University College of Southeast Norway

www.usn.no

FMH606 Master's Thesis 2017
Industrial IT and Automation

Improved flow management of Lake Toke

Kristian Dønheim Kvam

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Porsgrunn

Course: FMH606 Master's Thesis 2017
Title: *Improved flow management of Lake Toke*
Pages: *71*
Keywords: *Hydro power, MPC, SQL, MATLAB, Simulation and Modeling*

Student: *Kristian Dønheim Kvam*
Supervisor: *Bernt Lie*
External partner: *Beathe Furenes, Skagerak Energi*
Availability: *Open*

Approved for archiving: ____________________
(Bernt Lie)

Summary:
The Telemark University College (TUC) Flood server is an advisory system that assist in the regulation of the flood gates at Dalsfos hydroelectric dam. A large flood in September 2015 revealed that the model and data storage of the system had to be updated.
Historical measurement data was used to tune and test the model. The updated model showed improvement from the old model during simulation. A newly developed database was successfully integrated into the server for data storage.
The updated TUC flood server is ready to begin live testing in Skageraks system. Some work remains in regards to making the system more robust and user-friendly.

Preface

This is the final report of the master thesis as part of the Industrial IT and Automation course at the University College of South East Norway, Porsgrunn.
Due to time constraints, task 3. "Improve the efficiency of the MPC algorithm if possible" has not been completed.

I would like to offer thanks to Beathe Furenes and Åsmund Hasaas from Skagerak Energi for their assistance during the thesis work.

Porsgrunn, 22nd May 2017

Kristian Dønheim Kvam

Contents

Preface 3

Contents 5

1 Introduction 7
1.1 Background . . . 7
1.2 Previous work . . . 8
1.3 Overview of report . . . 8

2 System overview 10

3 Model 12
3.1 Updated model summary . . . 12
3.2 Historical data . . . 14
3.3 Gate flow parameter updating . . . 15
3.4 Inter-compartmental flow model reworking . . . 15
3.5 State estimator parameter tuning . . . 18
3.6 Model verification . . . 19

4 TUC Flood server 21
4.1 Data Handler . . . 22
4.2 Database . . . 24
4.3 MPC controller . . . 26
4.4 Simulator . . . 27

5 Discussion 29
5.1 Data Handler . . . 29
5.2 Database . . . 29
5.3 Model tuning . . . 30
5.3.1 Gate flow parameter updating . . . 30
5.3.2 Inter-compartmental flow model reworking . . . 30
5.3.3 Inter-compartmental flow model reworking MATLAB script . . . 30
5.4 Model verification . . . 31
5.5 Input data handling . . . 31
5.6 MPC controller . . . 32

6 Conclusions **33**
6.1 Future work . . . 33

References **34**

A Signed task description **35**

B Full database diagram **39**

C Vfit.m **40**

D getModelData.m **42**

E gateV.m **43**

F turbineV.m **44**

G paramFit.m **45**

H dhEstimator.m **46**

I tokeSimulator.m **47**

J findDataFuture.m **48**

K findDataFuture.m **51**

L manageToke.m **53**

M modLakeToke.m **55**

N prepTokeState.m **57**

O databaseGetFileNo.m **59**

P databaseRead.m **60**

Q storeOutput.m **62**

R matlab2gs2.m **64**

S databaseOutputFile.m **67**

T databaseWrite.m **68**

U Simulator.m **69**

Nomenclature

Catchment	The amount of rain, snow, hail, etc., that has fallen at a given place within a given period.
CSV file	Comma Separated Value: A file formatting used, amongst others, by MATLAB for storing and reading data.
GS2 file	A file formatting used in Skagerak Energis internal systems.
MATLAB	Matrix Laboratory, software by Mathworks.
Hydrologist	"Hydrology is the branch of science concerned with the properties of the earth's water, and especially its movement in relation to land." [1]

1 Introduction

The Dalsfos hydro power plant is located at the outlet of Lake Toke in Telemark, Norway. The waterways leading to Lake Toke, the rivers and lakes downstream of it and down to the ocean are known as the Kragerø waterway. Dalsfos is the first of five hydro power plants downstream of Lake Toke. Dalsfos consists of a dam with intakes to three turbines, two flood gates, and the power station itself just below the dam. The dam at Lake Toke creates a magazine for Dalsfos and the other four hydro power plants downstream, making these in effect run-of-the-river hydroelectricity plants that are dependent on flow from Dalsfos. The turbines are Kaplan turbines with a combined production capacity of just under 6 MW which equals to an output flow of 36m^3/s. The flood gates are controlled individually from a control room on the dam. The gates have a capacity of 450m^3/s each. Operating hydro power plants come with concession requirements from the Norwegian Water Resources and Energy Directorate (NVE). These dictate the maximum and minimum water levels of Lake Toke, the minimum water flow downstream and the maximum change of water flow.
The Dalsfos plant is owned by Skagerak Kraft, and is one of 46 plants they have ownership of (20 full, 26 they have partial ownership of). Skagerak Kraft is a subsidiary company of Skagerak Energi, in which two thirds are owned by Statkraft and the remaining third is owned by the municipalities of Skien, Porsgrunn and Bamble.

1.1 Background

The flood gates at Dalsfos are used to regulate the water level in Lake Toke during a flood. If the regulation is done successfully, it is possible to avoid property damage and risk of injury for people near Lake Toke and the Kragerø waterways. During a flood, water levels rise rapidly and if the flood gates are suddenly opened, the water will do extensive damage to the village down river of the dam and the roads along the river. Any people close to, or on, the river at the time would also be in severe danger. During the flood in September 2015, the water inflow peaked at 700m^3/s at hourly measurements. To put that into context, this is enough to fill an Olympic-size pool (50m x 25m x 2m) every 3.6 seconds. During this period, the roads downstream had to be closed and the operators had emptied the magazine for two weeks in advance to manage the inflow of water.

This project is a part of automating the flood gates at Dalsfos. The long-term goal for Skagerak Kraft is to ensure that the TUC flood server is functioning in a satisfactory manner under operating conditions so that the output from the system can be relied on. This requires storage of input- and output values, receiving correct outputs and being able to check how previous outputs compare to actual results.

1.2 Previous work

The Telemark University College (TUC) flood server is an advisory system that is currently running at Skagerak Energi office in Porsgrunn. This system calculates the optimal flood gate opening for the Dalsfos hydro power plant based on inflow forecast and Model Prediction Control (MPC). A MPC controller is, as the name implies, a model-based controller that simulates the modeled system for some time to find an optimal input for the next time step. For this specific case, the TUC flood server simulates the system for 10 days to return its suggestion for the flood gate openings for the next hour and then will recalculate this one hour later. How well a MPC controller performs is based on how accurate the model of the system is. The original model for Lake Toke was developed by Bjørn Glemmestad in 2013. It has been described and validated by previous Master theses at the University college of South East Norway (USN). During the flood in September 2015, it was discovered that the model for Lake Toke was insufficient for modeling such extreme conditions.
The MPC controller for TUC flood server was developed by Bernt Lie [2] and implemented in July 2014 with MATLAB. During the original development of the TUC flood server, Dalsfos had only one flood gate. Since then, a new gate has been added.
The TUC Flood Control Converter (TFCC) is the part of the TUC flood server that handles communication between the MPC controller and Skageraks internal systems and when to start the MPC controller. TFCC converts the GS2 input files to CSV input files for the MPC controller to read. Then it converts the CSV output files to GS2 output files. TFCC was developed by Nils-Olav Skeie [3].
The original TUC flood server did not store its data, making it difficult to identify errors when troubleshooting the software. The summer of 2016, master student Alexander Zhang Gjerseth was hired to develop a replacement for TFCC, the Data Handler and the Flood management database [4]. The Data handler reads the GS2 input files and stores the data from them in the Flood management database.

1.3 Overview of report

The main goal of this project is to further improve on the work previously done to automate the regulation of the flood gates at Dalsfos hydro power plant. The tasks to be undertaken in this project are:

- Integrate data storage with the existing system
- Improve the model of Lake Toke and the flood gates
- Create a simulator interface that can predict future values and compare historical data

Chapter 2 describes the physical and the software systems, and how they interact.
Chapter 3 shows the model, what was changed and the methods that were used.
Chapter 4 describes the parts of the TUC flood server and how they interact.
In chapter 5, the results, tools and methods used are discussed.
In chapter 6, the conclusions of the report are presented.

2 System overview

Figure 2.1 shows the catchment for Lake Toke. This is the area where any rain- and snowfall (or lack thereof) will affect the water level in Lake Toke.

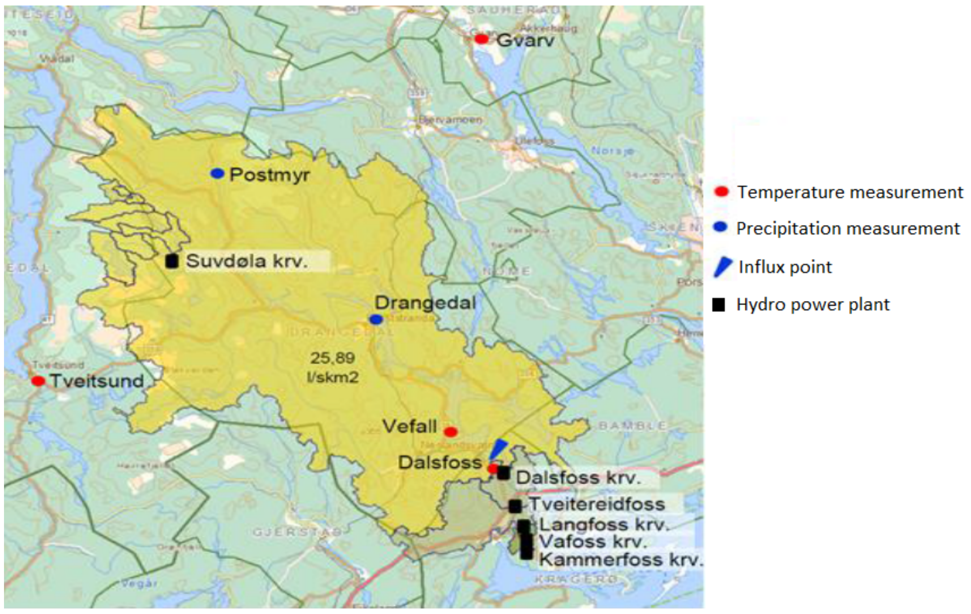


Figure 2.1: Catchment for Lake Toke (From [5], legend modified).

The catchment is 1156 km^2 large, making it difficult and time consuming to get an exact measurement of the catchment. Skagerak subscribes to a weather forecast service provided by `Storm.no`. The hydrologists at Skagerak analyze this data to find a forecast for the inflow to Lake Toke, as marked in Fig. 2.1, just above Dalsfos hydro power plant. The hydrological data is stored on an internal database (TRADE). From this database, the hydrological data is sent along with available measurements from the Dalsfos dam to the TUC flood server. In the TUC flood server, the input data is used to give a suggestion for the gate opening and both the input and output data are stored in an internal database. The output data is then sent to a measurement value comparison system named HIDACS that notifies the dam operator via text message if he or she should make any changes/action. HIDACS is also connected to sensors at the hydro power plant and sends these values to TRADE [5]. Any errors detected by the TUC

server are reported to Skagerak IT personnel through the Status folder.
A representation of the structure of how the TUC flood server interacts with the system at Skagerak is shown in figure 2.2.

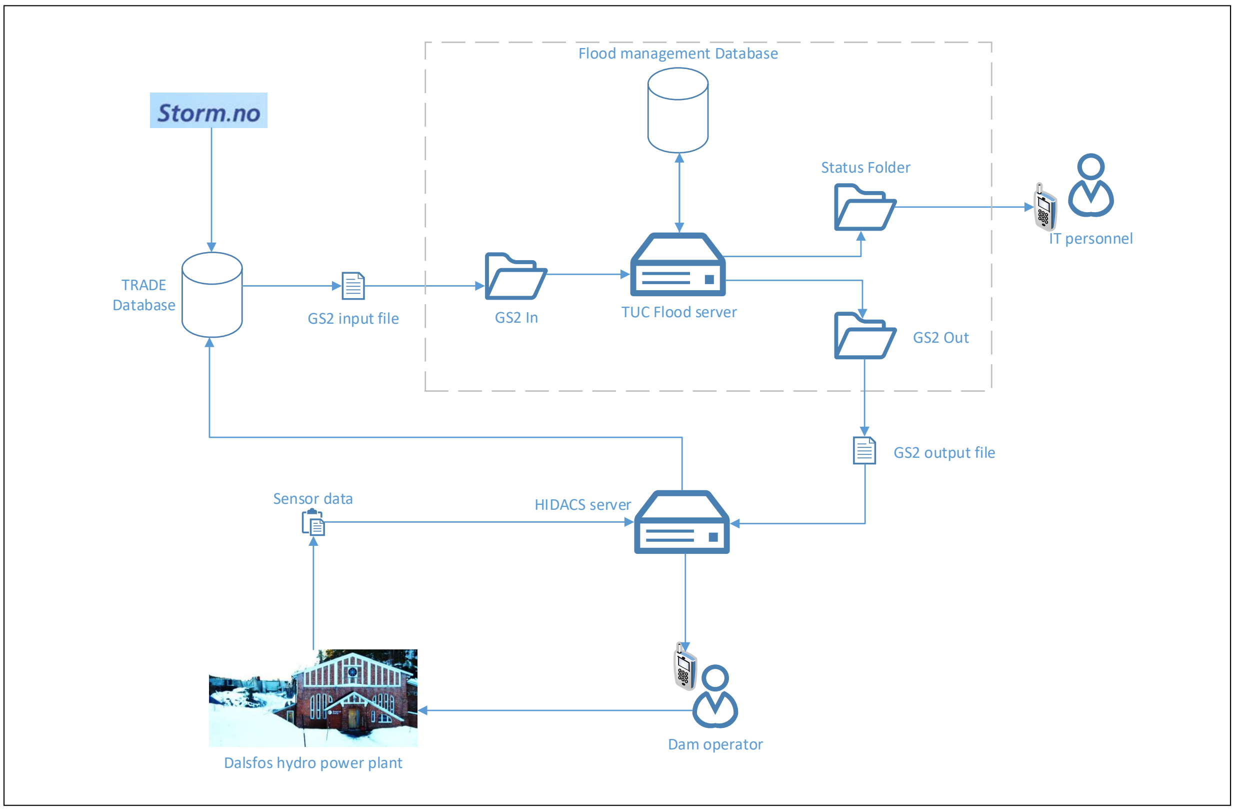


Figure 2.2: Network representation (From [5], simplified).

3 Model

Parts of the model for Lake Toke and Dalsfos hydro power plant have shown to be in need of tuning. This chapter presents the work done on the model. First, a summary of the updated model is presented, then which parts were updated and how are shown. Finally, a comparison of the old and the new model are presented.

3.1 Updated model summary

The model used in the updated TUC Flood server is presented below in equations 3.1 - 3.9. The parameters, and their values, are seen in table 3.1. Lastly, the polynomial coefficients are presented.

- h_1 and h_2 are the states of the model
- h_g is the control input (that can be varied at will)
- $\dot{V}_i$ is the inflow of water given by the hydrology model (disturbance)
- $\dot{W}_e$ is the planned power production (disturbance)
- The outputs are: x_M, x_D, $\dot{V}_t$ and $\dot{V}_g$

Calculating the change in the states:

$$\frac{dh_1}{dt} = \frac{1}{(1-\alpha)\ A(h_1)}\ [(1-\beta)\ \dot{V}_i - \dot{V}_{12}] \tag{3.1}$$

$$\frac{dh_2}{dt} = \frac{1}{\alpha\ A(h_2)}\ (\beta\dot{V}_i + \dot{V}_{12} - \dot{V}_t - \dot{V}_g) \tag{3.2}$$

Filling curve of Lake Toke:

$$A(h) = 28 \times 10^6 \cdot 1.1 \cdot h^{\frac{1}{10}} \tag{3.3}$$

Inter-compartment flow (volumetric flow within Lake Toke, from Merkebekk to Dalsfos):

$$\dot{V}_{12} = k_1 \cdot \sqrt{h_1 - h_2} + k_2 \cdot \sqrt[4]{h_1 - h_2} \tag{3.4}$$

Volumetric water flow through the flood gate:

$$\dot{V}_g = C_d \cdot \omega \ \cdot min(h_g, h_2) \cdot \sqrt{2g\ max(h_2, 0)} \tag{3.5}$$

The volumetric flow through the turbines is found by choosing the 2nd root of

$$0 = c_1 x_q^3 + (c_2 - c_1 x_D) x_q^2 + (c_3 - c_2 x_D + c_4 \dot{V}_g) x_q + \dot{W}_e - c_3 x_D - c_4 \dot{V}_g x_D - c_5 \tag{3.6}$$

and is inserted into:

$$\dot{V}_t = a \frac{\dot{W}_e}{x_D - x_q} + b \tag{3.7}$$

Water level at Dalsfos:

$$x_D = h_2 + x_{LRV}^{min} \tag{3.8}$$

Water level at Merkebekk:

$$x_M = h_1 + x_{LRV}^{min} \tag{3.9}$$

Table 3.1: Parameters for the original Lake Toke model.

Parameter	Value	Unit	Comment
α	0.01	-	Fraction of surface area in compartment 2
β	0.01	-	Fraction of inflow to compartment
C_d	0.7	-	Discharge coefficient, Dalsfos gate
ω_1	11.6	m	Width of Dalsfos gate 1
ω_2	11.0	m	Width of Dalsfos gate 2
x_{LRV}^{min}	55.75	m	Minimal low regulated level value
x_{HRV}^{max}	60.35	m	Maximum high regulated level value
g	9.81	m/s^2	Acceleration of gravity

The polynomial coefficients and their values are:

- $a = 124.69$
- $b = 3.161$
- $c = [0.1315,\ -9.5241,\ 172.34,\ -7.7045e-3,\ -87.359]$
- $k = [100,\ 1]$

3.2 Historical data

The data used for model fitting was provided by Beathe Furenes at Skagerak Energi. The data contains hourly measurements and their derived values, from the period from 01/01-2015 to 10/01-2017, totaling 17768 measurement points. The measurements for the water level in Lake Toke and the water inflow to the Dalsfos dam can be seen in figure 3.1.

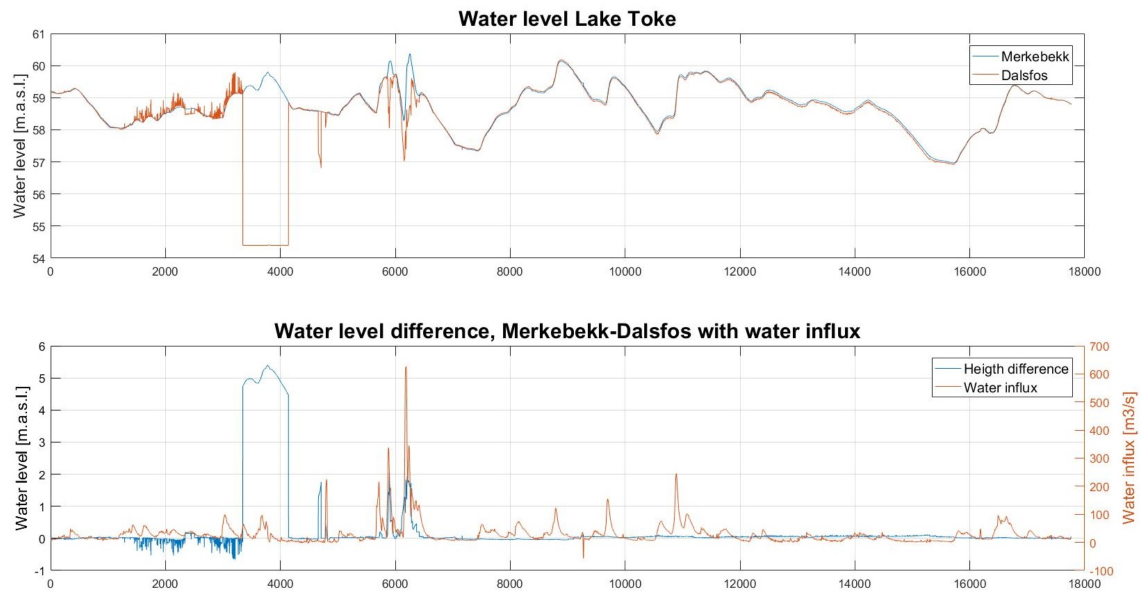


Figure 3.1: Water level in Lake Toke, 2 measuring points and the difference between them (All data).

When first evaluating the data, one can see from figure 3.1 that the first 4700 values seem to have some measurement errors. It does not make sense that the water level would be almost one meter higher at the point furthest downstream (points 1800-3300). Then the sensor at Dalsfos seems to fail for an extended period, stuck at the same value. These assumptions can be further verified by looking at the water inflow in the lower half of the figure. There is nothing to indicate the odd behavior that we see in the first 4700 data points.
Based on these observations, the first 4800 measurement points of data were excluded from further work, ending up with the data shown in 3.2.

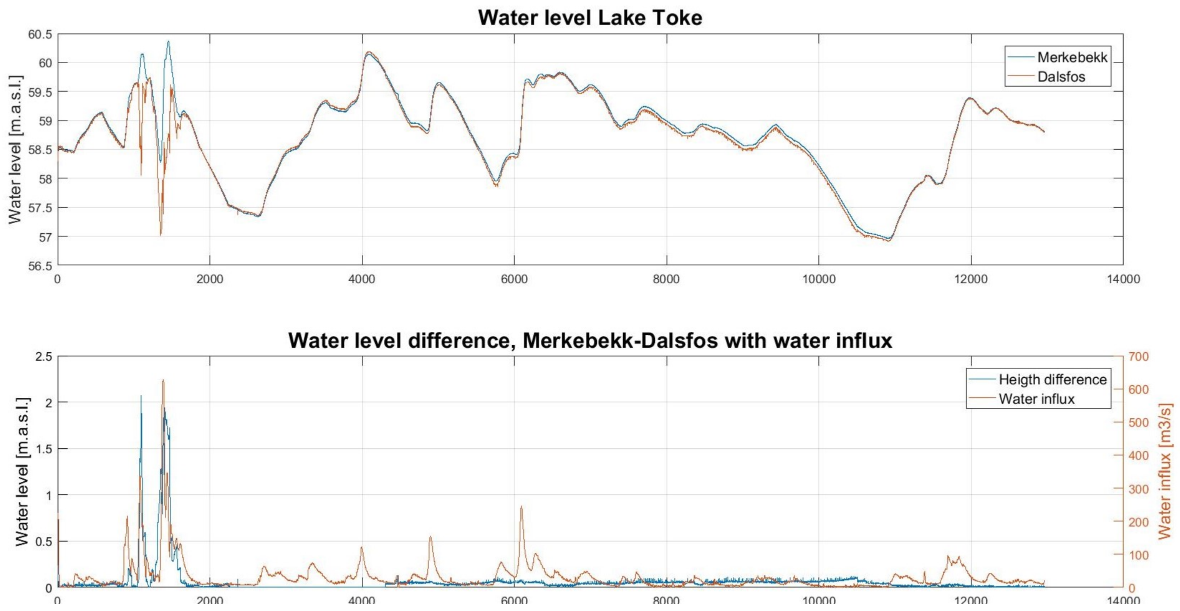


Figure 3.2: Water level in Lake Toke, 2 measuring points and the difference between them (First 4800 data points removed).

3.3 Gate flow parameter updating

The original formula for the gate flow is the same as the updated one, but the discharge coefficient C$_d$ was 1.0 and is now updated to 0.7. This is simply an update based on information from Norwegian Water Resources and Energy Directorate (NVE) [6, p. 10] supplied by Åsmund Hasaas at Skagerak Energi.

3.4 Inter-compartmental flow model reworking

The inter-compartmental flow for this model is the amount of water flowing from measuring point Merkebekk to Dalsfos based on the water height difference between these two points. The original equation (shown in equation 3.10) needed to be reworked based on historical data.

$$\dot{V}_{12} = K_{12}(h_1 - h_2)\sqrt{|h_1 - h_2|} \tag{3.10}$$

The available data was plotted as seen in figure 3.3. The total water flow is the combined flow through the turbines and the flood gates. The height difference in water level values are from measurements made at the same time as the water flow.

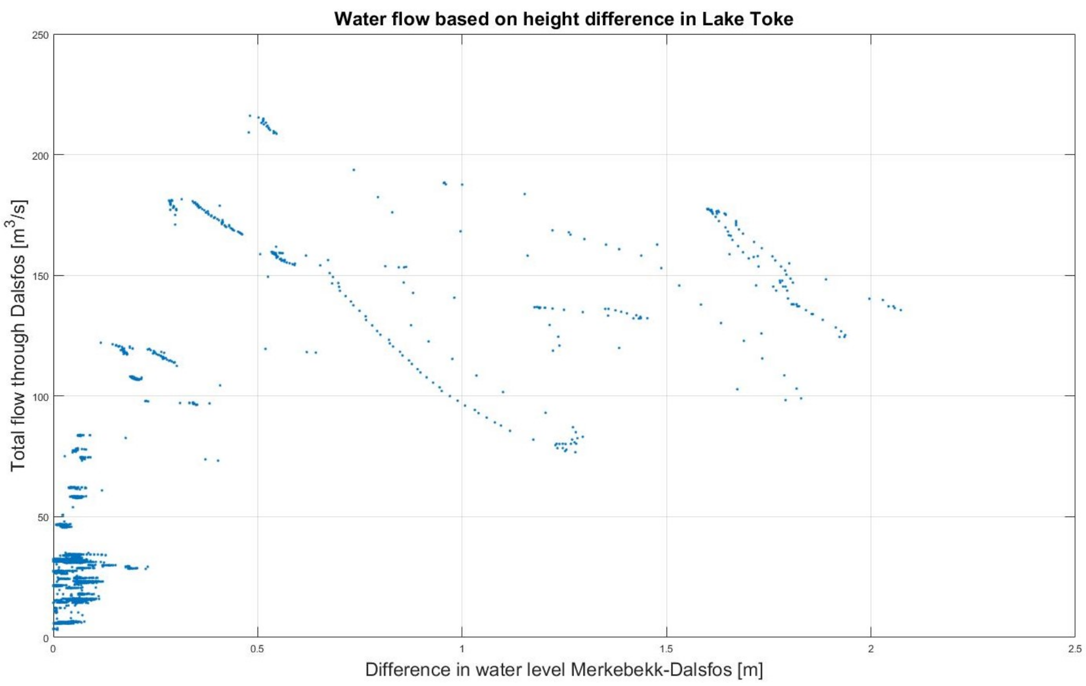


Figure 3.3: Total water flow through Dalsfos dam compared to height difference Merkebekk-Dalsfos.

The updated function was suggested by thesis supervisor Bernt Lie as seen in equation 3.4. To verify that the function was suitable for the available data, the polynomial coefficients had to be determined and it had to be ensured that the function fits the data.
The polynomial coefficients were found by use of the Least Squares method. This method solves parameter estimation by the use of vector algebra. The output (in this case, the total water flow) is defined as seen in equation 3.11 for N measurement points:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = k_1 \cdot \sqrt{x} + k_2 \cdot \sqrt[4]{x} + k_0 = \begin{bmatrix} \sqrt{x_1} & \sqrt[4]{x_1} & 1 \\ \sqrt{x_2} & \sqrt[4]{x_2} & 1 \\ \vdots & \vdots & \vdots \\ \sqrt{x_N} & \sqrt[4]{x_N} & 1 \end{bmatrix} \cdot \begin{bmatrix} k_1 \\ k_2 \\ k_0 \end{bmatrix} = \phi \cdot \theta \tag{3.11}$$

To find the values for k, equation 3.11 can be solved for θ, as seen in equation 3.12:

$$\theta = (\theta^T \cdot \theta)^{-}1 \cdot \theta^T \cdot y \tag{3.12}$$

There is an available function in MATLAB called *polyfit* what will solve this, but there is no option to edit or weigh the factors. So this function cannot be used since we want the function to go through (0,0) in the plane. This is to ensure that when there is no height difference between the different measurement points of Lake Toke, there is no flow of water between those points either. To achieve this, we have to force k_0 to be equal to zero. Thankfully, MATLAB is well suited for solving vector algebra and solving this problem is simple. Below is a simplification of the MATLAB code used for finding the polynomial coefficients theta. The whole script used for finding the parameters and plotting figure 3.4 can be found in appendix C.

```
x = waterLevelMerkebekk - waterLevelDalsfos;
y = waterFlowTurbines + waterFlowGates;
phi = [x.^1/2  x.^1/4];
theta = [phi\y ; 0];
```

This method of determining the polynomial coefficients was repeated three times with different degrees of the function to have the function from first to fourth order. These four functions were plotted in figure 3.4 against the measurement data to determine how well they fit the target data. In this figure we see that the first order function (green) falls outside off the data. The fourth order function (orange) results in a parabola which in is counterintuitive, the flow of water should increase with the height difference. Finally, we see that the second and third order functions are nearly identical and are a good fit for the data. Since the third degree offers no extra information, the second order function was chosen.

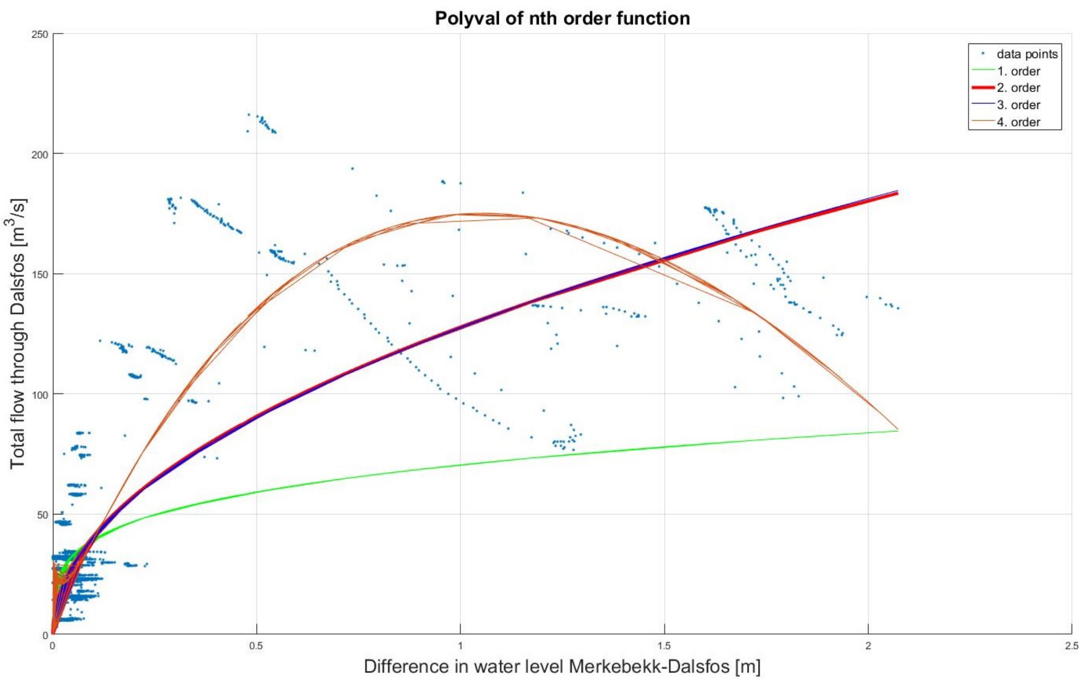


Figure 3.4: Polynomial evaluation, different order of fitted function.

3.5 State estimator parameter tuning

The α and β parameters were originally set to 0.05 and 0.02. This was an experience-based estimate done when the model first was developed.
To estimate new values, the built-in MATLAB function *lsqnonlin* was used.
"lsqnonlin is a nonlinear least-squares solver that solves nonlinear least-squares curve fitting problems of the form (seen in equation 3.13):" [7]

$$\underset{x}{min}||f(x)||_2^2 = \underset{x}{min}(f_1(x)^2 + f_2(x)^2 + ... + f_n(x)^2) \tag{3.13}$$

The MATLAB script paramFit.m (appendix G) was developed to tune the α and β parameters. The function call order for paramFit.m can be seen in figure 3.5. First paramFit.m retrieves the target historical data from the getModelData.m function (appendix D). This data and the dhEstimator.m function (appendix H) are sent as inputs to the *lsqnonlin* function. lsqnonlin will try different inputs for α and β over several iterations before returning the values that minimize the dhEstimator.m function. The dhEstimator.m function uses the built-in MATLAB function ode15s [8] to solve the differential equations returned from tokeSimulator.m (appendix I) to simulate the changes in water level in Lake Toke. tokeSimulator.m takes all the measurement data and the α and β parameters as inputs. It uses these values to first calculate the water flow through the turbines and the flood gates using their respective functions (appendix F and E). Finally, tokeSimulator.m calculates the derivatives for the water level and returns these values.

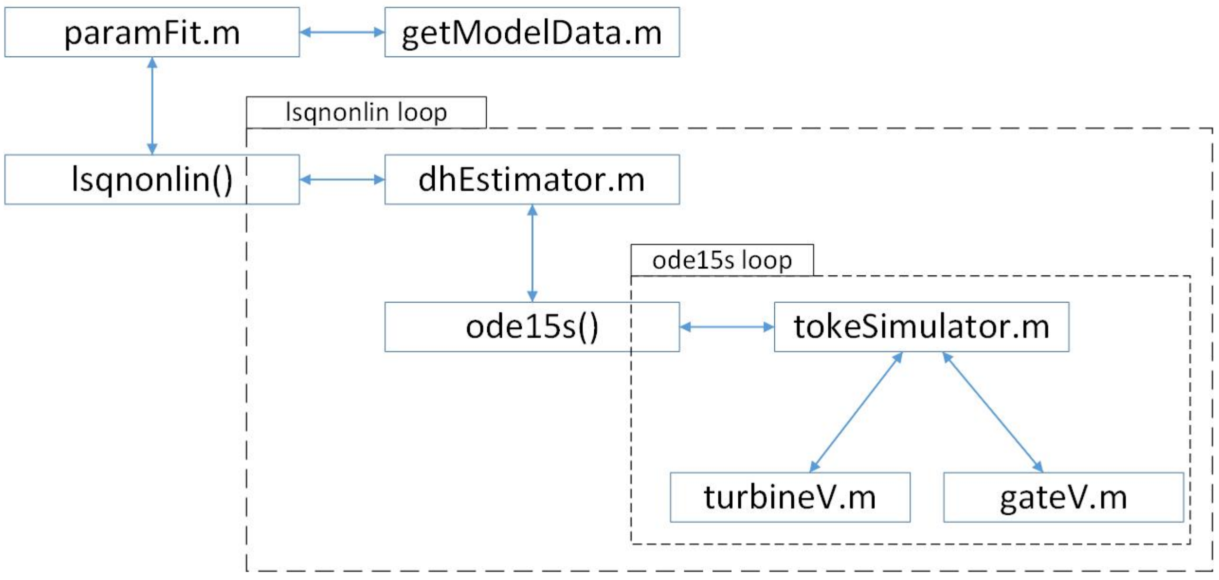


Figure 3.5: Function call order for MATLAB script paramFit.m.

3.6 Model verification

Figure 3.6 shows a comparison of the new and the old model in open loop simulation for 30 days. Each model was supplied with the initial measurement data at the starting time, the influx prognosis and the production plan ten days ahead.

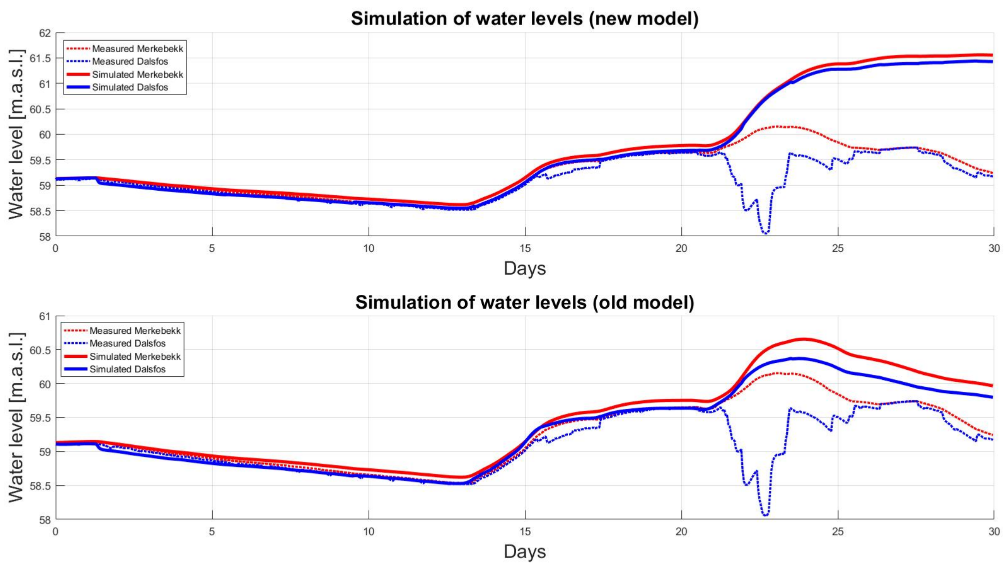


Figure 3.6: Comparison of old and new model in open loop simulation.

Figure 3.7 shows a comparison of the new and the old model in closed loop simulation for 30 days. Both models have been supplied with measurement data at each time step. This was done to see how the models respond to accurate data.

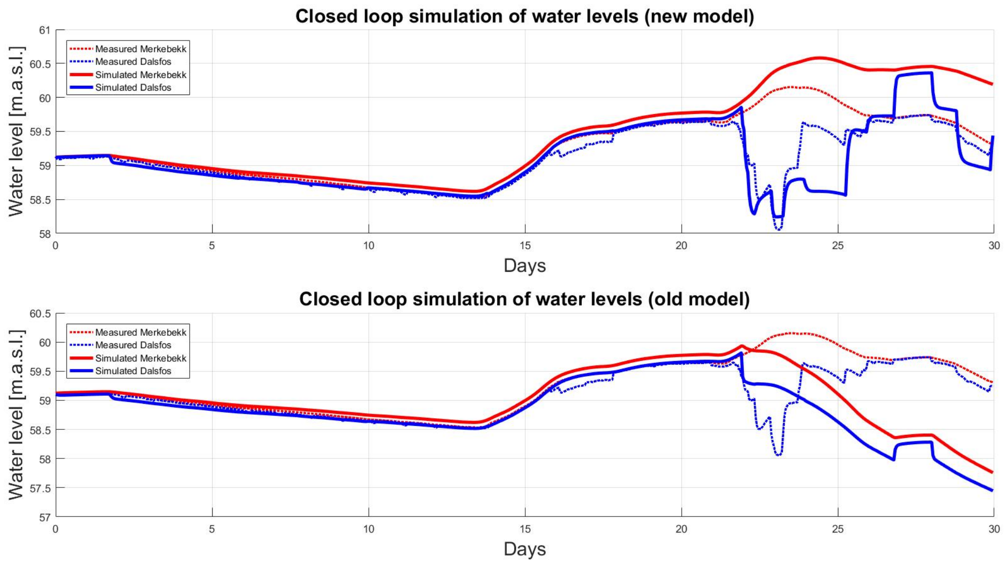


Figure 3.7: Comparison of old and new model in closed loop simulation.

4 TUC Flood server

The TUC flood server consists of three subsystems:

- Data Handler, a C# application
- Flood management, a Microsoft SQL database
- MPC controller, a MATLAB application

The Data Handler runs continuously, monitoring the input folder for new files and starts the MPC controller at chosen intervals. When the Data handler detects a new file, it reads the input values in the file stores all the input values in the Flood management database.
When the The MPC controller is started, it queries the database for the newest input values and imports these. This data is used by the controller for simulating the system and calculating future gate openings. The output values are stored in both an output GS2 file and in the Flood management database. How the subsystems interact is shown in figure 4.1. By using the functions developed for the MPC controller and gathering data from the Flood management database, a simulator was developed

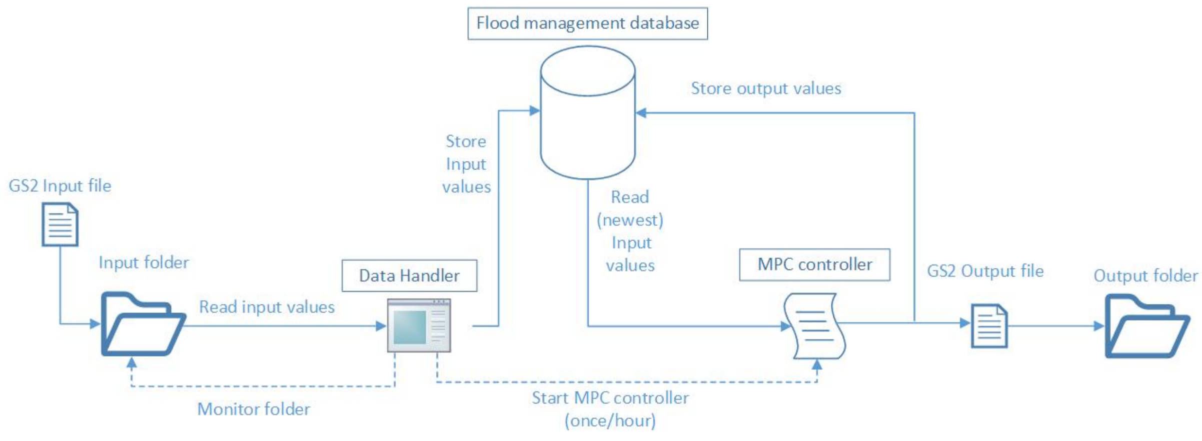


Figure 4.1: TUC flood server interaction.

As mentioned in the chapter 1.2, the Data Handler and the Flood management database were developed by Alexander Zhang Gjerseth. This has been documented in [4].

4.1 Data Handler

Figure 4.2 shows the sequence diagram for Data Handler (from [4]). This figure shows a simplified flow of the function calls. First, the main class loads the configuration and creates the timers. Timer.tick controls sets how often the main program runs, while a secondary timer (not shown in the figure) sets how often the MPC controller is started. The main program checks for new files at each Timer.tick. When it detects a new file, the file is read, stored in a local table and then written to the Flood management database.

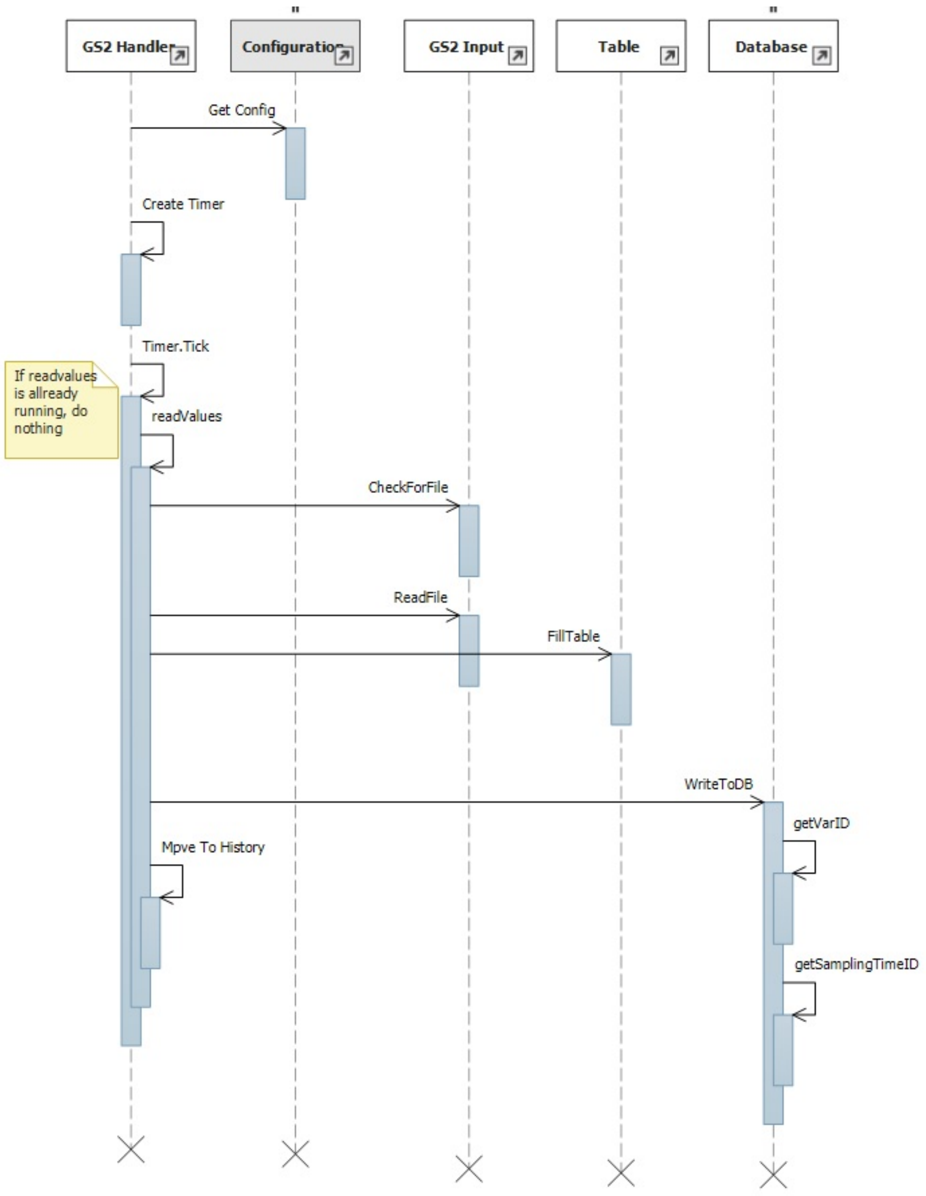


Figure 4.2: Sequence diagram of Data Handler.

The Data handler GUI is seen in figure 4.3. The user has the choice to start the Data handler, stop it from running and show Data. When the Data Handler application is started (from the executable), it will not execute any code until the Start button is pressed. Pressing stop will tell the program to finish its current tasks and halt the program from running again. It should be noted that all timers will reset when the Stop button is pressed. So the MPC controller will be not be started again until a full cycle of the timer has been completed after the Start button has been pressed (default one hour).

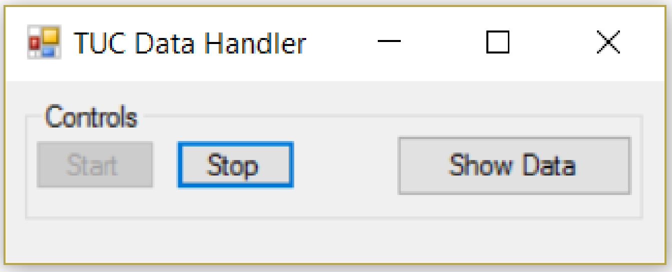
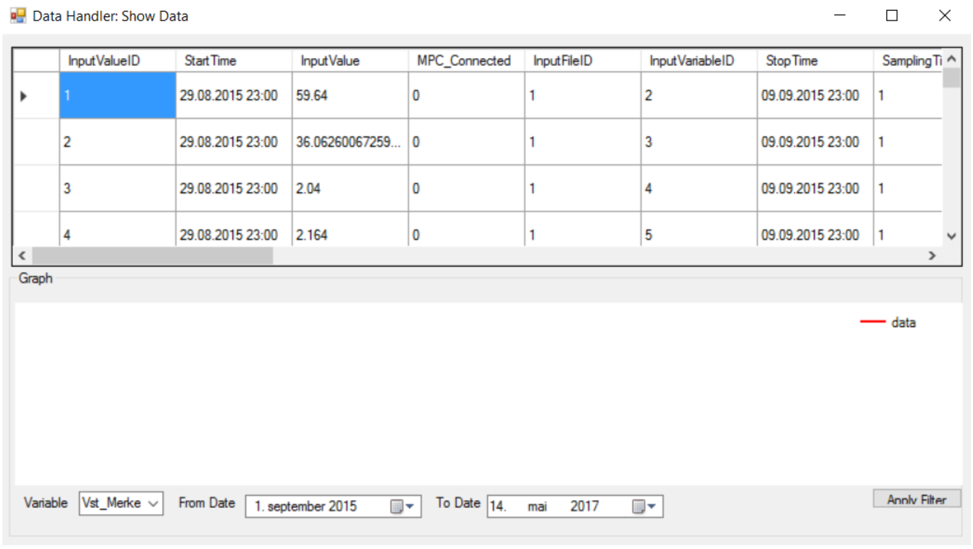


Figure 4.3: TUC Data Handler GUI.

When pressing the Show Data button, a new window will be opened, similar to the one shown in figure 4.4. This window retrieves and shows all the data stored in the INPUTVALUE table in the database.



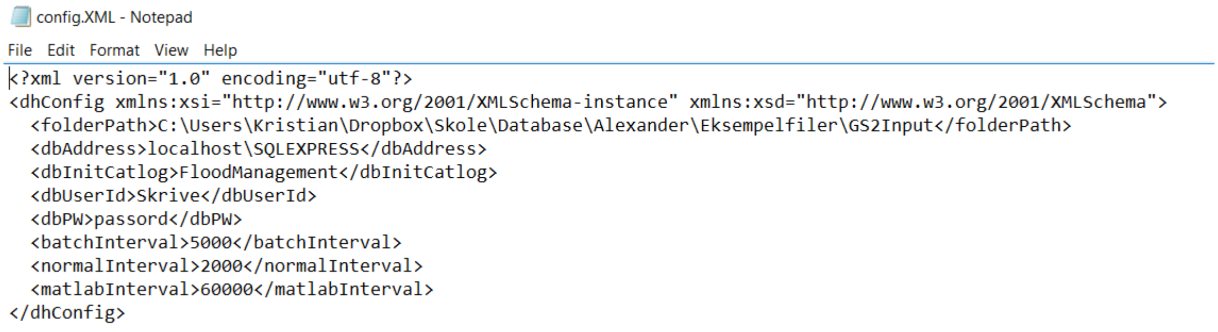
	InputValueID	StartTime	InputValue	MPC_Connected	InputFileID	InputVariableID	StopTime	SamplingTi
▸	1	29.08.2015 23:00	59.64	0	1	2	09.09.2015 23:00	1
	2	29.08.2015 23:00	36.06260067259...	0	1	3	09.09.2015 23:00	1
	3	29.08.2015 23:00	2.04	0	1	4	09.09.2015 23:00	1
	4	29.08.2015 23:00	2.164	0	1	5	09.09.2015 23:00	1

Variable: Vst_Merke From Date: 1. september 2015 To Date: 14. mai 2017

Figure 4.4: TUC Data Handler GUI, Show Data.

The settings for the Data Handler can be edited in the config.XML file, as seen in figure

4.5. When the Data Handler executable is run, it will check the input folder for a config file to load data from. If Data Handler can't find the config file, it will create one from with default values. Any changes by the user have to be done in this file. The parameters and a description of each are found in table 4.1.



```
<?xml version="1.0" encoding="utf-8"?>
<dhConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <folderPath>C:\Users\Kristian\Dropbox\Skole\Database\Alexander\Eksempelfiler\GS2Input</folderPath>
  <dbAddress>localhost\SQLEXPRESS</dbAddress>
  <dbInitCatlog>FloodManagement</dbInitCatlog>
  <dbUserId>Skrive</dbUserId>
  <dbPW>passord</dbPW>
  <batchInterval>5000</batchInterval>
  <normalInterval>2000</normalInterval>
  <matlabInterval>60000</matlabInterval>
</dhConfig>
```

Figure 4.5: TUC Data Handler configuration file.

Table 4.1: Data Handler configuration parameters.

Parameter	Comment
<folderPath>	Input folder, where input GS2 files are read
<dbAddress>	Database address
<dbInitCatlog>	Database initial catalog
<dbUserId>	Database user ID
<dbPW>	Database password
<batchInterval>	Main program timer interval [ms]
<normalInterval>	Alternate mode timer interval, not in use
<matlabInterval>	MPC controller start timer interval [ms]

4.2 Database

Figure 4.6 shows the diagram for the Flood management database. In this figure, only the tables containing dynamic information are shown. The tables that are collapsed contain static information that is unchanged during normal operation. The full diagram is shown in Appendix B.

With the structure of figure 4.6, all the input values are stored in a single table and are related to each other by their respective input- or output file. This gives the option of querying for variables based on both time and their file affiliation, making it simple to get

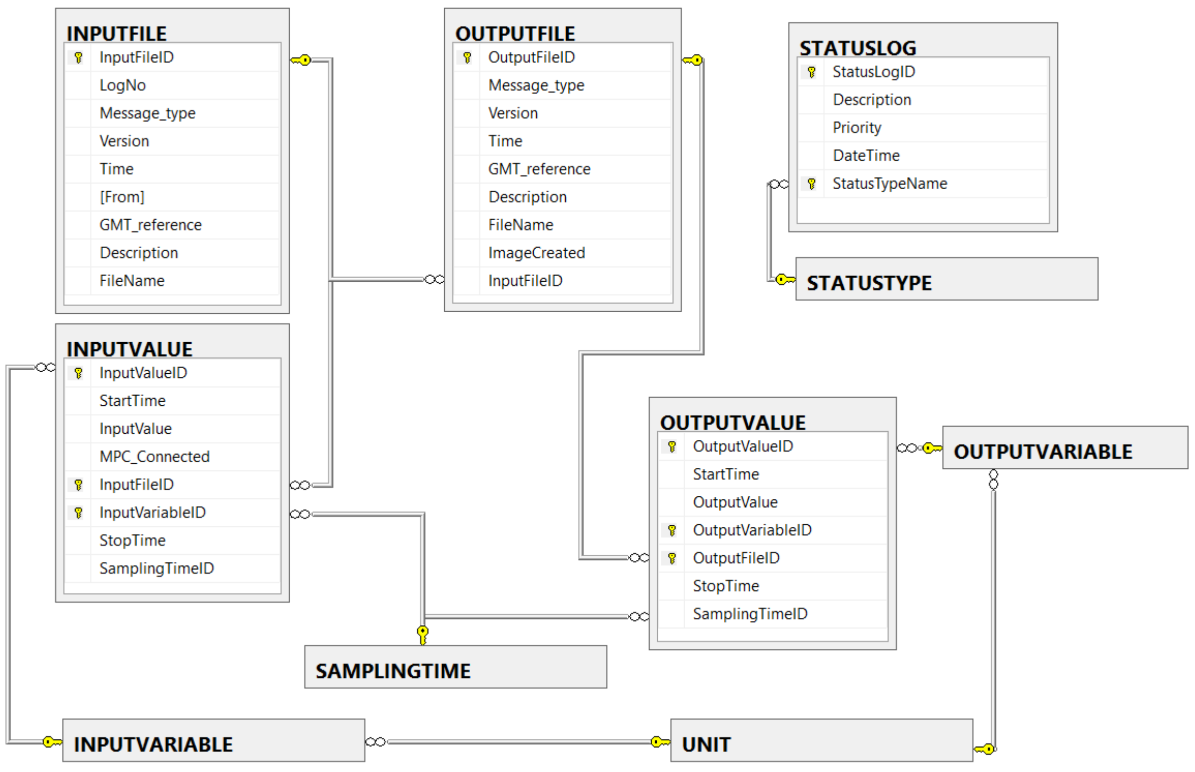


Figure 4.6: Database diagram.

data. Secondly, this structure makes it easy to store and read new variables; only their corresponding values in the -VARIABLE tables need to be added.

The STATUSLOG table is used for storing reports:

- Status reports are for monitoring the applications during regular operations, these have a priority value of 3.
- Warning reports are to indicate potential faults or irregularities during operations, these have a priority value of 2.
- Error reports are for critical faults that interrupts regular operations, these have a priority value of 1.

When the Data handler detects a file, when it is done with storing the data from the file, and when it starts the MPC controller, the Data handler stores a status report. If the Data Handler discovers any errors during operations, a error report will be stored. Report storing is not implemented in the MPC controller.

4.3 MPC controller

The main focus of this thesis has been to change how the MPC controller interacts with the rest of the TUC flood server and to update how its calculations are done. Thus, the main functionality of the MPC controller has not been changed during this process. This section details which of the original files have been altered, which are new and what changes have been made. The original MPC controller and its files are documented in [2]. Changes from the original MPC controller are summarized in table 4.2.

Table 4.2: MPC controller change log.

File name	Line	Comment	Appendix
findDataFuture.m	64	Updated model for Vdg	J
findDataPast.m	37	Updated model for Vdg	K
manageToke.m	52	Changed output to GS2 and database	L
modLakeToke.m	24-26	Updated parameters a, b and K12,	M
–”–	39-42	Updated models for Vd12, Vdg and Vdt	M
prepTokeState.m	26	Changed input to database	N

The updated MPC controller uses the measurement data and the hydrological data from Dalsfos to calculate a flood gate opening suggestion. Every time the MPC controller is started, it gathers data from the newest input file stored in the Flood management database. Once the MPC controller has finished calculating the output values, these are stored in a GS2 output file in the GS2 Out folder. An output file containing the same information as the GS2 output file is stored the Flood management database. Once the MPC controller has completed its operations, it terminates MATLAB.

Getting and storing data has been reworked as the previous version of the MPC controller communicated by use of CSV files. prepTokeState.m is the function that reads the input data. First, it gets the file number of the newest input file using database-GetFileNo.m (appendix O). Then, using the file number as an input to databaseRead.m (appendix P) to read the input data.
Storing the output data is handled by storeOutput.m (appendix Q), called by manage-Toke.m. It is a control function for exporting the output data to a GS2 file and the Flood management database. The function call order for storeOutput.m is shown in figure 4.7

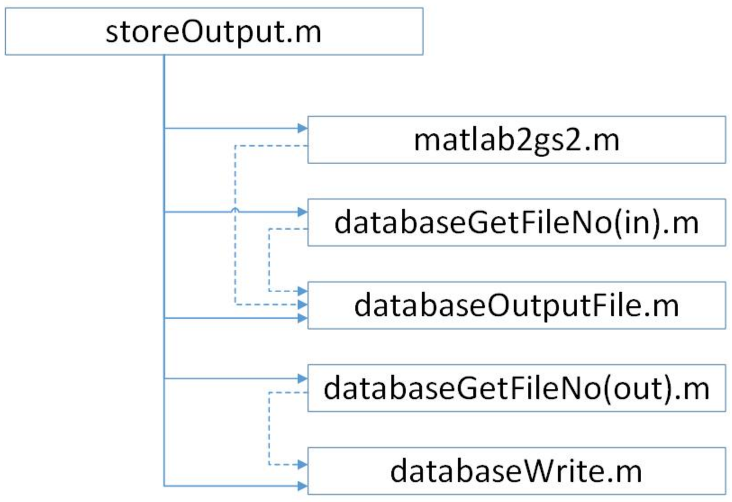


Figure 4.7: Function call order for MATLAB function storeOutput.m.

storeOutput.m is called with the output values as inputs. First, it connects to the Flood management database with the native ODBC Interface [9] and builds the required variables before calling the remaining functions:

1. matlab2gs2.m (appendix R): This function generates the output GS2 files and store them in the GS2 out folder. It returns the filename of the newly generated GS2 file.
2. databaseGetFileNo(in).m returns the file number of the newest input file
3. databaseOutputFile.m (appendix S): This function stores an entry for the output file currently being generated. This entry makes a connection between the GS2 file, the output values stored in the database and the input file.
4. databaseGetFileNo(out).m returns the file number of the newly generated output file entry in the database.
5. databaseWrite (appendix T): This function stores the output values in the database with relation to the newly generated output file.

4.4 Simulator

The script Simulator.m was developed for generating figure 3.6 in section 3.6. Figure 4.8 shows the function call order for Simulator.m. First it gets the measurement data for comparison from stored historical data using getModelData.m. Then Simulator.m gets the initial data for simulation from the database by getting the file number from databaseGetFileNo.m and sending that value to databaseRead.m. Then it sets up the

initial states and constraints by calling in order loadSetupToke.m, prepTokeState2.m, loadStateToke.m and readTokeConstraint.m. prepTokeState2.m is a modified version of the original prepTokeState.m that takes the data vector as an input value instead of getting it itself. With all the initial values and states ready, Simulator.m loops over the desired simulation time by calculating the flood gate opening with ctrlTokeMPC.m and inserting it into modLakeToke.m to get the water levels for the next time step. This loop is done twice, once for the new model and once for the old model. Once all the simulations are completed, the results is plotted.
The complete script used is found in appendix U

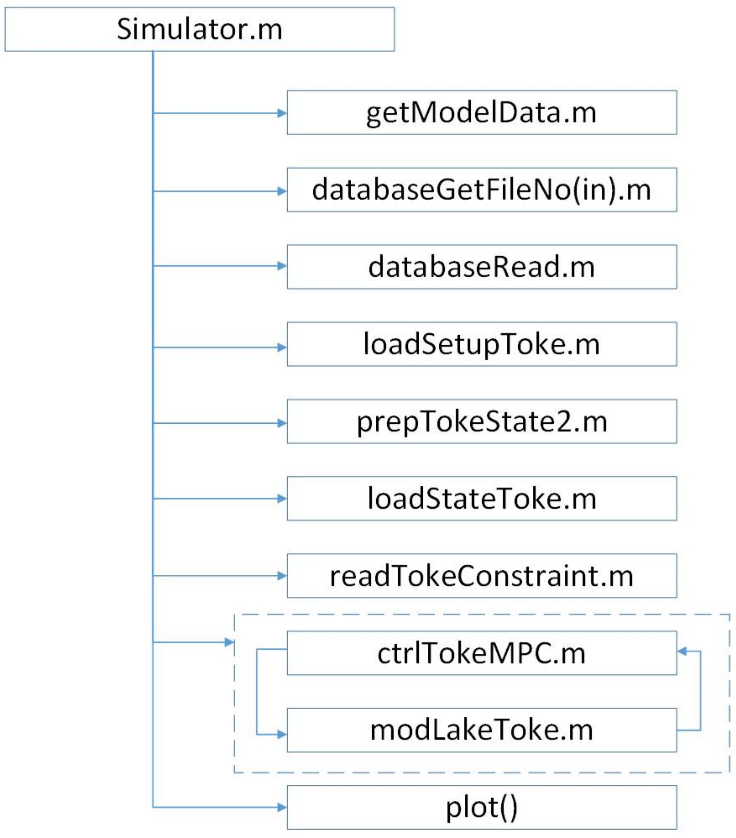


Figure 4.8: Function call order for MATLAB function Simulator.m.

5 Discussion

5.1 Data Handler

Error message are only written to the database and not the error files. This was not implemented when the Data Handler was designed and there has not been time to implement it during the thesis work. This feature that should be implemented for system robustness.
Any errors discovered during normal operations are reported in the database. A useful addition would be to show the status- and error messages on the front panel of the GUI so that the user does not have to access the database to view them.
Status updates for tasks started and completed by the Data Handler are reported to the database. It would be useful to monitor the database for the output from the MPC controller to check that it completes its task in time. The alternative of having the MPC controller itself write status updates to the database is simpler but would still require the Data Handler to monitor said status updates. But perhaps this is the better solution if error handling and reporting was implemented in the MPC controller. This would hopefully give the user an idea of why the MPC controller could not complete its task rather than it just a timeout message.

5.2 Database

The Flood management database was designed prior to the thesis work. During the thesis work, testing showed no issues so no changes have been made to it.
One possible addition to the database could be to implement a trigger that monitors the activity of the Data Handler. This trigger would give an error message if no activity was detected within a given time limit. This would be useful as a failsafe in case the Data Handler for some reason did not function properly. But if the Data Handler did not function, then the MPC controller would most likely not run and so not generate any output GS2 files. The need for a trigger would depend on whether the lack of an output GS2 file gives a suitable error message or not.

5.3 Model tuning

5.3.1 Gate flow parameter updating

In the task description (appendix A), under the second bullet point on the first page one can read: “In addition, a new model for flow through the flood gates should be implemented, based on available look-up data”. The look-up table mentioned is the same as the dam operators use to determine the gate opening to achieve a certain water flow. This table was originally implemented in the MPC controller but during discussions with Åsmund Hasaas at Skagerak, it was discovered that this table was derived from an equation supplied by NVE. It was chosen to rather implement the equation directly as it is simpler and requires noticeably less computation time.

5.3.2 Inter-compartmental flow model reworking

A weakness of the model is in the development of the inter-compartmental flow model presented in section (3.4). The assumption for this model is that the flow of water between the two measuring points in Lake Toke is equal to the flow through the Dalsfos dam. This is inaccurate since the flow through Dalsfos is controlled by the operator and not (directly) influenced by the difference in water height. The lack of a clear connection can be seen in figure 3.3.
To get a better model would require some method for interpreting the currently available measurement data (if possible). Another option would be to get measurement data of the water flow by the upper height measurement point. With the methods and measurement data available, the current assumption for the model is “as good as it gets”.
The original results when determining the value of k were [125.3877; 2.4384; 0]. This is slightly higher than the values presented in the model summary in section 3.1. After these values were found, the parameters for the state estimator were tuned as shown in section 3.5. The original values for k were used when tuning the state estimator. Once the new state estimator parameters were set, the values for k were then re-tuned using the new state estimator parameters, giving the values presented in the model summary.

5.3.3 Inter-compartmental flow model reworking MATLAB script

Vfit.m (appendix C) uses the built-in *polyval* function [10] for evaluation purposes. The function takes an array of inputs values and a function, evaluating the function at every input value. For our use, *polyval* is not entirely suited as the function has to expressed as an array of coefficients that are interpreted as a polynomial in descending power as seen in equation 5.1.

$$y = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1} \tag{5.1}$$

To work around this, the fourth root has been taken of the input data for finding the parameters and is then raised (back) to the power of four when plotting.

For plotting in figure 3.4, the MATLAB function *polyval* was used. To get the function to function properly, a trailing zero needed to be added to the values of k in the script. Further, one can see that some of the lines seem to deviate into several smaller ones, especially for the orange line. This is a side effect of there being several x-values for each y-value, resulting in several lines being returned.

5.4 Model verification

In figure 3.6 we see a comparison of the new and the old model in the open loop simulation. The data used starts at 11. August 2015. The simulation runs for 30 days, into the start of September, where Lake Toke experienced a major flood. Both models follow the measured values well for the first 13 days, and then relatively well until day 21 with slightly better results from the new model. That the simulation starts to deviate after 10 days and are getting progressively worse, is as expected since the simulators are only supplied with an influx prognosis for 10 days.
In figure 3.7 we see the simulator compared in a closed loop simulation with data from the same time period as the previous figure. The difference from the previous figure is that the simulators receive measurement data every time step. Both the simulators give good results for the first 21 days but struggle with the flood. Here we see an improvement of the new model compared to the old one. The new model follows the trends of the measurement data but still is slightly off.

5.5 Input data handling

There is no form of filtering or vetting done on the input data received neither in the Data Handler, nor in the MPC controller. Any errors in the input data is likely to produce errors in the output data or perhaps result in the MPC controller to abort is operation and thus give no output. Adding safeguards against invalid inputs that would produce a helpful error message (instead of aborting without notice) would make the controller more robust.

5.6 MPC controller

As mentioned in section 5.5, there is no vetting done on the input data. It would be an advantage to check for missing or old data to give an error message, and possibly get older data to be able to give an output. Whether to use old data or not for calculating outputs is another matter. Since the time constant of the system is very long, it should not make a huge difference to use data that is a few hours old. But there would have to be a limit as to how old data can be used, since the quality of the output would suffer when going far back. There is also the danger that by continuing to generate output files on old data, that the error that led to the lack of new data would not be discovered.
The information used for connecting to the database in storeOutput.m and prepTokeState.m is static. In storeOutput.m, the information used for generating and formatting the output data is also static. It would be better to gather all this information in a settings file to avoid duplicate data and making it simpler to modify settings for the user. Another alternative for getting the information for connecting to the database would be to make a function for reading the configuration file for the Data Handler. This would ensure that this information is only stored on place.

6 Conclusions

The new model has been thoroughly tested and has been found to be an improvement to the old one. The improvements have not been major, more of a needed tuning based on experience and measurement data. The same would probably be beneficial to do with the new model at a later stage, when new experience and data has been gathered. No model is perfect, the best one can hope for is ”good enough” for its intended use.
The updated TUC flood server has shown to solve its task well during testing. Now remains to deploy it live to validate that it works as intended. Integration of the database to the MPC controller was completed early during the thesis work. In hindsight, it would have been wise to implement this change on the live version once it was done. This would have made more measurement data available during the thesis work. Also, it would have made it possible to first test the database integration before model changes, making troubleshooting simpler. The simulator developed functions adequately but is very slow. Due to time constraints during development (of both the Data Handler and the MPC controller), these applications operate with little user feedback. The Data Handler has good error handling and -reporting, but none of these are easily available to the user or IT personnel at Skagerak. The MPC controller has no error handling or -reporting and no vetting of input variables that may cause it to fail.

6.1 Future work

- Test and integrate the updated TUC flood server live at Skagerak.
- Data Handler:
  - Create and send error files
  - Show status and error messages in the GUI
- MPC controller:
  - Implement error handling to avoid it from aborting without warning
  - Send status and errors reports to the database
- Optimize simulator to reduce computation time

References

[1] oxforddictionaries.com. Definition of hydrology. Article on the Internet. 5 May 2017. URL: https://en.oxforddictionaries.com/definition/hydrology.

[2] B. Lie. Final report: KONTRAKT NR INAN-140122 Optimal Control of Dalsfos Flood Gates - control algorithm. Tech. rep. Telemark University College, Porsgrunn, Norway, July 2014.

[3] N-O. Skeie. KONTRAKT NR INAN-140122 Optimal styring av Dalsfos Flomporter - Grensesnitt mellom GS2 Filformat og Flomprognosesystem med MPC fra Høgskolen i Telemark (HiT). Tech. rep. Telemark University College, Porsgrunn, Norway, June 2014.

[4] A.Z. Gjerseth. Systembeskrivelse. Tech. rep. Telemark University College, Porsgrunn, Norway, Aug. 2016.

[5] B. Furenes. MPC PÅ FLOMLUKEHÅNDTERING I KRAGERØVASSDRAGET. Presentation. Porsgrunn, Norway, 2016.

[6] Norges vassdrags-og energidirektorat. Retningslinjer for flomløp. Publication on the Internet. 17 Jan 2017. URL: http://publikasjoner.nve.no/retningslinjer/2005/retningslinjer2005_02.pdf.

[7] Mathworks. lsqnonlin. Article on the Internet. 20 Mar 2017. URL: https://se.mathworks.com/help/optim/ug/lsqnonlin.html.

[8] Mathworks. ode15s. Article on the Internet. 20 Mar 2017. URL: https://se.mathworks.com/help/matlab/ref/ode15s.html.

[9] Mathworks. Connecting to Database Using Native ODBC Interface. Article on the Internet. 10 Jan 2017. URL: https://se.mathworks.com/help/database/ug/connecting-to-database-using-native-odbc-interface.html.

[10] Mathworks. polyval. Article on the Internet. 10 Feb 2017. URL: https://se.mathworks.com/help/matlab/ref/polyval.html.

Appendix A

Signed task description

University College of Southeast Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: Improved flood management of Lake Toke

HSN supervisor: Bernt Lie, prof., University College of Southeast Norway
Nils-Olav Skeie, assoc. prof. (co-supervisor)

External partner: Skagerak Energi (dr. Beathe Furenes)

Task background:
A prototype of a flood management system was developed for Skagerak Energi in 2014, and based on the experience, it is of interest to update the system.

The flood management system considers management of Lake Toke in Telemark, and the Dalsfoss hydro power plant. Concession requirements for the hydro power plant puts constraints on (i) the change of flow through flood gates, (ii) minimum flow rate out of the lake, (iii) the level of the lake (seasonal variations in upper and lower levels), etc. The prototype flood management system receives information about inflow into the lake as well as production rates, water level in several locations, etc. Then, by combining measurements/information with a dynamic model of the level, the prototype attempts to compute the optimal flow rate through flood gates in such a way as to maximize the power production over time, while fulfilling the concession constraints.

Based on experience with the prototype, it is of interest to update the prototype in several directions:

- Information should be stored in a general database instead of in temporary MATLAB arrays, and it should be possible to access the data in the database in a simple manner, both on-line, and also to use historic data. In a summer-job in 2016, the framework for such a database has been developed, but it is necessary to make a final adjustment and testing of this database.
- A large flood in September 2015 revealed that the model of the flow from the lake (Merkebekk) down to the dam (Dalsfoss) needs to be improved. This part of the model should be tuned to historic data in the database, together with other parts of the model. In addition, a new model for flow through the flood gates should be implemented, based on available table look-up data. Overall, the model should be tested and validated.
- The current Model Predictive Control strategy should be evaluated, and, if possible, be improved/made more efficient. Note that the controller is not in automatic closed loop: currently, the MPC algorithm gives advice to an operator, who then evaluates the advice before changing the flood gate openings.
- Sometimes, the operator chooses to not follow the advice of the MPC algorithm. It is of interest to develop an off-line simulation mode for the system where it is possible to compare the actual operation of the flood gates by the operator, with a hypothetical operation as suggested by the MPC algorithm.

References:

Lie, Bernt (2014). Final report: KONTRAKT NR INAN-140122. Optimal Control of Dalsfos Flood Gates – control algorithm. Report, Telemark University College, Porsgrunn.

Skeie, Nils-Olav (2014). Endelig rapport: KONTRAKT NR INAN-140122. Optimal Control of Dalsfos Flood Gates: TFCC – TUCFloodControlConverter. Grensesnitt mot flomprognosesystem med MPC fra Høgskolen i Telemark (HiT). Funksjons- og konfigurasjonsbeskrivelse. Report, Telemark University College, Porsgrunn.

Task description:
Based on the task background, the following tasks are relevant:

1. Based on previous work, implement and test a database for handling information/measurements and results for the flood management system.
2. Using historic data in the database, improve/tune the model of Lake Toke using parameter estimation techniques.
3. Improve the efficiency of the MPC algorithm if possible, and consider how the results can be stored in the database.
4. Develop a strategy for checking how well a historic flood gate operation has been, compared to the hypothetical case of blindly obeying the advice from the MPC.
5. Report the work in the Master's Thesis, and possibly in a suitable conference/journal paper.

Student category: IIA students

Practical arrangements:

The workplace is Campus Kjølnes of University College of Southeast Norway. There will be weekly meetings with the supervisor from January until mid April, either face-to-face or via Skype, with hand-in of partial reports every 3 weeks.

The thesis work will start January 2, 2017, and the deadline for thesis hand-in is May 15 2017 at 14:00. An oral presentation with examination will take place no later than June 24, 2017.

Signatures (date and signature):

Kristian D. Kvam

Student: Kristian Dønheim Kvam, 150534

[illegible]

Supervisor: Bernt Lie

Appendix B

Full database diagram

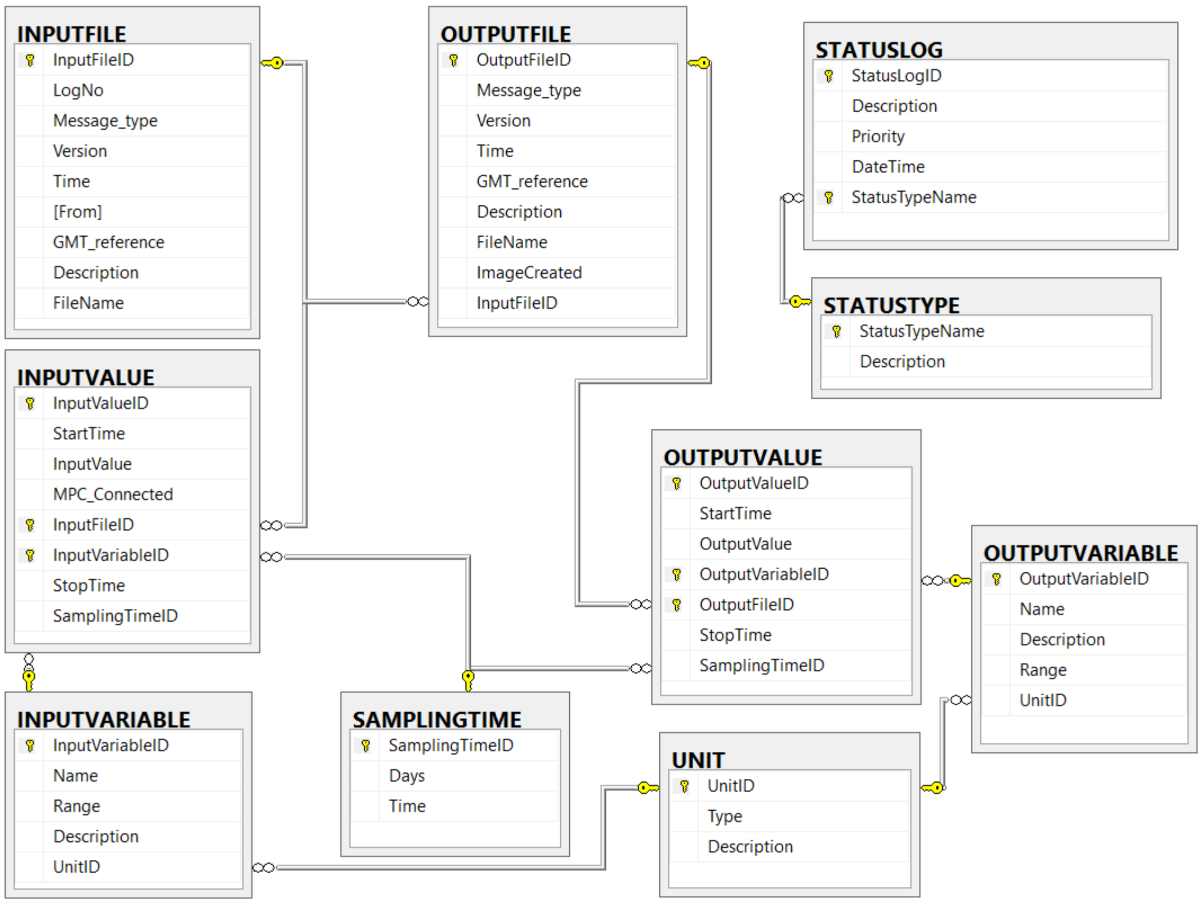


Figure B.1: Database diagram

Appendix C

Vfit.m

```
% Get measurement data
[xM, xD, hg1, hg2, Wde] = getModelData();
dh = xM - xD;
% Calculate water flow through flood gates and turbines
Vdg = zeros(length(Wde), 1);
Vdt = zeros(length(Wde), 1);
for i = 1:length(Wde)
    Vdg(i) = gateV(hg1(i), hg2(i), xD(i));
    Vdt(i) = turbineV(Wde(i),xD(i),Vdg(i));
end
VdTot = Vdg + Vdt;

% Removing data points with negative height difference
% Note: The x-values have the fourth roots taken to get
% polyval to give correct ouput.
idx = find(dh>=0);
x = (dh(idx)).^(1/4);
y = VdTot(idx);

% Making equation of different order
phi1 = [x.^1];
phi2 = [x.^2  x.^1];
phi3 = [x.^3 x.^2 x.^1];
phi4 = [x.^4 x.^3 x.^2 x.^1];
% Find parameter values for equations
p1 = [phi1\y; 0];
p2 = [phi2\y; 0];
p3 = [phi3\y; 0];
p4 = [phi4\y; 0];
```

```
% Plotting
% Note: The x-values are raised to the power of four to get the
% original values
figure(1)
title('Polyval␣of␣nth␣order␣function','fontsize',18)
ylabel('Total␣flow␣through␣Dalsfos␣[m^3/s]','fontsize',18)
xlabel('Difference␣in␣water␣level␣Merkebekk-Dalsfos␣[m]',...
    'fontsize',18)
hold on
plot(x.^4, y,'.','linewidth',6)
plot(x.^4, polyval(p1,x),'g-')
plot(x.^4, polyval(p2,x),'r-','linewidth',3)
plot(x.^4, polyval(p3,x),'b-')
plot(x.^4, polyval(p4,x))
l1 = legend('data␣points','1.␣order','2.␣order',...
    '3.␣order','4.␣order');
set(l1,'FontSize',12);
axis([0, 2.2, 0 , 250])
grid on
hold off
```

Appendix D

getModelData.m

```
function [xM,xD,hg1,hg2,Wde,Vdi,t,Vdg1,Vdg2,Vdt] = getModelData()

% Data source
T = table2array(readtable('Data_ryddet2.xlsx','ReadRowNames',true));
T(1:4862,:) = []; % Removing "bad" data
data = 1:12906; % Selecting remaining data
t = 0:1:length(data)-1;

% Getting measurement data
xM = (T(data,1)); % [m a.s.l.] Merkebekk
xD = (T(data,2)); % [m a.s.l.] Dalsfos
hg1 = (T(data,5)); % Gate 1 opening [cm]
hg2 = (T(data,7)); % Gate 2 opening [cm]
Wde = (T(data,9))+(T(data,10))+(T(data,11)); % Power prod [MW]
Vdi = (T(data,17)); % Tilsig Dalsfos [m3/s]

% Getting values drived from measurements (Skagerak formulaes)
Vdg1 = (T(data,6)); % Gate 1 volumetric flow [m3/s]
Vdg2 = (T(data,8)); % Gate 2 volumetric flow [m3/s]
Vdt = (T(data,12)); % Turbine volumetric flow [m3/s]
```

Appendix E

gateV.m

```
function Vdg = gateV(hg1,hg2, xD)
% Function for calculating water flow through flood gates

% Constants
Cd = 0.7;
wGate1 = 11.6;
wGate2 = 11.0;
xLRVmin = 55.75;
g = 9.81;

% Converting to cm
hg1 = hg1/100;
hg2 = hg2/100;

% Gate flow calculation
hD = xD - xLRVmin;
Vdg1 = Cd*wGate1*min(hg1,hD)*sqrt(2*g*max(hD',0));
Vdg2 = Cd*wGate2*min(hg2,hD)*sqrt(2*g*max(hD',0));

Vdg = Vdg1 + Vdg2;
```

Appendix F

turbineV.m

```
function Vdt = turbineV(Wde,xD,Vdg)
% Function for calculating water flow though the turbines

% polynomial coefficients for xQ estimator and Vdt estimator
c = [1.3152e-1; -9.5241; 1.7234e2; -7.7045e-3; -8.7359e-1];%
polVdt = [1.2469e2 (3.161)];

% Turbine flow calculation
xQroots = roots([c(1), (c(2)-c(1)*xD), (c(3)-c(2)*xD +...
    c(4)*Vdg),(Wde-c(3)*xD-c(4)*Vdg*xD-c(5))]);
xQ = xQroots(2);
dxt = xD-xQ;
Vdt = min(polVdt(1)*Wde/dxt + polVdt(2),36);
```

Appendix G

paramFit.m

```
[xM, xD, hg1, hg2, Wde, Vdi, t] = getModelData();

days = 30; % Time to simulate
span = 1:1:days*24; % Convert days to hours
shift = 0; % Increase to use other data (start time)
d = 1+shift:length(span)+shift;

% Allocate desired data
xM = xM(d);
xD = xD(d);
hg1 = hg1(d);
hg2 = hg2(d);
Wde = Wde(d);
Vdi = Vdi(d);

% Set up function, start values and limits
toke = @(p) dhEstimator(p, xM, xD, hg1, hg2, Wde, Vdi);
param0 = [0.01 0.01];
lb = [0 0];
ub = [1 1];

% Function for parameter estimation
param = lsqnonlin(toke,param0,lb,ub);
```

Appendix H

dhEstimator.m

```
function e = dhEstimator(p, xM, xD, hg1, hg2, Wde, Vdi)

% Allocate values
Xmeas = [xM,xD];
x0 = Xmeas(1,:)';
a = p(1);
b = p(2);
N = length(xM);
Xmod = zeros(N,2);
Xmod(1,:) = x0;

% Run simulator at one hour intervals to match measurement data
for i=1:N-1
    toke = @(t, x) tokeSimulator(t,x,hg1(i),hg2(i),Wde(i),...
    Vdi(i),a,b);
    [T,X] = ode15s(toke, [0 3600], x0);
    x0 = X(end,:)';
    Xmod(i+1,:) = x0';
end

% Find deviation between measured and simulated values
F = Xmeas-Xmod;
% Reshape output to a single vector to work with lsqnonlin
e = reshape(F,2*N,1);
```

Appendix I

tokeSimulator.m

```
function dxdt = tokeSimulator(t, x, hg1, hg2, Wde, Vdi, a, b)
xM = x(1);
xD = x(2);

% Calculate water flow through the gates and the turbines
Vdg = gateV(hg1,hg2,xD);
Vdt = turbineV(Wde,xD,Vdg);

% Constants
xLRVmin = 55.75;
K12 = [100 1];

% Convert to relative heigth and find height difference
hM = xM - xLRVmin;
hD = xD - xLRVmin;
dh = hM-hD;
% Removing negative height difference to avoid error
if (dh < 0)
    dh = 0;
end

% Calculate Intercompartmental flow and fill rates
Vd12 = K12(1)*nthroot(dh,2)+ K12(2)*nthroot(dh,4);
AhM = max(28e6*1.1*(hM)^0.1, 1e3);
AhD = max(28e6*1.1*(hD)^0.1, 1e3);

% Calculate derivatives
der_hM = ((1-b)*Vdi-Vd12)/((1-a)*AhM);
der_hD = (b*Vdi + Vd12 - Vdt - Vdg)/(a*AhD);

dxdt = [der_hM; der_hD];
```

Appendix J

findDataFuture.m

```
function dataFuture = findDataFuture(xM,xD,Hg,nUdim,tnUdim,...
    VdiWin,WdeWin,xLRVwin,xHRVwin,tClock,tHorizonFuture,tSamp)
%========================================
% Lake Toke Flood Management System
%
% Function for setting up future data for reportTokeData
% author:    Bernt Lie, Kristian Kvam
% location: Telemark University College, Porsgrunn
% date:      March 21, 2014    v 0.1
%            June 20, 2014     v 1.0
%            May 5, 2017       v 1.1 Updated model for Vdg
%
%========================================
dirFileToke;
reportFile = [ioDir,'/',reportDataFile];
load(reportFile);
mParToke;
hgMin_cm = 0;
hgMax_cm = mPar.hgMax*100;
xLRVmin = mPar.xLRVmin;
xHRVmax = mPar.xHRVmax;
cvec = mPar.cvec;
polVdt = mPar.polVdt;
xRef = mPar.xRef;

%
tWin = (0:tSamp:tHorizonFuture)';
hgWin = Hg(1)*ones(size(tWin));
for i=2:nUdim
    idx = find(tWin >= tnUdim(i));
    hgWin(idx) = Hg(i);
end
xMwin = xM*ones(size(hgWin));
xDwin = xD*ones(size(hgWin));
xQwin = zeros(size(xMwin));
xQ1win = xQwin;
xQ3win = xQwin;
xQLwin = xQwin;
xQ1Lwin = xQwin;
xQ3Lwin = xQwin;
VdtWin = zeros(size(xMwin));
VdgWin = zeros(size(xMwin));
%
% Loop through horizon
NH = length(tWin);
x = [xM;xD];
tSpan = [datenum(tClock),datenum(tClock)+tSamp]; % Sim time span
tSpan_s = tSpan*3600*24;  % Timespan in seconds
tSamp_s = tSamp*3600*24;
for I = 1:NH
    % Defining anonymous model function
    w = [VdiWin(I);WdeWin(I)];
    u = [hgWin(I);hgWin(I)];

    mToke = @(t,x) modLakeToke(t,x,u,w);
    % Simulating system
    [tm, X] = ode15s(mToke, tSpan_s, x);
    x = X(end,:)';
    xM = x(1);
    xD = x(2);
    xMwin(I) = xM;
    xDwin(I) = xD;
    hD = xD - xLRVmin;
    Vdg = gateV(u(1)*100,u(2)*100,xD);
    VdgWin(I) = Vdg;
    %-- turbine flow rate
    pol = zeros(1,4);
    pol(1) = cvec(1);
    pol(2) = cvec(2)-cvec(1)*xD;
    pol(3) = cvec(3)-cvec(2)*xD+cvec(4)*Vdg;
    pol(4) = WdeWin(I)-cvec(3)*xD-cvec(4)*Vdg*xD-cvec(5);
    xQroots = roots(pol)';
    xQwin(I) = xQroots(2);
    xQ1win(I) = xQroots(1);
    xQ3win(I) = xQroots(3);
    xQLwin(I) = isreal(xQroots(2));
    xQ1Lwin(I) = isreal(xQroots(1));
    xQ3Lwin(I) = isreal(xQroots(3));
    %
    dxt = xD-xQwin(I);
    VdtWin(I) = min(polVdt(1)*WdeWin(I)/dxt + polVdt(2),36);
    tSpan_s = [tm(end),tm(end)+tSamp_s];
    tWin(I) = tm(end)/24/3600;
end
%
VdoWin = VdtWin + VdgWin;
xHRVmaxWin = xHRVmax*ones(size(xHRVwin));
xLRVminWin = xLRVmin*ones(size(xLRVwin));
refLoWin = (1-xRef)*xLRVwin + xRef*xHRVwin;
refHiWin = xHRVwin - mPar.dHRV;

idx = find((VdiWin(1) >= mPar.VdiThr) & (VdiWin >= mPar.VdiThr));

refHiWin(idx) = mPar.xHRVmax - mPar.dHRV;
hgMaxWin_cm = hgMax_cm*ones(size(xLRVwin));
hgMinWin_cm = hgMin_cm*ones(size(xLRVwin));
hgWin_cm = hgWin*100;
nanWin = nan(size(tWin));
%
dataFuture = [xMwin,xDwin,xQwin,xQ1win,xQ3win,xQLwin,xQ1Lwin,...
    xQ3Lwin,nanWin,xLRVwin,xHRVwin,xLRVminWin,xHRVmaxWin,...
    refLoWin,refHiWin,nanWin,nanWin,hgWin_cm,hgMaxWin_cm,...
    hgMinWin_cm,VdiWin,VdtWin,VdgWin,VdoWin,nanWin,WdeWin,...
    nanWin,tWin];
%--> Return
```

Appendix K

findDataPast.m

```
function dataPast = findDataPast(xM,xD,xQ_NVE,Vdo_NVE,...
    xLRV,xHRV,hg1,hg2,hg,Vdi,Wde,tClock,tOpt)
%==========================================================
% Lake Toke Flood Management System
%
% Function for setting up past data for reportTokeData
% author:   Bernt Lie, Kristian Kvam
% location: Telemark University College, Porsgrunn
% date:     March 21, 2014   v 0.1
%           April 1, 2014    v 0.2
%           June 20, 2014    v 1.0
%           May 5, 2017      v 1.1 Updated model for Vdg
%
%==========================================================
dirFileToke;
reportFile = [ioDir,'/',reportDataFile];
load(reportFile);
mParToke;
hgMin_cm = 0;
hgMax_cm = mPar.hgMax*100;
hg_cm = hg*100;
hg1_cm = hg1*100;
hg2_cm = hg2*100;
xLRVmin = mPar.xLRVmin;
xHRVmax = mPar.xHRVmax;
xRef = mPar.xRef;
cvec = mPar.cvec;
polVdt = mPar.polVdt;
%
hD = xD - xLRVmin;
```

```
refLo = (1-xRef)*xLRV + xRef*xHRV;
if Vdi >= mPar.VdiThr
    refHi = mPar.xHRVmax - mPar.dHRV;
else
    refHi = xHRV - mPar.dHRV;
end
Vdg = gateV(hg1,hg2, xD);
% -- current turbine flow rate
pol = zeros(1,4);
pol(1) = cvec(1);
pol(2) = cvec(2)-cvec(1)*xD;
pol(3) = cvec(3)-cvec(2)*xD+cvec(4)*Vdg;
pol(4) = Wde-cvec(3)*xD-cvec(4)*Vdg*xD-cvec(5);
xQroots = roots(pol)';
xQ = xQroots(2);
xQ1 = xQroots(1);
xQ3 = xQroots(3);
xQL = isreal(xQ);
xQ1L = isreal(xQ1);
xQ3L = isreal(xQ3);
%
dxt = xD-xQ;
Vdt = min(polVdt(1)*Wde/dxt + polVdt(2),36);
Vdo = Vdg + Vdt;
%
tNow = datenum(tClock);
%
dataPast(1:end-1,:) = dataPast(2:end,:);
dataPast(end,:) = [xM,xD,xQ,xQ1,xQ3,xQL,xQ1L,xQ3L,xQ_NVE,...
    xLRV,xHRV,xLRVmin,xHRVmax,refLo,refHi,hg1_cm,hg2_cm,hg_cm,...
    hgMax_cm,hgMin_cm,Vdi,Vdt,Vdg,Vdo,Vdo_NVE,Wde,tOpt,tNow];
%--> Return
```

Appendix L

manageToke.m

```
%=========================================
% Lake Toke Flood Management System
%
% Managing Lake Toke Flood Gate based on file 'setupToke.mat'
% author:    Bernt Lie, Kristian Kvam
% location:  Telemark University College, Porsgrunn
% date:      March 11, 2014    v 0.1
%            March 14, 2014    v 0.2
%            April 24, 2014    v 0.3
%            June 19, 2014     v 0.4
%            June 20, 2014     v 1.0
%            May 5, 2017       v 1.1 Changed output from CSV to
%            GS2 and database
%
%=========================================
% Management operations
%-- Read current data from state file
tic;
St = loadStateToke();
%------------------------------------------------------------
%-- Set up constraint windows xLRVwin and xHRVwin
xRV = readTokeConstraint(St.tClock,sD.tHorizonFuture,...
    sD.tSamp,xConstraint);
%-----------------------------------------------------------
%-- Compute gate opening using MPC
Hg = ctrlTokeMPC(St.xM,St.xD,xRV.xLRVwin,xRV.xHRVwin,...
     St.VdiWin,St.WdeWin,St.hg1,St.hg2,sD.nUdim,...
     sD.tnUdim,sD.tHorizonFuture,sD.tSamp,mP);
hg = Hg(1);
%------------------------------------------------------------
```

```
%-- Report computed operation
xLRV = xRV.xLRVwin(1);
xHRV = xRV.xHRVwin(1);
Vdi = St.VdiWin(1);
Wde = St.WdeWin(1);
%
tOpt = toc;
dataPast = findDataPast(St.xM,St.xD,St.xQ_NVE,St.Vdo_NVE,...
    xLRV,xHRV,St.hg1,St.hg2,hg,Vdi,Wde,St.tClock,tOpt);
%--> function? to clean up code
dataFuture = findDataFuture(St.xM,St.xD,Hg,sD.nUdim,sD.tnUdim,...
    St.VdiWin,St.WdeWin,xRV.xLRVwin,xRV.xHRVwin,...
    St.tClock,sD.tHorizonFuture,sD.tSamp);
Vdg = dataFuture(1,23);
%-- Report
dirFileToke;
reportFile = [ioDir,'/',reportDataFile];
save(reportFile,'dataPast','dataFuture');
Plot results and save plots
okReportTokePlot = reportTokePlot(dataPast,dataFuture,sD.tSamp,...
   sD.nPLang);

% Report hg and Vdg
storeOutput(hg,Vdg);
```

Appendix M

modLakeToke.m

```
function der_x = modLakeToke(t,x,u,w)
%================================================
%
% Function for computing vector field of dynamic model
% for Lake Toke
% author:    Bernt Lie, Kristian Kvam
% location: Telemark University College, Porsgrunn
% date:      February 26, 2014   v 0.1
%            June 20, 2014       v 1.0
%            May 5, 2017         v 1.1 Updated model for Vd12, Vdg
%            and Vdt, parameters a, b and K12
%
%================================================
% Naming states
xM = x(1);           % m a.s.l.
xD = x(2);           % m a.s.l.
% Naming control input
hg1 = u(1);          % m
hg2 = u(2);          % m
% Naming disturbances
Vdi = w(1);          % m3/s, current inflow to Toke
Wde = w(2);          % MW, current elictricity production
% Naming parameters
a = 0.01;               % -, volume fraction in volume 1
b = 0.01;               % -, inflow fraction to volume 2
K12 = [100 1];
xLRVmin = 55.75;  % m a.s.l., minimal level of regulated value
%
% Computing algebraic expressions
% -- levels, areas, flows
```

```
hM = xM - xLRVmin;
hD = xD - xLRVmin;
dh = hM-hD;
if (dh < 0)
    dh = 0;
end
A1 = max(28e6*1.1*abs(hM)^0.1,1e3);
A2 = max(28e6*1.1*abs(hD)^0.1,1e3);
Vd12 = K12(1)*nthroot(dh,2)+ K12(2)*nthroot(dh,4);

Vdg = gateV(hg1,hg2,xD);
Vdt = turbineV(Wde,xD,Vdg);

% Setting up dynamic equations
der_xM = ((1-b)*Vdi-Vd12)/(1-a)/A1;
der_xD = (b*Vdi+Vd12-Vdt-Vdg)/a/A2;
%
der_x = [der_xM, der_xD]';
```

Appendix N

prepTokeState.m

```
function okPREP = prepTokeState(tHorizonFuture,tSamp)
%==========================================================
% Lake Toke Flood Management System
%
% Function for converting CST measurements to initTokeState file
% author:    Bernt Lie, Kristian Kvam
% location:  Telemark University College, Porsgrunn
% date:      March 14, 2014    v 0.1
%            June 20, 2014     v 1.0
%            August 28, 2014   v 1.1, bug fix wrt. dataVdiWin(idx) =
%            dataVdiWin(idx(1)-1);
%            May 5, 2017       v 1.2, changed input from CSV to
%            database
%
%===========================================================
%
rP = readRParToke();
dirFileToke;
stateFile = [ioDir,'/',stateDataFile];
if exist(stateFile,'file') == 2
    delete(stateFile);
end
%
% Connect to database and get data from newest file
conn = database.ODBCConnection('TokeData','Skrive','passord');
fileNo = databaseGetFileNo(conn, 'in', 'new');
data = databaseRead(fileNo, conn);
close(conn);
%
% INSTANCE values, measured
```

```
St.xM = data(1); % m a.s.l., current Merkebekk level
St.xD = data(2); % m a.s.l., current Dalsfos dam level
St.Vdo_NVE = data(3); % m3/s, downstream flow from NVE
a = rP.polVdo(1);
b = rP.polVdo(2);
c = rP.polVdo(3)-St.Vdo_NVE;
% m a.s.l., current Dalsfos quay level
St.xQ_NVE = (-b+sqrt(b^2-4*a*c))/2/a;
St.hg1Past_cm = data(8); % cm, past gate opening, Gate 1 (old)
St.hg2Past_cm = data(9); % cm, past gate opening, Gate 2 (new)
St.tClock = [data(50:54),0]; % current time; truncate seconds
%-- PROGNOSIS windows
dataVdiWin = [data(40:49)';data(49)];% m3/s, future inflow Toke
idx = find(isnan(dataVdiWin));
if length(idx) > 0
    dataVdiWin(idx) = dataVdiWin(idx(1)-1);
end
%
dataTWin = (0:1:length(dataVdiWin)-1)';
tWin = (0:tSamp:tHorizonFuture)';
% m3/s, future inflow to Toke
St.VdiWin = interp1(dataTWin,dataVdiWin,tWin);
St.VdiWin(1) = data(7); % m3/s, insert "observed" inflow Toke
%
% MW, next day total power production, Dalsfos
dataWdeWin = data(10)+data(20)+data(30);
% MW, future power consumption, Dalsfos
St.WdeWin = dataWdeWin*ones(size(tWin));
St.WdeWin(1) = sum(data(4:6)); % MW, total power production
% Save results to file
save(stateFile,'St');
okPREP = 1;
%
```

Appendix O

databaseGetFileNo.m

```
function fileNo = databaseGetFileNo(conn, sel, date)

q = char(39); % Get quotation mark to add to sql querry

if strcmp(sel, 'in')
    if strcmp(date, 'new')
        getFileNo = ['SELECT␣TOP␣1␣InputFileID' '␣FROM␣...
␣␣␣␣␣␣␣␣␣␣"FloodManagement"."dbo".INPUTFILE␣ORDER␣BY␣Time␣DESC'];
    else
        getFileNo = ['SELECT␣TOP␣1␣InputFileID␣FROM␣...
␣␣␣␣␣␣␣␣␣␣"FloodManagement"."dbo".INPUTFILE␣WHERE␣␣...
␣␣␣␣␣␣␣␣␣␣Time␣>=',q,date,q,'␣ORDER␣BY␣Time␣ASC'];
    end

elseif strcmp(sel, 'out')
    if strcmp(date, 'new')
        getFileNo = ['SELECT␣TOP␣1␣OutputFileID' '␣FROM␣...
␣␣␣␣␣␣␣␣␣␣"FloodManagement"."dbo".OUTPUTFILE␣ORDER␣BY␣Time␣DESC'];
    else
        getFileNo = ['SELECT␣TOP␣1␣OutputFileID␣FROM␣...
␣␣␣␣␣␣␣␣␣␣"FloodManagement"."dbo".OUTPUTFILE␣WHERE␣...
␣␣␣␣␣␣␣␣␣␣␣Time␣>=',q,date,q, '␣ORDER␣BY␣Time␣ASC'];
    end
end

curs = fetch(exec(conn, getFileNo));
% Converting from table to string
fileNo = mat2str(table2array(curs.Data));
close(curs);
```

Appendix P

databaseRead.m

```
function data = databaseRead(fileNo, conn)

data = ones(1,55)*-1;
%Set preferences with setdbprefs.
setdbprefs('DataReturnFormat', 'table');
setdbprefs('NullNumberRead', 'NaN');
setdbprefs('NullStringRead', 'null');

% Get timestamp from file
timeString = strcat(' FROM "FloodManagement"."dbo".INPUTFILE ...
 WHERE InputFileID=', fileNo);
getTime =['SELECT Time ' timeString];
cursTime = fetch(exec(conn, getTime));
cellTime = table2array(cursTime.Data);
fileTime = str2double(strsplit(char(cellTime),{'-',':',' ','.'}));

% Get data from file
newString = strcat(' FROM "FloodManagement"."dbo".INPUTVALUE ...
WHERE InputFileID=', fileNo);
getData =['SELECT InputValue ' newString ...
    'ORDER BY InputVariableID ASC'];
curs = fetch(exec(conn, getData));
cellData = table2array(curs.Data);

% Filling return array
data(1:6) = str2double(cellData(1:6));
data(7:9) = str2double(cellData(12:14));
data(10:19) = str2double(cellData(7));
data(20:29) = str2double(cellData(8));
data(30:39) = str2double(cellData(9));
```

```
% Ordering array Water influx EC00
allEC00 = str2double(strsplit(char(cellData(10)),',,'))';
data(40:49) = allEC00(1:10);
data(50:55) = fileTime(1:6);

close(curs);
```

Appendix Q

storeOutput.m

```
function storeOutput(hg,Vdg)

hg1 = hg(:,1)*100;
hg2 = hg(:,1)*100;

conn = database.ODBCConnection('TokeData','Skrive','passord');
refTime = datetime('now','Format', 'yyyy-MM-dd HH:mm:ss.ms');
startGS2 = datestr(refTime, 'yyyy-mm-dd.HH:MM:SS');
stopGS2 = datestr(refTime+hours(length(Vdg)),'yyyy-mm-dd.HH:MM:SS');
startDB = datestr(refTime, 'yyyy-mm-dd HH:MM:SS');
stopDB = datestr(refTime+hours(length(Vdg)),'yyyy-mm-dd HH:MM:SS');
step = '0000-00-00.01:00:00';
imgCreated = 1;
refStep = 1;

message = 'GateOpening-data';
ver = '1.2';
GMT = 1;
headerDescrip = 'MPC verdier fra HiT modul';
DBdescrip = 'MatLab MPC output file';

% Storing output in GS2 file
fileName = matlab2gs2(hg,Vdg,startGS2,stopGS2,step,message,ver,...
    GMT,headerDescrip);

% Get filenumber of newest input file in database
inputFileID = str2double(databaseGetFileNo(conn, 'in', 'new'));
% Creating and storing output file in database
databaseOutputFile(fileName,inputFileID,conn,message,ver,GMT,...
  startDB, DBdescrip,imgCreated);
```

```
% Get filenumber of newest output file in database
outputFileID = str2double(databaseGetFileNo(conn, 'out', 'new'));
% Storing output data in database
databaseWrite(conn,startDB,hg1,1,outputFileID,stopDB,refStep);
databaseWrite(conn,startDB,hg2,2,outputFileID,stopDB,refStep);
databaseWrite(conn,startDB,Vdg,3,outputFileID,stopDB,refStep);

close(conn);
```

Appendix R

matlab2gs2.m

```
function fileName = matlab2gs2(hg,Vdg,refStart, refStop,...
    refStep,message,ver,GMT,descrip)

% Rounding ouput values to get relevant decimals
Vdg = round(Vdg);
hg1 = round(hg(:,1)*100);
hg2 = round(hg1);

% Generate file in directory
outputFolder = 'output\';

tNow2 = datestr(now, 'yyyymmddHHMMSS');
tNow1 = [tNow2(1:12) '_'];
fileName = ['444_4_2_' tNow1 tNow1 tNow2 '000' '.exp'];
fullFileName = strcat(outputFolder, fileName);
fileID = fopen(fullFileName, 'w');

% Generating output values
message = strcat('#Message-type=', message);
ver = strcat('#Version=', ver);
GMT = strcat('#GMT-reference=', num2str(GMT));
descrip = strcat('#Description=', descrip);
timeNow = datestr(now, 'yyyy-mm-dd.HH:MM:SS');
inFileTimeNow = strcat('#Time=',timeNow);
start = strcat('#Start=', refStart);
stop = strcat('#Stop=', refStop);
step = strcat('#Step=', refStep);
refUnit1 = '#Unit=cm';
refUnit2 = '#Unit=m3/s';
hgString1 = [];
```

```
hgString2 = [];
VdgString = [];
for f = 1: length(hg1)
    hgString1 = strcat(hgString1, num2str(hg1(f)),'//');
end
value1 = strcat('#Value=<', hgString1 ,'>');
no1 = strcat('#No-of-values=', num2str(length(hg1)));
sum1 = strcat('#Sum=', num2str(sum(hg1)));

for l = 1: length(hg2)
    hgString2 = strcat(hgString2, num2str(hg2(l)),'//');
end
value2 = strcat('#Value=<', hgString2 ,'>');
no2 = strcat('#No-of-values=', num2str(length(hg2)));
sum2 = strcat('#Sum=', num2str(sum(hg2)));

for k = 1: length(Vdg)
    VdgString = strcat(VdgString, num2str(Vdg(k)),'//');
end
valueM = strcat('#Value=<', VdgString ,'>');
noM =strcat('#No-of-values=', num2str(length(Vdg)));
sumM = strcat('#Sum=', num2str(sum(Vdg)));

% Gathering output strings
header = ['##Start-message', message, ver, inFileTimeNow,...
    GMT, descrip];
gate1 = ['##Time-series', '#Reference=30019210-14', start,...
    stop, step, refUnit1, value1, no1, sum1, ...
    '#Description=1009␣Setp_luke_1␣fra␣MPC␣modul'];
gate2 = ['##Time-series', '#Reference=30019210-15', start,...
    stop, step, refUnit1, value2, no2, sum2,...
    '#Description=1009␣Setp_luke_2␣fra␣MPC␣modul'];

mpc = ['##Time-series', '#Reference=30019210-16', start,...
    stop, step, refUnit2, valueM, noM, sumM,...
    '#Description=1009␣Testparameter␣fra␣MPC␣modul'];

output=[header '@' gate1 '@' gate2 '@' mpc '@' '##End-message'];

% Writing to file
for i = 1:length(output)
    if (output(i) == '#' && output(i+1) == '#') || ...
```

```
         (output(i) == '#' && output(i-1) == '#')
    elseif output(i) == '#'
            fprintf(fileID, '\n');
    elseif output(i) == '@'
            fprintf(fileID, '\n\n');
            continue;
    end
    fprintf(fileID, char(output(i)));
end

% Closing file
fclose(fileID);
```

Appendix S

databaseOutputFile.m

```
function databaseOutputFile(fileName, inputFileID, conn,...
    message, ver,GMT, timeNow, des, imgCreated)

% Storing output file in database
tablename = 'OUTPUTFILE';
colnames = {'Message_type', 'Version', 'Time', 'GMT_reference', ...
 'Description', 'FileName', 'ImageCreated', 'InputFileID'};
data = {message, ver, timeNow, GMT, des, fileName, ...
imgCreated, inputFileID};
data_table = cell2table(data,'VariableNames', colnames);
datainsert(conn,tablename,colnames,data_table);
```

Appendix T

databaseWrite.m

```
function databaseWrite(conn, startTime, val, outVarID,...
    outFileID,stopTime, sampID)

tablename = 'OUTPUTVALUE';
colnames = {'StartTime', 'OutputValue', 'OutputVariableID', ...
'OutputFileID','StopTime', 'SamplingTimeID'};
outVarVal = num2str(round(val));
data = {startTime outVarVal outVarID outFileID stopTime sampID};
data_table = cell2table(data,'VariableNames', colnames);
datainsert(conn,tablename,colnames,data_table);
```

Appendix U

Simulator.m

```
% Simulation settings
tStart = '2015-08-22 00:00:00';
daysToSim = 30;
simH = daysToSim*24;

% Get measurement data
[xM, xD] = getModelData();
xM = xM(d);
xD = xD(d);

% Connect to database to get simulation data
conn = database.ODBCConnection('TokeData','Skrive','passord');
timeStr = datestr(tStart, 'yyyy-mm-dd HH:MM:SS');
fileNo = str2double(databaseGetFileNo(conn, 'in', timeStr));
data = databaseRead(num2str(fileNo), conn);
close(conn);
% Assigning start value for new model
xM0 = data(1);
xD0 = data(2);
% Assigning start value for old model
xM1 = data(1);
xD1 = data(2);
Wde = (data(10)+data(20)+data(30))*ones(simH,1);
hg1Past = data(8);
hg2Past = data(9);
c = 1;
d = data(40:49);

% Converting influx prognosis from daily to hourly values
for k = 1:daysToSim
```

```
    for g = 1:24
        Vdi(c) = d(k);
        c = c + 1;
    end
end

% Load settings
[okLoad,sD] = loadSetupToke();
okPREP = prepTokeState2(sD.tHorizonFuture,sD.tSamp, data);
St = loadStateToke();
hg1New = St.hg1;
hg2New = St.hg2;
hg1Old = St.hg1;
hg2Old = St.hg2;
xConstraint = loadOpConstraintToke();
xRV = readTokeConstraint([year(timeStr), month(timeStr),...
    day(timeStr)],sD.tHorizonFuture,sD.tSamp,xConstraint);

% Simulation with the new model
N = length(xM);

for k=1:N-1
    Hg = ctrlTokeMPC(xM0,xD0,xRV.xLRVwin,xRV.xHRVwin,St.VdiWin,...
    St.WdeWin,hg1New,hg2New,sD.nUdim,sD.tnUdim,...
    sD.tHorizonFuture,sD.tSamp,mPar);
    hgSim = Hg(1);
    tokeNew = @(t, x) modLakeToke(t,x,u,w);
    [T,X] = ode15s(tokeNew, [0 3600], x0New);
    x0New = X(end,:)';
    xM0 = x0New(1);
    xD0 = x0New(2);
    XmodNew(k+1,:) = x0New';
    hg1New = hgSim;
    hg2New = hgSim;
end

% Simulation with the old model
for k=1:N-1
    Hg = OldCtrlTokeMPC(xM1,xD1,xRV.xLRVwin,xRV.xHRVwin,...
    St.VdiWin,St.WdeWin,hg1Old,hg2Old,sD.nUdim,sD.tnUdim,...
    sD.tHorizonFuture,sD.tSamp,OldmPar);
    hgSim = Hg(1);
    u = [hgSim, hgSim];
    w = [Vdi(k), Wde(k)];
    tokeOld = @(t, x) OldModLakeToke(t,x,u,w);
    [T,X] = ode15s(tokeOld, [0 3600], x0Old);
    x0Old = X(end,:)';
    xM1 = x0Old(1);
    xD1 = x0Old(2);
    XmodOld(k+1,:) = x0Old';
    hg1Old = hgSim;
    hg2Old = hgSim;
end

t = (0:N-1)/24;
% Plotting results
subplot(211)
hold on
plot(t,xM,'r:','linewidth',2);
plot(t,xD,'b:','linewidth',2);
plot(t,XmodNew(:,1),'r-','linewidth',3);
plot(t,XmodNew(:,2),'b-','linewidth',3);
title('Simulation␣of␣water␣levels␣(new␣model)','fontsize',18)
legend('Measured␣Merkebekk','Measured␣Dalsfos',...
'Simulated␣Merkebekk','Simulated␣Dalsfos','fontsize',18)
ylabel('Water␣level␣[m.a.s.l.]','fontsize',18)
xlabel('Days','fontsize',18)
grid on

subplot(212)
hold on
plot(t,xM,'r:','linewidth',2);
plot(t,xD,'b:','linewidth',2);
plot(t,XmodOld(:,1),'r-','linewidth',3);
plot(t,XmodOld(:,2),'b-','linewidth',3);
title('Simulation␣of␣water␣levels␣(old␣model)','fontsize',18)
legend('Measured␣Merkebekk','Measured␣Dalsfos',...
'Simulated␣Merkebekk', 'Simulated␣Dalsfos','fontsize',18)
ylabel('Water␣level␣[m.a.s.l.]','fontsize',18)
xlabel('Days','fontsize',18)
grid on
```