1_A; h2_A];
u0 = [u1_A;u2_A;u_PA001_A];
y0 = x0;
mpcobj.Model.Nominal = struct('X',x0,'U',u0,'Y',y0);

% Set Estimator (None)
setEstimator(mpcobj, 'custom');

% Signal Scaling
mpcobj.OV(1).ScaleFactor = 1 - 0.13;    % Range of h1
mpcobj.OV(2).ScaleFactor = 0.4 - 0.02;  % Range of h2

% Weighting Matrices
mpcobj.Weights.OutputVariables = [1 1];                  % Q
mpcobj.Weights.ManipulatedVariablesRate = [0.1 0.1];    % S
mpcobj.Weights.ManipulatedVariables = [0 0];            % R

% Output (soft) Constraints
% mpcobj.OV(1).Max = 1;
% mpcobj.OV(1).Min = 0.13;
% mpcobj.OV(2).Max = 0.4;
% mpcobj.OV(2).Min = 0.02;

% % Output ECR (slack)
% mpcobj.OV(1).MinECR = 5;
% mpcobj.OV(1).MaxECR = 5;
% mpcobj.OV(2).MinECR = 5;
% mpcobj.OV(2).MaxECR = 5;

% Input (hard) Constraints
```

```
mpcobj.MV(1).Max = 0.9999;
mpcobj.MV(1).Min = 0.0001;
mpcobj.MV(2).Max = 0.9999;
mpcobj.MV(2).Min = 0.0001;

% Input Rate (soft) Constraints
% mpcobj.MV(1).RateMin = -0.1;
% mpcobj.MV(1).RateMax = 0.1;
% mpcobj.MV(2).RateMin = -0.1;
% mpcobj.MV(2).RateMax = 0.1;

% %Input Rate ECR (slack)
% mpcobj.MV(1).RateMinECR = 20;
% mpcobj.MV(1).RateMaxECR = 20;
% mpcobj.MV(2).RateMinECR = 20;
% mpcobj.MV(2).RateMaxECR = 20;

% MPC Optimizer Options:
% Interior-Point Solver and Option
% mpcobj.Optimizer.Algorithm = 'interior-point';
% mpcobj.Optimizer.InteriorPointOptions.MaxIterations = 8;
% mpcobj.Optimizer.InteriorPointOptions.ConstraintTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.OptimalityTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.ComplementarityTolerance = 1e-6;
% mpcobj.Optimizer.InteriorPointOptions.StepTolerance = 1e-7;

% Active-Set Solver and Options
% mpcobj.Optimizer.Algorithm = 'active-set';
% mpcobj.Optimizer.ActiveSetOptions.MaxIterations = 2;
% mpcobj.Optimizer.ActiveSetOptions.ConstraintTolerance = 1e-6;

% mpcobj.Optimizer.UseSuboptimalSolution = true;

%% SIM
Duration = 400;         % Simulation time
t = 0:ts:Duration;
N = length(t);

% Signal Previewing
ref = [ ones(1, N-1)*h1_A;
        ones(1, N-1)*h2_A];
u_PA001 = ones(1,N)*u_PA001_A;

% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 150;
t_upa_rise = 250;
t_hold = 70;
```

```matlab
idx_ref1 = round(t_h1_rise/ts);
idx_ref2 = round(t_h2_rise/ts);
idx_upa = round(t_upa_rise/ts);
idx_hold = round(t_hold/ts);
%
ref(1,idx_ref1:idx_ref1+idx_hold) = h1_A + 0.2;%0.2
ref(2,idx_ref2:idx_ref2+idx_hold) = h2_A - 0.2;%0.2
u_PA001(1, idx_upa:idx_upa+idx_hold) = u_PA001_A - 0.2;%0.2


%
x = x0;                          % Current state
xc = mpcstate(mpcobj);           % Controller state pointer
xN = x0;                         % Nominal (updated iteratively)
uN = u0;                         % Nominal (updated iteratively)
yN = y0;                         % Nominal (updated iteratively)
dxN = [0;0];                     % Nominal (updated iteratively)

% History Tracking
UU = zeros(2,N); UU(:,1) = u0(1:2);
MPCXX = zeros(2,N); MPCXX(:,1) = xc.Plant;
II = zeros(1,N-1);
XX_NL = zeros(2,N); XX_NL(:,1) = x0;
TIME_SPENT = zeros(1,N-1);


nsim = 1;   % Number of simulations

TIME_SPENT_AVERAGES = zeros(1,nsim);
COST = zeros(1,nsim);
for j = 1:nsim
hbar = waitbar(0, 'Simulation Progress');
for i = 1:(Duration/ts)
    if i <= N-p-1
        r1 = ref(1,i:i+p-1);
        r2 = ref(2,i:i+p-1);
        r = [r1' r2'];
        v = u_PA001(i:i+p-1)';
    else
        r1 = ref(1,i:end);
        r2 = ref(2,i:end);
        r = [r1' r2'];
        v = u_PA001(i:end)';
    end
    %r = ref(:,i)';      %Without signal preview (reference)
    %v = u_PA001(i);     %Without signal preview (disturbance)

    tic
    [mv, info] = mpcmoveAdaptive(mpcobj, xc, sys,...
        struct('X',xN,'U',uN,'Y',yN, 'DX', dxN), [], r, v); % Solve QP
    optimization problem
    TIME_SPENT(i) = toc;

    % Simulate Nonlinear System
    [~,XNL] = ode45(@(t,x)tankCT_NEW(x, [mv;v(1)]), [0 ts], XX_NL(:,i));
    xnl = [XNL(end,1);XNL(end,2)];

    % Update Controller State Based on Nonlinear System
    xc.Plant = xnl;
    xc.LastMove = mv;
    xc.Disturbance = [0;0];

    % Update Prediction Model and Nominal Solution
     sys = AdaptiveSys(xnl, [mv;v(1)], ts);
     xN = xnl;
     uN = [mv;v(1)];
     yN = xnl;
     dxN = xnl - XX_NL(:,i);

    % History tracking
    UU(:,i+1) = mv;
    MPCXX(:,i+1) = xc.Plant;
    II(i) = info.Iterations;
    XX_NL(:,i+1) = xnl;

    waitbar(i*ts/Duration, hbar);
end
close(hbar)

TIME_SPENT_AVERAGES(j) = mean(TIME_SPENT);

% Find Total Quadratic Cost
cost = 0;
Q = diag(mpcobj.Weights.OutputVariables);
Q(1,1) = Q(1,1)/mpcobj.OV(1).ScaleFactor;   % Scaling
Q(2,2) = Q(2,2)/mpcobj.OV(2).ScaleFactor;   % Scaling
S = diag(mpcobj.Weights.ManipulatedVariablesRate);
R = diag(mpcobj.Weights.ManipulatedVariables);
E = XX_NL(:,2:end) - ref;                   % OV reference tracking error
EU = UU(:,:) - u0(1:2,:);                   % MV target tracking error
mvRate = UU(:,1:end-1) - UU(:,2:end);       % MV rate
for l = 1:N-1
    cost = cost +  E(:,l)'*(Q^2)*E(:,l) + mvRate(:,l)'*(S^2)*mvRate(:,l)
    + EU(:,l)'*(R^2)*EU(:,l);
end
COST(j) = cost;
```

```matlab
end
sprintf('Execution Time Weighted Average: %e',mean(TIME_SPENT_AVERAGES))
sprintf('Execution Time Standard Deviation: %e',std(TIME_SPENT_AVERAGES))

sprintf('Cost Weighted Average: %e',mean(COST))
sprintf('Cost Standard Deviation: %e',std(COST))


%% Plot - Simulation
figure

% Tank 1
subplot(3,1,1)
plot(t, XX_NL(1,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(1,:), 'red', 'LineWidth',1, 'LineStyle','--')
%hold on
%plot(t, MPCXX(1,:), 'cyan', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
legend('$h_{1}$','$h_{1,\,ref}$', 'Interpreter','latex')
ylim([0 1]);
grid on
box on
set(gca,'YTick',0:0.05:1)

% Tank 2
subplot(3,1,2)
plot(t, XX_NL(2,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(2,:), 'red', 'LineWidth',1, 'LineStyle','--')
%hold on
%plot(t, MPCXX(2,:), 'cyan', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
legend('$h_{2}$','$h_{2,\,ref}$', 'Interpreter','latex')
ylim([0 1])
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1)

% Input Signals
subplot(3,1,3)
stairs(t, UU(1,:), 'magenta', 'LineWidth',1)
hold on
```

```
stairs(t, UU(2,:), 'green', 'LineWidth',1)
hold on
stairs(t, u_PA001, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('[-]', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    )
ylim([0 1])
grid on
box on
```

I.2.3 MPC_Linear_FINAL.m

```
%% Control of Two-Tank Using Adaptive MPC.
% Initial condition: {h1 = 0.5, h2 = 0.3, u_PA001 = 0.8}

% Author: Gent Luta

% Date: Spring 2023

%% System Parameters (For Simulink Model)
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

h1_max = 1;
h1_min = 0.13;
h2_max = 0.4;
h2_min = 0.02;
Kv_LV001   = 11.25;
Kv_LV002   = 11.25;
z_LV001 = 0:0.05:1;
f_LV001 = (exp(z_LV001.^1.2)-1)/(exp(1)-1);

z_LV002 = 0:0.05:1;
f_LV002 = f_LV001;
u_PA001_data = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
               0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
               8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s
```

```


%% Steady State Values
h1_A = 0.5;%0.5
h2_A = 0.3;%0.3
u_PA001_A = 0.8;%0.8

u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);

ts = 0.5;   % Sampling time

sys = AdaptiveSys([h1_A;h2_A], [u1_A;u2_A;u_PA001_A], ts);  % DT Linear
    System

%% Linear MPC
old_status = mpcverbosity('on');

p = 13;       % Prediction horizon
c = 13;        % Control horizon
%c = [1 4 8];      % Control horizon (blocking)



mpcobj = mpc(sys, ts, p, c);    % Linear MPC object

% Nominal Values
x0 = [h1_A; h2_A];
u0 = [u1_A;u2_A;u_PA001_A];
y0 = x0;
mpcobj.Model.Nominal = struct('X',x0,'U',u0,'Y',y0);

% Set Estimator (None)
setEstimator(mpcobj, 'custom');

% Signal Scaling
mpcobj.OV(1).ScaleFactor = 1 - 0.13;    % Range of h1
mpcobj.OV(2).ScaleFactor = 0.4 - 0.02;  % Range of h2

% Weighting Matrices
mpcobj.Weights.OutputVariables = [1 1];                 % Q
mpcobj.Weights.ManipulatedVariablesRate = [0.1 0.1];    % S
mpcobj.Weights.ManipulatedVariables = [0 0];            % R

% Output (soft) Constraints
% mpcobj.OV(1).Max = 1;
% mpcobj.OV(1).Min = 0.13;
% mpcobj.OV(2).Max = 0.4;
% mpcobj.OV(2).Min = 0.02;

% % Output ECR (slack)
% mpcobj.OV(1).MinECR = 5;
% mpcobj.OV(1).MaxECR = 5;
% mpcobj.OV(2).MinECR = 5;
% mpcobj.OV(2).MaxECR = 5;

% Input (hard) Constraints
mpcobj.MV(1).Max = 0.9999;
mpcobj.MV(1).Min = 0.0001;
mpcobj.MV(2).Max = 0.9999;
mpcobj.MV(2).Min = 0.0001;

% Input Rate (soft) Constraints
% mpcobj.MV(1).RateMin = -0.1;
% mpcobj.MV(1).RateMax = 0.1;
% mpcobj.MV(2).RateMin = -0.1;
% mpcobj.MV(2).RateMax = 0.1;

% %Input Rate ECR (slack)
% mpcobj.MV(1).RateMinECR = 20;
% mpcobj.MV(1).RateMaxECR = 20;
% mpcobj.MV(2).RateMinECR = 20;
% mpcobj.MV(2).RateMaxECR = 20;

% MPC Optimizer Options:
% Interior-Point Solver and Option
% mpcobj.Optimizer.Algorithm = 'interior-point';
% mpcobj.Optimizer.InteriorPointOptions.MaxIterations = 8;
% mpcobj.Optimizer.InteriorPointOptions.ConstraintTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.OptimalityTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.ComplementarityTolerance = 1e-6;
% mpcobj.Optimizer.InteriorPointOptions.StepTolerance = 1e-7;

% Active-Set Solver and Options
% mpcobj.Optimizer.Algorithm = 'active-set';
% mpcobj.Optimizer.ActiveSetOptions.MaxIterations = 2;
% mpcobj.Optimizer.ActiveSetOptions.ConstraintTolerance = 1e-6;

% mpcobj.Optimizer.UseSuboptimalSolution = true;

%% SIM
Duration = 400;         % Simulation time
t = 0:ts:Duration;
N = length(t);

% Signal Previewing
```

```
ref = [ ones(1, N-1)*h1_A;
        ones(1, N-1)*h2_A];
u_PA001 = ones(1,N)*u_PA001_A;

% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 150;
t_upa_rise = 250;
t_hold = 70;
idx_ref1 = round(t_h1_rise/ts);
idx_ref2 = round(t_h2_rise/ts);
idx_upa = round(t_upa_rise/ts);
idx_hold = round(t_hold/ts);
%
ref(1,idx_ref1:idx_ref1+idx_hold) = h1_A + 0.2;%0.2
ref(2,idx_ref2:idx_ref2+idx_hold) = h2_A - 0.2;%0.2
u_PA001(1, idx_upa:idx_upa+idx_hold) = u_PA001_A - 0.2;%0.2


%
x = x0;                         % Current state
xc = mpcstate(mpcobj);          % Controller state pointer
xN = x0;                        % Nominal (updated iteratively)
uN = u0;                        % Nominal (updated iteratively)
yN = y0;                        % Nominal (updated iteratively)
dxN = [0;0];                    % Nominal (updated iteratively)

% History Tracking
UU = zeros(2,N); UU(:,1) = u0(1:2);
MPCXX = zeros(2,N); MPCXX(:,1) = xc.Plant;
II = zeros(1,N-1);
XX_NL = zeros(2,N); XX_NL(:,1) = x0;
TIME_SPENT = zeros(1,N-1);


nsim = 1;   % Number of simulations

TIME_SPENT_AVERAGES = zeros(1,nsim);
COST = zeros(1,nsim);
for j = 1:nsim
hbar = waitbar(0, 'Simulation Progress');
for i = 1:(Duration/ts)
    if i <= N-p-1
        r1 = ref(1,i:i+p-1);
        r2 = ref(2,i:i+p-1);
        r = [r1' r2'];
        v = u_PA001(i:i+p-1)';
    else
```

```
        r1 = ref(1,i:end);
        r2 = ref(2,i:end);
        r = [r1' r2'];
        v = u_PA001(i:end)';
    end
    %r = ref(:,i)';     %Without signal preview (reference)
    %v = u_PA001(i);    %Without signal preview (disturbance)

    tic
    [mv, info] = mpcmoveAdaptive(mpcobj, xc, sys,...
        struct('X',xN,'U',uN,'Y',yN, 'DX', dxN), [], r, v); % Solve QP
    optimization problem
    TIME_SPENT(i) = toc;

    % Simulate Nonlinear System
    [~,XNL] = ode45(@(t,x)tankCT_NEW(x, [mv;v(1)]), [0 ts], XX_NL(:,i));
    xnl = [XNL(end,1);XNL(end,2)];

    % Update Controller State Based on Nonlinear System
    xc.Plant = xnl;
    xc.LastMove = mv;
    xc.Disturbance = [0;0];

    % Update Prediction Model and Nominal Solution
     sys = AdaptiveSys(xnl, [mv;v(1)], ts);
     xN = xnl;
     uN = [mv;v(1)];
     yN = xnl;
     dxN = xnl - XX_NL(:,i);

    % History tracking
    UU(:,i+1) = mv;
    MPCXX(:,i+1) = xc.Plant;
    II(i) = info.Iterations;
    XX_NL(:,i+1) = xnl;

    waitbar(i*ts/Duration, hbar);
end
close(hbar)

TIME_SPENT_AVERAGES(j) = mean(TIME_SPENT);

% Find Total Quadratic Cost
cost = 0;
Q = diag(mpcobj.Weights.OutputVariables);
Q(1,1) = Q(1,1)/mpcobj.OV(1).ScaleFactor;   % Scaling
Q(2,2) = Q(2,2)/mpcobj.OV(2).ScaleFactor;   % Scaling
S = diag(mpcobj.Weights.ManipulatedVariablesRate);
```

```matlab
R = diag(mpcobj.Weights.ManipulatedVariables);
E = XX_NL(:,2:end) - ref;                    % OV reference tracking error
EU = UU(:,:) - u0(1:2,:);                    % MV target tracking error
mvRate = UU(:,1:end-1) - UU(:,2:end);        % MV rate
for l = 1:N-1
    cost = cost +  E(:,l)'*(Q^2)*E(:,l) + mvRate(:,l)'*(S^2)*mvRate(:,l)
    + EU(:,l)'*(R^2)*EU(:,l);
end
COST(j) = cost;
end
sprintf('Execution Time Weighted Average: %e',mean(TIME_SPENT_AVERAGES))
sprintf('Execution Time Standard Deviation: %e',std(TIME_SPENT_AVERAGES))

sprintf('Cost Weighted Average: %e',mean(COST))
sprintf('Cost Standard Deviation: %e',std(COST))


%% Plot - Simulation
figure

% Tank 1
subplot(3,1,1)
plot(t, XX_NL(1,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(1,:), 'red', 'LineWidth',1, 'LineStyle','--')
%hold on
%plot(t, MPCXX(1,:), 'cyan', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
legend('$h_{1}$','$h_{1,\,ref}$', 'Interpreter','latex')
ylim([0 1]);
grid on
box on
set(gca,'YTick',0:0.05:1)

% Tank 2
subplot(3,1,2)
plot(t, XX_NL(2,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(2,:), 'red', 'LineWidth',1, 'LineStyle','--')
%hold on
%plot(t, MPCXX(2,:), 'cyan', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
legend('$h_{2}$','$h_{2,\,ref}$', 'Interpreter','latex')
ylim([0 1])
```

```
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1)

% Input Signals
subplot(3,1,3)
stairs(t, UU(1,:), 'magenta', 'LineWidth',1)
hold on
stairs(t, UU(2,:), 'green', 'LineWidth',1)
hold on
stairs(t, u_PA001, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('[-]', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    )
ylim([0 1])
grid on
box on
```

I.2.4 MPC_Nonlinear_FINAL.m

```
%% Control of Two-Tank Using Adaptive MPC.
% Initial condition: {h1 = 0.5, h2 = 0.3, u_PA001 = 0.8}

% Author: Gent Luta

% Date: Spring 2023

%% System Parameters (For Simulink Model)
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

h1_max = 1;
h1_min = 0.13;
h2_max = 0.4;
h2_min = 0.02;
Kv_LV001   = 11.25;
Kv_LV002   = 11.25;
z_LV001 = 0:0.05:1;
```

```
f_LV001 = (exp(z_LV001.^1.2)-1)/(exp(1)-1);

z_LV002 = 0:0.05:1;
f_LV002 = f_LV001;
u_PA001_data = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
                0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
                8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s


%% Steady State Values
h1_A = 0.5;%0.5
h2_A = 0.3;%0.3
u_PA001_A = 0.8;%0.8

u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);

ts = 0.5;   % Sampling time

sys = AdaptiveSys([h1_A;h2_A], [u1_A;u2_A;u_PA001_A], ts);  % DT Linear
    System

%% Linear MPC
old_status = mpcverbosity('on');

p = 13;       % Prediction horizon
c = 13;        % Control horizon
%c = [1 4 8];      % Control horizon (blocking)



mpcobj = mpc(sys, ts, p, c);    % Linear MPC object

% Nominal Values
x0 = [h1_A; h2_A];
u0 = [u1_A;u2_A;u_PA001_A];
y0 = x0;
mpcobj.Model.Nominal = struct('X',x0,'U',u0,'Y',y0);

% Set Estimator (None)
setEstimator(mpcobj, 'custom');

% Signal Scaling
mpcobj.OV(1).ScaleFactor = 1 - 0.13;    % Range of h1
mpcobj.OV(2).ScaleFactor = 0.4 - 0.02;  % Range of h2

```

```
% Weighting Matrices
mpcobj.Weights.OutputVariables = [1 1];                  % Q
mpcobj.Weights.ManipulatedVariablesRate = [0.1 0.1];     % S
mpcobj.Weights.ManipulatedVariables = [0 0];             % R

% Output (soft) Constraints
% mpcobj.OV(1).Max = 1;
% mpcobj.OV(1).Min = 0.13;
% mpcobj.OV(2).Max = 0.4;
% mpcobj.OV(2).Min = 0.02;

% % Output ECR (slack)
% mpcobj.OV(1).MinECR = 5;
% mpcobj.OV(1).MaxECR = 5;
% mpcobj.OV(2).MinECR = 5;
% mpcobj.OV(2).MaxECR = 5;

% Input (hard) Constraints
mpcobj.MV(1).Max = 0.9999;
mpcobj.MV(1).Min = 0.0001;
mpcobj.MV(2).Max = 0.9999;
mpcobj.MV(2).Min = 0.0001;

% Input Rate (soft) Constraints
% mpcobj.MV(1).RateMin = -0.1;
% mpcobj.MV(1).RateMax = 0.1;
% mpcobj.MV(2).RateMin = -0.1;
% mpcobj.MV(2).RateMax = 0.1;

% %Input Rate ECR (slack)
% mpcobj.MV(1).RateMinECR = 20;
% mpcobj.MV(1).RateMaxECR = 20;
% mpcobj.MV(2).RateMinECR = 20;
% mpcobj.MV(2).RateMaxECR = 20;

% MPC Optimizer Options:
% Interior-Point Solver and Option
% mpcobj.Optimizer.Algorithm = 'interior-point';
% mpcobj.Optimizer.InteriorPointOptions.MaxIterations = 8;
% mpcobj.Optimizer.InteriorPointOptions.ConstraintTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.OptimalityTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.ComplementarityTolerance = 1e-6;
% mpcobj.Optimizer.InteriorPointOptions.StepTolerance = 1e-7;

% Active-Set Solver and Options
% mpcobj.Optimizer.Algorithm = 'active-set';
% mpcobj.Optimizer.ActiveSetOptions.MaxIterations = 2;
% mpcobj.Optimizer.ActiveSetOptions.ConstraintTolerance = 1e-6;

% mpcobj.Optimizer.UseSuboptimalSolution = true;

%% SIM
Duration = 400;         % Simulation time
t = 0:ts:Duration;
N = length(t);

% Signal Previewing
ref = [ ones(1, N-1)*h1_A;
        ones(1, N-1)*h2_A];
u_PA001 = ones(1,N)*u_PA001_A;

% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 150;
t_upa_rise = 250;
t_hold = 70;
idx_ref1 = round(t_h1_rise/ts);
idx_ref2 = round(t_h2_rise/ts);
idx_upa = round(t_upa_rise/ts);
idx_hold = round(t_hold/ts);
%
ref(1,idx_ref1:idx_ref1+idx_hold) = h1_A + 0.2;%0.2
ref(2,idx_ref2:idx_ref2+idx_hold) = h2_A - 0.2;%0.2
u_PA001(1, idx_upa:idx_upa+idx_hold) = u_PA001_A - 0.2;%0.2


%
x = x0;                         % Current state
xc = mpcstate(mpcobj);          % Controller state pointer
xN = x0;                        % Nominal (updated iteratively)
uN = u0;                        % Nominal (updated iteratively)
yN = y0;                        % Nominal (updated iteratively)
dxN = [0;0];                    % Nominal (updated iteratively)

% History Tracking
UU = zeros(2,N); UU(:,1) = u0(1:2);
MPCXX = zeros(2,N); MPCXX(:,1) = xc.Plant;
II = zeros(1,N-1);
XX_NL = zeros(2,N); XX_NL(:,1) = x0;
TIME_SPENT = zeros(1,N-1);


nsim = 1;   % Number of simulations

TIME_SPENT_AVERAGES = zeros(1,nsim);
COST = zeros(1,nsim);
```

```
for j = 1:nsim
hbar = waitbar(0, 'Simulation Progress');
for i = 1:(Duration/ts)
    if i <= N-p-1
        r1 = ref(1,i:i+p-1);
        r2 = ref(2,i:i+p-1);
        r = [r1' r2'];
        v = u_PA001(i:i+p-1)';
    else
        r1 = ref(1,i:end);
        r2 = ref(2,i:end);
        r = [r1' r2'];
        v = u_PA001(i:end)';
    end
    %r = ref(:,i)';      %Without signal preview (reference)
    %v = u_PA001(i);     %Without signal preview (disturbance)

    tic
    [mv, info] = mpcmoveAdaptive(mpcobj, xc, sys,...
        struct('X',xN,'U',uN,'Y',yN, 'DX', dxN), [], r, v); % Solve QP
    optimization problem
    TIME_SPENT(i) = toc;

    % Simulate Nonlinear System
    [~,XNL] = ode45(@(t,x)tankCT_NEW(x, [mv;v(1)]), [0 ts], XX_NL(:,i));
    xnl = [XNL(end,1);XNL(end,2)];

    % Update Controller State Based on Nonlinear System
    xc.Plant = xnl;
    xc.LastMove = mv;
    xc.Disturbance = [0;0];

    % Update Prediction Model and Nominal Solution
     sys = AdaptiveSys(xnl, [mv;v(1)], ts);
     xN = xnl;
     uN = [mv;v(1)];
     yN = xnl;
     dxN = xnl - XX_NL(:,i);

    % History tracking
    UU(:,i+1) = mv;
    MPCXX(:,i+1) = xc.Plant;
    II(i) = info.Iterations;
    XX_NL(:,i+1) = xnl;

    waitbar(i*ts/Duration, hbar);
end
close(hbar)
```

```matlab

TIME_SPENT_AVERAGES(j) = mean(TIME_SPENT);

% Find Total Quadratic Cost
cost = 0;
Q = diag(mpcobj.Weights.OutputVariables);
Q(1,1) = Q(1,1)/mpcobj.OV(1).ScaleFactor;   % Scaling
Q(2,2) = Q(2,2)/mpcobj.OV(2).ScaleFactor;   % Scaling
S = diag(mpcobj.Weights.ManipulatedVariablesRate);
R = diag(mpcobj.Weights.ManipulatedVariables);
E = XX_NL(:,2:end) - ref;                   % OV reference tracking error
EU = UU(:,:) - u0(1:2,:);                   % MV target tracking error
mvRate = UU(:,1:end-1) - UU(:,2:end);       % MV rate
for l = 1:N-1
    cost = cost +  E(:,l)'*(Q^2)*E(:,l) + mvRate(:,l)'*(S^2)*mvRate(:,l)
    + EU(:,l)'*(R^2)*EU(:,l);
end
COST(j) = cost;
end
sprintf('Execution Time Weighted Average: %e',mean(TIME_SPENT_AVERAGES))
sprintf('Execution Time Standard Deviation: %e',std(TIME_SPENT_AVERAGES))

sprintf('Cost Weighted Average: %e',mean(COST))
sprintf('Cost Standard Deviation: %e',std(COST))


%% Plot - Simulation
figure

% Tank 1
subplot(3,1,1)
plot(t, XX_NL(1,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(1,:), 'red', 'LineWidth',1, 'LineStyle','--')
%hold on
%plot(t, MPCXX(1,:), 'cyan', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
legend('$h_{1}$','$h_{1,\,ref}$', 'Interpreter','latex')
ylim([0 1]);
grid on
box on
set(gca,'YTick',0:0.05:1)

% Tank 2
subplot(3,1,2)
plot(t, XX_NL(2,:), 'blue', 'LineWidth',1, 'LineStyle','-')
```

```matlab
hold on
plot(t(2:end), ref(2,:), 'red', 'LineWidth',1, 'LineStyle','--')
%hold on
%plot(t, MPCXX(2,:), 'cyan', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
legend('$h_{2}$','$h_{2,\,ref}$', 'Interpreter','latex')
ylim([0 1])
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1)

% Input Signals
subplot(3,1,3)
stairs(t, UU(1,:), 'magenta', 'LineWidth',1)
hold on
stairs(t, UU(2,:), 'green', 'LineWidth',1)
hold on
stairs(t, u_PA001, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('[-]', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    )
ylim([0 1])
grid on
box on
```

I.2.5 Experimental_Make_LQR.m

```matlab
%% Steady State Values
h1_A = 0.5;
h2_A = 0.2;
u_PA001_A = 0.8;

u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);
f1_A = ValveChar(u1_A);
f2_A = ValveChar(u2_A);

%% System Parameters (For Simulink Model)
rho = 1000;
g = 9.81;
A1 = 0.01;
```

```
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

h1_max = 1;
h1_min = 0.13;
h2_max = 0.4;
h2_min = 0.02;
Kv_LV001   = 11.25;
Kv_LV002   = 11.25;
z_LV001 = 0:0.05:1;
f_LV001 = (exp(z_LV001.^1.2)-1)/(exp(1)-1);

z_LV002 = 0:0.05:1;
f_LV002 = f_LV001;
u_PA001_data = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
              0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
              8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s

%% Function Handles
f1_handle = @ValveChar;
f2_handle = @ValveChar;
f3_handle = @PumpChar;

delta = 0.01;   % Step size when using forward difference

%% Linearization
% A - Matrix
a11 = - (sqrt(100000))/(7.2*10^8) * ((Kv1*f1_A*rho*g) / (A1*sqrt(rho*g*(
    h1_A + h_LV001))));
a12 = 0;
a21 = ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*10^8)) * ((Kv1*
    f1_A*rho*g) / (sqrt(rho*g*(h1_A + h_LV001))));
a22 = - ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*(10^8))) * ((Kv2
    *f2_A*rho*g) / (sqrt(rho*g*(h2_A + h_LV002))));
A = [a11 a12; a21 a22];

% B - Matrix
b11 = - ((Kv1)/(3600*A1)) * sqrt((rho*g*(h1_A + h_LV001))/100000) *
    forward_diff(f1_handle, u1_A, delta);
b12 = 0;
b21 = ((1)/(0.004 + 0.07*h2_A)) * (Kv1/3600) * sqrt((rho*g*(h1_A +
    h_LV001))/100000)*forward_diff(f1_handle, u1_A, delta);
b22 = -((1)/(0.004 + 0.07*h2_A)) * (Kv2/3600) * sqrt((rho*g*(h2_A +
    h_LV002))/100000)*forward_diff(f2_handle, u2_A, delta);
```

```
B = [b11 b12 ; b21 b22];

% C - Matrix
C = eye(2);

% D - Matrix
D = zeros(2);

% G - Matrix (Disturbance)
g11 = forward_diff(f3_handle, u_PA001_A, delta)/A1;
g21 = 0;
G = [g11; g21];

% Augmented B - Matrix
B_a = [B G];

% Augmented D - Matrix
D_a = zeros(2, 3);

%% State-Space-Model
ts = 0.1;                     % Sampling time
sys = ss(A, B_a, C, D_a);    % CT state-space model
sys = c2d(sys,ts);            % DT state-space model (ts sampling)
% Signal Names
sys.InputName = {'u1', 'u2', 'u_PA001'};
sys.OutputName = {'h1', 'h2'};
sys.StateName = {'h1', 'h2'};
% Signal Units
sys.InputUnit = {'-', '-', '-'};
sys.OutputUnit = {'m', 'm'};
sys.StateUnit = {'m', 'm'};
% Signal Types
sys = setmpcsignals(sys, 'MV', [1 2], 'MD', 3, 'MO', [1 2]);
% Check Controllability
controllability_matrix = ctrb(sys);
controllability_matrix_rank = rank(controllability_matrix);

%% SIM
Duration = 400;           % Simulation time
t = 0:ts:Duration;
N = length(t);

% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 250;
t_upa_rise = 450;
t_hold = 100;
%% LQR
```

```
sys_lqr_ct = ss(A, B, C, D);
sys_lqr_dt = c2d(sys_lqr_ct, ts);


Q = [10 0; 0 100];
R = eye(2);
[K, S, P] = lqr(sys_lqr_dt, Q, R);
A = sys_lqr_dt.A;
B = sys_lqr_dt.B;
```

I.2.6 Experimental_Make_MPCs.m

```
%% Mat. File for Making the MPCs Used During Experimental Validation
% Linearized model about the nominal solution {h1 = 0.5, h2 = 0.3,
    u_PA001 = 0.8}

% Author: Gent Luta

% Date: Spring 2023

%% Steady State Values
h1_A = 0.5;
h2_A = 0.2;
u_PA001_A = 0.8;

u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);
f1_A = ValveChar(u1_A);
f2_A = ValveChar(u2_A);

%% System Parameters (For Simulink Model)
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

h1_max = 1;
h1_min = 0.13;
h2_max = 0.4;
h2_min = 0.02;
Kv_LV001    = 11.25;
Kv_LV002    = 11.25;
z_LV001 = 0:0.05:1;
```

```
f_LV001 = (exp(z_LV001.^1.2)-1)/(exp(1)-1);

z_LV002 = 0:0.05:1;
f_LV002 = f_LV001;
u_PA001_data = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
               0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
               8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s

%% Function Handles
f1_handle = @ValveChar;
f2_handle = @ValveChar;
f3_handle = @PumpChar;

delta = 0.01;   % Step size when using forward difference

%% Linearization
% A - Matrix
a11 = - (sqrt(100000))/(7.2*10^8) * ((Kv1*f1_A*rho*g) / (A1*sqrt(rho*g*(
    h1_A + h_LV001))));
a12 = 0;
a21 = ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*10^8)) * ((Kv1*
    f1_A*rho*g) / (sqrt(rho*g*(h1_A + h_LV001))));
a22 = - ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*(10^8))) * ((Kv2
    *f2_A*rho*g) / (sqrt(rho*g*(h2_A + h_LV002))));
A = [a11 a12; a21 a22];

% B - Matrix
b11 = - ((Kv1)/(3600*A1)) * sqrt((rho*g*(h1_A + h_LV001))/100000) *
    forward_diff(f1_handle, u1_A, delta);
b12 = 0;
b21 = ((1)/(0.004 + 0.07*h2_A)) * (Kv1/3600) * sqrt((rho*g*(h1_A +
    h_LV001))/100000)*forward_diff(f1_handle, u1_A, delta);
b22 = -((1)/(0.004 + 0.07*h2_A)) * (Kv2/3600) * sqrt((rho*g*(h2_A +
    h_LV002))/100000)*forward_diff(f2_handle, u2_A, delta);
B = [b11 b12 ; b21 b22];

% C - Matrix
C = eye(2);

% D - Matrix
D = zeros(2);

% G - Matrix (Disturbance)
g11 = forward_diff(f3_handle, u_PA001_A, delta)/A1;
g21 = 0;
G = [g11; g21];
```

```matlab

% Augmented B - Matrix
B_a = [B G];

% Augmented D - Matrix
D_a = zeros(2, 3);

%% State-Space-Model
ts = 0.1;                      % Sampling time
sys = ss(A, B_a, C, D_a);      % CT state-space model
sys = c2d(sys,ts);             % DT state-space model (ts sampling)
% Signal Names
sys.InputName = {'u1', 'u2', 'u_PA001'};
sys.OutputName = {'h1', 'h2'};
sys.StateName = {'h1', 'h2'};
% Signal Units
sys.InputUnit = {'-', '-', '-'};
sys.OutputUnit = {'m', 'm'};
sys.StateUnit = {'m', 'm'};
% Signal Types
sys = setmpcsignals(sys, 'MV', [1 2], 'MD', 3, 'MO', [1 2]);
% Check Controllability
controllability_matrix = ctrb(sys);
controllability_matrix_rank = rank(controllability_matrix);

%% SIM
Duration = 400;          % Simulation time
t = 0:ts:Duration;
N = length(t);

% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 250;
t_upa_rise = 450;
t_hold = 100;

%% Linear MPC
old_status = mpcverbosity('on');

p = 13;          % Prediction horizon (var 13)
c = 2;          % Control horizon
m = c;
mpcobj = mpc(sys, ts, p, c);    % Linear MPC object

% Nominal Values
x0 = [h1_A; h2_A];
u0 = [u1_A;u2_A;u_PA001_A];
y0 = x0;
```

```
mpcobj.Model.Nominal = struct('X',x0,'U',u0,'Y',y0);

% Set Estimator (None)
setEstimator(mpcobj, 'custom');

% Signal Scaling
mpcobj.OV(1).ScaleFactor = 1 - 0.13;    % Range of h1
mpcobj.OV(2).ScaleFactor = 0.4 - 0.02;  % Range of h2

q1 = 100;
q2 = 200;
s1 = 50;
s2 = 50;
r1 = 0;
r2 = 0;
% Weighting Matrices
mpcobj.Weights.OutputVariables = [q1 q2];                 % Q
mpcobj.Weights.ManipulatedVariablesRate = [s1 s2];  % S
mpcobj.Weights.ManipulatedVariables = [r1 r2];          % R

% % Output (soft) Constraints
% mpcobj.OV(1).Max = 1;
% mpcobj.OV(1).Min = 0.13;
% mpcobj.OV(2).Max = 0.4;
% mpcobj.OV(2).Min = 0.02;
% % Output ECR (slack)
% mpcobj.OV(1).MinECR = 5;
% mpcobj.OV(1).MaxECR = 5;
% mpcobj.OV(2).MinECR = 5;
% mpcobj.OV(2).MaxECR = 5;
% Input (hard) Constraints
mpcobj.MV(1).Max = 0.9999;
mpcobj.MV(1).Min = 0.0001;
mpcobj.MV(2).Max = 0.9999;
mpcobj.MV(2).Min = 0.0001;


% Active-Set Solver and Options
% mpcobj.Optimizer.Algorithm = 'active-set';
% mpcobj.Optimizer.ActiveSetOptions.MaxIterations = 2;
% mpcobj.Optimizer.ActiveSetOptions.ConstraintTolerance = 1e-6;

% mpcobj.Optimizer.UseSuboptimalSolution = true;


%% Output dist:
sys_out_dist = getoutdist(mpcobj);

```

```matlab
%% Explicit MPC
% Define the Range:

mpcobj.PredictionHorizon = 13;
mpcobj.ControlHorizon = 2;

range = generateExplicitRange(mpcobj);

% Range of the States
range.State.Min(:) = [0.13 0.02 -2 -2];
range.State.Max(:) = [1    0.4   2  2];

% Range of the References
range.Reference.Min(:) = [0.13 0.02];
range.Reference.Max(:) = [1    0.4];

% Range of the Measured Disturbance
range.MeasuredDisturbance.Min = 0.00001;
range.MeasuredDisturbance.Max = 0.99999;

% Range of the Manipulated Variables
range.ManipulatedVariable.Min(:) = [0.00001 0.00001];
range.ManipulatedVariable.Max(:) = [0.99999 0.99999];

% Option for EMPC
opt = generateExplicitOptions(mpcobj);
opt.polyreduction = 1;

% Create EMPC Obj.
empcobj = generateExplicitMPC(mpcobj,range,opt);

mpcobj.PredictionHorizon = p;
mpcobj.ControlHorizon = c;

%% Nonlinear MPC
nx = 2;            % Number of states
ny = 2;            % Number of outputs
mvIndex = [1,2];   % Manipulated variable indices
mdIndex = 3;       % Measured disturbance indices
nlobj = nlmpc(nx,ny,'MV',mvIndex,'MD',mdIndex); % Nonlinear MPC object

nlobj.Ts = ts;  % Sampling time

% Horizon Lengths
%p = 13; % Prediction horizon
%m = [2];  % Control horizon
nlobj.PredictionHorizon = p;
nlobj.ControlHorizon = m;
```

```

% Nonlinear DT State Function
nlobj.Model.StateFcn = 'tankDT_NEW_One_Step';
nlobj.Model.IsContinuousTime = false;
nlobj.Model.NumberOfParameters = 1;

% DT Output Function
nlobj.Model.OutputFcn = 'tankOutputFcn';

% Output Jacobian (I.e., C - Matrix)
nlobj.Jacobian.OutputFcn = @(x,u,Ts) [1 0; 0 1];

% Weighting Matrices
nlobj.Weights.OutputVariables = [q1 q2];              % Q
nlobj.Weights.ManipulatedVariablesRate = [s1 s2]; % S
nlobj.Weights.ManipulatedVariables = [r1 r2];         % R

% Signal Scaling
nlobj.OV(1).ScaleFactor = 1 - 0.13;     % Range of h1
nlobj.OV(2).ScaleFactor = 0.4 - 0.02;  % Range of h2

% nlobj.OV(1).Max = 1;
% nlobj.OV(1).Min = 0.13;
% nlobj.OV(2).Max = 0.4;
% nlobj.OV(2).Min = 0.02;
% % Output ECR (slack)
% nlobj.OV(1).MinECR = 5;
% nlobj.OV(1).MaxECR = 5;
% nlobj.OV(2).MinECR = 5;
% nlobj.OV(2).MaxECR = 5;
% Input (hard) Constraints
nlobj.MV(1).Max = 0.9999;
nlobj.MV(1).Min = 0.0001;
nlobj.MV(2).Max = 0.9999;
nlobj.MV(2).Min = 0.0001;

nlobj.Optimization.SolverOptions.MaxIterations = 30;    % Limit SQP Solver to 30 Intervals
nlobj.Optimization.UseSuboptimalSolution = true;        % Toggle Sub-Optimal Solution

% Creat Bus Object for Simulink File
createParameterBus(nlobj,'totank_Live_Edited_060723_2020b/Nonlinear MPC Controller','myBusObject',{ts});

%nlobj.Optimization.RunAsLinearMPC = 'Adaptive';
```

I.2.7 Experimental_Make_PID.m

```
%% Steady State Values
h1_A = 0.5;
h2_A = 0.2;
u_PA001_A = 0.8;

u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);
f1_A = ValveChar(u1_A);
f2_A = ValveChar(u2_A);

%% System Parameters (For Simulink Model)
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

h1_max = 1;
h1_min = 0.13;
h2_max = 0.4;
h2_min = 0.02;
Kv_LV001   = 11.25;
Kv_LV002   = 11.25;
z_LV001 = 0:0.05:1;
f_LV001 = (exp(z_LV001.^1.2)-1)/(exp(1)-1);

z_LV002 = 0:0.05:1;
f_LV002 = f_LV001;
u_PA001_data = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
           0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
           8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s

%% Function Handles
f1_handle = @ValveChar;
f2_handle = @ValveChar;
f3_handle = @PumpChar;

delta = 0.01;   % Step size when using forward difference

%% Linearization
% A - Matrix
```

```
a11 = - (sqrt(100000))/(7.2*10^8) * ((Kv1*f1_A*rho*g) / (A1*sqrt(rho*g*(
    h1_A + h_LV001))));
a12 = 0;
a21 = ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*10^8)) * ((Kv1*
    f1_A*rho*g) / (sqrt(rho*g*(h1_A + h_LV001))));
a22 = - ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*(10^8))) * ((Kv2
    *f2_A*rho*g) / (sqrt(rho*g*(h2_A + h_LV002))));
A = [a11 a12; a21 a22];

% B - Matrix
b11 = - ((Kv1)/(3600*A1)) * sqrt((rho*g*(h1_A + h_LV001))/100000) *
    forward_diff(f1_handle, u1_A, delta);
b12 = 0;
b21 = ((1)/(0.004 + 0.07*h2_A)) * (Kv1/3600) * sqrt((rho*g*(h1_A +
    h_LV001))/100000)*forward_diff(f1_handle, u1_A, delta);
b22 = -((1)/(0.004 + 0.07*h2_A)) * (Kv2/3600) * sqrt((rho*g*(h2_A +
    h_LV002))/100000)*forward_diff(f2_handle, u2_A, delta);
B = [b11 b12 ; b21 b22];

% C - Matrix
C = eye(2);

% D - Matrix
D = zeros(2);

% G - Matrix (Disturbance)
g11 = forward_diff(f3_handle, u_PA001_A, delta)/A1;
g21 = 0;
G = [g11; g21];

% Augmented B - Matrix
B_a = [B G];

% Augmented D - Matrix
D_a = zeros(2, 3);

%% State-Space-Model
ts = 0.5;                   % Sampling time
sys = ss(A, B_a, C, D_a);   % CT state-space model
sys = c2d(sys,ts);          % DT state-space model (ts sampling)
% Signal Names
sys.InputName = {'u1', 'u2', 'u_PA001'};
sys.OutputName = {'h1', 'h2'};
sys.StateName = {'h1', 'h2'};
% Signal Units
sys.InputUnit = {'-', '-', '-'};
sys.OutputUnit = {'m', 'm'};
sys.StateUnit = {'m', 'm'};
```

```
% Signal Types
sys = setmpcsignals(sys, 'MV', [1 2], 'MD', 3, 'MO', [1 2]);
% Check Controllability
controllability_matrix = ctrb(sys);
controllability_matrix_rank = rank(controllability_matrix);

%% SIM
Duration = 400;          % Simulation time
t = 0:ts:Duration;
N = length(t);

% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 250;
t_upa_rise = 450;
t_hold = 100;


kp1 = -1.279;
ti1 = 43.47;
kp2 = -1.63;
ti2 = 64.1;
```

I.2.8 LMHE_KMPC_Simulink_init.m

```
close all; clear all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Master Thesis:
% Moving Horizon Estimation for the Two-tank System by Greta Bekeryte
% and
% Design and Implementation of Model Predictive Control for a Coupled
   Tank
% System by Gent Luta.
% May, 2023
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
run parameters
port_attribute = 2;
n = 2; % number of states? [3/2]
m = 1; % number of measurements available [2/1]
Ts = 1.5;
% Operating point for linearization
h1_0 = 0.3;
h2_0 = 0.2;
u_PA001_0 = 0.6;
%%%%%%%%%%%%%%%%%%%%%%%%%%%% MPC_init
% Steady State Values
```

```
h1_A = 0.3;
h2_A = 0.2;
u_PA001_A = 0.6;
u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);
f1_A = ValveChar(u1_A);
f2_A = ValveChar(u2_A);
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;
% Function Handles
f1_handle = @ValveChar;
f2_handle = @ValveChar;
f3_handle = @PumpChar;

delta = 0.01;   %Step size (numerical difference)
% Linearization
% A - Matrix
a11 = - (sqrt(100000))/(7.2*10^8) * ((Kv1*f1_A*rho*g) / (A1*sqrt(rho*g*(
    h1_A + h_LV001))));
a12 = 0;
a21 = ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*10^8)) * ((Kv1*
    f1_A*rho*g) / (sqrt(rho*g*(h1_A + h_LV001))));
a22 = - ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*(10^8))) * ((Kv2
    *f2_A*rho*g) / (sqrt(rho*g*(h2_A + h_LV002))));
A_mpc = [a11 a12; a21 a22];

% B - Matrix
b11 = - ((Kv1)/(3600*A1)) * sqrt((rho*g*(h1_A + h_LV001))/100000) *
    forward_diff(f1_handle, u1_A, delta);
b12 = 0;
b21 = ((1)/(0.004 + 0.07*h2_A)) * (Kv1/3600) * sqrt((rho*g*(h1_A +
    h_LV001))/100000)*forward_diff(f1_handle, u1_A, delta);
b22 = -((1)/(0.004 + 0.07*h2_A)) * (Kv2/3600) * sqrt((rho*g*(h2_A +
    h_LV002))/100000)*forward_diff(f2_handle, u2_A, delta);
B_mpc = [b11 b12 ; b21 b22];

% C - Matrix
C_mpc = eye(2);

% D - Matrix
D_mpc = zeros(2);

% G - Matrix (Disturbance)
```

```matlab
g11 = forward_diff(f3_handle, u_PA001_A, delta)/A1;
g21 = 0;
G_mpc = [g11; g21];

% B_a - Matrix (Augmented)
B_a = [B_mpc G_mpc];

% D_a - Matrix (Augmented)
D_a = zeros(2, 3);

% State-Space-Model
%ts = 0.5;                    % Sampling time
sys = ss(A_mpc, B_a, C_mpc, D_a);   % CT State-Space Model
sys = c2d(sys,Ts);            % DT State-Space Model (ts sampling)
% Signal Names
sys.InputName = {'u1', 'u2', 'u_PA001'};
sys.OutputName = {'h1', 'h2'};
sys.StateName = {'h1', 'h2'};
% Signal Units
sys.InputUnit = {'-', '-', '-'};
sys.OutputUnit = {'m', 'm'};
sys.StateUnit = {'m', 'm'};
% Signal Types
sys = setmpcsignals(sys, 'MV', [1 2], 'MD', 3, 'MO', [1 2]);
% Linear MPC
old_status = mpcverbosity('on');

p = 10;     % Prediction Horizon
%c = 1;      % Control Horizon
c = [2 3 5];     % Control Horizon (Blocking)

mpcobj = mpc(sys, Ts, p, c);    % Linear MPC object

% Nominal Values
x0 = [h1_A; h2_A];
u0 = [u1_A;u2_A;u_PA001_A];
y0 = x0;
mpcobj.Model.Nominal = struct('X',x0,'U',u0,'Y',y0);

% Set Estimator (None)
setEstimator(mpcobj, 'custom'); %N
%setoutdist(mpcobj,'integrators')

% Signal Scaling
mpcobj.OV(1).ScaleFactor = 1 - 0.13;    % Range of h1
mpcobj.OV(2).ScaleFactor = 0.4 - 0.02;  % Range of h2

% Weighting Matrices
```

```
mpcobj.Weights.OutputVariables = [1 1];                  % Q
mpcobj.Weights.ManipulatedVariablesRate = [0.1 0.1];  % S
mpcobj.Weights.ManipulatedVariables = [0 0];           % R
% Output (soft) Constraints
mpcobj.OV(1).Max = 1;
mpcobj.OV(1).Min = 0.13;
mpcobj.OV(2).Max = 0.4;
mpcobj.OV(2).Min = 0.02;
% Output ECR (slack)
mpcobj.OV(1).MinECR = 5;
mpcobj.OV(1).MaxECR = 5;
mpcobj.OV(2).MinECR = 5;
mpcobj.OV(2).MaxECR = 5;
% % Input (hard) Constraints
mpcobj.MV(1).Max = 1;
mpcobj.MV(1).Min = 0;
mpcobj.MV(2).Max = 1;
mpcobj.MV(2).Min = 0;

xc = mpcstate(mpcobj);
%%%%%%%%%%%%%%%%%%%%%%%%%%%% MPC_init end

%%%%%%%%%%%%%%%%%%%%%%%%%%%% MHE init
%2x2: 1,1; 2x1: 9e-1, 1; 3x2:1e-6, 1e-1, 1e-4
%MHE parameters
N_horizon = 10; %Horizon length
% Process disturbance used in simulation and MHE
cov_w1 = 9e-1;
cov_w2 = 1;
cov_w3 = 1e-4;

% Measurement noise used in simulation and MHE
cov_v1 = 2.8e-6; % measurement noise
cov_v2 = 1.6e-6; % measurement noise
if n == 2
    cov_w = diag([cov_w1, cov_w2]);
elseif n == 3
    cov_w = diag([cov_w1, cov_w2, cov_w3]);
end
if m == 2
    cov_v = diag([cov_v1, cov_v2]);
elseif m == 1
    cov_v = cov_v2;
end

% linearization
[A, B, C, D, u_LV001_0, u_LV002_0] = linearization(h1_0,h2_0,u_PA001_0,n,
    m);
```

```
%Discrete-time matrice A (Forward Euler)
A_disc = eye(size(A)) + Ts*A;
%Parameter structure
parameter_struct.n = n;
parameter_struct.m = m;
parameter_struct.N = N_horizon;
parameter_struct.Ts = Ts;
parameter_struct.A = A;
parameter_struct.B = B;
parameter_struct.C = C;
parameter_struct.D = D;
parameter_struct.h1_0 = h1_0;
parameter_struct.h2_0 = h2_0;
parameter_struct.u_LV001_0 = u_LV001_0;
parameter_struct.u_LV002_0 = u_LV002_0;
parameter_struct.u_PA001_0 = u_PA001_0;
parameter_struct.Q = cov_w;
parameter_struct.R = cov_v;
parameter_struct.A_disc = A_disc;

% Init MHE
horizon = zeros(1,N_horizon);
h1_hat_horizon = horizon;
h2_hat_horizon = horizon;
u_PA001_hat_horizon = horizon;
if n == 3
    u_PA001_hat_init = u_PA001_0;
    x_hat_horizon = [h1_hat_horizon; h2_hat_horizon; u_PA001_hat_horizon
   ];
    u_horizon = [horizon; horizon];
elseif n == 2
    x_hat_horizon = [h1_hat_horizon; h2_hat_horizon];
    u_horizon = [horizon; horizon; horizon];
end

h1_meas_horizon = horizon;
h2_meas_horizon = horizon;

% Lower and upper bound constraints for whole horizon
h1_max_arr = zeros(1,N_horizon) + h1_max;
h1_min_arr = zeros(1,N_horizon) + h1_min;
h2_max_arr = zeros(1,N_horizon) + h2_max;
h2_min_arr = zeros(1,N_horizon) + h2_min;
if n == 3
    u_PA001_max_arr = zeros(1,N_horizon) + 1; % From pump characteristic
    u_PA001_min_arr = zeros(1,N_horizon) + 0.45; % From pump
   characteristic
    x_ub = [h1_max_arr; h2_max_arr; u_PA001_max_arr];
    x_lb = [h1_min_arr; h2_min_arr; u_PA001_min_arr];
elseif n == 2
    x_ub = [h1_max_arr; h2_max_arr];
    x_lb = [h1_min_arr; h2_min_arr];
end
parameter_struct.x_ub = x_ub;
parameter_struct.x_lb = x_lb;

%Estimate error covariance matrix matrix used in arrival cost storage
P_k_N_arr = [];
if n == 3
    P_init = diag([0.001, 0.001, 0.001]); %initial P covariance matrix
    x_pred_k_N = [h1_0; h2_0; u_PA001_0];
elseif n == 2
    P_init = diag([0.001, 0.001]); %initial P covariance matrix
    x_pred_k_N = [h1_0; h2_0];
end
P_previous = P_init;
```

I.3 Figure Creation Files

I.3.1 Plot_Data_Visualization_Adaptive_MPC.m

```
%% Control of Two-Tank Using Adaptive MPC.
% Initial condition: {h1 = 0.5, h2 = 0.3, u_PA001 = 0.8}

% Author: Gent Luta

% Date: Spring 2023

%% System Parameters (For Simulink Model)
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

h1_max = 1;
h1_min = 0.13;
h2_max = 0.4;
h2_min = 0.02;
Kv_LV001   = 11.25;
Kv_LV002   = 11.25;
```

```
z_LV001 = 0:0.05:1;
f_LV001 = (exp(z_LV001.^1.2)-1)/(exp(1)-1);

z_LV002 = 0:0.05:1;
f_LV002 = f_LV001;
u_PA001_data = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
                0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
                8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s


%% Steady State Values
h1_A = 0.5;
h2_A = 0.3;
u_PA001_A = 0.8;
u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);

ts = 0.5;   % Sampling time

sys = AdaptiveSys([h1_A;h2_A], [u1_A;u2_A;u_PA001_A], ts);  % DT Linear
    System

%% Linear MPC
old_status = mpcverbosity('on');

Duration = 400;
t = 0:ts:Duration;
N = length(t);

nsim = 1;
COST = cell(1,nsim);
COST_Q = cell(1,nsim);
COST_S = cell(1,nsim);
COST_R = cell(1,nsim);
TIME_ARRAY_QP = cell(1,nsim);
TIME_ARRAY_SYS = cell(1,nsim);
ITERATIONS = cell(1,nsim);

for j = 1:nsim
p = 13;        % Prediction horizon
c = 13;         % Control horizon
%c = [1 4 8];       % Control horizon (blocking)

mpcobj = mpc(sys, ts, p, c);    % Linear MPC object

% Nominal Values
```

```matlab
x0 = [h1_A; h2_A];
u0 = [u1_A;u2_A;u_PA001_A];
y0 = x0;
mpcobj.Model.Nominal = struct('X',x0,'U',u0,'Y',y0);

% Set Estimator (None)
setEstimator(mpcobj, 'custom');

% Signal Scaling
mpcobj.OV(1).ScaleFactor = 1 - 0.13;     % Range of h1
mpcobj.OV(2).ScaleFactor = 0.4 - 0.02;  % Range of h2

% Weighting Matrices
mpcobj.Weights.OutputVariables = [1 1];                  % Q
mpcobj.Weights.ManipulatedVariablesRate = [0.1 0.1];     % S
mpcobj.Weights.ManipulatedVariables = [0 0];             % R

% Output (soft) Constraints
% mpcobj.OV(1).Max = 1;
% mpcobj.OV(1).Min = 0.13;
% mpcobj.OV(2).Max = 0.4;
% mpcobj.OV(2).Min = 0.02;

% % Output ECR (slack)
% mpcobj.OV(1).MinECR = 5;
% mpcobj.OV(1).MaxECR = 5;
% mpcobj.OV(2).MinECR = 5;
% mpcobj.OV(2).MaxECR = 5;

% Input (hard) Constraints
mpcobj.MV(1).Max = 0.9999;
mpcobj.MV(1).Min = 0.0001;
mpcobj.MV(2).Max = 0.9999;
mpcobj.MV(2).Min = 0.0001;

% Input Rate (soft) Constraints
% mpcobj.MV(1).RateMin = -0.1;
% mpcobj.MV(1).RateMax = 0.1;
% mpcobj.MV(2).RateMin = -0.1;
% mpcobj.MV(2).RateMax = 0.1;

% %Input Rate ECR (slack)
% mpcobj.MV(1).RateMinECR = 20;
% mpcobj.MV(1).RateMaxECR = 20;
% mpcobj.MV(2).RateMinECR = 20;
% mpcobj.MV(2).RateMaxECR = 20;

% MPC Optimizer Options:
```

```
% Interior-Point Solver and Option
% mpcobj.Optimizer.Algorithm = 'interior-point';
% mpcobj.Optimizer.InteriorPointOptions.MaxIterations = 8;
% mpcobj.Optimizer.InteriorPointOptions.ConstraintTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.OptimalityTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.ComplementarityTolerance = 1e-6;
% mpcobj.Optimizer.InteriorPointOptions.StepTolerance = 1e-7;

% Active-Set Solver and Options
% mpcobj.Optimizer.Algorithm = 'active-set';
% mpcobj.Optimizer.ActiveSetOptions.MaxIterations = 2;
% mpcobj.Optimizer.ActiveSetOptions.ConstraintTolerance = 1e-6;

% mpcobj.Optimizer.UseSuboptimalSolution = true;

%% SIM
% Signal Previewing
ref = [ ones(1, N-1)*h1_A;
        ones(1, N-1)*h2_A];
u_PA001 = ones(1,N)*u_PA001_A;

% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 150;
t_upa_rise = 250;
t_hold = 70;
idx_ref1 = round(t_h1_rise/ts);
idx_ref2 = round(t_h2_rise/ts);
idx_upa = round(t_upa_rise/ts);
idx_hold = round(t_hold/ts);
%
ref(1,idx_ref1:idx_ref1+idx_hold) = h1_A + 0.2;
ref(2,idx_ref2:idx_ref2+idx_hold) = h2_A - 0.2;
u_PA001(1, idx_upa:idx_upa+idx_hold) = u_PA001_A - 0.2;

%
x = x0;                         % Current state
xc = mpcstate(mpcobj);          % Controller state pointer
xN = x0;                        % Nominal (updated iteratively)
uN = u0;                        % Nominal (updated iteratively)
yN = y0;                        % Nominal (updated iteratively)
dxN = [0;0];                    % Nominal (updated iteratively)

% History Tracking
UU = zeros(2,N); UU(:,1) = u0(1:2);
XX_NL = zeros(2,N); XX_NL(:,1) = x0;
TIME_SPENT_QP = zeros(1,N-1);
TIME_SPENT_SYS = zeros(1,N-1);
```

```
II = zeros(1,N-1);

hbar = waitbar(0, 'Simulation Progress');
for i = 1:(Duration/ts)
    if i <= N-p-1
        r1 = ref(1,i:i+p-1);
        r2 = ref(2,i:i+p-1);
        r = [r1' r2'];
        v = u_PA001(i:i+p-1)';
    else
        r1 = ref(1,i:end);
        r2 = ref(2,i:end);
        r = [r1' r2'];
        v = u_PA001(i:end)';
    end
    %r = ref(:,i)';      %Without signal preview (reference)
    %v = u_PA001(i);     %Without signal preview (disturbance)

    tic
    [mv, info] = mpcmoveAdaptive(mpcobj, xc, sys,...
        struct('X',xN,'U',uN,'Y',yN, 'DX', dxN), [], r, v); % Solve QP
    optimization problem
    TIME_SPENT_QP(i) = toc;

    % Simulate Nonlinear System
    [~,XNL] = ode45(@(t,x)tankCT_NEW(x, [mv;v(1)]), [0 ts], XX_NL(:,i));
    xnl = [XNL(end,1);XNL(end,2)];

    % Update Controller State Based on Nonlinear System
    xc.Plant = xnl;
    xc.LastMove = mv;
    xc.Disturbance = [0;0];

    % Update Prediction Model and Nominal Solution
     tic
     sys = AdaptiveSys(xnl, [mv;v(1)], ts);
     xN = xnl;
     uN = [mv;v(1)];
     yN = xnl;
     dxN = xnl - XX_NL(:,i);
     TIME_SPENT_SYS(i) = toc;
    % History tracking
    UU(:,i+1) = mv;
    II(i) = info.Iterations;
    XX_NL(:,i+1) = xnl;

    waitbar(i*ts/Duration, hbar);
end
```

```
close(hbar)

TIME_ARRAY_QP{1,j} = TIME_SPENT_QP;
TIME_ARRAY_SYS{1,j} = TIME_SPENT_SYS;
ITERATIONS{1,j} = II;

cost = 0;
cost_Q = 0;
cost_S = 0;
cost_R = 0;

% Find Total Quadratic Cost
Q = diag(mpcobj.Weights.OutputVariables);
Q(1,1) = Q(1,1)/mpcobj.OV(1).ScaleFactor;   % Scaling
Q(2,2) = Q(2,2)/mpcobj.OV(2).ScaleFactor;   % Scaling
S = diag(mpcobj.Weights.ManipulatedVariablesRate);
R = diag(mpcobj.Weights.ManipulatedVariables);
E = XX_NL(:,2:end) - ref;                    % OV reference tracking error
EU = UU(:,:) - u0(1:2,:);                    % MV target tracking error
mvRate = UU(:,1:end-1) - UU(:,2:end);        % MV rate
for l = 1:N-1
    cost_Q = cost_Q + E(:,l)'*(Q^2)*E(:,l);
    cost_S = cost_S + mvRate(:,l)'*(S^2)*mvRate(:,l);
    cost_R = cost_R + EU(:,l)'*(R^2)*EU(:,l);
    cost = cost +  E(:,l)'*(Q^2)*E(:,l) + mvRate(:,l)'*(S^2)*mvRate(:,l)
   + EU(:,l)'*(R^2)*EU(:,l);
end
COST{1,j} = cost;
COST_Q{1,j} = cost_Q;
COST_S{1,j} = cost_S;
COST_R{1,j} = cost_R;
end

%% Plot - Simulation
figure

% Tank 1
subplot(3,1,1)
plot(t, XX_NL(1,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(1,:), 'red', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
legend('$h_{1}$','$h_{1,\,ref}$', 'Interpreter','latex')
ylim([0 1]);
grid on
box on
```

```
set(gca,'YTick',0:0.05:1)

% Tank 2
subplot(3,1,2)
plot(t, XX_NL(2,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(2,:), 'red', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
legend('$h_{2}$','$h_{2,\,ref}$', 'Interpreter','latex')
ylim([0 1])
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1)

% Input Signals
subplot(3,1,3)
stairs(t, UU(1,:), 'magenta', 'LineWidth',1)
hold on
stairs(t, UU(2,:), 'green', 'LineWidth',1)
hold on
stairs(t, u_PA001, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('[-]', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    )
ylim([0 1])
grid on
box on

%% Time spent QP
TIME_MAT = zeros(N-1, nsim);
Lable_struc = cell(1,nsim);
for i = 1:nsim
    TIME_MAT(:,i) = TIME_ARRAY_QP{1,i}';
    Lable_struc{i} = sprintf('Sim%i', i);
end
figure;
boxplot(TIME_MAT,"BoxStyle","outline",'MedianStyle','line','Notch','off',
    'OutlierSize',6,...
    'Symbol','.r', Jitter=0, Labels=Lable_struc)
xlabel('Simulations', Interpreter='latex')
ylabel('Time $[s]$', Interpreter='latex')
title('Box Plot w.r.t. Execution Time of QP Solver', Interpreter='latex')
box on
```

```
grid on

%% Time spent SYS
TIME_MAT = zeros(N-1, nsim);
Lable_struc = cell(1,nsim);
for i = 1:nsim
    TIME_MAT(:,i) = TIME_ARRAY_SYS{1,i}';
    Lable_struc{i} = sprintf('Sim%i', i);
end
figure;
boxplot(TIME_MAT,"BoxStyle","outline",'MedianStyle','line','Notch','off',
    'OutlierSize',6,...
    'Symbol','.r', Jitter=0, Labels=Lable_struc)
xlabel('Simulations', Interpreter='latex')
ylabel('Time $[s]$', Interpreter='latex')
title('Box Plot w.r.t. Execution Time of Prediction Model Update',
    Interpreter='latex')
box on
grid on

%% Time spent TOTAL
TIME_MAT = zeros(N-1, nsim);
Lable_struc = cell(1,nsim);
for i = 1:nsim
    TIME_MAT(:,i) = TIME_ARRAY_QP{1,i}' + TIME_ARRAY_SYS{1,i}';
    Lable_struc{i} = sprintf('Sim%i', i);
end
figure;
boxplot(TIME_MAT,"BoxStyle","outline",'MedianStyle','line','Notch','off',
    'OutlierSize',6,...
    'Symbol','.r', Jitter=0, Labels=Lable_struc)
xlabel('Simulations', Interpreter='latex')
ylabel('Time $[s]$', Interpreter='latex')
title('Box Plot w.r.t. Execution Time of QP Solver plus Prediction Model
    Update', Interpreter='latex')
box on
grid on

result = 0;
result_Q = 0;
result_S = 0;
for i = 1:nsim
    result = result + sum(TIME_ARRAY_QP{1,i} + TIME_ARRAY_SYS{1,i});
    result_Q = result_Q + sum(TIME_ARRAY_QP{1,i});
    result_S = result_S + sum(TIME_ARRAY_SYS{1,i});
end
avrg = result/10;
avrg_Q = result_Q/10;
```

```
avrg_S = result_S/10;

for i = 1:nsim
    i
    sum(TIME_ARRAY_SYS{1,i})
end

%% Quadratic Cost
COST_mat = cell2mat(COST);
COST_Q_mat = cell2mat(COST_Q);
COST_R_mat = cell2mat(COST_R);
COST_S_mat = cell2mat(COST_S);

sprintf('Mean Total Cost: %f',mean(COST_mat))
sprintf('Mean Total Cost - Q: %f',mean(COST_Q_mat))
sprintf('Mean Total Cost - S: %f',mean(COST_S_mat))
sprintf('Mean Total Cost - R: %f',mean(COST_R_mat))

%% Show Iterations and time taken on every control interval. Fill out
    this part!
figure
f = zeros(1,3);
yyaxis left
f(1) = stem(TIME_ARRAY_QP{1,9},'blue', 'LineWidth',.05, 'LineStyle','-',
    MarkerSize=0.001);
ylabel('Time $[s]$',Interpreter='latex')

hold on
f(2) = yline(2.02*10^-3, 'k', 'LineWidth',1, 'LineStyle','--');
%ylim([0 0.004])
yyaxis right

f(3) = stem(ITERATIONS{1,9},'Color',[255/255,69/255,0], 'LineWidth',.05,
    'LineStyle','-', MarkerSize=0.001);
ylabel('Iterations',Interpreter='latex')
set(gca,'YTick',0:2:40)
ylim([0 40])

ax = gca;
ax.YAxis(1).Color = 'b';
ax.YAxis(2).Color = [255/255,69/255,0];
ax.XAxis(1).Color = 'k';

title('Execution Time and Iterations of Every QP Solver Call',Interpreter
    ='latex')
xlabel('Control Interval $k$',Interpreter='latex')
legend(f(1:3),'Time Spent', 'Upper Adjacent', 'Iterations Required', '
    Interpreter','latex')
```

```
grid on
box on
```

I.3.2 Plot_Data_Visualization_Explicit_MPC_V2.m

```
%% Control of Two-Tank Using Explicit MPC.
% Linearized model about the nominal solution {h1 = 0.5, h2 = 0.3,
    u_PA001 = 0.8}

% Author: Gent Luta

% Date: Spring 2023

%% Steady State Values
h1_A = 0.5;
h2_A = 0.3;
u_PA001_A = 0.8;
u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);
f1_A = ValveChar(u1_A);
f2_A = ValveChar(u2_A);

%% System Parameters
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

%% Function Handles
f1_handle = @ValveChar;
f2_handle = @ValveChar;
f3_handle = @PumpChar;

delta = 0.01;   % Step size when using forward difference

%% Linearization
% A - Matrix
a11 = - (sqrt(100000))/(7.2*10^8) * ((Kv1*f1_A*rho*g) / (A1*sqrt(rho*g*(
    h1_A + h_LV001))));
a12 = 0;
a21 = ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*10^8)) * ((Kv1*
    f1_A*rho*g) / (sqrt(rho*g*(h1_A + h_LV001))));
```

```
a22 = - ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*(10^8))) * ((Kv2
    *f2_A*rho*g) / (sqrt(rho*g*(h2_A + h_LV002))));
A = [a11 a12; a21 a22];

% B - Matrix
b11 = - ((Kv1)/(3600*A1)) * sqrt((rho*g*(h1_A + h_LV001))/100000) *
    forward_diff(f1_handle, u1_A, delta);
b12 = 0;
b21 = ((1)/(0.004 + 0.07*h2_A)) * (Kv1/3600) * sqrt((rho*g*(h1_A +
    h_LV001))/100000)*forward_diff(f1_handle, u1_A, delta);
b22 = -((1)/(0.004 + 0.07*h2_A)) * (Kv2/3600) * sqrt((rho*g*(h2_A +
    h_LV002))/100000)*forward_diff(f2_handle, u2_A, delta);
B = [b11 b12 ; b21 b22];

% C - Matrix
C = eye(2);

% D - Matrix
D = zeros(2);

% G - Matrix (Disturbance)
g11 = forward_diff(f3_handle, u_PA001_A, delta)/A1;
g21 = 0;
G = [g11; g21];

% Augmented B - Matrix
B_a = [B G];

% Augmented D - Matrix
D_a = zeros(2, 3);

%% State-Space-Model
ts = 0.5;                       % Sampling time
sys = ss(A, B_a, C, D_a);      % CT state-space model
sys = c2d(sys,ts);              % DT state-space model (ts sampling)
% Signal Names
sys.InputName = {'u1', 'u2', 'u_PA001'};
sys.OutputName = {'h1', 'h2'};
sys.StateName = {'h1', 'h2'};
% Signal Units
sys.InputUnit = {'-', '-', '-'};
sys.OutputUnit = {'m', 'm'};
sys.StateUnit = {'m', 'm'};
% Signal Types
sys = setmpcsignals(sys, 'MV', [1 2], 'MD', 3, 'MO', [1 2]);

%% Linear MPC
old_status = mpcverbosity('on');
```

```

M = 1:9;
REGIONS = zeros(1,M(end));
EMPC = cell(1,M(end));
BYTES = zeros(1,M(end));
COST = zeros(1,M(end));
for m = M
p = 13;      % Prediction horizon
c = m;
%c = [1 2 2 3 5];       % Control horizon
%c = [2 4 7];       % Control horizon (blocking)

mpcobj = mpc(sys, ts, p, c);    % Linear MPC object

% Nominal Values
x0 = [h1_A; h2_A];
u0 = [u1_A;u2_A;u_PA001_A];
y0 = x0;
mpcobj.Model.Nominal = struct('X',x0,'U',u0,'Y',y0);

% Set Estimator (None)
setEstimator(mpcobj, 'custom');

% Signal Scaling
mpcobj.OV(1).ScaleFactor = 1 - 0.13;    % Range of h1
mpcobj.OV(2).ScaleFactor = 0.4 - 0.02;  % Range of h2

% Weighting Matrices
mpcobj.Weights.OutputVariables = [1 1];                 % Q
mpcobj.Weights.ManipulatedVariablesRate = [0.1 0.1];    % S
mpcobj.Weights.ManipulatedVariables = [0 0];            % R

% Output (soft) Constraints
% mpcobj.OV(1).Max = 1;
% mpcobj.OV(1).Min = 0.13;
% mpcobj.OV(2).Max = 0.4;
% mpcobj.OV(2).Min = 0.02;

% % Output ECR (slack)
% mpcobj.OV(1).MinECR = 5;
% mpcobj.OV(1).MaxECR = 5;
% mpcobj.OV(2).MinECR = 5;
% mpcobj.OV(2).MaxECR = 5;

% Input (hard) Constraints
mpcobj.MV(1).Max = 0.9999;
mpcobj.MV(1).Min = 0.0001;
mpcobj.MV(2).Max = 0.9999;
mpcobj.MV(2).Min = 0.0001;

% Input Rate (soft) Constraints
% mpcobj.MV(1).RateMin = -0.3;
% mpcobj.MV(1).RateMax = 0.3;
% mpcobj.MV(2).RateMin = -0.3;
% mpcobj.MV(2).RateMax = 0.3;

% Input Rate ECR (slack)
% mpcobj.MV(1).RateMinECR = 20;
% mpcobj.MV(1).RateMaxECR = 20;
% mpcobj.MV(2).RateMinECR = 20;
% mpcobj.MV(2).RateMaxECR = 20;

% Interior-Point Solver and Option
% mpcobj.Optimizer.Algorithm = 'interior-point';
% mpcobj.Optimizer.InteriorPointOptions.MaxIterations = 8;
% mpcobj.Optimizer.InteriorPointOptions.ConstraintTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.OptimalityTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.ComplementarityTolerance = 1e-6;
% mpcobj.Optimizer.InteriorPointOptions.StepTolerance = 1e-7;

% Active-Set Solver and Options
% mpcobj.Optimizer.Algorithm = 'active-set';
% mpcobj.Optimizer.ActiveSetOptions.MaxIterations = 2;
% mpcobj.Optimizer.ActiveSetOptions.ConstraintTolerance = 1e-6;

% mpcobj.Optimizer.UseSuboptimalSolution = true;

%% Explicit MPC
% Define the Range:
range = generateExplicitRange(mpcobj);

% Range of the States
range.State.Min(:) = [0.13 0.02 -0.001 -0.001];
range.State.Max(:) = [1    0.4   0.001  0.001];
% Range of the References
range.Reference.Min(:) = [0.13 0.02];
range.Reference.Max(:) = [1    0.4];
% Range of the Measured Disturbance
range.MeasuredDisturbance.Min = 0;
range.MeasuredDisturbance.Max = 1;
% Range of the Manipulated Variables
range.ManipulatedVariable.Min(:) = [0.00001 0.00001];
range.ManipulatedVariable.Max(:) = [0.99999 0.99999];

% Option for EMPC
opt = generateExplicitOptions(mpcobj);
```

```
opt.polyreduction = 1;
%opt.maxiterBS = 200;
%opt.maxiterNNLS = 1000;
%opt.maxiterQP = 400;
%opt.flattol = 1e-6;
%opt.normalizetol = 10;
%opt.removetol = 1e-2;
%opt.zerotol = 1e-10;

% Create EMPC Obj.
empcobj = generateExplicitMPC(mpcobj,range,opt);
%empcobj = generateExplicitMPC(mpcobj,range);
EMPC{1,m} = empcobj;
[~, reg] = size(empcobj.PiecewiseAffineSolution);
REGIONS(1,m) = reg;
s = whos('empcobj');
BYTES(1,m) = s.bytes;
end
%empcobjSimplified = simplify(empcobj, 'exact');

%% Plot EMPC Regions at Spesific Params.
empcobj = EMPC{4};
plotParams = generatePlotParameters(empcobj);

plotParams.State.Index = [3 4];
plotParams.State.Value = [0 0];

plotParams.ManipulatedVariable.Index = [1 2];
plotParams.ManipulatedVariable.Value = [0.2 0.2];

plotParams.Reference.Index = [1 2];
plotParams.Reference.Value = [0.2 0.35];

plotParams.MeasuredDisturbance.Index = 1;
plotParams.MeasuredDisturbance.Value = 0.9;

plotSection(empcobj,plotParams)
xlabel('$h_{1}(k)$', Interpreter='latex')
ylabel('$h_{2}(k)$', Interpreter='latex')
title('2-D Plot of Explicit MPC Polyhedral Partition', Interpreter='latex
    ')
box on
grid on
xlim([0 1])
ylim([0 0.4])

%% SIM
Duration = 400;          % Simulation time
```

```matlab
t = 0:ts:Duration;
N = length(t);

nsim = length(M);
COST = cell(1,nsim);
COST_Q = cell(1,nsim);
COST_S = cell(1,nsim);
COST_R = cell(1,nsim);
TIME_ARRAY = cell(1,nsim);

for j = 1:nsim
empcobj = EMPC{j};
% Signal Previewing
ref = [ ones(1, N-1)*h1_A;
        ones(1, N-1)*h2_A];
u_PA001 = ones(1,N)*u_PA001_A;

% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 150;
t_upa_rise = 250;
t_hold = 70;
idx_ref1 = round(t_h1_rise/ts);
idx_ref2 = round(t_h2_rise/ts);
idx_upa = round(t_upa_rise/ts);
idx_hold = round(t_hold/ts);
%
ref(1,idx_ref1:idx_ref1+idx_hold) = h1_A + 0.2;
ref(2,idx_ref2:idx_ref2+idx_hold) = h2_A - 0.2;
u_PA001(1, idx_upa:idx_upa+idx_hold) = u_PA001_A - 0.2;

%
x = x0;                         % Current state
xc = mpcstate(empcobj);         % Controller state pointer
%xc.Plant = x0;
%xc.LastMove = u0(1:2);

% History Tracking
UU = zeros(2,N); UU(:,1) = u0(1:2);
XX_NL = zeros(2,N); XX_NL(:,1) = x0;
TIME_SPENT = zeros(1,N-1);


hbar = waitbar(0, 'Simulation Progress');
for i = 1:(Duration/ts)
    r = ref(:,i)';      %Without signal preview (reference)
    v = u_PA001(i);     %Without signal preview (disturbance)

    tic
    [mv, info] = mpcmoveExplicit(empcobj, xc, [], r, v);     % Find
    optimal MV
    TIME_SPENT(1,i) = toc;
    if mv(1) < 0
        mv(1) = 0;
    end
    if mv(2) < 0
        mv(2) = 0;
    end

    % Simulate Linear System

    % Update Controller State Based on Linear System (Uncomment to Use)
    %xc.Plant = x; %N
    %xc.LastMove = mv; %N
    %xc.Disturbance = [0;0]; %N

    % Simulate Nonlinear System
    [~,XNL] = ode45(@(t,x)tankCT_NEW(x, [mv;v(1)]), [0 ts], XX_NL(:,i));
    xnl = [XNL(end,1);XNL(end,2)];

    % Update Controller State Based on Nonlinear System
    xc.Plant = xnl;
    xc.LastMove = mv;
    %xc.Disturbance = [0;0];

    %  History tracking
    UU(:,i+1) = mv;
    XX_NL(:,i+1) = xnl;

    waitbar(i*ts/Duration, hbar);
end
close(hbar)

TIME_ARRAY{1,j} = TIME_SPENT;

% Find Total Quadratic Cost
cost = 0;
cost_Q = 0;
cost_S = 0;
cost_R = 0;

Q = diag(mpcobj.Weights.OutputVariables);
Q(1,1) = Q(1,1)/mpcobj.OV(1).ScaleFactor;    % Scaling
Q(2,2) = Q(2,2)/mpcobj.OV(2).ScaleFactor;    % Scaling
S = diag(mpcobj.Weights.ManipulatedVariablesRate);
R = diag(mpcobj.Weights.ManipulatedVariables);
```

```
E = XX_NL(:,2:end) - ref;                      % OV reference tracking error
EU = UU(:,:) - u0(1:2,:);                      % MV target tracking error
mvRate = UU(:,1:end-1) - UU(:,2:end);          % MV rate
for l = 1:N-1
    cost_Q = cost_Q + E(:,l)'*(Q^2)*E(:,l);
    cost_S = cost_S + mvRate(:,l)'*(S^2)*mvRate(:,l);
    cost_R = cost_R + EU(:,l)'*(R^2)*EU(:,l);
    cost = cost +  E(:,l)'*(Q^2)*E(:,l) + mvRate(:,l)'*(S^2)*mvRate(:,l)
    + EU(:,l)'*(R^2)*EU(:,l);
end
COST{1,j} = cost;
COST_Q{1,j} = cost_Q;
COST_S{1,j} = cost_S;
COST_R{1,j} = cost_R;
end

%% Plot - Simulation
figure

% Tank 1
subplot(3,1,1)
%plot(t, XX(1,:),'blue', 'LineWidth',1)
%hold on
plot(t, XX_NL(1,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(1,:), 'red', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
%legend('$h_{1}(t)$', '$h_{1,\,ref}$','$h_{1,\,NL}$', 'Interpreter','
   latex')
legend('$h_{1}$', '$h_{1,\,ref}$', 'Interpreter','latex')
ylim([0 1]);
grid on
box on
set(gca,'YTick',0:0.05:1)

% Tank 2
subplot(3,1,2)
%plot(t, XX(2,:), 'blue', 'LineWidth',1)
%hold on
plot(t, XX_NL(2,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(2,:), 'red', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
```

```
%legend('$h_{2}(t)$', '$h_{2,\,ref}$','$h_{2,\,NL}$', 'Interpreter','
    latex')
legend('$h_{2}$', '$h_{2,\,ref}$', 'Interpreter','latex')
ylim([0 1])
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1)

% Input Signals
subplot(3,1,3)
stairs(t, UU(1,:), 'magenta', 'LineWidth',1)
hold on
stairs(t, UU(2,:), 'green', 'LineWidth',1)
hold on
stairs(t, u_PA001, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('[-]', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    )
ylim([0 1])
grid on
box on


%% Time Spent
TIME_MAT = zeros(N-1, nsim);
Lable_struc = cell(1,nsim);
for i = 1:nsim
    TIME_MAT(:,i) = TIME_ARRAY{1,i}';
    Lable_struc{i} = sprintf('Sim%i', i);
end
figure;
boxplot(TIME_MAT,"BoxStyle","outline",'MedianStyle','line','Notch','off',
    'OutlierSize',6,...
    'Symbol','.r', Jitter=0, Labels=Lable_struc)
xlabel('Simulations', Interpreter='latex')
ylabel('Time $[s]$', Interpreter='latex')
title('Box Plot w.r.t. Execution Time of QP Solver', Interpreter='latex')
box on
grid on

%% Quadratic Cost
COST_mat = cell2mat(COST);
COST_Q_mat = cell2mat(COST_Q);
COST_R_mat = cell2mat(COST_R);
COST_S_mat = cell2mat(COST_S);
```

```
sprintf('Mean Total Cost: %f',mean(COST_mat))
sprintf('Mean Total Cost - Q: %f',mean(COST_Q_mat))
sprintf('Mean Total Cost - S: %f',mean(COST_S_mat))
sprintf('Mean Total Cost - R: %f',mean(COST_R_mat))

%% figure
figure
yyaxis left
stem(M,cell2mat(COST),'blue', 'LineWidth',1, 'LineStyle','-',MarkerSize
    =10);
ylabel('Total Quadratic Cost',Interpreter='latex')

%ylim([0 0.004])
yyaxis right

stem(M,REGIONS,'Color',[255/255,69/255,0], 'LineWidth',1, 'LineStyle','-'
    , MarkerSize=10,Marker='x');
ylabel('Polyhedral Regions',Interpreter='latex')
%set(gca,'YTick',0:2:40)
%ylim([0 40])
set(gca, 'YScale', 'log')
ax = gca;
ax.YAxis(1).Color = 'b';
ax.YAxis(2).Color = [255/255,69/255,0];
ax.XAxis(1).Color = 'k';
xlim([0 10])
set(gca,'XTick',0:1:10)
title('Control Horizon v. Quadratic Cost and Explicit MPC Complexity',
    Interpreter='latex')
xlabel('Control Horizon $M$',Interpreter='latex')
legend('Total Quadratic Cost', 'Polyhedral Regions', 'Interpreter','latex
    ')
grid on
box on
```

I.3.3 Plot_Data_Visualization_Explicit_MPC.m

```
%% Control of Two-Tank Using Explicit MPC.
% Linearized model about the nominal solution {h1 = 0.5, h2 = 0.3,
    u_PA001 = 0.8}

% Author: Gent Luta

% Date: Spring 2023

```

```
%% Steady State Values
h1_A = 0.5;
h2_A = 0.3;
u_PA001_A = 0.8;
u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);
f1_A = ValveChar(u1_A);
f2_A = ValveChar(u2_A);

%% System Parameters
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

%% Function Handles
f1_handle = @ValveChar;
f2_handle = @ValveChar;
f3_handle = @PumpChar;

delta = 0.01;    % Step size when using forward difference

%% Linearization
% A - Matrix
a11 = - (sqrt(100000))/(7.2*10^8) * ((Kv1*f1_A*rho*g) / (A1*sqrt(rho*g*(
    h1_A + h_LV001))));
a12 = 0;
a21 = ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*10^8)) * ((Kv1*
    f1_A*rho*g) / (sqrt(rho*g*(h1_A + h_LV001))));
a22 = - ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*(10^8))) * ((Kv2
    *f2_A*rho*g) / (sqrt(rho*g*(h2_A + h_LV002))));
A = [a11 a12; a21 a22];

% B - Matrix
b11 = - ((Kv1)/(3600*A1)) * sqrt((rho*g*(h1_A + h_LV001))/100000) *
    forward_diff(f1_handle, u1_A, delta);
b12 = 0;
b21 = ((1)/(0.004 + 0.07*h2_A)) * (Kv1/3600) * sqrt((rho*g*(h1_A +
    h_LV001))/100000)*forward_diff(f1_handle, u1_A, delta);
b22 = -((1)/(0.004 + 0.07*h2_A)) * (Kv2/3600) * sqrt((rho*g*(h2_A +
    h_LV002))/100000)*forward_diff(f2_handle, u2_A, delta);
B = [b11 b12 ; b21 b22];

% C - Matrix
C = eye(2);
```

```

% D - Matrix
D = zeros(2);

% G - Matrix (Disturbance)
g11 = forward_diff(f3_handle, u_PA001_A, delta)/A1;
g21 = 0;
G = [g11; g21];

% Augmented B - Matrix
B_a = [B G];

% Augmented D - Matrix
D_a = zeros(2, 3);

%% State-Space-Model
ts = 0.5;                    % Sampling time
sys = ss(A, B_a, C, D_a);    % CT state-space model
sys = c2d(sys,ts);           % DT state-space model (ts sampling)
% Signal Names
sys.InputName = {'u1', 'u2', 'u_PA001'};
sys.OutputName = {'h1', 'h2'};
sys.StateName = {'h1', 'h2'};
% Signal Units
sys.InputUnit = {'-', '-', '-'};
sys.OutputUnit = {'m', 'm'};
sys.StateUnit = {'m', 'm'};
% Signal Types
sys = setmpcsignals(sys, 'MV', [1 2], 'MD', 3, 'MO', [1 2]);

%% Linear MPC
old_status = mpcverbosity('on');

p = 13;      % Prediction horizon
c = 2;       % Control horizon
%c = [2 4 7];       % Control horizon (blocking)

mpcobj = mpc(sys, ts, p, c);    % Linear MPC object

% Nominal Values
x0 = [h1_A; h2_A];
u0 = [u1_A;u2_A;u_PA001_A];
y0 = x0;
mpcobj.Model.Nominal = struct('X',x0,'U',u0,'Y',y0);

% Set Estimator (None)
setEstimator(mpcobj, 'custom');

```

```
% Signal Scaling
mpcobj.OV(1).ScaleFactor = 1 - 0.13;    % Range of h1
mpcobj.OV(2).ScaleFactor = 0.4 - 0.02;  % Range of h2

% Weighting Matrices
mpcobj.Weights.OutputVariables = [1 5];                   % Q
mpcobj.Weights.ManipulatedVariablesRate = [0.5 0.5];     % S
mpcobj.Weights.ManipulatedVariables = [0 0];             % R

% Output (soft) Constraints
% mpcobj.OV(1).Max = 1;
% mpcobj.OV(1).Min = 0.13;
% mpcobj.OV(2).Max = 0.4;
% mpcobj.OV(2).Min = 0.02;

% % Output ECR (slack)
% mpcobj.OV(1).MinECR = 5;
% mpcobj.OV(1).MaxECR = 5;
% mpcobj.OV(2).MinECR = 5;
% mpcobj.OV(2).MaxECR = 5;

% Input (hard) Constraints
mpcobj.MV(1).Max = 0.9999;
mpcobj.MV(1).Min = 0.0001;
mpcobj.MV(2).Max = 0.9999;
mpcobj.MV(2).Min = 0.0001;

% Input Rate (soft) Constraints
% mpcobj.MV(1).RateMin = -0.3;
% mpcobj.MV(1).RateMax = 0.3;
% mpcobj.MV(2).RateMin = -0.3;
% mpcobj.MV(2).RateMax = 0.3;

% Input Rate ECR (slack)
% mpcobj.MV(1).RateMinECR = 20;
% mpcobj.MV(1).RateMaxECR = 20;
% mpcobj.MV(2).RateMinECR = 20;
% mpcobj.MV(2).RateMaxECR = 20;

% Interior-Point Solver and Option
% mpcobj.Optimizer.Algorithm = 'interior-point';
% mpcobj.Optimizer.InteriorPointOptions.MaxIterations = 8;
% mpcobj.Optimizer.InteriorPointOptions.ConstraintTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.OptimalityTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.ComplementarityTolerance = 1e-6;
% mpcobj.Optimizer.InteriorPointOptions.StepTolerance = 1e-7;

% Active-Set Solver and Options
% mpcobj.Optimizer.Algorithm = 'active-set';
% mpcobj.Optimizer.ActiveSetOptions.MaxIterations = 2;
% mpcobj.Optimizer.ActiveSetOptions.ConstraintTolerance = 1e-6;

% mpcobj.Optimizer.UseSuboptimalSolution = true;

%% Explicit MPC
% Define the Range:
range = generateExplicitRange(mpcobj);

% Range of the States
range.State.Min(:) = [0.13 0.02 -0.1 -0.1];
range.State.Max(:) = [1    0.4   0.1  0.1];
% Range of the References
range.Reference.Min(:) = [0.13 0.02];
range.Reference.Max(:) = [1    0.4];
% Range of the Measured Disturbance
range.MeasuredDisturbance.Min = 0.00001;
range.MeasuredDisturbance.Max = 0.99999;
% Range of the Manipulated Variables
range.ManipulatedVariable.Min(:) = [0.00001 0.00001];
range.ManipulatedVariable.Max(:) = [0.99999 0.99999];

% Option for EMPC
opt = generateExplicitOptions(mpcobj);
opt.polyreduction = 1;
%opt.maxiterBS = 200;
%opt.maxiterNNLS = 1000;
%opt.maxiterQP = 400;
%opt.flattol = 1e-6;
%opt.normalizetol = 10;
%opt.removetol = 1e-2;
%opt.zerotol = 1e-10;

% Create EMPC Obj.
empcobj = generateExplicitMPC(mpcobj,range,opt);
%empcobj = generateExplicitMPC(mpcobj,range);
%empcobjSimplified = simplify(empcobj, 'exact');

%% Plot EMPC Regions at Spesific Params.
plotParams = generatePlotParameters(empcobj);

plotParams.State.Index = [3 4];
plotParams.State.Value = [0 0];

plotParams.ManipulatedVariable.Index = [1 2];
plotParams.ManipulatedVariable.Value = [0.2 0.2];

plotParams.Reference.Index = [1 2];
plotParams.Reference.Value = [0.2 0.35];

plotParams.MeasuredDisturbance.Index = 1;
plotParams.MeasuredDisturbance.Value = 0.9;

plotSection(empcobj,plotParams)
xlabel('$h_{1}(k)$', Interpreter='latex')
ylabel('$h_{2}(k)$', Interpreter='latex')
title('2-D Plot of Explicit MPC Polyhedral Partition', Interpreter='latex
    ')
box on
grid on
xlim([0 1])
ylim([0 0.4])

%% SIM
Duration = 400;         % Simulation time
t = 0:ts:Duration;
N = length(t);

nsim = 1;
COST = cell(1,nsim);
COST_Q = cell(1,nsim);
COST_S = cell(1,nsim);
COST_R = cell(1,nsim);
TIME_ARRAY = cell(1,nsim);

for j = 1:nsim

% Signal Previewing
ref = [ ones(1, N-1)*h1_A;
        ones(1, N-1)*h2_A];
u_PA001 = ones(1,N)*u_PA001_A;

% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 150;
t_upa_rise = 250;
t_hold = 70;
idx_ref1 = round(t_h1_rise/ts);
idx_ref2 = round(t_h2_rise/ts);
idx_upa = round(t_upa_rise/ts);
idx_hold = round(t_hold/ts);
%
ref(1,idx_ref1:idx_ref1+idx_hold) = h1_A + 0.2;
ref(2,idx_ref2:idx_ref2+idx_hold) = h2_A - 0.2;
u_PA001(1, idx_upa:idx_upa+idx_hold) = u_PA001_A - 0.2;
```

```

%
x = x0;                          % Current state
xc = mpcstate(empcobj);          % Controller state pointer
%xc.Plant = x0;
%xc.LastMove = u0(1:2);

% History Tracking
UU = zeros(2,N); UU(:,1) = u0(1:2);
XX_NL = zeros(2,N); XX_NL(:,1) = x0;
TIME_SPENT = zeros(1,N-1);


hbar = waitbar(0, 'Simulation Progress');
for i = 1:(Duration/ts)
    r = ref(:,i)';      %Without signal preview (reference)
    v = u_PA001(i);     %Without signal preview (disturbance)

    tic
    [mv, info] = mpcmoveExplicit(empcobj, xc, [], r, v);     % Find
    optimal MV
    TIME_SPENT(1,i) = toc;

    % Simulate Linear System

    % Update Controller State Based on Linear System (Uncomment to Use)
    %xc.Plant = x; %N
    %xc.LastMove = mv; %N
    %xc.Disturbance = [0;0]; %N

    % Simulate Nonlinear System
    [~,XNL] = ode45(@(t,x)tankCT_NEW(x, [mv;v(1)]), [0 ts], XX_NL(:,i));
    xnl = [XNL(end,1);XNL(end,2)];

    % Update Controller State Based on Nonlinear System
    xc.Plant = xnl;
    xc.LastMove = mv;
    %xc.Disturbance = [0;0];

    %  History tracking
    UU(:,i+1) = mv;
    XX_NL(:,i+1) = xnl;

    waitbar(i*ts/Duration, hbar);
end
close(hbar)

TIME_ARRAY{1,j} = TIME_SPENT;
```

```matlab

% Find Total Quadratic Cost
cost = 0;
cost_Q = 0;
cost_S = 0;
cost_R = 0;

Q = diag(mpcobj.Weights.OutputVariables);
Q(1,1) = Q(1,1)/mpcobj.OV(1).ScaleFactor;   % Scaling
Q(2,2) = Q(2,2)/mpcobj.OV(2).ScaleFactor;   % Scaling
S = diag(mpcobj.Weights.ManipulatedVariablesRate);
R = diag(mpcobj.Weights.ManipulatedVariables);
E = XX_NL(:,2:end) - ref;                   % OV reference tracking error
EU = UU(:,:) - u0(1:2,:);                   % MV target tracking error
mvRate = UU(:,1:end-1) - UU(:,2:end);       % MV rate
for l = 1:N-1
    cost_Q = cost_Q + E(:,l)'*(Q^2)*E(:,l);
    cost_S = cost_S + mvRate(:,l)'*(S^2)*mvRate(:,l);
    cost_R = cost_R + EU(:,l)'*(R^2)*EU(:,l);
    cost = cost +  E(:,l)'*(Q^2)*E(:,l) + mvRate(:,l)'*(S^2)*mvRate(:,l)
    + EU(:,l)'*(R^2)*EU(:,l);
end
COST{1,j} = cost;
COST_Q{1,j} = cost_Q;
COST_S{1,j} = cost_S;
COST_R{1,j} = cost_R;
end

%% Plot - Simulation
figure

% Tank 1
subplot(3,1,1)
%plot(t, XX(1,:),'blue', 'LineWidth',1)
%hold on
plot(t, XX_NL(1,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(1,:), 'red', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
%legend('$h_{1}(t)$', '$h_{1,\,ref}$','$h_{1,\,NL}$', 'Interpreter','
    latex')
legend('$h_{1}$', '$h_{1,\,ref}$', 'Interpreter','latex')
ylim([0 1]);
grid on
box on
set(gca,'YTick',0:0.05:1)
```

```matlab

% Tank 2
subplot(3,1,2)
%plot(t, XX(2,:), 'blue', 'LineWidth',1)
%hold on
plot(t, XX_NL(2,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(2,:), 'red', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
%legend('$h_{2}(t)$', '$h_{2,\,ref}$','$h_{2,\,NL}$', 'Interpreter','
    latex')
legend('$h_{2}$', '$h_{2,\,ref}$', 'Interpreter','latex')
ylim([0 1])
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1)

% Input Signals
subplot(3,1,3)
stairs(t, UU(1,:), 'magenta', 'LineWidth',1)
hold on
stairs(t, UU(2,:), 'green', 'LineWidth',1)
hold on
stairs(t, u_PA001, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('[-]', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    )
ylim([0 1])
grid on
box on


%% Time Spent
TIME_MAT = zeros(N-1, nsim);
Lable_struc = cell(1,nsim);
for i = 1:nsim
    TIME_MAT(:,i) = TIME_ARRAY{1,i}';
    Lable_struc{i} = sprintf('Sim%i', i);
end
figure;
boxplot(TIME_MAT,"BoxStyle","outline",'MedianStyle','line','Notch','off',
    'OutlierSize',6,...
    'Symbol','.r', Jitter=0, Labels=Lable_struc)
```

```matlab
xlabel('Simulations', Interpreter='latex')
ylabel('Time $[s]$', Interpreter='latex')
title('Box Plot w.r.t. Execution Time of QP Solver', Interpreter='latex')
box on
grid on

%% Quadratic Cost
COST_mat = cell2mat(COST);
COST_Q_mat = cell2mat(COST_Q);
COST_R_mat = cell2mat(COST_R);
COST_S_mat = cell2mat(COST_S);

sprintf('Mean Total Cost: %f',mean(COST_mat))
sprintf('Mean Total Cost - Q: %f',mean(COST_Q_mat))
sprintf('Mean Total Cost - S: %f',mean(COST_S_mat))
sprintf('Mean Total Cost - R: %f',mean(COST_R_mat))

%%
for i = 1:nsim
    i
    sum(TIME_ARRAY{i})
end
```

I.3.4 Plot_Data_Visualization_Linear_MPC_V2.m

```matlab
%% Control of Two-Tank Using Linear MPC.
% Linearized model about the nominal solution {h1 = 0.5, h2 = 0.3,
    u_PA001 = 0.8}

% NOTE: This File is used for testing and for visualization of data

% Author: Gent Luta

% Date: Spring 2023

%% Steady State Values
h1_A = 0.5;
h2_A = 0.3;
u_PA001_A = 0.8;
u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);
f1_A = ValveChar(u1_A);
f2_A = ValveChar(u2_A);

%% System parameters
rho = 1000;
```

```
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

h1_max = 1;
h1_min = 0.13;
h2_max = 0.4;
h2_min = 0.02;
Kv_LV001   = 11.25;
Kv_LV002   = 11.25;
z_LV001 = 0:0.05:1;
f_LV001 = (exp(z_LV001.^1.2)-1)/(exp(1)-1);

z_LV002 = 0:0.05:1;
f_LV002 = f_LV001;
u_PA001_data = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
              0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
              8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s

%% Function Handles
f1_handle = @ValveChar;
f2_handle = @ValveChar;
f3_handle = @PumpChar;

delta = 0.01;   %Step size (numerical difference)

%% Linearization
% A - Matrix
a11 = - (sqrt(100000))/(7.2*10^8) * ((Kv1*f1_A*rho*g) / (A1*sqrt(rho*g*(
    h1_A + h_LV001))));
a12 = 0;
a21 = ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*10^8)) * ((Kv1*
    f1_A*rho*g) / (sqrt(rho*g*(h1_A + h_LV001))));
a22 = - ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*(10^8))) * ((Kv2
    *f2_A*rho*g) / (sqrt(rho*g*(h2_A + h_LV002))));
A = [a11 a12; a21 a22];

% B - Matrix
b11 = - ((Kv1)/(3600*A1)) * sqrt((rho*g*(h1_A + h_LV001))/100000) *
    forward_diff(f1_handle, u1_A, delta);
b12 = 0;
b21 = ((1)/(0.004 + 0.07*h2_A)) * (Kv1/3600) * sqrt((rho*g*(h1_A +
    h_LV001))/100000)*forward_diff(f1_handle, u1_A, delta);
```

```
b22 = -((1)/(0.004 + 0.07*h2_A)) * (Kv2/3600) * sqrt((rho*g*(h2_A +
    h_LV002))/100000)*forward_diff(f2_handle, u2_A, delta);
B = [b11 b12 ; b21 b22];

% C - Matrix
C = eye(2);

% D - Matrix
D = zeros(2);

% G - Matrix (Disturbance)
g11 = forward_diff(f3_handle, u_PA001_A, delta)/A1;
g21 = 0;
G = [g11; g21];

% B_a - Matrix (Augmented)
B_a = [B G];

% D_a - Matrix (Augmented)
D_a = zeros(2, 3);

%% State-Space-Model
ts = 0.5;                    % Sampling time
sys = ss(A, B_a, C, D_a);    % CT State-Space Model
sys = c2d(sys,ts);           % DT State-Space Model (ts sampling)
% Signal Names
sys.InputName = {'u1', 'u2', 'u_PA001'};
sys.OutputName = {'h1', 'h2'};
sys.StateName = {'h1', 'h2'};
% Signal Units
sys.InputUnit = {'-', '-', '-'};
sys.OutputUnit = {'m', 'm'};
sys.StateUnit = {'m', 'm'};
% Signal Types
sys = setmpcsignals(sys, 'MV', [1 2], 'MD', 3, 'MO', [1 2]);
% Check Controllability
controllability_matrix = ctrb(sys);
controllability_matrix_rank = rank(controllability_matrix);
%% Linear MPC
old_status = mpcverbosity('on');
Duration = 400;
t = 0:ts:Duration;
N = length(t);

nsim = 10;
COST = cell(1,nsim);
COST_Q = cell(1,nsim);
COST_S = cell(1,nsim);
```

```
COST_R = cell(1,nsim);
TIME_ARRAY = cell(1,nsim);
ITERATIONS = cell(1,nsim);

for j = 1:nsim
p = 13;          % Prediction Horizon
c = 13;         % Control Horizon
%c = [2 4 7];     % Control Horizon (Blocking)

mpcobj = mpc(sys, ts, p, c);    % Linear MPC object

% Nominal Values
x0 = [h1_A; h2_A];
u0 = [u1_A;u2_A;u_PA001_A];
y0 = x0;
mpcobj.Model.Nominal = struct('X',x0,'U',u0,'Y',y0);

% Set Estimator (None)
setEstimator(mpcobj, 'custom'); %N
%setoutdist(mpcobj,'integrators')

% Signal Scaling
mpcobj.OV(1).ScaleFactor = 1 - 0.13;    % Range of h1
mpcobj.OV(2).ScaleFactor = 0.4 - 0.02;  % Range of h2

% Weighting Matrices
mpcobj.Weights.OutputVariables = [1 1];               % Q
mpcobj.Weights.ManipulatedVariablesRate = [0.1 0.1];  % S
mpcobj.Weights.ManipulatedVariables = [0 0];          % R

mpcobj.MV(1).Max = 0.9999;
mpcobj.MV(1).Min = 0.0001;
mpcobj.MV(2).Max = 0.9999;
mpcobj.MV(2).Min = 0.0001;

% MPC Optimizer Options:
% Interior-Point Solver and Option
mpcobj.Optimizer.Algorithm = 'interior-point';
% mpcobj.Optimizer.InteriorPointOptions.MaxIterations = 8;
% mpcobj.Optimizer.InteriorPointOptions.ConstraintTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.OptimalityTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.ComplementarityTolerance = 1e-6;
% mpcobj.Optimizer.InteriorPointOptions.StepTolerance = 1e-7;

% Active-Set Solver and Options
% mpcobj.Optimizer.Algorithm = 'active-set';
% mpcobj.Optimizer.ActiveSetOptions.MaxIterations = 2;
% mpcobj.Optimizer.ActiveSetOptions.ConstraintTolerance = 1e-6;

% mpcobj.Optimizer.UseSuboptimalSolution = true;

%% SIM

% Signal Previewing
ref = [ ones(1, N-1)*h1_A;
        ones(1, N-1)*h2_A];
u_PA001 = ones(1,N)*u_PA001_A;
% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 150;
t_upa_rise = 250;
t_hold = 70;
idx_ref1 = round(t_h1_rise/ts);
idx_ref2 = round(t_h2_rise/ts);
idx_upa = round(t_upa_rise/ts);
idx_hold = round(t_hold/ts);

ref(1,idx_ref1:idx_ref1+idx_hold) = h1_A + 0.2;
ref(2,idx_ref2:idx_ref2+idx_hold) = h2_A - 0.2;
u_PA001(1, idx_upa:idx_upa+idx_hold) = u_PA001_A - 0.2;

%
x = x0;                          % Current State
xc = mpcstate(mpcobj);

UU = zeros(2,N); UU(:,1) = u0(1:2);
XX_NL = zeros(2,N); XX_NL(:,1) = x0;
TIME_SPENT_LIST = zeros(1,N-1);
II = zeros(1,N-1);

hbar = waitbar(0, 'Simulation Progress');
for i = 1:(Duration/ts)
    if i <= N-p-1
        r1 = ref(1,i:i+p-1);
        r2 = ref(2,i:i+p-1);
        r = [r1' r2'];
        v = u_PA001(i:i+p-1)';
    else
        r1 = ref(1,i:end);
        r2 = ref(2,i:end);
        r = [r1' r2'];
        v = u_PA001(i:end)';
    end
    %r = ref(:,i)';      %Without preview (reference)
    %v = u_PA001(i);     %Without preview (Disturbance)

    % Find optimal mv
    tic
    [mv, info] = mpcmove(mpcobj, xc, [], r, v);
    TIME_SPENT_LIST(1,i) = toc;


    % Simulate Linear System
    x = sys.A*(x-x0) + sys.B*([mv;v(1)]-u0);
    x = round(x + x0, 5);

    % Simulate Nonlinear System
    [~,XNL] = ode45(@(t,x)tankCT_NEW(x, [mv;v(1)]), [0 ts], XX_NL(:,i));
    xnl = [XNL(end,1);XNL(end,2)];
    %xnl = round(xnl, 5);


    xc.Plant = xnl; %N
    xc.LastMove = mv; %N

    % History tracking
    UU(:,i+1) = mv;
    II(1,i) = info.Iterations;
    XX_NL(:,i+1) = xnl;

    waitbar(i*ts/Duration, hbar);
end
close(hbar)

TIME_ARRAY{1,j} = TIME_SPENT_LIST;
ITERATIONS{1,j} = II;
cost = 0;
cost_Q = 0;
cost_S = 0;
cost_R = 0;

Q = diag(mpcobj.Weights.OutputVariables);
Q(1,1) = Q(1,1)/mpcobj.OV(1).ScaleFactor;   % Scaling
Q(2,2) = Q(2,2)/mpcobj.OV(2).ScaleFactor;   % Scaling
S = diag(mpcobj.Weights.ManipulatedVariablesRate);
R = diag(mpcobj.Weights.ManipulatedVariables);
E = XX_NL(:,2:end) - ref;
EU = UU(:,:) - u0(1:2,:);
mvRate = UU(:,1:end-1) - UU(:,2:end);
for l = 1:N-1
    cost_Q = cost_Q + E(:,l)'*(Q^2)*E(:,l);
    cost_S = cost_S + mvRate(:,l)'*(S^2)*mvRate(:,l);
    cost_R = cost_R + EU(:,l)'*(R^2)*EU(:,l);
    cost = cost +  E(:,l)'*(Q^2)*E(:,l) + mvRate(:,l)'*(S^2)*mvRate(:,l)
   + EU(:,l)'*(R^2)*EU(:,l);
end
COST{1,j} = cost;
COST_Q{1,j} = cost_Q;
COST_S{1,j} = cost_S;
COST_R{1,j} = cost_R;
end

%% Plot - Simulation
figure
subplot(3,1,1)

plot(t, XX_NL(1,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(1,:), 'red', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
legend('$h_{1}$','$h_{1,\,ref}$', 'Interpreter','latex')
ylim([0 1]);
grid on
box on
set(gca,'YTick',0:0.05:1)

subplot(3,1,2)

plot(t, XX_NL(2,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(2,:), 'red', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
legend('$h_{2}$','$h_{2,\,ref}$', 'Interpreter','latex')
ylim([0 1])
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1)

subplot(3,1,3)
stairs(t, UU(1,:), 'magenta', 'LineWidth',1)
hold on
stairs(t, UU(2,:), 'green', 'LineWidth',1)
hold on
stairs(t, u_PA001, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('[-]', Interpreter='latex')
```

```matlab
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    )
ylim([0 1])
grid on
box on

%% Iterations
figure
histogram(ITERATIONS{1,2},EdgeColor='k',EdgeAlpha=1, FaceAlpha=1,
    FaceColor=[0.6350 0.0780 0.1840],...
    LineStyle='-', LineWidth=1)
xlabel('Necessary Iterations', Interpreter='latex')
ylabel('Number of QP Solver Calls', Interpreter='latex')
title('Simulation with Active-Set Algorithm', Interpreter='latex')
grid on
box on
%% Time Spent
TIME_MAT = zeros(N-1, nsim);
Lable_struc = cell(1,nsim);
for i = 1:nsim
    TIME_MAT(:,i) = TIME_ARRAY{1,i}';
    Lable_struc{i} = sprintf('Sim%i', i);
end
figure;
boxplot(TIME_MAT,"BoxStyle","outline",'MedianStyle','line','Notch','off',
    'OutlierSize',6,...
    'Symbol','.r', Jitter=0, Labels=Lable_struc)
xlabel('Simulations', Interpreter='latex')
ylabel('Time $[s]$', Interpreter='latex')
title('Box Plot w.r.t. Execution Time of QP Solver', Interpreter='latex')
box on
grid on
%% Quadratic Cost
COST_mat = cell2mat(COST);
COST_Q_mat = cell2mat(COST_Q);
COST_R_mat = cell2mat(COST_R);
COST_S_mat = cell2mat(COST_S);

sprintf('Mean Total Cost: %f',mean(COST_mat))
sprintf('Mean Total Cost - Q: %f',mean(COST_Q_mat))
sprintf('Mean Total Cost - S: %f',mean(COST_S_mat))
sprintf('Mean Total Cost - R: %f',mean(COST_R_mat))

%% Show Iterations and time taken on every control interval. Fill out
    this part!
figure
f = zeros(1,3);
```

```
yyaxis left
f(1) = stem(TIME_ARRAY{1,9},'blue', 'LineWidth',.05, 'LineStyle','-',
    MarkerSize=0.001);
ylabel('Time $[s]$',Interpreter='latex')

hold on
f(2) = yline(2.02*10^-3, 'k', 'LineWidth',1, 'LineStyle','--');
%ylim([0 0.004])
yyaxis right

f(3) = stem(ITERATIONS{1,9},'Color',[255/255,69/255,0], 'LineWidth',.05,
    'LineStyle','-', MarkerSize=0.001);
ylabel('Iterations',Interpreter='latex')
set(gca,'YTick',0:2:40)
ylim([0 40])

ax = gca;
ax.YAxis(1).Color = 'b';
ax.YAxis(2).Color = [255/255,69/255,0];
ax.XAxis(1).Color = 'k';

title('Execution Time and Iterations of Every QP Solver Call',Interpreter
    ='latex')
xlabel('Control Interval $k$',Interpreter='latex')
legend(f(1:3),'Time Spent', 'Upper Adjacent', 'Iterations Required', '
    Interpreter','latex')
grid on
box on

%%
```

I.3.5 Plot_Data_Visualization_Linear_MPC.m

```
%% Control of Two-Tank Using Linear MPC.
% Linearized model about the nominal solution {h1 = 0.5, h2 = 0.3,
    u_PA001 = 0.8}

% NOTE: This File is used for testing and for visualization of data

% Author: Gent Luta

% Date: Spring 2023

%% Steady State Values
h1_A = 0.5;
h2_A = 0.3;
```

```
u_PA001_A = 0.8;
u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);
f1_A = ValveChar(u1_A);
f2_A = ValveChar(u2_A);

%% System parameters
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

h1_max = 1;
h1_min = 0.13;
h2_max = 0.4;
h2_min = 0.02;
Kv_LV001    = 11.25;
Kv_LV002    = 11.25;
z_LV001 = 0:0.05:1;
f_LV001 = (exp(z_LV001.^1.2)-1)/(exp(1)-1);

z_LV002 = 0:0.05:1;
f_LV002 = f_LV001;
u_PA001_data = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
           0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
           8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s

%% Function Handles
f1_handle = @ValveChar;
f2_handle = @ValveChar;
f3_handle = @PumpChar;

delta = 0.01;   %Step size (numerical difference)

%% Linearization
% A - Matrix
a11 = - (sqrt(100000))/(7.2*10^8) * ((Kv1*f1_A*rho*g) / (A1*sqrt(rho*g*(
    h1_A + h_LV001))));
a12 = 0;
a21 = ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*10^8)) * ((Kv1*
    f1_A*rho*g) / (sqrt(rho*g*(h1_A + h_LV001))));
a22 = - ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*(10^8))) * ((Kv2
    *f2_A*rho*g) / (sqrt(rho*g*(h2_A + h_LV002))));
```

```matlab
A = [a11 a12; a21 a22];

% B - Matrix
b11 = - ((Kv1)/(3600*A1)) * sqrt((rho*g*(h1_A + h_LV001))/100000) *
    forward_diff(f1_handle, u1_A, delta);
b12 = 0;
b21 = ((1)/(0.004 + 0.07*h2_A)) * (Kv1/3600) * sqrt((rho*g*(h1_A +
    h_LV001))/100000)*forward_diff(f1_handle, u1_A, delta);
b22 = -((1)/(0.004 + 0.07*h2_A)) * (Kv2/3600) * sqrt((rho*g*(h2_A +
    h_LV002))/100000)*forward_diff(f2_handle, u2_A, delta);
B = [b11 b12 ; b21 b22];

% C - Matrix
C = eye(2);

% D - Matrix
D = zeros(2);

% G - Matrix (Disturbance)
g11 = forward_diff(f3_handle, u_PA001_A, delta)/A1;
g21 = 0;
G = [g11; g21];

% B_a - Matrix (Augmented)
B_a = [B G];

% D_a - Matrix (Augmented)
D_a = zeros(2, 3);

%% State-Space-Model
ts = 0.5;                       % Sampling time
sys = ss(A, B_a, C, D_a);       % CT State-Space Model
sys = c2d(sys,ts);              % DT State-Space Model (ts sampling)
% Signal Names
sys.InputName = {'u1', 'u2', 'u_PA001'};
sys.OutputName = {'h1', 'h2'};
sys.StateName = {'h1', 'h2'};
% Signal Units
sys.InputUnit = {'-', '-', '-'};
sys.OutputUnit = {'m', 'm'};
sys.StateUnit = {'m', 'm'};
% Signal Types
sys = setmpcsignals(sys, 'MV', [1 2], 'MD', 3, 'MO', [1 2]);
% Check Controllability
controllability_matrix = ctrb(sys);
controllability_matrix_rank = rank(controllability_matrix);
%% Linear MPC
old_status = mpcverbosity('on');
```

```
P = 20;
M = 20;
COST_CELL = cell(P,P);
TIME_CELL = cell(P,P);
TOTAL_TIME_CELL = cell(P,P);
for p = 1:P
    for m = 1:M
        if m > p
            COST_CELL{p,m} = nan;
            TIME_CELL{p,m} = nan;
            TOTAL_TIME_CELL{p,m} = nan;
        else
%p = 13;         % Prediction Horizon
%c = 13;        % Control Horizon
%c = [2 4 7];    % Control Horizon (Blocking)

mpcobj = mpc(sys, ts, p, m);    % Linear MPC object

% Nominal Values
x0 = [h1_A; h2_A];
u0 = [u1_A;u2_A;u_PA001_A];
y0 = x0;
mpcobj.Model.Nominal = struct('X',x0,'U',u0,'Y',y0);

% Set Estimator (None)
setEstimator(mpcobj, 'custom'); %N
%setoutdist(mpcobj,'integrators')

% Signal Scaling
mpcobj.OV(1).ScaleFactor = 1 - 0.13;    % Range of h1
mpcobj.OV(2).ScaleFactor = 0.4 - 0.02;  % Range of h2

% Weighting Matrices
mpcobj.Weights.OutputVariables = [1 1];               % Q
mpcobj.Weights.ManipulatedVariablesRate = [0.1 0.1];  % S
mpcobj.Weights.ManipulatedVariables = [0 0];          % R
% Output (soft) Constraints
%mpcobj.OV(1).Max = 1;
%mpcobj.OV(1).Min = 0.13;
%mpcobj.OV(2).Max = 0.4;
%mpcobj.OV(2).Min = 0.02;
% Output ECR (slack)
%mpcobj.OV(1).MinECR = 5;
%mpcobj.OV(1).MaxECR = 5;
%mpcobj.OV(2).MinECR = 5;
%mpcobj.OV(2).MaxECR = 5;
% % Input (hard) Constraints
mpcobj.MV(1).Max = 0.9999;
mpcobj.MV(1).Min = 0.0001;
mpcobj.MV(2).Max = 0.9999;
mpcobj.MV(2).Min = 0.0001;
% Input Rate (soft) Constraints
% mpcobj.MV(1).RateMin = -0.3;
% mpcobj.MV(1).RateMax = 0.3;
% mpcobj.MV(2).RateMin = -0.3;
% mpcobj.MV(2).RateMax = 0.3;
% Input Rate ECR (slack)
% mpcobj.MV(1).RateMinECR = 20;
% mpcobj.MV(1).RateMaxECR = 20;
% mpcobj.MV(2).RateMinECR = 20;
% mpcobj.MV(2).RateMaxECR = 20;

% MPC Optimizer Options:
% Interior-Point Solver and Option
% mpcobj.Optimizer.Algorithm = 'interior-point';
% mpcobj.Optimizer.InteriorPointOptions.MaxIterations = 8;
% mpcobj.Optimizer.InteriorPointOptions.ConstraintTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.OptimalityTolerance = 1e-5;
% mpcobj.Optimizer.InteriorPointOptions.ComplementarityTolerance = 1e-6;
% mpcobj.Optimizer.InteriorPointOptions.StepTolerance = 1e-7;

% Active-Set Solver and Options
% mpcobj.Optimizer.Algorithm = 'active-set';
% mpcobj.Optimizer.ActiveSetOptions.MaxIterations = 2;
% mpcobj.Optimizer.ActiveSetOptions.ConstraintTolerance = 1e-6;

% mpcobj.Optimizer.UseSuboptimalSolution = true;

%% SIM
Duration = 400;
t = 0:ts:Duration;
N = length(t);

% Signal Previewing
ref = [ ones(1, N-1)*h1_A;
        ones(1, N-1)*h2_A];
u_PA001 = ones(1,N)*u_PA001_A;
% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 150;
t_upa_rise = 250;
t_hold = 70;
idx_ref1 = round(t_h1_rise/ts);
idx_ref2 = round(t_h2_rise/ts);
idx_upa = round(t_upa_rise/ts);
idx_hold = round(t_hold/ts);
```

```

ref(1,idx_ref1:idx_ref1+idx_hold) = h1_A + 0.2;
ref(2,idx_ref2:idx_ref2+idx_hold) = h2_A - 0.2;
u_PA001(1, idx_upa:idx_upa+idx_hold) = u_PA001_A - 0.2;

%
x = x0;                           % Current State
xc = mpcstate(mpcobj);

nsim = 1;   % # of Simulations

XX = zeros(2,N); XX(:,1) = x0;
UU = zeros(2,N); UU(:,1) = u0(1:2);
MPCXX = zeros(2,N); MPCXX(:,1) = xc.Plant;
II = zeros(1,N-1, nsim);
XX_NL = zeros(2,N); XX_NL(:,1) = x0;
TIME_SPENT = zeros(1,N-1, nsim);

TIME_SPENT_AVERAGES = zeros(1,nsim);
COST = zeros(1,nsim);
COST_Q = zeros(1,nsim);
COST_S = zeros(1,nsim);
COST_R = zeros(1,nsim);

for j = 1:nsim
hbar = waitbar(0, 'Simulation Progress');
for i = 1:(Duration/ts)
    if i <= N-p-1
        r1 = ref(1,i:i+p-1);
        r2 = ref(2,i:i+p-1);
        r = [r1' r2'];
        v = u_PA001(i:i+p-1)';
    else
        r1 = ref(1,i:end);
        r2 = ref(2,i:end);
        r = [r1' r2'];
        v = u_PA001(i:end)';
    end
    %r = ref(:,i)';     %Without preview (reference)
    %v = u_PA001(i);    %Without preview (Disturbance)

    % Find optimal mv
    tic
    [mv, info] = mpcmove(mpcobj, xc, [], r, v);
    TIME_SPENT(1,i,j) = toc;


    % Simulate Linear System
    x = sys.A*(x-x0) + sys.B*([mv;v(1)]-u0);
    x = round(x + x0, 5);

    % Simulate Nonlinear System
    [~,XNL] = ode45(@(t,x)tankCT_NEW(x, [mv;v(1)]), [0 ts], XX_NL(:,i));
    xnl = [XNL(end,1);XNL(end,2)];
    %xnl = round(xnl, 5);


    xc.Plant = xnl; %N
    xc.LastMove = mv; %N

    % History tracking
    XX(:,i+1) = x;
    UU(:,i+1) = mv;
    MPCXX(:,i+1) = xc.Plant;
    II(1,i,j) = info.Iterations;
    XX_NL(:,i+1) = xnl;

    waitbar(i*ts/Duration, hbar);
end
close(hbar)
TIME_SPENT_AVERAGES(j) = mean(TIME_SPENT(:,:,j));

cost = 0;
cost_Q = 0;
cost_S = 0;
cost_R = 0;

Q = diag(mpcobj.Weights.OutputVariables);
Q(1,1) = Q(1,1)/mpcobj.OV(1).ScaleFactor;   % Scaling
Q(2,2) = Q(2,2)/mpcobj.OV(2).ScaleFactor;   % Scaling
S = diag(mpcobj.Weights.ManipulatedVariablesRate);
R = diag(mpcobj.Weights.ManipulatedVariables);
E = XX_NL(:,2:end) - ref;
EU = UU(:,:) - u0(1:2,:);
mvRate = UU(:,1:end-1) - UU(:,2:end);
for l = 1:N-1
    cost_Q = cost_Q + E(:,l)'*(Q^2)*E(:,l);
    cost_S = cost_S + mvRate(:,l)'*(S^2)*mvRate(:,l);
    cost_R = cost_R + EU(:,l)'*(R^2)*EU(:,l);
    cost = cost +  E(:,l)'*(Q^2)*E(:,l) + mvRate(:,l)'*(S^2)*mvRate(:,l)
   + EU(:,l)'*(R^2)*EU(:,l);
end
COST(j) = cost;
COST_Q(j) = cost_Q;
COST_S(j) = cost_S;
COST_R(j) = cost_R;
```

```matlab
end
COST_CELL{p,m} = cost;
TIME_CELL{p,m} = TIME_SPENT_AVERAGES;
TOTAL_TIME_CELL{p,m} = sum(TIME_SPENT(:,:,1));
        end
    end
end
%% Heatmap
Phm = 1:P;
Mhm = 1:M;
figure
COST_CELL_MAT = cell2mat(COST_CELL);
COST_CELL_MAT(:,1:7) = nan; % Use to remove Columns!
COST_CELL_MAT(1:9,:) = nan; % Use to remove Rows!
heatmap(Phm, Mhm, COST_CELL_MAT,"Colormap",jet)
xlabel('M')
ylabel('P')
title('Quadratic Cost v. Horizon Lengths')
[min_val,idx]=min(COST_CELL_MAT(:));
[row,col]=ind2sub(size(COST_CELL_MAT),idx);
sprintf('Best (P,M): (%f,%f) w/ Cost: %f',row, col, COST_CELL_MAT(row,col
    ))

[max_val,idx]=max(COST_CELL_MAT(:));
[row,col]=ind2sub(size(COST_CELL_MAT),idx);
sprintf('Worst (P,M): (%f,%f) w/ Cost: %f',row, col, COST_CELL_MAT(row,
    col))

Phm = 1:P;
Mhm = 1:M;
figure
TIME_CELL_MAT = cell2mat(TIME_CELL);
%TIME_CELL_MAT(:,40:-1:20) = nan; % Use to remove Columns!
%TIME_CELL_MAT(1:7,:) = nan; % Use to remove Rows!
heatmap(Phm, Mhm, TIME_CELL_MAT,"Colormap",jet)
xlabel('M')
ylabel('P')
title('Average Execution Time v. Horizon Lengths')

[min_val,idx]=min(TIME_CELL_MAT(:));
[row,col]=ind2sub(size(TIME_CELL_MAT),idx);
sprintf('Best (P,M): (%f,%f) w/ Time: %f',row, col, TIME_CELL_MAT(row,col
    ))

[max_val,idx]=max(TIME_CELL_MAT(:));
[row,col]=ind2sub(size(TIME_CELL_MAT),idx);
sprintf('Worst (P,M): (%f,%f) w/ Time: %f',row, col, TIME_CELL_MAT(row,
    col))
```

```matlab

Phm = 1:P;
Mhm = 1:M;
figure
TOTAL_TIME_CELL_MAT = cell2mat(TOTAL_TIME_CELL);
%TOTAL_TIME_CELL_MAT(:,1:7) = nan; % Use to remove Columns!
%TOTAL_TIME_CELL_MAT(1:7,:) = nan; % Use to remove Rows!

heatmap(Phm, Mhm, TOTAL_TIME_CELL_MAT,"Colormap",jet)
xlabel('M')
ylabel('P')
title('Total Execution Time v. Horizon Lengths')

[min_val,idx]=min(TOTAL_TIME_CELL_MAT(:));
[row,col]=ind2sub(size(TOTAL_TIME_CELL_MAT),idx);
sprintf('Best (P,M): (%f,%f) w/ Time: %f',row, col, TOTAL_TIME_CELL_MAT(
    row,col))

[max_val,idx]=max(TOTAL_TIME_CELL_MAT(:));
[row,col]=ind2sub(size(TOTAL_TIME_CELL_MAT),idx);
sprintf('Worst (P,M): (%f,%f) w/ Time: %f',row, col, TOTAL_TIME_CELL_MAT(
    row,col))


%%
sprintf('Execution Time Weighted Average: %e',mean(TIME_SPENT_AVERAGES))
sprintf('Execution Time Standard Deviation: %e',std(TIME_SPENT_AVERAGES))
sprintf('Cost Weighted Average: %e',mean(COST))
sprintf('Cost Standard Deviation: %e',std(COST))

% Plot - Simulation
figure
subplot(3,1,1)
%plot(t, XX(1,:),'green', 'LineWidth',1)
%hold on
plot(t, XX_NL(1,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(1,:), 'red', 'LineWidth',1, 'LineStyle','--')
%hold on
%plot(t, MPCXX(1,:), 'cyan', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
%legend('$h_{1,\,L}$','$h_{1,\,NL}$','$h_{1,\,ref}$', 'Interpreter','
    latex')
legend('$h_{1}$','$h_{1,\,ref}$', 'Interpreter','latex')
ylim([0 1]);
grid on
```

```
box on

subplot(3,1,2)
%plot(t, XX(2,:), 'green', 'LineWidth',1)
%hold on
plot(t, XX_NL(2,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(2,:), 'red', 'LineWidth',1, 'LineStyle','--')
%hold on
%plot(t, MPCXX(2,:), 'cyan', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
%legend('$h_{2,\,L}$','$h_{2,\,NL}$','$h_{2,\,ref}$', 'Interpreter','
    latex')
legend('$h_{2}$','$h_{2,\,ref}$', 'Interpreter','latex')
ylim([0 1])
xlim([0 t(end)])
grid on
box on

subplot(3,1,3)
stairs(t, UU(1,:), 'magenta', 'LineWidth',1)
hold on
stairs(t, UU(2,:), 'green', 'LineWidth',1)
hold on
stairs(t, u_PA001, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('[-]', Interpreter='latex')
title('Inputs', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    )
ylim([0 1])
grid on
box on

%% Iterations
figure
histogram(II(:,:,1),EdgeColor='k',EdgeAlpha=1, FaceAlpha=1,FaceColor
    =[0.6350 0.0780 0.1840],...
     LineStyle='-', LineWidth=1)
xlabel('Necessary Iterations', Interpreter='latex')
ylabel('Number of QP Solver Calls', Interpreter='latex')
title('Simulation with Active-Set Algorithm', Interpreter='latex')
grid on
box on
%% Time Spent
TIME_MAT = zeros(N-1, nsim);
```

```
Lable_struc = cell(1,nsim);
for i = 1:nsim
    TIME_MAT(:,i) = TIME_SPENT(:,:,i)';
    Lable_struc{i} = sprintf('Sim%i', i);
end
    figure
boxplot(TIME_MAT,"BoxStyle","outline",'MedianStyle','line','Notch','off',
    'OutlierSize',6,...
    'Symbol','.r', Jitter=0, Labels=Lable_struc)
xlabel('Simulations', Interpreter='latex')
ylabel('Time $[s]$', Interpreter='latex')
title('Box Plot w.r.t. Execution Time of QP Solver', Interpreter='latex')
box on
grid on
```

I.3.6 Plot_Data_Visualization_Nonlinear_MPC.m

```
%% Control of Two-Tank Using Nonlinear MPC.
% Initial condition: {h1 = 0.5, h2 = 0.3, u_PA001 = 0.8}

% Author: Gent Luta

% Date: Spring 2023

%% Steady State Values
h1_A = 0.5;
h2_A = 0.3;
u_PA001_A = 0.8;

u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);

%% Initial Condition
x0 = [h1_A; h2_A];
u0 = [u1_A;u2_A;u_PA001_A];
y0 = x0;

%% Nonlinear MPC
ts = 0.5;
Duration = 400;
t = 0:ts:Duration;
N = length(t);

nsim = 1;
COST = cell(1,nsim);
COST_Q = cell(1,nsim);
```

```
COST_S = cell(1,nsim);
COST_R = cell(1,nsim);
TIME_ARRAY = cell(1,nsim);
ITERATIONS = cell(1,nsim);

for j = 1:nsim
nx = 2;             % Number of states
ny = 2;             % Number of outputs
mvIndex = [1,2];    % Manipulated variable indices
mdIndex = 3;        % Measured disturbance indices
nlobj = nlmpc(nx,ny,'MV',mvIndex,'MD',mdIndex); % Nonlinear MPC object

ts = 0.5;    % Sampling time
nlobj.Ts = ts;  % Sampling time

% Horizon Lengths
p = 13; % Prediction horizon
m = 13;  % Control horizon
nlobj.PredictionHorizon = p;
nlobj.ControlHorizon = m;

% Nonlinear DT State Function
nlobj.Model.StateFcn = 'tankDT_NEW_One_Step';
nlobj.Model.IsContinuousTime = false;
nlobj.Model.NumberOfParameters = 1;

% DT Output Function
nlobj.Model.OutputFcn = 'tankOutputFcn';

% Output Jacobian (I.e., C - Matrix)
nlobj.Jacobian.OutputFcn = @(x,u,Ts) [1 0; 0 1];

% Weighting Matrices
nlobj.Weights.OutputVariables = [1 1];                % Q
nlobj.Weights.ManipulatedVariablesRate = [0.1 0.1]; % S
nlobj.Weights.ManipulatedVariables = [0 0];          % R

% Signal Scaling
nlobj.OV(1).ScaleFactor = 1 - 0.13;     % Range of h1
nlobj.OV(2).ScaleFactor = 0.4 - 0.02;   % Range of h2

% Input (hard) Constraints
nlobj.MV(1).Max = 0.9999;
nlobj.MV(1).Min = 0.0001;
nlobj.MV(2).Max = 0.9999;
nlobj.MV(2).Min = 0.0001;
%nlobj.Optimization.RunAsLinearMPC = 'Adaptive'; % REMOVE
```

```
nlobj.Optimization.SolverOptions.MaxIterations = 30;     % Limit SQP
    Solver to 30 Intervals
nlobj.Optimization.UseSuboptimalSolution = true;         % Toggle Sub-
    Optimal Solution

%% MPC Validation
validateFcns(nlobj,x0,u0(1:2),u0(3),{ts});
nloptions = nlmpcmoveopt;
nloptions.Parameters = {ts};

%% SIM

% Signal Previewing
ref = [ ones(1, N-1)*h1_A;
        ones(1, N-1)*h2_A];
u_PA001 = ones(1,N)*u_PA001_A;

% Reference and Disturbance Modification
t_h1_rise = 50;
t_h2_rise = 150;
t_upa_rise = 250;
t_hold = 70;
idx_ref1 = round(t_h1_rise/ts);
idx_ref2 = round(t_h2_rise/ts);
idx_upa = round(t_upa_rise/ts);
idx_hold = round(t_hold/ts);
%
ref(1,idx_ref1:idx_ref1+idx_hold) = h1_A + 0.2;%0.2
ref(2,idx_ref2:idx_ref2+idx_hold) = h2_A - 0.2;%0.2
u_PA001(1, idx_upa:idx_upa+idx_hold) = u_PA001_A - 0.2;%0.2

%
x = x0;          % Current state
mv = u0(1:2);    % Last MV

% History Tracking
UU = zeros(2,N); UU(:,1) = u0(1:2);
II = zeros(1,N-1);
XX_NL = zeros(2,N); XX_NL(:,1) = x0;
TIME_SPENT = zeros(1,N-1);

hbar = waitbar(0, 'Simulation Progress');
for i = 1:(Duration/ts)
    if i <= N-p-1
        r1 = ref(1,i:i+p-1);
        r2 = ref(2,i:i+p-1);
        r = [r1' r2'];
        v = u_PA001(i:i+p-1)';
    else
        r1 = ref(1,i:end);
        r2 = ref(2,i:end);
        r = [r1' r2'];
        v = u_PA001(i:end)';
    end
    %r = ref(:,i)';     %Without signal preview (reference)
    %v = u_PA001(i);    %Without signal preview (disturbance)

    tic
    [mv, nloptions, info] = nlmpcmove(nlobj,x,mv,r,v, nloptions); % Solve
     SQP optimization problem
    TIME_SPENT(i) = toc;

    % Simulate Nonlinear System
    [~,XNL] = ode45(@(t,x)tankCT_NEW(x, [mv;v(1)]), [0 ts], XX_NL(:,i));
    xnl = [XNL(end,1);XNL(end,2)];
    x = xnl;

    %  History tracking
    UU(:,i+1) = mv;
    II(i) = info.Iterations;
    XX_NL(:,i+1) = xnl;

    waitbar(i*ts/Duration, hbar);
end
close(hbar)

TIME_ARRAY{1,j} = TIME_SPENT;
ITERATIONS{1,j} = II;

cost = 0;
cost_Q = 0;
cost_S = 0;
cost_R = 0;


% Find Total Quadratic Cost
Q = diag(nlobj.Weights.OutputVariables);
Q(1,1) = Q(1,1)/nlobj.OV(1).ScaleFactor;   % Scaling
Q(2,2) = Q(2,2)/nlobj.OV(2).ScaleFactor;   % Scaling
S = diag(nlobj.Weights.ManipulatedVariablesRate);
R = diag(nlobj.Weights.ManipulatedVariables);
E = XX_NL(:,2:end) - ref;                  % OV reference tracking error
EU = UU(:,:) - u0(1:2,:);                  % MV target tracking error
mvRate = UU(:,1:end-1) - UU(:,2:end);      % MV rate
for l = 1:N-1
    cost_Q = cost_Q + E(:,l)'*(Q^2)*E(:,l);
    cost_S = cost_S + mvRate(:,l)'*(S^2)*mvRate(:,l);
    cost_R = cost_R + EU(:,l)'*(R^2)*EU(:,l);
    cost = cost +  E(:,l)'*(Q^2)*E(:,l) + mvRate(:,l)'*(S^2)*mvRate(:,l)
    + EU(:,l)'*(R^2)*EU(:,l);
end
COST{1,j} = cost;
COST_Q{1,j} = cost_Q;
COST_S{1,j} = cost_S;
COST_R{1,j} = cost_R;
end


%% Plot - Simulation
figure

% Tank 1
subplot(3,1,1)
plot(t, XX_NL(1,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(1,:), 'red', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
legend('$h_{1}$','$h_{1,\,ref}$', 'Interpreter','latex')
ylim([0 1]);
grid on
box on
set(gca,'YTick',0:0.05:1)

% Tank 2
subplot(3,1,2)
plot(t, XX_NL(2,:), 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t(2:end), ref(2,:), 'red', 'LineWidth',1, 'LineStyle','--')
xlabel('t\, [s]', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
legend('$h_{2}$','$h_{2,\,ref}$', 'Interpreter','latex')
ylim([0 1])
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1)

% Input Signals
subplot(3,1,3)
stairs(t, UU(1,:), 'magenta', 'LineWidth',1)
hold on
```

```
stairs(t, UU(2,:), 'green', 'LineWidth',1)
hold on
stairs(t, u_PA001, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('[-]', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    )
ylim([0 1])
grid on
box on

%% Time Spent
TIME_MAT = zeros(N-1, nsim);
Lable_struc = cell(1,nsim);
for i = 1:nsim
    TIME_MAT(:,i) = TIME_ARRAY{1,i}';
    Lable_struc{i} = sprintf('Sim%i', i);
end
figure;
boxplot(TIME_MAT,"BoxStyle","outline",'MedianStyle','line','Notch','off',
    'OutlierSize',6,...
    'Symbol','.r', Jitter=0, Labels=Lable_struc)
xlabel('Simulations', Interpreter='latex')
ylabel('Time $[s]$', Interpreter='latex')
title('Box Plot w.r.t. Execution Time of SQP Solver', Interpreter='latex'
    )
box on
grid on

%% Quadratic Cost
COST_mat = cell2mat(COST);
COST_Q_mat = cell2mat(COST_Q);
COST_R_mat = cell2mat(COST_R);
COST_S_mat = cell2mat(COST_S);

sprintf('Mean Total Cost: %f',mean(COST_mat))
sprintf('Mean Total Cost - Q: %f',mean(COST_Q_mat))
sprintf('Mean Total Cost - S: %f',mean(COST_S_mat))
sprintf('Mean Total Cost - R: %f',mean(COST_R_mat))

%% Show Iterations and time taken on every control interval. Fill out
    this part!
figure
yyaxis left
stem(TIME_ARRAY{1,1},'blue', 'LineWidth',.05, 'LineStyle','-',MarkerSize
    =0.001)
hold on
```

```
yline(20.77*10^-3, 'k', 'LineWidth',1, 'LineStyle','--')
hold on
yline(0.5, 'r', 'LineWidth',1, 'LineStyle','--')
ylabel('Time $[s]$',Interpreter='latex')
set(gca, 'YScale', 'log')

yyaxis right
stem(ITERATIONS{1,1},'Color',[255/255,69/255,0], 'LineWidth',.05, '
    LineStyle','-', MarkerSize=0.001);
ylabel('Iterations',Interpreter='latex')
%set(gca, 'YScale', 'log')
ylim([0 80])

ax = gca;
ax.YAxis(1).Color = 'b';
ax.YAxis(2).Color = [255/255,69/255,0];
ax.XAxis(1).Color = 'k';

title('Execution Time and Iterations of Every SQP Solver Call',
    Interpreter='latex')
xlabel('Control Interval $k$',Interpreter='latex')
legend('Time Spent', 'Upper Adjacent', 'Control Interval Length', '
    Iterations Required', 'Interpreter','latex')
grid on
box on

%%
```

I.3.7 Plot_Equal_Percentage_Valve_Char.m

```
%% Equal Percentage Valve Characteristic (E.P.V.C.)

% Author: Gent Luta

% Date: Spring 2023


%% E.P.V.C. with different rangeability (R)
z = 0:0.01:1;
R = [5, 10, 15, 20];
style = {'--', '-', ':', '-.'};
nr = 1;
figure
for r = R
    f = r.^(z-1);
    plot(z,f, LineWidth=1, Color='k', LineStyle=style(nr))
    nr = nr + 1;
    hold on
end
legend('R = 5', 'R = 10', 'R = 15','R = 20')
xlabel('Valve Opening $z(t)$', Interpreter='latex')
ylabel('Flow Rate $f(z(t))$', Interpreter='latex')
title('Equal Percentage Flow Characteristics', Interpreter='latex')
yticks([0,0.25, 0.5, 0.75, 1]);
yticklabels({'0%', '25%', '50%','75%', '100%'})
xticks([0,0.25, 0.5, 0.75, 1]);
xticklabels({'0%', '25%', '50%','75%', '100%'})
%line([0.6 0.6], [0 0.6], Linewidth = 2) Use when showing central
%difference.
box on
grid on

%% E.P.V.C. with R = 10 and Approximated V.C.
z = 0:0.01:1;
f = figure;
Real = 10.^(z-1);
Approx = (exp(z.^1.2) - 1) / (exp(1) - 1);
plot(z, Real, LineWidth=1, Color='k', LineStyle='--')
hold on
plot(z,Approx, LineWidth=1, Color='k', LineStyle='-')
box on
grid on
legend('Equal Percentage $R = 10$', 'Approximated Valve Char.', '
    Interpreter','latex')
yticks([0,0.25, 0.5, 0.75, 1]);
yticklabels({'0%', '25%', '50%','75%', '100%'})
xticks([0,0.25, 0.5, 0.75, 1]);
xticklabels({'0%', '25%', '50%','75%', '100%'})
xlabel('Valve Opening $z(t)$', Interpreter='latex')
ylabel('Flow Rate $f(z(t))$', Interpreter='latex')
title('Real and Approximated Flow Characteristics', Interpreter='latex')
exportgraphics(f,'ValveChar.png','Resolution',300)

%% Approximated V.C. and Forward difference
z = 0:0.01:1;
figure
Approx = (exp(z.^1.2) - 1) / (exp(1) - 1);
delta = 0.1;
point = 0.5;
Approx_Forward = (exp((point+delta)^1.2) - 1) / (exp(1) - 1);
Approx_Mid = (exp(point^1.2) - 1) / (exp(1) - 1);
Forward_Diff = (Approx_Forward - Approx_Mid)/(delta);
b = Approx_Mid - Forward_Diff*0.5;
Linear_line = Forward_Diff.*z + b;
```

```matlab
plot(z, Linear_line, LineWidth=1, Color='k', LineStyle=':')
hold on
plot(z,Approx, LineWidth=1, Color='k', LineStyle='-')
box on
grid on
ax = gca;
ax.TickLabelInterpreter = 'latex';
ax.GridColor = [0 0 0];
ax.GridLineStyle = '--';
legend('$\mathcal{L}_v$','Approximated Valve Char.', 'Interpreter','latex
    ')
yticks([Approx_Mid, Approx_Forward ]);
yticklabels({'$f(z(t))$', '$f(z(t) + h)$'})
xticks([point, point+delta]);
xticklabels({'$z(t)$', '$z(t) + h$'})
xlabel('Valve Opening $z(t)$', Interpreter='latex')
ylabel('Flow Rate $f(z(t))$', Interpreter='latex')
title('Forward Difference - Graphical Interpretation', Interpreter='latex
    ')
```

I.3.8 Plot_Experimental_Data_Delay_Noise.m

```matlab
%% Plot Two - Tank System Properties (Delay / Noise / Water Ripple)

% Author: Gent Luta

% Date: Spring 2023

% Load Scope Data File
load("Experimental_Folder_FINAL_DATA_080723\
    Experimental_Data_Dynamic_Noise.mat")

[upa, h1, ulv1, h2, ulv2] = ScopeData.signals.values;


% Index for 'Dynamic Noise', past 100 sec ->1430

t = ScopeData.time;

figure
% Tank 1
subplot(3,1,1)
plot(t, h1, 'blue', 'LineWidth',1, 'LineStyle','-')
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
```

```
legend('$h_{1}$', 'Interpreter','latex')
%ylim([0 1]);
xlim([100 200]);
grid on
box on
%set(gca,'YTick',0:0.05:1) % Use only when zooming in on Figure. Else,
    comment out

% Tank 2
subplot(3,1,2)
plot(t, h2, 'blue', 'LineWidth',1, 'LineStyle','-')
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
legend('$h_{2}$', 'Interpreter','latex')
%ylim([0 1])
xlim([100 200]);
grid on
box on
%set(gca,'YTick',0:0.05:1) % Use only when zooming in on Figure. Else,
    comment out

% Input Signals
subplot(3,1,3)
stairs(t, ulv1, 'magenta', 'LineWidth',1)
hold on
stairs(t, ulv2, 'green', 'LineWidth',1)
hold on
stairs(t, upa, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('$[-]$', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    );
%ylim([0 1])
xlim([100 200]);
grid on
box on
```

I.3.9 Plot_Experimental_Data_State_Estimation.m

```
%% Plot Experimental Data - With State Estimation

% Author: Gent Luta

% Date: Spring 2023
```

```matlab

% Load Scope Data File
%% case 1: 2x2 (estimation of 2 states using 2 measurements)
load("Simulink_mpc+mhe\Simulink_mpc+mhe\figures\scopedata_2x2_mpc+mhe.mat
    ")

% uPA
upa = ScopeData.signals(1).values(:,2); % Measured

% h1
h1_estimated = ScopeData.signals(2).values(:,1);
h1_real = ScopeData.signals(2).values(:,2);
h1_ref = ScopeData.signals(2).values(:,3);

% h2
h2_estimated = ScopeData.signals(3).values(:,1);
h2_real = ScopeData.signals(3).values(:,2);
h2_ref = ScopeData.signals(3).values(:,3);

% Input signals
ulv1 = ScopeData.signals(4).values(:,1);
ulv2 = ScopeData.signals(4).values(:,2);

% time
t = ScopeData.time;

figure
% Tank 1
subplot(3,1,1)
plot(t, h1_real, 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t, h1_ref, 'green', 'LineWidth',1, 'LineStyle','--')
hold on
plot(t, h1_estimated, 'red', 'LineWidth',1, 'LineStyle',':')
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
legend('$h_{1,\,measured}$','$h_{1,\,ref}$', '$h_{1,\,estimated}$', '
    Interpreter','latex')
ylim([0 1]);
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1) % Use only when zooming in on Figure. Else,
    comment out

% Tank 2
subplot(3,1,2)
```

```
plot(t, h2_real, 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t, h2_ref, 'green', 'LineWidth',1, 'LineStyle','--')
hold on
plot(t, h2_estimated, 'red', 'LineWidth',1, 'LineStyle',':')
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
legend('$h_{2,\,measured}$','$h_{2,\,ref}$', '$h_{2,\,estimated}$', '
    Interpreter','latex')
ylim([0 1])
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1) % Use only when zooming in on Figure. Else,
    comment out

% Input Signals
subplot(3,1,3)
stairs(t, ulv1, 'magenta', 'LineWidth',1)
hold on
stairs(t, ulv2, 'green', 'LineWidth',1)
hold on
stairs(t, upa, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('$[-]$', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    );
ylim([0 1])
xlim([0 t(end)])
grid on
box on




%% case 2: 2x1  (estimation of 2 states using 1 measurements)
load("Simulink_mpc+mhe\Simulink_mpc+mhe\figures\scopedata_2x1_mpc+mhe.mat
    ")

% uPA
upa = ScopeData.signals(1).values(:,2); % Measured

% h1
h1_estimated = ScopeData.signals(2).values(:,1);
h1_real = ScopeData.signals(2).values(:,2);
h1_ref = ScopeData.signals(2).values(:,3);
```

```matlab

% h2
h2_estimated = ScopeData.signals(3).values(:,1);
h2_real = ScopeData.signals(3).values(:,2);
h2_ref = ScopeData.signals(3).values(:,3);

% Input signals
ulv1 = ScopeData.signals(4).values(:,1);
ulv2 = ScopeData.signals(4).values(:,2);

% time
t = ScopeData.time;

figure
% Tank 1
subplot(3,1,1)
plot(t, h1_real, 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t, h1_ref, 'green', 'LineWidth',1, 'LineStyle','--')
hold on
plot(t, h1_estimated, 'red', 'LineWidth',1, 'LineStyle',':')
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
legend('$h_{1,\,measured}$','$h_{1,\,ref}$', '$h_{1,\,estimated}$', '
    Interpreter','latex')
ylim([0 1]);
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1) % Use only when zooming in on Figure. Else,
    comment out

% Tank 2
subplot(3,1,2)
plot(t, h2_real, 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t, h2_ref, 'green', 'LineWidth',1, 'LineStyle','--')
hold on
plot(t, h2_estimated, 'red', 'LineWidth',1, 'LineStyle',':')
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
legend('$h_{2,\,measured}$','$h_{2,\,ref}$', '$h_{2,\,estimated}$', '
    Interpreter','latex')
ylim([0 1])
xlim([0 t(end)])
grid on
```

```
box on
set(gca,'YTick',0:0.05:1) % Use only when zooming in on Figure. Else,
    comment out

% Input Signals
subplot(3,1,3)
stairs(t, ulv1, 'magenta', 'LineWidth',1)
hold on
stairs(t, ulv2, 'green', 'LineWidth',1)
hold on
stairs(t, upa, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('$[-]$', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    );
ylim([0 1])
xlim([0 t(end)])
grid on
box on






%% case 3: 3x2  (estimation of 3 states using 2 measurements)
load("Simulink_mpc+mhe\Simulink_mpc+mhe\figures\scopedata_3x2_mpc+mhe.mat
    ")

% uPA
upa = ScopeData.signals(1).values(:,2); % Measured
upa_estimate = ScopeData.signals(1).values(:,1);

% h1
h1_estimated = ScopeData.signals(2).values(:,1);
h1_real = ScopeData.signals(2).values(:,2);
h1_ref = ScopeData.signals(2).values(:,3);

% h2
h2_estimated = ScopeData.signals(3).values(:,1);
h2_real = ScopeData.signals(3).values(:,2);
h2_ref = ScopeData.signals(3).values(:,3);

% Input signals
ulv1 = ScopeData.signals(4).values(:,1);
ulv2 = ScopeData.signals(4).values(:,2);

```

```matlab
% time
t = ScopeData.time;

figure
% Tank 1
subplot(3,1,1)
plot(t, h1_real, 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t, h1_ref, 'green', 'LineWidth',1, 'LineStyle','--')
hold on
plot(t, h1_estimated, 'red', 'LineWidth',1, 'LineStyle',':')
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
legend('$h_{1,\,measured}$','$h_{1,\,ref}$', '$h_{1,\,estimated}$', '
    Interpreter','latex')
ylim([0 1]);
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1) % Use only when zooming in on Figure. Else,
    comment out

% Tank 2
subplot(3,1,2)
plot(t, h2_real, 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t, h2_ref, 'green', 'LineWidth',1, 'LineStyle','--')
hold on
plot(t, h2_estimated, 'red', 'LineWidth',1, 'LineStyle',':')
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
legend('$h_{2,\,measured}$','$h_{2,\,ref}$', '$h_{2,\,estimated}$', '
    Interpreter','latex')
ylim([0 1])
xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1) % Use only when zooming in on Figure. Else,
    comment out

% Input Signals
subplot(3,1,3)
stairs(t, ulv1, 'magenta', 'LineWidth',1)
hold on
stairs(t, ulv2, 'green', 'LineWidth',1)
hold on
```

```
stairs(t, upa, 'k', 'LineStyle',':', LineWidth=1)
hold on
stairs(t, upa_estimate, 'r', 'LineStyle','-.', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('$[-]$', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$','$u_{PA001,\,estimated
    }$', 'Interpreter','latex');
ylim([0 1])
xlim([0 t(end)])
grid on
box on
```

I.3.10 Plot_Experimental_Data.m

```
%% Plot Experimental Data

% Author: Gent Luta

% Date: Spring 2023

% Load Scope Data File
load("Experimental_Folder_FINAL_DATA_080723\
    Experimental_Data_Adaptive_08_1_100_60_10.mat")

% Tank 1 Data
h1 = ScopeData1{1}.Values.Data(:,1);
h1_ref = ScopeData1{1}.Values.Data(:,2);

% Tank 2 Data
h2 = ScopeData1{2}.Values.Data(:,1);
h2_ref = ScopeData1{2}.Values.Data(:,2);

% Input Signals Data
ulv1 = ScopeData1{3}.Values.Data(:,1);
ulv2 = ScopeData1{3}.Values.Data(:,2);
upa = ScopeData1{3}.Values.Data(:,3);
t = ScopeData1{1}.Values.Time;

figure
% Tank 1
subplot(3,1,1)
plot(t, h1, 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t, h1_ref, 'red', 'LineWidth',1, 'LineStyle','--')
xlabel('$t\, [s]$', Interpreter='latex')
```

```
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 1', Interpreter='latex')
legend('$h_{1}$','$h_{1,\,ref}$', 'Interpreter','latex')
ylim([0 1]);
grid on
box on
set(gca,'YTick',0:0.05:1) % Use only when zooming in on Figure. Else,
    comment out

% Tank 2
subplot(3,1,2)
plot(t, h2, 'blue', 'LineWidth',1, 'LineStyle','-')
hold on
plot(t, h2_ref, 'red', 'LineWidth',1, 'LineStyle','--')
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('Water level $[m]$', Interpreter='latex')
title('Tank 2', Interpreter='latex')
legend('$h_{2}$','$h_{2,\,ref}$', 'Interpreter','latex')
ylim([0 1])
%xlim([0 t(end)])
grid on
box on
set(gca,'YTick',0:0.05:1) % Use only when zooming in on Figure. Else,
    comment out

% Input Signals
subplot(3,1,3)
stairs(t, ulv1, 'magenta', 'LineWidth',1)
hold on
stairs(t, ulv2, 'green', 'LineWidth',1)
hold on
stairs(t, upa, 'k', 'LineStyle',':', LineWidth=1)
xlabel('$t\, [s]$', Interpreter='latex')
ylabel('$[-]$', Interpreter='latex')
title('Input Signals', Interpreter='latex')
legend('$u_{LV001}$', '$u_{LV002}$', '$u_{PA001}$', 'Interpreter','latex'
    );
ylim([0 1])
grid on
box on




%%
t_inx = 301;
t_indx_stop = 7001;
u0 = [0.5317;0.5680];
```

```
ts = 0.1;


cost = 0;
cost_Q = 0;
cost_S = 0;
cost_R = 0;

Q = [1 0 ;
     0 1];
%Q(1,1) = Q(1,1)/(1 - 0.13);    % Scaling
%Q(2,2) = Q(2,2)/(0.4 - 0.02);    % Scaling
S = [0 0 ;
     0 0];
R = [0 0;
     0 0];
E = [(h1(t_inx:t_indx_stop) - h1_ref(t_inx:t_indx_stop))' ; (h2(t_inx:
    t_indx_stop) - h2_ref(t_inx:t_indx_stop))'];
EU = [ulv1(t_inx:t_indx_stop)';ulv2(t_inx:t_indx_stop)'] - u0;
mvRate = [ulv1(t_inx:t_indx_stop-1)';ulv2(t_inx:t_indx_stop-1)'] - [ulv1(
    t_inx+1:t_indx_stop)';ulv2(t_inx+1:t_indx_stop)'];
for l = 1:length(t(t_inx:t_indx_stop-1))
    cost_Q = cost_Q + E(:,l)'*(Q^2)*E(:,l);
    cost_S = cost_S + mvRate(:,l)'*(S^2)*mvRate(:,l);
    cost_R = cost_R + EU(:,l)'*(R^2)*EU(:,l);
    cost = cost +  E(:,l)'*(Q^2)*E(:,l) + mvRate(:,l)'*(S^2)*mvRate(:,l)
    + EU(:,l)'*(R^2)*EU(:,l);
end
cost*ts
```

I.3.11 Plot_Forward_Euler.m

```
%% Plot Euler Method and ODE45

% Author: Gent Luta

% Date: Spring 2023

x0 = [0.5;0.5];
u0 = [0.5;0.5;0.8];

ts = 0.1;
h1 = 1;
h2 = 5;
h3 = 10;
h4 = 15;
```

```
h5 = 20;
h6 = 25;
h7 = 30;

Duration = 300;
t1 = 0:h1:Duration; N1 = length(t1);
t2 = 0:h2:Duration; N2 = length(t2);
t3 = 0:h3:Duration; N3 = length(t3);
t4 = 0:h4:Duration; N4 = length(t4);
t5 = 0:h5:Duration; N5 = length(t5);
t6 = 0:h6:Duration; N6 = length(t6);
t7 = 0:h7:Duration; N7 = length(t7);
tode = 0:ts:Duration; Node = length(tode);

XX1 = zeros(2,N1); XX1(:,1) = x0;
XX2 = zeros(2,N2); XX2(:,1) = x0;
XX3 = zeros(2,N3); XX3(:,1) = x0;
XX4 = zeros(2,N4); XX4(:,1) = x0;
XX5 = zeros(2,N5); XX5(:,1) = x0;
XX6 = zeros(2,N6); XX6(:,1) = x0;
XX7 = zeros(2,N7); XX7(:,1) = x0;
XXODE = zeros(2,Node); XXODE(:,1) = x0;
x1 = x0;
x2 = x0;
x3 = x0;
x4 = x0;
x5 = x0;
x6 = x0;
x7 = x0;
xode = x0;

for i = 2:(Duration/h1)+1
    x1 = x1 + h1*tankCT_NEW(x1,u0);
    XX1(:,i) = x1;
end

for i = 2:(Duration/h2)+1
    x2 = x2 + h2*tankCT_NEW(x2,u0);
    XX2(:,i) = x2;
end

for i = 2:(Duration/h3)+1
    x3 = x3 + h3*tankCT_NEW(x3,u0);
    XX3(:,i) = x3;
end

for i = 2:(Duration/h4)+1
    x4 = x4 + h4*tankCT_NEW(x4,u0);
    XX4(:,i) = x4;
end

for i = 2:(Duration/h5)+1
    x5 = x5 + h5*tankCT_NEW(x5,u0);
    XX5(:,i) = x5;
end

for i = 2:(Duration/h6)+1
    x6 = x6 + h6*tankCT_NEW(x6,u0);
    XX6(:,i) = x6;
end

for i = 2:(Duration/h7)+1
    x7 = x7 + h7*tankCT_NEW(x7,u0);
    XX7(:,i) = x7;
end

for i = 2:(Duration/ts)+1
    [~,XODE] = ode45(@(t,x)tankCT_NEW(x, u0), [0 ts], xode);
    xode = [XODE(end,1);XODE(end,2)];
    XXODE(:,i) = xode;
end
%% Plot of Tank 1
figure
plot(tode, XXODE(1,:), "Color","#A2142F", 'LineWidth',1)
hold on
plot(t1, XX1(1,:), "Color","#77AC30", 'LineWidth',1)
hold on
plot(t2,XX2(1,:), "Color","#D95319", 'LineWidth',1)
hold on
plot(t3,XX3(1,:), "Color","#EDB120", 'LineWidth',1)
hold on
plot(t4,XX4(1,:), "Color","#7E2F8E", 'LineWidth',1)
hold on
plot(t5,XX5(1,:), "Color","#0072BD", 'LineWidth',1)
hold on
plot(t6,XX6(1,:), "Color",'m', 'LineWidth',1)
hold on
plot(t7,XX7(1,:), "Color","cyan", 'LineWidth',1)
hold on
legend('\verb|ODE45|','$h_{Euler} = 1$', '$h_{Euler} = 5$', '$h_{Euler} =
     10$',...
    '$h_{Euler} = 15$', '$h_{Euler} = 20$', '$h_{Euler} = 25$', '$h_{
    Euler} = 30$', 'Interpreter','latex')
xlabel('Time', Interpreter='latex')
ylabel('$h_1 \,[m]$', Interpreter='latex')
title('Euler Method and \verb|ODE45|', Interpreter='latex')
```

```
box on
grid on

axes('position',[.65 .175 .25 .25])
box on % put box around new pair of axes
indexOfInterest = (tode < 120) & (tode > 50); % range of t near
    perturbation
plot(tode(indexOfInterest),XXODE(1,indexOfInterest),  "Color","#A2142F",
    'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t1 < 120) & (t1 > 50); % range of t near perturbation
plot(t1(indexOfInterest),XX1(1,indexOfInterest),  "Color","#77AC30", '
    LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t2 < 120) & (t2 > 50); % range of t near perturbation
plot(t2(indexOfInterest),XX2(1,indexOfInterest),  "Color","#D95319", '
    LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t3 < 120) & (t3 > 50); % range of t near perturbation
plot(t3(indexOfInterest),XX3(1,indexOfInterest),  "Color","#EDB120", '
    LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t4 < 120) & (t4 > 50); % range of t near perturbation
plot(t4(indexOfInterest),XX4(1,indexOfInterest),  "Color","#7E2F8E", '
    LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t5 < 120) & (t5 > 50); % range of t near perturbation
plot(t5(indexOfInterest),XX5(1,indexOfInterest),  "Color","#0072BD", '
    LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t6 < 120) & (t6 > 50); % range of t near perturbation
plot(t6(indexOfInterest),XX6(1,indexOfInterest),  "Color","m", 'LineWidth
    ',1) % plot on new axes
hold on
indexOfInterest = (t7 < 120) & (t7 > 50); % range of t near perturbation
plot(t7(indexOfInterest),XX7(1,indexOfInterest),  "Color","cyan", '
    LineWidth',1) % plot on new axes
axis tight


%% Plot of Tank 2
figure
plot(tode, XXODE(2,:), "Color","#A2142F", 'LineWidth',1)
hold on
plot(t1, XX1(2,:), "Color","#77AC30", 'LineWidth',1)
hold on
plot(t2,XX2(2,:), "Color","#D95319", 'LineWidth',1)
hold on
```

```
plot(t3,XX3(2,:), "Color","#EDB120", 'LineWidth',1)
hold on
plot(t4,XX4(2,:), "Color","#7E2F8E", 'LineWidth',1)
hold on
plot(t5,XX5(2,:), "Color","#0072BD", 'LineWidth',1)
hold on
plot(t6,XX6(2,:), "Color",'m', 'LineWidth',1)
hold on
plot(t7,XX7(2,:), "Color","cyan", 'LineWidth',1)
hold on
legend('\verb|ODE45|','$h_{Euler} = 1$', '$h_{Euler} = 5$', '$h_{Euler} =
    10$',...
    '$h_{Euler} = 15$', '$h_{Euler} = 20$', '$h_{Euler} = 25$', '$h_{
    Euler} = 30$', 'Interpreter','latex')
xlabel('Time', Interpreter='latex')
ylabel('$h_2 \,[m]$', Interpreter='latex')
title('Euler Method and \verb|ODE45|', Interpreter='latex')
box on
grid on

axes('position',[.65 .6 .25 .25])
box on % put box around new pair of axes
indexOfInterest = (tode < 120) & (tode > 50); % range of t near
    perturbation
plot(tode(indexOfInterest),XXODE(2,indexOfInterest),  "Color","#A2142F",
    'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t1 < 120) & (t1 > 50); % range of t near perturbation
plot(t1(indexOfInterest),XX1(2,indexOfInterest),  "Color","#77AC30", '
    LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t2 < 120) & (t2 > 50); % range of t near perturbation
plot(t2(indexOfInterest),XX2(2,indexOfInterest),  "Color","#D95319", '
    LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t3 < 120) & (t3 > 50); % range of t near perturbation
plot(t3(indexOfInterest),XX3(2,indexOfInterest),  "Color","#EDB120", '
    LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t4 < 120) & (t4 > 50); % range of t near perturbation
plot(t4(indexOfInterest),XX4(2,indexOfInterest),  "Color","#7E2F8E", '
    LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t5 < 120) & (t5 > 50); % range of t near perturbation
plot(t5(indexOfInterest),XX5(2,indexOfInterest),  "Color","#0072BD", '
    LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t6 < 120) & (t6 > 50); % range of t near perturbation
```

```
plot(t6(indexOfInterest),XX6(2,indexOfInterest),  "Color","m", 'LineWidth
    ',1) % plot on new axes
hold on
indexOfInterest = (t7 < 120) & (t7 > 50); % range of t near perturbation
plot(t7(indexOfInterest),XX7(2,indexOfInterest),  "Color","cyan", '
    LineWidth',1) % plot on new axes
axis tight
```

I.3.12 Plot_Pump_Char.m

```
%% Pump Plots

% Author: Gent Luta

% Date: Spring 2023

%% Pump Char.
u_PA001 = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
             0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
             8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s

f = figure;
plot(u_PA001, q_PA001,LineStyle="-", Color='k', LineWidth=1, Marker='*')
title('Pump Characteristics', Interpreter='latex')
xlabel('Control Signal $u_{PA001}(t)$', Interpreter='latex')
ylabel('Flow Rate $q_{PA001}(t)$', Interpreter='latex')
grid on
box on
%exportgraphics(f,'PumpChar.png','Resolution',300)

%% Forward Difference v Pump Char.
z = 0:0.01:1;
delta = 0.1;
Mid = interp1(u_PA001,q_PA001,0.65);
Forward = interp1(u_PA001,q_PA001,0.75);
Forward_Diff = (Forward - Mid)/(delta);
b = Mid - Forward_Diff*0.65;
Line = Forward_Diff.*z + b;

figure(3)
plot(z, Line, LineWidth=1, Color='k', LineStyle=':')
hold on
plot(u_PA001, q_PA001,LineStyle="-", Color='k', LineWidth=1, Marker='*')
```

```
title('Forward Difference - Graphical Interpretation', Interpreter='latex
    ')
xlabel('Control Signal $u_{PA001}(t)$', Interpreter='latex')
ylabel('Flow Rate $q_{PA001}(t)$', Interpreter='latex')
legend('$\mathcal{L}_p$','Pump Char.', 'Interpreter','latex')
box on
grid on
ax = gca;
ax.TickLabelInterpreter = 'latex';
ax.GridColor = [0 0 0];
ax.GridLineStyle = '--';
yticks([Mid, Forward]);
yticklabels({'$f(u(t))$', '$f(u(t) + h)$'})
xticks([0.65, 0.75]);
xticklabels({'$u(t)$', '$u(t) + h$'})
ylim([0 3.5*(10^-4)])
```

I.3.13 Plot_ZOH.m

```
%% Plot Euler Method and ODE45

% Author: Gent Luta

% Date: Spring 2023

%% Control of Two-Tank Using Linear MPC.
% Linearized model about the nominal solution {h1 = 0.5, h2 = 0.3,
    u_PA001 = 0.8}

% Author: Gent Luta

% Date: Spring 2023


%% Steady State Values
h1_A = 0.5;
h2_A = 0.3;
hnominal = [h1_A;h2_A];
u_PA001_A = 0.8;
u1_A = Valve_1_OP_New(h1_A, u_PA001_A);
u2_A = Valve_2_OP(u1_A, h1_A, h2_A);
unominal = [u1_A;u2_A;u_PA001_A];
f1_A = ValveChar(u1_A);
f2_A = ValveChar(u2_A);

%% System Parameters (For Simulink Model)
```

```
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

%% Function Handles
f1_handle = @ValveChar;
f2_handle = @ValveChar;
f3_handle = @PumpChar;

delta = 0.01;    % Step size when using forward difference

%% Linearization
% A - Matrix
a11 = - (sqrt(100000))/(7.2*10^8) * ((Kv1*f1_A*rho*g) / (A1*sqrt(rho*g*(
    h1_A + h_LV001))));
a12 = 0;
a21 = ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*10^8)) * ((Kv1*
    f1_A*rho*g) / (sqrt(rho*g*(h1_A + h_LV001))));
a22 = - ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*(10^8))) * ((Kv2
    *f2_A*rho*g) / (sqrt(rho*g*(h2_A + h_LV002))));
A = [a11 a12; a21 a22];

% B - Matrix
b11 = - ((Kv1)/(3600*A1)) * sqrt((rho*g*(h1_A + h_LV001))/100000) *
    forward_diff(f1_handle, u1_A, delta);
b12 = 0;
b21 = ((1)/(0.004 + 0.07*h2_A)) * (Kv1/3600) * sqrt((rho*g*(h1_A +
    h_LV001))/100000)*forward_diff(f1_handle, u1_A, delta);
b22 = -((1)/(0.004 + 0.07*h2_A)) * (Kv2/3600) * sqrt((rho*g*(h2_A +
    h_LV002))/100000)*forward_diff(f2_handle, u2_A, delta);
B = [b11 b12 ; b21 b22];

% C - Matrix
C = eye(2);

% D - Matrix
D = zeros(2);

% G - Matrix (Disturbance)
g11 = forward_diff(f3_handle, u_PA001_A, delta)/A1;
g21 = 0;
G = [g11; g21];

% Augmented B - Matrix
```

```matlab
B_a = [B G];

% Augmented D - Matrix
D_a = zeros(2, 3);

%% State-Space-Model

ts = [1 5 10 20 30 40 50];
Duration = 300;
t1 = 0:ts(1):Duration;
t2 = 0:ts(2):Duration;
t3 = 0:ts(3):Duration;
t4 = 0:ts(4):Duration;
t5 = 0:ts(5):Duration;
t6 = 0:ts(6):Duration;
t7 = 0:ts(7):Duration;

n = length(ts);
x0 = [0.45;0.35];
u0 = [0.5;0.5;0.8];
XX_hist = cell(1,n);
for i = 1:n
    t = 0:ts(i):Duration; N = length(t);
    sys = ss(A, B_a, C, D_a);      % CT state-space model
    sys = c2d(sys,ts(i));          % DT state-space model (ts sampling)
    XX = zeros(2,N); XX(:,1) = x0;
    x = x0;
    u = u0;
    for j = 2:(Duration/ts(i))+1
    x = sys.A*(x-hnominal) + sys.B*(u - unominal);
    x = x + hnominal;
    XX(:,j) = x;
    end
    XX_hist{1,i} = XX;
end

ts_ode = 0.1;
tode = 0:ts_ode:Duration; Node = length(tode);
XXODE = zeros(2,Node); XXODE(:,1) = x0;
xode = x0;
for i = 2:(Duration/ts_ode)+1
    [~,XODE] = ode45(@(t,x)tankCT_Linear(x, u0-unominal), [0 ts_ode],
   xode-hnominal);
    xode = [XODE(end,1);XODE(end,2)];
    xode = xode + hnominal;
    XXODE(:,i) = xode;
end

```

```matlab
%% Plot of Tank 1
figure
plot(tode, XXODE(1,:), "Color","#A2142F", 'LineWidth',1)
hold on
plot(t1, XX_hist{1}(1,:), "Color","#77AC30", 'LineWidth',1)
hold on
plot(t2, XX_hist{2}(1,:), "Color","#D95319", 'LineWidth',1)
hold on
plot(t3, XX_hist{3}(1,:), "Color","#EDB120", 'LineWidth',1)
hold on
plot(t4, XX_hist{4}(1,:), "Color","#7E2F8E", 'LineWidth',1)
hold on
plot(t5, XX_hist{5}(1,:), "Color","#0072BD", 'LineWidth',1)
hold on
plot(t6, XX_hist{6}(1,:), "Color",'m', 'LineWidth',1)
hold on
plot(t7, XX_hist{7}(1,:), "Color","cyan", 'LineWidth',1)
hold on
legend('\verb|ODE45|','$t_s (ZOH) = 1$', '$t_s (ZOH) = 5$', '$t_s (ZOH) =
     10$',...
    '$t_s (ZOH) = 20$', '$t_s (ZOH) = 30$', '$t_s (ZOH) = 40$', '$t_s (
    ZOH) = 50$', 'Interpreter','latex')
xlabel('Time', Interpreter='latex')
ylabel('$h_1 \,[m]$', Interpreter='latex')
title('Zero-Order Hold and \verb|ODE45|', Interpreter='latex')
box on
grid on

 axes('position',[.65 .175 .25 .25])
box on % put box around new pair of axes
indexOfInterest = (tode <= 80) & (tode >= 0); % range of t near
    perturbation
plot(tode(indexOfInterest),XXODE(1,indexOfInterest),  "Color","#A2142F",
    'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t1 <= 80) & (t1 >= 0); % range of t near perturbation
plot(t1(indexOfInterest),XX_hist{1}(1,indexOfInterest),  "Color","#77AC30
    ", 'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t2 <= 80) & (t2 >= 0); % range of t near perturbation
plot(t2(indexOfInterest),XX_hist{2}(1,indexOfInterest),  "Color","#D95319
    ", 'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t3 <= 80) & (t3 >= 0); % range of t near perturbation
plot(t3(indexOfInterest),XX_hist{3}(1,indexOfInterest),  "Color","#EDB120
    ", 'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t4 <= 80) & (t4 >= 0); % range of t near perturbation
```

```
plot(t4(indexOfInterest),XX_hist{4}(1,indexOfInterest),  "Color","#7E2F8E
    ", 'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t5 <= 80) & (t5 >= 0); % range of t near perturbation
plot(t5(indexOfInterest),XX_hist{5}(1,indexOfInterest),  "Color","#0072BD
    ", 'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t6 <= 80) & (t6 >= 0); % range of t near perturbation
plot(t6(indexOfInterest),XX_hist{6}(1,indexOfInterest),  "Color","m", '
    LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t7 <= 80) & (t7 >= 0); % range of t near perturbation
plot(t7(indexOfInterest),XX_hist{7}(1,indexOfInterest),  "Color","cyan",
    'LineWidth',1) % plot on new axes
axis tight

%% Plot Tank 2
figure
plot(tode, XXODE(2,:), "Color","#A2142F", 'LineWidth',1)
hold on
plot(t1, XX_hist{1}(2,:), "Color","#77AC30", 'LineWidth',1)
hold on
plot(t2, XX_hist{2}(2,:), "Color","#D95319", 'LineWidth',1)
hold on
plot(t3, XX_hist{3}(2,:), "Color","#EDB120", 'LineWidth',1)
hold on
plot(t4, XX_hist{4}(2,:), "Color","#7E2F8E", 'LineWidth',1)
hold on
plot(t5, XX_hist{5}(2,:), "Color","#0072BD", 'LineWidth',1)
hold on
plot(t6, XX_hist{6}(2,:), "Color",'m', 'LineWidth',1)
hold on
plot(t7, XX_hist{7}(2,:), "Color","cyan", 'LineWidth',1)
hold on
legend('\verb|ODE45|','$t_s (ZOH) = 1$', '$t_s (ZOH) = 5$', '$t_s (ZOH) =
     10$',...
     '$t_s (ZOH) = 20$', '$t_s (ZOH) = 30$', '$t_s (ZOH) = 40$', '$t_s (
    ZOH) = 50$', 'Interpreter','latex')
xlabel('Time', Interpreter='latex')
ylabel('$h_2 \,[m]$', Interpreter='latex')
title('Zero-Order Hold and \verb|ODE45|', Interpreter='latex')
box on
grid on

 axes('position',[.65 .175 .25 .25])
box on % put box around new pair of axes
indexOfInterest = (tode <= 80) & (tode > 0); % range of t near
    perturbation
```

```
plot(tode(indexOfInterest),XXODE(2,indexOfInterest),  "Color","#A2142F",
    'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t1 <= 80) & (t1 >= 0); % range of t near perturbation
plot(t1(indexOfInterest),XX_hist{1}(2,indexOfInterest),  "Color","#77AC30
    ", 'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t2 <= 80) & (t2 >= 0); % range of t near perturbation
plot(t2(indexOfInterest),XX_hist{2}(2,indexOfInterest),  "Color","#D95319
    ", 'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t3 <= 80) & (t3 >= 0); % range of t near perturbation
plot(t3(indexOfInterest),XX_hist{3}(2,indexOfInterest),  "Color","#EDB120
    ", 'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t4 <= 80) & (t4 >= 0); % range of t near perturbation
plot(t4(indexOfInterest),XX_hist{4}(2,indexOfInterest),  "Color","#7E2F8E
    ", 'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t5 <= 80) & (t5 >= 0); % range of t near perturbation
plot(t5(indexOfInterest),XX_hist{5}(2,indexOfInterest),  "Color","#0072BD
    ", 'LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t6 <= 80) & (t6 >= 0); % range of t near perturbation
plot(t6(indexOfInterest),XX_hist{6}(2,indexOfInterest),  "Color","m", '
    LineWidth',1) % plot on new axes
hold on
indexOfInterest = (t7 <= 80) & (t7 >= 0); % range of t near perturbation
plot(t7(indexOfInterest),XX_hist{7}(2,indexOfInterest),  "Color","cyan",
    'LineWidth',1) % plot on new axes
axis tight
```

I.4 .m Files for Simulink

I.4.1 SIM_AdaptiveSys.m

```
function sys = SIM_AdaptiveSys(x,u, ts)
%% Linearization point
h1_A = x(1);
h2_A = x(2);
u_PA001_A = u(3);
u1_A = u(1);
u2_A = u(2);
f1_A = ValveChar(u1_A);
f2_A = ValveChar(u2_A);
```

```

%% System parameters
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

%% Function Handles
f1_handle = @ValveChar;
f2_handle = @ValveChar;
f3_handle = @PumpChar;

delta = 0.01;   %Step size (numerical difference)

%% Linearization
% A - Matrix
a11 = - (sqrt(100000))/(7.2*10^8) * ((Kv1*f1_A*rho*g) / (A1*sqrt(rho*g*(
    h1_A + h_LV001))));
a12 = 0;
a21 = ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*10^8)) * ((Kv1*
    f1_A*rho*g) / (sqrt(rho*g*(h1_A + h_LV001))));
a22 = - ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*(10^8))) * ((Kv2
    *f2_A*rho*g) / (sqrt(rho*g*(h2_A + h_LV002))));
A = [a11 a12; a21 a22];

% B - Matrix
b11 = - ((Kv1)/(3600*A1)) * sqrt((rho*g*(h1_A + h_LV001))/100000) *
    forward_diff(f1_handle, u1_A, delta);
b12 = 0;
b21 = ((1)/(0.004 + 0.07*h2_A)) * (Kv1/3600) * sqrt((rho*g*(h1_A +
    h_LV001))/100000)*forward_diff(f1_handle, u1_A, delta);
b22 = -((1)/(0.004 + 0.07*h2_A)) * (Kv2/3600) * sqrt((rho*g*(h2_A +
    h_LV002))/100000)*forward_diff(f2_handle, u2_A, delta);
B = [b11 b12 ; b21 b22];

% C - Matrix
C = eye(2);

% D - Matrix
%D = zeros(2);

% G - Matrix (Disturbance)
g11 = forward_diff(f3_handle, u_PA001_A, delta)/A1;
g21 = 0;
G = [g11; g21];
```

```

% B_a - Matrix (Augmented)
B_a = [B G];

% D_a - Matrix (Augmented)
D_a = zeros(2, 3);

%% State-Space-Model
sys = ss(A, B_a, C, D_a);   % CT State-Space Model
sys = c2d(sys,ts);          % DT State-Space Model (ts sampling)
% Signal Names
sys.InputName = {'u1', 'u2', 'u_PA001'};
sys.OutputName = {'h1', 'h2'};
sys.StateName = {'h1', 'h2'};
% Signal Units
sys.InputUnit = {'-', '-', '-'};
sys.OutputUnit = {'m', 'm'};
sys.StateUnit = {'m', 'm'};
% Signal Types
sys = setmpcsignals(sys, 'MV', [1 2], 'MD', 3, 'MO', [1 2]);
```

I.4.2 SIM_Eval_Explicit_MPC.m

```
function mv = SIM_Eval_Explicit_MPC(empcobj, x, last_mv, r, v)
xc = mpcstate(empcobj);
xc.Plant = x(1:2);
xc.LastMove = last_mv;

mv = mpcmoveExplicit(empcobj, xc, [], r, v);
end
```

I.4.3 SIM_forward_diff.m

```
function derivative = SIM_forward_diff(f, u, h)
    derivative = (f(u+h) - f(u))/(h);
end
```

I.4.4 SIM_GetMatrix.m

```
function [A, B, C, D] = SIM_GetMatrix(sys)
A = sys.A;
B = sys.B;
C = sys.C;
```

```
D = sys.D;
end
```

I.4.5 SIM_PumpChar.m

```
function f = SIM_PumpChar(u)
u_PA001 = [0.00  0.45  0.46  0.47  0.48  0.49  0.50  0.55...
           0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00];
q_PA001 = [0.00  0.00  1.25  2.25  3.15  3.75  4.40  6.75...
           8.75 10.70 12.25 13.75 15.15 16.50 18.00 19.20 20.00];
q_PA001 = q_PA001/60000;  % liter/min -> m3/s
f = interp1(u_PA001,q_PA001,u);
end
```

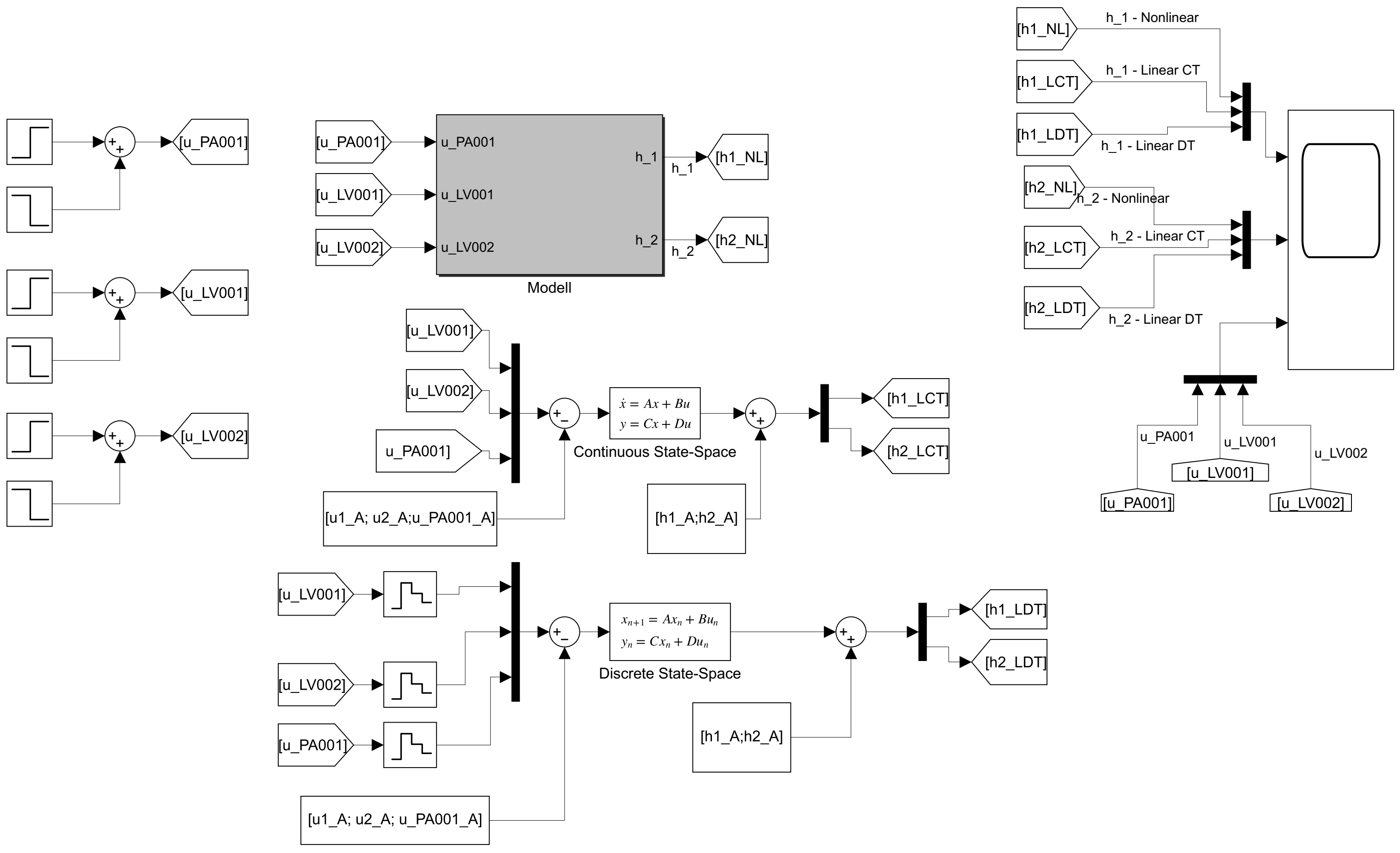
I.4.6 SIM_ValveChar.m

```
function f = SIM_ValveChar(u)
    %u = abs(u);
    f = (exp(u^1.2) - 1)/(exp(1) - 1);
end
```

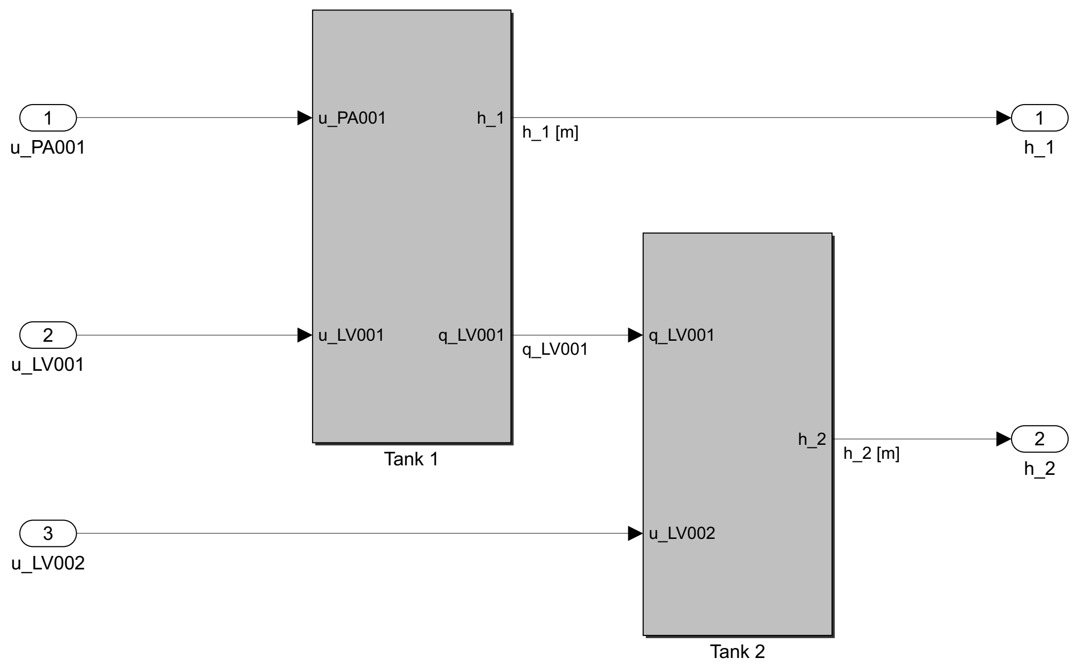
Appendix J

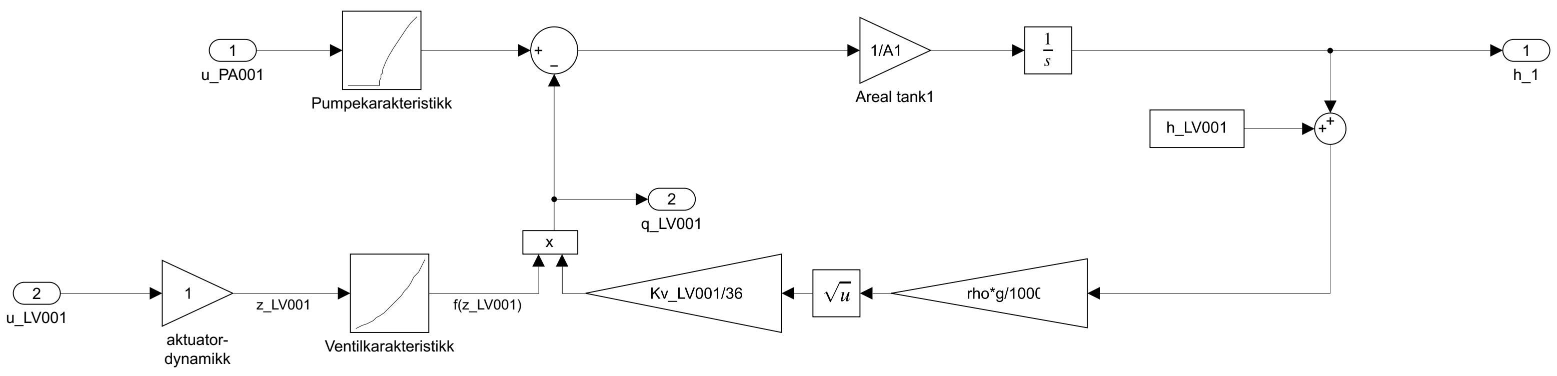
Simulink Schemes

J.1 SIM_Linear_VS_Nonlinear_Model.slx

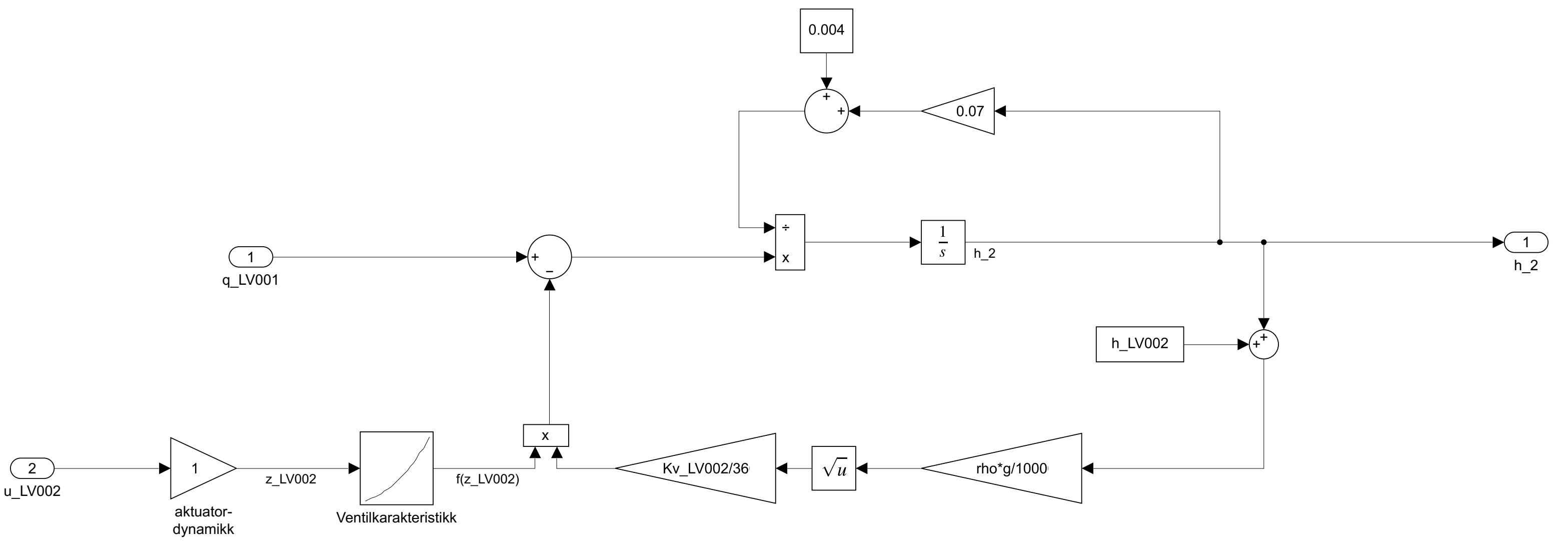


[u_PA001]
[u_LV001]
[u_LV002]
u_PA001
u_LV001
u_LV002
h_1
h_2
[h1_NL]
[h2_NL]
Modell
[u_LV001]
[u_LV002]
u_PA001
[u1_A; u2_A;u_PA001_A]
$\dot{x} = Ax + Bu$
$y = Cx + Du$
Continuous State-Space
[h1_A;h2_A]
[h1_LCT]
[h2_LCT]
[u_LV001]
[u_LV002]
[u_PA001]
[u1_A; u2_A; u_PA001_A]
$x_{n+1} = Ax_n + Bu_n$
$y_n = Cx_n + Du_n$
Discrete State-Space
[h1_A;h2_A]
[h1_LDT]
[h2_LDT]
[h1_NL]
h_1 - Nonlinear
[h1_LCT]
h_1 - Linear CT
[h1_LDT]
h_1 - Linear DT
[h2_NL]
h_2 - Nonlinear
[h2_LCT]
h_2 - Linear CT
[h2_LDT]
h_2 - Linear DT
u_PA001
u_LV001
u_LV002
[u_PA001]
[u_LV001]
[u_LV002]


1
u_PA001
2
u_LV001
3
u_LV002
u_PA001
h_1
u_LV001
q_LV001
Tank 1
h_1 [m]
q_LV001
q_LV001
h_2
u_LV002
Tank 2
h_2 [m]
1
h_1
2
h_2



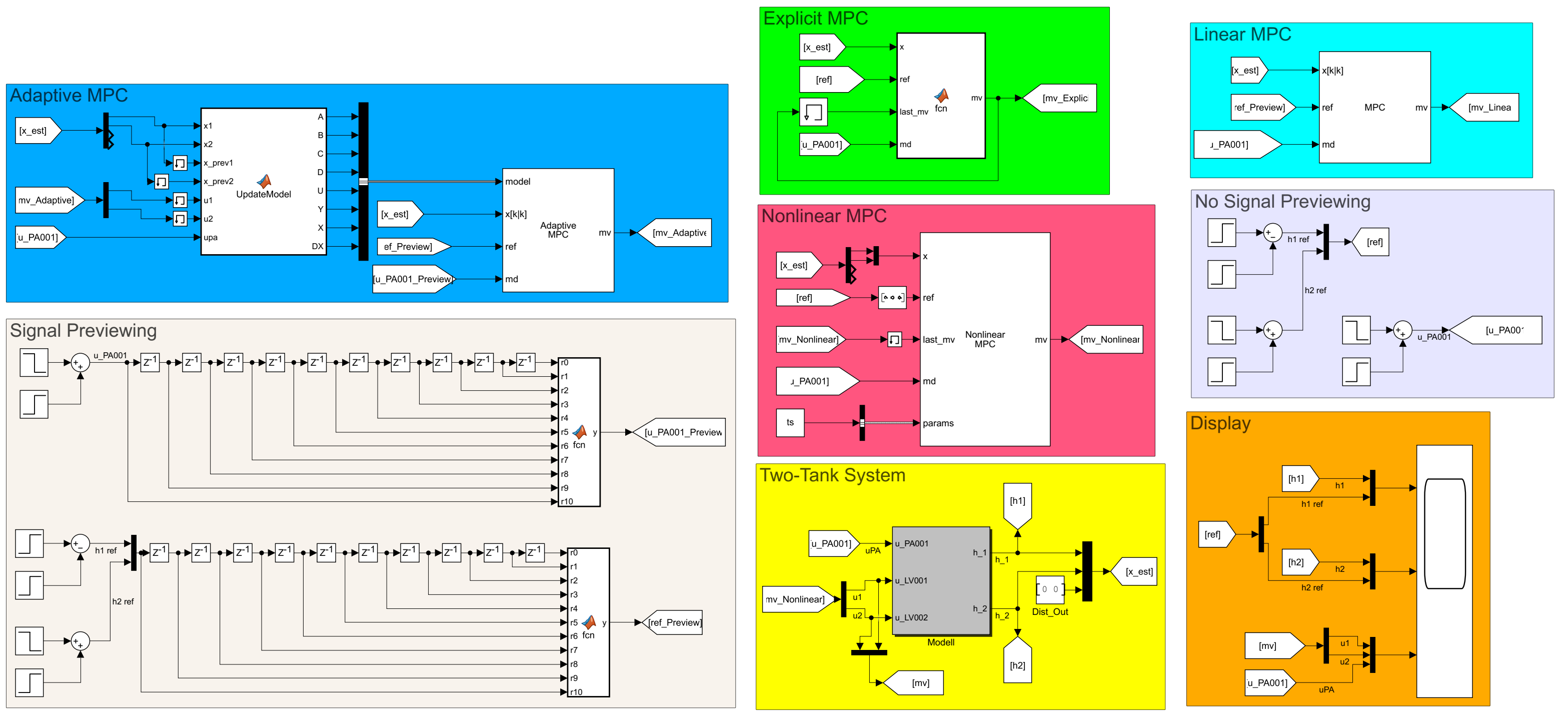
1
u_PA001
Pumpekarakteristikk
+
−
1/A1
Areal tank1
1/s
1
h_1
h_LV001
+
+
2
q_LV001
x
2
u_LV001
1
aktuator-
dynamikk
z_LV001
Ventilkarakteristikk
f(z_LV001)
Kv_LV001/36
√u
rho*g/1000


0.004
+
+
0.07
÷
x
1/s
h_2
1
q_LV001
+
–
1
h_2
h_LV002
+
+
2
u_LV002
1
aktuator-
dynamikk
z_LV002
Ventilkarakteristikk
f(z_LV002)
x
Kv_LV002/36
√u
rho*g/1000

Sample Times for 'SIM_Linear_VS_Nonlinear_Model'

Color	Annotation	Description	Value
⬛	Cont	Continuous	0
🟥	D1	Discrete 1	0.5
🟪	Inf	Constant	Inf
🟨	M	Multirate	N/A

J.2 SIM_MPC_Linear.slx



Adaptive MPC
[x_est]
x1
x2
x_prev1
x_prev2
u1
u2
upa
A
B
C
D
U
Y
X
DX
UpdateModel
[mv_Adaptive]
[u_PA001]
model
x[k|k]
ref
md
Adaptive MPC
mv
[mv_Adaptive
[x_est]
[ref_Preview]
[u_PA001_Preview]
Explicit MPC
[x_est]
[ref]
[u_PA001]
x
ref
last_mv
md
fcn
mv
[mv_Explic
Linear MPC
[x_est]
[ref_Preview]
[u_PA001]
x[k|k]
ref
md
MPC
mv
[mv_Linea
Nonlinear MPC
[x_est]
[ref]
[mv_Nonlinear]
[u_PA001]
ts
x
ref
last_mv
md
params
Nonlinear MPC
mv
[mv_Nonlinear
No Signal Previewing
h1 ref
h2 ref
[ref]
u_PA001
[u_PA00
Signal Previewing
u_PA001
z^{-1}
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
fcn
y
[u_PA001_Preview
h1 ref
h2 ref
[ref_Preview]
Two-Tank System
[u_PA001]
[mv_Nonlinear]
uPA
u1
u2
u_PA001
u_LV001
u_LV002
h_1
h_2
Modell
[h1]
[h2]
Dist_Out
[x_est]
[mv]
Display
[h1]
h1
h1 ref
[ref]
[h2]
h2
h2 ref
[mv]
u1
u2
[u_PA001]
uPA

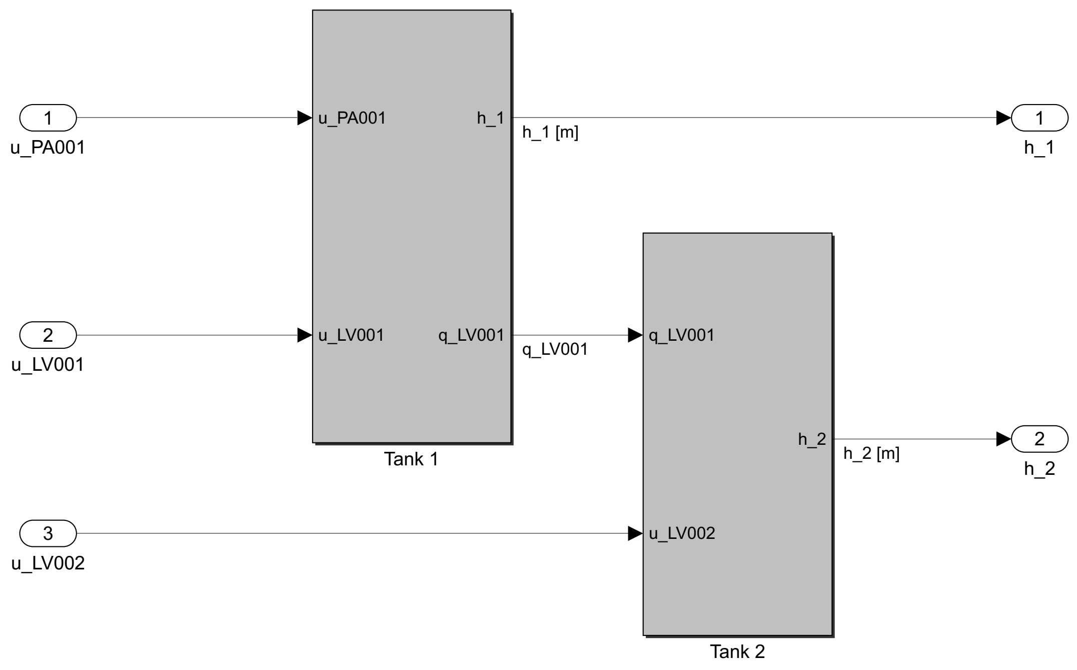
```
function y = fcn(r0,r1, r2, r3, r4, r5, r6, r7, r8, r9, r10)
y = [r0'; r1'; r2'; r3'; r4'; r5'; r6'; r7'; r8'; r9'; r10';];
```

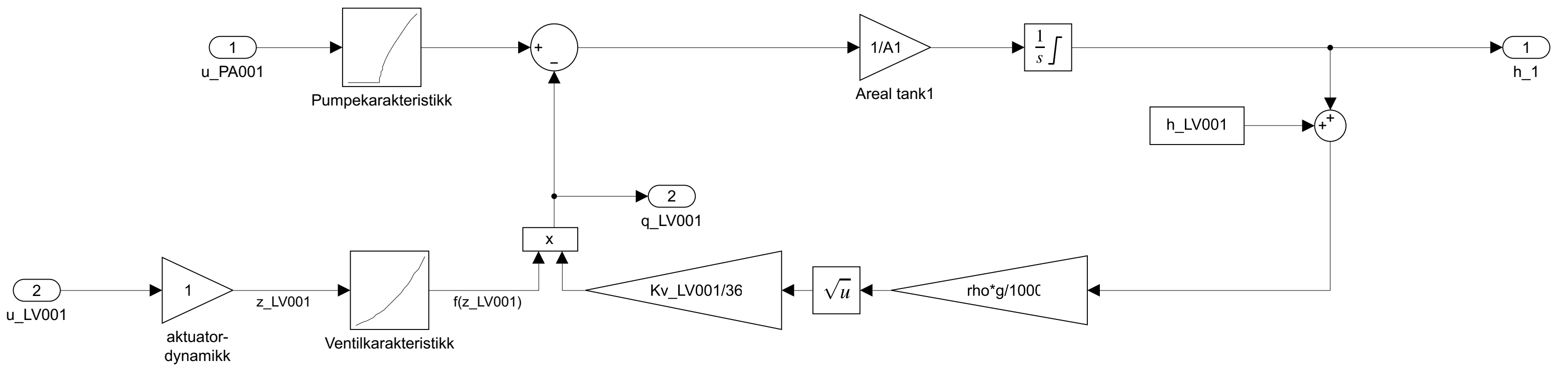
```
function mv = fcn(x,ref,last_mv, md)
coder.extrinsic('evalin')
coder.extrinsic('mpcmoveExplicit')
coder.extrinsic('assignin')
coder.extrinsic('SIM_Eval_Explicit_MPC')

mv = zeros(2,1);
persistent flag
if isempty(flag)
    mv = [evalin("base", 'u1_A');evalin("base", 'u2_A')];
    flag = 1;
else
    empcobj = evalin("base", 'empcobj');
    mv = SIM_Eval_Explicit_MPC(empcobj, x, last_mv, ref, md);
end
end
```

```
function y = fcn(r0,r1, r2, r3, r4, r5, r6, r7, r8, r9, r10)
y = [r0'; r1'; r2'; r3'; r4'; r5'; r6'; r7'; r8'; r9'; r10';];
```

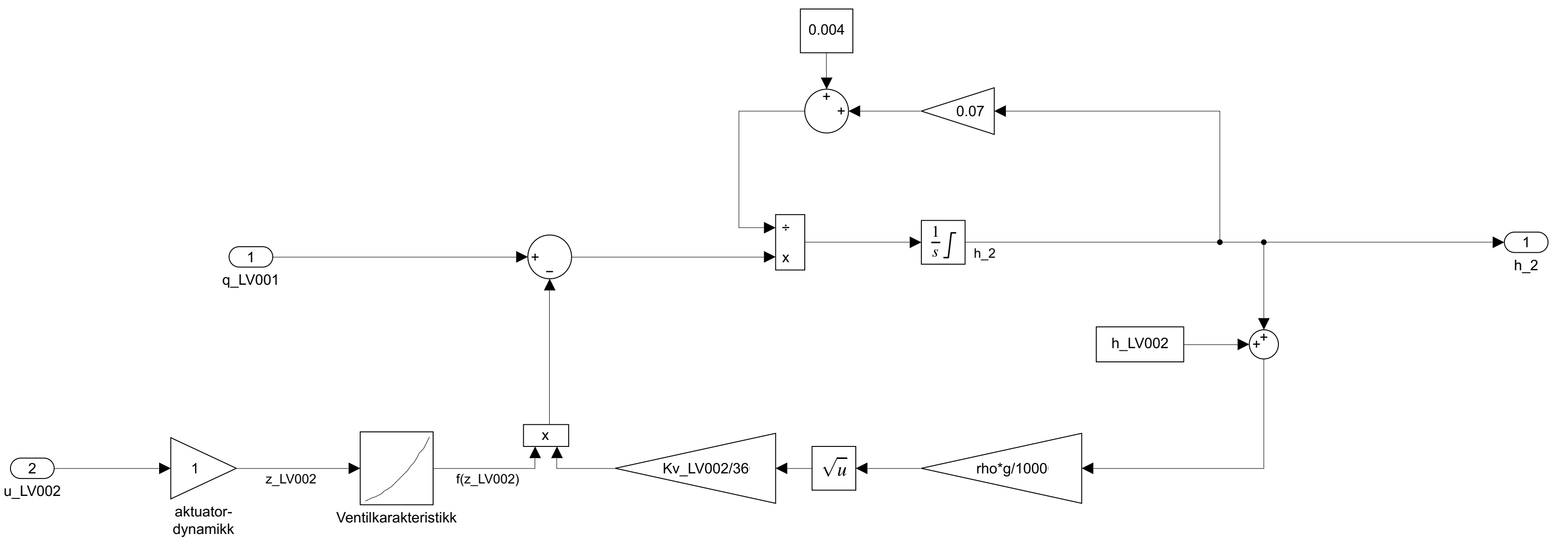
```
function [A, B, C, D, U, Y, X, DX] = UpdateModel(x1, x2, x_prev1, x_prev2, u1, u2, upa)
%coder.extrinsic('SIM_PumpChar')
%coder.extrinsic('SIM_ValveChar')
%coder.extrinsic('SIM_forward_diff')
coder.extrinsic('ss')
coder.extrinsic('c2d')
coder.extrinsic('SIM_GetMatrix')

ts = 0.5;
h1_A = x1;
h2_A = x2;
u_PA001_A = upa;
u1_A = u1;
u2_A = u2;
f1_A = SIM_ValveChar(u1_A);
f2_A = SIM_ValveChar(u2_A);

%% System parameters
rho = 1000;
g = 9.81;
A1 = 0.01;
Kv1 = 11.25;
Kv2 = 11.25;
h_LV001 = 0.05;
h_LV002 = 0.25;

%% Function Handles

delta = 0.01;   %Step size (numerical difference)

%% Linearization
% A - Matrix
a11 = - (sqrt(100000))/(7.2*10^8) * ((Kv1*f1_A*rho*g) / (A1*sqrt(rho*g*(h1_A + h_LV001))));
a12 = 0;
a21 = ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*10^8)) * ((Kv1*f1_A*rho*g) / (sqrt(rho*g*(h1_A + h_LV001))));
a22 = - ((1)/(0.004 + 0.07*h2_A)) * ((sqrt(100000))/(7.2*(10^8))) * ((Kv2*f2_A*rho*g) / (sqrt(rho*g*(h2_A + h_LV002))));
A = [a11 a12; a21 a22];

% B - Matrix
b11 = - ((Kv1)/(3600*A1)) * sqrt((rho*g*(h1_A + h_LV001))/100000) * ((SIM_ValveChar(u1_A + delta) - SIM_ValveChar(u1_A))/(delta));
b12 = 0;
b21 = ((1)/(0.004 + 0.07*h2_A)) * (Kv1/3600) * sqrt((rho*g*(h1_A + h_LV001))/100000)*((SIM_ValveChar(u1_A + delta) - SIM_ValveChar(u1_A))/
b22 = -((1)/(0.004 + 0.07*h2_A)) * (Kv2/3600) * sqrt((rho*g*(h2_A + h_LV002))/100000)*((SIM_ValveChar(u2_A + delta) - SIM_ValveChar(u2_A))/
B_a = [b11 b12 ; b21 b22];

% C - Matrix
C = eye(2);

% D - Matrix
%D = zeros(2);

% G - Matrix (Disturbance)
g11 = ((SIM_PumpChar(u_PA001_A + delta) - SIM_PumpChar(u_PA001_A))/(delta))/(A1);
g21 = 0;
G = [g11; g21];

% B_a - Matrix (Augmented)
```

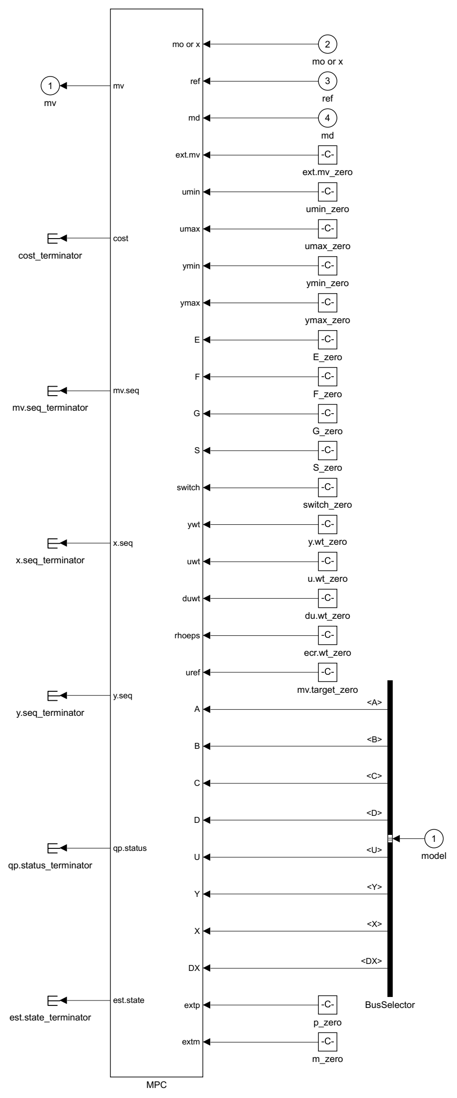
```
B = [B_a G];

% D_a - Matrix (Augmented)
D = zeros(2, 3);
%% State-Space-Model
sys = ss(A, B, C, D);   % CT State-Space Model
sys = c2d(sys,ts);          % DT State-Space Model (ts sampling)
[A, B, C, D] = SIM_GetMatrix(sys);

X = [x1;x2];
U = [u1;u2;upa];
Y = [x1;x2];
DX = [x1;x2] - [x_prev1;x_prev2];
```


1
u_PA001
2
u_LV001
3
u_LV002
u_PA001
h_1
h_1 [m]
u_LV001
q_LV001
q_LV001
Tank 1
q_LV001
h_2
h_2 [m]
u_LV002
Tank 2
1
h_1
2
h_2

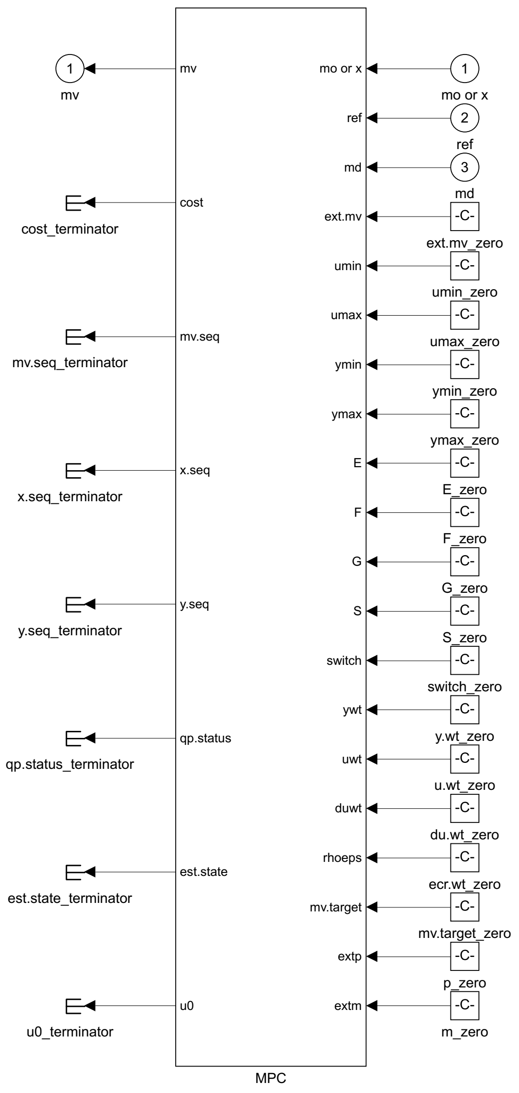


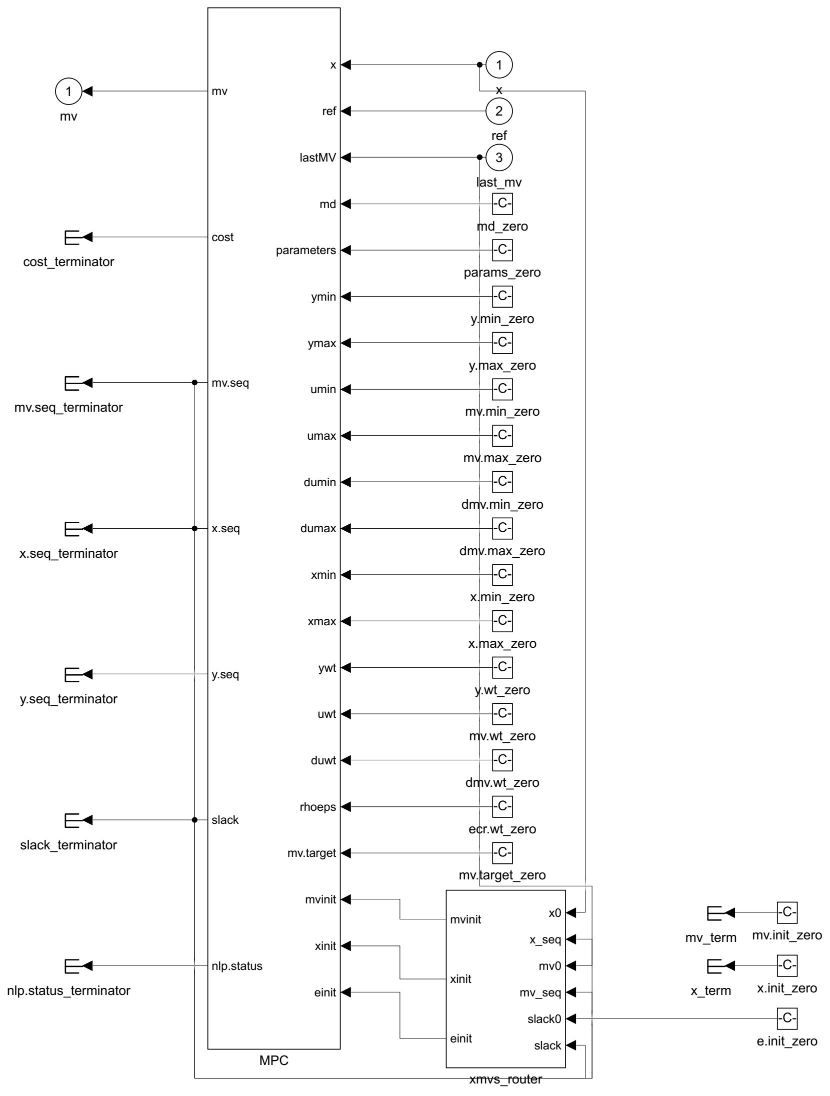
1
u_PA001
Pumpekarakteristikk
+
_
1/A1
Areal tank1
$\frac{1}{s}\int$
1
h_1
h_LV001
+
+
2
q_LV001
x
2
u_LV001
1
aktuator-
dynamikk
z_LV001
Ventilkarakteristikk
f(z_LV001)
Kv_LV001/36
$\sqrt{u}$
rho*g/1000


0.004
+
+
0.07
1
q_LV001
+
−
÷
x
$\frac{1}{s}\int$
h_2
1
h_2
h_LV002
+
+
2
u_LV002
1
aktuator-dynamikk
z_LV002
Ventilkarakteristikk
f(z_LV002)
x
Kv_LV002/36
$\sqrt{u}$
rho*g/1000



MPC
mo or x
ref
md
ext.mv
umin
umax
ymin
ymax
E
F
G
S
switch
ywt
uwt
duwt
rhoeps
uref
A
B
C
D
U
Y
X
DX
extp
extm
mv
cost
mv.seq
x.seq
y.seq
qp.status
est.state
1
mv
cost_terminator
mv.seq_terminator
x.seq_terminator
y.seq_terminator
qp.status_terminator
est.state_terminator
2
mo or x
3
ref
4
md
-C-
ext.mv_zero
umin_zero
umax_zero
ymin_zero
ymax_zero
E_zero
F_zero
G_zero
S_zero
switch_zero
y.wt_zero
u.wt_zero
du.wt_zero
ecr.wt_zero
mv.target_zero
<A>
<B>
<C>
<D>
<U>
<Y>
<X>
<DX>
BusSelector
1
model
p_zero
m_zero

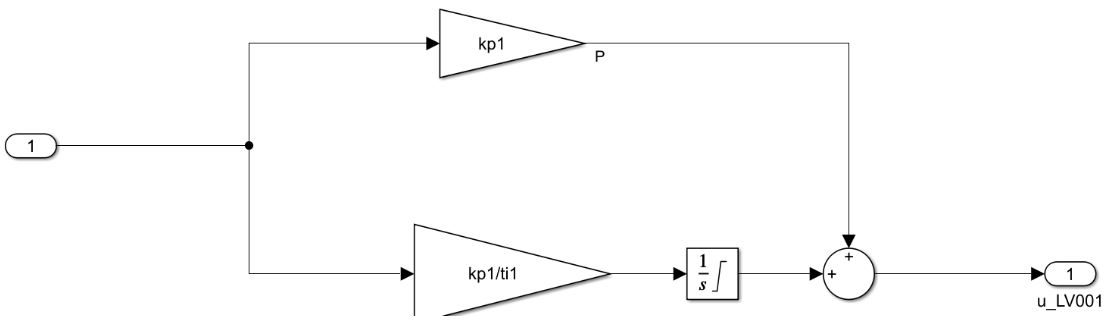

1
mv
cost_terminator
mv.seq_terminator
x.seq_terminator
y.seq_terminator
qp.status_terminator
est.state_terminator
u0_terminator
mv
cost
mv.seq
x.seq
y.seq
qp.status
est.state
u0
mo or x
ref
md
ext.mv
umin
umax
ymin
ymax
E
F
G
S
switch
ywt
uwt
duwt
rhoeps
mv.target
extp
extm
MPC
1
mo or x
2
ref
3
md
-C-
ext.mv_zero
umin_zero
umax_zero
ymin_zero
ymax_zero
E_zero
F_zero
G_zero
S_zero
switch_zero
y.wt_zero
u.wt_zero
du.wt_zero
ecr.wt_zero
mv.target_zero
p_zero
m_zero

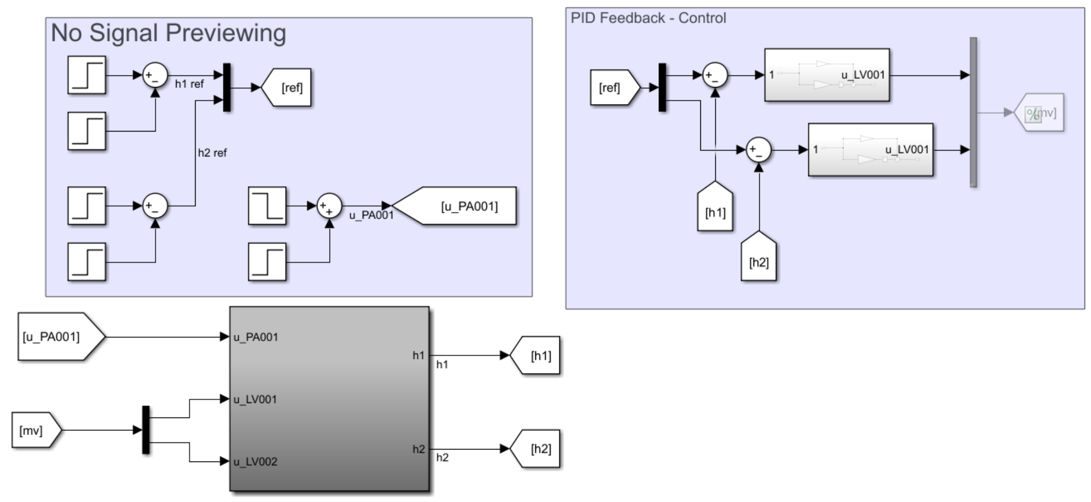

MPC
x
ref
lastMV
md
parameters
ymin
ymax
umin
umax
dumin
dumax
xmin
xmax
ywt
uwt
duwt
rhoeps
mv.target
mvinit
xinit
einit
mv
cost
mv.seq
x.seq
y.seq
slack
nlp.status
1 mv
cost_terminator
mv.seq_terminator
x.seq_terminator
y.seq_terminator
slack_terminator
nlp.status_terminator
1 x
2 ref
3 last_mv
md_zero
params_zero
y.min_zero
y.max_zero
mv.min_zero
mv.max_zero
dmv.min_zero
dmv.max_zero
x.min_zero
x.max_zero
y.wt_zero
mv.wt_zero
dmv.wt_zero
ecr.wt_zero
mv.target_zero
xmvs_router
mvinit
xinit
einit
x0
x_seq
mv0
mv_seq
slack0
slack
mv_term
mv.init_zero
x_term
x.init_zero
e.init_zero

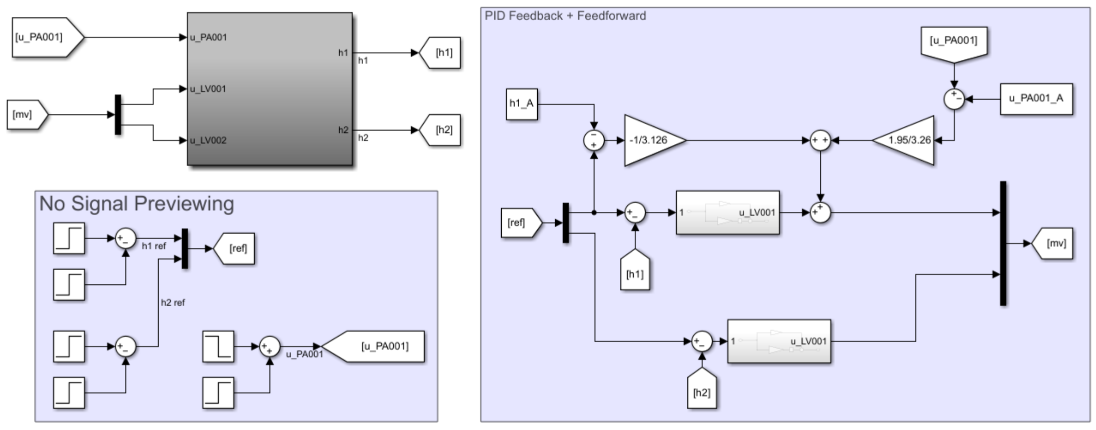
Sample Times for 'SIM_MPC_Linear'

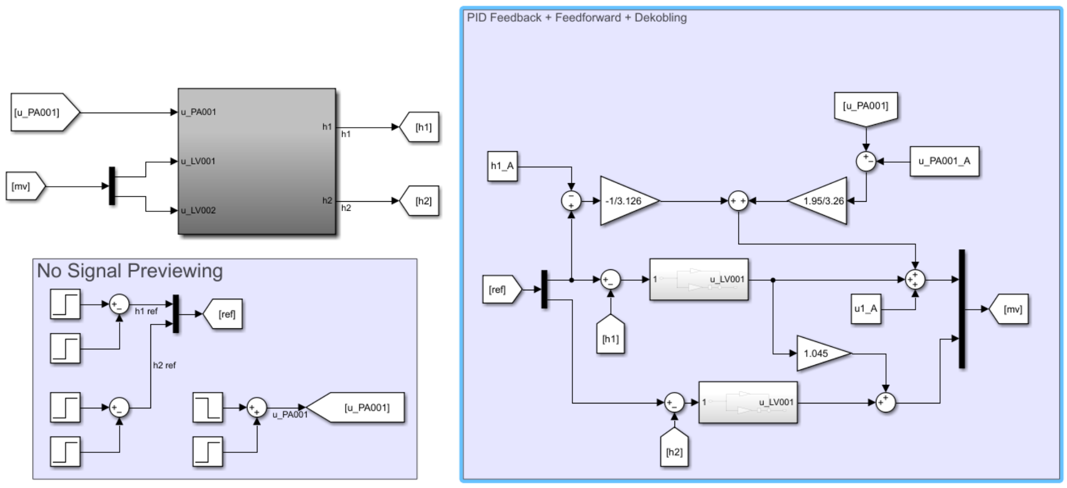
Color	Annotation	Description	Value
⬛	Cont	Continuous	0
🟥	D1	Discrete 1	0.5
🟪	Inf	Constant	Inf
🟨	M	Multirate	N/A

J.3 totank_Live_Edited_070723_PID.slx

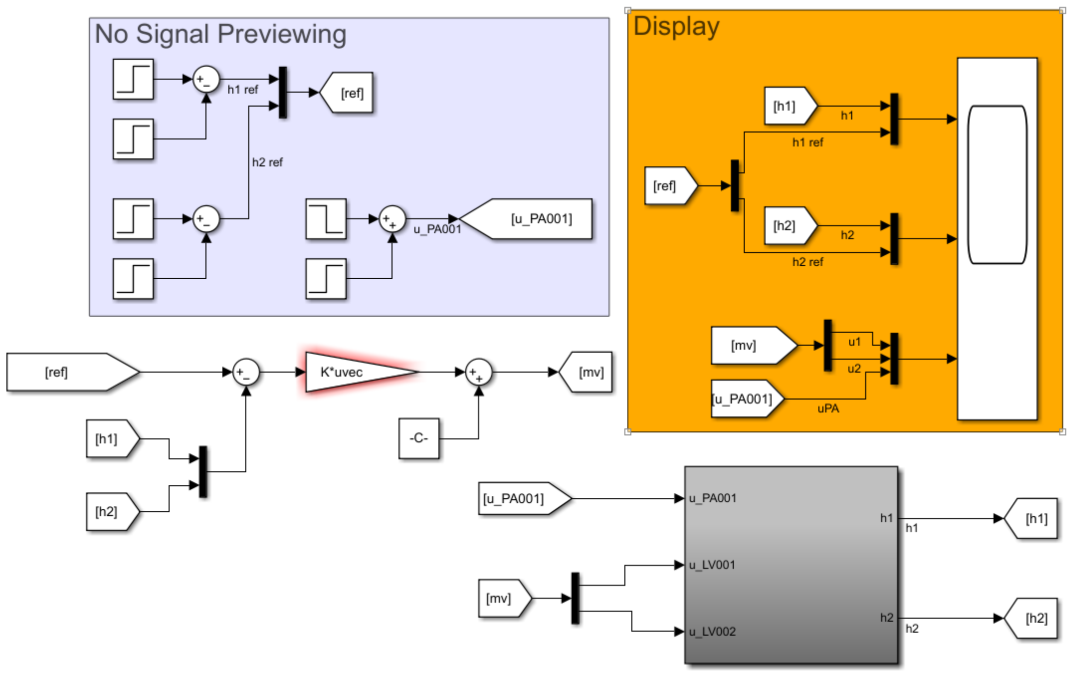




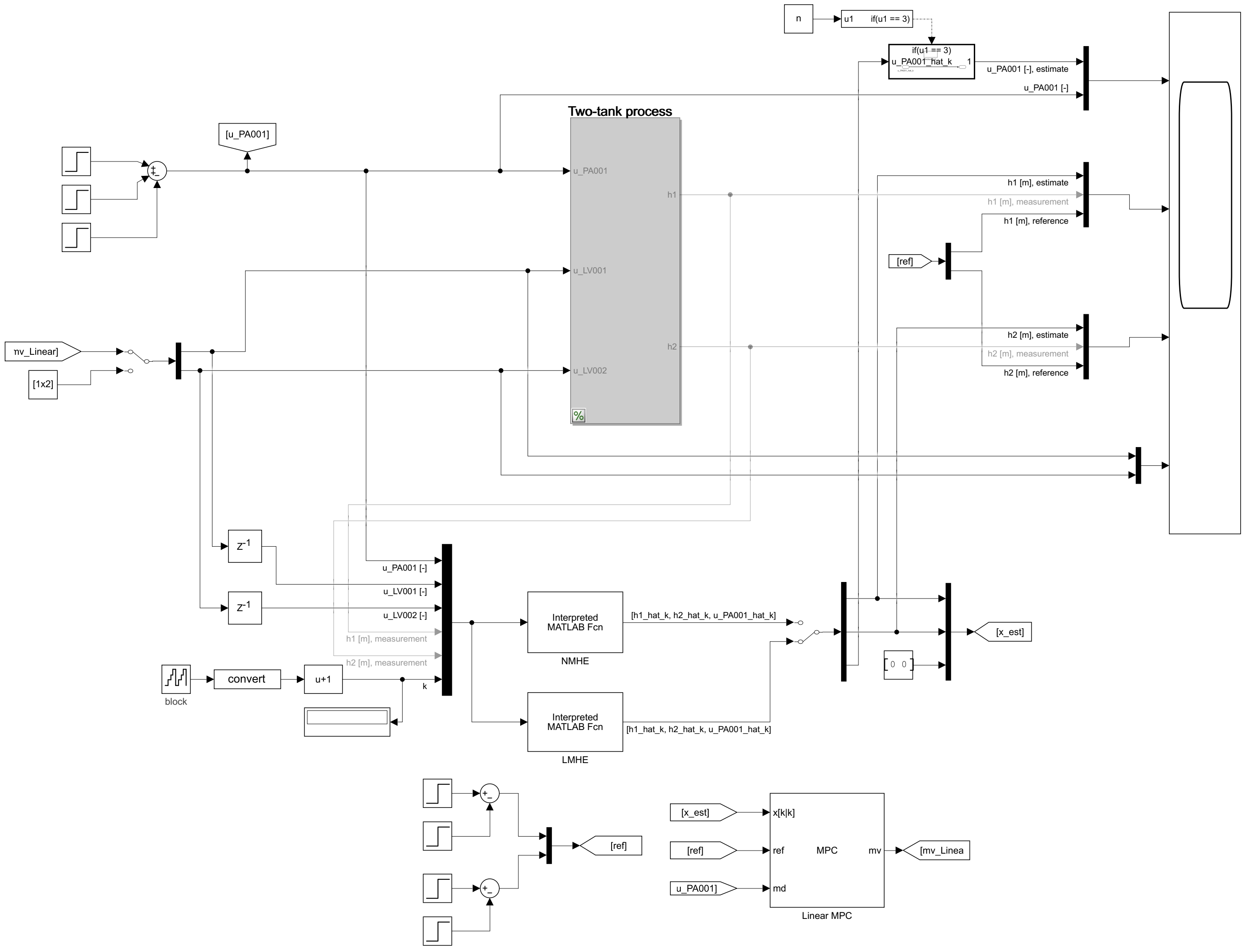


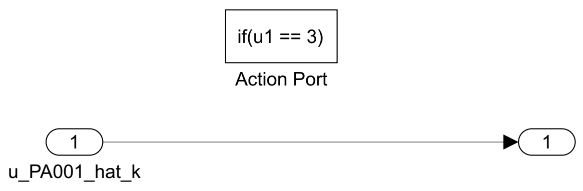

[u_PA001]
u_PA001
h1
h1
[h1]
[mv]
u_LV001
u_LV002
h2
h2
[h2]
No Signal Previewing
h1 ref
[ref]
h2 ref
u_PA001
[u_PA001]
PID Feedback + Feedforward + Dekobling
[u_PA001]
u_PA001_A
h1_A
-1/3.126
1.95/3.26
[ref]
1
u_LV001
[h1]
u1_A
[mv]
1.045
1
u_LV001
[h2]

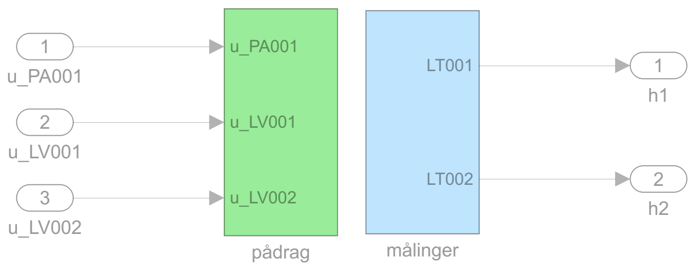
J.4 totank_Live_Edited_070723_LQR.slx

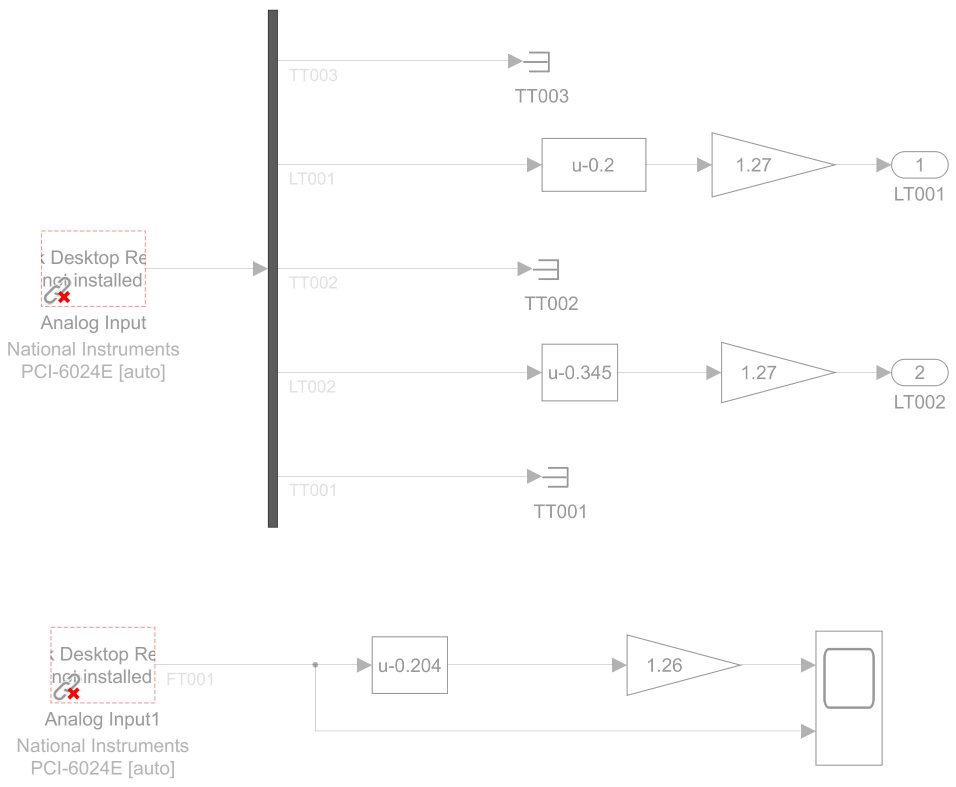


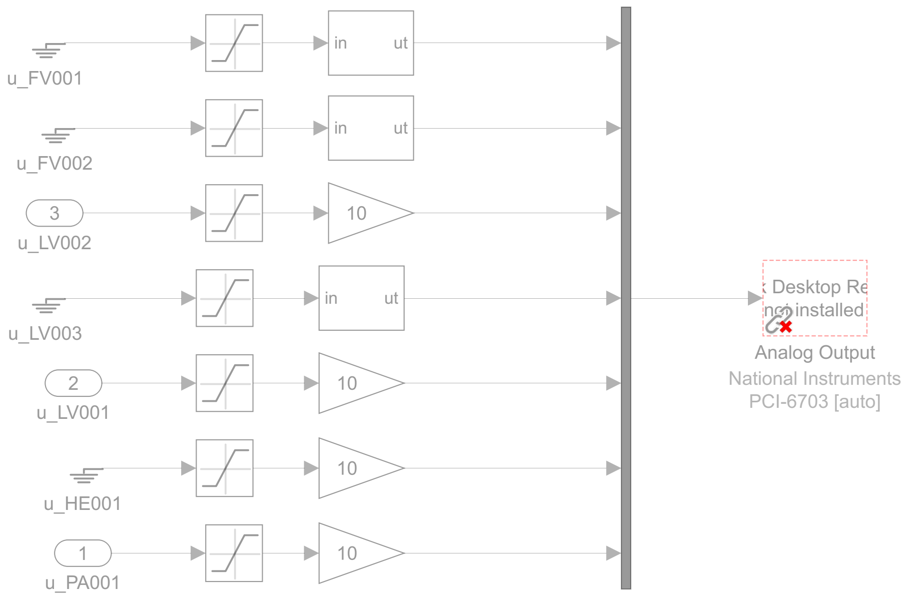
J.5 totank_mpc_and_mhe.slx


n
u1 if(u1 == 3)
if(u1 == 3)
u_PA001_hat_k 1
u_PA001 [-], estimate
u_PA001 [-]
Two-tank process
[u_PA001]
u_PA001
h1
u_LV001
h2
u_LV002
h1 [m], estimate
h1 [m], measurement
h1 [m], reference
[ref]
h2 [m], estimate
h2 [m], measurement
h2 [m], reference
nv_Linear]
[1x2]
z-1
z-1
u_PA001 [-]
u_LV001 [-]
u_LV002 [-]
h1 [m], measurement
h2 [m], measurement
k
block
convert
u+1
Interpreted MATLAB Fcn
NMHE
[h1_hat_k, h2_hat_k, u_PA001_hat_k]
Interpreted MATLAB Fcn
LMHE
[h1_hat_k, h2_hat_k, u_PA001_hat_k]
0 0
[x_est]
[ref]
[x_est]
[ref]
u_PA001]
x[k|k]
ref
md
MPC
mv
[mv_Linea
Linear MPC

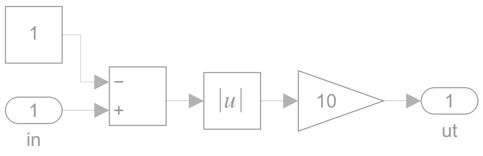

if(u1 == 3)
Action Port
1
u_PA001_hat_k
1

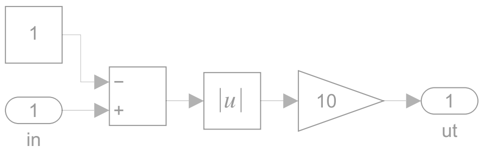

1
u_PA001
2
u_LV001
3
u_LV002
u_PA001
u_LV001
u_LV002
pådrag
LT001
LT002
målinger
1
h1
2
h2

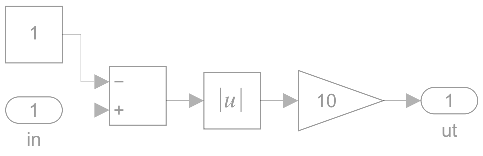

TT003
TT003
LT001
u-0.2
1.27
1
LT001
Desktop Re
not installed
Analog Input
National Instruments
PCI-6024E [auto]
TT002
TT002
LT002
u-0.345
1.27
2
LT002
TT001
TT001
Desktop Re
not installed
Analog Input1
National Instruments
PCI-6024E [auto]
FT001
u-0.204
1.26

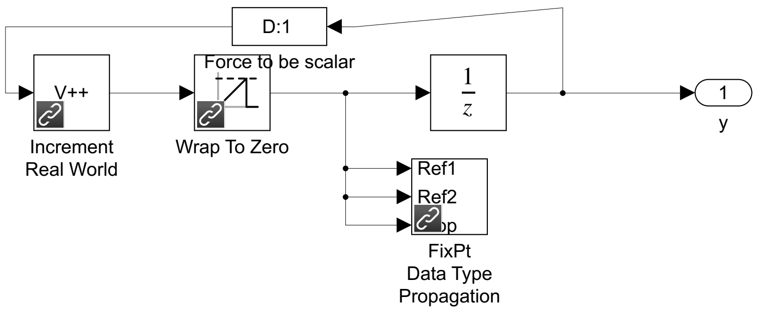


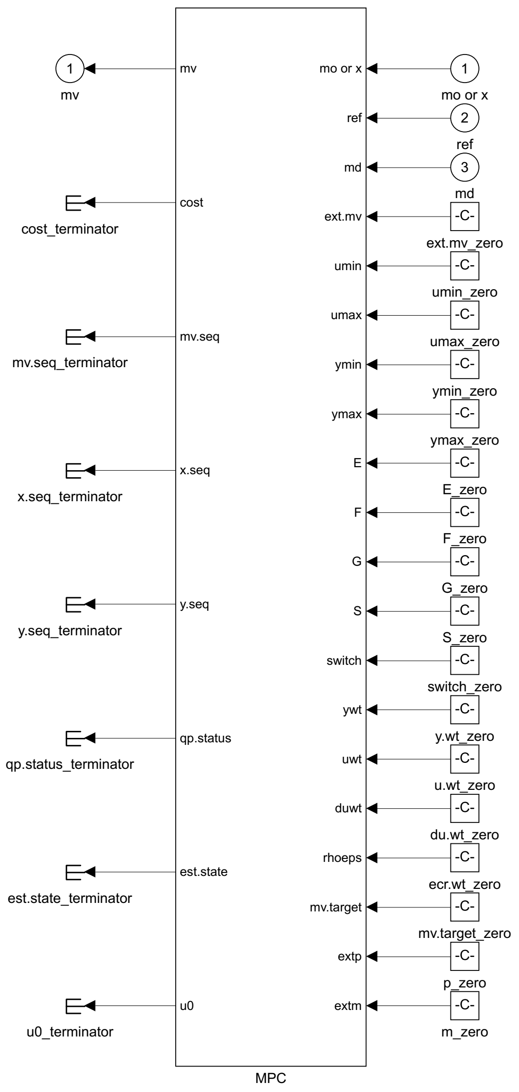
u_FV001
u_FV002
3
u_LV002
u_LV003
2
u_LV001
u_HE001
1
u_PA001
in
ut
10
Desktop Re
not installed
Analog Output
National Instruments
PCI-6703 [auto]


1
−
+
1
in
|u|
10
1
ut


1
1
in
−
+
|u|
10
1
ut


1
1
in
−
+
|u|
10
1
ut


D:1
Force to be scalar
V++
Increment
Real World
Wrap To Zero
1
z
1
y
Ref1
Ref2
op
FixPt
Data Type
Propagation


1
mv
cost_terminator
mv.seq_terminator
x.seq_terminator
y.seq_terminator
qp.status_terminator
est.state_terminator
u0_terminator
mv
cost
mv.seq
x.seq
y.seq
qp.status
est.state
u0
mo or x
ref
md
ext.mv
umin
umax
ymin
ymax
E
F
G
S
switch
ywt
uwt
duwt
rhoeps
mv.target
extp
extm
MPC
1
mo or x
2
ref
3
md
-C-
ext.mv_zero
umin_zero
umax_zero
ymin_zero
ymax_zero
E_zero
F_zero
G_zero
S_zero
switch_zero
y.wt_zero
u.wt_zero
du.wt_zero
ecr.wt_zero
mv.target_zero
p_zero
m_zero

Sample Times for 'totank_mpc_and_mhe'

Color	Annotation	Description	Value
⬛	Cont	Continuous	0
🟥	D1	Discrete 1	1.5
🟪	Inf	Constant	Inf
🟨	M	Multirate	N/A

Bibliography

[1] G. Tierie, *Cornelis Drebbel (1572-1633).* H. J. Paris, 1932.

[2] S. Bennett, "A brief history of automatic control," *IEEE Control Systems Magazine*, vol. 16, no. 3, pp. 17–25, 1996.

[3] D. S. Landes, *The Unbound Prometheus: Technological Change and Industrial Development in Western Europe from 1750 to the Present.* Cambridge University Press, 2nd ed., 2003.

[4] J. C. Maxwell, "On governors," *Proceedings of the Royal Society of London*, vol. 16, p. 270–283, 1868.

[5] S. Bennett, "Nicholas minorsky and the automatic steering of ships," *IEEE Control Systems Magazine*, vol. 4, no. 4, pp. 10–15, 1984.

[6] Adobe Acrobat Team, "Fast-forward : comparing a 1980s supercomputer to the modern smartphone," 2022. URL: https://blog.adobe.com/en/publish/2022/11/08/fast-forward-comparing-1980s-supercomputer-to-modern-smartphone, Accessed: 2023-04-12.

[7] J. Qin and T. Badgwell, "An overview of industrial model predictive control technology," *AIChE Symposium Series*, vol. 93, 1997.

[8] S. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control engineering practice*, vol. 11, no. 7, pp. 733–764, 2003.

[9] J. Richalet, A. Rault, J. L. Testud, and J. Papon, "Model predictive heuristic control: Applications to industrial processes," *Automatica*, vol. 14, no. 5, pp. 413–428, 1978.

[10] Intel, "Intel chips timeline." URL: https://www.intel.com/content/www/us/en/history/history-intel-chips-timeline-poster.html, Accessed: 2023-04-15.

[11] TSMC, "5nm technology." URL: https://www.tsmc.com/english/dedicatedFoundry/technology/logic/l_5nm, Accessed: 2023-04-15.

[12] R. Tedrake, “Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832),” 2023. URL: https://underactuated.csail.mit.edu, Accessed: 2023-05-03.

[13] R. E. Kalman *et al.*, “Contributions to the theory of optimal control,” *Bol. soc. mat. mexicana*, vol. 5, no. 2, pp. 102–119, 1960.

[14] B. Friedland, *Control system design : an introduction to state-space methods.* Mineola, New York: Dover Publications, 2005.

[15] H. Kwakernaak and R. Sivan, *Linear optimal control systems*, vol. 1072. Wiley-interscience New York, 1969.

[16] P. Whittle, *Risk-Sensitive Optimal Control.* Wiley Interscience Series in Systems and Optimization, Wiley, 1990.

[17] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital control of dynamic systems.* World Student Series, Menlo Park, Calif. ; Harlow, United States: Addison Wesley Longman, 3rd ed., 1998.

[18] J. B. Rawlings, “Tutorial overview of model predictive control,” *IEEE control systems magazine*, vol. 20, no. 3, pp. 38–52, 2000.

[19] S. Vichik and F. Borrelli, “Solving linear and quadratic programs with an analog circuit,” *Computers & Chemical Engineering - Manfred Morari Special Issue*, vol. 70, pp. 160–171, 2014.

[20] D. E. Seborg, T. F. Edgar, D. A. Mellichamp, and F. J. Doyle III, *Process dynamics and control.* John Wiley & Sons, 4th ed., 2016.

[21] F. Haugen, *Dynamiske systemer: modellering, analyse og simulering.* Tapir, 3rd ed., 2007.

[22] W. S. Levine, *The Control Handbook: Control System Advanced Methods.* CRC Press, 2nd ed., 2018.

[23] A. Bemporad, N. L. Ricker, and M. Morari, “Model Predictive Control Toolbox - User’s Guide,” 2023. URL: https://se.mathworks.com/help/pdf_doc/mpc/mpc_ug.pdf.

[24] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control : Theory and Algorithms.* Communications and Control Engineering, Springer London, 1st ed., 2011.

[25] J. Nocedal and S. J. Wright, *Numerical Optimization.* New York, United States: Springer, 1999.

[26] T. Drengstig, “Ele320 totank1 motivasjon modellering,” 2020. Lecture notes.

[27] G. Luta and J. H. Aarvåg, *Studie på anvendeligheten av Carleman embedding til kontroll av vannivået i en tank*. University of Stavanger, 2021. Bachelor thesis. URL: https://hdl.handle.net/11250/2985975.

[28] S. SARCO, “Control valve characteristics.” URL: https://www.spiraxsarco.com/learn-about-steam/control-hardware-electric-pneumatic-actuation/control-valve-characteristics, Accessed: 2023-05-17.

[29] R. A. Adams and C. Essex, *Calculus : a complete course*. Don Mills, Ont: Pearson, 9th ed., 2018.

[30] J. J. Duistermaat and J. A. C. Kolk, “Taylor expansion in several variables,” in *Distributions*, Cornerstones, Boston: Birkhäuser Boston, 2010.

[31] A. Hiorth, “Modeling and computational engineering,” 2022. Lecture notes.

[32] C. Edwards and D. Penny, *Elementary linear algebra*. Harlow: Pearson Education, 2010.

[33] S. Haykin and B. V. Veen, *Signals and systems*. New York: Wiley, 2nd ed., 2003.

[34] A. R. Hambley, *Electrical Engineering: Principles & Applications*. Harlow, United Kingdom: Pearson Education UK, 7th ed., 2018.

[35] J. R. Dormand and P. J. Prince, “A family of embedded runge-kutta formulae,” *Journal of Computational and Applied Mathematics*, vol. 6, no. 1, pp. 19–26, 1980.

[36] H. Moore, *MATLAB for engineers*. Boston: Pearson, 4th ed., 2015.

[37] The MathWorks Inc., *MATLAB version: 9.11.0 (R2021b)*. Natick, Massachusetts, United States: The MathWorks Inc., 2021. URL: https://www.mathworks.com.

[38] A. Bemporad, “A multiparametric quadratic programming algorithm with polyhedral computations based on nonnegative least squares,” *IEEE transactions on automatic control*, vol. 60, no. 11, pp. 2892–2903, 2015.

[39] G. Bekerytė, *Moving Horizon Estimation for the Two-tank System*. University of Stavanger, 2023. Master thesis.

[40] E. Bwambale, F. K. Abagale, and G. K. Anornu, “Data-driven model predictive control for precision irrigation management,” *Smart Agricultural Technology*, vol. 3, p. 100074, 2023.

[41] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll, "Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2397–2404, 2023.

[42] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye, *Probability & Statistics for Engineers & Scientists.* Harlow, United Kingdom: Pearson, 9th ed., 2016.