NTNU
Norwegian University of
Science and Technology

Real-Time Control of Temperature and Light Intensity in non-square Multiple-Input-Multiple-Output Plant

Sindre Johan Heggheim

December 19th, 2017

Chemical Engineering and Biotechnology	
Submission date:	December 19th, 2017
Supervisor:	Associate Professor Johannes Jäschke, IKP*
Co-supervisor:	PhD Candidate Tamal Das, IKP*

Norwegian University of Science and Technology
*Department of Chemical Engineering

Preface

This thesis was performed as a specialization project at Department of Chemical Engineering at Norwegian University of Science and Technology (NTNU). The main goal of the thesis is to study the model identification and possible control structures for a real-time Multiple-Input-Multiple-Output system.

I would like to thank my supervisor Associate Professor Johannes Jäschke and my co-supervisor Tamal Das for the project task and the help they have given me during the work.

Declaration of Compliance

I declare that this is an independent work according to the exam regulations of the Norwegian University of Science and Technology (NTNU).

Sindre Johan Heggheim

Trondheim, December 19th, 2017

Abstract

This work contain a study of a real-time Multiple-Input-Multiple-Output (MIMO) system connected to Matlab and Simulink. The system contain three manipulated inputs (power to bulb, power to fan and power to LED) and two controllable outputs (measured temperature and measured light intensity).

The purpose of the work was to identify models describing the interactions between the inputs and outputs in the system and implement controllers for the MIMO system with decentralized controllers. This was done with several Single-Input-Single-Output (SISO) controllers where it switches control structures for different control regions. The work also look into more centralized and optimized MIMO control structures where Linear Quadratic Regulator (LQR) and Model Predictive Control (MPC) were studied and tested.

The model identification was done by estimating transfer functions from measured step responses and the use of functions from “System Identification Toolbox” in Matlab. Estimated state space models was calculated from the estimated transfer functions using “Control System Toolbox”.

The SISO controllers in the system was tuned using Simple Internal Model Control (SIMC) tuning. The MPC object was made with “Model Predictive toolbox” using the estimated model. The LQR gain was calculated with “Control System Toolbox” and the state space model.

This work concluded that for the real-time plant the MPC is the easiest solution. The LQR will be a Linear Quadratic Gaussian (LQG) since it need to estimate the states not measured. The LQG has more difficult implementation and state estimations than the MPC, and it goes unstable. The MPC has the option to take the input constraints into consideration. The decentralized MIMO structure implemented needed to switch focus and has therefore conditions and several optional controllers which makes the control structure bigger and more difficult than the MPC.

It is recommended that the further work goes into the identification of the models. A better identification might also identify the noise and add it separately so it can be used to improve the LQG calculations and state estimation. It is also possibilities to implemented better decentralized MIMO structure that takes better care of the controlled Light Intensity.

Sammendrag

Dette arbeidet ser nærmere på et “real-time” system med flere pådrag og flere regulerte variabler (MIMO). Systemet har tre pådrag (strøm til lyspære, strøm til Vifte og strøm til LED-pære) og to regulerte variabler (målt temperatur og målt lys intensitet).

Hensikten med arbeidet var å identifisere modeller som beskriver interaksjonene mellom pådragene og de regulerte variablene i systemet og implementere desentraliserte regulatorer for MIMO-systemet. Dette ble gjort ved hjelp av Enkel-Inngang-enkel-utgang (SISO) regulatorer som endre struktur etter reguleringsområde. Arbeidet så også på mer sentraliserte og optimaliserte regulatorer som Linear Quadratic Regulator (LQR) og Modellbasert Predektiv Regulering (MPC).

Identifiseringen av modellene ble gjort ved å estimere overføringsfunksjoner fra målte responser og “System Identification Toolbox” in Matlab. Estimerte “state space” modeller ble funnet ved å bruke de estimerte overføringsfunksjoner og “Control System Toolbox” i Matlab.

SISO regulatorene ble stilt inn ved hjelp av enkel IMC (SIMC). MPC’en ble laget ved hjelp av “Model Predictive toolbox” i Matlab og den estimerte MIMO modellen. Forsterningen fra LQR ble funnet ved hjelp av state space modellen og “Control System Toolbox”.

Arbeidet konkluderte med at MPC gir den enkleste og beste løsningen på regulator for systemet. For å bruke LQR på dette systemet må tilstandene estimeres og LQR’en er derfor en “Linear Quadratic Gaussian” (LQG). LQG ga mer ustabile løsninger og vanskeligere implementeringer og estimeringer i strukturen. Siden MPC også kan implementere begrensninger på pådragene vil den ikke være like ustabil som LQG. Den desentraliserte løsningen var mer komplisert og var nødt til å bytte mellom strukturer etter hvordan tilstanden til systemet var.

Det foreslås at videre arbeid ser nærmere på model identifikasjonen, slik at også støyen estimeres slik at den kan brukes i LQG utregningen og tilstandestimeringen. Det foreslås også å se på mulighetene for å implementere bedre desentralisert MIMO struktur slik at lys intensitet reguleringen blir bedre.

Table of Content

List of Figures

List of Tables

1 Introduction

The easiest and most common part in control theory is Single-Input-Single-Output (SISO) systems that contains one Controlled Variable (CV) and one Manipulated Variable (MV). By measuring the system output the CV can be controlled in a feedback controller. According to Otto Mayr (1970) the feedback controller systems has been around since the water clock of Ktesibios in 250 B.C [1]. But the rightfully feedback controllers with closed loop mechanism was first used in the medieval clocks. The more modern three mode feedback PID-controller (Proportional control, Integral control and Derivative control) came first in the 1930's and is now the most used basis controller [2].

In industrial processes the systems is often, more complex and can be a Multiple-Input-Multiple-Output (MIMO) system. The number of inputs and outputs depends on how many MV's needed to have a good control of the CV in the whole range it is supposed to be controlled in. In MIMO systems the inputs might have interactions on several of the outputs and the complexity of the control increases. The selection of the MV-CV pairing in the multi-loop system, is not as easy when there is interactions between several inputs and outputs. A way of analysing the multivariable control system is the Relative Gain Array (RGA) method developed by Bristol in 1960's [2]. In decentralized control the system is controlled by several other controllers like SISO or Multiple-Input-Single-Output (MISO) controllers. Simple Internal Model Control (SIMC) is a method for tuning the PID-controllers. The method uses transfer function models of the response to perform the tuning.

A more centralized and optimized control of the full MIMO is to use Linear Quadratic Regulator (LQR) or Model Predictive Control (MPC). The LQR is a feedback controller where the gain is optimized. The MPC and the LQR is tuned by choosing the penalty-weights on the deviations in the inputs and outputs of the system. The MPC can also implement constraints on all variables and step constraints. MPC was first developed in the end of the 1970's and is popular alternative when controlling multivariable systems [2]. The MPC uses models of the process and measurements to predict and optimize a sequence of changes in the MV's, and implements the first step in the sequence and reoptimizes a new sequence of changes. State space models are used to optimize the tuning of these control structures.

1.1 Report Structure

The work starts with identification of models of the responses in the system. The models are then used to identify and tune control structures for the plant. Decentralized control and LQR and MPC will be tested on the plant.

The structure of the report is the following:

1. **Introduction** includes a brief description of goals of the project and the methods used. It also describes the structure of the report.
2. **Theory and System Description** includes a description of the equipment/plant to

be controlled. The section also describes the structure of the MIMO system and the software and programs that was used during the project. The section also goes through the theory of the project.

3. **Approach and Results** describes the approach during the work and is presenting the result and comments on the structures.

4. **Conclusion** gives the final comments on the methods and control structures.

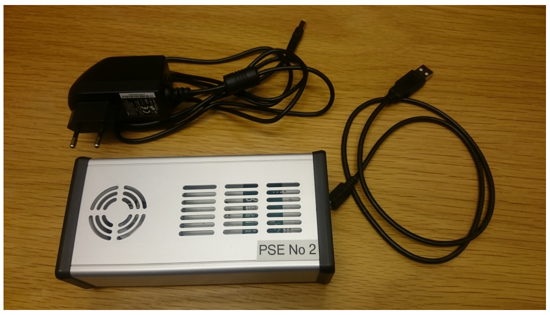
The identified models of the process responses and the plots of the SISO controls are given in the appendix. The main Matlab scripts and Simulink diagrams are also given in the appendix.

2 Theory and System Description

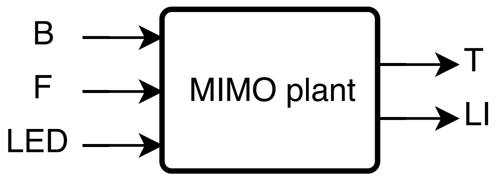
The objective of the project was to implement control structures in a real-time MIMO plant. The plant was easily connected to a computer and controlled with Simulink. The plant was looked at as a black box system where only the outputs and inputs was known and used further. No more theory or analysis of the plant processes was used.

2.1 System and Equipment/Plant

The plant is a small box that is connected to a computer with an universal serial bus USB. A picture of the plant is shown in Figure 1a. The plant has a bulb (B), a fan (F) and a light-emitting diode (LED). The plant can measure the temperature (T) and the light intensity (LI) inside the plant. The three MV's or system inputs are the power supplied to the bulb, the fan and the LED. The measured temperature and light intensity are the CVs or system outputs. The structure of the MIMO system with the inputs and outputs is shown in Figure 1b. The system inputs and outputs (power and measurements), will in this work be mentioned with capital front letters, where the physical parts of the plant will not. The order of the inputs and outputs given above will be used when numbering inputs and outputs later on in this work.



(a) Equipment/plant



(b) MIMO system structure

Figure 1: Picture of the plant/equipment and MIMO structure. The plant has a bulb, a fan and a LED and can measure the temperature and light intensity inside the plant. The plant has an external power supply and an USB to connect to the computer. The plant comes with installation files, and can be directly connected and manipulated inside Simulink.

The equipment came with installation files, Simulink library and initialization variables that allows the system to be connected and used in Simulink in real-time. The time unit used in Simulink as well as all time units in this work is given in seconds. The length of the time steps in the plant, T_s, was set to 0.1 sec. The equipment can measure more variables than the temperature and the light intensity, but the temperature and light intensity is the only measured output considered in this work. The units of the outputs are not considered in this work and the measurements are used directly. Temperature in this work is given in unspecified degrees (°). All inputs are unitless between 0 to 1.

The software that was used in this work is Matlab and Simulink. “System Identification toolbox” was used to estimate the response models and “Control System Toolbox” was used find state space models of the system and to calculate the LQR gain. “Model Predictive Control Tooolbox” was used during the building and testing of the MPC control structure.

2.2 Process Models

This work mainly work with transfer function models. The transfer functions are an alternative Laplace transform model of the ordinary differential equation (ODE), and is specified with the parameters: gain, k, time delay, θ and time constants, τ_i of orders i [2]. In first order responses the transfer function between the input, u, and the output, y, is

$$y = \frac{k}{\tau_1 s + 1} \mathrm{e}^{-\theta s} u \tag{1}$$

The gain k is the step in y from initial state to new steady state. τ_1 is the time constant and is the time from y starts to change until it reach $1 - 1/\mathrm{e} \approx 63\%$ of the gain step. θ is the delay present in the process. No delay present in the response gives that $e^{-\theta s} = 1$.

State space models is a general class of ODE [2]. A linear state space model is given as

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{2}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{3}$$

where $\mathbf{x}$, $\mathbf{u}$ and $\mathbf{y}$ is vectors of the states, outputs and inputs respectively. $\dot{\mathbf{x}}$ is the vector of time derivatives of the states and $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ is matrices describing the system combinations.

2.3 PID Tuning

PID-control can be tuned from transfer function describing the open loop response. In SIMC tuning the derivative part of the PID controller is zero if the open loop response has a first order dynamic, or if the approximated second order response has a second order time constant lower than the delay ($\tau_2 < \theta$) [3]. The PI controller and SIMC tuning of first order transfer functions are given by

$$c(s) = K_c \left(\frac{\tau_I s + 1}{\tau_I s} \right) \tag{4}$$

$$K_c = \frac{1}{k} \frac{\tau_1}{\tau_c + \theta} \tag{5}$$

$$\tau_I = \min\left(\tau_1, 4(\tau_c + \theta)\right) \tag{6}$$

where K_c is the proportional controller gain, τ_I is the integration time constant of the controller and τ_c is the tuning parameter.

Tight control is achieved by choosing $\tau_c = \theta$ and tuning the parameter larger gives a more slow and robust PI controller [3]. A way of calculating the maximum τ_c (no unnecessary large τ_c) is to calculate it from maximum K_c from known disturbances or changes. If the known input change, Δu, is needed to perform the output change, Δy, then the maximum need $K_c = \frac{\Delta u}{\Delta y}$ and $\tau_c = \left| \frac{1}{K_{c,\max}} \frac{\tau_1}{k} \right|$. This tuning gives a no saturation limit for a Δy step in the setpoint, given that the input has a Δu available before it saturates.

2.4 Relative Gain Array

In multivariable processes it can be difficult to identify the best pairing between MVs and CVs. The Relative Gain Array (RGA) method developed by Bristol in 1966 is a systematic approach for identifying the pairings and identify where the highest interactions will be in the process [2]. The elements λ_{ij} in the array is defined as the ratio between steady state gain with constant u and the steady state gain with constant y.

$$\lambda_{ij} = \frac{(\partial y_i/\partial u_j)_u}{(\partial y_i/\partial u_j)_y} \tag{7}$$

u_j and y_i is the system input number j and system output number i respectively. The first column in the RGA shows the different interactions (relative gains) on all outputs y_i, from the first input u_1. In a square non-singular plant the sums of the elements on a row or column is equal 1, and the best pairing will be the element nearest 1 [2]. If the element is 1 then the only effect from the corresponding input is on the corresponding output. In a non-square plant the sums of the RGA do not have to be 1, but the sums still give a indication on the good pairings [4]. If the sum of a column is much lower than 1, then the corresponding input have small interaction and there is other inputs that gives the main interactions on the process.

The RGA of the system with the steady state, square plant $\mathbf{G}$ is

$$\text{RGA}(\mathbf{G}) = \mathbf{G} \times (\mathbf{G}^{-1})^\intercal \tag{8}$$

where $\times$ is element-by-element multiplication [4]. For a non-square $\mathbf{G}$, the pseudo inverse, $\mathbf{G}^\dagger$ is used.

$$\text{RGA}(\mathbf{G}) = \mathbf{G} \times (\mathbf{G}^\dagger)^\intercal \tag{9}$$

In Matlab the RGA is calculated by

$$\text{RGA} = \text{G.*pinv(G.')}; \tag{10}$$

2.5 Model Predictive Control

The MPC uses a model of the process and the measurements to optimize a future sequence of input steps. The prediction horizon, p_h is the number of steps in the sequence. The main advantages [2] with MPCs is

1. The process model describes the interactions in the process.
2. Possible to implement constraints on CV and MV.
3. Can coordinate process control with optimal operation condition.
4. Good model predictions can give early warnings.

The summarized objective of the MPC is to drive the process to an optimal setpoint or reference point while maintaining constraints on where the variables can move and how they can move.

The process movements comes from minimizing the cost function J. The tuning of the MPC is to set weights on different part of the cost function containing deviations from optimal reference. The constraints in the MPC includes the constraints like operational limits for input and outputs and constraints of the physical process movements given in state space model.

The cost function used in the MPC in "Model predictive control toolbox" in Matlab [5] without soft constraints is

$$J = J_y + J_u + J_{\Delta u} \tag{11}$$

where J_y is the cost of deviations between the outputs, y_i, and their respective references, r_i, given as

$$J_y = \sum_{i=1}^{n_y} \sum_{p=1}^{p_h} \left\{ \frac{w_i^p}{s_i} \left[r_i^{k+p} - y_i^{k+p} \right] \right\}^2 \tag{12}$$

n_y is the number of output variables, p is the index of the step in the prediction horizon from the current time step k. w_i^p is the tuned weight on the output variable i at prediction step p and s_i is a scaling factor of the given output variable. The tuned weight is often equal during the prediction horizon. It can also have increased weight at the end of the prediction horizon.

J_u is the cost of deviations between the inputs, u_j, and their respective references or target values, r_j, given as

$$J_u = \sum_{j=1}^{n_u} \sum_{p=0}^{p_h-1} \left\{ \frac{w_j^p}{s_j} \left[u_j^{k+p} - r_j^{k+p} \right] \right\}^2 \tag{13}$$

n_u is the number of input variables. w_j^p is the tuned weight on the output variable j at prediction step p and s_j is the scaling factor of the output variable. $J_{\Delta u}$ is the cost of the changes in the inputs.

$$J_{\Delta u} = \sum_{j=1}^{n_u} \sum_{p=0}^{p_h-1} \left\{ \frac{w_{\Delta j}^p}{s_j} \left[u_j^{k+p} - u_j^{k+p-1} \right] \right\}^2 \tag{14}$$

The weights on the changes in input does not set any constraints on changes, but specifies what changes that are not as desirable. Any constraints on movement need to be specified separately together with the constraints and not in the cost function.

2.6 Linear Quadratic Regulator

In linear quadratic control there is no equality constraints and the only constraints are the state space model [6]. The LQR is a state feedback controller

$$u_t = -Kx_t \tag{15}$$

u_t is the system input and x_t the corresponding state and K is the state feedback gain that minimizes the objective function over the infinite time horizon. The objective function is given as

$$f^{\infty} = \sum_{t=0}^{\infty} \frac{1}{2} x_{t+1}^{\intercal} Q x_{t+1} + \frac{1}{2} u_t^{\intercal} R u_t \tag{16}$$

Where Q and R is matrices specifying the weightings and t is the index of the time step. The LQR gain is constant over the whole infinite time horizon and is given by

$$K = R^{-1} B^{\intercal} P (I + B R^{-1} B^{\intercal} P)^{-1} A) \tag{17}$$

$$P = Q + A^{\intercal} P (I + B R^{-1} B^{\intercal} P)^{-1} A \tag{18}$$

where $P = P^{\intercal} \leq 0$ [6]. A and B is the matrices from the state space model and I is an identity matrix.

The LQR need feedback of all states and if not all states are measured the LQR will need to estimate the states. LQR where the estimation of the states is done with Kalman filter is called a linear quadratic Gaussian (LQG) The structure of the LQG is illustrated in Figure 2

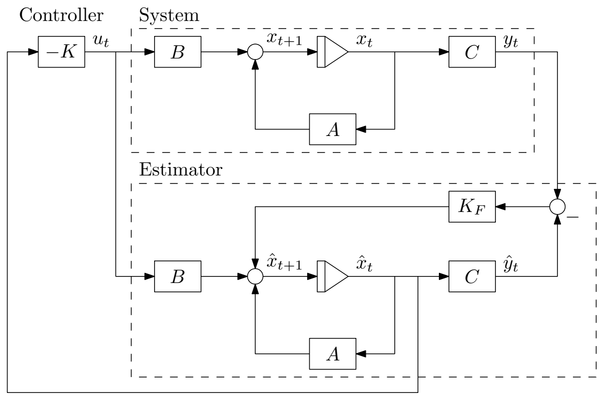


Figure 2: Illustration of LQG structure [6].

3 Approach and Results

The transfer function describing the system interactions was estimated from measured responses between the system inputs and outputs. The system of transfer functions was used to analyze the system with the RGA method, and also used to tune SISO, MISO and MIMO control structures. LQR and MPC control structures was also implemented and tested on the MIMO plant.

The approach and result and plots showing the control will be shown and commented continuously.

3.1 Model Identification

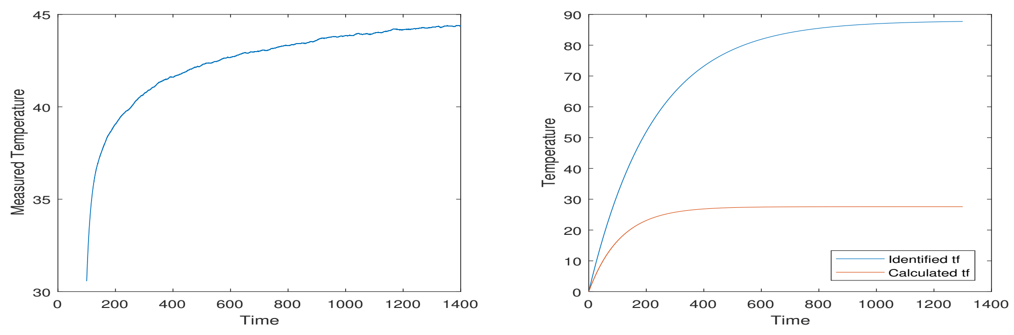
The plant was first installed and connected to Simulink, so it could communicate with Simulink in real-time. The system was then tested for step changes in all the inputs separately and the responses in the outputs measured. It was shown that responses of first order would give good estimated responses, where the fast responses in Light Intensity with nearly zero order dynamic could be estimated with small first order time constants τ_1.

The measured responses to the inputs was then used by the **tfest** function in Matlab to estimate the transfer functions with specified order. The response from Fan to Light Intensity and LED to Temperature was neglected and set as a 0 function. Any process delay was neglected ($\theta = 0$). It was found that the estimated functions describing the responses in Temperature had to high gain from Bulb and too low negative gain from Fan. The first order response parameters (τ_1, k) was therefore calculated directly from the Temperature response. The measured responses in temperature and the estimated responses are shown in Figure 3a and 3b. It was found that the estimated transfer functions describing the Light Intensity responses from Bulb and LED was good, and they was kept without any additional calculation. The measured responses in Light Intensity from Bulb and LED and the estimated response are shown in Figure 3c and 3d. The parameters describing the estimated first order responses are given in Table 1.

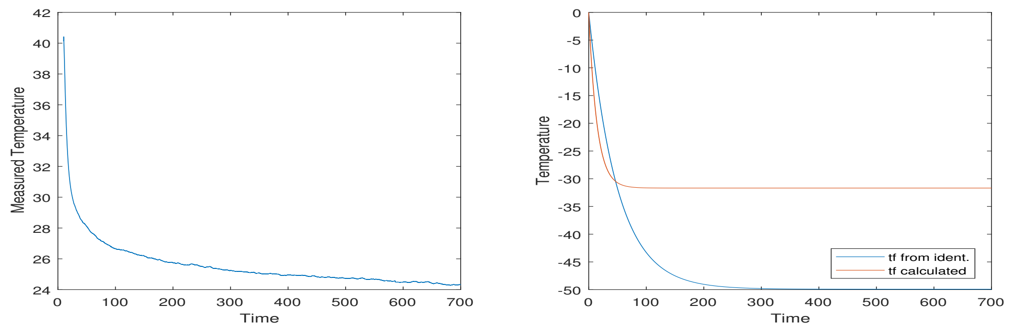
Table 1: First order response parameters of estimated models. The response from Fan to Light Intensity and from LED to Temperature was neglected. All process delays θ was neglected.

Response	k	τ_1
Bulb to Temperature	27.5800	110.0000
Fan to Temperature	-31.6800	14.3000
Bulb to Light Intensity	48.9019	0.1556
LED to Light Intensity	8.9218	0.0557

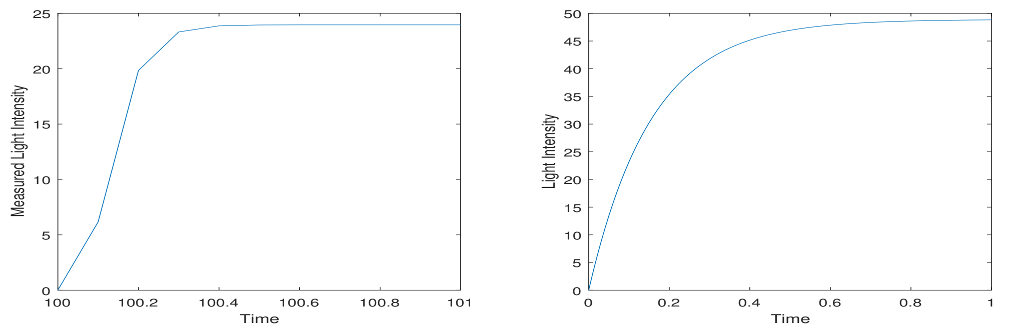
The full MIMO model was made by combining the transfer functions. The MIMO transfer function model was then used by the **ss** function in Matlab to calculate the state space model. The MIMO transfer function model and the state space model on the form given in (2) and (3) are shown in Appendix A.



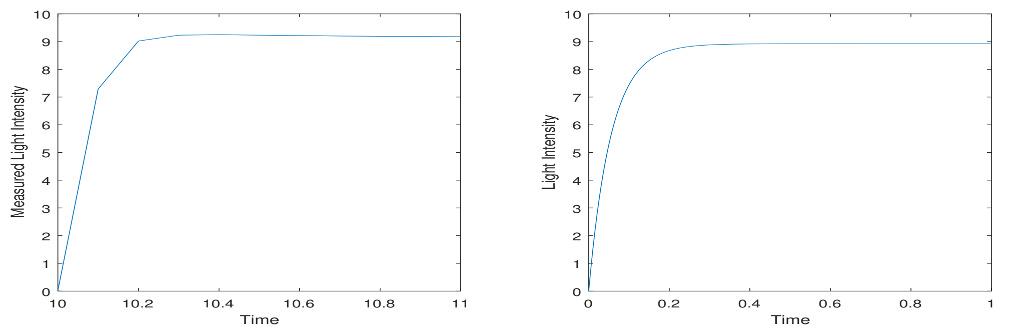
(a) Bulb to Temperature response.



(b) Fan to Temperature response.



(c) Bulb to Light Intensity response.



(d) LED to Light Intensity response.

Figure 3: Measured and estimated responses. The input steps during the step response test was 0.5 for Bulb and Fan and 1 for the LED. The input steps in the estimated responses shown are 1. Note that the different step times have no impact on the estimations and that there is no delay in any of the responses.

3.2 RGA Analysis

The models above was used in a RGA analysis to get get an overview of the main process responses and to get an indication of good control pairing and structures. The steady state gains of the plant system **G** was found from the transfer functions and the RGA calculated with equation (10). The **G** and RGA matrix with the sums of the rows and columns in the RGA is given as

$$G = \begin{bmatrix} 27.5800 & -31.6800 & 0 \\ 48.9019 & 0 & 8.9218 \end{bmatrix} \qquad RGA = \begin{array}{c} \begin{bmatrix} 0.0238 & 0.9762 & 0 \\ 0.9447 & 0 & 0.0553 \end{bmatrix} \begin{matrix} 1 \\ 1 \end{matrix} \\ \begin{matrix} 0.9686 & 0.9762 & 0.0553 \end{matrix} \quad \end{array}$$

The gain matrix **G** gives an indication of the inputs-output pairing because it shows where the highest output comes from. Since the LED has much lower gain than the Bulb on the Light Intensity the Bulb should be better to control the Light Intensity because it gives the highest output range. Since Fan has high negative gain on Temperature, the Fan can control the Temperature produces from Bulb to Light Intensity control. It can not hold the Temperature at minimum while the Light Intensity is maximized from Bulb, because the absolute value of the gain from Fan is lower than the gain from Bulb. The RGA confirms the thoughts about the control pairing. The LED has bad Light Intensity control compared to other process interactions. The RGA shows that the best pairing is to use Bulb to control Light Intensity and Fan to control Temperature. Note that low Light Intensity is not possible while the temperature is supposed to be high. How the plant operation is going to be is not specified, and several possible operational solution is possible. The RGA analysis concludes with the pairing of Bulb to Light Intensity and Fan to Temperature, but with the non-square plant it is more optimal to also include the extra input to the control structure.

3.3 SISO Control

SISO controllers was implemented and tested on the estimated models and the real-time plant. The tests was used to check the four possible pairings in real-time and to check the quality of the estimated models. During the SISO testing the other output was not taken into any consideration.

All the controllers was tuned with SIMC PI tuning and no delay. Since there is no delay, the tuning parameters, τ_c, could not be set equal to the delay. The Light Intensity controllers was tuned with $\tau_c = 0.5$. The controllers was first tuned and simulated with lower tuning parameters, but the tuning was not usable during the real-time control of the plant. Because of unstable oscillation, the parameter was tuned to be at least 0.5. This tuning gave a real-time control on the edge of going to unstable oscillation. Because of slow dynamics in the temperature, the Bulb to Temperature and Fan to Temperature was tuning slowly and robust to avoid any overshoots. It was assumed that a change of 5°C could be expected at operation and the $\tau_{c,\max}$ was calculated for this expected change and $\Delta u = 1$.

The SISO control on the models and the real-time plant is shown in Appendix B. The Temperature models show much similarities and gave good control, but the real-time Fan gives bad Temperature control near minimum Fan input. The assumed reason was that the fan has resistance and was not able to turn very slowly and respond to very small inputs. When the fan finely was able to turn it would give to high response and result in spikes. The real-time control on Light Intensity was less stable than the simulated control.

3.4 MISO Control

To improve the control of the plant outputs, MISO controllers was implemented and tested. The MISO controllers took use of the extra input to increase the total gain and the dynamic of the control. It was two possible MISO control structures for the plant. The first MISO uses Bulb and Fan to control Temperature and the second MISO uses Bulb and LED to control the Light Intensity. The MISO systems is illustrated in Figure 4.

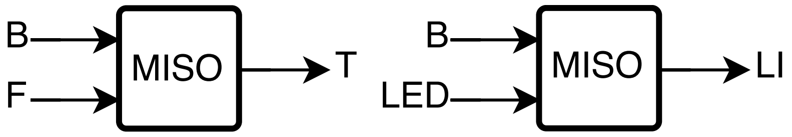


Figure 4: Illustration of possible MISO control structures.

3.4.1 Temperature

The MISO Temperature control structure combined the two SISO Temperature controllers. Since the Fan and Bulb had opposite sign on their gains the two controllers was two opposite constraint controllers. Overshoots from one of the inputs would be corrected by the other input. Because of the corrections between the inputs, the MISO control could be tuned much more aggressively than the earlier SISO controllers. The MISO control was tested on the estimated models and it was shown that the control was good with small Integrated Absolute Error (IAE) and with much faster response. The problem with the MISO control was that it stabilized at high input usages in both inputs. The control is shown in Figure 5.

The high input usage in the simulation was also shown in the real-time testing, where the plant got high input usage and oscillations. The problem was avoided by implementing "setpoint back-off" on one of the controllers. The same setpoints made the two controllers fight over the same position and the error in the real-time plant resulted in high oscillations in the system. The back-off allows the CV to stabilizes at one of the setpoints. This idea of the setpoint back-off is illustrated in Figure 6.

The MISO control was tuned with $\tau_c = 0.1$ for the Bulb and $\tau_c = 1$ for the Fan and the setpoint back-off was implemented with $0.15°$ higher setpoint for the Fan. The real-time control is shown in Figure 7. The real-time Temperature was shifted down $27°$.

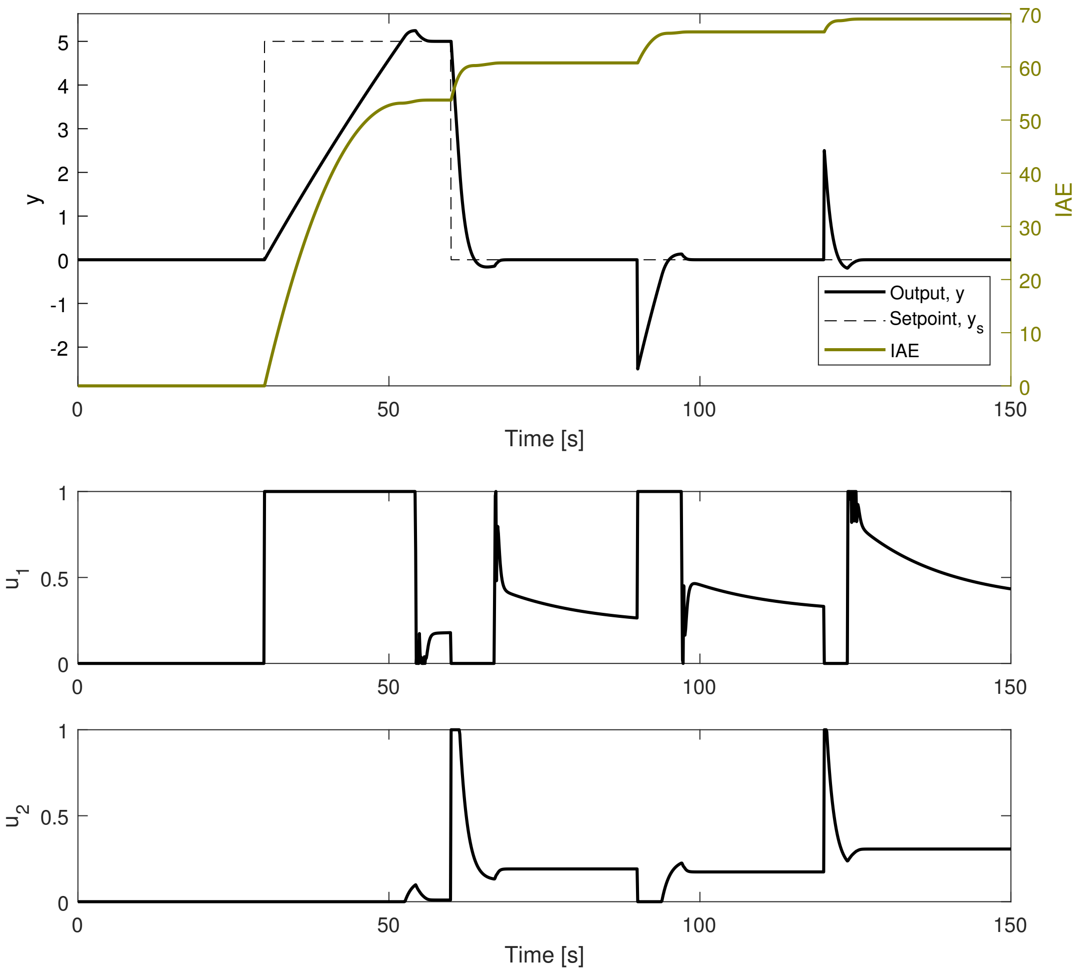


Figure 5: Simulation of MISO Temperature control. The two opposite constraint controllers resulted in high input usage in both directions. There was no setpoint back-off implemented in this simulation. The tuning was $\tau_c = 0.1$ for the Bulb and $\tau_c = 1$ for the Fan. u_1 and u_2 is Bulb and Fan respectively and y is Temperature.

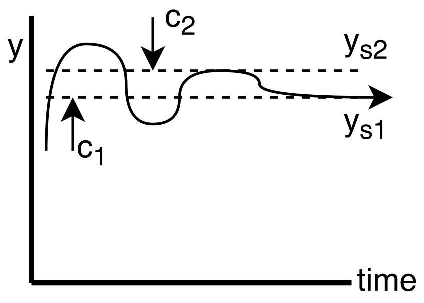


Figure 6: Illustration of setpoint back-off in two directional constraint control. The controller with negative gain (c_2) has a higher setpoint (y_{s2}) than the other controller (c_1). The two setpoints is for the same controlled output y.

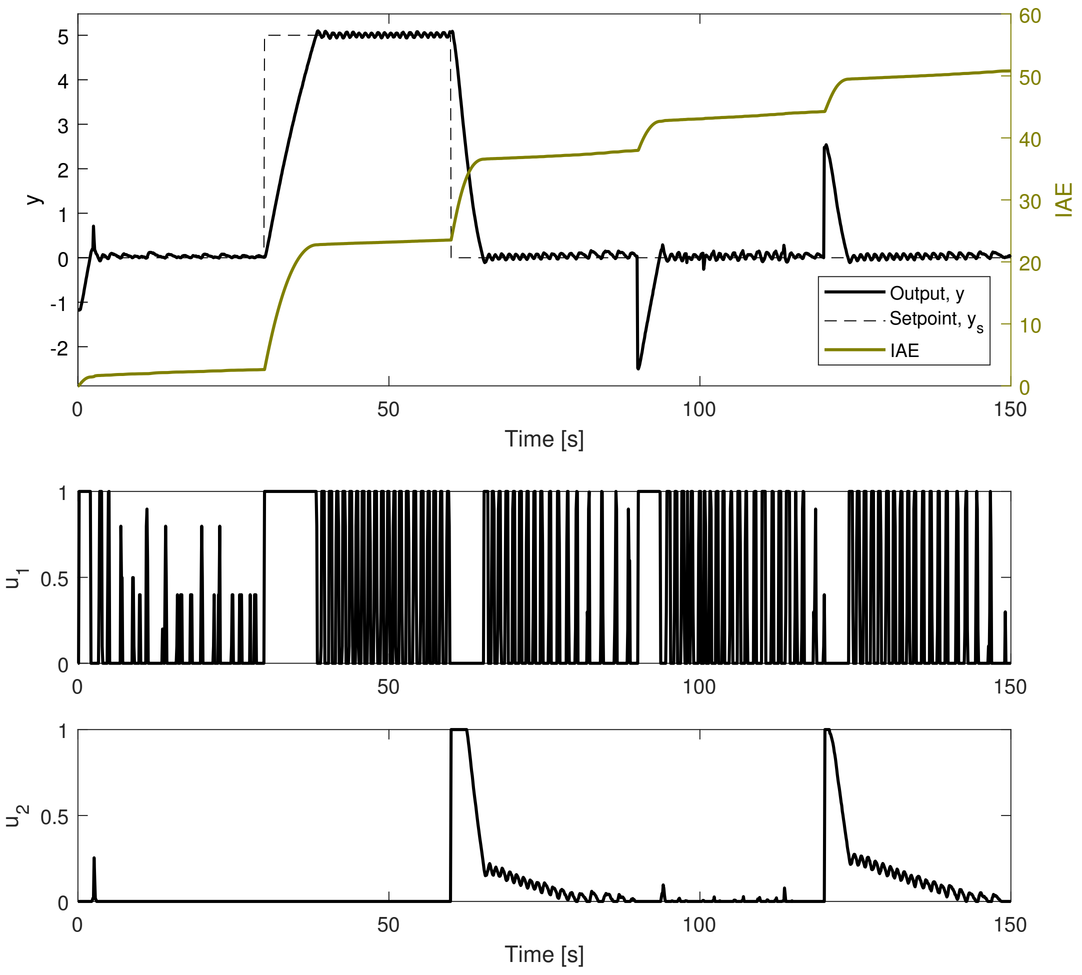


Figure 7: Real-Time MISO Temperature control with setpoint back-off. The Fan has a temperature setpoint 0.15° higher than the Bulb. The input usage with the back-off was much lower, but the plant would still show high input usage and oscillations. The tuning was $\tau_c = 0.1$ for the Bulb and $\tau_c = 1$ for the Fan. u_1 and u_2 is Bulb and Fan respectively and y is Temperature.

The setpoint back-off in the real-time control gave considerably reduction in the input usage compared to no back-off, where the Fan and Bulb would flash continuously. The implemented control with back-off still gave high oscillations and flashing in the Bulb. The high oscillations was reduced with implementation of alternative smooth control near the setpoints. The thresholds was set to 0.2° and the back-off in the new implemented structure was decreased to 0.1°. The alternative controllers was tuned with a 10 times larger parameter ($\tau_c = 1$ for Bulb and $\tau_c = 10$ for Fan). The real-time control with alternative smooth control near setpoint is shown in Figure 8.

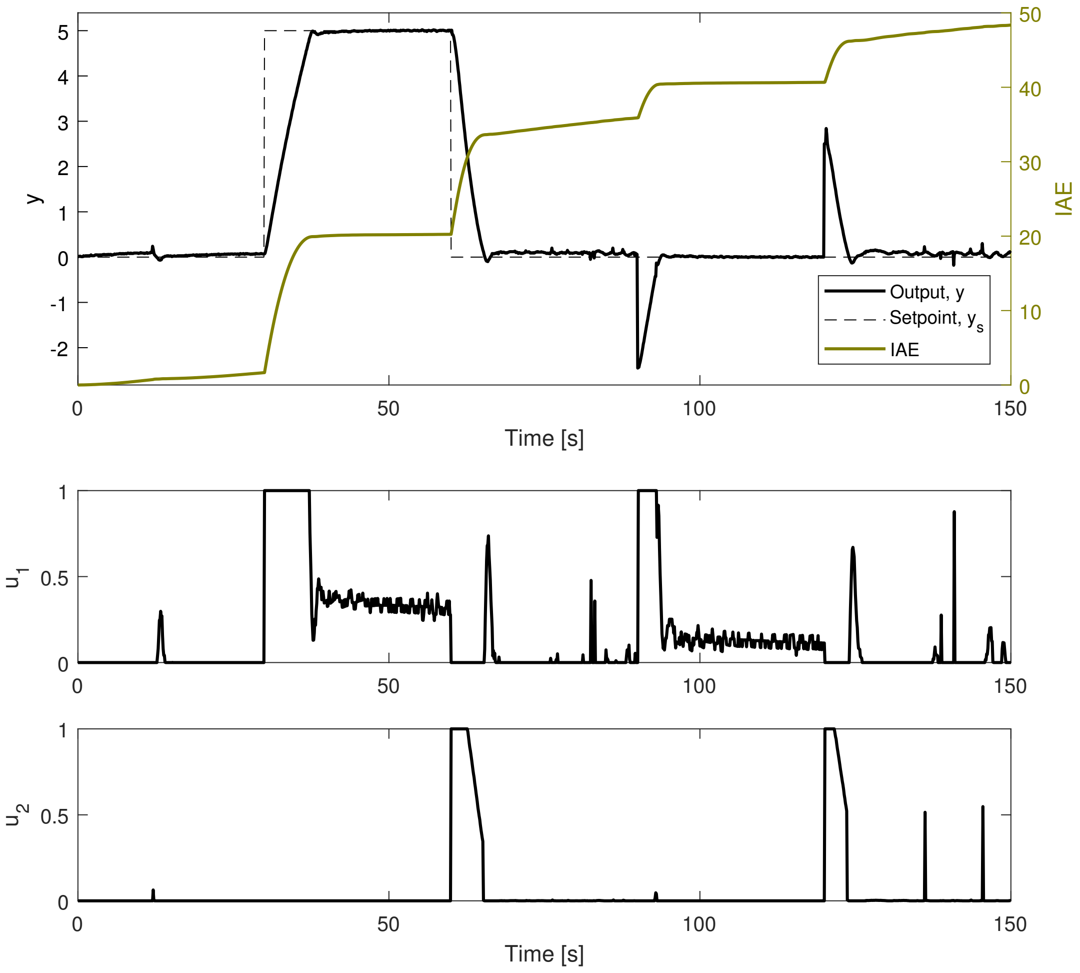


Figure 8: Real-Time control of MISO system with alternative smooth control near setpoint. The Fan had a Temperature setpoint with a 0.1° back-off. The tuning is $\tau_c = 0.1$ for the Bulb and $\tau_c = 1$ for the Fan. Near setpoint the tuning (τ_c) is 10 times larger. u_1 and u_2 is Bulb and Fan respectively and y is Temperature.

3.4.2 Light Intensity

In MISO Light Intensity control the main response is from Bulb, as shown in the RGA analysis in section 3.2. To get a good Light Intensity control the help from Bulb is necessary. There was not implemented any single test for MISO Light Intensity control, but some suggestions was considered and is discussed in this section. MISO Light Intensity control was implemented as a part of the MISO structure in Section 3.5.

One possible MISO control structure is split range control where the span of the Light Intensity is increased from adding Bulb when the LED is saturated. It is optimal to use the LED first because the use of Bulb do not result in any additional use of Fan if Temperature is taken into consideration.

If the plant is supposed to change the Light Intensity regularly, then changes in Bulb is not desirable since it also interact with the Temperature. A structure that can avoid most of these changes in the Temperature is using input resetting control. The Bulb is then tuned much slower than the LED, so that the LED have the fastest dynamic and takes care of the main changes in the Light Intensity. The Bulb is then used to control the LED input to a wanted operational LED input. A block diagram of the input resetting structure is illustrated in Figure 9.

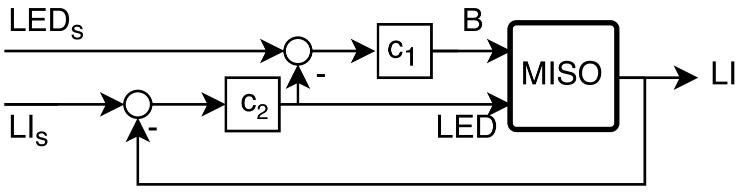


Figure 9: Illustration of input resetting in MISO Light Intensity control. LED$_s$ is the setpoint for the LED input at 50%. The Bulb uses the deviation from this setpoint to help LED back to 50%.

3.5 Decentralized MIMO control

A MIMO control structure was implemented using the structures discussed above. It was assumed that the Temperature was the main output and that deviations in Light Intensity was not as undesirable than deviations in Temperature. It was also assumed that tight control was a goal and that all inputs are to be used to optimize the control. Since the Temperature was the main objective the Bulb was mainly used to control Temperature, but if the temperature from Bulb to Light Intensity was higher than the Temperature setpoint the Fan could control the temperature by its own. Estimated Temperature from Bulb to Light Intensity control was therefore used to select which of the outputs the Bulb should focus on. The block diagram showing the steady state gains from Bulb is illustrated in Figure 10.

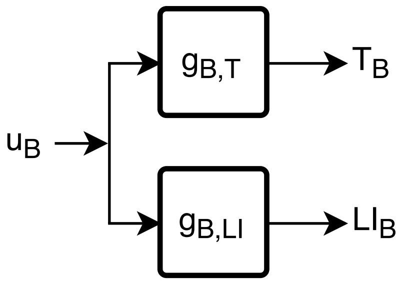


Figure 10: Block diagram of the interactions from Bulb.

The steady state Temperature and Light Intensity from a given Bulb input, u_B, is

$$T_B = g_{B,T} u_B \tag{19}$$

$$LI_B = g_{B,LI} u_B \tag{20}$$

$g_{j,i}$ is the steady state gain from input j to output i. If the Bulb controls the Light Intensity at the setpoint, LI_s, then

$$u_B = g_{B,LI}^{-1} LI_B = g_{B,LI}^{-1} LI_s \tag{21}$$

and the steady state Temperature from the Bulb becomes

$$T_B = g_{B,T} g_{B,LI}^{-1} LI_s \tag{22}$$

The temperature from the Bulb controlling Light Intensity can be estimated from steady state gains $g_{B,T}$ and $g_{B,LI}$ and the Light Intensity setpoint. This estimated temperature was implemented and used condition to switch between control structures in the MIMO control structure.

The nominal operational conditions was chosen as 31.51° in Temperature and 26.50 in Light Intensity. Temperature was shifted 27°. The control structure was tested on the real-time plant and is shown in Figure 11. Because the Light Intensity was taken into consideration in the MISO the tuning of Bulb to Temperature was much slower. This was to avoid the high interactions Bulb gives on Light Intensity. Because of the slower tuning there was no need of the alternative smooth controllers near setpoint and they was not implemented. Since the Bulb can switch between the focus on Temperature and the focus on Light Intensity the setpoint back-off on the Fan controller was implemented and set to 0.1°.

It was shown in the real-time control that the Fan still gave bad control near zero input. When the Bulb was controlling the Temperature the LED and Light Intensity became more noisy because of the changes in Bulb from the noisy Temperature control. The Bulb did switch good between the two outputs. The controlled output with the MISO Bulb, did show the best control, And as long as the temperature could be high enough for the Fan to control control the temperature alone the system was controlled very stable.

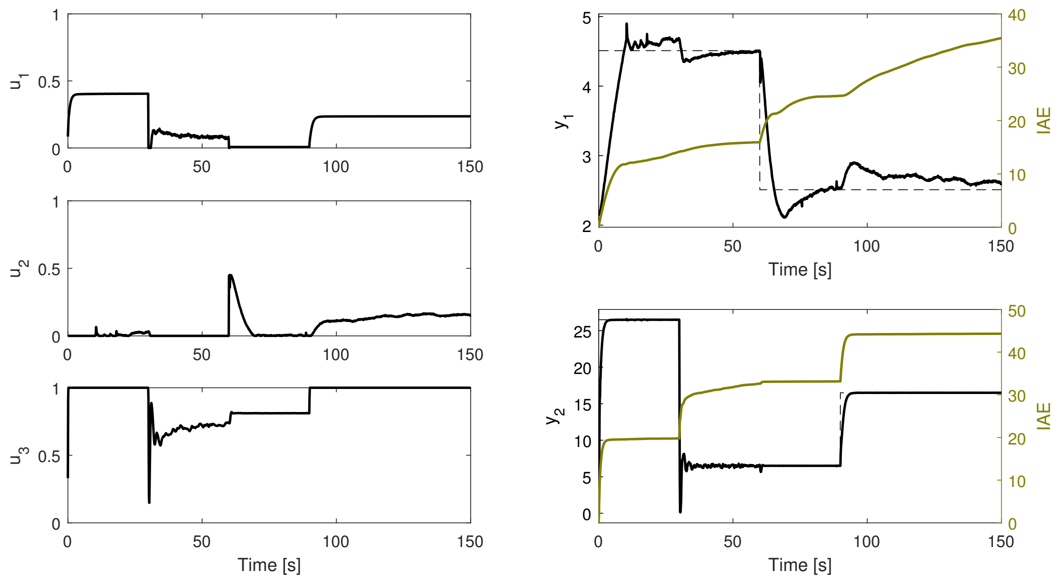


Figure 11: Real-Time MIMO Temperature and Light Intensity control. The tuning was $\tau_c = 5$ for the Bulb to Temperature and $\tau_c = 2$ for the Fan to Temperature. The Bulb to Light Intensity had $\tau_c = 1$ and LED to Light Intensity had $\tau_c = 0.5$. When the Bulb was controlling the Temperature the Light Intensity had more oscillations. The Fan had 0.1° back-off. The Bulb mainly helped the Light Intensity because it gave high enough temperature, but between 30 sec and 60 sec the wanted Light Intensity was low and the Bulb had to control the Temperature instead.

3.6 Model Predictive Control

The MPC was implemented using the "Model Predictive control toolbox" in Matlab. The toolbox contains a MPC block for Simulink and a function for making the MPC object used by the block. In this work the tuning of the MPC was done in Matlab. It was also possible to use a Graphical user interface (GUI) in Simulink to adjust and tune the MPC. The problem with the GUI was that it would need to be tuned on the model. The tuning of the MPC directly in Matlab made it possible to easily test the MPC tuning directly on the real-time plant. The nominal operational conditions chosen are given in Table 2.

Variable	Symbol	Size
Temperature	y_1	31.51
Light Intensity	y_2	26.50
Bulb	u_1	0.5
Fan	u_2	0.2
LED	u_3	0.5

Table 2: Nominal operation conditions.

The constraints implemented in the MPC was the saturation limits for the three inputs.

$$0 \leq u_j \leq 1 \tag{23}$$

The cost function in the MPC is given as

$$J = J_y + J_u + J_{\Delta u} \tag{24}$$

$$J_y = \sum_{p=1}^{p_h} \left[\left\{ \frac{w_T^p}{s_T} \left[r_T^{k+p} - y_T^{k+p} \right] \right\}^2 + \left\{ \frac{w_{LI}^p}{s_{LI}} \left[r_{LI}^{k+p} - y_{LI}^{k+p} \right] \right\}^2 \right] \tag{25}$$

$$J_u = \sum_{p=0}^{p_h-1} \left[\left\{ \frac{w_B^p}{s_B} [u_B^{k+p} - r_B^{k+p}] \right\}^2 + \left\{ \frac{w_F^p}{s_F} [u_F^{k+p} - r_F^{k+p}] \right\}^2 + \left\{ \frac{w_{LED}^p}{s_{LED}} [u_{LED}^{k+p} - r_{LED}^{k+p}] \right\}^2 \right] \tag{26}$$

$$J_{\Delta u} = \sum_{p=0}^{p_h-1} \left[\left\{ \frac{w_{\Delta B}^p}{s_B} [u_B^{k+p} - u_B^{k+p-1}] \right\}^2 + \left\{ \frac{w_{\Delta F}^p}{s_F} [u_F^{k+p} - u_F^{k+p-1}] \right\}^2 + \left\{ \frac{w_{\Delta LED}^p}{s_{LED}} [u_{LED}^{k+p} - u_{LED}^{k+p-1}] \right\}^2 \right] \tag{27}$$

The weight on the Temperature was chosen to be much greater than the weight on Light Intensity because of the slower dynamic compared to Light Intensity. Since the Bulb interacted with both outputs it was the only input with weight on the deviation from the target value, and it was also the input with most penalty on rate of change in its value. The tuned weights made the Bulb stay more stable than the other inputs and the other inputs was forced to take care of their respective outputs. Since the stable Bulb input was set, the Fan was forced to add input to hold the temperature down, and therefore avoiding the problem with Fan input near zero. The tuned parameters in the MPC are given in Table 3.

Table 3: Tuned MPC parameters.

Parameter	Symbol	Value
Step length	T_s	0.1
Prediction horizon	p_h	100
Control horizon	p_c	10
Scaling factor all inputs	s_j	1
Scaling factor Temperature	s_T	27.58
Scaling factor Light Intensity	s_{LI}	48.90
Weight on Temperature	w_T	1000
Weight on Light Intensity	w_{LI}	1
Weight on Bulb	w_B	1
Weight on Fan	w_F	0
Weight on LED	w_{LED}	0
Weight on Bulb rate	$w_{\Delta B}$	1
Weight on Fan rate	$w_{\Delta F}$	0.1
Weight on LED rate	$w_{\Delta LED}$	0.1

The tuned MPC was tested to check if it could be controlled and stabilized at the nominal point given in Table 2. The test is shown in Figure 12. The MPC stabilized the plant good, but the startup procedure of the MPC was bad. The MPC started with initial Bulb input and makde the temperature overshoot the setpoint before it started its control of the plant. The testing of setpoint changes on the MPC structure is shown in Figure 13. Apart from the startup the MPC did show good control of the plant.

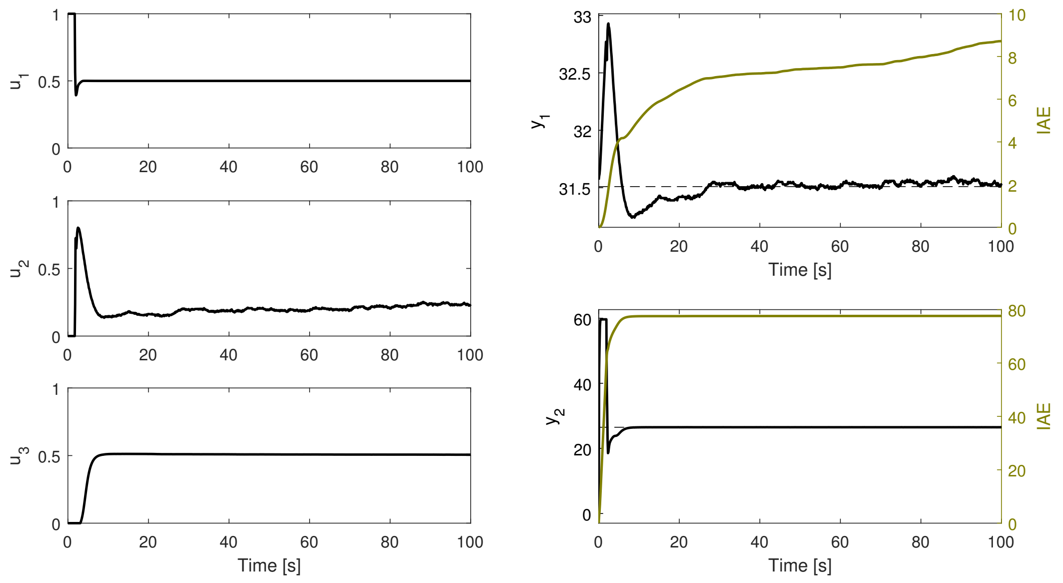


Figure 12: Stabilized control of plant with MPC. The nominal conditions is given in Table 2 and the tuned parameters of the MPC is given in Table 3. The MPC had bad startup procedure, where the initial Bulb made the temperature overshoot before the MPC started to control the plant.

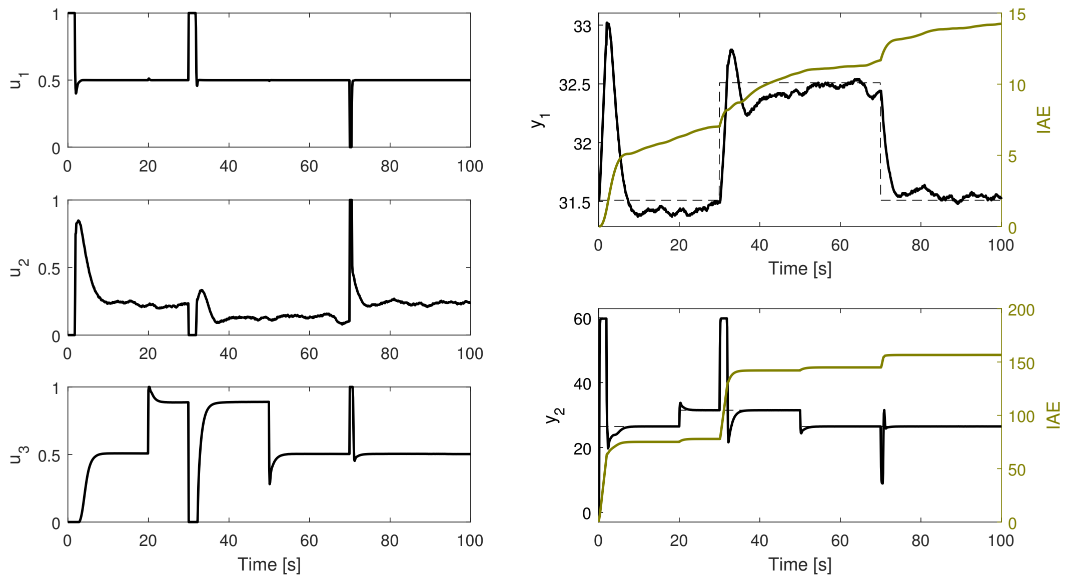


Figure 13: Test of setpoint changes on the MPC. The MPC had bad startup procedure. Apart from the startup, the control was good and the plant was controlled with stable Bulb, except for big changes in Temperature where the Bulb helps to change the Temperature. Because Temperature was assumed more important than the Light Intensity the big steps in Temperature setpoint result in errors in the Light Intensity. Since the LED was not weighted as much as the Bulb, the LED changed more easily and took care of most of the small changes in Light Intensity.

The problem with the bad startup could probably been avoided with implementation of a start condition or known initial state. If the initial states was known the MPC could start at these conditions and would not have to do anything before it was ready to do the calculations.

3.7 Linear Quadratic Gaussian

Since the plant did not have full measurement of the state, the states had to be estimated. The LQG control was not completed in this work and the last tests did go unstable. The Simulink diagram of the structure is given in Figure 26 in Appendix C. The input and output to the LQR gain was chosen as $x - x_0$ and $u - u_0$.

The LQG implementation was overridden in the favor of the MPC implementation described in the previous section. The MPC was instead implemented because of the possibility of constraints on the inputs. A reason for the unstable LQG is that it also tries to use negative inputs to control the system. Other reasons for the unstable control can be that it only was given bad tuning. The estimation could also been better if the model identification had taken care of the measured noise, and used it to tune the Kalman filter.

4 Conclusion

The responses in the system could be estimated as first order and therefore easy to estimate directly. The estimation from “system identification toolbox” did not give as good result as hoped, and it gave Temperature gains that was bad and also Light Intensity models that was not good. The Light Intensity was not a smooth first order response and should maybe not been estimated as a first order response.

The MIMO control of the plant was well enough and the switching between the placement of the MISO did work good. Comparing the difficult structure in the decentralized MIMO with the MPC, the tuning and implementation of the MPC was much easier. The control of the MPC was also better if the startup is not taken into consideration. The LQG had also easier structure than the big decentralized structure, but would probably not been any better than the MPC. The problem with the MPC implementation was that it had bad startup control. This could most likely been avoided with implementation start condition or known initial states.

It is recommended that any further work on the topic and plant look into better model identification, so that the noise also is a part of the identified model. The noise model can then be used to tune the LQG for better state estimation. The model of the system could also been better if it was derived from theory of Temperature and Light Intensity, but it would not have modeled the noise in the measurements and plant.

It is also recommended to look for a better decentralized MIMO structure that result in better Light Intensity control. An example is to implement the MISO input resetting structure into a MIMO structure. The MPC was able to do the same when weighting the LED to take care of the main changes in the Light Intensity and leaving the Bulb to run stable.

References

[1] Antonio M. Lepschy, Gian A. Mian, and Umberto Viaro. “Feedback Control in Ancient Water and Mechanical Clocks.” In: *IEEE Transactions on Education* 35.1 (Feb. 1992).

[2] Seborg et al. *Process Dynamics and Control.* Third edition. John Wiley & sons, Inc, 2011.

[3] Sigurd Skogestad and Chriss Grimholt. “The SIMC Method for Smooth PID Controller Tuning.” In: *PID Control in the Third Millennium: Lessons Learned and New Approaches.* Ed. by Ramon Vilanova and Antonio Visioli. London: Springer London, 2012, pp. 147–175. URL: `https://doi.org/10.1007/978-1-4471-2425-2_5`.

[4] Sigurd Skogestad and Ian Postlethwaite. *Multivariable Feedback Control, Analysis and design.* Second edition. John Wiley & sons, Inc, Aug. 2001.

[5] Mathworks. *Optimization Problem.* `https://se.mathworks.com/help/mpc/ug/optimization-problem.html`. Accessed: December 13th 2017. 2017.

[6] Bjarne Foss and Tor Aksel N.Heirung. “Merging Optimization and Control.” Department of Engineering Cybernetics, NTNU. 2016.

Abbreviations

B Bulb

CV Control Variable

F Fan

IAE Integrated Absolute Error

LED Light-Emitting Diode

LI Light Intensity

LQG Linear Quadratic Gaussian

MIMO Multiple-Input-Multiple-Output

MISO Multiple-Input-Single-Output

MPC Model Predictive Control

MV Manipulated Variable

ODE Ordinary Differential Equation

PID Proportional Integrating Derivative

RGA Relative Gain Array

SIMC Simple Internal Model Control

SISO Single-Input-Single-Output

T Temperature

USB Universal Serial Bus

Nomenclature

τ_i	Response time constant of order i
τ_c	Controller tuning parameter
τ_I	Controller integration time constant
λ_{ij}	Relative gain of input j and output i
θ	Response time delay
$\mathbf{A}$	System matrix
$\mathbf{B}$	Input matrix
$\mathbf{C}$	Output matrix
$\mathbf{D}$	Feedtrough matrix
$g_{j,i}$	Steady state gain from input j to output i
$\mathbf{G}$	Matrix of steady state gains
$\mathbf{G}^\dagger$	Pseudo inverse of matrix $\mathbf{G}$
I	Identity matrix
J	Total cost function
J_u	Cost function of input deviations
$J_{\Delta u}$	Cost function of input changes
J_y	Cost function of output deviations
K	State feedback gain
k	Response gain
k	Current time step
K_c	Proportionl controller gain
n_u	Number of inputs
n_y	Number of outputs
p	Prediction index
p_h	Prediction horizon in number of steps p
Q	Weight matrix on states
R	Weight matrix on inputs
r_i	Reference output i
r_j	Reference input j
s_i	Scaling factor of output i
s_j	Scaling factor of input j
t	Time index
T_s	Time step length
u	System input (MV)
Δu	Change in input
u_j	Input number j
$\mathbf{u}$	Vector of inputs
w_i^p	Weight on output i at prediction step p
w_j^p	Weight on input j at prediction step p
$\mathbf{x}$	Vector of states
$\dot{\mathbf{x}}$	Vector of time derivative states
x_t	state at time t
y	System output (CV)
$\mathbf{y}$	Vector of outputs
Δy	Change in output
y_i	Output number i

A Identified models

The matrices describing the state space model on the form given in (2) and (3) are given in equation (28), (29), (30) and (31). The combined transfer function system is given in equation (32).

$$A = \begin{bmatrix} \text{-0.009091} & 0 & 0 & 0 \\ 0 & \text{-6.426} & 0 & 0 \\ 0 & 0 & \text{-0.06993} & 0 \\ 0 & 0 & 0 & \text{-17.95} \end{bmatrix} \tag{28}$$

$$B = \begin{bmatrix} 0.5 & 0 & 0 \\ 16 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 16 \end{bmatrix} \tag{29}$$

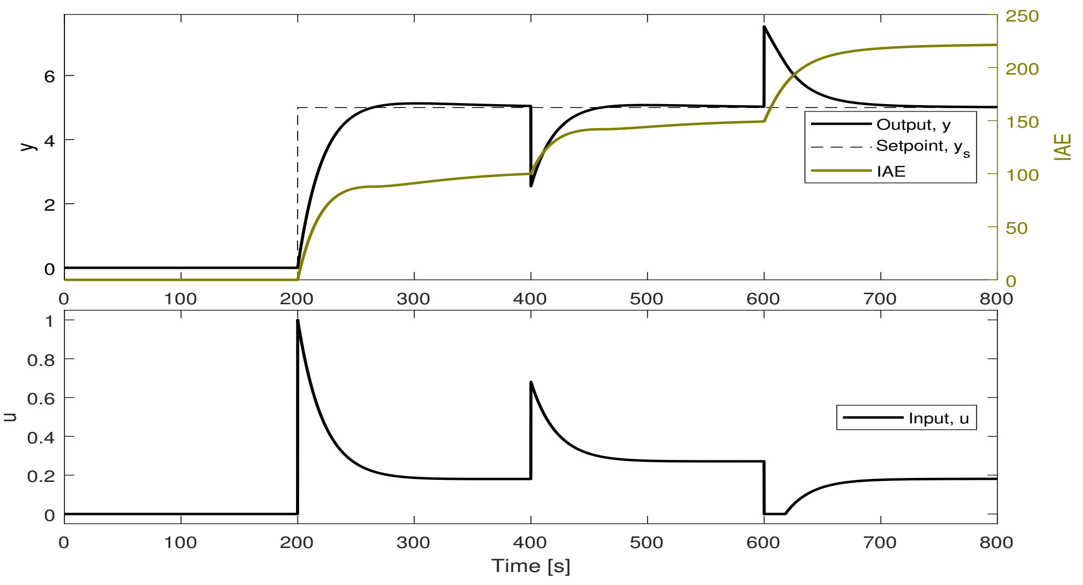
$$C = \begin{bmatrix} 0.5015 & 0 & \text{-1.108} & 0 \\ 0 & 19.64 & 0 & 10.01 \end{bmatrix} \tag{30}$$

$$D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{31}$$

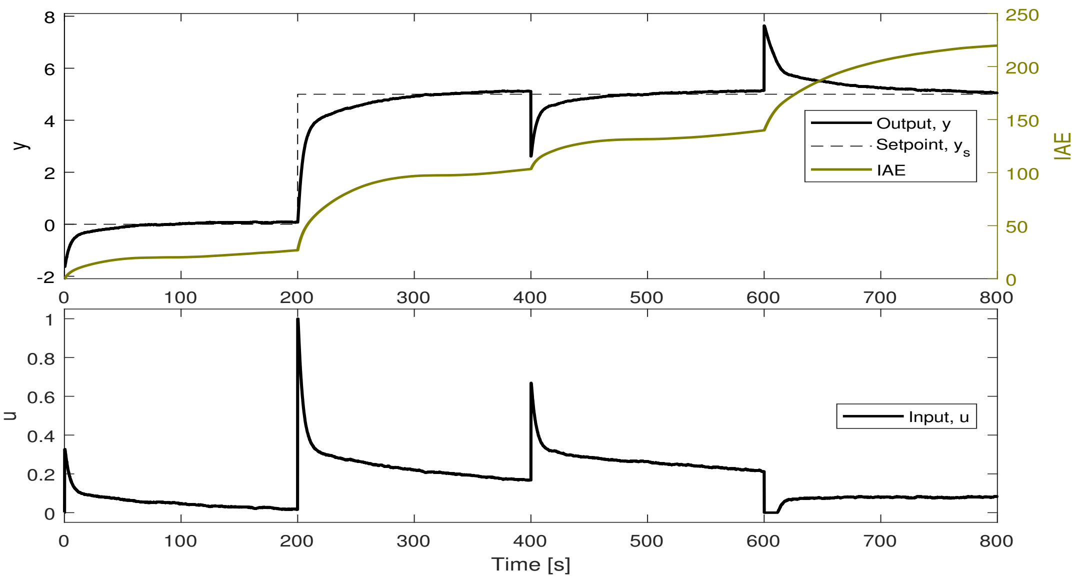
$$\begin{array}{c}\text{MIMO tf}\\\text{system}\end{array} = \begin{bmatrix} \frac{27.58}{110s+1} & \frac{-31.68}{14.3s+1} & 0 \\ \frac{48.9019}{0.1556s+1} & 0 & \frac{27.58}{0.0557s+1} \end{bmatrix} \tag{32}$$

B SISO control figures

This section contains the SISO control tests on the four possible pairings in the system.

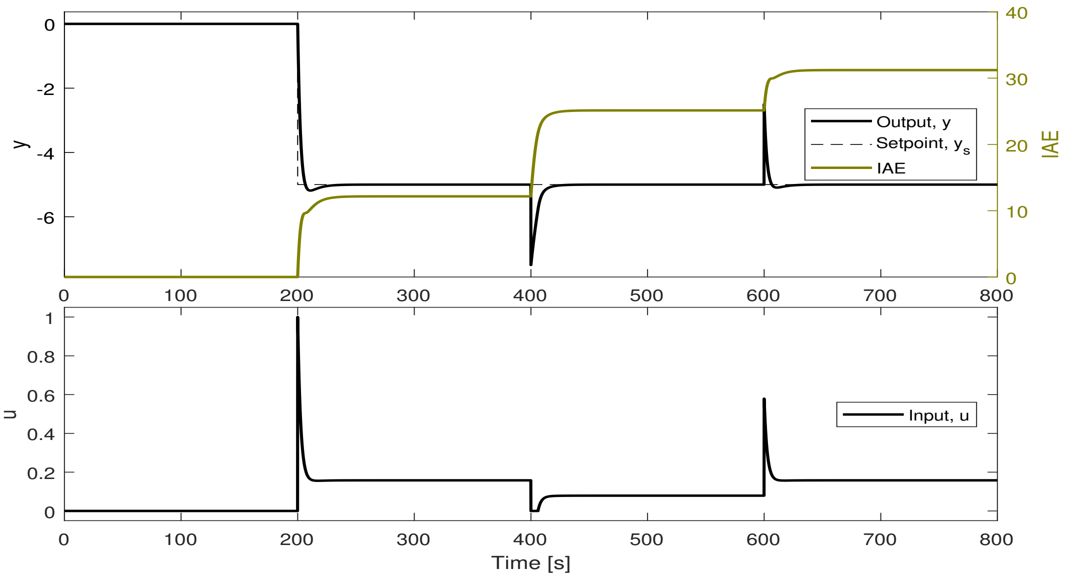


(a) Simulated

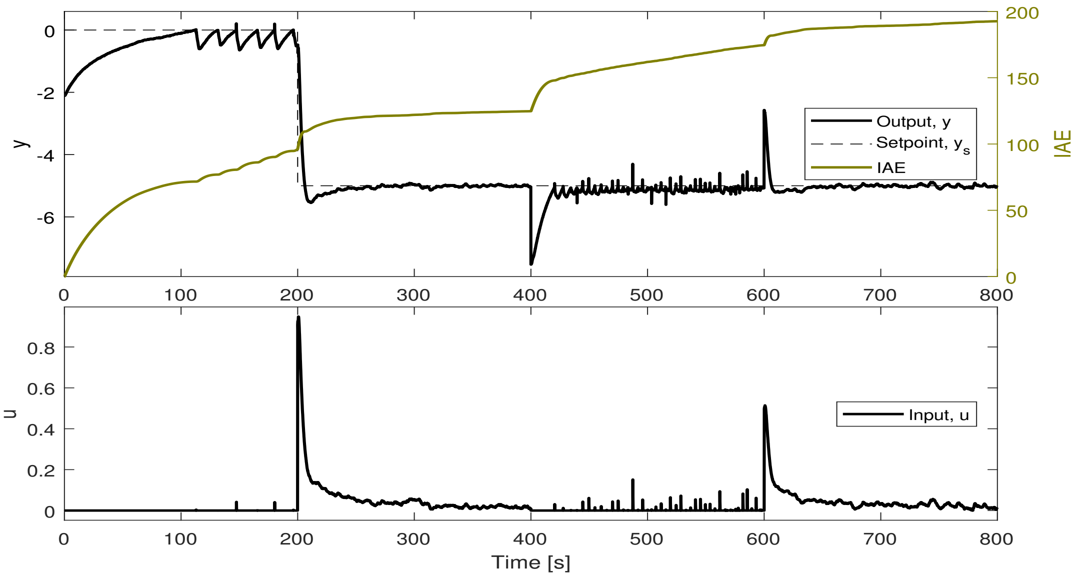


(b) Real-Time

Figure 14: Bulb (u) to Temperature (y) control. Figures shows both the control on the estimated model and on the real-time plant. Slow and robust tuning was chosen, avoiding saturation at 5° steps in Temperature. $\tau_c = 19.94$. The SISO control had better control on the real-time plant than on the model. Integral Absolute Error (IAE) is shown for the output.

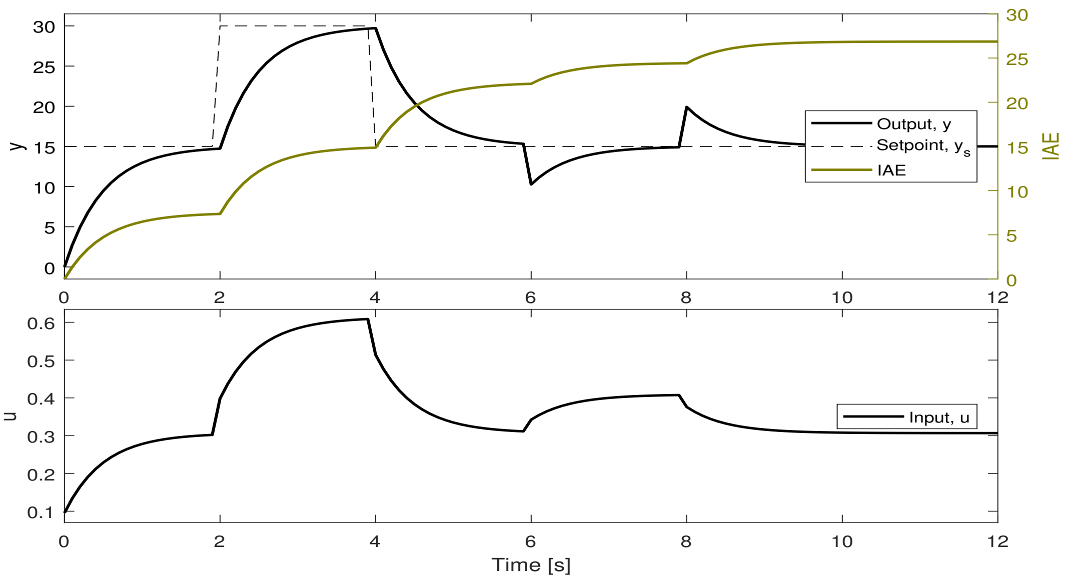


(a) Simulated

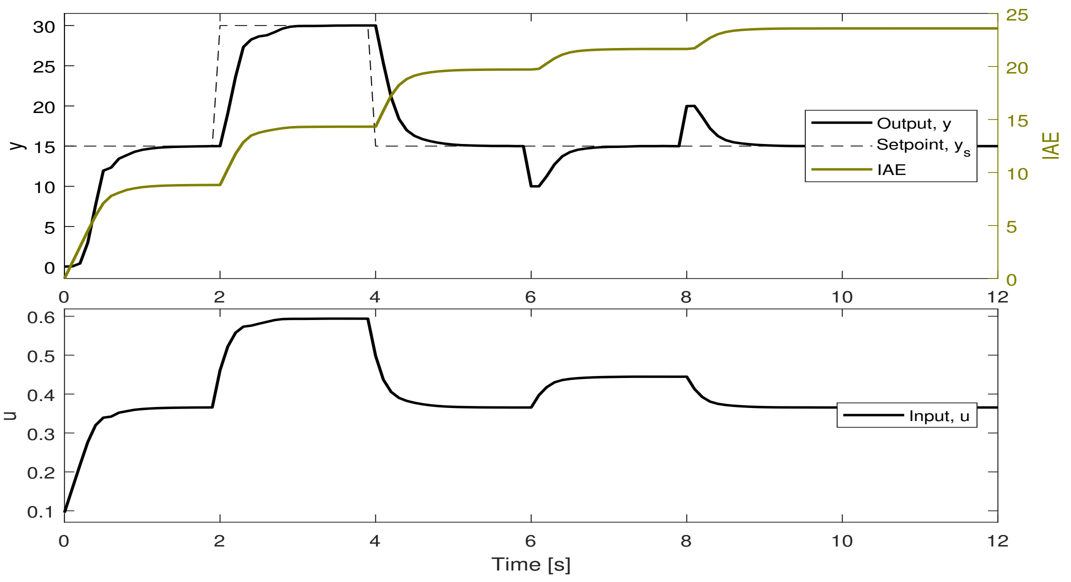


(b) Real-Time

Figure 15: Fan (u) to Temperature (y) control. Figures shows both the control on the estimated model and on the real-time plant. Slow and robust tuning was chosen, avoiding saturation at 5° steps in Temperature. $\tau_c = 2.257$. The real-time control shows more error because of the bad Fan control at low input. Integral Absolute Error (IAE) is shown for the output.

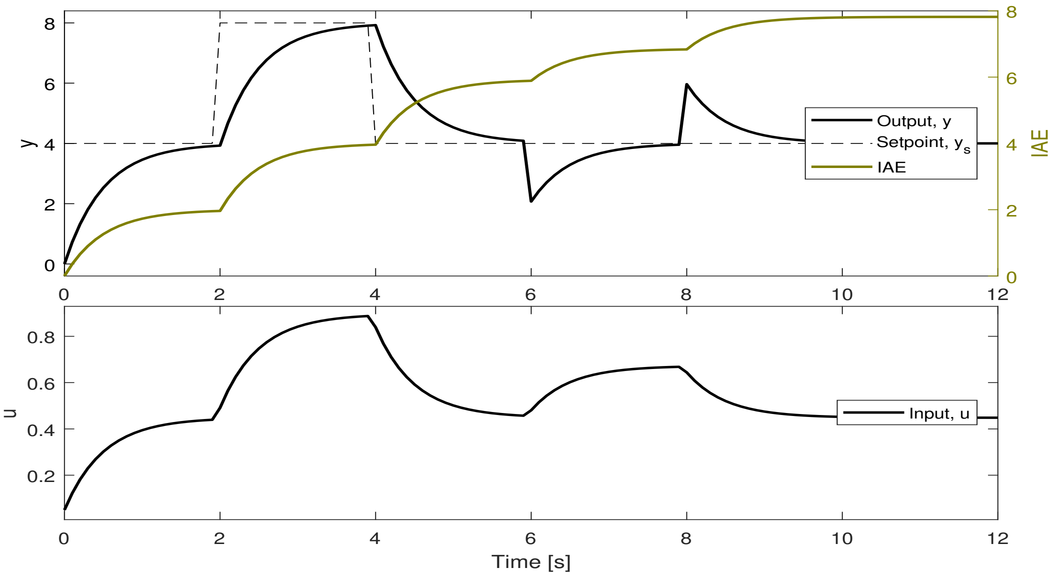


(a) Simulated

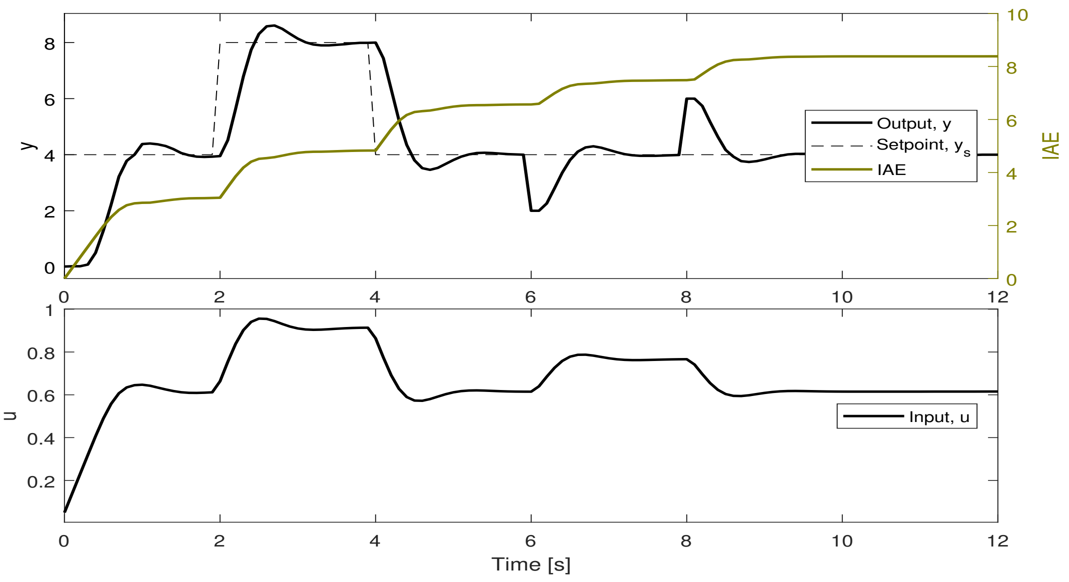


(b) Real-Time

Figure 16: Bulb (u) to Light Intensity (y) control. Figures shows both the control on the estimated model and on the real-time plant. $\tau_c = 0.5$ to avoid unstable oscillation in the real-time system. Integral Absolute Error (IAE) is shown for the output.



(a) Simulated



(b) Real-Time

Figure 17: LED (u) to Light Intensity (y) control. Figures shows both the control on the estimated model and on the real-time plant. $\tau_c = 0.5$ to avoid unstable oscillation in the real-time system. Integral Absolute Error (IAE) is also shown for the output.

C Simulink diagrams

This section contains the main Simulink diagrams used.

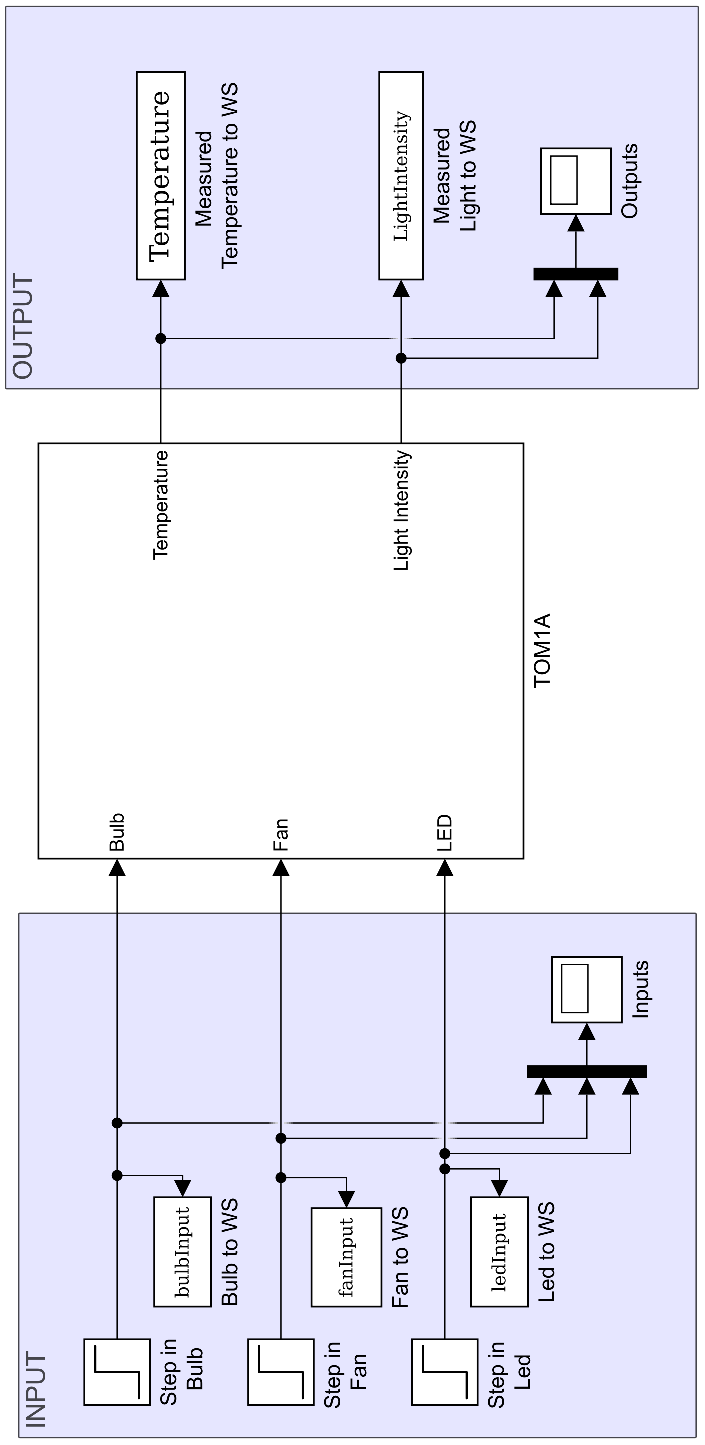


Figure 18: Simulink diagram of system. The diagram is used to perform step responses and measure standby temperature used when shifting Temperature. TOM1A is the real-time system.

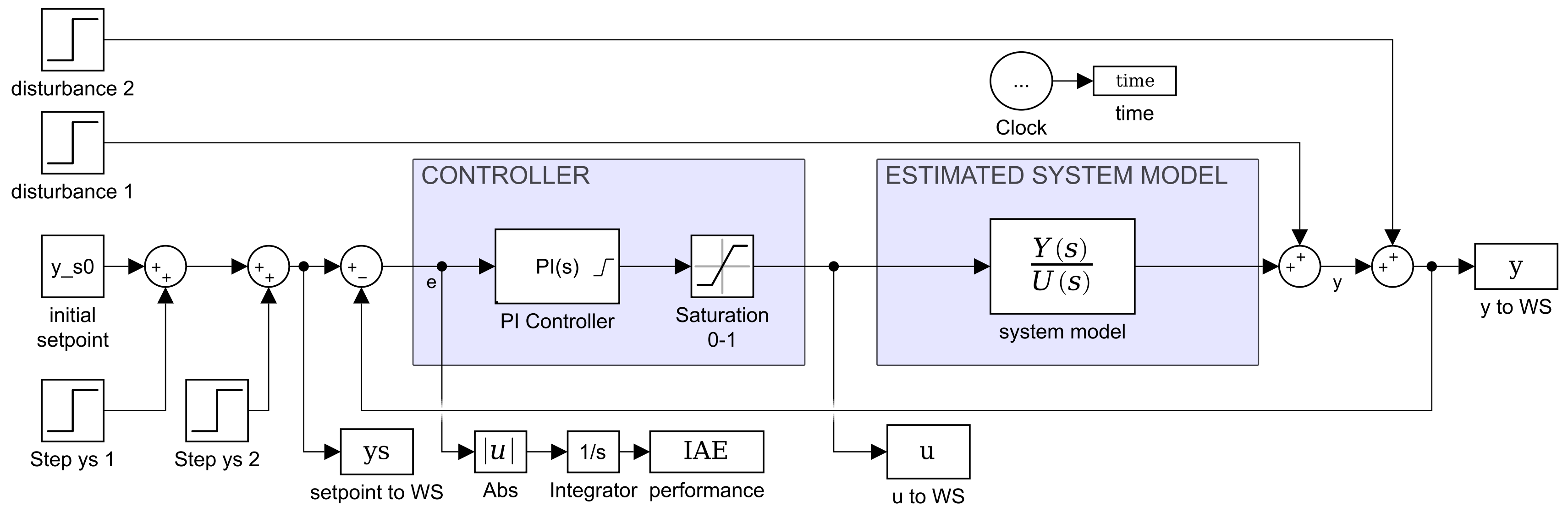


Figure 19: Simulink diagram of simulated SISO control.

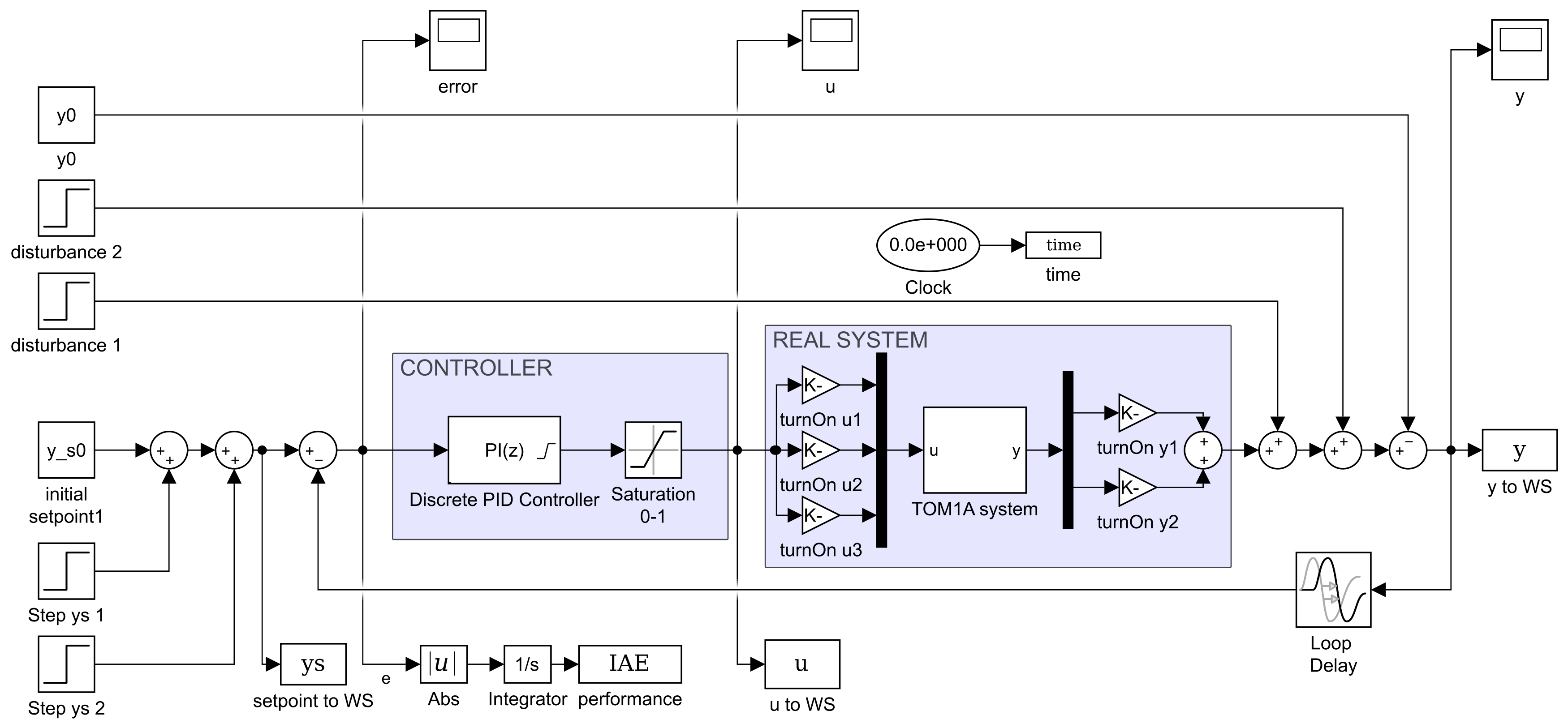


Figure 20: Simulink diagram of real-time SISO control. The real-time system needs a loop delay to avoid trying to calculate the whole system.

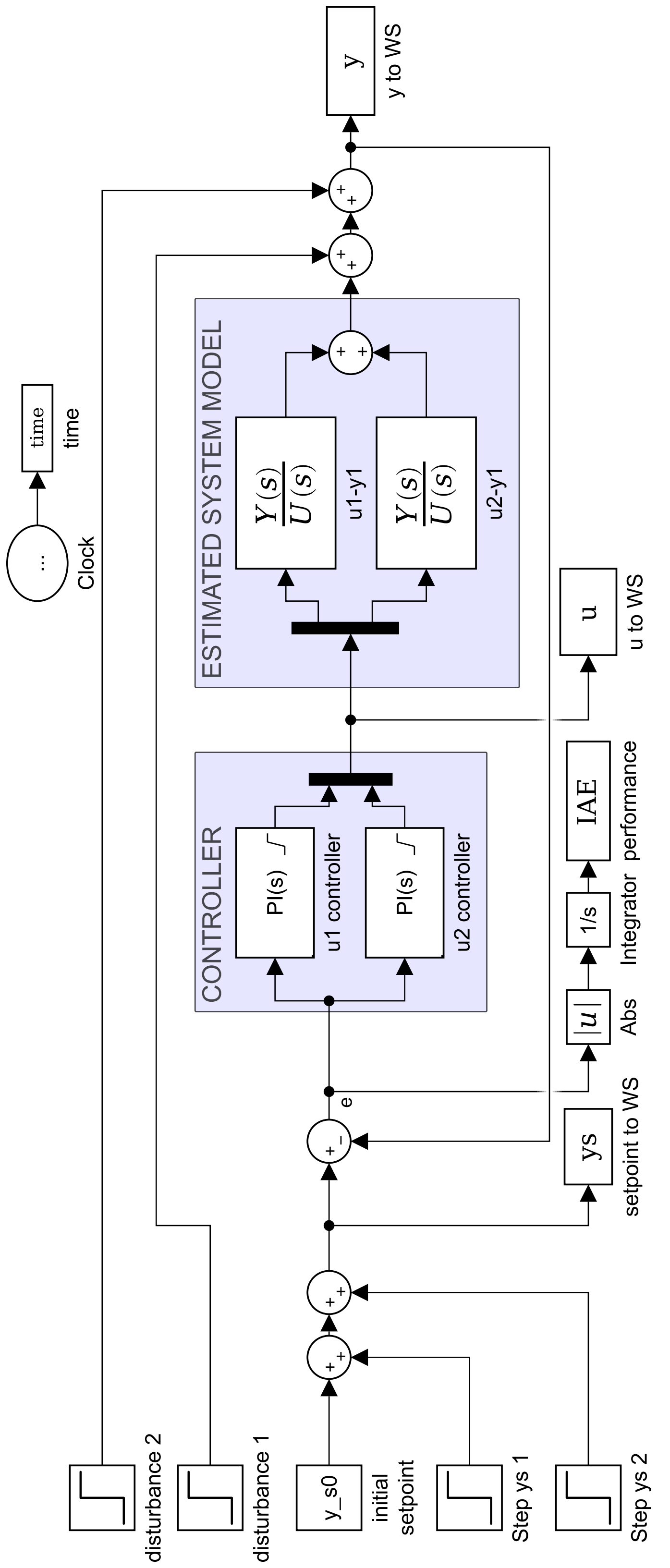


Figure 21: Simulink diagram of simulated MISO control.

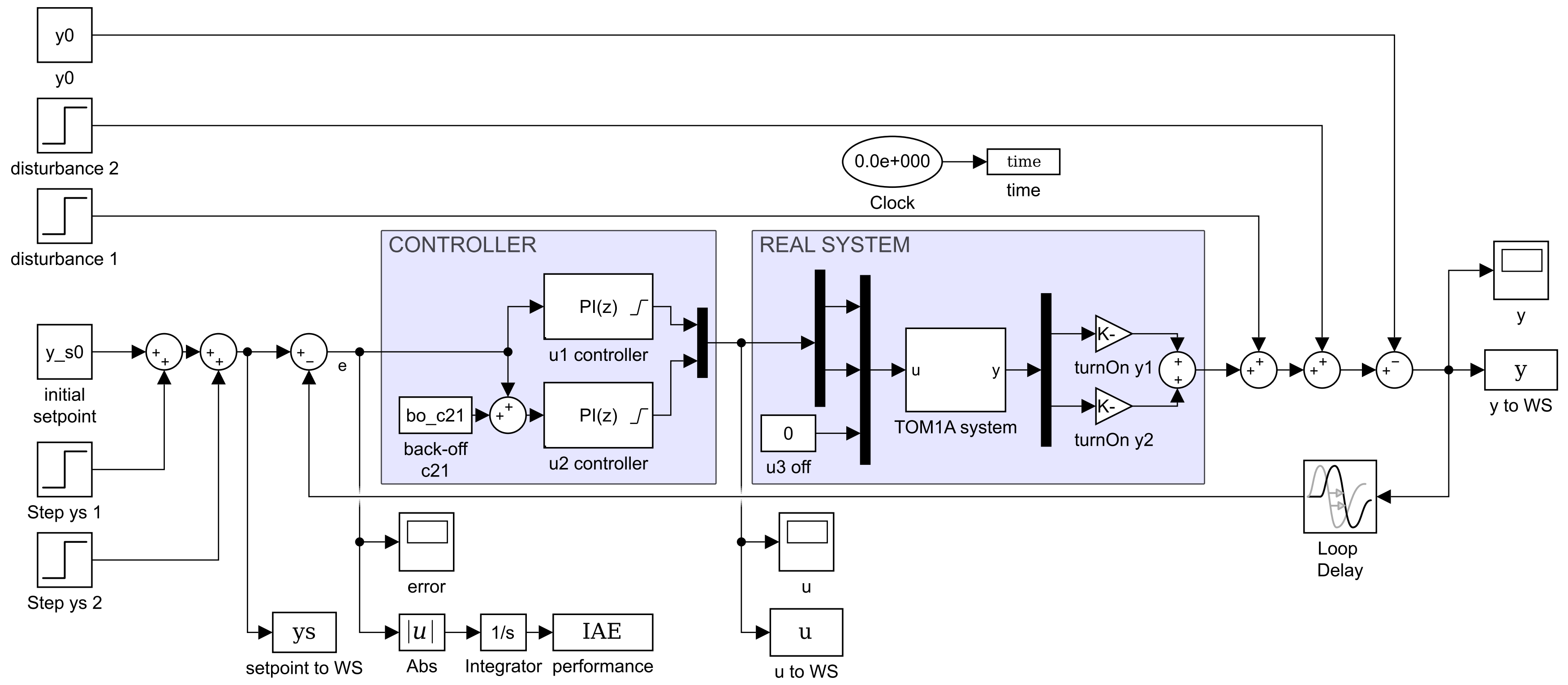


Figure 22: Simulink diagram of real-time MISO control. Bulb and Fan is used to control the Temperature. The Fan has a implemented setpoint back-off. The real-time system needs a loop delay to avoid trying to calculate the whole system.

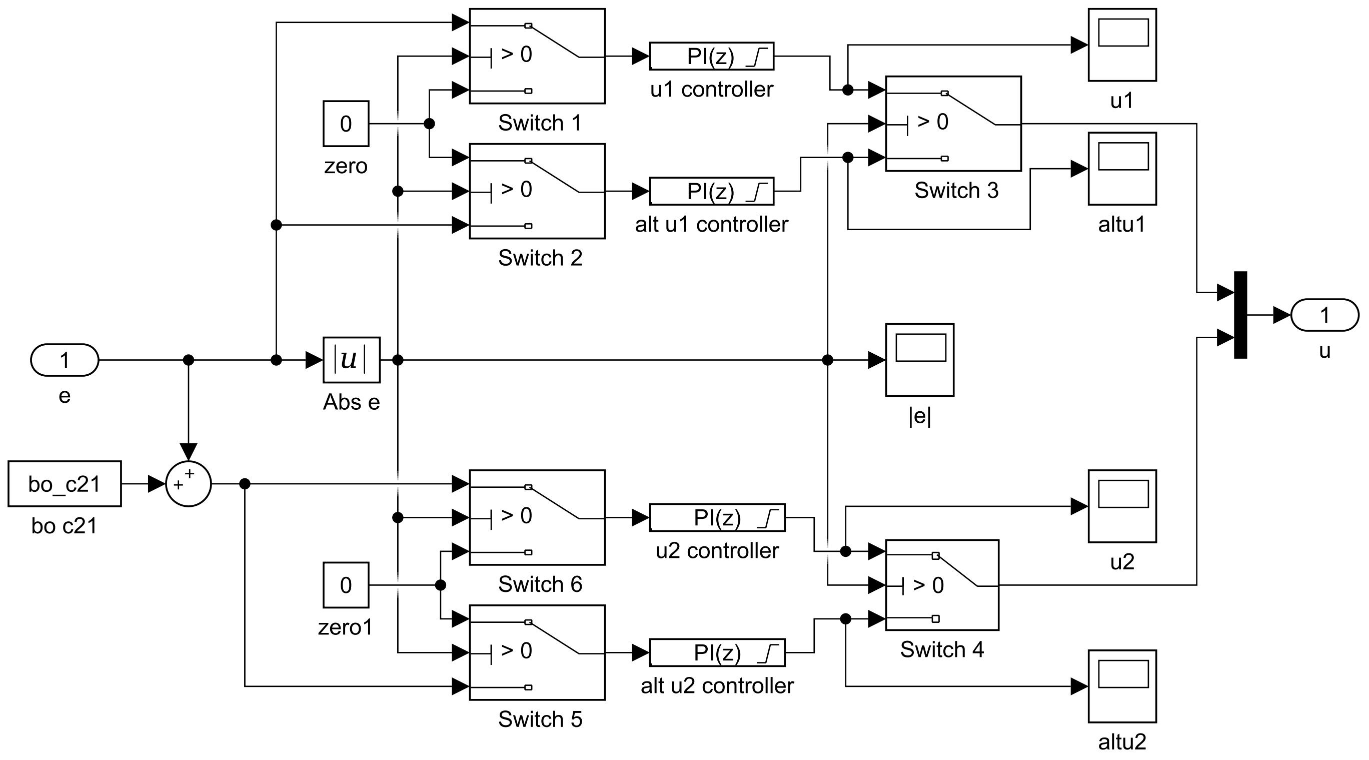


Figure 23: Control block in MISO Temperature control with alternative smooth control near setpoint. The block replaced is the CONTROLLER part in Figure 22.

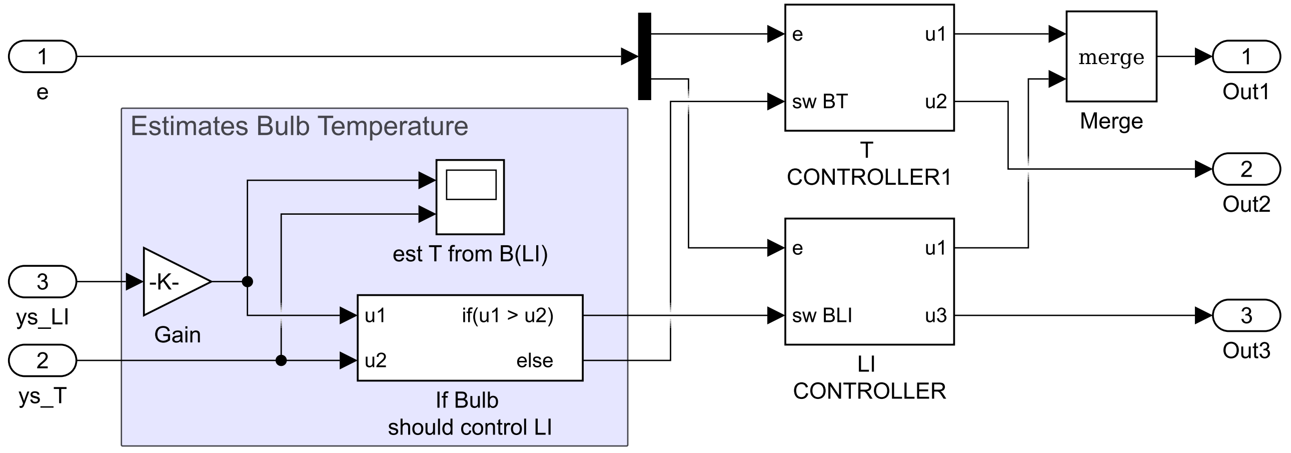


Figure 24: Simulink diagram of control block in MIMO structure. The controller uses the setpoints and the steady state gains to estimate temperature from B-LI control and uses it to switch control structure. The T and LI controller blocks has F-T and LED-LI controllers and an optional B-T and B-LI controller that turns on and off according to the estimated temperature.

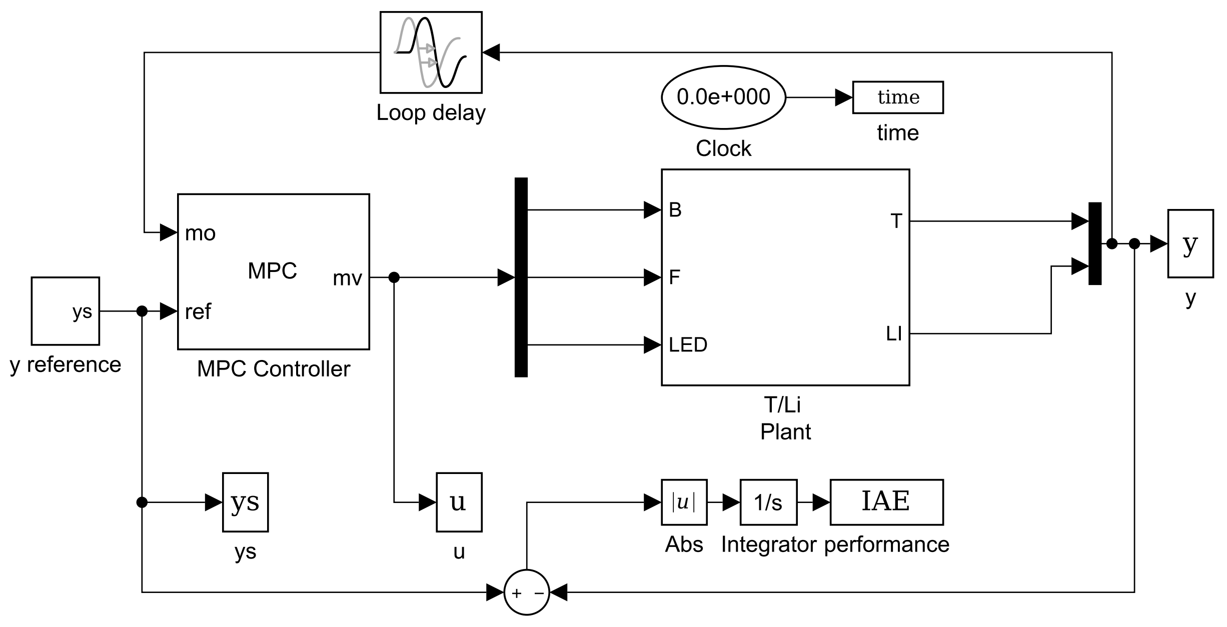


Figure 25: Simulink diagram of MPC implemented MIMO control.The real-time system needs a loop delay to avoid trying to calculate the whole system.

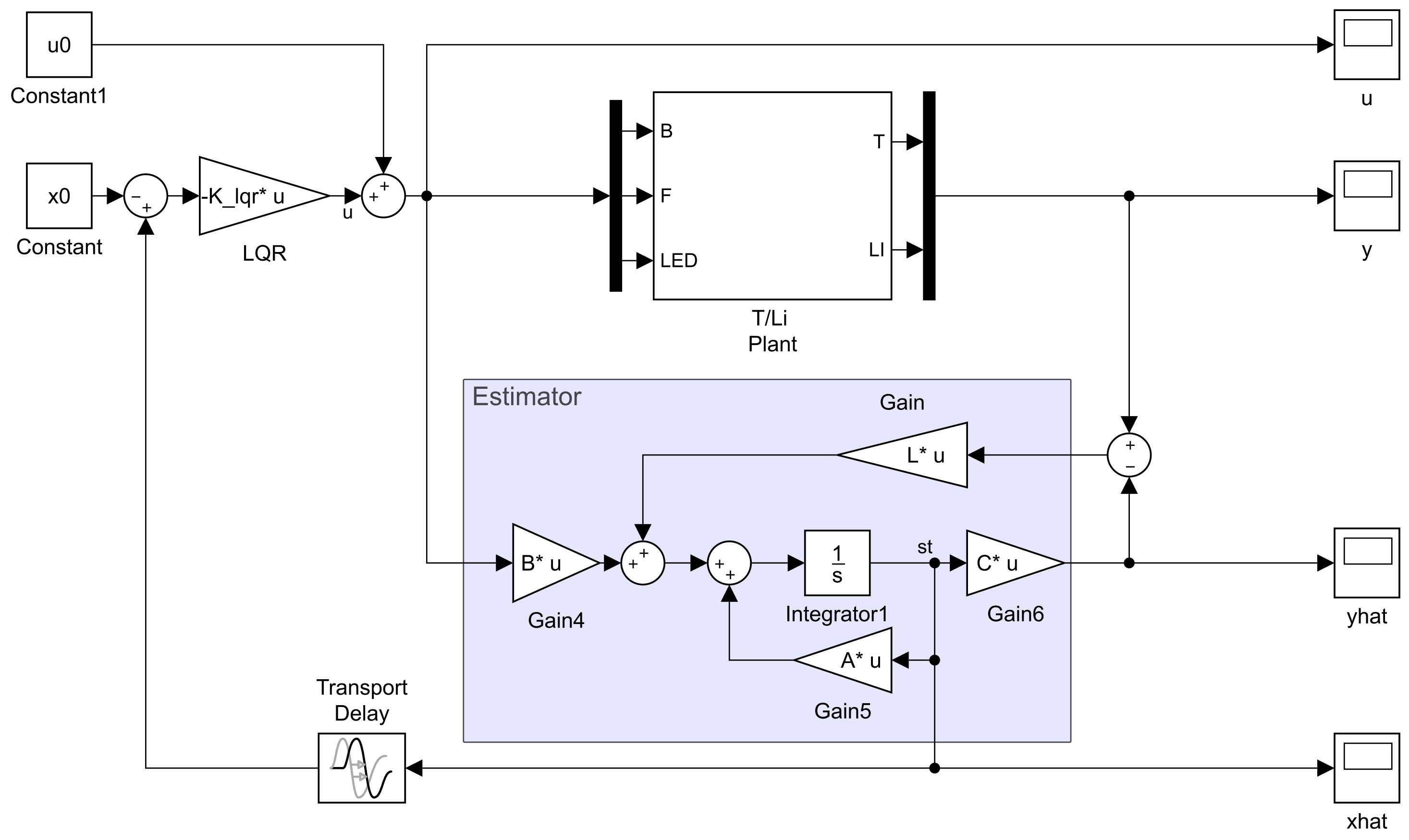


Figure 26: Simulink diagram of unfinished LQG structure.

D Scripts

This section contains the main scripts used.

D.1 identification.m

```
% Identifies transfer function models, state space models, Gains and RGA
% The data (WS_Bulb.mat, WS_Fan.mat and WS_Led.mat) of the responses
% was collected in runSteptest.m.
% The script uses the System Identification Toolbox to estimate the
% transfer functions. All non-zero responses was estimated as first order
% responses. There was no significant response in Temperature from LED and
% no response in Light Intensity from the Fan. All delays was neglected.
%
% See also RUNSTEPTEST.m TFEST SS

% Author: Sindre Johan Heggheim, sindrjh@stud.ntnu.no, November 20th, 2017
% Specialization project TKP4580 at Department of Chemical Engineering

close all
clear
clc

% figure settings
figProps.OuterPosition = [100 100 1000 500];
figProps.Color = [1 1 1];

%% Loading responses:
load('WS_Bulb')
load('WS_Fan')
load('WS_Led')

Ts = 0.1;
stepIndx_B = 100/Ts;  %100 sec
stepIndx_F = 10/Ts;   %10 sec
stepIndx_L = 10/Ts;  %10 sec
data_BT  = dataBulb(stepIndx_B:end,'Temperature','Bulb');
data_BLI = dataBulb(stepIndx_B:end,'Light','Bulb');
data_FT  = dataFan(stepIndx_F:end,'Temperature','Fan');
data_LLI = dataLed(stepIndx_L:end,'Light','Led');
timeBulb = timeBulb(stepIndx_B:end);
timeFan  = timeFan(stepIndx_F:end);
timeLed  = timeLed(stepIndx_L:end);

%% Bulb to Temperature

% Plot the measured and estimated response:
fig = figure();
set(fig,figProps);
subplot(1,2,1)
plot(timeBulb,data_BT.outputData);
xlabel('Time')
```

```
ylabel('Measured Temperature')

% estimated 1st order tf:
tf_BT = tfest(data_BT,1,0);
[T_est,t] = step(tf_BT,timeBulb(end)-timeBulb(1));
subplot(1,2,2)
plot(t,T_est)

% The esimated transfer function has to high gain. The transfer function
% is instead calculated with gain delta_y / delta_u and time constant
% at 63% of the step
u_init = data_BT.inputData(1);
u_end = data_BT.inputData(end);
y_init = data_BT.outputData(1);
y_end = data_BT.outputData(end);
gain_BT = (y_end-y_init)/(u_end-u_init);
tC_idx=1;
while data_BT.outputData(tC_idx)<y_init+(y_end-y_init)*0.63
    tC_idx=tC_idx+1;
end%while
tConst_BT = timeBulb(tC_idx)-timeBulb(1);
tf_BT_calc = tf([gain_BT],[tConst_BT 1]);

hold on
[T_est,t] = step(tf_BT_calc,timeBulb(end)-timeBulb(1));
plot(t,T_est)
xlabel('Time')
ylabel('Temperature')
legend('Identified tf','Calculated tf','Location','southeast')

% save figure
figIm = getframe(gcf);
imwrite (figIm.cdata, 'BT_tf.png', 'png');
hgexport(fig,'BT_tf.eps')

%% Bulb to Light Intensity
% plot measured and estimated response:
fig = figure();
set(fig,figProps);
subplot(1,2,1)
plot(timeBulb(2:12),data_BLI.outputData(2:12));
xlabel('Time')
ylabel('Measured Light Intensity')

% estimated 1st order tf
subplot(1,2,2)
tf_BLI = tfest(data_BLI,1,0);
[LI_est,t]=step(tf_BLI,1); % The esimated transfer function is a good fit.
plot(t,LI_est)
xlabel('Time')
ylabel('Light Intensity')

% save figure
figIm = getframe(gcf);
imwrite (figIm.cdata, 'BLI_tf.png', 'png');
```

```
hgexport(fig,'BLI_tf.eps')

%% Fan to Temperature
% plot measured and estimated response:
fig = figure();
set(fig,figProps);
subplot(1,2,1)
plot(timeFan,data_FT.outputData);
xlabel('Time')
ylabel('Measured Temperature')

% estimated 1st order tf
tf_FT = -tfest(data_FT,1,0); % negative response
[T_est,t]=step(tf_FT,[0:Ts:timeFan(end)]);
subplot(1,2,2)
plot(t,T_est)

% The esimated transfer function has to high gain. The transfer function
% is instead calculated with gain delta_y / delta_u and time constant
% at 63% of the step
hold on
u_init = data_FT.inputData(1);
u_end = data_FT.inputData(end);
y_init = data_FT.outputData(1);
y_end = data_FT.outputData(end);
gain_FT = (y_end-y_init)/(u_end-u_init);

tau1_idx=1;
while data_FT.outputData(tau1_idx)>y_init+(y_end-y_init)*0.63
    tau1_idx=tau1_idx+1;
end%while
tau1_FT = timeFan(tau1_idx)-timeFan(1);
tf_FT_calc = tf([gain_FT],[tau1_FT 1]);
hold on
[T_est,t]=step(tf_FT_calc,[0:Ts:timeFan(end)]);
plot(t,T_est)
legend('tf from ident.','tf calculated','Location','southeast')
xlabel('Time')
ylabel('Temperature')

% save figure
figIm = getframe(gcf);
imwrite (figIm.cdata, 'FT_tf.png', 'png');
hgexport(fig,'FT_tf.eps')

%% Led to Light
% plot measured and estimated response:
fig = figure();
set(fig,figProps);
subplot(1,2,1)
plot(timeLed(2:12),data_LLI.outputData(2:12));
xlabel('Time')
ylabel('Measured Light Intensity')

% estimated 1st order tf
```

```matlab
subplot(1,2,2)
tf_LLI = tfest(data_LLI,1,0);
[LI_est,t]=step(tf_LLI,1); % The estimated transfer function is a good fit.
plot(t,LI_est);
ylabel('Light Intensity')
xlabel('Time')
ylim([0,10])

% save figure
figIm = getframe(gcf);
imwrite (figIm.cdata, 'LLI_tf.png', 'png');
hgexport(fig,'LLI_tf.eps')

%% Save transfer functions to .mat file
tf_BT = tf_BT_calc;
tf_BT.InputName = 'Bulb';
tf_BT.OutputName = 'Temperature';
tf_BLI = 1*tf_BLI; % makes idtf to tf
tf_BLI.InputName = 'Bulb';
tf_BLI.OutputName = 'Light';
tf_FT = tf_FT_calc;
tf_FT.InputName = 'Fan';
tf_FT.OutputName = 'Temperature';
tf_LLI = 1*tf_LLI; % makes idtf to tf
tf_LLI.InputName = 'LED';
tf_LLI.OutputName = 'Light';
save('WS_tfs.mat','tf_BT','tf_BLI','tf_FT','tf_LLI')

%% Save MIMO-models to .mat file
MIMO_tf = [tf_BT, tf_FT, 0; tf_BLI, 0, tf_LLI ];
MIMO_css = ss(MIMO_tf);
save('WS_MIMO_models.mat','MIMO_tf','MIMO_css');

%% first order response parameters:
close all
clear
clc
load('WS_tfs')

numden = [cell2mat(tf_BT.num);cell2mat(tf_BT.den)]; %Bulb Temperature
BT_gain = numden(1,2)/numden(2,2);
BT_tau1 = numden(2,1)/numden(2,2);

numden = [cell2mat(tf_FT.num);cell2mat(tf_FT.den)]; % Fan Temperature
FT_gain = numden(1,2)/numden(2,2);
FT_tau1 = numden(2,1)/numden(2,2);

numden = [cell2mat(tf_BLI.num);cell2mat(tf_BLI.den)];% Bulb Light Intensity
BLI_gain = numden(1,2)/numden(2,2);
BLI_tau1 = numden(2,1)/numden(2,2);

numden = [cell2mat(tf_LLI.num);cell2mat(tf_LLI.den)]; % LED Light Intensity
LLI_gain = numden(1,2)/numden(2,2);
LLI_tau1 = numden(2,1)/numden(2,2);
```

```
tfParam  = [  BT_gain,  BT_tau1;...
                FT_gain,  FT_tau1;...
                BLI_gain, BLI_tau1;...
                LLI_gain, LLI_tau1];

%% Steady state gains and RGA:
close all
clear all
clc
load('WS_tfs')

G = zeros(2,3);
s=0;

num = cell2mat(tf_BT.num);
den = cell2mat(tf_BT.den);
G(1,1)  = (num(1)*s+num(2))/(den(1)*s+den(2)); % tf_BT

num = cell2mat(tf_FT.num);
den = cell2mat(tf_FT.den);
G(1,2)  = (num(1)*s+num(2))/(den(1)*s+den(2)); % tf_FT

G(1,3)  = 0;

num = cell2mat(tf_BLI.num);
den = cell2mat(tf_BLI.den);
G(2,1)  = (num(1)*s+num(2))/(den(1)*s+den(2)); % tf_BLI

G(2,2)  = 0;
num = cell2mat(tf_LLI.num);
den = cell2mat(tf_LLI.den);
G(2,3)  = (num(1)*s+num(2))/(den(1)*s+den(2)); % tf_LLI

RGA_Matrix =G.*pinv(G.');
save('WS_RGA.mat','RGA_Matrix','G')
```

D.2 runSteptest.m

```
% Initializes system and performs step respononses and save measured data.
% The three datasets is saved as the following .mat files:
% WS_Bulb.mat, WS_Fan.mat and WS_Led.mat

% Author: Sindre Johan Heggheim, sindrjh@stud.ntnu.no, November 20th, 2017

% The script uses runSystem.slx to run the MIMO system that is connected
% with the port spesified as teh 'com' variable.
% tom1ar8_lib.mdl, tom_lib.mdl and msfun_realtime_pacer.m is also needed
% to run the system.

%% Initialization of system

clc
clear
com = 'COM4';    % com port
```

```
delete(instrfind({'Port'},{com}));
baud = 250000;  % baud rate
Ts = 0.1;       % Sampling time period
fTt = 0.05;     % Filter time constant for temperature (0.05s - 10s)
fTl = 0.05;     % Filter time constant for light intensity (0.05s - 10s)
fTf = 0.1;      % Filter time constant for light intensity (0.1s - 10s)

t_sim = 1400;   % Simulation time
t_step = 100;   % Step time in simulation
stepSize = 1;   % step change in input in simulation

%% Response from Bulb
stepBulb = stepSize;
stepFan = 0;
stepLed = 0;

open('runSystem');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(3) %Waiting for simulink to save to WS
y = [Temperature,LightIntensity];            % Output, y
u = bulbInput;                               % Input, u
dataBulb = iddata(y,u,Ts);
dataBulb.InputName  = {'Bulb'};
dataBulb.OutputName = {'Temperature';'Light'};
timeBulb = time;
save('WS_Bulb.mat','dataBulb','timeBulb');

%% Response from Fan
t_sim = 700;    % Simulation time
t_step = 10;    % Step time
stepBulb = 0;
stepFan = stepSize;
stepLed = 0;

open('runSystem');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(3) %Waiting for simulink to save to WS
y = [Temperature,LightIntensity];            % Output, y
u = fanInput;                                % Input, u
dataFan = iddata(y,u,Ts);
dataFan.InputName  = {'Fan'};
dataFan.OutputName = {'Temperature';'Light'};
timeFan = time;
save('WS_Fan.mat','dataFan','timeFan');

%% Response from LED
t_sim = 50;    % Simulation time
```

```
t_step = 10;     % Step time
stepBulb = 0;
stepFan = 0;
stepLed = stepSize;

open('runSystem');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(3) %Waiting for simulink to save to WS
y = [Temperature,LightIntensity];            % Output, y
u = ledInput;                                % Input, u
dataLed = iddata(y,u,Ts);
dataLed.InputName  = {'Led'};
dataLed.OutputName = {'Temperature';'Light'};
timeLed = time;
save('WS_Led.mat','dataLed','timeLed');
```

D.3 SISOcontrol.m

```
% Tuning, simulation and testing of SISO PI controllers in MIMO system
% Temperature can be controlled by the Bulb and Fan and the Light Intensity
% can be controlled by the Bulb and LED. All delays (theta) are neglected
% and the derivitive part is not needed.
%
% The PI controllers PI: c(s) = K_c*( 1 + 1/(tau_I*s)) are tuned with
% SIMC PI tuning rules. The transfer functions used in the tuning was
% calculated in identification.m and saved as WS_tfs.mat
% The controllers in simulink uses Clamping as Anti-windup method.
% The script simulate the controllers in simSISOcontrol.slx and test it in
% realtime in runSISOcontrol.slx
%
% See also SIMCPI SIMCPI_nosat IDENTIFICATION

% Author: Sindre Johan Heggheim, sindrjh@stud.ntnu.no, November 20th, 2017
% Specialization project TKP4580 at Department of Chemical Engineering

%% Initialization

clear
close all
clc

% The script uses runSystem.slx to run the MIMO system that is connected
% with the port spesified as teh 'com' variable.
% tom1ar8_lib.mdl, tom_lib.mdl and msfun_realtime_pacer.m is also needed
% to run the system.
com = 'COM4';    % com port
delete(instrfind({'Port'},{com}));
baud = 250000;  % baud rate
Ts = 0.1;       % Sampling time period
fTt = 0.05;     % Filter time constant for temperature (0.05s - 10s)
```

```
fTl = 0.05;     % Filter time constant for light intensity (0.05s - 10s)
fTf = 0.1;      % Filter time constant for light intensity (0.1s - 10s)
t_sim = 400;    % Simulation time

load('WS_tfs')  % estimated transfer functions

%% Using Bulb as Temperature controller.
% Temperature is controlled by a SISO controller on the Bulb and the
% Light Intensity is not taken into any consideration.

% System transfer function:
sys_tf = tf_BT;
sys_num = cell2mat(sys_tf.num);
sys_den = cell2mat(sys_tf.den);

% Calculate smooth SIMC PI control parameters with so saturation on a 5
% degree step.
ys_max = 5;                         % max step in y without saturation.
[Kc,tauI,tauc] = SIMCPI_nosat(sys_tf,ys_max);

% Test controller on estimated system
y0 = 0;             % Shifting of y
y_s0 = y0;          % initial set-point
t_sim = 800;        % Simulation time
stepTime1 = 200;    % Time when first step
stepTime2 = 0;      % No step
step_ys1 = 5;       % Step in setpoint
step_ys2 = -0;      % No step
stepTime_d1 = 400;  % Time disturbance 1
stepTime_d2 = 600;  % Time disturbance 2
step_d1 = -2.5;     % Value of disturbance 1
step_d2 = +2.5;     % Value of disturbance 2

% Run simulation:
open('simSISOcontrol');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotSISOcontrol(y,u,ys,IAE,time);
figIm = getframe(gcf);
imwrite (figIm.cdata, 'SISOsimBT.png', 'png');
hgexport(fig,'SISOsimBT.eps')

%% Test Bulb-Temperature controller on real system:
% measure and shift with standby temperature
    t_step = 0; % no step
    t_sim  = 2;
    stepBulb = 0;
    stepFan = 0;
    stepLed = 0;
    open('runSystem');
    set_param(gcs,'StopTime',num2str(t_sim));
    set_param(gcs,'SimulationCommand','start');
    while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
        pause(1)
    end%while
    pause(1) %Waiting for simulink to save to WS
sb_T = Temperature(1,1); %standby temperature
sb_T = 27;    % most measured standby temperature
close_system('runSystem');

% Response and rejection:
y0 = sb_T;           % shift temperature
t_sim = 800;         % Simulation time

% Choose IO and fixed inputs in simulink system model
turnOn_u1 = 1; % Bulb in
turnOn_u2 = 0;
turnOn_u3 = 0;
turnOn_y1 = 1; % Temp out
turnOn_y2 = 0;
fixed_u1  = 0;
fixed_u2  = 0;
fixed_u3  = 0;

% Test controller:
open('runSISOcontrol');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotSISOcontrol(y,u,ys,IAE,time);
% figIm = getframe(gcf);
% imwrite (figIm.cdata, 'SISOtestBT.png', 'png');
% hgexport(fig,'SISOtestBT.eps')

%% Using Fan as temperature controller.
% Temperature is controlled by a SISO controller on the Fan and the
% Light Intensity is not taken into any consideration.

% System transfer function:
sys_tf = tf_FT;
sys_num = cell2mat(sys_tf.num);
sys_den = cell2mat(sys_tf.den);

% Calculate smooth SIMC PI control parameters with so saturation on a 5
% degree step.
% [Kc,tauI] = SIMCPI(sys_tf,0);
[Kc,tauI,tauc] = SIMCPI_nosat(sys_tf,5);

% Test controller on estimated system
```

```
y0 = 0;                 % Shifting of y
y_s0 = 0;               % initial set-point
t_sim = 800;            % Simulation time
stepTime1 = 0;          % no step
stepTime2 = 200;        % Time when first step
step_ys1 = 0;           % No step
step_ys2 = -5;          % Step in setpoint
stepTime_d1 = 400;      % Time disturbance 1
stepTime_d2 = 600;      % Time disturbance 2
step_d1 = -2.5;         % Value of disturbance 1
step_d2 = +2.5;         % Value of disturbance 2

% Run simulation:
open('simSISOcontrol');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotSISOcontrol(y,u,ys,IAE,time);
figIm = getframe(gcf);
imwrite (figIm.cdata, 'SISOsimFT.png', 'png');
hgexport(fig,'SISOsimFT.eps')

%% Test Fan-Temperature controller on real system:
% measure and shift with standby temperature
    t_step = 10; % nostep
    t_sim  = 2;
    stepBulb = 0;
    stepFan = 0;
    stepLed = 0;
    open('runSystem');
    set_param(gcs,'StopTime',num2str(t_sim));
    set_param(gcs,'SimulationCommand','start');
    while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
        pause(1)
    end%while
    pause(1) %Waiting for simulink to save to WS
sb_T = Temperature(1,1); %standby temperature
sb_T = 27;
close_system('runSystem');

y0 = sb_T + 10; % Need a higher temperature to test Fan on
t_sim = 800;

fixed_u1 = 0.25;% Needed to get higher T to test Fan on
fixed_u2 = 0;
fixed_u3 = 0;

% Choose IO:
turnOn_u1 = 0;
turnOn_u2 = 1;   % Fan
```

```
turnOn_u3 = 0;
turnOn_y1 = 1;  % Temperature
turnOn_y2 = 0;

open('runSISOcontrol');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotSISOcontrol(y,u,ys,IAE,time);
% figIm = getframe(gcf);
% imwrite (figIm.cdata, 'SISOtestFT.png', 'png');
% hgexport(fig,'SISOtestFT.eps')

%% Using Bulb as Light Intensity controller.
% Light Insensity is controlled by a SISO controller on the Bulb and the
% Temperature is not taken into any consideration.

% System transfer function:
sys_tf = tf_BLI;
sys_num = cell2mat(sys_tf.num);
sys_den = cell2mat(sys_tf.den);

% Because of unstable light control the controller time constant needs to
% be at least 0.5 sec.
[Kc,tauI] = SIMCPI(sys_tf,0.5);

% Test controller on estimated system
y0 = 0;                 % Shifting of y
y_s0 = 15;              % initial set-point
t_sim = 12;             % Simulation time
stepTime1 = 2;          % Time when first step
stepTime2 = 4;          % Time when second step
step_ys1 = 15;          % step in setpoint
step_ys2 = -15;         % Step in setpoint
stepTime_d1 = 6;        % Time disturbance 1
stepTime_d2 = 8;        % Time disturbance 2
step_d1 = -5;           % Value of disturbance 1
step_d2 = +5;           % Value of disturbance 2

% Test controller on estimated system
open('simSISOcontrol');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotSISOcontrol(y,u,ys,IAE,time);
```

```
figIm = getframe(gcf);
imwrite (figIm.cdata, 'SISOsimBLI.png', 'png');
hgexport(fig,'SISOsimBLI.eps')

%% Test Bulb - Light Intensity controller on real system:

% Choose IO
turnOn_u1 = 1; % Bulb in
turnOn_u2 = 0;
turnOn_u3 = 0;
turnOn_y1 = 0;
turnOn_y2 = 1; % Light Intensity
fixed_u1 = 0;
fixed_u2 = 0;
fixed_u3 = 0;

open('runSISOcontrol');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotSISOcontrol(y,u,ys,IAE,time);
% figIm = getframe(gcf);
% imwrite (figIm.cdata, 'SISOtestBLI.png', 'png');
% hgexport(fig,'SISOtestBLI.eps')

%% Using LED as Light Intensity controller.
% Light Insensity is controlled by a SISO controller on the ED and the
% Temperature is not taken into any consideration.

% System transfer function:
sys_tf = tf_LLI;
sys_num = cell2mat(sys_tf.num);
sys_den = cell2mat(sys_tf.den);
% Because of unstable light control the controller time constant needs to
% be at least 0.5 sec.
[Kc,tauI] = SIMCPI(sys_tf,0.5);

% Test controller on estimated system
y0 = 0;                 % Shifting of y
y_s0 = 4;               % initial set-point
t_sim = 12;             % Simulation time
stepTime1 = 2;          % Time when first step
stepTime2 = 4;          % Time when second step
step_ys1 = 4;           % step in setpoint
step_ys2 = -4;          % Step in setpoint
stepTime_d1 = 6;        % Time disturbance 1
stepTime_d2 = 8;        % Time disturbance 2
step_d1 = -2;           % Value of disturbance 1
step_d2 = +2;           % Value of disturbance 2
```

```
% Test controller on estimated system
open('simSISOcontrol');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotSISOcontrol(y,u,ys,IAE,time);
figIm = getframe(gcf);
imwrite (figIm.cdata, 'SISOsimLLI.png', 'png');
hgexport(fig,'SISOsimLLI.eps')

%% Test LED - Light Intensity controller on real system:

% Choose IO
turnOn_u1 = 0; % Bulb in
turnOn_u2 = 0;
turnOn_u3 = 1;
turnOn_y1 = 0;
turnOn_y2 = 1; % Light Intensity
fixed_u1  = 0;
fixed_u2  = 0;
fixed_u3  = 0;

open('runSISOcontrol');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotSISOcontrol(y,u,ys,IAE,time);
% figIm = getframe(gcf);
% imwrite (figIm.cdata, 'SISOtestLLI.png', 'png');
% hgexport(fig,'SISOtestLLI.eps')
```

D.4 MISOcontrol.m

```
% Tuning, simulation and testing of MISO PI controller on MIMO system.
% This script will tunes and implement a MISO Temperature controller with
% Fan and Bulb as MV's.
%
% The MISO control structure consist of the two SISO controllers.
% The controllers in the MISO control can be tuned more
% agressive than the SISO controllers because the two inputs has gains with
% different signs and any overshoot will be coorected by the other
% controller. Setpoint back-off and alternative smooth controllers
% near setpoint is also implemented.
% The script simulate the controllers in simMISOcontrol.slx and test it in
% realtime in runMISOcontrol.slx
%
% See also SIMCPI SISOCONTROL

% Author: Sindre Johan Heggheim, sindrjh@stud.ntnu.no, November 21th, 2017
% Specialization project TKP4580 at Department of Chemical Engineering

%% Initialization

clear
close all
clc

% The script uses runSystem.slx to run the MIMO system that is connected
% with the port spesified as teh 'com' variable.
% tom1ar8_lib.mdl, tom_lib.mdl and msfun_realtime_pacer.m is also needed
% to run the system.
Ts = 0.1;
com = 'COM4';   % com port
delete(instrfind({'Port'},{com}));
baud = 250000;  % baud rate
Ts  = 0.1;      % Sampling time period
fTt = 0.05;     % Filter time constant for temperature (0.05s - 10s)
fTl = 0.05;     % Filter time constant for light intensity (0.05s - 10s)
fTf = 0.1;      % Filter time constant for light intensity (0.1s - 10s)
t_sim = 800;    % Simulation time

% Simulink variables:
y0 = 0;             % Shifting temperature

% Disturbances
stepTime_d1 = 0;
step_d1= 0;
stepTime_d2 = 0;
step_d2 = 0;

%setpoints:
y1_s0 = 0;
stepTime_ys1 = 0;
step_ys1 = 0;

y2_s0 = 0;
stepTime_ys2 = 0;
step_ys2 = 0;

% tresholds alternative controllers:
altu1_threshold = 0;
altu2_threshold = 0;

%% Load response models and tune controllers:
load('WS_tfs')

% System transfer functions:
sys_tf11 = tf_BT;
sys_tf21 = tf_FT;
```

```
sys_num11 = cell2mat(sys_tf11.num);
sys_den11 = cell2mat(sys_tf11.den);
sys_num21 = cell2mat(sys_tf21.num);
sys_den21 = cell2mat(sys_tf21.den);

% calculate tight PI control parameters
[Kc11,tauI11] = SIMCPI(sys_tf11,0.1);    % tauc = 0.1
[Kc21,tauI21] = SIMCPI(sys_tf21,1);      % tauc = 1

%% Test controller on estimated system
y0 = 0;
y_s0 = 0;
stepTime_ys1 = 30;
stepTime_ys2 = 60;
step_ys1 = 5;
step_ys2 = -5;
stepTime_d1 = 90;
stepTime_d2 = 120;
step_d1 = -2.5;
step_d2 = +2.5;
t_sim = 150;

% Run simulation:
open('simMISOcontrol');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotMISOcontrol(y,u,ys,IAE,time);
% figIm = getframe(gcf);
% imwrite (figIm.cdata, 'MISOsimT.png', 'png');
% hgexport(fig,'MISOsimT.eps')

close_system('simMISOcontrol');

%% Test controller on real system
% measure and shift with standby temperature

t_step = 0; % no step
t_sim  = 2;
stepBulb = 0;
stepFan = 0;
stepLed = 0;
open('runSystem');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS
sb_T = Temperature(1,1); %standby temperature
sb_T = 27;
save_system('runSystem')
close_system('runSystem');

% Shift temperature:
y0 = sb_T+3;
t_sim = 150;
bo_c21 = +0.15; % Back-off on Fan
% The real system will always have error and it is not possible that both
% controllers pushes to the same setpoint from both directions. The result
% will be a very high input usage and oscillations. The back off helps the
% system to choose one of the two setpoints and be controlled by just one
% of the controllers.

% Choose IO
turnOn_u1 = 1; % Bulb in
turnOn_u2 = 1; % Fan in
turnOn_u3 = 0;
turnOn_y1 = 1; % Temp out
turnOn_y2 = 0;
fixed_u1 = 0;
fixed_u2 = 0;
fixed_u3 = 0;

open('runMISOcontrol');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotMISOcontrol(y,u,ys,IAE,time);
% figIm = getframe(gcf);
% imwrite (figIm.cdata, 'MISOtestT.png', 'png');
% hgexport(fig,'MISOtestT.eps')

%% Test controller on real system:
% Lowering the oscillations by decreasing the back-off in setpoint and
% adding alternative controllers with smooth control near the setpoint

bo_c21 = +0.1; %  Back-off on Fan
altu1_threshold = 0.2;
altu2_threshold = 0.2;

[Kc11,tauI11] = SIMCPI(sys_tf11,0.1);
[Kc21,tauI21] = SIMCPI(sys_tf21,1);
% alternative controllers has 10 times larger tauc and the backoff on the
[alt_Kc11,alt_tauI11] = SIMCPI(sys_tf11,1);
[alt_Kc21,alt_tauI21] = SIMCPI(sys_tf21,10);

open('alt_runMISOControl');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotMISOcontrol(y,u,ys,IAE,time);
figIm = getframe(gcf);
imwrite (figIm.cdata, 'MISOtestT_alt.png', 'png');
hgexport(fig,'MISOtestT_alt.eps')
```

D.5 SIMCPI.m

```
function [Kc,tauI] = SIMCPI(tf,tauc)
% Calculate SIMC PI tuning for 1 order response (tf) for given tauc

sys_num = cell2mat(tf.num);          % tf system
sys_den = cell2mat(tf.den);          % tf system
if sys_den(2)~=1                     % make tf on time constant format
    sys_num=sys_num/sys_den(2);
    sys_den=sys_den/sys_den(2);
end%if

% First order response parameters:
k = sys_num(2);                      % Plant gain
tau1 = sys_den(1);                   % Dominant lag time constant
theta = tf.IOdelay;                  % Delay

% SIMC PI tuning:
Kc = 1/k*tau1/(tauc+theta);          % controller gain
tauI = min(tau1,4*(tauc+theta));     % Integral time
end%fun
```

D.6 SIMCPI_nosat.m

```
function [Kc,tauI,tauc] = SIMCPI_nosat(tf,delta_ys_nosat)
% Calculate tuning of smooth control with no saturation for delta_ys_nosat
% The change in input is set as 1.

sys_num = cell2mat(tf.num);          % tf system
sys_den = cell2mat(tf.den);          % tf system
if sys_den(2)~=1                     % make tf on time constant format
    sys_num=sys_num/sys_den(2);
    sys_den=sys_den/sys_den(2);
end%if

% First order response parameters:
k = sys_num(2);                      % Plant gain
tau1 = sys_den(1);                   % Dominant lag time constant
theta = tf.IOdelay;                  % Delay
```

```
% Calculate tauc that does not saturate:
u_max = 1;
Kc_nosat = u_max/delta_ys_nosat;    % Kc that given no saturation
tauc = abs(1/Kc_nosat*tau1/k);      % tauc that max u without saturating

% SIMC PI tuning:
Kc = 1/k*tau1/(tauc+theta);         % controller gain (Kc_nosat)
tauI = min(tau1,4*(tauc+theta));    % Integral time

end%fun
```

D.7 MIMOcontrol.m

```
% Tuning and Implementation of MIMO control.
% The MIMO consist of SISO and MISO controllers and switches the focus of
% the Bulb between the two outputs. The estimated temperature from Bulb to
% Light Intensity is used to choose focus.
% The controllers are tuned with SIMC PI
%
% See also SIMCPI

% Author: Sindre Johan Heggheim, sindrjh@stud.ntnu.no, December 17th, 2017
% Specialization project TKP4580 at Department of Chemical Engineering

clear
close all
clc
run('init.m') % init files for TOM1A system and set Simulink variables to 0
load('WS_tfs') % Load response models and initialize system parameters

%% Tuning of Temperature controllers
% System transfer functions:
sys_tf11 = tf_BT;
sys_tf21 = tf_FT;
sys_num11 = cell2mat(sys_tf11.num);
sys_den11 = cell2mat(sys_tf11.den);
sys_num21 = cell2mat(sys_tf21.num);
sys_den21 = cell2mat(sys_tf21.den);

% % calculate PI control parameters
[Kc11,tauI11] = SIMCPI(sys_tf11,5);         % B-T
[Kc21,tauI21] = SIMCPI(sys_tf21,2);         % F-T
% [altKc11,alt_tauI11] = SIMCPI(sys_tf11,5);
% [altKc21,alt_tauI21] = SIMCPI(sys_tf21,2);

%% Tuning of Light Intensity controllers
sys_tf12 = tf_BLI;
sys_tf32 = tf_LLI;
sys_num11 = cell2mat(sys_tf12.num);
sys_den11 = cell2mat(sys_tf12.den);
sys_num21 = cell2mat(sys_tf32.num);
sys_den21 = cell2mat(sys_tf32.den);

% calculate PI control parameters
```

```
[Kc12,tauI12] = SIMCPI(sys_tf12,1);
[Kc32,tauI32] = SIMCPI(sys_tf32,0.5);

%% Operation at:
% run('steadyState') % runs system at stst.u to find stst.y
stst.u = [0.5,0.2,0.5]';
stst.y = [31.51,26.50]';

%% Temperature from B-LI control
% calculate steady state gain to estimate temperature from B-LI control
tf_LIT = tf_BT/tf_BLI;
T_est.num = cell2mat(tf_LIT.num);
T_est.den = cell2mat(tf_LIT.den);
stst_BTgain = T_est.num(2)/T_est.den(2);

%% Test controller on real system: Temperature controlled system
bo_c21 = 0.1;
y0 = 27;                 % shift temperature
%Shifted steady state setpoints:
y1_s0 = stst.y(1)-y0;    %4.5
y2_s0 = stst.y(2);       %26.5

step_ys1_1 = -2;
stepTime_ys1_1 = 60;
step_ys1_2 = -0;
stepTime_ys1_2 = 0;

step_ys2_1 = -20;
stepTime_ys2_1 = 30;
step_ys2_2 = +10;
stepTime_ys2_2 = 90;

t_sim = 150;
open('runMIMOcontrol');
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotMIMOcontrol(y,u,ys,0,IAE,time);
figIm = getframe(gcf);
imwrite (figIm.cdata, 'MIMOtestTemp.png', 'png');
hgexport(fig,'MIMOtestTemp.eps')
```

D.8 LightPlantMPC.m

```
% Implementation of MPC control of MIMO plant.
% The nominal operation point was choosen and the identified state space
% model saved in 'WS_MIMO_models' was used to make the MPC.
%
% See also makeMPCobj
% Author: Sindre Johan Heggheim, sindrjh@stud.ntnu.no, December 17th, 2017
% Specialization project TKP4580 at Department of Chemical Engineering

%% Initialization

clear
clc
run('init')
load('WS_MIMO_models');

% Find steady state at u = [0.5,0.2,0.5]'
% run('steadyState')
stst.y = [31.51,26.50]';
stst.u = [0.5,0.2,0.5]';

% Tune MPC object
Tsc = 0.1;
ph = 10;          % Prediction horizon in seconds
ch = 1;           % Control horizon in seconds
MPCobj = makeMPCobj(MIMO_tf,stst,Tsc,ph,ch);
MPCobj.Model.Nominal.Y = stst.y;
MPCobj.Model.Nominal.U = stst.u;

%% Test steady state control with MPC
t_sim = 100;
open('LightPlant_mpcModel')
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result:
fig = plotMIMOcontrol(y,u,ys,us,IAE,time);
% figIm = getframe(gcf);
% imwrite (figIm.cdata, 'MIMOtestMPC.png', 'png');
% hgexport(fig,'MIMOtestMPC.eps')

%% Test changes setpoints:
t_sim = 100;

% Temperature setpoints:
stepTime_ys1_1 = 30;
step_ys1_1 = +1;
stepTime_ys1_2 = 70;
step_ys1_2 = -1;

% Light Intensity setpoints:
stepTime_ys2_1 = 20;
step_ys2_1 = 5;
stepTime_ys2_2 =  50;
step_ys2_2 = -5;
```

```
open('LightPlant_mpcModel')
set_param(gcs,'StopTime',num2str(t_sim));
set_param(gcs,'SimulationCommand','start');
while get_param(gcs,'SimulationStatus')=='running' %Waiting on simulink
    pause(1)
end%while
pause(1) %Waiting for simulink to save to WS

% plot result
fig = plotMIMOcontrol(y,u,ys,us,IAE,time);
% figIm = getframe(gcf);
% imwrite (figIm.cdata, 'MIMOtestMPC2.png', 'png');
% hgexport(fig,'MIMOtestMPC2.eps')
```

D.9 makeMPCobj.m

```
function MPCobj = makeMPCobj(MIMO_tf,stst,Tsc,ph,ch)
% Building and tuning of MPC object used to control MIMO system.
% MIMO_tf:  2x3 transfer functions for MIMO system.
% stst:     Structure of steady state conditions with the 3 inputs '.u'
% Tsc:      Timesteps in MPC
% ph:       Prediction horizon in time units
% ch:       Control horizon in time units

% Author: Sindre Johan Heggheim, sindrjh@stud.ntnu.no, November 21th, 2017
% Specialization project TKP4580 at Department of Chemical Engineering

plant = MIMO_tf;
plant = setmpcsignals(plant,'MV',[1,2,3]); % define all input as MV

% Define MPC object:
nu = 3;
ny = 2;

p = ph/Tsc;                     % number of steps in prediction horizon
m = ch/Tsc;                     % number of steps in control horizon
MPCobj = mpc(plant,Tsc,p,m);

% Span of I/O variables:
Uspan = ones(nu,1);
[num,den] = tfdata(MIMO_tf);
numcell1 = num(1,1);
numcell2 = num(2,1);
dencell1 = den(1,1);
dencell2 = den(2,1);
num1 = cell2mat(numcell1);
den1 = cell2mat(dencell1);
num2 = cell2mat(numcell2);
den2 = cell2mat(dencell2);
Yspan = [num1(2)/den1(2), num2(2)/den2(2)]';

% Constraints on inputs:
consU = [0,1;0,1;0,1];
```

```
% Scaling factor and constraits on u:
for i = 1:nu
    MPCobj.MV(i).ScaleFactor = Uspan(i);
    MPCobj.MV(i).Min = consU(i,1);
    MPCobj.MV(i).Max = consU(i,2);
end%for

% Scaling factor for y:
for i = 1:ny
    MPCobj.OV(i).ScaleFactor = Yspan(i);
end%for

% MV targets:
MPCobj.MV(1).Target = stst.u(1);
MPCobj.MV(2).Target = stst.u(2);
MPCobj.MV(3).Target = stst.u(3);

% Weights
MPCobj.Weights.MV = [1,0,0];
MPCobj.Weights.MVRate = [1,0.1,0.1];

MPCobj.Weights.OV = [1000,1];

review(MPCobj)
end%fun
```

D.10 LightPlantLQG.m

```
% Calculate LQR gain and estiamtor gain
% unfinished code used to implement LQG on MIMO plant

% Author: Sindre Johan Heggheim, sindrjh@stud.ntnu.no, December 18th, 2017
% Specialization project TKP4580 at Department of Chemical Engineering
clear
clc
load('WS_MIMO_models.mat')
run('init')
nu = 3;
ny = 2;
nx = 4;

% run('steadyState')
stst.y = [31.51,26.50]';
stst.u = [0.5,0.2,0.5]';

% state space model:
ssmodel = ss(MIMO_tf);
[A,B,C,D] = ssdata(ssmodel);

% Kalman filter gain:
G = zeros(nx,1);
H = zeros(ny,1);
SSmodel = ss(A,[B G],C,[D H],Ts,'inputname',{'u1' 'u2' 'u3' 'w'},...
    'outputname',{'y1' 'y2'});
```

```
Qn = 0;
Rn = [1,0;0,1];
[K_f,L,P] = kalman(SSmodel,Qn,Rn);

% LQR gain:
Q = diag([0,1,0,0]);
R = diag([1,1e9,1e9]);
[K_lqr,S,e] = lqr(A,B,Q,R);
x0 = L*stst.y;
u0 = stst.u;
```