



Bachelor's thesis

# **DC motor control with quadrature encoder via Beckhoff EL7342 module**

*Adéla Čekalová*

Department of Mechanics, Biomechanics and Mechatronics

Supervisor: Ing. Martin Nečas, MSc., Ph.D.

August 9, 2020



## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Čekalová** Jméno: **Adéla** Osobní číslo: **466529**  
Fakulta/ústav: **Fakulta strojní**  
Zadávající katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**  
Studijní program: **Teoretický základ strojního inženýrství**  
Studijní obor: **bez oboru**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Řízení DC motoru s kvadrurním enkodérem prostřednictvím modulu Beckhoff EL7342**

Název bakalářské práce anglicky:

**DC motor control with quadrature encoder via Beckhoff EL7342 module**

Pokyny pro vypracování:

1. Proveďte rešerši na využití elektrické pohonové techniky v robotice.
2. Seznamte se se základními principy řízení DC motoru (PID regulátor, H-můstek, PWM).
3. Zprovozněte řízení DC motoru pomocí modulu Beckhoff EL7342 v softwaru TwinCat.
4. Implementujte systém nastavování parametrů PID regulátoru modulu EL7342 prostřednictvím Matlab/Simulink modulu TE1400 a realizujte systém jednoduchého autotuningu.
5. Kriticky zhodnoťte dosažené výsledky.

Seznam doporučené literatury:

- [1] NOVÁK, Martin. Technická měření. V Praze: České vysoké učení technické, 2018. ISBN 978-80-01-06388-0.
- [2] UHLÍŘ, Ivan. Elektrické stroje a pohony. Vyd. 2., přeprac. Praha: Nakladatelství ČVUT, 2007. ISBN 978-80-01-03730
- [3] HOFREITER, Milan. Základy automatického řízení. V Praze: České vysoké učení technické, 2012. ISBN 9788001050071.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

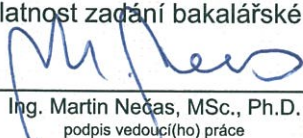
**Ing. Martin Nečas, MSc., Ph.D., odbor mechaniky a mechatroniky FS**


Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **30.04.2020**

Termín odevzdání bakalářské práce: **07.08.2020**

Platnost zadání bakalářské práce: \_\_\_\_\_

  
Ing. Martin Nečas, MSc., Ph.D.  
podpis vedoucí(ho) práce

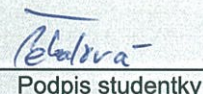
  
doc. Ing. Miroslav Španiel, CSc.  
podpis vedoucí(ho) ústavu/katedry

  
prof. Ing. Michael Valášek, DrSc.  
podpis děkana(ky)

## III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

15. 7. 2020  
Datum převzetí zadání

  
Podpis studentky

---

# Acknowledgements

I pay my deep sense of gratitude to my advisor, Ing. Martin Nečas, MSc., Ph.D., for his willingness, long-term support, and for guiding me through this whole bachelor thesis.

Then, I express my sincere thanks to my family, especially to my parents, for patience, love, advice, and encouragement through my studies, I truly appreciate it.

Last but not least, I would like to thank my friends who have helped me a lot.



---

# Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work for non-profit purposes only, in any way that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on August 9, 2020

.....

Czech Technical University in Prague

Faculty of Mechanical Engineering

© 2020 Adéla Čekalová. All rights reserved.

*This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Mechanical Engineering. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

**Citation of this thesis**

Čekalová, Adéla. *DC motor control with quadrature encoder via Beckhoff EL7342 module*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Mechanical Engineering, 2020.

---

# Abstrakt

Tato bakalářská práce je obecné seznámení s pohonnými jednotkami v robotice, jejich rozdíly, výhodami a nevýhodami. Práce se zaměřuje na DC motory, na jejich typy a na princip fungování. Následně jsou nastíněny možnosti realizace řízení DC motorů.

Cílem praktické části je návrh a implementace řízení DC motoru Moore Reed and Company Ltd 20 MPM 105 s repasovaným odměřováním. Řízení se provádí pomocí modulu Beckhoff EL7342 a programech vytvořených v TwinCATu a MATLAB/Simulinku, kde je implementován jednoduchý autotuning pro zaručení dobrých servo-vlastností pohonu.

**Klíčová slova** DC motor, H-můstek, Pulzně šířková modulace, PID regulace, Autotuning, Řízení



---

# Abstract

This bachelor thesis is a general introduction to power units in robotics, their differences, advantages, and disadvantages. This work focuses on DC motors, their types, and the principle of operation. Then, there are outlined possibilities of realization of DC motors control.

The practical part aims to design and implement control of DC motor Moore Reed and Company Ltd 20 MPM 105 with refurbished measuring. The control is performed by module Beckhoff EL7342 and programs created in TwinCAT and MATLAB/Simulink, where a simple autotuning is implemented in order to assure good servo-drive properties.

**Keywords** DC motor, H-Bridge, Pulse Width Modulation, PID Control, Autotuning, Control

---

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Motivation and goals</b>	<b>3</b>
<b>2 Theory</b>	<b>5</b>
2.1 Drive Systems in Robotics . . . . .	5
2.2 DC Motors . . . . .	7
2.3 DC Motor Control and Servo-control . . . . .	14
2.4 Tuning of PID Controllers . . . . .	24
2.5 TwinCAT 3 . . . . .	31
2.6 Beckhoff EL7342 Module . . . . .	35
<b>3 DC Motor Servocontrol</b>	<b>43</b>
3.1 Encoder Replacement . . . . .	43
3.2 Project Description . . . . .	45
3.3 Connection of TwinCAT and MATLAB/Simulink via the PLC	46
3.4 Connection of TwinCAT and MATLAB/Simulink via the PLC and the NC . . . . .	50
<b>4 Automatic Servotuning</b>	<b>55</b>
4.1 Testing of Gradient Descent Method in MATLAB . . . . .	55
4.2 Autotuning Algorithm in MATLAB/Simulink . . . . .	58
<b>5 Results</b>	<b>63</b>
<b>Conclusion</b>	<b>65</b>
<b>Bibliography</b>	<b>67</b>



---

## List of Figures

2.1	Shunt wound DC motor [5]	8
2.2	Series wound DC motor [5]	8
2.3	Compound wound DC motor [5]	9
2.4	DC motor - conceptual diagram [3]	10
2.5	Fleming's Left Hand Rule [6]	11
2.6	BLDC motor - construction [7]	12
2.7	BLDC motor - working principle [8]	12
2.8	Variable reluctance stepper motor [11]	14
2.9	H-Bridge [15]	15
2.10	H-Bridge - forward movement [15]	15
2.11	H-Bridge - backward movement [15]	15
2.12	Duty cycles [18]	17
2.13	Pulse width modulated waveform [19]	17
2.14	Operation of PWM [20]	18
2.15	A typical PID control system [22]	19
2.16	A static characteristic of P-controller [21]	20
2.17	Transient response of the ideal P-controller [21]	20
2.18	Transient response of the ideal I-controller [21]	21
2.19	Transient response of the ideal PI-controller [21]	22
2.20	Responses of P-controller, I-controller, and PI-controller [22]	22
2.21	Transient response of an ideal PD-controller [21]	23
2.22	Transient response of the ideal PID-controller [21]	24
2.23	The steepest descent direction to reach a point of local minima [26]	27
2.24	Cascade control block diagram [32]	30
2.25	TwinCAT 3 Engineering Environment [33]	32
2.26	Modular structure of the TwinCAT 3 Runtime Environment [33]	33
2.27	Ethernet - bus topology [35]	34
2.28	EtherCAT - ring topology [35]	35
2.29	EtherCAT's network topology [35]	35
2.30	The EL7342 EtherCAT terminal [36]	36

2.31	Possibilities to implement control loops using the EL7342 terminal [36] . . . . .	39
2.32	Relationship between user side (Commissioning) and installation [36]	40
2.33	New TwinCAT 3 project [36] . . . . .	41
3.1	The process of replacing damaged encoders with new ones . . . . .	44
3.2	The new encoders (type AEDA - 3300 TAM) attached to DC motors	44
3.3	Diagram describing the project using 3 PLC programs . . . . .	46
3.4	Beckhoff EK1100 EtherCAT coupler [39] . . . . .	48
3.5	Beckhoff EL9011 Bus end cover [40] . . . . .	48
3.6	Project created in TwinCAT . . . . .	48
3.7	Program created in MATLAB/Simulink . . . . .	50
3.8	Output graph from MATLAB/Simulink . . . . .	50
3.9	Output graph from TwinCAT . . . . .	50
3.10	Diagram describing the project using the NC configuration . . . . .	51
3.11	TwinCAT program using the PLC and the NC . . . . .	53
3.12	Code of the PLC program in TwinCAT . . . . .	53
3.13	Program created in MATLAB/Simulink for manual tuning . . . . .	54
4.1	Gradient descent MATLAB function . . . . .	56
4.2	Surface of the given function . . . . .	57
4.3	Point of minima of the given function . . . . .	57
4.4	MATLAB Command Window . . . . .	57
4.5	Difference between the actual position and the set value of the position . . . . .	58
4.6	MATLAB/Simulink optimization program with scope . . . . .	59
4.7	Triggered subsystem . . . . .	60
4.8	Generated pulses for simulation of optimization . . . . .	60
4.9	MATLAB function for optimization . . . . .	61

---

## List of Tables

2.1	Allowed combination of switches . . . . .	16
2.2	Calculation of controller's parameters using Ziegler-Nichols method [26] . . . . .	26
2.3	Technical data of the module EL7342 [36] . . . . .	37



---

# Introduction

Very fast development can be observed in robotics and automation. This impacts not only many technological fields but the whole society. Consumers demand more and more; therefore, the requirements are very high. The production of quality goods is crucial and a well-functioning product is the norm. When it comes to creating and manufacturing a robot, a great emphasis is placed on the positioning. The demanding aspects are accuracy and repeatability, which basically means to make the robot do the same thing over and over again accurately.

What is an industrial robot? The industrial robot is a device that is programmed and controlled by a human to perform preprogrammed, repetitive, and dangerous tasks with consistent precision and accuracy. These devices can work 7 days a week because of their automated functionality. They can be used in challenging and hazardous environments as well as increase productivity. However, all of this is preceded by developing the right hardware and software.

The whole world is driven to make everything faster, more effective, and cheaper. Hence, sophisticated control and regulation of robots' systems are essential when a company wants to maintain a good market position. There are a lot of different approaches to some kind of optimization, for example, autotuning.





---

# Motivation and goals

A motivation of this thesis is to broaden the knowledge of a robotics field, mainly in electric drive technology, which is developing very quickly. However, this work focuses more on DC motors and their controllers. Due to high requirements for efficiency, reliability, and accuracy, optimization algorithms are used to analyze a system and to find the right parameters to ensure the desired operation of the robot. The goal of this bachelor thesis is to optimize the control of DC motor Moore Reed and Company Ltd 20 MPM 105 by a module from Beckhoff company using a method called autotuning. Autotuning is a technique used to optimize an algorithm to find parameters to produce the best performance, such as a fast setting time with zero overshoot to ensure a highly dynamic running of the motor.

This thesis is divided into 7 chapters, including the introduction and the conclusion. The work is organized as follows:

- *A research on the use of drive technology in robotics, with a deeper focus on electric drive technology, especially on DC motors.*
- *The basic principles of DC motors control, which requires to understand the following terms:*
  - *PID Controllers*
  - *H-Bridge*
  - *Pulse Width Modulation*
- *DC motor control using the Beckhoff module EL7342.*
- *A simple autotuning system.*
- *Critical evaluation of achieved results.*

In the theory section, different types of DC motors, their constructions, working principles, their advantages, and disadvantages over each other are described. Also, the use and suitability of DC motors in particular systems are mentioned there. The following part of the theory deals with servo-control and DC motor control. In order to understand the controllers, essential key technical terms, such as the PID controller, PWM, and H-Bridge, are explained.

To control the examined DC motor, the Beckhoff EL7342 EtherCAT terminal is used.

Before approaching a practical part of this bachelor thesis, the motor's measuring must have been refurbished, which means that the damaged encoders were replaced with new ones. After making sure that the motor works properly, the practical part was started.

The practical part aims to develop and implement the optimizing autotuning algorithm created in MATLAB/Simulink. First, the motor along with the encoder is connected to a PC through the Beckhoff module EL7342. In order to control the DC motor, a project in TwinCAT is developed. It is desired to control the motors' parameters via MATLAB/Simulink where automatic servo tuning algorithm based on the gradient descent method is programmed. When both the TwinCAT and the Simulink programs are finished, it is possible to link these programs. Then, TwinCAT and MATLAB/Simulink are able to interface, cooperate, and transfer data to each other.

---

# Theory

## 2.1 Drive Systems in Robotics

### 2.1.1 Pneumatic Drive (Pneumatics)

Pneumatic power is used to drive about one-third of industrial robots, including the pneumatic-powered pick-and-place units; otherwise, the percentage is a bit less. It is mainly used to control grippers and sometimes also manipulator arms.

Pneu means “air” in Latin; therefore, pneumatic drives are air-driven, which means that the medium used to generate the power is compressed air. The compressed air (or similar gas) is used to transmit energy and force, to do so a motor-driven compressor alongside with a tank is needed. The motor and the compressor pressurize the air and store it inside the tank, from where the pressurized air can be delivered through the hoses to cylinders to make the robot or any other device move. Inside the cylinders, the mechanical energy of the compressed air is turned into a linear motion, or it could be a rotary motion as well. Valves play an important role, as they are used to regulate the flow of the air.

A big advantage is that after the process is done, the air can be exhausted into the atmosphere. However, it is quite challenging to remove the moisture out of the air and keep the lines clean and dry. Probably the biggest limitation of pneumatic drive systems is that the compressed air is a very expensive medium and moreover the torque that can be generated is low. [1]

### 2.1.2 Hydraulic Drive (Hydraulics)

Almost about half of industrial robots use hydraulic power. It is mostly because of the fact that hydraulic systems are capable of transmitting high forces.

For this reason, they are used in robust machines, such as loaders, excavators, and many other agricultural and construction machinery. Another example of usage is a hydraulic system of braking in automobiles.

Hydraulic systems are very similar to pneumatic systems. Hydro comes from Greek and means water. But instead of water, hydraulic systems use oil. There is a pump, which is powered by an electric motor, that puts a fluid under the pressure and pushes the liquid through the system. The pressurized liquid moves through the piping until it reaches a cylinder. The cylinder is extended to one or the other side by pressure of the fluid, which causes the movement of the desired part.

However, hydraulics have some limitations as well. For example, leaking is the main problem, the liquid is under big pressure and is seeking for a defect to run out through it. Another unpleasant point is the price. When taking into consideration the size, it is the most expensive technology out of the three drive systems mentioned. [1]

### 2.1.3 Electric Drive

The power source of electric drive systems are electric motors. They can be divided into two groups:

- *DC<sup>1</sup> motors - operated by direct current*
- *AC<sup>2</sup> motors - operated by alternating current*

Each type has its own pros and cons. Electric motors are controlled by a microprocessor or a computer, so then the torque and speed can be easily controlled.

Electric drive systems are not capable of lifting such big loads as hydraulic drive systems and they are still not completely safe to use in spray-painting atmospheres. However, they are very versatile and more importantly, they are able of very smooth control in starting, accelerating, decelerating, and stopping. Compared to the previous two types (hydraulics and pneumatics), electric robots have a higher repeatability, accuracy, efficiency, and easy installation. Therefore, electric drives are used in welding, assembly operations, machine loading, unloading, and simply in high-tech robots. Besides, electric actuators operate at a fraction of the cost of fluid-powered actuators. [1]

---

<sup>1</sup>DC = Direct Current

<sup>2</sup>AC = Alternating Current

## 2.2 DC Motors

DC motors are used in many types of industrial equipment, home appliances as well as in automobiles to drive electric seats and electric windows [2]. In general, DC motors are widely used in robotics, particularly in mobile and collaborative robots. The main reason for that is the possibility of the robot to be battery-powered, which makes a great advantage.

DC motor converts electrical energy to mechanical energy (rotational movement) and has two major parts [3]:

1. *Stator - a static part*
2. *Rotor - a rotating part*

DC motors have different designs with specific characteristics. The rotor does not change in basically all DC motors, but what changes is the stator. It is made of either permanent magnets or coil windings which divides DC motors into two groups [4]:

- *Permanent Magnet DC Motors*
- *Wound DC Motors - can be also differentiated according to the configuration [5]:*
  - *Shunt*
  - *Series*
  - *Compound*

DC motors can be also divided into following three groups [1]:

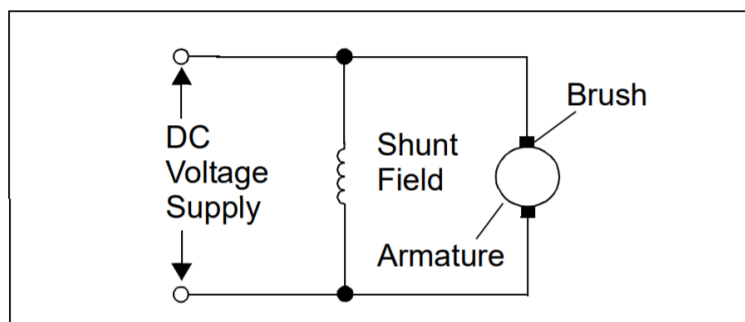
- *Brushed DC Motors*
- *Brushless DC Motors*
- *Stepper Motors*

Let's look at each type closely.

### 2.2.1 Shunt Wound DC Motor

A shunt wound DC motor has the armature (rotor) in parallel with the field coil (stator), as shown in Fig. 2.1. Based on this figure, it is clear that the total current is split into two parts [4]: the current through the armature  $I_a$  and the current through the field windings  $I_f$ . These currents are independent of each other and it is given that [4]:

$$I_{total} = I_a + I_f \tag{2.1}$$



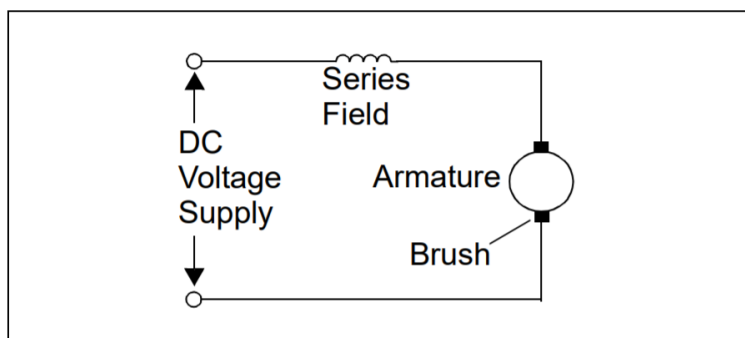
**Figure 2.1:** Shunt wound DC motor [5]

This configuration provides good speed control, but the starting torque is very low due to the low current at the startup. This is caused by the high resistance of field windings that are made of a thin wire with many turns. Hence, these motors are suitable for application where the constant speed characteristic is crucial, but the required starting torque is relatively low. [4], [5]

### 2.2.2 Series Wound DC Motor

In a series wound motor, the armature is connected in series with field windings, as depicted in Fig. 2.2. This ensures that the current through both of the parts in series is the same [4]:

$$I_{total} = I_a = I_f \quad (2.2)$$

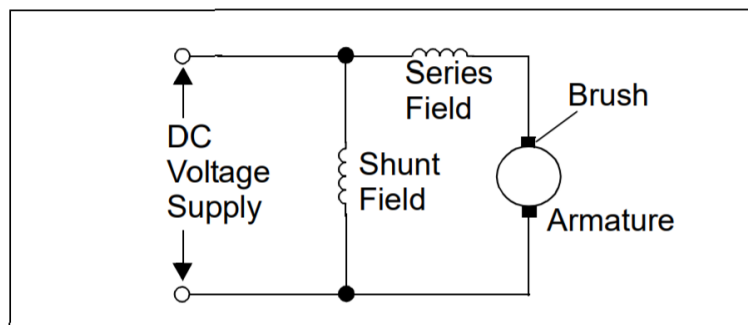


**Figure 2.2:** Series wound DC motor [5]

Compared to the shunt DC motor, the series one has field windings made of a thicker wire with fewer turns which provides lower resistance; therefore, the starting torque is higher. DC motors with this configuration are used where there is a need for a lot of torque at the startup, for instance, to lift or move something heavy. But the speed regulation is difficult. [4], [5]

### 2.2.3 Compound Wound DC Motor

As shown in Fig. 2.3, compound wound motors are a combination of series wound and shunt wound motors, which as well combines their characteristics and qualities. They have higher torque than shunt wound DC motors while providing better speed control than series wound DC motors. Their biggest limitation is that the series winding works against the shunt winding. [4], [5]



**Figure 2.3:** Compound wound DC motor [5]

### 2.2.4 Brushed DC Motor

Brushed DC (BDC) motors are widely available in all sizes and shapes. They are not very expensive, easy to control, and quite efficient. That gives them several advantages to ensure their future use.

Let's first start with the simplest brushed DC motor possible, shown in Fig. 2.4. The stator consists of either a permanent magnet or an electromagnet. As the name says, the permanent magnet provides a permanent magnetic field. This means that there is always a magnetic field unlike with an electromagnet. The electromagnet is made in a form of a coil. In order to create a magnetic field, an electric current needs to pass through the coil.

In case of this thesis, the motors that will be dealt with are DC motors with permanent magnets. Consequently, this design will be discussed in more detail.

The armature connected with commutators form the rotating part, called the rotor. The commutators are made from a copper separated by a mica insulator. These DC motors have carbon brushes, as the name implies.

The direct current is fed to the armature through a pair of commutator rings and brushes, it results into creating a magnetic field in the armature that interacts with the permanent magnetic field of the stator. The static magnetic field is created between South and North poles of the stator and has a set



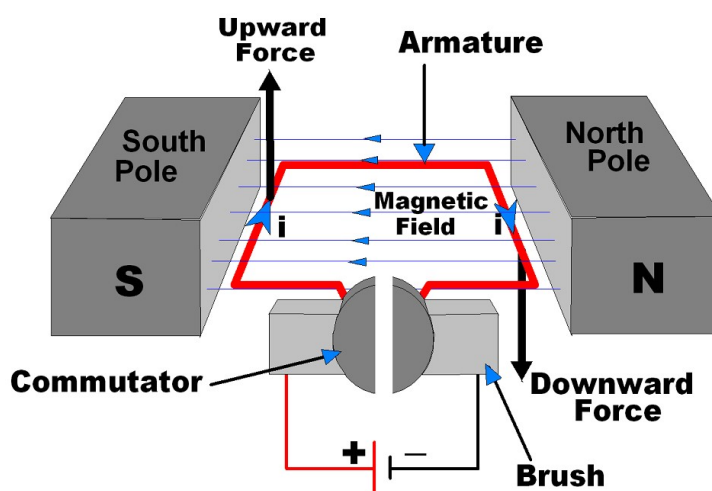


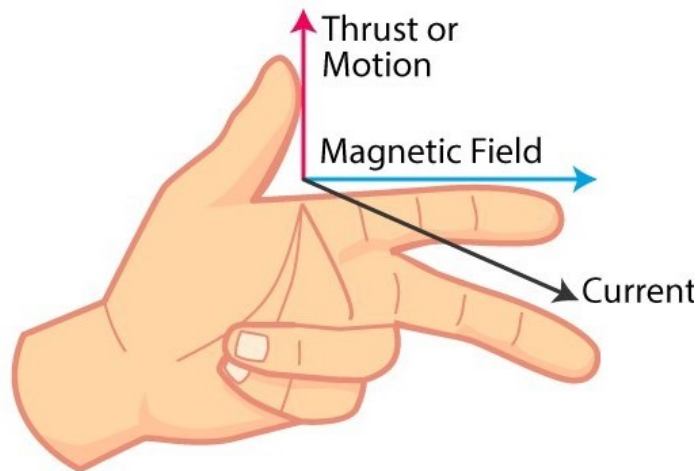
Figure 2.4: DC motor - conceptual diagram [3]

direction. When the current flows through the coil, the magnetic field generated in the coil is attracted to the permanent magnetic field which causes the armature to rotate. As the coil rotates, the commutators connect with the power source of different polarity. This makes the electricity on the left side of the coil always flow away and on the right side it flows towards. This provides the motion in the same direction and keeps the coil rotating. However, if taken into consideration only one armature loop, the motion will be irregular because when the coil is perpendicular to the magnetic flux, the torque action nears zero. To overcome this problem, more loops need to be added and the more loops are added, the smoother will be the motor rotation. Each loop has a pair of commutator rings. As the motor rotates, the following two commutator rings move over to make contact with the brushes that magnetized them. As a result, the next magnetized coil is attracted to the permanent magnet. Simply, the commutators are a switching device that provides the smooth motion and determines which coils are energized.[1], [2], [3]

The direction of rotation depends upon Fleming's Left Hand Rule. This rule is valid for electric motors, not generators, and is a mnemonic. It is a simple way to find out the direction of motion, as depicted in Fig. 2.5. The thumb points in the direction of the motion and also the direction of EMF<sup>3</sup>. The middle finger indicates the direction of the current and the forefinger represents the direction of the magnetic field. [6]

Changing the direction of rotation can be done by simply flipping the polarity of the electric current supplied. To control the speed of BDC motors, their

<sup>3</sup>EMF = Electromotive Force



**Figure 2.5:** Fleming's Left Hand Rule [6]

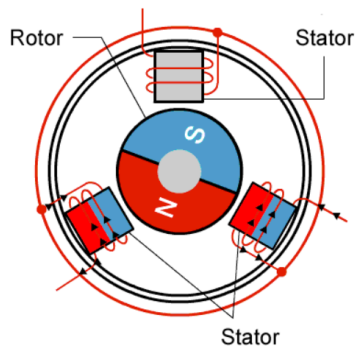
current or voltage or both can be regulated or it is also possible to insert a variable resistor in series with the field and adjust it to increase or decrease the voltage. [1]

### 2.2.5 Brushless DC Motor

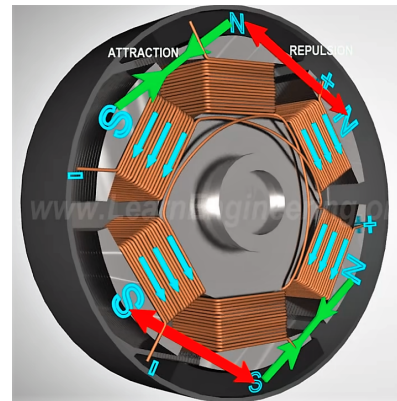
As the name suggests, brushless DC (BLDC) motors do not have brushes. They are basically the same as BDC motors but turned "inside-out". This means that the windings are not located on the rotor, but they are moved to the stator and remain stationary. Because the coils do not move, there is no need for brushes and commutators. On the other hand, the rotor consists of a permanent magnet that rotates. The rotor can be either inrunner or outrunner. An inrunner BLDC motor has the rotor inside the stator and an outrunner BLDC motor has the rotor outside of the stator. Inrunner BLDC is shown in Fig. 2.6. Also, note that the coil arrangement is different from BDC motors.

To move the rotor, direct current is fed to the coils on the stator which are then energized and become an electromagnet. The coils are energized one by one which attracts the permanent magnet and ensures that the rotor moves. When only one coil is powered, the two other coils reduce the power output. To make it more efficient, two coils can be energized at the same time with the same polarity current, as depicted in Fig. 2.7. Also, it can be observed that the BLDC motor depicted in this figure is the outrunner type.

In Fig. 2.7, the first coil attracts the poles of the rotor (marked with green forces) and the second coil repels the poles of the rotor (marked with red forces), which leads into a desired bigger torque. [8], [7]



**Figure 2.6:** BLDC motor - construction [7]



**Figure 2.7:** BLDC motor - working principle [8]

### 2.2.6 Brushed vs. Brushless DC Motor

This section is inspired by [7], [9].

Both of these types of DC motors are common and widely used in industrial and robotics applications. Based on the requirements and specifications of the desired application, it can be decided whether to use a brushed or brushless DC motor. Let's look at some advantages and drawbacks of BDC and BLDC motors.

#### Brushed DC Motors

They have a simple design, they are cheap and easy to control. However, the biggest drawback is that their brushes eventually wear out due to a continuous movement which causes the need for regular maintenance and a replacement of brushes. Also, the brushes may cause sparking, noise, and they limit the maximum speed of the motor. The efficiency of BDC motors is usually only 75-80%.

#### Brushless DC Motors

BLDC motors are more reliable, more efficient (85-90%). Because of the lack of brushes, they are less noisy, and there are no brushes to wear out which means it requires less maintenance and no sparking is generated. Overall, BLDC motors are lighter and smaller than BDC motors with the same power output. Besides, since a computer is used to control the motor instead of mechanical brushes, the delivered speed and torque are very precise, which reduces the heat generation and energy consumption. But they are more expensive.

In conclusion, today, BLDC motors are more commonly used than BDC motors, especially due to a long life span, precision, and efficiency. However, BDC motors can still be found in, for example, the automobile industry and home appliances, where such high demands are not always required.

### 2.2.7 Stepper Motors

Stepper (STP) motors are brushless DC motors. They consist of 2 main parts: a stator and a rotor. The rotor is a permanent magnet or created from a ferromagnetic material, the stator is made of coils. STP motors can control the angular position of the rotor without a closed feedback loop, which means that they operate in an open-loop. The motor divides a full rotation into several equal steps, that is where the name comes from.

Depending upon construction [10], there are three types of STP motors:

- *Permanent Magnet Stepper Motor*
- *Variable Reluctance Stepper Motor*
- *Hybrid Stepper Motor - the most common*

Not all the types of stepper motors will be discussed. Let's look at one of these types, for instance, Variable Reluctance Stepper motor, and describe the working principle which is almost the same for all of them. Variable Reluctance Stepper motor is one of the simplest types and is depicted in Fig. 2.8.

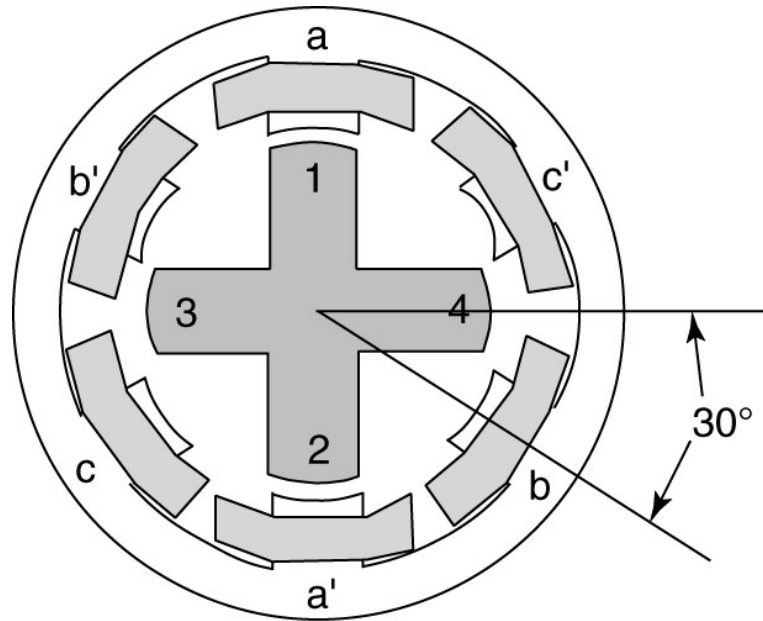
As can be seen in Fig. 2.8, this motor has 3 stator windings, which means 6 poles and 4 rotor teeth. When the coil "a" is energized, the respective teeth are attracted to it. Then if the coil "a" is de-energized and the next winding "b" is energized, the rotor will rotate CW<sup>4</sup>. If keeping energizing and de-energizing sequentially, the rotor will rotate continuously. At any point, only one pair of rotor teeth is aligned with the energized coil. In that case, the one-step size is 30°. To make it more precise, a method called half stepping can be used, it changes the one-step size to 15°. The half stepping means that the two coils next to each other are energized at the same time so as to move the rotor to a position in between them. Then only one coil is powered and then again the two next to each other. This ensures that the step angle is half of the size of the original one. [10], [12]

Some advantages of STP motors to mention, according to [13], [14], are:

- *High output torque at low angular velocity*
- *Precise positioning and repeatability of movement*

---

<sup>4</sup>CW = clockwise



**Figure 2.8:** Variable reluctance stepper motor [11]

- *Simple and robust construction resulting in good mechanical reliability*
- *Great response to starting, stopping, and reversing*

## 2.3 DC Motor Control and Servo-control

The ability to control a DC motor or any other motor is essential in creating any mechatronics system, for example, a robot. There are many tasks which concern the precise positioning and the speed of a certain mechanism. This can be obtained by simple devices, such as H-Bridge, PID Controllers, and so on. Let's look at them closely in the following chapters.

### 2.3.1 H-Bridge

This chapter comes from [15] and [16].

H-Bridge allows us to drive a DC motor in both directions, meaning forward and backward. In order to do that, the direction of a current must be controlled. H-Bridge is a simple circuit containing four switches. Thus, there are two to the four possible states but not all of them are relevant. The switches are usually made of series of bipolar transistors that route the current. Let's look at the electric circuit of an H-Bridge in Fig. 2.9. In this picture, it can be seen that the configuration resembles the letter H. The transistors on the top of the circuit are PNP type and are connected to a power supply. The

bottom ones are NPN types and are connected to the ground. When choosing the right transistor, it must be ensured that the transistor can handle enough current. For instance, if the motor takes 2 Amperes, the transistor needs to withstand at least that value of current; otherwise, it will burn out. In the middle, there is a motor that is desired to be controlled. The transistors in Fig. 2.9 are represented in Figures 2.10 and 2.11 as switches.

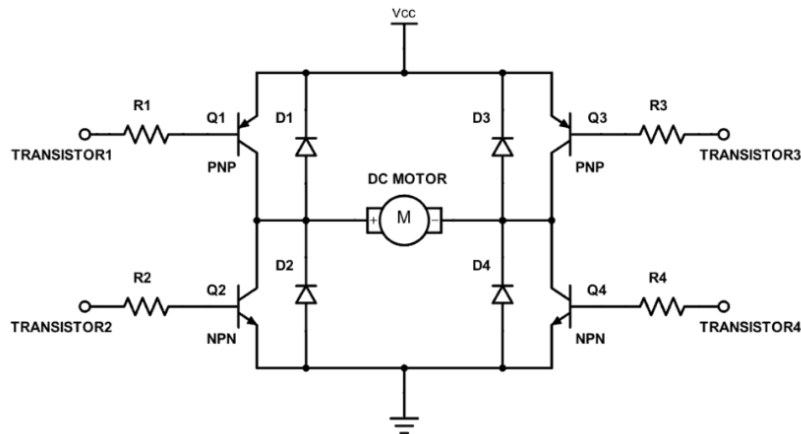


Figure 2.9: H-Bridge [15]

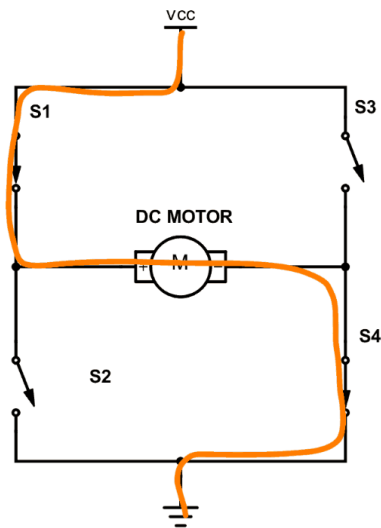


Figure 2.10: H-Bridge - forward movement [15]

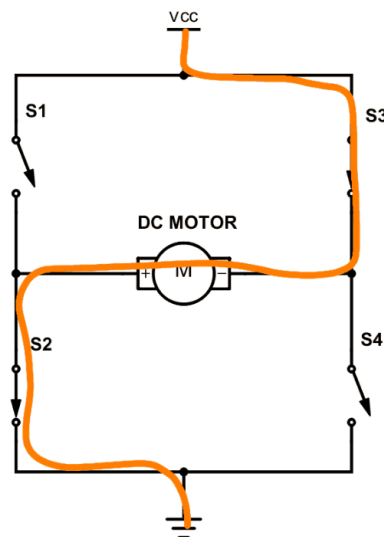


Figure 2.11: H-Bridge - backward movement [15]

Each of the switches can be closed or opened independently, but of course, there are some limitations. If switches S1 and S4 are closed, the left lead is connected to  $V_{CC}$  and the right lead is grounded. This results in the motor

starting to rotate in one direction, let's say forward, as demonstrated in Fig. 2.10. On the contrary, if switches S2 and S3 are closed, the motor gets energized in the reverse direction; therefore, it will spin in the opposite direction, meaning backward, which is illustrated in Fig. 2.11. If S1 and S3 or S2 and S4 are closed at the same time, both sides of the motor are seeing the same voltage, it is called a brake state, which means that the short circuit damping is used to slow down the motor. If one or none of the switches are closed, it is called a coast state, the voltage is whatever it wants to be, but no current is provided. Thus, the motor does not spin. Either both S1 and S2 or both S3 and S4 should never be closed simultaneously because it would be a short circuit from a positive voltage to the ground which would cause the burnout of the H-Bridge. Altogether, there are 9 possible states for an H-Bridge to be in, shown in Table 2.1.

S1	S2	S3	S4
close	open	open	close
open	close	close	open
close	open	close	open
open	close	open	close
close	open	open	open
open	close	open	open
open	open	close	open
open	open	open	close
open	open	open	open

**Table 2.1:** Allowed combination of switches

As expressed in Fig. 2.9, there are four diodes connected in the opposite direction than transistors. The diodes are called snubber diodes and are used to limit voltage transients in electrical systems running on DC current. While the motor is running, there is no current flowing through the diodes. But when the motor is turned off, there is a spike of voltage and then the current starts to flow through the diodes in order to not destroy the transistors. This means that the stored energy, that is generated by the motor, is dissipated through the snubber diodes from the circuit. [17]

### 2.3.2 PWM

The resources used for this section are [18], [19], [20].

PWM stands for Pulse Width Modulation. PWM is used to control the speed of a DC motor by varying the width of the pulse while keeping a constant frequency.

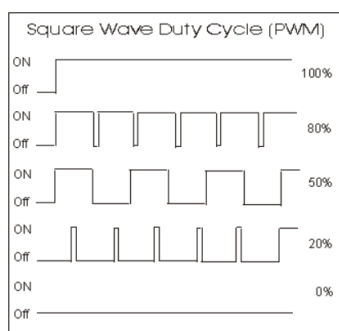
If the power of the DC motor is turned on, the motor does not reach a full speed immediately, but it takes some time before the motor runs at the full speed. If the power is switched off, the motor will start to slow down. If keeping switching the power on and off, the motor will run at some speed in between 0 speed and the full speed. This is exactly how the PWM works. PWM is a series of on and off pulses that create a square wave. Based on the width of on pulses compared to off pulses, the motor will receive a respective amount of voltage. As a consequence, the speed will change accordingly. The ratio of the on time to the time of the whole period is called the duty cycle, as demonstrated in Fig. 2.12. The duty cycle is always between 0 and 100 % and can be calculated, based on [20] like:

$$D = \frac{t_{on}}{T}, \quad (2.3)$$

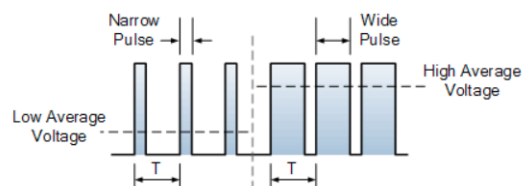
where  $t_{on}$  is a time for how long is the power on and  $T$  is the period. The lower the duty cycle, the lower the power that the motor receives. Regarding the switching speed, [18] states that the switching is usually done from a few kHz to tens of kHz for a motor drive.

As can be observed in Fig. 2.13, the left side waveform has a low average voltage because the width of on pulses is narrower than the width of off pulses. The right side waveform is exactly the opposite case which results in a high average voltage. Of course, the higher the average voltage, the bigger the speed and likewise.

While transitioning between on and off states, the voltage and the current change slightly; therefore, some power is dissipated. But since the time between on and off states is very small (in nanoseconds) compared to the fully on and fully off states, the average power loss is low.



**Figure 2.12:** Duty cycles [18]



**Figure 2.13:** Pulse width modulated waveform [19]



## 2. THEORY

---

PWM converts an analog signal to a digital signal. If the control voltage  $U_{ctr}$  is above the delta voltage  $U_D$ , then the power is switched on. Likewise, if  $U_{ctr}$  is below  $U_D$ , the power is switched off. This creates the sequence of square wave pulses which is the output voltage  $U_{out}$ . This can be observed in Fig. 2.14, where it is quite clearly depicted. Note that if  $U_{out} = 0$ , and  $t_{on}$  equals to  $t_{off}$ , then the duty cycle would be  $D = 0,5$ .

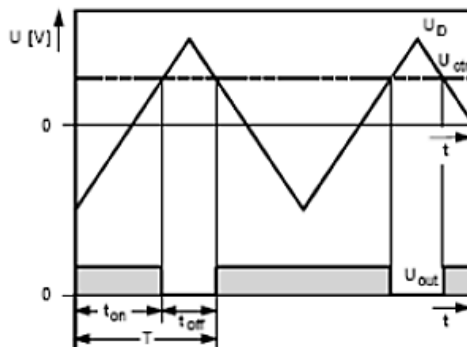


Figure 2.14: Operation of PWM [20]

PWM is often paired together with H-Bridge to form the basic building element of a motor to control the current in the winding.

### 2.3.3 PID type Controllers

This chapter and the following subchapters about different types of controllers are based on [21], [22].

Controllers, in general, can be divided [21]:

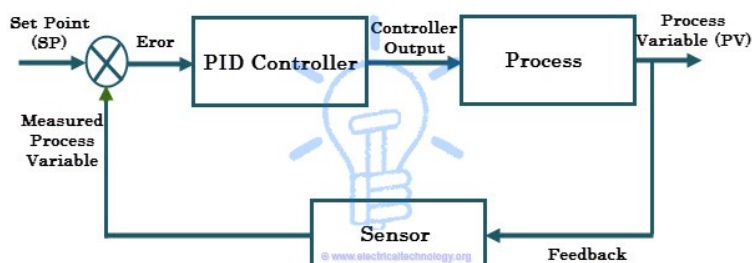
Based on:

- *Energy supply*
  - *Direct - no need of auxiliary energy*
  - *Indirect - there is a need of auxiliary energy (hydraulic, pneumatic, electric)*
- *Output waveform*
  - *Continuous*
  - *Discontinuous*
- *Linearity*
  - *Linear*
  - *Nonlinear*

PID controllers are indirect (mostly electric), continuous, and linear. PID consists of three parts: **P** means proportional, **I** stands for integral, and **D** is derivative. Depends on which components the regulators consist of, there are [21]:

- *P-Controller*
- *I-Controller*
- *PI-Controller*
- *PD-Controller*
- *PID-Controller*

These variants of controllers are used for different applications to get the best results. They are used to control a system. Let's look separately at each type of controllers in the following chapters.



**Figure 2.15:** A typical PID control system [22]

PID controllers are very widely used in various fields, also in automation and robotics. A PID control system is illustrated in Fig. 2.15. Assuming that the temperature in a room is to be controlled. The setpoint is the desired value, the process variable is the actual value (temperature in the room). The process variable is measured by a sensor, in this case, it would be probably a thermocouple. An error is a difference between the setpoint and the process variable that causes the actuator to act in a certain way, based on the type of the regulator, so as to achieve the desired temperature in the room. This whole system is known as a closed-loop feedback system.

### 2.3.3.1 Ideal P-Controller

An ideal P-controller or a proportional controller simply sets the control output  $u$  proportional to the error  $e$ . The error is the difference between the setpoint  $w$  and the process variable  $y$ :

$$e = w - y \quad (2.4)$$

The behavior of the P-controller can be described by the relation:

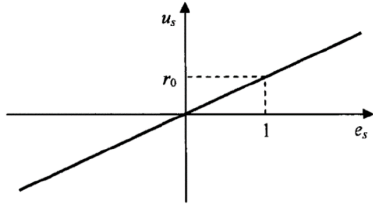
$$u(t) = r_0 \cdot e(t), \quad (2.5)$$

where  $r_0$  is the proportional gain of the P-controller.

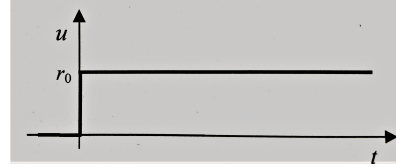
The transfer function  $R(s)$  of the ideal P-controller is

$$R(s) = \frac{U(s)}{E(s)} = r_0 \quad (2.6)$$

Fig. 2.16 depicts the static characteristics of the P-controller. Since it has a static characteristic, the whole system is static. Also, the P-controller cannot provide zero error state. Hence, there will always be a steady state error in the response of this controller. The response to a unit step change, which is called a transient response or a step response, is shown in Fig. 2.17.



**Figure 2.16:** A static characteristic of P-controller [21]



**Figure 2.17:** Transient response of the ideal P-controller [21]

### 2.3.3.2 Ideal D-Component

Only the D-controller on its own cannot be used because it reacts only to changes of  $e$ . Hence, when the error stabilizes, the derivative term equals zero. However, the derivative component is useful for predicting how fast the process variable will change in time and based on that the controller can adjust the output to the rate of change. The bigger the derivative constant is, the more aggressively the controller reacts, which is not usually desired. Thus, the derivative constant is mostly a small number as the derivative response is very sensitive to the noise in the process variable. If a big derivative constant is chosen, it leads to a very high output even for a small amount of noise.

The ideal D-component can be described by this equation where it is clear that the output equals to a derivative constant multiplied by the rate of change of error:

$$u(t) = r_D \cdot e'(t) \quad (2.7)$$

or

$$u(t) = r_0 \cdot (T_D \cdot e'(t)), \quad (2.8)$$

where  $T_D$  is the time derivative constant and  $r_D$  is the derivative constant.

The transfer function of the D-component is:

$$R(s) = r_D \cdot s. \quad (2.9)$$

### 2.3.3.3 Ideal I-Controller

An ideal I-controller or an integral controller sets the control output  $u$  proportionately to the integral of the error  $e$  so the behavior is expressed like:

$$u(t) = \frac{1}{T_I} \cdot \int_0^t e(\tau) d\tau + u(0) \quad (2.10)$$

or

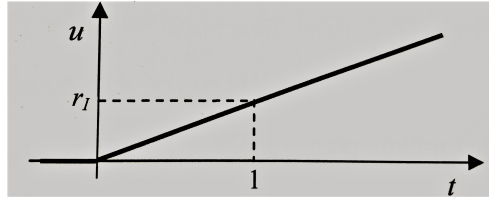
$$u(t) = r_I \cdot \int_0^t e(\tau) d\tau + u(0), \quad (2.11)$$

where  $T_I$  is the time integral constant and the inverse value of that,  $1/T_I$ , is  $r_I$  which is called the integral constant.

The transfer function  $R(s)$  of the ideal I-controller equals to:

$$R(s) = \frac{U(s)}{E(s)} = \frac{r_I}{s} \quad (2.12)$$

The I-controller is used to drive the error towards zero. The integral component sums the error over a period of time until it is eliminated. The controller responds even to a small error.



**Figure 2.18:** Transient response of the ideal I-controller [21]

When considering equation (2.10) or (2.11), it is clear that the steady state ( $u'(t) = 0$ ) can be achieved only if the steady state error equals to

zero. This system is astatic which means it does not have a static characteristic, but it has a transient response, demonstrated in Fig. 2.18. When comparing the transient response of the P and I-controller, it can be seen that the P-controller responds immediately, but there is always a steady state error. On the other hand, the I-controller responds slowly but maintains a zero steady state error.

### 2.3.3.4 Ideal PI-Controller

When combining P and I terms, the PI-controller is created which can be described by:

$$u(t) = r_0 \cdot e(t) + r_I \cdot \int_0^t e(\tau) d\tau + u(0) \quad (2.13)$$

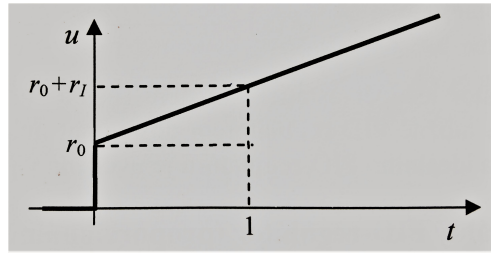
or

$$u(t) = r_0 \cdot (e(t) + \frac{1}{T_I} \cdot \int_0^t e(\tau) d\tau) + u(0). \quad (2.14)$$

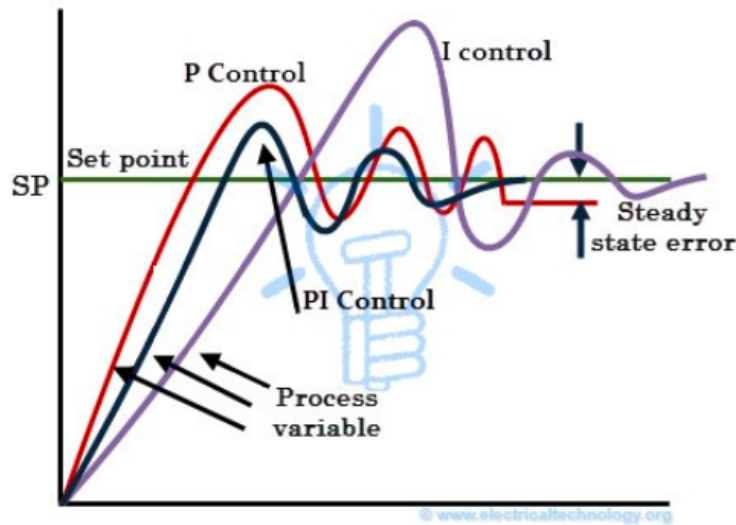
The transfer functions of P and I controllers are known; therefore, the transfer function of the PI-controller is:

$$R(s) = \frac{U(s)}{E(s)} = \frac{r_0 \cdot s + r_I}{s} = r_0 + \frac{r_I}{s} \quad (2.15)$$

The PI-controller is also an astatic system. Hence, it does not have a static characteristic, but the zero steady state response can be achieved. Since there is the P term present, the response of this system is faster than considering only I-controller. The transient response is depicted in Fig. 2.19. This type of controller is used in many industrial applications. To compare responses of P, I, and PI-controller, see Fig. 2.20.



**Figure 2.19:** Transient response of the ideal PI-controller [21]



**Figure 2.20:** Responses of P-controller, I-controller, and PI-controller [22]

### 2.3.3.5 Ideal PD-Controller

A PD-controller contains the proportional and the derivative term. By derivating the error  $e$  in the actual time  $t$ , it can predict how will the error behave in the future and based on that, it can change the controller output  $u$ .

The ideal PD-controller can be described by this equation:

$$u(t) = r_0 \cdot e(t) + r_D \cdot e'(t) \quad (2.16)$$

or

$$u(t) = r_0 \cdot (e(t) + T_D \cdot e'(t)), \quad (2.17)$$

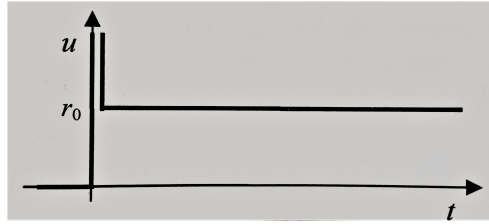
where  $T_D$  is the time derivative constant and  $r_D$  is the derivative constant.

The transfer function of the PD-controller is:

$$R(s) = r_0 + r_D \cdot s. \quad (2.18)$$

In steady state, the derivative equals to zero. For this reason, the D-term does not affect it. Hence, the static response is the same as for the P-controller that is shown in Fig. 2.16. This system is static since it has a static characteristic; therefore, a zero steady state error cannot be achieved.

When looking at the transient response of the ideal PD-controller in Fig. 2.21, it can be observed that because of the D-term, there is a big spike at  $t = 0$ . This spike goes ideally to infinity and then comes back to a constant value  $r_0$ .



**Figure 2.21:** Transient response of an ideal PD-controller [21]

A lot of industrial applications use very small derivative time  $T_D$  because the derivative response is sensitive to a noise in the process variable signal. This can result in producing high and quick changes in the controller output and leading to an unstable system. In order to prevent these undesired fast changes, filters are used.

### 2.3.3.6 Ideal PID-Controller

By combining proportional, integral, and derivative terms, the PID-controller is formed. It can be described by the following equation:

$$u(t) = r_0 \cdot e(t) + r_I \cdot \int_0^t e(\tau) d\tau + r_D \cdot e'(t) + u(0) \quad (2.19)$$

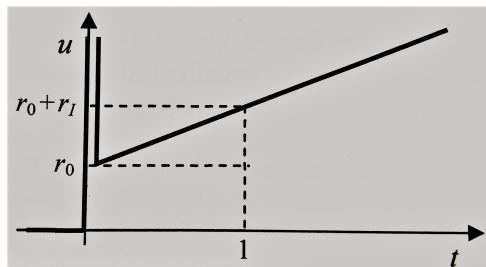
or

$$u(t) = r_0 \cdot (e(t) + \frac{1}{T_I} \cdot \int_0^t e(\tau) d\tau + T_D \cdot e'(t)) + u(0). \quad (2.20)$$

When derivating the equation 2.19, it is easy to get the transfer function of the PID-controller which is:

$$R(s) = \frac{r_0 \cdot s + r_I + r_D \cdot s^2}{s} = r_0 + \frac{r_I}{s} + r_D \cdot s. \quad (2.21)$$

Since the PID-controller contains the I-term, the system is astatic; thus, there is no static characteristic. Again, the steady state error equals zero. The transient response is demonstrated in Fig. 2.22.



**Figure 2.22:** Transient response of the ideal PID-controller [21]

Another step in using the PID-controller is to be able to tune it properly. Tuning means finding the right gains of proportional, integral, and derivative parameters so as to get the optimal response and achieve the desired value. There are different methods of tuning, let's talk about them closely in the following chapter.

## 2.4 Tuning of PID Controllers

A lot of different tuning methods for PID controllers have been developed in the last couple of decades. All the methods have their pros and cons. But the more information is known about the system, the easier and more precise the tuning is.

The optimal PID controller is quickly responsive, has minimal overshoot, has no steady state error, and is stable. Therefore, as said before, tuning methods are used to find the optimal proportional, derivative, and integral constants in order to get the desired response of the system. There is no one single way how to tune the PID controller, it depends on the characteristics of the system. Whether the system is linear or non-linear, whether it has a minimum or non-minimum phase, and whether there is a big delay or a manageable amount of delay. [23]

There are two general ways how to approach the tuning, either it is based on a model or a real physical system (hardware). From physical hardware, it is possible to develop a model.

In general, there are three methods of tuning [24]:

- *Manual Tuning*
- *Heuristic Methods*
- *Automatic Tuning*

### 2.4.1 Manual Tuning

The gains can be tweaked manually, it is an intuitive process for well-behaved systems since it is known how the P, I, and D terms affect the system. When increasing the proportional constant, it reduces the rise time. When increasing the derivative constant, it improves stability and decreases overshoot. [23]

There are three approaches to manual tuning based on [23]:

- *Trial and Error* - This is the simplest method of tuning a PID controller. It is based on the person, that tunes the controller, to guess the constants and run the system and just keep guessing values until he or she gets the desired response. Over the course of the time, some simple rules were developed. The first step is to start with just a  $P$  term, this means that the time integral constant is set to infinity (maximum), and the time derivative constant is set to 0. After the ideal proportional constant is found, the  $I$  term is added, and afterward, the  $D$  term is implemented as well.
- *Pole Placement Method* - finding the right poles in order to get the desired response. To make the system stable, all the poles must be on the left side of the complex plane.
- *Loop Shaping Method* - This method uses an open loop transfer function and the knowledge of Bode and Nyquist characteristics.

### 2.4.2 Heuristic Methods

Heuristic methods are slightly more sophisticated than the previous ones, but they still belong to the manual tuning methods. There is no need for a model when using heuristic methods. But these methods provide only an initial guess.

The two well-known heuristic methods are [23]:

- *Ziegler-Nichols Method*
- *Cohen-Coon Method*

Let's look only at the Ziegler-Nichols method since it is more used than the other one and it was also invented earlier.

#### Ziegler-Nichols Method

This method is one of the oldest and best-known tuning methods. It was developed by John G. Ziegler and Nathaniel B. Nichols in the year 1942.



The aim is to bring the system to the edge of stability and then record the critic values, such as the oscillation period  $T_u$  and the ultimate gain  $K_u$ . Using these values, it is possible to calculate the parameters of the controller. [25]

Ziegler-Nichols method follows these steps [25]:

1. First, let's eliminate the derivative and integral terms and focus only on the proportional term.
2. Bring the system to the oscillating stability limit by increasing the value  $r_0$ .
3. From the measured values, determine the period of oscillation, also called the ultimate period,  $T_u$ , and the ultimate gain  $K_u$ . Then calculate the optimal values for the type of the controller used, see table 2.2.

	$r_0$	$T_I$	$T_D$
<b>P</b>	$0,5 \cdot K_u$	Inf	0
<b>PI</b>	$0,45 \cdot K_u$	$T_u/1,2$	0
<b>PID</b>	$0,6 \cdot K_u$	$T_u/2$	$T_u/8$

**Table 2.2:** Calculation of controller's parameters using Ziegler-Nichols method [26]

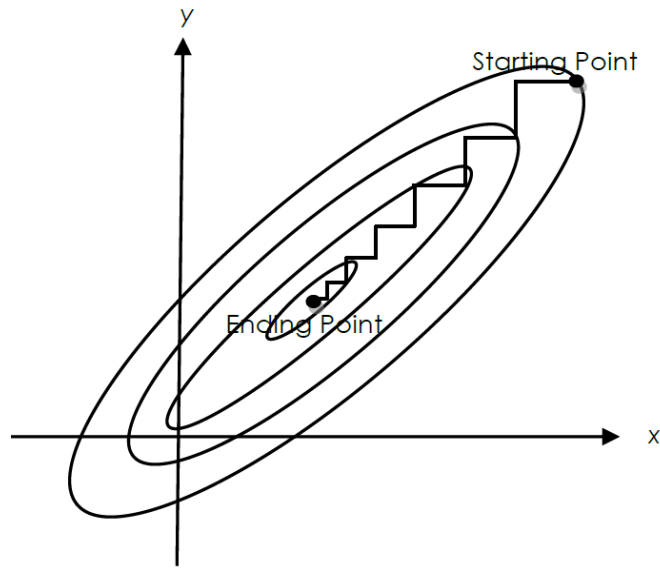
### 2.4.3 Automatic Tuning

Tuning, in general, depends on the operator's understanding of how tuning works and also on the dynamic behavior of the system and some other circumstances. Automatic tuning, also called autotuning, is a method that simplifies it for the operator because it let the controllers choose their own optimal parameters based on some sort of automated analysis of the controlled system's behavior. Autotuning basically works on the principle of satisfaction with the response of the controller. There is no one single technique that would be dominant. In most autotuning techniques, a mathematical model of the system is included. Autotuning can even run real-time and be constantly adjusting the gains. [23], [27]

Nowadays, most of the controllers have some kind of autotuning implemented inside of them. What it does is that the PID controller observes how the system behaves, responds to changes in setpoints and deals with disturbances. Based on all this knowledge, the autotuning software calculates appropriate P, I, and D constants and set them to the PID controller to get the optimal set of controller's parameters. One of the used autotuning methods is the gradient descent technique. [24]

### 2.4.4 Gradient Descent Method

In general, a gradient determines the direction of the greatest growth of a function and it is a first partial derivative of the given function. In the case of the gradient descent technique, as the name implies, the direction of the greatest descent is important. Gradient descent method is an algorithm used to reach a minimum point for the specific function because it is wanted to minimize the difference between the actual value and the desired value. The algorithm takes the starting point of the function and converts the solution to the negative direction of the gradient in order to obtain the minimum point of the function, as illustrated in Fig. 2.23. [26]



**Figure 2.23:** The steepest descent direction to reach a point of local minima [26]

The gradient can be expressed as [28]:

$$\text{grad } f(\mathbf{x}) = \nabla f(\mathbf{x}) = \vec{g}(\mathbf{x}), \quad (2.22)$$

where the  $\vec{g}$  function is a vector that contains partial derivatives [28]:

$$\vec{g}(\mathbf{x}) = \left[ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]^T. \quad (2.23)$$

The gradient can be calculated analytically or if it is too time-consuming or not possible at all, it can be calculated numerically as following [28]:

- *Single-sided difference:*

$$\frac{\partial f}{\partial x_1} = \frac{f(x_1 + \Delta x_1, x_2, \dots) - f(x_1, x_2, \dots)}{\Delta x_1} \quad (2.24)$$

- *Central difference:*

$$\frac{\partial f}{\partial \mathbf{x}_1} = \frac{f(\mathbf{x}_1 + \Delta \mathbf{x}_1, \mathbf{x}_2, \dots) - f(\mathbf{x}_1 - \Delta \mathbf{x}_1, \mathbf{x}_2, \dots)}{2\Delta \mathbf{x}_1} \quad (2.25)$$

- *And many other approaches*

The gradient method finds the minimum point faster than any other non-gradient technique because it is applied in the steepest direction. Consequently, the values of the function decrease at the fastest rate. It is an iterative process according to the equation 2.26 [26]:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \lambda_i \cdot \nabla f(\mathbf{x}) = \mathbf{x}_i - \lambda_i \cdot g(\mathbf{x}_i) \quad (2.26)$$

at which  $\lambda_i >$  satisfies:

$$f(\mathbf{x}_i - \lambda_i \cdot g(\mathbf{x}_i)) = \min_{\lambda_i > 0} f(\mathbf{x}_i - \lambda_i \cdot g(\mathbf{x}_i)), \quad (2.27)$$

where  $\lambda$  is the step size,  $\nabla$  denotes the gradient operator of the function, and  $g(x_i)$  indicates the gradient at the current point.

The algorithm tries to find the minimum value using this directional derivative [26]:

$$\frac{d}{d\lambda_i} f(\mathbf{x}_{i+1}) = \nabla f(\mathbf{x}_{i+1})^T \cdot \frac{d}{d\lambda_i} \mathbf{x}_{i+1} = -\nabla f(\mathbf{x}_{i+1})^T \cdot g(\mathbf{x}_i) \quad (2.28)$$

When reaching the local minimum, the equation 2.28 equals to zero. By applying the step size  $\lambda$  to the function  $\nabla f(X_{i+1})$  and  $g(X_i)$  at each iteration, the magnitude and the direction of the gradient is every time updated. That results in the directional function creating the zigzag pattern shown in Fig. 2.23.

It is important to make sure that the function diminishes at each iteration; therefore, the step size  $\lambda$  needs to be chosen accordingly. The  $\lambda$  must be bigger than zero. If a very small value for  $\lambda$  is chosen, then the optimization takes too much time. If the value is too high, it might skip the minimum value and start to diverge or oscillate. The right step size ensures stable conditions. There is also the question of whether to choose a fixed step size or gradually adjust it based on the algorithm progress. The parameters of the PID controller will be transformed by the algorithm until the stopping criteria met.

However, the goal of this technique is to find a global minimum, but the problem of the gradient descent method is that it may get caught in local minima. This method proceeds towards the steepest direction, which does not necessarily need to lead to the global minimum. Because in a local area, there might be a steeper direction for the method to follow. But when the system gets to a local minimum, it might get stuck in there and will not reach the desired solution. [26], [29]

### 2.4.5 PID Controller in the Laplace Domain

Let's convert the equation 2.20 from time domain to s-domain (Laplace domain) [30]:

$$U(s) = r_0 \cdot E(s) + \frac{r_I}{s} \cdot E(s) + r_D \cdot s \cdot E(s) \quad (2.29)$$

$$\frac{U(s)}{E(s)} = \frac{r_D \cdot s^2 + r_0 \cdot s + r_I}{s} = \frac{r_D \cdot (s^2 + \frac{r_0}{r_D} \cdot s + \frac{r_I}{r_D})}{s} \quad (2.30)$$

From equation 2.30, it can be observed that the PID controller has one pole at the origin (because of the single "s" in the denominator). But that is just the I part, that is integrated inside the PID controller. Let's look at the numerator. There are two zeros, their placement depends on the values of  $r_0$ ,  $r_I$ , and  $r_D$ . The whole tuning problem is where to place the zeros and how much gain to apply. But all of this applies to an ideal PID controller with the ideal derivative term. In the real PID controller, there is not a pure D term implemented, there is a filtered derivative instead. As a result, another pole arises. So the equation of the D term [30]:

$$r_D \cdot s \quad (2.31)$$

is replaced by the equation of the filtered derivative [30]:

$$\frac{r_D \cdot N}{1 + \frac{N}{s}} = \frac{r_D \cdot N \cdot s}{s + N}, \quad (2.32)$$

which makes it clear how the extra pole developed.

Now it is appropriate to use the methods that were mentioned in chapter 2.4.1, the pole placement, or the loop shaping. If it is known where the dominant poles should be in order to generate the desired response. It is possible to devise a PID controller that places the poles exactly at the spot where it is needed by adjusting the gains. This is the pole placement method. The loop shaping method deals with the frequency response, especially with the Bode plot. By adjusting the two zeros and the gains of the PID controller, it is possible to shape Bode plots to get the required frequency characteristics.

Both of these methods can be solved using math and the relevant equation.

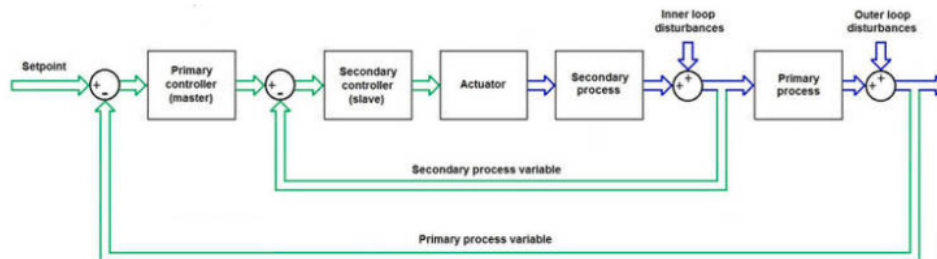
### 2.4.6 Cascade Control

This chapter is based on [31] and [32].

Cascade control is a widely used approach to improve process performance by reducing or eliminating a known disturbance. There are inner and outer loops

## 2. THEORY

and it is possible to distinguish them. They are called cascaded or nested loops because of the structure. The structure is to nest one feedback loop inside another feedback loop using two controllers. Let's describe a system with two cascaded loops, depicted in Fig. 2.24, but even more loops can be nested.



**Figure 2.24:** Cascade control block diagram [32]

From Fig. 2.24, it is noticeable that there are two controllers, two sensors, one actuator, and the processes are in series. The outer loop drives the setpoint of the inner loop, it means that the primary (master) controller generates the setpoint for the secondary (slave) controller. On the other hand, the inner loop generates a PV<sup>5</sup> that is the input of the primary process and affects the feedback path of the outer loop.

The inner loop works like a classic feedback control loop, it consists of a setpoint, a secondary controller, an actuator, and a process variable. The outer loop is also a feedback loop, except it uses the entire inner loop as the actuator.

Why design a system with cascaded loops and not a single loop [31]?

- *It can be easier to isolate problems.*
- *The inner loop can be tuned to respond quickly to local disturbances.*
- *The outer loop can be tuned more conservatively to reject sensor's noise and increase stability.*

The inner loop responds quickly and can detect the motor disturbances and adjust them so that the outer loop would not be affected by that. This allows the outer loop to be slower and respond to slow disturbances. Hence, when noise is detected by the outer loop, the response is not so aggressive as it would have been in a single loop system. It is much better to have a cascaded system with more loops targeting different setpoints and disturbances than a single loop system.

---

<sup>5</sup>PV = Process Variable

When the inner loop is 5-7 times faster than the outer loop, the loops can be tuned separately and the changes in the inner loop are so fast compared to the outer loop that the outer loop takes them as instantaneous. If the inner loop speed is about the same as the outer loop speed, then the tuning process is slightly more complicated because the inner loop performance affects the outer loop. The two controllers operating at the same speed might end up competing and driving the system unstable. There are 3 different ways to tune it [31]:

- *Iterative approach - Tune the inner loop with a guess and then tune the outer loop while the inner loop is running, change the inner loop guess and tune the outer loop again, and keeps iterating back and forth until the optimal response is found.*
- *Multi-input, Multi-output system - Both of the loops are tuned simultaneously.*
- *Using software to autotune the loops.*

The cascade control system can be, for example, created in MATLAB/Simulink.

Cascade control has some downsides. The extra sensor and controller increase the overall price. Also, there is twice as much tuning. However, cascade control performs better than the single loop control, but it is not worth the effort and costs for all applications.

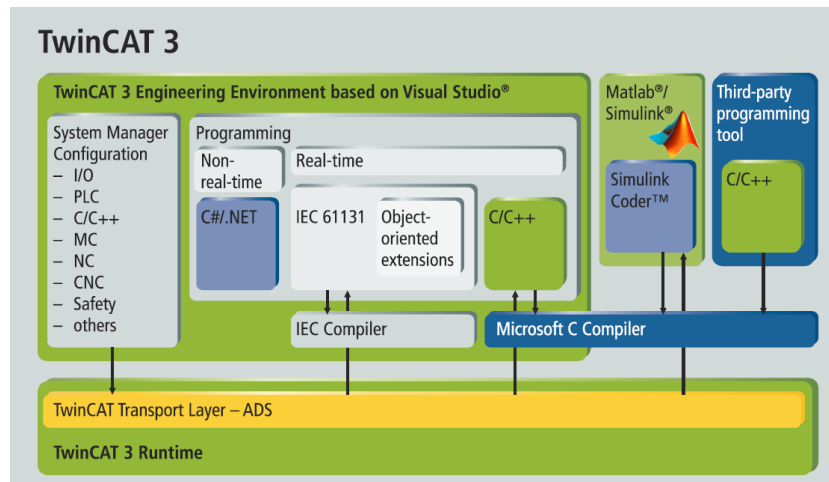
## 2.5 TwinCAT 3

TwinCAT 3 is a PC-based control software developed by Beckhoff Automation. This company claims that almost every kind of control application can be created with TwinCAT 3. To realize the applications, the user can access different programming languages, including PLC programming languages, the high-level languages C and C++ as well as MATLAB/Simulink. TwinCAT 3 philosophy is moved to modular control software which ensures that the complexity of modern machines is mastered and the engineering effort needed is decreased. It means that individual functions or machine units are considered as modules. TwinCAT 3 is deemed to be an eXtended Automation because it provides a combination of the latest IT technologies and scientific software tools with automation technology. TwinCAT 3 can be integrated into some of the existing software development environments such as Microsoft Visual Studio, which is used in the case of this thesis. Another feature that TwinCAT 3 provides is a real-time environment, where TwinCAT modules can be loaded, executed, and administrated. Each of the modules can be developed using different programming languages as mentioned above. Thus, this approach is convenient for many users and developers. The TwinCAT Engineering Envi-

## 2. THEORY

---

ronment and the modular structure of the TwinCat 3 Runtime Environment can be observed in Fig. 2.25 and Fig. 2.26 respectively. [33]



**Figure 2.25:** TwinCAT 3 Engineering Environment [33]

A further great feature of TwinCAT 3 is that it supports multi-core CPUs<sup>6</sup>. Thus, different modules and their tasks can be assigned to different cores of the CPU. This provides a great performance of the newest PCs that can be used up to their limits. [33]

The requirement of TwinCAT 3 are [33]:

### **TwinCAT 3 XAE (Engineering):**

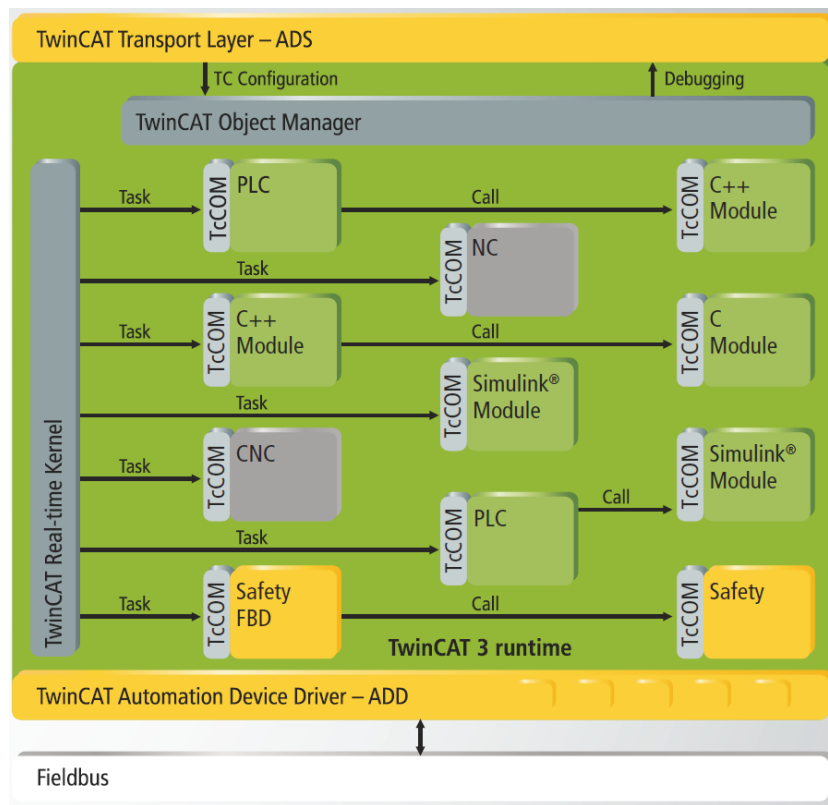
- *Windows XP with Service Pack 3 (x86) or Windows 7 (x86 or x64)*
- *Processor running at 1.6 GHz or higher*
- *2 GB RAM*
- *3 GB free hard disk space*
- *Graphics adapter supporting DirectX9, running at a minimum resolution of 1024x768*

### **TwinCAT 3 XAR (Runtime):**

- *x86-based Windows operating system: Windows XP with Service Pack 3, Windows 7, Windows Embedded Standard 2009, Windows Embedded Standard 7*

---

<sup>6</sup>CPU = Central Processing Unit



**Figure 2.26:** Modular structure of the TwinCAT 3 Runtime Environment [33]

### 2.5.1 Ethernet

Ethernet was created in the '80s to connect computers and other devices in a local environment. It is a communication standard IEE 802.3 that was standardized in 1983 by the Institute of Electrical and Electronics Engineers (IEEE). The local environment, also called LAN <sup>7</sup>, could be an administrative building, a home, or any other restricted area. LAN provides a space where multiple devices can be connected to share information with each other. Ethernet is a wired system. As a result, the devices are connected via cables. A coaxial cable used to be used. Nowadays, the technology was shifted to using twisted pair copper wiring and fiber optic wiring. The twisted pair cables, specifically CAT6a and CAT7, can handle speed up to 10 Gbps. They can work in either half or full-duplex mode. As the name suggests, the half-duplex mode allows data to be transferred in only one direction, and on the other hand, the full-duplex mode transmits data in both directions simultaneously. A fiber optic cable helps Ethernet to travel further at a higher speed. Overall,

<sup>7</sup>LAN = Local Area Network



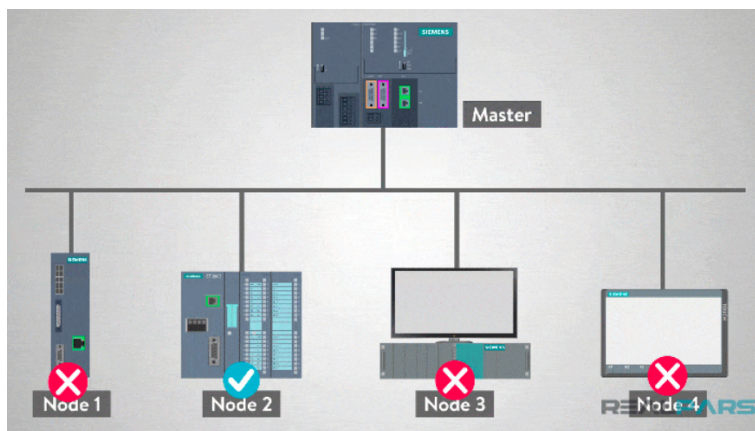
Ethernet is affordable, easy to install, and if it is connected to the internet, the possibilities are endless. [34]

### 2.5.2 EtherCAT

EtherCAT means Ethernet for Control Automation Technology. This technology is based on Ethernet but specifically targets an industrial automation. Unlike Ethernet, EtherCAT focuses more on [35]:

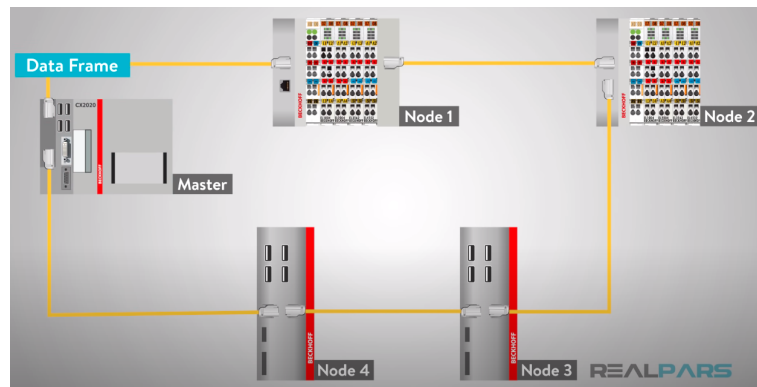
- *Quick response times => higher speed*
- *Minimal data requirements for each device => reduced data traffic*
- *Low costs of implementation*
- *More accurate data due to the distributed clock mechanism*

Ethernet works in a master/slave configuration, which means that the data are transferred only when the master or client requests it. It is not the most practical solution since it sends the frame of data to and from most of the devices, individually, even if it is meant to be only for a particular node, as depicted in Fig. 2.27, where the data frame is relevant only to the Node 2, but it reaches all the nodes. [35]



**Figure 2.27:** Ethernet - bus topology [35]

With EtherCAT, it works slightly different. Only the master is allowed to send the data frame that goes through the whole node network, as could be seen in Fig. 2.28. Each EtherCAT device can take and add some information to the data frame, as it passes through the nodes, which ensures real-time capability. Each device is equipped with two Ethernet ports, the first is the receiving one. The second port connects to the following node to provide the flow of the data. This approach assures the main advantage of EtherCAT which is that the data are processed on the fly. Of course, there is still a small

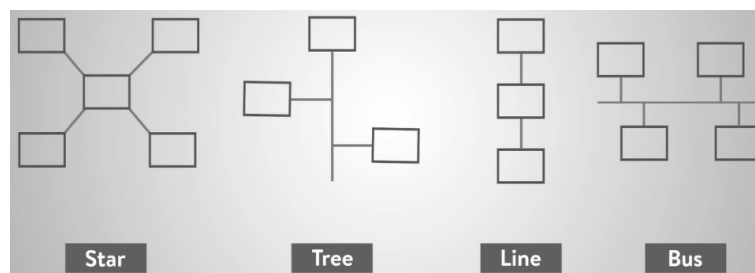


**Figure 2.28:** EtherCAT - ring topology [35]

delay, as the data frame passes through each node, but it is much faster than sending multiple frames via Ethernet. The only problem that can occur is that not all the devices can handle these high cycle times. Thus, the speed of the data transmission might need to be decreased which is possible. [35]

Another feature that EtherCAT provides is a distributed clock system for precise synchronization. As the data frame passes through devices, each node adds its timestamp to the data so then the master can calculate a delay for each node. With this mechanism, every single data transmission is very accurate. [35]

Based on the desired application, different topology can be realized. Ethernet's star, line, or bus topology can be used. Other than that, a tree, ring, or other topology could be implemented, see Fig. 2.29. The ring topology can be observed in Fig. 2.28. [35]



**Figure 2.29:** EtherCAT's network topology [35]

## 2.6 Beckhoff EL7342 Module

This chapter comes from the manual from the Beckhoff company [36], targeting the usage of the module EL7342.

### 2.6.1 Introduction

The EL7342 EtherCAT terminal is depicted in Fig. 2.30 and it enables direct operation of two DC motors. The speed and the position are determined by a 16-bit value from the automation device. When an incremental encoder is connected, a simple servo axis can be realized. There is also protection against overload and short-circuit. In Fig. 2.30, it can be seen that there are several LEDs<sup>8</sup> that indicate signal states of each of the two channels and provide a local diagnosis.

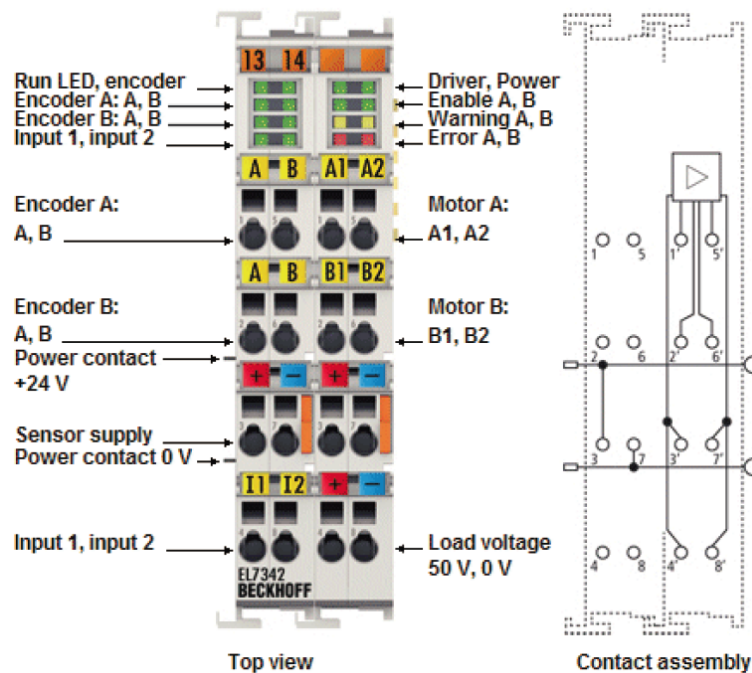


Figure 2.30: The EL7342 EtherCAT terminal [36]

<sup>8</sup>LED = Light Emitting Diode

## 2.6.2 Technical data

Technical data	EL7342
Number of channels	2 DC motors, 2 digital inputs, Encoder input
Rated load voltage	8 ... 50 V
Load type	DC brush motors, inductive
Output current without fan cartridge ZB8610	2 x 3.5 A (overload- and short-circuit-proof)
Output current with fan cartridge ZB8610	2 x 6.5 A (overload- and short-circuit-proof)
PWM clock frequency	30kHz with 180° phase shift each
Duty factor	0 .. 100 % (voltage-controlled)
Distributed Clocks	yes
Control resolution	max. 10 bits current, 16 bits speed
Supply voltage for electronic	via E-bus and power contacts
Electrical isolation	500 V (E-bus/field voltage)
Current consumption via E-bus	typ. 140 mA
Current consumption power contacts	typ. 70 mA
Current consumption sensor supply	typ. 20 mA
Encoder/input signal	Signal voltage "0": -3 V ... 1.5 V
	Signal voltage "1": 2.5 V ... 24 V
Nominal voltage of encoder signals	5 ... 24 V, 5 mA, single ended
Pulse frequency	400 000 increments/s, 4-fold evaluation
Weight	approx. 90 g
Permissible ambient temperature range during operation	0°C ... + 55°C
Permissible ambient temperature range during storage	- 25°C ... + 85°C
Permissible relative humidity	95 %, no condensation
Dimensions (W x H x D)	approx. 27 mm x 100 mm x 70 mm

Table 2.3: Technical data of the module EL7342 [36]

### 2.6.3 Technology

A compact Motion Control solution up to 200 W is integrated into the EL7342 DC motor terminal.

#### DC motor

Servo motors are more expensive than DC motors; therefore, when DC motors are operated with a sophisticated controller, they can replace servo motors. DC motors are easy to control because their speed is proportional to the voltage.

#### Two DC motor output staged for optimum use

The EL7342 EtherCAT terminal has a compact design. Using the terminal, a DC motor can be integrated simply into the control system and all the parameters are changeable via the fieldbus. The speed can be adjusted via the process data with the EL7342 terminal and it also contains a compensation of the internal resistance. This compensation maintains the desired velocity of the DC motor even for load changes.

#### Areas of application

There are two areas of application [36], both of these possibilities are depicted in Fig.2.31:

1. A simple controller with inexpensive processor power and low demands on the cycle time. By using the integrated travel distance control, the terminal can perform positioning drives independently without the use of NC <sup>9</sup>.
2. High-end positioning with integration in TwinCAT NC. When connected with the EtherCAT DC motor terminal, a DC motor is controlled under TwinCAT analogous to a servo terminal.

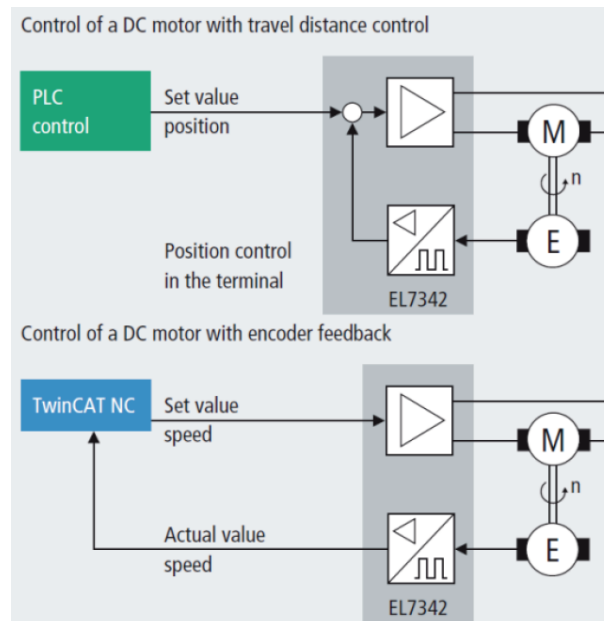
In order to accomplish tasks that require precise positioning, it is necessary to use a closed speed control loop with a feedback system.

Both of the approaches of possible applications that are expressed in Fig. 2.31 are implemented in the practical part to test them. For more information about how successful the methods are and to see their results go to sections 3.3 and 3.4.

The length of the cable between two EtherCAT devices must not be longer than 100 m because of the signal attenuation.

---

<sup>9</sup>NC = Numerical Control



**Figure 2.31:** Possibilities to implement control loops using the EL7342 terminal [36]

## 2.6.4 Basic Communication

### EtherCAT State Machine

The accessibility of individual functions depends on the state of the EtherCAT slaves, which is controlled via the EtherCAT State Machine. In each state of the slave, the EtherCAT master needs to send specific commands to the device.

Let's look at the states and their differences [36]:

- *Init* - The EtherCAT slave is in this state right after switching on and no communication is possible.
- *Pre-Operational* - When transitioning from *Init* state to *Pre-Op*, the mailbox is initialized; thus, the mailbox communication is possible in this state. Then the master initializes the sync channels for process data.
- *Safe-Operational* - Here are both the mailbox and the process data communication allowed. But only the input data are updated, the output data stay at a safe state.
- *Operational* - Before accessing this state, the slave transfers the output data of the master into its outputs. Again, process data and mailbox communication is available.

- *Boot* - This state can be reached through the *Init* state and is used for updating the slave firmware.

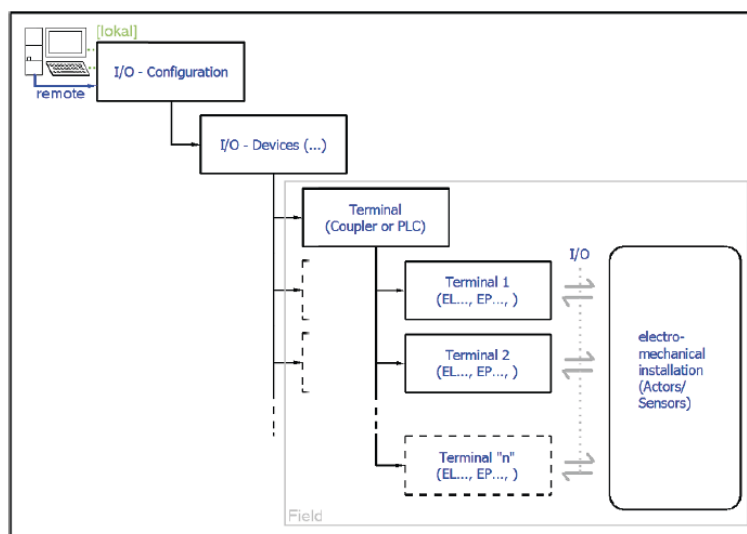
### The CoE Interface

The CoE Interface<sup>10</sup> is utilized for managing and adjusting parameters of the EtherCAT devices. These parameters are hierarchically arranged and can have different types, including integer or unsigned integer numbers, Boolean values, as well as string, and many others. The local CoE lists of slaves can be entered via EtherCAT by the EtherCAT master, either in write or read mode.

#### 2.6.5 Commissioning

TwinCAT is an environment where things are controlled in real-time. This environment maps the whole system and provides access to the controller. All inputs and outputs, both digital and analog, can be written or read directly.

When considering the relationship from user PC to the individual control elements, see Fig. 2.32



**Figure 2.32:** Relationship between user side (Commissioning) and installation [36]

The user can insert desired components, such as I/O<sup>11</sup> device, terminal, etc..

<sup>10</sup>CoE = CANopen over EtherCAT

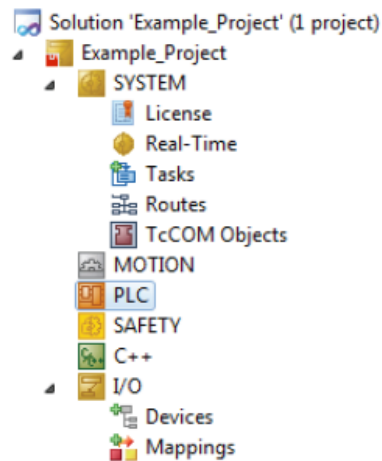
<sup>11</sup>I/O = Input/Output

### 2.6.5.1 TwinCAT 3

TwinCAT together, with Microsoft Visual Studio, shapes the development environment areas.

Once TwinCAT 3 is installed, it is possible to create a new project, which is captured in Fig. 2.33.

In general, there are two ways on how to use TwinCAT, it can operate in local or remote mode. In the case of this thesis, TwinCAT is used in local mode; hence, the next step is to add devices. This can be done by expanding "I/O" in the project folder explorer and then selecting "Devices". After that, right-click on "Devices" and select "Scan", the whole process goes through two stages. First, determine the devices. Second, determine the connected elements such as terminals, boxes, etc..



**Figure 2.33:** New TwinCAT 3 project [36]

### 2.6.5.2 TwinCAT Development Environment

TwinCAT 3 is an improved version of TwinCAT 2, so it includes the features that TwinCAT 2 has and also some additional ones.

Let's look at the features [36]: **TwinCAT 2**

- *Connects I/O to tasks in a variable-oriented manner*
- *Connects tasks to tasks in a variable-oriented manner*
- *Supports units at the bit level*
- *Supports synchronous or asynchronous relationships*
- *Exchange of consistent data areas and process images*
- *Integration of IEC 61131-3-Software-SPS, Software- NC and Software-CNC within Windows NT/2000/XP/Vista, Windows 7, NT/XP Embedded, CE*
- *More...*



**TwinCAT 3 (eXtended Automation) - additional features**

- *Visual-Studio-Integration*
- *Choice of the programming language*
- *Supports object oriented extension of IEC 61131-3*
- *Usage of C/C++ as programming language for real time applications*
- *Connection to MATLAB/Simulink*
- *Open interface for expandability*
- *Flexible run-time environment*
- *Active support of Multi-Core and 64-Bit-Operating system*
- *Automatic code generation and project creation with the TwinCAT Automation Interface*

---

## DC Motor Servocontrol

The DC motor 20 MPM 105 manufactured by company Moore Reed and Company Ltd is used for the practical part. Moore Reed and Company Ltd does not exist anymore, but the DC motor was dismantled from a robotic arm Stäubli PUMA 200. Stäubli is a Swiss company and PUMA is an abbreviation for "Programmable Universal Machine for Assembly", this robot can be seen in a top left picture of Fig. 3.1.

### 3.1 Encoder Replacement

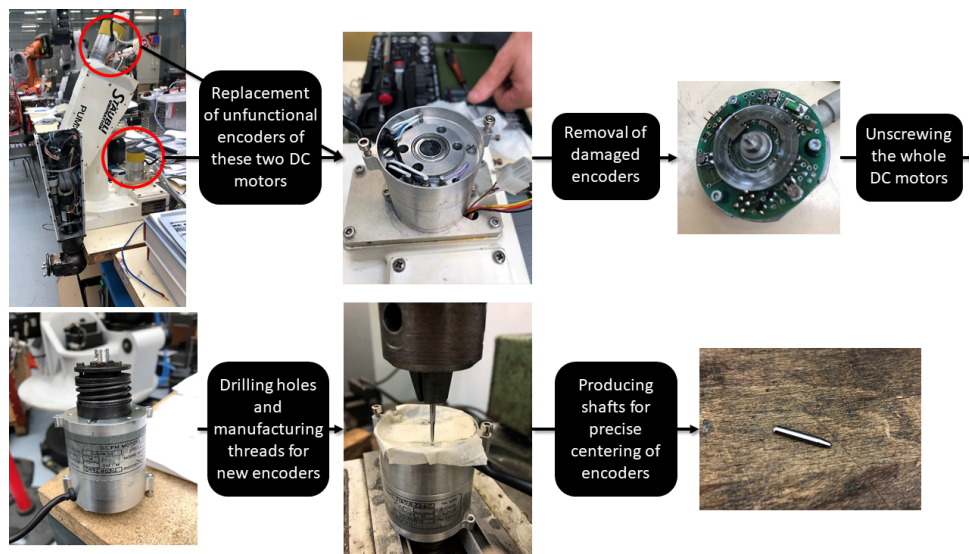
When beginning to approach the practical part of this thesis, some issues regarding position measuring were observed. It has been found that the encoders, that are attached to DC motors, do not function properly. Thus, the first step was to replace them with new ones which required some additional steps, that are captured in Fig. 3.1. Since the new encoders were smaller and had overall different dimensions, new holes for attaching encoders needed to be drilled. Inside the holes, threads were manufactured and also shafts with 2 mm diameter for centering were produced. Then, the last step was to mount the new encoders to the DC motors. All of the previously described steps are depicted in Fig. 3.1 and the result is shown in Fig. 3.2.

After completing all the steps, the motors with new encoders are fully functional again, meaning that the possibility of tracking the position is enabled.

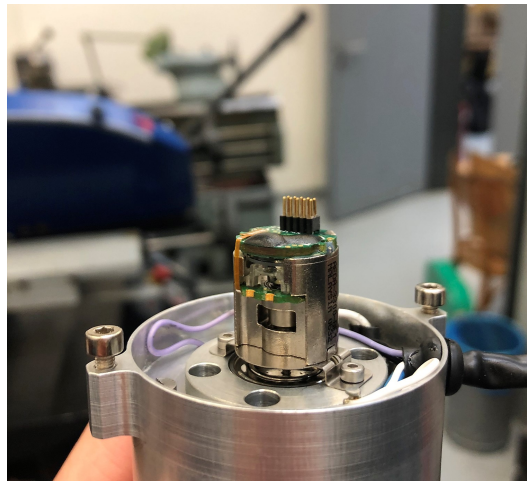
Let's look at the encoders' type and parameters closely in the following section.

### 3. DC MOTOR SERVOCONTROL

---



**Figure 3.1:** The process of replacing damaged encoders with new ones



**Figure 3.2:** The new encoders (type AEDA - 3300 TAM) attached to DC motors

#### 3.1.1 Encoders AEDA - 3300 TAM

This section is based on the manual of AEDA-3300 encoders [37].

AEDA-3300 encoders are three-channel optical incremental encoders produced by company Avago Technologies. Here are some features to be mentioned [37]:

- *Three channels output (quadrature A and B output with index channel)*

- *Resolution options from 600 to 20000 CPR* <sup>12</sup>
- *Cost effective*
- *-40°C to 125°C operating temperature*
- *Ultra miniature size - diameter: 17 mm*
- *Maximum 1 MHz operating frequency*
- *Maximum 12000 RPM* <sup>13</sup> *rotational speed*
- *Single 5V supply*
- *Integrated bearing stage for easy mounting*
- *Bottom-up or top-down mounting options*

These encoders are said to be very reliable and easy to install. They can withstand high temperatures and high operating frequencies. For this reason, they are used in many industrial automation and motion control applications.

The operation of the encoders is as follows. The LED emits light that goes through the code wheel and analog signals are generated. The analog signals are then converted to digital A, B, and index signals and their complements  $\bar{A}$ ,  $\bar{B}$ , and  $\bar{I}$ . There is always a 90 degrees phase shift between A and B signals. This phase difference can determine whether the motor rotates clockwise or counter-clockwise. When signal A leads signal B, the motor turns in a clockwise direction and vice versa. The index signal just indicates a reference position.

The encoder AEDA - 3300 TAM has its own specifications. T stands for Top-down with coupling plate mounting option and AM defines 3000 CPR.

## 3.2 Project Description

The goal of the practical part of this thesis is to develop the autotuning algorithm in MATLAB/Simulink to control the DC motor Moore Reed and Company Ltd 20 MPM 105. The algorithm is based on a gradient descent optimization in order to find ideal values of constants  $K_p$ ,  $K_i$ , and  $K_d$  of the PID-controller. Constants  $K_p$ ,  $K_i$ , and  $K_d$  are equivalent to  $r_0$ ,  $r_I$ , and  $r_D$ , that have been declared in chapter 2.3.3, respectively. Why is there a change in naming constants? It is simply because the PID-controller in TwinCAT is declared using  $K_p$ ,  $K_i$ , and  $K_d$  values. Hence, from now on, the new names

---

<sup>12</sup>CPR = Cycles Per Revolution

<sup>13</sup>RPM = Revolutions Per Minute

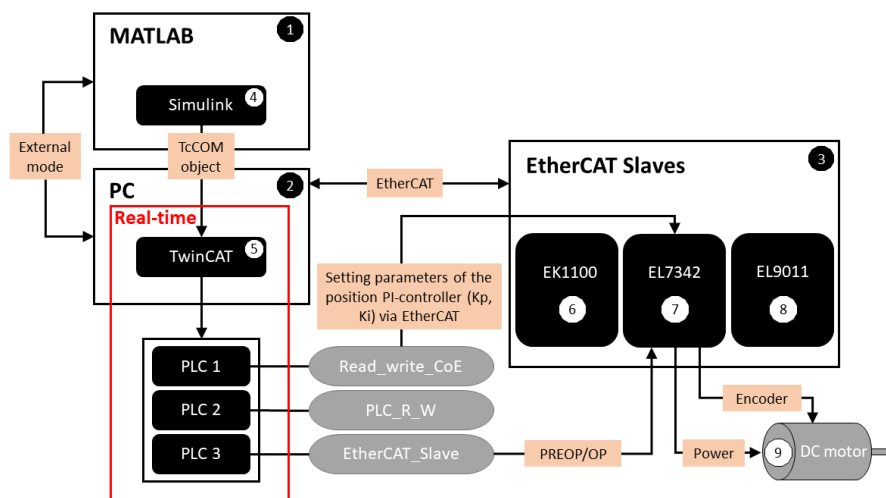
### 3. DC MOTOR SERVOCONTROL

of constants will be used to be consistent with TwinCAT. But the constants remain the same.

In order to transfer the data from the encoder to a PC, the EL7342 Beckhoff terminal is used. The Beckhoff terminal can be connected to TwinCAT and the aim of this third chapter is to create an interaction between TwinCAT and Simulink to be able to control the motor from Simulink.

### 3.3 Connection of TwinCAT and MATLAB/Simulink via the PLC

The first approach to implement control loops using the EL7342 Beckhoff terminal is control of a DC motor with travel position control, as illustrated in the top schema of Fig. 2.31. That means that all the controller's loops (position, velocity, and current) are inside the terminal. The current loop (the fastest loop) ensures a sufficiently fast current increase and thus, torque control. The speed loop regulates the velocity of the DC motor and the position loop, which is the slowest, ensures that the axis is kept in the desired position.



**Figure 3.3:** Diagram describing the project using 3 PLC programs

Fig. 3.3 shows relationships between all the main devices, their components, and software.

Simulink, marked in Fig. 3.3 by number 4, is an extension of MATLAB used for modeling and simulation of systems. The data can be then analyzed and visualized using MATLAB. To create a program in Simulink, block diagrams

are used. It is a very useful tool in control systems, electronics, robotics, signal processing, and many other areas.

TwinCAT is another convenient software that is applied to many automation applications, labeled by 5 in Fig. 3.3. To control the motor, PLC programs can be developed or axis can be added to the MOTION section in TwinCAT or it could be even a combination of both of them. TwinCAT communicates with the EL7342 Beckhoff terminal via EtherCAT, as shown in the diagram. It is important to mention that TwinCAT works in real-time.

When communicating between Simulink and TwinCAT, TE1400 TwinCAT Target for MATLAB/Simulink is used, it works based on the generated C/C++ code from the model in Simulink. TcCOM<sup>14</sup> modules or objects are created if the code generator works properly with TwinCAT Target. Special input and output function blocks, that are to be found in Beckhoff TwinCAT Target Library in Simulink, are used in order to generate TcCOM objects in TwinCAT. TcCOM objects are created when a project is built. [38]

Number 3 is used for marking EtherCAT slaves. There are three of them EK1100, EL7342, and EL9011 labelled by 6, 7, and 8, respectively. Let's explain each of them.

- *EK1100* - It is a coupler that connects the EtherCAT terminal EL7342 with EtherCAT. It is depicted in Fig. 3.4. [39]
- *EL7342* - This terminal is described in detail in chapter 2.6.
- *EL9011* - This is just a bus end cap demonstrated in Fig. 3.5. [40]

After deciding about the type of implementation, a project in TwinCAT is created. The EL7342 terminal is connected to the PC via the EtherCAT cable. The DC motor and the new AEDA encoder are wired to the terminal. Thus, the DC motor, the encoder, and the terminal can be added to the TwinCAT project to the I/O section. Now, all the data measured by the encoder and the terminal are provided to TwinCAT so the parameters can be graphed and adjusted.

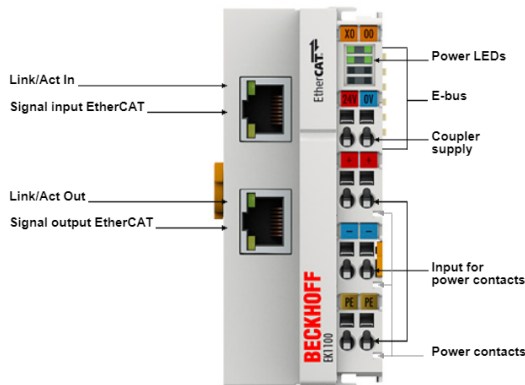
The second step is to set the controller to be a position controller and create PLC programs needed for effective control of the DC motor. For the needs of this project, 2 parameters ( $K_p$ ,  $K_i$ ) of the PI-controller are supposed to be controlled; therefore, 3 PLC programs are created inside TwinCAT. The first PLC program is called Read\_write\_CoE and is essential for setting parameters of the PI-controller. The second one is very simple and is called PLC\_R\_W. This program is responsible for taking the output from the EL7342 terminal and placing it as an input to TwinCAT. The name of the third PLC is EtherCAT\_Slave which was created for the purpose of switching states of the

---

<sup>14</sup>TcCOM = TwinCAT Object Model

### 3. DC MOTOR SERVOCONTROL

EL7342 terminal, specifically between PREOP and OP states. For more information about the EtherCAT State Machine see section 2.6.4. The main reason for developing the third PLC program is that the Read\_write\_CoE program can adjust parameters of the PI-controller only in the PREOP state and conversely, the DC motor can operate only in the OP mode. Consequently, there is a need for both of them.

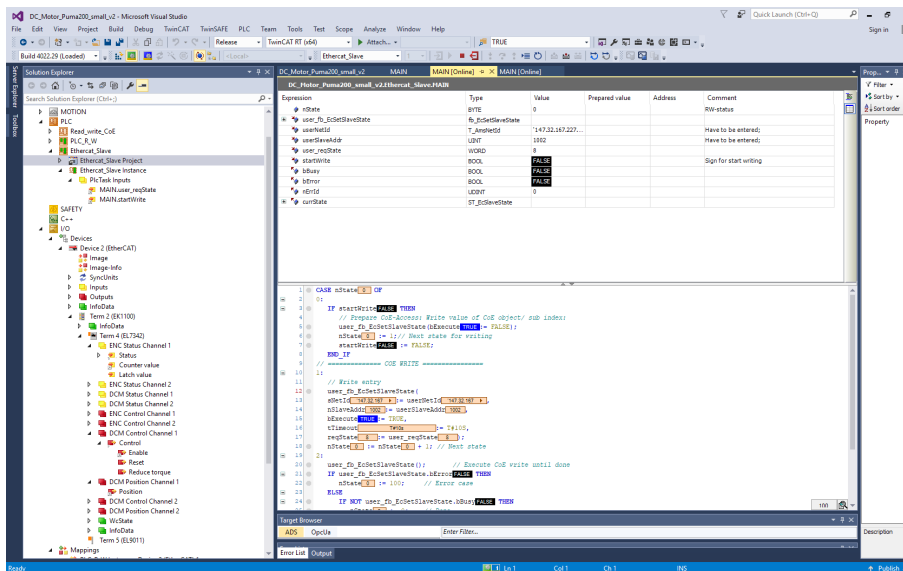


**Figure 3.4:** Beckhoff EK1100 EtherCAT coupler [39]



**Figure 3.5:** Beckhoff EL9011 Bus end cover [40]

In Fig. 3.6, one of the PLC programs and the whole TwinCAT environment of the project can be observed.



**Figure 3.6:** Project created in TwinCAT

In order to fulfill the goal of being able to control the DC motor from Simulink,

a Simulink program is created. The program in MATLAB/Simulink is captured in Fig. 3.7. There are two slider switches, one is for enabling and disabling the motor and the second one is for changing states of the terminal from PREOP to OP and for reset. The terminal needs to be reset after every single transition between PREOP and OP in order to be able to get to the next state and write parameters. It is so because the data are being written initially to the terminal and not directly to the PC. To change between states and reset, different numbers need to be inserted as a requested state and then the slider is dragged to "On" to write the data to the terminal and change the state. The specific numbers for each state are declared in PLC program EtherCAT\_Slave.

The numbers are:

- *OP* - 8
- *PREOP* - 2
- *reset* - 69

The very top function block ensures that Simulink can communicate with TwinCAT and vice versa. TC Module Input and Output blocks provide data transfer between Simulink and TwinCAT. The only problem occurred because of the restriction in the license of TE1400 TwinCAT Target where only 6 outputs or inputs to TwinCAT can be controlled via Simulink. Therefore, the part of the program that is responsible for adjusting the two parameters is disabled but can be seen in a light shade of gray.

As can be seen in Fig. 3.8 and in Fig. 3.9, the graphs are identical to each other. That is an indicator that both of the programs work properly because they are showing the same output curves. The regular rectangular pulse is the pulse that is desired to be achieved and the other one is supposed to be tuned to reach the set value. So far, it is clear that there is an overshoot and a steady state error, and both are to be ideally eliminated or at least reduced. The pulses oscillate around a value of 500 which was added in the Simulink program in order to eliminate problems with the initial oscillation around 0. There are possibilities to switch between states and to enable and disable the motor in the Simulink program. However, it is not realizable to change the parameters  $K_p$  and  $K_i$  via Simulink. Hence, this approach is not sufficient to accomplish the given goal. If the license provided more than 6 inputs/outputs, it would have been feasible to use this method.



### 3. DC MOTOR SERVOCONTROL

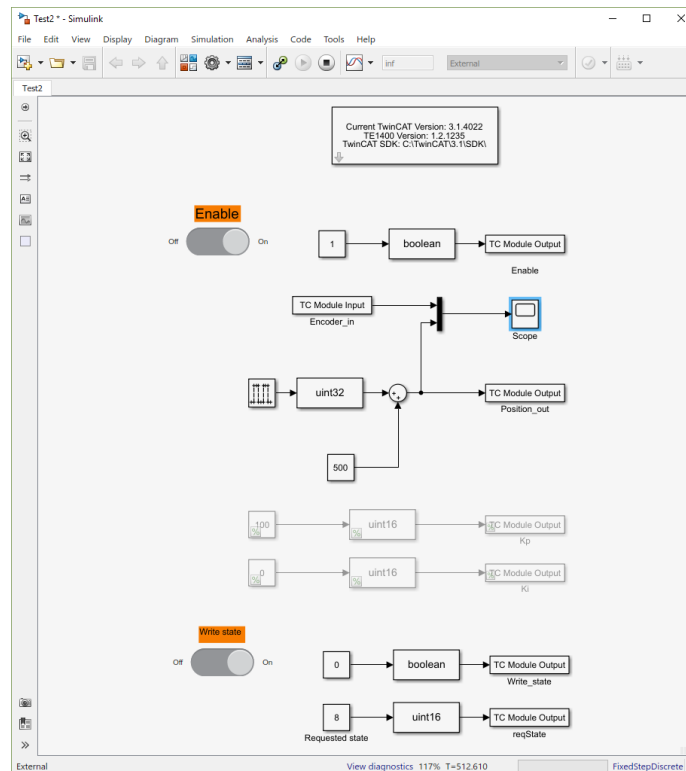


Figure 3.7: Program created in MATLAB/Simulink

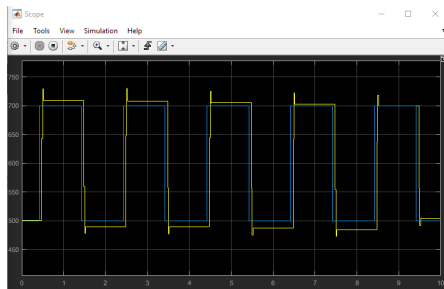


Figure 3.8: Output graph from MATLAB/Simulink

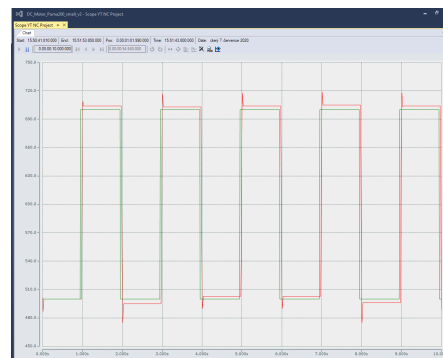


Figure 3.9: Output graph from TwinCAT

### 3.4 Connection of TwinCAT and MATLAB/Simulink via the PLC and the NC

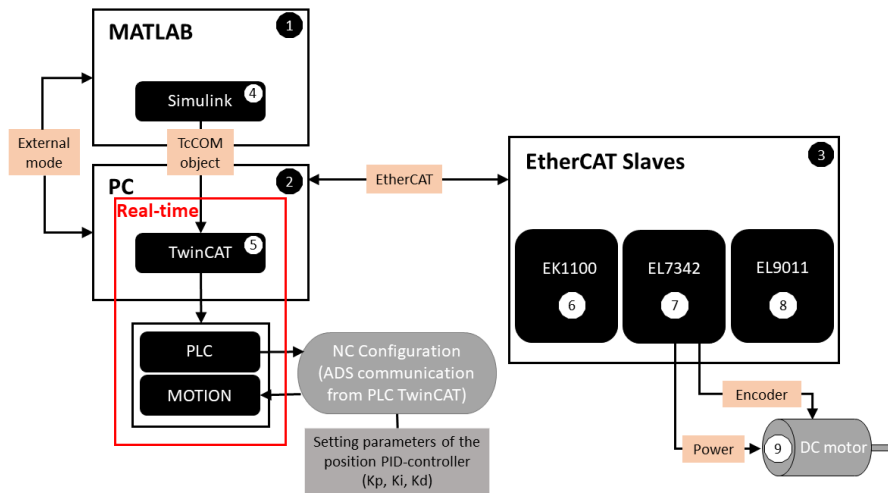
Let's address the task slightly differently and use the second application that can be observed in Fig. 2.31. It is the bottom schema, called control of a

### 3.4. Connection of TwinCAT and MATLAB/Simulink via the PLC and the NC

DC motor with encoder feedback. The difference from the previous method is that the controller is set to the velocity compact mode. Thus, instead of affecting the position, the speed is controlled. Also, velocity and current controller's loops are again inside the terminal, but the position loop is included in TwinCAT.

Another advantage is that unlike the previous technique, this one enables an instant rewriting of parameters in the OP state of the terminal. Therefore, there is no need for switching between PREOP and OP. And in addition, the parameters are written directly to the PC.

A schema of the second try is shown in Fig. 3.10. All software and components have already been described in the previous section, see 3.3.



**Figure 3.10:** Diagram describing the project using the NC configuration

For this approach, the project in TwinCAT needed to be changed because it uses both the PLC and the NC. The 3 PLC programs, that were used previously, are not used anymore. There is only one PLC program responsible for setting parameters. An axis of the DC motor must be added to the NC Configuration in the MOTION section of the TwinCAT project and linked to the EL7342 terminal. When it is done so, the velocity and the angle of rotation can be adjusted as required.

As can be observed in Fig. 3.10, the PLC program access the NC Configuration via ADS<sup>15</sup>, which is a transport layer inside TwinCAT. ADS provides communication between the PLC and the NC. In the NC Configuration, the

<sup>15</sup>ADS = Automatic Device Specification

### 3. DC MOTOR SERVOCONTROL

---

parameters  $K_p$ ,  $K_i$ , and  $K_d$  are set and then the data are transferred from the MOTION section via EtherCAT to the EL7342 module.

After examining which parameters affect the behavior of the DC motor the most, it was found that constants  $K_p$  and  $K_d$  make the most significant changes in the motor response. Also, the value of a reference velocity makes a huge difference in the motor's running; therefore, it needs to be set appropriately.

Based on information from Beckhoff, here's how to set a reference speed [41]:

1. *Set the  $K_p$  value to 0 and begin an empirical evaluation of the reference velocity.*
2. *Turn the motor and compare the setpoint velocity with the actual velocity.*
3. *Keep adjusting the reference velocity till the actual velocity is as identical to the setpoint velocity as it can get. Then record the reference velocity and set the final value as the reference velocity.*

The reference velocity was determined to be 17 800 revolutions/sec.. After that, the project is ready for tuning  $K_p$  and  $K_d$  values.

The new TwinCAT project is depicted in Fig. 3.11. On the left side of Fig. 3.11 under the tab PlcTask Inputs, it can be found that MAIN.user\_Write, MAIN.Kp\_value, and MAIN.Kd\_value have small green arrows beside them which indicate that these values are linked to Simulink. Therefore, they can be controlled from there. The declaration of variables of the only PLC program is demonstrated in this picture as well. There is always the name of the variable, type, and initial value. It is sometimes specified whether the variable is an input or an output, which is declared by either AT%I\* for input or AT%Q\* for output. Function blocks ADS\_write\_Kp and ADS\_write\_Kd are used for writing parameters  $K_p$  and  $K_d$ , respectively. The NETID and the PORT are included in order to know where the data are to be passed on.

Fig. 3.12 shows the actual code of the PLC program. At the top, there are two IF conditions for enabling and disabling the motor. Then, there is a state machine for writing the parameters. It starts with case 0 where "user\_Write = 69" is used for reset. Once "user\_Write = 1" and "startWrite = TRUE", CoE interface for adjusting parameters is accessed. This is done via an ADS-Write to the NETID of the device, PORT, IDXGRP (IndexGroup), and IDXOFFS (IndexOffset). Then, it moves to the state 1 where the writing itself happens. It keeps waiting there until the program is busy with adjusting parameters. If the program is not busy and no errors occur, it goes back to the state 0 and when it is reset, it is ready for writing new parameters. In case of an error, the program falls into the state 100 and the error needs to be handled before moving on.

### 3.4. Connection of TwinCAT and MATLAB/Simulink via the PLC and the NC

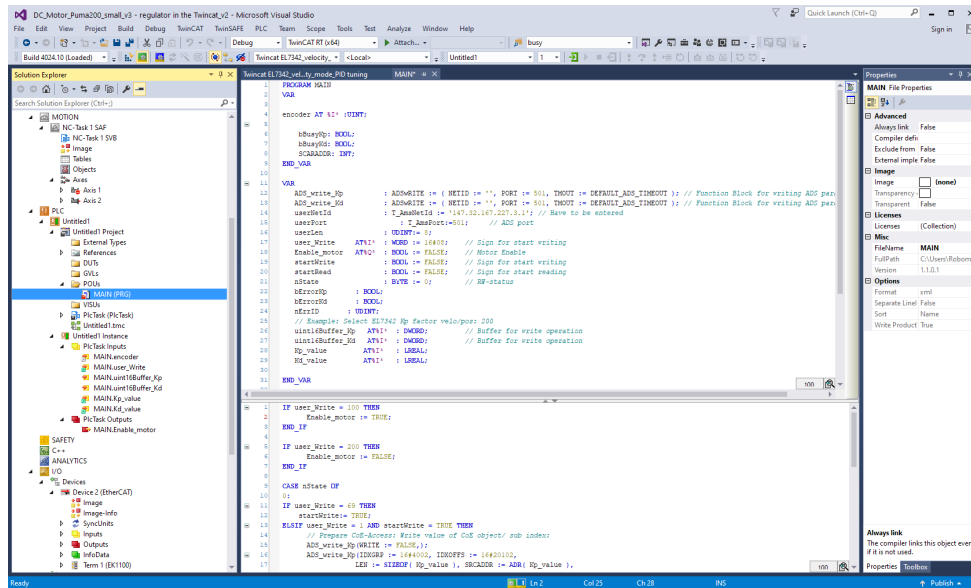


Figure 3.11: TwinCAT program using the PLC and the NC

```

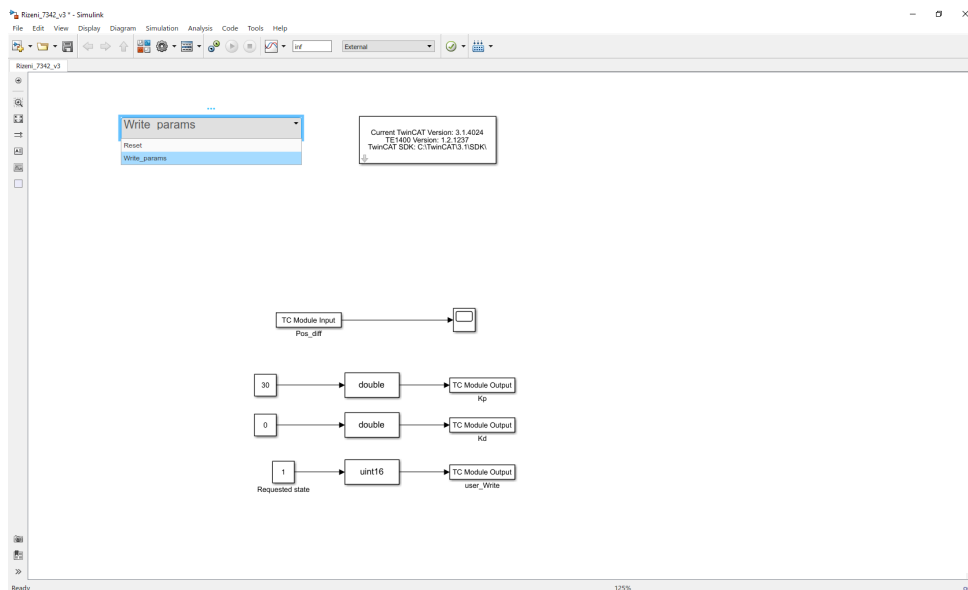
1  IF user_Write = 100 THEN
2      Enable_motor := TRUE;
3  END_IF
4
5  IF user_Write = 200 THEN
6      Enable_motor := FALSE;
7  END_IF
8
9  CASE nState OF
10  0:
11  IF user_Write = 69 THEN
12      startWrite:= TRUE;
13  ELSIF user_Write = 1 AND startWrite = TRUE THEN
14      // Prepare COE-Access: Write value of CoE object/ sub index:
15      ADS_write_Kp(WRITE := FALSE,);
16      ADS_write_Kp(IDXGRP := 16#4002, IDXOFFS := 16#20102,
17          LEN := SIZEOF( Kp_value ), SRCADDR := ADR( Kp_value ),
18          WRITE := TRUE );
19
20      ADS_write_Kd(WRITE := FALSE,);
21      ADS_write_Kd(IDXGRP := 16#4002, IDXOFFS := 16#20104,
22          LEN := SIZEOF( Kd_value ), SRCADDR := ADR( Kd_value ),
23          WRITE := TRUE );
24
25      nState := 1; // Next state for writing
26      startWrite := FALSE;
27  END_IF
28  1:
29      ADS_write_Kp( Write:= FALSE, BUSY=>bBusyKp, ERR=>bErrorKp, ERRID=>nErrID);
30      ADS_write_Kd( Write:= FALSE, BUSY=>bBusyKd, ERR=>bErrorKd, ERRID=>nErrID);
31      IF NOT (bBusyKp OR bBusyKd) THEN
32          IF NOT (bErrorKp OR bErrorKd) THEN
33              nState := 0; (* Success *)
34          ELSE
35              nState := 100; (* Error *)
36          END_IF
37      END_IF
38
39  100:
40      // Error handling..
41  END_CASE
42
43
44
45

```

Figure 3.12: Code of the PLC program in TwinCAT

### 3. DC MOTOR SERVOCONTROL

While developing the MATLAB/Simulink program for adjusting  $K_p$  and  $K_d$ , a great emphasis was placed on reducing the number of TC Module Input and Output blocks. The maximum allowed amount is 6 and, as can be seen in Fig. 3.13, only 4 TC Module blocks are used. Since the adjustment of parameters is done directly in the OP state, the need for switching between states is eliminated. Enabling and disabling options are also redundant because the motor is automatically enabled once the program is activated. However, the option for the motor to be disabled could have been added there from the safety point of view.



**Figure 3.13:** Program created in MATLAB/Simulink for manual tuning

Fig. 3.13 depicts only two possibilities of "Requested\_state" in the top left corner, they are "Write params" and "Reset" and are linked to the PLC code in TwinCAT. Simulink can send to TwinCAT either "user\_Write = 1" for writing parameters or "user\_Write = 69" for reset. The TC Module Input called "Pos\_diff" records the difference between the actual position and the set position. The difference, and its plot is discussed in the following chapter.

---

# Automatic Servotuning

This chapter is dedicated to the development and testing of an autotuning algorithm written in MATLAB/Simulink. The autotuning algorithm is supposed to tune any function automatically when the user starts it off. Since the goal is to minimize the difference between the actual value and the set (desired) value, the algorithm is based on the gradient descent method. This technique calculates a gradient, which is the steepest slope, of a function, and then moves in the opposite direction in order to reduce the difference. It keeps repeating this process until a global minimum of this function is reached. When the global minimum is achieved, the actual value overlaps with the set value.

In the previous chapter, MATLAB/Simulink was connected to TwinCAT. Consequently, parameters of the PID-controller are able to be manually adjusted from Simulink. It has been determined that  $K_p$  and  $K_d$  constants of PID-controller are the parameters that have the biggest influence on the course of the response. Now, let's look at the possibility of automatic tuning of two variables.

## 4.1 Testing of Gradient Descent Method in MATLAB

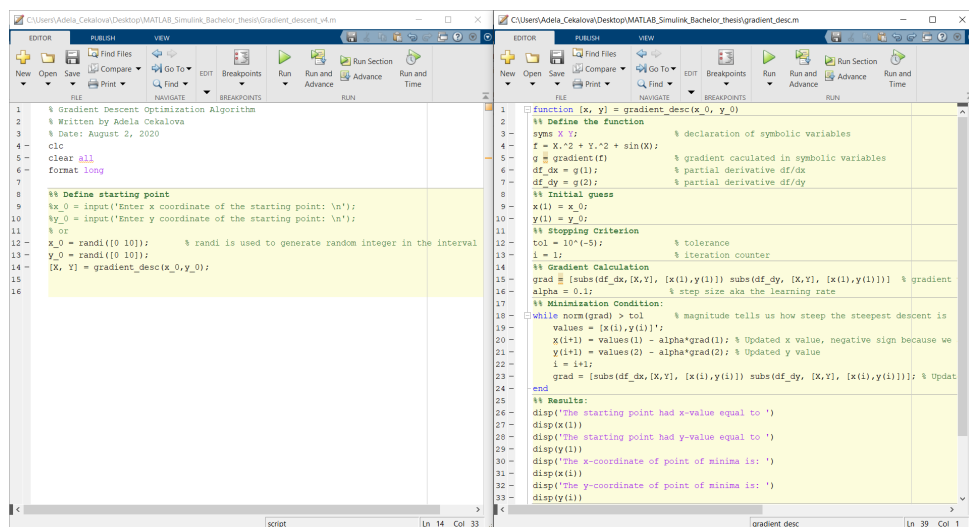
Since the aim is to tune two parameters of the PID-controller, the developed gradient descent algorithm in MATLAB is tested on a random function with two variables. The function with two variables is analogous to changing parameters  $K_p$  and  $K_d$ . Let's take, for example, a following function:

$$f = x^2 + y^2 + \sin(x) \quad (4.1)$$

## 4. AUTOMATIC SERVOTUNING

It is a function of a surface, as Fig. 4.2 depicts. A MATLAB function called `gradient_desc`, illustrated in Fig. 4.1 (right side), was created to find a point of minima of a given function. The function has 2 inputs `x_0`, `y_0`, and two outputs `x` and `y`. The algorithm is based on the gradient descent method, described in section 2.4.4. First, the given function and its variables are declared symbolically and the gradient is calculated. Then, the input values are set as initial values. After that, the stopping criterion is declared. There is a wide variety of stopping criteria, and the most commonly used is to stop the search when the Euclidean norm of the gradient has reached certain small pre-defined value, which is called the tolerance. Afterward, the gradient is calculated again, but the symbolic values are substituted with the initial values and the iterative process can begin. Every iteration the point moves in the opposite direction to the gradient and the gradient is updated with new values. The algorithm repeats itself until the stopping criterion is satisfied. The accuracy depends on the tolerance and the step size, also called the learning rate. The smaller the step size, the more precise results are reached but the longer it takes. In this algorithm, the step size was set to 0.1, it works well and it is quite fast.

The left side of Fig. 4.1 shows a script where the function `gradient_desc` is called and random integer values are generated as inputs of the function. It could have been done more user-friendly by uncommenting the lines 9, 10 and letting the user insert initial guesses.



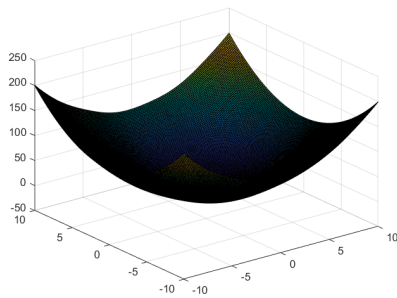
```
1 % Gradient Descent Optimization Algorithm
2 % Written by Adela Cekalova
3 % Date: August 2, 2020
4 =
5 clear all
6 format long
7
8 %% Define starting point
9 % x_0 = input('Enter x coordinate of the starting point: ');
10 % y_0 = input('Enter y coordinate of the starting point: ');
11 % or
12 x_0 = randi(10 10); % randi is used to generate random integer in the interval
13 y_0 = randi(10 10);
14 [X, Y] = gradient_desc(x_0,y_0);
15
16
```

```
1 function [x, y] = gradient_desc(x_0, y_0)
2 %% Define the function
3 syms x y; % declaration of symbolic variables
4 f = x.^2 + y.^2 + sin(x);
5 g = gradient(f) % gradient calculated in symbolic variables
6 df_dx = g(1); % partial derivative df/dx
7 df_dy = g(2); % partial derivative df/dy
8 %% Initial guess
9 x(i) = x_0;
10 y(i) = y_0;
11 %% Stopping Criterion
12 tol = 10^(-5); % tolerance
13 i = 1; % iteration counter
14 %% Gradient Calculation
15 grad = [subs(df_dx, [X, Y], [x(i), y(i)]) subs(df_dy, [X, Y], [x(i), y(i)])] % gradient
16 alpha = 0.1; % step size aka the learning rate
17 %% Minimization Condition:
18 while norm(grad) > tol % magnitude tells us how steep the steepest descent is
19 values = [x(i), y(i)]';
20 x(i+1) = values(1) - alpha*grad(1); % Updated x value, negative sign because we
21 y(i+1) = values(2) - alpha*grad(2); % Updated y value
22 i = i+1;
23 grad = [subs(df_dx, [X, Y], [x(i), y(i)]) subs(df_dy, [X, Y], [x(i), y(i)])]; % Updat
24 end
25 %% Results:
26 disp('The starting point had x-value equal to ')
27 disp(x(i))
28 disp('The starting point had y-value equal to ')
29 disp(y(i))
30 disp('The x-coordinate of point of minima is: ')
31 disp(x(i))
32 disp('The y-coordinate of point of minima is: ')
33 disp(y(i))
```

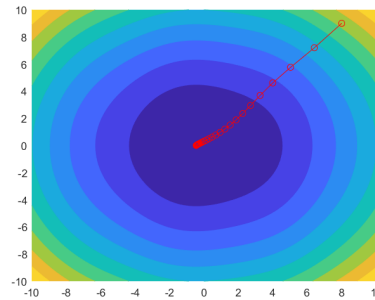
Figure 4.1: Gradient descent MATLAB function

But for now, let's examine the results with a random generation of the starting point. The plot of how the initial point moves towards the point of minima is demonstrated in Fig. 4.3. The darker the filling of the plot, the lower the

surface is getting. The movement of the points is marked by red circles. When the point of minima is reached, the function stops and shows the coordinates of the starting point and the coordinates of the point of minima in the Command window, see Fig. 4.4. Just to make sure that the code is written correctly, a gradient with symbolic values and with initial values is displayed in the Command window as well.



**Figure 4.2:** Surface of the given function



**Figure 4.3:** Point of minima of the given function

```

Command Window

g =

    2*X + cos(X)
           2*Y

grad =

[ cos(8) + 16, 18]

The starting point had x-value equal to
8

The starting point had y-value equal to
9

The x-coordinate of point of minima is:
-0.450183157571014

The y-coordinate of point of minima is:
4.519513249478410e-06

fx >>
    
```

**Figure 4.4:** MATLAB Command Window

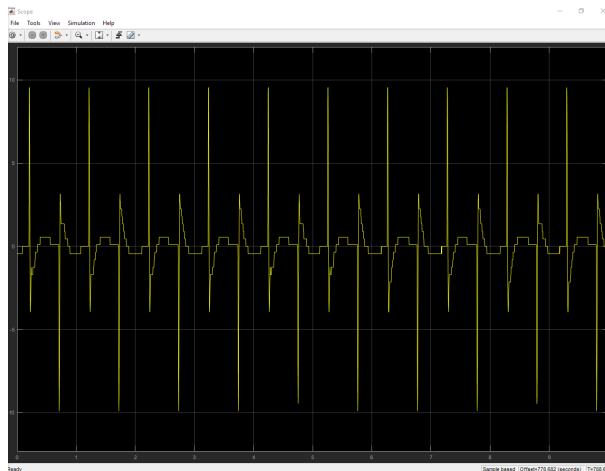
As can be found in the Command window, the x-value of the point of minima is -0.45 and the y-value is close to 0. By setting the tolerance equal to  $10e-12$  and decreasing the step size to 0.001. The resulting point of minima is  $[-0.4501836, 4.9943040e-13]$ , which is more precise, but the iteration process is very time-consuming. Thus, the step size and the stopping criterion must be chosen reasonably based on the application and precision needs.



## 4.2 Autotuning Algorithm in MATLAB/Simulink

The autotuning algorithm is based on the gradient descent method. When considering the real DC motor, a model of the DC motor's behavior is not provided. It is possible to track the functions of the actual position and the set value of the position. Then, subtract them from each other so that a difference of these functions is obtained. The difference, shown in Fig. 4.5, has big peaks that are caused by a slight offset between the actual and the set position functions, that are captured in Fig. 3.8. The difference must be logically minimized in order to get as close to the set value of the position as possible. To do that, an absolute value of the difference can be taken, which results in a curve on the positive side of the y-axis. The area under the curve can be calculated. To decrease the difference, the area needs to be reduced. With this idea in mind, a MATLAB/Simulink program is developed and is depicted in Fig. 4.6. Along with the Simulink program, there is also a final optimization scope for  $K_p$  parameter, which will be discussed later in this section.

As it has been examined earlier, the most significant effect on the motor's response have values  $K_p$  and  $K_d$ . After further experiments, it has been proven that the only obvious change in the difference between the actual and the set position shows  $K_p$ . Consequently, the system is tuned by only adjusting  $K_p$ , it means that the P-controller is used for this application.

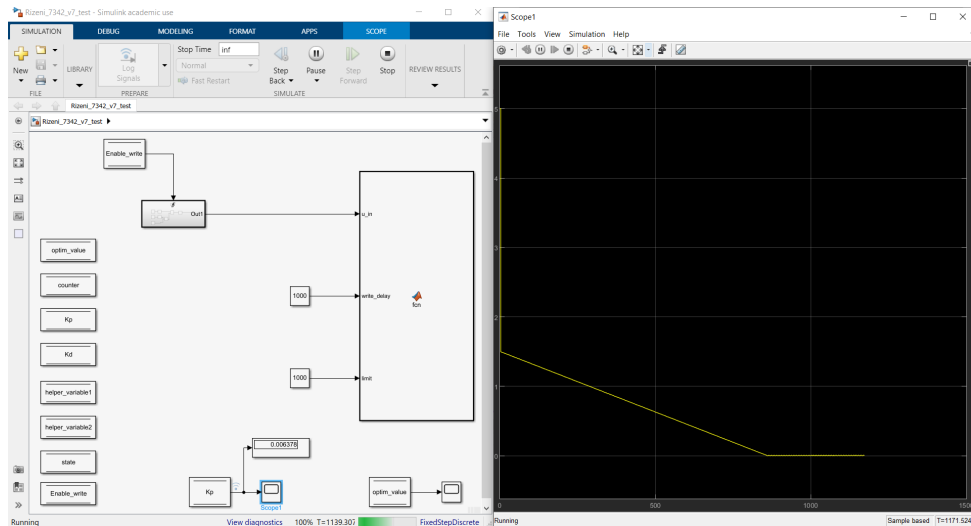


**Figure 4.5:** Difference between the actual position and the set value of the position

Let's explain what is happening in the Simulink program in Fig. 4.6. First of all, the Simulink program is not connected to TwinCAT because the optimization will be applied only to a simulation model to see if it works. On the

## 4.2. Autotuning Algorithm in MATLAB/Simulink

very left side, there are defined 8 global variables, it is convenient to declare the variables as global so that they can be accessed via any subsystem or function. The biggest function block is an embedded MATLAB function where the whole gradient descent optimization algorithm is hidden. Then, there is a triggered subsystem, which is the block that has Enable\_write as input. And on the bottom, the  $K_p$  value is tracked and plotted.

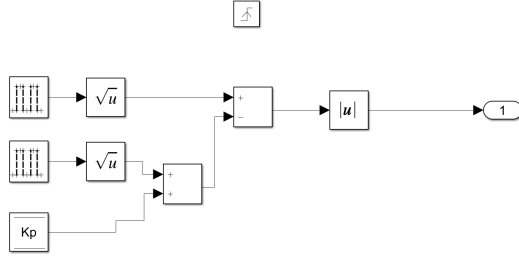


**Figure 4.6:** MATLAB/Simulink optimization program with scope

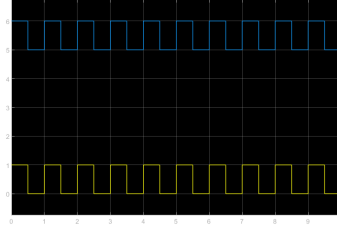
The triggered subsystem, demonstrated in Fig. 4.7, is where the set and the actual positions are generated. The two pulses in Fig. 4.8 are supposed to model the set value (yellow curve) and the actual position (blue curve). The blue pulse has an offset equal to  $K_p$  that is added to the signal. The initial value of  $K_p$  is 5, as can be observed in Fig. 4.8. Then, the two signals are subtracted to obtain the difference and an absolute value of the difference is taken to ensure that all values are positive. And the absolute value is the output of the triggered subsystem. The triggered subsystem is activated using some trigger, here the trigger is Enable\_write variable on the input of this subsystem. When Enable\_write equals 1, the subsystem is activated.

When calculating the area, there was a problem before the triggered subsystem was implemented. The problem was that the difference between the positions keeps changing at each sample time and to get precise results, the calculation of the area needs to be started always at the same point of the period. For this reason, the trigger subsystem was added and is activated only when the area is being determined and starts always at the same spot. The sample time and the base rate of Simulink are set to 1 ms. As a result, the whole simulation runs with a frequency of 1 kHz.

Let's get to the embedded MATLAB function and the code that is inside it,



**Figure 4.7:** Triggered subsystem



**Figure 4.8:** Generated pulses for simulation of optimization

the algorithm is shown in Fig. 4.9. It is programmed as a state machine using the "switch" function that transitions between different states using "cases". "Case 0" is only initialization and it moves right away to "Case 10" where Enable\_write is set to 1; therefore, the triggered subsystem is activated and the calculation of the area begins. The counter counts how many samples are supposed to be taken, it works until a specific limit is reached. The limit is an input of this function and can be adjusted in Simulink. The area is calculated by summing all values of samples, it is the same as integration, and is saved to optim\_value variable. In "Case 15", the  $K_p$  is adjusted by 0.00001 ( $\Delta K_p$ ) and the program moves to the following "Case 20". In the "Case 20", a new area with the updated  $K_p$  value is calculated and then, a gradient is determined. The gradient is calculated numerically by single-sided difference as the following formula shows:

$$\nabla = \frac{S(K_p + \Delta K_p) - S(K_p)}{\Delta K_p}, \quad (4.2)$$

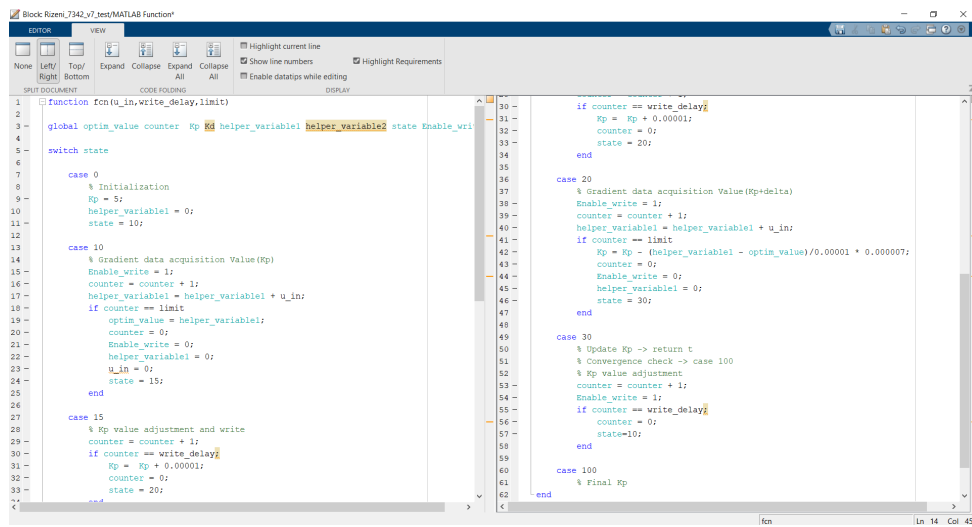
where S is the area. As the gradient descent method states, the direction of movement is against the direction of the gradient (the steepest slope) because a point of minima is desirable. Thus, a new  $K_p$  value is obtained as follows:

$$K_p = K_p - \nabla \cdot 0.000007, \quad (4.3)$$

where 0.000007 is the step size, also called the learning rate. The smaller the learning rate, the more precise results are achieved.

After that, the program goes to "Case 30" and writes the updated  $K_p$  value to TwinCAT. This process keeps repeating until the optimal  $K_p$  value is reached and then, it stays there.

## 4.2. Autotuning Algorithm in MATLAB/Simulink



```
function fcn(u_in, write_delay, limit)
global optim_value counter Kp Kd helper_variab1e1 helper_variab1e2 state Enable_wri

switch state
case 0
% Initialization
Kp = 5;
helper_variab1e1 = 0;
state = 10;
case 10
% Gradient data acquisition Value(Kp)
Enable_wri = 1;
counter = counter + 1;
helper_variab1e1 = helper_variab1e1 + u_in;
if counter == limit
optim_value = helper_variab1e1;
counter = 0;
Enable_wri = 0;
helper_variab1e1 = 0;
u_in = 0;
state = 15;
end
case 15
% Kp value adjustment and write
counter = counter + 1;
if counter == write_delay
Kp = Kp + 0.00001;
counter = 0;
state = 20;
end
case 20
if counter == write_delay
Kp = Kp - 0.00001;
counter = 0;
state = 20;
end
case 20
% Gradient data acquisition Value(Kp=delta)
Enable_wri = 1;
counter = counter + 1;
helper_variab1e1 = helper_variab1e1 + u_in;
if counter == limit
Kp = Kp - (helper_variab1e1 - optim_value)/0.00001 * 0.000007;
Enable_wri = 0;
helper_variab1e1 = 0;
state = 30;
end
case 30
% Update Kp -> return t
% convergence check -> case 100
% Kp value adjustment
counter = counter + 1;
Enable_wri = 1;
if counter == write_delay
counter = 0;
state=10;
end
case 100
% Final Kp
end
```

**Figure 4.9:** MATLAB function for optimization

When considering the generated pulses in Fig. 4.8, to make the actual position same as the set position,  $K_p$  needs to reach 0. The initial value of  $K_p$  is 5. While testing the MATLAB/Simulink program, the  $K_p$  value is recorded on the graph in Fig. 4.6. It is clear that the value started at 5, then, it dropped drastically, and afterward, it kept slowly moving towards zero. Once the  $K_p$  value reaches zero, it keeps oscillating around zero within a very small range. Hence, the predicted value of 0 was achieved.



---

## Results

The autotuning algorithm based on the gradient descent method to find optimal parameters of the PID-controller for the smooth running of the DC motor has been developed. It is now capable of tuning only one parameter, namely the value of  $K_p$ .  $K_p$  is the proportional constant of the PID-controller.

The autotuning algorithm has been tested in a simulation, and it works well. It has been proven that the expected  $K_p$  parameter was reached by the autotuning optimization. Therefore, it has been assumed that it would be applicable to solve the real problem of PID-controller tuning as well. However, it has not been applied to the real DC motor because the whole process of developing the autotuning algorithm, which was preceded by finding the right way to be able to adjust the parameters using MATLAB/Simulink, was too time-consuming. However, all of this was achieved and tested in the simulation. The greatest impact upon the computational time of the simulation have the values of the learning rate, also called the step size, and counter limit. Both values can be modified in the autotuning algorithm based on the user's needs and wants for accuracy and time consumption.

Apart from the autotuning algorithm in Simulink, a gradient descent optimization for finding a point of minima for a function with two variables has been introduced and successfully tested. Thus, expanding the autotuning algorithm by one more parameter would not be a problem.

Besides, two approaches of connecting TwinCAT to MATLAB/Simulink have been examined. But only the second one, where both the NC and the PLC were implemented, was successful.



---

## Conclusion

The content of this thesis begins with an introduction to power units in robotics with an emphasis on DC motors followed by an explanation of basic key terms and DC motors control itself. Furthermore, relevant information about the EL7342 Beckhoff terminal and TwinCAT 3 are covered.

The major target was to develop a simple autotuning algorithm that was achieved. The process of getting there was not an easy task. Mainly because the EL7342 Beckhoff module did not work consistently, as it was predicted. But despite that, it was possible to connect the DC motor to TwinCAT via the EL7342 terminal and create a MATLAB/Simulink program to be able to control parameters of the DC motor from there. Finally, the autotuning algorithm was created and applied to simulation. The performed simulation yielded satisfying results; thus, it proved the sustainability of DC motor control and can be reliably used for automatic tuning of the  $K_p$  parameter.

There is certainly room for further modifications of the autotuning algorithm, especially in expanding the number of possible tuneable parameters by adding the remaining constants  $K_i$  and  $K_d$ . That would lead to eliminating a steady state error and reducing or even getting rid of an overshoot. Simply, it would make the transition smoother and faster. When tuning a PID-controller, initial values of  $K_p$ ,  $K_i$ , and  $K_d$  have to be chosen reasonably. Otherwise, it may result in an unstable system and damage the DC motor.





---

# Bibliography

1. MILLER, Mark R.; MILLER, Rex. *Robots and Robotics - Principles, Systems, and Industrial Applications*. 1st ed. McGraw-Hill Education, 2017. ISBN 978-1-259-85978-6.
2. *Jak funguje stejnosměrný motor* [online]. Learn Engineering, 2014 [visited on 2020-03-08]. Available from: <https://www.youtube.com/watch?v=LAtPHANefQo>.
3. *DC Motor* [online]. ElectronicWings, 2020 [visited on 2020-03-08]. Available from: <https://www.electronicwings.com/sensors-modules/dc-motor>.
4. COLLINS, Danielle. *Best DC motors for high starting torque* [online]. Motion Control Tips, 2016 [visited on 2020-03-08]. Available from: <https://www.motioncontroltips.com/faq-whats-the-most-suitable-dc-motor-for-high-starting-torque/>.
5. CONDIT, Reston. *Brushed DC Motor Fundamentals* [online]. Microchip Technology Inc., 2004 [visited on 2020-03-08]. Available from: <http://ww1.microchip.com/downloads/en/appnotes/00905a.pdf>.
6. *Fleming's Left Hand Rule and Right Hand Rule* [online]. BYJU'S [visited on 2020-03-08]. Available from: <https://byjus.com/physics/flemings-left-hand-rule-and-right-hand-rule/>.
7. *What are Brushless DC Motors* [online]. Renesas [visited on 2020-03-08]. Available from: <https://www.renesas.com/us/en/support/technical-resources/engineer-school/brushless-dc-motor-01-overview.html>.

8. *Brushless DC Motor, How it works?* [online]. Learn Engineering, 2014 [visited on 2020-03-08]. Available from: <https://www.youtube.com/watch?v=LaTPHANEfQo>.
9. BRAIN, Marshall. *How Does a Brushless Electric Motor Work?* [online]. How Stuff Works, 2006 [visited on 2020-03-08]. Available from: <https://electronics.howstuffworks.com/brushless-motor.htm>.
10. *Stepper Motor* [online]. ElectronicWings, 2020 [visited on 2020-03-28]. Available from: <https://www.electronicwings.com/sensors-modules/stepper-motor>.
11. *Variable-reluctance stepper motor with four rotor poles and six stator poles* [online]. Electrical Academia [visited on 2020-08-07]. Available from: <https://electricalacademia.com/synchronous-machines/stepper-motor-types-working/attachment/variable-reluctance-stepper-motor-with-four-rotor-poles-and-six-stator-poles/>.
12. *How does a Stepper Motor work?* [online]. Learn Engineering, 2016 [visited on 2020-03-28]. Available from: <https://www.youtube.com/watch?v=eyqwLiowZiU>.
13. *Stepper Motor - Types, Advantages and Applications* [online]. EL-PRO-CUS, 2018 [visited on 2020-03-28]. Available from: <https://www.build-electronic-circuits.com/h-bridge/>.
14. CROWDER, Richard. *Electric Drives and Electromechanical Systems*. 1st ed. Elsevier, 2006. ISBN 978-0-7506-6740-1.
15. DAHL, Oyvind Nydal. *What is an H-Bridge?* [online]. Build Electronic Circuits, 2018 [visited on 2020-03-31]. Available from: <https://www.elprocus.com/stepper-motor-types-advantages-applications/>.
16. LYNCH, Kevin. *Driving DC motors, part 3/3: H-bridges* [online]. Northwestern Robotics, 2015 [visited on 2020-03-31]. Available from: <https://www.youtube.com/watch?v=fVgnUWIWzZ8>.
17. MCGRADY, Chris. *What is a Snubber and Why do You Want One?* [online]. Arrow, 2016 [visited on 2020-06-01]. Available from: <https://www.arrow.com/en/research-and-events/articles/what-is-a-snubber>.
18. *Pulse Width Modulation (PWM)* [online]. EL-PRO-CUS [visited on 2020-04-04]. Available from: <https://www.elprocus.com/pulse-width-modulation-pwm/>.

19. *Pulse Width Modulation* [online]. Electronics Tutorials, 2019 [visited on 2020-04-04]. Available from: <https://www.electronics-tutorials.ws/blog/pulse-width-modulation.html>.
20. HLÁVKA, Ing. Petr. *DC MOTOR Measurement - Speed Control of Pulse Width Modulation (PWM)* [online]. 2012 [visited on 2020-04-04]. Available from: <https://docplayer.cz/32425017-Dc-motor-measurement-speed-control-of-pulse-width-modulation-pwm.html>.
21. ING. MILAN HOFREITER, CSc. prof. *Základy automatického řízení*. 1st ed. České vysoké učení technické, 2012. ISBN 978-80-01-05007-1.
22. *What is a PID Controller, Their Types nad How does it Work?* [online]. Electrical Technology [visited on 2020-04-04]. Available from: <https://www.electricaltechnology.org/2015/10/what-is-pid-controller-how-it-works.html>.
23. *Understanding PID Control, Part 4: A PID Tuning Guide* [online]. MATLAB, 2018 [visited on 2020-04-18]. Available from: <https://www.youtube.com/watch?v=sFOEsA0Irjs>.
24. *How to tune a PID controller?* [online]. OMEGA [visited on 2020-04-18]. Available from: <https://www.omega.co.uk/technical-learning/tuning-a-pid-controller.html>.
25. BĚŤÁK, Jaroslav. *Řízení a autotuning synchronního AC servomotoru Baldor pomocí měniče Beckhoff AX5206*. České vysoké učení technické, 2019.
26. CHONG CHEE SOON, Rozaimi Ghazali. *PID controller tuning optimisation using gradient descent technique for an electrohydraulic servo system* [online]. Jurnal Teknologi, 2015 [visited on 2020-04-20]. Available from: <https://jurnalteknologi.utm.my/index.php/jurnalteknologi/article/view/6605/4367>.
27. *Pros and cons of autotuning control: Part 1* [online]. Control Engineering, 2018 [visited on 2020-06-01]. Available from: <https://www.controleng.com/articles/pros-and-cons-of-autotuning-control-part-1/>.
28. KROUPA, Tomáš. *Princip gradientních optimalizačních metod* [online]. 2014 [visited on 2020-04-20]. Available from: [https://www.kme.zcu.cz/kmet/tutorials/data/vsme\\_opvk/podpurne\\_studijni\\_materialy/optimalizace\\_princip\\_gradientni.pdf](https://www.kme.zcu.cz/kmet/tutorials/data/vsme_opvk/podpurne_studijni_materialy/optimalizace_princip_gradientni.pdf).
29. KUNII, Toshiyasu L. *Visual Computing*. 1st ed. Springer Science and Business Media, 2013. ISBN 9784431682042.

## BIBLIOGRAPHY

---

30. *Understanding PID Control, Part 6: Manual and Automatic Tuning Methods* [online]. MATLAB, 2018 [visited on 2020-04-18]. Available from: <https://www.youtube.com/watch?v=qj8vT01eIHo>.
31. *Understanding PID Control, Part 7: Important PID Concepts* [online]. MATLAB, 2018 [visited on 2020-04-18]. Available from: <https://www.youtube.com/watch?v=tbgV6caAVcs>.
32. VANDOREN, Vance. *Fundamentals of cascade control* [online]. Control Engineering, 2014 [visited on 2020-04-20]. Available from: <https://www.controleng.com/articles/fundamentals-of-cascade-control/>.
33. *TwinCAT 3* [online] [visited on 2020-05-21]. Available from: [https://download.beckhoff.com/download/document/catalog/TwinCAT\\_3\\_Booklet.pdf](https://download.beckhoff.com/download/document/catalog/TwinCAT_3_Booklet.pdf).
34. DIXON, Mary. *What is Ethernet?* [online]. RealPars, 2019 [visited on 2020-07-15]. Available from: <https://realpars.com/ethernet/>.
35. ANDERSON, Mondy. *What is EtherCAT?* [online]. RealPars, 2019 [visited on 2020-07-15]. Available from: <https://realpars.com/ethercat/>.
36. *Beckhoff EL73x2 Dokumentation* [online]. 2020 [visited on 2020-03-08]. Available from: <https://download.beckhoff.com/download/document/io/ethercat-terminals/el73x2en.pdf>.
37. *AEDA-3300 Series Data Sheet* [online]. 2006 [visited on 2020-03-08]. Available from: <https://datasheet.octopart.com/AEDA-3300-TAM-Avago-datasheet-8330581.pdf>.
38. *TC3 Target for Matlab/Simulink - TwinCAT 3* [online]. 2020 [visited on 2020-06-21]. Available from: [https://download.beckhoff.com/download/document/automation/twincat3/TE1400\\_TC3\\_Target\\_Matlab\\_EN.pdf](https://download.beckhoff.com/download/document/automation/twincat3/TE1400_TC3_Target_Matlab_EN.pdf).
39. BECKHOFF. *EK1100 EtherCAT Coupler* [online]. Beckhoff [visited on 2020-07-28]. Available from: <https://www.beckhoff.com/english.asp?ethercat/ek1100.htm>.
40. BECKHOFF. *EK1100 Bus end cover* [online]. Beckhoff [visited on 2020-07-28]. Available from: <https://www.beckhoff.com/EL9011/>.
41. BECKHOFF. *Reference velocity selection* [online]. Beckhoff [visited on 2020-08-04]. Available from: <https://infosys.beckhoff.com/english.php?content=../content/1033/el73x2/2075571595.html&id=>.