Digital Controller Design of an Industrial Sewing Station Using Evolutionary Algorithm

Eric Henrique Moretti - 39477

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Engenharia Industrial.

Trabalho orientado por:

Prof. Dr. João Paulo Coelho

Prof. Dr. Gilson Schiavon

Bragança

2019

Digital Controller Design of an Industrial Sewing Station Using Evolutionary Algorithm

Eric Henrique Moretti - 39477

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Engenharia Industrial.

Trabalho orientado por:

Prof. Dr. João Paulo Coelho

Prof. Dr. Gilson Schiavon

Bragança

2019

Acknowledgment

I want to thank my family, for all the effort made and all the support that they gave to me, encouraging and holding up everything although the distance.

A thanks to professor Dr. João Paulo Coelho, who encouraged and enlightened me through the development of this work.

I want to thank all the friends that I made on this journey, in Brazil and Portugal. I will remind them forever. A special thanks to Gustavo and Ighor, whose I will sing at their weddings.

Abstract

The PID controller is one of the most used systems in the industry due to its simplicity and efficiency. However, the main problem to implement this controller is to set the correct gain to get the desired system response.

The purpose of this thesis is to get the values of proportional, integral and derivative gain of the controller to obtain an adequate response to an industrial sewing workstation.

The sewing station consists of a sewing machine installed into a cart, which is moved by an induction motor guided by a rail. When the user accelerates the sewing machine, the motor must move the car in the speed as the sewing machine sews the fabric. Thereby, the controller needs to send the correct control signal to the motor system, to move the car at the set-point speed.

From this system, a mathematical model was obtained, that is used to develop the controller, which will be implemented in a digital microcontroller.

The controller was designed using the Genetic Algorithm, using the step response characteristics (e.g. overshoot, rise time, steady-state error) and the control signal to calculate the fitness function. Also was used the Particle Swarm Optimization to compare the results.

With the PID gains found as a result of the evolutionary algorithms, that with the better performance was implemented for tests on the company FactoryPlay, which design and produces inflatable structures for theme parks, based on the district of Bragança.

Keywords: PID controller, parameter estimation, digital control, genetic algorithm.

Resumo

O controlador PID é um dos sistemas mais utilizados na indústria devido à sua simplicidade e eficiência. No entanto, o principal problema para implementar este controlador é definir o ganho correto para obter a resposta do sistema desejada.

O objetivo desta tese é obter os valores de ganho proporcional, integral e derivativo do controlador para obter uma resposta adequada a uma estação de trabalho de costura industrial.

A estação de costura consiste em uma máquina de costura instalada em um carrinho, que é movida por um motor de indução guiado por um trilho. Quando o usuário acelera a máquina de costura, o motor deve mover o carro na velocidade em que a máquina de costura costura o tecido. Assim, o controlador precisa enviar o sinal de controle correto ao sistema do motor, para mover o carro na velocidade do ponto de ajuste.

A partir desse sistema, foi obtido um modelo matemático, usado para desenvolver o controlador, que será implementado em um microcontrolador digital.

O controlador foi projetado usando o algoritmo genético, usando as características da resposta ao degrau (e.g. *overshoot, rise time, steady-state error*) e o sinal de controle para calcular a função objetivo. Também foi utilizada a otimização por enxame de partículas para comparar os resultados.

Utilizando os ganhos do PID encontrados como resultado dos algoritmos evolutivos, foi implementado para testes na empresa FactoryPlay, que projeta e produz estruturas infláveis, situada no distrito de Bragança.

Palavras-chave: Controlador PID, Estimativa de Parâmetros, Controlo Digital, Algoritmos Evolutivos.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The company "FactoryPlay" is a industrial plant that design and produces inflatable toys, brand stands and all kind of inflatable structures. It is located at the industrial area of Mos, in the northeast region of Bragança, Portugal.

There are a large variety of products on their portfolio, such as inflatable sliders, castles, jumpers, toddler zones, obstacle courses and other toys. Some examples of those products are shown on Figure 1.1.

(a) (b)

(c) (d)

Figure 1.1: Some examples of inflatable structures. Images provided by FactoryPlay.

And the active brands catalog includes inflatable arches, columns, tents, totems, flags, among others. There are some example of this kind of product on the figure 1.2.

(a) (b)

(c) (d)

Figure 1.2: Brand Stands produced by the FactoryPlay. Images provided by FactoryPlay.1

Its inflatable products are made of a anti-fire treated sailcloth with a density of $620\,g/m^2$ [1]. To sew those fabrics, it is used an industrial electrical sewing machine.

The method used to sew this kind of objects is by placing the fabric in a large table while the sewing machine moves at the border of the table.

This method is implemented on the factory, and the sewing machine is moved using the operator feet. This may cause problems to the worker and also is an antiquate method of production.

To change this method to a modern approach, a new sewing station prototype was proposed, which is constituted by a cart with the sewing machine and a seat, moved by an electrical motor on a rail. Figure 1.3 shows a 3D model of the sewing station.

As shown in Figure 1.3, it is represented one piece of the table where the sailcloth will be sewed. Also is shown the cart, moved with wheels on rails. It is shown the desk where

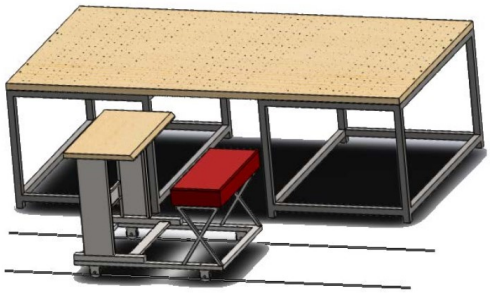


Figure 1.3: 3D model of the Sewing Station

will be installed the sewing machine.

The station is moved by the rail using a tree phase AC induction motor, which rotates at 1350 revolutions per minute. The system also uses a gear reduction box with a ratio of 30:1, reducing the output to 45 revolutions per minute. The system is controlled by a PI digital controller, installed on an Arduino board, but the PI parameters was adjusted using some aspects of the system response to set its constants.

1.1 Objectives

As this system already have a prototype built, the mainly work will happen over the digital controller.

The main objectives of this thesis are:

- Complete the instrumentation of the sewing station prototype;
- Obtain the dynamic response of the system;
- Develop a mathematical model of the system;
- Utilize classical techniques for PID controller design;
- Develop an evolutionary algorithm to design the PID controller and compare the results.

1.2 Structure of this work

This work is divided in mainly three parts:

Theoretical Background In this chapter it will be explained all the concepts and consolidated studies about the area of this project: mathematical modelling, PID controllers and evolutionary algorithms.

Modelling and Controller Desing of a Sewing Station In this chapter will be analysed and described the prototype of the sewing station and all of its particularities. Also will be develop in this section all the concepts shown on the chapter of the Theoretical Background, being implemented, to reach the objectives of this work. Also will have some discussions about the preliminary results of all steps over the development of the solutions.

Results, Conclusion and Future Work In these chapters will be analysed the results obtained from this work, discuss this solutions and propose the future works, if necessary.

1.3 Related Work

In this section will be analysed some other projects and solutions developed by the academy and the industry over the area of the sewing machines.

The sewing machines are the main tool of the textile industry. They were been developed to the human use and in the last decades, it is a point of interesting of the automation process.

One of the turning points of the automation of sewing machines was in 1983, with the paper "Vision Servo Control of a Robotic Sewing Machine" [2]. It shows the design of a sewing tool to be attached to a robotic arm, that performs the sewing proess and the manipulation of the cloth, using a camera to do the process.

In this project, it uses a PD controller to the system, and the K_p and K_d values was founded doing experimental tests changing these parameters and checking the results.

But the focus of this project do not was the controller, was the vision mechanism to control the robotic arm and the sewing tool.

In other approach to the sewing machines, the work "Adaptive Control of an Electromagnetically Actuated Presser-Foot for Industrial Sewing Machines" [3] describes an alternative method of read the presser-foot of the sewing machines to achieve a better control of the machine.

This project had analysed the presser, and it uses the output of an LVDT (linear variable differential transformer, that is used to measure linear movements, using a metal rod) to read the position of the presser and sensors in the machine to detect the stitches per minute.

These values are used to perform the control of the system. The paper uses initially a PID controller to control the machine, and later includes a Fuzzy Logic Control to improve the reference of the system, and the results were working as expected.

An other related paper is the "Study on Sensing and Monitoring of Sewing Machine for Textile Stream Smart Manufacturing Innovation" [4]. It uses the concepts of the *Industry 4.0* to enhance the production of the sewing and cloth production process, implementing some Internet of Things Concepts.

This system monitor all the system, from the order of the product, trough the production until the delivery of the finished job.

Besides that, it have a device for check and indicate the remaining of the under-thread sewing yarn of the sewing machine, to avoid the sewing without the under-thread yarn. It shows the amount of yarn on the machine on a human-machine interface device, using an bobbin with an encoder to detect the rotation of the bobbin yarn, using a photo sensor.

It also have a stitch control device and a sewing thread information detection system that record the current work and adjust the correct number of stitches to the kind of fabric and yarn used in the order, using the human-machine interface display and the controller in the sewing machine panel.

Another interesting paper is the "Parallel process decomposition of a dynamic manipulation task: robotic sewing" [5], that studies the implementation of a PUMA robot

system to manipulate the fabric which is being sewed in a sewing machine. It investigates the parallel decomposition of manipulation tasks that involve interaction with a dynamic environment.

Manipulate fabric is not a simple task. Robots manipulates solid objects easily, but fabric and cloths are very complicated due the large deformations, the very low bending resistance and its non-linearity.

To do so, it is used a "two finger" device which is controlled using some parallel techniques, and to control the actuator it is used a PI controller, which the K_p and K_i parameters are calculated using trial and error.

To improve these results, the paper "Model Reference Fuzzy Learning Force Control for Robotized Sewing" [6] uses a fuzzy logic controller to have a better result from the previous paper, but using a SCARA robot.

Another paper on the sewing machines area is the "Development of assistive technology to operate industrial grade sewing machines for differently-abled persons" [7], that uses a potentiometer as input to control the sewing machine in cases that the user have problems in the motion to control the sewing machine with the presser-foot pedal.

It have two approaches: one using a angular potentiometer, and other using a pulley to convert a linear motion to the angular motion of the potentiometer. Also, it uses a AC motor driver to control the system, and it uses a microcontroller to read the input from the potentiometer and convert to the control signal to the AC motor driver.

Chapter 2

Theoretical Background

2.1 Mathematical Modeling

There are many ways to design the mathematical model of a system, such as the analytical methods using the time response, analytical methods using the frequency response, numerical methods and the use of artificial intelligence. In this work, will be used an analytical method, where it is used the comparison of a known mathematical model with the real model to obtain the transfer function of the system.

With the plant modeled, it is possible do design an efficient compensator or controller.

There are usually two types of systems responses that are most commonly found: the first-order and the second order systems. Higher order systems exists, but they are out of the scope for this work. It is important to identify what kind of system is the plant to apply the mathematical analysis to find its function.

2.1.1 Continuous-Time System Responses

To have the mathematical model of the system it is required to have the system response to a known input test signals. It is important to have a basis to evaluate the performance of the system, or from a part of the system. Usually the test signal is the step function,

defined by the following parameters:

$$\begin{aligned} &r(t) = 0 \; \forall \; t \; < \; 0 \\ &r(t) = A \; \forall \; t \; > \; 0 \\ &r(t) \text{ is undefined, if } t = 0 \end{aligned} \tag{2.1}$$

Alternatively, in the Laplace domain:

$$R(s) = \frac{A}{s}, \qquad Re[s] > 0 \tag{2.2}$$

Where A is the amplitude of the signal. Also, other signals are used, such as the impulse, the ramp, the sinusoidal and others arbitrary functions, as the pseudo-random binary sequence. If the signal is mathematically described, it is possible to evaluate its response to achieve its temporal function and its transfer function. However, as the step function is the most used, it will be the focus of the analysis of the systems.

Using the step function, the transient period of the plant response is analyzed [8].

2.1.2 First-order Systems

A first-order system is a linear and time-invariant system, whose equation is given by

$$\tau \dot{y} + y = f(t), \quad \tau > 0 \tag{2.3}$$

Where *tau* is the time constant of the system. To find its transfer function, it is necessary to considerate the initial conditions $y_0 = 0$. Thus, applying the Laplace transform to 2.3 leads to:

$$\tau[sY(s) - y_0] + Y(s) = F(s) \tag{2.4}$$

Isolating $Y(s)/F(s)$,

$$\frac{Y(s)}{F(s)} = \frac{1}{\tau s + 1} \tag{2.5}$$

Therefore, the objective here is find the step response of the system, which is used to identify the τ constant in of the system [9]. Applying the step (2.2) to the system as the input function on the equation 2.4,

$$\tau[sT(s) - y_0] + Y(s) = \frac{A}{s} \tag{2.6}$$

Rearranging,

$$Y(s) = \frac{A}{s} + (y_0 - A)\frac{1}{s + \frac{1}{\tau}} \tag{2.7}$$

Lastly, its step response in time is

$$y(t) = A + (y_0 - A)e^{-t/\tau}, \quad t \geq 0 \tag{2.8}$$

Thus, the system response is represented in the Figure 2.1.

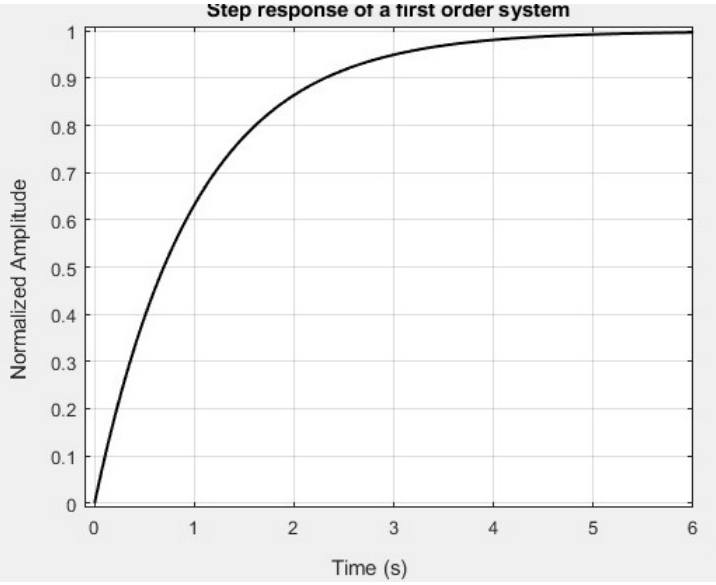


Figure 2.1: Step response of a first-order system using $\tau = 1$.

As can be seen from the figure 2.1, when the time reaches one time constant, the amplitude of the signal is $0.632 * A$ due to the characteristic of the exponential curve.

The most efficient way to find the transfer function of a first-order system is measure the time when the amplitude of the signal reaches $0.632 * A$ to find the value of τ for the (2.5) and (2.8).

2.1.3 Second-Order Systems

The equation for a standard second-order system is given by

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = f(t) \tag{2.9}$$

Where ζ is the damping ratio and the ω_n is the undamped natural frequency of the system. Those values are who describes the behavior of all second-order systems. To find the values of ζ and ω_n, it is used the step response of the system, as used in the first-order systems.[9]

The second-order systems are divided in tree instances:

1. Underdamped Case: Occurs when $1 > \zeta > 0$. In this case, the system's step response have a overshoot with a amplitude inverse of ζ: for lower ζ, have a higher overshoot and for higher ζ have a lower overshoot.

   In this case, the closed-loop poles are complex conjugates and lie in the left-half of the s plane. In this case, the transfer function can be written

   $$\frac{Y(s)}{F(s)} = \frac{\omega_n^2}{(s + \zeta\omega_n + j\omega_d)(s + \zeta\omega_n - j\omega_d)} \tag{2.10}$$

   Where $\omega_d = \omega_n\sqrt{1-\zeta^2}$. Applying a step function to the system, and calculating the Inverse Laplace Transform results in

   $$\mathcal{L}^{-1}[Y(s)] = y(t) = 1 - \frac{e^{-\zeta\omega_n t}}{\sqrt{1-\zeta^2}} \sin\left(\omega_d t + \tan^{-1}\frac{\sqrt{1-\zeta^2}}{\zeta}\right), \quad \forall\, t \geq 0. \tag{2.11}$$

Thus, from (2.11), the amplitude of $c(t) \to 1$ when $t \to \infty$, and the transient response is a exponential multiplied by a sine. At the transient period is when the figures of merit are shown. With these values is possible to find the values of ζ and ω_n. [10]

2. Critically Damped Case: This case occurs when $\zeta = 1$. The poles in the s plane are equal. In this case, the system response for a unit step can be written

$$Y(s) = \frac{\omega_n^2}{(s + \omega_n)^2 s} \tag{2.12}$$

And its inverse transform is

$$\mathcal{L}^{-1}[Y(s)] = y(t) = 1 - e^{-\omega_n t}(1 + \omega_n t), \quad \forall\, t \geq 0 \tag{2.13}$$

3. Overdamped Case: Happens when $\zeta > 1$. Its poles are negative, real and unequal. Appling a unit step input in this system, have the following response:

$$Y(s) = \frac{\omega_n^2}{(s + \zeta\omega_n + \omega_n\sqrt{\zeta^2 - 1})(s + \zeta\omega_n - \omega_n\sqrt{\zeta^2 - 1})} \tag{2.14}$$

It is very similar to the equation 2.10, with the difference that here don't have the j component of the imaginary plane, because in this system the poles are real. With some simplifications, the inverse Laplace transform results in

$$\mathcal{L}^{-1}[Y(s)] = y(t) = 1 - e^{-(\zeta - \sqrt{\zeta^2 - 1})\omega_n t}, \quad \forall t \geq 0. \tag{2.15}$$

The figure 2.2 shows the step response of a second order system with different values of ζ.

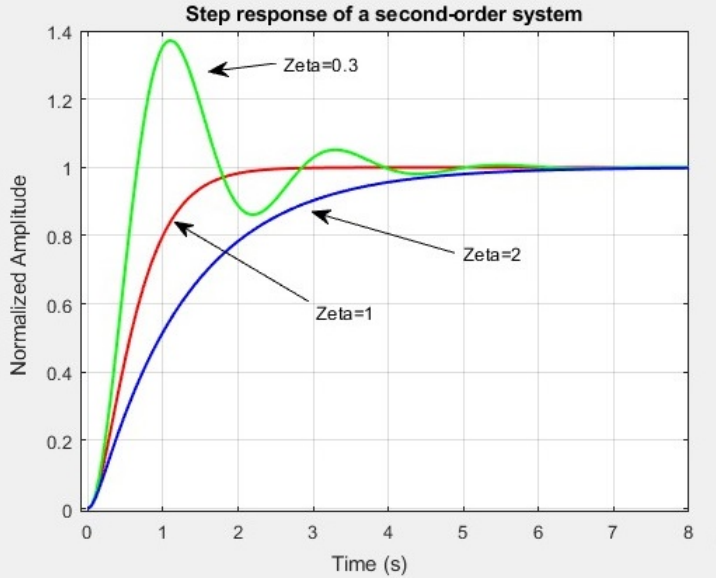


Figure 2.2: Variation of a Second Order ζ parameter.

2.1.4 Second-Order System's Figures of Merit

The figures of merit are the characteristic values on the transient period that can be used to find the required values to create the mathematical model of the system. The main figures of merit are shown on Figure 2.3.

With the values of some of those figures of merit it is possible to find the value of ζ and ω_n, which are the values desired to find the transfer function of the system, using the table 2.1.

Symbol	Name	Formula
M_p	Peak Overshoot	$M_p = 1 + e^{\frac{-\zeta\pi}{\sqrt{(1-\zeta^2)}}}$
t_p	Peak Time	$t_p = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}}$
t_s	Settling Time	$t_s = \frac{4}{\zeta\omega_n}$

Table 2.1: Main Figures of Merit used to find the transfer function of a second-order sytem

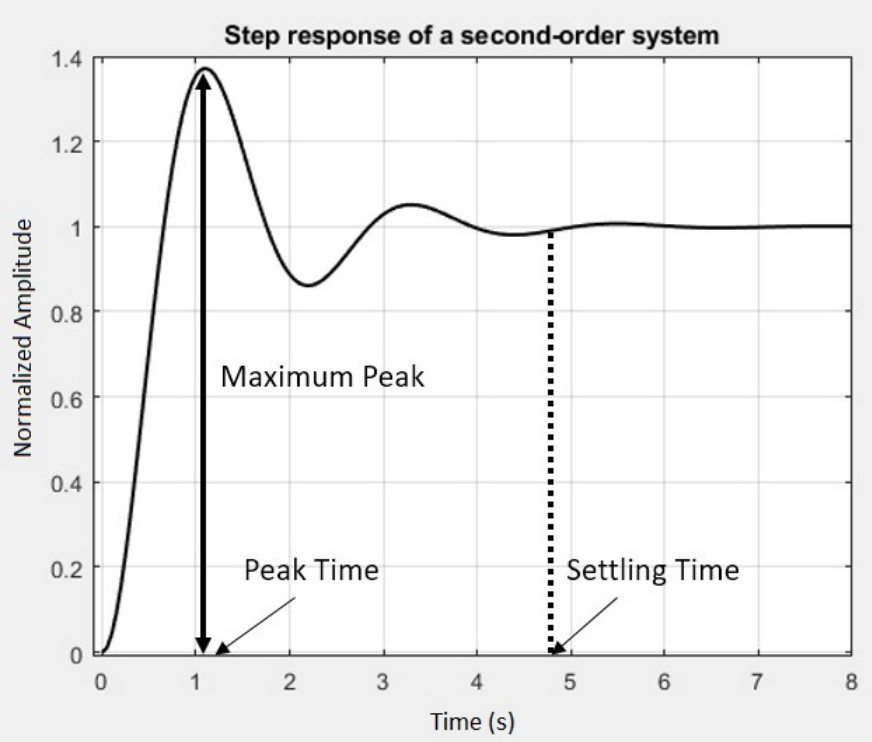


Figure 2.3: Figures of Merit used to calculate the transfer function

2.2 PID Tuning

The PID controller is widely used in the industry. It can be implemented as a digital or analog controller, depending on its application [11]. Considering $e(t)$ the input of the controller and $m(t)$ its output,the equation that defines the controller in continuous time domain is

$$m(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2.16}$$

And the transfer function of the controller G(s), from the Laplace Transform of the equation 2.16 is

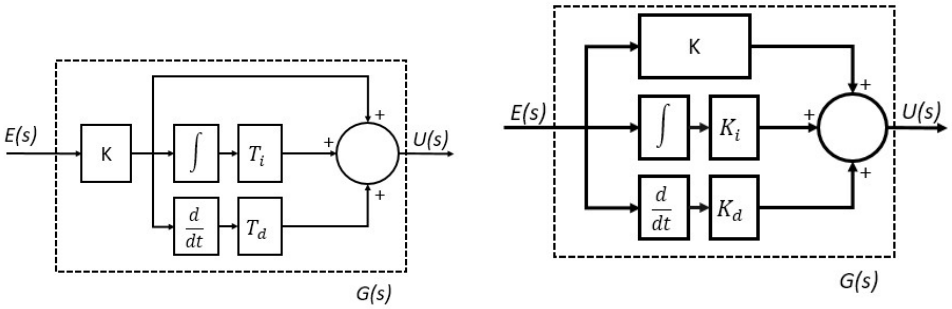
$$G(s) = \frac{M(s)}{E(s)} = K_p + \frac{K_i}{s} + K_d s \tag{2.17}$$

There is an alternative way to write the equation, that will be used on this work, that

uses the K_p isolated from the others constants, as shown on the following equation:

$$G(s) = K\left(1 + \frac{T_i}{s} + T_d s\right) \tag{2.18}$$

The block diagram for those systems are represented in the figure 2.4.



(a) (b)

Figure 2.4: Block Diagram of the PID controller

There is a modification to the PID controller which includes an extra pole to the system with the T_f constant, which works as a extra pole for the system to become realizable. It also is useful as a simple first order filter to the system. The constant T_f is calculated using the equation 2.19:

$$T_f = \frac{T_d}{N} \tag{2.19}$$

Where N is in the order of 5 to 10. Thus, the final form of the PID transfer function is

$$G(s) = K\left(1 + \frac{T_i}{s} + \frac{T_d s}{1 + T_f s}\right) \tag{2.20}$$

The main problem in the PID controller is to find the best values to the desired system response. There are practical methods, numerical methods and analytical methods that are used to find those values, but all of them have their limitations.

To solve this problem, there are some methods which will be explained in the next sections.

2.2.1 Classical PID tuning Methods

The classical methods uses the plot of the system response or the analysis of the poles and zeros to find the desired values of PID constants using objective equations. Some of the most used are listed below.

1. **Ziegler-Nichols Method**

   The Ziegler-Nichols Method is a practical method based on the "ultimate sensitivity", which is the value of proportional gain applied to the system that makes the system oscillate in a constant amplitude when $Kd = Ki = 0$. This value of gain is called Km, and the frequency of oscillation in the ultimate sensitivity is called ω_m [12], as the figure 2.5 shows.

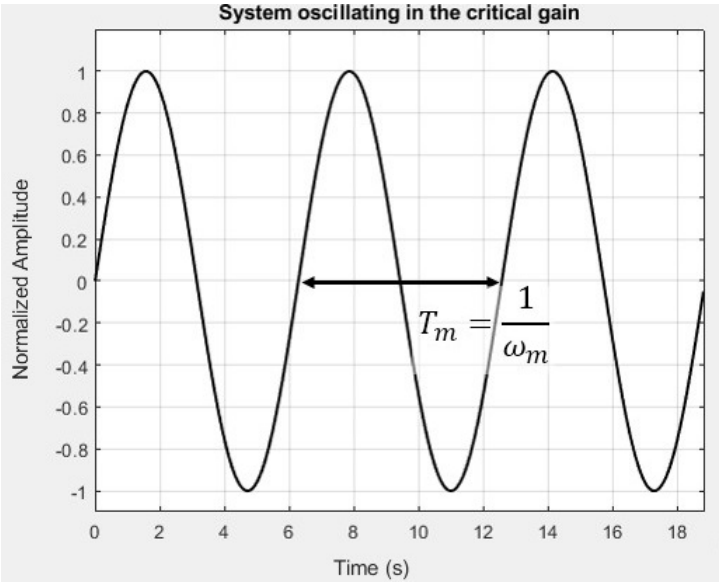


Figure 2.5: Example of system response in the ultimate sensitivity

With those values it is possible to find the values of Kp, Ki and Kd using the following equations:[13]

$$K_p = 0.6K_m \tag{2.21}$$

$$K_d = \frac{K_p \pi}{4\,\omega_m} \tag{2.22}$$

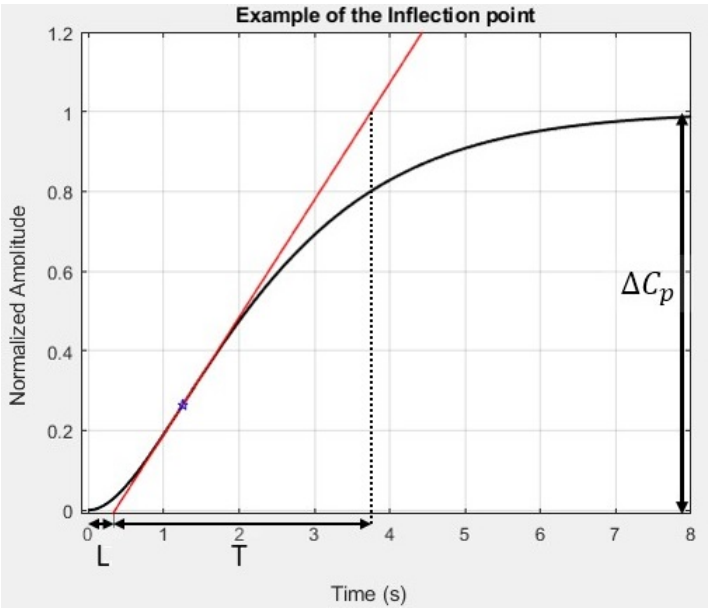


Figure 2.6: Analysis over the Inflection Point of the Step Response

$$K_i = \frac{K_p \, \omega_m}{\pi} \tag{2.23}$$

The main problem of this method is that it finds a subjective "good response". It leads to a response with a small overshoot, good stability and without steady-state error, but still having a overshoot — and in some cases it is undesirable. Also all the other parameters cannot be changed, as the rise time, for example.

2. **Ziegler-Nichols Reaction Curve Process**

   This method uses the open-loop step response to find the values of K, T_i and T_d. It will use the values based on the graphical representation of the step response [10].

   The first step on this method is find the inflection point of the curve. After that, it is drawn a tangent curve on this point. With the tangent point, it is find the values of L and T, as the figure 2.6 shows.

   The values found on this curve are in the table 2.2:

   With those values, the values of K, T_i and T_d are calculed using the following

Parameter	Description
L	Lag Time (min)
T	Time Constant Estimate (min)
δC_p	Change in the amplitude in response to step disturbance
$N = \frac{\Delta C_p}{T}$	Reaction Rate
$R = \frac{NL}{\Delta C_p}$	Lag Ratio

Table 2.2: Values of the Parameters from the Inflection Point

equations:

$$K = 1.2\left(\frac{P}{N\,L}\right) \tag{2.24}$$

$$T_i = 2L \tag{2.25}$$

$$T_d = 0.5L \tag{2.26}$$

3. **Cohen-Coon open-loop rules**

   This method is a set of tuning recommendations based on the Ziegler-Nichols Reaction Curve Process, enhanced by Cohen and Coon [14]. The parameters are calculed by the following equations:

$$K = \frac{P}{NL}\left(1.33 + \frac{R}{4}\right) \tag{2.27}$$

$$T_i = L\left(\frac{32 + 6R}{13 + 8R}\right) \tag{2.28}$$

$$T_d = L\left(\frac{4}{11 + 2R}\right) \tag{2.29}$$

4. **Method Chien-Hrones-Reswick**

   This method also uses the inflection point to find the same parameters of the table

3.2 [15]. The values of K, T_i and T_d are calculated with the following equations:

$$K = \frac{0,95T}{L} \tag{2.30}$$

$$T_i = 2.4L \tag{2.31}$$

$$T_d = 0.42L \tag{2.32}$$

2.3 Optimization methods using Artificial Intelligence

2.3.1 Genetic Algorithm

The genetic algorithm is a stochastic optimization algorithm inspired on the Darwin's Evolution of Species[16]. It was developed over the 60s by many computer scientists groups and engineers groups , with the goal to find solutions to problems that were too difficult to solve with analytical methods[17].

The evolutionary process occurs based on a initial population, which will reproduce between its individuals, creating the new generation. Each individual have its own *genetic code*, which have the information of every individual. When have a offspring, the genetic code of the parents will merge to a new genetic code of the child, resulting in a different individual, with characteristics of both parents.

To this process create a better individual, every individual have to be evaluated as how good it is to solve that problem — or to survive in the environment. This will determine the probability to the individual reproduce and transmit its genes to the next generation [16].

The process of reproduction happens in all new generations, until find the best solution as possible: being a predefined solution or to a open-ended solution, called artificial evolution and natural evolution, respectively. The artificial evolution have a specific goal,

which is used to find the solution to a predefined problem, while the natural evolution is used to find the best solution to a general problem, without restrictions.

For the usage of those concepts of the evolution to solve problems, it need to be described as computing functions and parameters to formulate a program to run this process.

A genetic algorithm program uses basically two functions to work: the fitness function and the function that makes the reproduction in the population.

Population The population is the group of individuals. Each individual have its own genetic code, that is coded by the characteristics of the individual. The characteristics must have information that can be used to solve the problem, which will be used in the fitness function to evaluate how good is this individual, each one being a potential solution for the problem.

Genetic Representation Each individual have its own genetic code, called chromosome. It contains all information about that specific individual that will be used to calculate its fitness, and the chromosome must be coded to work on the computing algorithms. There are four most used forms of strings to code the chromosome: the binary encoding and the real-number encoding [18].

The binary encoding was most used in the beginning of the development of the genetic algorithms and it is a quite effective solution to simple problems. The main problem of this representation is the drawbacks caused by the Hamming Cliffs, which is a pair of encodings that have a large Hamming distance, but a small Euclidean distance in the phenotype space. The problem with this is that is some cases, the mutation and crossover will have a small chance to reach all localities of points in the phenotype space[18].

Thus, the binary sill being used and it is the representation of the value in binary. A very common used variation is the usage of the gray system. It avoids the problem with the Hamming Cliffs.

The figure 2.7 shows the binary respresentation of three parameters.

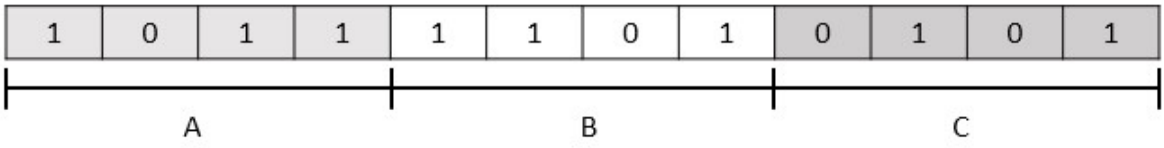


Figure 2.7: Example of the binary representation

For the engineering problems, it is very used the real-number encoding, which is a good solution to evaluating optimal values of functions. The point of the real-number encoding is it have its precision according the number of bits allowed by the application, that need to be enough to find the required precision, without have a large usage of memory.

Fitness Function The fitness function is a function that reads the genetic code from a chromosome and decodes it to associate a score from that chromosome. It is the way to determine the performance of each individual do decide if this individual will have a higher or lower rate to reproduce.

It is the part of the genetic algorithm that requires the most part of the processing power, because it is executed to every individual at all population, in every offspring. [17].

The choices of design of the the fitness function can guide the result of the genetic algorithm [19]. For example, if the genetic algorithm is designed to find the best dimensions to a product box, it can be the smallest size to fit the product, the most resistant shape or the cheapest solution. It will guide the algorithm to the solution that the programmer wants.

In a multiple objective function, might be used a sum of all the objectives fitness, and make a weighted sum of all objectives scores. If a score is most important than others, it have a higher weight, to ensure it will reach that objective primarily, but counting that the others will have their heights taken into account.

Genetic Operators There are two genetic operators in genetic algorithms. The first is the *crossover*, which makes the mixing of genotypes between individuals, and the mutation, which is a random operation that applies a change in the genetic code of a random

number of individuals to ensure a genetic variation [17].

Crossover The crossover is an operator that makes the interaction with the pairs of individuals, providing the reproduction. It takes a part of the genetic code of one individual and changes with other, generating a new different individual, with characteristics of the two parents. This phenomenal is called recombination of genotypes or genetic recombination.

There are three most common approaches to do the genetic recombination. The first, and most common is the *one-point crossover*, which is used in real-representation problems. It takes the parents and select a random point in the chromosome and swap all the genotype from that point until the end of the chromosome, resulting in two individuals with parts of the two parents, as shown in the figure 2.8.

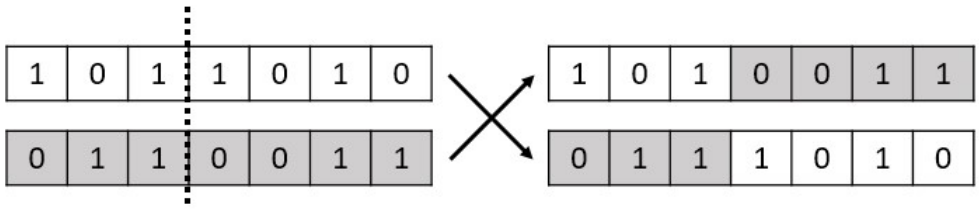


Figure 2.8: Representation of the One-point Crossover

The second crossover operator for real-number and binary representation is the *uniform crossover*, that swaps random genotypes of the parents, in random positions, as can be seen in the figure 2.9.

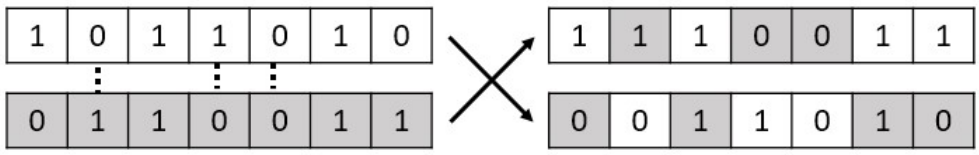


Figure 2.9: Representation of the Uniform Crossover

The third crossover operator is the *arithmetic crossover*, that creates a single genotype making the average of the parents genotype in randomly positions, as seen in the figure 2.10.

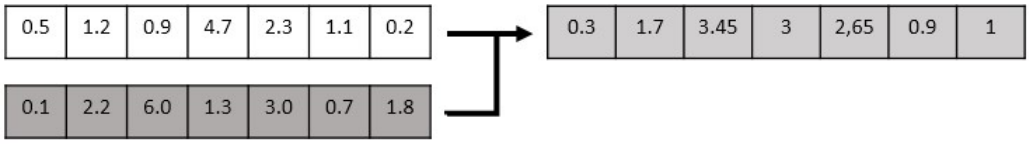


Figure 2.10: Representation of the Arithmetic Crossover

Mutation The mutation is a operator that changes the genotype of an individual. It happens randomly, and provides small changes with the objective of reach a larger range of solutions in the genotype.

It happens in random individuals of the population, following the mutation rate (usually around 0.01). The most common operation to mutate the individual is toggle a bit of the binary representation in a random position, if the individuals are represented in binary.

If the individuals are represented in real-number values, it can be added a random number in a Gaussian distribution in a random genotype of the chromosome. The figure

(a) Random bit toggle

(b) Adding a random number to a real-value representation

Figure 2.11: Representation of the mutation in an individual

Selection The selection is the part of the algorithm that defines the rate of each individual to survive for the next generation. With a bad selection algorithm, the search may terminate without finding the best solution or the progress will be too much slower [18].

Roulette Wheel One of the most used algorithm is the Roulette Wheel Selection [17], that determines the survivability rate of each individual based on its fitness. The name "roulette wheel" is based on the gamble game with the same name. The behavior of the roulette is simple: the numbers present on the original game are the individuals of the population. Every individual have its area on the roulette proportional to its fitness score. If the individual chromosome have a lower fitness, it will have a small area on the roulette. If have a higher fitness, will have a larger area on the roulette, as represented in the figure 2.12.

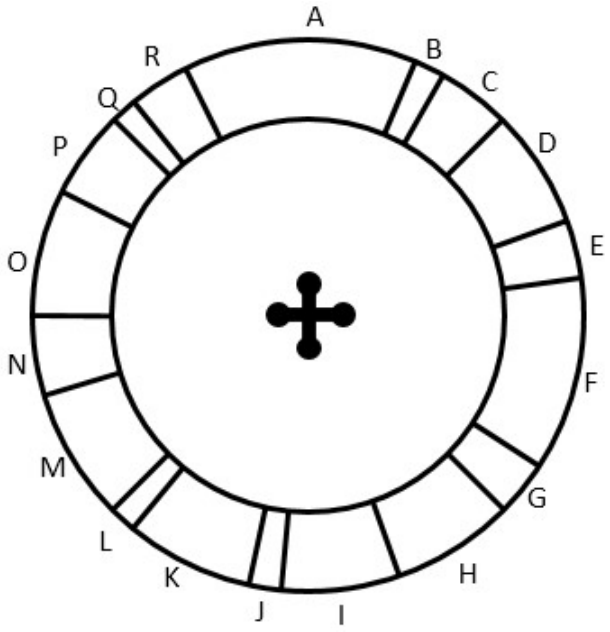


Figure 2.12: Representation of how the roulette area is distributed to all individuals. Each individual is represented by a letter, and its chance to reproduce is proportional to its area on the roulette.

The roulette is rolled the same number of times as the size of the population, to fill the offspring of the next population [17]. As the survivability rate is based on the fitness score, some individuals may have two copies of its chromosome on the next generation. If the fitness of that individual its too high, it can cause the system to converge to a premature local solution, not the global solution. Otherwise, if all the population have a similar fitness, the solution can became a random search.

Rank-based Selection The rank based selection is based on the Roulette Wheel, but the rate of every individual is made using a ranking besides the fitness. It is created a vector with the ranking of all individuals in the population. The higher rank individuals will have a larger area, but proportional to its respective ranking position, not based on its fitness, as the Roulette Wheel.

It avoids the situation of the individual have a very large fitness score at the Roulette Wheel. If it is used the fitness to find the area of the roulette, this individual may have a absurdly large portion of the wheel, while the others will have a very small area, that can takes it to converge to this individual, which may be a local maximum — and the rank-based selection avoids this problems.

Truncated Rank-Based Selection This selection makes the ranking of all populations individuals and takes copies of the best n individuals to the next offspring. This method will remove the lowest score individuals, but at the same time increases the rate of reproduction of the mean score individuals.

Tournament Selection The tournament selection picks a small random set of elements from the population and makes a ranking of their fitness. The individual with the highest ranking goes to the offspring. This process is repeated until fill all the new population.

Geneational Replacement Consists in keep all the offspring to the next generation, without select individuals. This may cause noise to find the best fitness, and may not converge to the best solution. To avoid this, it can be used the *elitism*. The elitism keeps a small part of the best fitness to the next generation.

Multiple Objective Function A multiple objective function is used when there are more then one criterion to be taken in consideration simultaneously [20]. For example: evolving a solution to find a better design for a airplane wing that have the best lift, minimum drag and it is composed by the minimum quantity of pieces [17].

Real problems using a genetic algorithm usually requires a multi-objective function to reach the solution. The only problem for this type of problem is very hard to find the perfect balance: it is quite rare to find a solution that solves all the problems.

One solution for the multi-objective problem is use a weighted sum to ensure all solutions will taken in account. The weights are based on how important that single objective is. The final result will be a single scalar value as fitness function, and the algorithm will use it to find the solution to the problem.

There are others solutions as the Random-Weight Generic Algorithm, the Vector Evaluated Genetic Algorithm and the Non-dominated Sorting Genetic Algorithm [20], but it wont will be used in thos work.

Constraints In some cases, it is necessary to use constraints in the genetic algorithm because some responses are not feasible, or some responses represents a solution that does not is affordable to realize it.

In those cases, it is possible to apply some constraints into the genetic algorithm to achieve the solution in a desired limit. The constraints must be inside of inequality limits. In every generation, it is evaluated the fitness score of every individual, and if the individual is out of the constraints, it is removed from the offspring, before the selection.

2.3.2 Particle Swarm Optimization

The Particle Swarm was originally created to simulate a simplified social system, with the intend to simulate a bird flock . Nowdays, it is used as optimization algorithm to find the minimum or maximum values of functions starting with a random population as the Genetic Algorithm. The differences from the Particle Swarm Optimization to the Genetic Algorithm is that each potential solution have a randomized velocity, and the potential solutions moves through the problem space [21], while the potential solutions in the Genetic Algorithm are combined to create new solutions.

This algorithm is based on some simple concepts that rules the behavior of socially organized populations in nature. It makes the population explore the search space with

stochastic moves. The best position of each particle is called *experience*, and it is saved on the memory [22]. All the experiences of the swarm are shared to determine the most promising regions so far, and it is used to do the next movement of the swarm.

The working process of the PSO begins with a initialization of a random population S, which must be randomly distributed over the space.

$$S = \{x_1, x_2, ..., x_n\} \tag{2.33}$$

Each element of the array population is a particle, and may be a scalar or a array of parameters.

The population is evaluated by a objective function f, similar to the fitness function in the Genetic Algorithm, where

$$f_i = f(x_i), \quad \forall\, i \tag{2.34}$$

With this set, it is possible to begin the interation. The particles will move with a velocity $\mathbf{v}$, which will have a determined velocity to each particle. At each interation, the velocity will be recalculated considering the potentially to find the global best solution.

All the positions found ever are saved in a array $\mathbf{P}$, which is called the memory, as mentioned before. To know the best solution, it is used an another array p_g of each particle, which will save the best position found.

Thus, the velocity and the position can be calculated by the following equations:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 R_1(p_{ij}(t) - x_{ij}(t)) + c_2 R_2(p_{gj}(t) - x_{ij}(t)) \tag{2.35}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{2.36}$$

Where R are random variables uniformly distributed within [0,1].

Those movements to each particle makes the swarm travel through the space and converge to the best solution, depending of the implementation of the algorithm. There

are some variables that can change the performance of the algorithm, as the random values **R**, on the equation 2.35.

2.3.3 PID Digital Controller Design

There are two most used methods to transform a continuous PID controller to a digital PID controller: the bilinear transform and the trapezoidal approximation.

Bilinear Transform The Bilinear Transform, also known as the Tustin algorithm is an approximation to the s-domain function to the z-domain function [8]. It is achieved substituting the terms of s using the equation 2.37.

$$s \rightarrow \left(\frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}} \right) \tag{2.37}$$

Where T is the sampling period.

This is a simple way to transform the continuous s-domain to a digital z-domain function, but have it limitation: it only works well to a limited frequency, but the only way to evaluate it is testing and checking its behaviour to higher frequencies.

With the z-domain transfer function, it is made the inverse z-transform, but to apply the time discrete function in a microcontroller, the results of the transfer function in z domain is usually a series of impulses with delays. To do the inverse z-transform of this type of transfer function, it is used the time delay propriety.

The time delay propriety uses the relation of the equation 2.38, that simplify the process, substituting the value of k.

$$\mathcal{Z}^{-1}\{z^{-k}X(z)\} = x[n-k] \tag{2.38}$$

The equation found in the discrete time will be used in the microcontroller as the controller equation.

Trapezoidal Approximation The trapezoidal approximation is a more intuitive implementation for the PID Controller. it uses directly the form in the continuous time from the PID controller and approximate it to the digital form, without pass trought the z-domain [23].

Using the PID controller represented as the sum of the tree terms

$$U(s) = U_P(s) + U_I(s) + U_D(s) \tag{2.39}$$

Each term of the equation can be calculated separately and be summed in the end of the process.

The proportional action is represented by the equation 2.40.

$$U_P(s) = KE(s) \tag{2.40}$$

Where K is the proportional gain and $E(s)$ is the error signal. It can be simply replacing its values in the kth sampling instant, as in the equation 2.41.

$$u_P(kh) = Ke(kh) \tag{2.41}$$

Where k is the sampling element and h is the sampling period.

The integral action is represented by the equation 2.42.

$$U_I(s) = K\frac{1}{T_i s}E(s) \tag{2.42}$$

It corresponds to the integral expression of the equation 2.43

$$u_I(t) = K\frac{1}{T_i}\int_0^t e(\tau)d\tau \tag{2.43}$$

Approximating the equation 2.43 to a summation, it can be written as the equation2.44

$$u_I(kh) = K\frac{h}{Ti}\sum_{n=0}^{k} e(nh) \tag{2.44}$$

Extracting the last term of the summation, it is obtained the equation 2.45

$$u_I(kh) = u_I(kh-h) + K\frac{h}{T_i}e(kh) = u_I(kh-h) + K\alpha e(kh) \tag{2.45}$$

Where

$$\alpha = \frac{h}{T_i} \tag{2.46}$$

The derivative action in the s-domain is represented by the equation

$$U_D(s) = -K\frac{T_d s}{1+T_f s}Y(s) \tag{2.47}$$

Replacing T_f by Td/N and expanding the equation will lead to

$$U_D(s) = NK\left[-Y(s) + \left(\frac{N}{T_d}/\left(s+\frac{N}{T_d}\right)\right)Y(s)\right] \tag{2.48}$$

Applying the backward differences, will lead to the final form of the derivative action, which can be written as

$$u_D(kh) = \beta u_D(kh-h) - K\frac{T_d}{h}(1-\beta)[y(kh) - y(kh-h)] \tag{2.49}$$

Where

$$\beta = \left(1+\frac{hN}{T_d}\right)^{-1} = \frac{T_d}{T_d+hN} \tag{2.50}$$

It is important to notice that the derivative action uses the output of the system, besides de error as its input. It is used like this to avoid the *derivative kick* when the input of the system is a step function or a inpulse. When are used these signals, it is created a abrupt change in the derivative form, which can lead to a abnormal behavior [23].

Chapter 3

Modelling and Controller Design of a Sewing Station

In this chapter will be analysed the case of the prototype of a sewing station which is being developed at the IPB to be used in the company FactoryPlay, which is situated in Bragança, Portugal.

The company FactoryPlay produces inflatables toys (as inflatable castles, sliders, jumpers, aquatic toys, etc.) and brand stands (as brand arches, columns, tents and etc.).

To sew the fabric of these products, it is used a industrial sewing machine, as the model shown on Figure 3.1.

This type of machine is widely used on the industry. It is build in steel, and its motor is installed under of the body of the sewing machine, as shown in Figure 3.2.

The machine is drive by a foot pedal, which accelerates the machine, as the user press the pedal. The pedal is shown in Figure 3.1b.

This system is being used in the FactoryPlay. The problem in the production of inflatables is because of the size of the fabric, the fabric stays on a large table and the sewing machine is moved on the border of the table, unlike the production of clothes, where the fabric moves and the sewing machine is immobile.

(a) Sewing Machine

(b) Body of the Sewing Machine

Figure 3.1: Pictures of the Sewing Machine

Currently, this set is used, with the sewing machine installed in a cart with wheels, and this cart is move by the user, using the user's legs to provide the power to move the cart.

To solve this problem, it is made a prototype of this sewing station, which uses the sewing machine installed on the cart, which is moved by an electrical motor. The station is shown on Figure 3.3.

The prototype of the sewing station was developed in 2018 and it is described in the article [1].

The motor must have to move in the same speed as the fabric is being sewed. That means, as fast the fabric is sewed, faster the motor must have to move the cart.

To do so, this system will need to measure the speed of the sewing machine to set the velocity of the motor. To measure the speed of the sewing machine and the speed of the motor, will be used a magnetic encoder, which will be used to set the value of the motor.

The speed of the motor is also measured because it will be used in a closed-loop controller. With a closed loop controller, the system reaches a better stability, a small error and it can be applied a large range of different controllers.

With the readings of motor speed and the machine speed, it is possible to use a

Figure 3.2: Back of the sewing machine. Detail to the motor, under of the body of the sewing machine.

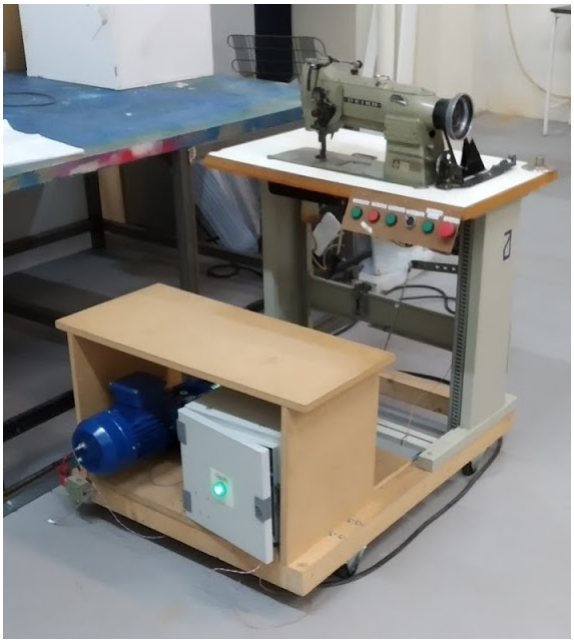


Figure 3.3: Overview of the Sewing Station

microcontroller to calculate the signal to send it to the motor driver. This setup is represented in Figure 3.4.

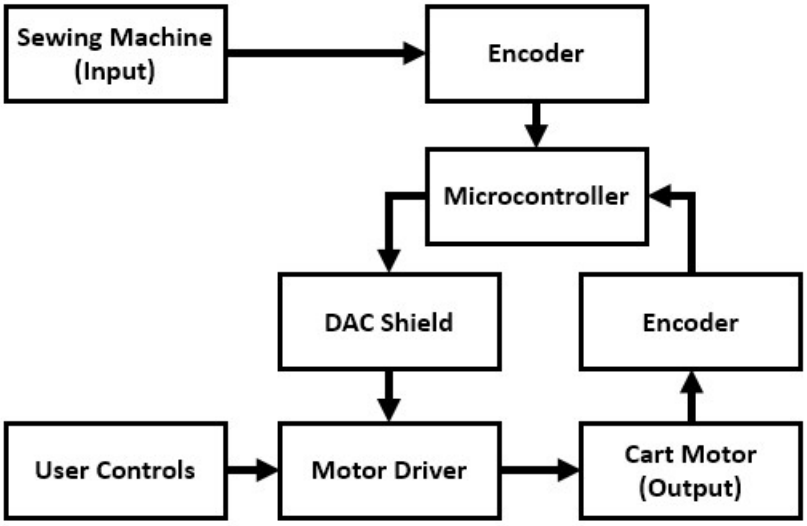


Figure 3.4: Diagram of the Sewing Station

Sewing Machine The sewing machine is controlled by a foot pedal, which is triggered by the user, who set the machine speed. The machine is powered by a electrical motor. In the end of the axis from this electrical motor is set placed the disk with magnets, that will be used to trigger the sensor and send the pulses to the microcontroller.

Encoder The encoder, as mentioned before is made of two elements: a disk of 50 mm diameter, with twelve neodymium magnets spaced with an arc of 30° between each magnet. A picture of the disk is shown in Figure 3.5.

The second element of the encoder is the US5881LUA Hall Effect sensor, which will be used to verify when have a magnetic field over it. The sensor is positioned in the perpendicular of the disk, in the radius where the magnets are installed.

The Hall effect sensor is installed on a board with a circuit that only purpose to serve as interface to connect the sensor to the Arduino. The board also have an LED to show when the sensor is being powered and if the sensor is measuring the magnetic field, blinking when the signal changes.

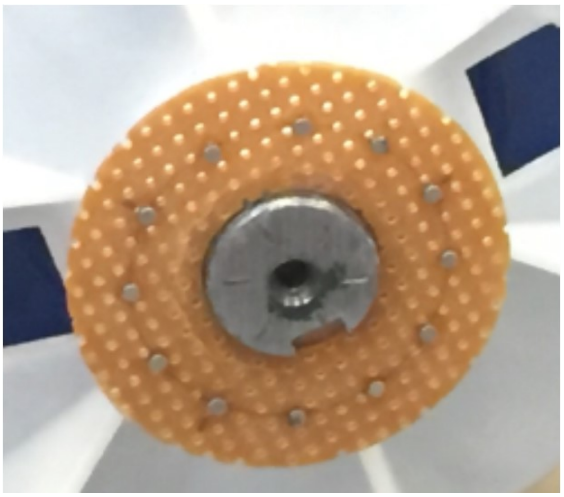


Figure 3.5: Encoder disk with the magnets

The output from the sensor will be a high level signal when the sensor is out of a magnetic field, and when the sensor is in the magnetic field, the signal will be in the low level.

Thus, when the motor rotates, the magnets will move and the output from the sensor will be a square wave, with the frequency proportional to the speed of the motor.

This set is used as an incremental encoder to calculate the speed of the motors. There is a routine in the microcontroller to count the pulses sent by the sensor. At the sampling period — which is 10 ms on this project — the value of the counter is the number of pulses done over that time, and it is computed, and the counter returns to zero. To know exactly the speed of the machine, its required to calculate the ratio of the number of pulses that are equivalent to the speed of the motor.

Microcontroller The microcontroller board used in this project is the Arduino Uno R3, shown in Figure 3.6. It will be user because of it is a simple and practical solution. *E.g.* already have the power circuit implemented on its board, have the serial communication set on USB, do not need an external device to send the program to the internal memory. It also have all features required to this project, as the processing clock required, internal and external interrupts.

The Arduino Uno R3 uses a ATmega328p as it microcontroller. The ATmega328p

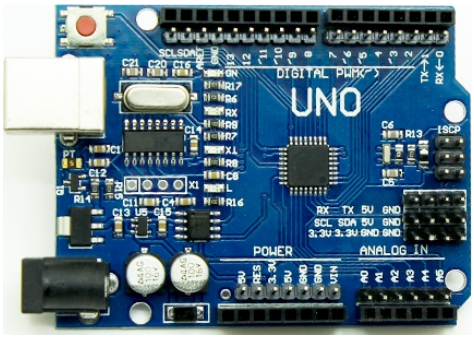


Figure 3.6: Arduino Uno R3

have 32KB of flash memory, nominal clock of 20MHz, 2 timers of 8 bits and one timer of 16 bits. Also have two external interrupts, on the digital port 2 and 3 and have PWM-ready ports. Other additional specification are available on the datasheet of the product [24].

The connections to the Arduino used are the following:

Power To power the microcontroller was used a external power supply, connected in the power connector from the Arduino.

Digital Ports 2 and 3 The digital ports was used as input to the sewing machine and motor encoders. Those ports was chosen because it have the external interrupt circuit installed on it, which is used to implement the incremental encoder.

Digital Port 11 It is the output of the system. It sends the controller signal to the motor driver trought a PWM singnal, using the *analogWrite()* command, from the Arduino IDE. That signal is amplified in the DAC shield.

Logic Lever Converter Shield The Logic Level Converter shield was designed to provide a better connection between the Arduino board and the sensors, and accommodate the output amplifier circuit. The prototype of the shield is shown in Figure 3.7. In the future, will be developed a final version of this circuit, with better connections.

To connect the sensors to the Arduino, was used PCB terminal blocks with 3 pins:

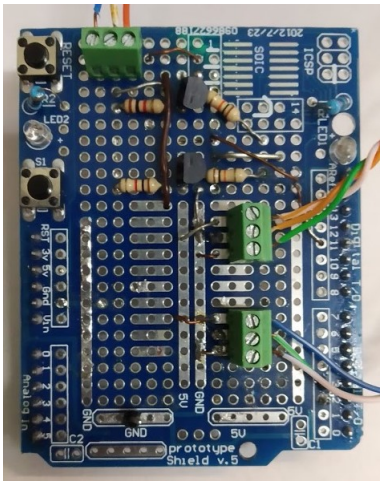


Figure 3.7: Picture of the shield prototype

one pin to ground, one pin to data and one pin to power. It was used one connected to the digital pin 2 and 3 of the Arduino as the input for the Hall Effect sensor, and one connected to the digital pin 11 as output of the PWM signal to the driver.

The power pin of the connector which will be used to connect the motor driver was used to connect the +10 V from the motor driver, because the Arduino does not provide that voltage, and the input of the motor driver range is from 0 V to 10 V. That is why it uses a amplifier on the board: to adequate the signal from Arduino to the motor driver input.

The amplifier circuit is represented on Figure 3.8.

This circuit is designed to apply a high gain on the transistors, that will saturate the transistors with any input signal higher then the threshold of the input of 0.8 V [25]. With the circuit being powered with 10 V, when the transistors saturate, the output will be 10 V.

In a previous version of this shield, was used a R-2R ladder resistor network DAC to convert an parallel digital output to a analog signal. But it show some problems: first, because of the characteristics of the circuit, was necessary a power supply higher then the output. That means it was requiring a power supply with more then 10 V.

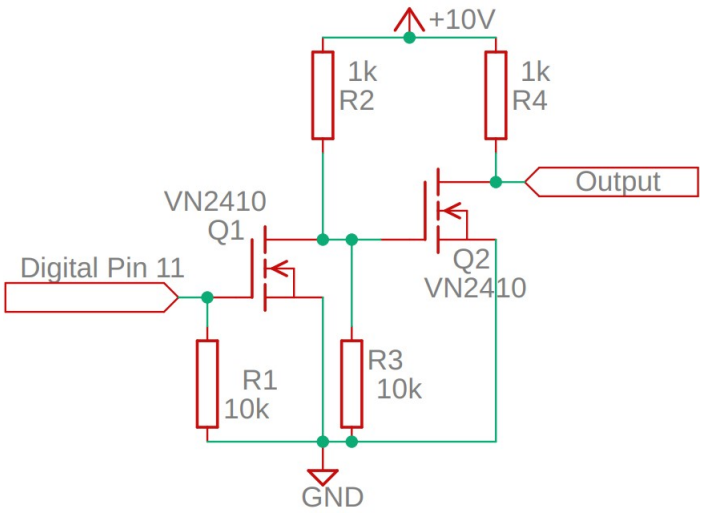


Figure 3.8: Logic Level Adapter Circuit.

As it would require more space and more components to work, was verified this new solution, using a modulated PWM output to the system. The function *analogWrite()* of the Arduino generates a PWM with the frequency of $980Hz$, and the duty cycle proportional to the output value.

Was verified on the motor driver that this type of signal is filtered, and the system works exactly as a real analog signal.

Thus, by the simplicity of this solution, this was the chosen one.

Cart Motor The motor used to push the cart is produced by the CEMER, and is a tri-phase alternating current motor. Its specifications are listed on the table 3.1.

Parameter	Value
Nominal Frequency	50 Hz
Nominal Voltage	400 V
Power	0.75 kW
Power Factor	0.78
Current	1.90 A
Revolutions per minute	1380 min^{-1}

Table 3.1: Cart motor specifications

A picture of the motor with the reduction box is in Figure 3.9.

Figure 3.9: Picture of the cart motor.

It also is used a speed reduction gearbox, with the ratio of 30:1 in the output of the cart motor to reduce the motor speed and increase the torque.

Motor Driver The motor driver used is the WEG CFW10, which is a motor inverter, shown in Figure 3.10. It allows the scalar control of the motor, which will be used on this project. It also can control motors up to 5 hp, which is over the necessary to the cart motor of 1hp.

The motor driver will be used to control which direction the motor will rotate, based on the user input. It will also control the cart motor speed, using the scalar value calculated by the microcontroller.

User Controls The user have a panel to control the station with five buttons and a switch, as shown in the image 3.11.

The function of each button are the following:

Automatic This green button turns on both motors: the sewing machine and the cart motor.

Figure 3.10: Picture of the motor driver WEG CFW10

Figure 3.11: Picture of the user's control panel

General Stop This red button turns off both motors.

Movement This green button turns on the cart motor

Front/Back This switch chooses the direction of movement of the cart: to the front or back of the user. If it is turned to the left, goes to the front (*Frente*); if it is on the right, the machine goes back (*Trás*).

Sewing Machine This red button turns the sewing machine on.

Emergency This red emergency button have the same function of the General Stop button: turns off both motors. This button is locked when its pressed, and need to twirl its button to release it.

The labels of the buttons are in Portuguese because the machine will be implemented on a Portuguese company, with Portuguese speakers employees.

3.1 Sewing Station Mathematical Modelling

3.1.1 Data Acquisition

With the Sewing Station ready, the first step to find the mathematical model is acquire the system response. For this, was developed a Arduino program that reads the engine speed, using the hall sensor and the magnets, similar to a encoder.

The program have an external interrupt set, and every time the external interrupt happens (due to the sensor output is change) an internal counter is incremented. Also have a timer set to every 10 milliseconds the counter goes to zero, and its value is sent to another register as the speed of the engine, and this register with the velocity is sent by the serial port to the computer.

With this code implemented in the Arduino Uno R3, it was possible to measure the RPM from the engine. Using the serial monitor from the Arduino API, was possible to acquire the values from the system.

With all set up, the system was run and measured ten times with a step as input. From those ten measurements, was removed 3 runs, because they were too out of the standard response. It was made because the purpose of this process is have the standard system response, not an abnormal or a noisy response. The remaining responses was used to make a average response, making the average response to each time step.

The result of those operations are represented on Figure 3.12:

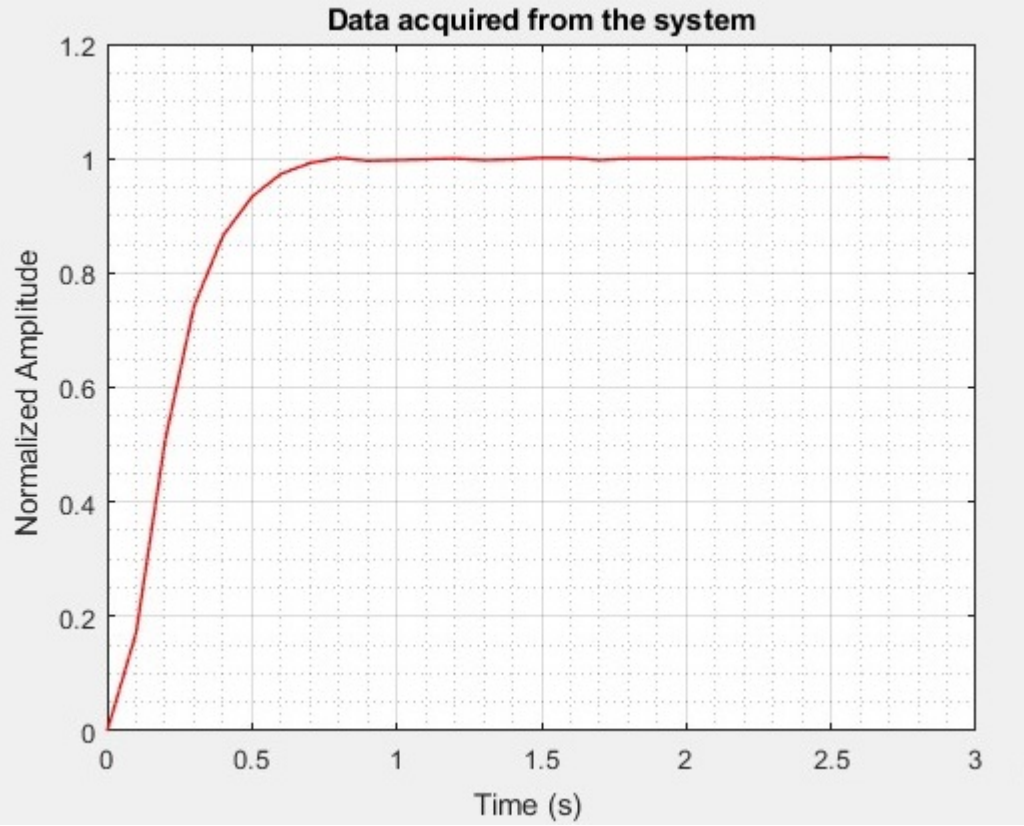


Figure 3.12: Acquired Engine Response

3.1.2 First-Order System Modelling

With the system response from Figure 3.12, based on the visual analysis it is possible to see the similarity with a second-order system. But, as the system have a very small overshoot and the rising curve is not very dumped, it is possible to check if the system fits over a first-order system.

As mentioned on the section 2.1.2, the first-order systems only depends of the parameter τ, which is found based on the time it takes to the system reaches to 63.2% of the steady-state amplitude.

Using the data cursor from the MATLAB on Figure, was found the following values:

$$X : 0.253$$

$$Y : 0.6325$$

Using the equation 2.5, it is possible to modelate the value of $\tau = 0.253$. Thus, the system can be represented as

$$G(s) = \frac{3.953}{s + 3.953} \tag{3.1}$$

The step response of the equation 3.1 is plotted in Figure

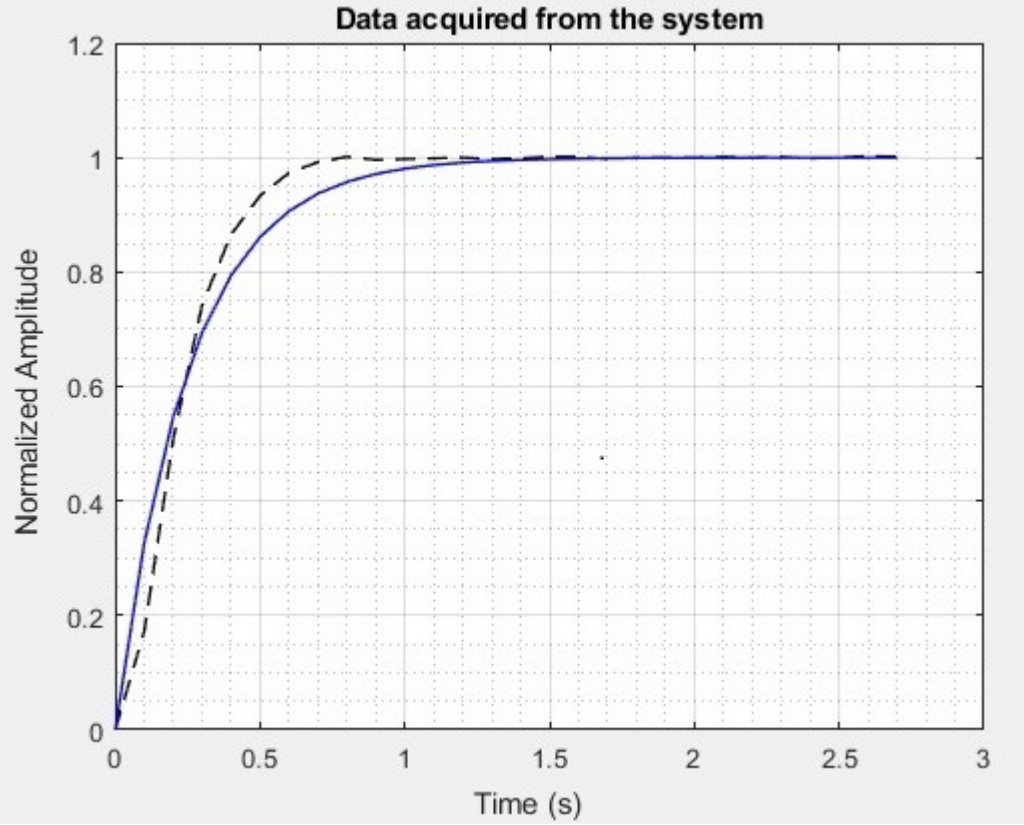


Figure 3.13: System response of the First-Order Model

3.1.3 Second-Order System Modelling

The main option to check is the second-order system model. To model it, is required to find two parameters: the ζ and the ω_n, as mentioned in the section 2.1.3. The best way to find it is using Figures of merit, shown in Section 2.1.4, using the overshoot (Mp) and the peak time (t_p)

Isolating the ζ from the first equation and ω_n the on the table 2.1, we will find the

following equations:

$$\zeta = \sqrt{\frac{\ln(Mp-1)^2}{\pi^2 \ln(Mp-1)^2}} \tag{3.2}$$

$$\omega_n = \frac{\pi}{t_p\sqrt{1-\zeta^2}} \tag{3.3}$$

From Figure 3.12, it is possible to find the values required for the equations: the overshoot and the peak time. As the overshoot is very small, it is more trustworthy use the values in the MATLAB instead a manual measure. The values are:

$$t_p = 0.9$$
$$Mp = 1.001481700992740$$

Applying the values on the equations 3.2 and 3.3, will find the values:

$$\zeta = 0.900733958553509$$
$$\omega_n = 8.036118730219977$$

Using these values on the equation 2.10, will lead us to the following transfer function:

$$H(s) = \frac{64.58}{s^2 + 14.48s + 64.58} \tag{3.4}$$

The step response of this system is shown in the Figure 3.14.

3.1.4 Results and Discussion

With a visual inspection it is possible to see that the second order system has a best system response to the problem, as figure 3.15 shows.

However, to measure the response quality, will be used the MATLAB System Identification Tool, which calculates the mean square error to find how much the system response

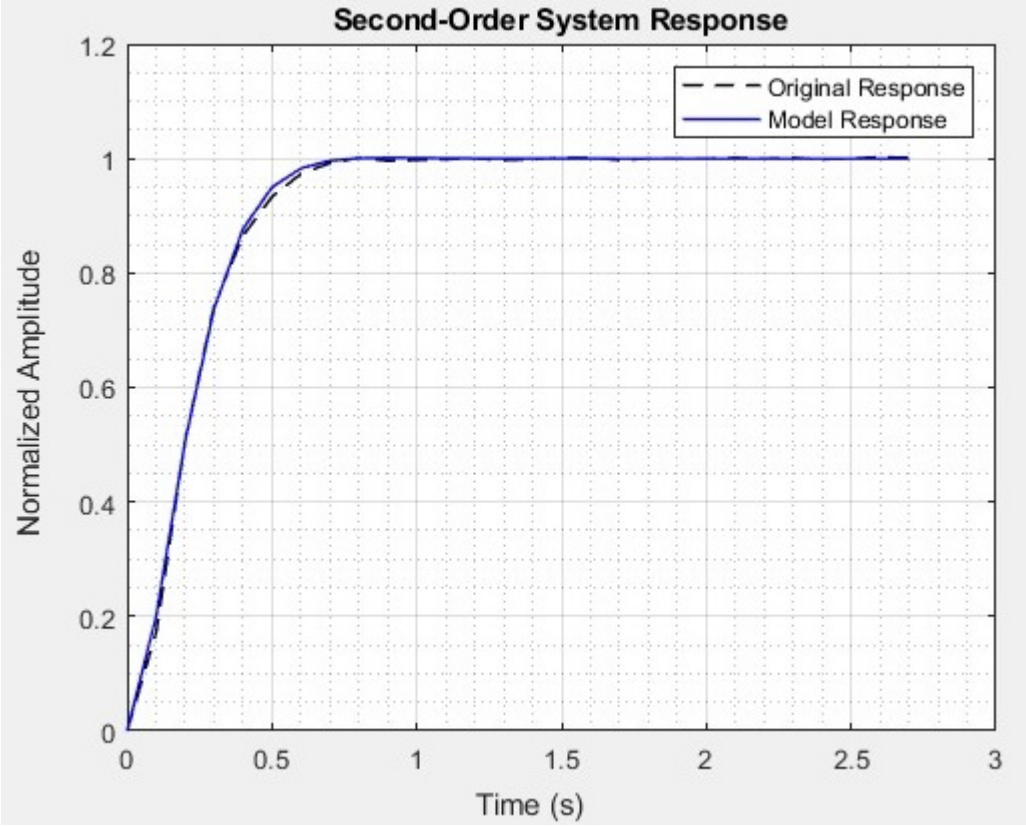


Figure 3.14: Step Response of the Second-Order System

fits over the original data. The results are the following:

First-Order System : 83.73%

Second-Order System : 97.6%

Based on this results, the second order is the best mathematical model to this system.

3.2 PID Parameter Estimation

3.2.1 Classical Models

There are some classical models that use some characteristics of the system response to calculate the parameters for the PID controller.

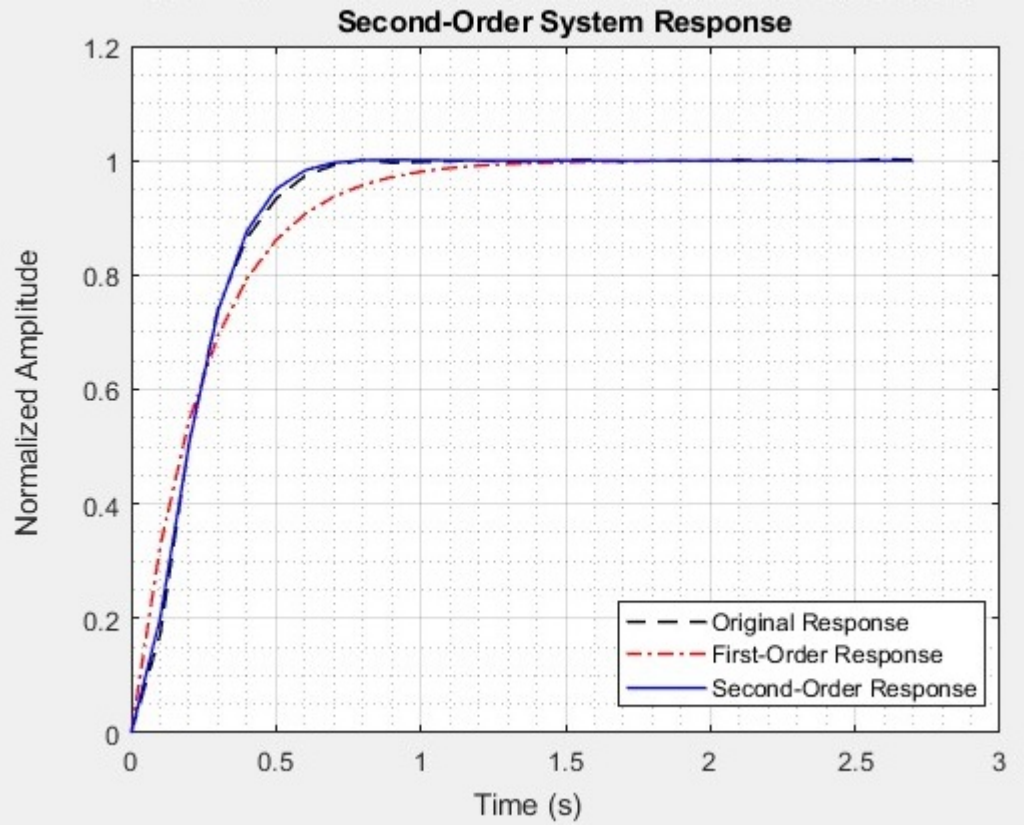


Figure 3.15: Comparison of the systems responses

Ziegler-Nichols Method

The firs method is the Ziegler-Nichols. To find the critical gain, the value of K was increased until the system oscillate. Unfortunately, the system does not oscillate, as can see in the root-locus plot in Figure 3.16.

As proven in the root-locus plot, the system will never oscillate because independent of the value of K, the system never goes to the right side of the plan.

Thus, this method is invalid to this system.

Ziegler-Nichols Reaction Curve Process

The reaction curve process is calculated using the open-loop step response of the system, as described on the section 2. The tangent curve found is in Figure 3.17.

The values of the parameters to these algorithms was calculated using the MATLAB.

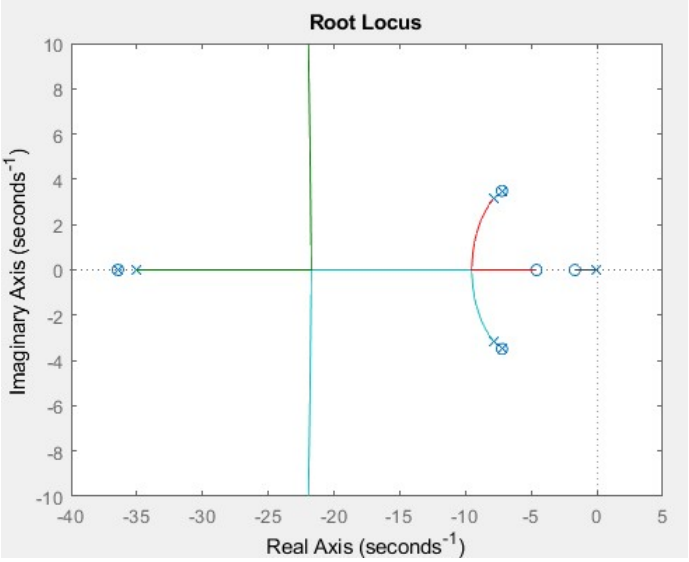


Figure 3.16: Root-locus of the system

All codes are listened in the appendix.

The values found for this system are in the table 3.2.

Parameter	**Description**
T	0.033182437444563
L	0.326751113709001
ΔC_p	1.001481700992739
N	30.181077043116016
R	9.847110064017857
P	1

Table 3.2: Table of parameters found with the inflection point

With this values, was calculated the PID parameters, and the result was

$$K = 0.121682867526252$$

$$Ti = 0.653502227418002$$

$$Td = 0.163375556854501$$

The step response with these values is in Figure 3.18

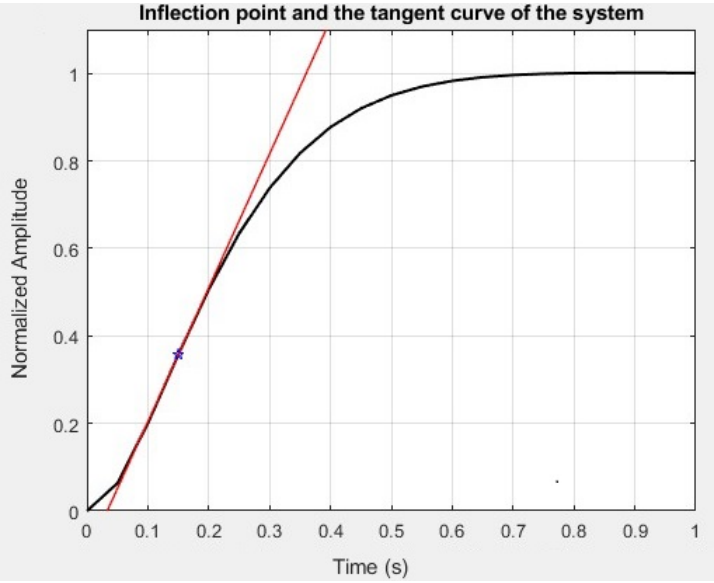


Figure 3.17: Tangent curve and the inflection point of the system

This response respects the limit of the controller effort, but have a very slow response, which is inadequate for the sewing station.

Cohen-Coon open-loop rules

Using the same parameters found with the Ziegler-Nichols Reaction Curve Process, was calculated the PID parameters using the equations of the Cohen-Coon open rules.

The PID parameters found was

$$K = 0.384495300974160$$

$$Ti = 0.324279497777744$$

$$Td = 0.042581451797246$$

The system response with these parameters is plotted in Figure 3.19.

This method have a faster response, in comparison with the Ziegler-Nichols method, but have an oversignal of the controller effort. With this signal, the real implementation will not work as the simulation.

Thus, this method also not is available to solve the sewing station problem.

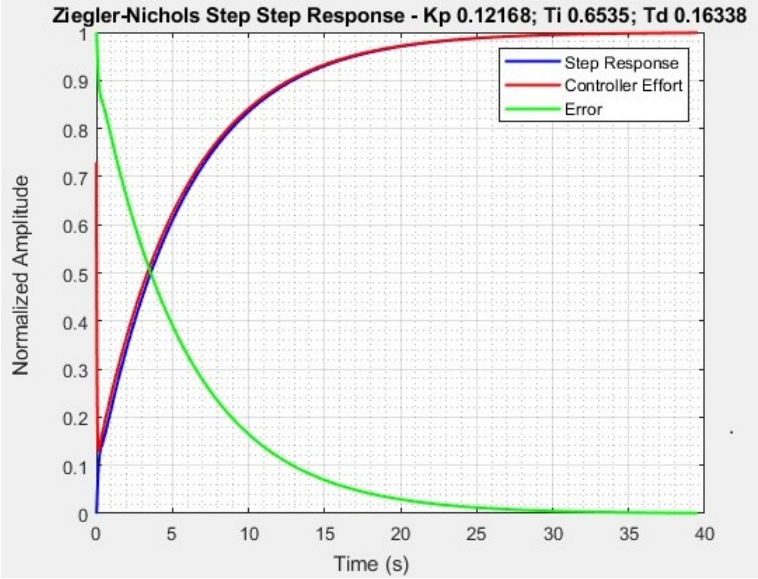


Figure 3.18: Caption

Method Chien-Hrones-Reswick

The Chien-Hrones-Reswick method was also calculated with the same parameters found in the Ziegler-Nichols method, and the result of the PID parameters was the following:

$$K = 0.096475005745226$$

$$Ti = 0.784202672901603$$

$$Td = 0.137235467757781$$

The system response with these parameters is plotted in Figure 3.20.

As the Ziegler Nichols Reaction Curve methor, this also have the reaction very slow, but does not pass the limit of the controller effort. Also, this is not a good solution for the sewing machine because of the time that it takes to accelerate.

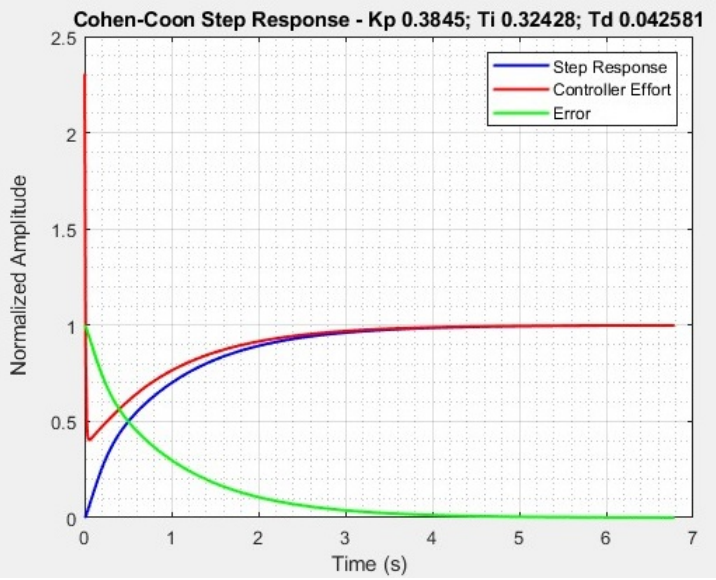


Figure 3.19: Cohen-Coon open-loop rules response

3.2.2 Genetic Algorithm

The genetic algorithm was implemented in MATLAB, using the function *ga*. The standard parameters of the genetic algorithm are described in the documentation of the function [26] and are shown in the following table:

Parameter	Value
Crossover Function	Scattered Crossover — also kown as Uniform Crossover, as in the section 2.3.1
Population Creation	Creates a random population, with uniform distribution
Limit of Generations	100 generations.
Mutation Function	Gaussian Mutation - witch applies a random mutation in the population with mean 0.
Population Size	50

There are others parameters, but they do not influence in this work.

Fitness Function

As this problem is a multi-objective problem, the fitness function is made of a sum of analysis from many perspectives of the system response. To elaborate the fitness function, will be used the model from the section 3.1.4.

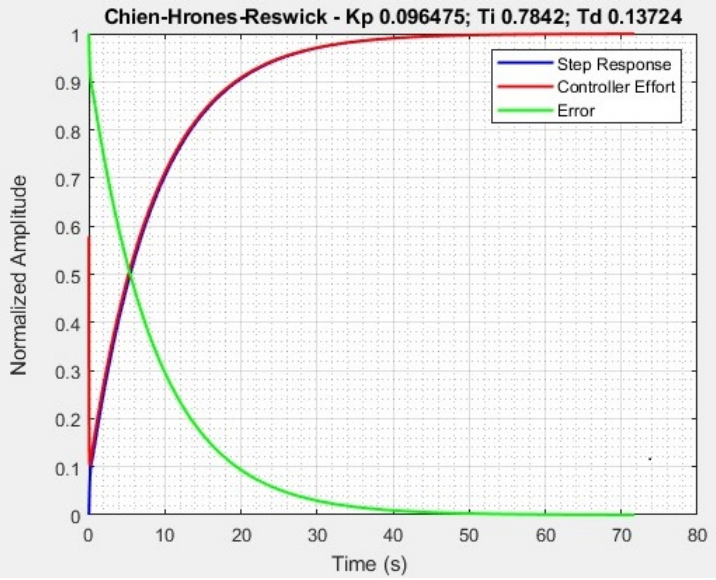


Figure 3.20: Response of the Method Chien-Hrones-Reswick

Overshoot The first objective is have the smallest overshoot from the system. This objective can be measured in two ways: the first way is just find the global maximum of the system response, using the function *max* from MATLAB in the vector of the step response. The other way to measure the overshoot is use the function *stepinfo*, which returns a structure with many performance indicators. As the function *stepinfo* will be useful in others analysis to the final fitness function, the overshoot can be obtained from this function using the command *stepinfo.Overshoot*.

Steady-State Error The second objective is do not have a steady-state error. To do so, it was created a array filled with ones, with the same size of the array from the system response. There are calculated the squared error from the system response to the array with ones after a certain point, to ensure the transitory period is not counted to this function. If the transitory period is counted, would be generated a wrong fitness score.

The code used to make this analysis is in the following listing:

```
1     [step_response_cl,t]=step(H)
2     ideal_response=ones(length(t),1);
3     time_transitory_limit=1;
```

```
4       idx_transitory_limit=find(t==time_transitory_limit);
5       sqr_error=(ideal_response-step_response_cl).^2;
6       sqr_error=sqr_error(idx_transitory_limit:end);
7       mean_error=sum(sqr_error)/length(sqr_error);
```

The final result of the operation will be in the variable *mean_error*, that will be used in the weighted sum.

Controller Effort Signal One of the concerns about the Arduino uses a range from 0 to 255 as output, which corresponds from 0 to 10 volts, the maximum controller effort is imitated to this values. In the simulation, it represents the value with range from 0 to 1.

Thus, was created a function to ensure the maximum value of the controller effort on a step response. To do so, the system was remodeled as Figure 3.21 to find the signal that the controller sends to the plant.

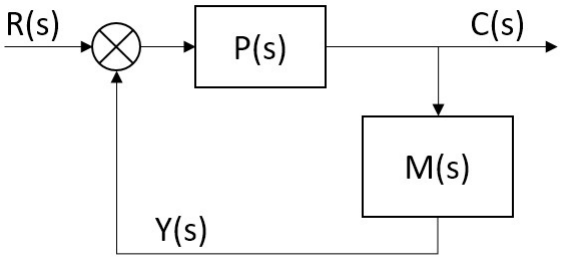


Figure 3.21: Block Diagram of the system to find the controller effort

Based on this configuration, the transfer function of this system is

$$\frac{C(s)}{R(s)} = \frac{P}{1 + PM} \tag{3.5}$$

Where P is the controller and M is the plant.

To compute this objective, was used the command *lsim*, which calculates the system response to an arbitrary input. Then, was used the step response to calculate the error to use it as the input to the system, and the transfer function to be calculated is the $P(s)$, as the block diagram in Figure 3.21.

The syntax used in the MATLAB was

```
step_response_cl=step(closed_loop,t);
error=1-step_response_cl;
[controller_effort,t,x]=lsim(P,error,t);
```

As result, it returns the *controller_effort* array with the controller effort, the t array with the time os simulation and the vector of state space x, which is not used in this solution.

To validate this function, was created a simulation in the *Simulink*, in the MATLAB to compare the results. The block diagram of the *Simulink* is shown on Figure 3.22. Both simulations was using the same parameters $K = 1$; $Ti = Td = 0.1$.

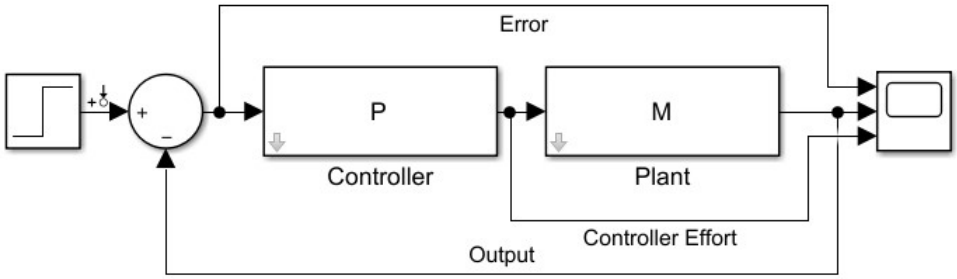


Figure 3.22: Block Diagram used in the Simulink

The result of the *Simulink* simulation and the code simulation are shown in Figures 3.23 and 3.24, respectively.

With this results, it is possible to confirm that the controller effort in the code is equivalent to the *Simulink* response.

Thus, being assure that the controller effort is being calculated correctly, the fitness function is calculated only finding the highest value of the array *controller_effort* and subtracting one. If the maximum signal effort is one, the output to the fitness function is zero, and it is the best value to this function.

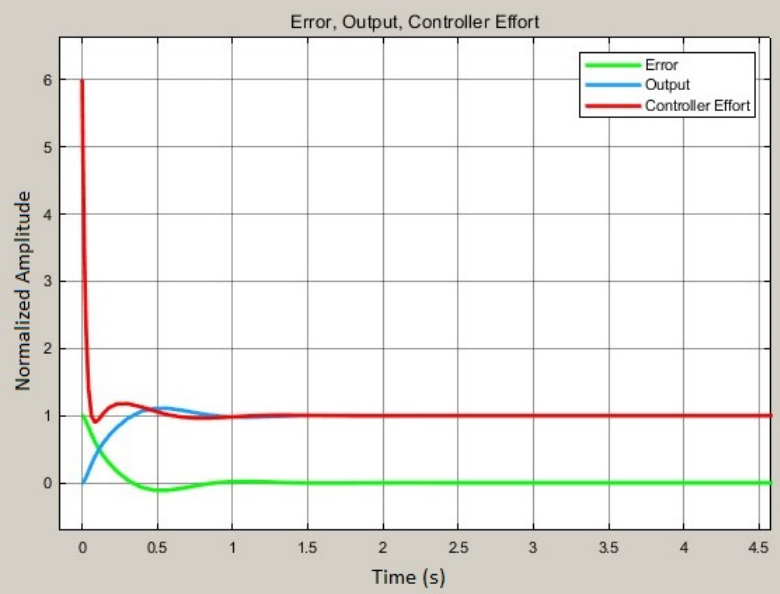


Figure 3.23: *Simulink* responses

Bandwidth The bandwidth of the closed loop system is an other parameter which can be used to check how good the system is. The limitation of the bandwidth means that the controller will work perfectly if the input is inside of the bandwidth, but if the controller is out of the bandwidth, the response will be distorted.

But, instead of the analysis of a communication system, where the bandwidth limit is a problem, in the control system the bandwidth limit can be used to remove the high frequency responses, which represent the fast change responses. It will work exactly as a filter.

As the application of this system will be used to transport a human, it is important move the person with most delicate movements, to prevent physiological damages.

To do so, it can be applied a $2Hz$ of limit on the bandwidth, which is computed using the MATLAB function *bandwidth(transfer_function)*, which returns a scalar with the bandwidth of the system.

Analysis of the Time Values Also had a parameters, called in the code as *delta_T*, which is the difference between the Rise Time and the Setteling Time. It was used as an attempt to avoid an anomalous rising curve. This anomaly was a "hump" that has

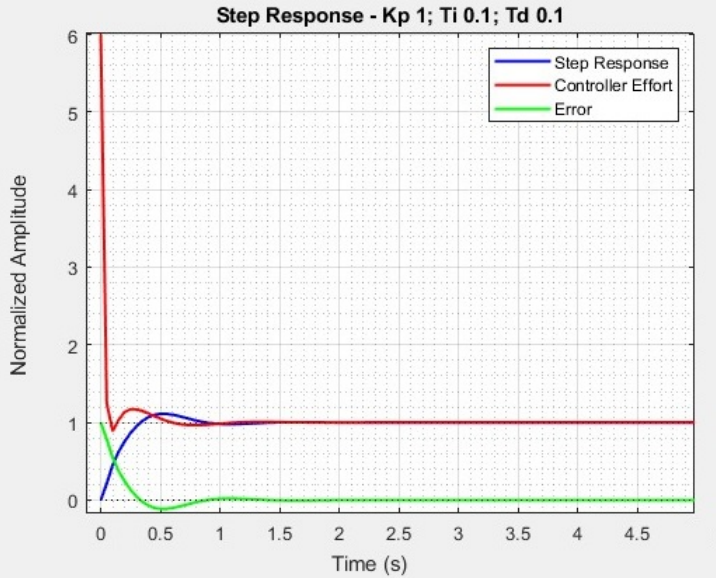


Figure 3.24: MATLAB code responses

occured in some cases, when the algorithm found a solution that has a good fitness score in all others parameters, but was not the desired response, as in Figure 3.25.

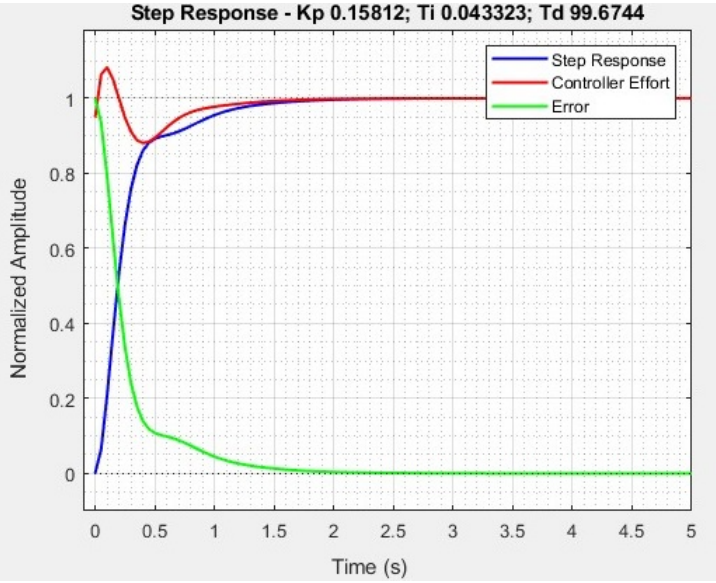


Figure 3.25: Example of the undesired hump

This "hump" is not a problem, but the most desired solution is a clean and straight rising curve.

This function also can be used to set the rise time from the system. In all tests, it was

used 1 second as default.

Implementation of the Genetic Algorithm The implementation of the Genetic Algorithm was using the function *ga*, using the minimal parameters, without constraints. The problem is because of the random creation of the population, some individuals was being created with negative gains, which will lead to a wrong response. To fix it, was made the constraint from 0.001 to 100 to K, Ti and Td.

With this constraints, the algorithm ran as the expected.

Also, this program to run the genetic algorithm was modified to be used as a function, so it became easily tested, using the weights of the final fitness sum as input arguments.

Also was added a code to save the input parameters, the final fitness achieved and the result of the genetic algorithm, K, Ti and Td.

As the genetic algorithm requires a lot of processing, a single run was lasting over 800 seconds. To reduce this time, was also changed the option of the genetic algorithm to use the parallel processing methods. With this change, the time of processing was reduced to around 250 seconds, each test.

Thus, the final code was the following:

```
function [x,fval] = alg_gen2(k_deltaT,k_overshoot,...
k_erro,k_bw,k_ctr)

format long;
rng default;

fun=@(coef)fitness_pid(coef,k_deltaT,k_overshoot,...
k_erro,k_bw,k_ctr);
nvars=3;
lb=[0.001 0.001 0.001];
ub=[100 100 100];

options = optimoptions('ga','UseParallel', true,...
'UseVectorized', false);

tic;
[x,fval] = ga(fun,nvars,[],[],[],[],lb,ub,[],[],options);
```

```
18      toc;
19
20      agora=datestr(now,'mm_dd_HH_MM_SS');
21      caminho='testesGA3\';
22      arq=strcat(caminho,agora);
23      save(arq,'x','fval','k_deltaT','k_overshoot',...
24      'k_erro','k_bw','k_ctr');
25
26      end
```

Genetic Algorithm Result The best result found with the Genetic Algorithm was found adjusting manually the weights of the fitness sum. The best combination of weight is in the table 3.3.

Weight	Value
$T_{settling} - T_{rise}$	5
Overshoot	0.3
Steady-State Error	100
Bandwidth	0
Controller Effort	1.8

Table 3.3: Best weights found manually to the Genetic Algorithm

And the result of this search with the Genetic Algorithm is

$$K = 0.169022678510371;$$

$$Ti = 0.072109163345392;$$

$$Td = 0.091354695916176;$$

The step response of the system with these parameters is on Figure 3.26.

Parameter	Value
RiseTime:	0.698112681810336
SettlingTime:	1.017660871135156
SettlingMin:	0.902733496897721
SettlingMax:	1.011839512145476
Overshoot:	1.183951214547574
Undershoot:	0
Peak:	1.011839512145476
PeakTime:	1.402968871098222

Table 3.4: Results from the step response of the manual search with the Genetic Algoritm

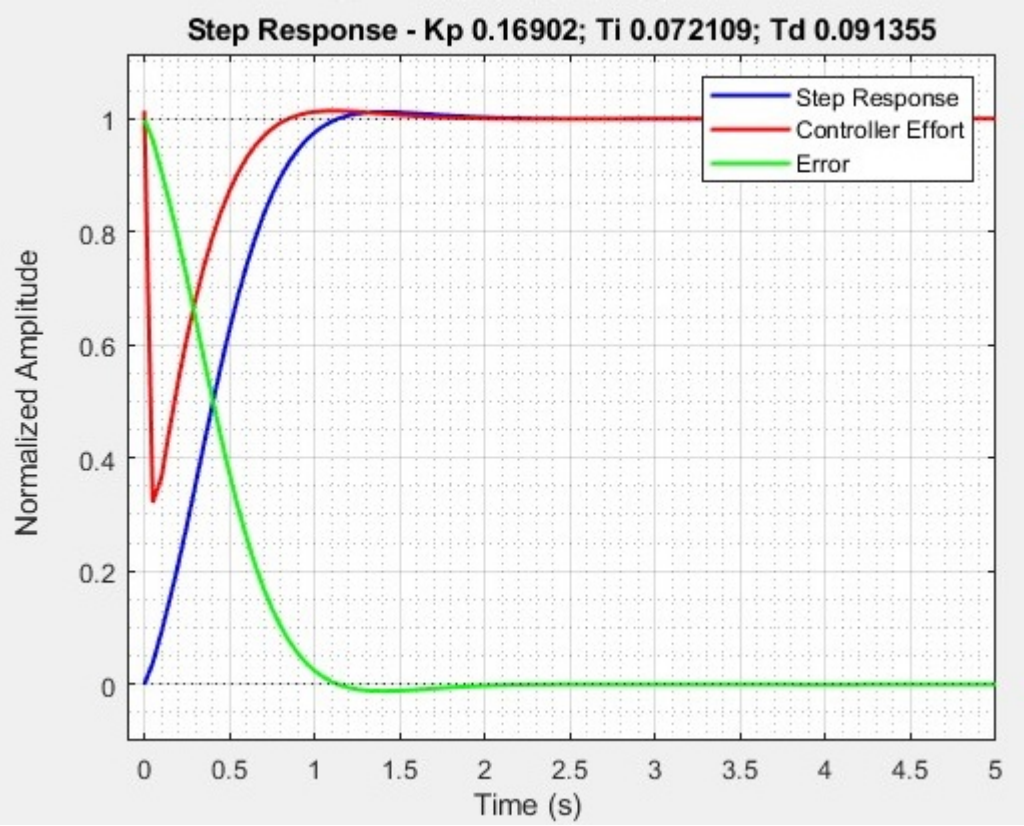


Figure 3.26: Best Result found with the Genetic Algorithm adjusting the parameters manually with the Genetic Algoritm

The characteristics of the step response are in the table 3.4.

After find this result, was made a search varying the parameters values around the best values to check if there are some best result nearby, changing the input parameters individually. The result from this search is in the appendix, and the best result found

was the following:

$$K = 0.167598529831592;$$

$$Ti = 0.072333574832971;$$

$$Td = 0.012110708117485;$$

This result was found using the input weight values from the table 3.5.

Weight	**Value**
$T_{settling} - T_{rise}$	3
Overshoot	0.3
Steady-State Error	100
Bandwidth	0
Controller Effort	1.8

Table 3.5: Best weights found with a neighbour search to the Genetic Algorithm

Using these parameters, the step response from the system is represented on Figure 3.27.

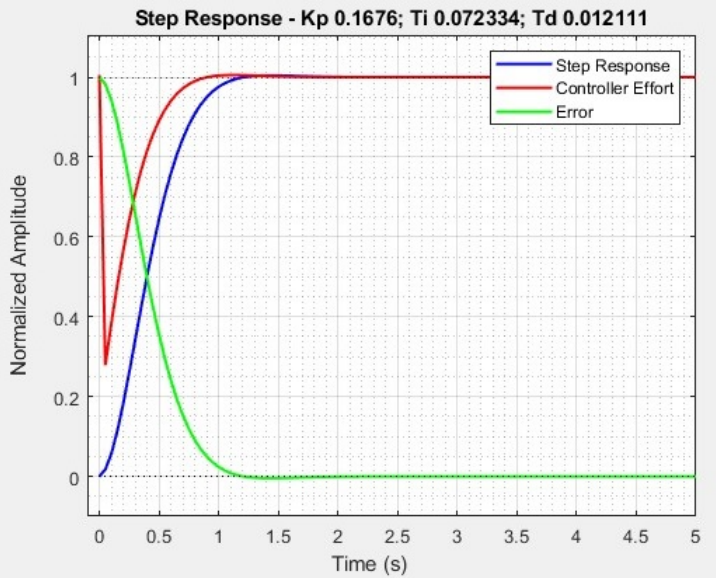


Figure 3.27: Best result found with the Genetic Algorithm, after the nearby search

The results of this step response are in the table 3.6.

Parameter	Value
RiseTime:	0.643727208289148
SettlingTime:	1.019770923961849
SettlingMin:	0.902619446088921
SettlingMax:	1.004549552630949
Overshoot:	0.454955263094869
Undershoot:	0
Peak:	1.004549552630949
PeakTime:	1.425543595367601

Table 3.6: Results from the step response of the nearby parameters search with the Genetic Algorithm

Both results are very similar, but the second result have a lower overshoot. Thus, these are the best parameters to the system.

On the appendix will have more simulations with other parameters, weights and other genetic algorithm functions (crossover, mutation).

3.2.3 Particle Swarm Optimization

The algorithm of the particle swarm was implemented exactly as the genetic algorithm was implemented. It was used the function *particleswarm*, with the same constraints and options.

With the particle swarm optimization, was found a result that is more faster than the result from the Genetic Algorithm. It can be useful if it is needed a faster response from the system, but it can be aggressive to the user who is sewing. Also have an other response found that is similar to the response found from the genetic algorithm.

The first result was found with the weights from the table 3.7

With these input, the result from the algorithm was

$$K = 0.091910134764164$$

$$Ti = 0.023054719862569$$

$$Td = 12.152111935926770$$

Weight	Value
$T_{settling} - T_{rise}$	5
Overshoot	0.3
Steady-State Error	100
Bandwidth	0
Controller Effort	7

Table 3.7: Fastest response found with the Particle Swarm Optimization

The step response with these gains is being represented in Figure 3.28.

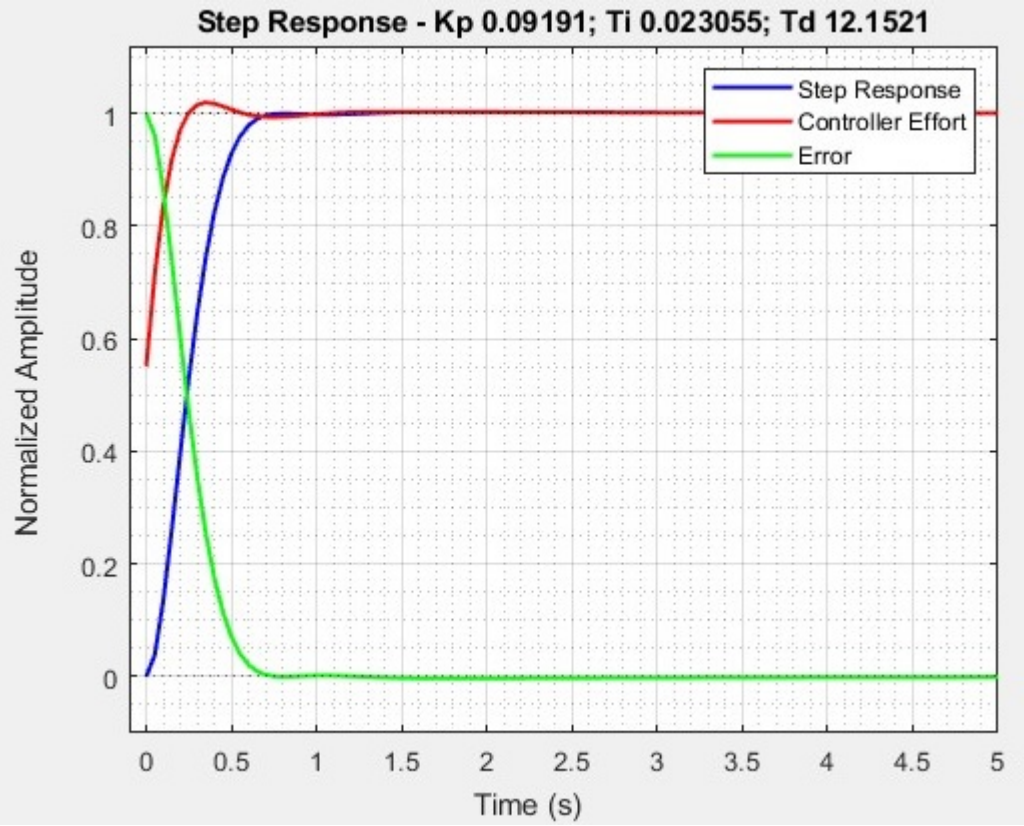


Figure 3.28: Step Response found with the Particle Swarm Optimization

The information from this step response are shown in the table 3.8.

This result have a faster Settling Time and keep the steady-state almost zero.

An other result found can be used, and it is very similar to the result found with the Genetic Algorithm. It was found using the weights in the table 3.9.

Parameter	Value
RiseTime:	0.380878399251545
SettlingTime:	0.604543733807215
SettlingMin:	0.901767659976360
SettlingMax:	0.999999986390869
Overshoot:	0
Undershoot:	0
Peak:	0.999999986390869
PeakTime:	0.806063641747438

Table 3.8: Information from the Step Response from the fastest response found with the Particle Swarm Optimization

Weight	Value
$T_{settling} - T_{rise}$	0.5
Overshoot	0.3
Steady-State Error	100
Bandwidth	0
Controller Effort	1.8

Table 3.9: Weitghs used in the PSO to find the result similar to the result found in the Genetic Algorithm

The PID gains foud using these parameters was

$$K = 0.166668533460323$$

$$Ti = 0.076890007199342$$

$$Td = 0.001000253319740$$

Which the K and Ti are quite similar as the values found on the Genetic Algorithm, but the Td is ten times smaller.

The step response with these parameters is shown in Figure 3.29.

The details from this step response is shown in the table 3.10.

With this results, any of the four sets of PID gains can be used in the system, which will lead to adequate response.

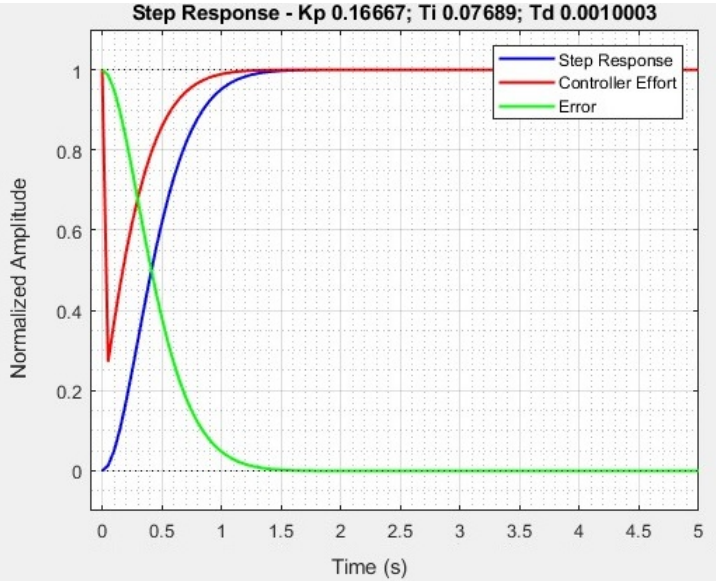


Figure 3.29: Particle Swarm result similar to the Genetic Algorithm result.

Parameter	Value
RiseTime:	0.697290763116349
SettlingTime:	1.166026956377534
SettlingMin:	0.903726189348649
SettlingMax:	1.000005487102953
Overshoot:	5.487102953116718e-04
Undershoot:	0
Peak:	1.000005487102953
PeakTime:	1.851049887812188

Table 3.10: Parameters from the step response with the gains found with the Particle Swarm.

3.3 Implementation

As the PID parameters was already found, the code was implemented in the Arduino, using a trapezoidal approximation to the PID Controller. The chosen PID gain was the second of the Genetic Algorithm, which have a smaller overshoot.

The pseudo-code of the controller used is in the following listing (the complete version is in the appendix), and uses the trapezoidal approximation of the PID, as explained in the Section 2.3.3.

```
double K=0.167598529831592; //Controller Gain
double Ti=0.072333574832971; //Integral Gain
double Td=0.012110708117485; //Derivative Controller
double Ts=0.1; //Sampling Time;
double N=5; //Filter Coefficient
//Parameter Calculation
double a_i = K * Ts / Ti;
double b = Td / (Td + N);
double b_1 = (K * Td * (1 - b))/Ts;

double u_p_k, u_i_k, u_i_k_1,a_1 =0;
double u_d_k, u_d_k_1, y_k, y_k_1 =0;
controller{
    //Proportional
    u_p_k = K * e_k;
    //Integral
    u_i_k = u_i_k_1 + a_i * e_k;
    //Derivative
    u_d_k = b * u_d_k_1 - b_1 * (y_k - y_k_1);
    //Final
    u_k = u_p_k + u_i_k + u_d_k;
    if (u_k < 0) u_k = 0;
    else if (u_k > 255) u_k = 255;
    //Update of the variables
    u_i_k_1=u_i_k;
    u_d_k_1=u_d_k;
    y_k_1=y_k;
}
```

To measure the accuracy of the simulation, it should be used a step response, the check the system. But, as the system input is the sewing machine, which have its own dump, it is impossible to do an ideal step response with it.

To solve this problems, there are two alternatives: the first is use a virtual step response, using a button, or a programmed trigger, and send the virtual step as the input to the system and measure the motor response. The second is record both the input from the sewing machine and the output from the motor, and use the sewing machine input to calculate the simulation, and compare the results.

As the second option is more fast to accomplish, and it is a real performance comparison, it will be the option to measure the system.

To do so, the sewing machine was accelerated (as a user would do, pressing the pedal) and the speed of the machine and the speed of the motor was recorded, using the gains from the system.

The results from this procedure are shown in the Figure 3.30.

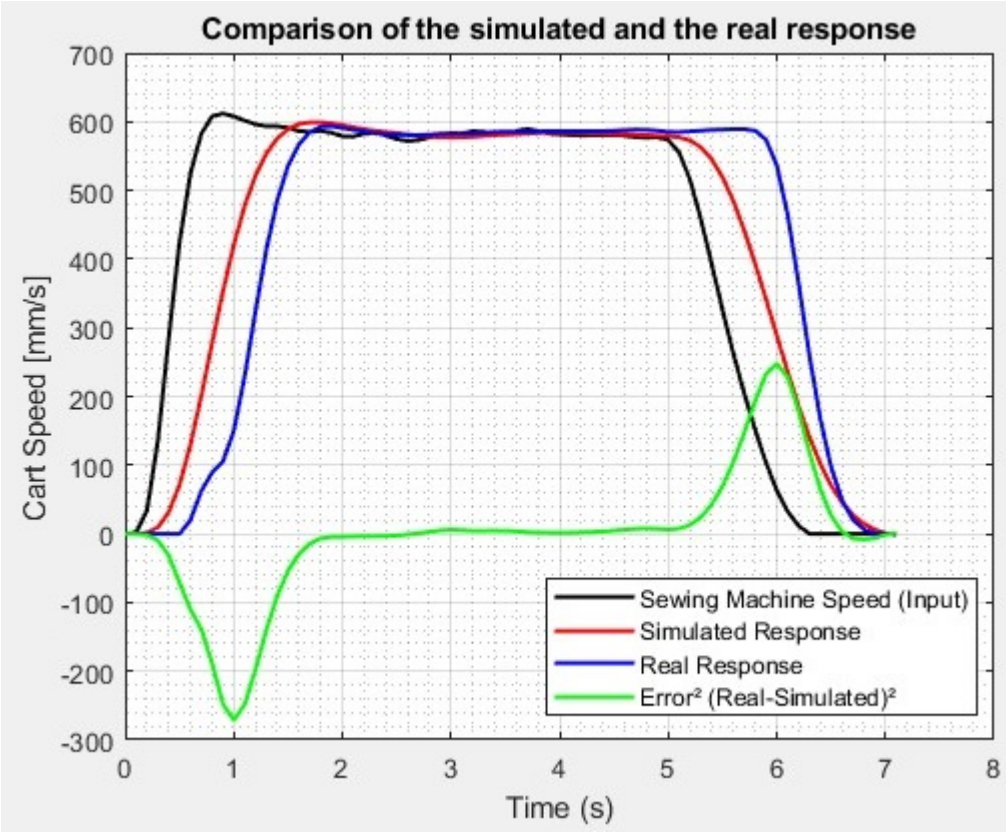


Figure 3.30: Real results using the PID parameters with the trapezoidal approximation

As shown on Figure 3.30, the system have a delayed response in comparison with the simulated response. It can be caused by the digital version of the PID algorithm, that uses the trapezoidal approximation. It also uses an different approach to the derivative action, which can causes this difference. Figure 3.31 is the plot of the absolute error from the difference of the simulation and the real test.

The problem is more evident in this graphic: in the transient period, happens the

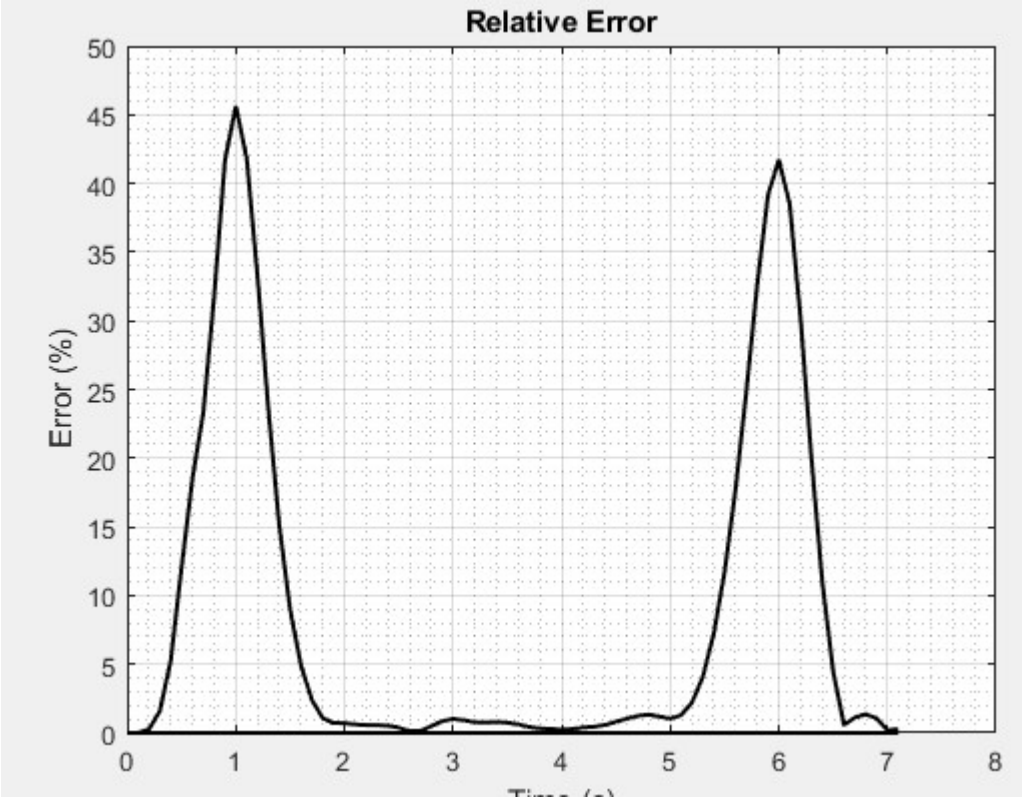


Figure 3.31: Difference from the simulation and the real test

peak ah 45% under the real value.

A tentative of solution was use the error as input to the derivative action instead the output of the system, as proposed on the theoretical background. It may cause the error from the system on the transient response, as the system was projected on the evolutionary algorithms was used the error as input to the derivative action.

Figure 3.32 shows the system response with the error as input to the derivative action.

And the difference of the real and the simulation is represented on Figure 3.33.

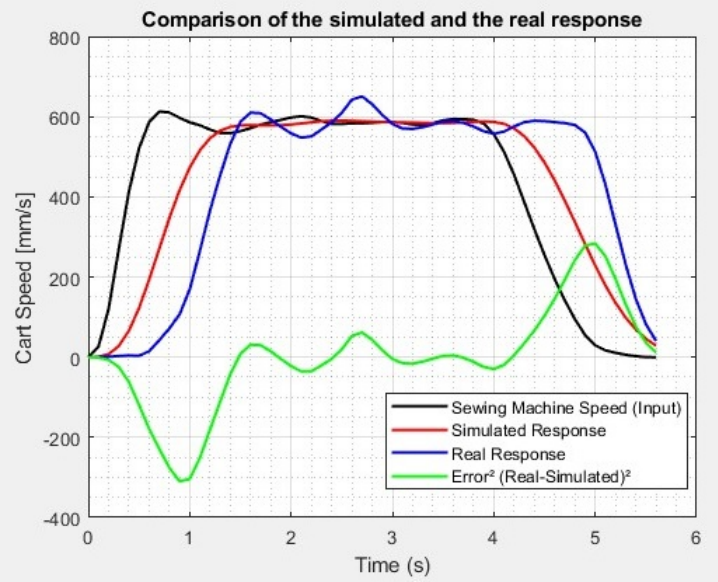


Figure 3.32: Real test with the trapezoidal approximation, using the error as input to the derivative action

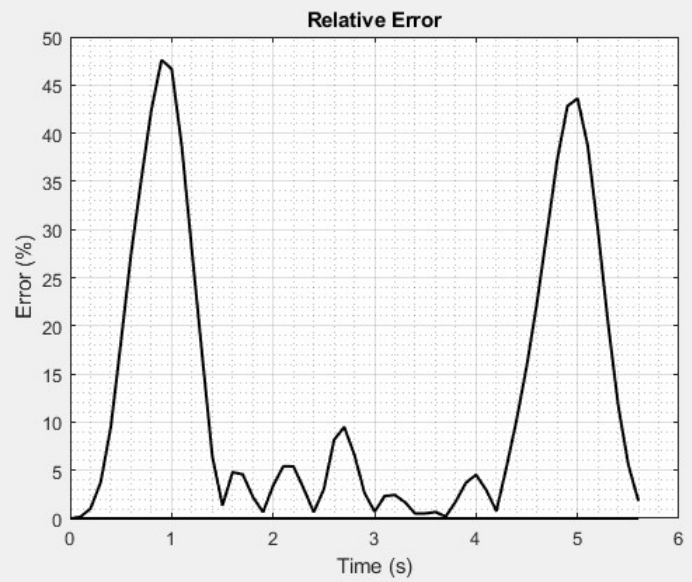


Figure 3.33: Difference of the simulation and the real test using the error as input to the derivative action

As Figure 3.33 shows, comparing with Figure 3.31, the results was worse than using the model using the output as input to the derivative action.

Thus, will be used the design of the controller using the z-transform to find the difference equation.

Thus, was designed a PID controller using the z-transform to find the difference equation to implement in the microcontroller. The code of this program is available in the appendix.

Using the same PID parameters, the result of the step response is represented in Figure 3.34, with the same information as the tests from the trapezoidal approximation.

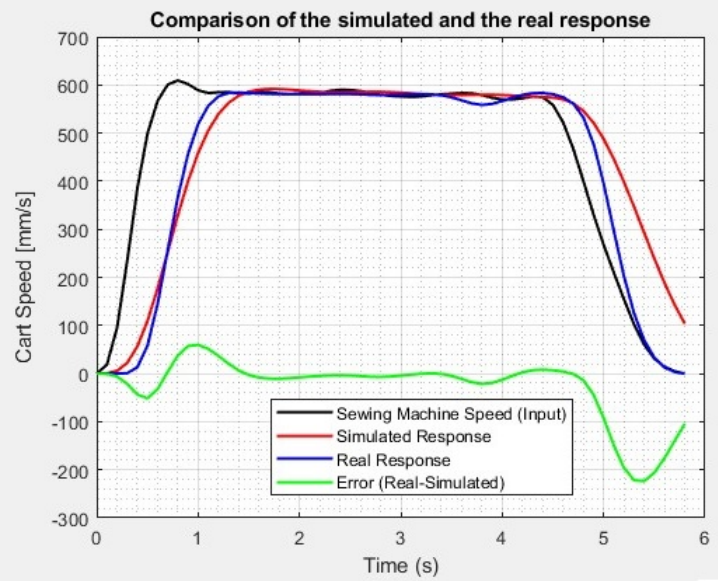


Figure 3.34: Real system response using the discrete controller

The error from the simulation is represented in Figure 3.35

The behaviour of the rise have a smaller error in comparison with the trapezoidal approximation controller, but in the fall, the error still high. Also, in the steady-state have more noise, or a more susceptible to disturbance controller.

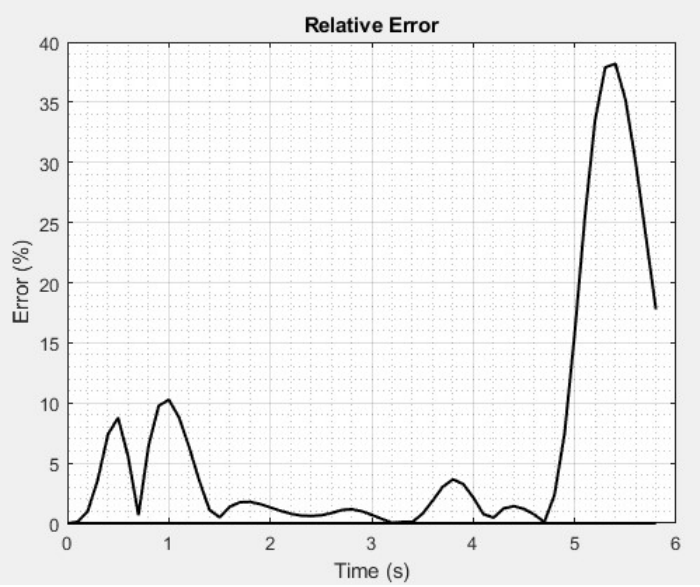


Figure 3.35: Error of the comparison of the real test and the simulation

Chapter 4

Results and Discussion

With all algorithms and tests implemented, it is possible to have some discussion about the results.

4.1 Mathematical Modelling

The mathematical modelling of the system with a first-order model was not a good response, specially in the transient period. It has a large difference from the real system.

However, the second-order system was a good model, with a good transient response and a small error, with an accurate of 97.6

This model was used to all simulations and in the evolutionary algorithms.

4.2 Classical PID parameter Models

None of the classical PID calculations obtained was a good response for the system. It happens because these models only have focus on a single objective of the system: have a small overshoot, or have a fast response. Factors as the controller effort are never taken in to account on these methods.

The Ziegler-Nichols Reaction Curve Process and the Chien-Hrones-Reswick method had a slow response, that was not a good solution for the sewing station.

The Cohen-Coon open-loop rules had a good response, but the controller effort was high on the begin of the step response. As the controller effort is limited on the sewing station, this solution will not happen if it is implemented on the system.

Therefore, all classical methods are not a good solution for this problem.

4.3 Genetic Algorithm

The results found with the Genetic Algorithm was a really good solution for this problem, as seen in the section 3.2.2. As the multi-objective genetic algorithm tries to accomplish all problems, it was only necessary do keep adjusting manually the parameters to try to find the best solution for the problem.

As shown on Figure 3.26 and 3.27, this solution might not be the best solution of all solution-space, but as this is a solution that solves the problem and satisfies all requirements, it is a good solution.

4.4 Particle Swarm Optimization

The Particle Swarm Optimization had different PID values from the Genetic Algorithm, as shown on Table 3.8 and Figure 3.28. But it also solves the problem and satisfies all requirements. Thus, the solution found was a good solution.

4.5 Real Implementation

The real implementation has two tests, using the trapezoidal approximation and the bilinear approximation. Both results can be implemented in the microcontroller, that would have a nice response.

The trapezoidal approximation using the output to calculate the derivative action has a better steady-state performance, but have a higher error in the transient period.

The trapezoidal approximation using the error to calculate the derivative action has a similar response in the transitory period, but have the steady-state more unstable.

The bilinear approximation have the best transient response in comparison with the others, and have the steady-state response with a lot of disturbance.

With these results, the best choice to implement in the machine is the trapezoidal approximation using the output to calculate the derivative action. As it have less disturbance, it is more comfortable to the user. As the user will stand on the sewing station for hours, it can cause physiological problems.

Chapter 5

Conclusion and Future Work

Analysing the results, the best result was achieved using the trapezoidal approximation using the output to calculate the derivative action. This system have a good response, a good stability in the steady state and it can be used in the industry.

To the future works, this system was not implemented in the industry. It needs to be tested, and verify its performance, check the feedback from the users and check the integrity over the time. In the mathematical modelling, it can be revised to check if the system had no changes over the time, for example. The wear of the mechanical parts can cause a more dumped response from the motor, that will cause a wrong system response.

The PID project using the genetic algorithm can be modified to apply some algorithms of optimization in the multi-objective function, to ensure the result is the global minimum of the space of results.

The genetic algorithm can also be upgraded to have a better control of the rise time. In this algorithm, it was fixed to one second, that was a good time response. It was not desired a fastest response because it can lead to a high acceleration of the user, but if became necessary a different time response, it would be useful to modify it.

To verify others approaches to this solution, wold be interesting to check some other controller algorithms, as fuzzy-logic controller, and robust control techniques to compare the results to this work.

Bibliography

[1] J. P. Coelho, P. Santos, T. M. Pinho, J. Boaventura-Cunha, and J. Oliveira, "Instrumentation and control of an industrial sewing station", *2018 13th APCA International Conference on Control and Soft Computing (CONTROLO)*, 2018. DOI: 10.1109/controlo.2018.8514280.

[2] D. Gershin and I. Porat, "Vision servo control of a robotic sewing system", *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, Apr. 1988.

[3] H. Carvalho, F. Soares, L. F. Silva, and F. Guhr, "Adaptive control of an electromagnetically actuated presser-foot for industrial sewing machines", *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*, Nov. 2010.

[4] J.-Y. Lee, D.-H. Lee, J.-H. Park, and J.-H. Park, "Study on sensing and monitoring of sewing machine for textile stream smart manufacturing innovation", *2017 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, Nov. 2017.

[5] D. Gershon, "Parallel process decomposition of a dynamic manipulation task: Robotic sewing", *IEEE Transactions on Robotics and Automation (Volume: 6 , Issue: 3 , Jun 1990)*, Jun. 1990.

[6] D. Triantafyllou, P. N. Koustoumpardis, and N. A. Aspragathos, "Model reference fuzzy learning force control for robotized sewing", *2011 19th Mediterranean Conference on Control & Automation (MED)*, Jun. 2011.

[7] K. Mani, D. Raja, and S. S. Suresh, "Development of assistive technology to operate industrial grade sewing machines for differently-abled persons", *Indian Journal of Fibre & Textile Research Vol 44, pp. 314-320*, Sep. 2019.

[8] C. H. Houpis and G. B. Lamont, *Digital Control Systems: Theory, Hardware, Software*, 2nd. The McGraw-Hill Companies, Inc., 1992, ISBN: 0070305005.

[9] H. V. Vu and R. S. Esfandiari, *Dynamic Systems: Modeling and Analysis*, 1st. The McGraw-Hill Companies, Inc., 1997, ISBN: 0070216738.

[10] K. Ogata, *Modern Control Engineering*, 4th. Upper Saddle River, New Jersey, U.S.A.: Prentice-Hall, Inc., 2002, ISBN: 0130432458.

[11] C. L. Phillips and R. D. Harbor, *Feedback Control Systems*, 4th. Upper Saddle River, New Jersey, U.S.A.: Prentice-Hall, Inc., 2000, ISBN: 0130161241.

[12] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers", *Transactions of the A.S.M.E, pp. 759-768*, Nov. 1942.

[13] B. Shahian and M. Hassul, *Control System Design using MATLAB ®*. 1st. Englewood Cliffs, New Jersey 07632, United States of America: Prentice-Hall, Inc., 1993, ISBN: 0130145572.

[14] W. Y. Svrcek, D. P. Mahoney, and B. R. Young, *A Real-Time Approach to Process Control*. West Sussex, England: John Wiley & Sons, Ltd., 2000.

[15] L. Maret, *Régulation Automatique*. EPFL-Ecublens, CH-1015, Suisse: Presses Polytechniques Romandes, 1987.

[16] S. Russell and P. Norvig, *Artifical Intelligence: A Modern Approach*, 1st. Upper Saddle River, New Jersey 07458, United States of America: Prentice-Hall, Inc., 1995, ISBN: 0131038052.

[17] D. Floreano and C. Mattiussi, *Bio-Inspired Artificial Intelligence: Theories, Methods and Technologies*, 1st. Cambridge, MA, U.S.A.: The MIT Press, 2008, ISBN: 9780262062718.

[18] M. Gen and R. Cheng, *Genetic algorithms and engineering optimization*. John Wiley & Sons, Inc., 2000.

[19] O. Kramer, *Genetic Algorithm Essentials*. Springer International Publishing, 2017.

[20] M. Gen, R. Cheng, and L. Lin, *Network Models and Optimization: Multiobjective Genetic Algorithm Approach*. Springer-Verlag London Limited, 2008, ISBN: 9781848001800.

[21] R. C. Eberhart and Y. Shi, "Particle swarm optimization: Developments, applications and resources", *Proc. of the 2001 Congress on Evolutionary Computation*, 2001.

[22] K. E. Parsopoulos and M. N. Vrahatis, *Particle Swarm Optimization and Intelligence: Advances and Applications*, 1st. Hershey, PA, U.S.A.: Information Science Reference, IGI Global, 2010.

[23] G. Olsson and G. Piani, *Computer Systems for Automation and Control*. Trowbridge, Wiltshire, UK: Prentice Hall International Ltd, 1992, ISBN: 0134575814.

[24] Atmel, *Atmega328p datasheet*, `http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf`, 2015.

[25] S. Inc., *Vn2410 datasheet*, `http://ww1.microchip.com/downloads/en/DeviceDoc/VN2410%20C081913.pdf`.

[26] I. The MathWorks, *Find minimum of function using genetic algorithm*. [Online]. Available: `https://www.mathworks.com/help/gads/ga.html`.

Appendix A

Appendix: Codes

In this appendix will be shown the main programs used on this project. All codes used in this project are available on the Github repository at the following adress:

https://github.com/morettieric/SewingStation

A.1 Matlab Codes

The main Matlab codes will be listened here. Some files have a second version (e.g. alg_gen2) and the previous versions (e.g. alg_gen) are available on the Github, at the section dump.

A.1.1 run.m

This is the code used to run the genetic algorithm and the particle swarm optimization. It can be used in a *for* loop to check the results of the weights of the multi-objective function. The genetic algorithm can be run directly on the prompt of the Matlab, but using it on a code editor is easily to check all the space of results.

```
clear all;
close all;
clc;
tic;
```

```
alg_gen2(5,0.3,100,0,1.8);
pso2(0.5,0.3,100,0,1.8);
toc;
```

A.1.2 alg_gen2.m

This program is the core program of the genetic algorithm. It uses the weights of each parameter of the weighted sum, and it returns the PID coefficients and the fitness result of the genetic algorithm.

```
1  function [x,fval] = alg_gen2(k_deltaT,k_overshoot,k_erro,k_bw,k_ctr)
2
3  format long;
4  rng default;
5
6  fun=@(coef)fitness_pid3(coef,k_deltaT,k_overshoot,k_erro,k_bw,k_ctr);
7  nvars=3;
8  lb=[0.001 0.001 0.001];
9  ub=[100 100 100];
10 options = optimoptions('ga','UseParallel', true, 'UseVectorized', false);
11 tic;
12 [x,fval] = ga(fun,nvars,[],[],[],[],lb,ub,[],[],options);
13 toc;
14
15 agora=datestr(now,'mm_dd_HH_MM_SS');
16 caminho='testesGA3\';
17 arq=strcat(caminho,agora);
18 save(arq,'x','fval','k_deltaT','k_overshoot','k_erro','k_bw','k_ctr');
19 tsc(x);
20 end
```

In the line 8 and 9 are set the upper and lower constraints to the genetic algorithm. It was set with these values because of the behaviour of the genetic algorithm without them. Whithout, the derivative action was set as zero, and the values of the proportional and integral actions was over one thousand.

In the line 10, was only used set the parameters to use the parallel processing, to achieve the results faster. Each run had from 150 to 800 seconds of processing using the

parallel processing.

The lines 15 to 18 are used to save the results in a file.

The function *tsc(x)*, in the line 19 calculates the results of the PID parameters found of the genetic algorithm result. It is also explained in this section.

A.1.3 fitness_pid3.m

This function is the fitness function of the genetic algorithm.

```
1  function fitness_final = fitness_pid(coef,k_deltaT,k_overshoot,k_erro,k_bw,k_ctr)
2
3  K=coef(1);
4  Ti=coef(2);
5  Td=coef(3);
6  N=5;
7  Tf=Td/N;
8  %valores do sistema
9  zeta = 0.900733958553509;
10 wn=8.036118730219977;
11 k=1;
12 t=0:0.005:5;
13
14 s=tf('s');
15
16 %sistemas
17 M=(wn^2)/(s^2+2*zeta*wn*s+wn^2);
18 P=K*(1+(1/(Ti*s))+(Td*s/(1+Tf*s)));
19 malha_aberta=P*M;
20 malha_fechada=(P*M)/(1+P*M);
21
22 %respostas
23 resp_deg_mf=step(malha_fechada,t);
24 erro=1-resp_deg_mf;
25 [sinal_controle,t,x]=lsim(P,erro,t);
26
27 %valor máximo do esforço do controlador
28 sinal_controle_max=(max(abs(sinal_controle))-1);
29 if sinal_controle_max<0
30     sinal_controle_max=0;
31 end
```

```
32
33 % calculo do erro quadrático
34 % limite_transitorio = o tempo desejado para o tempo transitorio ideal
35 % em segundos.
36 resposta_ideal=ones(length(t),1);
37 limite_transitorio=2;
38 idx_limite_transitorio=find(t==limite_transitorio);
39 erro_quad=(resposta_ideal-resp_deg_mf).^2;
40 erro_quad=erro_quad(idx_limite_transitorio:end);
41 erro_med=sum(erro_quad)/length(erro_quad);
42
43 %overshoot
44 info_step=stepinfo(malha_fechada);
45 oversh=info_step.Overshoot;
46
47 %diferença entre o RiseTime e o Settlingtime, para evitar o pico
48 %indesejado.
49 delta_T=info_step.SettlingTime-info_step.RiseTime;
50 if isnan(delta_T)==1
51     delta_T=0;
52 end
53
54 %bw do controlador obtido; ideal = Hz
55 bw=bandwidth(malha_fechada);
56 if isnan(bw)==1 %|| bw>100
57     bw=10000;
58 else
59     %o ideal é que bw=2, ou seja, se bw=2, o erro é zero.
60     bw=bw-2;
61 end
62
63 fitness_final=k_overshoot*oversh+k_erro*erro_med+k_bw*bw+k_ctr*sinal_controle_max+k_deltaT*delta_T;
64 %fprintf('Fit:%.2f\tOS: %.2f\tEM: %.2f\tBw: %.2f\tSC: %.2f\tDT: %.2f\tK: %.2f\tTi:
   ↪  %.2f\tTd: %.2f\n',fitness_final,k_overshoot*oversh, k_erro*erro_med, k_bw*bw,
   ↪  k_ctr*sinal_controle_max,k_deltaT*delta_T,K,Ti,Td);
65 end
```

The line 64 is used to show the fitness function and the all the parameters of the PID during the processing of the genetic algorithm, but it takes the processing long.

A.1.4 pso2.m

This code is used as the main code to the Particle Swarm Optimization. It is very similar to the *alg_gen2.m*.

```
function [x,fval] = pso2(k_deltaT,k_overshoot,k_erro,k_bw,k_ctr)

format long;
rng default;

fun=@(coef)fitness_pid3(coef,k_deltaT,k_overshoot,k_erro,k_bw,k_ctr);
nvars=3;
lb=[0.001,0.001,0.001];
ub=[100,100,100];
options = optimoptions('particleswarm','UseParallel', true, 'UseVectorized', false);
tic;

[x,fval] = particleswarm(fun,nvars,lb,ub,options)

x
fval
toc;

agora=datestr(now,'mm_dd_HH_MM_SS');
caminho='testesPSO3\';
arq=strcat(caminho,agora);
save(arq,'x','fval','k_deltaT','k_overshoot','k_erro','k_bw','k_ctr');
tsc(x);
end
```

A.1.5 tsc.m

This code is used to evaluate the PID parameters as its inputs.

```
function [y,c,t] = teste_sinal_controle(coef)
K=coef(1);
Ti=coef(2);
Td=coef(3);
N=5;
Tf=Td/N;
```

```
wn=8.036118730219977;
zeta=0.900733958553509;
k=1;
t=0:0.05:5;

s=tf('s');

%sistemas
M=(wn^2)/(s^2+2*zeta*wn*s+wn^2);
P=K*(1+(1/(Ti*s))+(Td*s/(1+Tf*s)));
malha_aberta=P*M;
malha_fechada=(P*M)/(1+P*M);

%respostas
resp_deg_mf=step(malha_fechada,t);
erro=1-resp_deg_mf;
[sinal_controle,t,x]=lsim(P,erro,t);

linha_1=ones(length(t),1);
linha_0=zeros(length(t),1);

%plot da resposta
figure (1);
plot(t,resp_deg_mf,'-b','LineWidth',1.5);
hold on
plot(t,sinal_controle,'-r','LineWidth',1.5);
plot(t,erro,'-g','LineWidth',1.5);
axis([-0.1 5 -0.1 max(sinal_controle)+0.1]);

plot(   t,linha_1,':k',...
        t,linha_0,':k');
grid on
grid minor
legend('Step Response','Controller Effort','Error');
title (['Step Response - Kp ', num2str(K),'; Ti ', num2str(Ti),'; Td ', num2str(Td)]);
hold off

%print dos dados importantes do sistema
%overshoot
info_step=stepinfo(malha_fechada);
oversh=info_step.Overshoot;

%bw
```

```
bw=bandwidth(malha_fechada);

% calculo do erro quadrático
% limite_transitorio = o tempo desejado para o tempo transitorio ideal
% em segundos.
resposta_ideal=1-(exp(-4*t));%ones(length(t),1);
limite_transitorio=1;
idx_limite_transitorio=find(t==limite_transitorio)
erro_quad=(resposta_ideal-resp_deg_mf).^2;
erro_quad=erro_quad(idx_limite_transitorio:end);
erro_med=sum(erro_quad)/length(erro_quad)

%diferença entre o RiseTime e o Settlingtime, para evitar o pico
%indesejado.
delta_T=info_step.SettlingTime-info_step.RiseTime

%sinal max de controle
resp_deg_mf_ctr=step(malha_fechada,t);
erro=1-resp_deg_mf_ctr;
%essa linha abaixo é diferente da função fitness em -1 pois aqui é mais
%interessante saber o valor real do sinal.
sinal_controle_max=max(abs(sinal_controle));

fprintf('Overshoot: %f\tErroMed: %f\tBw: %f\tSinContr: %f\n',oversh, erro_med, bw,
 ↪  sinal_controle_max);
info_step
end
```

A.1.6 classicalMethods.m

This code is used to calculate the PID parameters using the classical methods. It finds the inflection point and the tangent line and calculates the PID parameters of the system in all classical methods of this work.

```
clear all; close all; clc;
t=0:0.05:1;

wn=8.036118730219977;
zeta=0.900733958553509;

```

```
s=tf('s');
M=(wn^2)/(s^2+2*zeta*wn*s+wn^2);
[y,t]=step(M,t);
Nf=5;

d1y=gradient(y,t);
d2y=gradient(d1y,t);

t_infl = interp1(d1y, t, max(d1y));
y_infl = interp1(t, y, t_infl);
slope  = interp1(t, d1y, t_infl);   % Slope Defined Here As Maximum Of First
 ↪  Derivative
intcpt = y_infl - slope*t_infl;     % Calculate Intercept
tngt = slope*t + intcpt;

figure (1);
plot(t,y,'-k','LineWidth',1.5);
grid on
title('Inflection point and the tangent curve of the system');
axis([0 length(t)*0.05-0.05 0 1.1]);
hold on
plot(t, tngt, '-r', 'LineWidth',1); % Plot Tangent Line
plot(t_infl, y_infl, 'bp');

%parameters from the graph
T = interp1(tngt, t, 0)
L = (interp1(tngt, t, 1))-T
DeltaCp=max(y)
N=DeltaCp/T
R=N*L/DeltaCp
P=1                                 %from step function

%Ziegler-Nichols
fprintf('Zigler-Nichols')
K=1.2*P/(N*L)
Ti=2*L
Td=0.5*L
Tf=Td/Nf;

G=K*(1+(1/(Ti*s))+(Td*s/(1+Tf*s)));
malha_fechada=(G*M)/(1+G*M);
figure;
[resp_deg_mf,t]=step(malha_fechada);
```

```
erro=1-resp_deg_mf;
sinal_controle=lsim(G,erro,t);
figure;
plot(t,resp_deg_mf,'-b','LineWidth',1.5);
hold on
plot(t,sinal_controle,'-r','LineWidth',1.5);
plot(t,erro,'-g','LineWidth',1.5);
grid on
grid minor
legend('Step Response','Controller Effort','Error');
title (['Ziegler-Nichols Step Step Response - Kp ', num2str(K),'; Ti ', num2str(Ti),';
 ↪  Td ', num2str(Td)]);
hold off

%Cohen-Coon
fprintf('Cohen-Coon')
K=(P/(N*L))*(1.33+R/4)
Ti=L*((32+6*R)/(13+8*R))
Td=L*(4/(11+2*R))
Tf=Td/Nf;
G=K*(1+(1/(Ti*s))+(Td*s/(1+Tf*s)));
malha_fechada=(G*M)/(1+G*M);
[resp_deg_mf,t]=step(malha_fechada);
erro=1-resp_deg_mf;
[sinal_controle,t,x]=lsim(G,erro,t);
figure;
plot(t,resp_deg_mf,'-b','LineWidth',1.5);
hold on
plot(t,sinal_controle,'-r','LineWidth',1.5);
plot(t,erro,'-g','LineWidth',1.5);
grid on
grid minor
legend('Step Response','Controller Effort','Error');
title (['Cohen-Coon Step Response - Kp ', num2str(K),'; Ti ', num2str(Ti),'; Td ',
 ↪  num2str(Td)]);
hold off

%Chien-Hrones-Reswick
fprintf('Chien-Hrones-Reswick')
K=0.95*T/L
Ti=2.4*L
Td=0.42*L
Tf=Td/Nf;
```

```
G=K*(1+(1/(Ti*s))+(Td*s/(1+Tf*s)));
malha_fechada=(G*M)/(1+G*M);
[resp_deg_mf,t]=step(malha_fechada);
erro=1-resp_deg_mf;
[sinal_controle,t,x]=lsim(G,erro,t);
figure;
plot(t,resp_deg_mf,'-b','LineWidth',1.5);
hold on
plot(t,sinal_controle,'-r','LineWidth',1.5);
plot(t,erro,'-g','LineWidth',1.5);

grid on
grid minor
legend('Step Response','Controller Effort','Error');
title (['Chien-Hrones-Reswick - Kp ', num2str(K),'; Ti ', num2str(Ti),'; Td ',
 ↪  num2str(Td)]);
hold off
```

A.1.7 entrada_arbitraria.m

This code is used to simulate the system using an arbitrary input.

```
close all; clc; clear all;

K=0.169022678510371;
Ti=0.072109163345392;
Td=0.091354695916176;
N=5;
Tf=Td/N;

zeta = 0.900733958553509;
wn=8.036118730219977;
k=1;
t=0:0.005:5;

s=tf('s');

sistemas
M=(wn^2)/(s^2+2*zeta*wn*s+wn^2);
P=K*(1+(1/(Ti*s))+(Td*s/(1+Tf*s)));
malha_fechada=(P*M)/(1+P*M);
```

```
20
21 tecido_corrigido = tecido_corrigido';
22 t=0:0.1:(length(tecido_corrigido)*0.1)-0.1;
23 [simulado,t,x]=lsim(malha_fechada,tecido_corrigido,t);
24 %
25 erro=carro-simulado;
26
27 figure(1)
28 plot(t,tecido_corrigido,'-k','LineWidth',1.5);
29 hold on
30 plot(t,simulado,'-r','LineWidth',1.5);
31 plot(t,carro,'-b','LineWidth',1.5);
32 plot(t,erro,'-g','LineWidth',1.5);
33 grid on
34 grid minor
35 legend('Sewing Machine Speed (Input)','Simulated Response','Real Response','Error²
   ↪  (Real-Simulated)²');
36 title('Comparison of the simulated and the real response');
37 xlabel('Time (s)');
38 %
39 erro_relativo=sqrt(((carro-simulado)/carro).^2)*100;
40 figure(2)
41 plot(t,erro_relativo,'-k','LineWidth',1.5);
42 grid on
43 grid minor
44 title('Absolute Relative Error');
45 ylabel('Error (%)');
46 xlabel('Time (s)');
```

A.1.8 plot_resposta.m

This code uses the acquired data from the real system to calculate the ω_n and ζ parameters.

```
1 clear;
2 clc;
3 close;
4 format long;
5
6 t=0:0.1:2.7;
7 %tratamento dos dados
```

```
y=importdata('dados.txt');
y_ss=mean(y(11:28));
y_norm=y/y_ss;

y_norm_overshoot=max(y_norm(1:11));
t_pico =0.1* min(find(y_norm==y_norm_overshoot));

%plot dados
plot (t,y_norm,'--k','LineWidth',1.0);
grid on;
grid minor;
hold on;
title ('Data acquired from the system');
%plot (t_pico,y_norm_overshoot,'+')
%calculo dos parametros

%% modelo de 1a ordem
G=tf([1/0.253],[1 1/0.253])
resposta1order=step(t,G);
plot(t,resposta1order,'-.r','LineWidth',1.0);

%% modelo de 2a ordem
%plot resposta obtida
zeta=sqrt(((log(y_norm_overshoot-1)^2))/((pi^2)+((log(y_norm_overshoot-1))^2)));
wn=pi/(t_pico*sqrt(1-(zeta^2)));

H=tf([wn*wn],[1 2*wn*zeta wn*wn]);
resposta=step(t,H);

plot(t,resposta,'-b','LineWidth',1.0);
legend ('Original Response','First-Order Response','Second-Order Response');
title ('Second-Order System Response');
```

A.2 Arduino Codes

A.2.1 controller_trapezoidal_approx.ino

```
//- timer0  = Pins 5, 6
//- timer1  = Pins 9, 10
//- timer2 = Pins 11, 3
```

```
4  //
5  //Timer0 and timer2 are 8bit timers and timer1 is a 16bit timer
6
7  #define LED 13                // Luz piloto que indica o período de amostragem
8  #define CFW10 11              // Ligada à entrada analógica do CFW10
9  #define Scarro 1.55           // Sensibilidade da velocidade em função da frequencia
10 #define Stecido 3.42          // Sensibilidade da velocidade em função da frequencia
11 #define SPscale 0.6           // Fator de ajuste do set-point
12
13 volatile bool sample;         // Flag que ativa nova amostragem
14 volatile int  cartPulses = 0; // Conta o numero de transições observadas em cada 100ms
    ↪  para o carrinho
15 volatile int swingPulses = 0; // Conta o numero de transições observadas em cada 100ms
    ↪  para a máquina de costura
16
17 double x1_k = 0.0, x1_k_1 = 0.0, x1_k_2 = 0.0; // Variáveis de estado para o filtro
    ↪  passa-baixo para a medição da velocidade do
18 double y1_k = 0.0, y1_k_1 = 0.0, y1_k_2 = 0.0; // carro
19
20 double x2_k = 0.0, x2_k_1 = 0.0, x2_k_2 = 0.0; // Variáveis de estado para o filtro
    ↪  passa-baixo para a medição da velocidade do
21 double y2_k = 0.0, y2_k_1 = 0.0, y2_k_2 = 0.0; // máquina de costura
22
23 double Vcarro  = 0.0;  // Velocidade do carro em m/h
24 double Vtecido = 0.0;  // Velocidade do tecido em m/h
25 double Setpoint = 0.0;       // Setpoint em m/h
26 double u_k = 0.0;      // Sinal de controlo aplicado no instante k
27 double u_k_1 = 0.0;    // Sinal de controlo aplicado no instante k-1
28 double e_k = 0.0;      // Sinal de erro calculado no instante k
29 double e_k_1 = 0.0;    // Sinal de erro calculado no instante k-1
30
31 //constantes utilizadas na f.t. do controlador
32 double K=0.168944378473349; //Ganho do controlador
33 double Ti=0.097878329673360; //Constante de ganho Integral
34 double Td=0.680870631559591; //Constante de ganho Diferencial
35 double Ts=0.1; //Tempo de amostragem;
36 double N=5;
37 double a_i = K * Ts / Ti;
38 double b = Td / (Td + N);
39 double b_1 = (K * Td * (1 - b))/Ts;
40
41 double u_p_k, u_i_k, u_i_k_1,a_1 =0;
42 double u_d_k, u_d_k_1, y_k, y_k_1, e_k_11 =0;
```

```
43
44  //.........................
45
46  /*************************************************************
47            CONFIG TIMER 1 (sampling period)
48  *************************************************************/
49  void ConfigTimer1(void)
50  {
51    //---------------------- Configure TIMER 1
52    TCCR1A = 0x00;
53    // Timer mode with 256 prescaler (62500 Hz)
54    TCCR1B = (1 << CS12);
55    // 100 ms => (2^16-1) - 16e6/256*0.1
56    TCNT1  = 59285;
57    // Enable timer1 overflow interrupt
58    TIMSK1 = (1 << TOIE1) ;
59  }
60  /*************************************************************
61            Config INT 0
62  *************************************************************/
63  void ConfigInt0(void)
64  {
65    DDRD  &= ~(1 << PD2); // PD2 as input
66    PORTD |= (1 << PD2); //Turn On the Pull-up
67    EICRA |= (1 << ISC00); //Any logical change on INT0 generates
68    EIMSK |= (1 << INT0); //an interrupt request
69  }
70  /*************************************************************
71            Config INT 1
72  *************************************************************/
73  void ConfigInt1(void)
74  {
75    DDRD  &= ~(1 << PD3); // PD2 as input
76    PORTD |= (1 << PD3); //Turn On the Pull-up
77    EICRA |= (1 << ISC10); //Any logical change on INT1 generates
78    EIMSK |= (1 << INT1); //an interrupt request
79  }
80  /*************************************************************
81            TIMER 1 ISR
82  *************************************************************/
83  ISR (TIMER1_OVF_vect)
84  {
85    sample = true;
```

```
86    TCNT1 = 59285;    // for 100ms@16MHz
87  }
88  /*************************************************************
89            INT0 ISR
90  *************************************************************/
91  ISR (INT0_vect)
92  {
93    cartPulses++;
94  }
95  /*************************************************************
96            INT1 ISR
97  *************************************************************/
98  ISR (INT1_vect)
99  {
100   swingPulses++;
101 }
102 /*************************************************************
103           MAIN
104 *************************************************************/
105 void setup() {
106   cli();
107   Serial.begin(250000);
108   ConfigTimer1();
109   ConfigInt0();
110   ConfigInt1();
111   pinMode(LED, OUTPUT);
112   pinMode(CFW10, OUTPUT);
113   pinMode(7  , OUTPUT);
114   sei();
115   analogWrite(CFW10, 100);
116
117 }
118
119 void loop() {
120   // put your main code here, to run repeatedly:
121   if (sample)
122   {
123     cli();
124     digitalWrite(LED, !digitalRead(LED));
125     /*************************************************************
126       // Filtro passa-baixo de 2ª ordem (Butterwrth com Wc = 0.2)
127     *************************************************************/
128     x1_k_2 = x1_k_1; // Atualiza variáveis de estado
```

```
129     x1_k_1 = x1_k;   // Atualiza variáveis de estado
130     x1_k   = (double) (5.0 * cartPulses);
131     y1_k_2 = y1_k_1; // Atualiza variáveis de estado
132     y1_k_1 = y1_k;   // Atualiza variáveis de estado
133     y1_k = 0.0675 * x1_k + 0.135 * x1_k_1 + 0.0675 * x1_k_2 + 1.143 * y1_k_1 - 0.413 *
        ↪  y1_k_2;
134     if (y1_k < 0) y1_k = 0;
135     // y1_k é a frequência medida, em Hz, do carro
136     /*************************************************************
137       // Filtro passa-baixo de 2ª ordem (Butterwrth com Wc = 0.2)
138     *************************************************************/
139     x2_k_2 = x2_k_1; // Atualiza variáveis de estado
140     x2_k_1 = x2_k;   // Atualiza variáveis de estado
141     x2_k   = (double) (5.0 * swingPulses);
142     y2_k_2 = y2_k_1; // Atualiza variáveis de estado
143     y2_k_1 = y2_k;   // Atualiza variáveis de estado
144     y2_k = 0.0675 * x2_k + 0.135 * x2_k_1 + 0.0675 * x2_k_2 + 1.143 * y2_k_1 - 0.413 *
        ↪  y2_k_2;
145     if (y2_k < 0) y2_k = 0;
146     // y2_k é a frequência medida, em Hz, da máquina de costura
147     /*************************************************************
148       // Controlador
149     *************************************************************/
150     //.................................................. Velocidade do carro
151     Vcarro  = Scarro * y1_k;                // expressa em (m/h)
152     //.................................................. Velocidade do tecido
153     Vtecido = Stecido * y2_k;              // expressa em (m/h)
154     //.................................................. Velocidade da máquina de
        ↪  costura (setpoint)
155     Setpoint      = SPscale * Vtecido;
156     //.................................................. Erro de setpoint atual
157     e_k = Setpoint - Vcarro;  // expressa em (m/h)
158
159     //.................................................. Controlador PI
160
161     //Propocional
162     u_p_k = K * e_k;
163
164     //Integral
165     u_i_k = u_i_k_1 + a_i * e_k;
166
167     //Derivativo
168     u_d_k = b * u_d_k_1 - b_1 * (y_k - e_k_1);
```

```
169
170     //Final
171     u_k = u_p_k + u_i_k + u_d_k;
172
173     if (u_k < 0) u_k = 0;
174     else if (u_k > 255) u_k = 255;
175     //.................................................. Aplica variável de controlo
176     analogWrite(CFW10, (uint8_t)u_k);
177     //.................................................. Atualiza variáveis de
        ↪  estado
178     u_i_k_1=u_i_k;
179     u_d_k_1=u_d_k;
180     y_k_1=y_k;
181     e_k_1=e_k;
182
183     //.................................................. NOVA AMOSTRA
184     //Serial.print("\t"); //Erro:
185     Serial.print(e_k);
186     Serial.print("\t"); //\tCarro:
187     Serial.print(Vcarro);
188     Serial.print("\t"); //\tTecido:
189     Serial.print(Vtecido);
190     Serial.print("\t"); //\tCtr:
191     Serial.println(u_k);
192     //digitalWrite(7, !digitalRead(7));
193
194     cartPulses  = 0;
195     swingPulses = 0;
196     sample = false;
197     sei();
198   }
199 }
```

A.2.2 controller_tustin.ino

The controller using the tustin approximation is similar to the trapezoidal approximation. The difference is in the declaration of the variables, the calculation of the constants and the controller function. Only the differences are listened below:

```
double Kp = 0.167598529831592;
double Ti = 0.072333574832971;
double Kd = 0.012110708117485;
double Ki = 1 / Ti;
double Ts=0.1;
double a = K + (Ki*Ts / 2) + Kd / Ts;
double b = -Kp + (Ki*Ts / 2) - (2 * Kd / Ts);
double c = Kd / Ts;

[...]
    //controller function
    u_k = u_k_1 + a * e_k + b * e_k_1 + c * e_k_2;
    if (u_k < 0) u_k = 0;
    else if (u_k > 255) u_k = 255;
    //.................................................. Aplica variável de controlo
    analogWrite(CFW10, (uint8_t)u_k);
    //.................................................. Atualiza variáveis de
     ↪  estado
    e_k_1 = e_k;
    e_k_2 = e_k_1;
    u_k_1 = u_k;
```

Appendix B

Appendix: Simulations

In this chapter will be shown some runs of the evolutionary algorithms that have some interesting points to e analysed.

With the genetic Algorithms, the central point of the analysis is the result represented in the figure B.1, using the weights of the table B.1.

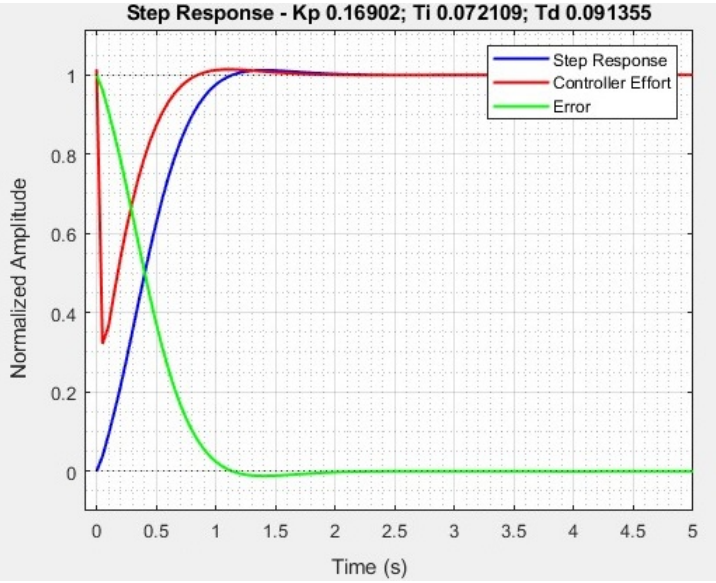


Figure B.1: Step response from the central point of the analysis

These values were found with a trial and error adjust, observing the obtained response and changing the weights. For example, if the overshoot was high, the weight of the overshoot was increased, run other simulation, check the result, and as it goes on.

Parameter	Value
k_bw	0
k_ctr	1.8
k_deltaT	5
k_erro	100
k_overshoot	0.3

Table B.1: Weights used in the central point

To verify some prorpieties of these parameters and how it affects the response, some others tests was done. The most interesting points will be shown in the following figures, with its weights and PID gains.

To a quick review, the parameters are:

- k_bw: Bandwidth of the closed-loop system;
- k_ctr: Maximum controller effort;
- k_deltaT: Difference of the rise time and the settling time;
- k_erro: Mean error in the steady-state;
- k_overshoot: Value of the amplitude of the overshoot.

Increasing the gain from the k_deltaT parameter, the result was the response of the figure B.2, table B.2.

Parameter	Value
k_bw	0
k_ctr	1.8
k_deltaT	10
k_erro	100
k_overshoot	0.3

Table B.2: Parameters with a changed k_deltaT

With a slightly change of the $k\_overshoot$ from 0.3 to 0.2 changes the balance and mess everything. The controller effort goes up, and the rise time get wrong, as shown in the figure B.3, table B.3.

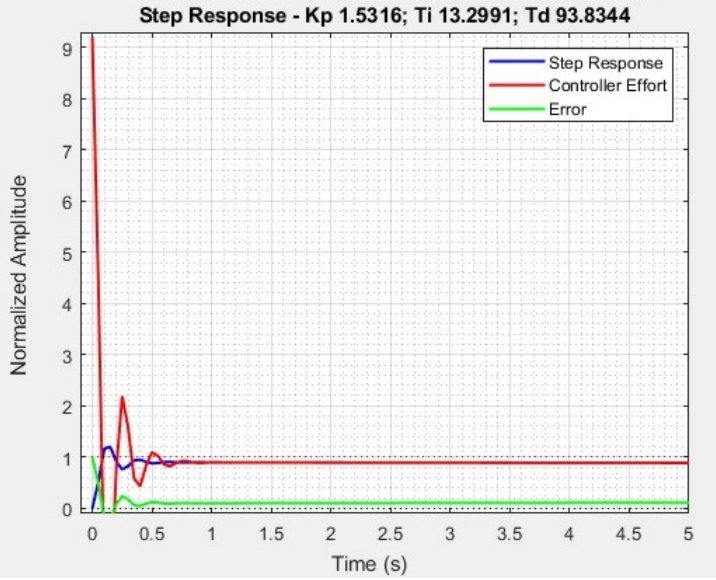


Figure B.2: Test changing the k_deltaT

Parameter	Value
k_bw	0
k_ctr	1.8
k_deltaT	5
k_erro	100
k_overshoot	0.2

Table B.3: Parameters with a changed k_overshoot

The parameter *k_bw* had a good expect with it to the results, but when the system is limited in the bandwidth, it became very complicated to keep all the parameters. With this weight set to 1, the result is shown in the figure B.4, table B.4, that begins to distort the curve. Increasing this weight only distorts more the result, as in the figure B.5, table B.5.

If the overshoot and the controller effort weight is changed, it also unbalances the results, resulting as in the figure B.6.

To have a better result, the best approach is increase the weights, never decrease. This method keeps the balance of all parameters and do not makes big unexpected changes.

These are only some of the performed tests. All tests are availabe on the Github repository, on the folder

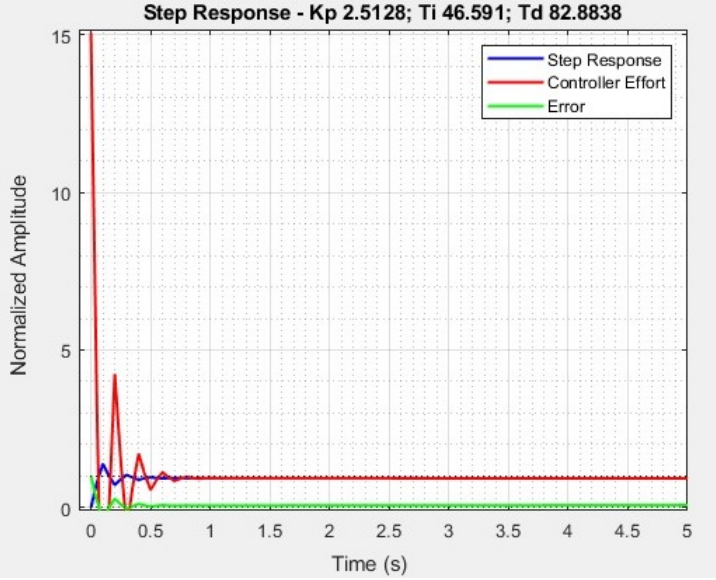


Figure B.3: Test changing the k_overshoot

Parameter	Value
k_bw	1
k_ctr	1.8
k_deltaT	5
k_erro	100
k_overshoot	0.3

Table B.4: Parameters with a little increasek of k_bw

https://github.com/morettieric/SewingStation/tree/master/dump/Matlab/forma_2/ga.

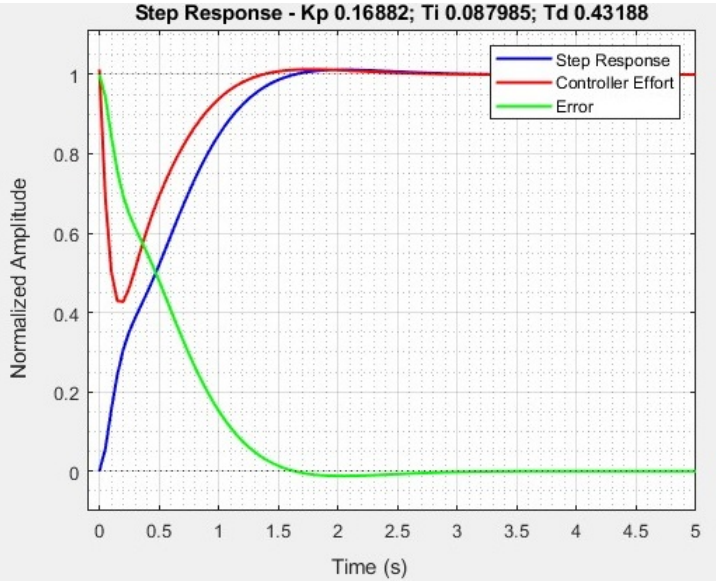


Figure B.4: Test increasing the k_bw to 1

Parameter	Value
k_bw	8
k_ctr	1.8
k_deltaT	5
k_erro	100
k_overshoot	0.3

Table B.5: Parameters with a great increase of k_bw.

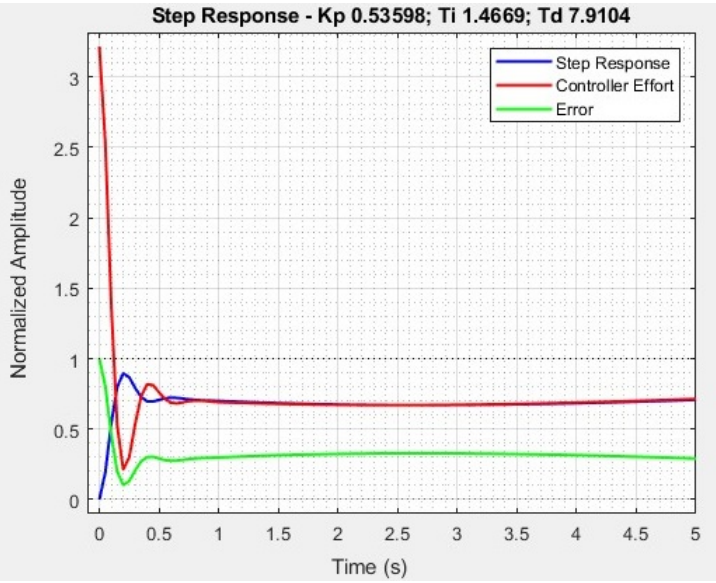


Figure B.5: Test increasing the k_bw to 8

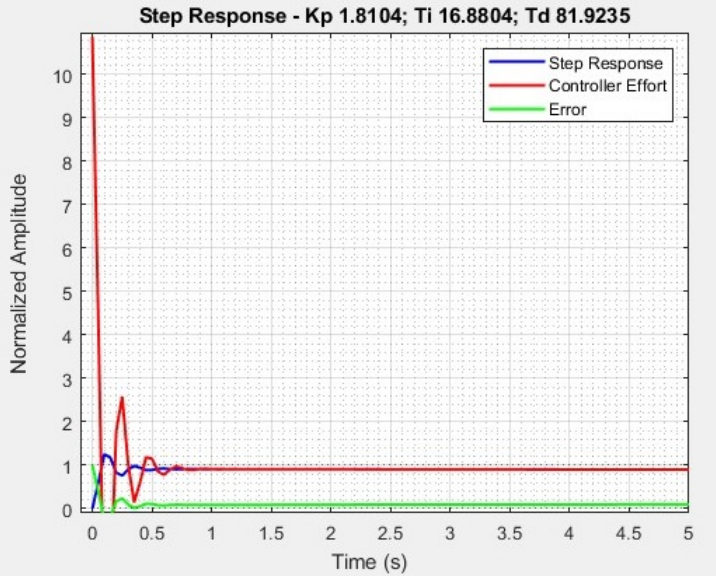


Figure B.6: Unbalance of the weights