TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

EERO HEINÄNEN

A METHOD FOR AUTOMATIC TUNING OF PID CONTROLLER FOLLOWING LUUS-JAAKOLA OPTIMIZATION

Master of Science Thesis

Examiners: Prof. José L. Martinez Lastra and Prof. Joni Kämäräinen

Examiner and topic approved by the Faculty Council of the Faculty of Engineering Sciences on 28th March 2018

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Automation
EERO HEINÄNEN: A Method for automatic tuning of PID controller following Luus-Jaakola optimization
Master of Science Thesis, 57 pages, 7 Appendix pages
October 2018
Major: Factory Automation and Industrial Informatics
Examiners: Prof. José L. Martinez Lastra and Prof. Joni Kämäräinen
Keywords: PID-controller, autotuning, optimization, machine learning, genetic algorithm, Luus-Jaakola, swarm intelligence

Tuning parameters of a robot axis PID controller manually requires resources and expertise. Even a skilled person cannot always produce optimal tuning parameters manually. In addition, even if two robots would be copies of each other and should perform equally well with same tuning parameters, manufacturing tolerances and other physical differences and errors between axes cause them to perform differently with the same parameter settings. Using lower gains to prevent oscillations results in suboptimal performance. A robust autotuning method would increase axis performance, decrease axis tuning expenses and allow finding optimal tuning parameters for each mass-produced axis individually.

This thesis investigates suitable machine learning approach for OptoFidelity's OptoDrive servo controller automatic tuning. Integrated squared error was used as a performance index to evaluate the PID controller tuning. Luus-Jaakola optimization was selected from the learning based optimization methods to optimize the tuning of velocity and position PID controllers in OptoDrive. Controller performance achieved with learning based autotuning was compared to results from manual tuning. To speed up the tuning process, a novel method to adjust model based tuning method with results from learning based method was developed. Both autotuning methods were superior to manual tuning of position controller by decreasing the squared error over 90 %. They also performed comparably to manual tuning of velocity controller. Significance of the results were tested with two-sample Kolmogorov-Smirnov test.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automation
EERO HEINÄNEN: PID-säätimen parametrien optimointi Luus-Jaakola optimointimenetelmällä
Diplomityö, 57 sivua, 7 liitesivua
Lokakuu 2018
Pääaine: Factory Automation and Industrial Informatics
Tarkastajat: Prof. José L. Martinez Lastra ja Prof. Joni Kämäräinen
Avainsanat: PID, säädin, virittäminen, optimointi, koneoppiminen, geneettiset algoritmit, Luus-Jaakola, parviälykkyys

Robotin servoakselin PID-säätimen parametrien manuaalinen viritys on hidasta ja vaatii ammattitaitoa. Inhimillisistä tekijöistä johtuen manuaalinen viritys ei tuota optimaalista säätöä. Tämän lisäksi tuotantotoleranssien ja -virheiden sekä akselien fyysisten erojen takia samat viritysparametrit eivät tuota samaa lopputulosta, vaikka akselit olisivat toistensa kopioita. Pienempien vahvistusten käyttäminen säätimessä vähentää akselin oskillointia, mutta samalla myös alentaa sen suorituskykyä. Luotettava automaattinen viritysmenetelmä kasvattaa suorituskykyä, vähentää manuaalisen työn tarvetta ja mahdollistaa massatuotettujen robottien akselien virittämisen yksitellen.

Työ käsittelee OptoFidelityn käyttämän servo-ohjaimen PID-säätimen automaattista virittämistä koneoppimisalgoritmeilla. PID-säätimen suorituskykyä arvioitiin integroimalla neliöityä paikka- ja nopeusvirhettä. Nopeus- ja paikkasäätimien parametrien optimointiin valittiin Luus-Jaakola-menetelmä. Automaattisen virittämisen jälkeen suorituskykyä verrattiin manuaalisesti viritetyn säätimen suorituskykyyn. Lisäksi työssä kehitettiin uusi mallipohjaisen virityksen optimointimenetelmä. Sen avulla voidaan nopeuttaa säätimen virittämistä tinkimättä merkittävästi säätimen suorituskyvystä. Luus-Jaakola -menetelmä sekä optimoitu, mallipohjainen Ziegler-Nichols -menetelmä tuottivat säätimille merkittävästi parempia virityksiä. Automaattisen virityksen avulla neliöity virhe laski paikkasäätimellä yli 90 % verrattuna manuaaliseen viritykseen. Nopeussäätimen suorituskyky pysyi samana. Tulosten merkittävyys testattiin kahden otoksen Kolmogorov-Smirnov -testillä.

PREFACE

This thesis combines my interests for automation and machine learning. It is great to see that the developed autotuning system is able to tune the axis controllers better than I ever could.

I would like to thank OptoFidelity Oy for the thesis topic and opportunity to work and write the thesis in parallel. Big thanks goes also to my supervisors, professors José L. Martinez Lastra and Joni Kämäräinen for guidance during the thesis work. My motivation always increased after having a conversation with you and I truely appreciate your input on the work. Thank you Hans Kuosmanen, SVP of OptoFidelity, for your ideas and thoughts on the topic and efforts for the thesis schedule. I am grateful to all OptoFidelity employees who gave input for the work and participated manual tuning tests, thank you. Special thanks goes to my friend and colleague Juha Koljonen, who has provided invaluable help, co-operation and ideas during the whole project. This project would not have been possible without your contributions, especially in building the test platform and developing the tuning methods. Huge thanks belongs also to my girlfriend, friends and family. Thank you for the support and constantly pushing me forward.

Shenzhen, 24th October 2018

Eero Heinänen

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

AGA	Adaptive Genetic Algorithm
AR	Augmented reality
CSV	Comma Separated Value, file format for storing data
GA	Genetic Algorithm
GS	Grid Search
IAE	Integral-Absolute-Error, performance measure calculated by integrating absolute error
IE	Integral-Error, performance measure calculated by integrating error
ISE	Integral-Squared-Error, performance measure calculated by integrating squared error
ISTES	Integral-Squared-Time-Error-Squared, performance measure calculated by integrating square of error multiplied by squared time
ISTSE	Integral-Squared-Time-Squared-Error, performance measure calculated by integrating squared error multiplied by squared time
ITAE	Integral-Time-Absolute-Error, performance measure calculated by integrating absolute error multiplied by time
ITSE	Integral-Time-Squared-Error, performance measure calculated by integrating squared error multiplied by time
KH	Koljonen-Heinänen, method for model based tuning method optimization
K-S	Kolmogorov-Smirnov, statistical test
ML	Machine Learning
PID	Proportional-Integral-Derivative, controller type
PSO	Particle Swarm Optimization
RS	Random Search
TCP/IP	Transmission Control Protocol/Internet Protocol, networking protocols
VR	Virtual reality
ZN	Ziegler-Nichols, model-based tuning method

LIST OF SYMBOLS

c_{1-2}	Coefficients for PSO velocity terms
e	Controller error value
f	Fitness value of the solution
f'	Fitness value of the better solution in GA crossover
FF	PID controller's feed forward term
f_{max}	Fitness of the best solution in population
$\overline{f}$	Average fitness of the population
G	The best solution of the current PSO swarm
j	Number of optimized parameters in GS
k_{1-4}	Constants for Adaptive Genetic Algorithm
K_d	PID controller's derivative gain
K_i	PID controller's integral gain $(1/T_i)$
K_p	PID controller's proportional gain
K_u	PID controller's critical P-gain
l_i	Number of possible values for ith parameter in GS
λ	Hype parameter for fitness function
M	Number of data points in one CSV file column
μ	Mean
N	Number of parameter combinations in search space
P	The best location individual PSO solution has recorded
p_c	Genetic algorithm crossover probability
p_m	Genetic algorithm mutation probability
r_t	Process variable setpoint
$rand$	Random number from range [0,1]
σ	Standard deviation
t	Time, [s]
T_d	PID controller's derivative time
T_i	PID controller's integration time
T_u	Output oscillation period at critical P-gain, [s]
u	Control variable
V_i	Velocity of one PSO solution candidate
X_i	PSO solution candidate
y	Process variable
y_m	Measured process variable

1. INTRODUCTION

Proportional-Integral-Derivative (PID) controller is the most used control algorithm in the industry and it is still used in 80-90 percent of controlled processes in industry [3, 6]. It is widely tested and accepted de facto control scheme in industry, and thus it is the first solution that should be tested when choosing the controller for a new system or process that utilizes feedback [3]. However, to get the maximum performance out of PID controlled systems, the parameters of the controllers should be determined and selected for each system individually. This is called controller tuning, and controller parameter combination found by tuning process is called tune. The performance as compared to optimal controllers in industry can be as low as 50 %, which means suboptimal usage of a large portion of control systems. Tuning controllers properly can significantly increase process performance. Depending on the process, increased performance produces better quality, increased speed and capacity or lowers downtime and risks of damage. Overall, more efficient systems save resources and create competitive advantage. [6]

Tuning of PID parameters is not a trivial task, especially when there are multiple controllers in system affecting each others. The applications utilizing PID-controllers varies from very slow systems with long delays to real time systems that should and can react to changes immediately. These applications have different requirements for the controller and therefore there cannot be general controller tuning method that would suit all cases. For any new application, the most suitable controller tuning method should be investigated based on the system requirements. Furthermore, an optimal tuning found for one system might not be optimal for a copy due to the hardware differences. Therefore tuning is not only application dependent and the best results are achieved when controllers are tuned individually.

This thesis is about automatic tuning of linear servo axis drive PID controller parameters. Tuning of servo drive controller parameters is required to increase controller performance. In addition, automating the sequence decreases costs of resource intensive manual tuning process. In this work, controller tuning is completed with learning based stochaistic methods and tested with real hardware consisting of OptoFidelity's electronic actuator controller, OptoDrive.

1.1 Background and motivation

OptoFidelity is a robot based testing company that creates test solutions for mobile devices, AR/VR headsets and other touch controllable and smart devices. One OptoFidelity mobile device tester is in Figure 1.1. Test robots usually consist of multiple axes with actuators and different types of applicators attached to them. There are high demands for the performance of the mobile device test robots. They are required to be reliable, fast and accurate, without forgetting competitive pricing for the customer. To meet these requirements, there is a need to have the axes controlled by high performance axis controllers. In OptoFidelity, requirements are fulfilled by using the company's own axis controllers that utilize PID controllers for torque, velocity and position controlling.

Figure 1.1 *OptoFidelity mobile device tester robot.*

Testing needs are usually different in each customer case and robots are often customized for each project. Therefore, the amount of test robots in company's product catalog is increasing. Differences among robots, including axis count and length, materials and component choices such as motors and encoders, result in multiple robots with different dynamics and properties. Varying configurations lead to setting the controller parameters individually for each robot type. Also, when delivering many robots of the same kind, the robots are still individuals and their physical properties vary slightly. Even if the robots would be copies of each others in theory, manufacturing tolerances, aging and errors in assembly cause differences in robot dynamics. That leads to a situation where each controller should be tuned

individually to reach the maximum performance. In practice, individual tuning of the controllers manually is not possible due to resource limitations. Especially mass delivery robots cannot be tuned manually one by one, but an average tune is set and used in all robots. Tune needs to be robust for system changes to work with all copy robots. Because of that, many robots are operating sub-optimally.

Currently in industry parameter tuning of axis controller is often done manually, which consumes lot of resources. It is not unusual that tuning of one axis controller can take hours or in some cases even days to achieve the desirable axis performance. However, even if the tuner would be an expert, manual tuning does not guarantee optimal or even near optimal result, as the goodness of the tune depends on tuner's subjective opinion about the performance. Humans have different understandings about axis performance goodness and there is no single performance index that could describe every aspect of axis performance. Therefore, performance index should be chosen so that controller behavior is objectively optimized based on desired behaviour.

The previously described tuning process and its problems create needs for an automatic axis tuning system. Increasing volumes of the robot deliveries cannot be carried out efficiently and fluently with manual tuning. When considering the whole business case of selling and supporting robotic measurement devices, automating system bring-up steps, such as controller tuning for each robot, creates a competitive advantage. A parallel thesis work of Juha Koljonen [15] studies well-known model based controller tuning methods for OptoFidelity robot controller tuning. In contrast to Juha's work, this thesis explores model free, machine learning (ML) algorithm based approaches. As the controller used in Optofidelity products has more parameters than the ideal textbook version of a PID controller, conventional model-based tuning methods cannot utilize all controller capabilities.

1.2 Objectives

As described in Section 1.1, the problem this thesis is attempting to solve is the inefficient, resource intensive and human-dependent manual servo drive controller tuning process with more automatic methods. This work investigates ML algorithms for the PID controller tuning with real hardware used by OptoFidelity. The main research question is: can machine learning algorithms be utilized to automatically tune OptoDrive servo controller parameters? What performance can be achieved with such algorithms as compared to conventional PID controller tuning methods and manual tuning? These are the research questions this thesis will attempt to answer.

Primary objective of this thesis is to select a suitable stochastic method for tuning OptoFidelity's servo drive controller and implement and test the method. Comparison of manual and selected tuning methods is done with measurements from tests with real hardware. Selection of performance metrics should be considered as learning based optimization methods will optimize only given this as tune fitness function.

Studying ML-based parameter optimization methods and choosing one for a closer study is the main scope of this thesis. Tuning the servo controller with conventional tuning methods is investigated and developed in separate, concurrent thesis work and its results are used for comparison in this work. The system is tested with a linear servo axis with different weights, but other types of axes such as ballscrew axes, are left out of scope. In addition to that, the tuning system will be tested with OptoFidelity's existing hardware and software, and may not be applicable for other servo systems as such.

1.3 Outline

After introduction in Chapter 1, Chapter 2 contains a literature review that includes a short introduction to PID controllers and their performance indices. In addition, Chapter 2 presents a set of parameter optimization methods, focusing mostly on learning based parameter optimization methods. In Chapter 3 the test system is described and the optimization methods and performance indices are discussed and selected for implementation. In Chapter 4 is a detailed description about the automatic tuning system implementation and testing. Also the achieved results are presented and analyzed. Finally, thesis results are concluded in Chapter 5.

2. THEORETICAL BACKGROUND

Controller is a system that attempts to keep the process variable close to the setpoint by adjusting the control signal. Short description of the used terminology is presented in Table 2.1. Process control can be divided to the two categories: open loop control and closed loop control. Closed loop controller can utilize measured value of the process variable and therefore compensate the disturbances in the controlled system. [2, pp. 5-6].

Table 2.1 *System control terminology.*

Term	Description
Control error	Difference between the setpoint and the process variable.
Control signal	Controller output, signal from controller to actuator.
Dead time	Time from setpoint change until process variable changes.
Delay margin	Amount of delay that can be added to process until it becomes unstable.
Overshooting	Process variable exceeding setpoint.
Process damping	Decay ratio of process oscillations.
Process variable	Output of the process.
Setpoint	Desired value of the process variable.
Settling time	Time from controller setpoint change to steady state.
Steady state	State where process variable has settled to defined range.
Steady state error	Difference between process value and setpoint in steady state.

In open loop control, the control command is calculated based on system model and desired action. Precalculated control is then sent to the system to produce desired action. The process is expected to be ideal as modelled and there is no mechanism to compensate errors or disturbances in the process. In contrast, closed loop control utilizes measured process values to adjust control commands, which allows error compensation. Measured process values are real feedback from the process and are compared to controller setpoint. When the difference between a measured process value and a controller setpoint is known, the controller modifies the control signal to reduce the error. [10, pp. 827-828]

The studied controller type in this work is the closed loop PID controller, which is presented in more detail in Section 2.1. Methods to evaluate controller's performance are studied in Section 2.2. After that, methods for optimizing controller's parameters is presented in Section 2.3.

2.1 PID Controller

By far the most popular control algorithm in the industry is a Proportional-Integral-Derivative (PID) controller [2, pp. 59]. It is a closed loop controller that utilizes process value measurements as a feedback from the system. Ideal PID controller has simple structure and can be adopted to a wide range of systems and processes and produce satisfactory control performance. The structure of a PID controller is presented in Figure 2.1. The process setpoint is denoted as r, process error as e, control signal as u, process value as y and measured process value as y_m. The PID controller parameters K_p, T_i and T_d are presented in more detail next.

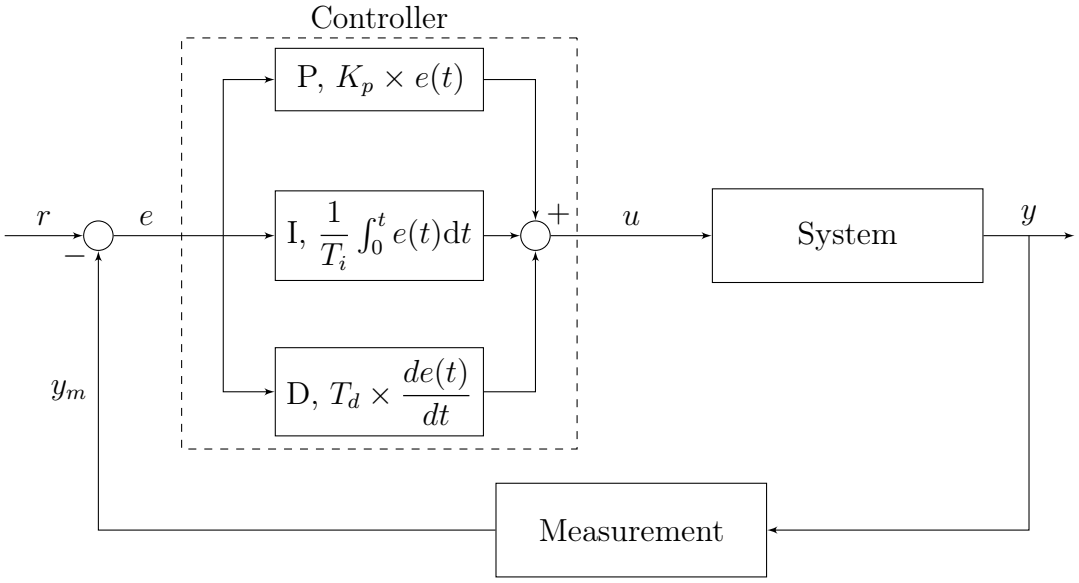


Figure 2.1 *The PID controller model (combined figures from [10, pp. 830] and [2, pp. 71]).*

Adjusting controller parameters to produce desired response of the controlled system is called controller tuning [2, pp. 5]. Controller tuning is required for each system to achieve the best performance of the controlled process. There are many proposed tuning methods available [22]. Each tuning method attempt to produce the optimal tune for a certain type of process. Targets of different algorithms are varying from robustness with big delay margin and process dampening to fast control that reacts strongly to even small changes in the process [39]. Sometimes slower but robust control without fast changes is desired even if reaching the setpoint takes longer, and sometimes quick controllers are preferred over settling time or overshoot minimization. Tuning method provides a tradeoff between attenuation of disturbances, decreasing effects of measurement noise, control robustness and reaction time to setpoint changes [2, pp. 196]. For example, stabilizing room temperature with automatic heater requires robust control and stability is more important than the fast response

time. In the other hand, quick control and short reacting time are important to unstable systems such as inverted pendulums. As can be determined from the various types of controllers and tuning methods, choosing a proper tuning method is crucial to get the desired result for the selected system.

Following sections describe ideal PID-controller structures and basic PID-controller tuning methods.

2.1.1 PID controller terms

Ideally, a PID controller is based on three controller terms that are computed and combined to get a new control value. These three, are proportional, integral and derivative controller terms.

The proportional term (P-term), called also the proportional gain, is a proportional correction term to the control error $e = r - y_m$. If only the proportional term is used, the control output u is proportional to the control error, which often leads to steady state error. To reduce the steady state error, integral of the control error is used. Integral term (I-term) ensures that even a small control error always leads to increasing or decreasing the control signal, until the error reaches zero. I-term is an automatically tuned bias term of the controller. Due to process dynamics, a change in process output does not immediately change after changing the control variable. Therefore, the control error does not change and the controller has a delay when correcting changes of the process. To produce more aggressive and reactive control and to decrease the control delay, derivative term (D-term) is utilized. The derivate of the control error indicates direction of the process change and thus reflects the immediate future behavior of the process. This prediction allows faster control, but due to nature of derivate, D-term is affected by measurement noise. [2, pp. 64-70]

By combining the three terms resulting equation of general PID controller in ideal noninteractive form is

$$u(t) = K_p \times \Big(e(t) + \frac{1}{T_i}\int_0^t e(t)\mathrm{d}t + T_d \times \frac{de(t)}{dt}\Big), \tag{2.1}$$

where t is time, u is the control variable and e is the control error. K_p, T_i and T_d are proportional gain, integral time and derivative time respectively.

Another, parallel form of the general PID controller equation is

$$u(t) = K_p \times e(t) + K_i \times \int_0^t e(t)\mathrm{d}t + K_d \times \frac{de(t)}{dt}, \tag{2.2}$$

where $K_i = \frac{K_p}{T_i}$ and $K_d = T_d \times K_p$.

Equation 2.1 is in noninteractive serial form. Equation 2.2 is in parallel form, where all terms are separated from each other. Both of the forms are commonly used in literature and can be found in [40, pp. 7-8]. The controller form should be considered when selecting PID controller tuning method, as model-based tuning methods are developed for certain controller structures [2, pp. 110].

In addition to the three presented controller actions, there is an additional action which allows utilizing information from former phases of the process. It is called feed forward action and it can be used to eliminate the effects of disturbances before they cause error to the controlled system. Feed forward term, as its name states, will forward information from earlier phase of the system to the controller. If feed forward is used, changes in process can be compensated in control before it has affected the system. For example, when considering a system that heats water flowing to a heater to a certain temperature, the temperature of the water can be measured before it enters the system. Then the heating power can be adjusted before the change in the heater output is seen. The other example is more related to this thesis: if the position setpoint change is forwarded to the velocity controller, the velocity controller can adjust the velocity before the position controller reacts to the position error. [2, pp. 281-284]

Originally PID-controllers were analog devices, but today it is more common to implement controllers with digital devices, such as microprocessors, which requires digital implementation of the PID control scheme. Digital implementation of PID controller is required to take care of sampling the process values. As discrete sampling is prone to signal aliasing, analog signal antialiasing filtering should be applied to a measurement signal before sampling. [2, pp. 93-100]

2.1.2 Cascade controller

Control performance can be increased by measuring more signals from the system and using them in control. Control of different parts of the system can be distributed to multiple controllers. If one controller output signal is an input for the other controller, the controllers are nested. For example, when the axis position needs to be controlled, it creates a need for the inner velocity controller. Furthermore, velocity controller requires nested torque controller. This type of controller combinations are called cascade controllers and they often increase the performance of more complex systems by providing possibility to individually control different characteristics of the system. The example block chart of a cascade controller is in Figure 2.2. [2, pp. 274-276]

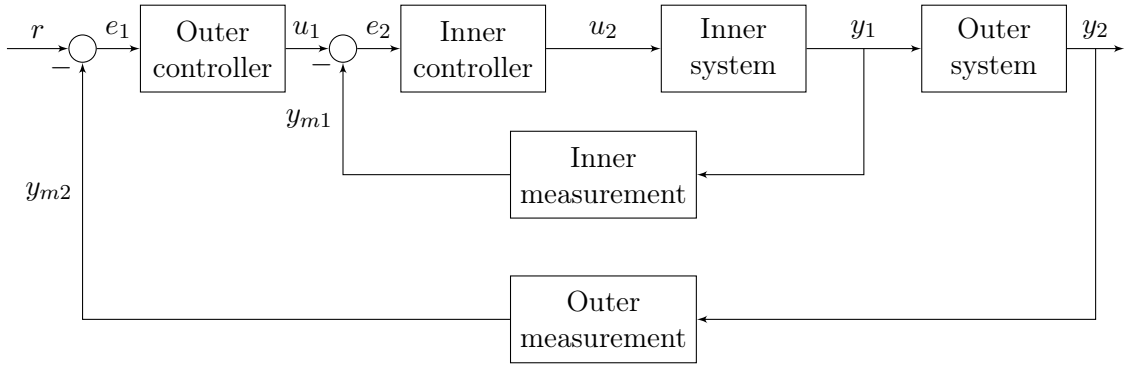


Figure 2.2 *Structure of a cascade PID controller. [2, pp. 275]*

As can be seen from the Figure 2.2, the outer controller output is the input of the inner controller. Cascade control is most beneficial when the process is slow, for example, due to long dead time. Often the inner loop of the cascade controller controls an actuator, while outer controller takes responsibility of actual process value. An example is a water tank of which level is controlled by valve controlled input. In this example, outer control loop uses water level measurement as feedback and gives setpoints to water input valve. If the process was controlled with only one controller, it would be required to have slow response time due to low velocity of the process. With a cascade controller, inner control loop uses valve orientation as feedback. Valve control is quicker process than whole water tank process and can therefore have higher gains. Fast inner process control makes it possible to increase the gains of the outer control loop as the system is more responsive. That results faster response in the whole system. The amount of controllers in cascade is not limited but there can be as many as required. The inner control loop of a cascade controller is recommended to be running at least five times faster than the outer loop to get the maximum benefit from the cascade topology. [2, pp. 274-277]. Downside of the cascade controller is that it has more parameters to tune and is therefore more complex to be tuned and its stability requires special attention [23].

2.1.3 Traditional tuning of a PID controller

Tuning of a PID controller is often done manually or with one of the well-known tuning methods such as Ziegler-Nichols (ZN) tuning method [45]. Comparisons of traditional tuning methods and their performances in different systems can be found in [36] and [31]. Manual tuning is often trial-error based and requires a lot of expertise.

ZN tuning method is based on finding the critical proportional gain for P controller. Critical gain is searched by increasing the controller proportional gain until a step change in the setpoint causes steady, constant amplitude sine wave to the system

output. When the critical gain K_u and the output oscillation period T_u are found, the PID controller parameters are calculated with coefficients in Table 2.2. [45]

Table 2.2 *Ziegler-Nichols tuning method P, I and D gains [45].*

Controller type	K_p	T_i	T_d
P	$0.5 \times K_u$	-	-
PI	$0.45 \times K_u$	$T_u/1.2$	-
PID	$0.6 \times K_u$	$T_u/2$	$T_u/8$

Vendors might offer their own PID controller tuning guides, such as National Instruments guide [33], Granite Devices guide [37], NovaTech guide [28] and Precision MicroControl guide [30]. The guides utilize each company's own software and hardware and might not be suitable for other controllers or systems. According to these guides, tuning is still an iterative process and guides are attempting to reduce amount of trial and error that is required to achieve a good tune. In these guides, performance of the tune is determined by user from tuning graphs, which make tuning process highly dependent on user knowledge and expertise.

2.2 PID controller performance indices

Performance indices of PID controllers can be divided to two categories. The first category includes characteristics that can be explicitly found from step response of the system, such as rise time, settling time, decay ratio, overshoot and steady-state error [2, pp.122, 127-128]. The indices in the second category are calculated from the step response by integrating the process error, thus called integral performance criteria [1]. Those are computationally more demanding than the indices in the first category, but they are more comprehensive since performance is calculated over the whole error data, not only from one feature of the system response.

There are four commonly used PID controller integral performance indices: Integral-Absolute-Error (IAE), Integral-Squared-Error (ISE), Integral-Time-Absolute-Error (ITAE) and Integral-Time-Squared-Error (ITSE). IAE is used for example in [1, 23, 25], ISE in [23] and ITAE in [2, 25]. All of them are also described in [2, pp. 128]. Other integrative indices such as Integral-Squared-Time-Error-Squared (ISTES) and Integral-Squared-Time-Squared-Error (ISTSE) are also available [22]. The comprehensive PID tuning rule summary in [22] shows that these integral performance indices are often utilized in PID controller performance comparisons.

Since the error is integrated a lower value means better performance. The equations

of the most commonly used performance indices are

$$IAE = \int_0^\infty |e(t)| \, \mathrm{d}t \tag{2.3}$$

$$ISE = \int_0^\infty e(t)^2 \, \mathrm{d}t \tag{2.4}$$

$$ITAE = \int_0^\infty t \times |e(t)| \, \mathrm{d}t \tag{2.5}$$

$$ITSE = \int_0^\infty t \times e(t)^2 \, \mathrm{d}t \tag{2.6}$$

As the name of IAE states, the measured error absolute value is integrated without any additional terms or multipliers. Absolute value of error is used to prevent errors with different sign from compensating each others, as would be the case with IE (Integral-Error) performance index [2]. All errors over time get equal weight when calculating IAE and effect depends linearly on the magnitude of error. In other words, time of occurrence of the error does not affect the IAE performance index and doubling the magnitudes of errors doubles also IAE. Similar to IAE, time does not affect the performance value of ISE. However, as error is squared, errors with larger magnitude are emphasized. Thus it can be seen that a PID controller tuning based on ISE will result in a smaller overshoot compared to IAE but longer settling time compared to time weighted ITAE and ITSE [44].

Alternatives to the step response indices can be achieved by weighing the error by the time of occurrence. ITAE and ITSE use time as a multiplier for the error during integration. That reduces the impact of the error caused by setpoint change in the beginning of the step response test. In addition to that, it emphasizes errors occurring later in the step response test, which will help minimizing remaining oscillations and steady state error. [44]

Presented performance indices measure different properties of the PID controller performance. Characteristics of the step response, such as rise and settling times and controller overshoot, can be used to measure the performance. If the system has requirements for those values, it is reasonable to choose them to evaluate the performance of a controller. For example, in some cases too high setpoint overshoot cannot be tolerated in the system and therefore measuring and optimizing that exact feature is justified. In contrast to indices calculated based on one key point of step response, integrative performance measures describes the overall setpoint error. This means that the same value for performance can be achieved with very different step responses. It should be noted that measurement length affects these performance indices as the sum of errors is calculated and even the smallest errors are noted. When the error is squared, it will emphasize the effect of larger errors

and correspondingly decrease the impact of smaller errors. Squaring the errors can be used when the maximum error should be minimized and tracking of the setpoint is important feature for the system control. Depending on the controller structure and the use case, also time can be used as a multiplier for the error. If it is crucial to get system to stabilize to the setpoint but the path to setpoint is not important, multiplying the error by time will emphasize the errors in later, more important parts of the control. Also, if the setpoint change is a sharp step without any smoothing, error will be large near the quick changes. In that case, if all error points are considered equally important, faster response controller tuning will have better performance than the slower controller with better stable state control. However, if the error is time weighted, the effect of error at setpoint change time is decreased and stable state performance is emphasized. Choosing the right performance index for a controller is not a trivial task as requirements and desired behavior are different for separate applications.

2.3 Parameter optimization

Alternatives for manual and conventional (eg. Ziegler-Nichols [45]) controller tuning methods are model-free, iterative, nonlinear optimization based tuning methods. These methods are often inspired by biology, such as evolution [16, 35] or behavior of animals [7, 13, 42]. A comparison between conventional PID-controller tuning methods and optimization methods is in [27]. Three nonlinear methods are presented in more detail in Section 2.3.1: Grid Search (GS), Random Search (RS) and Luus-Jaakola (LJ) optimization. From evolution based optimization methods Genetic Algorithm (GA) and Adaptive Genetic Algorithm (AGA) are studied and from the group of animal behavior based methods Particle Swarm Optimization (PSO) is presented.

2.3.1 Parameter search methods

Multi-dimensional parameter optimization can be done by searching the best solution from the parameter space by testing different parameter combinations and measuring their performances. The most used strategies for hyperparameter optimization are Grid Search, Random Search and Manual search [4]. These methods are general solutions to optimization problems. However, as they are trial-error based approaches, they require a lot of computation time.

Manual Search

Manual search (MS) requires a user to change the parameters, determine the new parameter set performance and repeat this sequence until the system performance is at the desired level. Experienced user can decrease iteration count by leaving clearly unsuitable parameter combinations out. However, reproducing MS optimization results is hard, if not impossible, as the whole optimization process depends on a individual user. Also, determining the performance manually slows down the optimization process. In PID tuning context, MS corresponds to manual tuning described in 2.1.3.

Grid Search

Grid Search (GS, parameter sweep) is a well-known brute force optimization method. In GS all possible parameter combinations are evaluated. Therefore, a finite set of values is defined for all parameters. Higher parameter resolution means denser grid and more accurate search. As a downside it also increases the number of parameter combinations. GS is guaranteed to find a global optimum in defined parameter space if the grid density approaches infinity. The solution is optimized based on the chosen fitness function. Fitness functions were described more closely in Chapter 2.2 in the form of the PID controller performance indices. Grid Search can be considered as a brute-force method for parameter optimization. The size N of a parameter space for GS can be calculated by Equation 2.7:

$$N = \prod_{i=1}^{j} l_i, \tag{2.7}$$

where j is the number of parameters and l_i is the number of possible values for the i^{th} parameter. As can be seen from Equation 2.7, parameter space size increases rapidly when the number of parameters and grid density increase. [4]

The search is exhaustive, which can be considered both positive and negative feature of the method. The globally optimal solution will be always found from the given parameter space, but to achieve that the whole parameter space needs to be investigated and parameter combinations tested. The GS algorithm is easy to implement. Grid Search requires a small amount of prior knowledge, including limits and resolutions for each optimizable parameter. However, it is often the case that the resources are limited and especially in the case of multi-parameter optimization, GS is too resource intensive method to be used. It can be used in hyperparameter optimization tasks and is implemented in commonly used Scikit-learn library [24].

Although the method can find the global optimum, it requires a lot of resources with higher dimensional problems [4].

One beneficial aspect of GS is that it is extremely parallelizable. Since all parameter combinations are known beforehand, they can be all tested in parallel if the system allows running parallel tests, for example in simulations.

Random Search

Random Search (RS, also Random Sampling, Monte Carlo method), is one of the simplest methods to search the optimal solution to an optimization problem [34, pp. 248-250]. While Grid Search has equidistant, predefined grid of parameters that is exhaustively searched through, in RS a collection of randomly generated parameter combinations is used. Because parameter combinations are randomly chosen from the parameter space, grid is not evenly formed and is able to choose any value in the parameter range. That is a desirable property of parameter optimization method when the amount of parameters is high and some parameters might be more important than others. It can be seen from visualization of GS and RS in Figure 2.3.

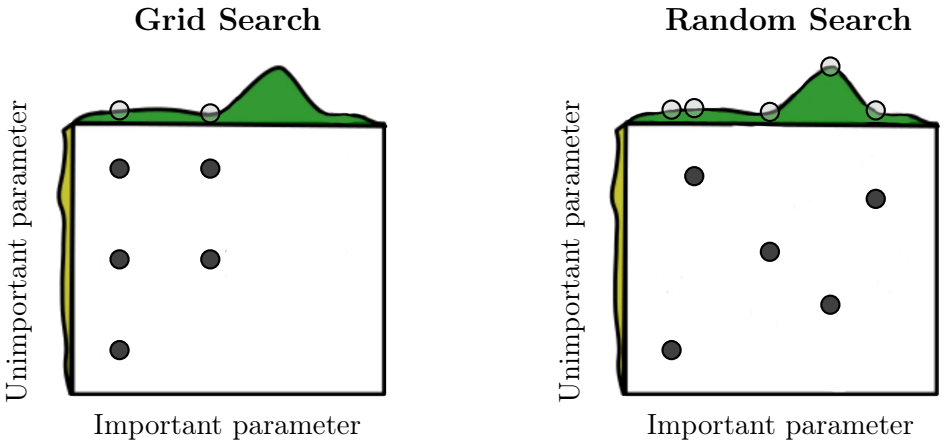


Figure 2.3 *Parameter space coverage comparison between GS and RS [4].*

Although the parameter combinations can cover the whole parameter space, the randomness is also the main downside of the method. Especially with a small amount of points it is highly likely that some parts of the parameter space are left unsearched. Evaluating more parameter combinations increases the probability of covering the whole parameter space but requires more computation time [4]. RS is particularily suitable for two kinds of applications. The first application type are those that have plenty of sufficient solutions and one can be found easily by randomly testing different candidates. In this case RS would be simple and efficient method for finding one sufficiently good solution. The second type includes applications where the search space is not coherent and solution does not tell anything about neighbouring

solutions. One example of this is the search of prime numbers. In that case, search should not be directed to any specific area or range as found prime numbers do not indicate if the numbers near them are prime numbers or not. Similar to GS, also RS is extremely parallelizable. [34, pp. 248-250]

Luus-Jaakola Optimization

In 1973 a general parameter optimization method Luus-Jaakola (LJ) was developed as a goal to create a simple optimization method for nonlinear problems [20]. The LJ procedure is described as a pseudocode in Algorithm 2.1.

Algorithm 2.1 Luus-Jaakola optimization.

```
1: procedure LuusJaakolaOptimization
2:     X̄ ← initialSolution                          ▷ Initialization
3:     r̄ ← initialRanges
4:     populationSize ← selectedPopulationSize
5:     while not TerminationCondition do             ▷ Optimization
6:         for i in range(populationSize) do         ▷ One generation
7:             ȳ_i ← RANDOM(−r̄/2, r̄/2)
8:             x̄_i ← X̄ + ȳ_i
9:             EVALUATEFITNESS(x̄_i)
10:        end for
11:        X̄ ← bestSolutionOfPopulation
12:        r̄ ← r̄ * (1 − ϵ)                          ▷ Shrink range, 0 < ϵ < 1
13:    end while
14:    return X̄                                     ▷ The final solution
15: end procedure
```

The proposed method achieved its goal as a simple, general parameter optimization method. The implementation is straightforward (Algorithm 2.1) and the method does not require complex calculations. Generality comes from low constraints to the problem. It is also semi-parallelizable: goodness of parameter combinations in one iteration can be calculated in parallel, but since the result of one iteration affects the next ones, it makes the method more sequential. The method is not guaranteed to find the global optimum, but when given enough iterations and shrinking the parameter region slowly enough, the method will usually find a good solution robustly and it is not affected by local optimums. As the method does not use gradient calculation at any point, it can be used in non-differentiable problems [20]. LJ method has been successfully used for optimization applications such as model reduction [19], optimizing optical coating [21], finding solutions for nonlinear system equations [29] and also in PID-controller tuning [26].

2.3.2 Evolutionary Algorithms

Evolutionary algorithms are general, population based optimization methods used widely in global optimization problems. They are inspired by evolution theory which appears to work in nature. In these algorithms, one generation consists of individual solution candidates for the optimization problem. The fitness of all solutions of the population is evaluated and the next generation is created based on the previous generation results. Following sections describe commonly used operators of evolutionary algorithms and two optimization methods that are grouped under evolutionary algorithms.

Genetic Operators

The evolutionary process in evolutionary algorithms is achieved with three genetic operations. Operations are inspired by real events of evolving genes. These three operations are crossover, mutation, and selection, and they are used as genetic operators in the algorithms. In addition, GA operators include fitness evaluation and termination operators. [16, pp. 11-12]

To understand these operators, concept of genotype-phenotype mapping should be understood. The genotype is a genetic expression that describes the real solution, the phenotype. Mapping abstracts the solution to easily combinable and modifiable form. Therefore, if the solution can be described in continuous space, the mapping might not be needed at all. If mapping is used, it should not introduce bias to the solution representation. That is to avoid biased phenotype change when genotype is modified in unbiased way. In digital context, bit sequences can be considered as genotype-phenotype mappings. Unbiased example of a bit sequence genotype-phenotype mapping is a status bit sequence: every bit in the sequence has specialized meaning and all bits are equally important. In contrast, if the phenotype is a number and genotype is binary representation of the number, unbiased change in bit representation will result in biased change in original value as some bits are more significant than others in binary expression. [16, pp. 15]

Crossover is the operator that combines genes or properties of parent solutions. Even though in nature the child has usually two parents, the GA operator can be extended to combine even more parents. If the gene is represented as a bit string, the common way to mix the parents is n-point crossover. The bit strings are divided at n positions and the child solution is created by randomly using parts from both parents. With this operation the resulting solution may have properties that either one of the parents did not have. Also continuous ranges can be used instead of bit strings: in that case crossover operation can be applied by calculating the mean of

each part of the n+1 parts the parents are divided to. For example, when one-point crossover is applied to solutions of 1010 and 1100 possible resulting offsprings are 1010, 1100, 1000 and 1110. For continuous values, consider crossover for solutions $(1, 5)$ and $(3, 3)$. The resulting offspring would be $(2, 4)$, the mean of the parent solutions. As can be seen from the examples, the offspring solution candidates may be different from any parent. That allows the evolution process to generate new, previously unseen solutions to test. [16, pp. 12-13]

The second GA operator is mutation. It changes the genes of the solution randomly which will result in new solution candidates. The basic version of mutation operator has three requirements: reachability, unbiasedness and scalability. Reachability means that the mutation operation should be able to modify the gene to any value in the range of the gene. The probability of reaching any point is not required to be equal, but reachability of any point will result in possibility to achieve the global optimum solution instead of a local one. Unbiasedness implies that there should be no preferred direction of the random change. Scalability of the strength of modification is the third requirement. All of the aforementioned requirements can be achieved for example by using Gaussian distribution in mutation. Gaussian distribution allows reaching any point in gene value range, is not biased to any direction and the strength of mutation is adaptable with standard deviation of the distribution. [16, pp. 13-15]

The third operator, selection, is used to select the parents for generating a new generation of solutions. The selection should prefer the fitter solutions for parents of the next generation to guide the population evolving towards the optimum solution. Elitist selection achieves this by selecting the best solutions based on evaluated fitness values. In contrast, comma selection selects survivor solutions randomly, but probability of each solution is calculated with its fitness. Plus selection extends comma selection by also selecting parents of selected solutions. Comma and plus selections enable suboptimal solution candidates to be selected as parents. Thus there can be child solutions generated from less optimal parents to get more coverage on the search space. However, even if it would be temporarily harmful for the search algorithm, the fittest solutions should not be inserted to next population. By discarding the fittest solutions the algorithm is able to overcome the problem of converging to the local minimum. [16, pp. 15-17]

Genetic Algorithm

Genetic algorithm (GA) is an optimization method under evolutionary algorithms. The operators described in Section 2.3.2 are used when utilizing the power of evolution for optimization problems. GA has been successfully used in optimization problems

such as antenna design in [8] and multiple times in PID controller tuning in [9, 17, 39]. GA optimization process is shown as pseudocode in Algorithm 2.2.

Algorithm 2.2 Genetic Algorithm optimization [16, pp. 12].

```
1: procedure Basic Genetic Algorithm
2:     Initialize population
3:     repeat
4:         repeat
5:             crossover
6:             mutation
7:             phenotype mapping
8:             fitness computation
9:         until population complete
10:        selection of parental population
11:    until termination condition
12: end procedure
```

If used as such, GA requires hyperparameter tuning for each problem. The method's hyperparameters are population size, mutation probability and crossover rate. Increasing population size increases computation time proportionally, as there are more solutions generated and tested. In return the robustness of the algorithm rises for the same reason. Mutation probability affects on how much the individual solutions are modified between the generations. Higher mutation probability allows wider search of the parameter space, but it decreases the optimization convergence rate as the solutions are more likely to move in the search space randomly [16, pp. 24-26]. Crossover rate describes the weights of the parents when applying crossover operation to create new offsprings [16, pp. 12-13]. Also, fitness function of the algorithm should be carefully chosen to describe the goodness of the solution properly as the whole optimization process is guided by the solution fitnesses [16, pp. 15-16]. In addition to those, the selection operator method and termination criteria should be selected based on the problem domain. Termination criteria often includes convergence of the optimization process and maximum iteration count criteria [16, pp. 17-18].

GA has also been criticized by its inefficiency: even though the evolution process undoubtedly will lead to decent solution, the question is more of if it can find better solution with less resources and complexity than other optimization methods. The problem of GA is described in [34, pp. 265-266]: *"Indeed, the analogy with evolution — where significant progress require millions of years — can be quite appropriate."*

Adaptive Genetic Algorithm

To overcome the problems with GA while still utilizing the power of evolution Adaptive Genetic Algorithm (AGA) was developed. It is strongly based on GA

and the only difference of these two is in adjusting crossover rate and mutation probability automatically during the optimization process. Because original GA has tendency to find local optimum with low mutation probability p_m and crossover rate p_c , those are adaptively adjusted. When the population average fitness approaches population's best solution fitness, p_m and p_c are increased to explore the parameter space with less fitter solution candidates. In contrast to that, those values are decreased when the population has scattered around with different fitnesses. The behavior is achieved with equations 2.8 and 2.9. Constants k_1 and k_2 values should be below one to constrain p_c and p_m to range $(0.0, 1.0)$. The fitness of the best solution of the population is denoted with f_{max}, f' is larger of the crossed solutions' fitness values, $\overline{f}$ is the population average fitness and f is the fitness value of the solution itself. [35]

$$p_c = \frac{k_1 \times (f_{max} - f')}{f_{max} - \overline{f}}, k_1 \leq 1.0 \tag{2.8}$$

$$p_m = \frac{k_2 \times (f_{max} - f)}{f_{max} - \overline{f}}, k_2 \leq 1 \tag{2.9}$$

By using these equations to adapt the GA parameters, solutions with good fitness are protected with low mutation probability and crossover rate while worse solutions are modified more to overcome local minimum problems. To prevent p_c and p_m getting values over one there are two more constrains 2.10 and 2.11 presented:

$$p_c = k_3, f' \leq \overline{f} \tag{2.10}$$

$$p_m = k_4, f \leq \overline{f}, \tag{2.11}$$

where values of k_3 and k_4 are constants from range $(0.0, 1.0)$.

AGA has been used in Alzheimer's disease progression prediction [38] for choosing the suitable feature set to use for disease diagnostics. Suitability and performance improvement over GA has also been shown in combat force allocation simulation [5] which is a multi-objective problem. Considering the topic of this thesis, it has also outperformed Ziegler-Nichols and classic GA methods in PID controller parameter tuning [18].

2.3.3 Swarm Intelligence

The third presented category of model-free optimization methods is Swarm Intelligence. The inspiration for these optimization algorithms comes from group behavior

in nature. For example, there are methods based on bee swarms [12], elephant herding [42] and many more. Compared to evolutionary algorithms, swarm intelligence methods does not generate new populations from previous ones but rather moves the solutions in swarm towards the more optimal solutions. Otherwise the approach is rather similar to evolutionary algorithms: swarm solution fitnesses are evaluated and solutions modified until some termination criteria is met. One of popular Swarm Intelligence method is Particle Swarm Optimization that is presented in more details next.

Particle Swarm Optimization

Until 1995 Swarm Intelligence methods were based on animal behaviors such as bird flocking and fish schooling. Then human social behavior based Particle Swarm Optimization (PSO) was developed. It extends the previous methods' relations to physical phenomena with cognitive and experiential variables of humans. As in the evolutionary algorithms, also PSO population, swarm, consists of solution candidates to the optimization problem. The solutions are called as particles and the i^{th} d-dimensional particle denoted as $X_i = (x_{i1}, x_{i2}, ...x_{id})$. The solutions dimensionality depends on the number of optimizable parameters. Each particle has also velocity $V_i = (v_{i1}, v_{i2}, ...v_{id})$ which allows particle to move in the search space in each iteration. After the fitnesses of particles are evaluated, velocities of particles are updated so that they would approach the best particle of the current swarm $G = (g_1, g_2, ...g_d)$ and the best location that particle itself has been at during optimization process $P = (p_1, p_2, ...p_d)$. As the best particles attracts the whole swarm towards themselves, the particles will approach the globally best particle after a reasonable amount of iterations. [13]

The equation for updating particle's velocity is

$$v_{id} = w \times v_{id} + c_1 \times rand_1 \times (p_{id} - x_{id}) + c_2 \times rand_2 \times (g_{id} - x_{id}) \tag{2.12}$$

where c_1 and c_2 are constant weights for velocity terms and $rand_1$ and $rand_2$ are random numbers from range $[0, 1]$. It has been proposed to use the weight factor w for current velocity but the optimal values have been found near one [32]. The recommended value for constants c_1 and c_2 is 2, since it results in particle to be directed to both global best and its own best locations with equal strength [32]. Also, with $c_1 = c_2 = 2$, the particles have enough momentum to move past their targets about half of the time. It allows more exploration of the parameter space and to avoid convergence to local optima [13]. After updating particle velocity, its position

is updated with straightforward Equation 2.13 [13].

$$x_{id} = x_{id} + v_{id} \tag{2.13}$$

The hyperparameters that should be chosen are swarm size, constants c_1 and c_2 and velocity weight factor w. In addition, a fitness function must be selected and optimization termination criteria should be defined. Considering the algorithm and its hyperparameters, it has achieved one of its goals as being a simple method to implement.

2.3.4 Parameter optimization summary

The described parameter optimization methods were divided to three categories: parameter search, evolutionary programming and swarm intelligence. Each category and method have their strengths and weaknesses and there is no method that would be better than others in every problem. Goodness of the method for specific optimization problem depends on the problem properties. As can be seen from Figure 2.4, simpler problems with uni-modal fitness function can be solved by greedy algorithms which will quickly find the only global optimum. If gradient of the fitness function is not known or it does not contain useful information for finding the optimum value, as is with multi-modal functions, randomized or stochastic optimization methods that do not require gradient calculations are more suitable [14, pp. 4-5].

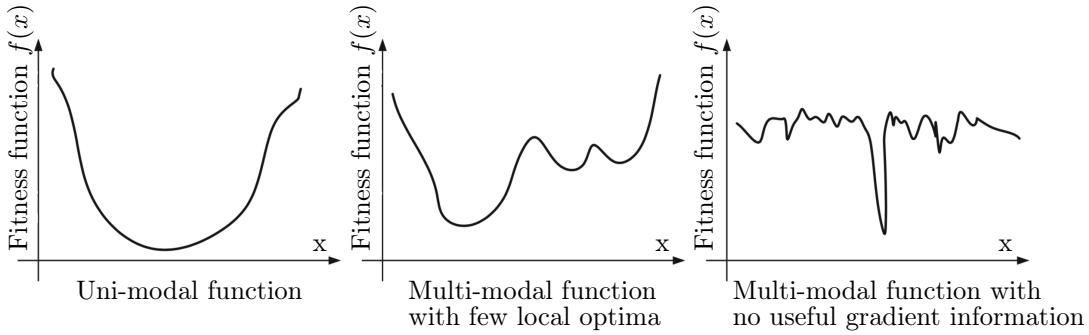


Figure 2.4 *Examples of uni-modal and multi-modal functions [14, pp. 4].*

Parameter search algorithms are generally easy to implement, but they often require more computation time to find a good solution than the evolutionary and the swarm intelligence algorithms. Grid Search and Random Search are brute force -like methods for finding the suitable solution from defined search space. They rely on searching the optimal solution from the whole parameter search space systematically and randomly. Luus-Jaakola algorithm succeeds at keeping the implementation simple while adding more intelligence to parameter searching. With LJ the optimal

solution is searched randomly, as with RS, but the search space size is reduced during optimization. Therefore the algorithm converges to small solution range or to one solution, depending on the method implementation.

In contrast to parameter search methods, evolutionary algorithms and swarm intelligence methods require more work at adapting them to the problem. The solution is required to be represented as a solution candidate that can be modified in a way that the method requires, for example moved in search space in PSO or mutated in GA. Evolutionary programming methods are a bit more complex to implement because of the number of the genetic operators. However, they often find a better solution faster than GS or RS as they are utilizing information from previously evaluated solutions in the search process. That said, finding a good solution quickly requires tuning of the method hyperparameters and the result is not guaranteed even after that. Swarm intelligence approaches utilize the same idea as evolutionary programming for finding the optimal parameters from the parameter space. However, they do not only rely on mutating existing solutions and generating new ones based on good solutions, but search the parameter space with the same solution candidates just by moving them around in search space with certain rules. Using solution candidates to search the less likely solution areas of the parameter space makes evolutionary algorithm and swarm intelligence methods more robust for finding a good solution.

3. METHOD FOR ROBOT AXIS TUNING

As stated in Section 2.3.4, there is no single parameter optimization method that would be the best option for all optimization problems. If the resources would be unlimited, brute force approaches such as GS and RS would be the solution, since those methods are able to evaluate the fitness of all possible solutions and choose the best. However, it is usually not the case with multidimensional optimization problems and more sophisticated optimization methods should be used to get the result in a reasonable time. To choose the optimization method the application and its features and limitations should be investigated more closely. In this thesis, optimization methods are used to adjust servo drive controller parameters to achieve the optimal performance of OptoFidelity robot systems. Combination of the controller parameters is called tune. The system is developed and tested with a typical servo axis hardware used in OptoFidelity, which is described in more details in the following sections. The optimization methods presented in the previous chapter require evaluating the goodness of tune, which can be done by calculating performance index for each tune. PID controller's performance is used as the fitness function for the optimization methods.

3.1 Test system description

To choose the tuning method and performance index to be used in this application, the test system should be investigated from the respects of hardware, software and controller. Since one objective is to create an autotuning system for OptoFidelity, a test system was built with OptoFidelity hardware and software components. One axis was used instead of a multi-axis robot to simplify the development and testing processes. The test system was created for developing and testing the chosen autotuning methods for this and the other parallel thesis work in [15].

3.1.1 Hardware

The hardware was chosen based on commonly used components at OptoFidelity. From hardware point of view, the requirements for a test system were one degree of

freedom with ability to accurately measure system movements. As the autotuning system is planned to be utilized in the final products, it should be possible to autotune axis controllers without additional sensors and measurement hardware. The system consists of the three main parts: an actuator, a controller and a position measurement device. All of the system hardware components are shown in Figure 3.1 and in more detais in Appendix A.

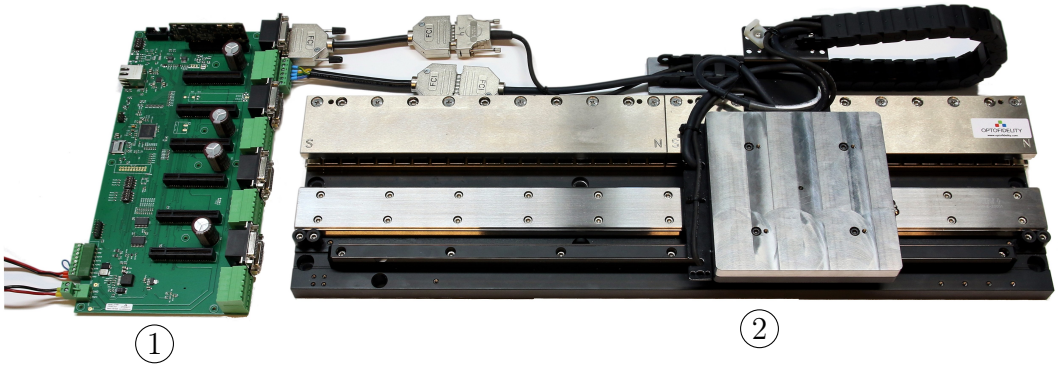


Figure 3.1 *Test system hardware overview. Controller motherboard is marked with 1 and linear servo axis with 2.*

The actuator of the test system is a linear servo axis typically used in OptoFidelity robots. The other option would have been a ballscrew driven axis with rotary motor. A linear motor was selected as target device of this project since OptoFidelity is focusing more on linear motors in its robot platforms. The actuator consists of a linear guide, a forcer, a carriage and magnets, all of which can be seen in Figure 3.2 where the actuator system is partially disassembled.

To provide feedback for the axis controller, an absolute linear optical encoder is used. The resolution of optical encoders vary typically from 0.1 μm to 10 μm. As this system is supposed to be the development platform, the most accurate encoder available was used to get the most accurate results, which led to the selection of 0.1 μm encoder. The servo drive card can read the encoder signals, from which the axis position is calculated. Position information is then utilized in position controller. Axis velocity is calculated from position feedback and used in velocity controller. In addition to components of the linear servo axis, the encoder can be seen in Figure 3.2 attached to carriage.

To test the autotuning performance more comprehensively with one axis, its dynamics were required to be modifiable. The simplest way to modify the test system dynamics was in this case to change its mass. Custom mass plates were manufactured to achieve this. The mass plates in Figure 3.3 can be attached to axis one by one or both at the same time. Two sizes of the mass plates were used to simulate different

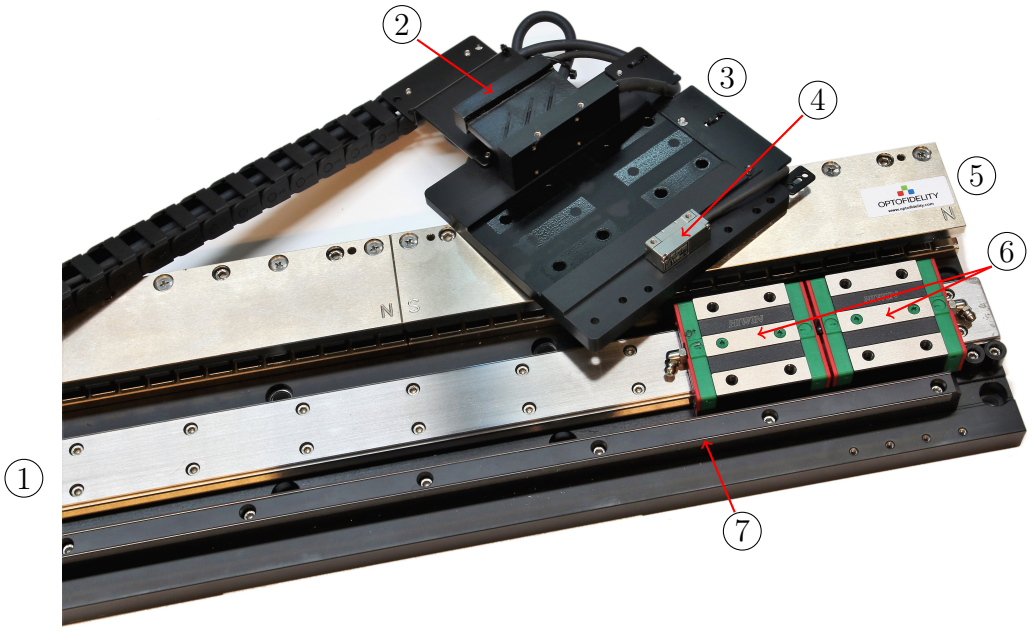


Figure 3.2 *Test system actuator. Linear guide is marked with 1, forcer with 2, carriage with 3, encoder reader with 4, magnets with 5, sliding blocks with 6 and encoder stripe with 7.*

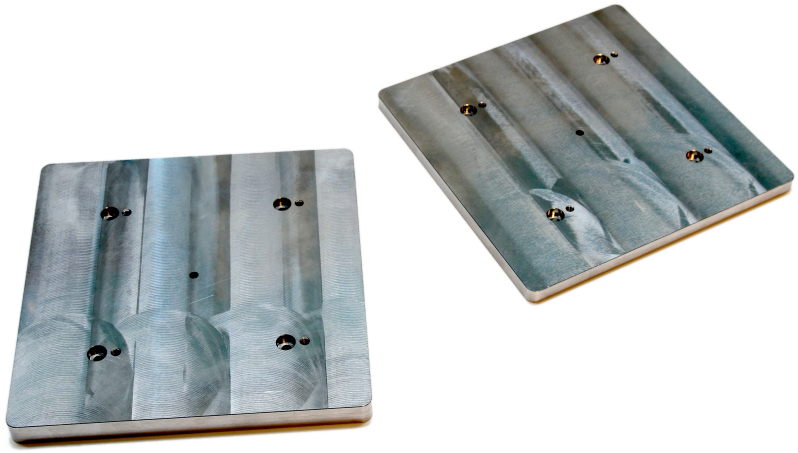


Figure 3.3 *Mass plates used for simulating different axis dynamics. Mass plate 1 on right and 2 on left.*

robot hardware that could be attached to axis. The weights of the mass plates are in Table 3.1.

However, during the preliminary test runs it was noticed that the axis maximum current limit was reached when the both mass plates were attached to axis carriage at the same time. To avoid breaking the hardware, only three different configurations were used in testing: axis without an additional mass, with the mass plate 1 and with the mass plate 2.

Table 3.1 *Mass plate weights.*

Mass plate	weight
1	1.8 kg
2	3.6 kg

The controller hardware is OptoFidelity product. The axis control board consists of a servo drive card (Figure 3.4) and a motherboard (at left in Figure 3.1) that can hold up to seven drive cards. Thus one motherboard can be used to control

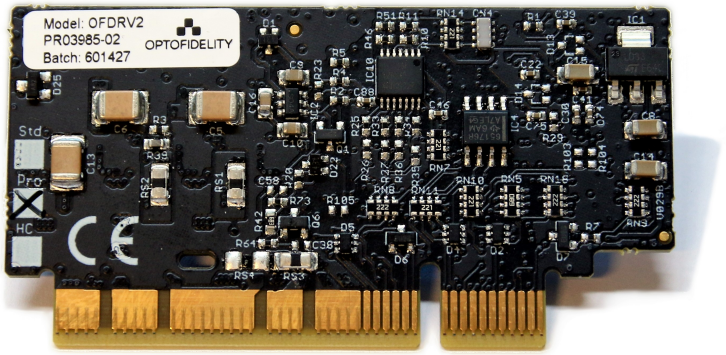


Figure 3.4 *OptoDrive servo driver card.*

seven different servo axes synchronously just by installing drive controller cards to a motherboard. Communication interface to motherboard is Ethernet port which is commonly used in home and office networks.

Drive cards are multipurpose and can be used to control wide variety of electrical actuators from open loop controlled steppers to closed loop controlled servos and other electronic actuators up to 500 watts of power. Suitability for many different electric actuators and control systems makes OptoDrive a reasonable choice for developing robotics platforms and therefore also this autotuning system.

3.1.2 Software and controller

While the selected hardware allows executing movement and measuring feedback, software is required for all of the controlling logic. In this system there are three levels of the software. The lowest level is running on OptoDrive servo controller card. Its task is to run all of the control loops and handle the interfaces to the axis hardware and abstract the hardware interface for higher level software. The middle level software handles communication between servo controllers and computers. A proprietary serial interface is used to communicate with the servo controller card to send commands and fetch system status. In addition to controlling axis, the

serial bus commands can also write new parameters to servo drive. This feature is utilized for example when axis motion controller tune parameters are changed. The highest level implements the logic to create sequences of setting parameters, for example axis speed and acceleration, and moving the axis to different locations along pre-calculated paths. As the hardware is abstracted and axis movements are handled by the servo controller and the serial bus commands, parameters and data can be transferred utilizing existing communication libraries. Therefore this thesis implementation focuses on the highest level of the software.

The logic of the system was developed using Python version 3.5.1. For data handling and algorithm implementation, very commonly used NumPy and SciPy libraries [41] were used. An existing communication library was used to communicate with axis drive motherboard, which then forwards the messages to all servo drive cards. The communication library supports commonly used TCP/IP protocol which makes it easy to control with almost any commercial computer.

The system controller is a cascade PID controller described in 2.1. It consists of three controllers in cascade: outermost controller being position controller, middle one velocity controller and the innest control loop controls axis torque. The position controller is a P-controller with a feed forward term. Integral and derivative actions are not added to the position controller structure. To create a controller that compensates steady state error, velocity controller has all PID actions, combined with velocity and acceleration feed forward terms. Since the velocity controller has integral action, even small position errors can be compensated and the system is able to reach zero steady state error. The torque controller has P and I actions. The axis driver software can calculate tuning parameters of torque controller automatically based on resistance and inductance of the motor, so it is left out of the scope of this thesis.

The controllers are digital and they are running in a loop with predefined controlling frequency. In position and velocity controllers the frequency is 2500 Hz and the torque control loop is running at 20000 Hz frequency. These are predefined values and they cannot be changed without modifying the driver card firmware.

3.2 Tuning algorithm

The core of the automatic controller tuning system is the tuning algorithm. There is no consensus of the most suitable stochaistic algorithm for PID controller tuning. It needs to be selected by considering the properties and requirements of an application in question. All of the presented learning based PID parameter tuning methods have been tested mostly with simulations and not with real hardware. If the simulation

model is made well and corresponds to the real system with disturbances, simulation can be used to compare different tuning methods. However, a real system can have additional disturbances, requirements and limitations compared to the simulation model that need to be taken into account when selecting the tuning algorithm.

Tuning the robot's servo drive controller is a one time task rather than a periodic one. Therefore, time consumed for tuning one axis is not that important property of autotuning system as long as the system is able to find the optimal tune without manual operations. If controller tuning is an automatic process, it can take even hours and the method would still be usable in commercial products. Faster tuning would be beneficial but not required. More important is that the method can produce good tunes repeatably. The autotuning system is required to generate a tune multiple times with constant performance, which would reduce need for manual work in tuning process. While the main advantage of an autotuning system is in removing expertise and manual work, it is also desired to get better performance than with manual tuning. In addition to aforementioned properties, the tuning process should be safe for the hardware. For example, it should cause as little oscillations as possible to prevent hardware failures. Also, safe movements of the axis should be enough to provide necessary information for tuning. Therefore shorter and slower movements are preferred over long or rapid movements in the tuning process.

Considering application requirements and the optimization algorithms presented in Section 2.3, the selection of tuning algorithm is not obvious. Even though it would be tempting to implement GS or RS to solve this problem, it would take too long time to find the optimal values for seven parameters with a reasonable parameter search resolution. Brute force approaches are left out for this reason. For these methods the parameter ranges should be large enough to ensure that a good solution is found, which leads to testing of many unsuitable parameter combinations. That would cause vibrations and therefore could cause damage in the test system hardware, which supports leaving out GS and RS tuning methods from the options. AGA is an improved version of GA with automatic adjusting of mutation and crossover probabilities, which is beneficial mechanism for the optimization method in the context of PID controller tuning [18]. Therefore also GA was left out. That leaves the three optimization algorithms: LJ, AGA and PSO.

LJ is the oldest of the three remaining algorithms. It is versatile, simple to implement and able to find a good solution robustly if given enough time, which in this case means shrinking the LJ parameter range slowly. Also the population size affects the performance, speed and robustness of the method, but that is the same for all algorithms and cannot therefore affect the selection. AGA is a more recent and sophisticated optimization method that will adapt to the optimization situation more than LJ. Its implementation is more complex compared to LJ and PSO and

utilizing AGA requires tuning of the method hyperparameters. When considering the autonomy of the autotuning system, tuning of hyperparameters makes system less autonomous and tuning the algorithm itself requires expertise and manual work. Genetic algorithms are also known as general but often inefficient methods for parameter optimization (Section 2.3.2). PSO algorithm lies between these two: it requires less hyperparameter tuning than AGA while still searching solutions in a more intelligent way than LJ. In preliminary tests it was noticed that the tuning process with PSO caused more oscillations on the axis than tuning with LJ. In addition, there was no significant difference between the resulting controller performance between these two methods in first tests. Therefore LJ was selected as the autotuning method of this research.

Undesired oscillations were also encountered when the initial parameter ranges were large enough to cover tuning axes with different dynamics. Amount of oscillations during tuning were remarkably decreased by using Ziegler-Nichols (ZN) method to get some initial tune and to calculate the initial parameter ranges. While tuning the controller with ZN does not result in the best tune, it was noticed to give good initial controller gains and often perform better than manual tuning [15]. However, since the OptoDrive controller has parameters beyond ZN tuning capabilities, all of the parameters cannot be optimized with ZN and the initial ranges and values needs to be defined manually. These parameters are feed forward terms, and determining their initial ranges will be discussed in more detail in Section 4.2.

3.3 Fitness function

The fitness function is a crucial part of the system optimization. Model-free optimization methods are able to classify system behavior only through the fitness function. For example, if the fitness is based on the maximum error, optimization methods will not take any other features into account and the steady state error may remain poor. In the PID controller context, minimizing the maximum error might seem a good idea at first, but it does not consider oscillations or steady state erros at all. On the other hand, if only oscillations would be minimized, it would result in a poor solution: if the controller gives zero control signal, the system does not oscillate at all, which would be considered as the ideal solution. However, the system would not react to a setpoint, which clearly is not the desired behaviour.

When considering test robots for mobile devices, positioning accuracy is one of the key properties. After the robot has moved to calculated position, all axes should stay in their setpoints to achieve as accurate measurements as possible. Thus the steady state error at the end of the movement should be as low as possible and axis should not oscillate after moving to desired position. As the controllers are in cascade,

velocity controller should not oscillate either to prevent position oscillations. The velocity controller should minimize the error and oscillations in setpoint following task to allow smooth movements of the axis. Minimizing the error in velocity controller results in an accurate speed and well-known path of the axis. When considering common applications of OptoFidelity robots, steady and robust movements are required in swipe tests, where the movement itself is measured instead of path endpoints. To get accurate measurements, axes should also be driven synchronously, which requires accurate trajectory following of all robot axes.

To minimize the errors of controllers, using integrated error of performance test movements is justified. All of the performance indices presented in Section 2.2 would therefore be suitable. The setpoint changes of the test system are not steps but smooth transitions and controllers should be able to follow the setpoints as accurately as possible during the whole transition. Therefore, time should not be used as multiplier for error. As there are no big steps in the test system controller setpoints, it is not necessary to reduce the weight of the errors at the beginning of the movement. Considering that, there are two options, IAE and ISE. ISE minimizes error peaks more efficiently than IAE because the error is squared. As the maximum error should be minimized in robot test cases, ISE is the performance index of choice. It is also widely used performance criteria when designing optimal control systems [17]. For example, it is used along with ITSE and ISTE in [39] and also in more recent research [43].

ISE has two major drawbacks that should be considered before employing it for controller optimization. Setpoint step change creates a step to controller error, which might result in ignoring smaller errors later in step response [44]. It can be avoided by smoothing the controller setpoints to trajectories that the system is able to follow. This is the case with OptoDrive servo drive controllers. The second drawback is that even though ISE by design minimizes errors and utilizing it in controller parameter optimization results in relatively small overshoot, steady state is not reached as quickly as with time weighted performance indices [17]. However, that can even be a desired property of fitness function, if trajectory following is as important as reaching the setpoint. Considering the properties of the four presented performance indices and the system, ISE is the most suitable performance index for this application.

4. IMPLEMENTATION AND TESTING

The methods selected in the previous chapter were implemented to test the methods and their performance with the described test system. A standardized test protocol was defined to get comparable results from different tuning methods. For comparison, manual tuning tests were executed and performances measured in the same test. The following sections present the test, detailed descriptions of the method implementations, followed by results and comparisons.

4.1 Experimental test setup

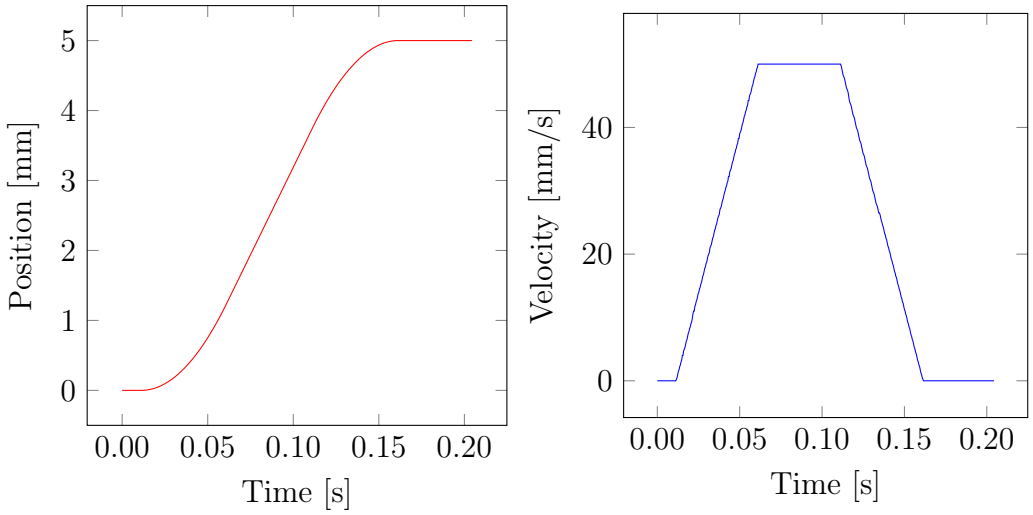
To get comparable results for different tuning parameters, an axis performance test was defined. It presents a common movement of the linear axis, which is in range from 1 mm to 500 mm in OptoFidelity robots. In addition to the trajectory length, also acceleration and velocity profiles were chosen based on typical use cases. The trajectory is required to include acceleration, constant velocity and deceleration parts, followed by axis stopping to position setpoint. Performance is measured from the whole axis usage. Since the tested controller is digital, the gathered data gathered is discrete and a suitable sampling frequency should be selected for the system. Higher sampling rate results in more accurate data from axis movements. However, the test system has a limited buffer size of 2048 samples for collecting data from one movement. That limits the sampling frequency when whole movement is required to be inspected. There are four parameters to be collected to calculate velocity and position controller performances: setpoints and actual values of velocity and position. That lowers the amount of samples from one parameter to $2048/4 = 512$.

The defined test is a tradeoff between aforementioned properties. Longer movement would allow higher acceleration and velocity to be used, but it would limit autotuning system usage to axes with larger movements ranges. Shorter movement allows higher sampling rate. Downside is that maximum velocity should be lowered to get enough data from the constant velocity part of the movement as axis acceleration is limited by maximum current of the servo.

During tuning process the axis controller will be unstable. Ziegler-Nichols tuning process is based on finding critical controller gain. Learning based approaches search

also unstable parts of the tuning parameter search space to find the globally optimal parameter values. With longer movements and higher velocity and acceleration unstable controller tune can cause more damage than with lower values.

The following values were selected empirically: axis movement of 5 mm, velocity of 50 mm/s and acceleration of 1000 mm/s^2. To collect data from the trajectory, the sampling rate was limited to 2500 Hz. The selected trajectory for axis position is presented in Figure 4.1a. Velocity profile of the trajectory is visualized in Figure 4.1b.



(a) *Position trajectory* (b) *Velocity trajectory*

Figure 4.1 *Ideal position and velocity trajectories of the test setup.*

For every test tune, the described test is executed six times: three runs to both directions. Mean values of position and velocity controller performance indices from all six runs were used in tune performance evaluation. Velocity and position trajectory errors from one test run are visualized in Figures 4.2 and 4.3. Resolution of the velocity controller is clearly visible in velocity tracking error plot in Figure 4.2. As the encoder resolution is 0.1 μm and the controller run at 2500 Hz frequency, the resulting velocity resolution is 0.25 mm/s.

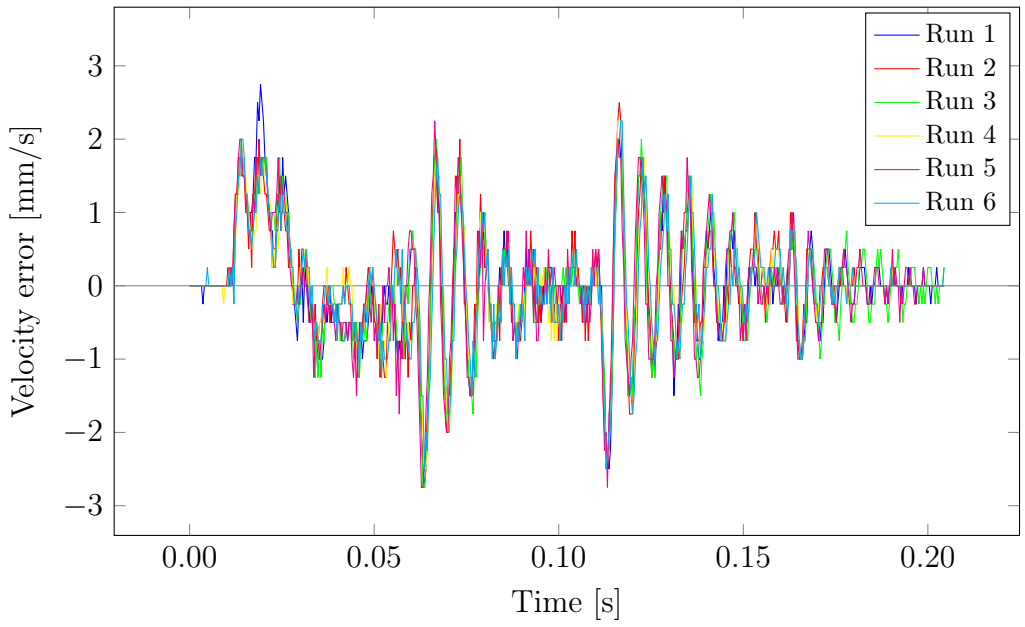


Figure 4.2 *Velocity error during six sample test movements.*

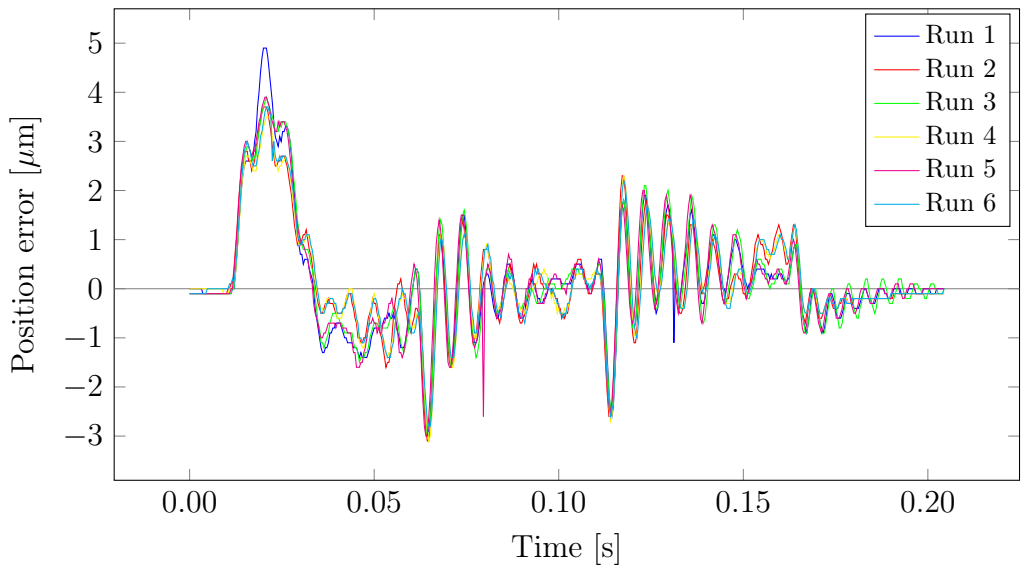


Figure 4.3 *Position error during six sample test movements.*

4.2 Implementation

Luus-Jaakola -parameter optimization method was implemented with ISE performance index as the fitness function. Python was used in the autotuning system implementation. It is the most used programming language within OptoFidelity and the OptoDrive axis controller library has a Python wrapper. In addition to that, Python has numeric and scientific calculation libraries NumPy and SciPy [41] that were utilized in data analysis in this project. The defined automatic tuning sequence is presented in Figure 4.4.

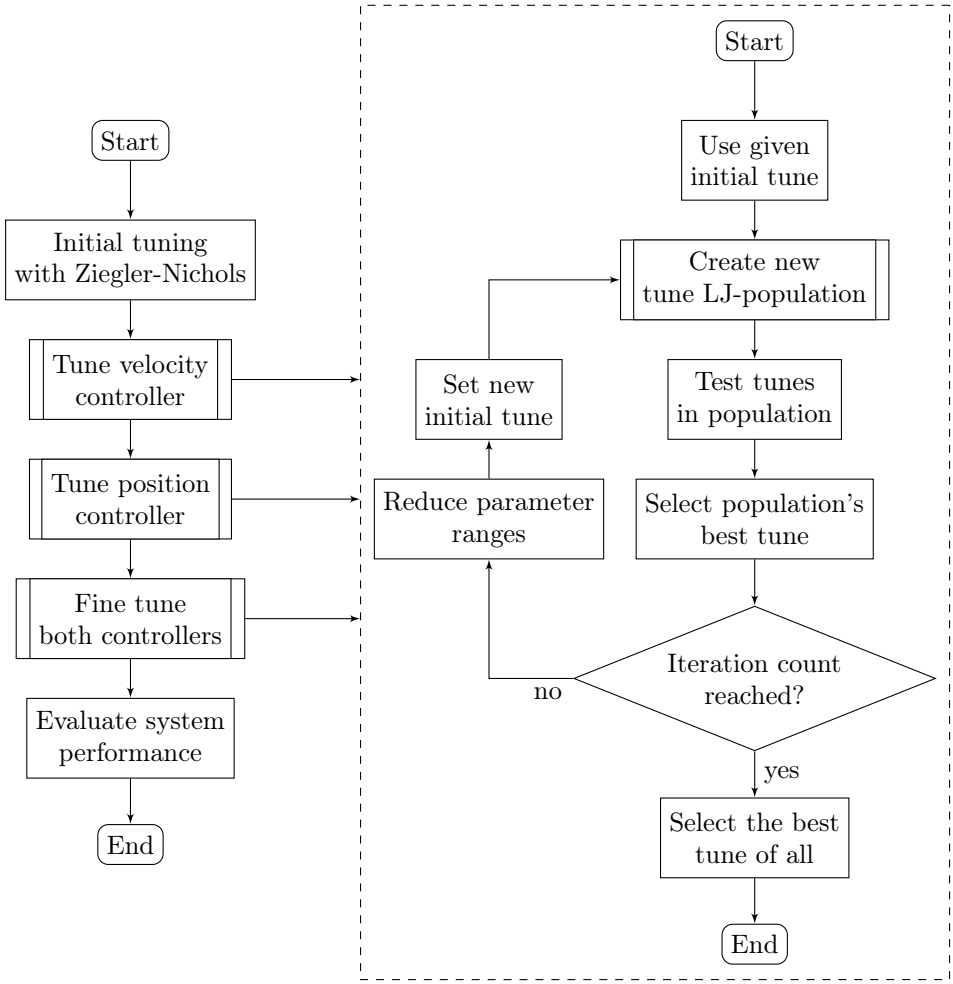


Figure 4.4 *Automatic tuning flowchart. Luus-Jaakola optimization method is used for all tuning steps.*

The tuning sequence was inspired by the manual tuning process. When tuning controllers in cascade, the controllers cannot be fully separated. Therefore, as often in manual tuning, the velocity controller is tuned first and position controller is tuned after the velocity controller gains have been found. As the previously tuned velocity controller is tuned with input from suboptimally tuned position controller, it is required to be tuned again with new position controller. However, after that

the position controller would be tuned with suboptimal velocity controller. To break the loop of tuning both controllers alternately and get the controllers tuned properly together, they are fine tuned simultaneously. That allows optimization algorithm to fit them together without problem of tuning other controller with a suboptimally tuned adjacent controller.

4.2.1 Performance index

The chosen performance index ISE was implemented for both the velocity and position controllers. The software generates a CSV file from the data gathered from the axis movements so the data handling is straightforward with NumPy-library of Python. Required data for calculating ISE consists of position and velocity setpoints and actual values, each of these as a column in a CSV file.

The data collected from the axis is scaled by the servo drive firmware and data downloaded from drive requires re-scaling to get SI-units. However, the correct scaling factors for both position and velocity data can be queried from the drive. For discrete data, the Equation 2.4 for ISE can be written as

$$ISE = \sum_{t=0}^{M} e(t)^2 \tag{4.1}$$

where M is the number of data points in one column of the resulting CSV file. As can be seen from the equation, ISE is simple to implement and fast to calculate. Performance index is calculated for both position and velocity errors.

4.2.2 Optimization method

Luus-Jaakola optimization method was implemented to tune the controllers. LJ-algorithm was implemented with slight modifications to fit for OptoDrive PID controller tuning. Tuning sequence was created to follow the order of manual tuning sequence, where controllers in cascade are tuned from inside out. Sequence is presented in Figure 4.4. Unlike in manual and model based tuning, all of controller's parameters are taken into optimization process.

The test system sets limitations to parameter values. Only the position controller feed-forward term uses floating point numbers. The six other controller parameters are integers, which leads to rounding and loss of accuracy. Also, all parameter values are non-negative. LJ parameter optimization method does not naturally limit the parameter types, ranges, or values and therefore application specific limitations were

implemented. Optimization algorithm is not modified, but the output values are validated and modified if necessary according to the limits. Negative parameter values are interpreted as zeros and integer type parameters are rounded to the nearest integer after LJ-iteration. The position controller feed-forward term is an exception and it is handled as a floating point number.

Initial values of the position controller P-gain and velocity controller PI-gains are gathered with the ZN-tuning algorithm. The position feed-forward term unit is percents and it is by default 100.0, which ideally forwards position errors directly to the velocity controller. The rest of the parameters are initialized to zero values. LJ-method also requires parameter range for each of the parameters. The parameters are not normalized to a same scale and therefore the same range cannot be used for all. For parameters initialized with ZN, the range was selected to depend on the parameter initial value. Other parameter ranges were selected manually for this controller.

The initial range was selected to be a parameter value itself. In practice, the initial value for the parameter is chosen randomly from the range $[0.75x; 1.25x]$, where x is the initial value of the parameter. Bigger $[0.5x; 1.5x]$ and smaller $[0.9x; 1.1x]$ initial ranges were also tested. The smaller range was faster to converge, but it limits the optimization search space and is more likely to converge to a local optimum. In contrast, the bigger range allows a larger search space which improves the changes to avoid local optimums. However, it is slower to converge, and will cause oscillations in the axis during the tuning process due to unsuitable controller gains. Optimization of the initial ranges remains as a future work.

Table 4.1 *Initial parameter values for automatic tuning with LJ.*

Parameter	Initial value	Initial range size
Position K_p	x_1 [1]	$[0.75x_1; 1.25x_1]$
Position FF	100.0	$[99.0; 101.0]$
Velocity K_p	x_2 [1]	$[0.75x_2; 1.25x_2]$
Velocity K_i	x_3 [1]	$[0.75x_3; 1.25x_3]$
Velocity K_d	0	$[0; 50]$
Velocity FF	0	$[0; 20]$
Acceleration FF	0	$[0; 20]$

The minimum range size is set to 2 to allow small modifications also in last iterations of optimization as the values are handled as integers. Initial values and ranges are collected to Table 4.1.

Luus-Jaakola optimization method reduces the parameter ranges in every iteration with a predefined multiplier. In this work, the value 0.75 was selected to decrease the

[1] Initial value from tuning created with Ziegler-Nichols-method.

parameter ranges fast to achieve reasonable run times in tests. With 20 iterations, the last iteration's region size is 3.17 % of the original range.

The test setup presented in Section 4.1 is used to collect the data and evaluate tune performance during the optimization process. Axis movement is alternated between positive and negative directions during testing to acquire data from the both moving directions. Other benefit from altering movement directions is speeding up the tuning process, since the axis is not moved to zero position after every measurement.

4.3 Results

Measurements were carried out to compare different tuning methods. Hypothesis was that automatic tuning results in more predictable controller performance than manual tuning, and also better absolute performance.

4.3.1 Manual tuning

Manual tuning performance results are required to evaluate goodness of automatic tuning algorithm. For comparison between performances achieved with the manual and automatic tuning, five experts at OptoFidelity were requested to create a tune for the axis presented in Section 3.1.1. Tuning was completed three times with each test subject: one without added weight, one with the plate 1 and one with the plate 2. The testing plan is in Appendix B. Before the experiments, the axis parameters were initialized to the default values that are able to move the axis. Default values were searched by rising the position controller P-gain and velocity controller P- and I-gains gradually, which is a very fast method to find approximate values, but will result in poor performance. Initial values of the parameters are listed in Table 4.2.

Table 4.2 *Initial controller parameter values for the manual tuning experiments.*

Parameter	Initial value
Position K_p	100
Position FF	100
Velocity K_p	100
Velocity K_i	100
Velocity K_d	0
Velocity FF	0
Acceleration FF	0

Manual tuning was performed with four parameters that are tuned in commercial robots of OptoFidelity. From position controller, the proportional gain and feed-forward parameters were used. The velocity controller proportional and integral

gains were the other two parameters. Rest of the parameters were left to their initial values (0) where they do not affect the performance.

It took from half an hour to one and half hours for one subject to tune the axis controller with each of the three weights. The first tuning took the longest time because the axis was new for the subjects. Subjects were allowed to use the previous tune as the basis for the next tunes, which decreased the system tuning time in the next rounds. Subjects' comments were recorded during the test. One comment describes industrial controller manual tuning process:

> *"There are no processes for this [tuning]. Or there is, but who has time to learn them."*

That comment strengthened the hypothesis that automatic tuning is beneficial.

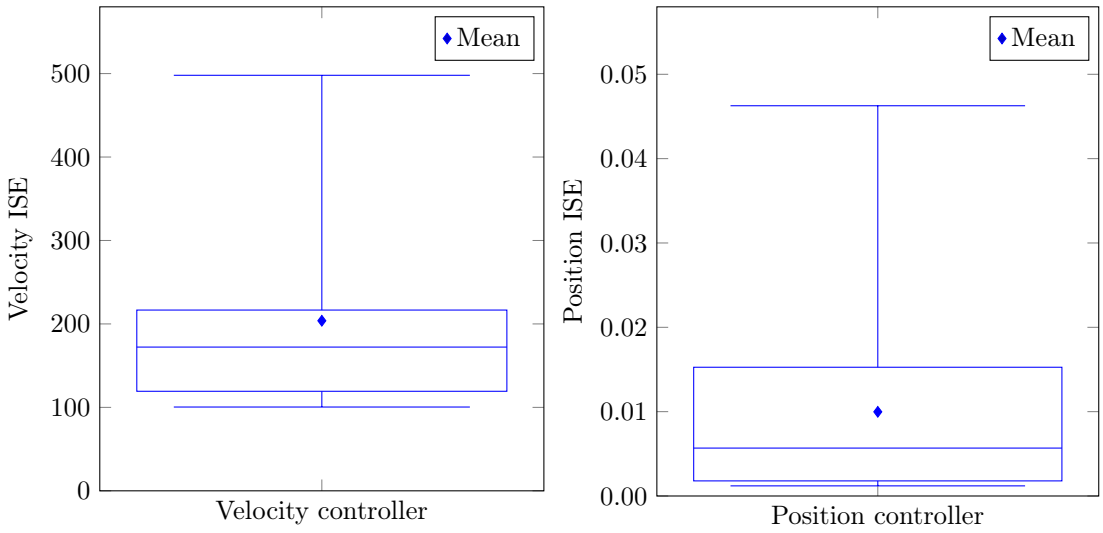


Figure 4.5 *Box plots of manually tuned controllers' performances.*

The result tuning parameters from all experiments can be found from measurement log in Appendix C. Summary of results is presented in Figure 4.5 as boxplot.

4.3.2 Automatic tuning

The implemented autotuning system was tested with the same axis configurations as manual tuning. Axis controllers were tuned ten times with each of the mass configurations, totaling 30 test runs. Running one tune test movement, including setting new parameters, running the test movement, collecting data and analysing it, took less than one second. With the selected LJ-population size of 20 and iteration count of 20, one tuning run lasts approximately $20 \times 20 \times 1\text{s} = 400\text{s}$. As there are three tuning steps, one full controller tuning sequence takes total of $400\text{s} \times 3 = 1200\text{s} = 20$ minutes.

It is not guaranteed that the last iteration would have the best performing individual tune, because in LJ optimization the best performing tune is not stored for next generation. Therefore tune that resulted in the best performance during the whole tuning process is considered as the result of tuning round, not the best tune of the very last generation.

Velocity controller tuning

Velocity controller is the inner one in the cascade controller and therefore it is tuned first. Velocity ISE was used as the fitness function for this LJ-optimization step. The position controller was tuned with ZN-method once with each configuration, and the same position controller tune was used in the mass plate configuration during velocity controller tuning. The optimization path from one velocity controller tuning run is presented in Figure 4.6. Y-axis scale is velocity controller ISE-performance index

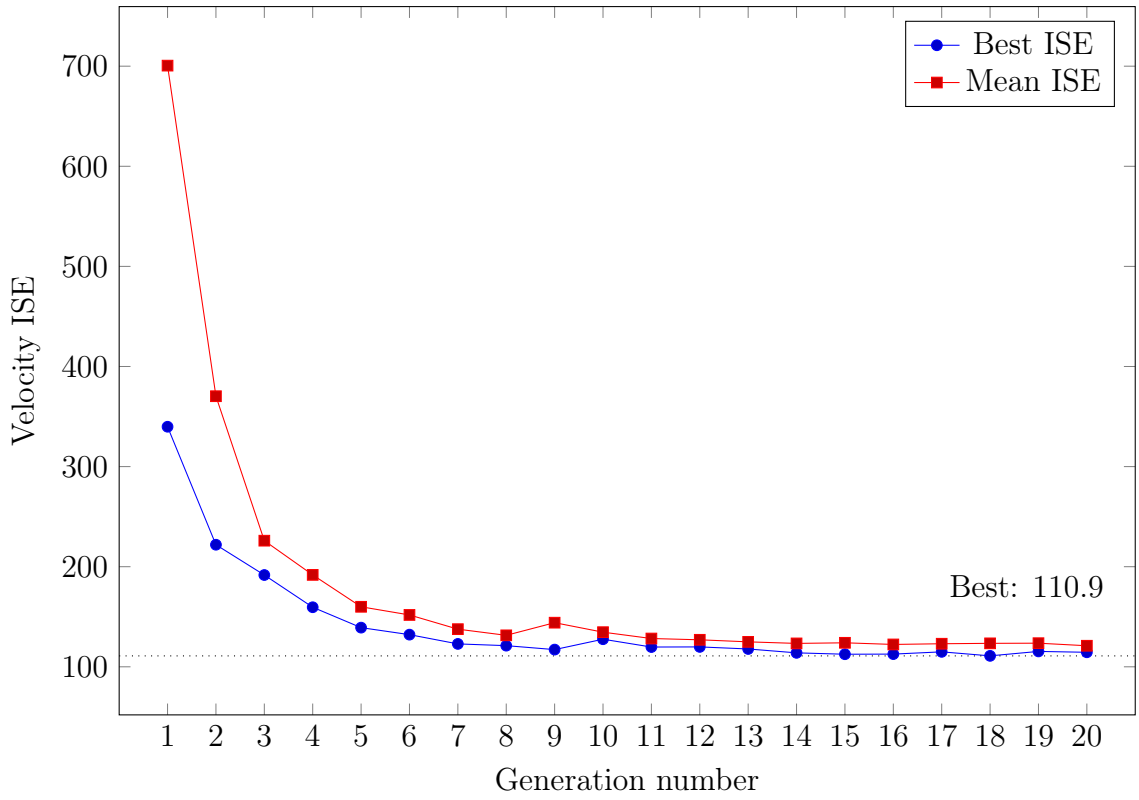


Figure 4.6 *Velocity controller performance over the generations.*

and X-axis consists of 20 tuning generations. Best velocity controller performance of each generation is plotted along with generation's mean performance. Smaller ISE-value is better and the figure shows that the performance is getting better during the tuning iterations. The best performance values of generations are converging to optimum around the fifteenth generation. Mean performance of the each generation

also approaches the best performance. From the data it is clear that 20 generations is enough to optimize this system.

The resulting velocity controller tuning parameters and ISE-performance index values are in Appendix D. Because the position controller was not tuned, only the velocity controller performances were measured. Measurements are divided to three tables based on mass plate configuration: D.1, D.2, and D.3. It should be noted that the velocity controller ISE values were measured with only one axis movement with corresponding controller tune. Tune producing the best performance is selected for the next tuning steps.

Position controller tuning

Position controller is the outer loop of the servo drive cascade controller and is thus tuned after the velocity controller. When the inner controller performs optimally, it produces the minimum amount of disturbance and error to outer controller. Therefore the starting point for position controller optimization is the ZN-tuned position controller and LJ-tuned velocity controller. Position ISE was used as the fitness function. The optimization path from one position controller tuning run is presented in Figure 4.7. Y-axis is position controller ISE-performance and X-axis is

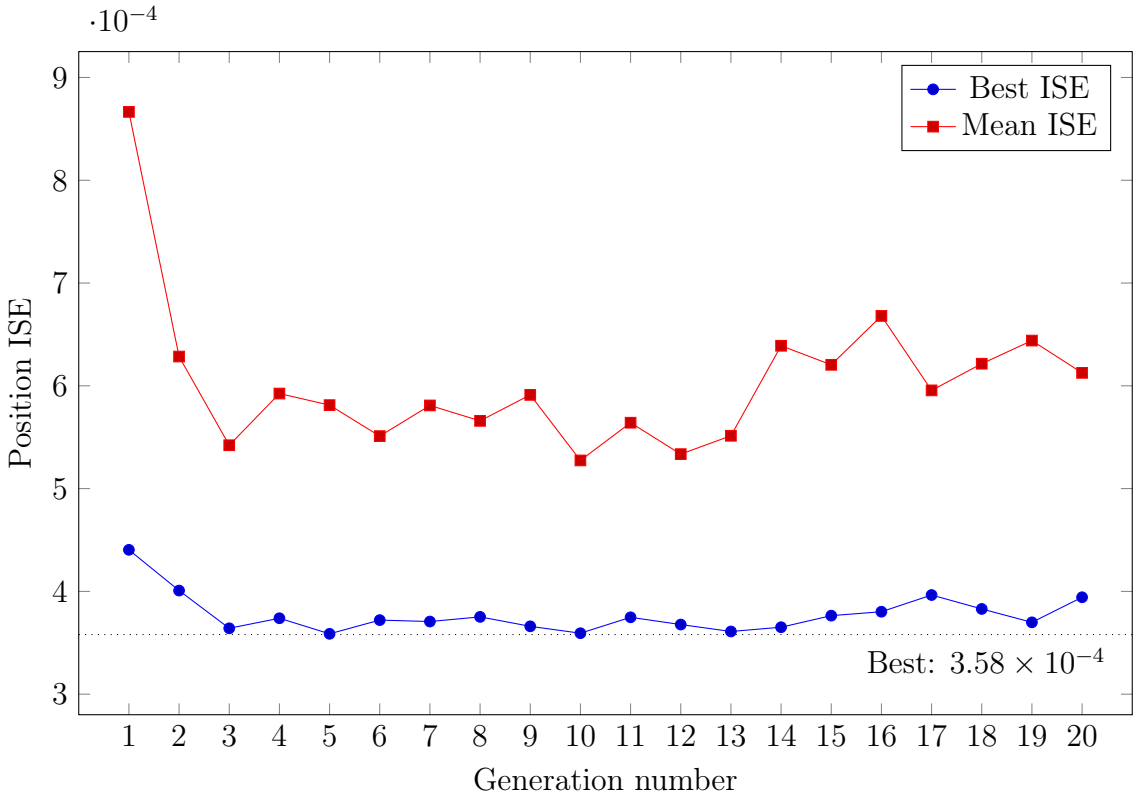


Figure 4.7 *Position controller performance over the generations.*

the generation index of the optimization process. Figure 4.7 shows that only a few generations from the beginning of the LJ-optimization improved the performance.

The position controller tunes and performance index values are in Appendix D. They are divided to Tables D.4, D.5, and D.6 based on mass configurations.

Fine tuning of the both controllers

After the both controllers were tuned individually, they were fine tuned together. This was done because velocity controller gets its setpoint from the suboptimally tuned position controller during the first tuning round. Then, the velocity controller parameters change, also the position controller should be re-tuned for optimal performance. Therefore it is reasonable to tune the controllers together.

Since the system output is the axis position, the position ISE is used as the performance index in fine tuning process. Optimization path of one fine tuning is presented in Figure 4.8. In this case, the best tune was found from the fourteenth generation, while the generation mean ISE optimum location was achieved in twelfth generation. Again, 20 generations proved to be enough to converge with the implemented method.

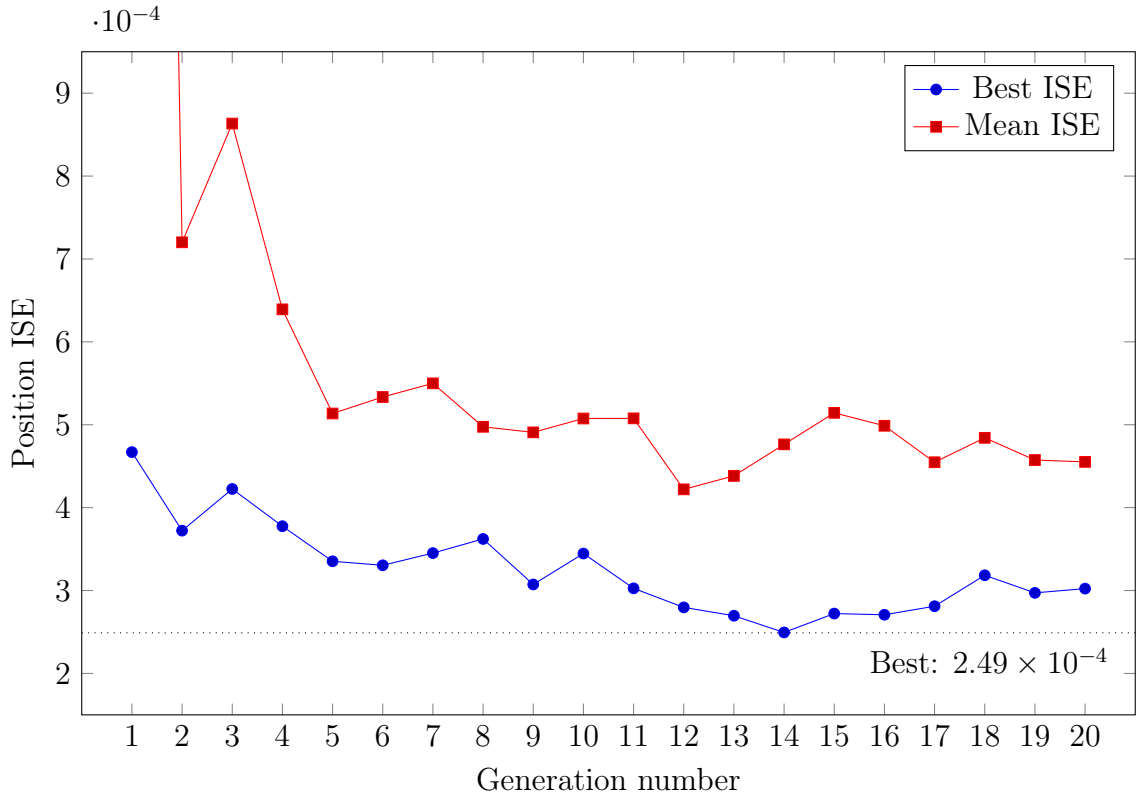


Figure 4.8 *Position controller performance during Luus-Jaakola tuning of the both controllers.*

The resulting tunes and position and velocity ISE performance values are in Appendix D in Tables D.7, D.8, and D.9.

Fine tuning with the modified fitness function

Even without further analysis, it is clear from the final tuning performance that velocity controller performance was significantly decreased after the position controller tuning and fine tuning of the both controllers. Because only the position ISE was used as perfromance index in the position based optimizations, the velocity controller performance can decrease as long as it does not harm the position controller performance. It is also possible that the position controller would compensate errors in the velocity controller, which can lead to decent position controller performance but decreased velocity controller performance. In that case there is a risk that with a different trajectory, the controllers would not work together as desired. Therefore it is reasonable to use both the position and velocity ISE values when fine tuning the controllers.

Different controller ISE values should be scaled to the same magnitude to utilize both of them in the optimization process. Currently, position ISE values are in magnitude 10^{-4} and velocity ISE values in magnitude 10^{2}. If they are summed without scaling, it is clear that the velocity ISE would lead the optimization and position ISE would have effect close to nothing. To get the magnitudes to the same scale, position ISE values were multiplied by 10^{6} during the optimization. The new ISE value for fine tuning fitness function is in Equation 4.2.

$$ISE_{combined} = \lambda \times ISE_{vel} + (1 - \lambda) \times 10^{6} \times ISE_{pos}, \tag{4.2}$$

where λ is a hype parameter from the range $(0, 1)$. When $\lambda = 1.0$ only velocity ISE is used. Respectively, when $\lambda = 0.0$, the optimization is based on position ISE. Relative weights of position and velocity ISE are application dependent. Initial value $\lambda = 0.5$ was selected for this work. In future applications, it should be adjusted based on the application requirements.

One optimization path with new fitness function that considers both controllers' performances is presented in Figure 4.9. The best tune was found after thirteen generations.

Tuning with modified fitness function was executed three times for each mass plate configuration. The tuning parameters and corresponding performance results are in Appendix D in Tables D.10, D.11, and D.12.

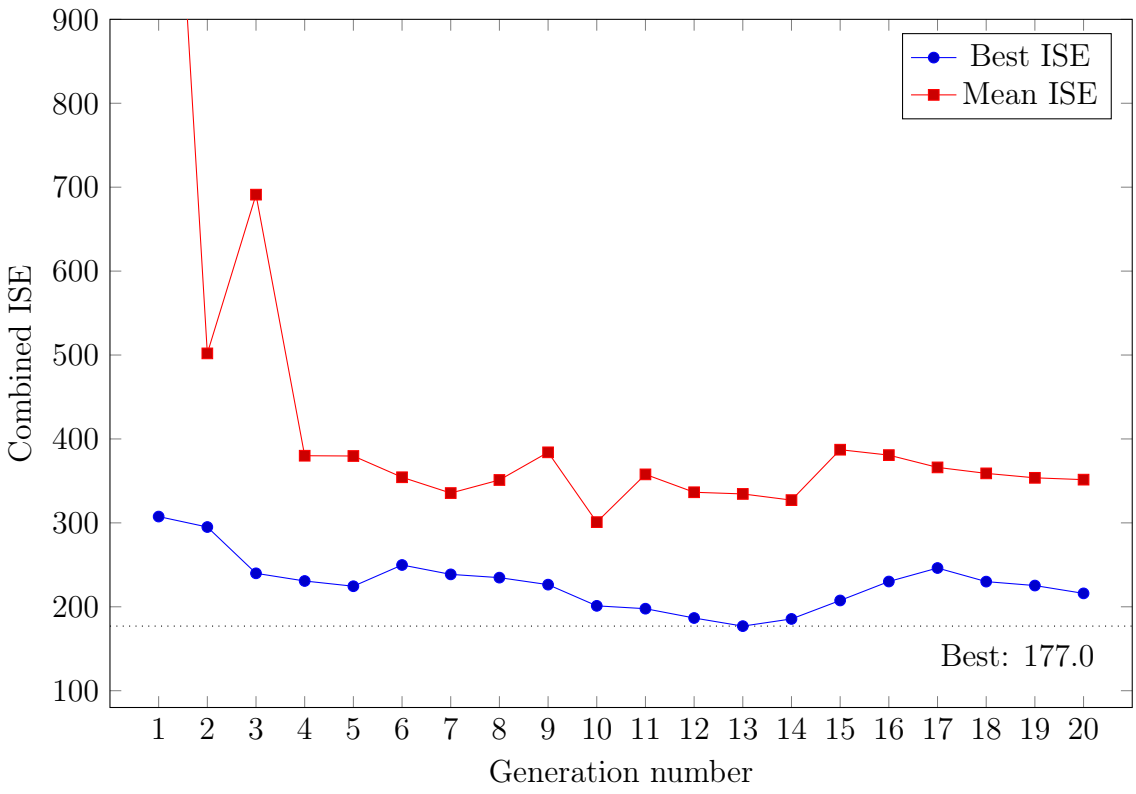


Figure 4.9 *Combined controller performance during Luus-Jaakola tuning of both controllers,* $\lambda = 0.5$.

4.3.3 Optimized ZN tuning

During the automatic tuning system development it was noted that the automated tuning with the population based parameter optimization method is too slow with real hardware. It was quickly noticed that when one tune evaluation takes one second, the iteration count and population size are very limited. When the measurements for the thesis of Juha Koljonen [15] were executed parallel with this thesis, they were found to be significantly faster to execute. Therefore, the model based tuning method was adjusted to produce the same tunes and performance than LJ-method. The new method for adjusting model based tuning methods was named Koljonen-Heinänen (KH) method.

At the time of developing this sequence, Adaptive Genetic Algorithm was used for controller tuning. However, most important is that the coefficients used by the model based PID controller tuning are adjusted based on the iterative method results, regardless of the iterative and model based methods. The flowchart of adjusting model based tuning method with KH method is in Figure 4.10.

At first, the test system is tuned once with iterative method. The result is used to adjust the parameters of the model based tuning method. In case of ZN-method,

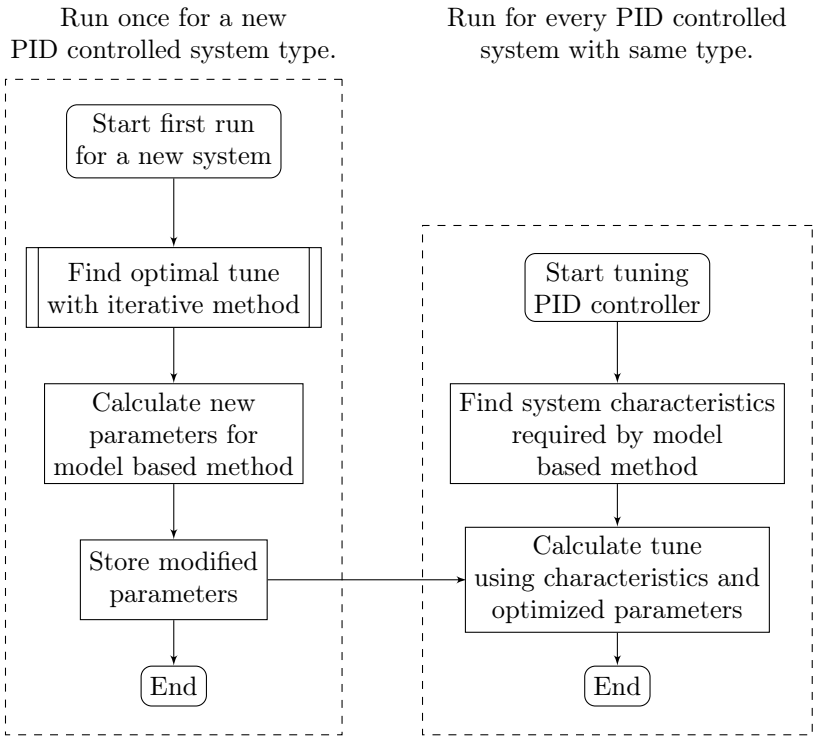


Figure 4.10 *KH method flowchart.*

modified parameters are coefficients of controller's critical P-gain and critical oscillation's period (Table 2.2). In short, the KH method adjusts model-based tuning methods to produce as good performance results as the optimization based methods. Novelty research has been done and patenting process is ongoing for the KH-method. Controller tuning and performance measurements with the new ZN-KH variant were conducted with the same test system in [15] and used in the tuning method comparisons.

4.4 Result analysis

The tuning performances were compared by calculating the mean values (μ) and standard deviations (σ). The mean value of performance index describes the expected performance of system when it is tuned with a specific method. On the other hand, the mean performance is not sufficient to represent the goodness of tuning method as the same average performance can be achieved with very different performance distributions. The population standard deviations were calculated to describe the repeatability of the tuning method. The lower the standard deviation, the more similar performance the tuning method produces in consecutive tuning iterations. It should be noted that as the velocity and position units are different, the ISE performance values of the controllers are not comparable to each others.

Only the final tune performance is measured in the case of manual tuning. Automatic LJ based tuning was completed in three stages and performances are presented after each stage. The most significant result is the controller performance after all three tuning stages. It is compared to results from manual tuning tests.

Two-sample Kolmogorov-Smirnov test (K-S test) [11] is utilized to deteremine the statistical significance of performance differences. It is a statistical method for testing if two empirical samples are drawn from the same distribution. The test compares sample cumulative distribution functions and expects the distributions to be continuous. The test is nonparametric and it considers both location and shape of the cumulative distribution function curves. Also, K-S test does not require test samples to be normally distributed. Therefore it is suitable for the experiments of this work.

4.4.1 Manual tuning

Statistics in Table 4.3 are calculated based on the manual tuning data in Appendix C.

Table 4.3 *Manual tuning performance average and standard deviation.*

Mass configuration	$\mu_{\text{Pos ISE}}$ $(\times 10^{-4})$	$\sigma_{\text{Pos ISE}}$ $(\times 10^{-4})$	$\mu_{\text{Vel ISE}}$	$\sigma_{\text{Vel ISE}}$
No mass plate	149.28	166.28	200.8	130.7
Mass plate 1	79.72	73.14	215.7	146.6
Mass plate 2	**70.36**	**67.93**	**194.9**	**59.6**
All	99.79	117.38	203.8	118.8

Manual tuning performances are the base values that automatic tuning performances are compared to. As can be seen in Table 4.3, the standard deviations are in the same magnitude as mean values. An ideal controller would have ISE values of zero and negative values are not possible, which means that performance index is distributed more to above the average of the data. For velocity performances standard deviations are about half of the mean values. While velocity controller performance remains almost the same regardless of the added mass plate, position controller performance gets better along the added mass. It indicates that the system with added mass is less sensitive, which makes it easier to tune manually.

4.4.2 Automatic tuning

To review the full automatic tuning process, the resulting tune performances from all three stages are presented in the tuning order. Velocity controller tuning results

are presented first, followed by the position controller tuning results, and finally the fine tuned controller performances.

Velocity controller performance

Statistics in Table 4.4 are calculated based on velocity controller tuning data in Appendix D.

Table 4.4 *Velocity controller performance. Tuning stage 1.*

Mass configuration	$\mu_{\text{Vel ISE}}$	$\sigma_{\text{Vel ISE}}$
No mass plate	137.7	15.1
Mass plate 1	**109.9**	**4.0**
Mass plate 2	113.1	9.2
All	120.2	16.3

After autotuning the velcoity controller, it can be seen that the controller's performance is better than manually tuned velocity controller. Average performance index value decreases to half of the manually tuned. In addition, the standard deviations decreased significantly, as it is about one seventh of manually tuned controller's. Best performance was achieved with mass plate 1. With that configuration the standard deviation was below 3 % of the corresponding manual tuning value.

Position controller performance

The statistics of the resulting position controller performance are collected to Table 4.5 from position controller tuning data in Appendix D.

Table 4.5 *Position controller tuning performance. Tuning stage 2.*

Mass configuration	$\mu_{\text{Pos ISE}}$ $(\times 10^{-4})$	$\sigma_{\text{Pos ISE}}$ $(\times 10^{-4})$
No mass plate	4.62	0.81
Mass plate 1	**3.58**	**0.39**
Mass plate 2	3.75	0.48
All	3.99	0.75

Position ISE, the system final performance value, is significantly lower than manual tuning performances in Table 4.3. The mean value is under one tenth, and the standard deviation is 100 times smaller than corresponding manual tuning values.

Fine tuned controllers' performance

The statistics of final LJ-tuning performance are calculated based on the fine tuning data in Appendix D and presented in Table 4.6.

Table 4.6 *Means and standard deviations of LJ-tuned controllers. Tuning stage 3.*

Configuration	$\mu_{\text{Pos ISE}}$ ($\times 10^{-4}$)	$\sigma_{\text{Pos ISE}}$ ($\times 10^{-4}$)	$\mu_{\text{Vel ISE}}$	$\sigma_{\text{Vel ISE}}$
No mass plate	5.05	0.93	360.5	103.3
Mass plate 1	**3.55**	**0.57**	245.0	**52.0**
Mass plate 2	3.37	0.77	**234.5**	108.8
All	3.99	1.08	280.0	108.0

It is clear that the worst performance values were again measured without the additional mass. In contrast, the measurements with mass plate 1 resulted in same or better performance and smaller variation than other mass configurations. After the last tuning stage, the performance of the position controller is significantly better than the results from manual tuning. The velocity controller performance is worse than after the first tuning step or manual tuning as the mean ISE values are up to 2.6 times higher. Standard deviations of the velocity controller performances are in the range of standard deviations of the manual tuning results.

Modified performance index fine tuned controllers' performance

Statistics from third tuning stage ran with the combined ISE fitness function are calculated from data in Appendix D and collected to Table 4.7.

Table 4.7 *Means and standard deviations of LJ-tuned controllers. Combined ISE performance index was used in last tuning step.*

Configuration	$\mu_{\text{Pos ISE}}$ ($\times 10^{-4}$)	$\sigma_{\text{Pos ISE}}$ ($\times 10^{-4}$)	$\mu_{\text{Vel ISE}}$	$\sigma_{\text{Vel ISE}}$
No mass plate	4.69	0.89	255.6	11.3
Mass plate 1	**2.98**	**0.17**	178.7	**11.1**
Mass plate 2	3.24	0.33	**147.4**	19.6
All	3.63	0.93	193.9	47.7

There are slight improvements in all of the final tuning results when the combined ISE was used. The standard deviations of velocity controller performances of individual mass configurations decreased 80-90 %. Also, the means and standard deviations from all measurements combined are decreased. That indicates more repeatable tuning with better performance results.

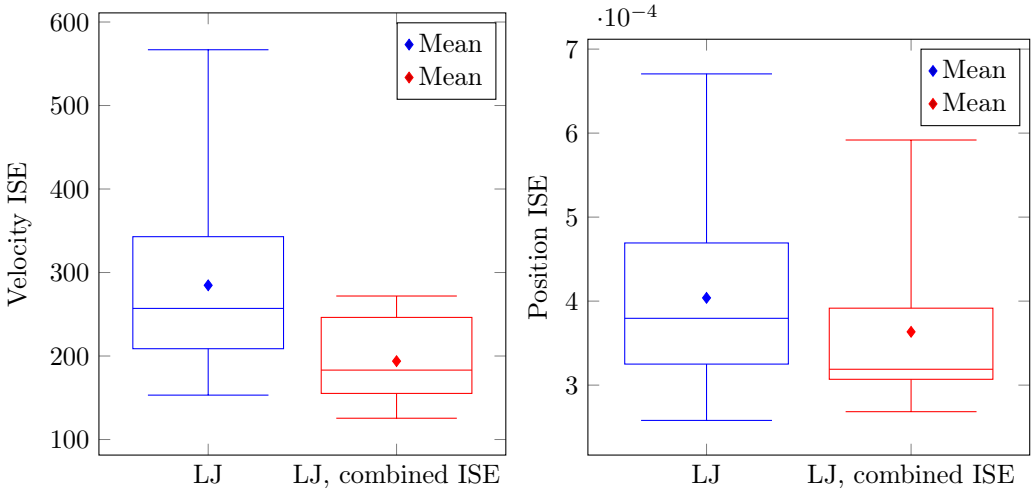
4.4.3 Performance comparison

Performances of the position and velocity controllers achieved with different methods are compared next. First, the results from Luus-Jaakola optimization with different fitness functions are compared to determine which one is the most suitable for this application. After that the best results from automatic tuning are compared to manual tuning and KH-modified ZN-tuning in [15].

From the presented statistical values it is seen that adding mass to the axis lowers ISE values of the both controllers. However, results with mass plate 1 are generally better than results with mass plate 2. There is no direct correlation between the additional mass and controller performance. Therefore the measurement results from all mass configurations are combined together and comparison between methods is made with the overall mean and standard deviations. Performance measurements are compared with two-sample Kolmogorov-Smirnov test that does not assume normally distributed samples.

Luus-Jaakola tuning comparison

The main concern in combining the ISE values for the final tuning step was that velcoity ISE could decrease the position ISE. The box plots of Luus-Jaakola tuned controller performances are in Figures 4.11a and 4.11b.



(a) *Velocity controller performance.* (b) *Position controller performance.*

Figure 4.11 *Box plots of the controllers fine tuned with the position ISE and the combined ISE.*

All means, medians and standard deviations of performances are smaller from the controllers that are tuned with the combined ISE. As position is the final output of

the axis, the hypothesis was that completing final tuning with position ISE would give the best results. The results shows otherwise. When the combined ISE is used the resulting velocity controller performance standard deviation decreased 56 % and mean value 31 % compared to tuning with only the position ISE value.Also, the best individual results for velocity controller were achieved by utilizing combined ISE. The effect is not as strong with the position controller, where mean decreased 9 % and standard deviation 13 %. Position controller best results were measured when only the position ISE was used as expected. Contrary to intuition, using also the velocity ISE in the final tuning stage reduced also standard deviation of position controller ISE. It shows that when the velocity controller is more accurate also the position controller performs better. Robustness of a tuning method is in this case preferred over the occasional top results. Therefore the results from LJ with combined ISE are selected for comparison between the manual and automatic tuning.

Automatic and manual tuning comparison

Two types of automatic tuning tests were conducted and the results are compared to manual tuning. ZN-KH tuning outperformed other model based tuning methods in [15] and therefore it is the only model based tuning method in comparison. The results from the manual tuning, ZN-KH and LJ with combined ISE are collected to Table 4.8 for easier comparison.

Table 4.8 *Final comparison. Means and standard deviations of automatic and manual tuning methods.*

Tuning method	$\mu_{\text{Pos ISE}}$ $(\times 10^{-4})$	$\sigma_{\text{Pos ISE}}$ $(\times 10^{-4})$	$\mu_{\text{Vel ISE}}$	$\sigma_{\text{Vel ISE}}$
Manual tuning	99.79	117.38	203.8	118.8
ZN-KH	10.90	2.02	**164.5**	**16.8**
LJ combined ISE	**3.63**	**0.93**	193.9	47.7

The data shows that the automatic tuning brings significant increase to the position controller performance. The mean position ISE with ZN-KH tuning is only 11 % of manual tuning, so the position ISE value has decreased by almost order of magnitude. With LJ tuning, the position ISE decreased to one third of the ZN-KH result. Same effect is seen in the standard deviations, where ZN-KH tuning position performance's standard deviation is 1.7 % of the manual tuning. LJ tuning position ISE standard deviation is 46 % of ZN-KH value, which means over two orders of magnitude difference to the manual tuning value.

Velocity controller performance does not improve as much as the position controller. ZN-KH mean ISE decreased by one fifth from manual tuning . LJ tuning improved manual tuning by 5 %. Significance of this change is calculated below. The standard

deviation of velocity ISE from ZN-KH tuning is only 14 % of manual tuning performance standard deviation, which means more repeatable tuning of the velocity controller. With LJ velocity controller tuning, the standard deviation is 60 % smaller than with manual tuning. ZN-KH method is therefore more repeatable for tuning velocity controller, as the standard deviation is only one third of value from LJ tuning.

Two sided two-sample Kolmogorov-Smirnov test is used to check if the performance values from different tuning methods are from the same or different distributions. The null hypothesis is that the both samples are drawn from the same distribution. The probabilities of the results being measured from the same distribution are collected to Table 4.9.

Table 4.9 *Kolmogorov-Smirnov test probabilities that two tuning performance results are drawn from same distribution.*

Tuning methods	Position controller [%]	Velocity controller [%]
Manual vs. ZN-KH	1.5×10^{-4}	5.9
Manual vs. LJ combined ISE	0.1×10^{-4}	30.8
LJ combined ISE vs. ZN-KH	5.6×10^{-8}	5.9

Probabilities of position controllers are very low for all tuning combinations, so all null hypotheses can be rejected. Therefore all three results are drawn from different distributions. The velocity controller case is not as clear as the position controller, as the probabilities are quite high. There is 5.9 % chance that manual tuning results and ZN-KH results are from the same distribution. Same probability is for LJ and ZN-KH results to be drawn from same performance distribution. With probability of 30.8 % it is quite likely that the LJ and manual tuning are equally good for velocity controller tuning. With the current information, the null hypothesis cannot be rejected and velocity controller performance is considered to be the same for all tuning methods.

4.5 Discussion

Based on the work results, test system can be tuned with iterative, learning based optimization methods. In addition to ideal PID controller parameters, also other test system specific controller parameters were successfully optimized. Considering the manual tuning measurements, there is a clear need for autotuning system for OptoDrive servo controller. Luus-Jaakola method selected from literature was proved to be suitable for autotuning both the velocity and position controllers. LJ method was able to produce tunes with consistent performance robustly in constant time. Automatic tuning adjusted controller parameters for different axis dynamics

successfully. The developed tuning system has potential to reduce the resource needs of axis tuning significantly. In the best case OptoFidelity will save worldwide travelling by automating the tuning sequence instead of sending an tuning expert to the customer premises to adjust the controller parameters.

The performance of tuning method surpassed the expectations by decreasing the mean position controller ISE by 96 % and standard deviation over 99 %. The velocity controller performance did not significantly change according to Kolmogorov-Smirnov test. In addition to implementing and testing the LJ autotuning method also a novel method for adjusting the model based tuning method to produce similar performance results was developed in cooperation with Juha Koljonen. The position controller tuned with ZN-KH method had slightly worse performance than the purely LJ tuned controller. Tuning time however is dropped significantly as ZN tuning requires only searching critical gain of the controller once. Also, model based ZN tuning cannot utilize all parameters of the controller which is assumably one reason for it to produce lower performance than the LJ method. The velocity controller performance achieved with the ZN-KH was actually better than performance of the LJ tuned controller. However, according to Kolmogorov-Smirnov test there is about 6 % chance that the performance distributions are not different, so superiority of ZN-KH method cannot be guaranteed.

In conclusion, ZN-KH method is the recommended method for OptoFidelity robot axis tuning when considering large amounts of robots. It is a fast tuning method and can produce consistent tunes whose performance is better than manual. For a small amount of robots, LJ method with final tuning completed with combined position and velocity ISE values is recommended, as it can utilize all controller parameters and therefore produced slightly better axis performance. LJ tuning sequence will take longer time than the model based tuning, and as it is searching the very optimum of the axis tune, optimization process is likely to cause some axis oscillations during the process.

The future work should be concentrated firstly on the fitness function of the optimization process. The selected ISE performance index is widely used in PID controller performance measurements, but it is not necessary the ultimately best fitness function for the servo drive case. As noticed in this work, combining different fitness functions can provide better results than utilizing single performance measurement. To find the optimal fitness function the desired and undesired properties of axis behaviour should be investigated more closely. For example, to prevent oscillations on undesired frequencies, the fitness function could utilize a frequency representation of the trajectory error. After finding the important properties they should be weighted so that the fitness function would be a compromise between them for each application. In addition to the fitness function, the current LJ method implementation could

be optimized by modifying the iteration count, population size and initial ranges. Alternatively, additional optimization methods such as Simulated Annealing and Differential Evolution could be used.

5. CONCLUSION

The targets of this thesis were to study nonlinear, discrete, learning based optimization methods for PID controller tuning to OptoFidelity OptoDrive. To provide a baseline for controller performance measurements, manual tuning tests were conducted. After selecting the best suitable method from the surveyed optimization methods and performance indices, the tuning and performance measurements were experimentally compared to manual tuning. The results showed that the automatic tuning with Luus-Jaakola method produces not only significantly better tunes for controllers, but also more repeatably and in the same or even less time than manual tuning. Expectations for model free autotuning method were surpassed and optimization search based tuning of OptoDrive controllers is the recommended option for individual robots. The results of the learning based methods were used to adjust the traditional model based Ziegler-Nichols method to produce nearly similar performance results. New, modified ZN tuning method is recommended to be used in a mass production where tuning time is one key resource. The method for adjusting the model based methods for a new controller is in patenting process.

The new OptoDrive auto-tuning system will be important part of the company's toolbox as it frees the resources from tuning to more important tasks. While the system is still under development and it still requires integration work to be user-friendly automatic tuning software, the tuning logic is implemented and tested to replace manual tuning. In addition to increased performance and less time in robot tuning at OptoFidelity, the optimization can be used in marketing of the products. These features will give OptoFidelity competitive advantage in the field of test robotics.

REFERENCES

[1] Bambang Argo et al. "Optimization of PID Controller Parameters on Flow Rate Control System Using Multiple Effect Evaporator Particle Swarm Optimization". In: *International Journal on Advanced Science, Engineering and Information Technology* 5 (Apr. 2015), p. 62.

[2] K. Åström and T. Hägglund. *PID Controllers: Theory, Design, and Tuning*. Vol. 2. Instrument society of America, Research Triangle Park, NC 27709, 1995. ISBN: 1-55617-516-7.

[3] K.J. Åström and T. Hägglund. "The future of PID control". In: *Control Engineering Practice* 9.11 (2001), pp. 1163–1175. ISSN: 0967-0661. DOI: https://doi.org/10.1016/S0967-0661(01)00062-4. URL: http://www.sciencedirect.com/science/article/pii/S0967066101000624.

[4] James Bergstra and Yoshua Bengio. "Random Search for Hyper-parameter Optimization". In: *Journal of Machine Learning Research* 13 (Feb. 2012), pp. 281–305. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2188385.2188395.

[5] Zafer Bingul. "Adaptive genetic algorithms applied to dynamic multiobjective problems". In: *Applied Soft Computing* 7.3 (2007), pp. 791–799. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2006.03.001. URL: http://www.sciencedirect.com/science/article/pii/S1568494606000251.

[6] Terrence L. Blevins. "PID Advances in Industrial Control". In: *International Federation of Automatic Control Proceedings Volumes* 45.3 (2012), pp. 23–28. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20120328-3-IT-3014.00004. URL: http://www.sciencedirect.com/science/article/pii/S1474667016309946.

[7] M. Dorigo and L. M. Gambardella. "Ant colony system: a cooperative learning approach to the traveling salesman problem". In: *IEEE Transactions on Evolutionary Computation* 1.1 (Apr. 1997), pp. 53–66. ISSN: 1089-778X. DOI: 10.1109/4235.585892.

[8] Gregory Hornby et al. "Automated Antenna Design with Evolutionary Algorithms". In: *Collection of Technical Papers - Space 2006 Conference* 1 (Aug. 2006). DOI: 10.2514/6.2006-7242. URL: https://doi.org/10.2514/6.2006-7242.

[9] Arturo Y. Jaen-Cuellar et al. "PID-Controller Tuning Optimization with Genetic Algorithms in Servo Systems". In: *International Journal of Advanced Robotic Systems* 10.9 (2013), p. 324. DOI: 10.5772/56697. eprint: https://doi.org/10.5772/56697. URL: https://doi.org/10.5772/56697.

[10] Reza N. Jazar. *Theory of Applied Robotics*. 2nd. Springer New York Dordrecht Heidelberg London, 2010. ISBN: 978-1-4419-1749-2.

[11] Frank J. Massey Jr. "The Kolmogorov-Smirnov Test for Goodness of Fit". In: *Journal of the American Statistical Association* 46.253 (1951), pp. 68–78. DOI: 10.1080/01621459.1951.10500769. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1951.10500769.

[12] Dervis Karaboga. "An Idea Based on Honey Bee Swarm for Numerical Optimization, Technical Report - TR06". In: *Technical Report, Erciyes University* (Jan. 2005).

[13] J. Kennedy and R. Eberhart. "Particle swarm optimization". In: *Neural Networks, 1995. Proceedings., IEEE International Conference on*. Vol. 4. Nov. 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[14] Serkan Kiranyaz, Moncef Gabbouj, and Turker Ince. *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*. English. Adaptation, Learning, and Optimization. Springer, 2014. ISBN: 978-3-642-37845-4. DOI: 10.1007/978-3-642-37846-1.

[15] Juha Koljonen. *Automatic model-based PID tuning of a servo axis*. Tampere University of Technology, Aug. 2018.

[16] Oliver Kramer. *Genetic Algorithm Essentials*. Vol. 679. Studies in Computational Intelligence. Springer International Publishing, 2017. ISBN: 978-3-319-52156-5. URL: https://link-springer-com.libproxy.tut.fi/book/10.1007/978-3-319-52156-5.

[17] R. A. Krohling and J. P. Rey. "Design of optimal disturbance rejection PID controllers using genetic algorithms". In: *IEEE Transactions on Evolutionary Computation* 5.1 (Feb. 2001), pp. 78–82. ISSN: 1089-778X. DOI: 10.1109/4235.910467. URL: https://ieeexplore.ieee.org/document/910467/.

[18] G. Lin and G. Liu. "Tuning PID controller using adaptive genetic algorithms". In: *2010 5th International Conference on Computer Science Education*. Aug. 2010, pp. 519–523. DOI: 10.1109/ICCSE.2010.5593559.

[19] Rein Luus. "Optimization in model reduction". In: *International Journal of Control* 32.5 (1980), pp. 741–747. DOI: 10.1080/00207178008922887. eprint: https://doi.org/10.1080/00207178008922887. URL: https://doi.org/10.1080/00207178008922887.

[20] Rein Luus and T H. I. Jaakola. "Optimization by Direct Search and Systematic Reduction in the Size of Search Region". In: *American Institute of Chemical Engineers Journal* 19 (July 1973), pp. 760–766.

[21] S.M. Al-Marzoug and R.J.W. Hodgson. "Luus–Jaakola optimization procedure for multilayer optical coatings". In: *Optics Communications* 265.1 (Sept. 2006), pp. 234–240. ISSN: 0030-4018. DOI: https://doi.org/10.1016/j.optcom.2006.03.039. URL: http://www.sciencedirect.com/science/article/pii/S0030401806002756.

[22] Aidan O'Dwyer. "A Summary of PI and PID Controller Tuning Rules for Processes with Time Delay. Part 1: PI Controller Tuning Rules". In: *International Federation of Automatic Control Proceedings Volumes* 33.4 (2000), pp. 159–164. ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)38237-X. URL: http://www.sciencedirect.com/science/article/pii/S147466701738237X.

[23] N. A. Patil and G. V. Lakhekar. "Design of PID controller for cascade control process using genetic algorithm". In: *International Conference on Intelligent Computing and Control Systems*. June 2017, pp. 1089–1095. DOI: 10.1109/ICCONS.2017.8250634. URL: https://ieeexplore.ieee.org/document/8250634/.

[24] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[25] K. T. Prasad and Y. V. Hote. "Optimal PID controller for Ball and Beam system". In: *International Conference on Recent Advances and Innovations in Engineering*. May 2014, pp. 1–5. DOI: 10.1109/ICRAIE.2014.6909125. URL: https://ieeexplore.ieee.org/document/6909125/.

[26] N. S. Rathore, V. P. Singh, and D. P. S. Chauhan. "ISE based PID controller tuning for position control of DC servo-motor using LJ". In: *2015 International Conference on Signal Processing, Computing and Control (ISPCC)*. Sept. 2015, pp. 125–128. DOI: 10.1109/ISPCC.2015.7375010.

[27] J. M. S. Ribeiro et al. "Comparison of PID controller tuning methods: analytical/classical techniques versus optimization algorithms". In: *2017 18th International Carpathian Control Conference (ICCC)*. May 2017, pp. 533–538. DOI: 10.1109/CarpathianCC.2017.7970458.

[28] Robert C. Rice. *PID Tuning Guide*. https://www.novatechweb.com/wp-content/uploads/2011/03/PID_Tuning_Guide_022810.pdf. NovaTech, 2010.

[29] W.F. Sacco and N. Henderson. "Finding all solutions of nonlinear systems using a hybrid metaheuristic with Fuzzy Clustering Means". In: *Applied Soft Computing* 11.8 (Dec. 2011), pp. 5424–5432. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2011.05.016. URL: http://www.sciencedirect.com/science/article/pii/S156849461100175X.

[30] *Servo Tuning Tutorial*. ftp://ftp.pmccorp.com/pub/support/docs/tutorials/Servo_tuning_tutorial.pdf. Precision MicroControl Corporation, 2015.

[31] Mohammad Shahrokhi and Alireza Zomorrodi. "Comparison of PID Controller Tuning Methods". In: Sharif University of Technology. 2012. URL: https://pdfs.semanticscholar.org/116c/e07bcb202562606884c853fd1d19169a0b16.pdf.

[32] Y. Shi and R. Eberhart. "A modified particle swarm optimizer". In: *1998 IEEE International Conference on Evolutionary Computation Proceedings*. May 1998, pp. 69–73. DOI: 10.1109/ICEC.1998.699146.

[33] *Simple Servo Motor Tuning and finding PID gains*. https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019MPASA2. National Instruments, Jan. 2018.

[34] Steven S. Skiena. *The Algorithm Design Manual*. 2nd. Springer Publishing Company, Incorporated, 2008. ISBN: 1848000693, 9781848000698.

[35] M. Srinivas and L.M. Patnaik. "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms". In: *IEEE Transactions on systems, man and cybernetic* 24.4 (Apr. 1994), pp. 656–667. URL: http://eprints.library.iisc.ernet.in/6971/2/adaptive.pdf.

[36] Wen Tan et al. “Comparison of some well-known PID tuning formulas”. In: *"Computers & Chemical Engineering"* 30 (May 2006), pp. 1416–1423.

[37] *Tuning position controller*. https://granitedevices.com/wiki/Tuning_position_controller. Granite Devices, 2015.

[38] Luke Vandewater et al. “An adaptive genetic algorithm for selection of blood-based biomarkers for prediction of Alzheimer’s disease progression”. In: *BMC Bioinformatics* 16.18 (Dec. 2015), S1. ISSN: 1471-2105. DOI: 10.1186/1471-2105-16-S18-S1. URL: https://doi.org/10.1186/1471-2105-16-S18-S1.

[39] A. Visioli. “Optimal tuning of PID controllers for integral and unstable processes”. In: *IEE Proceedings - Control Theory and Applications* 148.2 (Mar. 2001), pp. 180–184. ISSN: 1350-2379. DOI: 10.1049/ip-cta:20010197.

[40] A. Visioli. *Practical PID Control*. Advances in Industrial Control. Springer London, 2006. ISBN: 9781846285868. URL: https://books.google.fi/books?id=ymyAY01bEe0C.

[41] S. van der Walt, S. C. Colbert, and G. Varoquaux. “The NumPy Array: A Structure for Efficient Numerical Computation”. In: *Computing in Science Engineering* 13.2 (Mar. 2011), pp. 22–30. ISSN: 1521-9615. DOI: 10.1109/MCSE.2011.37.

[42] G. Wang, S. Deb, and L. d. S. Coelho. “Elephant Herding Optimization”. In: *2015 3rd International Symposium on Computational and Business Intelligence (ISCBI)*. Dec. 2015, pp. 1–5. DOI: 10.1109/ISCBI.2015.8.

[43] Yan Wengang, Zhu Yucai, and Zhao Jun. “Closed-loop Identification based PID Tuning without External Excitation”. In: *International Federation of Automatic Control - PapersOnLine* 50.1 (2017), pp. 3995–4000. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2017.08.713. URL: http://www.sciencedirect.com/science/article/pii/S2405896317311564.

[44] M. Zhuang and D. P. Atherton. “Tuning PID controllers with integral performance criteria”. In: *International Conference on Control*. Vol. 1. Mar. 1991, pp. 481–486.

[45] J.G. Ziegler and N.B. Nichols. “Optimum Setting for Automatic Controllers”. In: *Transactions of American Society of Mechanical Engineers* 64 (1942), pp. 759–768.

APPENDIX A: LIST OF HARDWARE

Component	Vendor	Model
Linear motor forcer	Chieftec	LM-PA-X2
Magnet track	Chieftec	LM-SA-X1
Linear guide	HIWIN	WER27R640P
Linear bearings	HIWIN	WEH27CA
Optical scale	RSF	AK MS15 version MK
Optical read head	RSF	AK MS15 TTLx100
Servo drive	OptoFidelity	OptoDrive
Power supply	ProPower	PS3003

APPENDIX B: MANUAL TUNING TESTING PLAN

Test phase	Component	Description
Introduction	Purpose of test	Explain the purpose of the test
	System overview	Introduce the testers to the system
Walk-through	Test flow	Describe the test flow to the testers
	Test output	List required parameters to tune
Practical phase	Tuning w/out a mass plate	Tune axis and report parameters
	Tuning w/ light mass plate	
	Tuning w/ heavy mass plate	
Feedback	Feedback to testers	Give feedback of how testing went
	Feedback to organizers	Gather feedback about test

APPENDIX C: MANUAL TUNING TEST LOG

Test hardware is presented in Section 3.1.1.

Table C.1 *Manual tuning without mass plate.*

Pos K_p	Pos FF	Vel K_p	Vel K_i	Pos ISE	Vel ISE
498	100.0	202	104	0.04627	457.1000
2000	100.0	250	200	0.00188	177.7625
1110	100.0	250	350	**0.00170**	123.3375
800	100.0	350	180	0.00781	144.8000
100	100.0	250	350	0.01698	**100.8250**

Table C.2 *Manual tuning with mass plate 1.*

Pos K_p	Pos FF	Vel K_p	Vel K_i	Pos ISE	Vel ISE
768	100.0	235	157	0.02163	497.9500
2000	100.0	350	350	**0.00120**	172.2500
790	100.0	440	455	0.00257	104.3375
1600	99.0	350	200	0.00568	203.6125
200	100.0	325	500	0.00878	**100.4125**

Table C.3 *Manual tuning with mass plate 2.*

Pos K_p	Pos FF	Vel K_p	Vel K_i	Pos ISE	Vel ISE
801	100.0	307	198	0.01678	282.3375
2500	100.0	400	350	**0.00120**	229.6250
1200	100.0	480	550	0.00140	144.8625
1900	100.0	350	300	0.00205	202.5000
150	100.0	325	500	0.01375	**115.2375**

APPENDIX D: AUTOMATIC TUNING

Table D.1 *Automatic velocity controller tuning without mass plate.*

Vel K_p	Vel K_i	Vel K_d	Vel FF	Acc FF	Velocity ISE
177	155	0	2	9	120.1875
144	164	3	3	10	128.4375
190	125	2	2	9	155.5625
151	145	0	2	8	152.5000
144	146	1	2	10	145.0000
216	128	8	2	7	132.6250
163	163	4	2	9	131.7500
146	208	5	2	9	**111.1875**
197	151	6	2	8	160.8125
197	162	5	1	7	139.0625

Table D.2 *Automatic velocity controller tuning with mass plate 1.*

Vel K_p	Vel K_i	Vel K_d	Vel FF	Acc FF	Velocity ISE
273	228	2	3	14	106.3125
300	236	1	2	12	108.4375
283	214	7	3	13	110.9375
268	243	3	2	13	107.8750
317	226	1	3	10	113.1250
302	239	8	1	9	117.4375
313	272	1	3	11	113.8125
306	189	4	3	15	111.0625
256	224	6	1	15	**102.8750**
307	263	3	2	11	107.3125

Table D.3 *Automatic velocity controller tuning with mass plate 2.*

Vel K_p	Vel K_i	Vel K_d	Vel FF	Acc FF	Velocity ISE
360	291	8	2	13	116.9375
394	253	0	3	11	109.1250
340	279	2	2	14	107.0625
362	270	11	3	14	109.4375
388	301	1	2	13	104.2500
367	288	5	3	9	129.5625
357	256	0	3	13	107.2500
332	317	5	0	14	**101.7500**
412	290	2	2	9	128.6250
395	280	7	4	10	117.0000

Table D.4 *Automatic position controller tuning without mass plate.*

Pos K_p	Pos FF	Vel K_p	Vel K_i	Vel K_d	Vel FF	Acc FF	Pos ISE
3712	100.49	177	155	0	2	9	0.00045653
4266	100.34	144	164	3	3	10	0.00040112
3988	100.31	190	125	2	2	9	0.00049021
4772	100.48	151	145	0	2	8	0.00047657
4207	100.53	144	146	1	2	10	0.00047781
3762	100.24	216	128	8	2	7	0.00058785
3736	100.05	163	163	4	2	9	0.00041323
4465	100.05	146	208	5	2	9	**0.00027470**
3595	100.55	197	151	6	2	8	0.00051115
3535	100.50	197	162	5	1	7	0.00053523

Table D.5 *Automatic position controller tuning with mass plate 1.*

Pos K_p	Pos FF	Vel K_p	Vel K_i	Vel K_d	Vel FF	Acc FF	Pos ISE
3963	100.30	273	228	2	3	14	0.00032851
4051	100.09	300	236	1	2	12	0.00033971
4326	100.00	283	214	7	3	13	0.00036989
3904	100.45	268	243	3	2	13	0.00034402
3586	100.47	317	226	1	3	10	0.00043159
3809	100.01	302	239	8	1	9	0.00039405
3319	100.58	313	272	1	3	11	0.00035958
3489	100.04	306	189	4	3	15	0.00042831
4301	100.22	256	224	6	1	15	**0.00030451**
3647	100.52	307	263	3	2	11	0.00036033

Table D.6 *Automatic position controller tuning with mass plate 2.*

Pos K_p	Pos FF	Vel K_p	Vel K_i	Vel K_d	Vel FF	Acc FF	Pos ISE
3436	100.19	360	291	8	2	13	0.00042445
3691	100.27	394	253	0	3	11	0.00047266
3752	100.56	340	279	2	2	14	**0.00029512**
3272	100.12	362	270	11	3	14	0.00033063
3620	100.56	388	301	1	2	13	0.00046432
3794	100.44	367	288	5	3	9	0.00040648
3594	100.57	357	256	0	3	13	0.00034267
3187	100.18	332	317	5	0	14	0.00037651
3391	100.26	412	290	2	2	9	0.00046337
3333	100.03	395	280	7	4	10	0.00048350

Table D.7 *Automatic controller fine tuning without mass plate.*

Pos K_p	Pos FF	Vel K_p	Vel K_i	Vel K_d	Vel FF	Acc FF	Pos ISE	Vel ISE
4786	100.01	125	160	3	1	10	0.00048472	309.8750
5033	100.46	140	177	4	2	9	0.00044121	360.0521
4227	100.44	134	151	0	3	9	0.00054928	263.9271
4972	100.11	76	203	5	2	13	**0.00037122**	257.0104
6456	100.23	98	117	2	3	9	0.00045960	378.2396
4388	100.25	165	145	11	1	7	0.00067049	355.5833
5496	100.19	117	182	4	2	8	0.00048245	418.8021
3637	100.30	142	255	4	1	11	0.00059362	483.9375
4999	100.49	114	165	0	2	10	0.00039041	**210.4271**
4192	100.56	138	210	5	2	11	0.00061040	566.7917

Table D.8 *Automatic controller fine tuning with mass plate 1.*

Pos K_p	Pos FF	Vel K_p	Vel K_i	Vel K_d	Vel FF	Acc FF	Pos ISE	Vel ISE
4793	100.16	132	320	8	1	18	0.00029126	222.8438
4428	100.34	205	278	2	2	18	0.00035907	328.5104
3475	100.07	227	299	4	2	16	0.00033374	208.7500
3770	100.35	243	343	0	2	15	0.00039882	342.9167
4314	100.35	189	303	7	4	15	0.00032502	232.9583
3462	100.41	213	343	5	3	16	0.00037952	264.0833
2746	100.04	300	345	0	2	16	0.00040089	257.4375
2639	100.17	332	212	1	2	20	0.00046924	**170.3021**
4987	100.46	158	288	4	2	18	**0.00025798**	203.2917
3662	100.32	203	319	10	3	16	0.00033849	218.5938

Table D.9 *Automatic controller fine tuning with mass plate 2.*

Pos K_p	Pos FF	Vel K_p	Vel K_i	Vel K_d	Vel FF	Acc FF	Pos ISE	Vel ISE
3740	100.01	339	355	8	3	16	0.00052527	532.0000
4225	100.47	276	300	2	2	18	0.00030561	240.5208
3966	100.06	289	306	2	2	15	0.00030168	172.4375
3298	100.28	278	345	1	3	21	**0.00026530**	153.1563
5064	100.18	244	229	1	3	18	0.00033172	299.3854
3280	100.19	379	322	0	4	12	0.00042691	237.6875
3875	100.43	215	323	2	2	19	0.00026865	**144.2083**
3187	100.56	272	405	3	3	17	0.00031786	186.0521
2998	100.25	363	355	11	2	16	0.00034895	207.2083
4536	100.31	271	259	3	3	15	0.00028135	172.8229

Table D.10 *Automatic controller fine tuning with combined ISE, without mass plate.*

Pos K_p	Pos FF	Vel K_p	Vel K_i	Vel K_d	Vel FF	Acc FF	Pos ISE	Vel ISE
3731	100.27	160	140	10	2	12	0.00059178	246.8542
5441	100.54	116	165	9	2	9	**0.00035732**	**245.6771**
5557	100.57	95	163	0	2	10	0.00040248	266.7708
4874	100.44	134	146	14	3	11	0.00044239	271.8542
4009	100.15	172	156	1	3	9	0.00054944	246.7917

Table D.11 *Automatic controller fine tuning with combined ISE, with mass plate 1.*

Pos K_p	Pos FF	Vel K_p	Vel K_i	Vel K_d	Vel FF	Acc FF	Pos ISE	Vel ISE
3494	100.10	236	275	1	2	16	0.00031664	183.7292
4446	100.38	194	278	1	2	13	0.00031032	192.3646
4860	100.48	183	233	2	1	16	0.00030353	183.1250
4617	100.45	224	247	1	2	14	**0.00026838**	174.5625
3924	100.56	242	250	1	3	16	0.00029033	**159.5625**

Table D.12 *Automatic controller fine tuning with combined ISE, with mass plate 2.*

Pos K_p	Pos FF	Vel K_p	Vel K_i	Vel K_d	Vel FF	Acc FF	Pos ISE	Vel ISE
3316	100.15	232	360	13	1	19	0.00031472	150.8542
2356	100.29	352	341	7	2	19	0.00038071	**125.4896**
4563	100.55	191	289	7	3	16	0.00031888	181.3333
3114	100.26	337	358	5	4	14	0.00032746	148.2813
2947	100.20	344	344	2	3	17	**0.00027700**	130.8854