AALBORG UNIVERSITY

STUDENT REPORT

IRS10-1-F19

Optimization and Modeling of Transient of a Parallel Multiple Variable-Speed Water Pump System

Students:
Andrius Kulšinskas
Allan Gjerlevsen

Supervisors:
Petar Durdevic Løhndorf
Zhenyu Yang

May 31, 2019

AALBORG UNIVERSITY
STUDENT REPORT

School of Information and Communication Technology
Niels Bohrs Vej 8
DK-6700 Esbjerg
http://sict.aau.dk

Title:
Optimization and Modeling of Transient of a Parallel Multiple Variable-Speed Water Pump System

Theme:
Scientific Theme

Project Period:
Spring Semester 2019

Project Group:
IRS10-1-F19

Participant(s):
Andrius Kulšinskas
Allan Gjerlevsen

Supervisor(s):
Petar Durdevic Løhndorf
Zhenyu Yang

Copies: 1

Page Numbers: 64

Date of Completion:
May 31, 2019

Abstract:

The project is aimed towards improvement of efficiency in multiple water pump systems. The problem is defined as lack of proper optimization algorithms in water pump systems worldwide, which affects power consumption, environment and water loss. Three methods are proposed as possible solutions to the problem. The first method is an efficiency optimization algorithm which was previously developed with faulty results. The method is re-implemented with improvements to the calculation process and accompanying data and tested on a real life water pump platform. Over and underpressuring during transitions in system are identified as one of the contributors to the problem. Experimental data is gathered, showing system's products' change as operating conditions - pumps in action, input speed and choke valve percentage - change over time. A decoupling method is then designed based on similar work, which attempts to remove coupling between the pumps' pressures and reduce spikes in head during transitions. Another method is designed from scratch, creating a model of pump system using a sum of two transfer functions to represent each pumps' performance. The two methods are then tested in offline simulations. The observed results are evaluated and a conclusion is drawn.

Contents

Preface

The project entitled *Optimization and Modeling of Transient of a Parallel Multiple Variable-Speed Water Pump System* was made by two students from the Intelligent Reliable Systems programme at Aalborg University Esbjerg, for the P10 project during the tenth semester.

From hereby on, every mention of 'we' refers to the two co-authors listed below.

Aalborg University, May 31, 2019.

Andrius Kulšinskas
<kulsinskas.andrius@gmail.com>

Allan Gjerlevsen
<agjerl13@student.aau.dk>

Chapter 1

Introduction

There are conflicting reports about the energy consumption in water pump systems worldwide - various sources list this number at anywhere between 2 and 10% [1], [2], [3], [4] and it is believed that the percentage of world's energy consumed by the pumps will keep increasing in the coming years. However small or large this number may be, it is not uncommon for those pump systems to be inefficient. An inefficient pump costs more to operate, requires more maintenance and can result in loss of water it is pumping. Therefore, it is important to try and optimize the water pump systems, in order to reduce the power consumption and the strain on the environment. This can be done in several ways - using multiple pumps to distribute the load, utilizing variable speed pumps or adjustable valve opening to optimize the efficiency and power consumption or reduce the damage to the system so it performs as expected for longer duration. There have already been various implementations of efficiency algorithms that try to minimize the power consumption - [5], [6], [7], [8], [9]. A previous project done by the authors of this report implemented one of the methods on a multiple pump platform [10]. It was shown that while the method worked, the transitional period when switching pump combinations put system in an unknown state until steady state was reached, which lead to over or under pressuring. Therefore, in addition to making improvements to the efficiency algorithm, this report will investigate modeling of the transitional state of the pumps on a multiple variable speed pump system.

Chapter 2

Problem analysis

2.1 Problem description

In order to optimize a pump system, it is necessary to have a model of the system's performance under various conditions, so that operation scheduling can be made. For a single pump, only input variables available are the pump's speed and the pressure valve. However, a multiple pump system can switch between combinations of pumps in operation as well, creating a bigger range of operation. In various works using a multiple pump system, regardless of whether a parallel or series system is used, a steady-state model is often obtained to show the pump curves. A generic pump curve can be seen in figure 2.1, where a pressure and flow plot shows how system's performance changes with two input conditions - pump speed if the pump allows it and the choke valve opening.

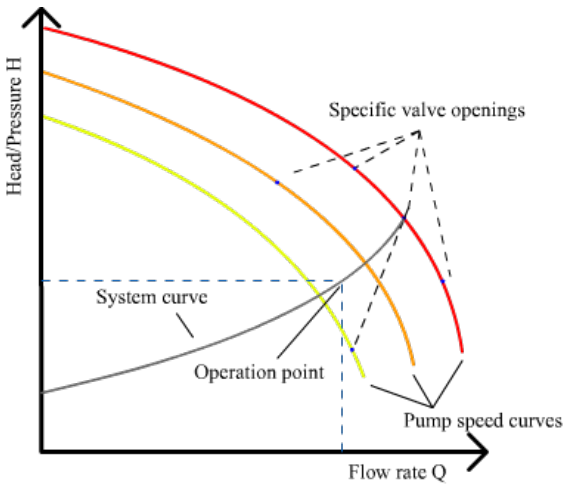


Figure 2.1: A generic pump curve with different valve openings and speeds [10].

Commonly the pressure represents the difference in pressure between inlet and outlet of the pump, while the flow measures the rate at which the liquid flows through the pump. In a multiple pump case, if the system is running in parallel, the pumps

share common pressure, but produce individual flow rates which are then summed up. However, the flow rate is not doubled when running two pumps compared to just one [11]. In a multi-pump case, the flow rate can be significantly smaller than expected due to increase in friction. When the system is running in series, the pumps share a common flow rate, while the pressure increases with the number of pumps. Therefore, depending on requirements, one can increase the flow rate or the pressure by utilizing multiple pumps. However, efficiency algorithms only optimize based on system's steady state - as the system transitions from one operating condition to another, its performance in terms of pressure, flow and power are unknown. Coupling between pumps in a multi-pump case makes identification of this transitional performance difficult, as seen in figure 2.2. Remaining at same operating conditions but turning on another pump affects the head of both pumps.

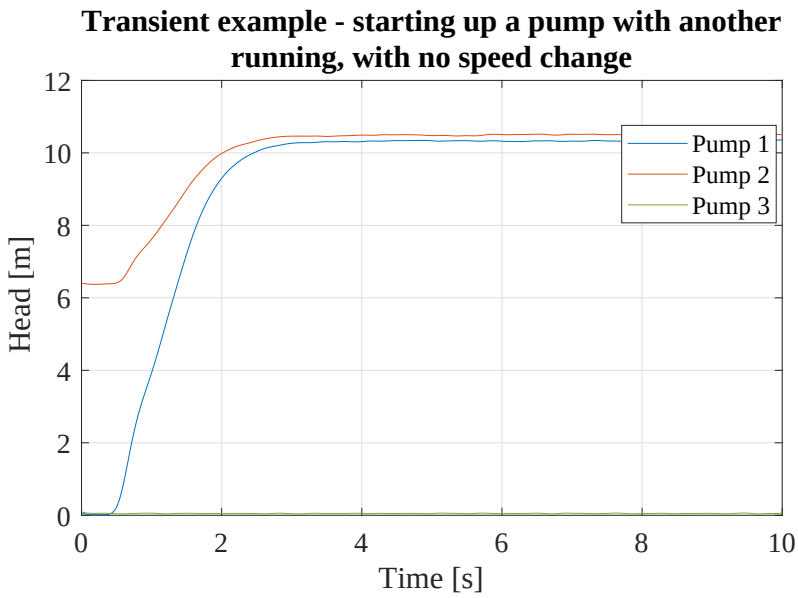


Figure 2.2: Example of transition, where the pump speed is kept the same but another pump is introduced.

Therefore, there is a need to find a way to model this transition to reduce risk of damaging the system. There is not much research available regarding this matter. One idea we will investigate, that was proposed in [12], is decoupling of pump's head measurements, in order to know how much pressure each pump is producing even with multiple pumps in operation. Additionally, we will attempt to model the coupled performance as a time-varying first order system.

2.2 Methods

An efficiency optimization algorithm, which was initially developed in a previous project, uses similar approach this time as well - the goal is to pick the most efficient pump combination to use given the known requirements - reference head and

some valve opening percentage. A static model derived offline from pump curves is used to choose the most optimal combination, in terms of efficiency, online and set the appropriate speed. An online estimator based on pre-calculated data is used to determine system curve coefficient k_1 when valve opening changes. The main difference is that instead of considering all pumps to be different and have only similar speeds, the current approach sums up pumps' performances and sets identical speeds.

The differences in the methods between the projects can be seen in table 2.1.

Configuration	**Previous project**	**Current project**
Speed range	10-100%	50-100%
Speed steps	5%	5%
Valve opening range	5-100%	25-75%
Valve opening steps	5%	2%
Combination performance	Assumes dissimilar pumps	Assumes identical pumps

Table 2.1: Comparison between projects

The speed range was reduced due to fault in the frequency converters in the pumps, which resulted in faulty power readings at low pump speeds. As seen in figure 2.3, despite all products of the pump, as well as the speed, increasing, the current drops as soon as it reaches 1 amp for the first time. The choke valve range was also reduced to minimize the noise in the system when running the system at high speeds. To compensate, more steps were used.

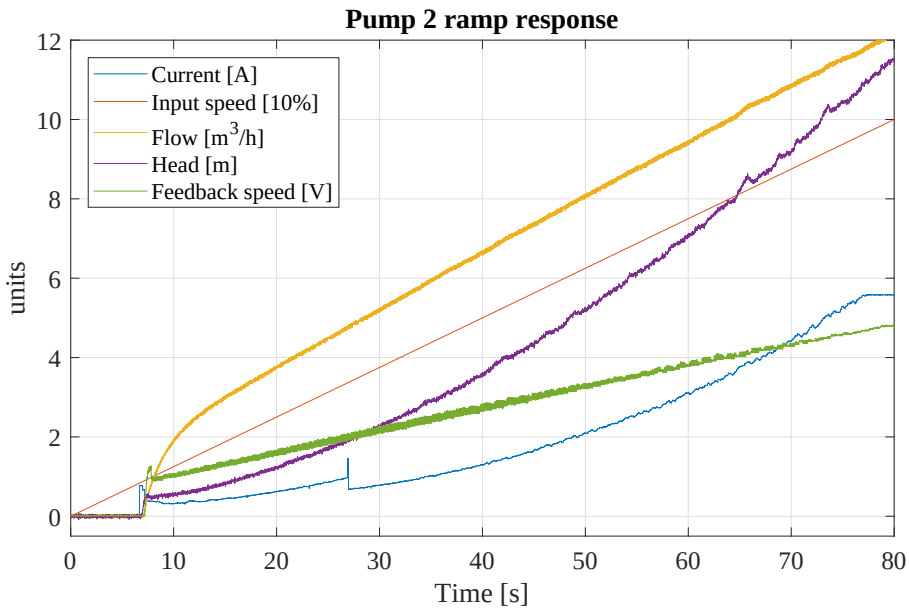


Figure 2.3: Example of transition, where the pump speed is kept the same but another pump is introduced.

For transitional performance, it was decided to gather as much data about various transitions as possible. Same speed and valve opening ranges, were used, with steps

of 10% in both. The transitions themselves will be defined in a later chapter.
The decoupling system will be designed to calculate one pump's contribution to another pump's head. The method will then attempt to subtract this addition due to coupling, in order to calculate pump's performance without any additional effects from other pumps.

2.3 Chapter conclusion

In order to improve the power consumption for a multiple variable-speed parallel pump system, an optimization algorithm will be adopted. An attempt to improve the algorithm will be made through the use of more data points on the system curve, fixes to parameter estimator's algorithm in the edge cases and new efficient combination speed calculation method. Data of system changing operating conditions will also be obtained and later represented as first order systems. A decoupling function will be designed to avoid over pressure and under pressure in transitions.

Chapter 3

Mathematical modeling

The modeling of equations used in efficiency algorithm was based on [6]. The equations start with affinity law:

$$\frac{Q(\omega_1)}{Q(\omega_2)} = \frac{\omega_1}{\omega_2} \tag{3.1}$$

$$\frac{H(\omega_1)}{H(\omega_2)} = \frac{\omega_1^2}{\omega_2^2} \tag{3.2}$$

$$\frac{P(\omega_1)}{P(\omega_2)} = \frac{\omega_1^3}{\omega_2^3} \tag{3.3}$$

We can then express head - pressure between inlet and outlet of the pump - H and power P as a function of speed ω, flow rate Q and polynomial coefficients a and p [6].

$$H(\omega) = a_0\omega^2 + a_1\omega Q(\omega) + a_2(Q(\omega))^2 \tag{3.4}$$

$$P(\omega) = p_0\omega^3 + p_1\omega^2 Q(\omega) + p_2\omega(Q(\omega))^2 + p_3(Q(\omega))^3 \tag{3.5}$$

The coefficients are obtained from curve fitting 2nd and 3rd order polynomials to flow-head and flow-power curves, respectively.

The efficiency of a combination n is then calculated as [6]:

$$\eta_i = \frac{\rho g(a_0^i\omega_i^2 + a_1^i\omega_i Q_i(\omega_i) + a_2^i Q_i(\omega_i))^2)Q_i}{p_0^i\omega_i^3 + p_1^i\omega_i^2 Q_i(\omega_i) + p_2^i\omega_i(Q_i(\omega_i))^2 + p_3^i(Q_i(\omega_i))^3} \times 100\% \tag{3.6}$$

System curve H_{sys} is head-flow relation that includes static head k_0 and dynamic head k_1Q^2 [6].

$$H_{sys} = k0 + k1Q^2 \tag{3.7}$$

Static head does not exist in our prototype, while dynamic head coefficient $k1$ is obtained from data tests for each valve opening and pump combination.

Chapter 4

System identification

4.1 Pump characteristics

Pump curves have been identified using the testing conditions defined in 2.1. A single pump's curves can be seen in figure 4.1, where input speed range and choke valve percentage was used as described in table 2.1, with speed steps of 5% for more detailed information. A new curve is a new speed step across the entire valve opening range.

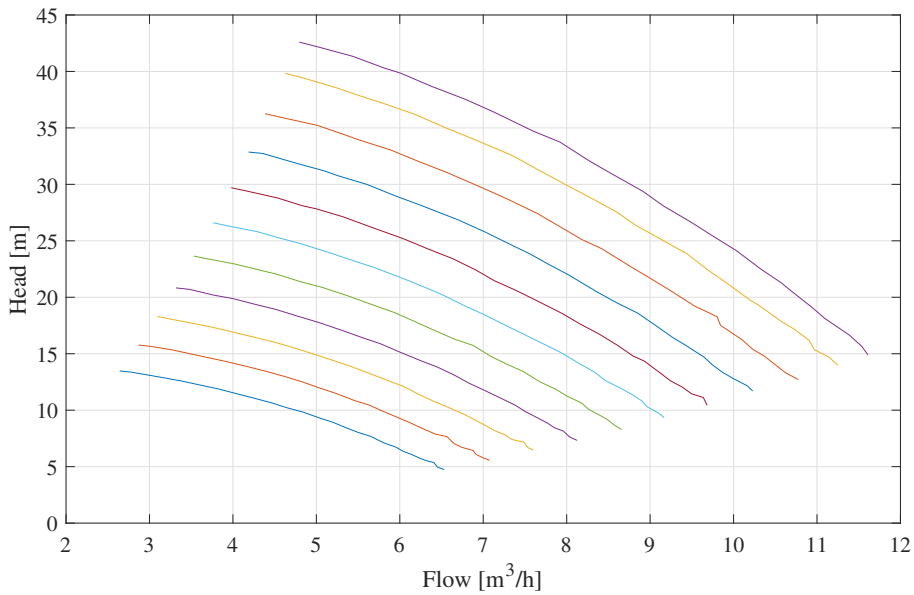


Figure 4.1: 2nd pump's curves.

In figure 4.2, we can see how the power curves look like, showing the relationship between power and flow rate.

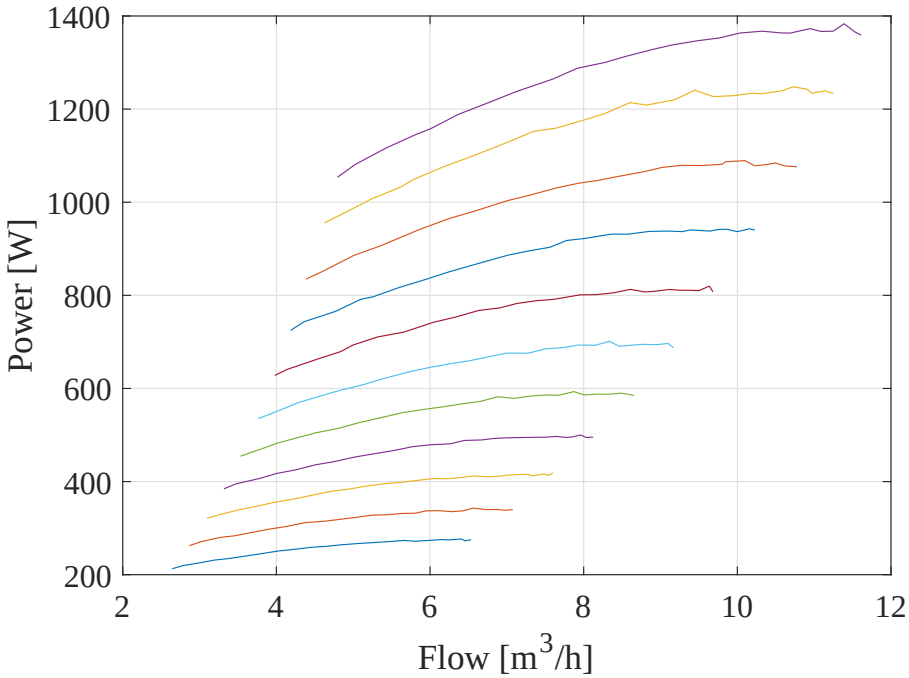


Figure 4.2: 2nd pump's power curves.

When combining pumps' performance, we can see how friction affects the system (figure 4.3). The lengthier pipeline and additional fittings and other loss coefficients cause the flow rate to be significantly below the expected amount - triple the single pump's product.

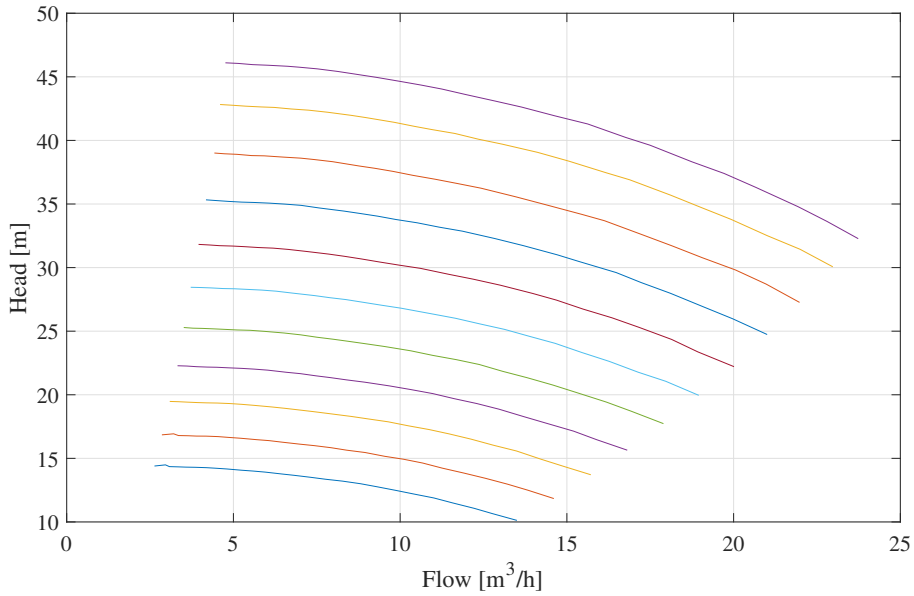


Figure 4.3: Flow-head plot with all 3 pumps running.

Efficiency depends on the flow rate, as seen in figure 4.4. As a result, the efficiency is dependent on the valve opening, since opening or closing the valve affects the flow

rate. For example, a single pump at 75% valve opening produces a flow rate of 11.7 m^3/h, but little pressure, resulting in a small hydraulic power and overall efficiency. At same flow rate level, both 2 and 3 pump combinations have higher efficiency due to smaller choke valve opening.

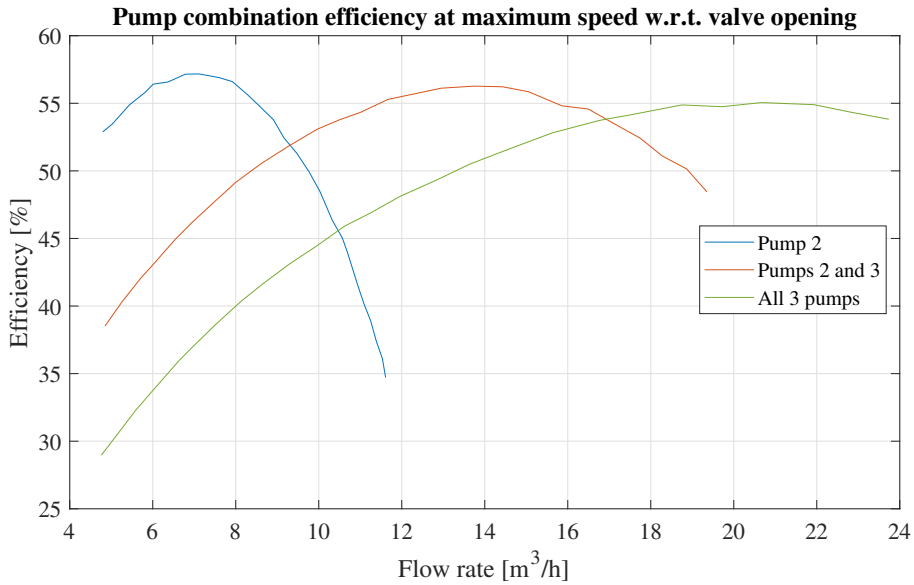


Figure 4.4: Three pump combination efficiencies

With this data, we can fit coefficients to 2nd and 3rd order polynomials for head and power, as described in equations 3.4 and 3.5.

4.2 Obtaining transitional data

It is necessary to observe transitions with various variables in play, in order to understand how each pump's transition is affected by itself and due to coupling with others. A transition is defined as a change from one pump combination to another, while also potentially changing the speed and observing the head, flow and power for 10 seconds when the change happens. The transitions are observed with valve openings of 25% to 75%, with increments of 10%. At any given combinations of pumps running, we can either switch on one or more pumps that are not currently running or turn off any amount of pumps that are in operation. We do not allow the data acquiring algorithm to both turn on a pump and turn off another pump in the same step, as the transition would be more difficult to analyze. As such, we initially obtained 38 different possible transitions, often referred to as "cases". After looking at some of the experiments, it was decided to obtain data for three more cases. At the beginning of these new transitions, a single pump is running, while during the transition only the end speed is changed.

4.2.1 Definition of cases

First, we define pump combinations. With 3 pumps on the platform, there are 8 different possible combinations of pumps running, listen below in table 4.1.

Combination number	Pump 1 usage	Pump 2 usage	Pump 3 usage
1	on	off	off
2	off	on	off
3	off	off	on
4	on	on	off
5	on	off	on
6	off	on	on
7	on	on	on
8	off	off	off

Table 4.1: The eight possible combinations of pumps.

Then, knowing speed at the beginning of transition and the end, we can determine which pumps were running at the start and what was the combination at the end. A matrix was then made to determine the case number - defining which transition number it is - using information about the pump combinations at start and beginning. Using 8×8 matrix, each row number indicates the combination at the beginning of transition, while the columns show the combination number at the end of the transition. The matrix can be seen in 4.1. For example, if we were running pumps 1 and 2 at the beginning and started up pump 3 during the transition, it would mean we have combinations 4 and 7 at the beginning and end of the transition, respectively. Then, looking at 4th row and 7th column in the matrix, we see that the case number is 14.

$$\begin{bmatrix} 39 & 0 & 0 & 2 & 3 & 0 & 4 & 20 \\ 0 & 40 & 0 & 6 & 0 & 7 & 8 & 24 \\ 0 & 0 & 41 & 0 & 10 & 11 & 13 & 28 \\ 21 & 25 & 0 & 0 & 0 & 0 & 14 & 35 \\ 22 & 0 & 29 & 0 & 0 & 0 & 15 & 36 \\ 0 & 26 & 30 & 0 & 0 & 0 & 12 & 37 \\ 23 & 27 & 31 & 32 & 33 & 34 & 0 & 38 \\ 1 & 5 & 9 & 16 & 17 & 18 & 19 & 0 \end{bmatrix} \tag{4.1}$$

4.2.2 Definition of speed changes

Additionally, while there is both at least one pump at the beginning and the end of the transition, we can perform 36 different speed changes - starting with 50% pump speed, we can either keep it at the same speed or change it with increments of 10% all the way to 100%. Additionally, when no pumps are in operation at the beginning, we can go to anywhere between 50% and 100% speed, again, with increments of 10%.

Similarly, when turning off pumps completely, we start at speeds between 50% and 100%, going down to 0% in those cases. When a pump is starting up or shutting down, we include 12 more possible speed changes - starting at 0% when a pump is powering up and going to one of the steps between 50% and 100%, or, similarly for shutting off case - going down from a specific speed percentage down to 0%. Finally, to keep the structure even, an unused 0% to 0% speed change scenario was included, to make looping through the data easier. In total, there were 49 speed changes defined, which are based on input speeds before and during the transition. The indexes of each change and the speed step percentages can be seen in table 4.2.

Index	**Step**	**Index**	**Step**	**Index**	**Step**	**Index**	**Step**	**Index**	**Step**	**Index**	**Step**	**Index**	**Step**
1	0-0	8	50-0	15	60-0	22	70-0	29	80-0	36	90-0	43	100-0
2	0-50	9	50-50	16	60-50	23	70-50	30	80-50	37	90-50	44	100-50
3	0-60	10	50-60	17	60-60	24	70-60	31	80-60	38	90-60	45	100-60
4	0-70	11	50-70	18	60-70	25	70-70	32	80-70	39	90-70	46	100-70
5	0-80	12	50-80	19	60-80	26	70-80	33	80-80	40	90-80	47	100-80
6	0-90	13	50-90	20	60-90	27	70-90	34	80-90	41	90-90	48	100-90
7	0-100	14	50-100	21	60-100	28	70-100	35	80-100	42	90-100	49	100-100

Table 4.2: 49 defined input speed changes.

4.2.3 Data filtering

In total, we obtain 998 different changes in terms of speed and combinations per specific valve opening. With the 6 possible valve opening percentages, we end up with 5988 transitions to analyze in total. There is always at least one sample of transition data available, although some transitions occurred multiple times. In those cases, the mean value of the sum of the data was taken, which helped reducing the effects of random behaviors we observed, which will be discussed later. Although three channels of data per pump were obtained - flow, head and power - our main concern was the pressure, so all analysis from hereon was done on head measurements only. Additionally, 30th order low-pass FIR filter with 10 Hz cutoff frequency was added to clean up the data. A script was then made to load all test data and sort it based on valve opening, case number and speed change, while filtering the head. The script can be found in appendix chapter 9.1.

4.3 Correlation

It was decided to first try and analyze the data statistically, due to huge amount of it and attempt to make a simplified model based on correlations between the pumps and their variables. Figure 4.5 shows the correlation between first and second pumps' pressure, when the first pump is running and second one is starting up. The correlations are done at 50% and 100% pump speeds, indicated by labels $P1$ 5 as 50% input for pump one, $P1$ 10 as pump one at 100% and similarly for the second pump with labels $P2$... . At low valve opening, it shows high correlation between the pumps when they are operating at same speed, but the correlation coefficient drops when the speed is altered. Moreover, the correlation coefficient is smaller

when comparing same pump at different speeds, leading to a conclusion that an accurate linear model cannot be made from statistical data alone for pump speed changes.

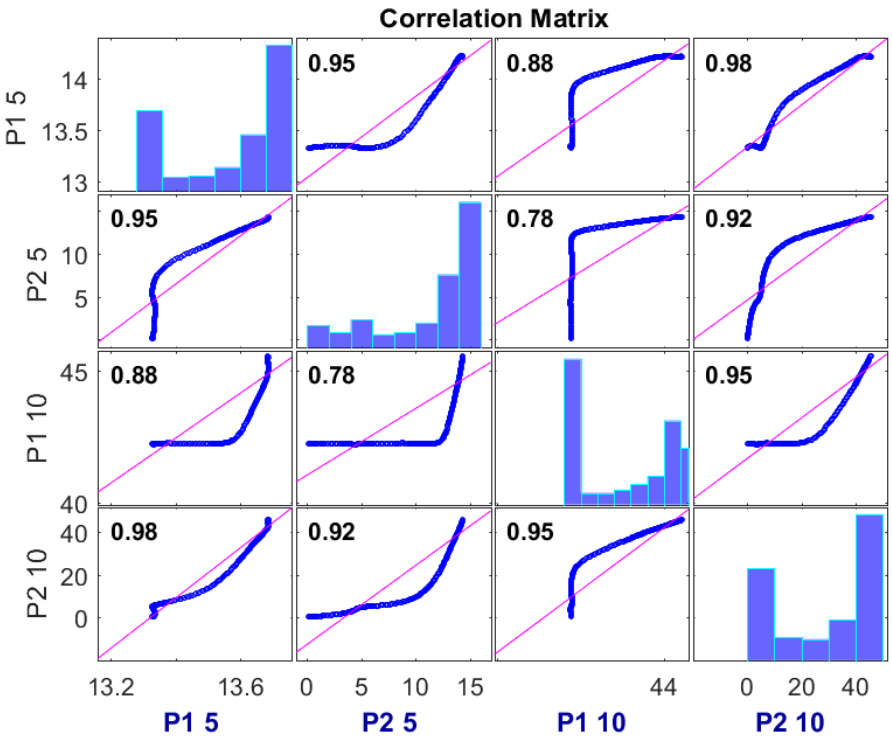


Figure 4.5: Correlations between pressure in the pumps at low valve opening.

In figure 4.6, the same test is performed with a high valve opening. The correlation coefficients are now higher between all points of data, showing that the coupling is stronger at higher valve openings.

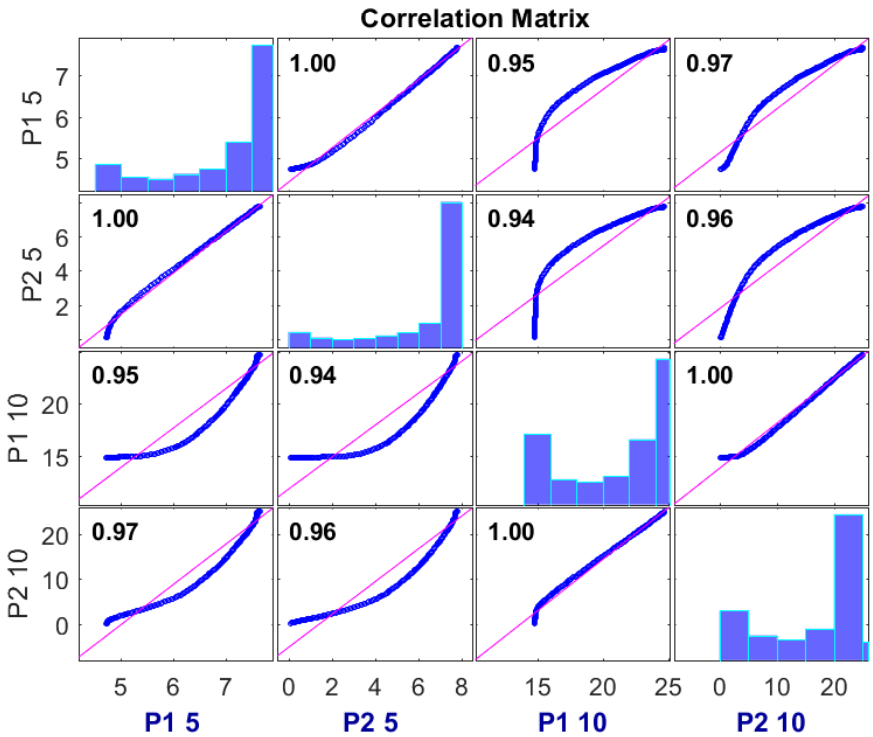


Figure 4.6: Correlations between pressure in the pumps at high valve opening.

We then tested the correlation when shutting down a pump. Figure 4.7 shows the correlation coefficients when turning off third pump, while pumps one and two are still running during the transition, with valve opening of 25%. The correlation coefficient between third pump and the other two was negative, indicating that the pressure for the two pumps increased as the third pump's head dropped. This was thought to be a result of small choke valve percentage.

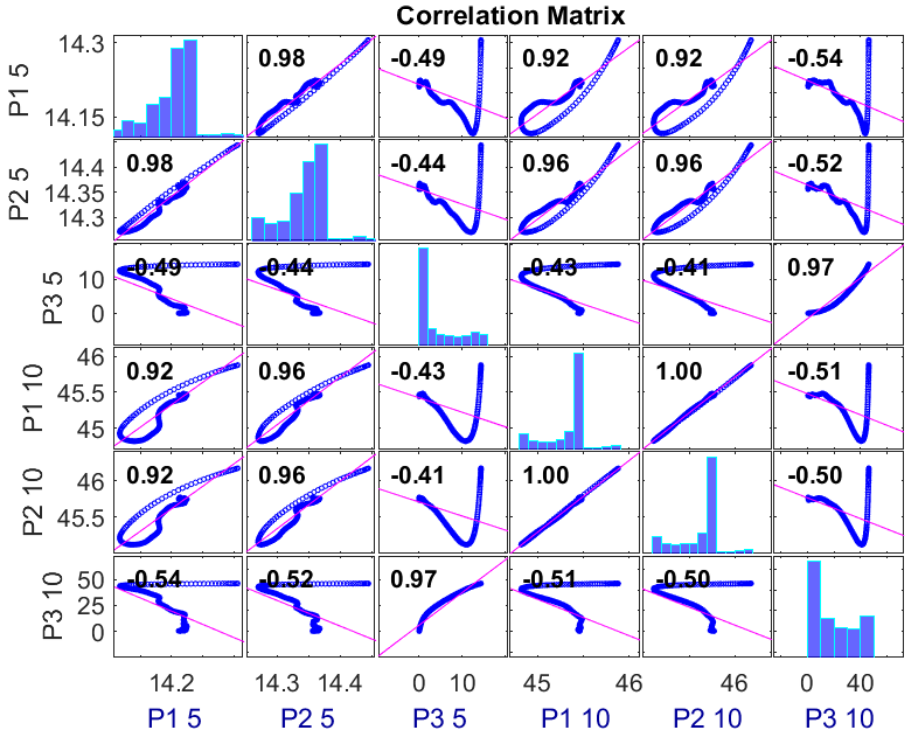


Figure 4.7: Correlation between the three pumps while turning off third one and running the other two.

The same transition test was then performed with 75% valve opening. The results, shown in figure 4.8, now display high positive correlation coefficients, showing that the coupling is dependent on valve opening percentage.

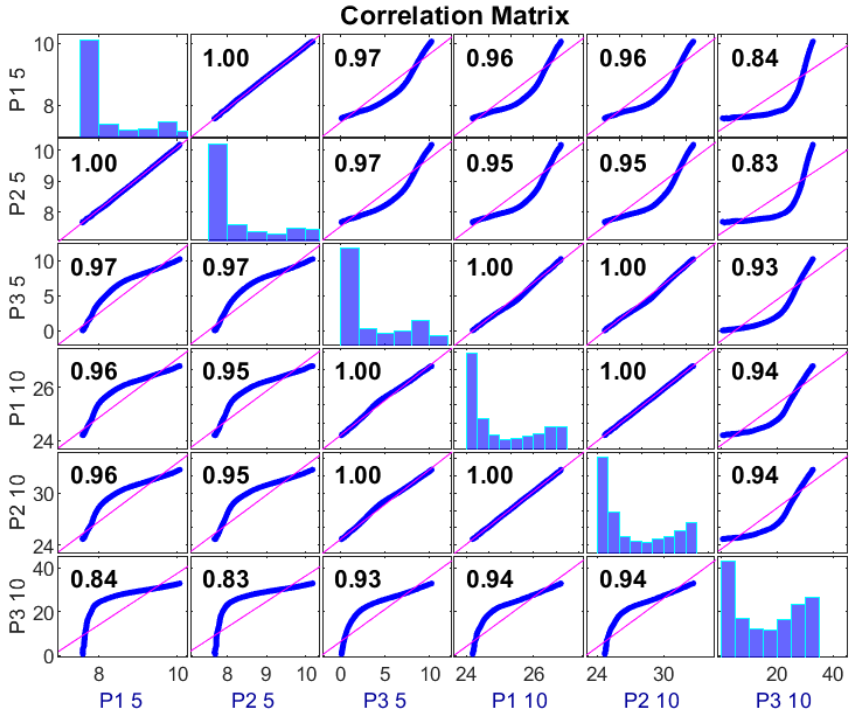


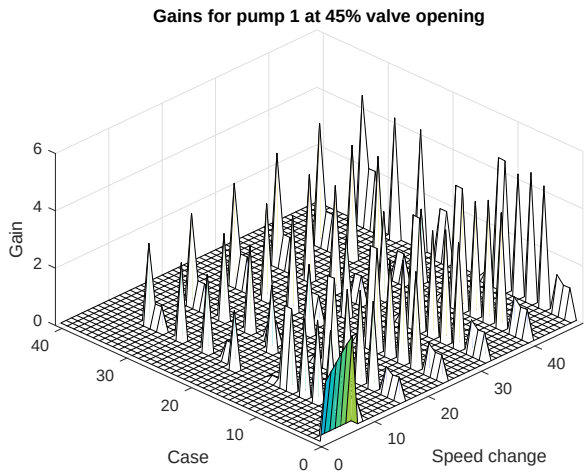
Figure 4.8: Correlation between the three pumps while turning off third one and running the other two.

From correlation coefficients at various operating points, we see that the pumps generally display strong coupling - as one pump changes its pressure, so do the other pumps. However, in some cases, the correlation can be misleading, as was the case in figure 4.7, where shutting down a pump - and thus removing its coupling effect from others - showed negative correlation between the pressures of the pumps, stating that turning off a pump increases head. From correlation between pumps' pressure in some scenarios, we see that it is difficult to generalize the changes from a purely statistical point of view, as different operating conditions affect transitions in different ways. Therefore, an attempt will be made to analyze the transfer function coefficients for each transition individually.

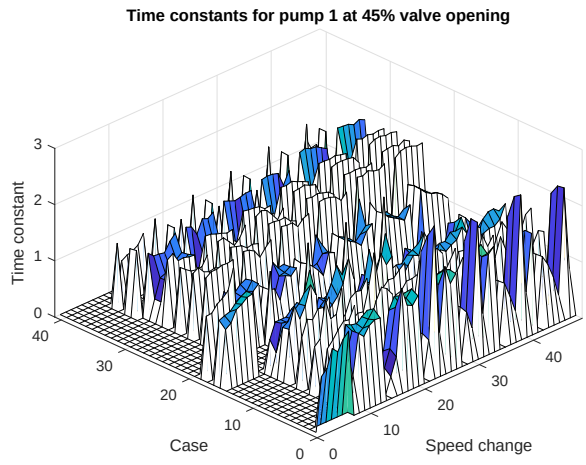
4.4 Analyzing data

4.4.1 Calculating system coefficients

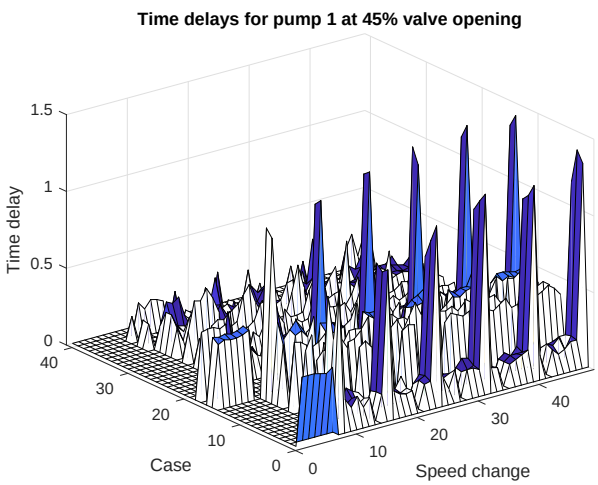
All stored transitional data was then put into a script to obtain DC gains and time constants for scenarios where it was possible to identify the transition. The full script can be seen in appendix chapter 9.2. A sample of this can be seen in figure 4.9.



(a) Gains plotted against case and speed change numbers at specific valve opening.



(b) Time constants plotted against case and speed change numbers at specific valve opening.



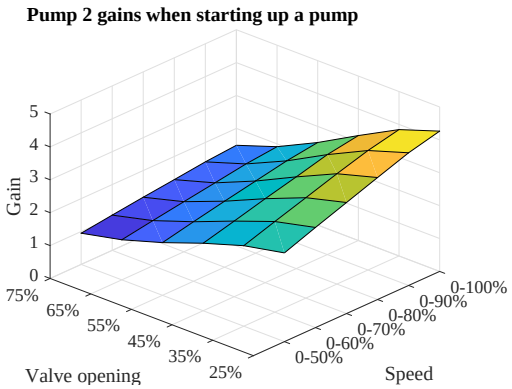
(c) Time delays plotted against case and speed change numbers at specific valve opening.

Figure 4.9: Obtained values for gains 4.9a, time constants 4.9b and time delays 4.9c, based on case and speed change index.

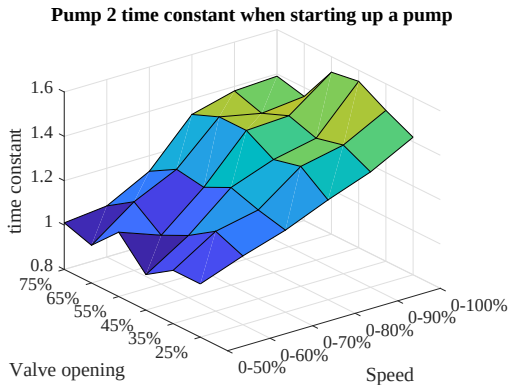
We had decided to investigate values only where it mattered. For example, DC gain is only dependent on the steady state value, so it does not matter which input speed was used at the beginning of the transition, but only the end point. Moreover, looking at transitions such as 2.2, a hypothesis was formed that a transition can be closely represented as a sum of two first order transfer functions - one showing pump's performance because of input voltage and another one caused due to coupling with other pumps. As such, when investigating pump's performance with one or more additional pumps running, we calculated only the gain from coupling, as the pump's individual performance is then represented by another transfer function while it is running alone. Similarly for time constant, we are only interested in the rise time of a pump either while it is running alone, or when coupled with other pumps but not having its own input voltage changed. As calculating time constant requires knowing the time delay in the system, these values have also been investigated.

4.4.2 Coefficient sorting

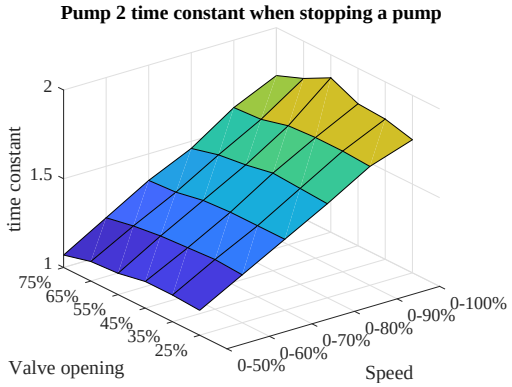
We then started looking for patterns in the data, searching for a way to generalize the values describing the transfer functions so it would be possible to easily represent them for a wide range of inputs. It was observed that the order of pumps did not matter - if a pump is running and then another pump is started, the coupling is independent of which pump is started up. Moreover, the gain is independent whether we start up or shut off pumps, as it is related to the steady state value. Time constant, on the other hand, is different when we start up a pump than it is when we turn it off. Finally, due to differences between pumps themselves, each pump has had the coefficients derived based on its performance with respect to other pumps. An example of sorted gains and time constants, as well as time delays can be seen in figure 4.10, where it is shown how the values change when input speed and valve opening are altered.



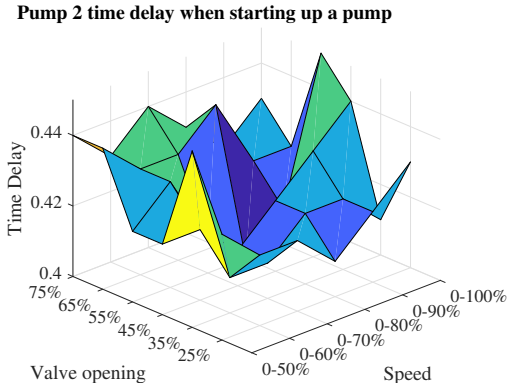
(a) Example of sorted gains.



(b) Example of sorted time constants with pumps starting up.



(c) Example of sorted time constants with pumps shutting down.



(d) Example of sorted time delays.

Figure 4.10: Sorting pattern examples for DC gain 4.10a, time constant when powering up coupled pump(s) 4.10b and when shutting down 4.10c and time delays 4.10d

We can see that overall, values can be fitted to a curve to extrapolate coefficients at different speed or valve opening inputs.

4.4.3 Patterns

In total, there were 6 patterns found for the DC gain, 8 for time constants and 3 for time delays. The patterns found are described in tables 4.3, 4.4 and 4.5 for DC gain, time constant and time delay, respectively.

Pattern	DC Gain description
1	Pump starting up
2	Pump running, starting up another one
3	Two pumps running, starting up last one
4	Pump running, starting up two pumps
5	Pump starting up with one already running
6	Pump starting up with two already running

Table 4.3: Different sorting patterns for DC gain.

Pattern	Time constant description
1	Pump starting up
2	Pump running, starting up another one
3	Pump running, starting up two
4	Starting pump with another running
5	Starting pump with two running
6	Starting two pumps with one running
7	Two pumps running, starting up last one
8	One pump running, changes speeds

Table 4.4: Different sorting patterns for time constant.

Pattern	Time delay description
1	Pump starting up/shutting down
2	Pump running
3	Running pump coupled with a new pump with no speed change

Table 4.5: Different sorting patterns for time delay.

With these patterns, we now know how DC gain and time constant changes for each pump with respect to input voltage and choke valve opening during specific transitions. The time delay was found to be nearly constant in first two cases described in table 4.5, being independent of valve opening or speed. It allows us to model both individual pump's performance and the coupling of the pumps. The values will later be used in a simulation in order to test the validity of these findings. The data will be fit to a first order polynomial in terms of speed, with fixed valve opening, in order to prove our hypothesis first.

4.5 Coupling

Because of coupling, changing one pump's speed will influence the other pump's head. With our initial efficiency algorithm implementation, we saw spikes in pressure during transitions, which can produce over or under-pressuring, potentially

damaging the system. A possible solution was proposed to decouple the pumps, to remove the pumps effects from each other. The idea is to make a cross decoupler function that subtracts the effect from one pump by calculating how much the input in G_{p12} will decrease the need for input in G_{p11}. A block diagram of the decoupled system for two pumps can be seen in figure 4.11, where G_{p11} is first pump's transfer function, G_{p22} is second pump's transfer function, T_{12} and T_{21} are the two cross decoupler functions and G_{p12} and G_{p21} are the transfer functions describing the coupling effect between them. G_{c1} and G_{c2} are the two controllers for the pumps.

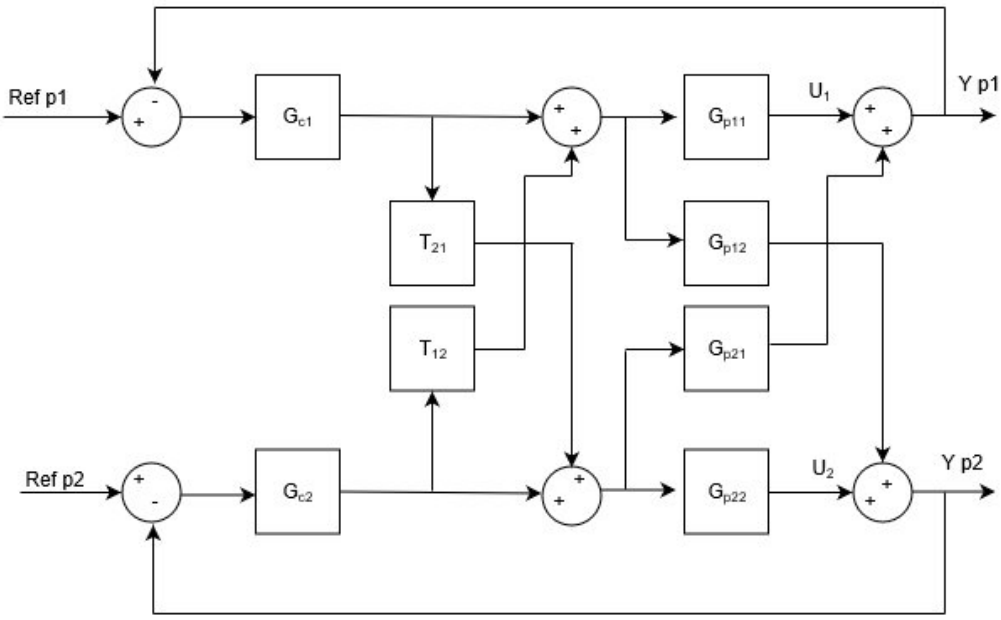


Figure 4.11: Block diagram for decoupled system.

We already know that the system describing pump's performance - either on its own or coupling part - varies with operating conditions. Therefore, the decoupler functions will be modeled as first order systems in the form seen in equation 4.2.

$$G(s) = \frac{K}{\tau s + 1} \tag{4.2}$$

The two decoupling functions, seen in equations 4.3 and 4.4, then subtract the head produced by G_{p12} and G_{p21}.

$$T_{12} = -\frac{G_{p12}}{G_{p11}} \tag{4.3}$$

$$T_{21} = -\frac{G_{p21}}{G_{p22}} \tag{4.4}$$

The procedure could be expanded for three pump system, requiring additional transfer functions describing each pump's performance, as well as extra cross decoupler for

each pump. For simplification purposes, the testing will be done with a two-pump system only.

4.6 Chapter conclusion

In this chapter, we have identified pumps' characteristics in order to obtain pump curves necessary for efficiency algorithm. We have also looked at transient modeling - we have obtained experimental data showing transitions with various changes in operating conditions. From that, we have derived patterns for DC gain, time constant and time delay. The obtained data can then be used in a cross decoupling system, to decouple the pumps effects on each other, as well as modeling of the transient periods themselves.

Chapter 5

Implementation

5.1 Efficiency algorithm and parameter estimator

5.1.1 Algorithms' logic

The efficiency algorithm will be implemented using the mathematical model derived from the experimental data. The coefficients will be loaded into memory and used to calculate the efficiency value for each combination at a speed, required to produce the desired head. The parameter estimator will attempt to determine choke valve percentage when a change in it occurs. The block diagram of this system can be seen in figure 5.1.

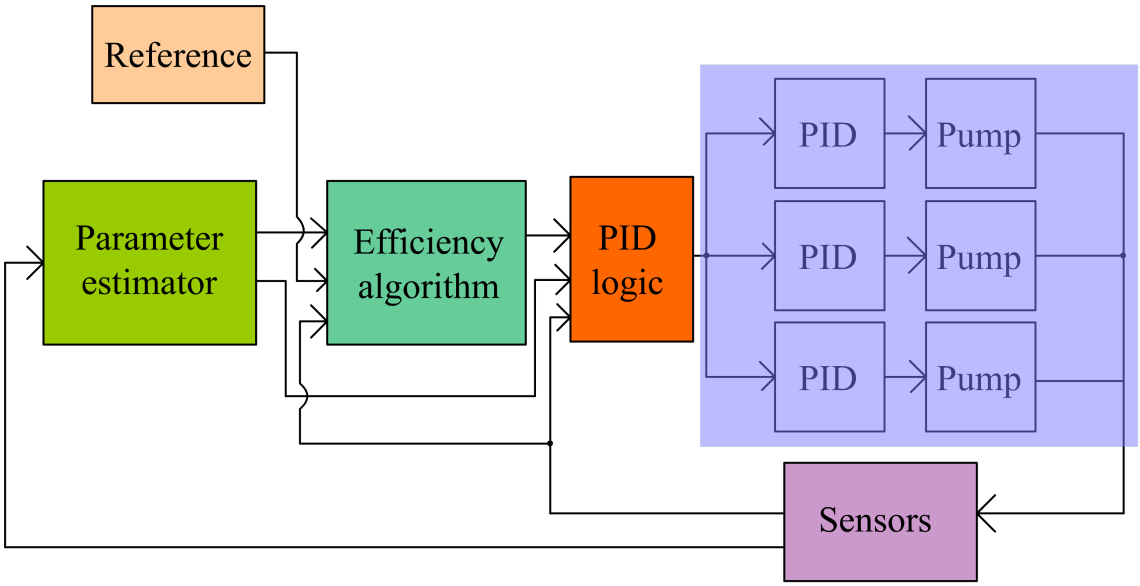


Figure 5.1: Block diagram of the system.

The implemented system will follow the logic seen in figure 5.2.

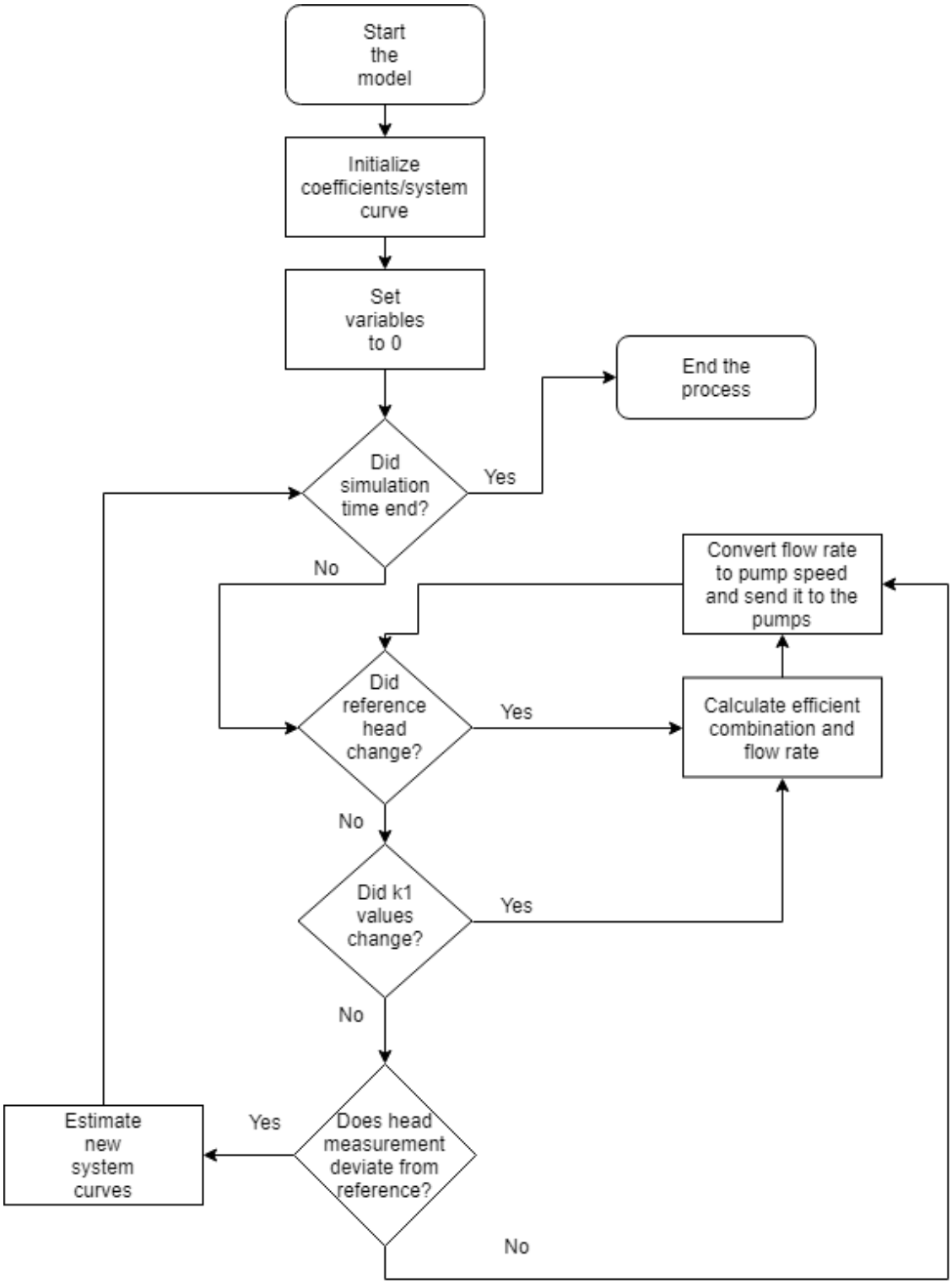


Figure 5.2: Flowchart of implementation logic.

5.1.2 Optimization equations

The logic of efficiency algorithm remains the same as in previous project. However, the code has undergone some changes to accommodate for the alteration of data and methods used to derive the coefficients. The idea behind the efficiency algorithm is to load the coefficients into the memory of the Simulink model, then allow it to pick the most optimal pump combination and pump speeds based on desired head and choke valve percentage. Knowing desired head H, we can find what flow rate would each pump combination would produce with current system curve [6]:

$$Q^{\star} = \sqrt{\frac{H\star}{k1}} \tag{5.1}$$

Knowing the flow rate for each combination, we can solve the following equation to find the pump speeds needed to produce the desired head [6]:

$$H(\omega) = a_0\omega^2 + a_1\omega Q(\omega) + a_2(Q(\omega))^2 \tag{5.2}$$

With speed also known, we can then find power required for each combination [6]:

$$P(\omega) = p_0\omega^3 + p_1\omega^2 Q(\omega) + p_2\omega(Q(\omega))^2 + p_3(Q(\omega))^3 \tag{5.3}$$

q Finally, having all other variables, we can compute each combination's hydraulic efficiency [6]:

$$\eta_i = \frac{\rho g(a_0^i\omega_i^2 + a_1^i\omega_i Q_i(\omega_i) + a_2^i Q_i(\omega_i))^2)Q_i}{p_0^i\omega_i^3 + p_1^i\omega_i^2 Q_i(\omega_i) + p_2^i\omega_i(Q_i(\omega_i))^2 + p_3^i(Q_i(\omega_i))^3} \times 100\% \tag{5.4}$$

The algorithm then picks the combination with highest efficiency and outputs the solution's pump speeds. Parts of the algorithm's code can be seen in appendix chapters 9.3.1 and 9.3.2.

5.1.3 Parameter estimator

The parameter estimator is another algorithm, which triggers recalculation of system curve if the head measurement deviates from the expected head by at least 2 meters. It then uses current head and flow rate to compute the k_1 value that the system curve should have under new operating conditions. The value is then compared with known system curves to determine approximate valve opening percentage. This percentage is then translated into each combination's new k_1 value, again using known data. Part of parameter estimator's code can be seen in appendix chapter 9.3.3.

5.2 Transient modeling

5.2.1 Block diagram

A block diagram of the transition simulation model can be seen in figure 5.3.

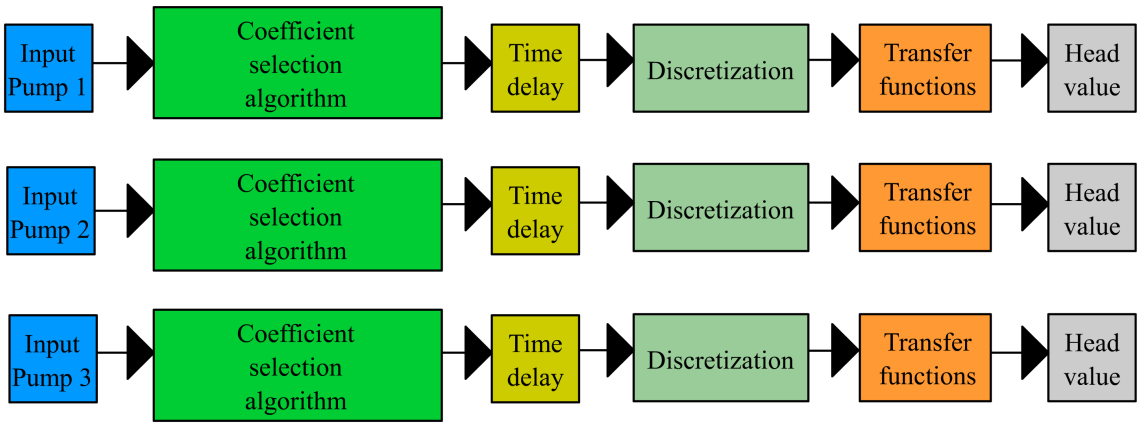


Figure 5.3: Block diagram of simulation model.

5.2.2 Simulation model

An algorithm, which picks the transient based on previous and current input speeds, using first order system coefficients derived in previous chapter, was also designed. The algorithm can know which pattern of DC gain and time constant changes best describe particular pump's performance when the inputs change. A sample of such algorithm can be seen in appendix chapter 9.3.4.

As input speeds change, the algorithm points to the right values in the curve fitting coefficient matrices, which are then used in combination with the pump speed. The output is the DC gain and time constant values for both transfer functions of a particular pump in a specific transition, with appropriate time delay. The values are then passed through a time delay, in accordance to the delay in that particular transition. Through testing, it was discovered that a relatively small time varying time delay was keeping the system causal due to the nature of step delay block, thus no issues had arisen. The new transfer function was then discretized using zero order hold method with a sampling time of 0.01s, to assure that the step of the system is as close to the real one as possible. Finally, the discretized system's values are then fed into a time varying discrete function block, which produces the outputs for both the system when pump changes speeds and the system produced due to coupling in the pumps.

5.3 Decoupling model

The logic for the decoupled model can be seen in figure 5.4. To simplify the implementation, only a few possible scenarios will be simulated and inputs as well as the transfer functions will be pre-calculated. The model will serve as a proof of concept rather than a model doing all the calculations and estimations online.

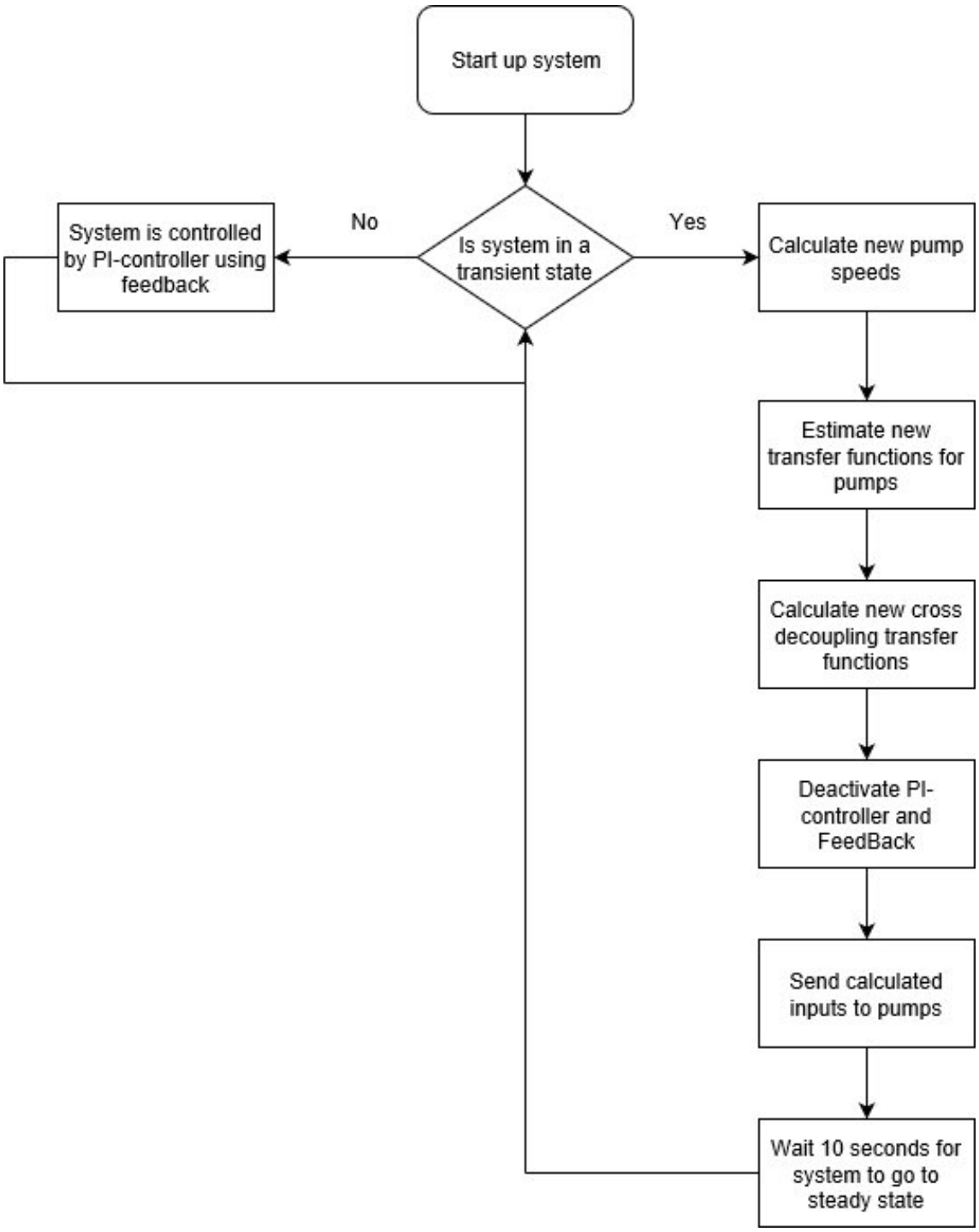


Figure 5.4: Flowchart of implementation logic.

The Simulink model used for the simulation can be seen in figure 5.5. A two pump system simulation was made to test decoupler's performance. Each pump has a transfer function describing the pump's individual performance, contribution coming from the other pump and a decoupler function that separates the two.

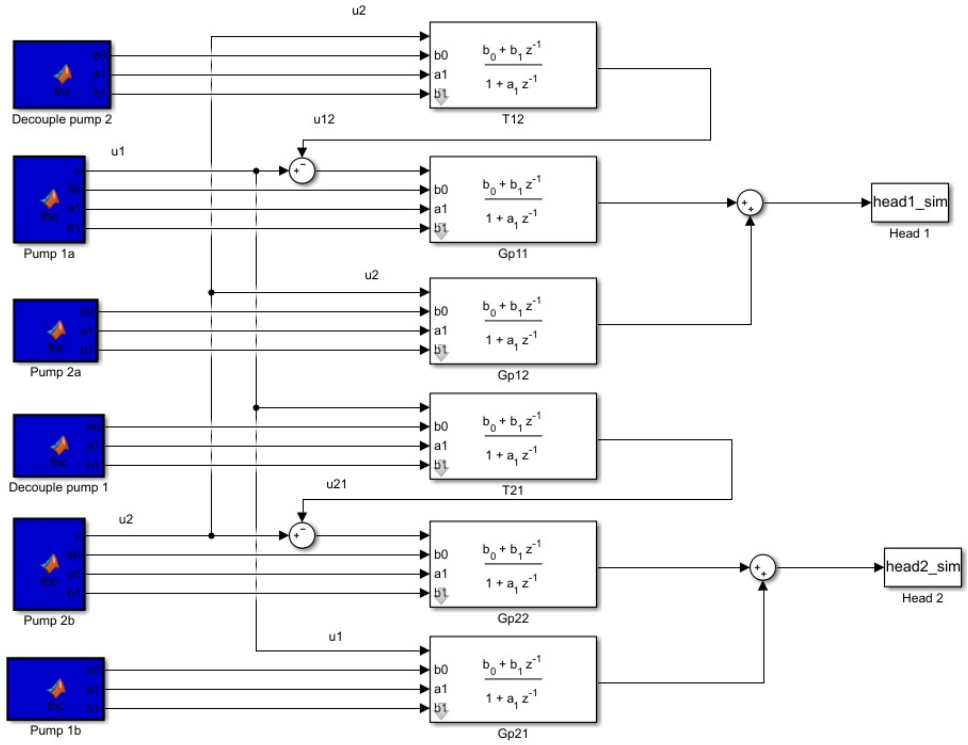


Figure 5.5: Decoupling model.

5.4 Chapter conclusion

The three methods have been implemented in different environments - the efficiency optimization algorithm and accompanying parameter estimator will be tested on a real life platform. Due to limitations with the software on the platform, we are unable to test the transient modeling system and the decoupling model. These two methods will be tested in offline simulations to at least validate method themselves in a best-case scenario.

Chapter 6

Results

6.1 Testing efficiency algorithm

The efficiency algorithm and the parameter estimator were tested on a real life platform in a 1445 seconds long test. The idea was to test a condensed version of pump's daily operation, where requirements can change every hour and see if the algorithms can still keep up with the references at the end of the simulation. Thus, the test makes change in requirements every minute. The algorithms will attempt to follow the reference head, which changes every minute, while the pressure valve is changed every 6 seconds to modify the system curve. The model will not know the real valve opening percentage past the first step and will have to estimate it based on pre-calculated data and real time measurements. Based on figure 4.4, we expect the optimal combination, based on efficiency, to change with changes in pressure valve. Data will be gathered for head, flow, power, input speeds as well as estimated valve opening percentage to compare with expected data. Finally, the algorithm's performance in this test will be compared with experiments without efficiency algorithm on, where one, two or three pumps will be in operation for the entire experiment. Figure 6.1 shows that once steady state is reached, the parameter estimator can determine approximate valve opening percentage. However, the estimation is not entirely accurate, causing the efficiency with the algorithms to be slightly below that of a system without optimization. On the other hand, when pressure valve percentage is changed, the efficiency algorithm jumps to the correct pump combination to achieve the most optimal pump operation schedule with the given requirements. The estimated valve opening percentage is within 2% error margin, second and third step.

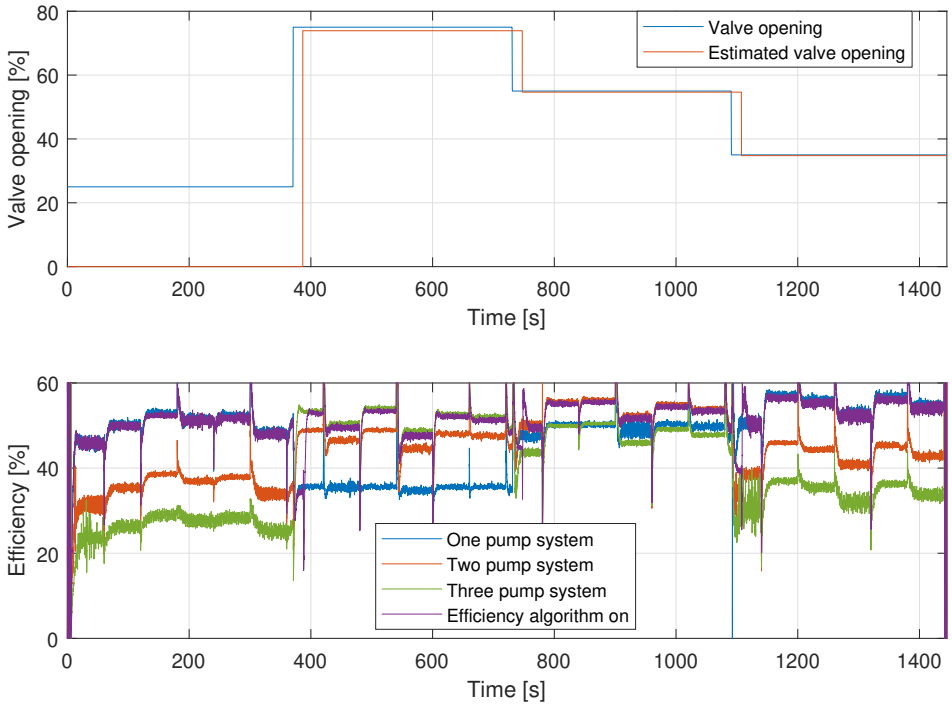


Figure 6.1: Estimated and real pressure valve percentage; Efficiency with algorithms and without them.

We see that the efficiency is also changing with changes in reference head, which is modified every minute. We can then evaluate the algorithms' performance in terms of efficiency by comparing the mean efficiency percentages throughout entire experiment for all four tests. After removing spikes in efficiency that were above 60% or below 0% - which were caused due to an oversight in the code - we see that the test with algorithms in place had the highest efficiency throughout the entire experiment period:

$$\eta_{optimized} = 51.4021\% \tag{6.1}$$

$$\eta_{1pump} = 47.2830\% \tag{6.2}$$

$$\eta_{2pumps} = 44.7365\% \tag{6.3}$$

$$\eta_{3pumps} = 39.9409\% \tag{6.4}$$

Both the head and the flow are following the reference head and expected flow, respectively, in steady state, as seen in figure 6.2. However, changes in operating conditions causes the PI controller to overshoot the system, while valve opening changes require system to reach steady state before any estimations can be done.

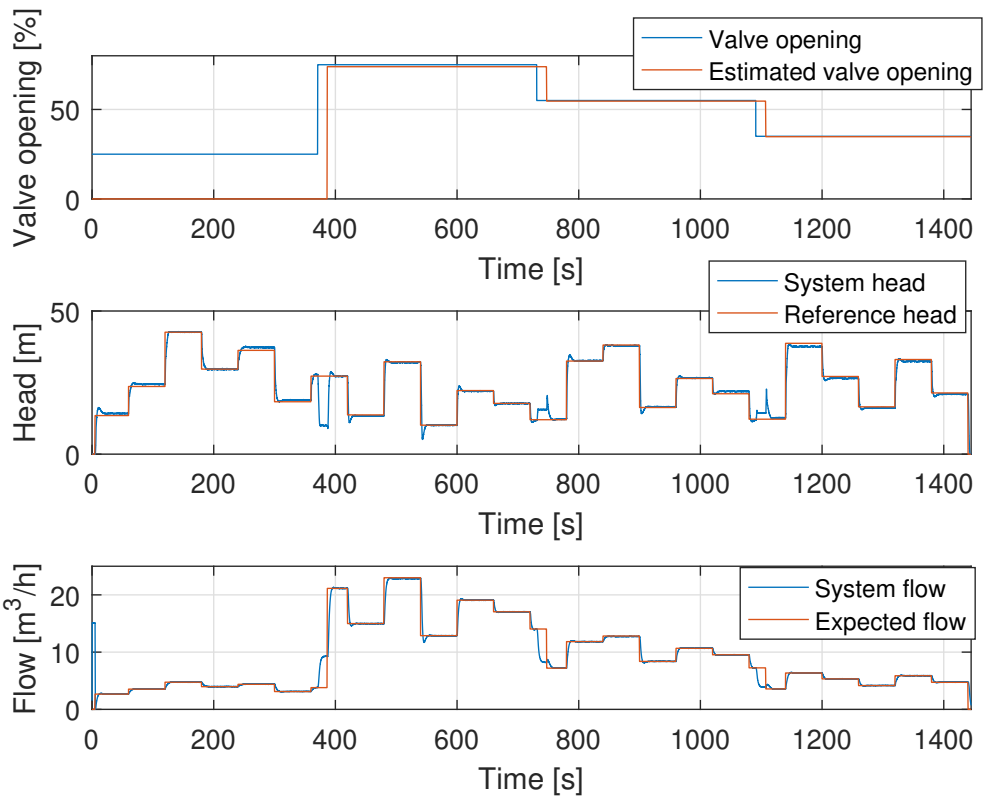


Figure 6.2: Real and estimated valve opening; Reference head and measured head; Expected flow and measured flow.

The head was found to be within 1.3 meters of error margin, apart for transition periods, showing that the steady state system is accurate as long as the estimated system curve is correct. Finally, the estimated pump speeds were compared to the actual pump speeds, as seen in 6.3.

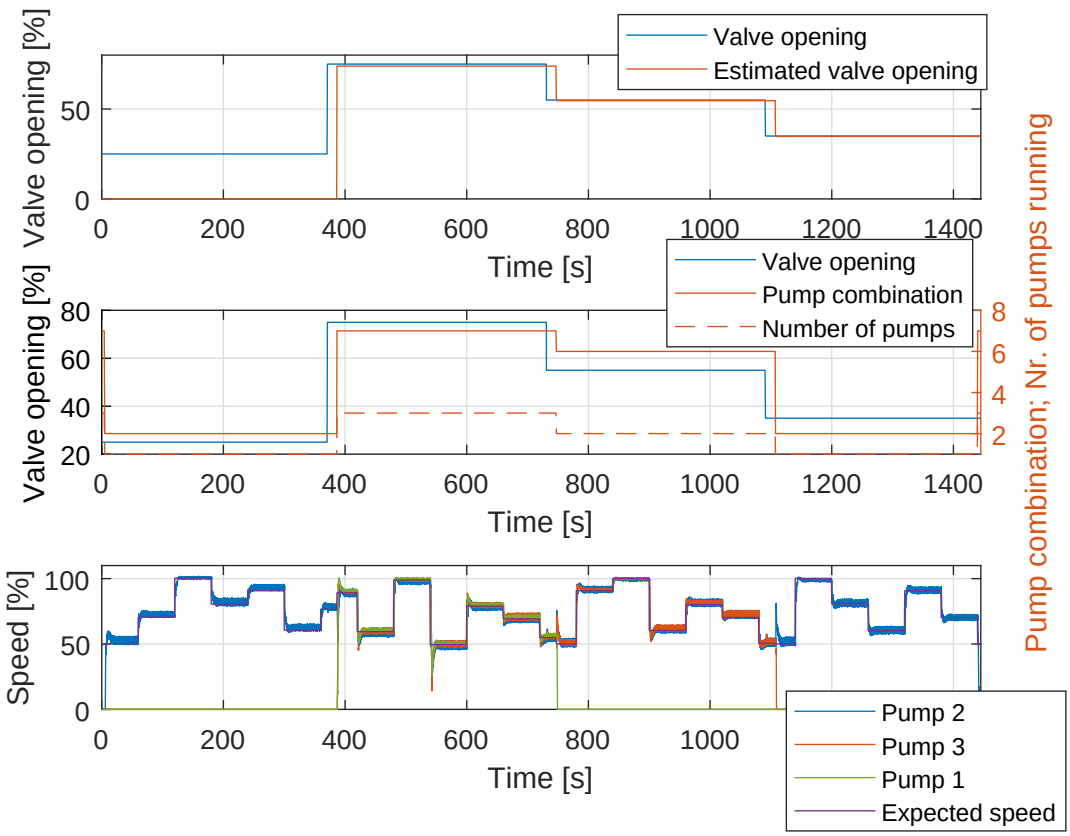


Figure 6.3: Real and estimated valve opening; Pump combination and number of pump change with respect to valve opening; Expected and measured speed.

Although the pumps are thought to be identical, in order to simplify the calculations, in reality their performance differs. Each pump requires different amounts of speed to produce the desired flow and pressure. While the mathematical model assumes that the pumps are identical, the PI controllers make necessary adjustments to ensure that each pump is able to produce the required part of the products.
A benefit of using multiple pump system with efficiency algorithm is that it allows the system to meet the increased requirements in both head and flow. In figure 6.4, we see that a single pump is unable to meet the demands at certain valve opening percentages, because a single pump, even pushed to maximum speed, cannot produce the desired head level. As a result, the flow rate is unable to keep up with the expected values as well, resulting in low efficiency at those operating conditions

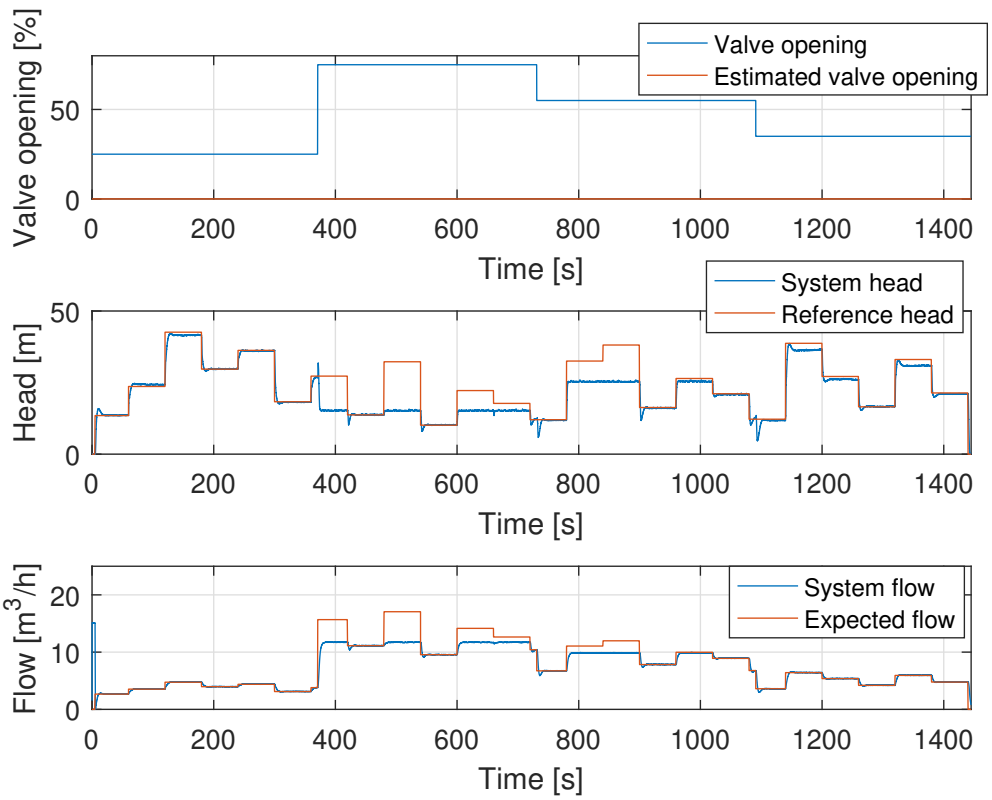


Figure 6.4: Single pump's head and flow during the previous experiment.

Two additional experiments were done with different requirements in head as well as different changes in pressure valve percentage, in order to ensure that the algorithms' optimized solutions were not a fluke. The results can be seen in appendix chapter 9.3.5. There, the optimization algorithm still performs as expected, picking the best solution it can, in terms of hydraulic efficiency, based on the data it has. At one point, the parameter estimator returns valve opening estimation with about 2% error, causing the efficient combination to be incorrect, because based on the data the algorithm has, it estimates a different pump - which has efficiency close to the real optimized solution - to be the best solution. So, while the algorithm is heavily dependent on the parameter estimator to approximate the system curve accurately, even with errors it can choose a solution close to the real optimized operating condition. Moreover, the parameter estimator is able to adjust its mistake over time, as the measured pressure deviates from the reference one, eliminating the error in the long run.

6.2 Simulating system transitions

Two tests were programmed to simulate and validate the transition model. The first test was done by simulating a pump test at 25% valve opening, where initially one pump starts up at 50% speed. After 10 seconds, another pump is started up and

the pump speed is then changed to 100%. After another 10 seconds, the last pump is powered on. The simulation data is then compared with the real data obtained from transition tests and can be seen in figure 6.5. It is evident that although the overall idea of transition is followed, the real data does not fit to a first order system exactly. The steps are less smooth than those of the simulation and the steady state looks more similar to the second order performance than the first one, due to small oscillations. Regardless, it was found that the MSE between the simulation and real data is 3.6469 for pump 1, 3.5804 for pump 2 and 3.6 for pump 3, where the error primarily comes from the steps themselves.

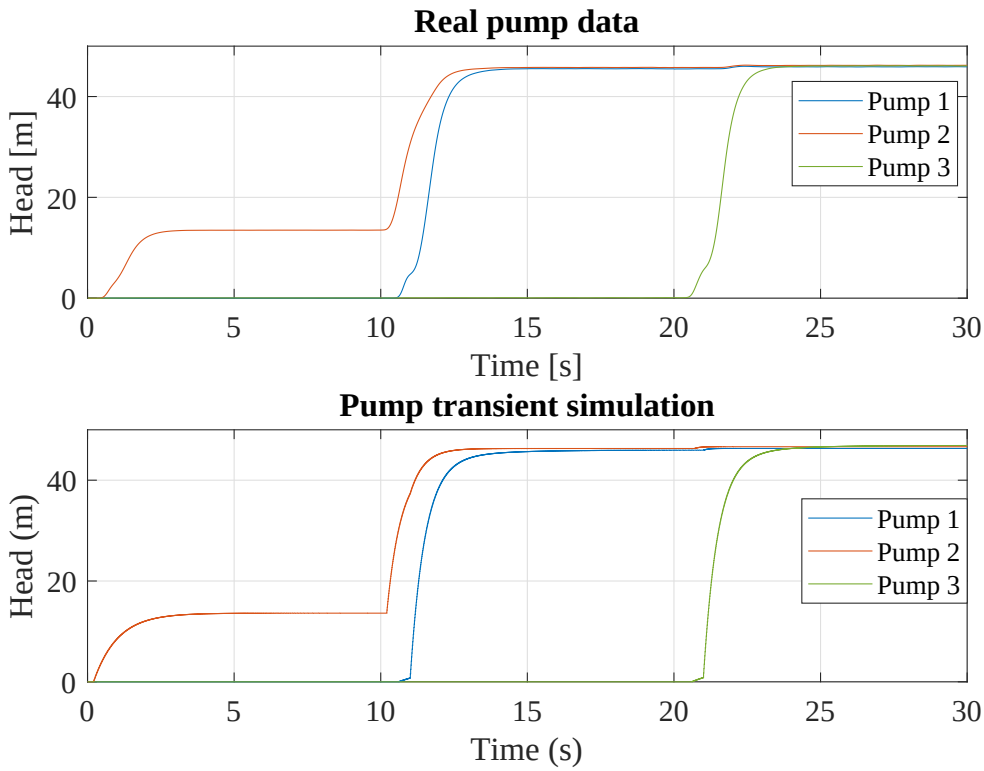


Figure 6.5: Comparison of real pump transient data and simulation of the scenario.

Another simulation was done with coefficients derived at 75% pressure valve opening. This time, at the beginning of the transition, a pump was running at 80% speed. During the transition, this speed was reduced by 60%, which then decreased the performance of the running pump. However, another pump was also started up, running at the same 60% speed at the end. Both the real data and the simulation result can be seen in figure 6.6. As before, we see that the simulation is able to approximately reproduce the transition, but the curves are too sharp in simulations. The speed change affects the running pump first, causing its head measurement to drop. After about a second, the effects of the new pump start to kick in, and coupling causes both pumps heads to go up. The steady state error is still minimal and the MSE for the two pumps is 0.4243 and 1.9476, showing that the model is still relatively valid.

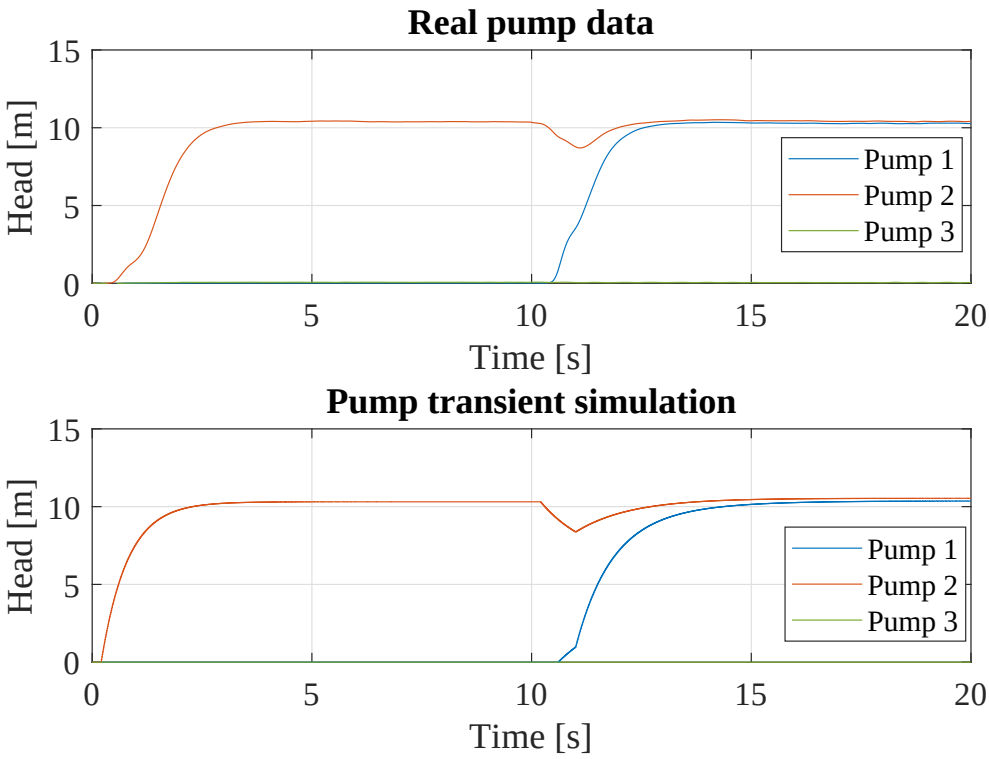


Figure 6.6: Comparison of real pump transient data and simulation of the scenario.

6.3 Testing decoupling model

A test was done by simulating the start up of first pump with input needed to produce 25 meters of head. At 40 second mark, second pump is started up with same input. reference head 25 meters after 20 seconds. Then after 40 seconds pump 2 is starting up with same reference head. In figure 6.7, we see that the two pumps are being decoupled when the second one starts up. After 60 seconds into the simulation, the input is changed to produce 10 meters of head. At 80 seconds of simulation, second pump is turned off. The simulation was done without varying the transfer functions, in order to validate the decoupling itself first. Based on simulation results, we can see that the decoupler is indeed capable of removing the additional head in the first pump.

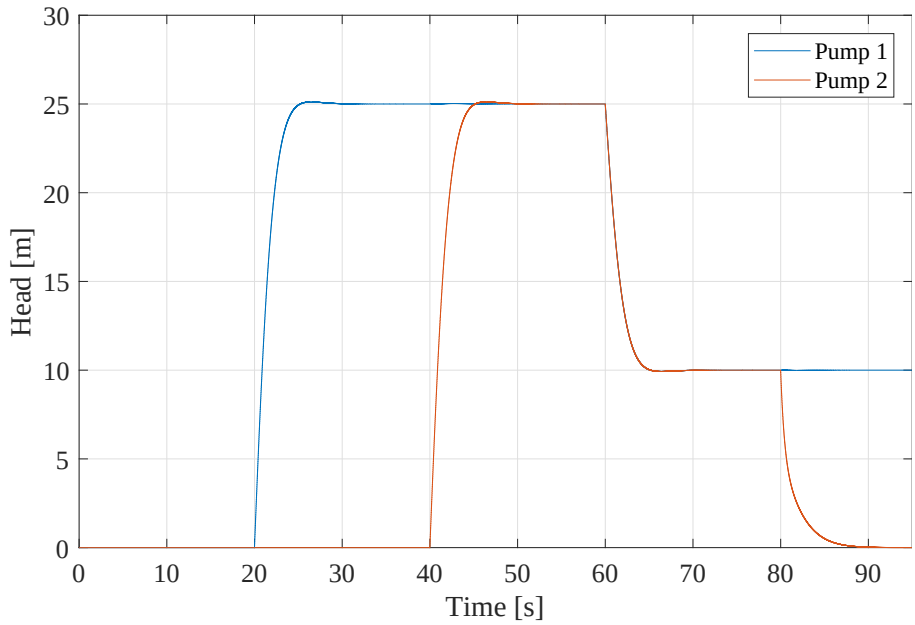


Figure 6.7: Decoupling two pumps.

For comparison, a similar simulation was done for the model, where decoupling feature was replaced with PI controllers and a feedback loop. Figure 6.8 shows that first pump has a sudden increase in pressure when the second pump is started up/shut down.

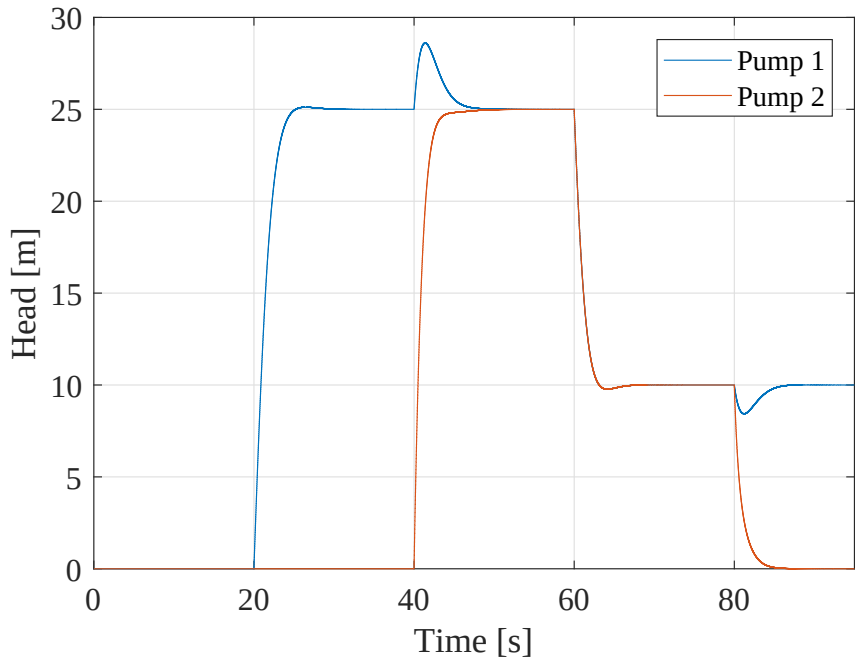


Figure 6.8: Simulation with PI controller instead of decoupler.

Since we expect the transfer functions to vary with time, a simulation was then setup

to test decoupler's functionality with time-varying transfer functions. In figure 6.9, we see that when the second pump is introduced, the decoupler still works. When the input speed changes, the transfer functions are modified as well, but decoupling function is able to keep up. However, when the second pump is stopped, the first pump's head encounters a small spike before settling down to the right steady state value.

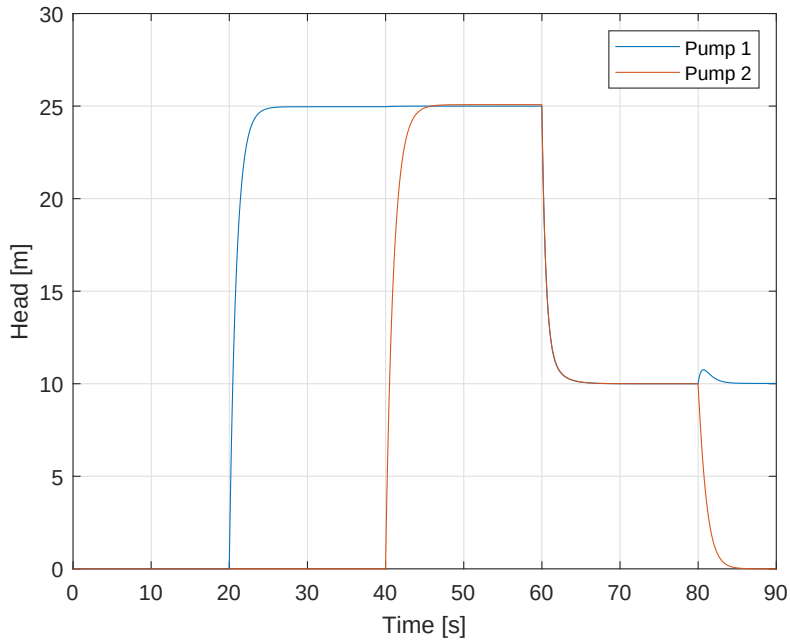


Figure 6.9: Decoupling two pumps.

The spike in head at towards the end of the simulation is thought to be due to changes in the transfer functions T_{12} and G_{p12} but not their initial conditions, causing the apparent overshoot, although the real cause was never found, despite our attempts to change how the transfer functions are handled.

6.4 Chapter conclusion

The designed methods have been implemented in either simulations or real life system. We can see positive results, meaning that the methods perform the expected operations. There are errors in the values, where the simulations do not match the real life data or expected results perfectly. However, given the amount of approximations and curve fittings done to derive the models, it is an acceptable trade-off. On the other hand, we see that our hypothesis, that the transitions can be represented as a sum of two transfer functions, is holding up.

Chapter 7

Discussion

7.1 Current performance

From the three implemented methods, we see that there are some errors in values in both simulations and real life tests. However, with algorithms using only approximations and curve fitted values, the accuracy of the methods is dependent on the coefficients themselves. With more data, as well as higher order polynomials for curve fitting, it would be possible to reduce the errors. Nevertheless, the efficiency algorithm is always capable to select the most efficient pump combination in terms of hydraulic efficiency, especially when the valve opening changes. The pressure in the system always stays within 2 meter margin of error in steady state, based on threshold for parameter estimator trigger. The parameter estimator is also accurate when attempting to estimate the choke valve percentage, being within 2% of error margin. If the pressure sensor on the valve is faulty, the system can still keep track of the valve opening percentage, producing correct system curves. The algorithm is also capable of providing reliable pump speeds, as the steady state head of the system closely follows the desired head level. The measured values are inaccurate during the transitions, though, since the system only knows the expected steady state values and not the time it takes to get there or the shape of the step curve. To help alleviate this problem, both a model of transitions and a decoupling system were designed and tested in simulations. Although modeling of transient period was only tested in few specific scenarios, each one of them was able to approximately represent how a real system would behave when changing both the number of pumps in operation and their speeds. The steady state is again close to the real value, as was the case with efficiency algorithm, staying within around 0.5 meter margin of error this time. The shape of the step curve, however, appears more steep in simulations than it is in real life, as was seen in figures 6.5 and 6.6. As we saw through numerous transition tests, though, when another pump is introduced or shut off, both its curve and its effect on other pumps is not just a first order system. In some cases, seemingly random physical phenomena can cause the head of a pump to briefly go down despite pump speed demanding otherwise. Similarly, the decoupling of pumps is as accurate as the data used to design it is. While it is able to separate

effects of one pump on another, it cannot produce precise results due to transitions being more complicated than a first order system. Regardless, both methods show potential, giving us a better idea of how the head of pumps change when operating conditions are altered. On the other hand, either one of the methods require immense amounts of data to understand when and how the transfer function coefficients should change. While patterns are fairly easy to recognize for both DC gains and time constants in a three pump system, the data required for a proper analysis would scale with the number of pumps. Additionally, the tests were only done under the assumption that each step the system would only either turn on or shut off a pump. If it were to allow to do both at the same step, understanding and modeling the changes of head another measurements of pumps and their effects on each other would be more complicated.

7.2 Future work

Although the idea of the methods works, the results only approximately represent the real data. Several ideas have been considered that could improve the overall performance of the system in the future. Having more test data to derive coefficients for efficiency algorithm and transitional transfer functions would help in some cases. As we have seen, the time constants are more scattered when pump is starting up than they are when a pump is shutting down. In such scenarios, more data could provide a better idea of how the mean values look like. However, using more data alone would still result in the process being inefficient. Utilizing machine learning, it would be possible to train the computer to recognize the patterns for changes in DC gain and time constant. With all inputs fed into the learning algorithm, the computer could spend more time understanding how the values change when pump speeds, pump combinations and choke valve percentages change the measurements, potentially providing better results. The errors could also be minimized through the use of controllers. However, given the usage of time-varying transfer functions, the controller coefficients would also have to vary based on operating conditions. While a PI controller was able to help efficiency algorithm keep pump speeds at right values for each pump, the controller produced overshoot in some cases due to changes in operating conditions, as was shown in figure 6.1. With gain and time constant being dependent on input speeds, choke valve and the combination the system is running at, so too should the controller values change, in order to fit more precisely to expected steady state value, settling time as well as to remove the overshoot. Finally, all of this would still need to be implemented and tested on a real life system in real time. The time varying transfer function and a controller would also rely on a time varying time delay. In order to ensure that the system remains causal, the memory of time delay block would have to be managed and sorted by another algorithm that would ensure input data is not being overwritten when the time delay increases or decreases. Furthermore, all of modeling and coupling was done for a single channel of data - the head of the pumps. The methods would have to be re-used for both flow rate and power to make it possible to evaluate the system's efficiency and overall performance during the transition.

7.3 Chapter conclusion

In the simulated scenarios, the methods are capable of achieving desired results. Through the use of machine learning, in combination with more data, it would be possible to reduce the errors further as well as gain better understanding of how the transitions work. Implementation of the system would also require care, as a time varying controller would be needed to control the system through feedback, while a time varying delay could cause system to be non-causal in real time without additional algorithms.

Chapter 8

Conclusion

Initially, we set out to make changes in the efficiency algorithm we have implemented in a previous project. We have tested the changed algorithms on a real life platform and have observed improvements in performance. The algorithm now chose the most efficient combination at every step, because of the increased accuracy of input speed and efficiency calculations. We then investigated the transient state of the system by obtaining large amounts of experimental data from transitions in the system with various inputs. We have observed how the pressure changes based on number of pumps in operation, their speed and choke valve percentage. We then set out to test our hypothesis, that a transient step of a coupled pump system can be represented by a sum of two transfer functions. A simulation model was made to test two scenarios where pumps are assumed to be running at certain initial speed and choke valve percentage. The system is later assumed to turn on one or more pumps, while also altering the speed. Model's results were then compared with experimental data from same transitions and was found to be an approximate representation of the transitions. The steady state error was found to be minimal, the settling time was similar. However, the step curve in simulations was too smooth to properly represent the transitions and a higher order system could potentially be needed. Similar approach was then tested in a decoupling model, which attempted to decouple one pump's effects from another. The decoupling system was able to decouple the two pumps when their transfer functions were not affected directly. However, when the transfer functions are derived based on changes in speed, the decoupler produces small overshoot when coupled pump is powered down. Moreover, while both methods provide good results in some aspects and a decent starting point in modeling of the transient state, neither model was tested on a real life system due to time and software limitations. Nevertheless, groundwork for improvements was laid and further improvements have been suggested. To sum up, it can be said that the proposed methods show promise in improving the efficiency of water pump systems, but the subject could use more work.

Bibliography

[1] Ioan Sarbu. A study of energy optimisation of urban water distribution systems using potential elements. page 593, 2016.

[2] Muhammad Wakeel and et al. Energy consumption for water use cycles in different countries: A review. In *Applied Energy*, pages 868–885, 2016.

[3] Grundfos. The surprising truth about pumps and sustainability. https://www.grundfos.com/water-energy/meet-the-water-and-energy-challenge-now/facts-about-pumps-and-sustainability.html.

[4] Danfoss. Water and energy loss put pressure on the water industry. https://www.danfoss.com/en/markets/water-and-wastewater/dcs/water-pumps-and-pressure/.

[5] Kasper L. Jepsen, Leif Hansen, Christian Mai, and Zhenyu Yang. Power consumption optimization for multiple parallel centrifugal pumps. In *Control Technology and Applications (CCTA), 2017 IEEE Conference on*, pages 806–811, 2017.

[6] Zhenyu Yang and Hakon Børsting. Optimal scheduling and control of a multi-pump boosting system. In *Control Applications (CCA), 2010 IEEE International Conference on*, pages 2071–2076, 2010.

[7] Xiangtao Zhuan and Xiaohua Xia. Development of efficient model predictive control strategy for cost-optimal operation of a water pumping station. In *IEEE transactions on control systems technology, vol. 21, No. 4 July 2013*, pages 1449–1454, 2013.

[8] Juha Viholainen and et al. Energy efficient control strategy for variable-speed driven parallel pumping systems. In *Springer Energy Efficiency Journal*, 2013.

[9] M.F.K. Pasha and K. Lansey. Optimal pump scheduling by linear programming. In *World Environmental and Water Resources Congress*, pages 395–404, 2009.

[10] Allan Gjerlevsen and Andrius Kulsinskas. Efficiency Optimization of a Multiple Variable-Speed Water Pump System, 2018.

[11] U.S. Department of Energy. Optimize parallel pumping systems. https://www.energy.gov/sites/prod/files/2014/05/f16/optimize_parallel_pumping.pdf, 2006.

[12] C. Ramadevi1 and V. Vijayan. Design of decoupled pi controller for quadruple tank system. In *International Journal of Science and Research (IJSR)*, pages 318–323, 2014.

Chapter 9

Appendix

9.1 Data storing script

```
clear all
coder.extrinsic('GetCaseNum');
load('filter');
%run nums for each valve opening and test (Yx6)
runnums =  [387 389 390 391 394 395 396 397 398 399 460 461 465 471 501
     483 489 495 501 502 550 556
             400 401 402 403 404 405 406 407 408 409 410 411 466 472 478
     484 490 496 496 503 551 557
             412 413 414 415 416 417 418 419 420 421 422 423 467 473 479
     485 491 497 497 504 552 558
             424 425 426 427 428 429 430 431 432 433 434 435 468 474 480
     486 492 498 498 505 553 559
             436 437 438 439 440 441 442 443 444 445 446 447 469 475 481
     487 493 499 499 506 554 560
             448 449 450 451 452 453 454 455 456 457 458 459 470 476 482
     488 494 500 500 507 555 561];
%valve opening index, 1 to 6
vo = 1;
%load the data
for i=1:length(runnums)
    run_num = runnums(vo,i);
    clear pump2_meas
    if ~exist('pump2_meas','var')
        ImportData2Workspace
    end
    for n=1:3
        %generate offsets
        offsetQ = mean(pump2_meas(1).data(1:500,n*7-7+6));
        offsetH = mean(pump2_meas(1).data(1:500,n*7-7+2));
        offsetP = mean(pump2_meas(1).data(1:500,n*7-7+4));
        %get data
        head(i,n).head = pump2_meas.data(:,n*7-7+2) - offsetH;
        flow(i,n).flow = pump2_meas.data(:,n*7-7+6) - offsetQ;
        power(i,n).power = pump2_meas.data(:,n*7-7+4) - offsetP;
        speed(i,n).speed = pump2_meas.data(:,22+n);
```

```
    end
end
%clear variables we're not using anymore
clear a filename folder n offsetH offsetP offsetQ pump2_meas run_num
    str2find version
a = 1; % filter denum
%declare empty matrices
head1_total = [];
head2_total = [];
head3_total = [];
flow1_total = [];
flow2_total = [];
flow3_total = [];
power1_total = [];
power2_total = [];
power3_total = [];
speed1_total = [];
speed2_total = [];
speed3_total = [];
%for each test run concatenate data in the empty matrices
for i=1:length(runnums)
    headnew = head(i,1).head;
    head1_total = [head1_total headnew'];
    headnew = head(i,2).head;
    head2_total = [head2_total headnew'];
    headnew = head(i,3).head;
    head3_total = [head3_total headnew'];
    flownew = flow(i,1).flow;
    flow1_total = [flow1_total flownew'];
    flownew = flow(i,2).flow;
    flow2_total = [flow2_total flownew'];
    flownew = flow(i,3).flow;
    flow3_total = [flow3_total flownew'];
    powernew = power(i,1).power;
    power1_total = [power1_total powernew'];
    powernew = power(i,2).power;
    power2_total = [power2_total powernew'];
    powernew = power(i,3).power;
    power3_total = [power3_total powernew'];
    speednew = speed(i,1).speed;
    speed1_total = [speed1_total speednew'];
    speednew = speed(i,2).speed;
    speed2_total = [speed2_total speednew'];
    speednew = speed(i,3).speed;
    speed3_total = [speed3_total speednew'];
end
%clear old variables
clear speednew flownew headnew powernew head flow power speed
%fill two matrices with zeros in a [case]x[speed_step] matrix
%data_sum - summed up data for each [case]x[speed_step] and the count
%for each case in last row
%data_div - each case but divided by the count number
for i=1:41
    for j=1:49
        data_sum(i,j).data = zeros(12,1000);
        data_sum(i,j).data(13,1) = 0;
```

```
        data_div(i,j).data = zeros(12,1000);
    end
end
%declare previous combination and speed
speed_last = [0 0 0];
%loop through every transient-sample in the data
for k=2:floor(length(speed1_total)/1000)
    %shift which sample it uses to get the speed value based on which files
    %we're looking at due to error
    if k < 2161
        z = 1;
    else
        z = 50;
    end
    %declare current speed value
    speed1 = speed1_total(1,k*1000-1000+z);
    speed2 = speed2_total(1,k*1000-1000+z);
    speed3 = speed3_total(1,k*1000-1000+z);
    %concatenate speed values
    speed_current = [speed1 speed2 speed3];
    %find current and previous speed in the system (since all pumps use
    %the same speed)
    speedprev = max(speed_last);
    speednew = max(speed_current);
    %find the column number based on current and previous speed
    if speedprev == 0
        if speednew == 0
            j = 1;
        else
            j = 1 + (speednew/10-4);
        end
    else
        if speednew == 0
            j = (speedprev/10-4)*7+1;
        else
            j = (speedprev/10-4)*7+1 + (speednew/10-4);
        end
    end
    %get case num from combination
    %but only if the speed wasn't 0->0
    %which can happen when we switch from one file to another
    if rem((k-1),180) ~= 0 || k == 3421 % throw out cases when we switch files
        %throw out weird speed changes that shouldn't happen
        if j ~= 1 && j ~= 50 && j ~= 51 && j ~= 4.1 && j ~= -26.3 && j ~= 24.5 && j ~= 21.5 && j ~= -25.3 && j ~= -21.8 && j ~= 39.6
            %get case number based on current and previous speed vectors
            i = GetCaseNum(speed_last, speed_current);
            if i ~= 0
                %sum the data in the matrix
                head1_filt = filtfilt(Num,a,head1_total(1,k*1000-1000:k*1000-1));
                head2_filt = filtfilt(Num,a,head2_total(1,k*1000-1000:k*1000-1));
```

```
                    head3_filt = filtfilt(Num,a,head3_total(1,k*1000-1000:k*1000-1));
                    data_sum(i,j).data(1,:) = data_sum(i,j).data(1,:) + flow1_total(1,k*1000-1000:k*1000-1);
                    data_sum(i,j).data(2,:) = data_sum(i,j).data(2,:) + flow2_total(1,k*1000-1000:k*1000-1);
                    data_sum(i,j).data(3,:) = data_sum(i,j).data(3,:) + flow3_total(1,k*1000-1000:k*1000-1);
                    data_sum(i,j).data(4,:) = data_sum(i,j).data(4,:) + head1_filt;
                    data_sum(i,j).data(5,:) = data_sum(i,j).data(5,:) + head2_filt;
                    data_sum(i,j).data(6,:) = data_sum(i,j).data(6,:) + head3_filt;
                    data_sum(i,j).data(7,:) = data_sum(i,j).data(7,:) + power1_total(1,k*1000-1000:k*1000-1);
                    data_sum(i,j).data(8,:) = data_sum(i,j).data(8,:) + power2_total(1,k*1000-1000:k*1000-1);
                    data_sum(i,j).data(9,:) = data_sum(i,j).data(9,:) + power3_total(1,k*1000-1000:k*1000-1);
                    data_sum(i,j).data(10,:) = data_sum(i,j).data(10,:) + speed1_total(1,k*1000-1000:k*1000-1);
                    data_sum(i,j).data(11,:) = data_sum(i,j).data(11,:) + speed2_total(1,k*1000-1000:k*1000-1);
                    data_sum(i,j).data(12,:) = data_sum(i,j).data(12,:) + speed3_total(1,k*1000-1000:k*1000-1);
                    data_sum(i,j).data(13,1) = data_sum(i,j).data(13,1) + 1;
                end
            end
        end
        %set last speed to the one we had in this sample frame
        speed_last = speed_current;
end
%loop through entire matrix and divide it based on number of occurances
for i=1:41
    for j=1:49
        for k=1:12
            if data_sum(i,j).data(13,1) ~= 0
                data_div(i,j).data(k,:) = data_sum(i,j).data(k,:)./data_sum(i,j).data(13,1);
            end
        end
    end
end
%clear all other variables
clear i j k head1_total head2_total head3_total flow1_total flow2_total flow3_total power1_total power2_total
clear power3_total speed1_total speed2_total speed3_total speednew speedprev speed_current speed_last
clear speed1 speed2 speed3 a head1_filt head2_filt head3_filt Num o z runnums
%store the data into a separate file, name it after the valve opening
temp = num2str(vo*10+15);
fileName = strcat('TransientData_',temp);
save(fileName,'data_sum','data_div');
```

```
clear vo fileName temp
```

Code Listing 9.1: Full script to load raw experimental data and store it based on cases and speed steps

9.2 Coefficient sorting script

```
%% setup
load('FullTransientData');
load('MapTau');
load('MapK');
p1gain = [1 2 3 4 6 8 10 12 13 14 15 16 17 19 20 21 22 23 25 27 29 31 32 33 34 35 36 38 39];
p2gain = [2 4 5 6 7 8 11 12 13 14 15 16 18 19 21 23 24 25 26 27 30 31 32 33 34 35 37 38 40];
p3gain = [3 4 7 8 9 10 11 12 13 14 15 17 18 19 22 23 26 27 28 29 30 31 32 33 34 36 37 38 41];
p1start = [1 6 8 10 12 13 16 17 19];
p2start = [2 4 5 11 13 15 16 18 19];
p3start = [3 4 7 8 9 14 17 18 19];
p1off = [20 25 27 29 31 34 35 36 38];
p2off = [21 23 24 30 31 33 35 37 38];
p3off = [22 23 26 27 28 32 36 37 38];
p1p1 = [14 15];
p2p1 = [12 14];
p3p1 = [12 15];
SpeedIndex = [9 17 25 33 41 49 10 11 12 13 14 16 18 19 20 21 23 24 26 27 28 30 31 32 34 35 37 38 39 40 41 44 45 46 47 48];
SpeedIndex2 = [10 11 12 13 14 1];
SpeedIndex3 = [44 45 46 47 48 1];
%% script

p = 2; %pump
clear GainMatrix TauMatrixRising TauMatrixFalling
for n=1:6 %vo
    clear tau K wstar wend ydiff TD
    TD = zeros(41,49);
    for i=1:41 %cases
        for j=1:49 %speed changes
            if mean(fullData(n).data(i,j).data(3+p,:)) > 0.5 %check that there is data there

                % find time constant
                y = fullData(n).data(i,j).data(3+p,:);
                yi=y(end);
                clear idx
                idx=min(find(abs(y-yi)<=abs(0.37*(yi-mean(y(1:10)))) + 0.00005)) * 0.01;


                found = false;
                deriv(1:5)=0;
                for k=6:999
```

```
                    deriv(k)=(y(k-1) - y(k+1))/(2*0.01);
                    deriv_mean(k)=mean(deriv(k-5:k));
                    if abs(deriv_mean(k))>0.5
                        td = k-2;
                        found = true;
                        break
                    end
                end

                TD(i,j) = td/100;

                % acccount for time delay
                if idx >= 9
                    tau(i,j) = 0;
                else
                    if ~isempty(idx)
                        if found
                            tau(i,j) = idx-(td*0.01);
                        end
                    else
                        tau(i,j) = 0;
                    end
                end

                doGain = false;

                if i == 1 || i == 5 || i == 9 || i == 20 || i == 24 || i == 28 %1 & 3
                    doGain = true;
                elseif i == 2 || i == 3 || i == 6 || i == 7 || i == 10 || i == 11 || i == 12 || i == 14 || i == 15 || i == 21 || i == 22 || i == 25 || i == 26 || i == 29 || i == 30 || i == 32 || i == 33 || i == 34
                    if j == 9 || j == 17 || j == 25 || j == 33 || j == 41 || j == 49 %2 & 4
                        doGain = true;
                    end
                end

                if doGain
                    % obtain gain
                    ystart = mean(fullData(n).data(i,j).data(3+p,1:10));
                    yend = mean(fullData(n).data(i,j).data(3+p,900:1000));
                    ydiff(i,j) = yend - ystart;
                    temp = ceil(j/7);
                    if temp == 1
                        wstart(i,j) = 0;
                    else
                        wstart(i,j) = (temp-1)*10+40;
                    end
                    temp2 = j - (temp*7-7);
                    if temp2 == 1
                        wend(i,j) = 0;
                    else
```

```
                        wend(i,j) = (temp2-1)*10+40;
                    end
                    wgain(i,j) = (wend(i,j) - wstart(i,j))/10;
                    if wgain(i,j) ~= 0
                        K(i,j) = ydiff(i,j)/wgain(i,j);
                    end
                else
                    K(i,j) = 0;
                end

            else %if no data, set everything to 0
                tau(i,j) = 0;
                K(i,j) = 0;
            end
        end
    end

    %obtain gain for multi-pump transitions separately
    for i=1:41
        for j=1:49
            doGain = false;
            if i == 1 || i == 5 || i == 9 || i == 20 || i == 24 || i == 28 %1 & 3
                if p == 1 && ismember(i,p1gain)
                    doGain = true;
                elseif p == 2 && ismember(i,p2gain)
                    doGain = true;
                elseif p == 3 && ismember(i,p3gain)
                    doGain = true;
                end
            elseif i == 2 || i == 3 || i == 4 || i == 6 || i == 7 || i == 8 || i == 10 || i == 11 || i == 12 || i == 13 || i == 14 || i == 15 || i == 21 || i == 22 || i == 25 || i == 26 || i == 29 || i == 30 || i == 32 || i == 33 || i == 34
                if j == 9 || j == 17 || j == 25 || j == 33 || j == 41 || j == 49 %2 & 4
                    if p == 1 && ismember(i,p1gain)
                        doGain = true;
                    elseif p == 2 && ismember(i,p2gain)
                        doGain = true;
                    elseif p == 3 && ismember(i,p3gain)
                        doGain = true;
                    end
                end
            end

            if doGain
                % obtain gain
                ystart = mean(fullData(n).data(i,j).data(3+p,1:10));
                yend = mean(fullData(n).data(i,j).data(3+p,900:1000));
                ydiff(i,j) = yend - ystart;
                temp = ceil(j/7);
                if temp == 1
                    wstart(i,j) = 0;
                else
                    wstart(i,j) = (temp-1)*10+40;
```

```
                end
                temp2 = j - (temp*7-7);
                if temp2 == 1
                    wend(i,j) = 0;
                else
                    wend(i,j) = (temp2-1)*10+40;
                end
                wgain(i,j) = (wend(i,j) - wstart(i,j))/10;
                if wgain(i,j) == 0
                    if i == 2 || i == 3 || i == 4 || i == 6 || i == 7 || i == 8 || i == 10 || i == 11 || i == 12 || i == 13 || i == 14 || i == 15
                        wgain(i,j) = wend(i,j)/10;
                        ydiff_temp = yend;
                        if ydiff_temp ~= 0
                            gain_temp = ydiff_temp/wgain(i,j);
                            if p == 1
                                if ismember(i,p1p1)
                                    if i == 14
                                        gain_old = K(2,j) + K(p*4-4+1, temp2);
                                    elseif i == 15
                                        gain_old = K(3,j) + K(p*4-4+1, temp2);
                                    end
                                elseif ~ismember(i,p1start)
                                    gain_old = K(p*4-4+1,temp2);
                                else
                                    gain_old = 0;
                                end
                            elseif p == 2
                                if ismember(i,p2p1)
                                    if i == 12
                                        gain_old = K(7,j) + K(p*4-4+1, temp2);
                                    elseif i == 14
                                        gain_old = K(6,j) + K(p*4-4+1, temp2);
                                    end
                                elseif ~ismember(i,p2start)
                                    gain_old = K(p*4-4+1,temp2);
                                else
                                    gain_old = 0;
                                end
                            elseif p == 3
                                if ismember(i,p3p1)
                                    if i == 12
                                        gain_old = K(11,j) + K(p*4-4+1, temp2);
                                    elseif i == 15
                                        gain_old = K(10,j) + K(p*4-4+1, temp2);
                                    end
                                elseif ~ismember(i,p3start)
                                    gain_old = K(p*4-4+1,temp2);
                                else
```

```
                                        gain_old = 0;
                                    end
                                end
                                K(i,j) = gain_temp - gain_old;
                            end
                        elseif i == 21 || i == 22 || i == 25 || i == 26 || i == 29 || i == 30 || i == 32 || i == 33 || i == 34
                            wgain(i,j) = -wstart(i,j)/10;
                            ydiff_temp = -ystart;
                            if ydiff_temp ~= 0
                                gain_temp = ydiff_temp/wgain(i,j);
                                if p == 1
                                    if ~ismember(i,p1off)
                                        gain_old = K(p*4-4+20,temp2*7-14+8);
                                    else
                                        gain_old = 0;
                                    end
                                elseif p == 2
                                    if ~ismember(i,p2off)
                                        gain_old = K(p*4-4+20,temp2*7-14+8);
                                    else
                                        gain_old = 0;
                                    end
                                elseif p == 3
                                    if ~ismember(i,p3off)
                                        gain_old = K(p*4-4+20,temp2*7-14+8);
                                    else
                                        gain_old = 0;
                                    end
                                end

                                K(i,j) = gain_temp - gain_old;
                            end
                        end
                    end
                end
            end
        end
    %plot time constant w.r.t. case and speed steps

    %       figure
    %       [X,Y] = meshgrid(1:49,1:41);
    %       surf(X,Y,K);
    %       xlabel('Speed');
    %       ylabel('Case')
    %       zlabel('Gain')
    %       title(['Gains for pump ' num2str(p) ' at ' num2str(n*10+15) '% vo']);
    %       colormap(MapK)

    %
    %       figure
    %       [X,Y] = meshgrid(1:49,1:41);
```

```
%       surf(X,Y,tau);
%       xlabel('Speed');
%       ylabel('Case')
%       zlabel('Time constant')
%       title(['Time constants for pump ' num2str(p) ' at ' num2str(n*10+15) '% vo']);
%       colormap(MapTau)

        figure
        [X,Y] = meshgrid(1:49,1:41);
        surf(X,Y,TD);
        xlabel('Speed');
        ylabel('Case')
        zlabel('Time delay')
        title(['Time delays for pump ' num2str(p) ' at ' num2str(n*10+15) '% vo']);
        colormap(MapTau)


%separate the gains
for m=1:6
        GainMatrix(1,1).data(n,m) = K(1,m+1);
        GainMatrix(2,1).data(n,m) = max(K(2,m*8+1),K(3,m*8+1));
        GainMatrix(3,1).data(n,m) = max(K(14,m*8+1),K(15,m*8+1));
        GainMatrix(4,1).data(n,m) = K(4,m*8+1);
        GainMatrix(5,1).data(n,m) = max(K(6,m*8+1),K(10,m*8+1));
        GainMatrix(6,1).data(n,m) = K(12,m*8+1);
        TauMatrixRising(1,1).data(n,m) = tau(1,m+1);
        TauMatrixRising(2,1).data(n,m) = tau(16,m+1);
        TauMatrixRising(3,1).data(n,m) = tau(17,m+1);
        TauMatrixRising(4,1).data(n,m) = tau(19,m+1);
        TauMatrixFalling(1,1).data(n,m) = tau(20,m*7-7+8);
        TauMatrixFalling(2,1).data(n,m) = tau(35,m*7-7+8);
        TauMatrixFalling(3,1).data(n,m) = tau(36,m*7-7+8);
        TauMatrixFalling(4,1).data(n,m) = tau(38,m*7-7+8);

        TDMatrix(1,2).data(n,m) = TD(5,m+1);
        GainMatrix(1,2).data(n,m) = K(5,m+1);
        GainMatrix(2,2).data(n,m) = max(K(6,m*8+1),K(7,m*8+1));
        GainMatrix(3,2).data(n,m) = max(K(12,m*8+1),K(14,m*8+1));
        GainMatrix(4,2).data(n,m) = K(8,m*8+1);
        GainMatrix(5,2).data(n,m) = max(K(2,m*8+1),K(11,m*8+1));
        GainMatrix(6,2).data(n,m) = K(15,m*8+1);
        TauMatrixRising(1,2).data(n,m) = tau(5,m+1);
        TauMatrixRising(2,2).data(n,m) = tau(16,m+1);
        TauMatrixRising(3,2).data(n,m) = tau(18,m+1);
        TauMatrixRising(4,2).data(n,m) = tau(19,m+1);
        TauMatrixFalling(1,2).data(n,m) = tau(24,m*7-7+8);
        TauMatrixFalling(2,2).data(n,m) = tau(35,m*7-7+8);
        TauMatrixFalling(3,2).data(n,m) = tau(37,m*7-7+8);
        TauMatrixFalling(4,2).data(n,m) = tau(38,m*7-7+8);


        GainMatrix(1,3).data(n,m) = K(9,m+1);
        GainMatrix(2,3).data(n,m) = max(K(10,m*8+1),K(11,m*8+1));
        GainMatrix(3,3).data(n,m) = max(K(12,m*8+1),K(15,m*8+1));
```

```
        GainMatrix(4,3).data(n,m) = K(13,m*8+1);
        GainMatrix(5,3).data(n,m) = max(K(3,m*8+1),K(7,m*8+1));
        GainMatrix(6,3).data(n,m) = K(14,m*8+1);
        TauMatrixRising(1,3).data(n,m) = tau(9,m+1);
        TauMatrixRising(2,3).data(n,m) = tau(17,m+1);
        TauMatrixRising(3,3).data(n,m) = tau(18,m+1);
        TauMatrixRising(4,3).data(n,m) = tau(19,m+1);
        TauMatrixFalling(1,3).data(n,m) = tau(28,m*7-7+8);
        TauMatrixFalling(2,3).data(n,m) = tau(36,m*7-7+8);
        TauMatrixFalling(3,3).data(n,m) = tau(37,m*7-7+8);
        TauMatrixFalling(4,3).data(n,m) = tau(38,m*7-7+8);
    end
    for m=1:6
        %% pump1
        TauMatrixRising(5,1).data(n,m) = tau(2,SpeedIndex(m));
        TauMatrixRising(6,1).data(n,m) = tau(3,SpeedIndex(m));
        TauMatrixRising(7,1).data(n,m) = tau(4,SpeedIndex(m));
        TauMatrixRising(8,1).data(n,m) = tau(6,SpeedIndex(m));
        TauMatrixRising(9,1).data(n,m) = tau(10,SpeedIndex(m));
        TauMatrixRising(10,1).data(n,m) = tau(12,SpeedIndex(m));
        TauMatrixRising(11,1).data(n,m) = tau(8,SpeedIndex(m));
        TauMatrixRising(12,1).data(n,m) = tau(13,SpeedIndex(m));
        TauMatrixRising(13,1).data(n,m) = tau(14,SpeedIndex(m));
        TauMatrixRising(14,1).data(n,m) = tau(15,SpeedIndex(m));
        TauMatrixRising(15,1).data(n,m) = tau(39,SpeedIndex2(m));
        TauMatrixFalling(5,1).data(n,m) = tau(21,SpeedIndex(m));
        TauMatrixFalling(6,1).data(n,m) = tau(22,SpeedIndex(m));
        TauMatrixFalling(7,1).data(n,m) = tau(23,SpeedIndex(m));
        TauMatrixFalling(8,1).data(n,m) = tau(25,SpeedIndex(m));
        TauMatrixFalling(9,1).data(n,m) = tau(29,SpeedIndex(m));
        TauMatrixFalling(10,1).data(n,m) = tau(27,SpeedIndex(m));
        TauMatrixFalling(11,1).data(n,m) = tau(31,SpeedIndex(m));
        TauMatrixFalling(12,1).data(n,m) = tau(32,SpeedIndex(m));
        TauMatrixFalling(13,1).data(n,m) = tau(33,SpeedIndex(m));
        TauMatrixFalling(14,1).data(n,m) = tau(34,SpeedIndex(m));
        TauMatrixFalling(15,1).data(n,m) = tau(39,SpeedIndex3(m));
        %% pump2
        TauMatrixRising(5,2).data(n,m) = tau(6,SpeedIndex(m));
        TauMatrixRising(6,2).data(n,m) = tau(7,SpeedIndex(m));
        TauMatrixRising(7,2).data(n,m) = tau(8,SpeedIndex(m));
        TauMatrixRising(8,2).data(n,m) = tau(2,SpeedIndex(m));
        TauMatrixRising(9,2).data(n,m) = tau(11,SpeedIndex(m));
        TauMatrixRising(10,2).data(n,m) = tau(15,SpeedIndex(m));
        TauMatrixRising(11,2).data(n,m) = tau(4,SpeedIndex(m));
        TauMatrixRising(12,2).data(n,m) = tau(13,SpeedIndex(m));
        TauMatrixRising(13,2).data(n,m) = tau(12,SpeedIndex(m));
        TauMatrixRising(14,2).data(n,m) = tau(14,SpeedIndex(m));
        TauMatrixRising(15,2).data(n,m) = tau(40,SpeedIndex2(m));
        TauMatrixFalling(5,2).data(n,m) = tau(25,SpeedIndex(m));
        TauMatrixFalling(6,2).data(n,m) = tau(26,SpeedIndex(m));
        TauMatrixFalling(7,2).data(n,m) = tau(27,SpeedIndex(m));
        TauMatrixFalling(8,2).data(n,m) = tau(21,SpeedIndex(m));
        TauMatrixFalling(9,2).data(n,m) = tau(30,SpeedIndex(m));
        TauMatrixFalling(10,2).data(n,m) = tau(23,SpeedIndex(m));
        TauMatrixFalling(11,2).data(n,m) = tau(31,SpeedIndex(m));
        TauMatrixFalling(12,2).data(n,m) = tau(32,SpeedIndex(m));
```

```
        TauMatrixFalling(13,2).data(n,m) = tau(34,SpeedIndex(m));
        TauMatrixFalling(14,2).data(n,m) = tau(33,SpeedIndex(m));
        TauMatrixFalling(15,2).data(n,m) = tau(40,SpeedIndex3(m));
        %% pump 3
        TauMatrixRising(5,3).data(n,m) = tau(10,SpeedIndex(m));
        TauMatrixRising(6,3).data(n,m) = tau(11,SpeedIndex(m));
        TauMatrixRising(7,3).data(n,m) = tau(13,SpeedIndex(m));
        TauMatrixRising(8,3).data(n,m) = tau(3,SpeedIndex(m));
        TauMatrixRising(9,3).data(n,m) = tau(7,SpeedIndex(m));
        TauMatrixRising(10,3).data(n,m) = tau(14,SpeedIndex(m));
        TauMatrixRising(11,3).data(n,m) = tau(4,SpeedIndex(m));
        TauMatrixRising(12,3).data(n,m) = tau(8,SpeedIndex(m));
        TauMatrixRising(13,3).data(n,m) = tau(12,SpeedIndex(m));
        TauMatrixRising(14,3).data(n,m) = tau(15,SpeedIndex(m));
        TauMatrixRising(15,3).data(n,m) = tau(41,SpeedIndex2(m));
        TauMatrixFalling(5,3).data(n,m) = tau(29,SpeedIndex(m));
        TauMatrixFalling(6,3).data(n,m) = tau(30,SpeedIndex(m));
        TauMatrixFalling(7,3).data(n,m) = tau(31,SpeedIndex(m));
        TauMatrixFalling(8,3).data(n,m) = tau(22,SpeedIndex(m));
        TauMatrixFalling(9,3).data(n,m) = tau(26,SpeedIndex(m));
        TauMatrixFalling(10,3).data(n,m) = tau(23,SpeedIndex(m));
        TauMatrixFalling(11,3).data(n,m) = tau(27,SpeedIndex(m));
        TauMatrixFalling(12,3).data(n,m) = tau(33,SpeedIndex(m));
        TauMatrixFalling(13,3).data(n,m) = tau(34,SpeedIndex(m));
        TauMatrixFalling(14,3).data(n,m) = tau(32,SpeedIndex(m));
        TauMatrixFalling(15,3).data(n,m) = tau(41,SpeedIndex3(m));
    end
end

titleGainMatrix = ['starting up a pump                        '
    'running and starting up another pump   '
    'running 2 and starting up another pump '
    'running and starting up 2 pumps        '
    'starting up a pump with another running'
    'starting up a pump with 2 other running'];
titleTauRisingMatrix = ['starting up a pump                        '
    'starting up 2 pumps                    '
    'starting up 2 different pumps          '
    'starting up 3 pumps                    '
    'running a pump and starting up another  '
    'running a pump and starting up different'
    'running 2 pumps and starting up last    '
    'starting a pump with one running        '
    'starting a pump with anther running     '
    'starting a pump with 2 running          '
    'starting 2 pumps with 1 running         '
    'starting 2 pumps with different running '];

titleTauFallingMatrix = ['stopping a pump                        '
    'stopping 2 pumps                       '
    'stopping 2 different pumps             '
    'stopping 3 pumps                       '
    'running a pump and stopping another    '
    'running a pump and stopping different  '
    'running 2 pumps and stopping last      '
    'stopping a pump with one running       '
```

```
    'stopping a pump with anther running     '
    'stopping a pump with 2 running          '
    'stopping 2 pumps with 1 running         '
    'stopping 2 pumps with different running '
    'stopping 2 pumps with different running '
    'stopping 2 pumps with different running '];

%plot separated cases

% for i=1:6
%     fig
%     [X,Y] = meshgrid(1:6,1:6);
%     surf(X,Y,GainMatrix(i,p).data);
%     xlabel('Speed');
%     ylabel('Valve opening')
%     zlabel('Gain')
%   xticks([1 2 3 4 5 6])
%     xticklabels({'0-50%','0-60%','0-70%','0-80%','0-90%','0-100%'})
%     yticks([1 2 3 4 5 6])
%     yticklabels({'25%','35%','45%','55%','65%','75%'})
%     title(['Pump ' num2str(p) ' gains when ' titleGainMatrix(i,:)]);
% end

fig
[X,Y] = meshgrid(1:6,1:6);
surf(X,Y,TDMatrix(1,p).data);
xlabel('Speed');
ylabel('Valve opening')
zlabel('Time Delay')
xticks([1 2 3 4 5 6])
xticklabels({'0-50%','0-60%','0-70%','0-80%','0-90%','0-100%'})
yticks([1 2 3 4 5 6])
yticklabels({'25%','35%','45%','55%','65%','75%'})
title(['Pump ' num2str(p) ' time delay when ' titleGainMatrix(1,:)]);



%
% for i=1:4
%     fig
%     [X,Y] = meshgrid(1:6,1:6);
%     surf(X,Y,TauMatrixRising(i,p).data);
%     xlabel('Speed');
%     ylabel('Valve opening')
%     zlabel('time constant')
%     xticks([1 2 3 4 5 6])
%     xticklabels({'0-50%','0-60%','0-70%','0-80%','0-90%','0-100%'})
%     yticks([1 2 3 4 5 6])
%     yticklabels({'25%','35%','45%','55%','65%','75%'})
%     title(['Pump ' num2str(p) ' tau when ' titleTauRisingMatrix(i,:)
    ]);
% end
%
% for i=1:4
%     fig
%     [X,Y] = meshgrid(1:6,1:6);
```

```
%       surf(X,Y,TauMatrixFalling(i,p).data);
%       xlabel('Speed');
%       ylabel('Valve opening')
%       zlabel('time constant')
%       xticks([1 2 3 4 5 6])
%       xticklabels({'0-50%','0-60%','0-70%','0-80%','0-90%','0-100%'})
%       yticks([1 2 3 4 5 6])
%       yticklabels({'25%','35%','45%','55%','65%','75%'})
%       title(['Pump ' num2str(p) ' tau when ' titleTauFallingMatrix(i,:)
    ]);
% end



%%
% for i=5:12
%       figure
%       [X,Y] = meshgrid(1:6,1:6);
%       surf(X,Y,TauMatrixRising(i,p).data);
%       xlabel('Speed');
%       ylabel('Valve opening')
%       zlabel('time constant')
%       title(['Pump ' num2str(p) ' tau when ' titleTauRisingMatrix(i,:)
    ]);
% end

%% pump1
TauRisingNew(1,1).data = TauMatrixRising(1,1).data;
TauRisingNew(2,1).data = TauMatrixRising(5,1).data;
TauRisingNew(3,1).data = TauMatrixRising(7,1).data;
TauRisingNew(4,1).data = TauMatrixRising(8,1).data;
TauRisingNew(5,1).data = TauMatrixRising(10,1).data;
TauRisingNew(6,1).data = TauMatrixRising(11,1).data;
TauRisingNew(7,1).data = TauMatrixRising(13,1).data;
TauRisingNew(8,1).data = TauMatrixRising(15,1).data;
TauFallingNew(1,1).data = TauMatrixFalling(1,1).data;
TauFallingNew(2,1).data = TauMatrixFalling(5,1).data;
TauFallingNew(3,1).data = TauMatrixFalling(7,1).data;
TauFallingNew(4,1).data = TauMatrixFalling(8,1).data;
TauFallingNew(5,1).data = TauMatrixFalling(14,1).data;
TauFallingNew(6,1).data = TauMatrixFalling(10,1).data;
TauFallingNew(7,1).data = TauMatrixFalling(12,1).data;
TauFallingNew(8,1).data = TauMatrixFalling(15,1).data;

%% pump2
TauRisingNew(1,2).data = TauMatrixRising(1,2).data;
TauRisingNew(2,2).data = TauMatrixRising(5,2).data;
TauRisingNew(3,2).data = TauMatrixRising(7,2).data;
TauRisingNew(4,2).data = TauMatrixRising(8,2).data;
TauRisingNew(5,2).data = TauMatrixRising(10,2).data;
TauRisingNew(6,2).data = TauMatrixRising(11,2).data;
TauRisingNew(7,2).data = TauMatrixRising(13,2).data;
TauRisingNew(8,2).data = TauMatrixRising(15,2).data;
TauFallingNew(1,2).data = TauMatrixFalling(1,2).data;
TauFallingNew(2,2).data = TauMatrixFalling(5,2).data;
TauFallingNew(3,2).data = TauMatrixFalling(7,2).data;
```

```
TauFallingNew(4,2).data = TauMatrixFalling(8,2).data;
TauFallingNew(5,2).data = TauMatrixFalling(14,2).data;
TauFallingNew(6,2).data = TauMatrixFalling(10,2).data;
TauFallingNew(7,2).data = TauMatrixFalling(12,2).data;
TauFallingNew(8,2).data = TauMatrixFalling(15,2).data;


%% pump3
TauRisingNew(1,3).data = TauMatrixRising(1,3).data;
TauRisingNew(2,3).data = TauMatrixRising(5,3).data;
TauRisingNew(3,3).data = TauMatrixRising(7,3).data;
TauRisingNew(4,3).data = TauMatrixRising(8,3).data;
TauRisingNew(5,3).data = TauMatrixRising(10,3).data;
TauRisingNew(6,3).data = TauMatrixRising(11,3).data;
TauRisingNew(7,3).data = TauMatrixRising(13,3).data;
TauRisingNew(8,3).data = TauMatrixRising(15,3).data;
TauFallingNew(1,3).data = TauMatrixFalling(1,3).data;
TauFallingNew(2,3).data = TauMatrixFalling(5,3).data;
TauFallingNew(3,3).data = TauMatrixFalling(7,3).data;
TauFallingNew(4,3).data = TauMatrixFalling(8,3).data;
TauFallingNew(5,3).data = TauMatrixFalling(14,3).data;
TauFallingNew(6,3).data = TauMatrixFalling(10,3).data;
TauFallingNew(7,3).data = TauMatrixFalling(12,3).data;
TauFallingNew(8,3).data = TauMatrixFalling(15,3).data;

x = 50:10:100;
y = 25:10:75;

for i=1:8
    for k=6:6
        for j=1:3
            if i < 7
                clear A
                A = fit(x',GainMatrix(i,j).data(k,:)','poly1');
                GainFit(i,j).data(1) = A.p1;
                GainFit(i,j).data(2) = A.p2;
                clear A
                A = fit(x',TauRisingNew(i,j).data(k,:)','poly1');
                TauRisingFit(i,j).data(1) = A.p1;
                TauRisingFit(i,j).data(2) = A.p2;
                clear A
                A = fit(x',TauFallingNew(i,j).data(k,:)','poly1');
                TauFallingFit(i,j).data(1) = A.p1;
                TauFallingFit(i,j).data(2) = A.p2;
            else
                clear A
                A = fit(x',TauRisingNew(i,j).data(k,:)','poly1');
                TauRisingFit(i,j).data(1) = A.p1;
                TauRisingFit(i,j).data(2) = A.p2;
                clear A
                A = fit(x',TauFallingNew(i,j).data(k,:)','poly1');
                TauFallingFit(i,j).data(1) = A.p1;
                TauFallingFit(i,j).data(2) = A.p2;
            end
        end
    end
```

```
end
%% Gains:
%1) pump just starting up
%2) pump running, another starting up (close enough to use same value for
%either pump?)
%3) 2 pumps running, last one starting up
%4) pump running, 2 other starting up (higher gain than with 1 on higher
%vos)
%5) pump starting up with another running (different case; close enough
%numbers to use same value with diff pump)
%6) pump starting up with 2 others running

%shutting off mirrors turning off cases for any gain

%% Tau Rising:
%1) pump starting up
%2) pump running, another starting up
%3) pump running, starting up 2
%4) starting up a pump with another running
%5) starting up a pump with 2 running
%6) starting up a pump +1 with another running
%7) pump +1 running, starting up last one
%8) pump changing speeds (increasing)

%% Tau Falling:
%1) turning off a pump -> 1
%2) running 2 pumps and turning off another -> 2
%3) running 3 pumps and turning off other two -> 3
%4) running 2 pumps and turning off the pump -> 4
%5) running 3 pumps and turning the pump off -> 5
%6) running 3 pumps and turning off the pump +1 -> 6
%7) running 3 pumps and turning off another pump -> 7
%8) pump changing speeds (reducing)
```

Code Listing 9.2: Script to load experimental data, obtain gains, time delays and constants as well as plot them

9.3 Code examples

9.3.1 Combination's speed

```
%% Href = reference head
for n=1:7 %for every combination
    a0 = coeffH(n,3); %load the coefficients
    a1 = coeffH(n,2);
    a2 = coeffH(n,1);
    %% use Bisection method to find a solution for speed to produce the desired head
    wa = 0;   %speed point a
    wb = 10;  %speed point b
```

```
    wc = 0;   %speed point c
    threshold = 0.0001;   %set threshold (error margin for head)
    residual = 3.6;       %initial value (can be some value as long as it's not below threshold)
    for it=1:1000 %make a thousand interations, although the solution is usually found within few steps
        if residual < threshold %if we're below the threshold, the solution is found - break the loop
            break
        end
        wc = 0.5*(wa+wb); %calculate c point - middle between a and b
        head = a0*wc^2 + a1*wc*Qstar(n,1) + a2*Qstar(n,1)^2; %put the values into the head equation
        residual = abs(Href-head); %calculate the residual
        if head > Href %if we're over the reference, lower the b boundary
            wb = wc;
        else
            wa = wc; %otherwise, lower a boundary
        end
    end
    speedI(n,1) = wc; %set input speed based on the solution
    %constrain the speed artificially
    if w == 5 %if the solution was not found, it is trying to run the system beyond known limits
        speed(n,1) = 500; %as a result, trick it into running at high speed
    end
  if w <= 2.7 %if its approximately below 50%, force it back to 50%
        speedI(n,1) = 2.7;
    end
end
```

Code Listing 9.3: Excerpt of the code to find combination's pump speed

9.3.2 Optimal solution

```
%find consumed power for each combination based on expected flow rate and coefficients
for n=1:7
  p0 = coeffP(n,4);
  p1 = coeffP(n,3);
  p2 = coeffP(n,2);
  p3 = coeffP(n,1);
  P(n,1) = p0*speed(n,1)^3 + p1*speed(n,1)^2*Q(n,1) + p2*speed(n,1)*Q(n,1)^2 + p3*Q(n,1)^3;
end
rho=1000; %desnity of water
g=9.82; %gravitational force

%find efficiency of each combination
for n=1:7
  eta(n,1) = ((rho*g*Href*(Q(n,1)/3600))/P(n,1) * 100);
```

```
  if speed(n,1) == 500 %if the combination's speed is impossible to achieve
    eta(n,1) = 0; %set efficiency to zero, as that combination is invalid
  end
end
%minimize for power
powermin = min(P);
%and maximize for efficiency
etamax = max(eta);
%loop through all combinations to find the best solutions
for n=1:7
  if P(n,1) == powermin %if we're minimizing for power
    Npower = n; %set optimized combination
    wpower = -34 + 42.588*speed(n,1) + (-5.862*speed(n,1)^2) + 0.5764*speed(n,1)^3; %convert voltage to percentage
  end
  if eta(n,1) == etamax %if we're maximizing for efficiency
    Neff = n;
    weff = -34 + 42.588*speed(n,1) + (-5.862*speed(n,1)^2) + 0.5764*speed(n,1)^3;
  end
end
```

Code Listing 9.4: Excerpt of the code to find power and most optimal solution

9.3.3 Valve estimator

```
k1new = Hmeasurement/(Qnew^2); %obtain new k1 value for the current system
found = 0; %start looking for valve opening percentage
i = 1;
for z=1:26 %loop through all known steps
  if k1new > Valve(Neff,z) %and try to find between which steps it lays
    i = z;
    found = 1;
    break
  end
end

if found == 1 %if it was within known boundaries (25-75%)
  if i ~= 1 %and it wasn't 25%
    b = Valve(Neff,i-1); %set the 'a' and 'b' coefficients based on known data
    a = (Valve(Neff,i)-b)/2;
  else %else interpolate to get 'a' coefficient based on approximte 23% value
    b = Valve(Neff,i);
    a = 0.16;
  end
else %if it was instead above 75%
  b = Valve(Neff,26); %interpolate again
  a = 0.0047;
  i = 26;
```

```
end
%use `y = ax + b` to find `x`
voSolv = (k1new - b)/a;
%add the "base" percentage to get the valve opening percentage approximation
vo = voSolv + i*2-2 + 23;

%now use the same step and the small percentage inbetween steps to find new k1 values for each combination
voTemp = voSolv;
for n=1:7
  b = Valve(n,i);
  a = (Valve(n,i+1)-b)/2;
  voEst = a * voTemp + b; %again use `y = ax + b`
  k1est(n,1) = voEst; %set the value to estimated k1 coefficient matrix
end
```

Code Listing 9.5: Excerpt of the code to estimate valve opening

9.3.4 Coefficient selector

```
%N - case number, determined from another function block
...% get coefficients for transfer function based on case
elseif N == 5
    Gain1Coeff = [0;0];
    Td1 = 0;
    Tau1Coeff = [0;0];
    Td2 = 0.2;
elseif N == 6
    Gain1Coeff = [K_a(2,p);K_b(2,p)];
    Td1 = 1;
    Tau1Coeff = [TauF_a(2,p);TauF_b(2,p)];
    Td2 = 0.2;
    addGain = true;
elseif N == 14
    Gain1Coeff = [K_a(3,p);K_b(3,p)];
    Td1 = 0.7;
    Tau1Coeff = [TauF_a(7,p);TauF_b(7,p)];
    Td2 = 0.2;
    addGain = true;
end
% If we have switched combinations in the current step
if Nprev ~= N
    K1Temp = speed*Gain1Coeff(1) + Gain1Coeff(2); %recalculate new gain
    if addGain % check if the case demands gain to be added to the previous one
        K1Temp = K1Temp + K1prev;
    end
  K1prev = K1Temp; %keep track of values used in this step for next one
    Gain1prev = Gain1Coeff(1);
    Gain2prev = Gain1Coeff(2);
    Nprev = N;
else
    K1Temp = K1prev;
```

```
end
%check whether or not we changed input speeds
if speedprev == speed
    K2Temp = K2prev;
    Tau2Temp = Tau2prev;
else %if so, recalculate new base coefficients
    if speedprev > speed %check whether pump is starting up or being shut down
        Gain2Coeff = [K_a(1,p);K_b(1,p)];
        Tau2Coeff = [TauF_a(8,p);TauF_b(8,p)];
    else
        Gain2Coeff = [K_a(1,p);K_b(1,p)];
        Tau2Coeff = [TauR_a(8,p);TauR_b(8,p)];
    end
    K2Temp = speed*Gain2Coeff(1) + Gain2Coeff(2); %put the coefficients into line equations
    K2prev = K2Temp;
    Tau2Temp = speed*Tau2Coeff(1) + Tau2Coeff(2);
    Tau2prev = Tau2Temp;
end
%output the values we calculated into the time varying TF block
speedprev = speed;
K1 = K1Temp;
K2 = K2Temp;
Tau1 = speed*Tau1Coeff(1) + Tau1Coeff(2);
Tau2 = Tau2Temp;
```

Code Listing 9.6: Algorithm to select DC gain and time constant values based on input.

9.3.5 Test runs

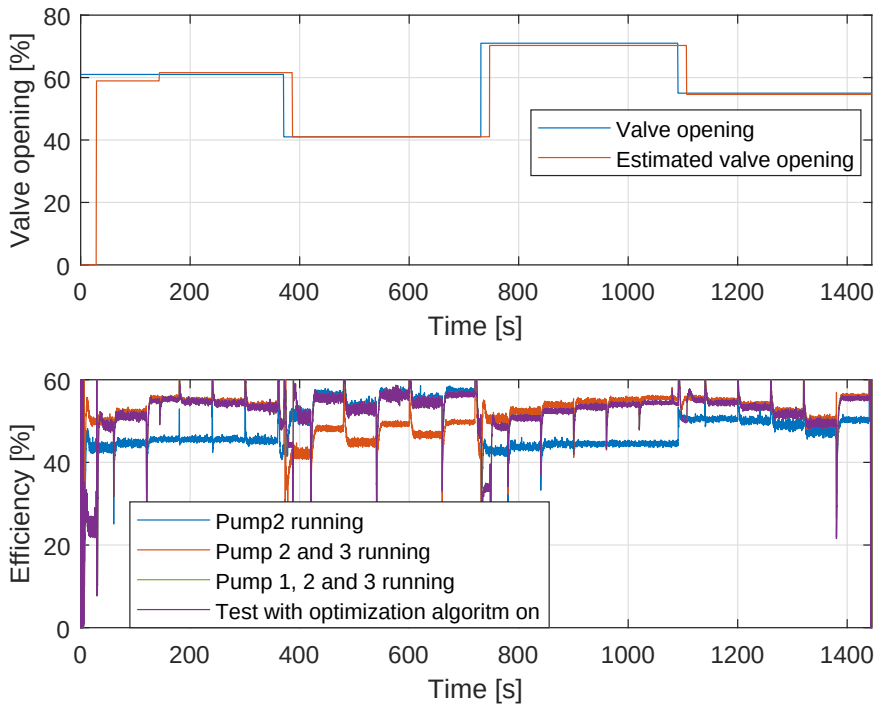


Figure 9.1: Comparison of efficiency.

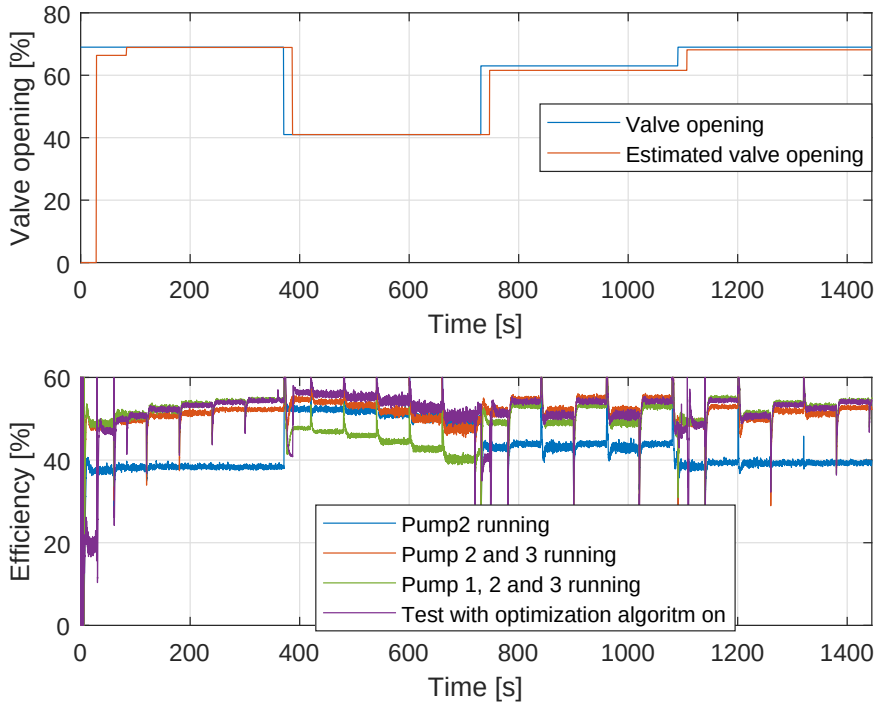


Figure 9.2: comparison of efficiency.