

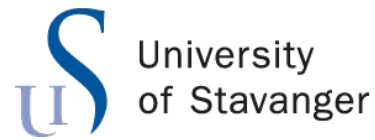


University of
Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Master in Robot Technology and Signal Processing	Spring semester, 2023 <u>Open</u> / Restricted access
Writer: Greta Bekerytė	<i>Greta Bekerytė</i> (Writer's signature)
Faculty supervisor: Damiano Rotondo External supervisor(s):	
Thesis title: Moving Horizon Estimation for the Two-tank System	
Credits (ECTS): 30	
Key words: Moving Horizon Estimation, LMHE, NMHE, KF, EKF, state estimator, optimization, two-tank system, nonlinear system	Pages:120..... + enclosure:101..... Stavanger,15/06/2023... Date/year



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

Moving Horizon Estimation for the Two-tank System

Master's Thesis in Robot Technology and Signal Processing

by

Greta Bekerytė

Internal Supervisors

Damiano Rotondo

June 14, 2023

Declaration of Authorship

I, Greta Bekerytė, declare that this thesis titled, ‘Moving Horizon Estimation for the Two-tank System’ and the work presented in it are my own. I confirm that:

- This work was done wholly 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: *Greta Bekeryte*

Date: *15.06.2023*

“Not everything that can be counted counts, and not everything that counts can be counted.”

Albert Einstein

Abstract

This thesis presents the application and evaluation of Moving Horizon Estimation (MHE) for the nonlinear two-tank system. MHE is an iterative optimization-based approach that continuously updates the estimates of the states by solving an optimization problem over a fixed-size, receding horizon. Linear and nonlinear MHE-based estimators are designed and implemented in Matlab for evaluation in simulation environment and Simulink for on-line realization and validation. The linear and nonlinear MHE are evaluated in comparison with the Kalman and Extended Kalman filter through extensive simulations and experimental validation, assessing their accuracy, efficiency, and overall performance. The results of the two-tank state and unmeasured disturbance estimation shows the benefit of the MHE.

Acknowledgements

I would like to express my deepest appreciation to my supervisor Damiano Rotondo providing with guidance, insightful comments, suggestions and patience throughout the duration of this project.

I would also like to thank my family for their support. A particular thanks goes to Bernardas, without your tremendous understanding, patience and encouragement in the past few years, it would be impossible for me to complete my study.

Finally, I wish to express gratitude to my colleagues and supervisors at Zaptec, for encouragement and giving me the flexibility to take classes while working.

Contents

Declaration of Authorship	iii
Abstract	v
Acknowledgements	vi
List of Figures	xi
List of Figures	xv
List of Tables	xv
List of Tables	xvii
Abbreviations	xvii
Symbols	xix
1 Introduction	1
1.1 Background and Motivation	1
1.2 Objectives	2
1.3 Approach and Contributions	3
1.4 Outline	3
2 State Estimation	5
2.1 Dynamical System Representation and Notation	6
2.2 Observability and Detectability	7
2.3 Luenberger Observer	10
2.4 Kalman Filter	11
2.5 Extended Kalman Filter	14
2.6 Full Information Estimation	15
2.7 Moving Horizon Estimation	17
2.7.1 Arrival Cost Update	18
2.7.2 MHE Constraints	20
2.7.3 Estimation of System Parameters and Disturbances	20
2.7.4 Tuning Parameters	21

3	Optimization	23
3.1	Optimization Problem	23
3.2	Optimization Classes and Methods	25
3.2.1	Least-Squares Optimization	26
3.2.2	Linear Programming	27
3.2.3	Convex Programming	28
3.2.4	Nonlinear Programming	32
3.3	Matlab In-built Functions	35
4	Implementation	37
4.1	Linear Moving Horizon Estimation	37
4.1.1	Linearization of Two-tank System	40
4.1.2	Implementation	44
4.1.3	In-built <i>fmincon</i> for Constrained Optimization	49
4.1.4	Verification of LMHE	51
4.2	Nonlinear Moving Horizon Estimation	58
4.2.1	Implementation	58
4.2.2	Verification of NMHE	58
4.3	Simulink Implementation	63
5	Results	67
5.1	Simulation Environment	67
5.1.1	LMHE	67
5.1.2	NMHE	72
5.1.3	MHE Efficiency	77
5.2	Real Data Estimation	82
5.3	Experimental MPC and MHE Results	92
6	Conclusions	97
A	The Two-tank System	99
A.1	Process Description	99
A.2	Valves and Pump Characteristics	101
A.3	Dynamic Model of Tank 1	104
A.4	Dynamic Model of Tank 2	104
B	Additional Results	107
B.1	NMHE Verification	108
B.2	Simulation Environment Results	110
B.2.1	LMHE	110
B.2.2	NMHE	115
B.2.3	LMHE and NMHE (increased measurement noise)	121
B.3	Real Data Estimation Results	128
B.3.1	LMHE	128
B.3.2	NMHE	129
B.3.3	On-line Results	131

C	Matlab Code	135
C.1	LMHE.m	135
C.2	NMHE.m	154
C.3	LMHE_Simulink_init.m	168
C.4	NMHE_Simulink_init.m	170
C.5	Functions	173
C.5.1	Amatrix.m	173
C.5.2	ExtendedKalmanFilter.m	174
C.5.3	KalmanFilter.m	176
C.5.4	linearization.m	177
C.5.5	ObjectiveFunction.m	179
C.5.6	LMHE_simulink.m	182
C.5.7	NMHE_simulink.m	186
C.5.8	parameters.m	191
D	Project Poster and Project Plan	193
	Bibliography	197

List of Figures

2.1	State observer structure.	10
2.2	Kalman filter loop.	13
2.3	Extended Kalman filter loop.	15
2.4	Moving Horizon Estimation.	17
3.1	Optimization problem.	24
3.2	Example of convex and non-convex functions.	25
3.3	Example of a linear programming polytope together with a possible path (red) taken by the simplex method to solve the corresponding LP [1].	28
4.1	Eigenvalues of the observability Grammian plotted over the feasible region of the two-tank system. The subplot A represents the eigenvalues of u_{PA001} and h_1 when one measurement is available. Subplot B compares the eigenvalues of u_{PA001} when one measurement is available versus when two measurements are available.	44
4.2	An overview of Moving Horizon Estimation iteration.	46
4.3	Typical <i>fmincon</i> optimization process.	50
4.4	LMHE Case 1: Estimation of h_1 and h_2 given the measurements of h_1 and h_2 . No process or measurement noise.	52
4.5	LMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . No process or measurement noise.	53
4.6	LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . No process or measurement noise.	54
4.7	LMHE Case 3 estimation using different weights in the model covariance matrix. The second element (representing h_2) is kept constant, $Q_2 = 1$	56
4.8	LMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . No process or measurement noise.	57
4.9	NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . No process or measurement noise.	61
4.10	NMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . No process or measurement noise.	62
4.11	Simulink model for the two-tank system with LMHE and NMHE.	63
4.12	Simulink implementation of LMHE and NMHE.	64
5.1	LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$	68
5.2	LMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$	70

5.3	LMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. Sample time increased to 0.5s.	72
5.4	NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$	73
5.5	EKF Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$	74
5.6	NMHE Case 3: u_{PA001} estimation using global optimization methods. Simulated measurement noise, $\sigma^2 = 10^{-9}$	75
5.7	NMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. Sample time increased to 0.5s.	76
5.8	Absolute values of the estimation error variations using <i>sqp</i> , <i>sqp</i> with reduced tolerances, <i>active-set</i> and <i>interior-point</i> algorithms. The <i>active-set</i> performance is equal to <i>interior-point</i> , therefore invisible.	79
5.9	Absolute values of the estimation error variations for the horizon length $N = 5$, $N = 10$, $N = 15$ and $N = 20$	80
5.10	Training dataset used to determine measurement noise covariance matrix. Dataset <i>data3.mat</i>	82
5.11	LMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Dataset <i>data1.mat</i>	83
5.12	LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Dataset <i>data1.mat</i>	84
5.13	LMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . Dataset <i>data1.mat</i>	86
5.14	NMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Dataset <i>data1.mat</i>	87
5.15	https://youtu.be/AeMLxtVhUqw	88
5.16	NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Dataset <i>data1.mat</i>	89
5.17	Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Dataset <i>data2.mat</i>	90
5.18	Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Dataset <i>data2.mat</i>	91
5.19	Simulink model for the two-tank system with MHE and MPC.	93
5.20	Control of the two-tank system with LMPC using LMHE. Case 1: estimation of h_1 and h_2 given the measurements h_1 and h_2	94
5.21	Control of the two-tank system with LMPC using LMHE. Case 2: estimation of h_1 , h_2 given the measurement h_2	95
5.22	Control of the two-tank system with LMPC using LMHE. Case 3: estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2	96
A.1	A schematic sketch and picture of the two-tank system located at the University of Stavanger, KE E-459.	100
A.2	A simplified version of the two-tank system schematic sketch.	101
A.3	Valve characteristic for LV001 and LV002[2].	103
A.4	Pump PA001 characteristic [2].	103
A.5	Simplified sketch of tank 2.	105

B.1	NMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . No process or measurement noise.	108
B.2	NMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . No process or measurement noise.	109
B.3	LMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. <i>Sqp</i> algorithm with the horizon length $N = 10$	110
B.4	LMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. <i>Sqp</i> algorithm with the horizon length $N = 10$	111
B.5	LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. <i>Sqp</i> algorithm with the horizon length $N = 10$. <i>OptimalityTolerance</i> = <i>StepTolerance</i> = 10^{-10}	112
B.6	LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. <i>Sqp</i> algorithm with the horizon length $N = 5$	113
B.7	LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. <i>Interior – point</i> algorithm with the horizon length $N = 10$	114
B.8	NMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. <i>Sqp</i> algorithm with the horizon length $N = 10$	115
B.9	NMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. <i>Sqp</i> algorithm with the horizon length $N = 10$	116
B.10	NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. <i>Active-set</i> algorithm with the horizon length $N = 10$	117
B.11	NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. <i>MultiStart</i> method with 5 start points using <i>sqp</i> algorithm with the horizon length $N = 10$	118
B.12	NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. <i>GlobalSearch</i> method using <i>sqp</i> algorithm with the horizon length $N = 10$	119
B.13	NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. <i>PatternSearch</i> method using <i>sqp</i> algorithm with the horizon length $N = 10$	120
B.14	LMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.	121
B.15	LMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.	122
B.16	LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.	123
B.17	LMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.	124
B.18	NMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.	125
B.19	NMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.	126

B.20 NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s. . .	127
B.21 LMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . <i>Sqp</i> algorithm with the horizon length $N = 10$. Dataset <i>data1.mat</i> . . .	128
B.22 NMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . <i>Sqp</i> algorithm with the horizon length $N = 10$. Dataset <i>data1.mat</i> . . .	129
B.23 Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . Dataset <i>data2.mat</i>	130
B.24 LMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . <i>Sqp</i> algorithm with the horizon length $N = 10$	131
B.25 NMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . <i>Sqp</i> algorithm with the horizon length $N = 10$	132
B.26 LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . <i>Sqp</i> algorithm with the horizon length $N = 10$	133
B.27 NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . <i>Sqp</i> algorithm with the horizon length $N = 10$	134

List of Tables

5.1	Time usage of the LMHE using different algorithms, parameters and horizon lengths.	78
5.2	PC-1 time usage of the NMHE using different methods.	78
5.3	Estimation error means for the NMHE, EKF, LMHE and KF in Case 1, 2 and 3. Dataset <i>data2.mat</i>	91
A.1	Tank 2 dimensions [2].	105
A.2	Two-tank system variables [2].	106

Abbreviations

BFGS	B royden- F letcher- G oldfarb- S hanno
EKF	E xtended K alman F ilter
KF	K alman F ilter
KKT	K arush- K uhn- T ucker
LMHE	L inear M oving H orizon E stimation
LP	L inear P rogramming
LTI	L inear T ime I nvariant
MHE	M oving H orizon E stimation
MPC	M odel P redictive C ontrol
NMHE	N onlinear M oving H orizon E stimation
PHB	P opov- B elevitch- H autus
ODE	O rdinary D ifferential E quations
SQP	S uccessive Q uadratic P rogramming

Symbols

The list of the symbols is not exhaustive.

\mathbb{R}	Set of real numbers.
\mathbb{R}^n	Euclidean space of dimension n .
$\mathbb{R}^{n \times n}$	Matrix dimension $n \times n$.
\mathbb{X}	Set of constraints on the states.
\mathbb{W}	Set of constraints on the process noise.
\mathbb{V}	Set of constraints on the measurement error.
e	Estimation error.
u	Vector of control inputs.
v	Vector of measurement error.
w	Vector of non-modelled process disturbances.
x	Vector of system states.
x_0	Vector of initial system states.
\bar{x}	A priori state estimate.
\hat{x}	A posteriori state estimate.
y	Vector of outputs.
A	State transition matrix.
B	Input matrix.
C	Output matrix.
D	Feed-through matrix.
H	Hessian matrix.
J	Objective function.

K	Kalman gain matrix.
L	Luenberger observer gain matrix.
N	Horizon Length.
O	Observability matrix/Operating point.
P, P^{-1}	Prior weighting and confidence matrices.
R, R^{-1}	Measurement covariance and confidence matrices.
Q, Q^{-1}	Process covariance and confidence matrices.
W	Observability Grammian matrix.
$\hat{(\cdot)}$	Optimization variable.
$\tilde{(\cdot)}$	Trajectory.
$\dot{(\cdot)}$	Derivative.
$(\cdot)^*$	Optimal solution.
$(\cdot)^T$	Matrix/vector transposition.
$(\cdot)_k$	Discrete-time variable at time k .
$(\cdot)_{i k}$	Discrete-time variable at time i calculated at time k .
$\ x\ _2^2$	Euclidean norm $x^T x$.
$\ x\ _W^2$	Weighted Euclidean norm $x^T W x$.
$\Gamma(\cdot)$	Arrival cost function.
$\nabla(\cdot)$	Gradient.
$L(\cdot)$	Langrangian function.
Δ	Change in variable.
λ	Langrange multiplier/eigenvalue.
ϵ	Tolerance.

Chapter 1

Introduction

1.1 Background and Motivation

In many physical systems it is often impossible to directly measure all the necessary states. This may be attributed to the inaccessibility of certain states, unobservability of specific variables due to limitations in the available sensors or measurement techniques, complexity of the system, or high cost associated with deploying multiple sensors. These systems therefore often require inferring unobservable variables or augmenting limited data to obtain a comprehensive understanding of the systems state using state estimators. Additionally, in environments characterized by high levels of noise, employing a state estimator can be advantageous for improving the accuracy and reliability of measurements used in decision-making processes, control strategies, or system monitoring.

The Kalman filter is widely used to estimate the state of linear unconstrained systems in the presence of noisy measurements and unmeasured process disturbances. For the systems corrupted with white Gaussian noise, the Kalman filter operates as an optimal estimator. However, in the majority of real-world applications, the systems under consideration are nonlinear and can not directly utilize the Kalman filter. The Extended Kalman filter is a natural extension of the Kalman filter which iteratively linearize the nonlinear system around the current state estimate. Nevertheless, the extended Kalman filter is not an optimal estimator and its stability is not guaranteed. In some cases, additional information about the process is available in the form of equality or inequality

constraints. However, when incorporating these constraints, general recursive solutions like Kalman filtering are not applicable [3].

Moving Horizon Estimation is an optimization-based approach that considers the constraints as part of the optimization problem over a fixed-size estimation window of most recent measurements and control inputs. An increasing success and adoption of model predictive control (MPC), which shares a conceptual similarity with the MHE, has sparked a growing interest in MHE as well. The MHE can improve the state estimation performance, however, at the cost of increased computational load and is thus, more suitable for processes where there are sufficient computational resources available, e.g., in the case of systems with slow dynamics. This project aims at designing and implementing an MHE-based state estimator to observe the state of the two-tank system available at the University of Stavanger in the laboratory KE E-458 and compare its performance against more conventional state estimation techniques.

1.2 Objectives

The main objectives in this thesis can be segmented into the following:

- Literature study and analysis of the state of the art of Moving Horizon Estimation.
- Study and analysis of more conventional state estimation strategies such as the Luenberger observer, the Kalman Filter and the Extended Kalman filter.
- Study and analysis of the fundamental principles behind optimization techniques.
- Implementation and evaluation of linear and nonlinear MHE for the two-tank system in simulation environment.
- Implementation and evaluation of Kalman and Extended Kalman filter for the two-tank system in simulation environment.
- Implementation and experimental validation of the designed MHE-based estimator on the two-tank system located at the University of Stavanger in the laboratory KE E-458.
- Comparison of MHE against Kalman and Extended Kalman filter.

1.3 Approach and Contributions

In this work linear and nonlinear MHE is considered for the two-tank state and disturbance estimation. Implementation is done in Matlab for evaluation in simulation environment and Simulink for real-time estimation. The MHE-based state estimator performance is compared against more conventional state estimation strategies, namely the Kalman and Extended Kalman filter.

Prior to the project start, the project plan was developed to outline the key milestones, activities and timelines. Throughout the duration of the project, the execution adhered closely to the initial plan, with some minor deviations that were efficiently managed. The project plan is available in Appendix D.

1.4 Outline

This thesis is organized as follows:

- **Chapter 2** begins by establishing the theoretical background on observability and detectability notions. Subsequently, the chapter delves into the discussion of the most commonly used estimation techniques, namely the Luenberger observer, Kalman Filter, and Extended Kalman Filter. Each technique is presented with its underlying principles, mathematical formulation. The chapter then progresses to introduce the Full Information Estimation framework followed by a more detailed theoretical presentation on Moving Horizon Estimation, providing an in-depth understanding of this estimation approach.
- **Chapter 3** starts by providing theoretical background on optimization problem formulation. Next, it offers an overview of commonly used optimization classes, including least-squares optimization, linear programming, convex programming, and nonlinear programming. Each class is discussed in terms of its characteristics, and specific problem formulations. Furthermore, the chapter delves into the fundamental theoretical principles behind the most widely used optimization algorithms employed to solve these optimization problems. It covers algorithms such as gradient descent, Newton's method, Interior-point and SQP.

- **Chapter 4** begins with a more detailed theoretical overview of linear Moving Horizon Estimation (LMHE), providing a deeper understanding of the underlying principles, followed by an implementation overview, outlining the steps and considerations for implementing the LMHE. Then the implementation verification is provided. Next, the chapter transitions to nonlinear Moving Horizon Estimation (NMHE). The implementation overview section follows, offering insights into the practical implementation aspects. Then the implementation verification of NMHE is provided. Lastly, the chapter concludes with a brief overview of the Simulink implementation.
- Chapter 5 presents the evaluation of MHE performance in various scenarios. The chapter begins with an analysis of the results obtained from the simulation environment, showcasing the estimation accuracy and robustness of linear and nonlinear MHE techniques compared with the performance of Kalman filter and Extended Kalman filter. Furthermore, the MHE efficiency is examined, considering computational time and overall performance. Further, the chapter proceeds to evaluate the estimation results using real two-tank data. These results are again compared with the performance of the Kalman filter and Extended Kalman filter. Lastly, the chapter concludes with experimental the MPC and MHE results.
- Appendix A provides detailed description of the two-tank system and derive the mathematical expressions of the dynamical process model.
- Appendix B contains additional results.
- Appendix C provides the Matlab code.
- Appendix D contains the project poster and project plan.

Chapter 2

State Estimation

Information on the variables that uniquely defines the state of the system at any given time is the essential condition for effective monitoring and control of a process. In most practical applications, due to the lack of sensors, all physical states of the system are seldom accessible. If observed states of a system are sufficient to determine the dynamic of the system, unmeasurable states can be reconstructed using a state estimator. The use of state estimator is also beneficial if measurements are corrupted by excessive levels of noise. State estimation objective is to accommodate a model prediction with available measurement information, considering model error, noisy or incomplete measurements and disturbances. The primary aim is to obtain a state estimate such that the difference between the actual and estimated value decays to zero asymptotically. However, depending on the application, the requirement for the state estimate may vary. Instead, the desired behavior could be that the error decays to zero within a finite time or that the error remains bounded within certain limits by taking into account the modeling uncertainties and the properties of noise and disturbances. In control theory, a dynamic system that provides a state trajectory estimate for a given real system is called a state observer (or state estimator)[4][5].

This chapter provides some relevant background on observability and detectability notions, Section 2.2, most commonly used state estimation techniques developed throughout the years, namely the Luenberger Observer, Section 2.3, the Kalman filter, Section 2.4, the Extended Kalman filter, Section 2.5, and the Full Information Estimation, Section 2.6.

Section 2.7 delves into more detailed theoretical presentation on the Moving Horizon Estimation.

2.1 Dynamical System Representation and Notation

A general dynamical system model consists of two equations. The first equation is a state transition equation that describes the dynamics of the system states and consists of a set of ordinary differential equations (ODE). The second is an output equation, that describes measurement mapping of the system.

The following dynamical system representation and notation will be used throughout the report:

- Nonlinear continuous-time invariant system:

$$\begin{cases} \dot{x}(t) = f(x, u, t) + w(t) \\ y(t) = h(x, u, t) + v(t) \end{cases}, \quad (2.1)$$

where $x \in \mathbb{R}^n$ is the vector of the states, $u \in \mathbb{R}^p$ is the vector of the control inputs, $y \in \mathbb{R}^m$ is the vector of the outputs, $w \in \mathbb{R}^n$ is the vector of non-modelled or unknown process disturbance (model error), $v \in \mathbb{R}^m$ is the vector of measurement error, $f(\cdot)$ and $h(\cdot)$ denotes nonlinear functions and $t \in \mathbb{R}$ is the time. In this paper, the probabilistic distribution of the process disturbance w and measurement noise v are considered to be zero-mean Gaussian white-noise. Furthermore, the dynamic model equations are assumed to be disturbed by additive process and measurement noise.

- Linear continuous-time invariant system:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) + w(t) \\ y(t) = Cx(t) + Du(t) + v(t) \end{cases}, \quad (2.2)$$

where $A \in \mathbb{R}^{n \times n}$ is the state transition matrix, $B \in \mathbb{R}^{n \times p}$ is the input matrix, $C \in \mathbb{R}^{m \times n}$ is the output matrix and $D \in \mathbb{R}^{m \times p}$ is the feed-through matrix. In this paper the feed-through matrix is considered to be zero, such that (2.2) is reduced

to:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) + w(t) \\ y(t) = Cx(t) + v(t) \end{cases} \quad (2.3)$$

- In practise, the majority of the applications uses a discrete-time system representation. This is due to the fact that state measurements are often sampled and processed digitally. Discrete-time invariant system representations used in this report are

$$\begin{cases} x_{k+1} = Ax_k + Bu_k + w_k \\ y_k = Cx_k + v_k \end{cases} \quad (2.4)$$

for the linear case, and

$$\begin{cases} x_{k+1} = f(x, u, k) + w_k \\ y_k = h(x, u, k) + v_k \end{cases} \quad (2.5)$$

for nonlinear case. k is the discrete time index, such that $t_k = kT_s$, where T_s is the sampling time.

2.2 Observability and Detectability

Observability and detectability are important notions to be considered when designing a state estimator. The concept of observability for linear dynamic systems was introduced by Rudolf E. Kalman in 1960 [6]. Observability concept considers the ability to estimate the state of a system from its measured outputs. Detectability, on the other hand, considers the ability to estimate the state of a system with a bounded error. Detectability concept was introduced by Rudolf E. Kalman, Peter L. Falb and Michael A. Arbib in 1969 [7].

Definition 2.1 (Detectability). A system is said to be detectable if all unobservable states are asymptotically stable.

Definition 2.2 (Observability for linear systems). A linear continuous¹-time invariant system

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t), x(t_0) = x_0 \\ y(t) = Cx(t) \end{cases} \quad (2.6)$$

¹Equivalent definition and theorems for linear discrete-time invariant systems can be found in [8] p.171.

is said to be observable if, for a given time interval $[t_0, t_f]$, it is possible to uniquely determine any initial state x_0 by using the input $u(t)$ and the output $y(t)$ for $t \in [t_0, t_f]$. Otherwise, the system is said to be unobservable.

Observability is given by the following theorem [8]:

Theorem 2.3. *The state equation (2.6) is observable if and only if the $n \times n$ observability Grammian matrix*

$$W_0(t) = \int_0^t e^{A^T \tau} C^T C e^{A \tau} d\tau \quad (2.7)$$

is non singular for any $t > 0$.

If the matrix A is stable, the observability Grammian can be found as the solution to Lyapunov equation given by

$$A^T W_0 + W_0 A = -C^T C. \quad (2.8)$$

Theorem 2.4. *The system is observable if matrix A is stable, and the unique solution of (2.8) is positive definite, i.e., all eigenvalues of W_0 are strictly positive.*

The observability Grammian provides information on observability "degree" in different state space directions. E.g., eigenvalues to the observability Grammian that are close to zero corresponds to low observability. In practise, instead of calculating the observability Grammian, observability of a system can be determined using simpler equivalent statements than Theorems 2.3 and 2.4 [8]:

Theorem 2.5.

1. *The n -dimensional pair (A, C) is observable.*
2. *The LTI system (2.6) is observable if and only if $nm \times n$ observability matrix*

$$O = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \quad (2.9)$$

has rank n (full column rank).

3. The LTI system (2.6) is observable if and only if $(n + m) \times n$ matrix

$$\begin{bmatrix} A - \lambda I \\ C \end{bmatrix} \quad (2.10)$$

has full column rank for every eigenvalue of A .

The item 3. in Theorem 2.5 is also called a Popov-Belevitch-Hautus (PHB) test.

The PHB test for detectability states that [9]:

Lemma 2.6.

1. The n -dimensional pair (A, C) is detectable.

2. The LTI system (2.6) is asymptotically observable, i.e., detectable if and only if $(n + m) \times n$ matrix

$$\begin{bmatrix} A - \lambda I \\ C \end{bmatrix} \quad (2.11)$$

has full column rank for every eigenvalue of A for which $\Re(\lambda) \geq 0$.

Observability of the system is a sufficient condition for state variables estimation, while detectability is a necessary and sufficient condition. However, observability is a necessary and sufficient condition for an estimator with fully adjustable convergence rate to exist, meaning that, if such an estimator exists, measurable state variables contain useful indirect information on all state variables of the system.

For linear systems both observability and detectability are global properties that do not depend on the operating region. The evaluation of observability and detectability for nonlinear systems is more challenging than for linear systems. In this case, observability/detectability are local properties that do depend on the operating region [4].

2.3 Luenberger Observer

The Luenberger observer is a deterministic state estimation technique introduced by David G. Luenberger in 1964 [10]. For linear discrete-time dynamic system

$$\begin{cases} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{cases} \quad (2.12)$$

a linear system observer is given by

$$\begin{cases} \hat{x}_{k+1} &= A\hat{x}_k + Bu_k + L(y_k - \hat{y}_k) \\ \hat{y}_k &= C\hat{x}_k \end{cases}, \quad (2.13)$$

where \hat{x} is the state estimate, $L \in \mathbb{R}^{n \times m}$ is the observer gain, \hat{y} is the observer output and $(y_k - \hat{y}_k)$ is a correction of the estimation equation with a feedback from estimation error. The state observer structure is shown in Fig. 2.1. The state estimation error at

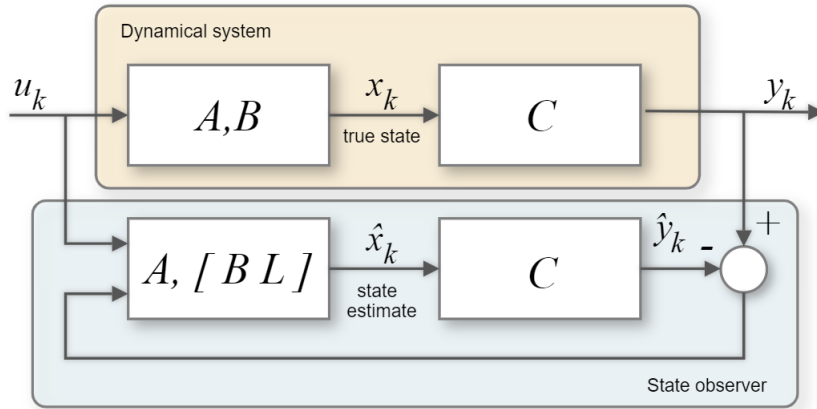


Figure 2.1: State observer structure.

time $k + 1$ is given by

$$e_{k+1} = x_{k+1} - \hat{x}_{k+1}. \quad (2.14)$$

Substituting (2.12) and (2.13), the observer error in (2.14) becomes

$$e_{k+1} = (A - LC)e_k. \quad (2.15)$$

The observer is called asymptotically stable if the estimation error converges to zero when $k \rightarrow \infty$, i.e., the state estimate \hat{x} converges to the true state of x . For discrete-time system, the Luenberger observer is asymptotically stable if the eigenvalues of the matrix

$A - LC$ are placed inside the unit circle. The eigenvalues of $A - LC$ can be placed arbitrary by appropriate choice of the observer gain L , if the pair (A, C) is observable. The Luenberger estimator is a straightforward method for state estimation, however, it relies on an accurate mathematical system model and is not designed to account for process noise.

2.4 Kalman Filter

The Kalman filter is an optimization based estimation approach introduced by Rudolf E. Kalman in 1960 [11]. It is one of the most important and unique accomplishments in estimation theory, and is widely used in numerous applications of science and engineering. The objective of the Kalman filter, also called the linear quadratic estimator or the linear least squares estimator, is to minimize the mean-squared estimation error for a linear dynamic system with white Gaussian disturbance and measurement noise. The filter contains a set of recursive equations that provides statistically optimal estimates [12].

Considering a linear unconstrained discrete-time dynamic system:

$$x_{k+1} = Ax_k + Bu_k + w_k, \quad (2.16a)$$

$$y_k = Cx_k + v_k, \quad (2.16b)$$

where $w_k \in \mathbb{R}^{n \times n}$ and $v_k \in \mathbb{R}^{m \times m}$ are the process and measurement noise respectively. w_k and v_k are assumed to be uncorrelated, zero mean, $E\{w_k\} = E\{v_k\} = 0$, Gaussian variables with covariance matrices Q and R , respectively.

The Kalman filter consists of two phases, namely prediction and correction. It considers the following state observer equations:

$$\bar{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Bu_k, \quad (2.17a)$$

$$\hat{x}_{k|k} = \bar{x}_{k|k-1} + K_k(y_k - \bar{x}_{k|k-1}), \quad (2.17b)$$

where $\bar{x}_{k|k-1}$ is the a priori state estimate at sample k given knowledge of the system prior to sample k , i.e., measurement at sample k is not taken into account. $\hat{x}_{k|k}$ is the a posteriori state estimate at sample k given measurement y_k , i.e., a linear combination of prediction and a weighted difference between a prediction and an actual measurement.

The term $(y_k - C\bar{x}_{k|k-1})$ is also called measurement innovation, or residual. K_k is the time-varying gain. The a priori and a posteriori estimation errors can then be defined as

$$e_{k|k-1} \equiv x_k - \bar{x}_{k|k-1}, \quad (2.18a)$$

$$e_{k|k} \equiv x_k - \hat{x}_{k|k}. \quad (2.18b)$$

The a priori and a posteriori estimate error covariance matrices are given by

$$P_{k|k-1} = E\{e_{k|k-1}e_{k|k-1}^T\}, \quad (2.19a)$$

$$P_{k|k} = E\{e_{k|k}e_{k|k}^T\}. \quad (2.19b)$$

The goal is to find the gain factor K_k such that it minimizes the a posteriori error covariance in (2.19b). Minimization can be achieved by combining equations (2.16b), (2.17b), (2.18b) and (2.19b), performing the indicated expectations and optimizing with respect to K_k , the details can be found in [12]. Hence, the optimal Kalman gain K_k that minimizes the mean square estimation error is given by

$$K_k = P_{k|k-1}C^T(CP_{k|k-1}C^T + R)^{-1} \quad (2.20)$$

and the covariance matrix associated with the optimal estimate is

$$P_{k|k} = P_{k|k-1} - P_{k|k-1}C^T(CP_{k|k-1}C^T + R)^{-1}CP_{k|k-1}. \quad (2.21)$$

Assuming that w_k in (2.16a) is uncorrelated and it has zero mean, the next prediction is given by

$$\bar{x}_{k+1|k} = A\hat{x}_{k+1|k} + Bu_k. \quad (2.22)$$

The a priori covariance matrix associated with the next prediction $\bar{x}_{k+1|k}$ can be expressed substituting $e_{k+1|k} = x_{k+1} - \hat{x}_{k+1|k}$:

$$P_{k+1|k} = E\{e_{k+1|k}e_{k+1|k}^T\} = AP_kA^T + Q. \quad (2.23)$$

Thus, (2.17b), (2.20), (2.21), (2.22) and (2.23) are the Kalman filter recursive equations [12]. A transition from a posteriori to a priori is called time update, whereas transition from a priori to a posteriori is called measurement update. The Fig. 2.2 illustrates high-level Kalman filter loop. The initial a priori estimate \bar{x}_0 and the error covariance

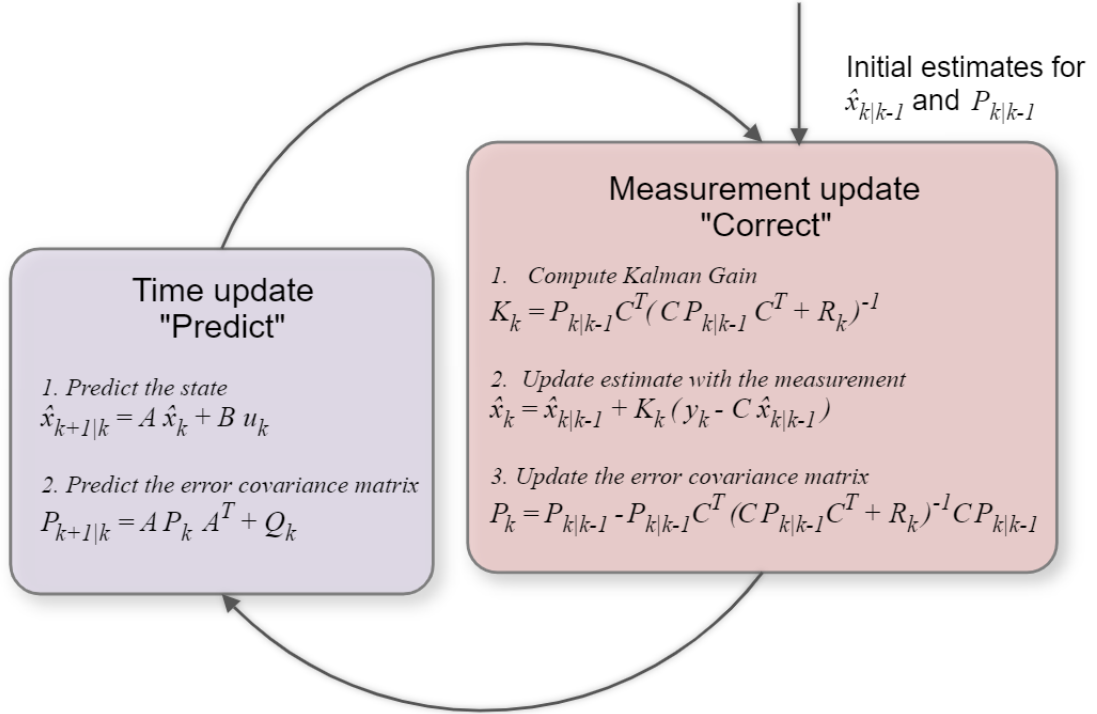


Figure 2.2: Kalman filter loop.

matrix P_0 is assumed to be known or can be calculated using model equations.

The measurement noise covariance matrix R_k and the process noise covariance matrix Q_k are filter tuning parameters. R_k is, usually, estimated using off-line sample measurements or according to the sensor accuracy. The estimation of Q_k , on the other hand, is more challenging [13]. It can, for example, be estimated using system identification techniques or determined during off-line testing.

If R_k and Q_k are treated as constants, the estimation covariance matrix P_k and the Kalman gain K_k will, eventually, reach a steady state. In this case, the steady state value of $P_k = P_\infty$ can be pre-computed off-line. By multiplying the steady state solution of (2.21) and (2.23) with matrix A and A^T , P_∞ can be obtained as a solution of discrete-time algebraic Riccati equation given by

$$P_\infty = A P_\infty A^T - A P_\infty C^T (C P_\infty C^T + R)^{-1} C P_\infty A^T + Q. \quad (2.24)$$

2.5 Extended Kalman Filter

The Kalman filter, presented in the section above, is a conventional approach to solve the state estimation problem for linear unconstrained systems. However, most of the real world applications are nonlinear processes. The Extended Kalman filter (EKF) is a nonlinear version and a straightforward extension of the linear Kalman filter. The fundamental notion behind the EKF is to linearize the nonlinear system around the current state estimate using the partial derivatives of the system. The nonlinear discrete-time unconstrained system under consideration is

$$\begin{aligned}x_{k+1} &= f(x_k, u_k) + w_k, \\y_k &= h(x_k) + v_k.\end{aligned}\tag{2.25}$$

As for linear Kalman filter, process and measurement noise, w_k and v_k , respectively, are considered stationary, zero mean, Gaussian white noise and mutually uncorrelated with x_0 . The recursive Kalman filter equations (2.17b) and (2.22) can be rearranged as follows [13]:

$$\hat{x}_{k|k} = \bar{x}_{k|k-1} + K_k(y_k - h(\bar{x}_{k|k-1}))\tag{2.26}$$

and

$$\bar{x}_{k+1|k} = f(\hat{x}_{k+1|k}, u_k).\tag{2.27}$$

The linearized system of (2.25) is given by

$$\begin{aligned}x_{k+1} &= f(\hat{x}_k, u_k) + A_k(x_k - \hat{x}_k) + w_k, \\y_k &= h(x_{k|k-1}) + C_k(x_k - x_{k|k-1}) + v_k,\end{aligned}\tag{2.28}$$

where A_k is the Jacobian matrix of partial derivatives of f with respect to $\hat{x}_{k-1|k}$ and C_k is the Jacobian matrix of partial derivatives of h with respect to $\hat{x}_{k|k-1}$. The Fig. 2.3 illustrates high-level Extended Kalman filter loop.

It is important to note that the EKF is not an optimal estimator. In addition, for highly nonlinear processes the Extended Kalman filter can be insufficient because the filter design is based on a linear approximation of the nonlinear process model. This may cause poor estimation performance due to the non globally asymptotically stable estimation error dynamics [4].

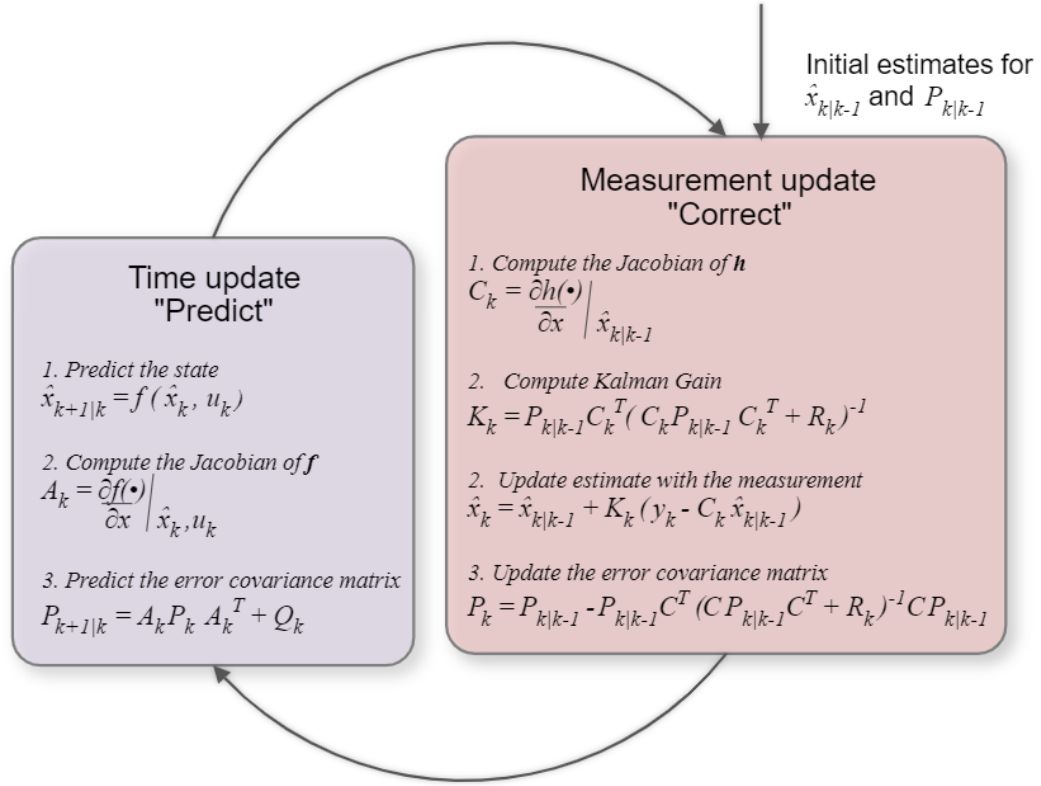


Figure 2.3: Extended Kalman filter loop.

2.6 Full Information Estimation

One of the substantial disadvantages of the Kalman filter and Extended Kalman filter is the inability to directly address or incorporate constraints in the optimization process. Full information estimation, on the other hand, is an optimization based approach that considers the constraints as part of the optimization problem.

The full information estimation objective function is formulated as a least-squares estimation problem and is given by [14]

$$\min_{x_0, \{w_k\}_{k=0}^{T-1}} \|x_0 - \bar{x}_0\|_{P_0^{-1}}^2 + \sum_{k=0}^{T-1} \|w_k\|_{Q_k^{-1}}^2 + \sum_{k=0}^T \|v_k\|_{R_k^{-1}}^2 \quad (2.29)$$

Subject to:

$$x_0 - \bar{x}_0 \in \mathbb{X}, \quad w_k \in \mathbb{W}, \quad v_k \in \mathbb{V}.$$

where \bar{x}_0 is the a priori estimate of the initial values of the state variables, w_k and v_k is the sequences of process and measurement noise in time interval $k = [0, \dots, T-1]$ and $k = [0, \dots, T]$ respectively. P_0^{-1} , Q_k^{-1} and R_k^{-1} are the confidence matrices, where P_0 , Q_k

and R_k are the covariance matrices of x_0 , w_k and v_k respectively. All these matrices are assumed to be symmetric positive definite, and thus invertible. \mathbb{X} , \mathbb{W} and \mathbb{V} are the subsets of real numbers, i.e., the constraints of the system.

Given a discrete-time dynamic system

$$\begin{aligned}x_{k+1} &= f(x_k, u_k) + w_k, \\y_k &= h(x_k) + v_k,\end{aligned}\tag{2.30}$$

where $f(\cdot)$ and $h(\cdot)$ denote linear or nonlinear functions, the optimization problem formulation, (2.29), can be rewritten as follows:

$$\begin{aligned}\min_{\mathbf{x}} \quad & \|x_0 - \bar{x}_0\|_{P_0^{-1}}^2 + \sum_{k=0}^{T-1} \|x_{k+1} - f(x_k, u_k)\|_{Q_k^{-1}}^2 + \sum_{k=0}^T \|y_k - h(x_k)\|_{R_k^{-1}}^2 \\ \text{Subject to:} & \\ & x_0 - \bar{x}_0 \in \mathbb{X}, \\ & x_{k+1} - f(x_k, u_k) \in \mathbb{W}, \\ & y_k - h(x_k) \in \mathbb{V}.\end{aligned}\tag{2.31}$$

The equation (2.31) is solved by minimizing a weighted sum of squared errors of initial condition, system model dynamics and measurements in the time interval $[0, T]$. The solution of the objective function (2.31) yields the sequence of the optimal state estimates $\hat{x} = [\hat{x}_0^T, \dots, \hat{x}_T^T]^T$. In [14] and [15], it is proven that the full information estimation has the best theoretical properties with regard to stability and optimality.

However, the full information estimation is not suited for online implementation due to increasing computational load with each time step. As the size of the optimization problem increases, it will eventually reach a point where it becomes infeasible to solve. Moreover, the measurements far in the past may no longer be relevant or representative of the current state of the system [5]. The practical estimator design technique that preserves the properties of the full information estimation while maintaining a feasible computational burden is called Moving Horizon Estimation.

2.7 Moving Horizon Estimation

In comparison to the full information estimation, MHE considers only a finite sequence of most recent measurements and control inputs. The principle is to continuously update the estimates of the states by solving an optimization problem over a fixed-size moving window. This is illustrated in Fig. 2.4. Considering a finite horizon of the length N , the

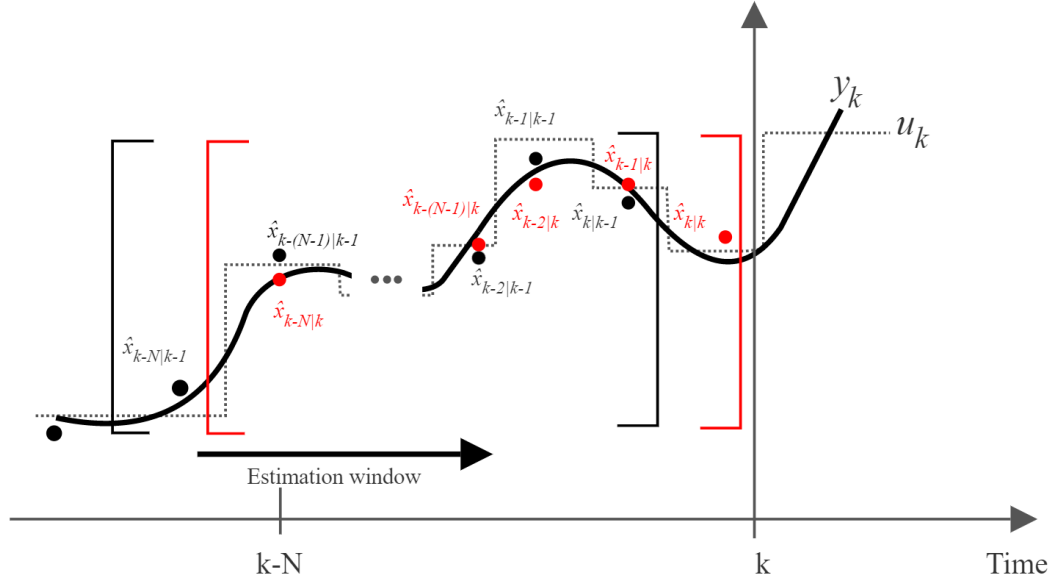


Figure 2.4: Moving Horizon Estimation.

full information estimation objective function, (2.29), can be reformulated as follows:

$$\begin{aligned} \min_{x_0, \{w_i\}_{i=0}^{k-1}} & \sum_{i=k-N}^{k-1} \|w_i\|_{Q^{-1}}^2 + \sum_{i=k-N}^k \|v_i\|_{R^{-1}}^2 + \\ & + \sum_{i=0}^{k-N-1} \|w_i\|_{Q^{-1}}^2 + \sum_{i=0}^{k-N} \|v_i\|_{R^{-1}}^2 + \|x_0 - \bar{x}_0\|_{P_0^{-1}}^2, \end{aligned} \quad (2.32)$$

where (2.29) is divided into two intervals $[k-N, k-1]$ and $[0, k-N-1]$. The first two sums in (2.32) are only dependent on the information within the estimation window, therefore, it can be rewritten as [16]

$$\min_{x_{k-N}, \{w_i\}_{i=k-N}^{k-1}} \sum_{i=k-N}^{k-1} \|w_i\|_{Q^{-1}}^2 + \sum_{i=k-N}^k \|v_i\|_{R^{-1}}^2 + \Gamma_{k-N}(x_{k-N}), \quad (2.33)$$

where

$$\Gamma_{k-N}(x_{k-N}) = \min_{x_0, \{w_i\}_{i=0}^{k-N-1}} \sum_{i=0}^{k-N-1} \|w_i\|_{Q^{-1}}^2 + \sum_{i=0}^{k-N} \|v_i\|_{R^{-1}}^2 + \|x_0 - \bar{x}_0\|_{P_0^{-1}}^2. \quad (2.34)$$

The minimization of the objective function (2.33) yields the sequence of the optimal state estimates $\hat{x} = [\hat{x}_{k-N}^T, \dots, \hat{x}_k^T]^T$. The term $\Gamma_{k-N}(x_{k-N})$ in (2.33), is referred to as the arrival cost which summarizes past information outside the estimation window. To bound the MHE optimization problem as k grows, the arrival cost function in (2.34) has to be rewritten as an algebraic expression. For unconstrained linear systems with the horizon length $N = 1$, MHE optimization problem simplifies to the standard Kalman filter, thus the arrival cost can be expressed explicitly. For nonlinear or constrained systems, the arrival cost has to be approximated [16].

2.7.1 Arrival Cost Update

Moving horizon estimation reduces the computational burden of solving the full information estimation problem by considering a finite horizon of the measurements, however, to determine the arrival cost is a challenging task. The choice of the arrival cost influences the performance and stability of the MHE, thus it is an important design aspect when constructing the estimator.

One of the simplest choices is zero prior weighting, i.e., setting $\Gamma_i(x_i) = 0$. This means that the states are estimated using information within the estimation window and all prior data is completely discarded. The drawbacks of the zero prior weighting is that the system has to be observable rather than detectable, for solution to the MHE problem to exist and it may require a large horizon to achieve an acceptable performance [14].

For constrained linear systems, a common approach is to use an approximate arrival cost based on the arrival cost for the unconstrained problem [3]. The arrival cost in (2.34) for time $k > N$ is approximated by the following quadratic approximation:

$$\tilde{\Gamma}_{k-N}(x_{k-N}) = \|x_{k-N} - \bar{x}_{k-N}\|_{P_{k-N}^{-1}}^2, \quad (2.35)$$

where \bar{x}_{k-N} is the state prediction at time $k - N$ and P_{k-N}^{-1} is the prior confidence matrix at time $k - N$. It is also assumed that P_{k-N} and \bar{x}_{k-N} precisely describe the approximate arrival cost, meaning that for time $k \leq N$, P_k and \bar{x}_k are updated recursively subject to initial condition P_0 and \bar{x}_0 , where \bar{x}_k is the estimate of the states at time k given measurements up until $k = N - 1$. (2.35) can be solved using Kalman filter and its

covariance update equation given by (2.21) and (2.23):

$$P_k = AP_{k-1}A^T - AP_{k-1}C^T(CP_{k-1}C^T + R)^{-1}CP_{k-1}A^T + Q \quad (2.36)$$

For constrained nonlinear systems, [16] estimates the arrival cost by approximating a constrained nonlinear system as an unconstrained linear time-variant system. Assuming that the model functions $f(\cdot)$ and $h(\cdot)$ are sufficiently smooth, the nonlinear system can be approximated using a first-order Taylor series approximation around the estimated trajectory, thus (2.35) can be solved using Extended Kalman filter covariance update. Complex constrained nonlinear systems, may require more advanced techniques to approximate the arrival cost, e.g., unscented Kalman filter or particle filter. A list of different types of approximations is presented in [17].

Combining (2.30) and (2.35), (2.33) can be rewritten as [18] [14]:

$$\begin{aligned} \min_{x_{k-N}, \dots, x_k} \quad & \frac{1}{2} \underbrace{\sum_{i=k-N}^{k-1} \|x_{i+1} - f(x_i, u_i)\|_{Q^{-1}}^2}_{\text{Dynamical system model error}} + \frac{1}{2} \underbrace{\sum_{i=k-N}^k \|y_i - h(x_i)\|_{R^{-1}}^2}_{\text{Measurement error}} + \\ & + \frac{1}{2} \underbrace{\|x_{k-N} - \bar{x}_{k-N}\|_{P_{k-N}^{-1}}^2}_{\text{Arrival cost}}. \end{aligned} \quad (2.37)$$

The MHE objective function is solved by minimizing the influence of the measurement error, the non-measured process disturbance and the arrival cost over the estimation horizon.

When sliding from one estimation window to the next, the state prediction \bar{x}_{k-N} in the arrival cost has to be updated. This can be done by a filtering update or a smoothing update. The filtering update uses the state prediction of $\bar{x}_{k-N|k-N}$, that is the solution of \bar{x}_{k-N} obtained at time $k-N$. This method requires storage of the last $k-N$ MHE estimates. The smoothing update, on the other hand, is more convenient and uses the state prediction of $\bar{x}_{k-N|k}$. In practice, it means that the state prediction to the sample $x_{k-N|k}$ is the optimal estimate to the sample $k-N$ obtained at time $k-1$ [14].

2.7.2 MHE Constraints

Since MHE is derived from the full information estimation presented in Section 2.6, MHE has the ability to directly address or incorporate constraints in the optimization process. Constraints can be modeled as:

- Inequality constraints:

$$h(x) \leq 0 \quad (2.38)$$

where $h(\cdot)$ is a linear or nonlinear function.

- Equality constraints:

$$g(x) = 0 \quad (2.39)$$

where $g(\cdot)$ is a linear or nonlinear function.

- Box constraints, i.e., lower and upper bounds:

$$x_{lb} \leq x \leq x_{ub} \quad (2.40)$$

Equality and inequality constraints define the relation between the optimization variables, while box constraints define the range of optimization variables, i.e., the region of search for the optimization problem.

2.7.3 Estimation of System Parameters and Disturbances

In state estimation, it is often desirable to estimate unknown system parameters or external disturbances. The additional information on the system can be used to improve estimation accuracy, enhance robustness by accounting for uncertainties, enable adaptability to changing system dynamics, and provide valuable insights into the underlying system behavior. This can be achieved by modeling the parameters or disturbances as state variables, thus by augmenting the original system model. The system disturbances/parameters are estimated from measured process variables and are assumed to remain constant if no explicit model for the variations is available. Therefore, the

parameter/disturbance p can be represented as a random walk process²:

$$\dot{p}(t) = 0 + w_p(t) \quad (2.41)$$

for continuous-time, and

$$p_{k+1} = p_k + w_{p_k} \quad (2.42)$$

for discrete-time, where $w_p(t)$ and w_{p_k} are additional process noise. Considering a nonlinear discrete-time state transition equation (2.5) and the augmented state vector defined as $\tilde{x}_k = [x_k^T, p_k^T]^T$, $\tilde{x}_k \in \mathbb{R}^{n_x+n_p}$, the augmented state transition equation has the following form:

$$\underbrace{\begin{bmatrix} x_{k+1} \\ p_{k+1} \end{bmatrix}}_{\tilde{x}_{k+1}} = \underbrace{\begin{bmatrix} f_x(x_k, p_k, u_k) \\ p_k \end{bmatrix}}_{\tilde{f}(x_k, u_k)} + \underbrace{\begin{bmatrix} w_{x_k} \\ w_{p_k} \end{bmatrix}}_{\tilde{w}_k}. \quad (2.43)$$

The only restriction for the augmented system is detectability [14]. Detectability of the augmented system model can be checked using PHB test presented in Section 2.2

2.7.4 Tuning Parameters

The performance of the MHE can be fine-tuned by adjusting the following parameters:

- **The estimation horizon length N .**

In general, the larger N , the more accurate estimation can be expected, however, this will increase the computational burden.

- **The covariance matrix Q .**

The covariance matrix Q is often a diagonal matrix where each diagonal element represents the variance of the corresponding dynamic model error. If correlations among the variables are known, they can be included as nondiagonal elements.

- **The covariance matrix R .**

The covariance matrix R is a diagonal matrix where each diagonal element represents the variance of the corresponding measurement error. R can be considered as a tuning factor, but the best practice is to set it to the actual variance of a

²A random walk is a mathematical model used to describe a process where a variable takes a series of steps that are determined by sequence of random values, i.e., future behaviour is independent of past history.

measurement series of the real plant. If correlations among the measurement errors are known, they can be included as nondiagonal elements.

- **Initial arrival cost covariance matrix P_0 .**

The Initial arrival cost covariance matrix, P_0 , has to be chosen as close to the covariance of the state distribution as possible.

The covariance matrices, Q , R and P , can also be interpreted as prior knowledge of the system and adjusted accordingly, or they can be used to account for errors due to approximations.

Chapter 3

Optimization

The concept of optimization is an important tool in the fields of science, engineering, finance and economics. Optimization plays a crucial role in finding optimal solutions to complex problems such as minimize costs, optimize system performance or maximize profits. MHE, presented in Section 2.7, is an iterative approach that relies on solving an optimization problem to determine the optimal state estimates.

In this chapter, Section 3.1 provides theoretical background on optimization problem formulation, Section 3.2 gives a fundamental insight to the principles behind the optimization techniques and an overview of most widely used optimization algorithms. Section 3.3 summarizes in-built optimization functions available in Matlab.

3.1 Optimization Problem

A typical engineering problem is often described with some mathematical equations subject to a single or multiple performance criterion, e.g., a cost function. The goal of optimization process is to find the best solution, among a range of possible alternatives that satisfy certain constraints and yields the best (maximum or minimum) value of performance criterion. Optimization is, simply, a collection of mathematical principles and methods utilized to solve quantitative problems. There are different ways to formulate an optimization problem, but most of them share similar structures. Every optimization problem contains three fundamental elements [19]:

- An objective function to be optimized (necessary).
- Equality constraints (optional).
- Inequality constraints (optional).

The set of variables that satisfies the constraints is called feasible solutions, whereas the set of feasible solutions that provides an optimal value to the objective function is called an optimal solution to the optimization problem. Fig 3.1 illustrates a minimization problem¹, where $x = [x_1, x_2]$ are the optimization variables, orange lines indicates the constraints, f is the objective function and $[x_{1,optimal}, x_{2,optimal}]$ is the set of optimal solutions that minimizes the function f . A typical mathematical formulation of the

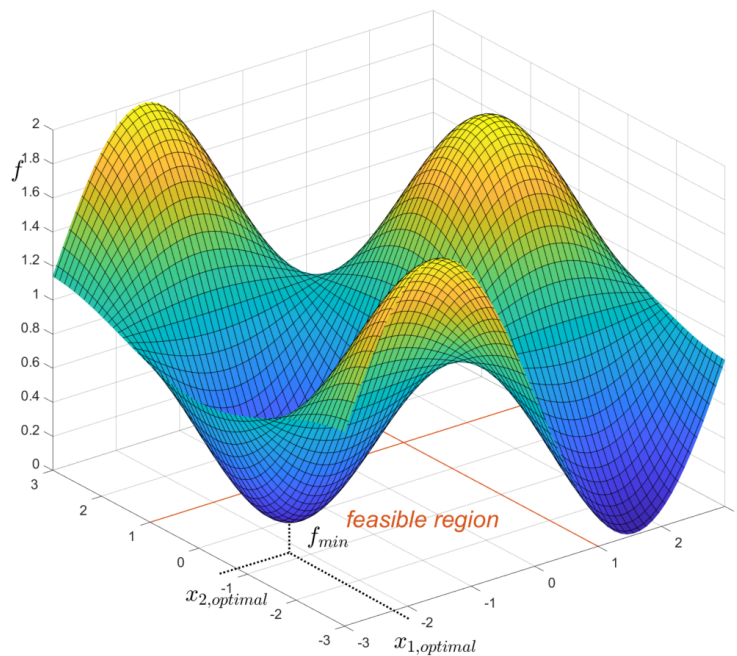


Figure 3.1: Optimization problem.

optimization problem is the following: *For a given mathematical model of the process, find the values of the vector \mathbf{x} that minimizes the objective function $f(\mathbf{x})$:*

$$\begin{aligned}
 & \min_{\mathbf{x}} f(\mathbf{x}) \\
 & \text{Subject to:} \\
 & \mathbf{g}(\mathbf{x}) = 0, \quad \text{Equality constraints.} \\
 & \mathbf{h}(\mathbf{x}) \leq 0, \quad \text{Inequality constraints.}
 \end{aligned}
 \tag{3.1}$$

¹For the rest of this paper, optimization problem will be considered as minimization problem.

Optimization problems are categorized into convex and non-convex problems. Convex problems are those where both objective and constraints functions are convex, i.e., they satisfy the inequality

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y) \quad (3.2)$$

for all $x, y \in \mathbb{R}$ and $\alpha, \beta \in \mathbb{R}$, where $\alpha + \beta = 1$. Convex problems have unique feasible region with a unique set of globally optimal solutions. Non-convex optimization problems, on the other hand, are those where the objective or constraint functions are non-convex. Non-convex problems may have multiple feasible regions and locally optimal solutions within each region. In general, non-convex optimization is a hard task to solve, since there may be potentially many local minima, saddle points and very flat regions [20]. Examples of convex and non-convex function are shown in Fig. 3.2.

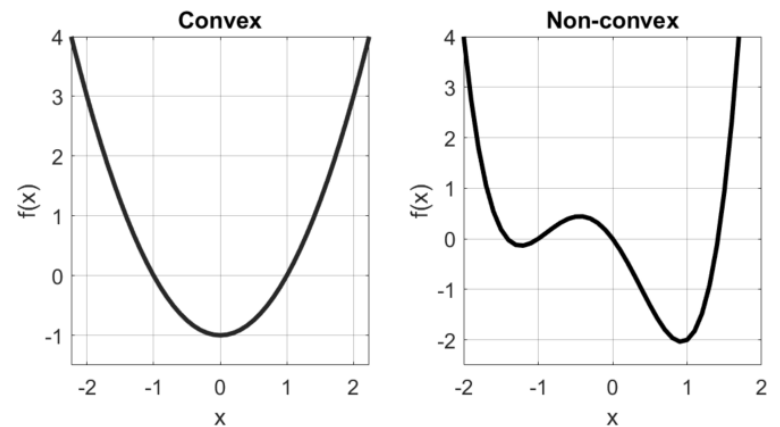


Figure 3.2: Example of convex and non-convex functions.

3.2 Optimization Classes and Methods

Due to the fact that optimization is a fundamental tool used in many fields, where each field has its own unique set of optimization problems, optimization problems are divided into different classes. Each class reflects the diversity of applications and mathematical properties of optimization problems, and requires different techniques or analytical tools to solve the problems effectively. Examples of widely used optimization problem classes in various fields include linear programming, convex programming, nonlinear programming, quadratic optimization, stochastic programming, network optimization, combinatorial optimization, geometric programming and many others. This section covers a short

introduction to the most popular optimization classes and corresponding methods used to solve the optimization problem.

3.2.1 Least-Squares Optimization

A basic least-squares optimization problem is an unconstrained problem with an objective function which is a sum of squares of terms $a_j^T x - b_j$:

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 = \sum_{j=1}^k (a_j^T \mathbf{x} - b_j)^2, \quad (3.3)$$

where $x \in \mathbb{R}^n$ is the vector of optimization variables and $A \in \mathbb{R}^{k \times n}$, where k is the number of rows in A . The solution to least-squares problem can be obtained by solving a set of linear equations

$$(A^T A)\mathbf{x} = A^T \mathbf{b}. \quad (3.4)$$

Thus, least-squares problem has an analytical solution $\mathbf{x} = (A^T A)^{-1} A^T \mathbf{b}$. There are many reliable and efficient methods that can solve large least-squares problems with hundreds of optimization variables in a few seconds. The computation time is approximately proportional to $n^2 \cdot k$ [20].

Regression analysis, data fitting, optimal control, and numerous parameter estimation techniques are built upon the foundation of the least-squares problem. Weighted least-squares is a frequently used variation of basic least-squares problem:

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_W^2 = \sum_{j=1}^k w_j (a_j^T \mathbf{x} - b_j)^2, \quad (3.5)$$

where w_1, \dots, w_k are positive weights. The weighted least-squares formulation provides more flexibility to express different levels of importance for the magnitudes of the terms $a_j^T \mathbf{x} - b_j$. In state estimation, the weights are often used to express influence of unequal variances that corrupts the error terms.

Furthermore, (3.5) can be improved by introducing a regularization term that adds the ability to penalize the original cost term:

$$\min_{\mathbf{x}} \sum_{j=1}^k w_j (a_j^T \mathbf{x} - b_j)^2 + \sum_{j=1}^k v_j (c_j^T \mathbf{x} - d_j)^2. \quad (3.6)$$

The choice of weights w_j and v_j permits a trade-off between reducing the magnitude of the first term while ensuring that the second term does not become too large, or vice versa [20].

3.2.2 Linear Programming

Linear programming problems involve optimizing a linear objective function whose variables are subject to linear equality and inequality constraints. The standard formulation is given by

$$\begin{aligned} \min_{\mathbf{x}} \quad & c^T \mathbf{x} \\ \text{Subject to:} \quad & \\ & a_i^T x \leq b_i, \quad i = 1, \dots, l \\ & x \geq 0, \end{aligned} \tag{3.7}$$

where $c, a_1, \dots, a_l \in \mathbb{R}^n$ are the vectors, $b_1, \dots, b_l \in \mathbb{R}$ are the scalars and l is the number of constraint equations. The feasible region, defined by (3.7), in geometric terms is a convex polytope. There is no simple analytical expression to solve a linear programming problem, but there are some effective and reliable algorithms such as Dantzig's Simplex or Interior-point algorithms [20]. For linear objective functions with linear constraints an optimal solution occurs at extreme point of the feasible region. However, an optimal solution may not be unique (if there is more than one extreme point). Hence, linear programming problem reduces to evaluating which extreme point corresponds to the lowest value of the objective function. The Simplex algorithm is an iterative algorithm that starts with an initial extreme point and evaluates if the solution is optimal. This is done using an algebraic specification of the problem. If a current extreme point is not an optimal solution, the algorithm moves to an adjacent extreme point along an edge in the direction for which the objective function is minimized at fastest rate. The search and evaluation of extreme points is repeated until optimal solution is found [21]. Fig. 3.3 illustrates an example of a linear programming problem and optimal solution search by Simplex algorithm.

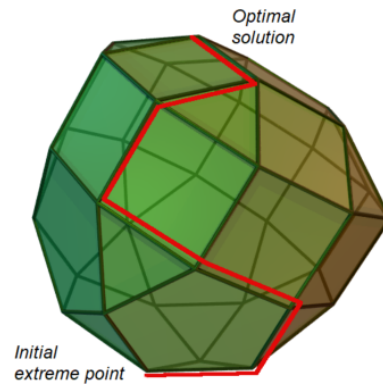


Figure 3.3: Example of a linear programming polytope together with a possible path (red) taken by the simplex method to solve the corresponding LP [1].

3.2.3 Convex Programming

A convex optimization problem has the following form:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f_0(\mathbf{x}) \\ \text{Subject to:} \quad & \\ & f_i(\mathbf{x}) \leq b_i, \quad i = 1, \dots, l, \end{aligned} \tag{3.8}$$

where functions f_0, \dots, f_l are convex, i.e., they satisfy the equation (3.2). The least-squares and linear programming problems, presented above, are special cases of the general convex optimization problem. There are a variety of effective approaches for solving convex optimization problems. Some of them are: gradient methods, Quasi-Newton method and Interior-point. If the optimization problem is convex and unconstrained, the solution may be found using Gradient descent, Steepest descent or Newton methods.

Descent Methods

Descent methods generate a minimizing sequence x_k , $k = 1, \dots$:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + t_k \Delta \mathbf{x}(k), \tag{3.9}$$

where k denotes iteration, $\Delta \mathbf{x} \in \mathbb{R}^n$ is a vector called the step or search direction and $t_k > 0$ (except when \mathbf{x}_k is optimal, $t_k = 0$) is referred to as the step size or step length at iteration k . A search direction must minimize the objective function such that the

following holds:

$$f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k), \quad \text{unless } x_k \text{ is an optimal solution.} \quad (3.10)$$

A search direction $\Delta \mathbf{x}$ is called a descent direction if it satisfies

$$\nabla f(\mathbf{x})^T \Delta \mathbf{x} < 0. \quad (3.11)$$

In geometry setting, (3.11) means that the search directions points towards the direction where the objective function gradient is negative. Hence, given a starting point a general descent algorithm alternates between two steps: determining a descent direction $\Delta \mathbf{x}$, and then choosing a step size t_k . Internal iteration continues until a stopping criterion is satisfied, e.g., $|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \epsilon$.

The gradient descent method determines the step using

$$\Delta x_k = -\nabla f(x_k), \quad (3.12)$$

such that (3.9) has the following form [19]

$$\mathbf{x}_{k+1} = \mathbf{x}_k - t_k \nabla f(x_k). \quad (3.13)$$

The step size t_k is determined using line search methods: exact line search or inexact line search. In exact line search t_k is chosen to minimize f along $\{x_k + t_k \Delta x_k | t \geq 0\}$:

$$t_k = \operatorname{argmin}_{s \leq 0} f(x_k + s \Delta x_k). \quad (3.14)$$

An example of inexact line search method that is commonly used in practise is backtracking line search. The method depends on two constants, namely, α and β , where $0 < \alpha < 0.5$ and $0 < \beta < 1$. The algorithm starts with a unit step and reduces it by the factor β until the stopping criterion

$$f(x_k + t_k \Delta x_k) < f(x_k) + \alpha t_k \nabla f(x_k)^T \Delta x_k \quad (3.15)$$

is satisfied [20]. The parameter α is used to account for prediction error or uncertainty while the parameter β corresponds to the level of approximation used.

Newton Method

Similarly to descent methods, Newton method is also an iterative gradient based algorithm. Assuming that the objective function is twice differentiable, Newton method approximates an objective function using second order Taylor expansion, i.e., a quadratic approximation of $f(\mathbf{x})$ at \mathbf{x}_k :

$$f(\mathbf{x}) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \Delta \mathbf{x}_k + \frac{1}{2} (\Delta \mathbf{x}_k)^T H(\mathbf{x}_k) \Delta \mathbf{x}_k, \quad (3.16)$$

where $H(\mathbf{x}_k)$ is the Hessian matrix² of $f(\mathbf{x})$. By employing the information obtained from the second partial derivatives of $f(\mathbf{x})$, it is possible to take into account the curvature of $f(\mathbf{x})$ at \mathbf{x}_k when searching for directions. The search direction, also called Newton step, can be obtained by differentiating (3.16) with respect to $\Delta \mathbf{x}$, and setting the expression to zero [19]:

$$\begin{aligned} \nabla f(\mathbf{x}) &= \nabla f(\mathbf{x}_k) + H(\mathbf{x}_k) \Delta \mathbf{x}_k = 0 \\ \Rightarrow \Delta \mathbf{x}_k &= -[H(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k). \end{aligned} \quad (3.17)$$

Thus, substituting (3.17) into (3.9) results in

$$\mathbf{x}_{k+1} = \mathbf{x}_k - t_k [H(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k). \quad (3.18)$$

To avoid calculating the inverse of a matrix in (3.18), it is common to solve the following set of linear equations to determine the Newton step:

$$H(\mathbf{x}_k) \Delta \mathbf{x}_k = -\nabla f(\mathbf{x}_k). \quad (3.19)$$

Newton method iterates in the same manner as descent methods described above. Commonly used stopping criterion for Newton method is $\lambda^2/2 \geq \epsilon$, where λ is a quantity called the Newton decrement at x_k and is given by [20]

$$\lambda(\mathbf{x}) = (\nabla f(\mathbf{x}_k)^T [H(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k))^{\frac{1}{2}}. \quad (3.20)$$

Newton method can also be extended to include linear equality constraints. The extended version of Newton method is almost the same, except for two differences: the initial guess must be feasible, i.e., satisfy the constraints, and the Newton step definition is

²The matrix of second partial derivatives with respect to \mathbf{x} .

slightly modified to account for equality constraints. By approximating linear constraints of the form $A\mathbf{x} = \mathbf{b}$ using second-order Taylor expansion, the objective function in (3.16) is minimized subject to $A\Delta\mathbf{x} = 0$. The optimality of a quadratic objective function is determined using Karush-Kuhn-Tucher (KKT) optimality condition. An optimization problem can be approximated using the Langrangian function

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T(A\mathbf{x} - \mathbf{b}), \quad (3.21)$$

where $\boldsymbol{\lambda}$ is the vector of Langrange multipliers. The KKT optimality condition states, that Langrangian function, $L(\mathbf{x}, \boldsymbol{\lambda})$, is minimized at optimal solutions $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$, thus its gradient must be zero at $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$:

$$\begin{aligned} A\mathbf{x}^* &= \mathbf{b}, & \text{primal feasibility condition.} \\ \nabla f(\mathbf{x}^*) + A^T\boldsymbol{\lambda}^* &= 0, & \text{dual feasibility condition.} \end{aligned} \quad (3.22)$$

Hence, the KKT condition for (3.16) with constraints $A\Delta\mathbf{x} = 0$ are

$$\begin{aligned} H(\mathbf{x}_k)\Delta\mathbf{x}_k + \nabla f(\mathbf{x}_k) + A^T\mathbf{w} &= 0, \\ A\Delta\mathbf{x}_k &= 0, \end{aligned} \quad (3.23)$$

where \mathbf{w} is the optimal dual variable for the quadratic problem. The Newton step can be found by solving the linear equations in (3.23). These can also be rewritten in matrix form

$$\begin{bmatrix} H(\mathbf{x}_k) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_k \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} -\nabla f(\mathbf{x}_k) \\ 0 \end{bmatrix}. \quad (3.24)$$

The matrix:

$$\begin{bmatrix} H(\mathbf{x}_k) & A^T \\ A & 0 \end{bmatrix} \quad (3.25)$$

is also called KKT matrix. The Newton step is uniquely determined only at points where the KKT matrix is nonsingular [20].

Quasi-Newton Method

Quasi-Newton method is an alternative to Newton method that replaces $H(\mathbf{x}_k)$ in equation (3.18) with a positive-definite approximation of \tilde{H}_k to reduce computational

time. \tilde{H}_k is initialized as a positive-definite symmetric matrix, and continuously updated after each line search using the changes in \mathbf{x}_k and $\nabla f(\mathbf{x})$:

$$\mathbf{d}_k = \mathbf{x}_{k+1} - \mathbf{x}_k \quad (3.26)$$

and

$$\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k). \quad (3.27)$$

Among the most efficient and frequently employed techniques to update the Hessian is Broyden–Fletcher–Goldfarb–Shanno (BFGS) update that is given by [19]

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{d}_k^T \mathbf{y}_k} - \frac{\tilde{H}_k \mathbf{d}_k (\tilde{H}_k \mathbf{d}_k)^T}{\mathbf{d}_k^T \tilde{H}_k \mathbf{d}_k}. \quad (3.28)$$

3.2.4 Nonlinear Programming

Nonlinear programming refers to an optimization problem, where the objective or constraint functions are nonlinear, and they are assumed not to be necessarily convex. Therefore, even a "simple" nonlinear problem may be extremely challenging to solve. Methods for the general nonlinear problems, are often, based on different approaches compromising between finding the true optimal solution and optimization efficiency. Using local optimization approach the solution is, possibly, only locally optimal, meaning that the objective function is minimized among feasible points that are near the minima, but not necessarily near the global minima. The advantages of local optimization methods are efficient computational time and ability to handle large-scale problems. The only requirement is that both objective and constraints functions are smooth, i.e., differentiable at every point in its domain. However, there are several disadvantages. Other than possibly not globally optimal solutions, local optimization methods require initial guess for the optimization variables. Thus, the choice of initial guess may considerably affect the value of the objective function as well as local optimal solution. In addition, these type of methods may be sensitive to algorithm parameters. Local optimization methods that can handle constrained nonlinear problems are Successive Quadratic Programming (SQP) and Interior-point. The other approach in solving nonlinear optimization problems is global optimization. These type of methods focus on finding a global optimal solution but at the cost of efficiency. Global optimization methods are, usually, used for problems with few optimization variables and where computational time is not an issue [20].

SQP Method

SQP is gradient based method that approximates a nonlinear programming problem to a sequence of quadratic programming sub-problems. In addition, nonlinear constraints are linearized around selected point and inequality constraints transformed to equality constraints using slack³ variables. Thus, the quadratic problems have a quadratic objective function with linear constraints. The quadratic objective functions are approximated using the Langrangian function. Given a general, simplified version of nonlinear optimization problem with equality constraints

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{Subject to:} \quad & \\ & g_i(\mathbf{x}) = b_i, \quad i = 1, \dots, l, \end{aligned} \tag{3.29}$$

the Langrangian function is given by

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^l \lambda_i (g_i(\mathbf{x}) - b_i). \tag{3.30}$$

SQP method uses the KKT optimality conditions, presented in previous section, thus the KKT conditions for the optimization problem in (3.29) is

$$\nabla_{\mathbf{x}} L = \nabla f(\mathbf{x}) + \sum_{i=1}^l \lambda_i \nabla \mathbf{g}_i(\mathbf{x}) = 0 \tag{3.31}$$

and

$$g_i(\mathbf{x}) = b_i. \tag{3.32}$$

Given some initial guess the equations (3.31) and (3.32) can be solved using Newton method, where both equations are replaced by first-order Taylor approximations around initial guess. That is

$$\nabla_{\mathbf{x}} L + \nabla_{\mathbf{x}}^2 L \Delta \mathbf{x} + \nabla \mathbf{g}^T \Delta \boldsymbol{\lambda} = 0 \tag{3.33}$$

and

$$\mathbf{g} + \nabla \mathbf{g} \Delta \mathbf{x} = 0, \tag{3.34}$$

³Slack variable is non-negative variable that represents the amount by which a constraint can be violated. E.g., Introducing slack variable $s \geq 0$ for inequality constraint $Ax \leq b$ results in equality constraint $Ax + s = b$.

where $\nabla_{\mathbf{x}}^2 L$ is the Hessian of the Lagrangian and ∇g is the Jacobian matrix of \mathbf{g} , both evaluated at initial guess. Considering quadratic optimization problem in (3.29) instead of general nonlinear problem, Taylor approximation in (3.33) becomes [19]

$$\nabla_{\mathbf{x}} L^T \Delta \mathbf{x} + \frac{1}{2} \Delta \mathbf{x} \nabla_{\mathbf{x}}^2 L \Delta \mathbf{x} + \nabla \mathbf{g}^T \Delta \boldsymbol{\lambda} = 0. \quad (3.35)$$

Hence, the Newton step $(\Delta \mathbf{x}, \Delta \boldsymbol{\lambda})$ can be found by solving equations (3.35) and (3.34). The algorithm terminates when the KKT optimality conditions are satisfied.

Interior-Point Method

Considering a nonlinear optimization problem with linear equality and nonlinear inequality constraints

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{Subject to:} \quad & \\ & g_i(\mathbf{x}) \geq 0, \quad i = 1, \dots, l, \\ & A(\mathbf{x}) = \mathbf{b}, \end{aligned} \quad (3.36)$$

where f and g_1, \dots, g_l are convex and twice differentiable functions. The Interior-point methods solve an optimization problem in (3.36) by applying Newton method to a sequence of linear equality constrained problems. Meaning that after the optimization problem is reformulated with a sequence of linear equality constraints, Interior-point methods employs Newton method to reduce the equality constrained problem to a sequence of quadratic constrained problems as presented in previous section.

One type of the methods used to transform inequality constraints into equality constraints is called barrier methods. The main idea is to modify an objective function with an additional term such that the value of objective function increases if constraints are violated. An optimization problem with logarithmic barrier function is formulated as

$$\min_{\mathbf{x}} \quad f(\mathbf{x}) - r \sum_{i=1}^l \ln(g_i(\mathbf{x})). \quad (3.37)$$

Barrier methods convert a constrained optimization problem into a series of unconstrained problems. The optimal solutions to these unconstrained problems converge to the constrained solution as barrier parameter r is approaching zero. If parameter r is positive,

the function, (3.37), is defined only inside the feasible region, where $\mathbf{g}(\mathbf{x})$ is positive. As \mathbf{x} approaches the boundary of the constraints, $\mathbf{g}(\mathbf{x})$ approaches zero, and the term $\ln(\mathbf{g}(\mathbf{x}))$ approaches infinity. Barrier parameter is increased iteratively such that the optimal solution moves closer to the feasible region, until the solution converges to the optimal solution of the original constrained optimization problem [19].

3.3 Matlab In-built Functions

Matlab has a variety of in-built functions that are capable to solve different optimization problems, such as linear programming, mixed-integer linear programming, quadratic programming, second-order cone programming, nonlinear programming, constrained least squares, nonlinear least squares and nonlinear equations [22]. Some of these are⁴:

- ***linprog***. Solves linear constrained programming problems of the form (3.7). Available algorithms are Dual-Simplex, Interior-point-legacy(a variant of Mehrotra's predictor-corrector algorithm) and Interior-point.
- ***quadprog***. Solves quadratic linear constrained and unconstrained programming problems. Available algorithms are Interior-point-convex, Trust-region-reflective (based on interior-reflective Newton method) and Active-set.
- ***lsqlin***. Solves standard linear least-squares constrained programming problems of the form (3.3). Available algorithms are Interior-point (based on Interior-point-convex), Trust-region-reflective (based on interior-reflective Newton method) and Active-set.
- ***lsqnonlin***. Solves standard nonlinear least-squares constrained programming problems. Available algorithms are Interior-point, Trust-region-reflective (based on interior-reflective Newton method) and Levenberg-marquardt.
- ***fminunc***. Solves nonlinear unconstrained programming problems. Available algorithms are Quasi-Newton and Trust-region.
- ***fmincon***. Solves nonlinear constrained multivariable programming problems. Available algorithms are Interior-point, SQP, Active-Set and Trust-Region-Reflective.

⁴Information presented in this section is compiled from MathWorks documentation [23].

Trust-Region-Reflective algorithm requires a pre-defined for the objective and constraints functions.

- ***patternsearch***. Solves nonlinear constrained programming problems. Available algorithms are Pattern Search (based on an adaptive mesh that is aligned with the coordinate directions) and Nonuniform Pattern Search.
- ***ga***. Solves nonlinear constrained programming problems. Genetic algorithm.

In general, the choice of optimization method or algorithm depends on the type of the problem, i.e., the type of the objective function, the nature of the constraints and the number of variables [19]. Thus, the effectiveness of different methods varies and may differ in accuracy, computational efficiency and the ability to solve the optimization problem.

Chapter 4

Implementation

Chapter 2 discussed the most commonly used estimation techniques and provided detailed theoretical presentation on MHE. Chapter 3 presented the fundamental background on optimization problem formulation and theoretical principles behind the most widely used optimization algorithms. In this chapter, Section 4.1 and 4.2 disclose implementation and verification of the LMHE and NMHE for the two-tank system in simulation environment, and Section 4.3 gives a brief overview of the Simulink implementation.

4.1 Linear Moving Horizon Estimation

In a LMHE estimation an objective function formulation involves describing the system dynamics using linear equations. Consequently, it allows for a linear state-space model representation. Considering a linear discrete-time dynamic system of the form

$$\begin{aligned}x_{k+1} &= Ax_k + Bu_k + w_k, \\y_k &= Cx_k + v_k,\end{aligned}\tag{4.1}$$

the general constrained MHE formulation (2.37), can be rewritten as follows:

$$\begin{aligned}
\min_{x_{k-N}, \dots, x_k} \quad & \underbrace{\frac{1}{2} \sum_{i=k-N}^{k-1} \|x_{i+1} - Ax_i - Bu_i\|_{Q^{-1}}^2}_{\text{Dynamical system model error}} + \underbrace{\frac{1}{2} \sum_{i=k-N}^k \|y_i - Cx_i\|_{R^{-1}}^2}_{\text{Measurement error}} + \\
& + \underbrace{\frac{1}{2} \|x_{k-N} - \bar{x}_{k-N}\|_{P_{k-N}^{-1}}^2}_{\text{Arrival cost}} \quad (4.2)
\end{aligned}$$

Subject to:

$$x_i, \bar{x}_{k-N} \in \mathbb{X}, \quad x_{i+1} - Ax_i - Bu_i \in \mathbb{W}, \quad y_i - Cx_i \in \mathbb{V},$$

where \bar{x}_{k-N} is the state prediction at time $k - N$ and generic $f(\cdot)$ and $h(\cdot)$ functions are replaced with the corresponding linear functions.

The state variables for the two-tank system, presented in Appendix A, namely, the water level in tank 1, the water level in tank 2 and, potentially, the water flow from the pump are physically constrained. The water level is bounded by the tank size while the water flow rate is limited by the pump capacity. Thus, the two-tank MHE problem is subject to the box constraints, i.e., the lower and upper bounds of the state variables. The constraints in (4.2) can, therefore, be reformulated as

$$\mathbb{X} = \prod_{i=1}^n \prod_{j=1}^N [x_{i,lb}(j), x_{i,ub}(j)], \quad (4.3)$$

where N is the horizon length and n is the number of state variables. The products should be interpreted in the Cartesian sense.

In the literature it is quite common to formulate the LMHE problem in (4.2) as a quadratic programming problem, i.e., a second order Taylor approximation of the following form

$$\min_{\mathbf{z}} \frac{1}{2} \mathbf{z}^T H \mathbf{z} + \mathbf{z}^T \nabla f, \quad (4.4)$$

where $\mathbf{z} \in \mathbb{R}^{Nn}$ is the vector of the state variables, H is the Hessian and ∇f is the gradient of the objective function. After some algebra, the equation (4.2) can be transformed into (4.4). The Hessian and the gradient of the objective function can be obtained by, firstly, considering the horizon length, $N = 1$ and expanding the squares. This gives the

following expression:

The first term:

$$x_2^T Q^{-1} x_2 - x_2^T Q^{-1} A x_1 - x_2^T Q^{-1} B u_1 - x_1^T A^T Q^{-1} x_2 - u_1^T B^T Q^{-1} x_2 + x_1^T A^T Q^{-1} A x_1 + x_1^T A^T Q^{-1} B u_1 + u_1^T B^T Q^{-1} A x_1 + u_1^T B^T Q^{-1} B u_1 +$$

The second term:

$$+ y_1^T R^{-1} y_1 - y_1^T R^{-1} C x_1 - x_1^T C^T R^{-1} y_1 + x_1^T C^T R^{-1} x_1 +$$

The third term:

$$+ \bar{x}_1^T P_1^{-1} \bar{x}_1 - \bar{x}_1^T P_1^{-1} x_1 - x_1^T P_1^{-1} \bar{x}_1 + x_1^T P_1^{-1} x_1 \quad (4.5)$$

Simplifying the expression further and collecting all quadratic parts of x_i in a matrix results in

$$H^{(N=1)} = \begin{bmatrix} A^T Q^{-1} A + C^T R C + P_1^{-1} & -A^T Q^{-1} \\ -Q^{-1} A & Q^{-1} \end{bmatrix}. \quad (4.6)$$

Considering the horizon length, $N = 2$ and repeating the same steps, the Hessian becomes

$$H^{(N=2)} = \begin{bmatrix} A^T Q^{-1} A + C^T R C + P_1^{-1} & -A^T Q^{-1} & 0 \\ -Q^{-1} A & Q^{-1} + A^T Q^{-1} A + C^T R C & -A^T Q^{-1} \\ 0 & -Q^{-1} A & Q^{-1} \end{bmatrix}. \quad (4.7)$$

In (4.7) the additional step have only influenced the element (2, 2). Thus the Hessian with the horizon N is given by

$$H^{(N)} = \begin{bmatrix} A^T Q^{-1} A + C^T R C + P_{k-N}^{-1} & -A^T Q^{-1} & 0 & \dots & 0 \\ -Q^{-1} A & Q^{-1} + A^T Q^{-1} A + C^T R C & -A^T Q^{-1} & \dots & 0 \\ 0 & -Q^{-1} A & Q^{-1} + A^T Q^{-1} A + C^T R C & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & -A^T Q^{-1} \\ 0 & 0 & 0 & -Q^{-1} A & -Q^{-1} \end{bmatrix} \quad (4.8)$$

The gradient can be obtained in the same manner, but collecting all linear terms of x_i in (4.5)

$$\nabla f^{(i)} = \begin{bmatrix} A^T Q^{-1} B u_1 & 0 & \dots & 0 \\ -Q^{-1} B u_1 & A^T Q^{-1} B u_2 & \ddots & 0 \\ 0 & -Q^{-1} B u_2 & \ddots & 0 \\ \vdots & \vdots & \ddots & A^T Q^{-1} B u_i \\ 0 & 0 & 0 & -Q^{-1} B u_i \end{bmatrix} + \begin{bmatrix} -C^T R^{-1} y_1 - P_{k-N}^{-1} \bar{x}_{k-N} \\ -C^T R^{-1} y_2 \\ \vdots \\ -C^T R^{-1} y_i \\ 0 \end{bmatrix}. \quad (4.9)$$

The constant term $f(u, y, \tilde{x}_{k-N})$ is neglected due to the fact that it does not influence the optimal solution.

4.1.1 Linearization of Two-tank System

The two-tank plant is second-order nonlinear system, detailed description of the system is provided in Appendix A. LMHE implementation requires a linear dynamical model, thus the nonlinear equations of the two-tank system has to be linearized. A linear approximation of the two-tank system is obtained by using first-order Taylor series expansion around a state trajectory associated to the control inputs of the system and the specific initial states. The two-tank system is described by the following state equations

$$\begin{cases} \dot{h}_1(t) &= g_1(h_1(t), u_{PA001}(t), u_{LV001}(t)) \\ \dot{h}_2(t) &= g_2(h_1(t), h_2(t), u_{LV001}(t), u_{LV002}(t)) \end{cases}, \quad (4.10)$$

where g_1 and g_2 denotes the nonlinear functions of tank 1 and tank 2, respectively. Assuming that the the following holds

$$\begin{cases} \tilde{h}_1(t) &= g_1(\tilde{h}_1(t), \tilde{u}_{PA001}(t), \tilde{u}_{LV001}(t)) \\ \tilde{h}_2(t) &= g_2(\tilde{h}_1(t), \tilde{h}_2(t), \tilde{u}_{LV001}(t), \tilde{u}_{LV002}(t)) \\ \tilde{h}_1(t_0) &= h_{1,0}, \tilde{h}_2(t_0) = h_{2,0}, \tilde{u}_{PA001}(t_0) = u_{PA001,0}, t \geq t_0 \end{cases}, \quad (4.11)$$

the Taylor series approximation is

$$\Delta \dot{h}_1(t) = \left. \frac{\partial g_1}{\partial h_1} \right|_O \Delta h_1(t) + \left. \frac{\partial g_1}{\partial u_{PA001}} \right|_O \Delta u_{PA001}(t) + \left. \frac{\partial g_1}{\partial u_{LV001}} \right|_O \Delta u_{LV001}(t) \quad (4.12)$$

for tank 1 and

$$\begin{aligned} \Delta \dot{h}_2(t) &= \left. \frac{\partial g_2}{\partial h_1} \right|_O \Delta h_1(t) + \left. \frac{\partial g_2}{\partial h_2} \right|_O \Delta h_2(t) + \left. \frac{\partial g_2}{\partial u_{LV001}} \right|_O \Delta u_{LV001}(t) \\ &\quad + \left. \frac{\partial g_2}{\partial u_{LV002}} \right|_O \Delta u_{LV002}(t) \end{aligned} \quad (4.13)$$

for tank 2, where O denotes the operating point for the partial derivatives, namely, $\tilde{h}_1, \tilde{h}_2, \tilde{u}_{LV001}, \tilde{u}_{LV002}$ and \tilde{u}_{PA001} . Δ denotes the deviation variable around the operating point.

u_{LV001} , u_{LV002} and u_{PA001} are independent variables of the functions f_1 , f_2 and f_3 . Therefore, the partial derivatives $\frac{\partial g_1}{\partial u_{LV001}}$, $\frac{\partial g_1}{\partial u_{PA001}}$, $\frac{\partial g_2}{\partial u_{LV001}}$ and $\frac{\partial g_2}{\partial u_{LV002}}$ can be found using supply chain rule

$$\frac{\partial g_1}{\partial u_{LV001}} = \frac{\partial g_1}{\partial f_1} \cdot \frac{\partial f_1}{\partial u_{LV001}}, \quad \frac{\partial g_1}{\partial u_{PA001}} = \frac{\partial g_1}{\partial f_3} \cdot \frac{\partial f_3}{\partial u_{PA001}}, \quad (4.14)$$

$$\frac{\partial g_2}{\partial u_{LV001}} = \frac{\partial g_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial u_{LV001}}, \quad \frac{\partial g_2}{\partial u_{LV002}} = \frac{\partial g_2}{\partial f_2} \cdot \frac{\partial f_2}{\partial u_{LV002}}, \quad (4.15)$$

where the partial derivative $\frac{\partial g_i}{\partial u_{xx00i}}$, can be found by considering a small deviation Δ around the operating point

$$\left. \frac{\partial g_i}{\partial u_{xx00i}} \right|_O = \frac{f_i(u_{xx00i,0} + \Delta) - f_i(u_{xx00i,0})}{(u_{xx00i,0} + \Delta) - u_{xx00i,0}}. \quad (4.16)$$

In practice, Δ is set to 0.01.

The corresponding operating points for \tilde{u}_{LV001} and \tilde{u}_{LV002} , such that (4.11) holds, can be determined by setting the differential equations (A.11) and (A.16) to zero:

$$\begin{aligned} \dot{h}_1(t) = 0 &\Rightarrow q_{LV001,0} = f_1(u_{LV001,0}) = \frac{3600 f_3(u_{PA001,0})}{K_{v,LV001} \sqrt{\frac{\rho g (h_1(t) + h_{LV001})}{100000}}} \\ &\Rightarrow u_{LV001,0} = f_1^{-1}(q_{LV001,0}), \end{aligned} \quad (4.17)$$

$$\begin{aligned} \dot{h}_2(t) = 0 &\Rightarrow q_{LV002,0} = f_2(u_{LV002,0}) = \frac{3600}{K_{v,LV002} \sqrt{\frac{\rho g (h_2(t) + h_{LV002})}{100000}}} \\ &\quad \cdot \frac{K_{v,LV001} f_1(u_{LV001,0})}{3600} \sqrt{\frac{\rho g (h_1(t) + h_{LV001})}{100000}} \\ &\Rightarrow u_{LV002,0} = f_2^{-1}(q_{LV002,0}). \end{aligned} \quad (4.18)$$

Similarly, as in (4.11), the output trajectories, $\tilde{y}_i(t)$, i.e., the trajectories of the measurements, are given by

$$\begin{cases} \tilde{y}_1(t) = \tilde{h}_1(t) \\ \tilde{y}_2(t) = \tilde{h}_2(t) \end{cases}. \quad (4.19)$$

The dynamics of the sensors are considered fast, compared with the two-tank process and, therefore, neglected. Thus, the output trajectory is determined by the state trajectory.

This results in the output equation approximation

$$\begin{cases} y_1(t) = \Delta h_1(t) \\ y_2(t) = \Delta h_2(t) \end{cases}. \quad (4.20)$$

The linearized two-tank model, equations (4.12) and (4.13), can be rewritten in compact matrix form

$$\begin{cases} \Delta \dot{x}(t) = A \Delta x(t) + B \Delta u(t) \\ y(t) = C \Delta x(t) \end{cases}, \quad (4.21)$$

where x denotes the vector of the states, u denotes the vector of the control inputs, y denotes the vector of outputs and matrices A and B are the Jacobian matrices $\frac{Dg_i}{Dx}(\cdot)$ and $\frac{Dg_i}{Du}(\cdot)$.

In this work, to evaluate MHE possibilities, the following discrete-time two-tank dynamical model representations will be considered:

- **Case 1.** Original two-tank system with two state variables, h_1 and h_2 , and two available measurements. In this case u_{PA001} is considered a known, controllable process disturbance. Thus, the linearized two-tank dynamical model is given by

$$\underbrace{\begin{bmatrix} \Delta h_{1,k+1} \\ \Delta h_{2,k+1} \end{bmatrix}}_{\Delta x_{k+1}} = \underbrace{\begin{bmatrix} \left. \frac{\partial g_1}{\partial h_1} \right|_O & 0 \\ \left. \frac{\partial g_2}{\partial h_1} \right|_O & \left. \frac{\partial g_2}{\partial h_2} \right|_O \end{bmatrix}}_A \cdot \underbrace{\begin{bmatrix} \Delta h_{1,k} \\ \Delta h_{2,k} \end{bmatrix}}_{\Delta x_k} + \underbrace{\begin{bmatrix} \left. \frac{\partial g_1}{\partial u_{LV001}} \right|_O & 0 & \left. \frac{\partial g_1}{\partial u_{PA001}} \right|_O \\ \left. \frac{\partial g_2}{\partial u_{LV001}} \right|_O & \left. \frac{\partial g_2}{\partial u_{LV002}} \right|_O & 1 \end{bmatrix}}_B \cdot \underbrace{\begin{bmatrix} \Delta u_{LV001,k} \\ \Delta u_{LV002,k} \\ \Delta u_{PA001,k} \end{bmatrix}}_{\Delta u_k}, \quad (4.22)$$

$$\underbrace{\begin{bmatrix} y_{1,k} \\ y_{2,k} \end{bmatrix}}_{y_k} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_C \cdot \underbrace{\begin{bmatrix} \Delta h_{1,k} \\ \Delta h_{2,k} \end{bmatrix}}_{\Delta x_k}. \quad (4.23)$$

- **Case 2.** Original two-tank system with two state variables, h_1 and h_2 , and one available measurement. In this case the state transition equation is the same as for Case 1, (4.22). Assuming that one of the measurements is not available, observability is maintained only if it is the measurement of water level in tank 1. Meaning that the measurement of water level in tank 2, $h_2(t)$, is necessary

condition for the system to be observable. Thus, the output equation for the Case 2 is

$$\underbrace{y_{2,k}}_{y_k} = \underbrace{\begin{bmatrix} 0 & 1 \end{bmatrix}}_C \cdot \underbrace{\begin{bmatrix} \Delta h_{1,k} \\ \Delta h_{2,k} \end{bmatrix}}_{\Delta x_k}. \quad (4.24)$$

- **Case 3.** Augmented two-tank system with three state variables, h_1 , h_2 and u_{PA001} , and two available measurements. The system model is augmented as presented in Section 2.7.3. In this case the linearized two-tank dynamical model has the following form

$$\underbrace{\begin{bmatrix} \Delta h_{1,k+1} \\ \Delta h_{2,k+1} \\ \Delta u_{PA001,k+1} \end{bmatrix}}_{\Delta x_{k+1}} = \underbrace{\begin{bmatrix} \left. \frac{\partial g_1}{\partial h_1} \right|_O & 0 & \left. \frac{\partial g_1}{\partial u_{PA001}} \right|_O \\ \left. \frac{\partial g_2}{\partial h_1} \right|_O & \left. \frac{\partial g_2}{\partial h_2} \right|_O & 0 \\ 0 & 0 & 1 \end{bmatrix}}_A \cdot \underbrace{\begin{bmatrix} \Delta h_{1,k} \\ \Delta h_{2,k} \\ \Delta u_{PA001,k} \end{bmatrix}}_{\Delta x_k} \quad (4.25)$$

$$+ \underbrace{\begin{bmatrix} \left. \frac{\partial g_1}{\partial u_{LV001}} \right|_O & 0 \\ \left. \frac{\partial g_2}{\partial u_{LV001}} \right|_O & \left. \frac{\partial g_2}{\partial u_{LV002}} \right|_O \\ 0 & 0 \end{bmatrix}}_B \cdot \underbrace{\begin{bmatrix} \Delta u_{LV001,k} \\ \Delta u_{LV002,k} \end{bmatrix}}_{\Delta u_k},$$

$$\underbrace{\begin{bmatrix} y_{1,k} \\ y_{2,k} \end{bmatrix}}_{y(t)} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_C \cdot \underbrace{\begin{bmatrix} \Delta h_{1,k} \\ \Delta h_{2,k} \\ \Delta u_{PA001,k} \end{bmatrix}}_{\Delta x_k}. \quad (4.26)$$

- **Case 4.** Augmented two-tank system with three state variables, h_1 , h_2 and u_{PA001} , and one available measurement. In this case the state trajectory remains as for Case 3, (4.25), the output equation is adjusted as in Case 2

$$\underbrace{y_{2,k}}_{y_k} = \underbrace{\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}}_C \cdot \underbrace{\begin{bmatrix} \Delta h_{1,k} \\ \Delta h_{2,k} \\ \Delta u_{PA001,k} \end{bmatrix}}_{\Delta x_k}. \quad (4.27)$$

All four cases presented above maintain observability of the two-tank system, however, the observability of the disturbance u_{PA001} is highly reduced if only one measurement is present, i.e., Case 4. As mentioned in Section 2.2, the observability Grammian provides

information on observability "degree" in different state space directions. The system is said to be observable if all eigenvalues to the observability Grammian are strictly positive, thus the eigenvalues close to zero indicate low observability. Figure 4.1 gives an overview of 1000 operating points inside the feasible region of the two-tank system and corresponding observability Grammian eigenvalues. The subplot A, show the relation between the eigenvalues of u_{PA001} and h_1 at different operating points when only one measurement is present. The subplot B, compares the eigenvalues of u_{PA001} given one measurement versus two measurements. It is evident that despite the variations from one operating point to another, the eigenvalue of u_{PA001} given one measurement is in range of 10^{-5} , i.e., are close to zero. The eigenvalues of u_{PA001} given two measurements, on the other hand, lies in range of 10^{-2} .

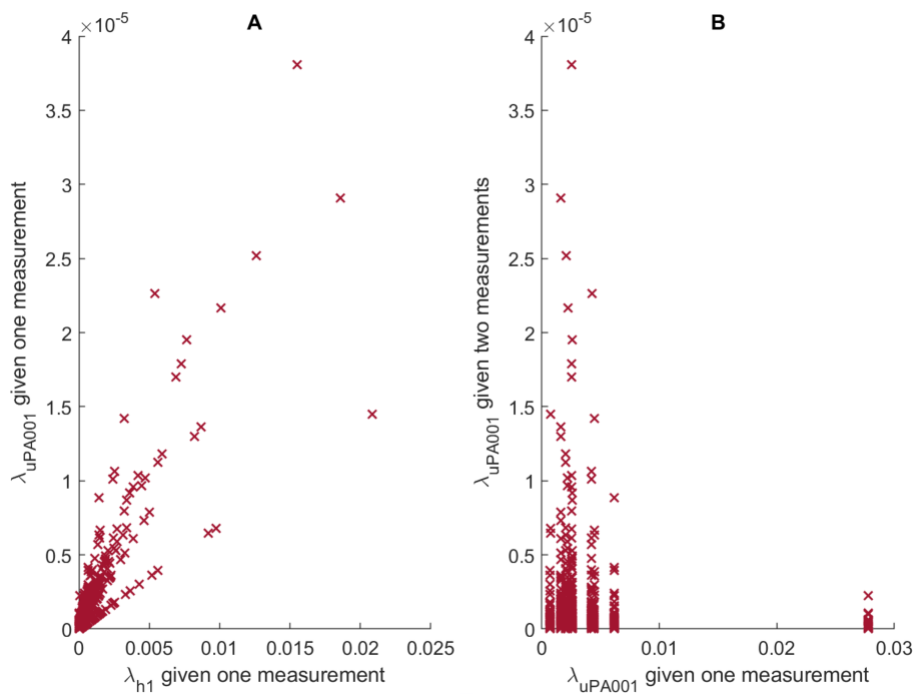


Figure 4.1: Eigenvalues of the observability Grammian plotted over the feasible region of the two-tank system. The subplot A represents the eigenvalues of u_{PA001} and h_1 when one measurement is available. Subplot B compares the eigenvalues of u_{PA001} when one measurement is available versus when two measurements are available.

4.1.2 Implementation

Simulation and implementation of the dynamical two-tank system models and LMHE is done in Matlab.

The Two-tank Sstem Simulation

The two-tank dynamical models both nonlinear and linearized are implemented as continuous-time systems and discretized using forward Euler method. Discretization is necessary due to the fact that MHE works in discrete time using measurements available at discrete time samples. Code 4.1 shows the snippet from *LMHE.m*. In addition, Euler integration is bounded by the systems physical constraints. The control inputs for the valves and the pump, u_{LV001} , u_{LV002} , u_{PA001} are defined by the user, as well as initial states, sample time and simulation length. Both state and output equations can be simulated with or without process or measurement noise. The noise is generated as normally distributed random numbers based on user defined variances.

```

% Linear Process model
if n == 3
    dx_dt_k = A*(x_k - [h1_0; h2_0; u_PA001_0]) +
              B*(u_k - [u_LV001_0; u_LV002_0]) + w_k;
elseif n == 2
    dx_dt_k = A*(x_k - [h1_0; h2_0]) +
              B*(u_k - [u_LV001_0; u_LV002_0; u_PA001_0]) + w_k;
end
x_k_ny = Ts*dx_dt_k + x_k + v_k; %Euler integration
%Measurements
h_meas_k = C*x_k + D*0 + v_k;
...
...
% Nonlinear Process model
f3_k = interp1(u_PA001,q_PA001, u_PA001_k);
f1_k = interp1(u_LV001,f_LV001,u_LV001_k);
f2_k = interp1(u_LV002,f_LV002,u_LV002_k);

dh1_dt_k = (1/A1)*(f3_k-((Kv_LV001*f1_k)/3600
                    *sqrt((rho*g*(h1_k+h_LV001))/100000)));
dh2_dt_k = (1/(0.07*h2_k+0.004))*(((Kv_LV001*f1_k)/3600
                    *sqrt((rho*g*(h1_k+h_LV001))/100000) - (Kv_LV002*f2_k)/3600
                    *sqrt((rho*g*(h2_k+h_LV002))/100000)));

h1_k_ny = Ts*dh1_dt_k + h1_k + w1_k; %Euler integration
h2_k_ny = Ts*dh2_dt_k + h2_k + w2_k;

h1_meas_k = h1_k + v1_k;
h2_meas_k = h2_k + v2_k;

```

Code 4.1: Linear and nonlinear process models.

LMHE

Figure 4.2 illustrates the concept behind one time instance of MHE. The main idea is to provide MHE with the horizons of the control inputs, measurements and initial guesses of the state variables, where $u_{horizon} \in \mathbb{R}^{p \times N}$, $y_{horizon} \in \mathbb{R}^{m \times N}$ and $x_{horizon} \in \mathbb{R}^{n \times N}$. There are two main tasks to be performed at each time instance: update the arrival cost covariance matrix that is used to approximate the arrival cost in the objective function, and solve the optimization problem that returns the estimates of the state variables.

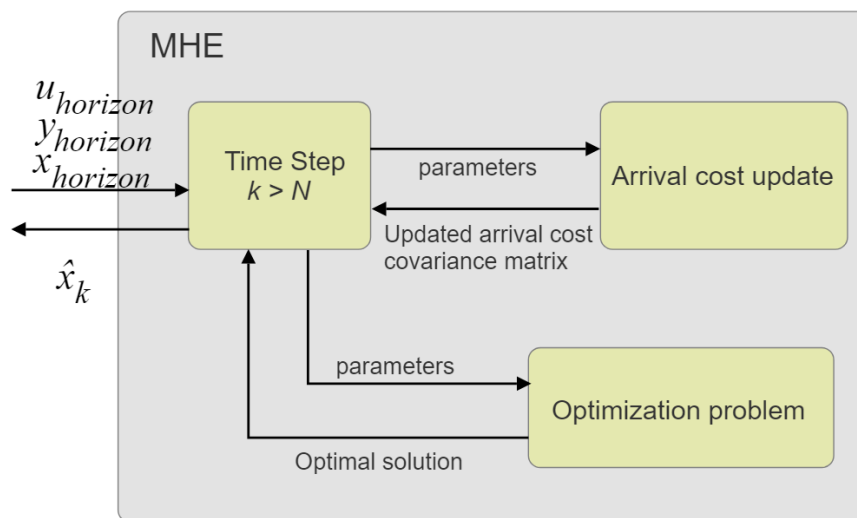


Figure 4.2: An overview of Moving Horizon Estimation iteration.

As described in the previous sections, LMHE is based on linear dynamical model, thus the linear two-tank system representation is used in both the objective function formulation and the next sample prediction. More detailed description of LMHE algorithm is given in Algorithm 4.1. The algorithm starts with the initialization of the main parameters such as the number of states, n , the number of available measurements, m , the horizon length, N , the sample time, T_s , covariance matrices of the MHE, the operating point of the state variables, x_0 , and control input/state variable $u_{PA001,0}$. The two-tank system parameters are stored in separate file called *parameters.m*. The first step is linearization of the two-tank system around provided state trajectory.

```
[A,B,C,D,u_LV001_0,u_LV002_0] = linearization(h1_0,h2_0,u_PA001_0,n,m)
```

The function `linearization(·)` calculates and returns the Jacobians of linearized continuous-time two-tank system, and associated control inputs $u_{LV001,0}$ and $u_{LV002,0}$. The arrangement of system matrices are defined by the inputs n and m . The box constraints

\mathbf{x}_{lb} and \mathbf{x}_{ub} are defined based on the systems state variable upper and lower bounds. The box constraints are $n \times N$ matrices. The horizons of the state estimates, $\hat{\mathbf{x}}_{horizon}$, control inputs, $\mathbf{u}_{horizon}$, and the measurements $\mathbf{y}_{horizon}$ are initialized as zero vectors of length N . The MHE optimization problem is solved repeatedly at each time instance. At each iteration, a new initial state is the second element from the previous estimation horizon. Thus, the horizons of control inputs and measurements are updated at the beginning of each time instance. The horizons are shifted to the right, i.e., taking into consideration the newest measurement and control inputs while dropping the oldest.

Algorithm 4.1 LMHE

Initialization: $n, m, N, Ts, Q, R, P_0, \mathbf{x}_0, \mathbf{u}_0$
 $\mathbf{x} \leftarrow \mathbf{x}_0$
Run *parameters.m* ▷ Parameters of the two-tank system
Perform *linearization*(\cdot)
Determine the constraints: $\mathbf{x}_{lb}, \mathbf{x}_{ub}$
Pre-allocation of horizons: $\hat{\mathbf{x}}_{horizon}, \mathbf{u}_{horizon}, \mathbf{y}_{horizon}$
Pre-allocation of array for storage: $P_{k-N, arr}$
for $k = 1 : t_length$ **do**
 $\mathbf{u}_{horizon} \leftarrow [\mathbf{u}_{horizon}(:, 2 : N), \mathbf{u}_k]$ ▷ Update horizons
 $\mathbf{y}_{horizon} \leftarrow [\mathbf{u}_{horizon}(:, 2 : N), \mathbf{y}_k]$

 if $k \leq N$ **then** ▷ MHE Initialization
 $[\hat{\mathbf{x}}_k, P] \leftarrow KalmanFilter(\cdot)$
 $\hat{\mathbf{x}}_{horizon}(k, :) \leftarrow \hat{\mathbf{x}}_k$ ▷ Update horizons
 $\mathbf{x} \leftarrow \hat{\mathbf{x}}_k$
 $P_{k-N, arr}(k) \leftarrow P$
 end if
 if $k > N$ **then**
 $x_{pred, k} \leftarrow LinearModel(\cdot)$ ▷ Predict the state and update horizons
 $\hat{\mathbf{x}}_{horizon} \leftarrow [\hat{\mathbf{x}}_{horizon}(2 : N), \mathbf{x}_{pred, k}]$
 $P \leftarrow RiccatiUpdate(\cdot)$
 $P_{k-N, arr}(k) \leftarrow P$
 $\bar{x}_{k-N} \leftarrow \hat{\mathbf{x}}_{horizon}(:, 1)$
 $P_{k-N} \leftarrow P_{k-N, arr}(k)$
 $\hat{\mathbf{x}}_{k, horizon}^* \leftarrow \mathbf{minimize} \quad ObjectiveFunction(\cdot)$ ▷ Linear objective function
 $\hat{\mathbf{x}}_{horizon} \leftarrow \hat{\mathbf{x}}_{k, horizon}^*$
 end if
end for

As mentioned in Section 2.7.1, to ensure MHE stability, the arrival cost approximation requires that the arrival cost covariance matrix P_{k-N} , and the state prediction \bar{x}_{k-N} , are updated recursively subject to the initial condition P_0 and \bar{x}_0 , and the measurements up until time $k = N - 1$. Therefore, while $k \leq N$, the state variables and covariance matrix, P , are estimated using a Kalman filter.

```
[x_hat,P_hat] = KalmanFilter(x_hat,P_hat,y_meas_k,u_k,parameter_struct,Q,R)
```

where *parameter_struct* is the structure utilized to store a variety of parameters and facilitate their transfer between functions. The function `KalmanFilter(·)` is implemented as described in Section 2.4. At each iteration, while $k \leq N$, the initial $\hat{\mathbf{x}}_{horizon}$ is updated with the Kalman estimates. The covariance matrix, P , is also stored for the future use.

At the time instance $k > N$, the horizons of all variables are full. The optimization algorithms requires a good initial guess of the optimization variables. The best initial guess of the state variables at time k is the optimal solution found at time $k - 1$. The only issue is that the optimal solutions from the previous horizon, $\hat{\mathbf{x}}_{k-1,horizon}^*$, are the solutions from time sample $k - N - 1$ up until $k - 1$. Thus, the sample $k - N - 1$ needs to be left out and the prediction for time index k needs to be added. The state prediction at time k is calculated using the system model, i.e., linearized two-tank dynamical model:

```
[x_next, y] = LinearModel(x,u,parameter_struct)
```

The updated $\hat{\mathbf{x}}_{horizon}$ is supplied to the optimizer as the initial guess, and the sample $k - N$ is the state prediction, \bar{x}_{k-N} , used in arrival cost. At this point, the arrival cost covariance matrix P is updated using the `RiccatiUpdate(·)` function, that is the Eq. (2.36).

```
P = RiccatiUpdate(P, parameter_struct)
```

The objective function to be minimized is implemented as (4.2) and has the following required inputs:

```
J = ObjectiveFunction(x,x_pred_k_N,u,y_meas,parameter_struct,P,type)
% Returns: objective function value
% Inputs:
% x          -- The horizon vector of optimal state estimates
% x_pred_k_N -- The prediction vector of the states at time k-N
% u          -- The horizon vector of control inputs
% y_meas     -- The horizon vector of measured outputs
% parameter_struct -- The struct with parameters. For Linear MHE required
%                parameters are the sytem matrices A, B, C, D and covariance
%                matrices Q, R. For nonlinear MHE required parameters are
%                all system parameters and covariance matrices Q and R.
%                Other required parameters are: the sampling
%                time,Ts,the horizon length, N, the number of states,
%                n, and the number of measurements, m.
```

```

% P          -- Estimate error covariance matrix
% type      -- The string of type of the MHE, where 'L' stands for
%           linear and 'N' stands for nonlinear MHE

```

Most of the in-built solvers in Matlab require that the objective function is passed to the solver via a function handler. Hence, the solver iterates from an initial guess towards the local minimum by repeatedly evaluating the objective function with new sets of potentially optimal values to the optimization variables. The solution provided by optimizer yields the sequence of the optimal state estimates $\hat{\mathbf{x}} = [\hat{x}_{k-N}^T, \dots, \hat{x}_k^T]^T$, where \hat{x}_k^T is the optimal state estimates for the time sample k .

4.1.3 In-built *fmincon* for Constrained Optimization

Based on the optimization problem at hand the optimizer that can handle multivariable constrained programming problems is the *fmincon* function. *fmincon* is a nonlinear programming solver that minimizes a problem of the following form [24]

$$\min_{\mathbf{x}} f(\mathbf{x}), \quad \text{subject to} \begin{cases} c(\mathbf{x}) \leq 0, \\ c_{eq}(\mathbf{x}) = 0, \\ A\mathbf{x} \leq \mathbf{b}, \\ A_{eq}\mathbf{x} = \mathbf{b}_{eq}, \\ lb \leq \mathbf{x} \leq ub, \end{cases} \quad (4.28)$$

where \mathbf{b} and \mathbf{b}_{eq} are vectors, A and A_{eq} are matrices, $c(\mathbf{x})$ and $c_{eq}(\mathbf{x})$ are functions that return vectors, and $f(x)$ is a function that returns a scalar. $f(\mathbf{x})$, $c(\mathbf{x})$, and $c_{eq}(\mathbf{x})$ can be nonlinear functions. lb and ub can be passed as vectors or matrices. *fmincon* has five algorithm options: *interior-point*, *trust-region-reflective*, *sqp*, *sqp-legacy* and *active-set*. Matlab recommends to use the *interior-point* algorithm first. This is because it handles large, sparse problems, as well as small dense problems. It is a large-scale¹ algorithm. The next recommendation is to use *sqp* algorithm in order to obtain more speed on small- or medium-size problems. Lastly, *active-set* should be tried if even more speed is required. *sqp-legacy* is similar to *sqp*, but usually much slower and uses

¹Large-scale algorithms use sparse linear algebra, which means that the algorithms do not need to store or operate on full matrices. Instead, they use sparse matrices, which take up less memory and can be computed more efficiently.

more memory, thus *sqp-legacy* will not be considered. *Trust-region-reflective* require a pre-defined gradient of the objective function, therefore the algorithm will not be included in this paper. Both *sqp* and *active-set* are very similar algorithms and are the variants of SQP method presented in 3.2. Both algorithms uses a quasi-Newton method to iterate toward a solution such that KKT condition holds. The Hessian is approximated using BFGS update. The main differences between *sqp* and *active-set* algorithms are that *sqp* only takes steps in the feasible region while *active-set* can take some steps outside the feasible region. In addition, *sqp* uses some techniques to adjust the step length in case an objective function return *Nan*, *Inf* or a complex value [25]. Fig. 4.3 shows a typical *fmincon* optimization process.

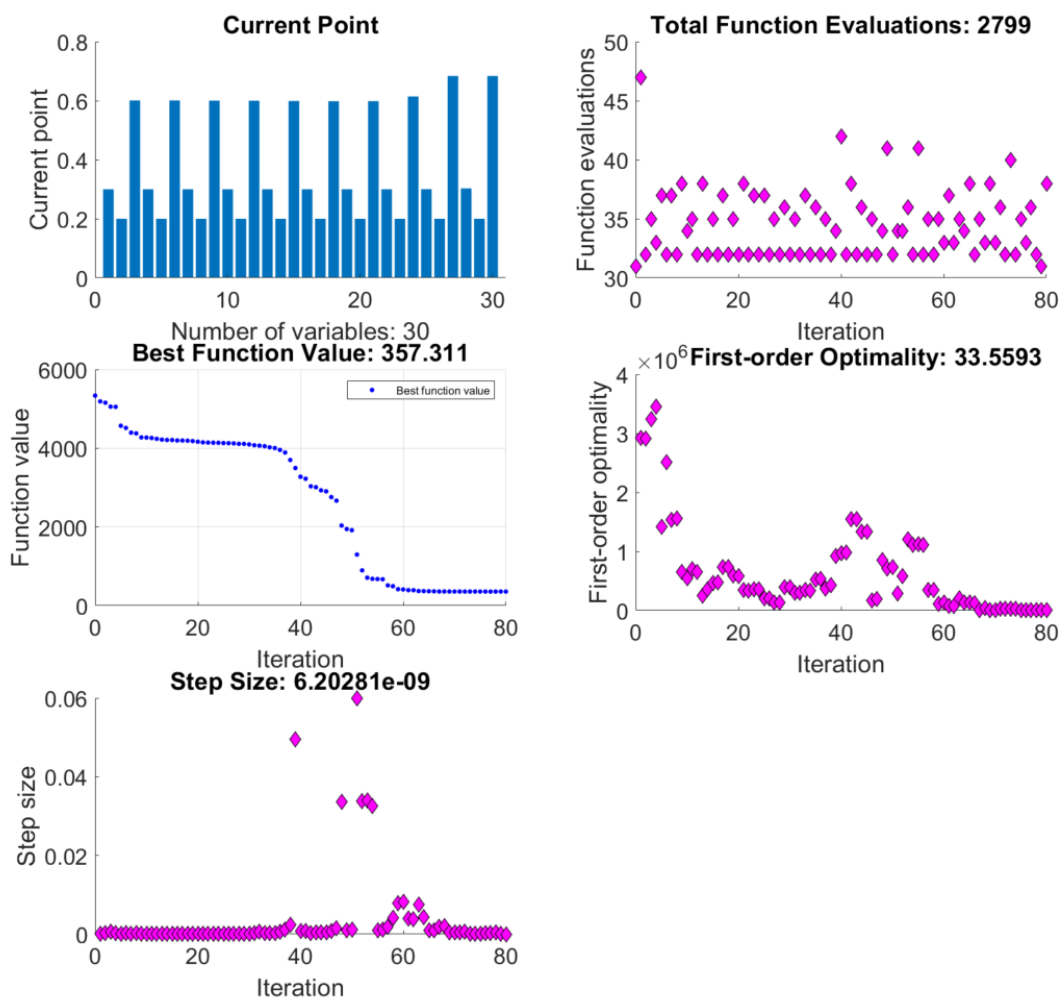


Figure 4.3: Typical *fmincon* optimization process.

The *Current Point* subplot displays all optimization variables and their current values, i.e., the optimal solution. The subplot *Function Evaluation* shows an overview of number

of function evaluations at each iteration. The subplot *Function value* plots the objective function value at each iteration. The decrease in the function value indicates a progress, *fmincon* moves towards the local minimum. The subplot *First-Order Optimality* plots first-order optimality value versus iteration. First-order optimality is a measure of how close a point x is to optimal. This is a necessary but not a sufficient condition. This means that the first-order optimality measure must be zero at a minimum, but a point with first-optimality equal to zero is not necessarily a minimum [24]. The last subplot plots the Newton step size at various iterations.

The *fmincon* function provides a set of global and algorithm-dependent options that can be adjusted using *optimoptions* function. The user can adjust the maximum number of function evaluations, maximum number of iterations, optimality tolerance, constraint tolerance etc. Matlab also provides an option to compute the gradients of the objective function and the constraints in parallel using Parallel Computing Toolbox for more efficient processing.

4.1.4 Verification of LMHE

The implementation of the LMHE algorithm is verified by simulating the linearized two-tank system without process or measurement noise. The sample time is set to $0.2s$. The test is conducted for all four cases described in 4.1.1. For comparison, the estimation is performed both by LMHE and Kalman filter. Fig 4.4 illustrates the Case 1, i.e., estimation of h_1 and h_2 , given the measurements of h_1 and h_2 . The optimization for the LMHE is solved using default *sqp* algorithm and the horizon length $N = 10$. Since the two-tank process is simulated without noise, the covariance matrices for both the LMHE and KF are set with small values:

$$Q = R = \begin{bmatrix} 10^{-12} & 0 \\ 0 & 10^{-12} \end{bmatrix}. \quad (4.29)$$

The system is noise free, thus both the model and the measurement errors are weighted equally. Both estimators gives good results²:

LMHE :

Estimation error mean h1: 4.45e-07

Estimation error mean h2: 3.43e-07

²The estimation error mean is calculated using the absolute value of the estimation error.

KF:

Estimation error mean h_1 : $9.14e-06$

Estimation error mean h_2 : $1.09e-08$

The mismatch between the estimators is small and varies in time, an example of estimation error is illustrated in the enhanced sections of the plot.

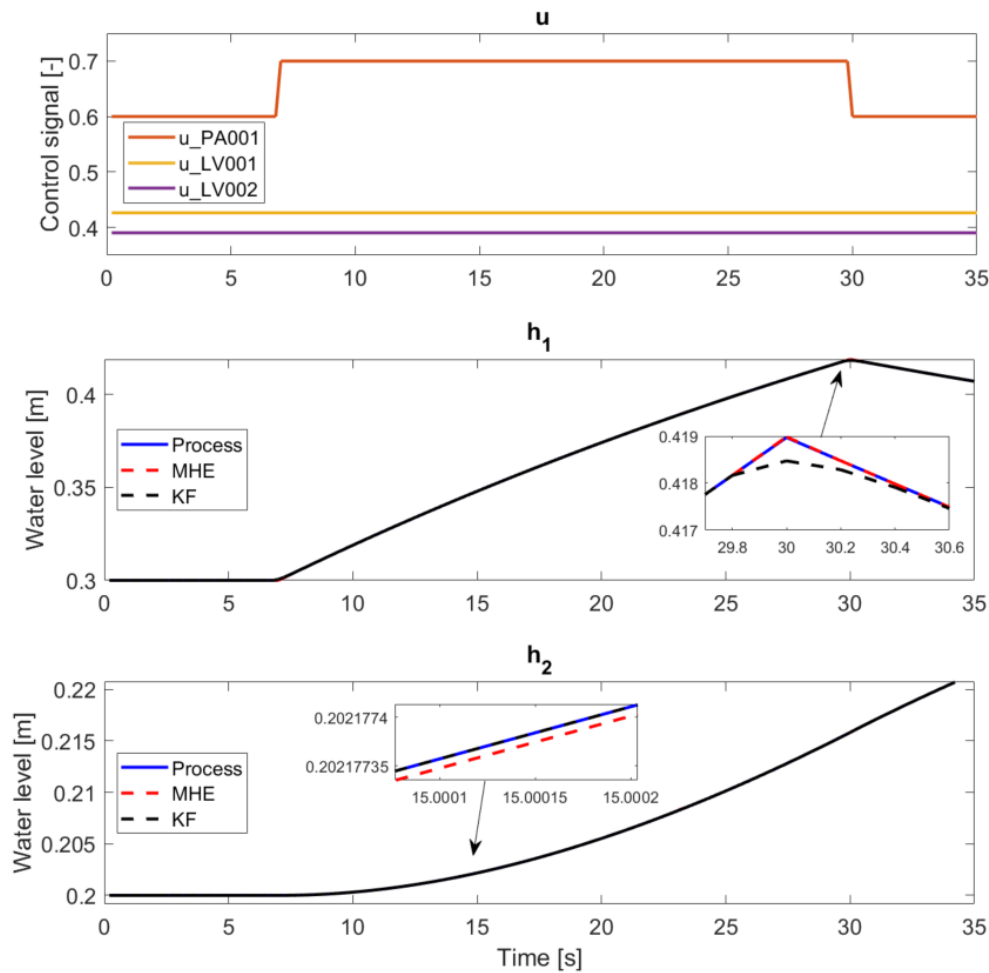


Figure 4.4: LMHE Case 1: Estimation of h_1 and h_2 given the measurements of h_1 and h_2 . No process or measurement noise.

The same simulation is done for the Case 2, i.e., estimation of h_1 and h_2 , given the measurement h_2 . The results are shown in Fig. 4.5. In this case the model error covariance matrices, Q , are adjusted to weigh the model error term in the objective function more than the measurement error term. The first element in Q_{MHE} is decreased more compared to the second element. This means that there is more confidence in the model describing h_1 . The same approach is applied to the Kalman filter, but since MHE uses the inverse of the Q and R in the objective function, the result is opposite, i.e., the

first element in Q_{KF} is increased:

$$Q_{MHE} = \begin{bmatrix} 10^{-16} & 0 \\ 0 & 10^{-14} \end{bmatrix}, \quad Q_{KF} = \begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{-12} \end{bmatrix}, \quad (4.30)$$

$$R_{MHE} = R_{KF} = 10^{-12}.$$

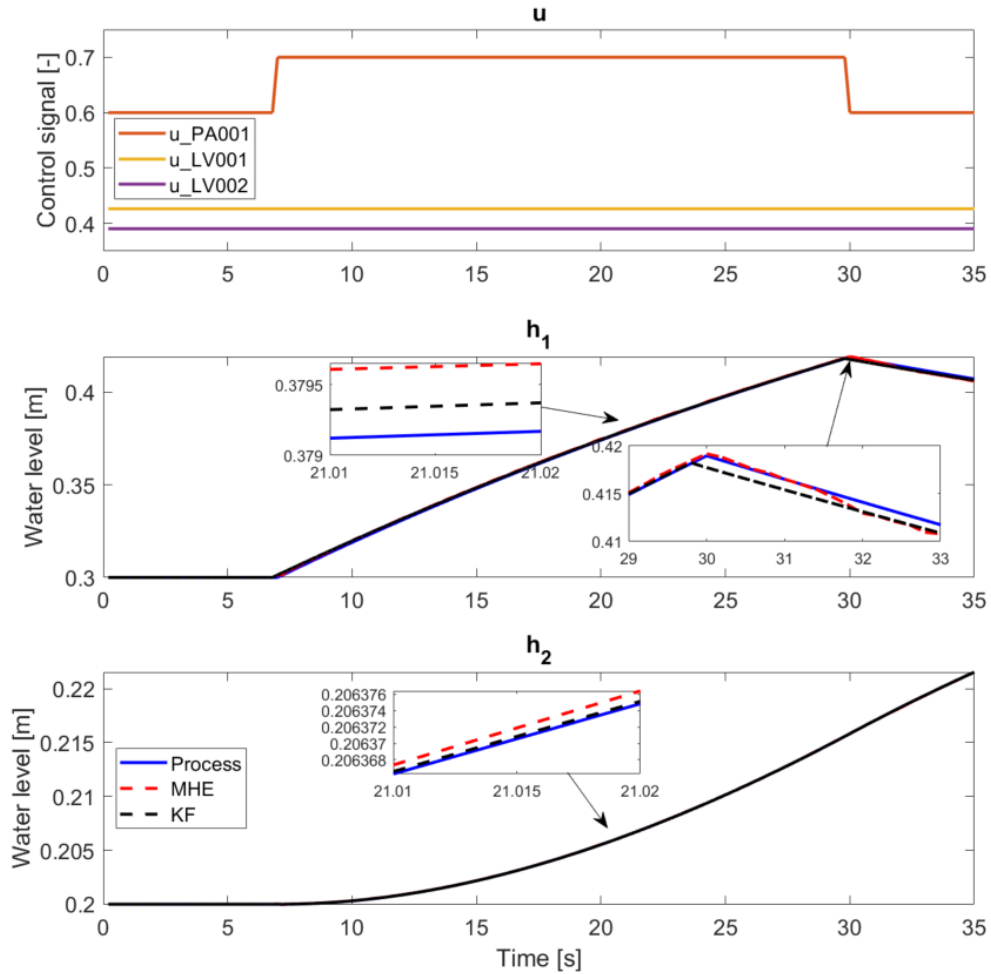


Figure 4.5: LMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . No process or measurement noise.

The estimation error mean has increased compared with Case 1, but it is still very low:

LMHE :

Estimation error mean h1: 4.46e-04

Estimation error mean h2: 6.80e-06

KF :

Estimation error mean h1: 4.02e-04

Estimation error mean h2: 5.46e-07

The performance of both estimators can be further improved by adjusting covariance matrices to find the best match. The MHE have even more tuning parameters, such as horizon length, the type of algorithm used and various algorithm related parameters.

The results for the Case 3, the estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 , is shown in Fig. 4.6. In this case the covariance matrices to MHE are changed to:

$$Q_{MHE} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 8 \cdot 10^3 \end{bmatrix}, \quad Q_{KF} = \begin{bmatrix} 10^{-12} & 0 & 0 \\ 0 & 10^{-12} & 0 \\ 0 & 0 & 10^{-8} \end{bmatrix}, \quad (4.31)$$

$$R_{MHE} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R_{KF} = \begin{bmatrix} 10^{-12} & 0 \\ 0 & 10^{-12} \end{bmatrix}.$$

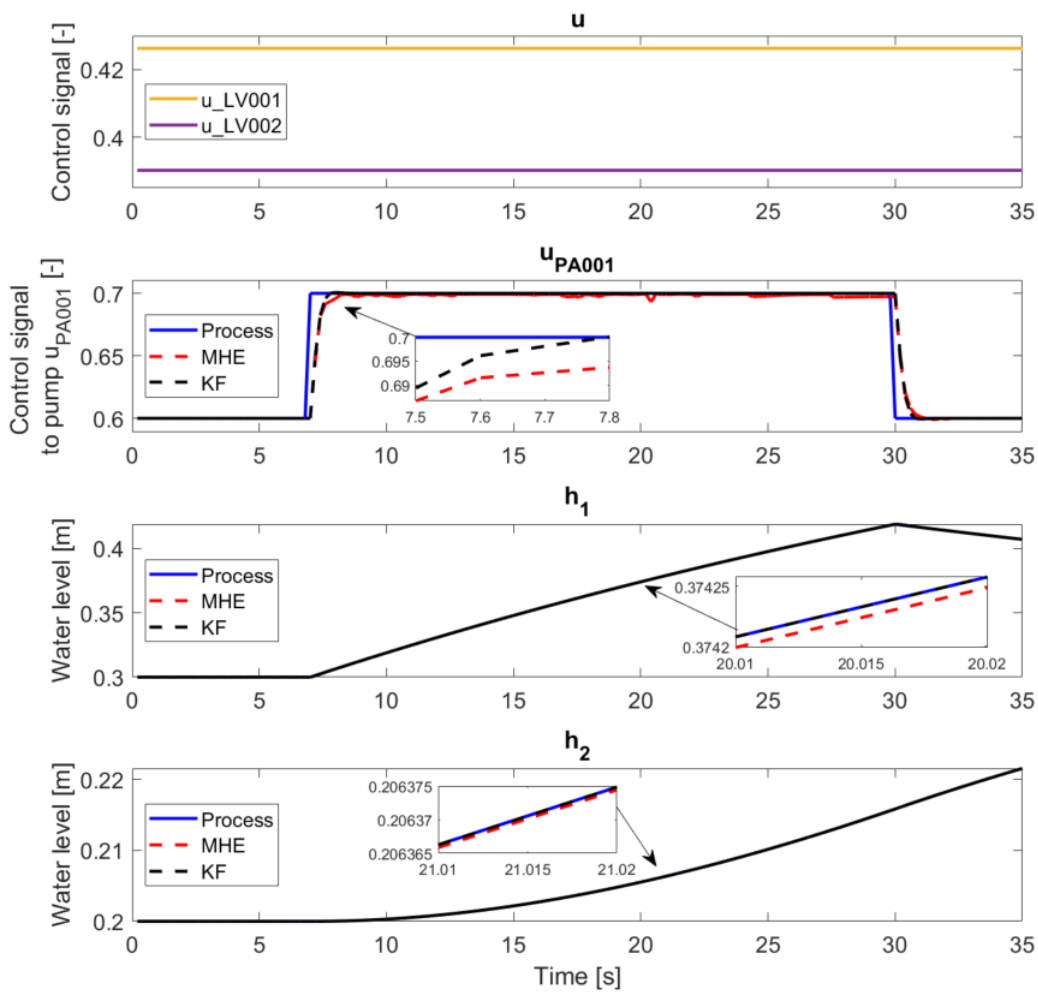


Figure 4.6: LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . No process or measurement noise.

During the testing of different combinations of weighting matrices it has been observed that very small values results in oscillations. The estimator becomes very sensitive to the choice of the weights. This can be due to extremely high values of the objective function. The same proportion between Q_{MHE} and R_{MHE} but multiplied with 10^{12} gives more stable estimation. As for the Kalman filter, the response is similar in both cases, if the proportion of the Q_{KF} and R_{KF} is kept the same.

The model and the measurement errors, h_1 and h_2 , are weighted equally, whereas the element representing u_{PA001} in the model error covariance matrix is weighted less, i.e., the confidence that the disturbance u_{PA001} is constant is reduced.

The estimation error means for the LMHE and KF are the following

```

LMHE:
Estimation error mean h1: 1.55e-05
Estimation error mean h2: 1.46e-07
Estimation error mean u_PA001: 3.05e-03
KF:
Estimation error mean h1: 4.44e-06
Estimation error mean h2: 4.53e-09
Estimation error mean u_PA001: 1.96e-03

```

The search process of the model error covariance matrix is based on the trial-and-error procedure. The stopping criterion is based on the desirable estimator performance. Fig. 4.7 illustrates the LMHE Case 3 estimation performance using different weights, the first or third element in matrix Q_{MHE} is increased or decreased by factor 10.

By increasing Q_1 (light blue curve), the confidence in the model describing h_1 is reduced, thus the measurement of h_1 is trusted more. This results in more accurate estimate of h_2 but less accurate estimate of h_1 and u_{PA001} . Decreasing Q_1 (green curve) results in opposite performance, the accuracy of h_1 and u_{PA001} estimates is improved and the accuracy of h_2 estimate is degraded. By decreasing Q_3 (yellow curve) instead of increasing Q_1 , the confidence that the disturbance u_{PA001} is constant is increased, thus compared with the light blue curve the estimate of h_1 is less accurate while the estimate of u_{PA001} is improved. By decreasing the confidence that the disturbance u_{PA001} is constant (purple curve) results in faster response and more accurate estimates. It is evident that the choice of model error covariance matrix weights permits a trade-off between the accuracy of estimating different states, allowing for prioritization based on their importance.

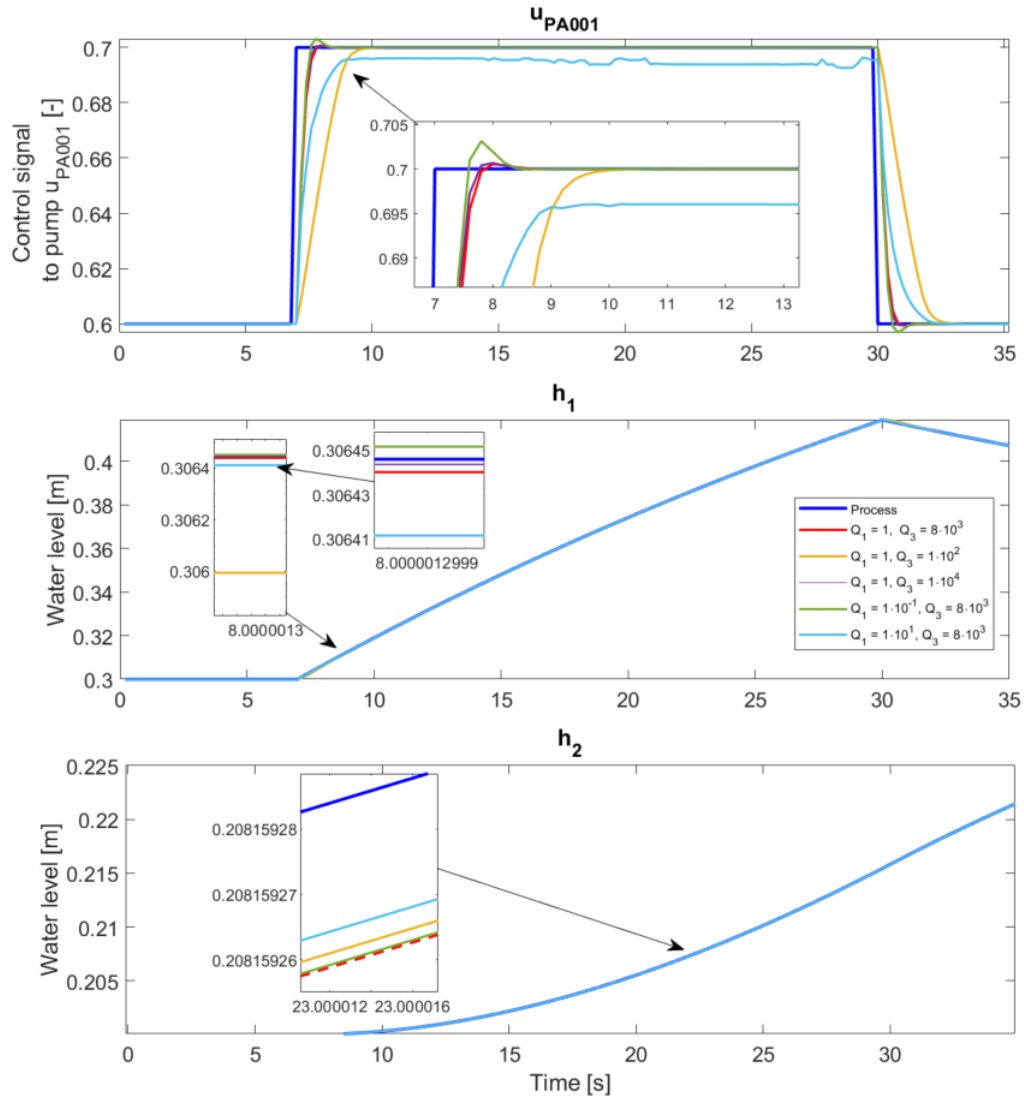


Figure 4.7: LMHE Case 3 estimation using different weights in the model covariance matrix. The second element (representing h_2) is kept constant, $Q_2 = 1$.

The results for the Case 4, estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 , are shown in Fig. 4.8. The covariance matrices are adjusted as follows:

$$Q_{MHE} = \begin{bmatrix} 10^{-7} & 0 & 0 \\ 0 & 10^{-10} & 0 \\ 0 & 0 & 5 \cdot 10^{-1} \end{bmatrix}, \quad Q_{KF} = \begin{bmatrix} 10^{-6} & 0 & 0 \\ 0 & 10^{-12} & 0 \\ 0 & 0 & 10^{-2} \end{bmatrix}, \quad (4.32)$$

$$R_{MHE} = 10^{-10}, \quad R_{KF} = 10^{-12}.$$

The weights in Q are distributed such that the trust in h_2 is highest while the trust in u_{PA001} is lowest, i.e., the model describing h_1 and h_2 is trusted more than the assumption that the disturbance u_{PA001} is a constant.

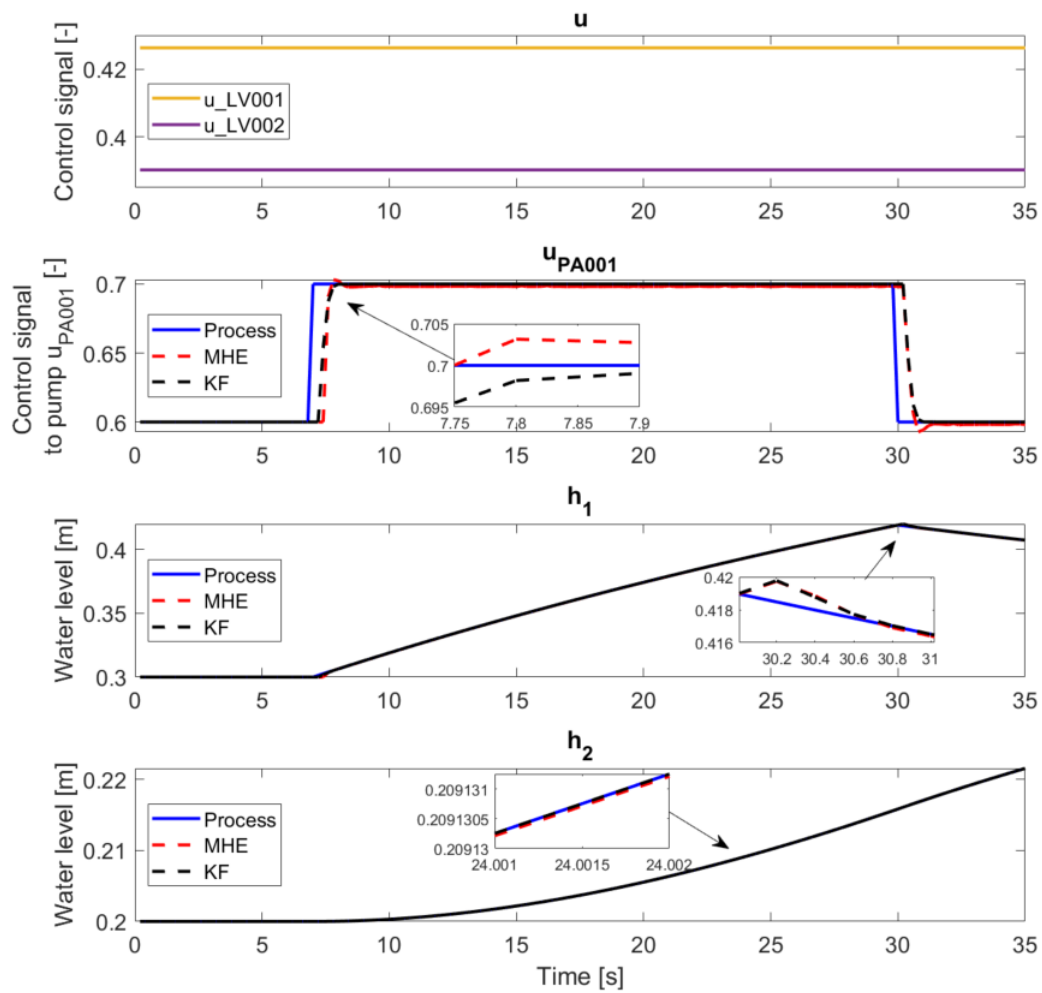


Figure 4.8: LMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . No process or measurement noise.

The results are very good considering that only one measurement is used. The response of MHE u_{PA001} is delayed by one sample and has a small overshoot, however, the error mean for both the MHE and the KF is similar:

```

LMHE:
Estimation error mean h1: 6.45e-05
Estimation error mean h2: 5.04e-08
Estimation error mean u_PA001: 4.53e-03
KF:
Estimation error mean h1: 2.68e-05
Estimation error mean h2: 2.02e-09
Estimation error mean u_PA001: 2.97e-03

```

In general, the MHE and the KF estimator performances are good in all four cases. The Kalman filter is more stable and less sensitive to the small changes in covariance matrices.

This is not the case for the MHE. In addition, there are many other parameters that may improve or deteriorate the performance of the estimator. Thus, the tuning of MHE can become quite a challenging and time consuming task.

4.2 Nonlinear Moving Horizon Estimation

4.2.1 Implementation

The NMHE algorithm implementation uses the same approach as LMHE presented in the previous section. High-level NMHE algorithm is illustrated in Algorithm 4.2. The main difference between linear and nonlinear moving horizon estimation is that for NMHE the optimization problem is formulated using nonlinear differential equations. For the nonlinear case, the arrival cost approximation and the first horizon estimation is done using Extended Kalman filter. The implementation of EKF is according to equations derived in Section 2.5.

The function `Amatrix(·)` in Algorithm 4.2 calculates the Jacobians to the state transition matrix around the current state trajectory.

```
A = Amatrix(h1_0, h2_0, u_PA001_0, u_LV001_0, u_LV002_0, parameter_struct)
```

The function returns discretized state transition matrix used to update the arrival cost covariance matrix P .

4.2.2 Verification of NMHE

The implementation of the NMHE algorithm is verified the same way as the LMHE. However, the two-tank system is simulated using nonlinear model and the estimator performance is compared to the Extended Kalman filter rather than Kalman filter. In addition, to be able to compare linear and nonlinear MHE results, the two tank system is simulated around the same state trajectory, the optimization problem is solved by default *sqp* algorithm and the horizon length $N = 10$.

Algorithm 4.2 NMHE

Initialization: $n, m, N, Ts, Q, R, P_0, \mathbf{x}_0, \mathbf{u}_0$
 $\mathbf{x} \leftarrow \mathbf{x}_0$
Run *parameters.m* ▷ Parameters of the two-tank system
Determine the constraints: $\mathbf{x}_{lb}, \mathbf{x}_{ub}$
Pre-allocation of horizons: $\hat{\mathbf{x}}_{horizon}, \mathbf{u}_{horizon}, \mathbf{y}_{horizon}$
Pre-allocation of array for storage: $P_{k-N, arr}$
for $k = 1 : t_length$ **do**
 $\mathbf{u}_{horizon} \leftarrow [\mathbf{u}_{horizon}(:, 2 : N), \mathbf{u}_k]$ ▷ Update horizons
 $\mathbf{y}_{horizon} \leftarrow [\mathbf{u}_{horizon}(:, 2 : N), \mathbf{y}_k]$

 if $k \leq N$ **then** ▷ MHE Initialization
 $[\hat{\mathbf{x}}_k, P] \leftarrow ExtendedKalmanFilter(\cdot)$
 $\hat{\mathbf{x}}_{horizon}(k, :) \leftarrow \hat{\mathbf{x}}_k$ ▷ Update horizons
 $\mathbf{x} \leftarrow \hat{\mathbf{x}}_k$
 $P_{k-N, arr}(k) \leftarrow P$
 end if
 if $k < N$ **then**
 $x_{pred, k} \leftarrow NonlinearModel(\cdot)$ ▷ Predict the state and update horizons
 $\hat{\mathbf{x}}_{horizon} \leftarrow \mathbf{x}_{pred, k}$
 $A \leftarrow Amatrix(\cdot)$ ▷ Linearization around current state trajectory
 $P \leftarrow RiccatiUpdate(\cdot)$
 $P_{k-N, arr}(k) \leftarrow P$
 $\bar{x}_{k-N} \leftarrow \hat{\mathbf{x}}_{horizon}(:, 1)$
 $P_{k-N} \leftarrow P_{k-N, arr}(k)$
 $\hat{\mathbf{x}}_k^* \leftarrow \mathbf{minimize} \quad ObjectiveFunction(\cdot)$ ▷ Nonlinear objective function
 $\hat{\mathbf{x}}_{horizon} \leftarrow \hat{\mathbf{x}}_k^*$
 end if
end for

The covariance matrices for the NMHE and EKF for the Case 1 is set to:

$$Q = R = \begin{bmatrix} 10^{-12} & 0 \\ 0 & 10^{-12} \end{bmatrix}. \quad (4.33)$$

Since the system is simulated without noise, the model and measurement error covariance matrices are weighted equally. The results are very similar to the LMHE and KF:

NMHE:

Estimation error mean h1: 8.13e-07

Estimation error mean h2: 7.05e-07

EKF:

Estimation error mean h1: 8.24e-06

Estimation error mean h2: 9.02e-09

The figure illustrating the Case 1 simulation is available in Appendix B, Fig. B.1.

In the Case 2, to achieve at least the same accuracy as for the LMHE, the covariance matrices for the NMHE and EKF are modified as follows:

$$Q_{MHE} = \begin{bmatrix} 10^{-1} & 0 \\ 0 & 10^5 \end{bmatrix}, \quad Q_{EKF} = \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-12} \end{bmatrix}, \quad (4.34)$$

$$R_{MHE} = 1, \quad R_{EKF} = 10^{-12}.$$

The search process of the model error covariance matrix is based on the trial-and-error approach. The NMHE has a slightly lower h_2 estimation error mean compared with the LMHE:

```

NMHE:
Estimation error mean h1: 2.67e-04
Estimation error mean h2: 1.05e-08
EKF:
Estimation error mean h1: 2.06e-04
Estimation error mean h2: 2.13e-07

```

The EKF performance in the Case 2 is identical to the KF. The results are available in Appendix B, Fig. B.2.

During the testing it has been observed that too small values of the elements in the covariance matrices or highly unbalanced covariance matrices³ results in a singular arrival cost covariance matrix, P , i.e., P is not invertible. This yields the EKF and the NMHE since both estimators are using the same covariance matrix update technique. The cause of the matrix P becoming singular can be attributed to the limitations of the linearization approximation used in the EKF. The Kalman filter, on the other hand, assumes linearity and can analytically propagate uncertainty through the linear system equations, thus always produce a non-singular matrix P . The results for the Case 3 are shown in Fig. 4.9. In this case to find a good match of covariance matrices is a more challenging task due to the matrix P issue. The best results are achieved using the following:

$$Q_{MHE} = \begin{bmatrix} 10^2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 5 \cdot 10^{-2} \end{bmatrix}, \quad Q_{EKF} = \begin{bmatrix} 10^2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (4.35)$$

$$R_{MHE} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R_{EKF} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

³One of the elements in the matrix is very high/low compared to the other elements.

As seen in the Fig. 4.9, the EKF has a slower response to estimate the first change in the u_{PA001} than the NMHE. However, the response of the second and third change is almost as fast as the NMHE. This can occur due to the covariance matrix P requiring more iterations to converge. The estimation error means in the Case 3 are:

NMHE:

Estimation error mean h_1 : $3.95e-07$

Estimation error mean h_2 : $4.29e-08$

Estimation error mean u_{PA001} : $2.74e-03$

EKF:

Estimation error mean h_1 : $8.50e-07$

Estimation error mean h_2 : $1.69e-09$

Estimation error mean u_{PA001} : $7.70e-03$

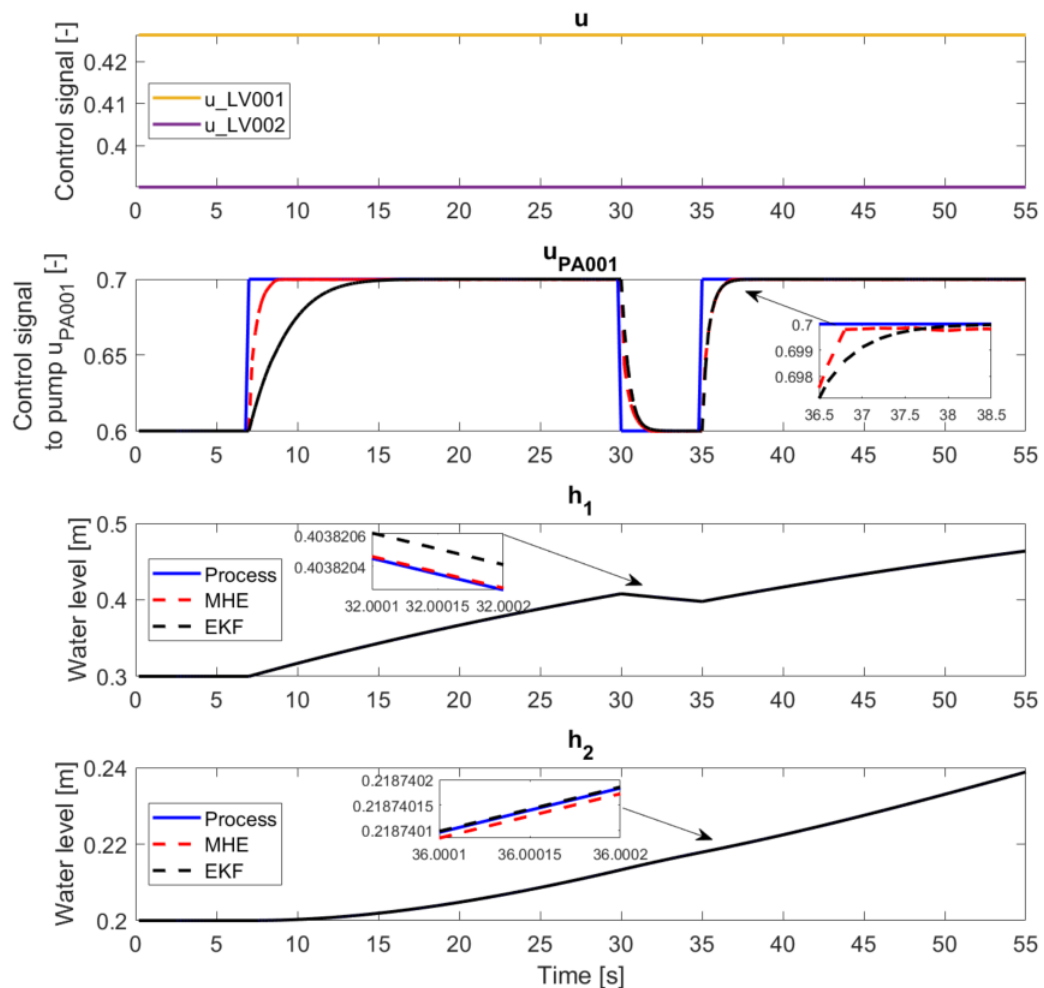


Figure 4.9: NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . No process or measurement noise.

After trying many different combinations of the covariance matrices for the Case 4, the estimation of u_{PA001} by the EKF could not be improved. The results are shown in Fig. 4.10. Many experiments resulted in singular covariance matrix, P , or estimates were unstable (oscillating).

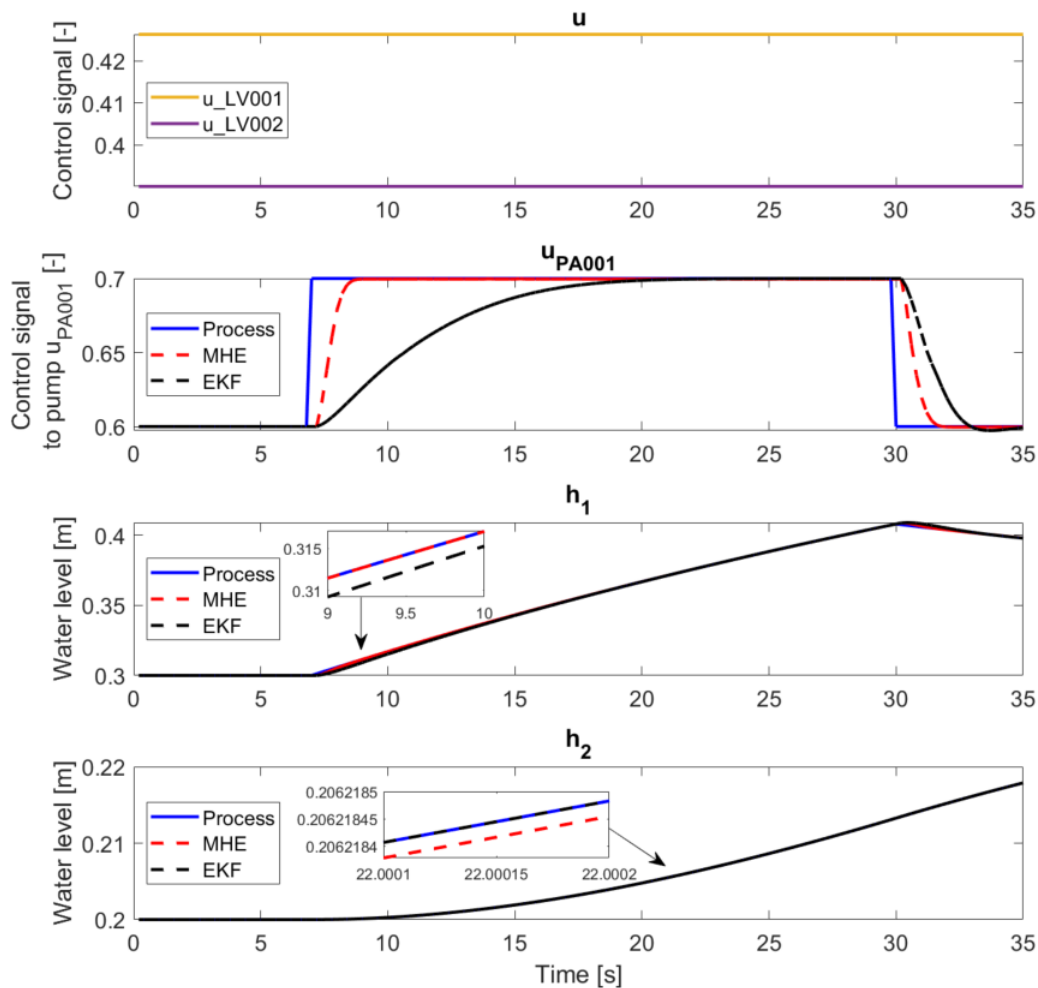


Figure 4.10: NMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . No process or measurement noise.

The NMHE and EKF requires 2 samples to detect the change in u_{PA001} and h_1 , but in general, the overall response is good. The covariance matrices used for the Case 4 are:

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^{-5} & 0 & 0 \\ 0 & 10^{-10} & 0 \\ 0 & 0 & 10^{-8} \end{bmatrix}, & Q_{EKF} &= \begin{bmatrix} 10^3 & 0 & 0 \\ 0 & 10^{-12} & 0 \\ 0 & 0 & 4 \end{bmatrix} \\
 R_{MHE} &= 10^{-10}, & R_{EKF} &= 10^{-12},
 \end{aligned} \tag{4.36}$$

and the resulting estimation error means are:

NMHE:

Estimation error mean h1: 7.33e-05

Estimation error mean h2: 4.67e-08

Estimation error mean u_PA001: 4.97e-03

EKF:

Estimation error mean h1: 4.54e-04

Estimation error mean h2: 4.876e-17

Estimation error mean u_PA001: 1.68e-02

4.3 Simulink Implementation

The real two-tank plant is operated based on a Simulink interface. Thus, in order to test the moving horizon estimation in real time requires a Simulink model. The MHE implementation is built on the Simulink model, *totank1_motivasjon.slx*, provided in the course ELE320 Reguleringssteknikk [2]. The model includes the required interface building blocks to interact with the plant. The control inputs to the valves and the pump can be controlled manually by the user. The upgraded Simulink model is illustrated in Fig. 4.11 and Fig. 4.12. The implementation of MHE is done using the Interpreted

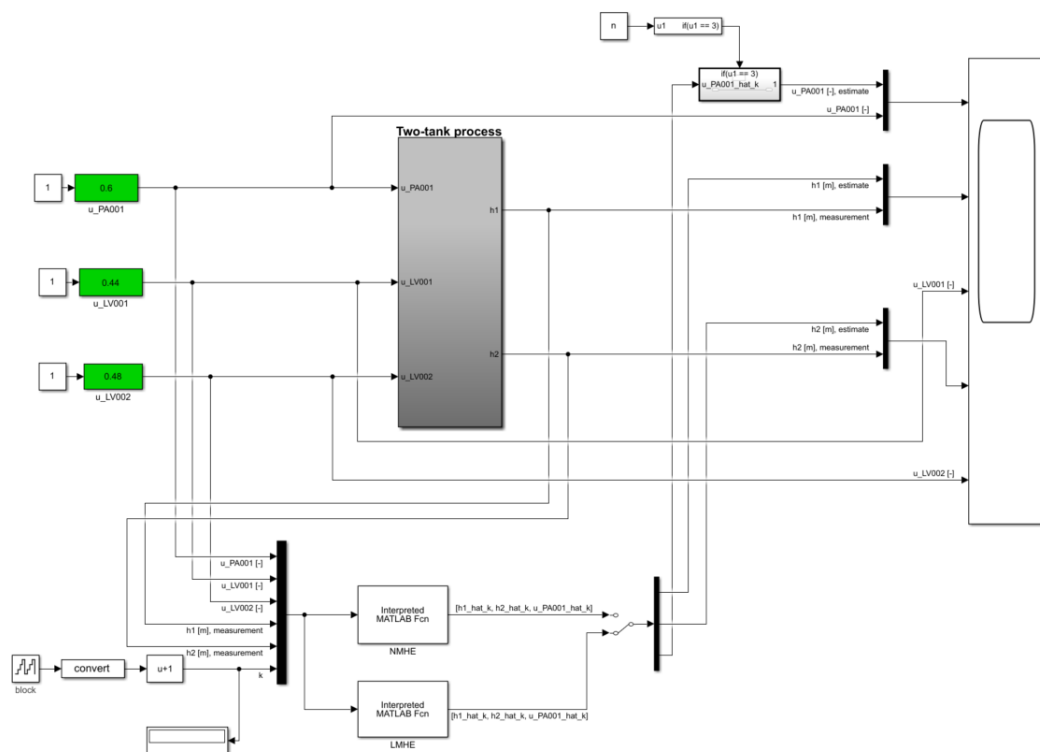


Figure 4.11: Simulink model for the two-tank system with LMHE and NMHE.

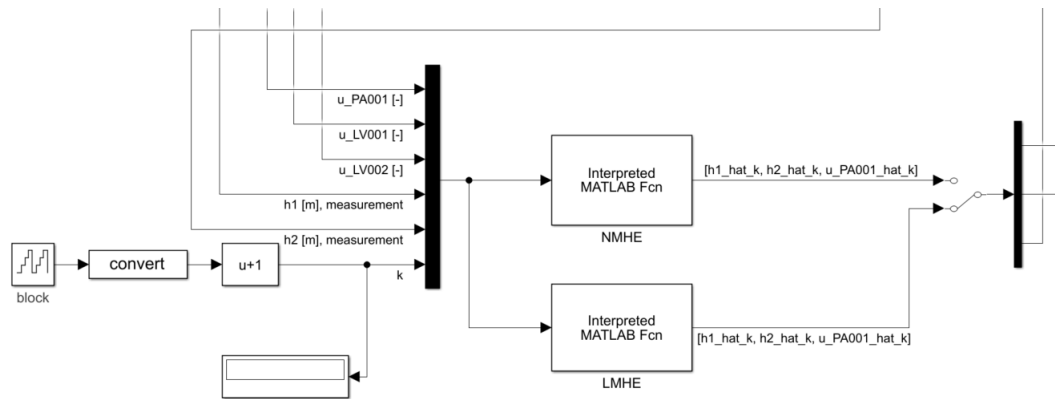


Figure 4.12: Simulink implementation of LMHE and NMHE.

Matlab Function building block. This building block applies a user-defined Matlab function to the input for evaluation. One limitation of the Interpreted Matlab Function block is its incompatibility with multivariable signals. As a result, it can only process 1D vectors as inputs and return 1D vectors as outputs. Therefore, LMHE and NMHE algorithms, presented in the previous sections, are slightly modified and converted to the Matlab functions, *LMHE_simulink.m* and *NMHE_simulink.m*. The input signals, such as the measurements, and the control inputs are stacked to 1D vector using a Mux block, analogously, the signal unpacking is done by Demux block. The MHE functions requires a time index as one of the inputs. The time index is used to initialize MHE, i.e., to estimate the first horizon, and to store/retrieve the arrival cost covariance matrix. The time is counted using the Free-Running Counter block.

Prior to the simulation, the simulink model has to be initialized with the respective init file, namely, *LMHE_Simulink_init.m* or *NMHE_Simulink_init.m*. Initialized horizons and other parameters are, therefore, stored in the base workspace. However, the simulink variables, the measurements, control inputs and the MHE function outputs, are stored in the model workspace. In addition to the input variables from Simulink model the LMHE/NMHE function requires the horizons of the previous measurement and control input samples to calculate the current state estimates. The Matlab functions are using local/temporary workspace to perform calculations, thus, the variables in the base workspace are not directly accessible by the function. This is solved with *evalin()* and *assignin()* functions.

```
function [output] = LMHE_simulink(input)
...
parameter_struct = evalin('base','parameter_struct');
h1_meas_horizon = evalin('base','h1_meas_horizon');
h2_meas_horizon = evalin('base','h2_meas_horizon');
h1_hat_horizon = evalin('base','h1_hat_horizon');
h2_hat_horizon = evalin('base','h2_hat_horizon');
u_PA001_hat_horizon = evalin('base','u_PA001_hat_horizon');
x_hat_horizon = evalin('base','x_hat_horizon');
P_k_N_arr = evalin('base','P_k_N_arr');
P_previous = evalin('base','P_previous');
x_pred_k_N = evalin('base','x_pred_k_N');
u_horizon = evalin('base','u_horizon');
...
assignin('base','h1_hat_horizon',h1_hat_horizon);
assignin('base','h2_hat_horizon',h2_hat_horizon);
assignin('base','u_PA001_hat_horizon',u_PA001_hat_horizon);
assignin('base','h1_meas_horizon',h1_meas_horizon);
assignin('base','h2_meas_horizon',h2_meas_horizon);
assignin('base','x_hat_horizon',x_hat_horizon);
assignin('base','u_horizon',u_horizon);
assignin('base','P_previous',P_previous);
assignin('base','P_k_N_arr',P_k_N_arr);
assignin('base','x_pred_k_N',x_pred_k_N);
```

The *evalin()* is used to retrieve the parameters and current horizons from the base workspace, while *assignin()* is used to update the horizons for the next time step.

Complete overview of Matlab code is available in [Appendix C](#).

Chapter 5

Results

Chapter 4 presented the practical implementation and verification of the LMHE and NMHE estimators. In this chapter, Section 5.1 analyze and discuss the LMHE and NMHE results in different scenarios obtained in the simulation environment whereas Section 5.2 provides evaluation of the estimation results using real two-tank data. Section 5.3 presents experimental results of combining the Model Predictive Control and Moving Horizon Estimation.

5.1 Simulation Environment

To explore the performance of the estimators, the two-tank system is simulated with measurement noise. The noise is generated as normally distributed random numbers with variance $\sigma^2 = 10^{-9}$ and $\sigma^2 = 10^{-6}$. The same test is carried out for all four different cases. The search process of the model error covariance matrices in different scenarios is based on the trial-and-error procedure presented in the previous section. The performances of the LMHE and NMHE are compared with the KF and EKF respectively.

5.1.1 LMHE

Fig. 5.1 illustrates the Case 3, the estimation of h_1 , h_2 and u_{PA001} , given noisy measurements of h_1 and h_2 . The optimization for the LMHE is solved using default *sqp* algorithm and the horizon length $N = 10$. The covariance matrices for the LMHE and

KF are set to:

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^{-9} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^{-5} \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^4 \end{bmatrix}, \\
 R_{MHE} &= R_{KF} = \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix}.
 \end{aligned} \tag{5.1}$$

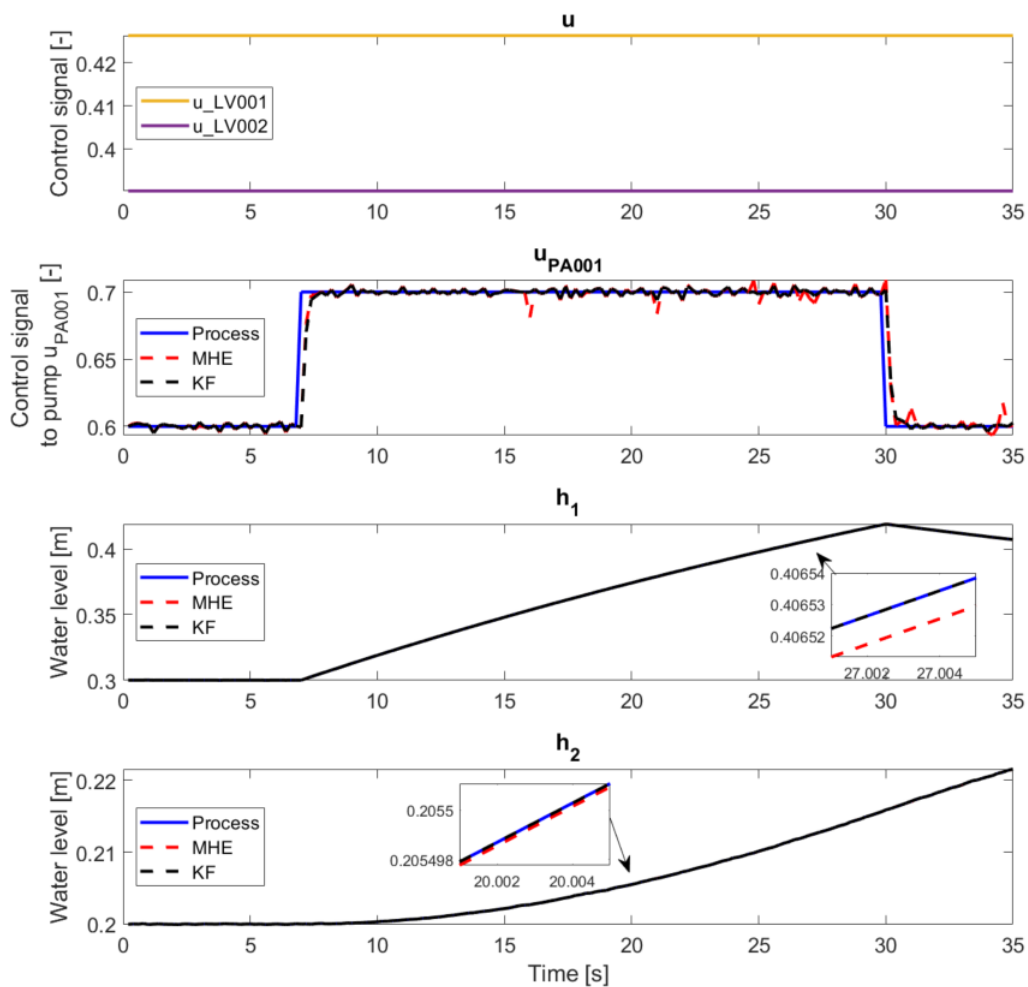


Figure 5.1: LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$.

The results are quite good, however, the performance of the KF is better. The estimates are more smooth and accurate. The resulted estimation error means are:

```

LMHE:
Estimation error mean h1: 2.96e-06
Estimation error mean h2: 6.01e-07
Estimation error mean u_PA001: 3.94e-03
KF:
Estimation error mean h1: 2.19e-14
Estimation error mean h2: 3.49e-14
Estimation error mean u_PA001: 3.30e-03

```

By adjusting¹ the *fmincon* parameters, *OptimalityTolerance* and *StepTolerance*, the spikes seen in u_{PA001} estimate, Fig. 5.1, can be reduced but not eliminated, thus there is no significant improvement when it comes to reducing the estimation errors. In general, setting small tolerances does not always result in more accurate results. Instead, the *fmincon* can fail to recognize when it has converged [25].

Nevertheless, the LMHE performance is improved by solving the optimization problem with *interior-point* algorithm instead of *sqp*. The estimate of u_{PA001} is more smooth and estimation errors are reduced:

```

LMHE:
Estimation error mean h1: 2.20e-06
Estimation error mean h2: 1.06e-08
Estimation error mean u_PA001: 3.38e-03

```

The same improvement is also achieved by reducing the horizon length to $N = 5$. Theoretically, the larger N , the more accurate estimation can be expected. However, the larger number of variables, can lead to a point where the optimization problem becomes infeasible or hard to solve. In this case, for the horizon length $N = 10$ and three state variables, the optimization problem is minimized subject to 30 variables. The additional Case 3 results can be found in Appendix B, Section B.2.1.

The Case 4 results are suboptimal compared to the Case 3, these are presented in Fig. 5.2. The optimization problem of LMHE is solved using *sqp* algorithm, the horizon length $N = 10$, and tolerance parameters set to 10^{-10} . The best possible outcome is given by the following covariance matrices:

¹For both parameters the default value 10^{-6} is set to 10^{-10} .

$$Q_{MHE} = \begin{bmatrix} 10^{-2} & 0 & 0 \\ 0 & 10^{-8} & 0 \\ 0 & 0 & 10^{-2} \end{bmatrix}, \quad Q_{KF} = \begin{bmatrix} 10^{-7} & 0 & 0 \\ 0 & 10^{-9} & 0 \\ 0 & 0 & 10^{-9} \end{bmatrix}, \quad (5.2)$$

$$R_{MHE} = R_{KF} = \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix}.$$

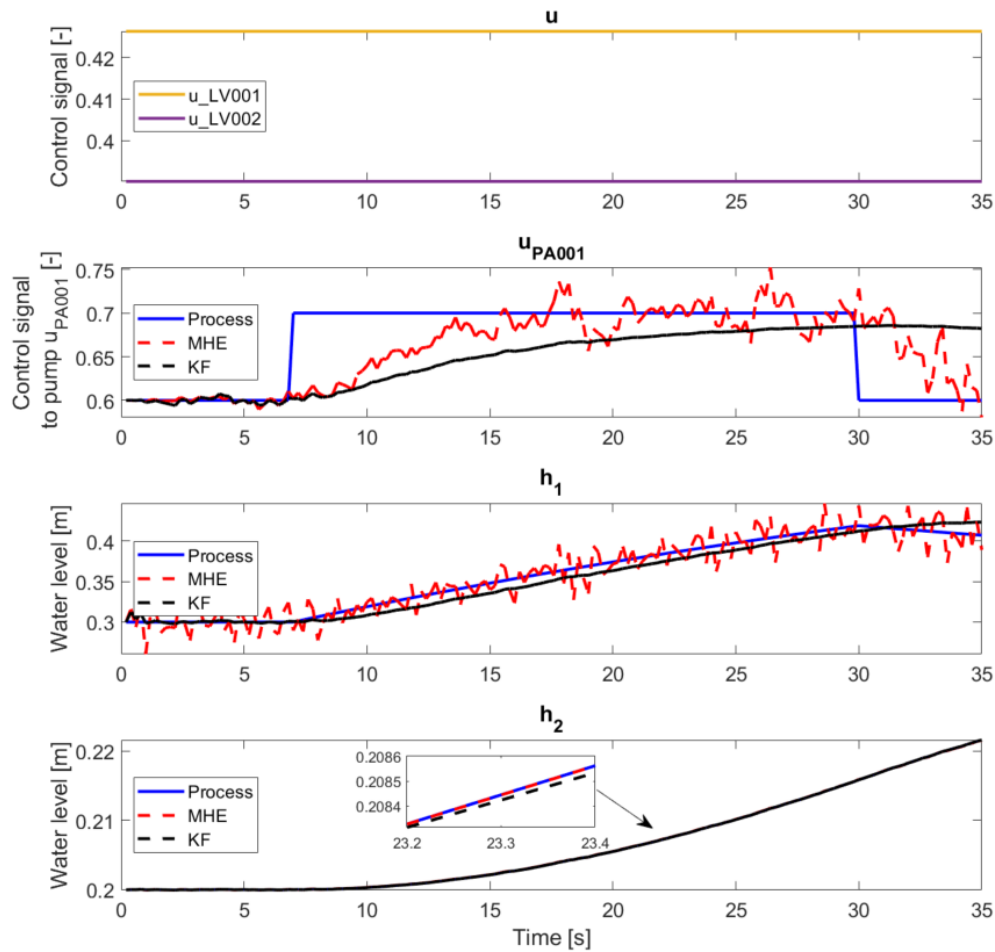


Figure 5.2: LMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$.

LMHE:

Estimation error mean h1: 1.26e-02

Estimation error mean h2: 7.18e-07

Estimation error mean u_PA001: 2.75e-02

KF:

Estimation error mean h1: 7.78e-03

Estimation error mean h2: 1.45e-05

Estimation error mean u_PA001: 4.20e-02

As seen in the figure, the LMHE estimate of u_{PA001} is noisy. The response times for both estimators are slow, yet the LMHE is faster than the KF. The estimation of h_1 is better but not as good as in the Case 3. The KF has a bias and slow response, this can be observed in the time interval [30s, 35s].

As presented in Section 4.1.4, the performance of both estimators is great when the system is noise free. However, relative small amount of measurement noise has a big impact to the estimates. The fact that both the LMHE and KF performs poorly, indicates that the problem may lie in the two-tank system rather than the estimator. As discussed in Section 4.1.1, the observability of u_{PA001} in the Case 4 is low. In this particular operating point, the eigenvectors and eigenvalues of the observability Grammian are

$$\lambda_1 = 0.13 \cdot 10^{-3}, \quad \lambda_2 = 1, \quad \lambda_3 = 0.5 \cdot 10^{-6},$$

$$e_1 = \begin{bmatrix} -0.99 \\ -0.1 \cdot 10^{-3} \\ 0.19 \cdot 10^{-2} \end{bmatrix}, \quad e_2 = \begin{bmatrix} 0.10 \cdot 10^{-3} \\ -0.99 \cdot 10^{-3} \\ -0.61 \cdot 10^{-7} \end{bmatrix}, \quad e_3 = \begin{bmatrix} 0.19 \cdot 10^{-2} \\ 0.13 \cdot 10^{-6} \\ 0.99 \end{bmatrix}, \quad (5.3)$$

where λ_3 corresponds to u_{PA001} . Low observability means that information or characteristics of the state is not fully captured or reflected in the measurements. Hence, increased uncertainty due to the measurement noise in addition to low observability of the state may result in a poor estimate.

However, it has been observed that lower sampling rate results in less noisy estimates. This can be due to the fact that the change in the state when sampled less frequently is bigger. The change in the state variable compared to the measurement noise is more "obvious" for the optimizer resulting in more accurate estimates. The example of the same simulation presented in the Fig. 5.2 where the sampling time is increased from 0.2s to 0.5s is shown in Fig. 5.3. The response time have not changed, but the LMHE estimates are less noisy.

The results for the LMHE Case 1 and Case 2 simulations are available in Appendix B, Section B.2.1.

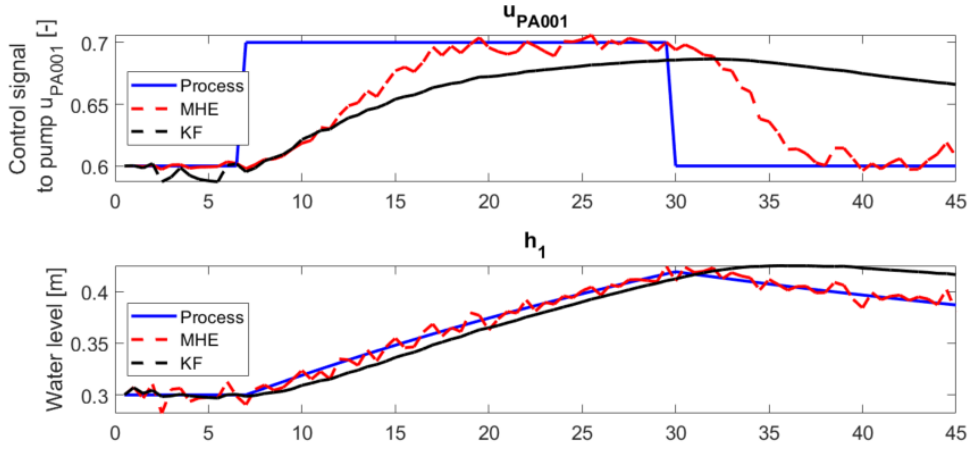


Figure 5.3: LMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. Sample time increased to $0.5s$.

5.1.2 NMHE

The NMHE and EKF results for the Case 3 are presented in Fig. 5.4. The optimization problem of NMHE is solved with *sqp* algorithm and the horizon length $N = 10$. The covariance matrices used are the following:

$$Q_{MHE} = \begin{bmatrix} 10^{-2} & 0 & 0 \\ 0 & 10^{-2} & 0 \\ 0 & 0 & 10^{-4} \end{bmatrix}, \quad Q_{EKF} = \begin{bmatrix} 6 \cdot 10^1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (5.4)$$

$$R_{MHE} = R_{EKF} = \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix}.$$

NMHE:

Estimation error mean h1: 4.80e-07
 Estimation error mean h2: 5.64e-07
 Estimation error mean u_PA001: 1.08e-02

EKF:

Estimation error mean h1: 1.55e-15
 Estimation error mean h2: 3.48e-14
 Estimation error mean u_PA001: 5.68e-03

The results of the NMHE estimates, h_1 and h_2 , are good but the estimate of u_{PA001} is noisy. The EKF estimate, on the other hand, is more accurate but has a slower response time. Therefore, additional experiments were conducted to investigate if the NMHE performance can be improved. Instead of the *sqp* algorithm the optimization problem is tried to be solved using *interior-point* algorithm. However, the *interior-point* fails in

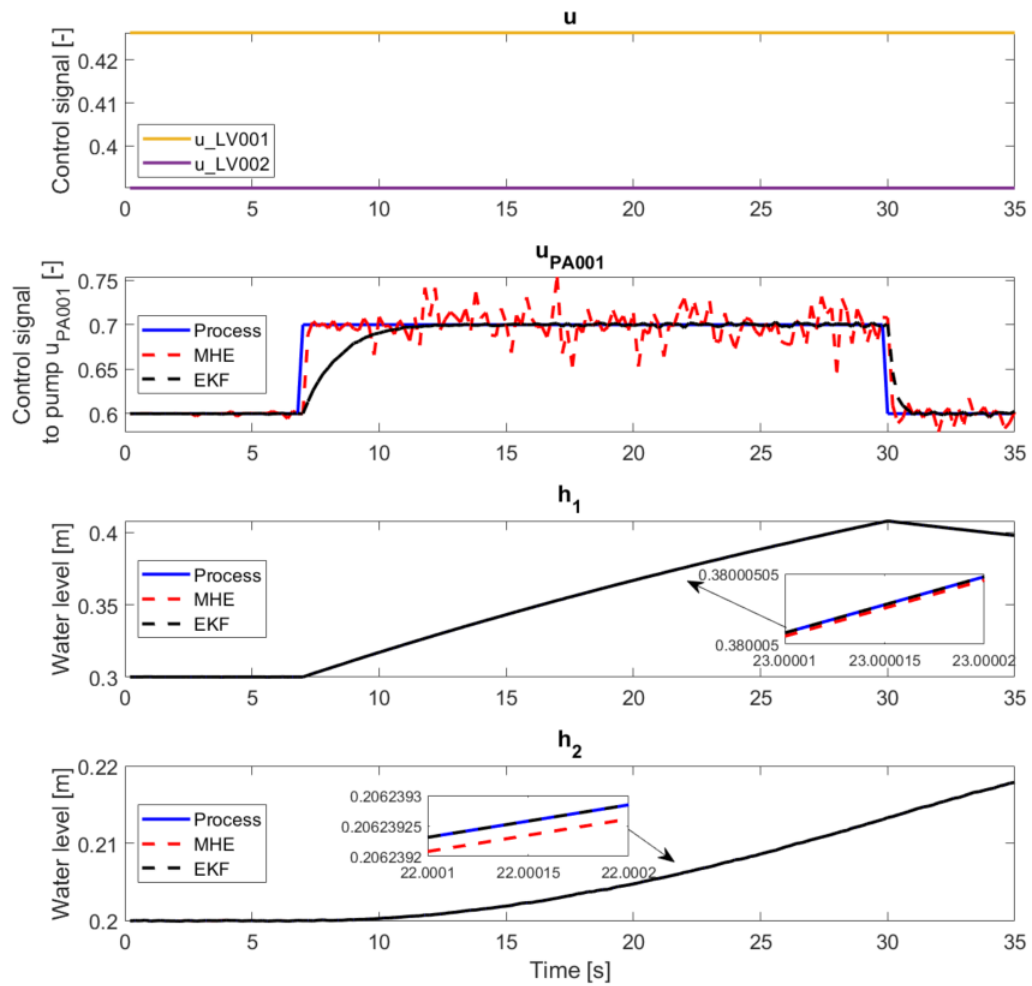


Figure 5.4: NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$.

estimating u_{PA001} , which is estimated as being constant during the whole simulation time. The *active-set* algorithm, on the other hand, behaves better than *sqp*. The resulting estimation error mean is reduced and u_{PA001} estimate has smaller peaks. These results can be found in Appendix B, Fig. B.10. In addition, as for the LMHE, the estimator performance is improved by reducing the horizon length N . Despite the fact that the above mentioned approaches gives better results than presented in the Fig. 5.4, the estimation of u_{PA001} is noisy.

The NMHE is a nonlinear estimator, unlike the LMHE or EKF. Thus, the objective function that is to be minimized is nonlinear. The oscillation seen in u_{PA001} estimate can be due to the fact that the optimization problem when introducing measurement noise is nonconvex. The *fmincon* function is a local minimizer, therefore, for the optimization problems with possibly multiple minima, *fmincon* may fail to find global minima.

During the testing it has also been observed that, even though, estimation performance of the EKF is good, i.e., the weighting matrices are correctly chosen, suddenly the EKF can fail. This happens when the system reach some specific operating point where the error of linear approximation is big enough to result in sudden estimation performance change and the covariance matrix, P , becoming singular. Fig. 5.5 shows an example where the EKF suddenly fails. This is the same simulation as in Figure 5.4, but the first element in Q_{EKF} is 10 instead of 60. The response time of the u_{PA001} estimate is much faster than presented in the Fig. 5.4 but the estimator becomes unstable and fails at time 25s.

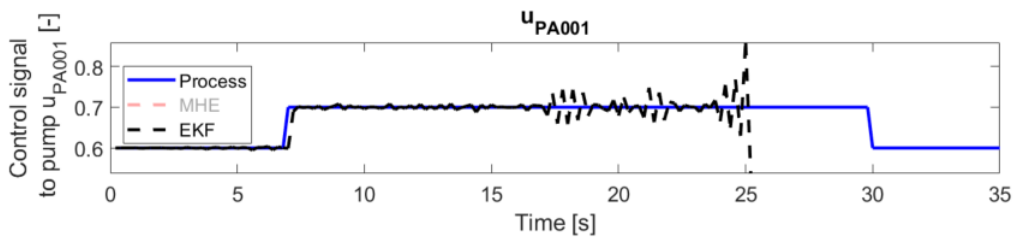


Figure 5.5: EKF Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$.

Global Optimization

Matlab provides several global optimization based solvers such as *MultiStart*, *GlobalSearch*, *PatternSearch*, *ga* and *simulannealbnd*.

The *MultiStart* and *GlobalSearch* are gradient-based global optimization solvers. Both methods employ a local solver, such as *fmincon*, to find a local minimum for given starting points. The *MultiStart*, in addition to user-supplied starting point, generates a set of uniformly distributed starting points within the bounds. These are evaluated by a local solver. The algorithm returns the best value as a global minimum. The *GlobalSearch* method is more complicated. The starting points are generated using a scatter-search mechanism. The algorithm tries to determine in advance if the trial points will result in improvement. These are evaluated using score functions, thus only the starting points with highest score are passed to a local solver [26].

simulannealbnd performs a random search. Simulated annealing is a stochastic optimization method inspired by annealing process, i.e., a heat treatment process used to

alter the properties of a material by heating it to a specific temperature and then slowly cooling it [26].

Ga, genetic algorithms, are inspired by the process of natural selection. The objective function is interpreted as a measure of fitness and each point as the genes of an individual. At each iteration, randomly selected individuals, called parents, are used to create a new generation of individuals, called children. Thus, the method generates better points iteratively.

PatternSearch method, also called direct search algorithm, polls a number of neighboring points by computing their objective function values and evaluating them to determine which direction to search for even lower objective function values. *PatternSearch* algorithm is not using derivatives, thus it is very useful if the objective function is not differentiable.

Fig. 5.6 illustrates the Case 3 u_{PA001} estimation results obtained by *fmincon* using *sqp* algorithm (as in 5.4) and global optimization methods: *Multistart*, *GlobalSearch* and *PatternSearch*. *Multistart* and *GlobalSearch* uses *fmincon* (*sqp* algorithm) as a local solver. The number of starting points to *MultiStart* is set to 5.

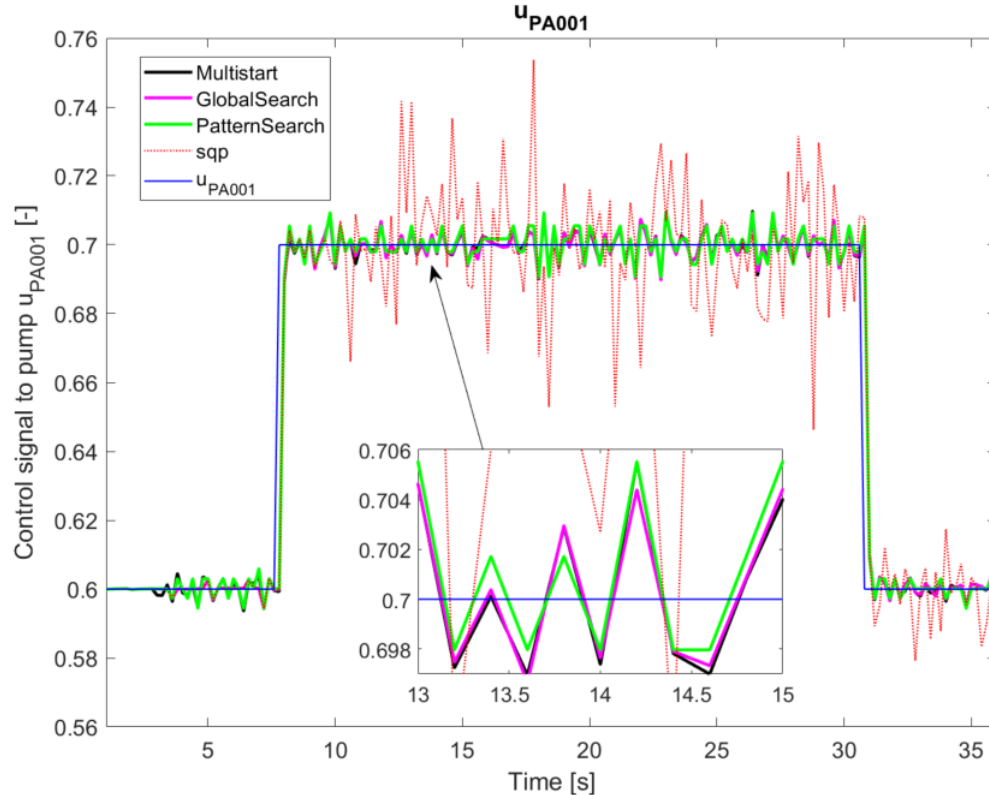


Figure 5.6: NMHE Case 3: u_{PA001} estimation using global optimization methods. Simulated measurement noise, $\sigma^2 = 10^{-9}$.

The improvement by using global optimization solvers is obvious. All three global optimization methods have very similar results, indicating that resulting estimates are globally optimal solutions. Small deviations may be caused by different tolerances within the algorithms. As discussed in Section 3.2, the accuracy of the global optimization techniques comes with a cost, i.e., computational efficiency. The time consumption for different methods is discussed in more detail in Section 5.1.3.

The Case 4 for the NMHE is simulated with increased sampling time, i.e., $0.5s$ instead of $0.2s$. The results are presented in Fig. 5.7. As mentioned in Section 5.1.1, the estimation improves with higher sampling time. Due to the measurement noise, low observability of u_{PA001} and nonconvex objective function the NMHE fails when sampling time is $0.2s$.

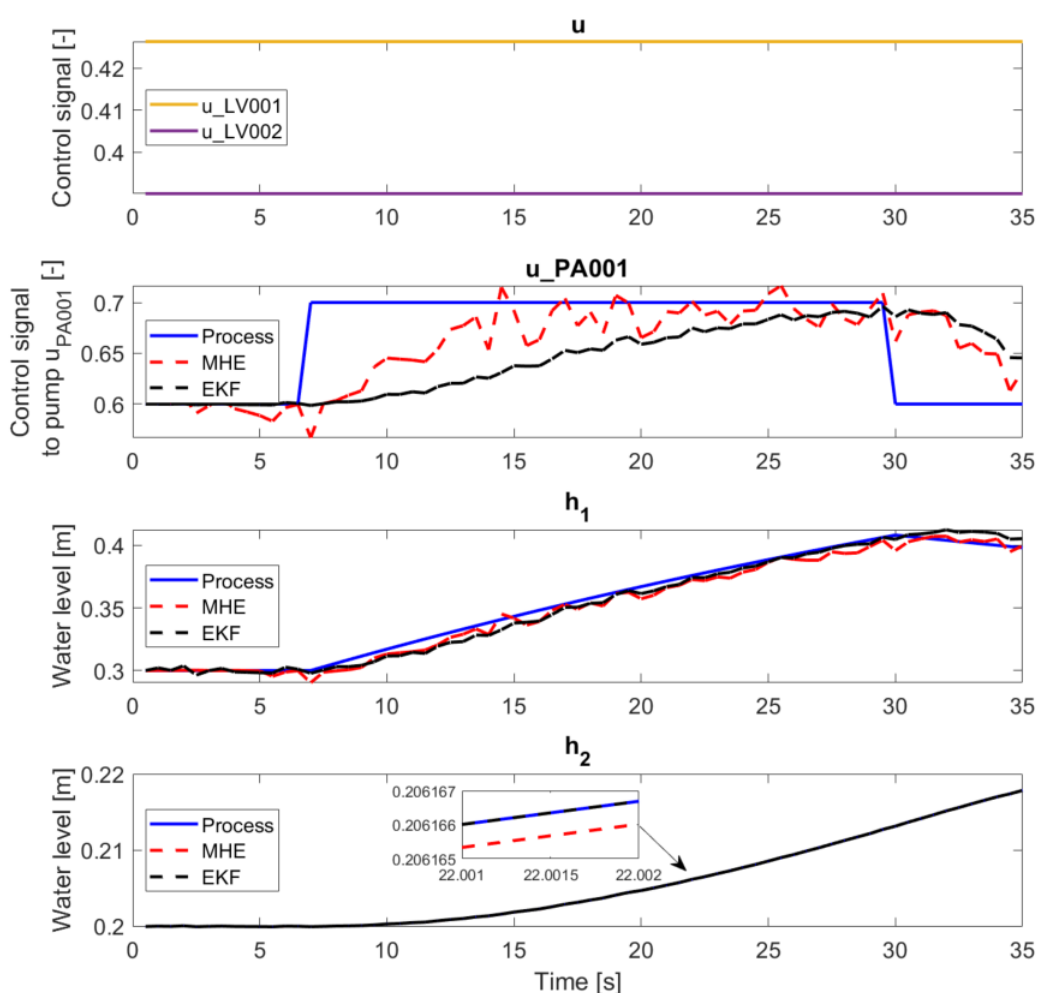


Figure 5.7: NMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. Sample time increased to $0.5s$.

The covariance matrices for the NMHE and EKF are set to:

$$Q_{MHE} = \begin{bmatrix} 10^{-6} & 0 & 0 \\ 0 & 2 \cdot 10^{-7} & 0 \\ 0 & 0 & 10^{-10} \end{bmatrix}, \quad Q_{EKF} = \begin{bmatrix} 10^4 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^1 \end{bmatrix}, \quad (5.5)$$

$$R_{MHE} = R_{KF} = 10^{-9}.$$

The estimation error means are very similar to the LMHE and KF:

```

NMHE:
Estimation error mean h1: 4.27e-03
Estimation error mean h2: 3.24e-07
Estimation error mean u_PA001: 3.07e-02
EKF:
Estimation error mean h1: 4.37e-03
Estimation error mean h2: 1.25e-14
Estimation error mean u_PA001: 4.50e-02

```

The results for the NMHE Case 1 and Case 2 simulations are available in Appendix B, Section [B.2.2](#).

5.1.3 MHE Efficiency

In general, the MHE is a computationally expensive and time-consuming estimation technique, especially for large-scale systems. This is due to the computational complexity of solving the optimization problem. The time required to perform the estimation can be critical in many applications, particularly in real-time systems. The MHE is often considered in systems where the computational time is negligible compared to the sample time. However, all systems are limited to critical minimum sampling time that is determined by the system dynamics and desirable closed-loop stability. Too low sampling frequency may result in poor system performance and even instability. Therefore, some testing is done to investigate the efficiency of the estimators. The results of time usage are reported in Tables [5.1](#) and [5.2](#). Figures [5.8](#) and [5.9](#) illustrate how different algorithms and different choices of horizon lengths impact the LMHE accuracy.

The process time may vary enormously depending on what hardware is used. Since *fmincon* provides parallel pooling for faster and more efficient processing, the tests have been conducted on two different PC's for comparison:

- **PC-1:** HP Pavilion Notebook, Intel(R) Core(TM) i5-6200U, 2.4GHz, 8GB RAM, 2 cores.
- **PC-2:** Dell XPS 15 9500, Intel(R) Core(TM) i7-10750H, 2.6GHz, 16GB RAM, 6 cores.

The PC-1 is used in most of the simulation because it has similar hardware to the PC utilized in the two-tank system.

LMHE Case 3 with measurement noise, $N = 10$	Average internal alg. iterations	Average time usage per sample, [s]	Max. time usage per sample, [s]
<i>interior-point</i> algorithm, PC-1	83.0	1.18	3.39
<i>active-set</i> algorithm, PC-1	60.0	1.79	3.49
<i>sqp</i> algorithm, PC-1	65.5	0.74	2.46
<i>sqp</i> algorithm, PC-2	65.5	0.34	2.06
<i>sqp</i> algorithm, reduced tolerances, PC-1	65.7	0.87	2.25
<i>sqp</i> algorithm, parallel pooling, PC-1	65.5	3.77	8.86
<i>sqp</i> algorithm, parallel pooling, PC-2	65.5	2.46	3.45
KF, PC-1	1	$9.8 \cdot 10^{-5}$	$3.9 \cdot 10^{-3}$
<i>sqp</i> algorithm, $N = 5$, PC-1	45.8	0.20	1.74
<i>sqp</i> algorithm, $N = 10$, PC-1	65.5	0.74	2.46
<i>sqp</i> algorithm, $N = 15$, PC-1	72.6	1.44	3.06
<i>sqp</i> algorithm, $N = 20$, PC-1	74.9	2.64	7.31

Table 5.1: Time usage of the LMHE using different algorithms, parameters and horizon lengths.

NMHE Case 3 with measurement noise, $N = 10$	Average time usage per sample, [s]	Max. time usage per sample, [s]
<i>sqp</i> algorithm	1.65	3.54
<i>MultiStart</i> method using <i>sqp</i> algorithm	10.8	28.2
<i>MultiStart</i> method using <i>sqp</i> algorithm, parallel pooling	9.69	63.2
<i>GlobalSearch</i> method using <i>sqp</i> algorithm	6.0	9.6
<i>PatternSearch</i> method	3.30	8.61
EKF	$6.9 \cdot 10^{-4}$	$1.9 \cdot 10^{-2}$

Table 5.2: PC-1 time usage of the NMHE using different methods.

Based on the results presented in the Table 5.1 and Figures 5.8 and 5.9, the *Active-set* algorithm provides the highest accuracy but is the most time-consuming, whereas the *sqp* algorithm is the most efficient. When it comes to horizon length, considering the process time versus the accuracy it seems that the horizon length between $N = 5$ and $N = 10$ is the best choice. It is also worth noting that the process time of NMHE using *sqp* algorithm is two times higher than LMHE.

The global optimization methods introduced in the previous section are more accurate than *fmincon*, as seen in Fig. 5.6, yet at the cost of processing time. Compared with *fmincon* the global optimization methods are approximately 6 – 10 times slower. The exception is the *PatternSearch* method that is approximately three times slower.

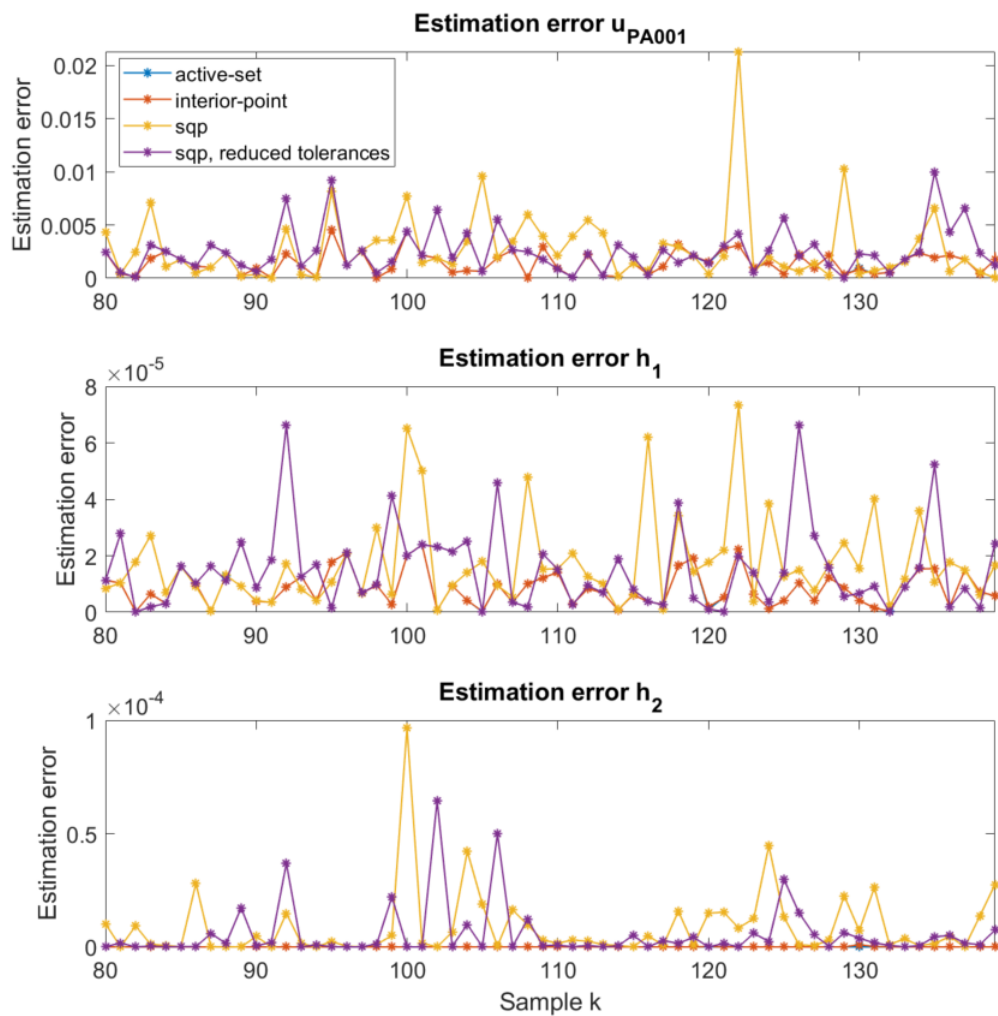


Figure 5.8: Absolute values of the estimation error variations using *sqp*, *sqp* with reduced tolerances, *active-set* and *interior-point* algorithms. The *active-set* performance is equal to *interior-point*, therefore invisible.

The results of parallel pooling are unexpected. The processing time using parallel pooling is extremely higher than serial processing. Doing some research on the topic it turns out, that this is often the case, mainly for two reasons [27]:

- Communications overhead - the data transfer to the separate processes for each worker and back to the client. Hence, if the amount of work performed during each iteration of the parallel loop is insufficient, the overhead of communication outweighs any benefits gained from parallel processing.
- When dealing with sufficiently large matrices Matlab utilizes high-performance multi-threaded libraries that utilize all available cores. However, when running parallel pooling, each worker is allocated only one core. As a result, although the same multi-threaded libraries are called, all the computations are performed sequentially, and thus, incurring the overhead of setting up the library calls.

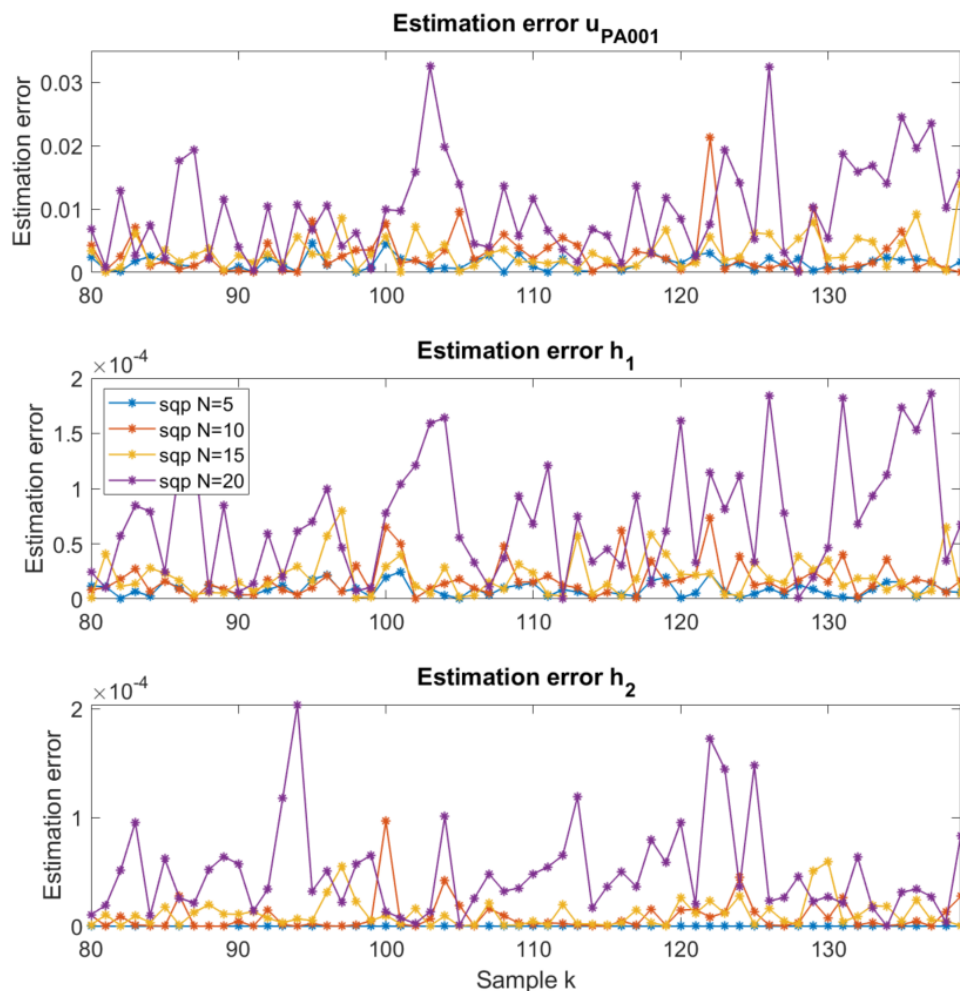


Figure 5.9: Absolute values of the estimation error variations for the horizon length $N = 5$, $N = 10$, $N = 15$ and $N = 20$.

Sampling Time for the Two-tank System

A general rule of thumb when choosing the sampling time for a system, is to set the sampling time to be at least 10 times faster than the fastest dynamics of the system. This ensures that the sampling rate is sufficiently fast to capture the dynamics of the system. The easiest way to determine the time constants to the two-tank system is by determining the eigenvalues of the linearized state-space model. The corresponding time constants can be approximated using

$$\tau_i = -\frac{1}{\lambda_i}. \quad (5.6)$$

The other way to determine the time constants is by finding the transfer functions to each tank:

$$\begin{aligned} H_p(s) = \frac{y(s)}{u(s)} &\Rightarrow H_{p,1}(s) = \frac{\Delta h_1(s)}{\Delta u_{LV001}(s)}, & H_{p,2}(s) &= \frac{\Delta h_2(s)}{\Delta u_{LV002}(s)}, \\ H_v(s) = \frac{y(s)}{v(s)} &\Rightarrow H_{v,1}(s) = \frac{\Delta h_1(s)}{\Delta u_{PA001}(s)}, & H_{v,2}(s) &= \frac{\Delta h_2(s)}{\Delta u_{LV001}(s)}. \end{aligned} \quad (5.7)$$

where $H_{p,i}$ is the process transfer function and $H_{v,i}$ is the disturbance transfer function to the tank number i . The approximated Laplace transfer functions can be obtained by bringing the system to the desired operating point and simulating a step at the input. The time constant can be determined by measuring the point in the resulting step response where the output reaches 63% of the total change. The operating point $u_{PA001} = 0.8, h_1 = 0.5, h_2 = 0.2 \Rightarrow u_{LV001} = 0.531, u_{LV002} = 0.568$ gives the following time constants [s]:

$$\begin{aligned} \tau_{p,1} &= 43.4, & \tau_{v,1} &= 43.4, \\ \tau_{p,2} &= 65, & \tau_{v,2} &= 52.3. \end{aligned} \quad (5.8)$$

Thus the tank 1 represents the fastest dynamics within the system. According to the rule of thumb the sampling time has to be at most $\approx 4s$. However, the time constants will vary depending on the operating point, hence the sampling time should be chosen accordingly. Based on the MHE time consumption in Case 3, i.e., 3 state variables, the *sqp* algorithm with the horizon length up to $N = 15$ for the LMHE, and $N = 10$ for the NMHE is the best fit.

The simulation results with more realistic measurement noise, $\sigma^2 = 10^{-6}$, are available in Appendix B, Section B.2.3. The measurement noise in the real two-tank system lies

in range of $\sigma^2 = 10^{-6}$, thus the simulation results are very similar to the real data estimation. Therefore, the discussion is covered in Section, 5.2.

5.2 Real Data Estimation

Since the two-tank system is a slow process most of the testing is performed offline. Evaluation of different estimators in different cases is done using two datasets, namely, *data1.mat* and *data2.mat*. In both datasets the two-tank process is running at the same operating point. Dataset 1 has a 0.1 step in u_{PA001} , the valve openings remain constant at all times. Dataset 2, on the other hand, has a 0.15 step in u_{PA001} and multiple steps in the valve openings. Data is sampled every 1.05s. Dataset 1 is used to evaluate the LMHE and KF, and the NMHE and EKF individually while dataset 2 is used to compare linear and nonlinear model based estimators.

The measurement noise covariance matrix R is determined using training data, *data3.mat*. The dataset has 2838 samples and is illustrated in Fig. 5.10.

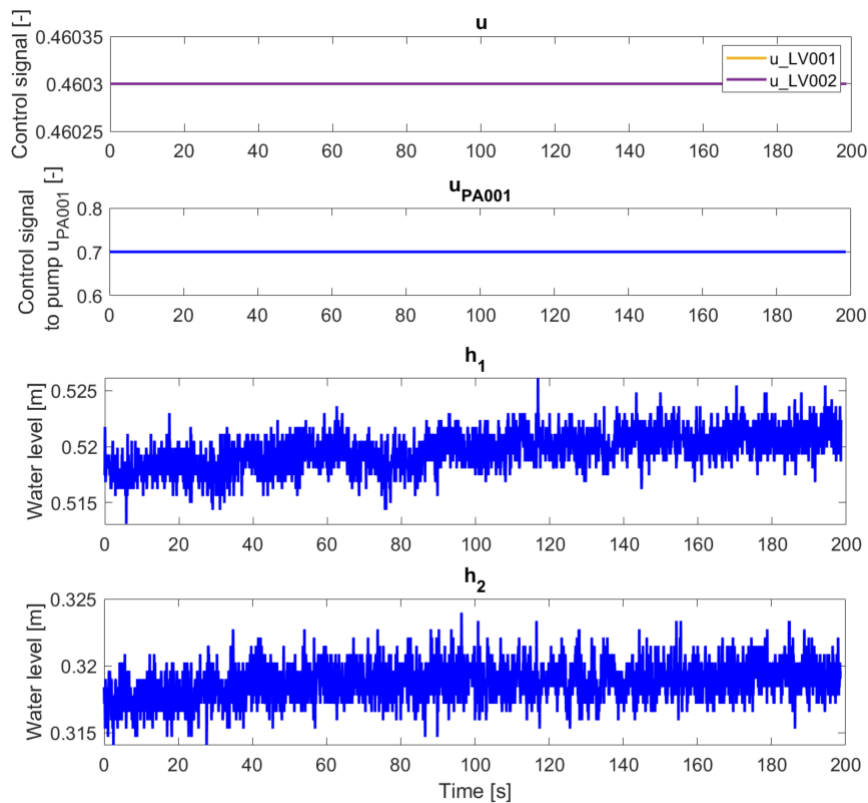


Figure 5.10: Training dataset used to determine measurement noise covariance matrix. Dataset *data3.mat*.

Resulting variances for h_1 and h_2 measurements are:

$$\sigma_{h_1}^2 = 2.8 \cdot 10^{-6}, \quad \sigma_{h_2}^2 = 1.6 \cdot 10^{-6}. \quad (5.9)$$

To determine the model error to the actual two-tank system is quite a challenging task. Hence, real data estimation is done using the covariance matrices, obtained via the trial-and-error procedure in simulation environment.

LMHE

Fig. 5.11 illustrates the Case 2. The water levels, h_1 and h_2 , are estimated using the measurement of h_2 . The optimization problem for LMHE is solved with *sqp* algorithm and the horizon length $N = 10$.

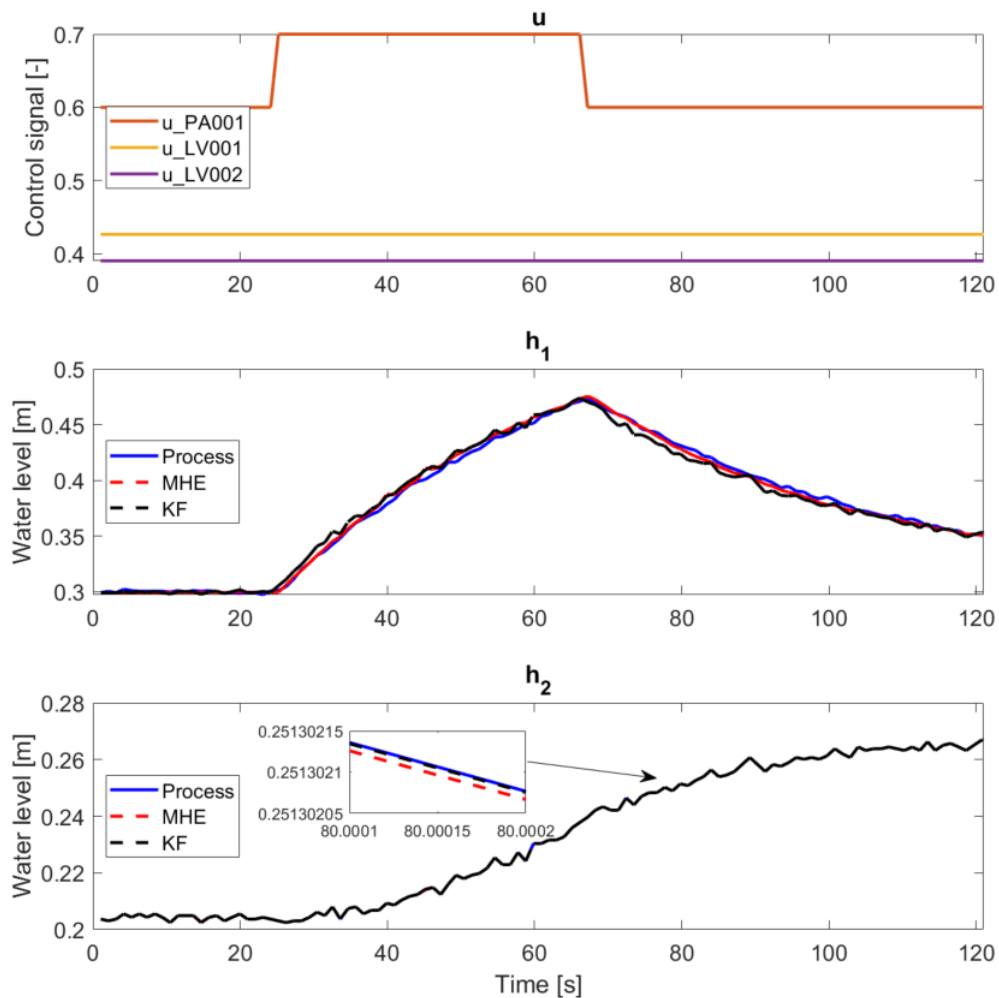


Figure 5.11: LMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Dataset *data1.mat*.

The covariance matrices are set to:

$$Q_{MHE} = \begin{bmatrix} 9 \cdot 10^{-1} & 0 \\ 0 & 1 \end{bmatrix}, \quad Q_{KF} = \begin{bmatrix} 10^1 & 0 \\ 0 & 1 \end{bmatrix}, \quad (5.10)$$

$$R_{MHE} = R_{KF} = 1.6 \cdot 10^{-6}.$$

The LMHE and KF give good results. The LMHE estimate of h_1 is slightly better than the KF, yet the estimate of h_2 is less accurate than the KF:

LMHE:

Estimation error mean h_1 : 2.27e-03

Estimation error mean h_2 : 2.49e-06

KF:

Estimation error mean h_1 : 4.27e-03

Estimation error mean h_2 : 2.32e-09

The Case 3 results for the LMHE and KF are presented in Fig. 5.12.

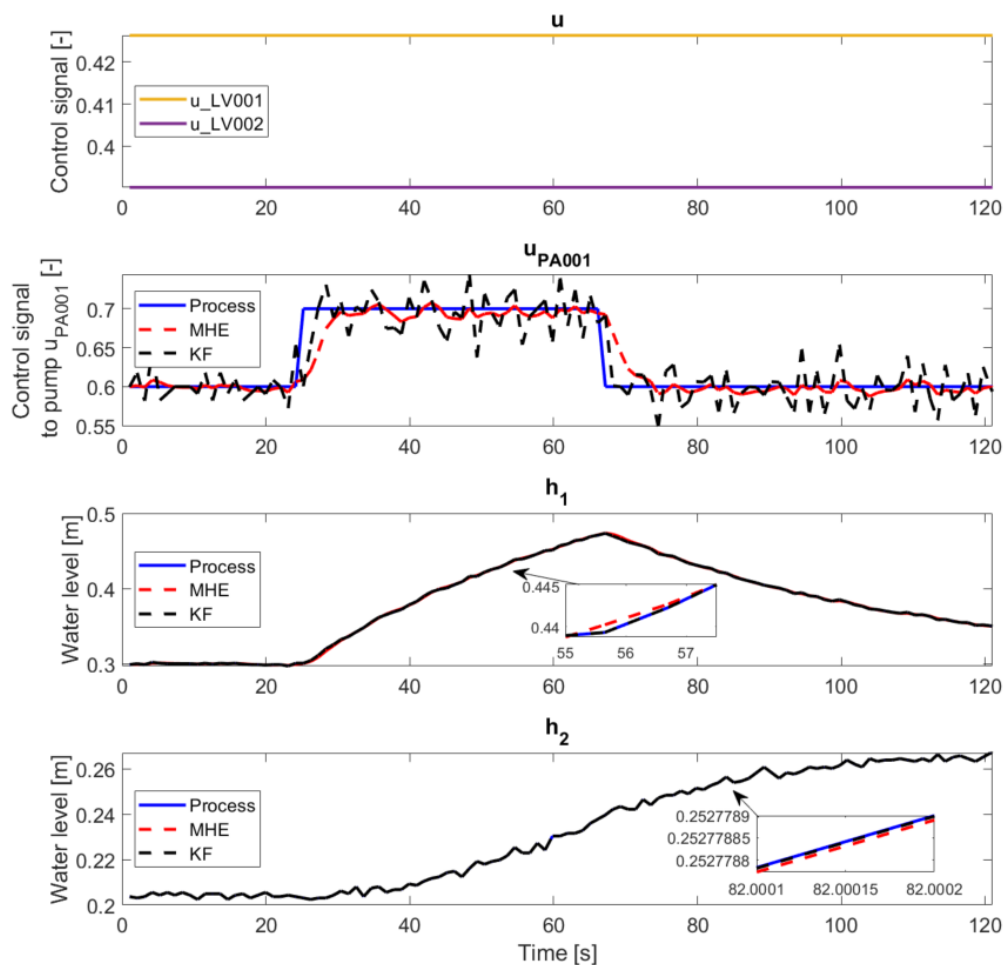


Figure 5.12: LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Dataset *data1.mat*.

The covariance matrices used, are the following:

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^{-6} & 0 & 0 \\ 0 & 10^{-1} & 0 \\ 0 & 0 & 10^{-4} \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^7 \end{bmatrix}, \\
 R_{MHE} = R_{KF} &= \begin{bmatrix} 2.8 \cdot 10^{-6} & 0 \\ 0 & 1.6 \cdot 10^{-6} \end{bmatrix}.
 \end{aligned} \tag{5.11}$$

The choice of covariance matrix Q in the LMHE depends on desired response. Reduced diagonal elements, e.g., $[10^{-7}, 10^{-1}, 10^{-5}]$ results in slower but more accurate u_{PA001} estimate but at the cost of h_1 accuracy. However, increased values give faster response time and more noisy estimate of u_{PA001} where h_1 accuracy is improved. The KF, on the other hand, is less responsive to the tuning. The overall performance is good for both estimators. However, the LMHE is better in estimating u_{PA001} , the accuracy of h_1 and h_2 is a bit lower than the KF, but still very high:

```

LMHE:
Estimation error mean h1: 5.95e-04
Estimation error mean h2: 4.61e-08
Estimation error mean u_PA001: 9.01e-03
KF:
Estimation error mean h1: 5.86e-13
Estimation error mean h2: 2.34e-09
Estimation error mean u_PA001: 2.06e-02

```

Since the measurement noise in the real-two tank system is much higher than presented in the previous section, the Case 4 estimation, Fig. 5.13, is worse compared with the results in the Fig. 5.1. The covariance matrices

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^{-5} & 0 & 0 \\ 0 & 10^{-7} & 0 \\ 0 & 0 & 10^{-4} \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 10^{-4} & 0 & 0 \\ 0 & 10^{-6} & 0 \\ 0 & 0 & 10^{-3} \end{bmatrix}, \\
 R_{MHE} = R_{KF} &= 1.6 \cdot 10^{-6}
 \end{aligned} \tag{5.12}$$

gave the following estimation error means:

```

LMHE:
Estimation error mean h1: 1.43e-02
Estimation error mean h2: 8.37e-04
Estimation error mean u_PA001: 2.69e-02
KF:

```

Estimation error mean h_1 : $1.56e-02$
 Estimation error mean h_2 : $5.09e-04$
 Estimation error mean u_{PA001} : $2.90e-02$

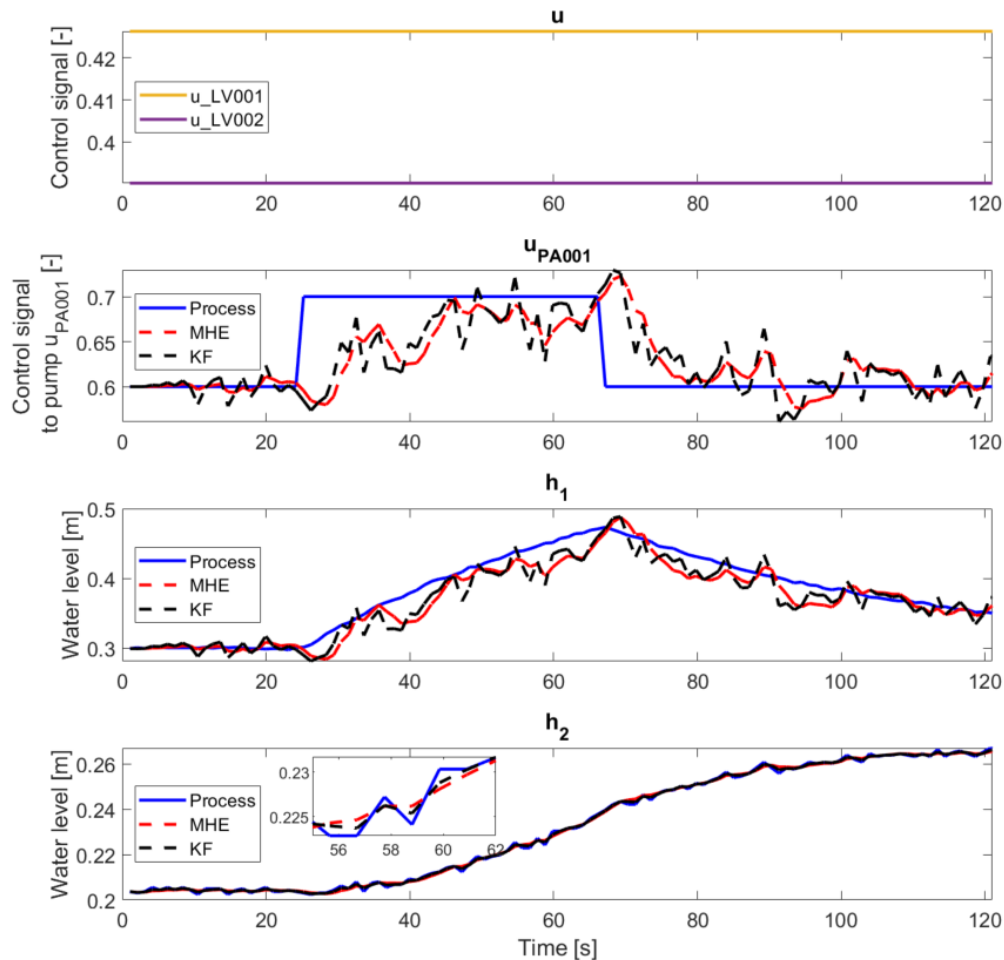


Figure 5.13: LMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . Dataset *data1.mat*.

The resulting estimation of both the LMHE and KF is rough. The LMHE estimates are smoother version of the KF estimates. Both estimators manage to detect the change in u_{PA001} , but the response is slow and noisy.

The results for the Case 1 are available in Appendix B, Section B.3.1. The performance of both estimators is good. The estimation error means for the LMHE and KF are very similar.

NMHE

The results of the NMHE and EKF in the Case 2 are presented in Fig. 5.14. The optimization problem of NMHE is solved with *sqp* algorithm and the horizon length $N = 10$.

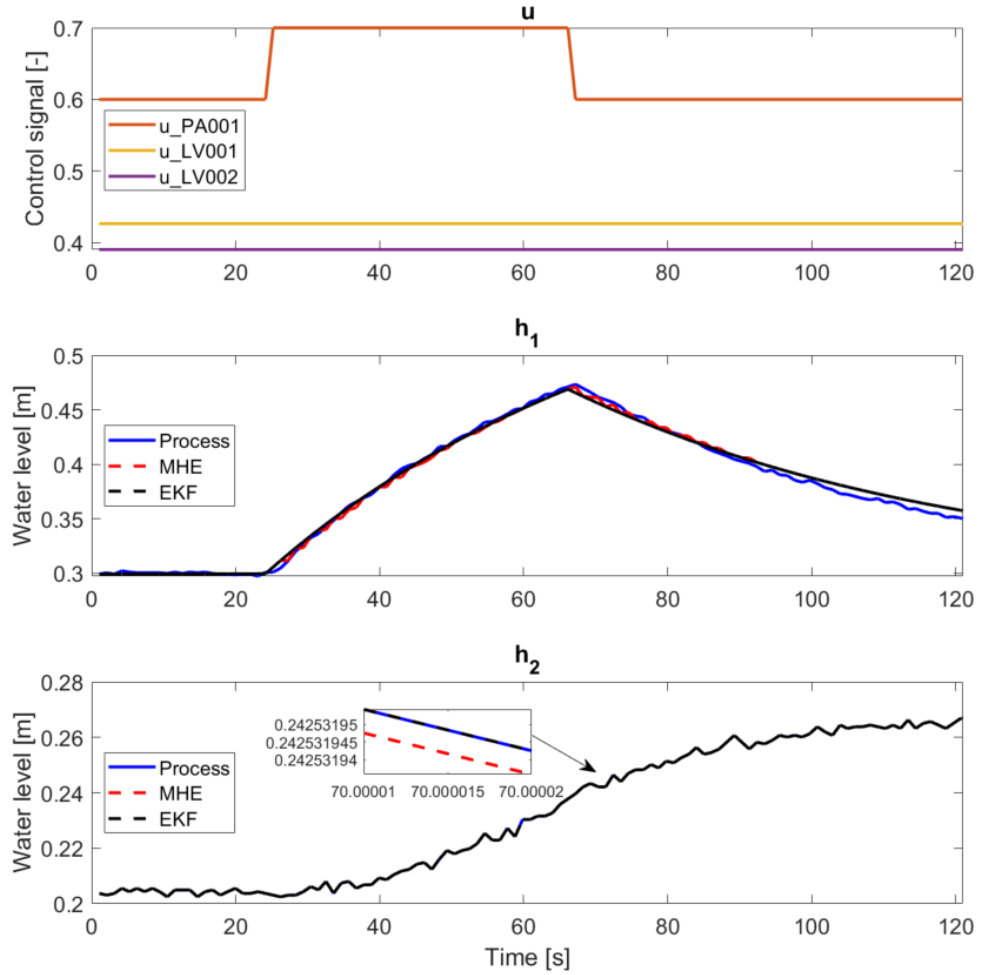


Figure 5.14: NMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Dataset *data1.mat*.

The covariance matrices used are:

$$Q_{MHE} = \begin{bmatrix} 9 \cdot 10^2 & 0 \\ 0 & 10^6 \end{bmatrix}, \quad Q_{EKF} = \begin{bmatrix} 10^{-1} & 0 \\ 0 & 10^2 \end{bmatrix}, \quad (5.13)$$

$$R_{MHE} = R_{EKF} = 1.6^{-6}.$$

The NMHE and EKF results are very similar to the LMHE and KF:

```

NMHE:
Estimation error mean h1: 3.49e-03
Estimation error mean h2: 6.95e-09
EKF:
Estimation error mean h1: 3.37e-03
Estimation error mean h2: 2.34e-11

```

The estimation error seen in time interval [90s – 120s] can be caused by slightly different process noise in this particular operating point. This can be an indicator that the process noise to the two-tank system vary and is operating point dependent.

The results for the Case 3 are shown in Fig. 5.16. The covariance matrices are adjusted as follows:

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^1 & 0 & 0 \\ 0 & 10^1 & 0 \\ 0 & 0 & 10^{-5} \end{bmatrix}, & Q_{EKF} &= \begin{bmatrix} 10^2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^{-1} \end{bmatrix}, \\
 R_{MHE} = R_{EKF} &= \begin{bmatrix} 2.8^{-6} & 0 \\ 0 & 1.6^{-6} \end{bmatrix}
 \end{aligned} \tag{5.14}$$

The performance of both estimators are similar, yet the NMHE estimate of u_{PA001} is less noisy:

```

NMHE:
Estimation error mean h1: 7.05e-09
Estimation error mean h2: 6.715e-09
Estimation error mean u_PA001: 1.15e-02
EKF:
Estimation error mean h1: 5.295e-11
Estimation error mean h2: 2.345e-09
Estimation error mean u_PA001: 1.53e-02

```

The video of the NMHE Case 3 estimation showcasing internal optimization process is available here:



Figure 5.15: <https://youtu.be/AeMlxtVhUqw>

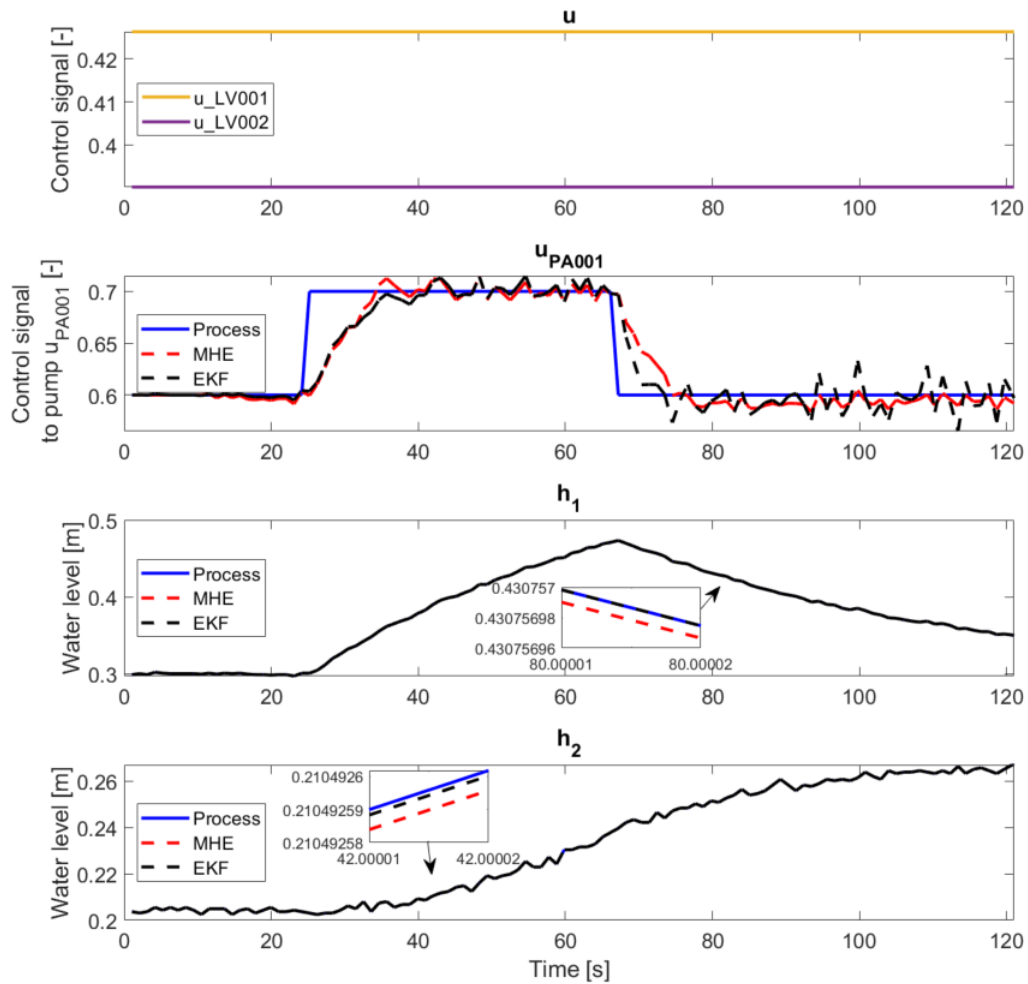


Figure 5.16: NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Dataset *data1.mat*.

As for the Case 4, the performance of the NMHE is worse than the LMHE. The estimates are very noisy. In contrast to LMHE, NMHE introduces additional complexity associated with nonconvexity, along with the uncertainties arising from measurement noise and low observability. This may be the cause of poor performance. The EKF is also unstable. The Case 1 results are available in Appendix B, Section B.3.2.

The performance comparison of all four estimators using *data2.mat* is presented in Fig. 5.17 the Case 2, and Fig. 5.18 the Case 3. In addition, estimation error means are reported in Table 5.3. Both the LMHE and NMHE are using the default *sqp* algorithm and the horizon length $N = 10$. The covariance matrices are the same ones used while testing *data1.mat*.

It is evident from the Fig. 5.17 that the NMHE and EKF is more accurate in estimating h_1 . Higher estimation error for the LMHE and KF is due to linearization errors. The total change in control input signals results in deviation from the original operating point. The deviation from the original operating point is the biggest at time interval $[95s - 120s]$ as the LMHE and KF estimation error. The LMHE is slightly better in compensating for the linearization error than the KF.

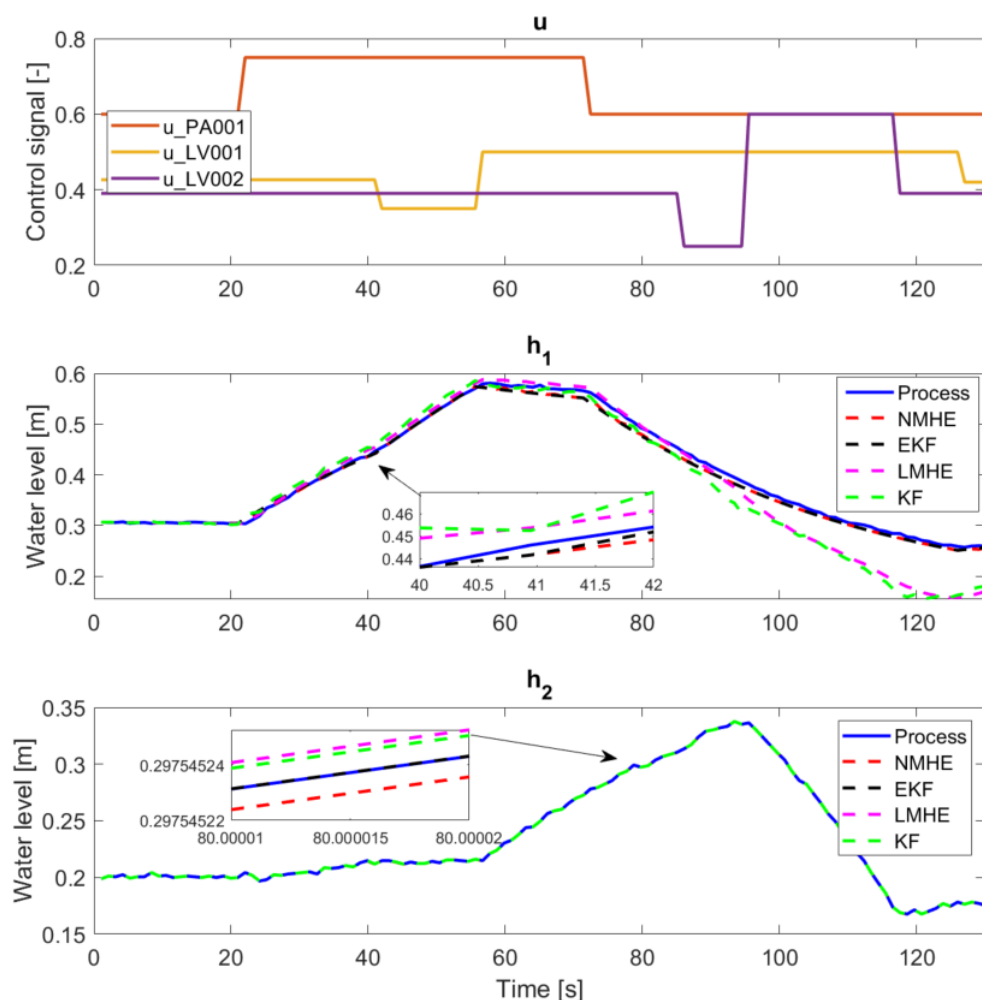


Figure 5.17: Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Dataset *data2.mat*.

In the Case 3, the Fig. 5.18, the NMHE has the most accurate estimate of u_{PA001} but also the slowest response time. In addition, the NMHE is the best in estimating the new position of u_{PA001} . The LMHE has a small deviation. This indicates the tendency of increased linearization error. In case of higher step than 0.15 would result in even bigger error. The estimate of EKF and KF, as seen in previous experiments, is more noisy than

moving horizon estimators. However, both keeps low estimation error means on h_1 and h_2 estimates.

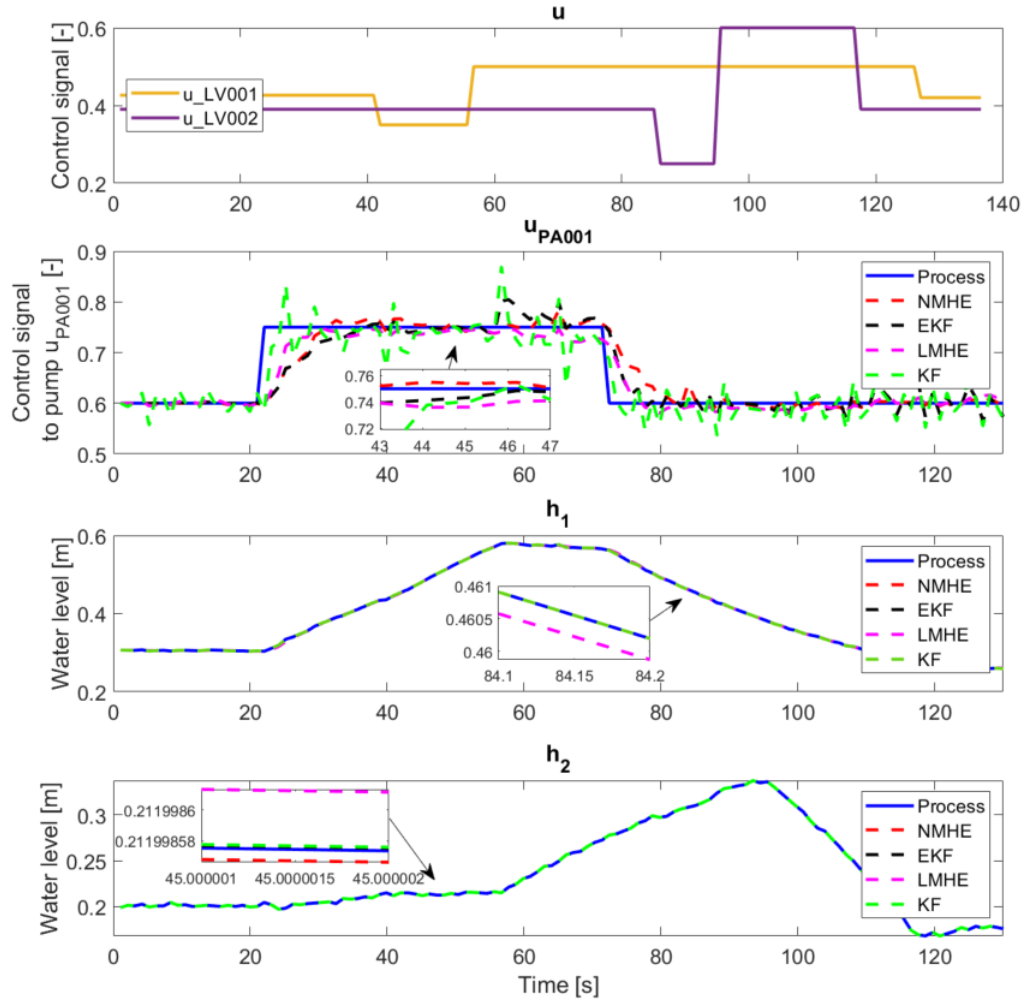


Figure 5.18: Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Dataset *data2.mat*.

Case	Estimator/ State	NMHE	EKF	LMHE	KF
Case 1	h_1	$2.08 \cdot 10^{-8}$	$4.55 \cdot 10^{-9}$	$8.87 \cdot 10^{-9}$	$4.86 \cdot 10^{-9}$
	h_2	$1.58 \cdot 10^{-8}$	$3.31 \cdot 10^{-9}$	$6.88 \cdot 10^{-9}$	$3.58 \cdot 10^{-9}$
Case 2	h_1	$6.19 \cdot 10^{-3}$	$6.70 \cdot 10^{-3}$	$2.74 \cdot 10^{-2}$	$3.24 \cdot 10^{-2}$
	h_2	$4.78 \cdot 10^{-6}$	$3.28 \cdot 10^{-11}$	$6.94 \cdot 10^{-9}$	$3.33 \cdot 10^{-9}$
Case 3	h_1	$1.19 \cdot 10^{-8}$	$6.41 \cdot 10^{-11}$	$6.66 \cdot 10^{-4}$	$3.23 \cdot 10^{-12}$
	h_2	$1.81 \cdot 10^{-8}$	$3.31 \cdot 10^{-9}$	$6.34 \cdot 10^{-8}$	$3.58 \cdot 10^{-9}$
	u_{PA001}	$1.50 \cdot 10^{-2}$	$1.87 \cdot 10^{-2}$	$1.38 \cdot 10^{-2}$	$2.54 \cdot 10^{-2}$

Table 5.3: Estimation error means for the NMHE, EKF, LMHE and KF in Case 1, 2 and 3. Dataset *data2.mat*

The estimation of Case 1 using *data2.mat* is equally good for both the LMHE and KF, and the NMHE and EKF. Since both measurements are present, the confidence in measurements are high, resulting in low process noise related errors. The estimation errors lies in the range of 10^{-9} , see Table 5.3. The Case 1 results can be found in Appendix B, Fig. B.23.

As mentioned earlier, the real data estimation is performed using the model error covariance matrices obtained in simulation environment. Comparing the real data estimation results presented in this section with the simulation results in Appendix B, Section B.2.3, it is evident that the resulting performance of the estimators are very similar. Although the simulation is performed with measurement noise $\sigma^2 = 10^{-6}$ and without process noise, the simulation model is a good representation of the real two-tank system and can be used to find tuning parameters.

On-line Estimation Results

The on-line verification of both LMHE and NMHE is performed on the two-tank plant located at the University of Stavanger in the laboratory KE E-458. Moving horizon estimation is accomplished using the Simulink model presented in Section 4.3. Based on the required processing time results, presented in Tables 5.1 and 5.2, the sampling time is adjusted to $T_s = 1.5s$, except NMHE Case 3, where the sampling time is set to $T_s = 2.5s$. Estimation by the LMHE and NMHE is performed using *sqp* algorithm and the horizon length $N = 10$. The results of both LMHE and NMHE Case 2 and Case 3 are available in Appendix B, Section B.3.3.

5.3 Experimental MPC and MHE Results

In this section, the experimental results are obtained in collaboration with Gent Luta². The aim of this experiment is to evaluate the MHE performance when combined with a controller. The MHE and MPC are somehow dual as both rely on solving an optimization problem that incorporate a dynamical system model with constraints over a fixed-size horizon. More information on the duality between MHE and MPC can be found in [28].

²Master's Thesis "Design and Implementation of Model Predictive Control for a Coupled Tank System".

In order to test the MHE and MPC in real time the Simulink model in 4.3 is modified as shown in Fig. 5.19. The experiments are conducted using linear version of MHE and MPC. The predictive controller generates a control action by solving an open-loop optimal control problem to predict systems behavior over a finite, receding horizon in which the current state of the system is used as the initial state³. The current state of the system is estimated by the MHE and supplied to the MPC, thus the control of the system is based on the state estimates rather than the real measurements.

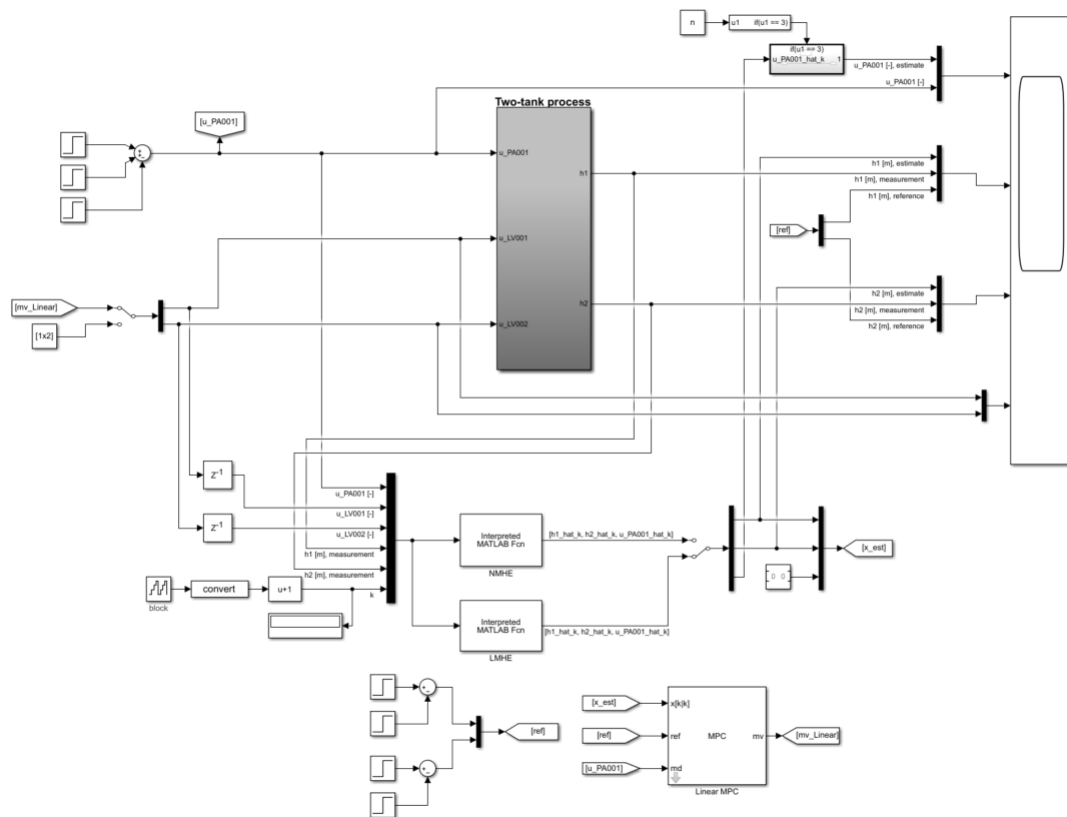


Figure 5.19: Simulink model for the two-tank system with MHE and MPC.

The control of the two-tank system is tested in three different scenarios: Case 1, 2 and 3. The sampling time of the two-tank system is set to $T_s = 1.5s$. The horizon lengths are set to $N_{LMHE} = 10$, $N_{LMPC} = 13$, $N_{LMPC,C} = 1$, where N_{LMPC} is the prediction horizon and $N_{LMPC,C}$ is the control horizon. The results for the Case 1 are shown in Fig. 5.20. The MPC is supplied with h_1 and h_2 estimates. Since in this case the estimation error is very small, the resulting MPC performance is as good as using actual measurements. The error means are:

³Detailed description of MPC implementation can be found in Master's Thesis "Design and Implementation of Model Predictive Control for a Coupled Tank System" by Gent Luta.

```

Estimation error mean h1: 2.32e-08
Estimation error mean h2: 1.55e-08
Reference tracking error mean h1: 6.32e-03
Reference tracking error mean h2: 4.85e-03

```

The response time to bring the states to the desired operating point or reference tracking errors are subject to the MPC design parameters, hence it is not discussed in this paper.

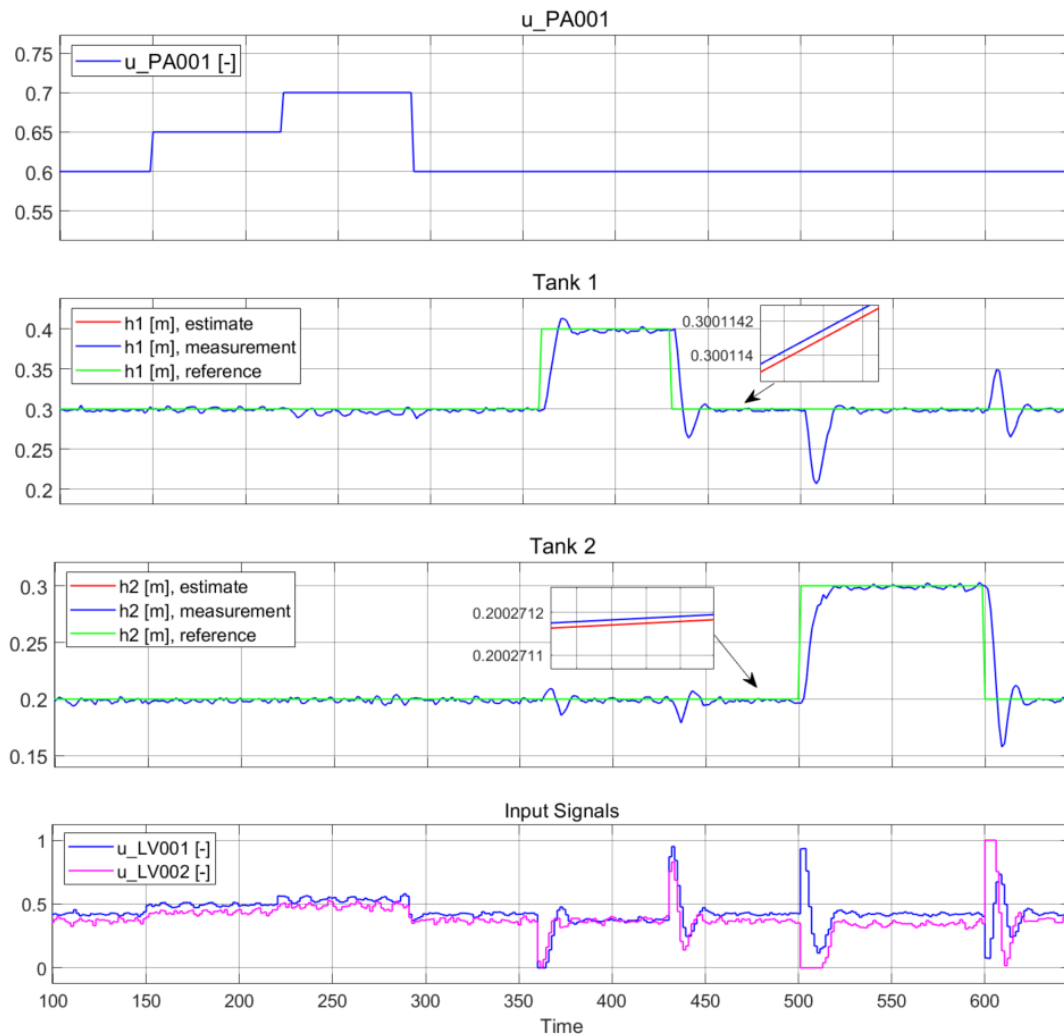


Figure 5.20: Control of the two-tank system with LMPC using LMHE. Case 1: estimation of h_1 and h_2 given the measurements h_1 and h_2 .

The Case 2 results are presented in Fig. 5.21. The states h_1 and h_2 are estimated using only h_2 measurement. As seen in the figure, the estimate of h_1 is less accurate, along with the control action. h_1 estimation error is highest when the system deviates from the initial operating point, indicating that the discrepancy is due to linearization errors. The error means are:

```

Estimation error mean h1: 1.55e-02
Estimation error mean h2: 7.98e-09
Reference tracking error mean h1: 1.70e-02
Reference tracking error mean h2: 4.67e-03

```

The decision whether the resulting performance is good enough depends on desired outcome or the specific application requirements and tolerances.

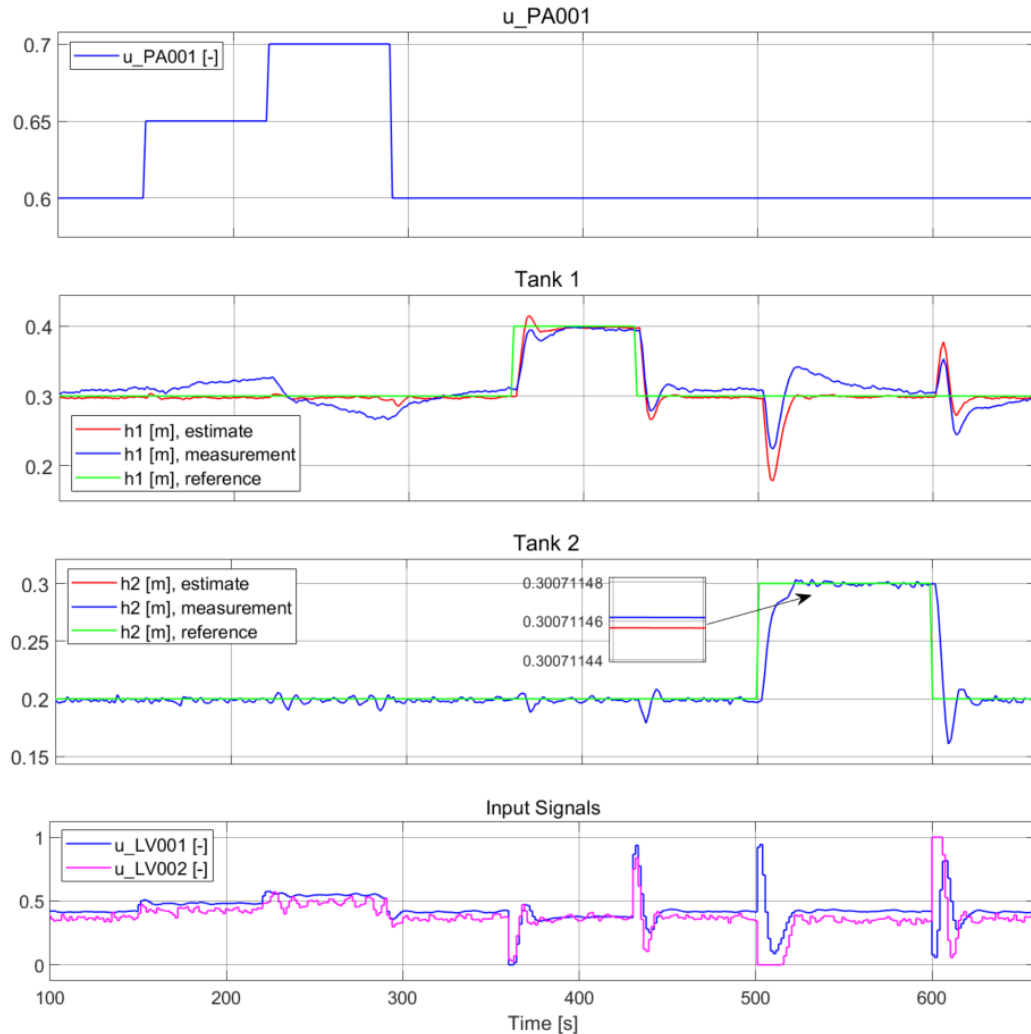


Figure 5.21: Control of the two-tank system with LMPC using LMHE. Case 2: estimation of h_1 , h_2 given the measurement h_2 .

The results for the Case 3 are shown in Fig. 5.22. The MPC is supplied with h_1 , h_2 and u_{PA001} estimates. The estimate of u_{PA001} is noisy due to linearization errors, however, the resulting reference tracking is good. The error means are:

```

Estimation error mean h1: 7.61e-04
Estimation error mean h2: 6.72e-08

```

Estimation error mean u_{PA001} : $1.52e-02$
 Reference tracking error mean h_1 : $7.57e-03$
 Reference tracking error mean h_2 : $4.70e-03$

Compared with the Case 1, where u_{PA001} is a measured disturbance, the reference tracking error mean for h_1 has increased from $6.32 \cdot 10^{-3}$ to $7.57 \cdot 10^{-3}$, while for h_2 it has decreased from $4.85 \cdot 10^{-3}$ to $4.70 \cdot 10^{-3}$, i.e., the MPC performance is almost as good as in the Case 1. Hence, it can be concluded that the u_{PA001} estimate is accurate enough and could replace the measurement. However, in the scenarios where the operating range to the system is wide, utilization of the NMHE should be considered.

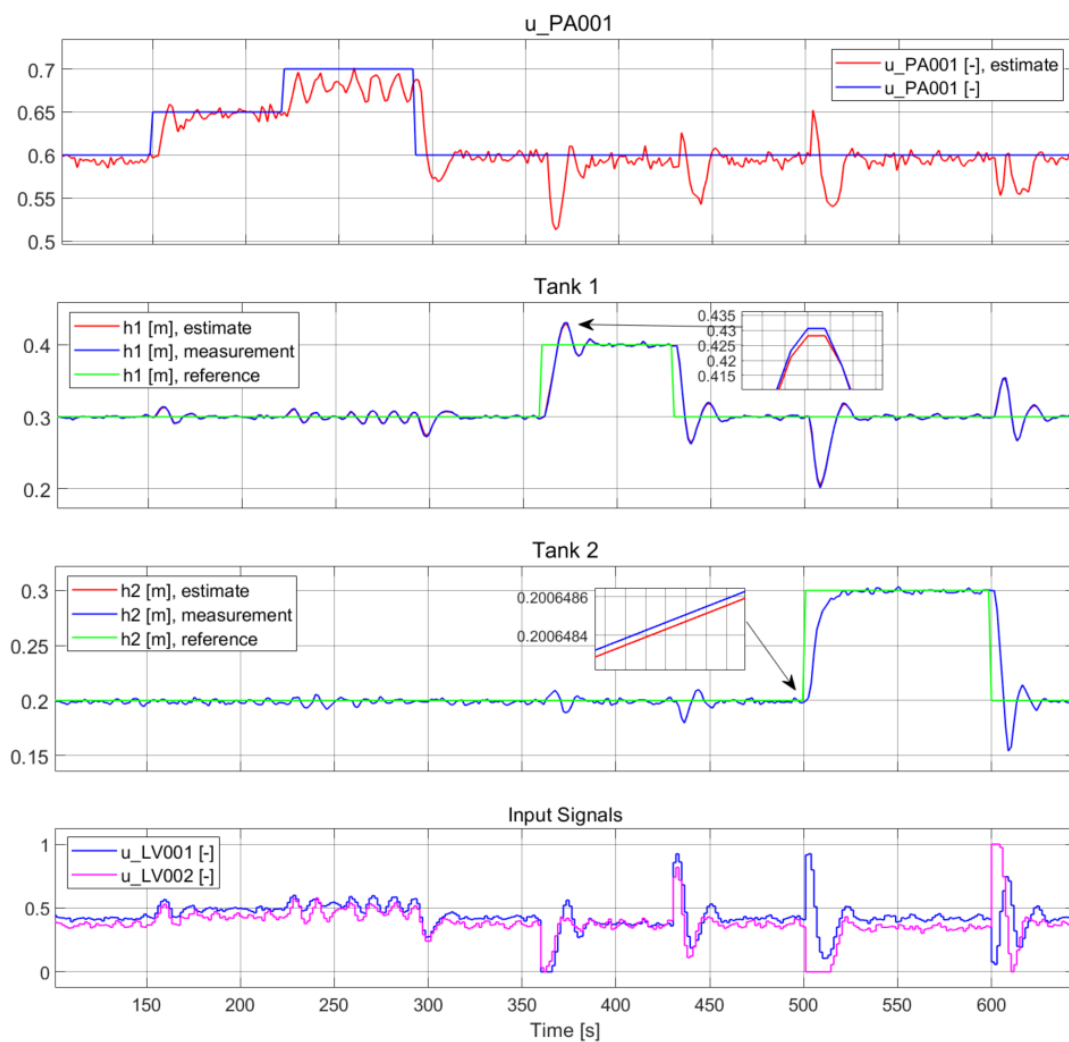


Figure 5.22: Control of the two-tank system with LMPC using LMHE. Case 3: estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 .

Chapter 6

Conclusions

The main goal of this thesis was to design and implement an MHE-based state estimator to observe the state of the two-tank system. A linear and nonlinear MHE were successfully implemented in Matlab for evaluation in simulation environment and Simulink for real-time estimation on the two-tank process plant. In addition, the LMHE was implemented and validated with the LMPC controller.

The performance of the LMHE and NMHE was compared against the Kalman and Extended Kalman filter. Moreover, the estimators were evaluated in different two-tank system configurations: estimation of two states, namely the water level in tank 1 and tank 2, and estimation of three states, which includes the water level in tank 1 and tank 2, as well as the unmeasured disturbance from the pump. An effort was made to investigate whether the states of the system could still be accurately estimated by removing one of the available measurements.

Based on simulations and experimental results of the two state estimation, the overall performance of all four estimators is good. However, as expected, due to linearization errors the estimation error for the LMHE and Kalman filter increases when the system deviates from the initial operating point, especially if only one measurement is present. Considering the three state estimation, the LMHE and NMHE estimate of the disturbance is more accurate than the KF and EKF estimate. The three state estimation using one measurement was achieved by the LMHE and KF, however the resulting estimates were delayed and inaccurate. The NMHE and EKF, on the other hand, were unsuccessful. It

has been concluded that this is due to low observability of the system and the uncertainties arising from relative high measurement noise.

In brief, MHE is powerful, and yet, complex and computationally inefficient estimation technique. In addition, the MHE performance relies on good optimization performance, adequate parameters and right choice of weighting matrices. Therefore, the tuning can become quite a challenging and time consuming task. The two-tank system is constrained solely by upper and lower bounds of the state variables, thereby the full utilization of the main MHE advantage to incorporate multiple equality and inequality constraints is limited. Hence, the two state estimation using the KF or EKF seems less complex and more computationally efficient choice. However, considering the three state estimation, the MHE outperforms the Kalman filters.

The LMHE is simpler and more computationally efficient than the NMHE. Furthermore, in contrast to the LMHE, the NMHE introduces additional complexity associated with nonconvexity, thus a local optimizer may provide a locally optimal solution instead of globally optimal. This may result in suboptimal and inaccurate estimates. Therefore, if the operating range of the two-tank system is narrow, the LMHE is a favored option.

Future Directions

Even though, the implementation of the LMHE and NMHE was successful, there are further improvements and interesting aspects that could be considered. Some of these are:

- Implementation and evaluation of more advanced arrival cost function approximation techniques.
- The computational efficiency was not a key goal in the MHE implementation, thus an effort could be made to improve the MHE application efficiency.
- Designing and implementing an MHE-based state estimator to the system with multiple equality and inequality constraints.

Appendix A

The Two-tank System

The theoretical and practical implementation results presented in this project are applied to the two-tank system located at the University of Stavanger in the laboratory KE E-459. In this chapter, Section [A.1](#) provides detailed description of the two-tank processing plant, Section [A.2](#) presents valves and pump characteristics, Section [A.3](#) and [A.4](#) derive the dynamical process model for the two-tank system. The appendix A is based on the first assignment on the two-tank system in course ELE320 Reguleringssteknikk [2].

A.1 Process Description

The process plant, shown in Fig. [A.1](#), consists of two water containers, tank 1 and tank 2. Tank 1 has a rectangular shape while tank 2 has a conical shape. Tank 1 has two inlets, from the pump PA001 and the mixer tap LV003, and two outlets, to the valve FV001 via hose coil and to the tank 2 via the valve LV001. Tank 1 is also equipped with a $2kW$ heating flask HE001.

Tank 2 has one inlet, which is from tank 1 via valve LV001, and two outlets, via the valve FV002 and via the valve LV002, whereas both outlets lead to the same collection container. The water in collection container is pumped back into tank 1 by the pump PA001.

Both tank 1 and tank 2 are open containers, thus the water surface pressure is the atmospheric pressure. The plant is outfitted with diverse types of instruments:

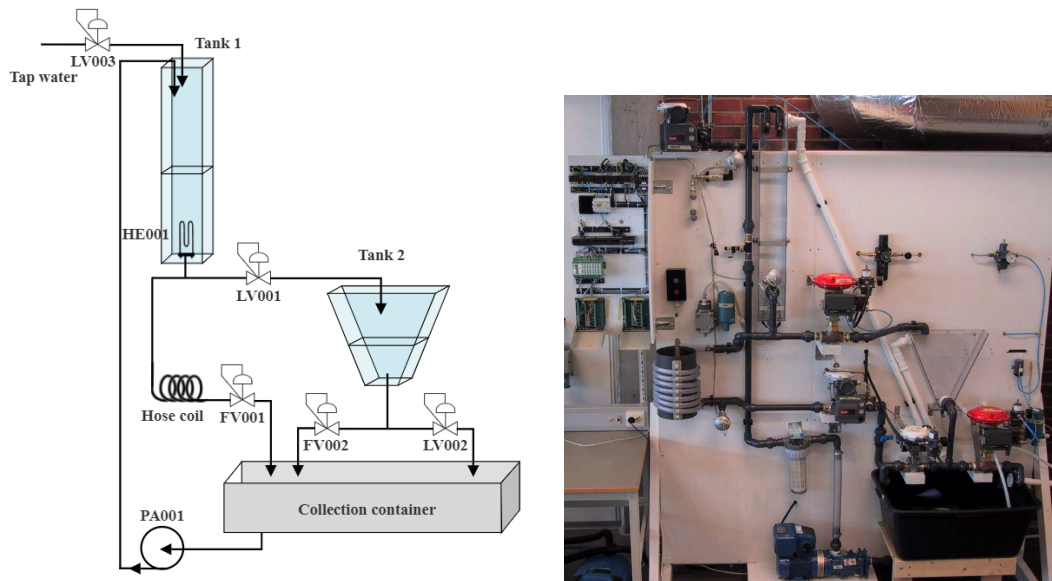


Figure A.1: A schematic sketch and picture of the two-tank system located at the University of Stavanger, KE E-459.

- Pressure gauge, PT001, measures the pressure in the tap water.
- Temperature meter, TT003, measures the temperature of the tap water.
- Temperature meter, TT001, measures the temperature of water in tank 1.
- Level meter, LT001, measures the level of the water in tank 1.
- Temperature meter, TT002, measures the temperature of the water from tank 1, which is delayed through the hose coil.
- Level meter, LT002, measures the water level in tank 2.
- Flow meter, FT001, measures the water flow from the pump PA001.

The two-tank system is designed to simulate a variety of different industrial processes. In this project, the two-tank system is limited to the valves LV001, LV002, the pump PA001 and the instruments FT001, LT001 and LV002. The Fig. A.2 shows a simplified version of the two-tank system schematic sketch, whereas Table A.2 gives an overview of the two-tank system variables.

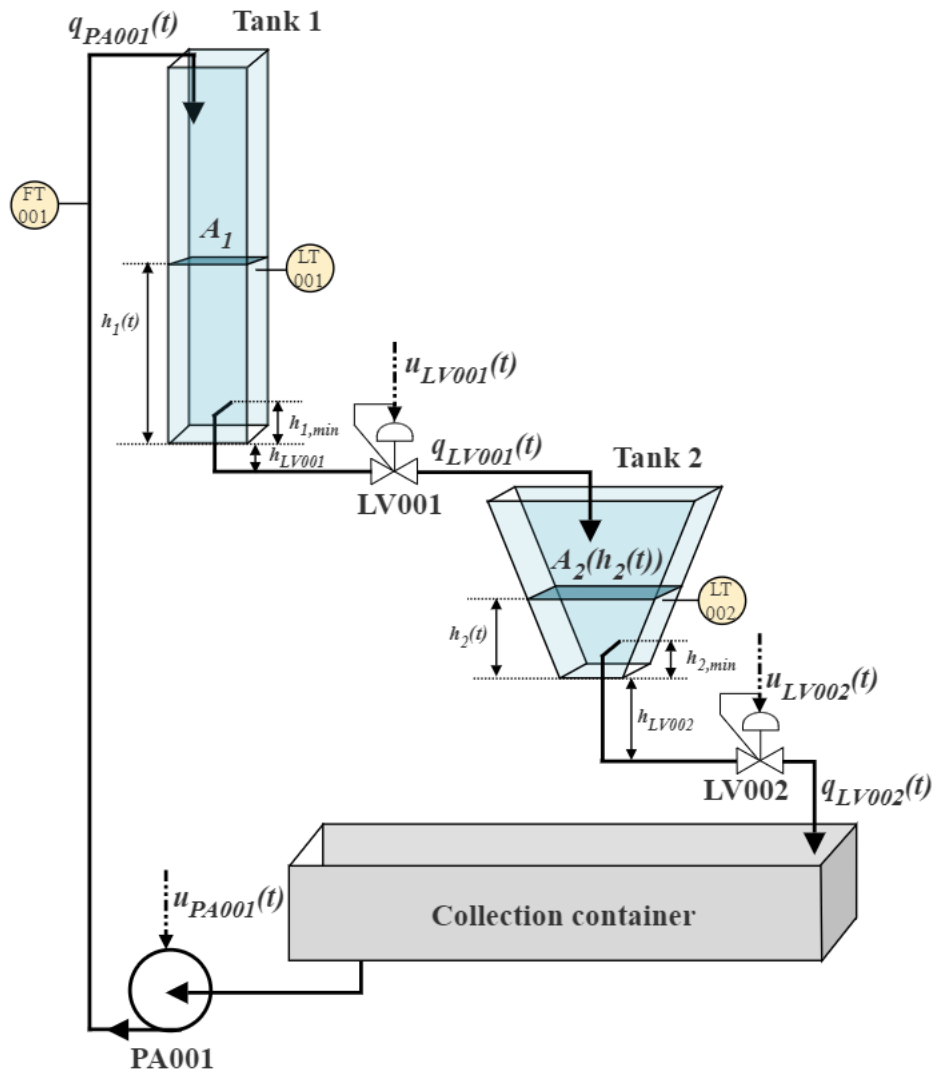


Figure A.2: A simplified version of the two-tank system schematic sketch.

A.2 Valves and Pump Characteristics

The mathematical model of the valves describes the volume flow $q[m^3/h]$ through the valve as a function of the valve size $K_v[\frac{m^3/h}{\sqrt{bar}}]$, valve opening z , and pressure drop across the valve $\Delta p[bar]$, thus the valve equation is given by

$$q(t) = K_v f(z(t)) \sqrt{\Delta p(t)}, \quad (\text{A.1})$$

where $f(z(t))$ is the valve characteristic. The pressure upstream the valve consist of the atmospheric pressure p_0 and water pressure p while the pressure downstream is only the

atmospheric pressure. Therefore, the differential pressure $\Delta p[Pa]$ is given by

$$\Delta p(t) = p(t) = \rho g h(t), \quad (\text{A.2})$$

where ρ is the density of the liquid, g is the gravitational constant and $h(t)$ is the height of the liquid above the valve.

The valve characteristic describes the relation between the valve opening $z(t)$ and the flow $f(\cdot)$ through the valve. The relation for both the valve LV001 and LV002 is approximated by

$$f(z(t)) = \frac{e^{z(t)^{1.2}} - 1}{e^1 - 1}. \quad (\text{A.3})$$

The valves LV001 and LV002 are fail open valves and are controlled using compressed air. The dynamics of the actuator is fast compared with the two-tank process and, therefore, neglected. Hence the valve opening $z(t)$ is determined by the control input $u(t)$ and (A.3) can be rewritten as

$$f_i(u_{LV00i}(t)) = \frac{e^{u_{LV00i}(t)^{1.2}} - 1}{e^1 - 1}, \quad (\text{A.4})$$

where $i = 1, 2$ represents the valve LV001 and LV002, respectively. The relation between $f_i(t)$ and $u_{LV00i}(t)$ is shown in Fig. A.3.

Combining equations (A.1) and (A.2), and converting them into SI units, the volume flow of the valves LV001 and LV002 is represented by equation

$$q_{LV00i}(t) = \frac{K_{v,LV00i} f_i(u_{LV00i}(t))}{3600} \sqrt{\frac{\rho g h_i(t)}{100000}}. \quad (\text{A.5})$$

Pump characteristic is given by the relation

$$q_{PA001}(t) = f_3(u_{PA001}(t)), \quad (\text{A.6})$$

where $q_{PA001}(t)$ is the flow through the pump, and $u_{PA001}(t)$ is the control signal. Fig. A.4 presents the relation between the flow through the pump and control signal provided in [2].

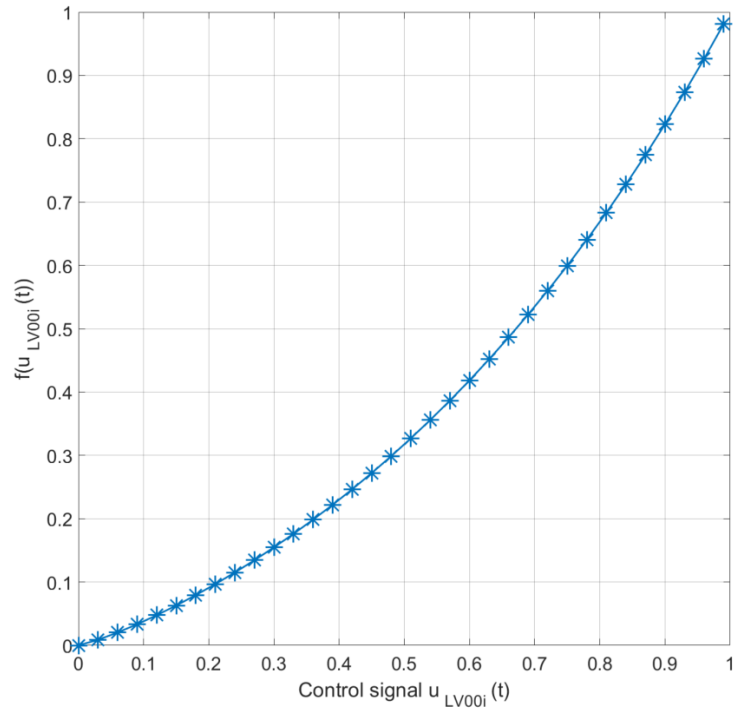


Figure A.3: Valve characteristic for LV001 and LV002[2].

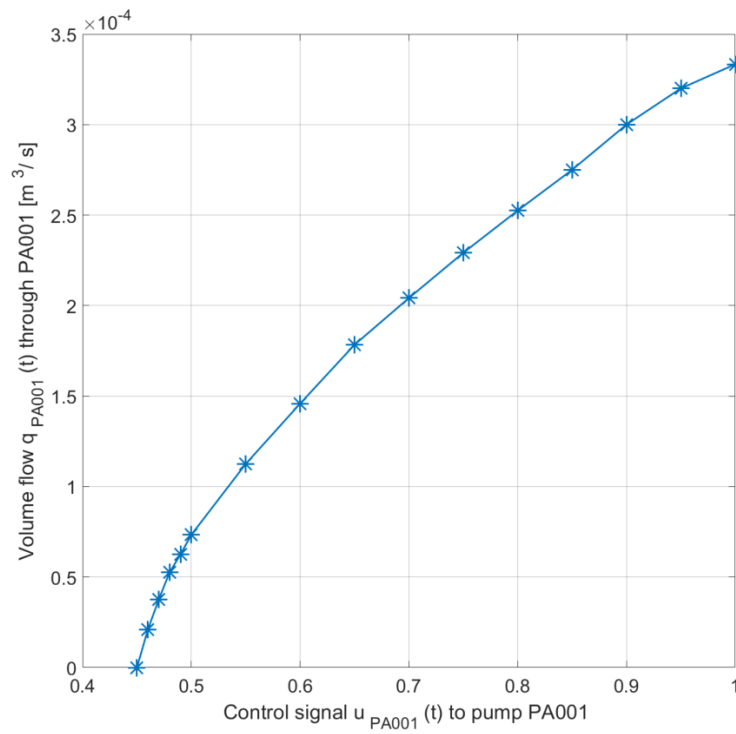


Figure A.4: Pump PA001 characteristic [2].

A.3 Dynamic Model of Tank 1

Considering the system boundaries and the dynamics of the valves and the pump presented in the Section A.1, the dynamical model for the two-tank system can be obtained using the mass conservation law

$$\frac{dm(t)}{dt} = \sum w_{in}(t) - \sum w_{out}(t), \quad (\text{A.7})$$

where $\frac{dm(t)}{dt}$ is the rate of mass accumulation, $w_{in}(t)$ is the rate of mass in and $w_{out}(t)$ is the rate of mass out.

The mass flows in tank 1 can be expressed as

$$w_{in}(t) = \rho q_{PA001}(t) = \rho f_3(u_{PA001}(t)) \quad (\text{A.8})$$

and

$$w_{out}(t) = \rho q_{LV001}(t) = \rho f_3(u_{PA001}(t)). \quad (\text{A.9})$$

Substituting (A.5) into (A.9) and considering that if $h_i(t) > h_{i,min}$, $\Delta p_i(t) = \rho g (h_i(t) + h_{i,min})$, the equation (A.9) can be rewritten as

$$w_{out}(t) = \rho \frac{K_{v,LV001} f_1(u_{LV001}(t))}{3600} \sqrt{\frac{\rho g (h_1(t) + h_{1,min})}{100000}}. \quad (\text{A.10})$$

Substituting (A.8) and (A.9) into (A.7), where $m_1(t) = \rho V(t) = \rho A_1 h_1(t)$, the differential equation, describing the dynamics of the water level in tank 1, has the following form

$$\frac{dh_1(t)}{dt} = \frac{1}{A_1} \left(f_3(u_{PA001}(t)) - \frac{K_{v,LV001} f_1(u_{LV001}(t))}{3600} \sqrt{\frac{\rho g (h_1(t) + h_{1,min})}{100000}} \right). \quad (\text{A.11})$$

A.4 Dynamic Model of Tank 2

The dynamical model for tank 2 can be derived applying the same procedure as for tank 1. The mass flows in tank 2 is given by

$$w_{in}(t) = \rho q_{LV001}(t) = \rho \frac{K_{v,LV001} f_1(u_{LV001}(t))}{3600} \sqrt{\frac{\rho g (h_1(t) + h_{1,min})}{100000}} \quad (\text{A.12})$$

and

$$w_{out}(t) = \rho q_{LV002}(t) = \rho \frac{K_{v,LV002} f_2(u_{LV002}(t))}{3600} \sqrt{\frac{\rho g (h_2(t) + h_{2,min})}{100000}}. \quad (\text{A.13})$$

Due to the canonical shape, the volume, as well as the surface area, of the liquid in tank 2 is a function of the height $h_2(t)$. The rate of height can be derived as follows

$$\frac{dm_2(t)}{dt} = \rho \frac{dV_2(h_2(t))}{dt} = \rho \frac{dV_2(h_2(t))}{dh_2} \cdot \frac{dh_2(t)}{dt} = \rho A_2(h_2(t)) \frac{dh_2(t)}{dt}, \quad (\text{A.14})$$

where $m_2(t) = \rho V_2(h_2(t))$. It can be shown that using geometry and dimensions, provided in Table A.1 and Fig. A.5, the surface area of the liquid in is given by

$$A_2(h_2(t)) = A_{2,0} + h_2(t) \cdot \frac{b_{2,max} - b_{2,min}}{h_{2,max}} \cdot d_2 = 0.004 + 0.07 h_2(t). \quad (\text{A.15})$$

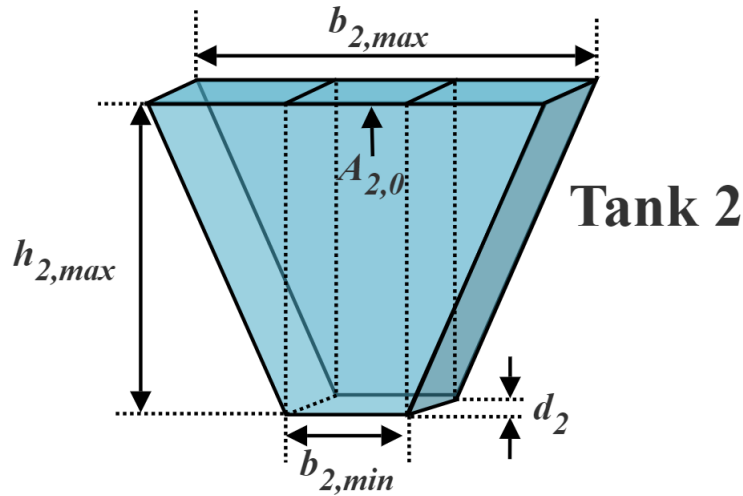


Figure A.5: Simplified sketch of tank 2.

Name	Description	Value
$A_{2,0}$	Area of tank 2 bottom	0.004 m
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
$h_{2,max}$	Height of tank 2	0.4 m

Table A.1: Tank 2 dimensions [2].

Hence, combining (A.12), (A.13), (A.14) and (A.15), and substituting into (A.7), the differential equation for tank 2 has the following form:

$$\frac{dh_2(t)}{dt} = \frac{1}{0.004 + 0.07 h_2(t)} \left(\frac{K_{v,LV001} f_1(u_{LV001}(t))}{3600} \sqrt{\frac{\rho g (h_1(t) + h_{1,min})}{100000}} - \frac{K_{v,LV002} f_2(u_{LV002}(t))}{3600} \sqrt{\frac{\rho g (h_2(t) + h_{2,min})}{100000}} \right). \quad (\text{A.16})$$

The dynamical model of the two-tank system consists of two nonlinear ODE differential equations, where (A.11) describes the dynamics of the water level in tank 1 and (A.16) describes the dynamics of the water level in tank 2, thus the system is second-order nonlinear system.

Name	Description	Unit	Value
$h_1(t)$	Water level in tank 1 measured with LT001	m	0-1
$h_2(t)$	Water level in tank 2 measured with LT002	m	0-0.4
A_1	Area for tank 1	m^2	0.0096
$A_2(h_2(t))$	Area for tank 2	m^2	0.025-0.07
ρ	Water density	kg/m^3	1000
g	Gravity acceleration	m/s^2	9.81
q_{PA001}	Volume flow from pump PA001	m^3/s	0-12
u_{PA001}	Control signal to pump PA001	-	0-1
q_{LV001}	Volume flow from valve LV001	m^3/s	0-17
u_{LV001}	Control signal to valve LV001	-	0-1
q_{LV002}	Volume flow from valve LV002	m^3/s	0-17
u_{LV002}	Control signal to valve LV002	-	0-1
$K_{v,LV001}$	Valve constant for valve LV001	$\frac{m^3}{time\sqrt{bar}}$	11.25
$K_{v,LV002}$	Valve constant for valve LV002	$\frac{m^3}{time\sqrt{bar}}$	11.25
$h_{1,min}$	Height between tank 1 bottom and tank 1 outlet	m	0.14
$h_{2,min}$	Height between tank 2 bottom and tank 2 outlet	m	0.03
h_{LV001}	Height between tank 1 bottom and valve LV001	m	0.05
h_{LV002}	Height between tank 2 bottom and valve LV002	m	0.25

Table A.2: Two-tank system variables [2].

Appendix B

Additional Results

B.1 NMHE Verification

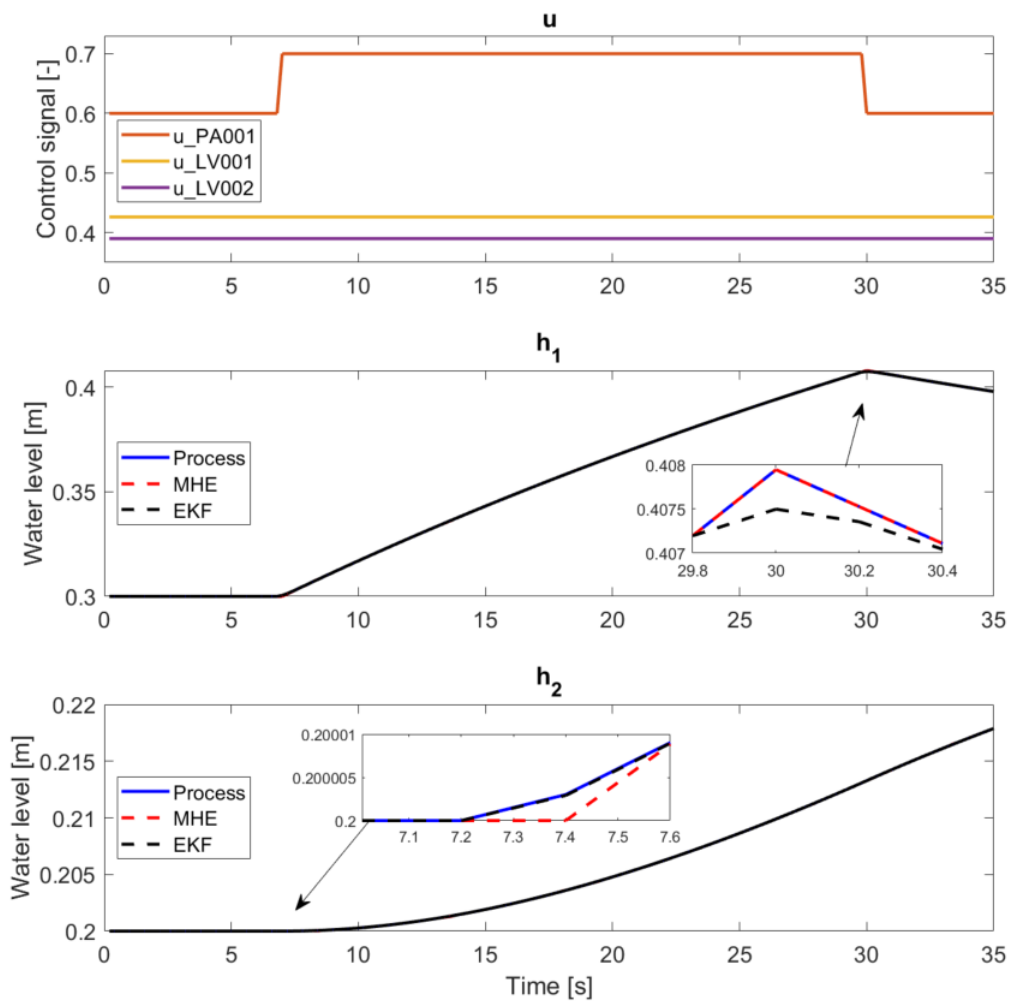


Figure B.1: NMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . No process or measurement noise.

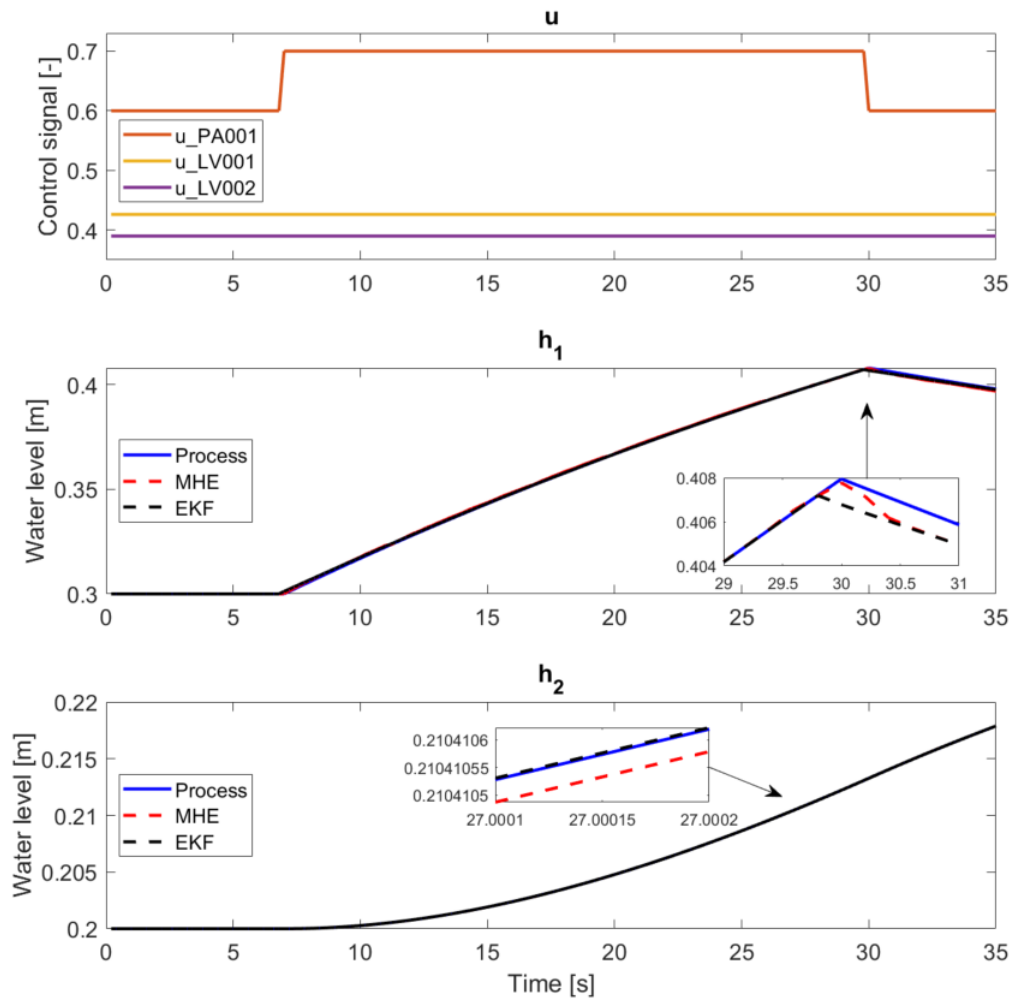


Figure B.2: NMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . No process or measurement noise.

B.2 Simulation Environment Results

B.2.1 LMHE

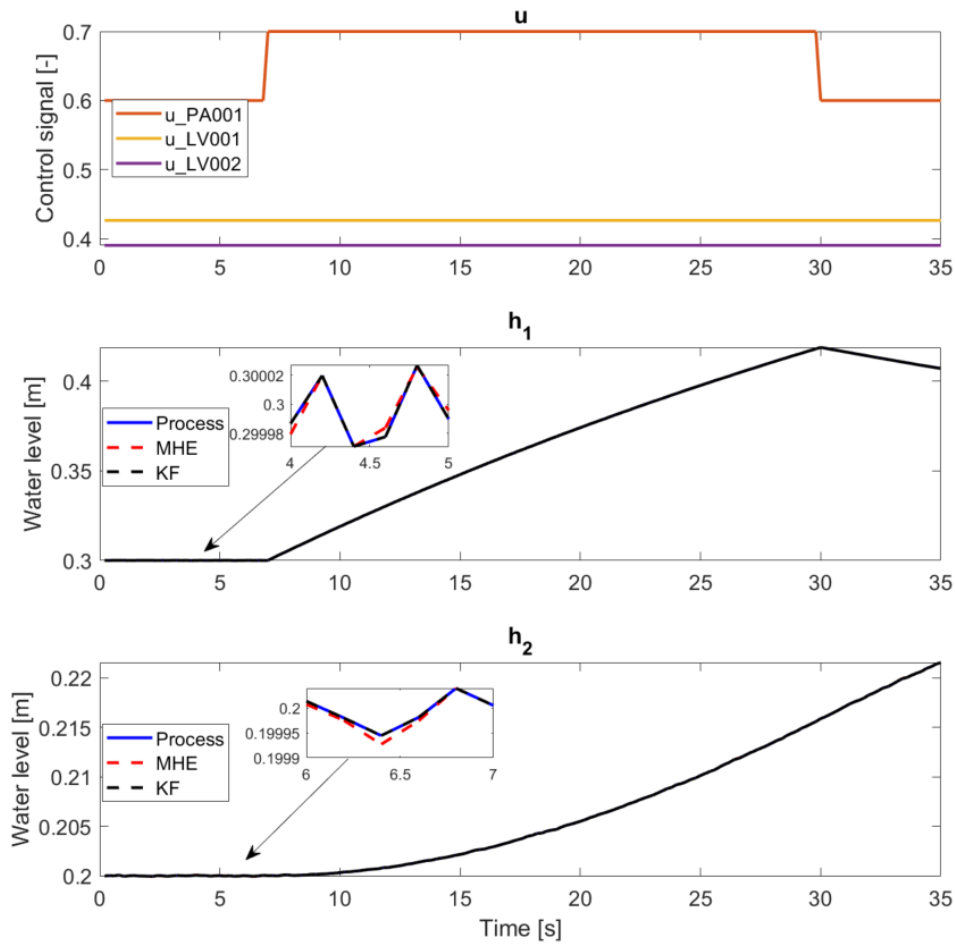


Figure B.3: LMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. *Sqp* algorithm with the horizon length $N = 10$.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^1 & 0 \\ 0 & 10^1 \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \\
 R_{MHE} = R_{KF} &= \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix}
 \end{aligned} \tag{B.1}$$

LMHE:

Estimation error mean h1: 6.87e-07

Estimation error mean h2: 9.23e-07

KF:

Estimation error mean h1: 5.24e-14

Estimation error mean h2: 3.48e-14

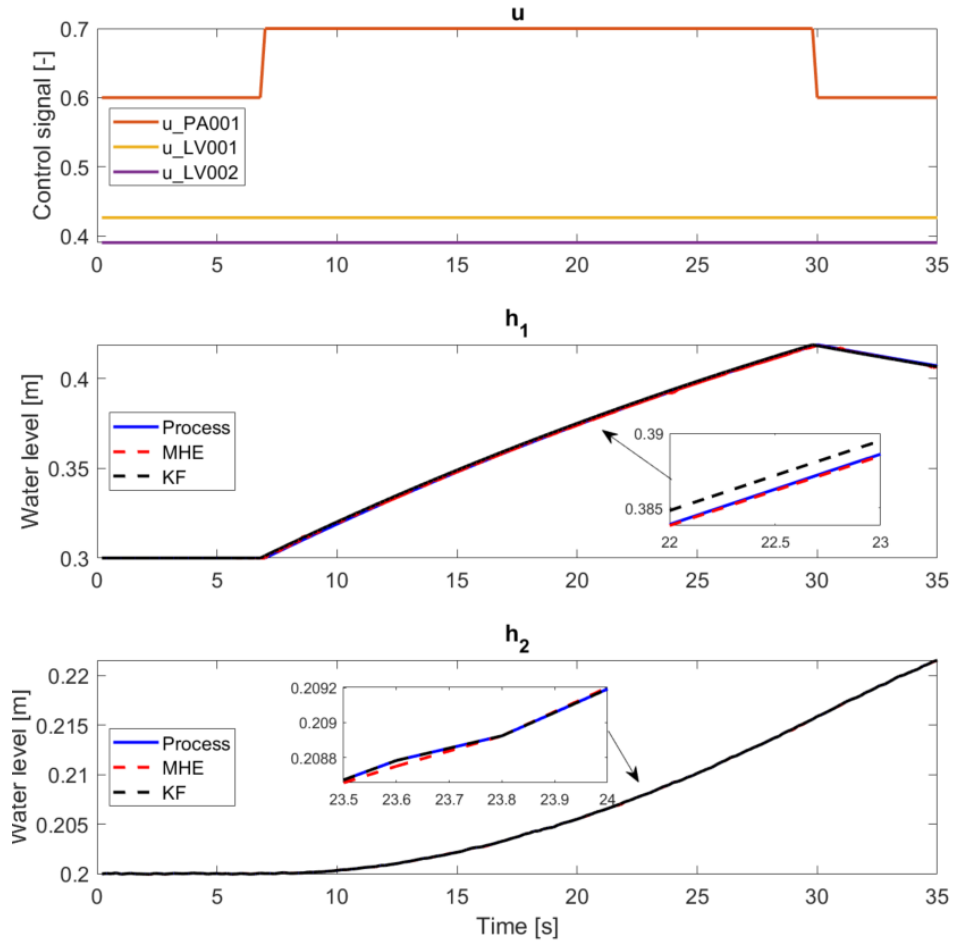


Figure B.4: LMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. Sqp algorithm with the horizon length $N = 10$.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^{-3} & 0 \\ 0 & 10^{-2} \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 10^{-2} & 0 \\ 0 & 1 \end{bmatrix}, \\
 R_{MHE} &= R_{KF} = 10^{-9}
 \end{aligned}
 \tag{B.2}$$

LMHE :

Estimation error mean h1: 3.13e-04

Estimation error mean h2: 3.44e-06

KF :

Estimation error mean h1: 7.42e-04

Estimation error mean h2: 3.48e-14

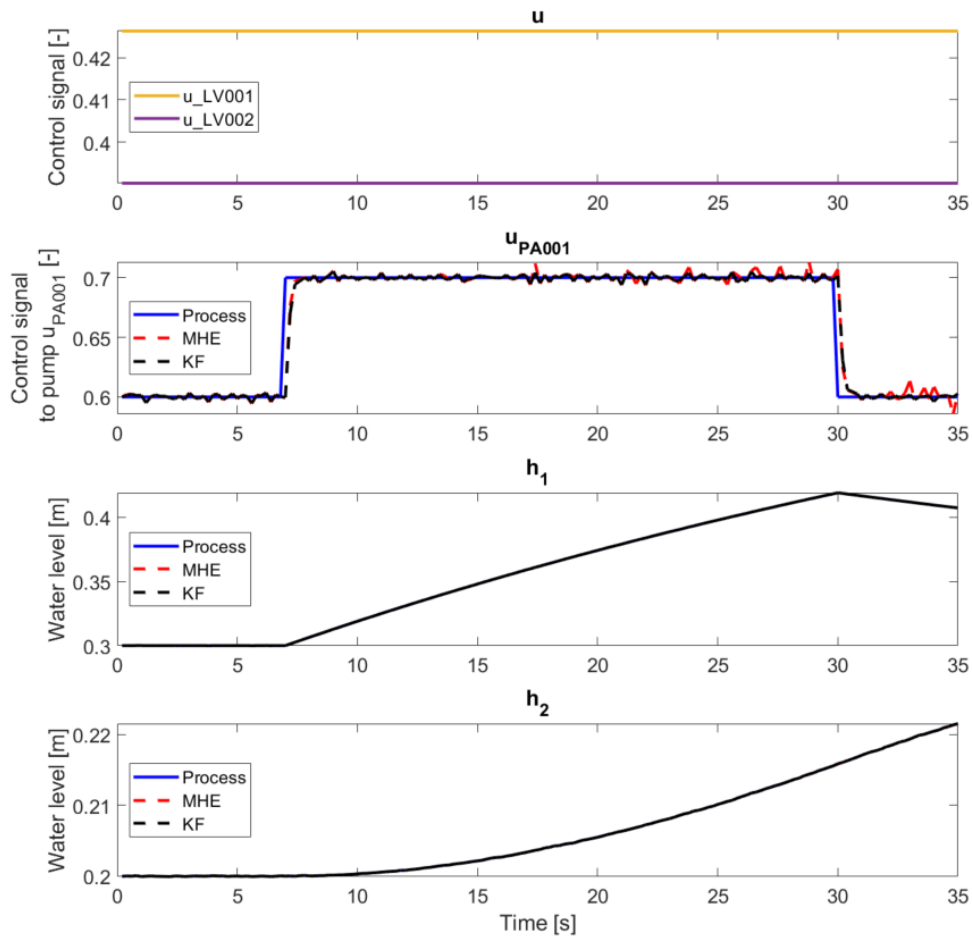


Figure B.5: LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. *Sqp* algorithm with the horizon length $N = 10$. *OptimalityTolerance* = *StepTolerance* = 10^{-10} .

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^{-9} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^{-5} \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^4 \end{bmatrix}, \\
 R_{MHE} = R_{KF} &= \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix}
 \end{aligned} \tag{B.3}$$

LMHE:

Estimation error mean h1: 3.72e-06

Estimation error mean h2: 1.17e-06

Estimation error mean u_PA001: 3.94e-03

KF:

Estimation error mean h1: 2.19e-14

Estimation error mean h2: 3.48e-14

Estimation error mean u_PA001: 3.30e-03

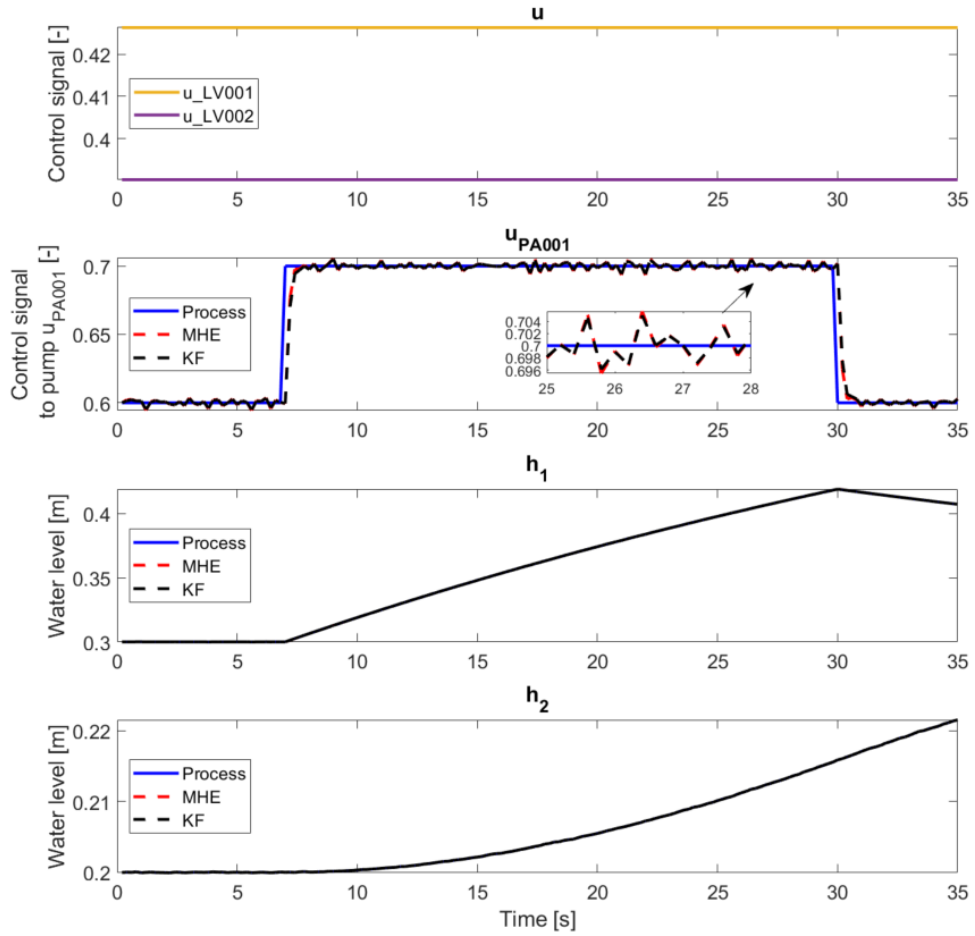


Figure B.6: LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. *Sqp* algorithm with the horizon length $N = 5$.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^{-9} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^{-5} \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^4 \end{bmatrix}, \\
 R_{MHE} = R_{KF} &= \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix}
 \end{aligned} \tag{B.4}$$

LMHE :

Estimation error mean h1: 2.18e-06

Estimation error mean h2: 7.01e-09

Estimation error mean u_PA001: 3.39e-03

KF :

Estimation error mean h1: 2.19e-14

Estimation error mean h2: 3.48e-14

Estimation error mean u_PA001: 3.30e-03

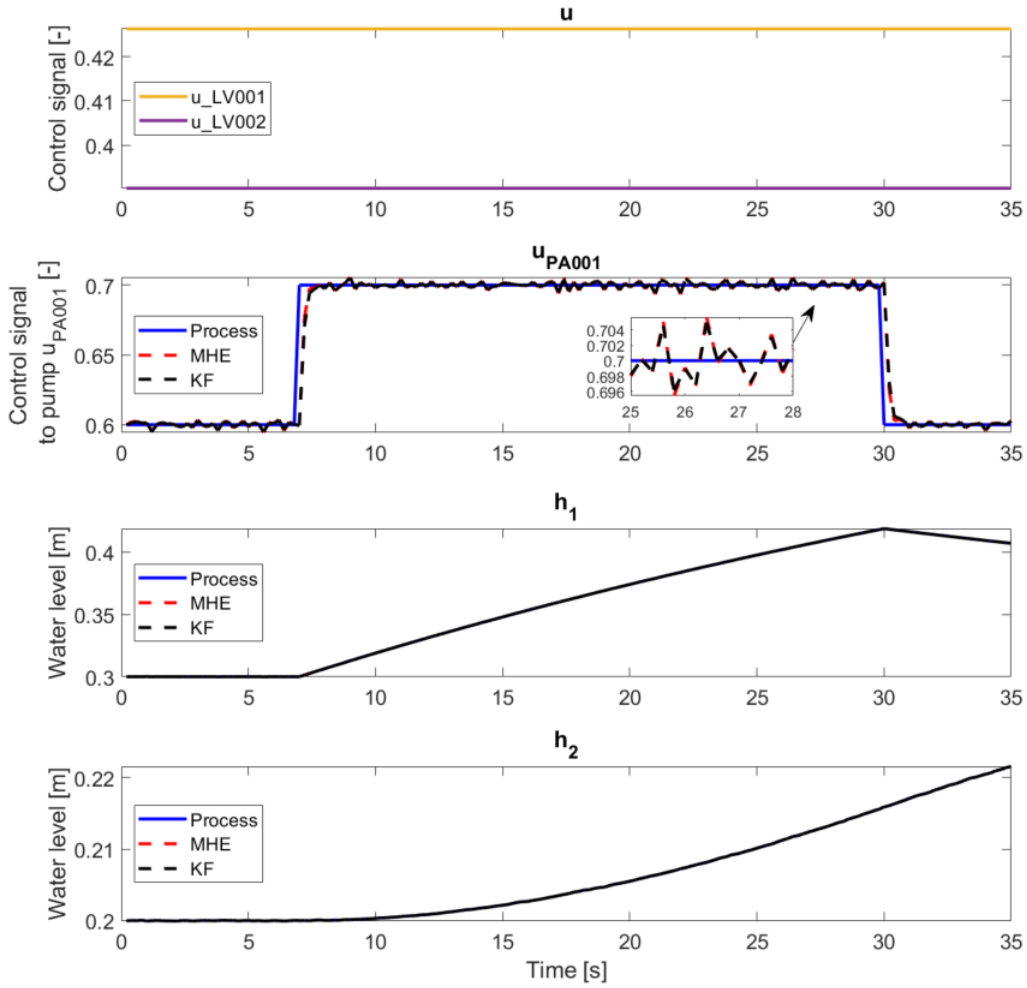


Figure B.7: LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. *Interior – point* algorithm with the horizon length $N = 10$.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^{-9} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^{-5} \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^4 \end{bmatrix}, \\
 R_{MHE} = R_{KF} &= \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix}
 \end{aligned} \tag{B.5}$$

LMHE:

Estimation error mean h1: 2.20e-06

Estimation error mean h2: 1.06e-08

Estimation error mean u_PA001: 3.38e-03

KF:

Estimation error mean h1: 2.19e-14

Estimation error mean h2: 3.48e-14

Estimation error mean u_PA001: 3.30e-03

B.2.2 NMHE

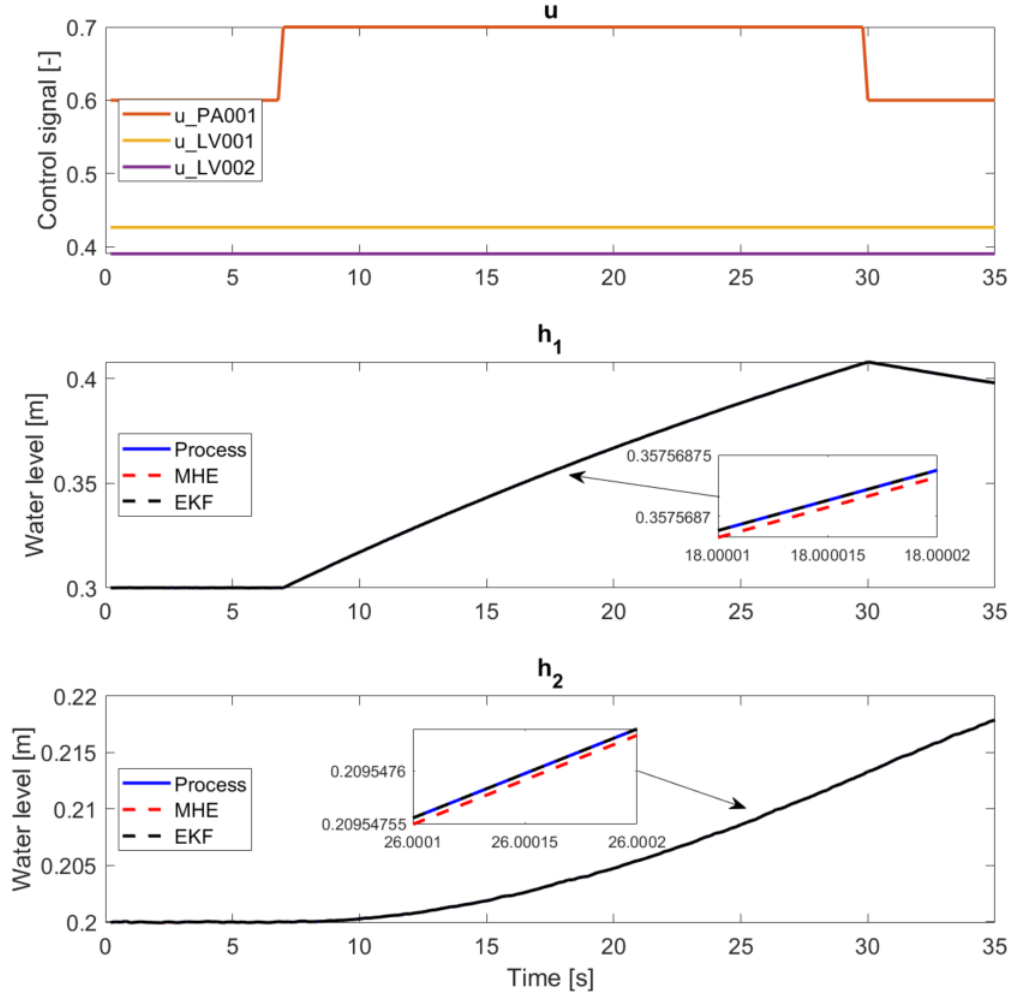


Figure B.8: NMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. *Sqp* algorithm with the horizon length $N = 10$.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^1 & 0 \\ 0 & 10^1 \end{bmatrix}, & Q_{EKF} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \\
 R_{MHE} = R_{EKF} &= \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix}
 \end{aligned} \tag{B.6}$$

NMHE :

Estimation error mean h1: 6.74e-07

Estimation error mean h2: 7.20e-07

EKF :

Estimation error mean h1: 5.09e-14

Estimation error mean h2: 3.48e-14

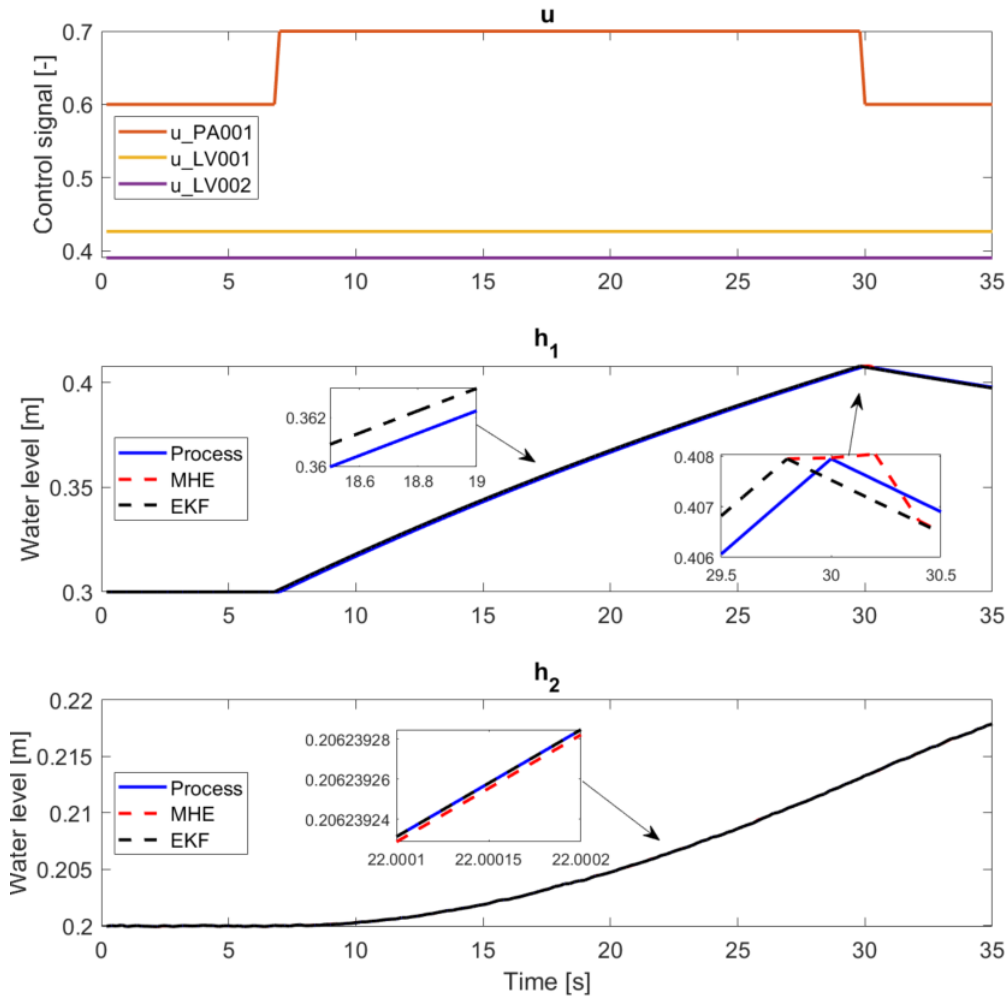


Figure B.9: NMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. Sqp algorithm with the horizon length $N = 10$.

$$Q_{MHE} = \begin{bmatrix} 6 \cdot 10^1 & 0 \\ 0 & 10^3 \end{bmatrix}, \quad Q_{EKF} = \begin{bmatrix} 10^{-4} & 0 \\ 0 & 10^{-1} \end{bmatrix}, \quad (B.7)$$

$$R_{MHE} = R_{EKF} = 10^{-9}$$

NMHE:

Estimation error mean h1: 6.75e-04

Estimation error mean h2: 1.75e-06

EKF:

Estimation error mean h1: 6.75e-04

Estimation error mean h2: 3.48e-13

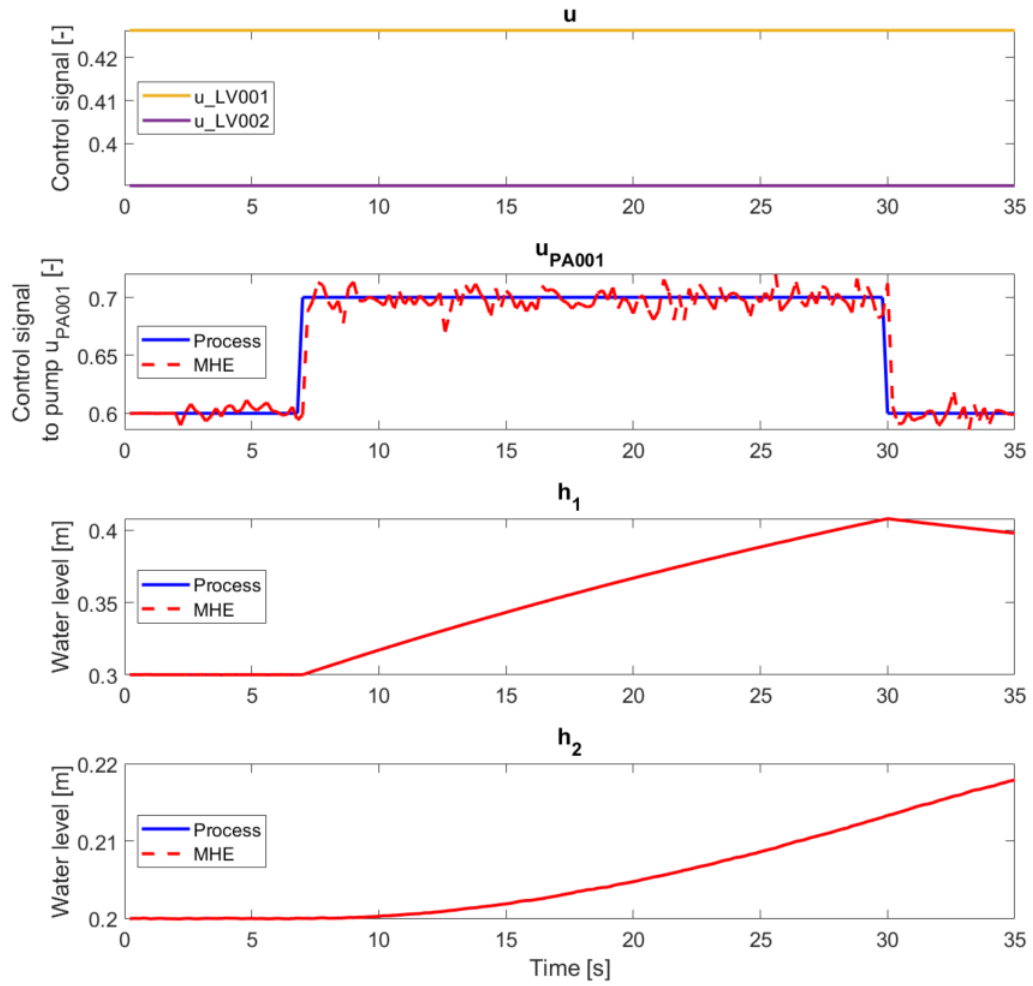


Figure B.10: NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. *Active-set* algorithm with the horizon length $N = 10$.

$$Q_{MHE} = \begin{bmatrix} 10^{-2} & 0 & 0 \\ 0 & 10^{-2} & 0 \\ 0 & 0 & 10^{-4} \end{bmatrix}, \quad R_{MHE} = \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix} \quad (\text{B.8})$$

NMHE :

Estimation error mean h1: 7.21e-09

Estimation error mean h2: 7.68e-09

Estimation error mean u_PA001: 7.53e-03

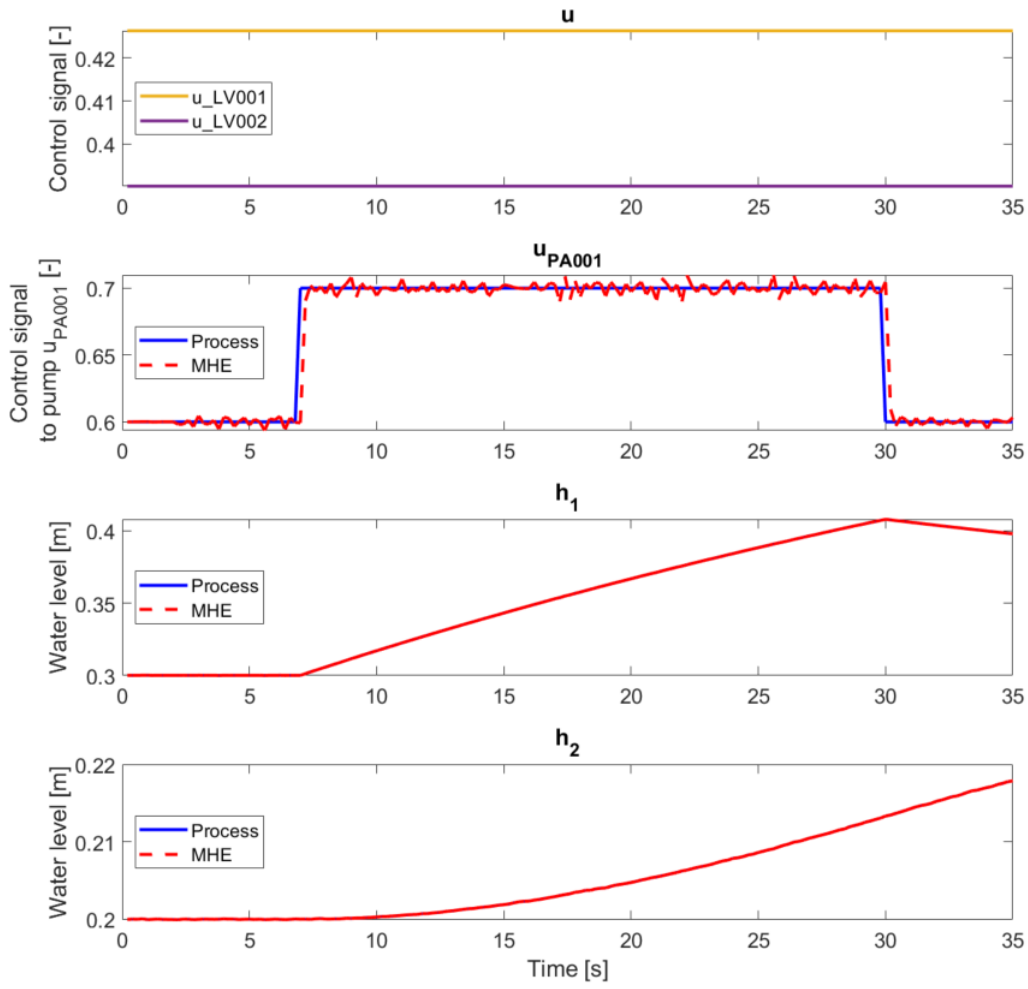


Figure B.11: NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. *MultiStart* method with 5 start points using *sqp* algorithm with the horizon length $N = 10$.

$$Q_{MHE} = \begin{bmatrix} 10^{-2} & 0 & 0 \\ 0 & 10^{-2} & 0 \\ 0 & 0 & 10^{-4} \end{bmatrix}, \quad R_{MHE} = \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix} \quad (\text{B.9})$$

NMHE:

Estimation error mean h1: 6.93e-09

Estimation error mean h2: 6.95e-09

Estimation error mean u_PA001: 4.14e-03

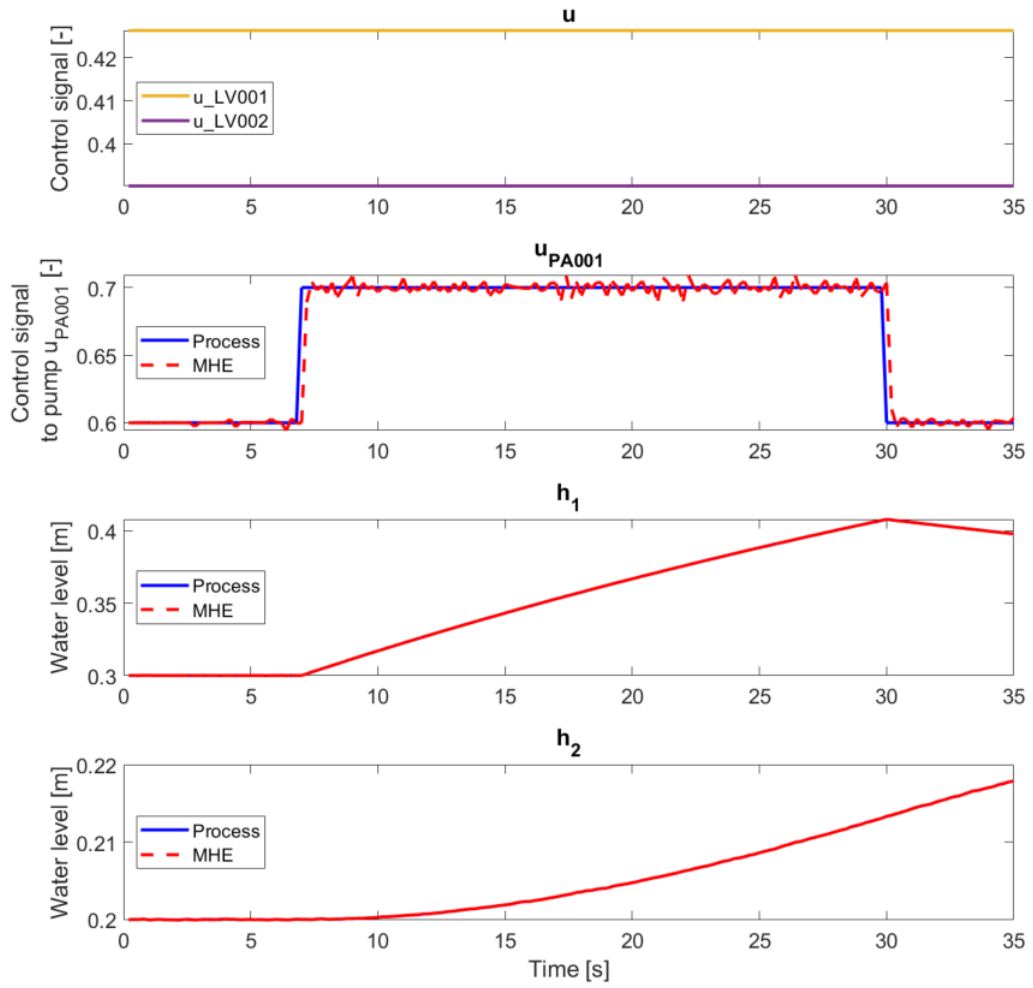


Figure B.12: NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. *GlobalSearch* method using *sqp* algorithm with the horizon length $N = 10$.

$$Q_{MHE} = \begin{bmatrix} 10^{-2} & 0 & 0 \\ 0 & 10^{-2} & 0 \\ 0 & 0 & 10^{-4} \end{bmatrix}, \quad R_{MHE} = \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix} \quad (\text{B.10})$$

NMHE :

Estimation error mean h1: 4.76e-07

Estimation error mean h2: 5.60e-07

Estimation error mean u_PA001: 3.92e-03

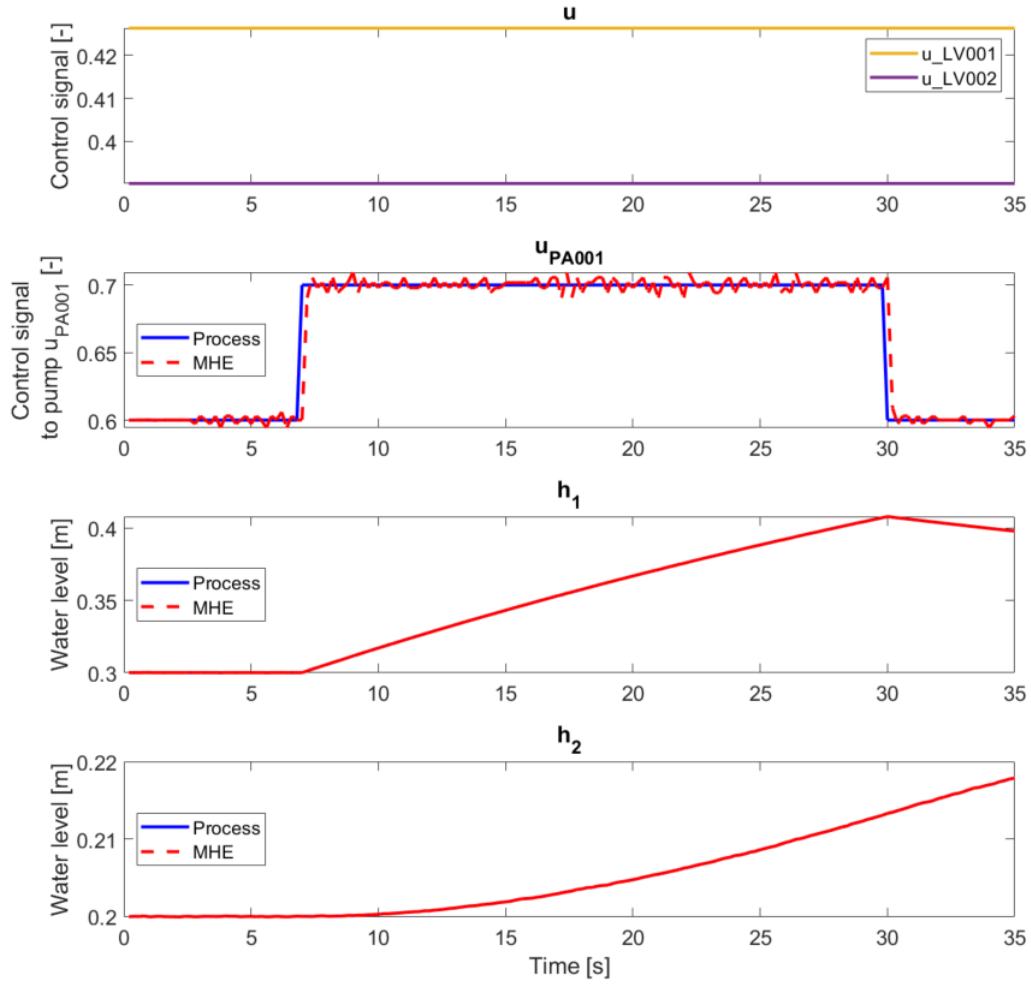


Figure B.13: NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-9}$. *PatternSearch* method using *sqp* algorithm with the horizon length $N = 10$.

$$Q_{MHE} = \begin{bmatrix} 10^{-2} & 0 & 0 \\ 0 & 10^{-2} & 0 \\ 0 & 0 & 10^{-4} \end{bmatrix}, \quad R_{MHE} = \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix} \quad (\text{B.11})$$

NMHE:

Estimation error mean h1: 4.69e-07

Estimation error mean h2: 4.66e-07

Estimation error mean u_PA001: 4.22e-03

B.2.3 LMHE and NMHE (increased measurement noise)

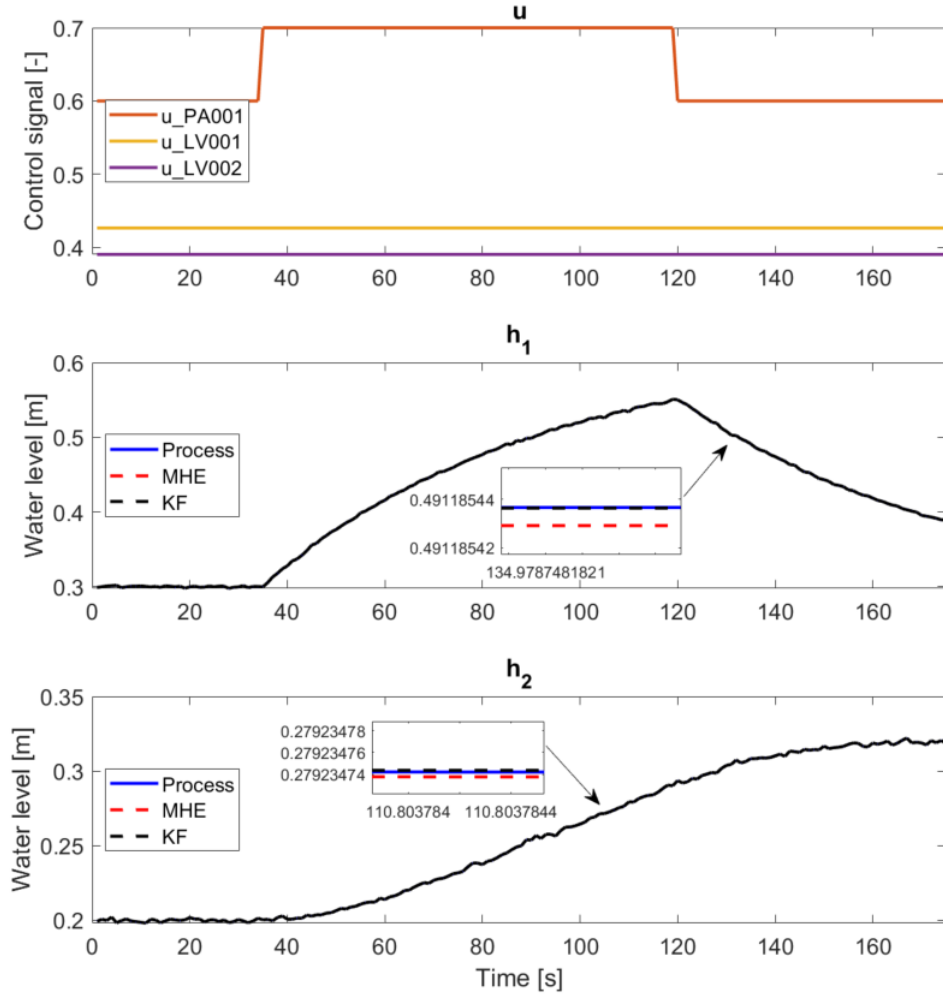


Figure B.14: LMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \\
 R_{MHE} = R_{EKF} &= \begin{bmatrix} 10^{-6} & 0 \\ 0 & 10^{-6} \end{bmatrix}
 \end{aligned} \tag{B.12}$$

LMHE :

Estimation error mean h1: 6.96e-09

Estimation error mean h2: 7.07e-09

KF :

Estimation error mean h1: 1.23e-09

Estimation error mean h2: 1.09e-09

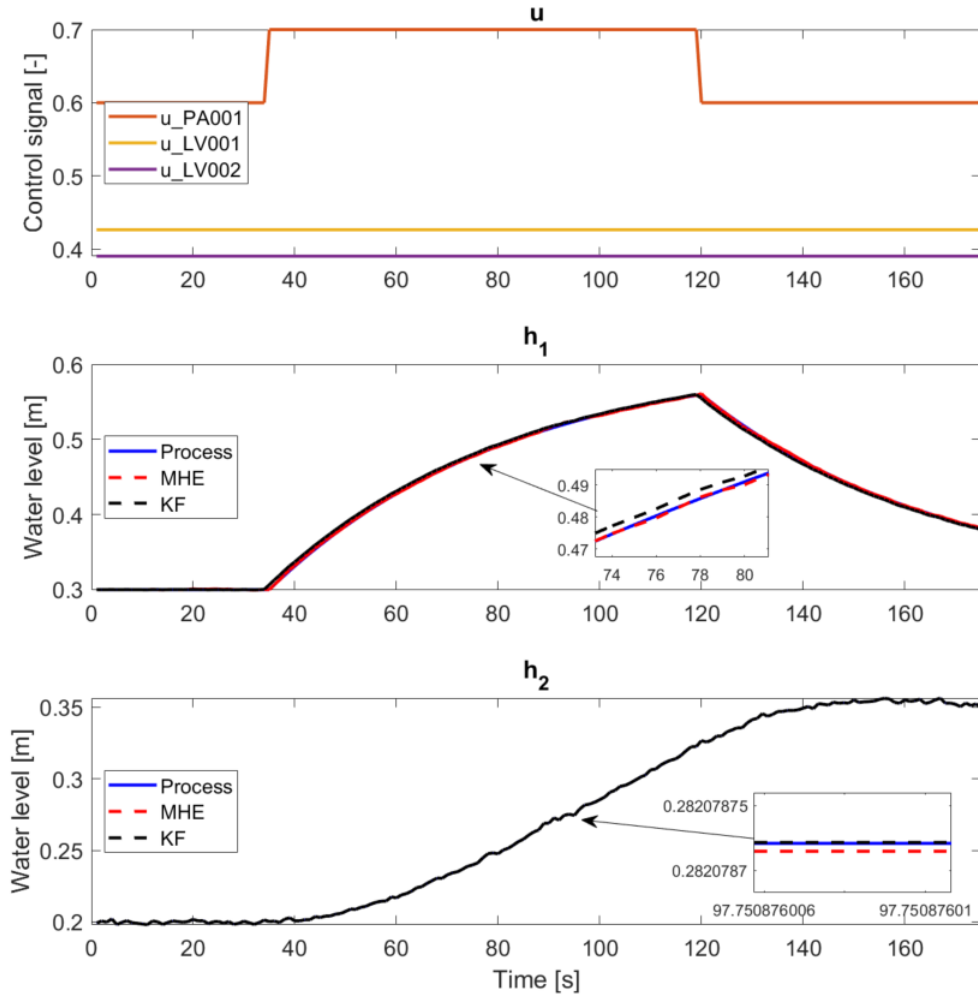


Figure B.15: LMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 9 \cdot 10^{-1} & 0 \\ 0 & 1 \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 10^1 & 0 \\ 0 & 1 \end{bmatrix}, \\
 R_{MHE} &= R_{KF} = 10^{-6}
 \end{aligned}
 \tag{B.13}$$

LMHE:

Estimation error mean h1: 5.08e-04

Estimation error mean h2: 7.92e-09

KF:

Estimation error mean h1: 2.38e-03

Estimation error mean h2: 1.09e-09

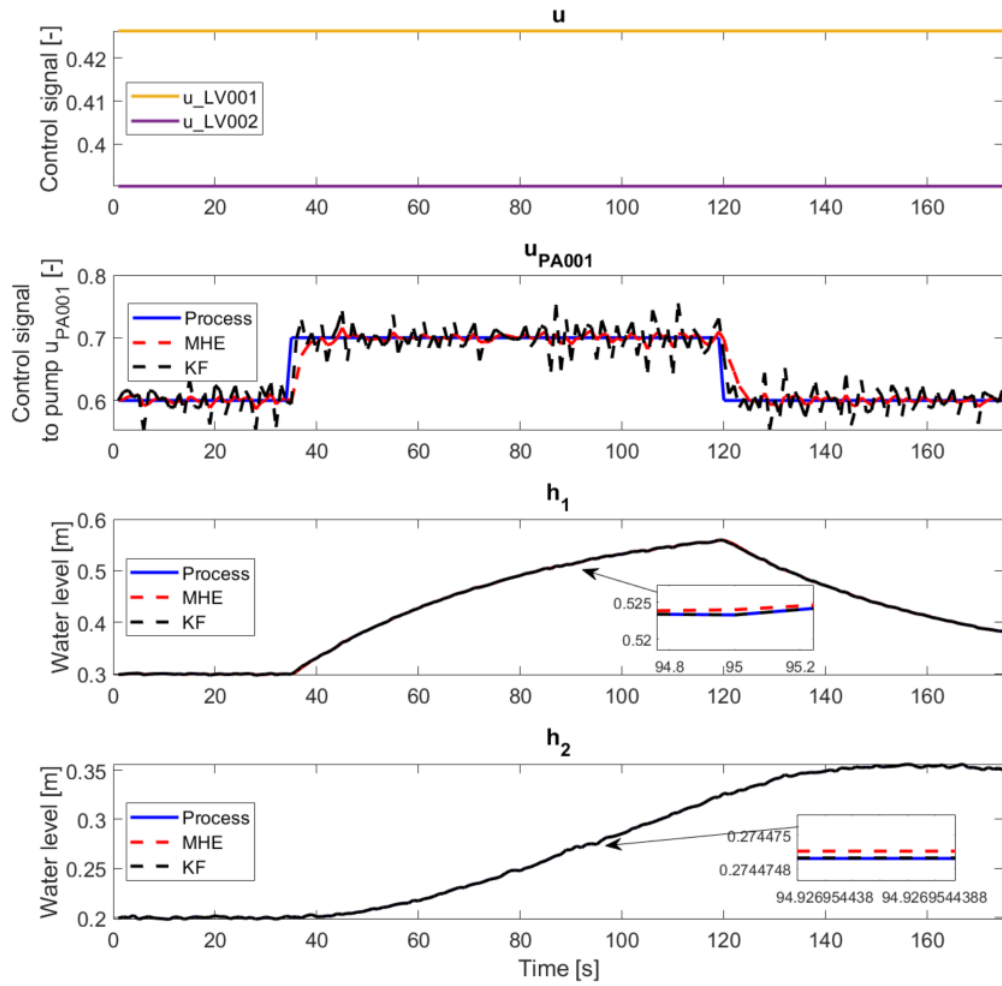


Figure B.16: LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^{-6} & 0 & 0 \\ 0 & 10^{-1} & 0 \\ 0 & 0 & 10^{-4} \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^7 \end{bmatrix}, \\
 R_{MHE} = R_{KF} &= \begin{bmatrix} 10^{-6} & 0 \\ 0 & 10^{-6} \end{bmatrix}
 \end{aligned} \tag{B.14}$$

LMHE :

Estimation error mean h1: 3.59e-04
 Estimation error mean h2: 2.18e-08
 Estimation error mean u_PA001: 7.05e-03

KF :

Estimation error mean h1: 6.35e-12
 Estimation error mean h2: 1.09e-09
 Estimation error mean u_PA001: 1.88e-02

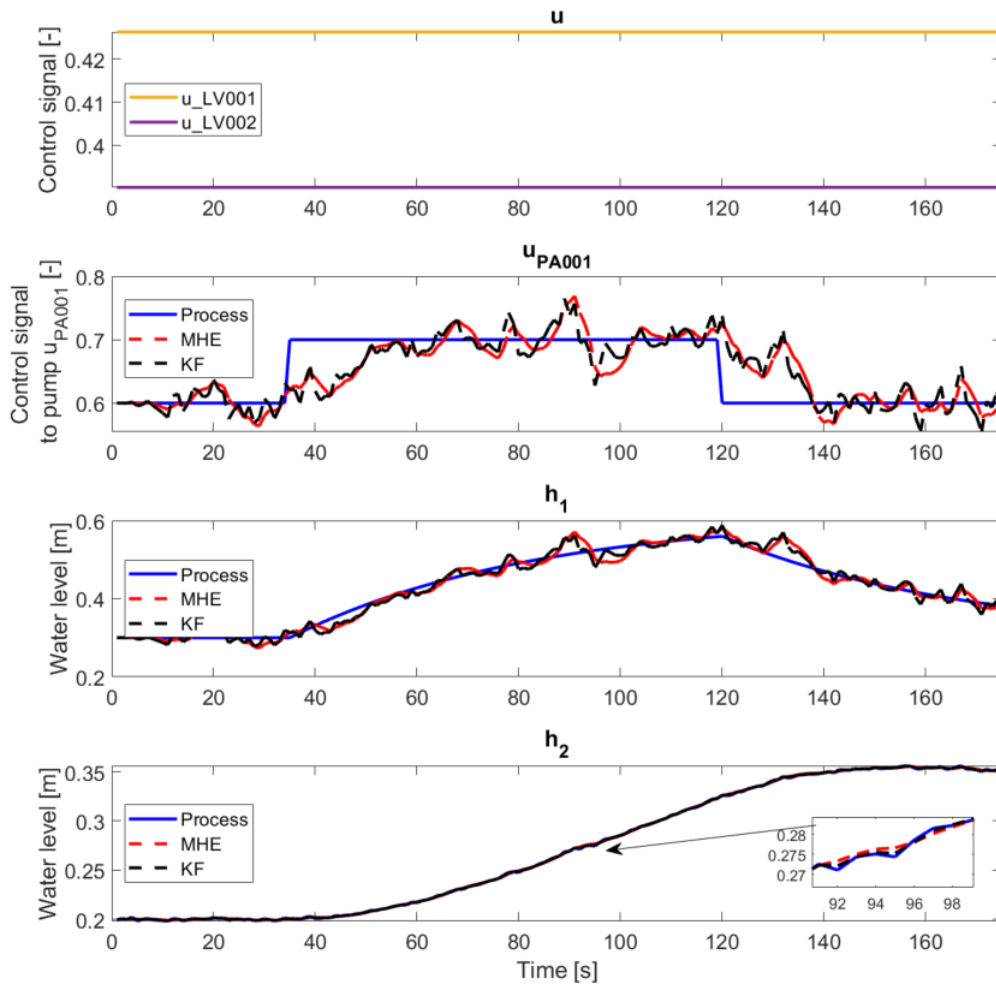


Figure B.17: LMHE Case 4: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^{-5} & 0 & 0 \\ 0 & 10^{-7} & 0 \\ 0 & 0 & 10^{-4} \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 10^{-4} & 0 & 0 \\ 0 & 10^{-6} & 0 \\ 0 & 0 & 10^{-3} \end{bmatrix}, \\
 R_{MHE} &= R_{KF} = 10^{-6}
 \end{aligned} \tag{B.15}$$

LMHE:

Estimation error mean h1: 1.27e-02

Estimation error mean h2: 6.36e-04

Estimation error mean u_PA001: 1.88e-02

KF:

Estimation error mean h1: 1.29e-02

Estimation error mean h2: 3.23e-04

Estimation error mean u_PA001: 2.61e-02

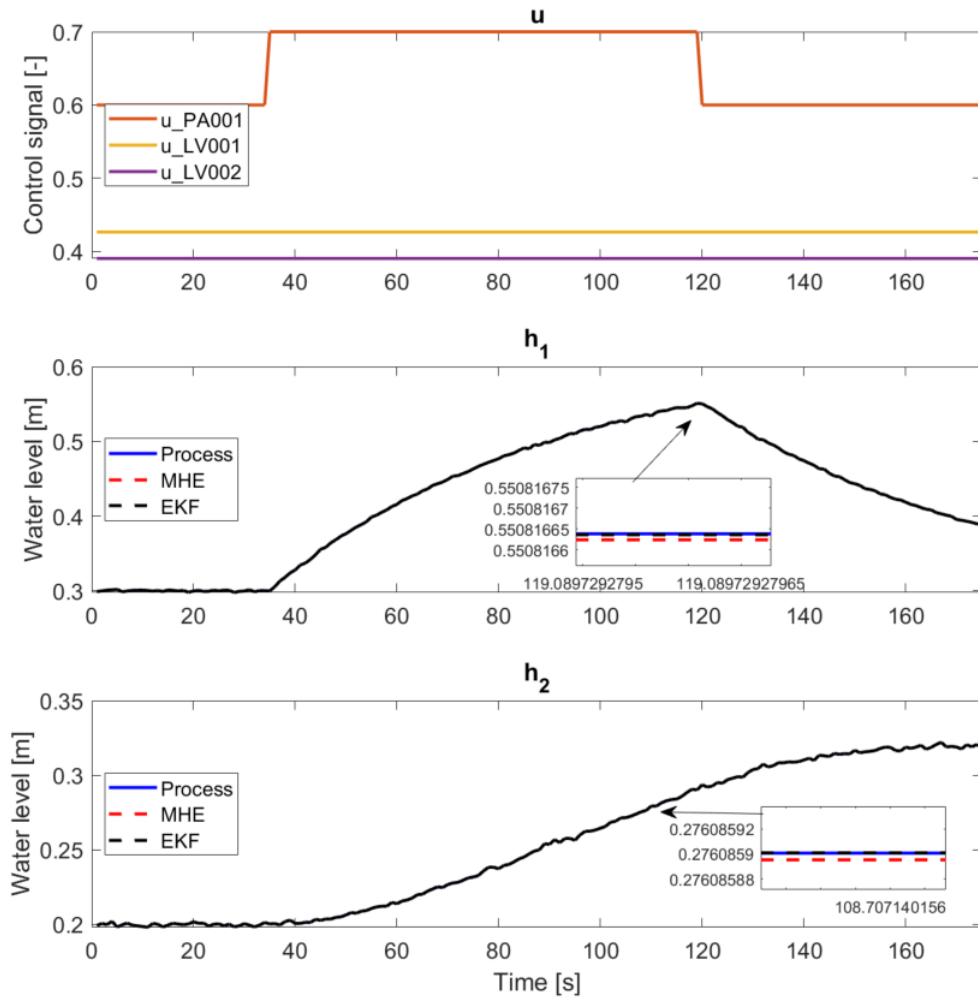


Figure B.18: NMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, & Q_{EKF} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \\
 R_{MHE} = R_{EKF} &= \begin{bmatrix} 10^{-6} & 0 \\ 0 & 10^{-6} \end{bmatrix}
 \end{aligned} \tag{B.16}$$

NMHE :

Estimation error mean h1: 6.49e-07

Estimation error mean h2: 5.63e-07

EKF :

Estimation error mean h1: 1.22e-09

Estimation error mean h2: 1.09e-09

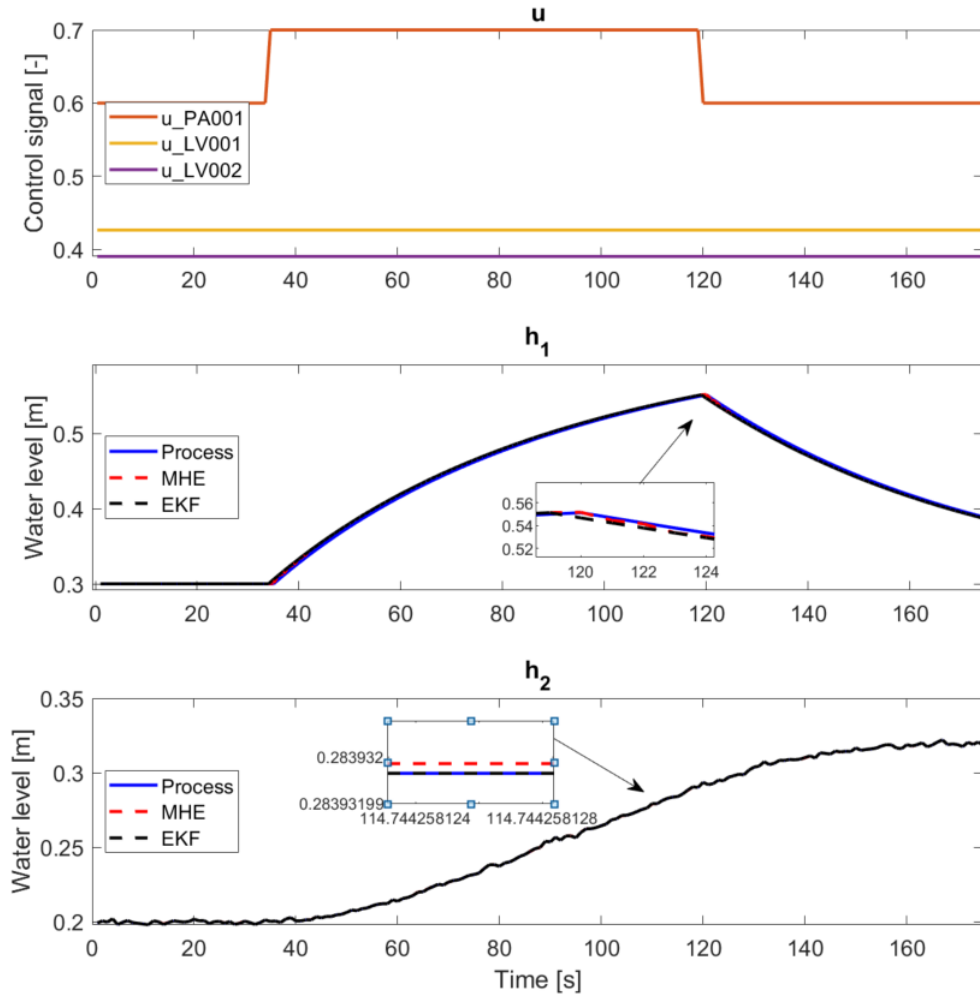


Figure B.19: NMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.

$$Q_{MHE} = \begin{bmatrix} 9 \cdot 10^2 & 0 \\ 0 & 10^6 \end{bmatrix}, \quad Q_{EKF} = \begin{bmatrix} 10^{-1} & 0 \\ 0 & 10^2 \end{bmatrix}, \quad (B.17)$$

$$R_{MHE} = R_{EKF} = 10^{-6}$$

NMHE:

Estimation error mean h1: 2.23e-03

Estimation error mean h2: 1.24e-05

EKF:

Estimation error mean h1: 2.37e-03

Estimation error mean h2: 1.09e-11

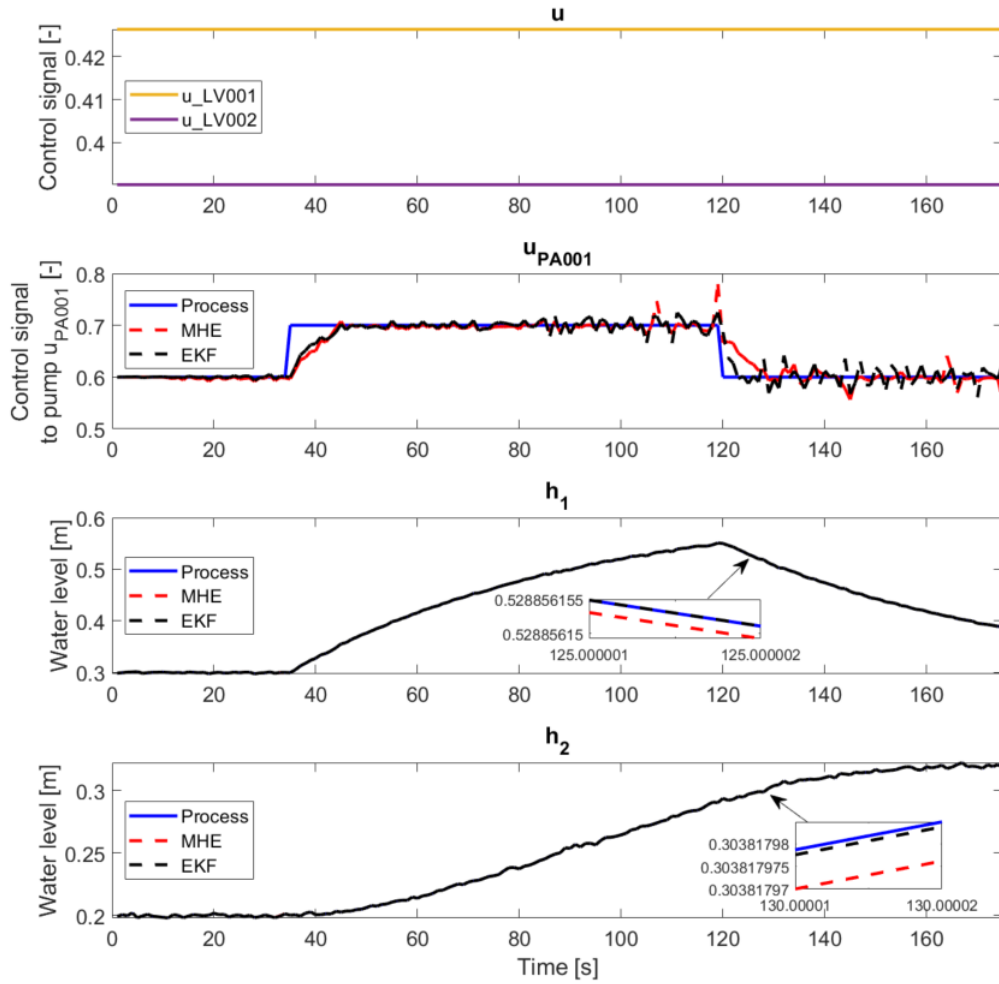


Figure B.20: NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurements h_1 and h_2 . Simulated measurement noise, $\sigma^2 = 10^{-6}$. Sample time 1s.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 10^1 & 0 & 0 \\ 0 & 10^1 & 0 \\ 0 & 0 & 10^{-5} \end{bmatrix}, & Q_{EKF} &= \begin{bmatrix} 10^2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^{-1} \end{bmatrix}, \\
 R_{MHE} = R_{EKF} &= \begin{bmatrix} 10^{-6} & 0 \\ 0 & 10^{-6} \end{bmatrix}
 \end{aligned} \tag{B.18}$$

NMHE :

Estimation error mean h1: 1.32e-06
 Estimation error mean h2: 1.58e-06
 Estimation error mean u_PA001: 1.05e-02

EKF :

Estimation error mean h1: 1.68e-11
 Estimation error mean h2: 1.09e-09
 Estimation error mean u_PA001: 1.18e-02

B.3 Real Data Estimation Results

B.3.1 LMHE

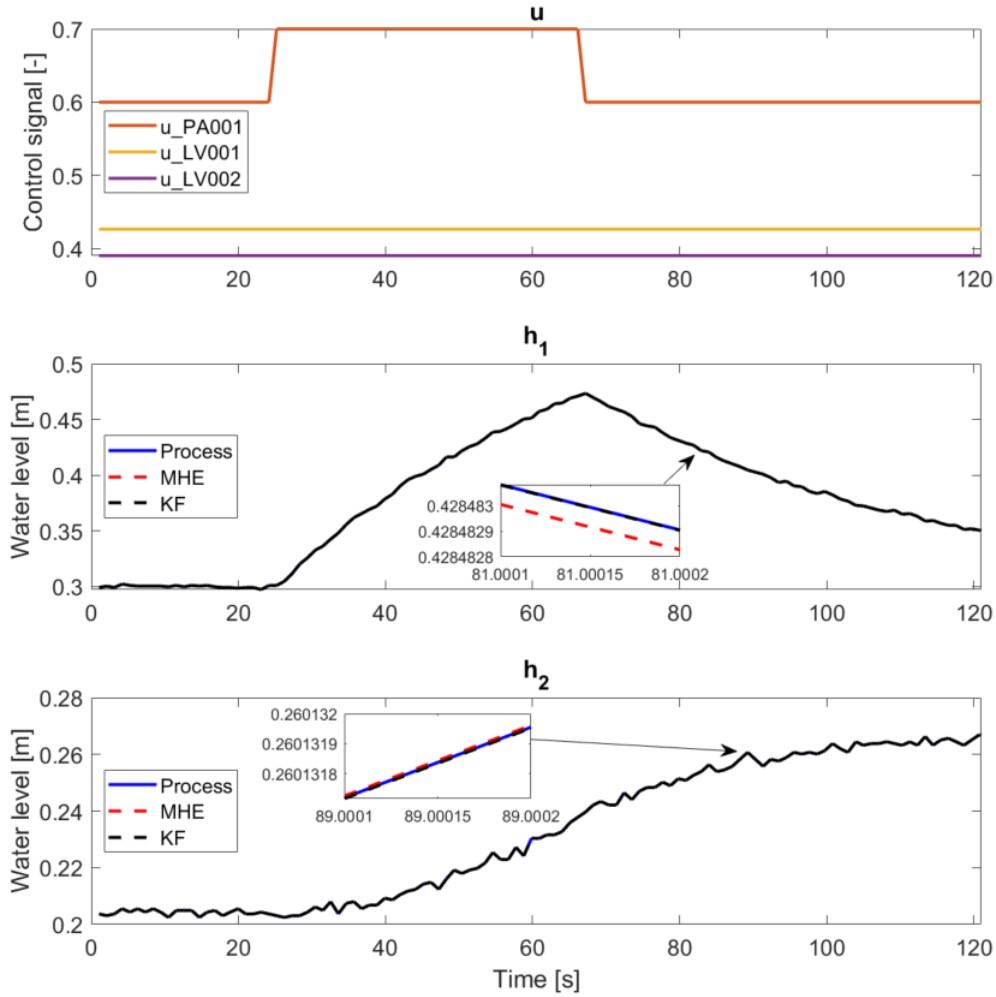


Figure B.21: LMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . *Sqp* algorithm with the horizon length $N = 10$. Dataset *data1.mat*.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, & Q_{KF} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \\
 R_{MHE} = R_{KF} &= \begin{bmatrix} 2.8 \cdot 10^{-6} & 0 \\ 0 & 1.6 \cdot 10^{-6} \end{bmatrix}
 \end{aligned} \tag{B.19}$$

LMHE:

Estimation error mean h1: 1.78e-08

Estimation error mean h2: 1.11e-08

KF:

Estimation error mean h1: 3.94e-09

Estimation error mean h2: 2.34e-09

B.3.2 NMHE

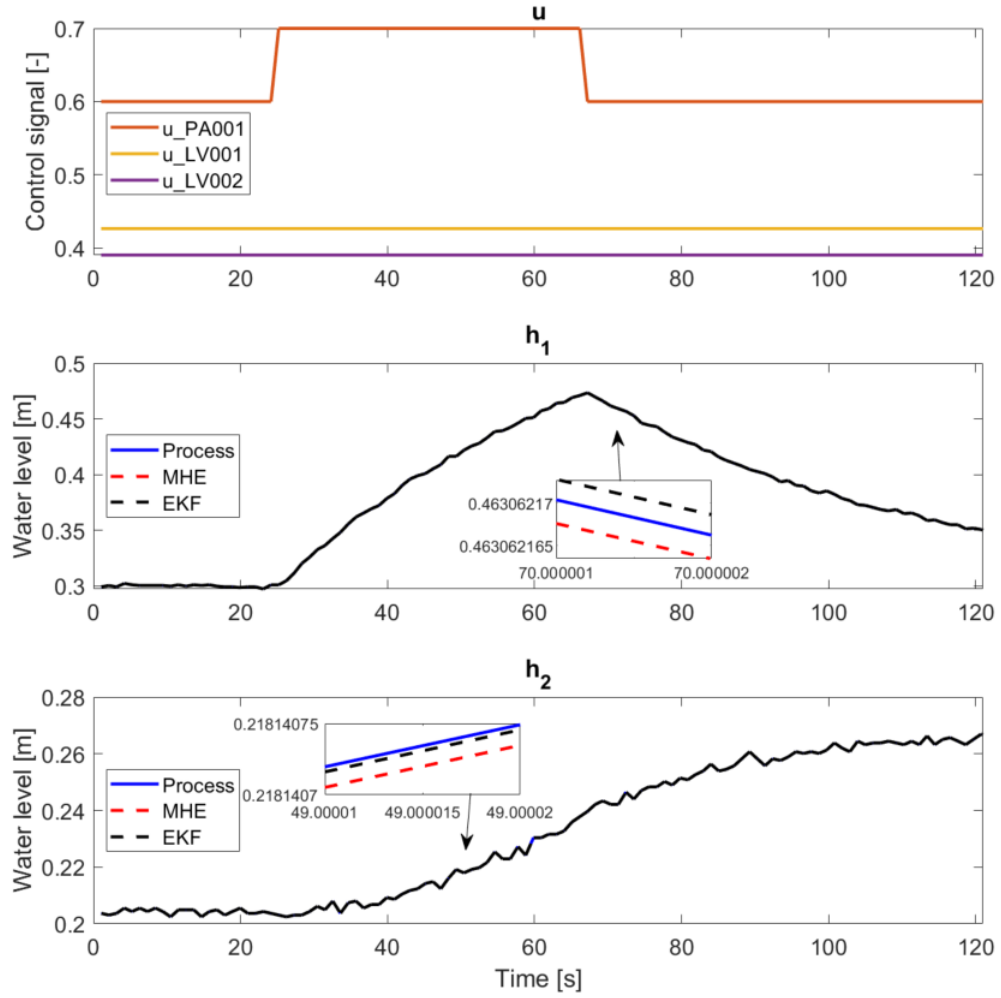


Figure B.22: NMHE Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . *Sqp* algorithm with the horizon length $N = 10$. Dataset *data1.mat*.

$$\begin{aligned}
 Q_{MHE} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, & Q_{EKF} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \\
 R_{MHE} = R_{EKF} &= \begin{bmatrix} 2.8 \cdot 10^{-6} & 0 \\ 0 & 1.6 \cdot 10^{-6} \end{bmatrix}
 \end{aligned} \tag{B.20}$$

LMHE:

Estimation error mean h1: 8.78e-09

Estimation error mean h2: 7.75e-09

EKF:

Estimation error mean h1: 3.95e-09

Estimation error mean h2: 2.34e-09

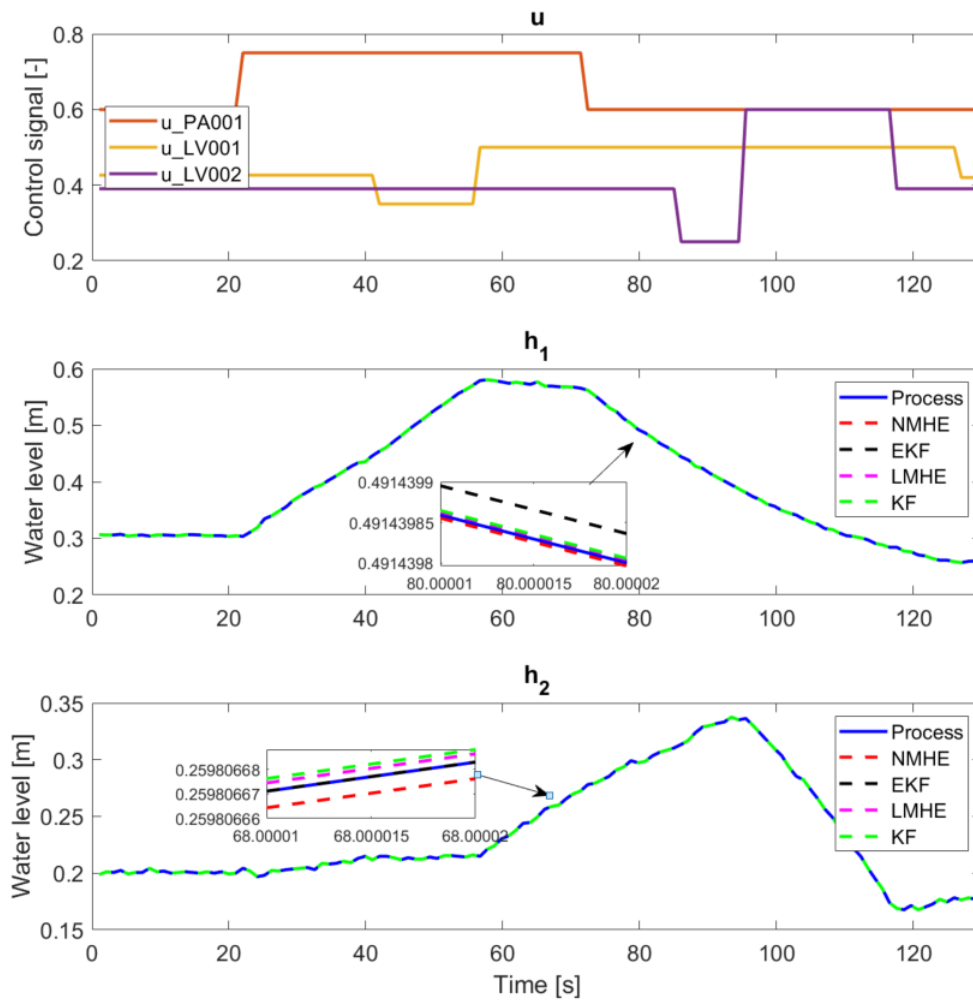


Figure B.23: Case 1: Estimation of h_1 and h_2 given the measurements h_1 and h_2 . Dataset *data2.mat*.

B.3.3 On-line Results

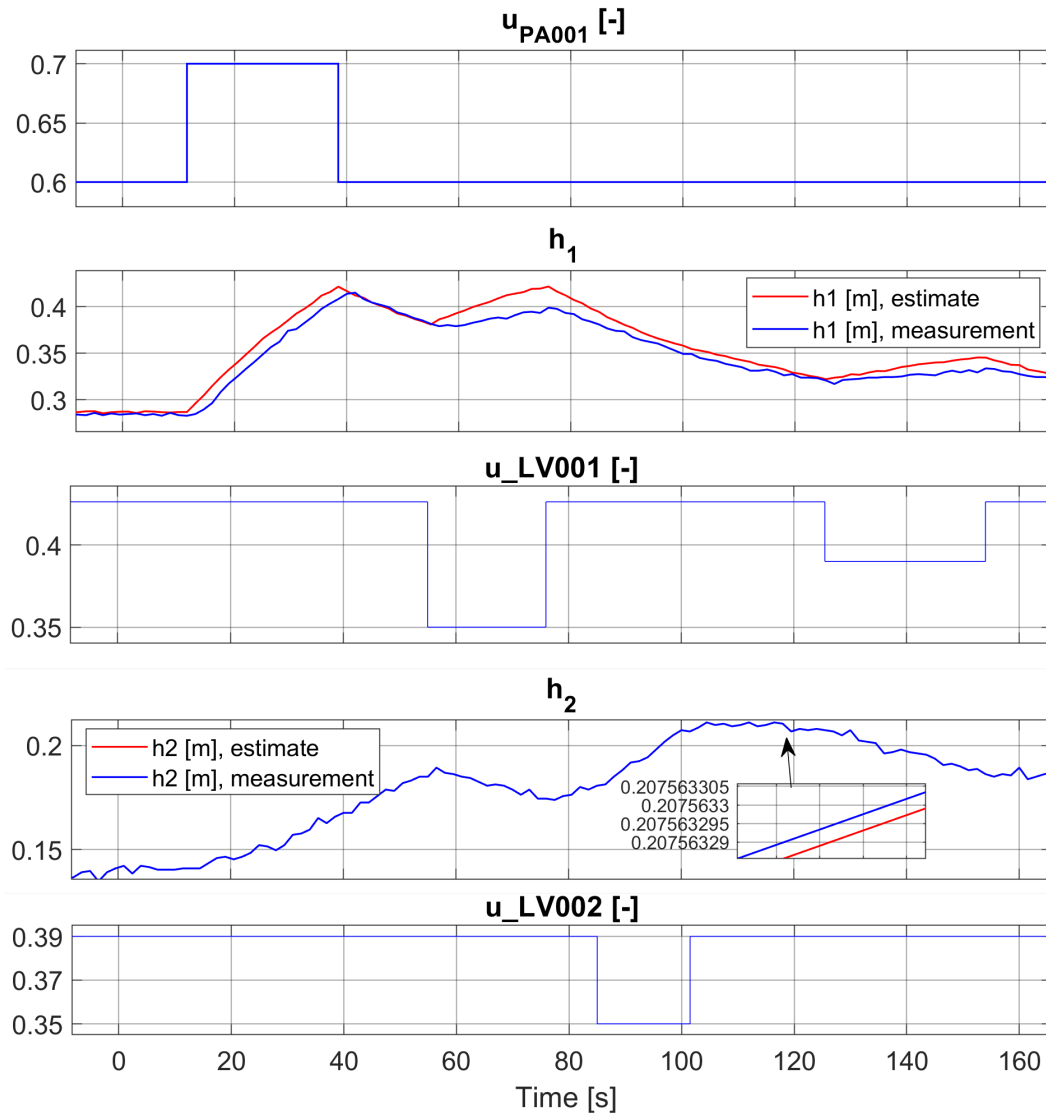


Figure B.24: LMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . Sqp algorithm with the horizon length $N = 10$.

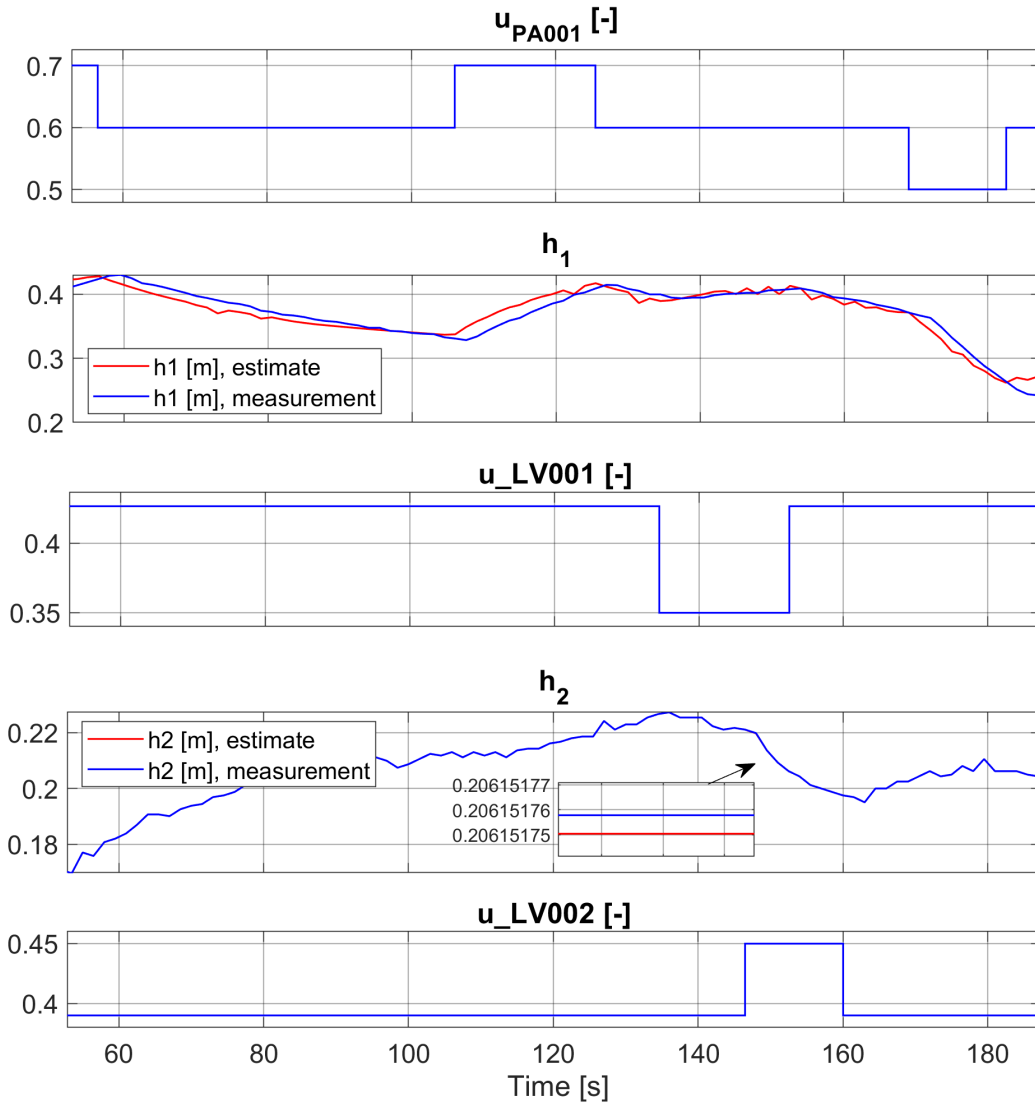


Figure B.25: NMHE Case 2: Estimation of h_1 and h_2 given the measurement h_2 . *Sqp* algorithm with the horizon length $N = 10$.

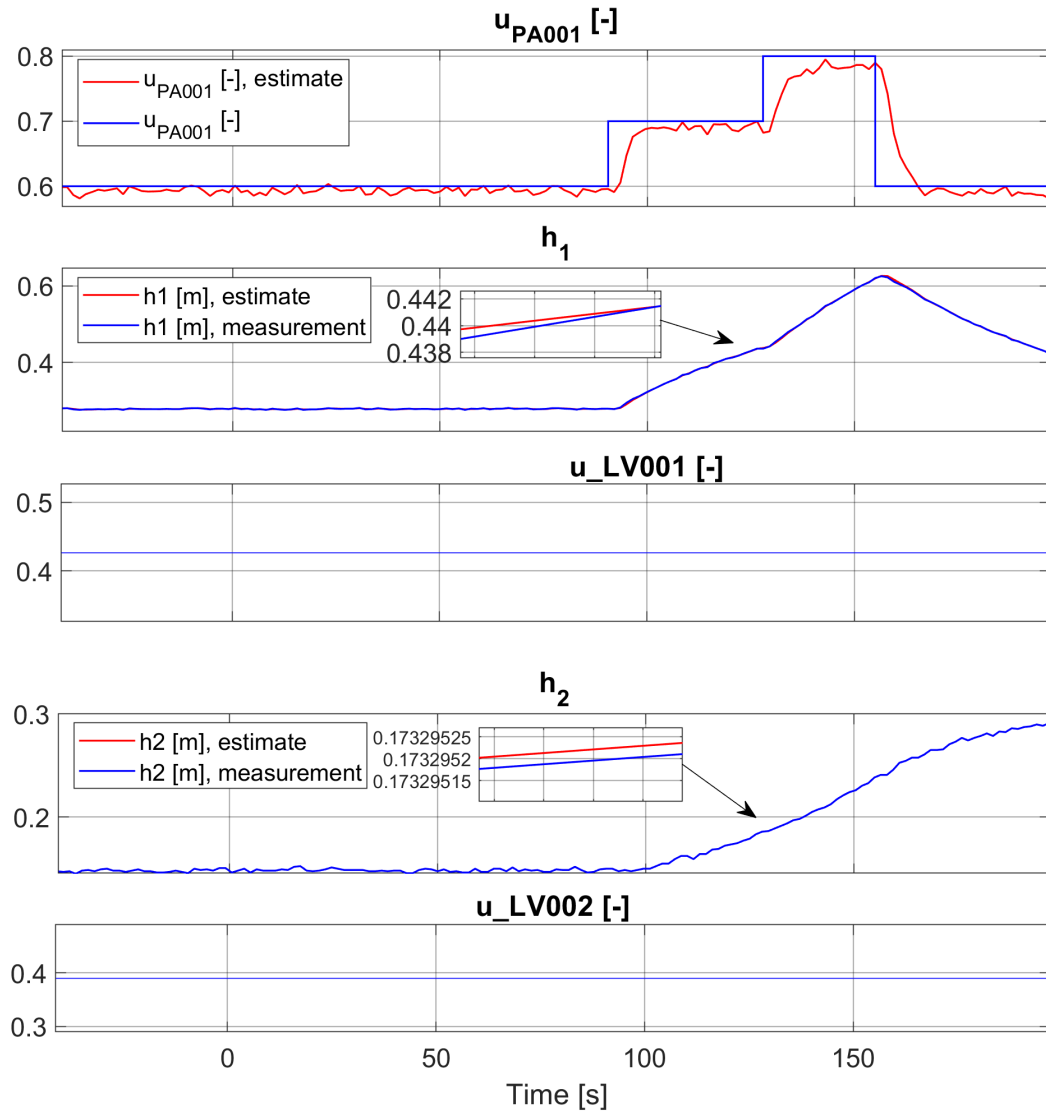


Figure B.26: LMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . *Sqp* algorithm with the horizon length $N = 10$.

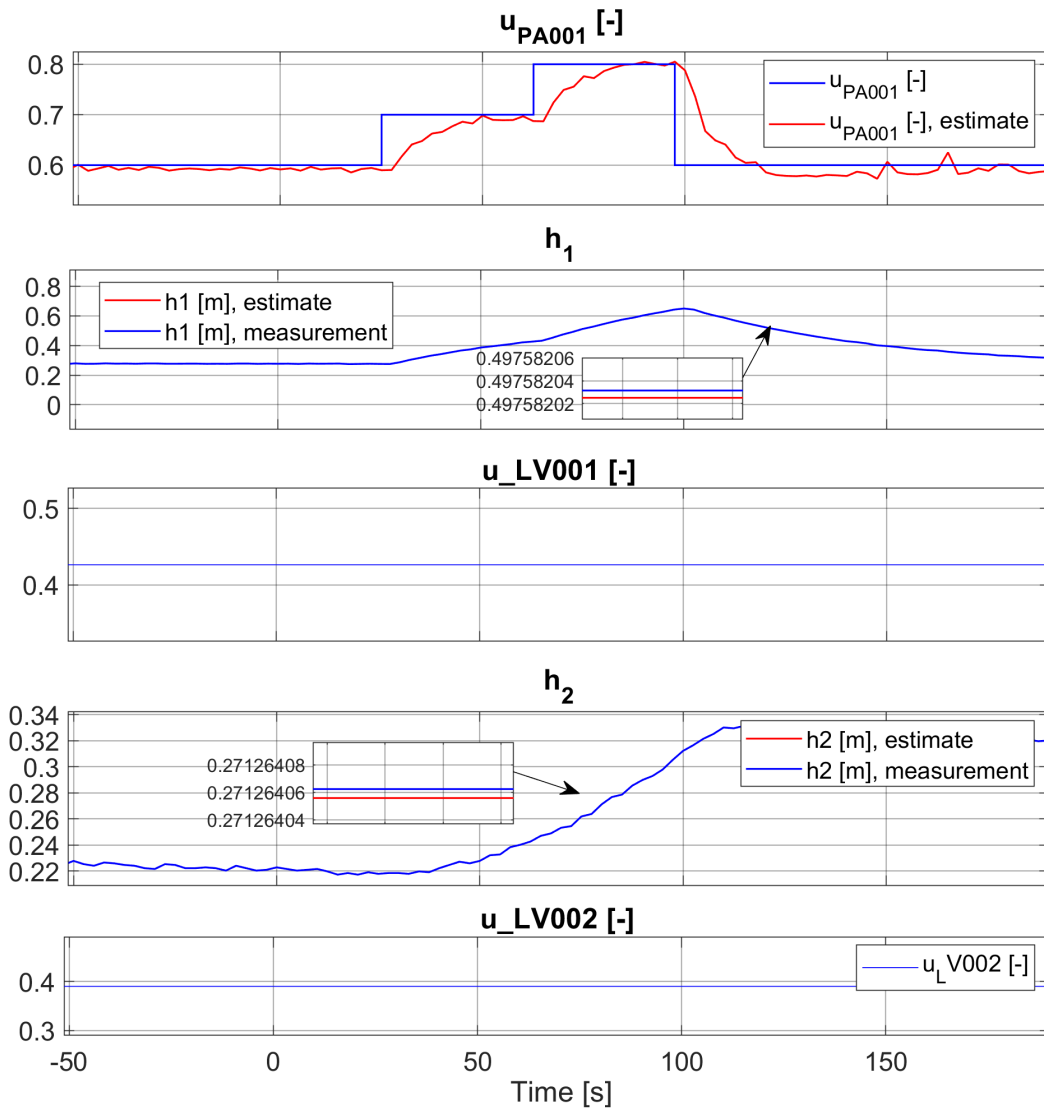


Figure B.27: NMHE Case 3: Estimation of h_1 , h_2 and u_{PA001} given the measurement h_1 and h_2 . *Sqp* algorithm with the horizon length $N = 10$.

Appendix C

Matlab Code

C.1 LMHE.m

```
clear all; close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Master Thesis:
% Moving Horizon Estimation for the Two-tank System by Greta Bekeryte.
% May, 2023
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
run parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Operating point for linearization
h1_0 = 0.3;
h2_0 = 0.2;
u_PA001_0 = 0.6;
n = 3; % number of states [3/2]
m = 2; % number of measurements available [2/1]
%MHE parameters
N = 10; %Horizon length
% Simulation parameters
sim_model = 'L'; % Linear or nonlinear simulation model? ['L'/'N'/'data']
sys_noise = 'Y'; % Simulate system with noise? ['Y'/'N']
Kalman_filter = 'Y'; % Estimation using Kalman filter for comparison ['Y'/'N']
Ts = 1; % Samplings time. For dataset1 and dataset2: 1.05

if strcmp('data',sim_model)
    load('data1.mat ')
    h1_s = ScopeData.signals(2).values(:,1);
    h2_s = ScopeData.signals(4).values(:,1);
    u_LV001_s = ScopeData.signals(3).values;
    u_LV002_s = ScopeData.signals(5).values;
```

```

    u_PA001_s = ScopeData.signals(1).values;
    t_time = ScopeData.time;
    t_length = length(t_time);
    t_storage = zeros(1,t_length);
    t_start = 0;
    t_end = t_length;
    u_PA001_0 = u_PA001_s(1);
    %initial water levels in tank 1 and tank 2
    h1_0 = h1_s(1); %h1_min;
    h2_0 = h2_s(1); %h2_min;
    % Init
    h1_init = h1_0; %operating point
    h2_init = h2_0;
    if n == 3
        u_PA001_init = u_PA001_0;
        u_PA001_k = u_PA001_init;
    end
end
if strcmp('L',sim_model) || strcmp('N',sim_model)
    t_start = 0;
    t_end = 35*5;
    step_up = 7*5;
    step_down = 30*4;
    step_size = 0.1;
    step_in = 'u_PA001'; % ['u_LV001'/'u_LV002'/'u_PA001']
    %Simulation time parameters
    t_length = t_end/Ts + 1;
    t_storage = zeros(1,t_length);
    % Init
    h1_init = h1_0; %operating point
    h2_init = h2_0;
    if n == 3
        u_PA001_init = u_PA001_0;
        u_PA001_k = u_PA001_init;
    end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% linearization
[A, B, C, D, u_LV001_0, u_LV002_0] = linearization(h1_0,h2_0,u_PA001_0,n,m);
%Parameter structure
parameter_struct.N = N;
parameter_struct.n = n;
parameter_struct.m = m;
parameter_struct.Ts = Ts;
parameter_struct.A = A;
parameter_struct.B = B;
parameter_struct.C = C;

```

```
parameter_struct.D = D;
%Set points
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;

% Process disturbance MHE
Q_w1 = 1e-6;
Q_w2 = 1e-1;
Q_w3 = 1e-4;
% Measurement noise MHE
R_v1 = 1e-6;%2.8e-6;
R_v2 = 1e-6;%1.6e-6;

if n == 2
    Q = diag([Q_w1, Q_w2]);
elseif n == 3
    Q = diag([Q_w1, Q_w2, Q_w3]);
end
if m == 2
    R = diag([R_v1, R_v2]);
elseif m == 1
    R = R_v2;
end
parameter_struct.Q = Q;
parameter_struct.R = R;

rng(1)
noise = randn(5,t_length);
if sys_noise == 'Y'
    % Process disturbance used in simulation
    cov_w1 = 0;
    cov_w2 = 0;
    cov_w3 = 0;
    % Measurement noise used in simulation
    cov_v1 = 1e-6;
    cov_v2 = 1e-6;
elseif sys_noise == 'N'
    % Process disturbance used in simulation
    cov_w1 = 0;
    cov_w2 = 0;
    cov_w3 = 0;
    % Measurement noise used in simulation
    cov_v1 = 0;
    cov_v2 = 0;
end
```

```
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

%Simulation variable storage arrays
%Control inputs
u_LV001_arr = zeros(1,t_length);
u_LV002_arr = zeros(1,t_length);
%Disturbance or state parameter
u_PA001_arr = zeros(1,t_length);
%State variables
h1_arr = zeros(1,t_length);
h2_arr = zeros(1,t_length);
%Outputs
if m == 2
    h1_meas_arr = zeros(1,t_length);
end
h2_meas_arr = zeros(1,t_length);
% MHE estimates
h1_hat_arr = zeros(1,t_length);
h2_hat_arr = zeros(1,t_length);
if n == 3
    u_PA001_hat_arr = zeros(1,t_length);
end

h1_k = h1_init;
h2_k = h2_init;

% Init MHE
horizon = zeros(1,N);
h1_hat_horizon = horizon;
h2_hat_horizon = horizon;
if n == 3
    u_PA001_hat_init = u_PA001_0;
    u_PA001_hat_horizon = horizon;
    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
```



```

if m == 2
    h1_meas_horizon = horizon;
end
h2_meas_horizon = horizon;

% Lower and upper bound constraints for whole horizon
h1_max_arr = zeros(1,N) + h1_max;
h1_min_arr = zeros(1,N) + h1_min;
h2_max_arr = zeros(1,N) + h2_max;
h2_min_arr = zeros(1,N) + h2_min;
if n == 3
    u_PA001_max_arr = zeros(1,N) + 1; % From pump characteristic
    u_PA001_min_arr = zeros(1,N) + 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

%Estimate error covariance matrix matrix used in arrival cost storage
P_k_N_arr = cell(1,t_length);
P_k_N_arr{1,t_length} = [];
if n == 3
    P_init = diag([0.00001, 0.00001, 0.00001]); %initial P covariance matrix
    x_pred_k_N = [h1_init; h2_init; u_PA001_init];
elseif n == 2
    P_init = diag([0.001, 0.001]); %initial P covariance matrix
    x_pred_k_N = [h1_init; h2_init];
end
P_previous = P_init;

%%%%%%%%%%%%%% Kalman filter init
A_disc = eye(size(A)) + Ts*A; % Discrete-time matrix A (Forward Euler)
parameter_struct.A_disc = A_disc;
if Kalman_filter == 'Y'
    if n == 2
        K_x_hat = [h1_init; h2_init];
    elseif n == 3
        K_x_hat = [h1_init; h2_init; u_PA001_init];
    end
    % storage of Kalman estimates
    K_h1_hat_arr = zeros(1,t_length);
    K_h2_hat_arr = zeros(1,t_length);
    if n == 3
        K_u_PA001_hat_arr = zeros(1,t_length);
    end
    K_P_hat = P_init;

```

```

if n == 2
    K_Q = diag([1, 1]);
elseif n == 3
    K_Q = diag([1, 1, 1e7]);
end
if m == 2
    K_R = diag([1e-6, 1e-6]);
elseif m == 1
    K_R = 1e-6;
end
end

for k = 1:t_length
    t_k = k*Ts;

    if strcmp('L',sim_model)
        %Simulation of control inputs
        if t_k < step_up
            u_LV001_k = u_LV001_0;
            u_LV002_k = u_LV002_0;
            u_PA001_k = u_PA001_0;
        end
        if t_k >= step_up
            if strcmp('u_LV001',step_in)
                u_LV001_k = u_LV001_0 + step_size;
            elseif strcmp('u_LV002',step_in)
                u_LV002_k = u_LV002_0 + step_size;
            elseif strcmp('u_PA001',step_in)
                u_PA001_k = u_PA001_0 + step_size;
            end
        end
        if t_k >= step_down
            if strcmp('u_LV001',step_in)
                u_LV001_k = u_LV001_0;
            elseif strcmp('u_LV002',step_in)
                u_LV002_k = u_LV002_0;
            elseif strcmp('u_PA001',step_in)
                u_PA001_k = u_PA001_0;
            end
        end
    end
    %Disturbances and noise
    if n == 3
        w_k = [sqrt(cov_w1)*noise(1,k); sqrt(cov_w2)*noise(2,k);...
              sqrt(cov_w3)*noise(3,k)];
    elseif n == 2
        w_k = [sqrt(cov_w1)*noise(1,k); sqrt(cov_w2)*noise(2,k)];
    end
end

```

```

if m == 2
    v_k = [sqrt(cov_v1)*noise(4,k); sqrt(cov_v2)*noise(5,k)];
elseif m == 1
    v_k = sqrt(cov_v2)*noise(5,k);
end

if n == 3
    u_k = [u_LV001_k; u_LV002_k];
    x_k = [h1_k; h2_k; u_PA001_k];
elseif n == 2
    % when u_PA001_k is known disturbance
    u_k = [u_LV001_k; u_LV002_k; u_PA001_k];
    x_k = [h1_k; h2_k];
end

%%%%%%%%%%%%%% Linear Process model: derivatives and integration
if sys_noise == 'Y'
    if n == 3
        dh_dt_k = A*(x_k - [h1_0; h2_0; u_PA001_0]) + B*(u_k -...
            [u_LV001_0; u_LV002_0]); % + w_k;
    elseif n == 2
        dh_dt_k = A*(x_k - [h1_0; h2_0]) + B*(u_k -...
            [u_LV001_0; u_LV002_0; u_PA001_0]); % + w_k;
    end
elseif sys_noise == 'N'
    if n == 3
        dh_dt_k = A*(x_k - [h1_0; h2_0; u_PA001_0]) + B*(u_k...
            -[u_LV001_0; u_LV002_0]);
    elseif n == 2
        dh_dt_k = A*(x_k - [h1_0; h2_0]) + B*(u_k -...
            [u_LV001_0; u_LV002_0; u_PA001_0]);
    end
end

x_k_ny = Ts*dh_dt_k + x_k; %Euler integration
%Physical constraints
if x_k_ny(1) < h1_min
    x_k_ny(1) = h1_min;
end
if x_k_ny(2) < h2_min
    x_k_ny(2) = h2_min;
end
if x_k_ny(1) > h1_max
    x_k_ny(1) = h1_max;
end
if x_k_ny(2) > h2_max
    x_k_ny(2) = h2_max;
end
if n == 3
    if x_k_ny(3) < 0.45

```

```

        x_k_ny(3) = 0.45;
    end
    if x_k_ny(3) > 1
        x_k_ny(3) = 1;
    end
end
if sys_noise == 'Y'
    h_meas_k = C*x_k + D*0 + v_k;
elseif sys_noise == 'N'
    h_meas_k = C*x_k + D*0;
end
% Storing simulation data
t_storage(k) = t_k;
u_LV001_arr(k) = u_LV001_k;
u_LV002_arr(k) = u_LV002_k;
u_PA001_arr(k) = u_PA001_k;
h1_arr(k) = h1_k;
h2_arr(k) = h2_k;
if m == 2
    h1_meas_arr(k) = h_meas_k(1);
end
h2_meas_arr(k) = h_meas_k(2);

%Update for k+1
h1_k = x_k_ny(1);
h2_k = x_k_ny(2);
if n == 3
    u_PA001_k = x_k_ny(3);
end
%Update horizons
u_horizon = [u_horizon(:, 2:N), u_k]; %sliding horizon to the right
if m == 2
    h1_meas_horizon = [h1_meas_horizon(2:N), h_meas_k(1)];
end
h2_meas_horizon = [h2_meas_horizon(2:N), h_meas_k(2)];

if m == 2
    h_meas_horizon = [h1_meas_horizon; h2_meas_horizon];
elseif m == 1
    h_meas_horizon = h2_meas_horizon;
end
elseif strcmp('N',sim_model)
%Simulation of control inputs
if t_k < step_up
    u_LV001_k = u_LV001_0;
    u_LV002_k = u_LV002_0;
    u_PA001_k = u_PA001_0;
end

```

```

if t_k >= step_up
    if strcmp('u_LV001',step_in)
        u_LV001_k = u_LV001_0 + step_size;
    elseif strcmp('u_LV002',step_in)
        u_LV002_k = u_LV002_0 + step_size;
    elseif strcmp('u_PA001',step_in)
        u_PA001_k = u_PA001_0 + step_size;
    end
end
if t_k >= step_down
    if strcmp('u_LV001',step_in)
        u_LV001_k = u_LV001_0;
    elseif strcmp('u_LV002',step_in)
        u_LV002_k = u_LV002_0;
    elseif strcmp('u_PA001',step_in)
        u_PA001_k = u_PA001_0;
    end
end
%Disturbances and noise
if n == 3
    w_k = [sqrt(cov_w1)*noise(1,k); sqrt(cov_w2)*noise(2,k);...
           sqrt(cov_w3)*noise(3,k)];
elseif n == 2
    w_k = [sqrt(cov_w1)*noise(1,k); sqrt(cov_w2)*noise(2,k)];
end
if m == 2
    v_k = [sqrt(cov_v1)*noise(4,k); sqrt(cov_v2)*noise(5,k)];
elseif m == 1
    v_k = sqrt(cov_v2)*noise(5,k);
end

if n == 3
    u_k = [u_LV001_k; u_LV002_k];
    x_k = [h1_k; h2_k; u_PA001_k];
elseif n == 2
    % when u_PA001_k is known disturbancenn
    u_k = [u_LV001_k; u_LV002_k; u_PA001_k];
    x_k = [h1_k; h2_k];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Nonlinear Process model: derivatives and integration
f3_k = interp1(u_PA001,q_PA001, u_PA001_k);
f1_k = interp1(u_LV001,f_LV001,u_LV001_k);
f2_k = interp1(u_LV002,f_LV002,u_LV002_k);

dh1_dt_k = (1/A1)*(f3_k-((Kv_LV001*f1_k)/3600)*sqrt((rho*g*(h1_k+...
            h_LV001))/100000));
dh2_dt_k = (1/(0.07*h2_k+0.004))*(((Kv_LV001*f1_k)/3600)*...
            sqrt((rho*g*(h1_k+h_LV001))/100000)...

```

```

        -(Kv_LV002*f2_k)/3600*sqrt((rho*g*(h2_k+h_LV002))/100000));
if sys_noise == 'Y'
    h1_k_ny = Ts*dh1_dt_k + h1_k;% + w1_k;    %Euler integration
    h2_k_ny = Ts*dh2_dt_k + h2_k;% + w2_k;
elseif sys_noise == 'N'
    h1_k_ny = Ts*dh1_dt_k + h1_k;    %Euler integration
    h2_k_ny = Ts*dh2_dt_k + h2_k;
end
%Physical constraints
if h1_k_ny < h1_min
    h1_k_ny = h1_min;
end
if h2_k_ny < h2_min
    h2_k_ny = h2_min;
end
if h1_k_ny > h1_max
    h1_k_ny = h1_max;
end
if h2_k_ny > h2_max
    h2_k_ny = h2_max;
end

if sys_noise == 'Y'
    if m == 2
        h1_meas_k = h1_k + v_k(1);
        h2_meas_k = h2_k + v_k(2);
    elseif m == 1
        h2_meas_k = h2_k + v_k;
    end

elseif sys_noise == 'N'
    if m == 2
        h1_meas_k = h1_k;
    end
    h2_meas_k = h2_k;
end
%Physical constraints
if h2_meas_k < h2_min
    h2_meas_k = h2_min;
end
if h2_meas_k > h2_max
    h2_meas_k = h2_max;
end
if m == 2
    if h1_meas_k < h1_min
        h1_meas_k = h1_min;
    end
    if h1_meas_k > h1_max

```

```

        h1_meas_k = h1_max;
    end
end
% Storing simulation data
t_storage(k) = t_k;
u_LV001_arr(k) = u_LV001_k;
u_LV002_arr(k) = u_LV002_k;
u_PA001_arr(k) = u_PA001_k;
h1_arr(k) = h1_k;
h2_arr(k) = h2_k;

if m == 2
    h1_meas_arr(k) = h1_meas_k;
    h2_meas_arr(k) = h2_meas_k;
    h_meas_k = [h1_meas_k; h2_meas_k];
elseif m == 1
    h2_meas_arr(k) = h2_meas_k;
    h_meas_k = h2_meas_k;
end

%Update for k+1
h1_k = h1_k_ny;
h2_k = h2_k_ny;

%Update horizons
u_horizon = [u_horizon(:,2:N), u_k]; %sliding horizon to the right

h2_meas_horizon = [h2_meas_horizon(2:N), h2_meas_k];
if m == 2
    h1_meas_horizon = [h1_meas_horizon(2:N), h1_meas_k];
    h_meas_horizon = [h1_meas_horizon; h2_meas_horizon];
elseif m == 1
    h_meas_horizon = h2_meas_horizon;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp('data',sim_model)
% Storing simulation data
t_storage(k) = t_k;
u_LV001_k = u_LV001_s(k);
u_LV002_k = u_LV002_s(k);
u_PA001_k = u_PA001_s(k);

if n == 3
    u_k = [u_LV001_k; u_LV002_k];
elseif n == 2
    u_k = [u_LV001_k; u_LV002_k; u_PA001_k];
end
% Process derivatives and integration

```

```

f3_k = interp1(u_PA001,q_PA001, u_PA001_k);
f1_k = interp1(u_LV001,f_LV001,u_LV001_k);
f2_k = interp1(u_LV002,f_LV002,u_LV002_k);

h1_meas_k = h1_s(k);
h2_meas_k = h2_s(k);
u_LV001_arr(k) = u_LV001_k;
u_LV002_arr(k) = u_LV002_k;
u_PA001_arr(k) = u_PA001_k;
h1_arr(k) = h1_meas_k;
h2_arr(k) = h2_meas_k;

if m == 2
    h1_meas_arr(k) = h1_meas_k;
    h2_meas_arr(k) = h2_meas_k;
    h_meas_k = [h1_meas_k; h2_meas_k];
elseif m ==1
    h2_meas_arr(k) = h2_meas_k;
    h_meas_k = h2_meas_k;
end
%Update horizons
u_horizon = [u_horizon(:,2:N), u_k]; %sliding horizon to the right

h2_meas_horizon = [h2_meas_horizon(2:N), h2_meas_k];
if m == 2
    h1_meas_horizon = [h1_meas_horizon(2:N), h1_meas_k];
    h_meas_horizon = [h1_meas_horizon; h2_meas_horizon];
elseif m == 1
    h_meas_horizon = h2_meas_horizon;
end
end

%%%%%%%%%%%%%% Kalman filter
if Kalman_filter == 'Y'
    tic;
    [K_x_hat, K_P_hat] = KalmanFilter(K_x_hat,K_P_hat,h_meas_k,u_k,...
        parameter_struct,K_Q,K_R);

    %Storing
    K_h1_hat_arr(k) = K_x_hat(1);
    K_h2_hat_arr(k) = K_x_hat(2);
    if n == 3
        K_u_PA001_hat_arr(k) = K_x_hat(3);
    end
end

%%%%%%%%%%%%%%
if k <=N
    [x_hat, P_previous] = KalmanFilter(x_pred_k_N,P_previous,h_meas_k,u_k,...

```



```

        parameter_struct,Q,R);

%Storing
h1_hat_horizon(1,k) = x_hat(1);
h2_hat_horizon(1,k) = x_hat(2);
if n==3
    u_PA001_hat_horizon(1,k) = x_hat(3);
    x_hat_horizon = [h1_hat_horizon; h2_hat_horizon;...
                    u_PA001_hat_horizon];
else
    x_hat_horizon = [h1_hat_horizon; h2_hat_horizon];
end
% estimates for plot
h1_hat_arr(k) = x_hat(1);
h2_hat_arr(k) = x_hat(2);
if n == 3
    u_PA001_hat_arr(k) = x_hat(3);
end
P_k_N_arr{k} = P_previous;
x_pred_k_N(1) = x_hat(1);
x_pred_k_N(2) = x_hat(2);
if n == 3
    x_pred_k_N(3) = x_hat(3);
end
end

if k > N %enough samples for the first horizon

% Using model to predict h_k based on estimated state x_{k-1}
if n == 3
    M_dx_dt_k = A*(x_hat_horizon(:,N) - [h1_0; h2_0; u_PA001_0]) +...
                B*(u_k -[u_LV001_0; u_LV002_0]);
elseif n == 2
    M_dx_dt_k = A*(x_hat_horizon(:,N) - [h1_0; h2_0]) +...
                B*(u_k -[u_LV001_0; u_LV002_0; u_PA001_0]);
end

x_pred_k = Ts*M_dx_dt_k + x_hat_horizon(:,N);

% Updating estimate matrix, sliding horizon to the right and
% adding predicted x_k
if n == 3
    x_hat_horizon = [[h1_hat_horizon(2:N),x_pred_k(1)];...
                    [h2_hat_horizon(2:N),x_pred_k(2)];...
                    [u_PA001_hat_horizon(2:N),x_pred_k(3)]];
elseif n == 2
    x_hat_horizon = [[h1_hat_horizon(2:N),x_pred_k(1)];...
                    [h2_hat_horizon(2:N),x_pred_k(2)]];
end
end

```

```

P = A_disc*P_previous*A_disc' - (A_disc*P_previous*C'*(C*P_previous*C'...
    + R)^(-1)*C*P_previous*A_disc') + Q;
P_k_N_arr{k} = P;
P_previous = P;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x_pred_k_N = x_hat_horizon(:,1); %used in arrival cost function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
P_k_N = P_k_N_arr{k-N}; % used at time k, but calculated at time k-N
%%% Objective function handle %%%

objective_function_handle = @(x_hat_horizon) ...
    ObjectiveFunction(x_hat_horizon,x_pred_k_N,...
        u_horizon,h_meas_horizon,parameter_struct,...
        P_k_N,'L');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
options = optimoptions(@fmincon, 'Display', 'none', 'Algorithm','sqp',...
    'OptimalityTolerance',1e-10,'StepTolerance',1e-10);

% Plott
% options = optimoptions('fmincon','Display','iter','PlotFcn',...
%     {'optimplotx','optimplotfunccount',...
%     'optimplotfvalconstr','optimplotfval',...
%     'optimplotconstrviolation','optimplotstepsize',...
%     'optimplotfirstorderopt'});

x_hat_optimal = fmincon(objective_function_handle,x_hat_horizon,...
    [],[],[],[],x_lb,x_ub,[],options);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Updating with optimal solution, adding setting point
h1_hat_horizon = x_hat_optimal(1,:);
h2_hat_horizon = x_hat_optimal(2,:);
if n == 3
    u_PA001_hat_horizon = x_hat_optimal(3,:);
end

%Updating storage variables
h1_hat_arr(k) = h1_hat_horizon(end);
h2_hat_arr(k) = h2_hat_horizon(end);
if n == 3
    u_PA001_hat_arr(k) = u_PA001_hat_horizon(end);
end

end

%Plott
if n == 2 && Kalman_filter == 'Y'

```

```

figure(3)
if (k>1 && k<t_length)
    subplot(3,1,1)
    plot([t_storage(k-1),t_storage(k)],[u_PA001_arr(k-1),u_PA001_arr(k)],
        ...'Color','#D95319','LineWidth',2)
    title('u')
    xlim([t_start,t_end])
    hold on
    plot([t_storage(k-1),t_storage(k)],[u_LV001_arr(k-1),u_LV001_arr(k)],
        ...'Color','#EDB120','LineWidth',2)
    hold on
    plot([t_storage(k-1),t_storage(k)],[u_LV002_arr(k-1),u_LV002_arr(k)],
        ...'Color','#7E2F8E','LineWidth',2)
    legend('u\_PA001','u\LV001','u\LV002')
    set(legend,'Location','west');
    ylabel('Control signal [-]')
    hold on
    subplot(3,1,2)
    if m == 2
        plot([t_storage(k-1),t_storage(k)],[h1_meas_arr(k-1),...
            h1_meas_arr(k)],'b',[t_storage(k-1),t_storage(k)],...
            [h1_hat_arr(k-1),h1_hat_arr(k)],'--r',[t_storage(k-1),...
            t_storage(k)],[K_h1_hat_arr(k-1),K_h1_hat_arr(k)],...
            '--k','LineWidth',2)
    elseif m == 1
        plot([t_storage(k-1),t_storage(k)],[h1_arr(k-1),h1_arr(k)],...
            'b',[t_storage(k-1),t_storage(k)],[h1_hat_arr(k-1),...
            h1_hat_arr(k)],'--r',[t_storage(k-1),t_storage(k)],...
            [K_h1_hat_arr(k-1),K_h1_hat_arr(k)],'--k','LineWidth',2)
    end
    title('h_1')
    xlim([t_start,t_end])
    legend('Process','MHE','KF')
    set(legend,'Location','west');
    ylabel('Water level [m]')
    hold on
    subplot(3,1,3)
    plot([t_storage(k-1),t_storage(k)],[h2_meas_arr(k-1),...
        h2_meas_arr(k)]'b',[t_storage(k-1),t_storage(k)],[h2_hat_arr(k-1),...
        h2_hat_arr(k)],'--r',[t_storage(k-1),t_storage(k)],
        ...[K_h2_hat_arr(k-1),K_h2_hat_arr(k)],'--k','LineWidth',2)
    title('h_2')
    xlim([t_start,t_end])
    legend('Process','MHE','KF')
    set(legend,'Location','west');
    xlabel('Time [s]')
    ylabel('Water level [m]')
    hold on

```

```

end
elseif n == 2 && Kalman_filter == 'N'
    figure(3)
    if (k>1 && k<t_length)
        subplot(3,1,1)
        plot([t_storage(k-1),t_storage(k)], [u_PA001_arr(k-1),...
            u_PA001_arr(k)], 'Color', '#D95319', 'LineWidth', 2)
        title('u')
        xlim([t_start,t_end])
        hold on
        plot([t_storage(k-1),t_storage(k)], [u_LV001_arr(k-1),...
            u_LV001_arr(k)], 'Color', '#EDB120', 'LineWidth', 2)
        hold on
        plot([t_storage(k-1),t_storage(k)], [u_LV002_arr(k-1),...
            u_LV002_arr(k)], 'Color', '#7E2F8E', 'LineWidth', 2)
        legend('u\_PA001', 'u\LV001', 'u\LV002')
        set(legend, 'Location', 'west');
        ylabel('Control signal [-]')
        hold on
        subplot(3,1,2)
        if m == 2
            plot([t_storage(k-1),t_storage(k)], [h1_meas_arr(k-1),...
                h1_meas_arr(k)], 'b', [t_storage(k-1),t_storage(k)],...
                [h1_hat_arr(k-1),h1_hat_arr(k)], '--r', 'LineWidth', 2)
        elseif m == 1
            plot([t_storage(k-1),t_storage(k)], [h1_arr(k-1),...
                h1_arr(k)], 'b', [t_storage(k-1),t_storage(k)],...
                [h1_hat_arr(k-1),h1_hat_arr(k)], '--r', 'LineWidth', 2)
        end
        title('h_1')
        xlim([t_start,t_end])
        ylabel('Water level [m]')
        legend('Process', 'MHE')
        set(legend, 'Location', 'west');
        hold on
        subplot(3,1,3)
        plot([t_storage(k-1),t_storage(k)], [h2_meas_arr(k-1),...
            h2_meas_arr(k)], 'b', [t_storage(k-1),t_storage(k)],...
            [h2_hat_arr(k-1),h2_hat_arr(k)], '--r', 'LineWidth', 2)
        title('h_2')
        xlim([t_start,t_end])
        legend('Process', 'MHE')
        set(legend, 'Location', 'west');
        xlabel('Time [s]')
        ylabel('Water level [m]')
        hold on
    end
end
elseif n == 3 && Kalman_filter == 'Y'

```

```

figure(3)
if (k>1 && k<t_length)
    subplot(4,1,1)
    plot([t_storage(k-1),t_storage(k)], [u_LV001_arr(k-1),...
        u_LV001_arr(k)], 'Color', '#EDB120', 'LineWidth', 2)
    xlim([t_start,t_end])
    hold on
    plot([t_storage(k-1),t_storage(k)], [u_LV002_arr(k-1),...
        u_LV002_arr(k)], 'Color', '#7E2F8E', 'LineWidth', 2)
    title('u')
    legend('u\_LV001', 'u\_LV002')
    set(legend, 'Location', 'west');
    ylabel('Control signal [-]')
    hold on
    subplot(4,1,2)
    plot([t_storage(k-1),t_storage(k)], [u_PA001_arr(k-1),...
        u_PA001_arr(k)], 'b', [t_storage(k-1),t_storage(k)],...
        [u_PA001_hat_arr(k-1),u_PA001_hat_arr(k)],...
        '--r', [t_storage(k-1),t_storage(k)],...
        [K_u_PA001_hat_arr(k-1),K_u_PA001_hat_arr(k)],...
        '--k', 'LineWidth', 2)
    title('u_{PA001}')
    xlim([t_start,t_end])
    hold on
    legend('Process', 'MHE', 'KF')
    set(legend, 'Location', 'west');
    ylabel('Control signal to pump u_{PA001} [-]')
    subplot(4,1,3)
    if m == 2
        plot([t_storage(k-1),t_storage(k)], [h1_meas_arr(k-1),...
            h1_meas_arr(k)], 'b', [t_storage(k-1),t_storage(k)],...
            [h1_hat_arr(k-1),h1_hat_arr(k)], '--r',...
            [t_storage(k-1),t_storage(k)], [K_h1_hat_arr(k-1),...
            K_h1_hat_arr(k)], '--k', 'LineWidth', 2)
    elseif m == 1
        plot([t_storage(k-1),t_storage(k)], [h1_arr(k-1),...
            h1_arr(k)], 'b', [t_storage(k-1),t_storage(k)],...
            [h1_hat_arr(k-1),h1_hat_arr(k)], '--r',...
            [t_storage(k-1),t_storage(k)],...
            [K_h1_hat_arr(k-1),K_h1_hat_arr(k)],...
            '--k', 'LineWidth', 2)
    end
    title('h_1')
    xlim([t_start,t_end])
    legend('Process', 'MHE', 'KF')
    set(legend, 'Location', 'west');
    ylabel('Water level [m]')
    hold on

```

```

subplot(4,1,4)
plot([t_storage(k-1),t_storage(k)], [h2_meas_arr(k-1),...
    h2_meas_arr(k)], 'b', [t_storage(k-1),t_storage(k)],...
    [h2_hat_arr(k-1),h2_hat_arr(k)], '--r',...
    [t_storage(k-1),t_storage(k)], [K_h2_hat_arr(k-1),...
    K_h2_hat_arr(k)], '--k', 'LineWidth', 2)
title('h_2')
xlim([t_start,t_end])
legend('Process','MHE','KF')
set(legend,'Location','west');
xlabel('Time [s]')
ylabel('Water level [m]')
hold on
end
elseif n == 3 && Kalman_filter == 'N'
figure(3)
if (k>1 && k<t_length)
subplot(4,1,1)
plot([t_storage(k-1),t_storage(k)], [u_LV001_arr(k-1),...
    u_LV001_arr(k)], 'Color', '#EDB120', 'LineWidth', 2)
xlim([t_start,t_end])
hold on
plot([t_storage(k-1),t_storage(k)], [u_LV002_arr(k-1),...
    u_LV002_arr(k)], 'Color', '#7E2F8E', 'LineWidth', 2)
title('u')
legend('u_LV001','u_LV002')
set(legend,'Location','west');
ylabel('Control signal [-]')
hold on
subplot(4,1,2)
plot([t_storage(k-1),t_storage(k)], [u_PA001_arr(k-1),...
    u_PA001_arr(k)], 'b', [t_storage(k-1),t_storage(k)],...
    [u_PA001_hat_arr(k-1),u_PA001_hat_arr(k)],...
    '--r', 'LineWidth', 2)
title('u_{PA001}')
xlim([t_start,t_end])
hold on
legend('Process','MHE')
set(legend,'Location','west');
ylabel('Control signal to pump u_{PA001} [-]')
subplot(4,1,3)
if m == 2
plot([t_storage(k-1),t_storage(k)], [h1_meas_arr(k-1),...
    h1_meas_arr(k)], 'b', [t_storage(k-1),t_storage(k)],...
    [h1_hat_arr(k-1),h1_hat_arr(k)], '--r', 'LineWidth', 2)
elseif m == 1
plot([t_storage(k-1),t_storage(k)], [h1_arr(k-1),...
    h1_arr(k)], 'b', [t_storage(k-1),t_storage(k)],...

```

```

        [h1_hat_arr(k-1),h1_hat_arr(k)], '--r', 'LineWidth', 2)
    end
    title('h_1')
    xlim([t_start,t_end])
    legend('Process','MHE')
    set(legend,'Location','west');
    ylabel('Water level [m]')
    hold on
    subplot(4,1,4)
    plot([t_storage(k-1),t_storage(k)],[h2_meas_arr(k-1),...
        h2_meas_arr(k)], 'b',[t_storage(k-1),t_storage(k)],...
        [h2_hat_arr(k-1),h2_hat_arr(k)], '--r', 'LineWidth', 2)
    title('h_2')
    xlim([t_start,t_end])
    legend('Process','MHE')
    set(legend,'Location','west');
    xlabel('Time [s]')
    ylabel('Water level [m]')
    hold on
end
end
end
hold off

if m == 2
    error_h1 = mean(abs(h1_hat_arr - h1_meas_arr));
    fprintf('Estimation error mean h1: %d\n',error_h1);
elseif m == 1
    error_h1 = mean(abs(h1_hat_arr - h1_arr));
    fprintf('Estimation error mean h1: %d\n',error_h1);
end
error_h2 = mean(abs(h2_hat_arr - h2_meas_arr));
fprintf('Estimation error mean h2: %d\n',error_h2);
if n == 3
    error_u_PA = mean(abs(u_PA001_hat_arr - u_PA001_arr));
    fprintf('Estimation error mean u_PA001: %d\n',error_u_PA);
end
if Kalman_filter == 'Y'
    fprintf('KF: \n');
    if m == 2
        error_h1 = mean(abs(K_h1_hat_arr - h1_meas_arr));
        fprintf('Estimation error mean h1: %d\n',error_h1);
    elseif m == 1
        error_h1 = mean(abs(K_h1_hat_arr - h1_arr));
        fprintf('Estimation error mean h1: %d\n',error_h1);
    end
    error_h2 = mean(abs(K_h2_hat_arr - h2_meas_arr));
    fprintf('Estimation error mean h2: %d\n',error_h2);
end

```

```

if n == 3
    error_u_PA = mean(abs(K_u_PA001_hat_arr - u_PA001_arr));
    fprintf('Estimation error mean u_PA001: %d\n',error_u_PA);
end
end
end

```

C.2 NMHE.m

```

close all; clear all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Master Thesis:
% Moving Horizon Estimation for the Two-tank System by Greta Bekeryte.
% May, 2023
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
run parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n = 3; % number of states? [3/2]
m = 2; % number of measurements available [2/1]
%MHE parameters
N = 10; %Horizon length
% Simulation parameters
sys_noise = 'Y'; % Simulate system with noise? [Y/N]
sim_model = 'N'; % Linear or nonlinear simulation model? ['N'/data]
Kalman_filter = 'Y'; % Estimation useing Kalman filter for comparison ['Y'/'N']
Ts = 1; % Samplings time. For dataset1 and dataset2: 1.05

if strcmp('data',sim_model)
    load('data1.mat')
    h1_s = ScopeData.signals(2).values(:,1);
    h2_s = ScopeData.signals(4).values(:,1);
    u_LV001_s = ScopeData.signals(3).values;
    u_LV002_s = ScopeData.signals(5).values;
    u_PA001_s = ScopeData.signals(1).values;
    t_time = ScopeData.time;
    t_length = length(t_time);
    t_storage = zeros(1,t_length);
    t_start = 0;
    t_end = t_length;
    u_LV001_0 = u_LV001_s(1);
    u_LV002_0 = u_LV002_s(1);
    u_PA001_0 = u_PA001_s(1);
    %initial water levels in tank 1 and tank 2
    h1_init = h1_s(1); %h1_min;
    h2_init = h2_s(1); %h2_min;
end

```



```

if strcmp('L',sim_model) || strcmp('N',sim_model)
    t_start = 0;
    t_end = 35*5;
    %initial control inputs for simulation
    u_LV001_0 = 0.4263;
    u_LV002_0 = 0.3902;
    u_PA001_0 = 0.6;
    step_up = 7*5;
    step_down = 30*4;
    step_size = 0.1;
    step_in = 'u_PA001'; % ['u_LV001'/'u_LV002'/'u_PA001']
    %initial water levels in tank 1 and tank 2
    h1_init = 0.3; %h1_min;
    h2_init = 0.2; %h2_min;
    %Simulation time parameters
    t_length = t_end/Ts + 1;
    t_storage = zeros(1,t_length);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parameter structure
parameter_struct.Ts = Ts;
parameter_struct.n = n;
parameter_struct.m = m;
parameter_struct.N = N;
parameter_struct.A1 = A1;
parameter_struct.Kv_LV001 = Kv_LV001;
parameter_struct.Kv_LV002 = Kv_LV002;
parameter_struct.rho = rho;
parameter_struct.g = g;
parameter_struct.h_LV001 = h_LV001;
parameter_struct.h_LV002 = h_LV002;
parameter_struct.u_LV001 = u_LV001;
parameter_struct.u_LV002 = u_LV002;
parameter_struct.f_LV001 = f_LV001;
parameter_struct.f_LV002 = f_LV002;
parameter_struct.u_PA001 = u_PA001;
parameter_struct.q_PA001 = q_PA001;

% Process disturbance MHE
Q_w1 = 1e1;
Q_w2 = 1e1;
Q_w3 = 1e-5;
% Measurement noise MHE
R_v1 = 1e-6; %2.8e-6;
R_v2 = 1e-6; %1.6e-6;

```

```
if n == 2
    Q = diag([Q_w1, Q_w2]);
elseif n == 3
    Q = diag([Q_w1, Q_w2, Q_w3]);
end
if m == 2
    R = diag([R_v1, R_v2]);
elseif m == 1
    R = R_v2;
end
parameter_struct.Q = Q;
parameter_struct.R = R;

rng(1)
noise = randn(5,t_length);
if sys_noise == 'Y'
    % Process disturbance used in simulation
    cov_w1 = 0;
    cov_w2 = 0;
    cov_w3 = 0;
    % Measurement noise used in simulation
    cov_v1 = 1e-6;
    cov_v2 = 1e-6;
elseif sys_noise == 'N'
    % Process disturbance used in simulation
    cov_w1 = 0;
    cov_w2 = 0;
    cov_w3 = 0;
    % Measurement noise used in simulation
    cov_v1 = 0;
    cov_v2 = 0;
end

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

%Simulation variable storage arrays for plot
%Control inputs
u_LV001_arr = zeros(1,t_length);
u_LV002_arr = zeros(1,t_length);
```

```
%Disturbance or state parameter
u_PA001_arr = zeros(1,t_length);
%State variables
h1_arr = zeros(1,t_length);
h2_arr = zeros(1,t_length);
%Outputs
if m == 2
    h1_meas_arr = zeros(1,t_length);
end
h2_meas_arr = zeros(1,t_length);
% MHE estimate variable storage arrays for plot
h1_hat_arr = zeros(1,t_length);
h2_hat_arr = zeros(1,t_length);
if n == 3
    u_PA001_hat_arr = zeros(1,t_length);
end

% Simulation Init
h1_k = h1_init;
h2_k = h2_init;
if n == 3
    u_PA001_init = u_PA001_0;
    u_PA001_k = u_PA001_init;
end

% Init MHE
horizon = zeros(1,N);
h1_hat_horizon = horizon;
h2_hat_horizon = horizon;
if n == 3
    u_PA001_hat_init = u_PA001_0;
    f3_hat_init = interp1(u_PA001,q_PA001, u_PA001_hat_init);
    f3_hat_horizon = horizon;
    x_hat_horizon = [h1_hat_horizon; h2_hat_horizon; f3_hat_horizon]; %matrix
    u_horizon = [horizon; horizon];
elseif n == 2
    x_hat_horizon = [h1_hat_horizon; h2_hat_horizon]; % f3_hat_horizon]; %matrix
    u_horizon = [horizon; horizon; horizon];
end
if m == 2
    h1_meas_horizon = horizon;
end
h2_meas_horizon = horizon;

% Lower and upper bound constraints for whole horizon
h1_max_arr = zeros(1,N) + h1_max;
h1_min_arr = zeros(1,N) + h1_min;
h2_max_arr = zeros(1,N) + h2_max;
```

```

h2_min_arr = zeros(1,N) + h2_min;
if n == 3
    f3_max_arr = zeros(1,N) + 0.000333; % From pump characteristic
    f3_min_arr = zeros(1,N) + 0; % From pump characteristic
    x_ub = [h1_max_arr; h2_max_arr; f3_max_arr];
    x_lb = [h1_min_arr; h2_min_arr; f3_min_arr];
elseif n == 2
    x_ub = [h1_max_arr; h2_max_arr];
    x_lb = [h1_min_arr; h2_min_arr];
end

%Estimate error covariance matrix used in arrival cost storage
P_k_N_arr = cell(1,t_length-N);
P_k_N_arr{1,t_length-N} = [];
if n == 3
    %initial P covariance matrix
    P_init = diag([0.0000001, 0.0000001, 0.0000001]);
    %state prediction prior first horizon
    x_pred_k_N = [h1_init; h2_init; u_PA001_init];
    if m == 2
        C = [1 0 0; 0 1 0];
    elseif m == 1
        C = [0 1 0];
    end
elseif n == 2
    P_init = diag([0.00001, 0.00001]); %initial P covariance matrix
    x_pred_k_N = [h1_init; h2_init]; %state prediction prior first horizon
    if m == 2
        C = [1 0 ; 0 1 ];
    elseif m == 1
        C = [0 1];
    end
end
end
P_previous = P_init;
parameter_struct.C = C;

%%%%%%%%%% Extended Kalman filter init
if Kalman_filter == 'Y'
    if n == 2
        K_x_hat = [h1_init; h2_init];
    elseif n == 3
        f3_init = interp1(u_PA001,q_PA001,u_PA001_init);
        K_x_hat = [h1_init; h2_init; f3_init];
    end
    % storage of Kalman estimates for plot
    K_h1_hat_arr = zeros(1,t_length);
    K_h2_hat_arr = zeros(1,t_length);
    if n == 3

```

```
        K_u_PA001_hat_arr = zeros(1,t_length);
    end
end
K_P_hat = P_init;
if n == 2
    K_Q = diag([1e-1, 1e2]);
elseif n == 3
    K_Q = diag([1e2, 1, 1e-1]);
end
if m == 2
    K_R = diag([1e-6, 1e-6]);
elseif m == 1
    K_R = 1e-6;
end

for k = 1:t_length
    t_k = k*Ts;

    if strcmp('N',sim_model)
        %Simulation of control inputs
        if t_k < step_up
            u_LV001_k = u_LV001_0;
            u_LV002_k = u_LV002_0;
            u_PA001_k = u_PA001_0;
        end
        if t_k >= step_up
            if strcmp('u_LV001',step_in)
                u_LV001_k = u_LV001_0 + step_size;
            elseif strcmp('u_LV002',step_in)
                u_LV002_k = u_LV002_0 + step_size;
            elseif strcmp('u_PA001',step_in)
                u_PA001_k = u_PA001_0 + step_size;
            end
        end
        if t_k >= step_down
            if strcmp('u_LV001',step_in)
                u_LV001_k = u_LV001_0;
            elseif strcmp('u_LV002',step_in)
                u_LV002_k = u_LV002_0;
            elseif strcmp('u_PA001',step_in)
                u_PA001_k = u_PA001_0;
            end
        end
    end
    %
    if t_k >= 35
        %
        if strcmp('u_LV001',step_in)
            u_LV001_k = u_LV001_0 + step_size;
        %
        elseif strcmp('u_LV002',step_in)
            u_LV002_k = u_LV002_0 + step_size;
        %
    end
end
```

```

%         elseif strcmp('u_PA001',step_in)
%             u_PA001_k = u_PA001_0 + step_size;
%         end
%     end
%
%
%Disturbances and noise
w1_k = sqrt(cov_w1)*noise(1,k);
w2_k = sqrt(cov_w2)*noise(2,k);
w3_k = sqrt(cov_w3)*noise(3,k);
v1_k = sqrt(cov_v1)*noise(4,k);
v2_k = sqrt(cov_v2)*noise(5,k);

if n == 3
    u_k = [u_LV001_k; u_LV002_k];
elseif n == 2
    u_k = [u_LV001_k; u_LV002_k; u_PA001_k];
end

% Process derivatives and integration
f3_k = interp1(u_PA001,q_PA001, u_PA001_k);
f1_k = interp1(u_LV001,f_LV001,u_LV001_k);
f2_k = interp1(u_LV002,f_LV002,u_LV002_k);

dh1_dt_k = (1/A1)*(f3_k-((Kv_LV001*f1_k)/3600)*...
    sqrt((rho*g*(h1_k+h_LV001))/100000));
dh2_dt_k = (1/(0.07*h2_k+0.004))*(((Kv_LV001*f1_k)/3600)*...
    sqrt((rho*g*(h1_k+h_LV001))/100000)...
    -(Kv_LV002*f2_k)/3600*sqrt((rho*g*(h2_k+h_LV002))/100000));
if sys_noise == 'Y'
    h1_k_ny = Ts*dh1_dt_k + h1_k;    %Euler integration
    h2_k_ny = Ts*dh2_dt_k + h2_k;
elseif sys_noise == 'N'
    h1_k_ny = Ts*dh1_dt_k + h1_k;    %Euler integration
    h2_k_ny = Ts*dh2_dt_k + h2_k;
end

%Physical constraints
if h1_k_ny < h1_min
    h1_k_ny = h1_min;
end
if h2_k_ny < h2_min
    h2_k_ny = h2_min;
end
if h1_k_ny > h1_max
    h1_k_ny = h1_max;
end
if h2_k_ny > h2_max

```

```
        h2_k_ny = h2_max;
    end

    if sys_noise == 'Y'
        if m == 2
            h1_meas_k = h1_k + v1_k;
        end
        h2_meas_k = h2_k + v2_k;
    elseif sys_noise == 'N'
        if m == 2
            h1_meas_k = h1_k;
        end
        h2_meas_k = h2_k;
    end

    %Physical constraints
    if h2_meas_k < h2_min
        h2_meas_k = h2_min;
    end
    if h2_meas_k > h2_max
        h2_meas_k = h2_max;
    end
    if m == 2
        if h1_meas_k < h1_min
            h1_meas_k = h1_min;
        end
        if h1_meas_k > h1_max
            h1_meas_k = h1_max;
        end
    end

    % Storing simulation data
    t_storage(k) = t_k;
    u_LV001_arr(k) = u_LV001_k;
    u_LV002_arr(k) = u_LV002_k;
    u_PA001_arr(k) = u_PA001_k;
    h1_arr(k) = h1_k;
    h2_arr(k) = h2_k;
    if m == 2
        h1_meas_arr(k) = h1_meas_k;
        h2_meas_arr(k) = h2_meas_k;
        h_meas_k = [h1_meas_k; h2_meas_k];
    elseif m == 1
        h2_meas_arr(k) = h2_meas_k;
        h_meas_k = h2_meas_k;
    end

    %Update for k+1
    h1_k = h1_k_ny;
    h2_k = h2_k_ny;
```



```

if Kalman_filter == 'Y'
    tic;
    [K_x_hat, K_P_hat] = ExtendedKalmanFilter(K_x_hat,K_P_hat,...
        h_meas_k,u_k,...
        parameter_struct,K_Q,K_R);

    %Storing
    K_h1_hat_arr(k) = K_x_hat(1);
    K_h2_hat_arr(k) = K_x_hat(2);
    if n == 3
        K_u_PA001_hat = interp1(q_PA001,u_PA001,K_x_hat(3));
        K_u_PA001_hat_arr(k) = K_u_PA001_hat;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%initial Arrival cost update given measurements up to time k<=N
if k <=N
    if n == 3
        f3_pred_k_N = interp1(u_PA001,q_PA001,x_pred_k_N(3));
        x_pred_k_N = [x_pred_k_N(1:2);f3_pred_k_N];
    end
    [x_hat, P_previous] = ExtendedKalmanFilter(x_pred_k_N,P_previous,...
        h_meas_k,u_k,parameter_struct,Q,R);

    %Storing
    h1_hat_horizon(1,k) = x_hat(1);
    h2_hat_horizon(1,k) = x_hat(2);
    if n == 3
        f3_hat_horizon(1,k) = x_hat(3);
        x_hat_horizon = [h1_hat_horizon; h2_hat_horizon; f3_hat_horizon];
    else
        x_hat_horizon = [h1_hat_horizon; h2_hat_horizon];
    end
    % estimates for plotet
    h1_hat_arr(k) = h1_hat_horizon(k);
    h2_hat_arr(k) = h2_hat_horizon(k);
    if n == 3
        % Estimating f3, plotting u_PA001
        u_PA001_hat_arr(k) = interp1(q_PA001,u_PA001,f3_hat_horizon(k));
    end
    P_k_N_arr{k} = P_previous;
    x_pred_k_N(1) = x_hat(1);
    x_pred_k_N(2) = x_hat(2);
    if n == 3
        x_pred_k_N(3) = interp1(q_PA001,u_PA001,x_hat(3));
    end
end

if k > N %enough samples for the first horizon

```

```

% Using model to predict x_k based on estimated state x_k-1
if n == 3
    M_dh1_dt_k = (1/A1)*(x_hat_horizon(3,N)-((Kv_LV001*f1_k)/3600)*...
        sqrt((rho*g*(x_hat_horizon(1,N)+h_LV001))/100000));
elseif n == 2
    M_dh1_dt_k = (1/A1)*(f3_k-((Kv_LV001*f1_k)/3600)*...
        sqrt((rho*g*(x_hat_horizon(1,N)+h_LV001))/100000));
end
M_dh2_dt_k = (1/(0.07*x_hat_horizon(2,N)+0.004))*...
    (((Kv_LV001*f1_k)/3600)*sqrt((rho*g*...
    (x_hat_horizon(1,N)+h_LV001))/100000)-...
    (Kv_LV002*f2_k)/3600*sqrt((rho*g*...
    (x_hat_horizon(2,N)+h_LV002))/100000));
M_df3_dt_k = 0;
h1_pred_k = Ts*M_dh1_dt_k + x_hat_horizon(1,N);
h2_pred_k = Ts*M_dh2_dt_k + x_hat_horizon(2,N);

if n == 3
    f3_pred_k = Ts*M_df3_dt_k + x_hat_horizon(3,N);
end

%Physical constraints
if h1_pred_k < h1_min
    h1_pred_k = h1_min;
end
if h2_pred_k < h2_min
    h2_pred_k = h2_min;
end
if h1_pred_k > h1_max
    h1_pred_k = h1_max;
end
if h2_pred_k > h2_max
    h2_pred_k = h2_max;
end
if n == 3
    if f3_pred_k < 0
        f3_pred_k = 0;
    end
    if f3_pred_k > 0.000333
        f3_pred_k = 0.000333;
    end
end

% Updating estimate matrix, sliding horizon to the right and adding
% predicted x_k
if n == 3
    x_hat_horizon = [[h1_hat_horizon(2:N),h1_pred_k];...
        [h2_hat_horizon(2:N),h2_pred_k];...

```

```

        [f3_hat_horizon(2:N),f3_pred_k]];

elseif n == 2
    x_hat_horizon = [[h1_hat_horizon(2:N),h1_pred_k];...
                    [h2_hat_horizon(2:N),h2_pred_k]];

end
% updating error covariance matrix used in Arrival cost
if n == 3
    u_PA001_tr = interp1(q_PA001,u_PA001,x_hat_horizon(3,end));
    A = Amatrix(x_hat_horizon(1,end),x_hat_horizon(2,end),...
                u_PA001_tr,u_LV001_k,u_LV002_k,parameter_struct);
elseif n == 2
    A = Amatrix(x_hat_horizon(1,end),x_hat_horizon(2,end),...
                u_PA001_k,u_LV001_k,u_LV002_k,parameter_struct);
end
P = A*P_previous*A' - (A*P_previous*C'*(C*P_previous*C' +...
    R)^(-1)*C*P_previous*A') + Q;
P_k_N_arr{k} = P;
P_previous = P;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%new
x_pred_k_N = x_hat_horizon(:,1); %used in arrival cost function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
P_k_N = P_k_N_arr{k-N}; % used at time k, but calculated at time k-N

%%% Objective function handle %%%
objective_function_handle = @(x_hat_horizon) ...
    ObjectiveFunction(x_hat_horizon,...
        x_pred_k_N,u_horizon,...
        h_meas_horizon,parameter_struct,...
        P_k_N,'N');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rng default %for reproducibility globaloptimization
options = optimoptions(@fmincon,'Display', 'none','Algorithm','sqp');
%, 'OptimalityTolerance',1e-10,'StepTolerance',...
1e-10,'MaxIterations', 1000, 'MaxFunctionEvaluations',10000);

% Plott
% options8 = optimoptions('fmincon','Display','iter',...
%     'PlotFcn',{ 'optimplotx',...
%     'optimplotfunccount',...
%     'optimplotfvalconstr',...
%     'optimplotfval',...
%     'optimplotconstrviolation',...
%     'optimplotstepsize',...
%     'optimplotfirstorderopt'},...

```

```

%           'Algorithm','sqp',...
%           'OptimalityTolerance',1e10,...
%           'StepTolerance',1e-10,...
%           'MaxIterations', 1000,...
%           'MaxFunctionEvaluations',10000);
x_hat_optimal = fmincon(objective_function_handle,...
    x_hat_horizon,[],[],[],[],x_lb,x_ub,[],options);

%%%%%%%%Global Search%%%%%%%%
%problem = createOptimProblem('fmincon','x0',x_hat_horizon,...
%    'objective',objective_function_handle,'lb',x_lb,...
%    'ub',x_ub,'options',options);
%gs = GlobalSearch;%( 'FunctionTolerance',2e4,'NumTrialPoints',2000);
%x_hat_optimal = run(gs,problem);

%%%%%%%%MultiStart%%%%%%%%
%problem = createOptimProblem('fmincon','x0',x_hat_horizon,...
%    'objective',objective_function_handle,'lb',x_lb,'ub'...
%    ,x_ub,'options',options);
%ms = MultiStart(gs);
%ms = MultiStart('StartPointsToRun','bounds');
%,'XTolerance',1e-8,'FunctionTolerance',1e-, 'UseParallel',true,);
%x_hat_optimal = run(ms,problem,5);

%%%%%%%%Annealing%%%%%%%%
%options2 = optimoptions(@simulannealbnd,'MaxIterations', 1000,...
%    'MaxFunctionEvaluations',10000);
%x_hat_optimal = simulannealbnd(objective_function_handle,...
%    x_hat_horizon,x_lb,x_ub,options2);

%%%%%%%%GA%%%%%%%%
% GA !to use GA, the first line in objective_funtion.m has to be
% uncommented
%options3 = optimoptions('ga','Display','iter');
%x_hat_optimal = ga(objective_function_handle,n*N,[],[],[],[],...
%    [x_lb(1,:) x_lb(2,:) x_lb(3,:)],[x_ub(1,:) x_ub(2,:)...
%    x_ub(3,:)],[],options3)
%x_hat_optimal = reshape(x_hat_optimal,[n,N]);

%%%%%%%%PatternSearch%%%%%%%%
%options4 = optimoptions('patternsearch','Display','none');
%x_hat_optimal = patternsearch(objective_function_handle,...
%    x_hat_horizon,[],[],[],[],x_lb,x_ub,[],options4);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Updating with optimal solution, adding setting point
h1_hat_horizon = x_hat_optimal(1,:);
h2_hat_horizon = x_hat_optimal(2,:);

```

```

        if n == 3
            f3_hat_horizon = x_hat_optimal(3,:);
        end
        %Updating storage variables
        h1_hat_arr(k) = h1_hat_horizon(end);
        h2_hat_arr(k) = h2_hat_horizon(end);
        if n == 3
            % Estimating f3, plotting u_PA001
            u_PA001_hat_arr(k) = interp1(q_PA001,u_PA001,f3_hat_horizon(end));
        end
    end
end
%Plott

... Same as in LMHE.m

if m == 2
    error_h1 = mean(abs(h1_hat_arr - h1_meas_arr));
    fprintf('Estimation error mean h1: %d\n',error_h1);
elseif m == 1
    error_h1 = mean(abs(h1_hat_arr - h1_arr));
    fprintf('Estimation error mean h1: %d\n',error_h1);
end
error_h2 = mean(abs(h2_hat_arr - h2_meas_arr));
fprintf('Estimation error mean h2: %d\n',error_h2);
if n == 3
    error_u_PA = mean(abs(u_PA001_hat_arr - u_PA001_arr));
    fprintf('Estimation error mean u_PA001: %d\n',error_u_PA);
end
if Kalman_filter == 'Y'
    fprintf('EKF: \n');
    if m == 2
        error_h1 = mean(abs(K_h1_hat_arr - h1_meas_arr));
        fprintf('Estimation error mean h1: %d\n',error_h1);
    elseif m == 1
        error_h1 = mean(abs(K_h1_hat_arr - h1_arr));
        fprintf('Estimation error mean h1: %d\n',error_h1);
    end
    error_h2 = mean(abs(K_h2_hat_arr - h2_meas_arr));
    fprintf('Estimation error mean h2: %d\n',error_h2);
    if n == 3
        error_u_PA = mean(abs(K_u_PA001_hat_arr - u_PA001_arr));
        fprintf('Estimation error mean u_PA001: %d\n',error_u_PA);
    end
end
end

```

C.3 LMHE_Simulink_init.m

```

close all; clear all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Master Thesis:
% Moving Horizon Estimation for the Two-tank System by Greta Bekeryte
% May, 2023
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
run parameters
port_attribute = 2;
n = 3; % number of states? [3/2]
m = 2; % number of measurements available [2/1]
Ts = 0.5;
% Operating point for linearization
h1_0 = 0.3;
h2_0 = 0.2;
u_PA001_0 = 0.6;

%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 MHE
cov_w1 = 1;
cov_w2 = 1;
cov_w3 = 1e3;

% Measurement noise used in MHE
cov_v1 = 1; %2.8e-6;
cov_v2 = 1; %1.6e-6;
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;
if n == 3
    u_PA001_hat_init = u_PA001_0;
    u_PA001_hat_horizon = horizon;
    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
if m == 2
    h1_meas_horizon = horizon;
end
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;

```

C.4 NMHE_Simulink_init.m

```

close all; clear all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Master Thesis:
% Moving Horizon Estimation for the Two-tank System by Greta Bekeryte
% May, 2023
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
run parameters

port_attribute = 1;
n = 3; % number of states? [3/2]
m = 2; % number of measurements available [2/1]
Ts = 0.5;
%initial water levels in tank 1 and tank 2
h1_init = h1_min; %h1_min;
h2_init = h2_min; %h2_min;
%For model integration
h1_0 = h1_min;
h2_0 = h2_min;
u_PA001_hat_init = 0.6;
%MHE parameters
N_horizon = 10; %Horizon length
% Process disturbance used in simulation and MHE
cov_w1 = 1e2;
cov_w2 = 1;
cov_w3 = 0.05;
% Measurement noise used in MHE
cov_v1 = 1; %2.8e-6
cov_v2 = 1; %1.6e-6

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
parameter_struct.n = n;
parameter_struct.m = m;
parameter_struct.N = N_horizon;
parameter_struct.Ts = Ts;
parameter_struct.A1 = A1;
parameter_struct.Kv_LV001 = Kv_LV001;
parameter_struct.Kv_LV002 = Kv_LV002;
parameter_struct.rho = rho;
parameter_struct.g = g;
parameter_struct.h1_min = h1_min;
parameter_struct.h1_max = h1_max;
parameter_struct.h1_min = h2_min;
parameter_struct.h1_max = h2_max;
parameter_struct.h_LV001 = h_LV001;
parameter_struct.h_LV002 = h_LV002;
parameter_struct.u_LV001 = u_LV001;
parameter_struct.u_LV002 = u_LV002;
parameter_struct.f_LV001 = f_LV001;
parameter_struct.f_LV002 = f_LV002;
parameter_struct.u_PA001 = u_PA001;
parameter_struct.q_PA001 = q_PA001;
parameter_struct.Q = cov_w;
parameter_struct.R = cov_v;

% Init MHE
horizon = zeros(1,N_horizon);
h1_hat_horizon = horizon;
h2_hat_horizon = horizon;
f3_hat_horizon = horizon;
if n == 3

        f3_hat_init = interp1(u_PA001,q_PA001, u_PA001_hat_init);
        x_hat_horizon = [h1_hat_horizon; h2_hat_horizon; f3_hat_horizon]; %matrix
        x_hat_optimal = [h1_hat_horizon; h2_hat_horizon; f3_hat_horizon];
        u_horizon = [horizon; horizon];
elseif n == 2
        x_hat_horizon = [h1_hat_horizon; h2_hat_horizon]; % f3_hat_horizon]; %matrix
        x_hat_optimal = [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
    f3_max_arr = zeros(1,N_horizon) + 0.000333; % From pump characteristic
    f3_min_arr = zeros(1,N_horizon) + 0; % From pump characteristic
    x_ub = [h1_max_arr; h2_max_arr; f3_max_arr];
    x_lb = [h1_min_arr; h2_min_arr; f3_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 used in arrival cost storage
P_k_N_arr = [];
if n == 3
    P_init = diag([0.00001, 0.00001, 0.00001]); %initial P covariance matrix
    %state prediction prior first horizon
    x_pred_k_N = [h1_init; h2_init; u_PA001_hat_init];
    if m == 2
        C = [1 0 0; 0 1 0];
    elseif m == 1
        C = [0 1 0];
    end
elseif n == 2
    P_init = diag([1e-6, 1e-6]); %initial P covariance matrix
    x_pred_k_N = [h1_init; h2_init]; %state prediction prior first horizon
    if m == 2
        C = [1 0 ; 0 1 ];
    elseif m == 1
        C = [0 1];
    end
end
end
parameter_struct.C = C;
P_previous = P_init;
```

```

if n == 3
    par_der_f3 = 1/A1; % dg1/df3
    f3_delta = interp1(u_PA001,q_PA001,(u_PA001_0 +0.01));
    par_der_u_PA001 = (f3_delta - f3_0)/((u_PA001_0 + 0.01)-u_PA001_0);
    bv1 = par_der_f3*par_der_u_PA001;
    bv2 = 0;
end

par_der_h1_2 =(Kv_LV001*f1_0*g*rho)/(720000000*((7*h2_0)/100 + 1/250)*...
    ((g*rho*(h1_0 + h_LV001))/100000)^(1/2)); % dg2/dh1
par_der_h2 = -(7*((Kv_LV001*f1_0*((g*rho*(h1_0 +...
    h_LV001))/100000)^(1/2))/3600 - (Kv_LV002*f2_0*...
    ((g*rho*(h2_0 + h_LV002))/100000)^(1/2))/3600))/...
    (100*((7*h2_0)/100 + 1/250)^2) - (Kv_LV002*f2_0*g*rho)/...
    (720000000*((7*h2_0)/100 + 1/250)*((g*rho*(h2_0 +...
    h_LV002))/100000)^(1/2)); % dg2/dh2

a11 = par_der_h1;
a12 = 0;
a21 = par_der_h1_2;
a22 = par_der_h2;
if n == 2
    A = [a11 a12; a21 a22]; % state transition matrix n = 2
    %Forward Euler discretization
    A = eye(size(A)) + Ts*A;
elseif n ==3
    A = [a11 a12 bv1; a21 a22 bv2; 0 0 0]; % state transition matrix n = 3
    %Forward Euler discretization
    A = eye(size(A)) + Ts*A;
else
    disp('Number of states is out of bound.')
end
end
end

```

C.5.2 ExtendedKalmanFilter.m

```

function [x_hat,P_hat] = ExtendedKalmanFilter(x_hat,P_hat,y_meas_k,u_k,...
    parameter_struct,Q,R)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Master Thesis:
% Moving Horizon Estimation for the Two-tank System by Greta Bekeryte.
% May, 2023
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Ts = parameter_struct.Ts;
n = parameter_struct.n;
A1 = parameter_struct.A1;

```

```

Kv_LV001 = parameter_struct.Kv_LV001;
Kv_LV002 = parameter_struct.Kv_LV002;
rho = parameter_struct.rho;
g = parameter_struct.g;
h_LV001 = parameter_struct.h_LV001;
h_LV002 = parameter_struct.h_LV002;
u_LV001 = parameter_struct.u_LV001;
u_LV002 = parameter_struct.u_LV002;
f_LV001 = parameter_struct.f_LV001;
f_LV002 = parameter_struct.f_LV002;
u_PA001 = parameter_struct.u_PA001;
q_PA001 = parameter_struct.q_PA001;
f1_k = interp1(u_LV001,f_LV001,u_k(1));
f2_k = interp1(u_LV002, f_LV002,u_k(2));
C = parameter_struct.C;

% Time update
if n == 3
    dh1_dt_pred = (1/A1)*(x_hat(3)-((Kv_LV001*f1_k)/3600)*sqrt((rho*g*...
        (x_hat(1)+h_LV001))/100000));
elseif n == 2
    f3_k = interp1(u_PA001,q_PA001,u_k(3));
    dh1_dt_pred = (1/A1)*(f3_k-((Kv_LV001*f1_k)/3600)*sqrt((rho*g*...
        (x_hat(1)+h_LV001))/100000));
end
dh2_dt_pred = (1/(0.07*x_hat(2)+0.004))*(((Kv_LV001*f1_k)/3600)*...
    sqrt((rho*g*(x_hat(1)+h_LV001))/100000)-...
    (Kv_LV002*f2_k)/3600*sqrt((rho*g*(x_hat(2)+...
    h_LV002))/100000));
df3_dt_pred = 0;
h1_pred = Ts*dh1_dt_pred + x_hat(1);
h2_pred = Ts*dh2_dt_pred + x_hat(2);

if n == 3
    f3_pred = Ts*df3_dt_pred + x_hat(3);
    x_pred = [h1_pred; h2_pred; f3_pred];
elseif n == 2
    x_pred = [h1_pred; h2_pred];
end

if n == 3
    u_PA001_pred = interp1(q_PA001,u_PA001,f3_pred);
    A_pred = Amatrix(h1_pred,h2_pred,u_PA001_pred,u_k(1),u_k(2),...
        parameter_struct);
elseif n == 2
    A_pred = Amatrix(h1_pred,h2_pred,u_k(3),u_k(1),u_k(2),...
        parameter_struct);
end
end

```

```

P_pred = A_pred*P_hat*A_pred' + Q;
% Measurement update
K = P_pred*C'*inv(C*P_pred*C' + R);
x_hat = x_pred + (K*(y_meas_k - C*x_pred));
P_hat = (eye(size(P_pred,1)) - K*C)*P_pred;
end

```

C.5.3 KalmanFilter.m

```

function [x_hat,P_hat] = KalmanFilter(x_hat,P_hat,y_meas_k,u_k,...
                                     parameter_struct,Q,R)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Master Thesis:
% Moving Horizon Estimation for the Two-tank System by Greta Bekeryte.
% May, 2023
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Ts = parameter_struct.Ts;
n = parameter_struct.n;
h1_0 = parameter_struct.h1_0;
h2_0 = parameter_struct.h2_0;
u_LV001_0 = parameter_struct.u_LV001_0;
u_LV002_0 = parameter_struct.u_LV002_0;
u_PA001_0 = parameter_struct.u_PA001_0;
A = parameter_struct.A;
B = parameter_struct.B;
C = parameter_struct.C;
A_disc = parameter_struct.A_disc;

% Time update
if n == 3
    dx_dt_pred = A*(x_hat - [h1_0; h2_0; u_PA001_0]) + B*(u_k - ...
        [u_LV001_0; u_LV002_0]);
elseif n == 2
    dx_dt_pred = A*(x_hat - [h1_0; h2_0]) + B*(u_k - ...
        [u_LV001_0; u_LV002_0; u_PA001_0]);
end
x_pred = Ts*dx_dt_pred + x_hat;
P_pred = A_disc*P_hat*A_disc' + Q;
% Measurement update
K = P_pred*C'*inv(C*P_pred*C' + R);
x_hat = x_pred + (K*(y_meas_k - C*x_pred));
P_hat = (eye(size(P_pred,1)) - K*C)*P_pred;
end

```

C.5.4 linearization.m

```

function [A,B,C,D,u_LV001_0,u_LV002_0] = linearization(h1_0,h2_0,...
                                                    u_PA001_0,n,m)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Master Thesis:
% Moving Horizon Estimation for the Two-tank System by Greta Bekeryte.
% May, 2023
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Dynamic system equations and partial derivatives for linearization
% Tank 1 dynamic model
%tank1_model = (1/A1)*(f3-((Kv_LV001*f1)/3600)*sqrt((rho*g*...
% (h1+h_LV001))/100000));
% Tank 2 dynamic model
%tank2_model = (1/(0.07*h2+0.004))*(((Kv_LV001*f1)/3600)*...
% sqrt((rho*g*(h1+h_LV001))/100000)-(Kv_LV002*f2)/3600*...
% sqrt((rho*g*(h2+h_LV002))/100000));
run parameters.m

% Operating points Tank 1
f3_0 = interp1(u_PA001,q_PA001, u_PA001_0);
f3_delta = interp1(u_PA001,q_PA001,(u_PA001_0 +0.01));

f1_0 = (3600*f3_0)/(Kv_LV001*((g*rho*(h1_0 + h_LV001))/100000)^(1/2));
u_LV001_0 = interp1(f_LV001,u_LV001,f1_0);
f1_delta = interp1(u_LV001,f_LV001,(u_LV001_0 +0.01));

% Partial derivatives Tank 1
par_der_u_PA001 = (f3_delta - f3_0)/((u_PA001_0 + 0.01)-u_PA001_0);
par_der_u_LV001 = (f1_delta - f1_0)/((u_LV001_0 + 0.01)-u_LV001_0);
par_der_h1 = -(Kv_LV001*f1_0*g*rho)/(720000000*A1*((g*rho*(h1_0 +...
h_LV001))/100000)^(1/2)); % dg1/dh1
par_der_f1 = -(Kv_LV001*((g*rho*(h1_0 + h_LV001))/100000)^(...
(1/2))/(3600*A1); % dg1/df1
par_der_f3 = 1/A1; % dg1/df3

% Operating points Tank 2
f2_0 = (Kv_LV001*f1_0*((g*rho*(h1_0 + h_LV001))/100000)^(1/2))/...
(Kv_LV002*((g*rho*(h2_0 + h_LV002))/100000)^(1/2));
u_LV002_0 = interp1(f_LV002,u_LV002,f2_0);
f2_delta = interp1(u_LV002,f_LV002,(u_LV002_0 +0.01));

% Partial derivatives Tank 2
par_der_u_LV002 = (f2_delta - f2_0)/((u_LV002_0 + 0.01)-u_LV002_0);
par_der_h1_2 =(Kv_LV001*f1_0*g*rho)/(720000000*((7*h2_0)/100 +...
1/250))*((g*rho*(h1_0 + h_LV001))/100000)^(1/2)); % dg2/dh1
par_der_h2 = -(7*((Kv_LV001*f1_0*((g*rho*(h1_0 + h_LV001))/100000)^(1/2))/...

```

```

3600 - (Kv_LV002*f2_0*((g*rho*(h2_0 + ...
h_LV002))/100000)^(1/2))/3600))/(100*((7*h2_0)/100 + ...
1/250)^2) - (Kv_LV002*f2_0*g*rho)/(720000000*...
((7*h2_0)/100 + 1/250)*((g*rho*(h2_0 +...
h_LV002))/100000)^(1/2));% dg2/dh2
par_der_f1_2 = (Kv_LV001*((g*rho*(h1_0 + h_LV001))/100000)^(1/2))/...
(3600*((7*h2_0)/100 + 1/250));% dg2/df1
par_der_f2 = -(Kv_LV002*((g*rho*(h2_0 + h_LV002))/100000)^(1/2))/(3600*...
((7*h2_0)/100 + 1/250));% dg2/df2

%State space matrices
a11 = par_der_h1;
a12 = 0;
a21 = par_der_h1_2;
a22 = par_der_h2;
b11 = par_der_f1*par_der_u_LV001;
b12 = 0;
b21 = par_der_f1_2*par_der_u_LV001;
b22 = par_der_f2*par_der_u_LV002;
bv1 = par_der_f3*par_der_u_PA001;
bv2 = 0;

% System with 2 states
A = [a11 a12; a21 a22];           % state transition matrix
B = [b11 b12 bv1; b21 b22 bv2]; % input matrix when disturbance is known
C = [1 0; 0 1];                   % output matrix both h1 and h2 available
D = [0 0 0; 0 0 0];               % feed-through matrix 2 measurements available

C1 = [0 1];                        % output matrix only h2 measurement
D1 = [0 0 0];                      % feed-through matrix 1 measurement available

% System with 3 states
A3 = [a11 a12 bv1; a21 a22 bv2; 0 0 0]; % state transition matrix
B3 = [b11 b12; b21 b22; 0 0];          % input matrix when disturbance is known
C3 = [1 0 0; 0 1 0];                   % output matrix both h1 and h2 available
D3 = [0 0; 0 0];                       % feed-through matrix with 2
                                         % measurements available

C312 = [0 1 0];                       % output matrix h2 available
D31 = [0 0];                           % feed-through matrix 1 measurement
                                         % available

C311 = [1 0 0];                       % output matrix h1 available
% Rearranging system matrices according to number of states and measurements
if m == 1
    C = C1;
    D = D1;
end

```



```

if n == 3
    A = A3;
    B = B3;
    if m == 2
        C = C3;
        D = D3;
    elseif m == 1
        C = C312;
        D = D31;
    else
        disp('Number of measurements is out of bound.')
    end
end
end
end

```

C.5.5 ObjectiveFunction.m

```

function J = ObjectiveFunction(x,x_pred_k_N,u,y_meas,parameter_struct,P,type)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Master Thesis:
% Moving Horizon Estimation for the Two-tank System by Greta Bekeryte.
% May, 2023
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%x = reshape(x,[n,N]); %used only when Ga algorithm is applied

% Returns : scalar value of the objective function
% Inputs :
% x          -- The horizon vector of optimal state estimates
% x_pred_k_N -- The prediction vector of the states at time k - N
% u          -- The horizon vector of control inputs
% y_meas     -- The horizon vector of measured outputs
% parameter_struct -- The struct with parameters. For Linear MHE required
%                   parameters are the sytem matrices A, B, C, D and
%                   covariance matrices Q, R. For nonlinear MHE required
%                   parameters are all system parameters and covariance
%                   matrices Q and R. Other required parameters are: the
%                   sampling time, Ts, the horizon length, N, the
%                   number of states, n, and the number of measurements, m.
% P          -- Estimate error covariance matrix
% type       -- The string of type of the MHE, where L stands for
%                   linear and N stands for nonlinear MHE
N = parameter_struct.N;
Ts = parameter_struct.Ts;
n = parameter_struct.n;
m = parameter_struct.m;

```

```

if type == 'L' % the linear case
    A = parameter_struct.A; % state transition matrix
    B = parameter_struct.B; % input matrix
    C = parameter_struct.C; % output matrix
    D = parameter_struct.D; % feed-through matrix
    Q = parameter_struct.Q; % Covariance matrix of the estimated state noise
    R = parameter_struct.R; % Covariance matrix of output measuremen noise

%Set points
x1_0 = parameter_struct.h1_0;
x2_0 = parameter_struct.h2_0;
u1_0 = parameter_struct.u_LV001_0;
u2_0 = parameter_struct.u_LV002_0;

if n == 2
    u3_0 = parameter_struct.u_PA001_0;
elseif n == 3
    x3_0 = parameter_struct.u_PA001_0; % n=3
else
    disp('Number of states is out of bound.')
end

J_arrival = (1/2)*((x(:,1)-x_pred_k_N)'inv(P)*(x(:,1)-x_pred_k_N));

for i = 1:N
    x_i = x(:,i); % Both states, at time i
    u_i = u(:,i);
    y_meas_i = y_meas(:,i); % all measurements, at time i

    if i <= N-1
        x_i_next = x(:,i+1);

        if n == 2
            dx_dt_i = A*(x_i - [x1_0; x2_0]) + B*(u_i -[u1_0; u2_0; u3_0]);
        elseif n == 3
            dx_dt_i = A*(x_i - [x1_0; x2_0; x3_0]) + B*(u_i -[u1_0; u2_0]);
        else
            disp('Number of states is out of bound.')
        end
        w_i = x_i_next - (Ts*dx_dt_i + x_i);
    end
    J_cost_i = (1/2)*(w_i'inv(Q)*w_i) + (1/2)*((y_meas_i - ...
        C*x_i)'inv(R)*(y_meas_i - C*x_i));
    J_i = J_cost_i + J_arrival;

    % Update for i+1
    J_arrival = J_i;
end
end

```

```

J = J_i;

elseif type == 'N' % The nonlinear case
    A1 = parameter_struct.A1;
    Kv_LV001 = parameter_struct.Kv_LV001;
    Kv_LV002 = parameter_struct.Kv_LV002;
    rho = parameter_struct.rho;
    g = parameter_struct.g;
    h_LV001 = parameter_struct.h_LV001;
    h_LV002 = parameter_struct.h_LV002;
    u_LV001 = parameter_struct.u_LV001;
    u_LV002 = parameter_struct.u_LV002;
    f_LV001 = parameter_struct.f_LV001;
    f_LV002 = parameter_struct.f_LV002;
    Q = parameter_struct.Q;
    R = parameter_struct.R;
    if n == 2
        u_PA001 = parameter_struct.u_PA001;
        q_PA001 = parameter_struct.q_PA001;
    end

J_arrival = (1/2)*((x(:,1)-x_pred_k_N)'*inv(P)*(x(:,1)-x_pred_k_N));

for i = 1:N
    x_i = x(:,i); % All states, at time i
    u_i = u(:,i);

    f1_i = interp1(u_LV001,f_LV001,u_i(1));
    f2_i = interp1(u_LV002,f_LV002,u_i(2));

    y_meas_i = y_meas(:,i); % all measurements, at time i
    if m == 1
        v2_i = y_meas_i - x_i(2);
        v_i = v2_i;
    elseif m == 2
        v1_i = y_meas_i(1) - x_i(1);
        v2_i = y_meas_i(2) - x_i(2);
        v_i = [v1_i; v2_i];
    end

    if i <= N-1
        x_i_next = x(:,i+1);

        if n == 2
            f3_i = interp1(u_PA001,q_PA001,u_i(3));
            dx1_dt_i = (1/A1)*(f3_i-((Kv_LV001*f1_i)/3600)*...
                sqrt((rho*g*(x_i(1)+h_LV001))/100000));
            dx2_dt_i = (1/(0.07*x_i(2)+0.004))*(((Kv_LV001*f1_i)/3600)*...

```

```

        sqrt((rho*g*(x_i(1)+h_LV001))/100000)-...
        (Kv_LV002*f2_i)/3600*sqrt((rho*g*...
        (x_i(2)+h_LV002))/100000));
w1_i = x_i_next(1) - (Ts*dx1_dt_i + x_i(1));
w2_i = x_i_next(2) - (Ts*dx2_dt_i + x_i(2));
w_i = [w1_i; w2_i];

elseif n == 3
    dx1_dt_i = (1/A1)*(x_i(3)-((Kv_LV001*f1_i)/3600)*...
        sqrt((rho*g*(x_i(1)+h_LV001))/100000));
    dx2_dt_i = (1/(0.07*x_i(2)+0.004))*(((Kv_LV001*f1_i)/3600)*...
        sqrt((rho*g*(x_i(1)+h_LV001))/100000)-...
        (Kv_LV002*f2_i)/3600*sqrt((rho*g*...
        (x_i(2)+h_LV002))/100000));
    dx3_dt_i = 0;
    w1_i = x_i_next(1) - (Ts*dx1_dt_i + x_i(1));
    w2_i = x_i_next(2) - (Ts*dx2_dt_i + x_i(2));
    w3_i = x_i_next(3) - (Ts*dx3_dt_i + x_i(3));
    w_i = [w1_i; w2_i; w3_i];
else
    disp('Number of states is out of bound.')
end
end

J_cost_i = (1/2)*(w_i'*inv(Q)*w_i) + (1/2)*(v_i'*inv(R)*v_i);
J_i = J_cost_i + J_arrival;

% Update for i+1
J_arrival = J_i;
end
J = J_i;

else
    disp('The type of the MHE chosen do not exist.');
```

C.5.6 LMHE_simulink.m

```

function [output] = LMHE_simulink(input)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Master Thesis:
% Moving Horizon Estimation for the Two-tank System by Greta Bekeryte.
% May, 2023
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
port_attribute = evalin('base','port_attribute');
```

```

if port_attribute == 2
    parameter_struct = evalin('base','parameter_struct');
    h1_meas_horizon = evalin('base','h1_meas_horizon');
    h2_meas_horizon = evalin('base','h2_meas_horizon');
    h1_hat_horizon = evalin('base','h1_hat_horizon');
    h2_hat_horizon = evalin('base','h2_hat_horizon');
    u_PA001_hat_horizon = evalin('base','u_PA001_hat_horizon');
    x_hat_horizon = evalin('base','x_hat_horizon');
    P_k_N_arr = evalin('base','P_k_N_arr');
    P_previous = evalin('base','P_previous');
    x_pred_k_N = evalin('base','x_pred_k_N');
    u_horizon = evalin('base','u_horizon');

    u_PA001_k = input(1);
    u_LV001_k = input(2);
    u_LV002_k = input(3);
    h1_meas_k = input(4);
    h2_meas_k = input(5);
    k = input(6);

    A = parameter_struct.A; % state transition matrix
    B = parameter_struct.B; % input matrix
    C = parameter_struct.C; % output matrix
    D = parameter_struct.D; % feed-through matrix
    A_disc = parameter_struct.A_disc;
    Q = parameter_struct.Q;
    R = parameter_struct.R;
    n = parameter_struct.n;
    m = parameter_struct.m;
    N = parameter_struct.N;
    Ts = parameter_struct.Ts;
    x_lb = parameter_struct.x_lb;
    x_ub = parameter_struct.x_ub;
    %Set points
    h1_0 = parameter_struct.h1_0;
    h2_0 = parameter_struct.h2_0;
    u_PA001_0 = parameter_struct.u_PA001_0;
    u_LV001_0 = parameter_struct.u_LV001_0;
    u_LV002_0 = parameter_struct.u_LV002_0;

    if n == 3
        u_k = [u_LV001_k; u_LV002_k];

    elseif n == 2
        % when u_PA001_k is known disturbance
        u_k = [u_LV001_k; u_LV002_k; u_PA001_k];
    end
end

```

```

if m == 2
    h_meas_k = [h1_meas_k; h2_meas_k];
elseif m ==1
    h_meas_k = h2_meas_k;
end

%Update horizons
u_horizon = [u_horizon(:,2:N), u_k]; %sliding horizon to the right
h2_meas_horizon = [h2_meas_horizon(2:N), h2_meas_k];
if m == 2
    h1_meas_horizon = [h1_meas_horizon(2:N), h1_meas_k];
    h_meas_horizon = [h1_meas_horizon; h2_meas_horizon];
elseif m == 1
    h_meas_horizon = h2_meas_horizon;
end

if k <=N
    [x_hat, P_previous] = KalmanFilter(x_pred_k_N,P_previous,h_meas_k,u_k,...
        parameter_struct,Q,R);
    %Storing
    h1_hat_horizon(1,k) = x_hat(1);
    h2_hat_horizon(1,k) = x_hat(2);
    if n==3
        u_PA001_hat_horizon(1,k) = x_hat(3);
        x_hat_horizon = [h1_hat_horizon; h2_hat_horizon; ...
            u_PA001_hat_horizon];
    else
        x_hat_horizon = [h1_hat_horizon; h2_hat_horizon];
    end

    P_k_N_arr = cat(2,P_k_N_arr,P_previous);
    x_pred_k_N = [x_hat(1);x_hat(2)];
    if n == 3
        x_pred_k_N = [x_hat(1);x_hat(2);x_hat(3)];
    end

    %Plot
    output(1) = h1_hat_horizon(k); %h1_hat_k
    output(2) = h2_hat_horizon(k); %h2_hat_k
    output(3) = u_PA001_k;
    if n == 3
        % Estimating f3, plotting u_PA001
        output(3) = u_PA001_hat_horizon(k); %u_PA001_hat_k
    end
end

if k > N %enough samples for the first horizon
    % Using model to predict h_k based on estimated state x_k-1
    if n == 3

```



```

    %Updating with optimal solution, adding setting point
    h1_hat_horizon = x_hat_optimal(1,:);
    h2_hat_horizon = x_hat_optimal(2,:);
    if n == 3
        u_PA001_hat_horizon = x_hat_optimal(3,:);
    end
    %Plot
    output(1) = h1_hat_horizon(end); %h1_hat_k
    output(2) = h2_hat_horizon(end); %h2_hat_k
    output(3) = u_PA001_k;
    if n == 3
        % Estimating f3, plotting u_PA001
        output(3) = u_PA001_hat_horizon(end); %u_PA001_hat_k
    end
end
end
assignin('base','h1_hat_horizon',h1_hat_horizon);
assignin('base','h2_hat_horizon',h2_hat_horizon);
assignin('base','u_PA001_hat_horizon',u_PA001_hat_horizon);
assignin('base','h1_meas_horizon',h1_meas_horizon);
assignin('base','h2_meas_horizon',h2_meas_horizon);
assignin('base','x_hat_horizon',x_hat_horizon);
assignin('base','u_horizon',u_horizon);
assignin('base','P_previous',P_previous);
assignin('base','P_k_N_arr',P_k_N_arr);
assignin('base','x_pred_k_N',x_pred_k_N);
end
end

```

C.5.7 NMHE_simulink.m

```

function [output] = NMHE_simulink(input)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Master Thesis:
% Moving Horizon Estimation for the Two-tank System by Greta Bekeryte.
% May, 2023
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
port_attribute = evalin('base','port_attribute');
if port_attribute == 1
    parameter_struct = evalin('base','parameter_struct');
    h1_meas_horizon = evalin('base','h1_meas_horizon');
    h2_meas_horizon = evalin('base','h2_meas_horizon');
    h1_hat_horizon = evalin('base','h1_hat_horizon');
    h2_hat_horizon = evalin('base','h2_hat_horizon');
    f3_hat_horizon = evalin('base','f3_hat_horizon');
    x_hat_horizon = evalin('base','x_hat_horizon');
    P_k_N_arr = evalin('base','P_k_N_arr');

```



```
P_previous = evalin('base','P_previous');
x_pred_k_N = evalin('base','x_pred_k_N');
u_horizon = evalin('base','u_horizon');

u_PA001_k = input(1);
u_LV001_k = input(2);
u_LV002_k = input(3);
h1_meas_k = input(4);
h2_meas_k = input(5);
k = input(6);

A1 = parameter_struct.A1;
Kv_LV001 = parameter_struct.Kv_LV001;
Kv_LV002 = parameter_struct.Kv_LV002;
rho = parameter_struct.rho;
g = parameter_struct.g;
h1_min = parameter_struct.h1_min;
h1_max = parameter_struct.h1_max;
h2_min = parameter_struct.h1_min;
h2_max = parameter_struct.h1_max;
h_LV001 = parameter_struct.h_LV001;
h_LV002 = parameter_struct.h_LV002;
u_LV001 = parameter_struct.u_LV001;
u_LV002 = parameter_struct.u_LV002;
f_LV001 = parameter_struct.f_LV001;
f_LV002 = parameter_struct.f_LV002;
u_PA001 = parameter_struct.u_PA001;
q_PA001 = parameter_struct.q_PA001;
Q = parameter_struct.Q;
R = parameter_struct.R;
n = parameter_struct.n;
m = parameter_struct.m;
N = parameter_struct.N;
Ts = parameter_struct.Ts;
C = parameter_struct.C;
x_lb = parameter_struct.x_lb;
x_ub = parameter_struct.x_ub;

if n == 3
    u_k = [u_LV001_k; u_LV002_k];
elseif n == 2
    u_k = [u_LV001_k; u_LV002_k; u_PA001_k];
end

% Process derivatives and integration
f3_k = interp1(u_PA001,q_PA001, u_PA001_k);
f1_k = interp1(u_LV001,f_LV001,u_LV001_k);
f2_k = interp1(u_LV002,f_LV002,u_LV002_k);
```

```

if m == 2
    h_meas_k = [h1_meas_k; h2_meas_k];
elseif m ==1
    h_meas_k = h2_meas_k;
end

%Update horizons
u_horizon = [u_horizon(:,2:N), u_k]; %sliding horizon to the right
h2_meas_horizon = [h2_meas_horizon(2:N), h2_meas_k];
if m == 2
    h1_meas_horizon = [h1_meas_horizon(2:N), h1_meas_k];
    h_meas_horizon = [h1_meas_horizon; h2_meas_horizon];
elseif m == 1
    h_meas_horizon = h2_meas_horizon;
end

%initial Arrival cost update given measurements up to time k<=N
if k <=N
    if n == 3
        f3_pred_k_N = interp1(u_PA001,q_PA001,x_pred_k_N(3));
        x_pred_k_N = [x_pred_k_N(1:2);f3_pred_k_N];
    end
    [x_hat, P_previous] = ExtendedKalmanFilter(x_pred_k_N,P_previous,...
                                             h_meas_k,u_k,...
                                             parameter_struct,Q,R);

    %Storing
    h1_hat_horizon(1,k) = x_hat(1);
    h2_hat_horizon(1,k) = x_hat(2);

    if n == 3
        f3_hat_horizon(1,k) = x_hat(3);
        x_hat_horizon = [h1_hat_horizon; h2_hat_horizon; f3_hat_horizon];
    else
        x_hat_horizon = [h1_hat_horizon; h2_hat_horizon];
    end

    P_k_N_arr = cat(2,P_k_N_arr,P_previous);
    x_pred_k_N = [x_hat(1);x_hat(2)];
    if n == 3
        x_pred_k_N = [x_hat(1);x_hat(2);interp1(q_PA001,u_PA001,x_hat(3))];
    end
    %Plot
    output(1) = h1_hat_horizon(k); %h1_hat_k
    output(2) = h2_hat_horizon(k); %h2_hat_k
    output(3) = u_PA001_k;
    if n == 3
        % Estimating f3, plotting u_PA001

```

```

        output(3) = interp1(q_PA001,u_PA001,f3_hat_horizon(k));
        %u_PA001_hat_k
    end
end

if k > N %enough samples for the first horizon

    % Using model to predict x_k based on estimated state x_k-1
    if n == 3
        M_dh1_dt_k = (1/A1)*(x_hat_horizon(3,N)-((Kv_LV001*f1_k)/3600)*...
            sqrt((rho*g*(x_hat_horizon(1,N)+h_LV001))/100000));
    elseif n == 2
        M_dh1_dt_k = (1/A1)*(f3_k-((Kv_LV001*f1_k)/3600)*...
            sqrt((rho*g*(x_hat_horizon(1,N)+...
            h_LV001))/100000));
    end
    M_dh2_dt_k = (1/(0.07*x_hat_horizon(2,N)+0.004))*(((Kv_LV001*f1_k)/...
        3600)*sqrt((rho*g*(x_hat_horizon(1,N)+...
        h_LV001))/100000)-(Kv_LV002*f2_k)/3600*sqrt((rho*g*...
        (x_hat_horizon(2,N)+h_LV002))/100000));
    M_df3_dt_k = 0;
    h1_pred_k = Ts*M_dh1_dt_k + x_hat_horizon(1,N);
    h2_pred_k = Ts*M_dh2_dt_k + x_hat_horizon(2,N);

    if n == 3
        f3_pred_k = Ts*M_df3_dt_k + x_hat_horizon(3,N);
    end

    %Physical constraints
    if h1_pred_k < h1_min
        h1_pred_k = h1_min;
    end
    if h2_pred_k < h2_min
        h2_pred_k = h2_min;
    end
    if h1_pred_k > h1_max
        h1_pred_k = h1_max;
    end
    if h2_pred_k > h2_max
        h2_pred_k = h2_max;
    end
    if n == 3
        if f3_pred_k < 0
            f3_pred_k = 0;
        end
        if f3_pred_k > 0.000333
            f3_pred_k = 0.000333;
        end
    end
end

```

```

end

% Updating estimate matrix, sliding horizon to the right and adding
% predicted x_k
if n == 3
    x_hat_horizon = [[h1_hat_horizon(2:N),h1_pred_k];...
                    [h2_hat_horizon(2:N),h2_pred_k];...
                    [f3_hat_horizon(2:N),f3_pred_k]];
elseif n == 2
    x_hat_horizon = [[h1_hat_horizon(2:N),h1_pred_k];...
                    [h2_hat_horizon(2:N),h2_pred_k]];
end

% updating error covariance matrix used in Arrival cost
if n == 3
    u_PA001_tr = interp1(q_PA001,u_PA001,x_hat_horizon(3,end));
    A = Amatrix(x_hat_horizon(1,end),x_hat_horizon(2,end),...
                u_PA001_tr,u_LV001_k,u_LV002_k,parameter_struct);
elseif n == 2
    A = Amatrix(x_hat_horizon(1,end),x_hat_horizon(2,end),...
                u_PA001_k,u_LV001_k,u_LV002_k,parameter_struct);
end

P = A*P_previous*A' - (A*P_previous*C'*(C*P_previous*C' + ...
    R)^(-1)*C*P_previous*A') + Q;
P_k_N_arr = cat(2,P_k_N_arr,P);
P_previous = P;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x_pred_k_N = x_hat_horizon(:,1); %used in arrival cost function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
idx = size(P_k_N_arr, 2) - N*n + 1;
% used at time k, but calculated at time k-N
P_k_N = P_k_N_arr(:, idx:idx+n-1);

%%% Objective function handle %%%
objective_function_handle = @(x_hat_horizon) ...
    ObjectiveFunction(x_hat_horizon,...
                    x_pred_k_N,...
                    u_horizon,...
                    h_meas_horizon,...
                    parameter_struct,...
                    P_k_N,'N');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
options = optimoptions(@fmincon,'Display','none','Algorithm',...
    'active-set');
%, 'OptimalityTolerance',1e-10,'StepTolerance',1e-10);

x_hat_optimal = fmincon(objective_function_handle,x_hat_horizon,...

```

```

[], [], [], [], x_lb, x_ub, [], options);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Updating with optimal solution, adding setting point
h1_hat_horizon = x_hat_optimal(1,:);
h2_hat_horizon = x_hat_optimal(2,:);
if n == 3
    f3_hat_horizon = x_hat_optimal(3,:);
end
%Plot
output(1) = h1_hat_horizon(end); %h1_hat_k
output(2) = h2_hat_horizon(end); %h2_hat_k
output(3) = u_PA001_k;
if n == 3
    % Estimating f3, plotting u_PA001
    output(3) = interp1(q_PA001, u_PA001, f3_hat_horizon(end));
    %u_PA001_hat_k
end
end
end
assignin('base', 'h1_hat_horizon', h1_hat_horizon);
assignin('base', 'h2_hat_horizon', h2_hat_horizon);
assignin('base', 'f3_hat_horizon', f3_hat_horizon);
assignin('base', 'h1_meas_horizon', h1_meas_horizon);
assignin('base', 'h2_meas_horizon', h2_meas_horizon);
assignin('base', 'x_hat_horizon', x_hat_horizon);
assignin('base', 'u_horizon', u_horizon);
assignin('base', 'P_previous', P_previous);
assignin('base', 'P_k_N_arr', P_k_N_arr);
assignin('base', 'x_pred_k_N', x_pred_k_N);
end
end

```

C.5.8 parameters.m

```

%% Parameters of two-tank system
%clear all; close all
% The following code is from ELE320 Refuleringsteknikk that describes
% Pump and valve characteristics and system parameters.

rho = 1000;           % water density [kg/m^3]
g = 9.81;            % gravitational acceleration [m/s ^2]
c_p = 4200;          % water heat capacity [j/kg*K]
Kv_LV001 = 11.25;    % valve constant LV001 [m^3/h] @1bar pressure drop
h_LV001 = 0.05;      % height LV001 [m]
h1_max = 1;          % max height tank 1 [m]
h1_min = 0.13;       % min height tank 1 [m]

```

```
A1 = 0.01;           % area tank 1 [m ^2]
Kv_LV002 = 11.25;    % ventilkonstant LV002 [m3/h]
h_LV002 = 0.25;      % height between the bottom of the tank 2 and LV002
h2_max = 0.4;        % max height tank 2 [m]
h2_min = 0.02;       % min height tank 2 [m]

% Pump characteristic
u_PA001 = [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 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; % liters / time -> m3/s q_PA001=f3(u_PA001)

%figure(1)
%plot(u_PA001,q_PA001,'*-')
%title('Pump characteristic ')
%xlabel('Control signal u_{ PA001 }(t) to pump PA001 ')
%ylabel('Volume flow q_{ PA001 }(t) through PA001 [m ^3/ s]')

% Ventilkarakteristikk
u_LV001 = 0:0.03:1;
f_LV001 = (exp( u_LV001.^1.2)-1)/(exp(1)-1);
u_LV002 = u_LV001;
f_LV002 = f_LV001;

%figure(2)
%plot (u_LV00i,f_LV001,'*-')
%title('Valve characteristic for LV001 og LV002 ')
%xlabel('Control signal u_{ LV00i }(t)')
%ylabel('f(u_{ LV00i }(t))')

% End of the code from ELE320 Reguleringssteknikk
```

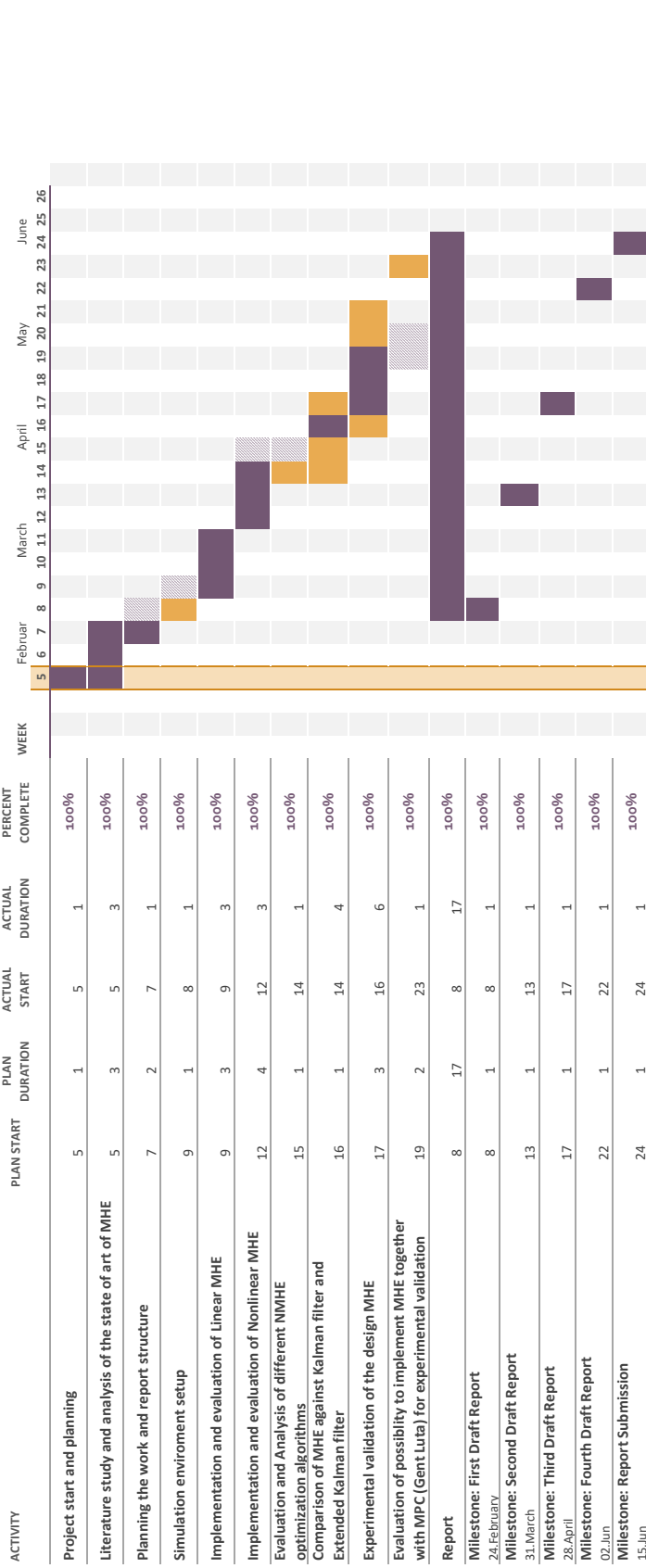
Appendix D


Project Poster and Project Plan

Moving Horizon Estimation for the two-tank system

Greta Bekeryte, 234929

Period Highlight: 5






Universitetet
i Stavanger

Moving Horizon Estimation for the Two-tank System

Greta Bekeryté, Master Thesis
Universitetet i Stavanger



Universitetet
i Stavanger

Abstract

Information on the variables that uniquely defines the state of the system at any given time is the essential condition for effective monitoring and control of a process. One of the substantial disadvantages of commonly used estimation techniques such as Kalman filter or Luenberger observer is the inability to directly address or incorporate constraints in the optimization process. Moving Horizon Estimation (MHE) is an optimization-based approach that considers the constraints as part of the optimization problem.

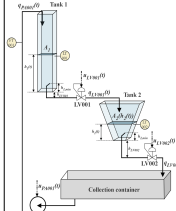


Figure 1. A simplified version of the two-tank system schematic sketch.

MHE can improve the state estimation performance, however, at the cost of increased computational load and is thus, more suitable to be applied to processes where there is availability for enough computational resources, e.g., in the case of systems with slow dynamics. This project aims at designing and implementing an MHE-based state estimator to observe the state of the two-tank system (shown in Fig. 1) available in the laboratory at KE E-458 and compare its performance against more conventional state estimation techniques.

Moving Horizon Estimation

The principle of Moving Horizon Estimation is to continuously update the estimates of the states by solving an optimization problem using a finite sequence of most recent measurements and control inputs. Hence, the optimization problem is solved over a fixed-size moving window. This is illustrated in Fig. 2.

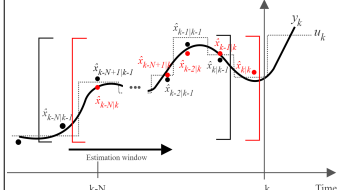


Figure 2. Moving Horizon Estimation.

The MHE objective function is formulated as a least-squares estimation problem and is given by:

$$\min_{x_{k-N}, \dots, x_k} \frac{1}{2} \sum_{i=k-N}^{k-1} \|x_{i+1} - f(x_i, u_i)\|_{Q_i}^2 + \frac{1}{2} \sum_{i=k-N}^k \|y_i - h(x_i)\|_{R_i}^2 + \frac{1}{2} \|x_{k-N} - \bar{x}_{k-N}\|_{P_{k-N}}^2$$

Dynamical system model error Measurement error Arrival cost

Subject to:

$$x_{k-N} - \bar{x}_{k-N} \in \bar{X},$$

$$x_{i+1} - f(x_i, u_i) \in \mathcal{W},$$

$$y_i - h(x_i) \in \mathcal{V}$$

The objective is to minimize the influence of the measurement error, the non-measured process disturbance and the arrival cost (summarizes past information outside the estimation window) over the estimation horizon subject to the constraints of the system. The constraints can be modelled as equality, inequality or box constraints. Thus, the optimization problem is solved by minimizing a weighted sum of squared errors of initial condition, system model dynamics and measurements in the time interval $[k-N, k]$. The solution of the objective function yields the sequence of the optimal state estimates, $\hat{x} = [\hat{x}_{k-N}^T, \dots, \hat{x}_k^T]^T$.

Implementation

Figure 3, illustrates the concept behind one time instance of MHE.

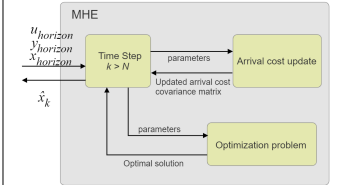


Figure 3. An overview of one Moving Horizon Estimation iteration.

At each iteration, a new initial state is the second element from the previous estimation horizon, thus, the horizons are shifted continuously to the right, taking into consideration the newest measurements and control inputs while dropping the oldest as shown in the Fig. 2. The MHE optimization problem is solved repeatedly at each time instance using nonlinear programming solver. The arrival cost covariance matrix, P_{k-N} , is updated recursively subject to initial condition, P_0 , using Kalman filter covariance update equation.

In this project two variants of MHE are considered, linear moving horizon estimation (LMHE) and nonlinear moving horizon estimation (NMHE). In a LMHE an objective function formulation involves describing the system dynamics using linear equations. Since the two-tank system is nonlinear, the LMHE uses linearized two-tank dynamical model, whereas the NMHE incorporates nonlinear differential equations. The state variables for the two-tank system, the water level in tank 1, the water level in tank 2 and the water flow from the pump are physically constrained. The water level is bounded by the tank size while the water flow rate is limited by the pump capacity. Hence, the two-tank MHE problem is subject to the box constraints, i.e., the lower and upper bounds of the state variables.

The LMHE and NMHE are implemented in Matlab. The optimization problem is solved using in-built Matlab function `fmincon`.

Results

The performance of LMHE is compared with widely used Kalman filter, while NMHE is compared with Extended Kalman filter. In addition, different estimators are evaluated at four different two-tank dynamical model scenarios:

- Estimation of two states (h_1 and h_2) given two measurements (h_1 and h_2), and one measurement (h_2).
 - Estimation of three states (h_1 , h_2 and u_{p0001}) given two measurements (h_1 and h_2), and one measurement (h_2).
- Some of the results are presented in figures 4, 5 and 6. The estimation by MHE is done with horizon length $N = 10$.

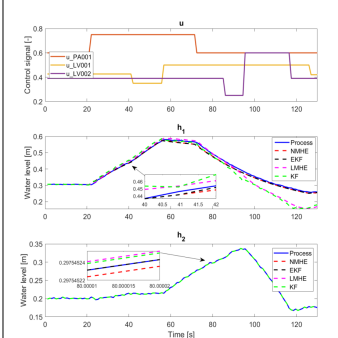


Figure 4. Estimation of h_1 and h_2 given the measurement h_2 .

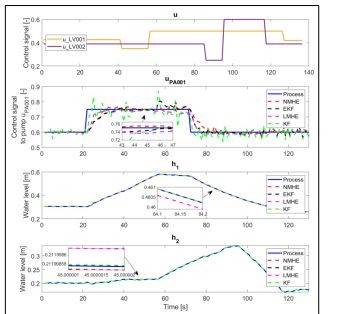


Figure 5. Estimation of h_1 , h_2 and u_{p0001} given the measurements h_1 and h_2 .

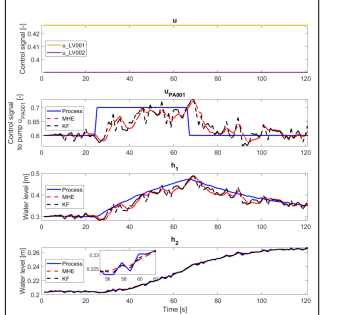


Figure 6. Estimation of h_1 , h_2 and u_{p0001} given the measurement h_2 .

Conclusions

The overall performance of all four estimators is good. However, as expected, due to linearization errors the estimation error for LMHE and Kalman filter increases when the system deviates from the original operating point. As in the Fig. 4, estimation error becomes even bigger if only one measurement is available. The confidence in the model is high resulting in high process noise related errors.

In general, the MHE performance is better than Kalman filters (KF and EKF) in estimating three states. The resulting estimate of the disturbance, u_{p0001} , is less noisy. However, at the cost of water level estimates accuracy.

The LMHE and KF estimation of the three states given one measurement, presented in Fig. 6, is noisy, delayed and inaccurate. This is due to low observability and relative high measurement noise. But it is evident that the LMHE performs better than KF. The performance of the NMHE and EKF is worse. In contrast to LMHE, NMHE introduces additional complexity associated with nonconvexity, along with the uncertainties arising from measurement noise and low observability.

In brief, MHE is powerful, and yet, complex and computationally inefficient estimation technique. In addition, the MHE performance relies on good optimization performance, adequate parameters and right choice of weighting matrices. The LMHE is simpler and more efficient than NMHE. If the operating range of the two-tank system is narrow, the LMHE is the preferred choice.

Bibliography

- [1] User Sdo. File:simplex-method-3-dimensions.png, 2006. URL <https://en.wikipedia.org/wiki/File:Simplex-method-3-dimensions.png#filelinks>. Accessed: 2023-04-20.
- [2] Tormod Drengstig. Totankøving 1. motivasjon og modellering, 2018. ELE320 Reguleringssteknikk.
- [3] C.V. Rao, J.B. Rawlings, and Jay H. Lee. Constrained linear state estimation - a moving horizon approach. *Automatica*, 37:1619–1628, 2001.
- [4] Masoud Soroush. State and parameter estimations and their applications in process control. *Computers & Chemical Engineering*, 23(Issue 2):229–245, 1998.
- [5] Max Boegli. *Real-Time Moving Horizon Estimation for Advanced Motion Control. Application to Friction State and Parameter Estimation*. PhD thesis, KU Leuven – Faculty of Engineering Science, 2014.
- [6] Rudolf E. Kalman. On the general theory of control systems. *IFAC Proceedings Volumes*, 1(Issue 1):491–502, 1960.
- [7] Rudolf E. Kalman, Peter L. Falb, and Michael A. Arbib. *Topics in Mathematical System Theory*. International series in pure and applied mathematics. McGraw-Hill, 1969.
- [8] Chi-Tsong Chen. *Linear System Theory and Design*. The Oxford Series in Electrical and Computer Engineering. Oxford University Press, Inc., third edition, 1999.
- [9] Eduardo D. Sontag. *Mathematical Control Theory*. Texts in Applied Mathematics. Springer New York, NY, second edition edition, 2013.

- [10] David G. Luenberger. Observing the state of a linear system. *IEEE Transactions on Military Electronics*, 8(Issue 2:74–80, 1964).
- [11] Rudolf E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D:35–45, 1960).
- [12] Robert G. Brown. *Introduction to random signal analysis and Kalman filtering*. John Wiley Sons, Inc., 1983.
- [13] Greg Welch and Gary Bishop. An introduction to the kalman filter. *Department of Computer Science University of North Carolina*, page 16, 1999.
- [14] James B. Rawlings, David Q. Mayne, and Moritz M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob-Hill, 2017.
- [15] James B. Rawlings and Luo Ji. Optimization-based state estimation: current status and some new results. *Journal of Process Control*, 22:1439–1444, 2012.
- [16] C.V. Rao, J.B. Rawlings, and D.Q. Mayne. Constrained state estimation for nonlinear discrete-time systems: Stability and moving horizon approximations. *IEEE transactions on automatic control*, 48(2):246–258, 2003.
- [17] S Ungarala. Computing arrival cost parameters in moving horizon estimation using sampling based filters. *Journal of Process Control*, 19(9):1576–1588, 2009.
- [18] Douglas G. Robertson, Jay H. Lee, and James B. Rawlings. A moving horizon-based approach for least-squares estimation. *AIChE Journal*, 42(Issue 8):2209–2224, 1996.
- [19] Thomas F. Edgar, David M. Himmelblau, and Leon Lasdon. *Optimization of Chemical Processes*. McGraw-Hill Chemical Engineering Series. McGraw-Hill, second edition edition, 2001.
- [20] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, seventh edition edition, 2009.
- [21] Hédi Nabli. An overview on the simplex algorithm. *Applied Mathematics and Computation*, 210(2):479–489, 2009.
- [22] MathWorks. Optimization toolbox, 2023. URL https://se.mathworks.com/help/optim/index.html?s_tid=CRUX_lftnav. Accessed: 2023-04-20.

-
- [23] MathWorks. Documentation, 2023. URL https://se.mathworks.com/help/index.html?s_tid=CRUX_lftnav. Accessed: 2023-04-10.
- [24] MathWorks. Documentation, 2023. URL https://se.mathworks.com/help/optim/ug/fmincon.html#responsive_offcanvas. Accessed: 2023-04-15.
- [25] Thomas Coleman, Mary Ann Branch, and Andrew Grace. *Optimization Toolbox For Use with Matlab*. User's Guide. The MathWorks Inc., second version edition, 1999.
- [26] MathWorks. *Global Optimization Toolbox R2011b Matlab*. User's Guide. The MathWorks Inc., 2011.
- [27] MathWorks. Improving performance with parallel computing, 2023. URL <https://se.mathworks.com/help/optim/ug/improving-performance-with-parallel-computing.html>. Accessed: 2023-05-15.
- [28] Graham Clifford Goodwin, Jose A. De Dona, Maria M. Seron, and Xiang W. Zhuo. Lagrangian duality between constrained estimation and control. *Automatica*, 41(6): 935–944, 2005.