UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING AND ENERGY TECHNOLOGY

AUTOMATION

Joni Vesterback

PRODUCING PID CONTROLLERS FOR TESTING CLUSTERING

Investigating novelty detection for use in classifying PID parameters

Master's thesis for the degree of Master of Science in Technology submitted for inspection, Vaasa 28.06.2013

Supervisor	Jarmo Alander
Instructor	Vladimir Bochko

FOREWORDS

This thesis was done for University of Vaasa, as a part of the Cluster project, which the Automation group of University of Vaasa made for Wärtsilä Finland Ltd. I wish to thank Professor Jarmo Alander, Doctor Vladimir Bochko both from University of Vaasa and Fredrik Östman from Wärtsilä Finland Ltd, for their support, discussions, and idea meetings. Special thanks to Doctor Vladimir Bochko for creating the analyser that is discussed in this work and for taking care of many of the arrangements regarding the publication of this research, and also to Professor Jarmo Alander for providing me with the opportunity to make this work and also taking care of the arrangements behind this work.

Vaasa 28.06.2013

Joni Vesterback

TABLE OF CONTENTS

SYMBOLS AND ABBREVIATIONS

A	Weight coefficient for sum of absolute error in fitness function
A, B, C	Probability areas for Gaussian distribution
B	Weight coefficient for max error in fitness function
C	Weight coefficient for final error in fitness function
C(*s*)	Controller transfer function
D	Derivative parameter of PID controller
d	Differential operator sign
E(*s*)	Control error, Laplace domain
e, *e*(t)	Control error, time domain
e_{∞}	Final error
e_{max}	Maximum error
EVD	Extreme Value Distribution
EVT	Extreme Value Theory
e_{Σ}	Sum of absolute error
F_n	Probability distribution function
$f_n, f_n(\mathbf{X})$	Denotation of probability density function
FPR	False Positive Rate
GA	Genetic Algorithm
GMM	Gaussian Mixture Model
GMVC	Generalized Minimum Variance Control
H(*s*)	Process transfer function
I	Integration parameter of PID controller
IT	Information Technology
j	Imaginary unit
k	Threshold in novelty detection
K_d	Proportional coefficient of PID controller for derivation
K_i	Proportional coefficient of PID controller for integration
K_p	Proportional coefficient of PID controller
L	Laplace operator
MEVS	Multivariate Extreme Value Statistics
n_{anorm}	Set of abnormal 104 PID parameters for testing the analyser
n_{norm}	Set of 104 normal PID parameters for testing the analyser

n_t	Set of 104 normal PID parameters for training the analyser
o_{max}	Maximum overshoot
P	Proportional parameter of PID controller
$\mathbf{P}$	Probability
PCA	Principal Component Analysis.
PD	Proportional and Derivative
pdf	Probability density function
PI	Proportional and Integral
PID	Proportional, Integral and Derivative
s	Laplace variable
t	Time
T_d	Derivation time
T_i	Integration time
TPR	True Positive Rate
t_s	Settling time
$U(s)$	Input to process Laplace domain
$u(t)$	Input to process (control variable)
VMM	Variational Mixture Model
$\mathbf{X}$, $\mathbf{X}_n$	Test data (example)
$Y(s)$	Output from process, Laplace domain
y, $y(t)$	Output from process, time domain
$Y_{sp}(s)$	Setpoint, Laplace domain
y_{sp}, $y_{sp}(t)$	Set point, time domain
σ	Decay ratio
ω	Angular frequency

LIST OF FIGURES

LIST OF TABLES

VAASAN YLIOPISTO
Teknillinen tiedekunta

Tekijä:	Joni Vesterback
Diplomityön nimi:	PID-säätäjien luominen ryvittämisen testaamista varten. – Tutkimus ei-Gaussisten havaintamenetelmän käytöstä PID säätäjien luokittelussa.
Valvojan nimi:	Jarmo Alander
Ohjaajan nimi:	Vladimir Bochko
Tutkinto:	Diplomi-Insinööri
Oppiaine:	Automaatiotekniikka
Opintojen aloitusvuosi:	2004
Diplomityön valmistumisvuosi:	2013 **Sivumäärä:** 73

TIIVISTELMÄ

PID säätäjän toimivuus riippuu siitä miten hyvin sen parametrit on konfiguroitu. Säätäjän konfigurointi ei ole helppoa. Siihen käytetään kokemusta ja intuitiota tai automaattisia ohjelmia. Esitämme tavan arvioida säätäjien konfiguroinnin laatua tilastollisin menetelmin. Metodi perustuu usean muuttujan ääriarvojen tilastollisiin ominaisuuksiin. Tässä työssä esitetään myös analysaattori joka käyttää tätä tilastotieteellistä teoriaa hyväksi. Analysaaattori vertaa uusia PID konfigurointeja tunnettuihin PID konfigurointeihin, jotka ovat todettu hyvin toimiviksi. Tämä työkalu auttaa ongelmien torjunnassa PID säätäjien konfiguroinnissa. Tavanomaiset epänormaaliuuksien havaintamenetelmät perustuvat Gaussin jakaumaan. Työssä esitettävä analysaattori käyttää vaihtelevaa jakautumaa. Siksi sen käyttäminen teki datan sovittamisen jakautumaan helpommaksi käyttäjälle.

Työn yksi osa oli PID konfiguraatioiden luominen, joilla testaisimme analysaattoria. Tähän tarvitsimme sekä hyvin että huonosti säädettyjä konfiguraatioita. Molemmista tapauksista tarvittiin useita esimerkkejä. Geneettinen algoritmi nähtiin tähän työhön erittäin sopivana työkaluna. Geneettisiä algoritmeja on ennenkin käytetty sekä PID säätäjien konfigurointiin että testidatan luomiseen. Geneettinen algoritmi ohjelmoitiin Matlabissa. PID säätäjiä simuloitiin Simulink mallilla, jota käytettiin hyvyysfunktiossa.

PID konfiguraatiot simuloitiin ja niiden askelvasteen kuvaajat piirrettiin. Parhaimmat geneettisen algoritmin löytämät konfiguraatiot tuottavat vähän virhettä tavoitearvoon verrattuna. Virhe näytti myös kasvavan geneettisen algoritmin antaman hyvyysluvun laskiessa. Testiparametreille käytettiin kolmea kriteeriä: maksimiylitys, asettumisaika sekä eron itseisarvon summa. Jokaiselle kriteerille annettiin raja-arvo. Konfiguraatio joka ylitti yhdenkin näistä luokiteltiin epänormaaliksi.

Analysaattorin toimintaa arvioitiin näillä testikonfiguraatioilla. Analysaattori opetettiin ensin normaaleilla konfiguraatioilla ja sen jälkeen testattiin ensin joukolla normaaleja konfiguraatioita ja sitten joukolla epänormaaleja parametreja. Tuloksista löytyi 2 väärää hälytystä molemmissa tapauksissa 104:stä mahdollisesta. Tämä antoi 98%:n tarkkuuden, joka on erittäin korkea epänormaalisuuksien havaintamenetelmälle.

AVAINSANAT: Ääriarvo tilastotiede useilla muuttujilla, PID säätäjä, testidatan luonti, geneettinen algoritmi, vaihteleva jakautuma.

VASA UNIVERSITET
Tekniska fakulteten

Författare:	Joni Vesterback	
Titel:	Skapande av PID-kontrollers för att testa klustrering. – Undersökning av att använda upptäckt av onormalheter för att klassificera PID parametrar.	
Examinator:	Jarmo Alander	
Handledare:	Vladimir Bochko	
Examen:	Diplom-Ingenjör	
Huvudämne:	Automationsteknik	
Intagningsår:	2004	
Utgivningsår:	2013	**Antal sidor:** 73

REFERAT

Prestanda av PID kontrollers är beroende av deras inställningar. Det är inte enkelt att konfigurera en PID kontroller och flera använder deras erfarenhet och intuition, eller automatiska program för konfigurering. I det här arbetet presenterar vi en metod för att testa kvaliteten av PID kontrollers med hjälp av statistiska metoder. Metoden använder sig av extremvärde statistik med flera variabler. Med analysatorn som presenteras i det här arbetet kan man jämföra nya PID inställningar till de som man vet har fungerat väl. Konventionellt använder man Gaussisk fördelning i extremvärde statistik. Analysatorn i det här arbetet använder en varierande fördelning i stället. Det här gjorde det enklare för användaren att anpassa data till fördelningen.

En del av det här arbetet var att producera PID parameter konfigurationer för att testa analysatorn med. Vi behövde flera exempel av så väl bra inställda parametrar som dåligt inställda parametrar. Vi såg att en genetisk algoritm var det perfekta verktyget för det här jobbet. Genetiska algoritmer har förr använts för både generering av test parametrar och för inställning av PID kontrollers. Genetiska algoritmen var skriven i Matlab. PID kontrollerna simulerades med hjälp av en Simulink modell.

PID konfigurationerna simulerades och graferna av deras stegsvar ritades. De bästa konfigurationerna enligt genetiska algoritmen hade bara litet fel jämfört med målvärdet. Felet mellan målvärde och utmatning steg enligt godhetsvärdet som genetiska algoritmen hade givit. Vi gav tre kriterier för varje konfiguration som vi testade analysatorn med: max översläng, insvängningstid och summan av absoluta fel. Alla dessa kriterier fick ett gränsvärde. Om en konfiguration översteg endast ett av de här gränsvärdena så blev kontrollern klassad som onormal.

Analysatorns prestanda undersöktes med hjälp av dessa konfigurationer. Först var analysatorn tränad med en grupp av normala parametrar. Efter det var den testad med en grupp normala och en grupp av onormala parametrar. Resultaten av båda grupperna gav två felbedömda konfigurationer ut av 104 möjliga. Det här betydde att analysatorns precision var 98%, vilket är ett högt värde för en extremvärde-statistik-applikation.

NYCKELORD: extremvärde statistik med flera variabler, PID kontroller, test data generering, genetiska algoritmer, varierande fördelningar.

UNIVERSITY OF VAASA	
Faculty of technology	
Author:	Joni Vesterback
Topic of the Thesis:	Producing PID controllers for testing clustering. – Investigating novelty detection for use in classifying PID parameters.
Supervisor:	Jarmo Alander
Instructor:	Vladimir Bochko
Degree:	Master of Science in Technology
Major of Subject:	Automation Engineering
Year of Entering the University:	2004
Year of Completing the Thesis:	2013 **Pages:** 73

ABSTRACT

PID controllers performance depend on how they are tuned. Tuning a controller is not easy either and many use their experience and intuition, or automatic software for tuning. We present a way to test the quality of controllers using statistics. The method uses multivariate extreme value statistics with novelty detection. With the analyser presented in this paper one can compare fresh PID parameters to those that have been tuned well. This tool can help in troubleshooting with PID controller tuning. Conventional novelty detection methods use a Gaussian mixture model, the analyser here uses a variational mixture model instead. This made the fitting process easier for the user.

Part of this work was to create PID parameter configurations to test the analyser with. We needed both well tuned and poorly tuned parameters for testing the algorithm, as well as several examples of both cases. A genetic algorithm was seen as a tool that would meet these requirements. Genetic algorithms have previously been used for both test parameters generation and PID controller tuning in many applications. The genetic algorithm was written in Matlab. The reason for using Matlab is that the genetic algorithm uses a Simulink model of a PID control process in its fitness function.

The parameters were simulated and plots of their step response were drawn. The best configurations according to the genetic algorithm had little error compared to the reference value. The error seemed to rise according to the index of goodness used by the genetic algorithm. We set three criterions on the parameters: maximum overshoot, settling time, and sum of absolute error. Each of these criterions had a threshold. Each parameter configuration that crossed at least one of these thresholds were classed abnormal.

The performance of the analyser was assessed with these parameters. The analyser were first trained with a set of normal parameters, then tested with a set of normal and a set of abnormal parameters. The results showed 2 false alarms in both cases out of 104 possible. This gave us an accuracy of 98%, which is a very high one for a novelty detection method.

KEYWORDS: Novelty detection, multivariate extreme value statistics, PID controller, test data generation, genetic algorithms, variational mixture model.

1. INTRODUCTION

PID controllers provide many important features, such as proportional feedback, regulation of steady state control error and future error predicion by derivation. This is why PID controllers are popular controllers in industry today. It takes only three parameters to configure a PID controller. Still, tuning them requires much knowledge and experience. Their performance may vary greatly depending on how they are configured and the parameters can be chosen from a wide range. Also knowing what to measure according to the goal is crucial. This is why tuning a controller is not easy and many applications have had poorly tuned controllers over the past years. (Åström & Hägglund 1995: 1-2)

According to Blevins (2012: 1-2) a research showed that factories which put effort into analysing their use of PID controllers improved significantly their production. Such researches concerned monitoring tools, personnel, single processes and overall use in the whole factory processes. Because of the popularity of PID controllers, new methods for tuning and other tools are constantly being investigated. (Blevins 2012: 1-2) Over the past 10 years the controllers have gotten the attention of academics (Åström & Hägglund 2001: 1163).

During that time PID controllers have taken huge leaps in development. The development has focused on automatic tuning methods, adaptive control, monitoring and making the controllers in such a way that the user does not need much knowledge of controllers. (Blevins 2012: 1-2)

This thesis investigates a completely new method, where one can assess the quality of PID controllers before even running them in a process. This method uses a technique called novelty detection. University of Vaasa was investigating a completely new application of the novelty detection method, which they called ‘PID outlier detection’. In this thesis we will produce an experiment for this method, analyse the results and assess how well the method functions.

The request to make this investigation came from Wärtsilä, Finland Ltd. Wärtsilä is a worldwide company, producing combustion diesel engine power solutions for ships and energy markets. They emphasize technological innovation and efficiency. Power plants made by Wärtsilä are based on diesel engines. Due to this, in this thesis we will mostly focus the outlier detection problem on diesel power plants. (Wärtsilä Ltd 2012)

A program based on the theory for novelty detection methods had been built by University of Vaasa. The program takes PID parameter triplets as input. First it is trained with parameters classified as normal. Then it tests any other sets of parameters and gives an estimate of their normality.

With the analyser presented in this work, one can predetermine PID parameter quality without even having to run them in a process. An analyser like this is useful for countering issues on the field, where there might be poorly tuned controllers. By being able to predetermine parameter quality, one can shorten the troubleshooting times.

In this work we will use a genetic algorithm to produce PID parameters to test how well novelty detection works in this field. Then the results will be assessed, both the method used to create parameters and the analyser program.

1.1. Introducing a state of the art method

Novelty detection is a statistical analysis method used to determine whether data are normal or abnormal when compared to a set of data considered normal. The method is often used in jet engine, manufacturing processes, power generating facilities and patient health monitoring. The best use for the method is when examples of abnormal behaviour are hard to find. The method works by comparing data that are known to represent the target in its normal condition, to data which condition is unknown. Then we use these to assess the quality of other configurations. (Clifton, Hugeny & Tarassenko 2011: 371). For engine health the parameters might be e.g. engine vibrations (Clifton, Hugueny & Tarassenko 2009: 15). For patient health monitoring these might be heart rate, respiration rate, blood pressure, body temperature, etc. (Clifton, Hugeny & Tarassenko

2010: 5). The goal of PID outlier detection is to be able to find out if a set of PID parameters are normal or abnormal.

Usually novelty detection methods have used Gaussian Mixture Models (GMM) for distributions. This work presents a new way for distributing data, using a Variational Mixture Model (VMM). The method was introduced in Vesterback, Bochko, Ruohonen, Alander, Bäck, Nylund, Dal & Östman (2012: 405, 412). We will also explain what it means to use a VMM instead of a GMM, why it is better and what consequences it brings.

1.2. Previous work

1.2.1. Novelty detection and PID controllers

Novelty detection has been used in a number of applications. According to a review of the method, written by Miljkovic (2010) these applications include system monitoring, aerospace and railroad systems, IT security applications, image processing and video surveillance, and even topic detection in text mining, as well as the previously mentioned engine and patient health monitoring systems. Several different variants of novelty detection have been used in each of these applications.

However the use of the statistical approach, using a GMM to create a probability distribution when determining the novelty threshold using extreme value theory (EVT) is very limited. The method was introduced in Roberts 1999, where he showed using the EVT approach that it worked better than the previously commonly used heuristics method to set the novelty threshold. Later Roberts wrote another paper on the same subject in 2000. (Roberts 1999; Roberts 2000)

However, this method is weak when used in multivariate, multimodal problems. A solution was suggested by Clifton et al. (2009). The solution was to use multivariate extreme value statistics (MEVS) for these problems instead of classical EVT. The authors investigated these approaches to patient health monitoring and jet engine health monitoring in several works, for example in Clifton et al. (2010) and Clifton et al. (2009).

Using novelty detection to determine the quality of PID controllers is a new application of the method and the only existing work is Vesterback et al. (2012), which was based on the same research as this thesis. The same research suggested using a VMM instead of the conventional GMM, which is also a new development.

Because the engine might break, if the output error is too large and it is costly to run a process the larger the error is, it is natural to think normal mode is when the process output is a reasonably small error. With this in mind, PID controllers can exhibit several examples of abnormal behavior. There are few examples of normal behavior, even fewer of optimal behavior. There are so many abnormal examples that it is easier to create all examples of normal behavior. Since it is easier to find examples of normal behavior, PID controllers are interesting topic to research novelty detection.

1.2.2. Genetic algorithms

The use of a genetic algorithm was to create several PID parameters that were realistic. These parameters would be used to test the analyser with. Since we needed both well tuned and poorly tuned parameters, the genetic algorithm would have to tune PID parameters. Genetic algorithms have been used for both tuning PID- and other control systems and creating test parameters for computer software. Below is described some example works of these.

There are several previous works on using genetic algorithms to tune PID controllers and other control systems. In Törmänen (1997), genetic algorithms were used for tuning directly the PID parameters of a controller. In Goldberg (1985) genetic algorithms were used in a learning classifier system for tuning parameters of a simulated natural gas pipeline.

In Kwok & Sheng (1994) genetic algorithms were used to tune six PID controllers, each one controlling one joint of a robot arm. In the same work a method of using simulated annealing for the same purpose was used. The results were compared to a random search method and an empirical method in two experiments of following a circle and step motion tracking. Both genetic algorithms and simulated annealing outperformed the two other methods. Out of the two, genetic algorithms slightly out performed simulated annealing.

In Mitsukura, Yamamoto & Kaneda (1999) a genetic algorithm was used as a part of a self-tuning PID process. The self tuning was done online, so the genetic algorithm found parameters for a generalized minimum variance control (GMVC) system, which was used to derive PID parameters.

There are also several examples of genetic algorithms being used in producing test data for software. Mantere & Alander (2005) mentions uses in interface testing by digital or analog input, Ethernet calls and even finding out how fast users learn to use an interface. Another example is Srivastava & Kim (2009), where a genetic algorithm was used to find vulnerabilities in software. Here the genetic algorithm outperformed exhaustive and local search techniques.

In Michael, McGraw, Schatz & Walton (1997) a genetic algorithm was compared to a random test data generator. The methods were used to create test data for a closed loop fuzzy controller and an automatic pilot controller system. Also a library of 10 different math problems, such as bubble sort and computing the median was used to test the methods against. The genetic algorithm outperformed or was at least as good as the random generator in all tests. Especially it performed better in more complex problems.

In Mantere & Alander (2001) genetic algorithms were investigated for test image generation. The test images were created for testing different halftoning methods. It is concluded that the genetic algorithm was successful in most cases. In the other cases, the genetic algorithm found values close to the highest value reached with a static image test.

1.3. Structure of this thesis

In chapter 2 we explain the theory behind the method used in this thesis. Novelty detection, PID controllers and genetic algorithms is presented. In chapter 3 we explain how the methods are modified from their typical applications and how they are used in this solution. In this chapter we present the fitness function and Simulink model used in simulations. Also the criterions for selecting normal and abnormal parameters are presented in this chapter.

Chapter 4 explains the setup of the experiment, presents results of the experiment and assesses the results. The first part presents the genetic algorithm and the parameters for it. The resulting PID parameters are assessed whether they are suitable for testing. We explain how we put together the sets for testing the analyser from the resulting parameters. The second part focuses on the experiments conducted on the analyser. We present the results and assess the performance of the analyser.

Chapter 5 gives an overview of the problem, gives a summary of the experiments conducted and the results from them. Finally there is a discussion about results and ideas for future work is presented.

2. THEORY

The theory for this thesis is drawn from three different areas: novelty detection, PID controllers and genetic algorithms. Novelty detection is the theory by which we are investigating whether it is suitable for classifying PID controllers by their normality. The analyser is based on this theory.

To test the analyser we need to know beforehand if the PID controller values are normal or abnormal. To do this we need to simulate PID controllers, here the theory for PID controllers come in. A genetic algorithm is used to produce the parameters for testing the analyser and thus plays an essential role in this work.

2.1. Novelty Detection

Novelty detection is best used in an application where examples of normal behaviour are easily found but examples of abnormal behaviour is more difficult to find. When we know what are the signs of normal behaviour, we can classify the behaviour of unknown examples. (Clifton et al. 2011: 371)

Novelty detection uses extreme value theory, which is a method originally used in statistics and economics. The original application of novelty detection were only concerned with a single variable in a two dimensional space, so they use a univariate approach. Later novelty detection began to be used in other applications, such as patient and engine health monitoring. In these cases we need to look at several variables, which is why a multivariate version of EVT has been developed. (Clifton et al. 2011: 371-373)

First we assume a set of normal data $\{\mathbf{x}_1,\ldots, \mathbf{x}_n\}$ is independent and identically distributed. Then we assume they are distributed according to a probability density function (pdf) denoted f_n. The pdf is a representation of the data space of possible values and is approximated using a mixture model, usually GMM is used. Then we set a threshold k, such that any test data $\mathbf{x}$ is considered abnormal if $f_n(\mathbf{x}) < k$, where k is defined using multivariate extreme value statistics. (Clifton et al. 2011: 372-373)

Now we introduce a probability distribution function F_n. With F_n we can define a probability such that if we were to draw a value from f_n it would fall outside the novelty threshold with probability $1 - F_n$. A calculation of F_n is presented in Equation (1). (Clifton et al. 2011: 372-373)

$$F_n = \int_{f^{-1}_{]0, f_n(\mathbf{x})]}} f_n(\mathbf{x})\,\mathrm{d}\,\mathbf{x} \tag{1}$$

We approximate the multivariate multimodal distribution by a single Gaussian kernel. We do it only for the ends of $f_n(\mathbf{x})$, i.e. where f_n is close to zero. This makes the probability distribution a univariate function. After approximation we can use extreme value theory for univariate data. In this case we look at the distribution of the data and find the areas with the most extreme densities. Here we consider the most extreme cases to be the ones that are the most improbable ones. That is if we took a set of sample normal data from f_n and used it as our extreme value distribution (EVD) and put a novelty threshold on it. Thus any dataset which is more improbable than the EVD values will be considered abnormal. (Clifton et al. 2011: 372-377, 381-384; Vesterback et al. 2012: 410-411)

Usually a GMM has been used for data distribution. However, using this distribution requires stating the number of clusters. This means that when one uses the GMM, one has to try different numbers of clusters and evaluate how well each one fits the data. This is called fitting. The analyzer uses a VMM, which chooses the number of clusters automatically. Using the VMM means it will produce varying results. This is why one should still run the algorithm a few times and choose the simplest fitting model. This means the model with least Gaussian kernels, or clusters. This is still easier than giving different numbers of clusters and comparing the results. (Vesterback et al. 2012: 405, 412)

2.2. PID controllers

Process control has a wide field of applications. Different feedback control systems, including PID controllers have uses in household appliances, industrial automation devices, autopilots for airplanes, temperature control, and power plants to name a few. (D'Azzo & Houpis 1995: 1-6; Chen 1993: 2-7) Different control applications have existed throughout history, the first widely used control device is believed to be Watt's governor in 1788, which regulated steam engines. Maxwell (1868) made his contribution for governors and control in general in his paper. However modern control began to be developed in the beginning of the 1900's. Especially Black's invention of the negative feedback loop for telephone amplifiers in his paper 1934. Negative feedback loops are always used in PID control systems. (D'Azzo et al. 1995: 8-12; Black 1934; Maxwell 1868; Chen 1993: 552-558, 561-563)

PID controllers were also invented during the beginning of the 1900's. Ziegler & Nichols (1947) mentions that it was common that controllers at their time had some combinations of three components: a proportional action, an automatic reset action i.e. integral and a predictive i.e. derivative action. These are the three components of a PID controller. In this paper Ziegler and Nichols laid ground for empirical tuning rules that carries its authors name and is still used today. PID controllers have carried on until today; Mantz & Tacconi (1989: 1465) mentions them at the time being the most popular. Åström et al. (1995: 1) mentions them being popular and recently published Blevins (2012: 1) also mentions them being the most popular controller types.

The idea of a process control system is to manipulate the input of a process so that the output will be the desired reference value. In this section we refer to the engine or machine being controlled as a process. Usually in control applications a negative feedback loop is applied, an example is presented in Figure 1. It is simply the process output measurement value $y(\mathrm{t})$ connected backwards to influence the input value of the process. The output $y(\mathrm{t})$ will be subtracted from the desired value $y_{sp}(\mathrm{t})$ to get the error $e(\mathrm{t})$. (Åström et al. 1995: 5-6)

This corrects disturbances and also changes the output when needed. If the output y(t) is larger than the reference value, setpoint y_{sp}(t), the error e(t) will be negative and the process output starts falling. If a controller is added, it adjusts the output in such a way that the process will achieve the reference value faster. (Åström et al. 1995: 5-6)

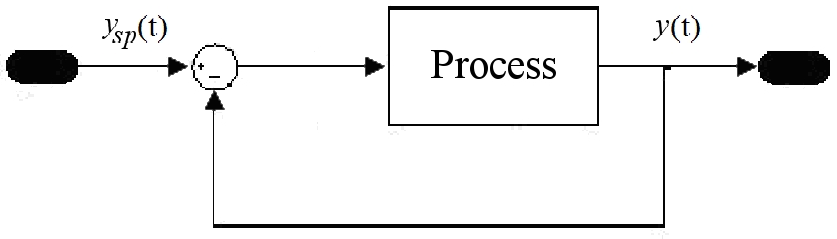

Figure 1. Blockdiagram of a process with a feedback loop. The process block signifies the engine, y(t) is the measured output value from the process, y_{sp}(t) is the setpoint, i.e. desired value for the process output.

The relationship between y(t) and y_{sp}(t) is determined by the process model. The units for y(t) and y_{sp}(t) doesn't have to be the same, as long as they are related. (Fröhr & Orttenburger 1982: 11-19) To obtain a process model one uses basic physics formulas to calculate the effects of the process on the outcome. Usually in linear systems these relationships are expressed by differential equations where the functions are functions of time t. The methods for control are universal and can be applied in several different fields, such as mechanical-, electrical-, and hydraulic systems. (Chen 1993:14-27)

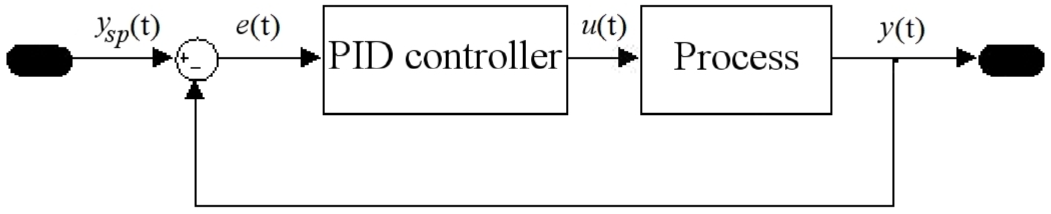

Figure 2. Example block diagram of a process with a PID controller in time domain. The PID controller block is added to signify the controller, e(t) is the error signal, which is the separation between y_{sp}(t) and y(t). Between controller and process there is the control value u(t).

Figure 2 shows a simple PID controller system. The input signal $y_{sp}(\mathrm{t})$ is called set point, which is the desired value of the process, $y(\mathrm{t})$ is the system output called process variable, $u(\mathrm{t})$ is called control variable, $e(\mathrm{t})$ is the control error, which is the difference between the setpoint and process variable $y_{sp}(\mathrm{t}) - y(\mathrm{t})$. In other words, this is the error between the desired value and system output, which is then given to the PID controller for correction. (Åström et al. 1995: 5-6)

Next we will introduce the Laplace domain before we go to look at the PID components more closely. Sometimes differential equations are difficult to calculate with. To make it easier to perform calculations on control systems, one can transform the model from the time domain into the Laplace domain. When the model is transformed into the Laplace domain, derivatives and integrals are replaced with the Laplace variable s and $1/s$. Now one can use ordinary arithmetics when calculating with these functions. (Kreyszig 2011: 203-204) Another use of the Laplace domain is one can find the transfer function of the system and each subsystem. The transfer function is the relationship between input and output (Fröhr et al. 1982: 216-221).

Equation (2) shows the Laplace transform on a time domain function and shows how the Laplace variable s becomes the function variable. (Kreyszig 2011: 204-205) The Laplace variable s is a complex number, presented in Equation (3). The symbols stand for decay ratio σ, and angular frequency ω. If one substitutes s with $j\omega$, one can study the frequency response of the system. This is why the Laplace domain is also called the frequency domain. (Fröhr et al. 1982: 37-39).

$$\mathrm{U}(s) = \mathrm{L}\{u(\mathrm{t})\} = \int_{0^-}^{\infty} u(\mathrm{t})\,\mathrm{e}^{-s\,\mathrm{t}}\mathrm{dt} \tag{2}$$

$$s = \sigma + j\omega \tag{3}$$

Figure 3 shows the model given in Figure 2 in the Laplace domain. Here we have taken the Laplace transform of each signal in Figure 2. Now one can find the transfer function for the system. (Åström et al. 1995: 5-6, 64-70; Chen1993: 39-44, 94-98)

PID stands for Proportional, Integral and Derivative. These are the three parts of the controller and each one has its own influence on the control system behavior. All of them use the control error *e* in some way. (Åström et al. 1995: 70)

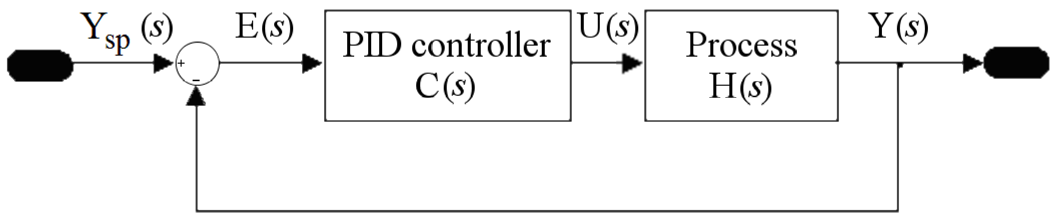

Figure 3. Block diagram of PID controller and process in Laplace domain. Now the functions are denoted with capital letters because they are in Laplace domain. The function argument *s* is the Laplace variable. (Chen 1993: 39-40, 567)

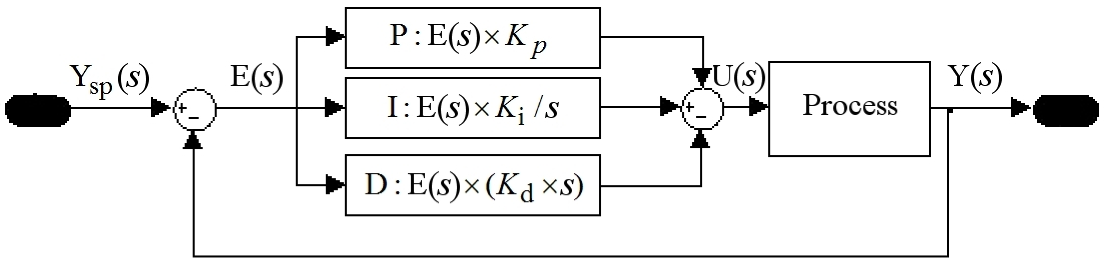

Figure 4. Block diagram of PID controller and Process. This Figure shows in detail the components of a PID controller. *P*-, *I*- and *D*-blocks signify the three components in a PID controller. K_p is the proportional coefficient, K_i is the integration coefficient, and K_d is the derivative coefficient. U(*s*) is found by summing all these three components. (Åström et al. 1995: 71)

The Proportional part of the system has no time delay and reacts on the control error. It adds to the control value to the process by multiplying the error *e* with a gain constant K_p. (Fröhr et al. 1982: 125-129; Åström et al. 1995: 64-67)

The integral part is used to eliminate steady state error. The integral part has a time delay depending on the integration time T_i. If the T_i is increased, the output will slowly creep towards the setpoint, if T_i is smaller the setpoint will often be reached faster but the output will oscillate more. In theory the integral part sums all the error from each measurement, i.e. it takes into account all the previous error, and multiplies it with a weight constant K_i before adding it to the control value. (Fröhr et al. 1982: 129-133, 156-163; Åström et al. 1995: 67-69)

The derivative component derives the input, i.e. it tries to predict future error. This is done by linear extrapolation and thus also has a time delay T_d. The derivative part also multiplies the output with its own weight constant K_d. The output from these three parts are summed to determine the input for the process. (Åström et al. 1995: 64-70)

Equation (4) presents the PID controller equation in time domain, where coefficients K_p, T_i, and T_d are presented in this Equation. Equation (5) and (6) presents how these are related to K_p, K_i and K_d. These are integration coefficient and derivative coefficient. One can usually calculate K_i and K_d with the formulas given in Equations (5) and (6). To make things simple we will use only K_p, K_i and K_d. We will also refer to them as *P*, *I* and *D* respectively when we speak of them as PID controller parameters. Equation (12) presents the transfer function of a PID controller, which is the Laplace transform of the time domain representation. (Åström et al. 1995: 64, 70-72)

$$u(\mathrm{t}) = K_p\left(1 + \frac{1}{T_i}\int_0^{\mathrm{t}} e(\tau)\,\mathrm{d}\tau + T_d\frac{\mathrm{d}e(\mathrm{t})}{\mathrm{dt}}\right) \tag{4}$$

$$K_i = \frac{K_p}{T_i} \tag{5}$$

$$K_d = K_p \times T_d \tag{6}$$

(Åström et al. 1995: 64, 70-72)

Equation (12) presents the PID controller equation in the Laplace domain (Åström et al. 1995: 70). To get from Equation (4) to Equation (12), one has to apply the Laplace transform on Equation (4). The Laplace transforms for integrals and derivatives are well defined and presented in Equations (7) and (8) respectively (Kreyszig 2011: 211-213). In the applications of this work the initial value u(0 s) will always be 0, because we use these functions in simulations, which start at time t=0. First we multiply K_p into the parenthesis of Equation (4) to get to Equation (9). Then we use Equations (5) and (6) on Equation (9) to get Equation (10). Now we apply the Laplace transform on both sides. The Laplace transform mainly uses integration, which is linear, so we can apply it separately on each of the components, as presented in Equation (11). When we make the appropriate transforms on Equation (11), using Equations (7) and (8) in the process, we get Equation (12).

$$\mathrm{L}\{\frac{\mathrm{d}}{\mathrm{dt}}u(\mathrm{t})\} = s\,\mathrm{U}(s) - u(0\,\mathrm{s}) \tag{7}$$

$$\mathrm{L}\{\int_0^{\mathrm{t}} u(\tau)\,\mathrm{d}\tau\} = \frac{\mathrm{U}(s)}{s} \tag{8}$$

$$u(\mathrm{t}) = K_p + \frac{K_p}{T_i}\int_0^{\mathrm{t}} e(\tau)\,\mathrm{d}\tau + K_p T_d \frac{\mathrm{d}e(\mathrm{t})}{\mathrm{dt}} \tag{9}$$

$$u(\mathrm{t}) = K_p + K_i\int_0^{\mathrm{t}} e(\tau)\,\mathrm{d}\tau + K_d \frac{\mathrm{d}e(\mathrm{t})}{\mathrm{dt}} \tag{10}$$

$$\mathrm{U(s)} = \mathrm{L}\{u(\mathrm{t})\} = \mathrm{L}\{K_p\} + \mathrm{L}\{K_i\int_0^{\mathrm{t}} e(\tau)\,\mathrm{d}\tau\} + \mathrm{L}\{K_d \frac{\mathrm{d}e(\mathrm{t})}{\mathrm{dt}}\} \tag{11}$$

$$\mathrm{U}(s) = K_p + K_i\frac{1}{s} + K_d s \tag{12}$$

With the transfer function of a system one can find out whether the system is stable. A system is stable when a finite input produces an output which is a finite value when time approaches infinity. An analytical way of investigating stability is studying the system transfer function. If all the poles of the transfer function are in the left half complex plane, the system is stable. That is all the real parts of the roots are less than zero. (Chen 1993: 125-129)

Controller performance depends on the tuning of the controller. There are a number of ways to tune a controller. Most of them are based on sets of empirical rules, others are analytical. Today though, lots of controllers come with automatic tuning functions (Åström et al. 1995: 120-121, 134-164, 234) and there are also many automatic tools to help tuning (Mazeda & de Prada 2012: 1), e.g. Matlab having many of them (MathWorks 2012a). Another example of a tuning tool is IFTtune, which you can read about in Mazeda et al. (2012).

The quality of the PID controller and the whole system is measured by the process output with respect to the reference value. There are several different approaches and characteristic values one can measure to judge the quality of a controller. When deciding what values one wants to measure and/or calculate, one should think about the goal of the controller and system one uses to judge a controller. Typical goals are attenuation of load disturbance, sensitivity to measurement noise, robustness or setpoint following.

The output can be measured in different ways. One is in time domain, when one studies the time response. The other one is the frequency response, which is studied in the Laplace domain. In this work we will focus on the time domain. (Chen 1993: 195-197, 270-272) What we will investigate is the transient and steady state behaviour of the controller. Transient behaviour happens when the input value changes. This will result in an error between the reference value and output, which will be corrected by the controller with a time lag. If the error isn’t corrected in a finite time, it is steady state error. (Fröhr et al. 1982: 20-24) When measuring the transient output one usually looks at how well the output follows the setpoint. (Chen 1993: 195-197)

This method is called setpoint following, where we look at the error between the two signals. After choosing the test function, we are presented with choices between different performance criterions. Examples of different criterions that can be used is presented in Figure 5. (Åström et al. 1995: 127-129)

When one has decided the goal, there are a number of criterions one can look at when assessing the quality of a controller. In the case of setpoint following some examples are: rise time; the time it takes for the signal to reach the setpoint, settling time; the time it takes before the signal reaches a certain threshold within the setpoint, attenuation; ratio between two spikes following each other also, overshoot; the places where the output exceeds the setpoint, and the sum of absolute errors; is the integral of the absolute value of the difference between the output and setpoint functions. These examples are also presented in Figure 5. (Åström et al. 1995: 121, 127-129) Last but certainly not least is stability. Stability is a very important criterion because unstable systems might wear out faster over time or even break, because of a single change in the reference value. (Chen 1993: 125)

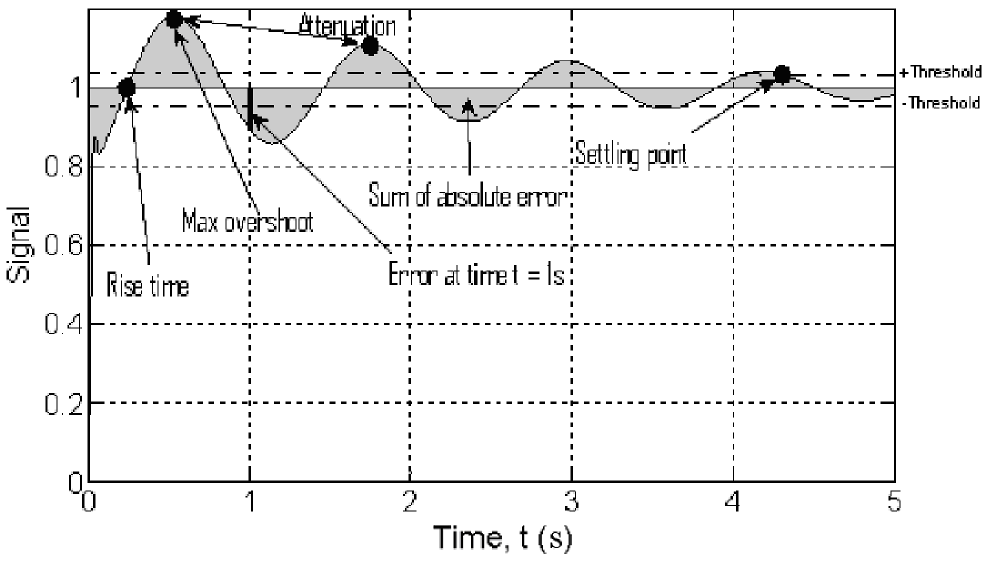

Figure 5. Different criterions for evaluation of a PID controller with step response. The error between setpoint and output is marked with the gray area.

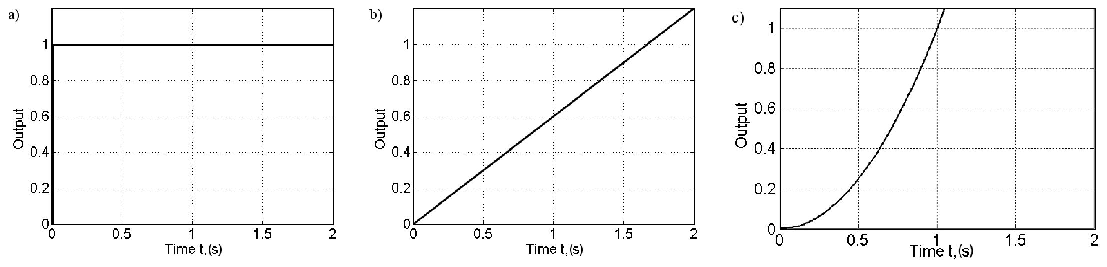

Figure 6. Examples of different step functions that can be used as reference value y_{sp}(t). In image a) y_{sp}(t) = 1 after t = 0 s, in b) y_{sp}(t) = t and in c) y_{sp}(t) = t^2.

Another important choice before simulation is the choice of test function. A test function is a typical input values, which will be used as reference value y_{sp}(t) when simulating. Some examples of test functions are the step function, ramp function and acceleration function, all of them presented in Figure 6. Choice of step function should be made according to what the probable reference value will be in the practical application. (Chen 1993: 138-141)

2.3. Computational intelligence

Just as many other optimisation methods, genetic algorithms work by adjusting current values to move towards more optimal solutions on the cost surface. With genetic algorithms, adjusting is based on statistical theory. The algorithms work with both continuous and discrete values. They also use only an objective cost function, which doesn't need derivatives of the cost function or other auxiliary information, but only the goodness values of the solutions.

Cost surface is the outcome of all possible values from the cost function, also called parameter space. Cost surfaces varies between high and low spikes. High spikes can be thought of as hills and low spikes as valleys. In minimum seeking algorithms the aim is to find the lowest spike. A usual problem with optimization algorithms is that the deepest valley found might not be the lowest point on the whole surface. The lowest point on a valley is called the local minimum or optimum, while the lowest point on the whole cost surface is called the global minimum or optimum. The same applies in reverse for maximum values. Many optimization methods often get stuck in local optimum.

Optimisation algorithms usually have a cost function and the goal is to find the parameters that produce the optimal outcome for that cost function. In the field of genetic algorithms, these are usually called fitness functions. There are other optimisation methods as well, like trial and error, brute force and analytical, which uses calculus on a cost function. Optimization problems also differ whether they are discrete or continuous, static or dynamic, single or multiple variable. Choice of optimization method should be chosen according to the problem. (Goldberg 1989: 2-7, 10, 75-76, 202-204)

2.3.1. Genetic algorithms

A genetic algorithm is an optimisation method which mimics the evolution process of nature. The idea was developed by John Holland and his colleagues over a the 1960s and –70s, and had been proven robust even in complex search spaces both theoretically and empirically. (Goldberg 1989: 1-2; Holland 1992: 66-67, 71-72)

The advantage of a genetic algorithm includes that it simultaneously searches a large range of the cost surface, can deal with problems with a large number of variables, can analyse complex cost surfaces, works on multimodal search spaces and, can give a set of optimal values instead of only one solution. However genetic algorithms is not the best method for all problems. E.g. if auxiliary information, such as derivatives are easily attainable, the genetic algorithm might perform worse than solutions designed for the particular problem. (Goldberg 1989: 7-9, 15-20; De Jong 2006: 6-19)

The most basic idea of genetic algorithms is combining pieces of different ideas or configurations, the so called building blocks, that show good potential. In addition to combining ideas the algorithm uses an operator that slightly changes the configuration, called the mutation operator. The best of these new genes are selected and new configurations are created out of these in the same manner.

The reason why this works is, when repeatedly combining parts of good solutions and always selecting the best out of those, some of the solutions which show good results will have more copies in the population. Also the top solutions can be different, but some of them will have similarities, i.e. part of the configurations have the same value, this is called implicit parallelism. The partial configurations are called building blocks, or schemata. The best building blocks getting more representation is called them getting more market share growth and the phenomenon is called the building block hypothesis, according to Goldberg (2002: 7) and De Jong (2006: 192-199) this was presented in Holland (1975).

There is no need for separate bookkeeping on which building blocks are good or bad. We can be sure that the best building blocks get market share growth because the best solutions come from certain regions in the search space and thus the best selected will have some similar parts. This will cause the genetic algorithm to converge towards local optimums. New local optimums can be found through the combination of building blocks. The neighborhood is explored better with the mutation operator. The mutation operator also ensures there's no important genetic information lost during the way. This prevents premature convergence, where the algorithm converges fast towards a few local optimums without exploring the cost surface thoroughly. (Goldberg 1989: 6-14, 18-23; Goldberg 2002: 3-6)

2.3.2. Genetic algorithm terminology

A genetic algorithm is used to create parameters of PID controllers in this thesis. Therefore it is important to know some of the terminology and concepts of genetic algorithms.

Individual, chromosome, gene, fitness value

An individual in a genetic algorithm is one possible solution to the problem. Individuals can also be called chromosomes or a point on the cost surface. One individual contains one solution to the problem. These values are called genes. In the case of the PID controllers an individual can consist of three positive integer values P, I, and D.

Each individual also has a fitness value. The fitness value measures the goodness of one solution on the cost surface. When an individual has a fitness value, the individual can be compared with other individuals and they can be ranked. (Goldberg 1989: 10, 21)

Conventionally configuration parameters for chromosomes have been encoded with binary strings or real values. Both of them have strengths and weaknesses as well as different ways one can program the algorithm with. With binary encoding one can only use discrete values. Also binary numbers can produce redundant values. E.g. representing the number 18 in binary numbers would require 5 digits but if 18 is the maximum value for a parameter, numbers 10011_2-11111_2 will not have anything to represent. Strengths of binary coding include that it can produce entirely new numbers in crossover, depending on the crossing points.

Binary representation uses a low cardinal alphabet; only 1's and 0's are used to represent values. In alphabets with higher cardinality one is forced to use higher population sizes to get all the different alphabet members represented. Also binary representation supports the so important implicit parallelism better. (Herrera, Lozano & Verdegay 1998: 276-277)

Real valued representation doesn't have to be decoded for fitness calculation and thus saves processing time. It is found that this is more useful for problems with continuous values requiring precision. Also more useful when the parameters can have many different values and the binary representation of a chromosome would get very long. (Herrera et al. 1998: 281-282, 287-300)

Allele, schemata, building blocks

An allele is a genetic trait or characteristic. For example blue eyes is a trait. An allele can be the value for a parameter. In the case of PID controllers, the value 5 for the *P*-parameter is a characteristic of that individual.

A schemata is similar configuration for an individual. Let's say we have chromosomes, which have five binary numbers as their parameters. Two of these individuals could be 11001 and 11110. These individuals have the same schemata of 11***, the stars in this example are any value. Also schemata can be spread over the chromosome, e.g. 01110 and 01010. These have the common schemata of 01*10. At the same time they have the common schemata of 0***0 and *1*1* among others. Building blocks are short and highly fit, that survive over generations. They get combined with different building blocks and this way gets more market share growth. (Goldberg 1989: 21, 40-41)

Population

The population is a group of individuals. In this work we usually refer to the current generation by 'population'. The number of chromosomes in a population is usually set at the beginning of the genetic algorithm. (Goldberg 1989: 60-62) It is known that the quality of the following populations correlate on the first population (Alander 1991: 1318; Goldberg 2002: 114).

Generation, initial population

A generation starts from selecting individuals according to their fitness value from the current population. Then the next step is to create new individuals by recombining genes from two or more individuals, in a process called crossing. The next step is mutation where we explore the area around an individual by tweaking its genes. Then comes ranking of individuals, using the fitness function. At this stage we have a completely new population than before we started selecting. This is what we call the new generation, which starts from the selection step over again. (Goldberg 1989: 15-18)

The first generation is also called the initial population and is usually generated randomly. Another possibility is to give it ready made chromosomes, which reside in areas where one assumes the best solutions to be. (Alander 1991: 1313,1316)

Fitness function, fitness landscape

Fitness function is the formula which the genetic algorithm uses to calculate the fitness value for each individual. The formula is decided by the programmers according to what they want to analyse. (Goldberg 1989: 10-11; Goldberg 2002: 3). For easier thought, the fitness function can also be called fitness landscape. The performance of the genetic algorithm depends highly on the fitness landscape. (Alander, Zinchenko & Sorokin 2004: 2933-3934)

A genetic algorithm mimics natural selection. In nature the environment often decide what species and what individual traits will be preserved. The ones that are best at coping with the environment survive. (Goldberg 2002: 3, 31-37) In the same way it is also important to choose the right criterions, because they shape the fitness landscape. If chosen correctly the algorithm will output chromosomes with desired values. (Alander et al. 2004: 2934).

Selection

Selection is one operator in the genetic algorithm. When the individuals of a generation have been ranked, the program will select a number of them. The ones not selected will be discarded, and the ones selected will “survive” to the next generation. There are many ways of selecting individuals. Some of the selection methods allows for some of the worse individuals to survive. (Goldberg 2002: 3-4) This is to keep the population diverse, though it is shown in Mantere (2006: 66) that this works poorly.

One category is stochastic selection methods, where each individual is assigned a probability of being selected. The probability is in accordance to the individuals fitness function in relation to the fitness of the population. From here a number of individuals are selected randomly. Examples of proportional selection methods are roulette selection and stochastic universal sampling.

Then there is deterministic selection methods. An example of this is elitism, where the topmost individuals are selected and copied until the population is filled. (De Jong 2006: 54-55)

Crossing

Crossing is one operator where one combines parts of well fit individuals. The aim of crossing is to find new combination of parameters leading to local optimums. Through selection one can also find new local optimums when using binary encoding. This is because one can split up the parameters in bits. Then if the parents differ much, their offspring can also differ a lot from both parents.

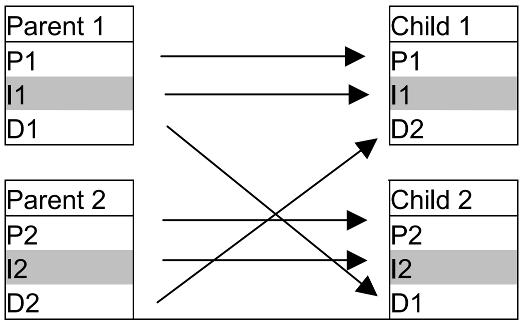

Figure 7. Example of how traits are passed on between chromosomes in uniform crossover. In this case we use *P*, *I*, and *D* as parameters as genes.

Crossing is done after the survivors of the previous generation have been selected. Usually two individuals are selected by random to become parents of two new individuals. Then statistically genes are selected from both parents randomly. These genes are combined to create a new individual and the other halves of the parents are used to create another individual. Often two parents with the same set of genes or two very close to each other are not allowed to cross. This is another rule made in order to keep the population diverse.

There are also other ways of crossover. If the individual is coded as a binary string, the parents can give single binary values. Also the binary strings of chromosomes can be divided into groups of bits. Then the parents give away a number of their groups to their children. An example is shown in Figure 8.

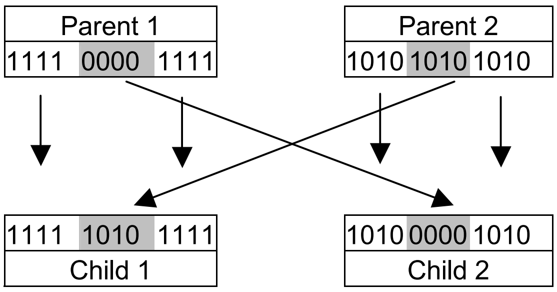

Figure 8. The change of traits when using binary coding. The change is made in certain break points. This example uses 2-point crossover.

A concern with crossing might be how it affects building blocks. In actuality it doesn't affect building blocks much unless the building blocks are far from each other. E.g. if a building block like 1***1* is more likely to be separated than a building block like 11***. (Goldberg 1989: 12- 20; Goldberg 2002: 3-5) This is used in genetics to map traits into chromosomes or distance measurements, and is called genetic linkage. (McClean 1997)

Mutation

The mutation step occurs once for every generation. In the mutation step the parameters of randomly selected chromosomes are slightly modified. When programming bits, the bits are chosen randomly from all bits of the whole population. When programming with real numbers, the mutated individual is first chosen, then one or more of its genes are adjusted by a random positive or negative number. The interval of change is given by the programmers at the beginning, usually the number is small. How many individuals at each generation are mutated is decided randomly according to a mutation percentage set by the user. Mutation rate should be set low, because then it doesn't affect building block growth. This is because this is not the purpose of mutation in the algorithm, but to keep around some information that might be important.

The use of the mutation step is that it helps searching a larger area of the cost surface by exploring the neighborhood of the local optimum where the point is located on the cost surface. If the change is positive, the area will be explored more when the individual gets to mate. (Goldberg 1989: 13-20; Goldberg 2002: 4-6)

2.3.3. The genetic algorithm

Now that we know the basic terms of a genetic algorithm, we can take a look on the basic loop of the algorithm in more detail:

```
1. Generate initial population.
2. Calculate fitness.
3. Selection.
4. Crossover.
5. Mutation.
6. Repeat from step 2 until max number of generations are
   reached or a threshold fitness value is reached.
```

The first step is to create the initial population. Then the fitness is calculated for each individual. After that the individuals are ranked according to their fitness value. This is for the selection step, where the survivors of the generation are chosen, where either the best fitness value survives or the ones with a better fitness value gets better chances at surviving.

Next is crossover, where the survivors mate and have offspring based on their own genes. After crossing comes mutation, where a certain percentage of the survivors get their genes modified. The modification is usually not large and is done on one of the genes of the individual at hand.

After these steps lots of new individuals have been created and some of the old population have been selected to stay in the population. Now we have a completely new population which we call the new generation.

From here the genetic algorithm loop starts. The first thing done with a new generation is that we calculate the fitness values for the individuals. Then the algorithm goes trough all the same steps and goes back to calculating fitness. The loop is iterated a number of times until a satisfactory fitness value has been reached or until a set number of generations have been reached.

(Goldberg 2002: 2-4)

3. METHODOLOGY

A method for predetermining PID controller quality was to be investigated. The method uses novelty detection with MEVS, to classify PID configurations without having to run them in the process. With novelty detection one has to define normal and abnormal behaviour. (Vesterback et al. 2012: 404-405) With PID controllers one is usually concerned about the error between the reference value and process output, and stability. The first one can be calculated with transient response. For the second one, investigating transfer functions is one way to determine stability. (Åström et al. 1995: 5-10) From this it was determined that normal would be a controller that follows well the reference value, with stability as another criterion.

To test the analyser properly we would need a bunch of parameters of varying quality. Part of the goal of this thesis was then to create PID controller parameters for testing the analyser. The well tuned parameters would be used as normal parameters to train the program. Then we would have another set of normal parameters to see how many of those it would erroneously classify as abnormal. Then the last set would be a set of abnormal parameters. This would be used to see how many of those the analyser erroneously classify as normal. After this we can calculate the accuracy of the analyser.

We focused on PID controllers. PI or PD controllers should be treated separately from PID controllers because of having one dimension less. The results would make no sense if these were to be mixed together.

3.1. Genetic algorithms and criterions used in parameter selection

To create several PID configurations of varying quality, it would be good to have a program come up with them randomly instead of trying to tune them by hand. One is a random number generator. There are a few problems with a random number generator; one is that it would not necessarily come up with stable parameters, the other reason was it would come up with parameters that normally wouldn’t even be considered and lastly it would not necessarily come up with any well tuned configurations.

A genetic algorithm was used to create a group of parameters overall good according to an index of goodness presented later. After this we would use a second set of criterions to define them as normal or abnormal. We divide them into three sets: training, normal test, and abnormal test set. When we had the parameters we would test them in the analyser. Then we would see if it can be used in this manner to assess PID controller quality.

Genetic algorithms was well suited for the task at hand because it would produce a list of parameters, not just one. It could come up with well tuned parameters for the task, as well as poorly tuned ones. The parameters would be realistic and the program could be made so that it accepts only stable parameters. The program would also come up with many parameters, of which many would be of good and bad quality. From this set we could choose the most interesting parameters for testing.

Even though genetic algorithms use random number generators throughout the whole session, it is more intelligent than pure random number search generators. This is because most of the new random values are based on the best values from the previous generation. Random number generation is only used to guide the decision making process (Goldberg 1989: 10).

3.1.1. Fitness function

Every genetic algorithm has a way of ranking its individuals. This is done in the fitness function according to a formula. The genetic algorithm would be used in this work to come up with a set of overall good parameters. In the fitness function we used Simulink to simulate the PID controller. The unit step function was used as input.

The output would be used in a formula to calculate fitness. The formula was chosen according to the goal of creating overall good parameters. A similar function was used in the Törmänen (1997), where the genetic algorithm combined with fuzzy logic was employed to configure PID controllers. This function produced good results to find reasonably good parameters in that work. The function is presented in Equation (13) (Törmänen 1997: 32).

$$\frac{1}{e_{\Sigma} \times \mathbf{A} + e_{\max} \times \mathbf{B} + e_{\infty} \times \mathbf{C} + 1} \tag{13}$$

A, **B** and **C** in Equation (13) are weight coefficients. They were chosen by Törmänen (1997: 37) as follows: **A** = 0.02, **B** = 100 and **C** = 1. In other words we used three criterions to calculate fitness, each one with a weight on them: *sum of absolute error* e_{Σ}, *maximum error* e_{max}, and *final error* e_{∞}. Since our genetic algorithm has a single fitness function we take a weighted mean from all of these values. An example of the criterions is visualized in Figure 9. An other, more complicated option would have been to use Pareto optimization, which optimizes multiple objectives on a Pareto optimal front (Goldberg 1989: 197-201). We chose to follow Törmänen's example for of a more simple application.

The criterions Törmänen used are also mentioned in many works as good ways of estimating performance of a PID controller (Åström et al. 1995: 127; Haugen, Fjelddalen, Dunia & Edgar 2007: 8-12). e_{Σ} gives an indication of the stationary error (Åström et al. 1995: 128). The $e_{\max}$ is interesting because if it is too high it might break the machine operating the process (Haugen et al. 2007: 8-9), while e_{∞} also indicates the stationary error. $e_{\max}$ is also an indicator of how well the process reacts to change and e_{Σ} is an indicator of response times.

The stability of the PID controller was also important; there is no sense in trying out unstable ones in the analyser. This was the fourth criterion. Any unstable controller configuration would be discarded by the genetic algorithm automatically.

In section 2.2 we stated that the test function should be chosen according to the practical application. We chose the unit step function was used as reference value, because it tests the system for sudden changes in setpoint or disturbances. It is also the easiest to use because it only needs a starting and finishing value. Any test function could have been used, as long as the same function was used to evaluate each controller. (Fröhr et al. 1982: 22-23)

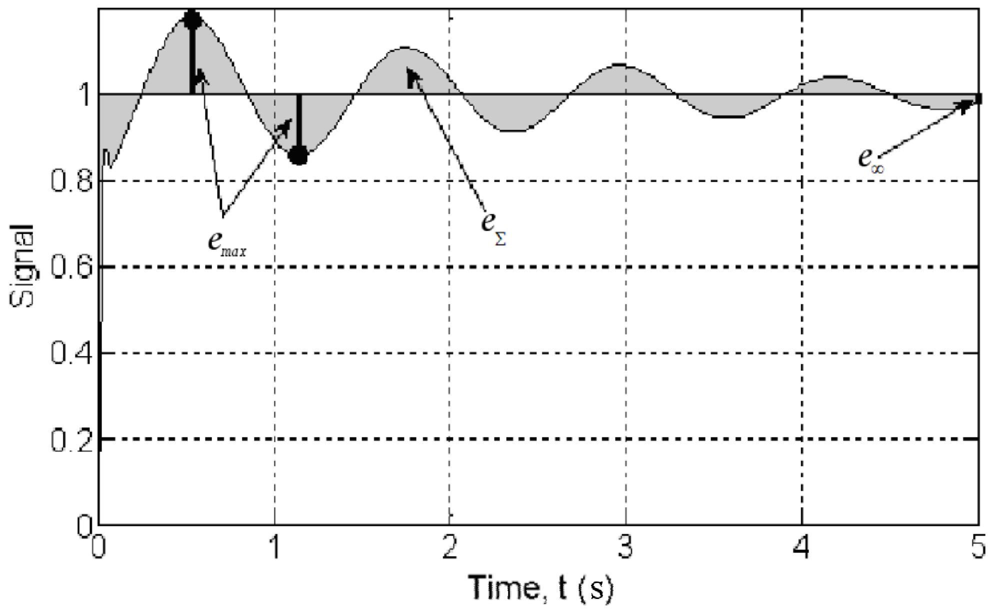

Figure 9. Step response of criterions used in our fitness function. The step function is the straight line at point 1 on the signal axis, the process output is the curving line. The gray area is the error between output signal and reference signal. e_{max} is found by finding the largest deviance of the output and reference, on the image this is either one of the dots on the output signal. e_{Σ} is found by calculating the surface gray area. e_{∞} is found by taking the absolute value of the separation between reference value and the output signal.

3.1.2. Selecting normal and abnormal PID parameters

After the genetic algorithm had produced parameters for us we would use a second set of criterions for classifying the parameters as normal or abnormal. These criteria were, *settling time* t_{s}, and *the maximum overshoot* o_{max}. e_{Σ} was also used in this case and is the same as in the fitness function of the genetic algorithm. t_{s} is the point in time when the output signal reaches a boundary within the setpoint and never goes over it again. o_{max} is the error where the output signal is at its highest after the setpoint. If the output never goes over the setpoint, maximum overshoot will be zero. An example of these are visualized in Figure 10.

Settling time t_s is also a very interesting criterion in practice, (Haugen et al. 2007: 10-11). Also previously we had $e_{\max}$ after reaching the reference value for the first time, now maximum overshoot was used. $o_{\max}$ is much like $e_{\max}$, except that $e_{\max}$ indicates more the extra expense of running this controller, which is already taken into account in the sum of error.

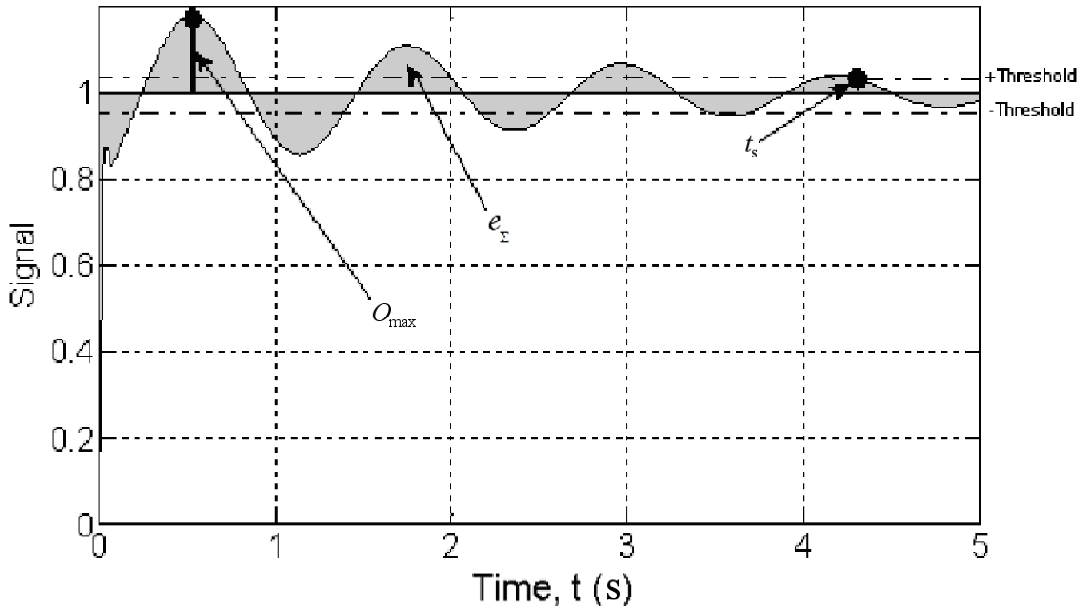

Figure 10. Step response of secondary criteria used. The dashed line marks the threshold set where the function will have to settle within. Settling time t_s, is the point in time when the output goes within these two lines and never crosses them again. Overshoot $o_{\max}$ is the value where the output signal is at its highest point over the reference value. Sum of absolute errors e_Σ is the same as in Figure 9.

Each criteria has its own threshold. If a parameter configuration would exceed one of these thresholds, it would be classified as abnormal. The thresholds we used were designed to give reasonable and efficient results. The threshold for $o_{\max}$ was set to 10%. Settling time t_s was put to reach 2% range of reference value within 0.75 s. After investigating e_Σ from the previous run and comparing them to the plot of the same parameter configuration we found that some configurations wouldn't be disqualified by settling time nor maximum overshoot, but weren't really good either. The sum of error e_Σ mostly will cut controllers with high constant error and some with a combined high $o_{\max}$

and t_s. We drew the line for e_Σ at 450, because it seemed to pass configurations that were a little bit better than mediocre.

3.2. Simulation

To calculate the fitness we would have to find out the output signal for each configuration. To do this we used Simulink to simulate the PID controllers.

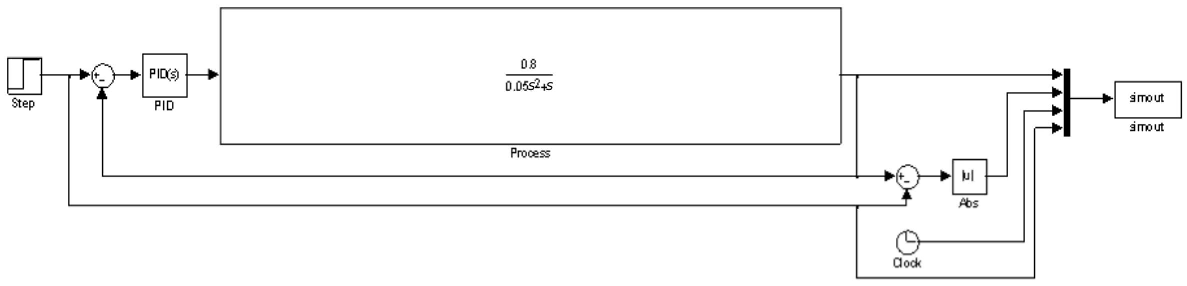

Figure 11. Simulink model used by the genetic algorithm to simulate PID controllers.

Figure 11 presents the Simulink model used to simulate PID controllers. The model would be run from the fitness function. The `Step` – block represents the input and produces a unit step signal. The `PID` – block represents the PID controller, its values are inserted by Matlab before each simulation. The `process` – block represents the engine. The `Abs` – block outputs the absolute value of what goes in. In this case it is the difference between reference signal and system output. The thin black box on the right side with four signals going in is the `Mux` – block. The `simout` – block transfers its input to Matlab workspace.

We chose a simple engine transfer function to avoid unnecessary complexity. Inspiration for the function was taken from Mohammed & Koivo (2005: 1-2). The transfer function is from a diesel engine with actuator. It is presented in Equation (14).

$$\frac{0.8}{0.05s^2 + s} \qquad (14)$$

For setpoint we used the unit step function, which rises to one and stays there for the rest of the simulation. The unit step function is a popular way of investigating the transient response of a system. The simulation time was set to 5 s, so it would not cause too many measurement values for the algorithm to become too slow. (Åström et al. 1995: 9)

The model also had to be given a sampling time or else different runs would turn up with a different amount of measurements. This would cause the controllers to get unfair evaluation for the fitness function when calculating e_{Σ}. We gave the model a sampling time of 0.0001 s. With a 5 s simulation time, this meant 50 001 measurements in one simulation.

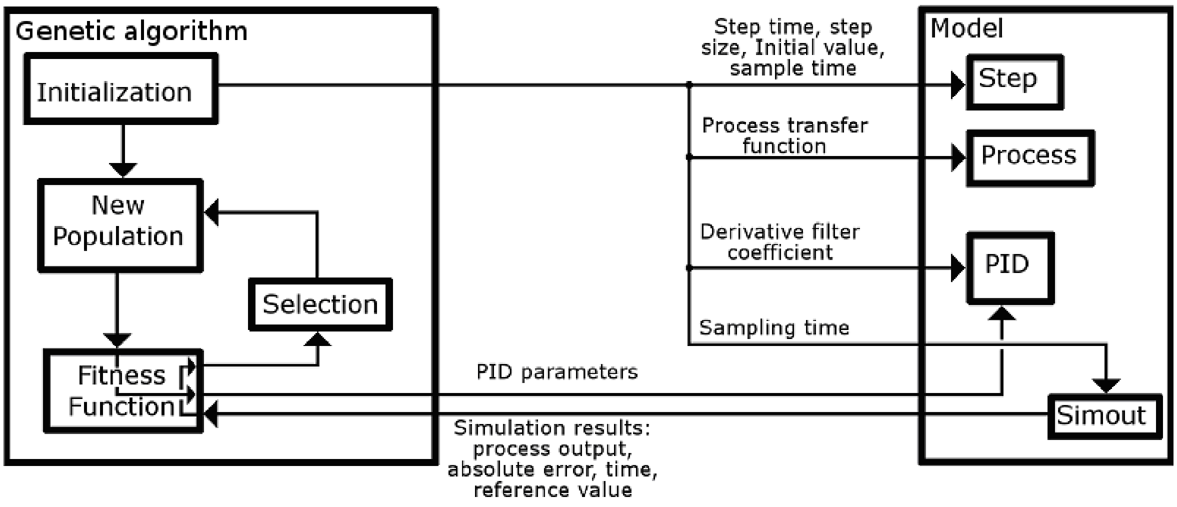
Genetic algorithm
Initialization
New Population
Selection
Fitness Function
Step time, step size, Initial value, sample time
Process transfer function
Derivative filter coefficient
Sampling time
PID parameters
Simulation results: process output, absolute error, time, reference value
Model
Step
Process
PID
Simout

Figure 12. A block diagram of the relationship between the genetic algorithm and the Simulink model. At the beginning the genetic algorithm gives the Simulink model parameters for the step function (step time, step size, initial value and sample time), process (process transfer function), PID-block (derivative filter coefficient), and sampling time to the simout-block. The fitness function in the genetic algorithm gives each time new parameters for the PID controller, and uses the data from the Simout-block to calculate fitness for the individual.

Figure 12 presents the relationship between the genetic algorithm and the Simulink model. At the beginning the genetic algorithm gives the model step time, step size, initial value, sample time for the step block, transfer function, derivative filter coefficient and sampling time for the `simout` – block. The fitness function gives the controller it's PID parameters and runs the simulation. The results are used in the fitness function to calculate the fitness value for the individual. (Vesterback et al. 2012: 408-409)

3.3. Calculating fitness

Before calculating fitness we would create a transfer function in Matlab with the `tf` – function. The transfer function of the PID controller was based on its own parameters as presented in Equation (12). After this we would combine this transfer function with the engine transfer function by using the `feedback` – function. This function returns the transfer function of the processes in series with each other, with a feedback loop around them. This is called the system transfer function. When we have the system transfer function, we can use it to calculate the stability of the controller by using the `isstable` – function. (MathWorks 2012b)

When the simulation was complete the `simout` – block, presented in Figure 11, would transfer to workspace 4 vectors. They included 50 001 measurements each, taken in steady intervals over the simulation. The first vector is the output from the system $y(\mathrm{t})$, the second is the absolute value of the error $e(\mathrm{t})$, the third is the time t at each measurement point and the last one is the setpoint y_{sp}.

To calculate fitness we need to know e_{Σ}, $e_{\max}$, and e_{∞}. The sum of errors e_{Σ} was calculated by summing all the values in e. This was implemented with the `sum` – function. To find $e_{\max}$, we first found out at which point in time the output value y reaches the setpoint y_{sp}. Then we find out which value is the largest in vector e after that point. That value would then become the maximum error. The asymptotic error e_{∞} was simply found by taking the value at $e(5\ \mathrm{s})$.

After finding these three values we would take the predetermined coefficients **A**, **B**, and **C** for calculation of the fitness as presented by Equation (13). Out of the four vectors we have not still used vector 3, the time. This is used later when we select normal and abnormal parameters.

4. EXPERIMENTS

4.1. Modelling the engine PID parameters

Usually when choosing parameters for a genetic algorithm, one wants to find a balance between converging fast enough to find good solutions and converging slow enough to explore the cost surface enough. (De Jong 2006: 49-69) This means one has to have a mix of mechanics that makes the algorithm find values from larger areas and still converge to some point or points. With our goal of finding as many parameters as possible, it was more important to search a wide area than to converge fast. According to Alander (1992: 65) execution efficiency is only a factor when choosing population size, which can lead to long processing time or that the algorithm doesn't find the global optimum before the maximum number of generations is reached. This is why we chose to have the algorithm search for a large number of generations, using a large population. We also did not have it stop when a satisfactory fitness value had been found, but had it continue until a predetermined number of generations was reached.

We chose a high number of generations and a large population size to get a huge number of different parameters. Also crossing was not allowed for parameters between parents with exact same genes. The genetic algorithm was set to run for 50 generations with an initial population of 1000 randomly created individuals and a normal population size of 1000.

The range for the parameters were set to range from 0.5 to 1 500, where the minimum difference between two parameter values were 0.5. We tried to run the algorithm between 0.5 and 15 000, but it returned worse results.

Because mutation at a low rate doesn't affect building block processing (Goldberg 1989: 20), mutation rate was set to 1%. This was implemented by randomizing at each individual between 1-100. If it was 1, the individual would mutate. The best individual was not allowed to mutate. When an individual was selected, one of its parameters was selected. Then a number between 0.5 and 10 was selected. The parameter was changed up or down, also selected randomly, according to that number. The parameter could not go below 0.5 for *P* and not below 0 for *I* and *D*. Mutation is an operator for exploring the neighborhood and ensure that genetic information is not lost too early. That is why we didn't have a too large mutation-interval that the parameter could change.

For crossing we had the top 100 to survive to the next generation and selected 300 individuals through roulette selection. Then each of the survivors were crossed. The pairings were randomly selected so each individual crossed once. After that we had 80% filled of the new population. The remaining 20% were filled by creating new individuals randomly.

Table 1. Top 10 parameters given by the genetic algorithm.

#	*P*	*I*	*D*	Index of goodness
1	47.5	3	4.5	0.13297
2	47.5	3	2.5	0.13040
3	43	3	4.5	0.12059
4	47.5	5.5	4.5	0.11961
5	47.5	5.5	2.5	0.11642
6	47.5	7	4.5	0.11391
7	47.5	8	4.5	0.11078
8	47.5	7	2.5	0.1105
9	43	5.5	4.5	0.10800
10	47.5	8	2.5	0.10725

The program returned about 22 000 unique parameter configurations. To assess whether these parameters were what we wanted them to be, we have given the ten best parameters in Table 1. Also plots of the step response for a few selected PID configurations have been drawn in Figures 13–17. Figure 13 shows the step response plot for the best parameter set, i.e. the one with the best index of goodness from the genetic algorithm. To get a fair evaluation of whether these parameters were what we needed, the simulation results for these plots were produced the same way as in the genetic algorithms fitness function. This meant that they were simulated with a step response and a 5 s transient time. Here we see no overshoot at all, short settling time, final error and steady state error isn't large either. Figure 14 shows the plot for the worst parameter set selected for testing. Its parameter values were $P = 43$, $I = 1382.5$, $D = 8.5$. This configuration broke all our secondary criterions, which determine normal or abnormality; too large overshoot, too slow settling time and too great of a steady state error.

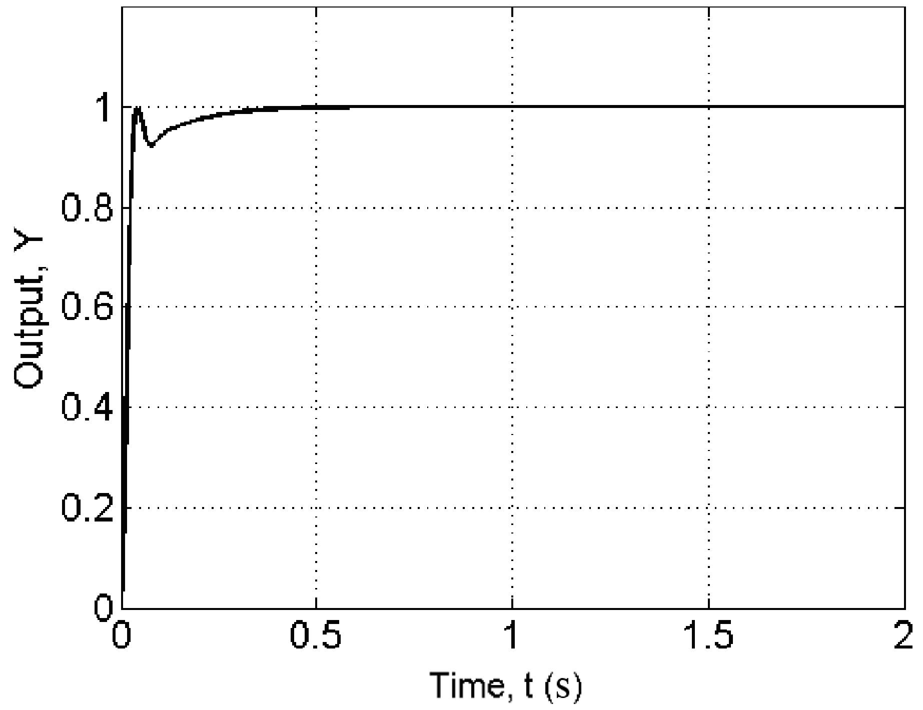

Figure 13. Step response for the best parameter set given by the genetic algorithm. Parameters: $P = 47.5$, $I = 3$, $D = 4.5$.

Figures 15–17 show a plot of the output of the controller for each criterion, where the controller barely makes it within the normal limit. Figure 15 shows an output where the controller barely meets the sum of absolute error criterion. The sum of absolute error of the controller is 417.87 and its parameters: $P = 47.5$, $I = 13$, $D = 4.5$. This is due to the fall or undershoot at the beginning and after settling the error is still higher than on most for the controllers, this one has next to no overshoot and a fast settling time.

Figure 16 shows a plot where the controller barely makes the settling time criterion. Settling time here was 0.61 s, and parameters were $P = 54$, $I = 144.5$, $D = 4.5$. Here it simply takes a long time for the output to decrease after the second rise. This one has a very low error when settled, so the *sum of the absolute error* doesn't rise too high.

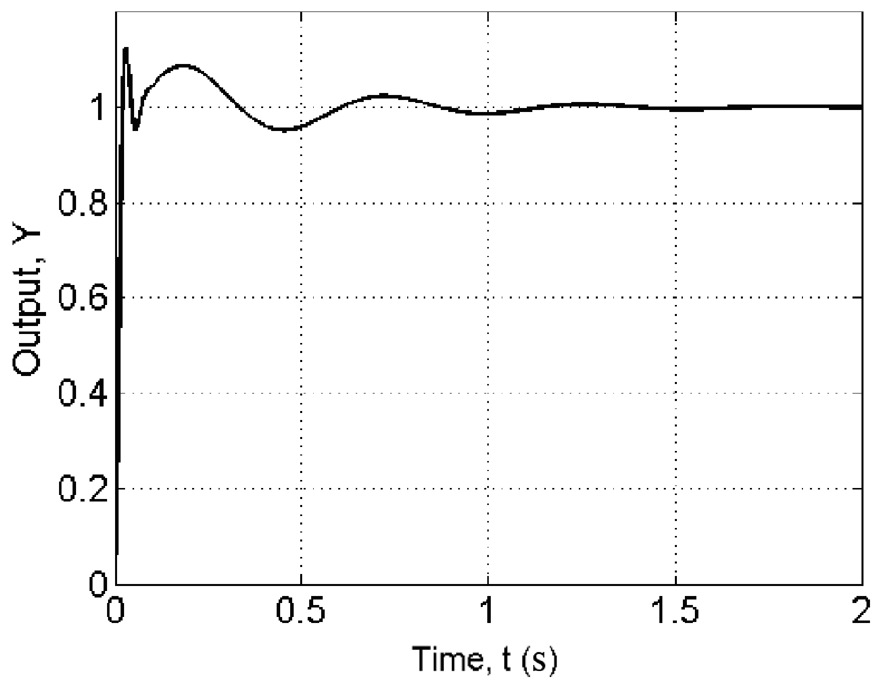

Figure 14. Step response for the worst parameter set we used for testing. Parameters: $P = 43$, $I = 1382.5$, $D = 8.5$.

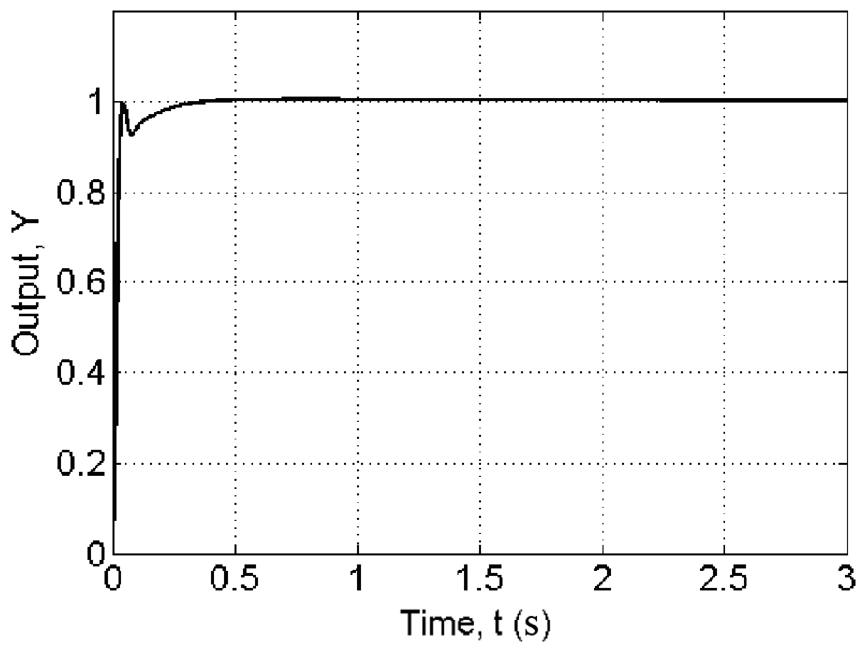

Figure 15. Step response of a controller barely making the sum of absolute error criterion. Sum of absolute error is 417.87. Parameters: $P = 47.5$, $I = 13$, $D = 4.5$.

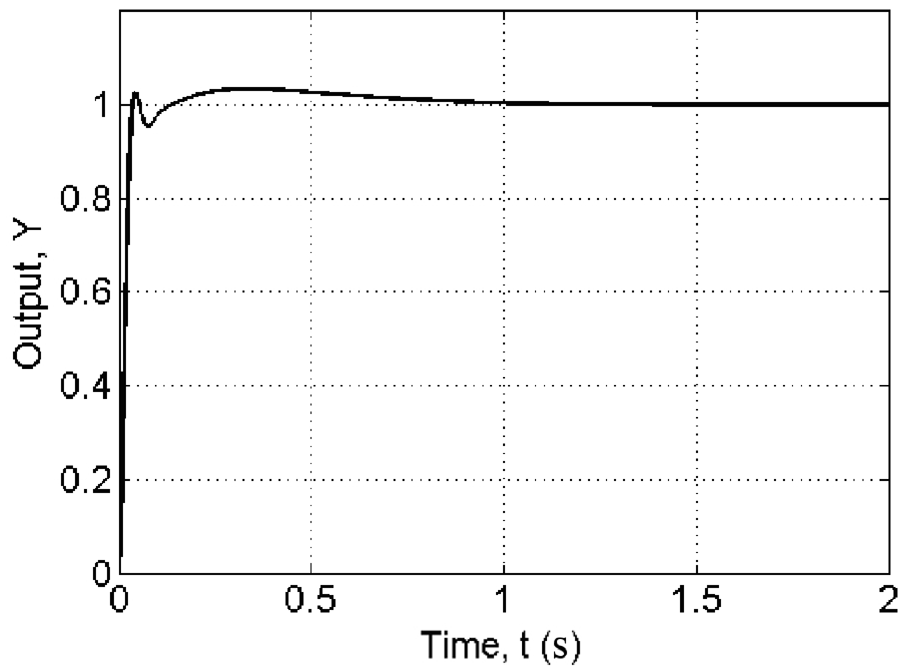

Figure 16. Step response of a controller barely making the settling time criterion. Settling time was 0.61 s. Parameters: $P = 54$, $I = 144.5$, $D = 4.5$.

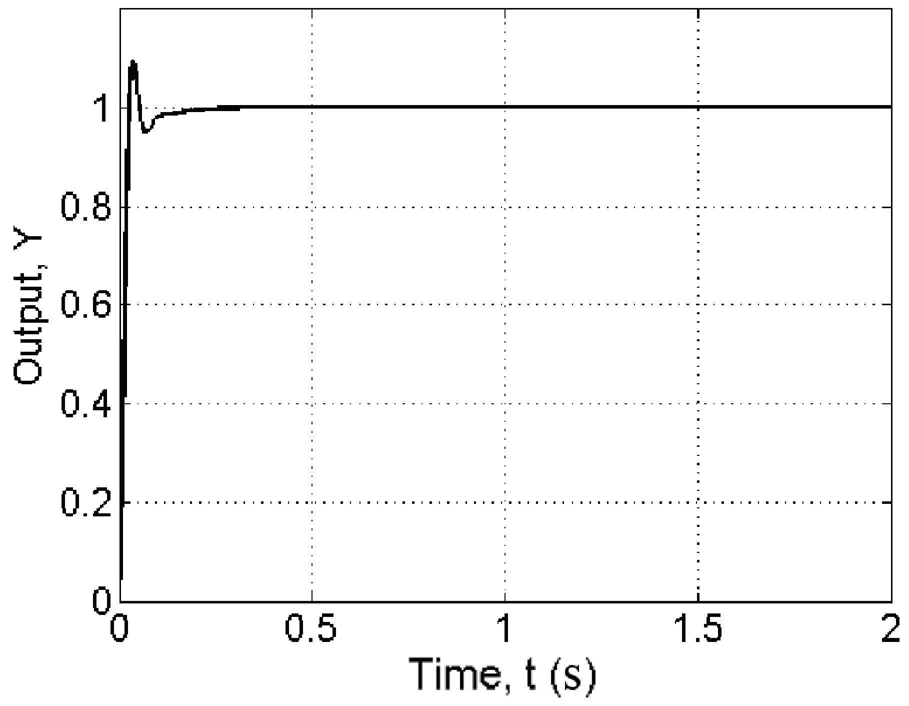

Figure 17. Step response of a controller barely making the overshoot criterion. Overshoot was 9.3%. Parameters: $P = 82.5$, $I = 13$, $D = 5.5$.

Figure 17 shows the output when the controller barely meets the overshoot criterion which here was 9.3% and parameters were P = 82.5, I = 13, D = 5.5. This one has a very low error when settled, so the sum of the absolute error doesn't rise too high and settling time works well also.

When investigating the worse behaviours, the error seen in the output reflected their index of goodness. Also interpreting the parameters in Table 1, we notice that the values are close to each other, which means they come from close by regions. This indicates the algorithm converged to a peak on the cost surface. From this we can conclude that the algorithm has found well working parameters.

Now we have to apply the second criteria to find out which parameter sets were normal and which not. To do this we analyzed the step responses of the parameters, to see if they exceeded any of the new criterions. The analyser program was built in such a way that it took 104 parameters for training. For this we made a training set n_t of 104 parameters. For testing we would also use 104 normal and 104 abnormal parameters, we will call these the normal test set n_{norm} and the abnormal test set n_{anorm}. In total 312 parameter sets of varying quality was to be generated for one test run.

We took the first 8 normal parameter sets and put them into the training set. Then we took the next 8 normal parameter sets and put them into the normal test set. We continued like this until we had two sets of 104 parameter sets. Then we chose abnormal parameters and made parameter sets based on why they were abnormal; we chose at least 8 parameters for every criterion that crossed only one of the criterions, 8 for each combination of criterions where the parameters broke exactly two of the criterions and finally at least 8 that broke all 3 criterions. There were some controllers with D or I at 0 proposed by the genetic algorithm. We left those out in this experiment and chose only PID controllers.

4.2. Engine PID parameter analysis

We trained the analyser with n_{t}. We used a threshold of 0.999 for training. This gave us a two-dimensional region shown in Figure 18. The circles mark the data points and all parameters inside the area are expected to be normal. In this case, all data points should be inside because the parameters used for training defines what is normal.

Due to using a VMM, which is stochastic, the program may produce different regions when fitting (Vesterback 2012: 410). We tried fitting a few times until we would get as simple shape as possible. We stopped at the region drawn by the contour in Figure 18. The scaled principal components having unit variations in Figure 18 and the following images comes from dimensionality reduction of the PID parameters from 3D-space to 2D-space, using principal component analysis. In previous fitting rounds we saw regions of different shapes, some with islands, some with more clusters and other shapes as well. The reason why we considered this the most reliable was that it was a very compact region without any islands, it had a small number of clusters. The region consists of three clusters, presented in Figure 19. *A* is a circle-like shape, *B* and *C* are two ellipsoid-like shapes, a large and a thin one. All the areas overlap a little, the thin area strikes trough the two others.

Next we tested the analyser using n_{norm}. Figure 20 shows results of this. Most data points are inside the region, which is good and tells us that the program is working. There are two so called false alarms. The parameters for these could be taken from the analyser, they were: $P = 56.5$, $I = 185$, $D = 4$ and $P = 62$, $I = 211$, $D = 4$. The analyser estimates the data with a probability of 0.999, which were the threshold used. That means the parameters classified as abnormal are such with a probability of $\mathbf{P} > 0.999$. (Vesterback 2012: 412)

Figure 21 shows the results from testing the analyser with n_{anorm}. Again, most of the parameters are where they should be, this time outside of the region. Here we have also a few points inside. These are called false positives. (Vesterback et al. 2012: 412)

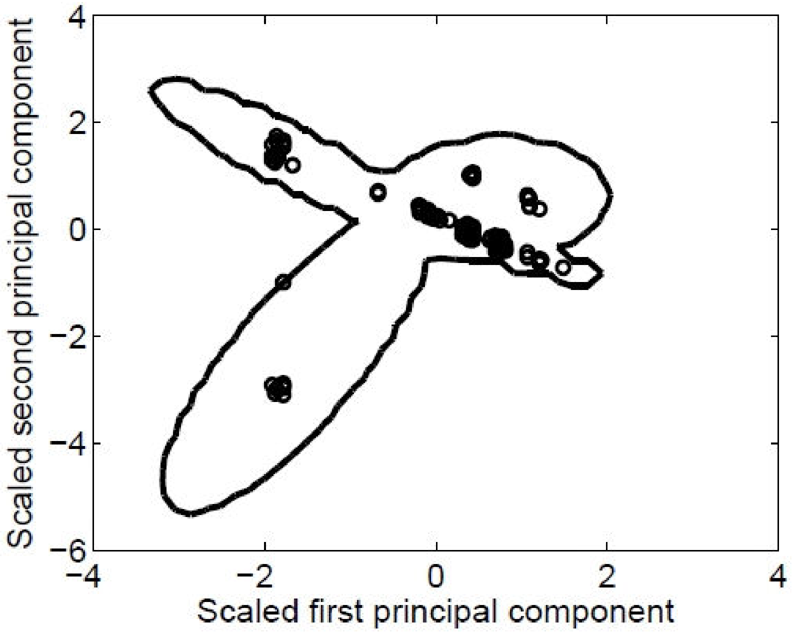

Figure 18. Region after training the analyser. Circles mark PID parameter sets. Normal parameter values are expected to be inside this region.

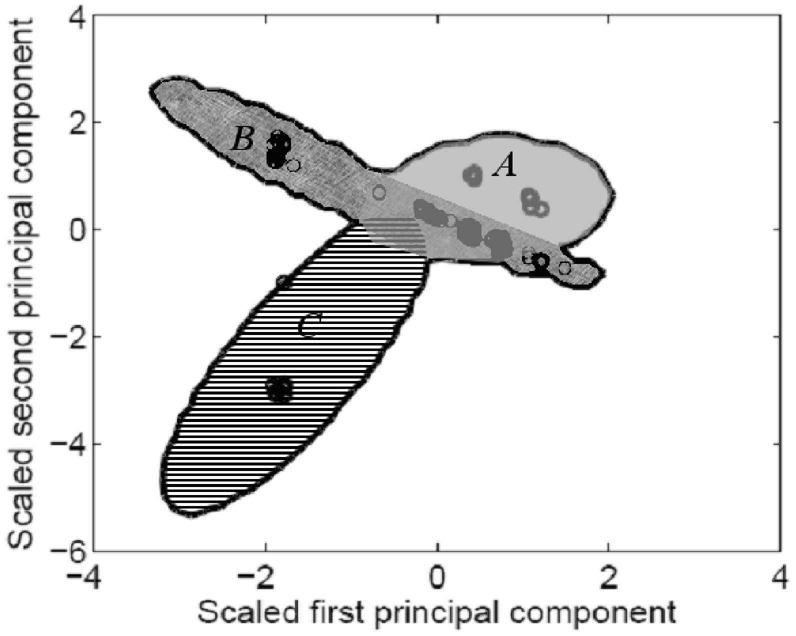

Figure 19. The three main clusters, *A* (light gray area), *B* (dark gray area) and *C* (striped area).

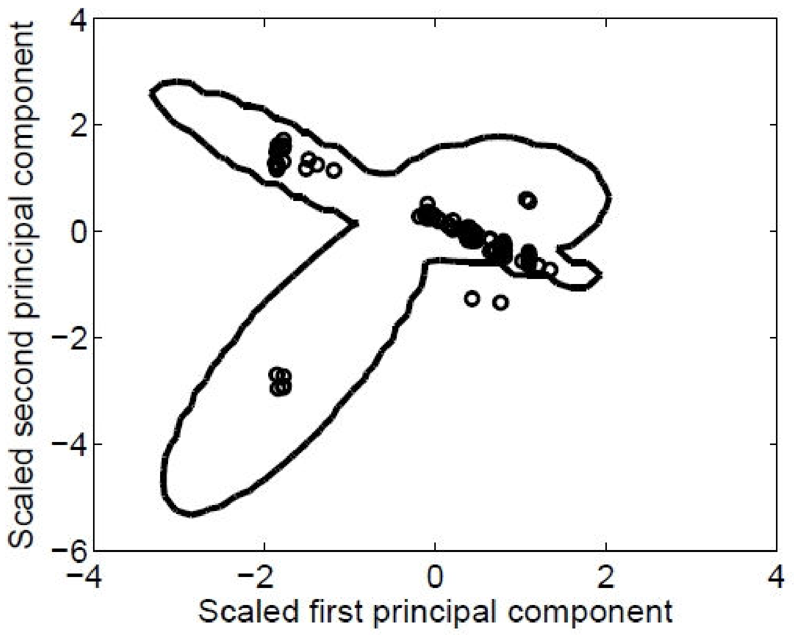

Figure 20. Results from testing normal data. Here we have two false alarms, but most of them fall inside the region.

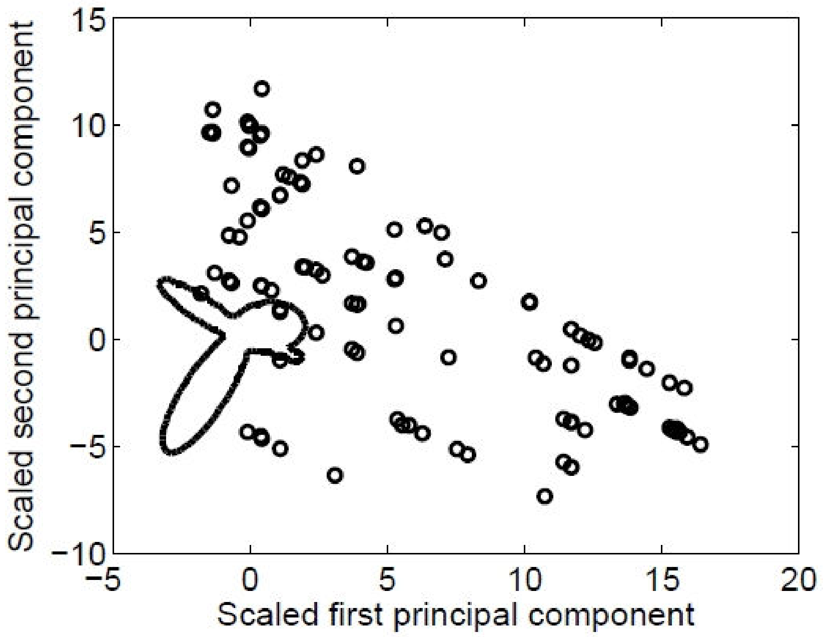

Figure 21. Results from testing abnormal data. Here we can see a few false positive results, but most of them fall outside the region.

In addition to the above experiments, we tried two other experiments. We trained the analyser with n_{norm} and used n_{t} as a test set for normal parameters. In the other experiment we used parameters generated in the same run with the genetic algorithm as we used the previous parameters. We chose parameters we had not used previously and made three more normal sets of 104 parameter triplets and two abnormal sets of 104 triplets. The results from both of these experiments gave about the same results as the first experiment.

With 102 right evaluations out of 104, we get a true positive rate of (TPR) ~98%, and a false positive rate (FPR) of ~2%. The rates were about the same in the following experiments. This gives us an accuracy of 98%, which means the analyzer works well. (Vesterback et al. 2012: 412-414)

5. SUMMARY, CONCLUSIONS AND FUTURE WORK

Many control systems have poorly tuned controllers today. This is because it requires lots of expertise to tune a PID controller. Many reports from the field state problems with closed loop control systems. If one could find a way to predetermine whether the parameters are poorly tuned without even running them in the process, it would speed up troubleshooting on the field.

We set out to investigate whether it was possible to use novelty detection for classifying PID controller quality. This would be exactly a way of predetermining PID parameter quality before running them in a process. An analyser program were built by University of Vaasa for this based on theory from novelty detection and multivariate extreme value statistics. The analyser would be trained with a set of parameters known to be normal, then one could test the quality of new parameters with regard to this training set.

The goal of the thesis was to create test parameters for the analyser and then assess how well the analyser worked. We used a genetic algorithm to produce these. Then we used a set of criterions based on step response to classify the parameters as normal or abnormal.

5.1. The genetic algorithm

There was not much new use of genetic algorithms in this thesis. They have been used to create test parameters for computer programs in many cases before. Also using genetic algorithms to tune PID controllers had been done. Genetic algorithms was a perfect tool for the job though. This was because it could find a huge set of parameters, where some would be well tuned and some would not and also anything in between.

The genetic algorithm was built in Matlab. It used a Simulink model for simulation in its fitness function. We used the step function as reference value and measurements were made with a 5 s transient time. To test whether the genetic algorithm had done what we wanted it to, we drew the step response plots for some of the parameters in the environment it was simulated in. We analysed these responses and found out the best parameters according to the genetic algorithm had a small error in regard to the criterions we had set on them. Also the worse parameters performed worse according to one or more of the criterions we set on them. Then we concluded the parameters produced were realistic and could be used as testing subjects for the PID outlier analyser.

5.2. PID outlier analyser

We chose three criteria for the PID parameters: sum of absolute error, maximum overshoot, and settling time. We set a threshold on each of these. If a configuration would cross any one of these, it would be classed as abnormal. We trained the program with parameters classified normal by these criteria. After that we tested the program with a set of normal and another set of abnormal parameters. The analyser correctly identified most normal parameters as normal and most abnormal parameters as abnormal.

We got a false positive rate of 98% and a true positive rate of 2%. Thus we conclude that the analyser works well. The analyser showed some false positives and false negatives. These cases are the exceptions that still should occur. We conducted a similar experiment and got similar false positive and true positive rates.

5.3. Novelty detection and variational mixture model

In novelty detection a variational mixture model was used instead of the conventional Gaussian mixture model. This made the fitting part easier for the user. The analyser could automatically determine the numbers of Gaussian clusters. Normally one would have to run the analyser several times, each time determining the numbers of clusters manually. After this one would have to compare the results and choose the most fitting result.

With the VMM the only thing needed to run the analyser was a click on a mouse and we would have a region. The fitting process would have to be run a few times to get a few regions. Out of these the most compact was selected, which is the region with the fewest clusters.

5.4. Discussion and future work

In this work the PID parameters were meant for engine control and were created by simulation. The only thing separating engines when simulating is their transfer function. The PID values could be created for engines with different transfer functions. Considering one could change the transfer function, the work might be applicable to any process using PID controllers. A suggestion would be to research how changing the transfer function affects the outcome. Different types of transfer functions could be investigated from different applications.

Since this is only one test in a single application, this number might not be very accurate. To get a closer reading one could run the test with lots of other transfer functions and/or criterions or index of goodness. Also running tests with real life PID values would be a future research to get empirical knowledge of the analyser.

This work only focused on PID controllers. PI- and PD controllers have to be analysed separately. Because of having one dimension less, they can't be analysed together with with the PID values. A possible future research might be to investigate if it's possible to analyse PI- or PD controllers the same way and find out whether there are some big differences to using the analyser with PID values.

Now we have found the analyser to be working by simulation. The next step is to verify the analyser in the field, after which hopefully we have made tuning and troubleshooting problems with PID controllers a little bit easier and faster.

REFERENCES

Alander, Jarmo T. (1991). On finding the optimal genetic algorithms for robot control problems. In: *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on. IEEE*, 1313-1318.

Alander, Jarmo T. (1992). On optimal population size of genetic algorithms. In: *Proceedings of CompEuro 92, IEEE Computer*, 65-70.

Alander, Jarmo T., Zinchenko, Lyudmila A. & Sorokin, Sergey N. (2004). Comparison of fitness landscapes for evolutionary design of dipole antennas. In: *IEEE Transactions on Antennas and Propagation* 52:11, 2932-2940.

Åström, Karl J. & Hägglund, Tore (1995). *PID Controllers: Theory, Design, and Tuning.* 2nd ed. Instrument Society of America. ISBN 1-55617-516-7.

Åström, Karl J. & Hägglund, Tore (2001). *The future of PID control.* In: Control engineering practice 9:11, 1163–1175.

Black, Harold S. (1934). *Stabilized feed-back amplifiers*. In: Transactions of the American Institute of Electrical Engineers 53:1, 114–120.

Blevins, Terrence L. (2012). PID advances in industrial control. In: *Preprints IFAC Conference on Advances in PID Control*. Brescia, Italy.

Chen, Chi-Tsong (1993). *Analog and digital control system design: transfer-function, state-space, and algebraic methods.* Orlando, Florida: Saunders College Publishing. ISBN: 0-03-094070-2.

Clifton, David Andrew, Hugeny, Samuel & Tarassenko, Lionel (2009). A comparison of approaches to multivariate extreme value theory for novelty detection. *IEEE Statistical Signal Processing*, 13-16. Cardiff, United Kingdom.

Clifton, David Andrew, Hugeny, Samuel & Tarassenko, Lionel (2010). Probabilistic patient monitoring using extreme value theory. In: *Biomedical Engineering Systems and Technologies*, 5-12. Valencia, Spain: Springer Science+Business Media.

Clifton, David Andrew, Hugeny, Samuel & Tarassenko, Lionel (2011). Novelty detection with multivariate extreme value statistics. *Journal of Signal Processing Systems* 65:3, 371-389.

D'Azzo, John J. & Houpis, Constantine H. (1995). *Linear control system analysis and design: conventional and modern (International edition).* 4th edition. Singapore: McGraw-Hill. ISBN: 0-07-113295.

De Jong, Kenneth A. (2006). *Evolutionary Computation: A Unified Approach*. Cambridge, Massachusetts: MIT Press. ISBN 0-262-04194-4.

Fröhr, Friedrich & Orttenburger, Fritz (1982). *Introduction to Electronic Control Engineering.* London, United Kingdom: Heyden & Son Ltd. ISBN: 0-85501-290-0.

Goldberg, David (1985). Dynamic system control using rule learning and genetic algorithms. In: *International Joint Conference on Artificial Intelligence - IJCAI*, 588-592. Los Angeles, California, USA.

Goldberg, David (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading: Addison-Wesley Professional. ISBN 978-0201157673.

Goldberg, David (2002). *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Boston: Kluwer Academic Publishers. ISBN 1-4020-7098-5.

Haugen, Finn, Fjelddalen, Eivind, Dunia, Ricardo & Edgar, Thomas F. (2007). Demonstrating PID control principles using an Air Heater and LabVIEW. *CACHE News (Computer Aids for Chemical Engineering)*.

Herrera, F., Lozano, M. & Verdegay, J.L. (1998). Tackling real-coded genetic algorithms: operators and tools for behavioural analysis. *Artificial Intelligence Review* 12:4, 265-319.

Holland, John (1975). *Adaptation in artificial and natural systems*. Ann Arbor: The University of Michigan Press.

Holland, John (1992). Genetic algorithms. *Scientific American* 267:1, 66-72.

Kreyszig, Erwin (2011). *Advanced engineering mathematics (international student version).* 10th edition. Singapore: John Wiley & Sons Inc. ISBN 978-0-470-64613-7.

Kwok, D. P. & Sheng, Fang (1994). Genetic algorithm and simulated annealing for optimal robot arm PID control. *IEEE World Congress on Computational Intelligence*, 707-713. Orlando, Florida, USA.

MathWorks (2012a). *PID Control with MATLAB and Simulink. Design and implement PID controllers* [online]. [Cited: 12.10.2012]. Available from Internet: MathWorks < URL: http://www.mathworks.se/discovery/pid-control.html >

MathWorks (2012b). Matlab documentation center [online] [Cited: 12.10.2012].. Available from Internet: MathWorks < URL: http://www.mathworks.se/help/ >

Mantere, Timo (2006). A min-max genetic algorithm with alternating multiple sorting for solving constrained problems. In: *Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence*, 61-67.

Mantere, Timo & Alander, Jarmo T. (2001). Automatic test image generation by genetic algorithms for testing halftoning methods – comparing results using wavelet filtering. In: *2001 Finnish Signal Processing Symposium*, 55-58.

Mantere, Timo & Alander, Jarmo T. (2005). Evolutionary software engineering, a review. In: *Applied Soft Computing* 5:3, 315-331.

Mantz, Ricardo J. & Tacconi, Eugenio J. (1989). Complementary rules to Ziegler and Nichols' rules for a regulating and tracking controller. In: *International Journal of control* 49:5, 1465-1471.

Maxwell, Clerk. J. (1868). On governors. *Proceedings of the Royal Society of London,* 270-283. London, United Kingdom.

Mazeda, Rogelio & de Prada, César (2012). IFTtune: a PID automatic tuning software tool. *IFAC Conference on Advances in PID Control*. Brescia, Italy.

McClean, Phillip (1997). *Recombination and estimating the distance between genes*. [Online]. North Dakota: North Dakota State University. [Cited: 9.4.2013] Available from internet: < URL: http://www.ndsu.edu/pubweb/~mcclean/plsc431/lin kage /linkage2.htm >.

Michael, C. Christoph, McGraw, E. Gary, Schatz, A. Michael & Walton, Curtis C. (1997). Genetic algorithms for dynamic test data generation. *Automated Software Engineering – ASE*, 307-308.

Miljkovic, Dubravko (2010). Review of novelty detection methods. In: *MIPRO international convention*, 593 – 598. Opatija, Croatia.

Mitsukura, Yasue, Yamamoto, Toru & Kaneda, Masahiro (1999). A design of self-tuning PID controllers using a genetic algorithm. In: *American Control Conference 1999*, 1361–1365.

Mohammed, Faisal A. & Koivo, Heikki N. (2005). Diesel engine systems with genetic algorithm self tuning PID controller. In: *International Conference on Future Power Systems*. The Netherlands: Delft Univ. of Tech.

Roberts, S. J. (1999). Novelty detection using extreme value statistics. In: *IEEE Proceedings-Vision, Image and Signal Processing*, 124-129. United Kingdom: Michael Faraday House.

Roberts S. J. (2000). Extreme value statistics for novelty detection in biomedical data processing. *IEEE Proceedings: Science, Measurements and Technology*, 363-367. United Kingdom: Michael Faraday House.

Srivastava, Praveen Ranjan & Kim, Tai-hoon (2009). Application of genetic algorithm in software testing. In: *International Journal of Software Engineering and Its Applications* 3:4, 87-96.

Törmänen, Pasi (1997). *Geneettisten algoritmien käyttö tutkittaessa sumean logiikan hyötyä PID-säädössä : esimerkkinä taajuusmuuttajan sisäiset säätäjät.* [*Evaluating the benefit of fuzzy logic for PID control by means of genetic algorithms – case: frequency controller*]. Master of Economics thesis, University of Vaasa.

Vesterback, Joni, Bochko, Vladimir, Ruohonen, Mika, Alander, Jarmo, Bäck, Andreas, Nylund, Martin, Dal, Allan & Östman, Fredrik (2012). Engine parameter outlier detection: verification by simulating PID controllers generated by genetic algorithm. In: *IDA2012 Intelligent Data Analysis conference*, 404-415. Ed. Jaakko Hollmén, Frank Klawonn & Alland Tucker. Spriner: Heidelberg.

Wärtsilä Ltd. *Mission, Vision and Strategy* [online]. [Cited: 4.8.2012]. Available from Internet: <URL: http:// www.wartsila.com/en/about/company-management/strategy/mission-vision >

Ziegler, J. G. & Nichols, N. B. (1947). Optimum settings for automatic controllers. trans. *American Society of Mechanical Engineers Transactions* 64:11, 759-768.

APPENDIX A

List of parameters used in testing
Training parameters:

P	I	D	Index	Sum of error	Max Overshoot	Settling time (2%)
47.5	3.0	4.5	0.132970	318.1246668	0.001567802	0.2159
47.5	3.0	2.5	0.130400	325.3523254	0.001604443	0.1225
43.0	3.0	4.5	0.120593	355.0714787	0.001894413	0.2460
47.5	5.5	4.5	0.119616	354.0160972	0.002780049	0.2111
47.5	5.5	2.5	0.116419	364.9839493	0.002882279	0.1188
47.5	7.0	4.5	0.113910	371.4635485	0.003476934	0.2083
47.5	8.0	4.5	0.110778	381.6016188	0.003930547	0.2066
47.5	7.0	2.5	0.110499	384.2503566	0.003629401	0.1165
43.0	8.0	4.5	0.099923	426.7422380	0.004706377	0.2338
47.5	13.0	3.5	0.098649	425.4459774	0.006261103	0.1623
47.5	13.0	2.5	0.096358	436.2961470	0.006501594	0.0589
47.5	21.5	4.5	0.092408	443.9168084	0.009420037	0.1868
56.5	13.0	2.5	0.092116	373.8474906	0.023773231	0.0733
57.5	13.0	2.5	0.089670	370.0189997	0.027499490	0.0766
65.5	7.0	4.5	0.089417	270.1123655	0.043968382	0.1223
65.0	7.0	4.5	0.088663	271.8739942	0.042757075	0.1243
57.5	185.0	4.5	0.086349	341.0073864	0.037607210	0.5560
57.0	180.0	4.5	0.086285	343.0117446	0.037292229	0.5626
57.0	181.0	4.5	0.086195	342.8894530	0.037437729	0.5619
59.0	13.0	2.5	0.086185	365.0013625	0.033013127	0.0804
57.0	182.0	4.5	0.086105	342.7708411	0.037582899	0.5611
59.0	149.0	4.5	0.085956	343.2757488	0.037683897	0.5651
56.5	176.0	4.5	0.085915	344.9768255	0.037113346	0.5687
57.0	184.5	4.5	0.085879	342.4893827	0.037944401	0.5593
66.0	13.0	4.5	0.085369	303.3270532	0.045572297	0.1172
56.5	184.5	4.5	0.085357	343.9745229	0.038359812	0.5622
56.5	185.0	4.5	0.085312	343.9235346	0.038432386	0.5618
59.0	176.0	4.5	0.085311	338.3183722	0.039554155	0.5524
59.0	180.0	4.5	0.085195	337.7258594	0.039831970	0.5500
59.0	181.0	4.5	0.085165	337.5854496	0.039901452	0.5494
59.0	182.0	4.5	0.085135	337.4480595	0.039970932	0.5488
59.0	184.5	4.5	0.085057	337.1182148	0.040144624	0.5472
57.5	205.5	4.5	0.084499	339.3455627	0.040475823	0.5408
59.0	201.0	4.5	0.084478	335.4138290	0.041290703	0.5363
59.0	137.0	4.5	0.084465	345.8666812	0.036854464	0.5672
57.0	201.5	4.5	0.084338	341.0943780	0.040351093	0.5464
59.0	205.5	4.5	0.084303	335.0806798	0.041603182	0.5332
55.5	185.0	4.5	0.084244	347.0864489	0.039284834	0.5676
60.0	163.5	4.5	0.084199	338.0826689	0.041149229	0.5515
58.0	144.5	4.5	0.084102	346.8689501	0.034889102	0.5755
56.5	200.5	4.5	0.083901	342.7244889	0.040643196	0.5497
57.0	209.0	4.5	0.083660	340.7249823	0.041385977	0.5406
60.0	185.0	4.5	0.083635	334.6685356	0.042633455	0.5406
57.5	144.5	4.5	0.083454	348.2377505	0.033640806	0.5801
56.5	205.5	4.5	0.083447	342.4762264	0.041340995	0.5458

57.0	211.5	4.5	0.083435	340.6300144	0.041727489	0.5387
56.5	155.0	4.5	0.083428	348.6571891	0.033922916	0.5833
57.0	149.0	4.5	0.083353	348.5969501	0.032702430	0.5825
56.5	211.5	4.5	0.082905	342.2535053	0.042169020	0.5410
54.0	182.0	4.5	0.082865	352.5660535	0.040164578	0.5787
57.0	144.5	4.5	0.082805	349.6355474	0.032387954	0.5846
56.5	149.0	4.5	0.082701	350.0001967	0.032979618	0.5868
56.5	214.5	4.5	0.082635	342.1702308	0.042579326	0.5387
62.0	99.0	4.5	0.082589	347.2446530	0.041632686	0.4654
54.0	185.0	4.5	0.082588	352.3310973	0.040617194	0.5760
59.0	245.5	4.5	0.082513	333.9447085	0.044404042	0.5056
62.0	136.0	4.5	0.082066	338.5189384	0.044149386	0.5347
62.0	137.0	4.5	0.082046	338.3228243	0.044217578	0.5351
53.5	180.0	4.5	0.082046	354.6019169	0.040314613	0.5833
53.5	185.0	4.5	0.082021	354.2210752	0.041076246	0.5787
69.5	13.0	4.5	0.081954	290.4897323	0.053907931	0.1092
55.5	236.5	5.0	0.081938	340.0660221	0.044029702	0.5447
57.0	137.0	4.5	0.081895	351.4554413	0.031864924	0.5873
62.0	144.5	4.5	0.081894	336.9016630	0.044728964	0.5370
53.5	176.0	4.5	0.081640	354.9692051	0.039698651	0.5870
53.0	182.0	4.5	0.081508	356.3876920	0.041083850	0.5842
53.0	185.0	4.5	0.081446	356.1851725	0.041543032	0.5814
59.0	182.0	5.0	0.081430	330.3412557	0.046737110	0.5673
53.0	181.0	4.5	0.081410	356.4616000	0.040930051	0.5852
55.5	149.0	4.5	0.081396	352.9116786	0.033749466	0.5951
59.0	185.0	5.0	0.081361	329.9942634	0.046910169	0.5654
53.0	180.0	4.5	0.081312	356.5389837	0.040775877	0.5861
57.5	245.5	4.5	0.080975	338.7120954	0.045752299	0.5112
62.0	184.5	4.5	0.080939	330.4458046	0.047459918	0.5280
62.0	185.0	4.5	0.080926	330.3758078	0.047494092	0.5277
53.0	176.0	4.5	0.080915	356.8834198	0.040155389	0.5899
57.5	185.0	5.0	0.080913	334.4065780	0.043533111	0.5750
55.5	144.5	4.5	0.080860	354.0146032	0.033020390	0.5978
56.5	3.0	4.5	0.080755	267.0440482	0.021292710	0.1683
59.0	267	4.5	0.080710	334.2794967	0.047043999	0.4912
62.0	205.5	4.5	0.080331	327.9339792	0.048897411	0.5172
52.0	185.0	4.5	0.080277	360.3447891	0.042500476	0.5866
62.5	185.0	4.5	0.080273	329.3798293	0.048699514	0.5244
52.0	184.5	4.5	0.080257	360.3714152	0.042423072	0.5871
54.0	234.5	5.0	0.080218	346.1536625	0.045163370	0.5522
62.0	211.0	4.5	0.080152	327.4415583	0.049274414	0.5141
53.5	205.5	4.5	0.080136	353.3959148	0.044108758	0.5598
62.0	211.5	4.5	0.080135	327.4002867	0.049308684	0.5138
55.5	234.5	4.5	0.079792	345.6281673	0.046199630	0.5270
59.0	99.0	4.5	0.079764	356.2066274	0.034232601	0.5292
63.0	180.0	4.5	0.079757	329.1016156	0.049559776	0.5230
56.5	247.5	4.5	0.079755	342.2092212	0.046942839	0.5134
56.5	185.0	5.0	0.079730	337.6300875	0.041266984	0.5812
59.0	236.5	5.0	0.079725	327.7277606	0.049885128	0.5301
62.0	13.0	4.5	0.079669	320.5863972	0.035815425	0.1327
53.5	236.5	5.0	0.079643	348.3825987	0.045884315	0.5525

53.5	234.5	5.0	0.079545	348.3221563	0.045636207	0.5542
62.5	211.0	4.5	0.079522	326.3856734	0.050475004	0.5115
62.5	211.5	4.5	0.079505	326.3423176	0.050509221	0.5112
54.0	245.5	5.0	0.079485	346.5105357	0.046507502	0.5431
62.5	214.5	4.5	0.079407	326.0937486	0.050714515	0.5095
53.5	214.5	4.5	0.079325	353.3256700	0.045397770	0.5518
62.0	234.5	4.5	0.079302	326.0702844	0.050886503	0.5003
53.5	153.5	4.5	0.079264	358.3258947	0.036114378	0.6073

Normal test parameters:

P	I	D	Index	Sum of error	Max Overshoot	Settling time (2%)
43.0	5.5	4.5	0.108003	396.1377864	0.003342362	0.2396
47.5	8.0	2.5	0.107255	395.4799053	0.004120346	0.1148
44.0	7.0	4.5	0.105240	405.0041619	0.003999088	0.2295
43.0	7.0	4.5	0.102760	415.6152326	0.004169657	0.2361
44.0	8.0	4.5	0.102335	415.9052107	0.004515524	0.2274
53.0	13.0	2.5	0.101412	390.9451141	0.010401195	0.0508
47.5	13.0	4.5	0.100321	417.8703552	0.006087049	0.1984
53.5	13.0	2.5	0.100021	388.0891881	0.012343574	0.0503
47.5	33.5	4.5	0.088420	447.3915829	0.013613910	0.1735
65.0	8.0	4.5	0.087650	279.0212311	0.042823424	0.1236
65.0	8.5	4.5	0.087183	282.3914626	0.042856598	0.1232
57.5	180.0	4.5	0.086798	341.6098324	0.036888143	0.5595
57.5	181.0	4.5	0.086708	341.4824505	0.037032600	0.5588
57.5	182.0	4.5	0.086619	341.3584944	0.037176733	0.5581
58.0	185.0	4.5	0.086478	339.6359184	0.037708932	0.5530
57.5	184.5	4.5	0.086394	341.0638131	0.037535664	0.5563
57.0	184.5	4.5	0.085879	342.4893827	0.037944401	0.5593
57.0	185.0	4.5	0.085834	342.4356740	0.038016459	0.5589
59.0	155.0	4.5	0.085827	342.0697729	0.038099626	0.5630
56.5	180.0	4.5	0.085765	344.4725228	0.037702947	0.5657
56.5	181.0	4.5	0.085675	344.3557138	0.037849499	0.5649
56.5	182.0	4.5	0.085584	344.2424534	0.037995716	0.5642
56.5	173.0	4.5	0.085567	345.3950101	0.036667514	0.5709
59.0	144.5	4.5	0.085398	344.2176937	0.037372423	0.5663
59.0	185.0	4.5	0.085041	337.0545884	0.040179362	0.5469
59.0	187.5	4.5	0.084960	336.7479850	0.040353040	0.5453
56.5	167.5	4.5	0.084922	346.2549607	0.035841817	0.5749
59.0	190.0	4.5	0.084876	336.4605396	0.040526707	0.5436
57.5	155.0	4.5	0.084743	345.9285420	0.034372111	0.5753
55.5	180.0	4.5	0.084702	347.5795952	0.038544843	0.5717
65.5	13.0	4.5	0.084656	305.3238362	0.044365860	0.1187
55.5	182.0	4.5	0.084519	347.3718063	0.038841865	0.5701
59.0	211.0	4.5	0.084080	334.7408172	0.041986054	0.5294
59.0	211.5	4.5	0.084059	334.7134249	0.042020889	0.5291
57.5	149.0	4.5	0.084005	347.2239621	0.033954092	0.5783
57.5	211.0	4.5	0.084004	339.0859253	0.041225126	0.5367
59.0	213.5	4.5	0.083976	334.6092713	0.042160225	0.5277
57.5	211.5	4.5	0.083959	339.0656420	0.041292842	0.5363
65.0	13.0	4.5	0.083943	307.3644536	0.043155503	0.1203
59.0	214.5	4.5	0.083933	334.5604988	0.042229890	0.5270
62.0	8.0	4.5	0.083169	290.9421048	0.035478261	0.1375
57.0	214.5	4.5	0.083166	340.5330885	0.042135088	0.5364
54.5	185.0	4.5	0.083147	350.5134671	0.040165714	0.5732
60.0	201.5	4.5	0.083097	332.8383055	0.043774105	0.5306
55.5	163.5	4.5	0.083094	349.8958879	0.036039388	0.5848
56.5	211.0	4.5	0.082950	342.2691271	0.042100399	0.5414
59.0	236.5	4.5	0.082949	333.9669211	0.043762044	0.5118
60.5	185.0	4.5	0.082944	333.5411089	0.043854772	0.5375
56.5	185.0	4.0	0.082483	353.2116702	0.040594493	0.5414

55.5	205.5	4.5	0.082369	345.8523626	0.042234095	0.5506
53.5	182.0	4.5	0.082247	354.4396437	0.040620360	0.5815
54.0	189.0	4.5	0.082218	352.0622338	0.041215768	0.5723
56.5	144.5	4.5	0.082157	351.0637491	0.032262080	0.5891
53.5	181.0	4.5	0.082146	354.5190247	0.040467671	0.5824
55.5	234.5	5.0	0.082107	340.0244411	0.043787501	0.5462
53.5	184.5	4.5	0.082067	354.2554544	0.041000491	0.5791
55.5	211.0	4.5	0.081871	345.6963623	0.043003752	0.5460
55.5	211.5	4.5	0.081826	345.6852254	0.043073293	0.5456
62.0	149.0	4.5	0.081797	336.0875484	0.045035748	0.5374
56.5	141.5	4.5	0.081794	351.7955483	0.031778748	0.5905
53.5	187.5	4.5	0.081789	354.0605106	0.041453690	0.5764
57.5	236.5	4.5	0.081749	338.6373072	0.044598321	0.5178
62.0	153.5	4.5	0.081698	335.2993817	0.045342495	0.5373
53.5	189.0	4.5	0.081650	353.9732027	0.041679101	0.5750
56.5	137.0	4.5	0.081252	352.9283391	0.031046026	0.5922
62.5	144.5	4.5	0.081238	335.7688388	0.045940670	0.5319
57.0	236.5	4.5	0.081225	340.3049090	0.045054323	0.5197
56.5	231.0	4.5	0.081173	341.9962689	0.044794093	0.5259
62.0	176.0	4.5	0.081158	331.6845255	0.046878893	0.5316
62.0	180.0	4.5	0.081057	331.0915603	0.047152333	0.5299
62.0	181.0	4.5	0.081031	330.9458141	0.047220688	0.5295
62.0	182.0	4.5	0.081005	330.8013593	0.047289042	0.5291
56.5	237.0	4.5	0.080652	342.0350142	0.045582758	0.5213
62.0	211.0	4.0	0.080531	338.1032987	0.046554394	0.4953
53.5	201.5	4.5	0.080500	353.4776356	0.043527880	0.5635
62.0	201.0	4.5	0.080472	328.3894476	0.048588912	0.5197
62.5	13.0	4.5	0.080380	318.2604180	0.037048328	0.1304
62.5	181.0	4.5	0.080376	329.9432671	0.048426531	0.5261
62.5	182.0	4.5	0.080350	329.8008992	0.048494779	0.5257
53.5	163.5	4.5	0.080343	356.5345948	0.037733516	0.5985
62.0	213.5	4.5	0.080068	327.2407595	0.049445760	0.5127
53.0	167.5	4.5	0.080058	357.8220360	0.038815995	0.5980
62.0	214.5	4.5	0.080034	327.1642966	0.049514296	0.5121
65.0	36.0	4.5	0.080007	351.1821765	0.044686453	0.1108
63.5	144.5	4.5	0.079957	333.5718574	0.048353470	0.5216
56.5	245.5	4.5	0.079924	342.1664888	0.046685691	0.5149
54.0	214.5	4.5	0.079892	351.2991693	0.044909063	0.5497
62.0	13.0	2.5	0.079849	357.2285713	0.043776139	0.0861
53.5	211.0	4.5	0.079639	353.3354874	0.044899384	0.5549
59.0	7.0	4.5	0.079631	296.6520155	0.027921920	0.1527
55.5	236.5	4.5	0.079620	345.6589978	0.046465112	0.5255
53.5	237.0	5.0	0.079601	348.3983416	0.045946181	0.5520
53.5	211.5	4.5	0.079594	353.3327657	0.044970804	0.5544
53.0	205.5	4.5	0.079562	355.4570954	0.044596761	0.5621
54.5	144.5	4.5	0.079561	357.1131261	0.033805376	0.6064
65.0	43.5	4.5	0.079551	352.5823766	0.045185483	0.1086
62.0	236.5	4.5	0.079224	326.0020093	0.051023975	0.4991
72.0	13.0	4.5	0.079208	282.4253282	0.059751895	0.1051
62.0	237.0	4.5	0.079204	325.9860225	0.051058342	0.4988
65.0	59.0	4.5	0.079191	350.2829215	0.046219638	0.1050

53.0	211.0	4.5	0.079066	355.4193949	0.045392719	0.5570
56.5	256.0	4.5	0.079043	342.4368707	0.048026017	0.5072
65.5	7.0	2.5	0.079037	308.1706715	0.054877147	0.0897
53.0	211.5	4.5	0.079021	355.4186283	0.045464618	0.5566

Abnormal parameters:

P	I	D	Index	Sum of error	Max Overshoot	Settling time (2%)	Broken criterion
43.0	144.5	2.5	0.060260	479.9748280	0.059952387	0.5786	sum of error
47.5	182.0	2.5	0.060247	459.9704656	0.063990186	0.5106	sum of error
47.5	185.0	2.5	0.059948	459.8663347	0.064837033	0.5071	sum of error
38.5	236.5	4.5	0.058547	462.2660870	0.068349120	0.5783	sum of error
50.0	237.0	2.5	0.057070	451.7441754	0.074874934	0.4481	sum of error
38.5	236.5	5.5	0.056810	451.1039979	0.060899803	0.6234	sum of error
35.0	236.5	5.5	0.052697	492.2411654	0.066307577	0.6337	sum of error
43.0	51.0	5.5	0.052542	456.5175589	0.020781559	0.7417	sum of error
94.5	13.0	4.5	0.060203	237.8653675	0.108523719	0.0867	max overshoot
72.0	300.0	6.5	0.059726	283.0628977	0.100818778	0.4866	max overshoot
82.5	182.0	5.5	0.059653	285.3994643	0.100556247	0.3658	max overshoot
82.5	185.0	5.5	0.059622	285.1590734	0.100691165	0.3718	max overshoot
72.0	308.5	6.5	0.059610	283.1857801	0.101120452	0.4833	max overshoot
82.5	212.5	5.5	0.059332	283.0779295	0.101927595	0.4030	max overshoot
72.0	339.0	6.5	0.059173	283.9475034	0.102206134	0.4714	max overshoot
72.0	357.0	6.5	0.058908	284.5418774	0.102847525	0.4643	max overshoot
47.5	122.0	6.5	0.057992	380.6110604	0.052196616	0.7736	settling time
43.0	471.0	7.5	0.057687	417.1924341	0.079911386	0.7795	settling time
43.0	482.0	7.5	0.057541	417.7146851	0.080244941	0.7848	settling time
47.5	144.5	7.0	0.057207	377.3887820	0.065134083	0.7627	settling time
43.0	564.5	7.5	0.056457	421.8765558	0.082749366	0.7665	settling time
43.0	99.0	5.5	0.056381	417.7411882	0.032647478	0.8174	settling time
43.0	92.0	5.5	0.055944	420.8930366	0.031134783	0.8281	settling time
50.0	99.0	7.5	0.055908	373.7251363	0.079559957	0.7824	settling time
38.5	440.0	6.5	0.056821	461.5568558	0.073680140	0.8192	sum of error,settling time
38.5	455.0	6.5	0.056396	462.7284659	0.074772545	0.8119	sum of error,settling time
38.5	458.0	6.5	0.056312	462.9660966	0.074988075	0.8103	sum of error,settling time
38.5	560.5	7.5	0.055934	463.5411532	0.075639209	0.8005	sum of error,settling time
38.5	440.0	7.5	0.054083	457.2958750	0.071941275	0.8659	sum of error,settling time
38.5	564.5	6.5	0.053624	471.9361967	0.082094894	0.7541	sum of error,settling time
38.5	455.0	5.5	0.052970	472.5601128	0.084272881	0.7603	sum of error,settling time
35.0	339.0	5.5	0.052848	502.0914074	0.078802910	0.8589	sum of error,settling time
53.0	144.5	8.5	0.053363	349.8735263	0.107421742	0.7545	max overshoot,settling time
53.0	149.0	8.5	0.053360	349.3826002	0.107529062	0.7513	max overshoot,settling time
50.0	185.0	9.0	0.050529	363.5069830	0.115204400	0.7554	max overshoot,settling time
43.0	560.5	8.5	0.050523	418.3939237	0.104252562	0.8055	max overshoot,settling time
53.0	144.5	9.0	0.050472	350.5343633	0.118022323	0.7674	max overshoot,settling time
53.0	149.0	9.0	0.050471	350.0703724	0.118120022	0.7643	max overshoot,settling time
53.5	153.5	9.0	0.050432	347.2757791	0.118830846	0.7555	max overshoot,settling time
43.0	180.0	9.0	0.050396	410.4272567	0.106342837	0.8119	max overshoot,settling time
43.0	569.0	4.5	0.049251	458.8405855	0.101272767	0.6265	sum of error,max overshoot
43.0	582.5	4.5	0.048827	461.0076692	0.102601888	0.6219	sum of error,max overshoot
43.0	602.5	4.5	0.048210	464.4050764	0.104544042	0.6150	sum of error,max overshoot
53.0	392.5	2.5	0.048103	457.8439429	0.106318425	0.3441	sum of error,max overshoot
43.0	1340.0	7.5	0.047815	462.8744815	0.106565830	0.7465	sum of error,max overshoot
47.5	738.0	4.5	0.047801	450.8449869	0.109031577	0.5536	sum of error,max overshoot
43.0	1348.5	7.5	0.047737	463.2487005	0.106829828	0.7447	sum of error,max overshoot
43.0	1360.5	7.5	0.047629	463.7762215	0.107202877	0.7422	sum of error,max overshoot
43.5	1062.0	6.5	0.049318	460.3700962	0.100691421	0.7518	all 3

43.0	1040.0	6.5	0.049249	463.3915813	0.100370531	0.7601	all 3
43.0	1212.0	7.5	0.049013	457.1353046	0.102602047	0.7729	all 3
43.0	1213.5	7.5	0.048998	457.2039293	0.102648531	0.7726	all 3
43.0	1062.0	6.5	0.048946	464.8783933	0.101331158	0.7596	all 3
43.0	1220.0	7.5	0.048936	457.5009595	0.102849950	0.7713	all 3
38.5	704.5	8.5	0.048862	464.4039440	0.101777400	0.7670	all 3
43.0	1229.5	7.5	0.048845	457.9335467	0.103144307	0.7694	all 3
43.0	45.5	8.5	0.047677	453.4407182	0.091678506	0.2579	sum of error
35.0	236.5	8.0	0.047525	487.9895140	0.072966106	0.7327	sum of error
35.0	300.0	8.5	0.047486	489.9933455	0.086961700	0.6736	sum of error
35.0	293.5	8.5	0.047419	489.8259291	0.086800100	0.6804	sum of error
35.0	262.5	8.5	0.047103	489.0668557	0.086029762	0.7160	sum of error
43.0	36.0	8.5	0.046957	467.1754056	0.091447224	0.2720	sum of error
35.0	245.5	8.5	0.046927	488.7155326	0.085609318	0.7378	sum of error
35.0	300.0	9.0	0.046779	490.1938297	0.098808011	0.6891	sum of error
43.0	45.5	8.5	0.047677	453.4407182	0.091678506	0.2579	max overshoot
35.0	236.5	8.0	0.047525	487.9895140	0.072966106	0.7327	max overshoot
56.5	987.0	4.5	0.047539	423.6030597	0.115632216	0.4707	max overshoot
53.0	944.0	8.5	0.047532	368.8632572	0.126609873	0.6276	max overshoot
72.0	1062.0	7.5	0.047521	306.0502637	0.139222463	0.3130	max overshoot
53.5	1360.5	7.5	0.047519	390.2283599	0.122395524	0.5141	max overshoot
72.0	339.0	9.0	0.047511	278.5186544	0.144774079	0.5272	max overshoot
53.5	944.0	8.5	0.047506	366.2670395	0.127244126	0.6254	max overshoot
35.0	1062.0	8.5	0.045489	519.0701479	0.106018864	0.9031	all 3
35.0	1087.0	8.5	0.045317	520.0980610	0.106649833	0.8944	all 3
35.0	1191.5	7.5	0.045226	541.7103985	0.102771975	0.8176	all 3
35.0	1111.0	8.5	0.045153	521.0665389	0.107255432	0.8861	all 3
38.5	1100.5	6.5	0.045080	513.9721974	0.109035325	0.7851	all 3
35.0	1128.0	8.5	0.045038	521.7408294	0.107684320	0.8803	all 3
35.0	455.0	9.5	0.045038	492.9834202	0.113437754	0.9778	all 3
43.0	1382.5	8.5	0.044516	450.3496651	0.124566331	0.7733	all 3
35.0	185.0	7.5	0.047583	486.2904419	0.058547780	0.7954	sum of error,settling time
35.0	179.5	7.5	0.047505	486.2206432	0.058379101	0.8056	sum of error,settling time
35.0	961.5	7.5	0.047443	529.0755517	0.094964226	0.8956	sum of error,settling time
35.0	799.5	8.5	0.047417	507.3713137	0.099421264	0.9946	sum of error,settling time
33.0	440.0	5.5	0.047371	542.9973982	0.092501661	0.8102	sum of error,settling time
35.0	714.0	6.5	0.047329	527.2188172	0.095840974	0.9255	sum of error,settling time
35.0	982.5	7.5	0.047219	530.2996950	0.095720769	0.8879	sum of error,settling time
35.0	92.0	6.5	0.047173	489.5559617	0.037067440	0.9825	sum of error,settling time
35.0	185.0	7.5	0.047583	486.2904419	0.058547780	0.7954	max overshoot,settling time
43.0	647.5	9.0	0.047464	419.7893571	0.116729411	0.7879	max overshoot,settling time
43.0	694.0	9.0	0.047175	421.0532836	0.117766219	0.7685	max overshoot,settling time
43.5	1227.0	8.5	0.045541	441.1746473	0.121347813	0.7990	max overshoot,settling time
43.0	1227.0	8.5	0.045508	445.2225073	0.120698961	0.8041	max overshoot,settling time
43.0	185.0	10.0	0.045327	412.5033864	0.128119883	0.8349	max overshoot,settling time
43.0	1276.0	8.5	0.045188	446.8974985	0.121916909	0.7951	max overshoot,settling time
43.0	1340.0	8.5	0.044781	449.0007982	0.123509350	0.7821	max overshoot,settling time
43.0	1369.0	7.5	0.047552	464.1491249	0.107467090	0.7404	sum of error,max overshoot
47.5	1040.0	5.5	0.047487	451.7388720	0.110234469	0.5218	sum of error,max overshoot
42.5	1390.5	7.5	0.047325	469.5381147	0.107397959	0.7380	sum of error,max overshoot
43.0	45.5	9.0	0.047169	452.4998876	0.103365600	0.2651	sum of error,max overshoot

```
47.5   1062.0    5.5    0.047112   454.1264981   0.111436878          0.5169  sum of error,max overshoot
43.0    640.0    4.5    0.047081   471.4860775   0.108104618          0.6020  sum of error,max overshoot
43.0   1428.0    7.5    0.047027   466.7246436   0.109300334          0.7282  sum of error,max overshoot
47.5    768.5    4.5    0.047005   455.4985423   0.111642556          0.5456  sum of error,max overshoot
```