UNIVERSITE KASDI MERBAH OUARGLA

Faculté des Sciences Appliquées Département de Génie Electrique

**Mémoire**

MASTER ACADEMIQUE

Domaine : Sciences et technologies Filière : Génie électrique

Spécialité : Machines électriques

Présenté par :

BELKACEM BECHOUA …… BABZIZ MOHAMMED NACER

**Thème :**

# Speed and position control of DC motor by Arduino

Soumis au jury composé de :

| | | | | |
|---|---|---|---|---|
| BENAOUADJ | MAHDI | MCB | PRESIDENT | Encadreur/rapporteur UKM Ouargla |
| REZOUG | REDHA | MCA | EXAMINATEUR | Encadreur/rapporteur UKM Ouargla |
| DJARAH | DJALAL | MCB | ENCADREUR | Encadreur/rapporteur UKM Ouargla |
| BENMAKHLOUF | ABDESLAM | MAA | CO-ENCADREUR | Encadreur/rapporteur UKM Ouargla |

**Année universitaire 2022/2023**

# Thanks

We would like to begin by thanking Allah Almighty for granting us the strength, courage, and patience to accomplish this task.

We extend our heartfelt thanks and appreciation to all those who contributed to and supported the completion of this work. It is impossible to summarize all the words that express my deep gratitude to everyone who provided assistance and guidance throughout this academic journey.

We thank our families for their unlimited support and belief in our abilities. They have been the pillar on which we relied during this crucial period of our academic lives.

I would like to express my deep gratitude to my mentor, Mr.Djarah Djalal , for sharing his knowledge, expertise, and guidance during this project. You have been a source of inspiration, and I have learned so much thanks to your wisdom and direction.

I am grateful to my fellow students and friends who shared this academic journey with me. We had moments of challenge and mutual encouragement, and I cannot thank them enough for the support and collaboration they provided.

Lastly, I would like to offer special thanks to the educational institution that provided me with this opportunity and platform for academic growth. You have been a crucial factor in my educational journey, and I am grateful for every opportunity given to me.

Thank you once again for your boundless support and dedication in assisting me in achieving this accomplishment.

I will carry these beautiful memories and valuable experiences with me throughout my life.

# Dedicates

**Dedication**

**To our beloved parents.**

**with heartfelt gratitude for all the years of sacrifice and encouragement.**

**To our siblings and extended family.**

**To all our friends...**

**To the teachers who have guided us.**

**To all those I hold dear,**

**I dedicate this thesis with profound appreciation and gratitude to all of you who have been an invaluable source of support, inspiration, and motivation throughout this academic journey. Your love, encouragement, and trust in me have been the essential pillars of my success. May this dedication reflect my immense gratitude and deep respect for all of you.**

## Abstract

This work presents a study of controlling the position and speed of a DC motor using Arduino

In this study, the hardware aspects of the system, including the DC motor, encoder, and L298 drivers, were explored. Then the methodology used in system and control modeling was presented, while specifying the approach used to develop mathematical models, design control algorithms, and implement software solutions. The results obtained from the implemented system were analyzed and compared with the expected behavior based system models and control algorithms. Factors affecting results, such as sensor resolution and system dynamics, are discussed in detail. The effectiveness of the control system in achieving precise engine positioning and speed regulation was evaluated.

## Résumé

Ce travail présente une étude de contrôle de la position et de la vitesse d'un moteur à courant continu à l'aide d'Arduino

Dans cette étude, les aspects matériels du système, y compris le moteur à courant continu, l'encodeur et les pilotes L298, ont été explorés. Ensuite, la méthodologie utilisée dans la modélisation des systèmes et du contrôle a été présentée, tout en précisant l'approche utilisée pour développer des modèles mathématiques, concevoir des algorithmes de contrôle et mettre en œuvre des solutions logicielles. Les résultats obtenus à partir du système mis en œuvre ont été analysés et comparés au comportement attendu sur la base de modèles de système et d'algorithmes de contrôle. Les facteurs affectant les résultats, tels que la résolution du capteur et la dynamique du système, sont discutés en détail. L'efficacité du système de contrôle pour obtenir un positionnement précis du moteur et une régulation de la vitesse a été évaluée.

**ملخص**

يمثل هذه العمل دراسة التحكم في موضع وسرعة محرك التيار المستمر باستعمال الاردوينو

في هذه الدراسة ، تم استكشاف جوانب الأجهزة في النظام ، بما في ذلك محرك التيار المستمر والتشفير وبرامج تشغيل . L298 تم تقديم المنهجية المستخدمة في نموذجة النظام والتحكم ، مع تحديد النهج المستخدم لتطوير النماذج الرياضية ، وخوارزميات التحكم في التصميم ، وتنفيذ الحلول البرمجية. و تم تحليل النتائج التي تم الحصول عليها من النظام المنفذ ومقارنتها بالسلوك المتوقع بناءً على نماذج النظام وخوارزميات التحكم. العوامل التي تؤثر على النتائج ، مثل دقة المستشعر وديناميكيات النظام ، تمت مناقشتها بالتفصيل. وتم تقييم فعالية نظام التحكم في تحقيق التموضع الدقيق للمحرك وتنظيم السرعة

# CONTENTS



## List of tables



## List of figures

## List of abbreviations

## Chapter I: Project Hardware

## Chapter II: Methodology for System Modeling and Control

## Chapter III: results and discussions

# General introduction

# General Introduction

In the field of automation and robotics, precise control of the position and speed of DC motors is crucial for many applications. DC motors are widely used due to their simplicity, affordability, and adaptability to a variety of systems. However, to ensure accurate motion control, it is necessary to use sensors such as encoders to measure the motor's position and speed.

Arduino, an open-source electronics development platform, provides a practical solution for controlling DC motors with encoders. Arduino combines user-friendly programming with hardware flexibility, making it a popular choice for motor control projects.

In this project, we focus on controlling the position and speed of a DC motor using an encoder and the L298 driver in conjunction with Arduino. The L298 driver is a specialized integrated circuit designed for controlling high-current DC motors. It offers advanced features such as motor direction control and speed regulation.

The main objective of this project is to implement a control system that can precisely position the DC motor using information from the encoder while regulating its speed. Our aim is to create a system capable of performing accurate and repeatable movements, which are essential for applications such as robotics, mechanical arms, and position tracking systems.

In our approach, we will use Arduino to receive signals from the encoder, which measures the motor's position and speed. The L298 driver then amplifies the control signals generated by Arduino, enabling control of the motor's direction and regulation of its speed based on the encoder information.

We will present the different steps required to realize this control system, including the hardware connection between Arduino, the L298 driver, the DC motor, and the encoder. We will also explain the configuration of the Arduino code to receive signals from the encoder and command the motor accordingly.

In summary, this project aims to provide a comprehensive control solution for the position and speed of a DC motor equipped with an encoder, using the L298 driver and Arduino. By combining the advantages of the L298 driver in terms of direction

control and speed regulation with the flexibility of Arduino, we can achieve precise and reproducible motions, opening up possibilities for various applications in the field of automation and robotics.

The research findings and implementation details are documented in a dissertation that is divided into three chapters:

Chapter 1: Project Hardware

Chapter 1 focuses on the hardware aspects of the project, providing essential background information on DC motors, encoders, and L298 drivers.

Chapter 2: Methodology for System Modeling and Control

Chapter 2 focuses on the methodology employed for system modeling and control.

It outlines the approach used to develop mathematical models of the system and design control algorithms. The chapter explains the step-by-step methodology for deriving the system equations based on the characteristics of the DC motor, encoder, and L298 driver.

Additionally, the chapter delves into the software implementation aspects. It covers the programming techniques utilized, including the selection of programming languages. It also discusses the simulation software employed to validate the system models and test the control algorithms. The chapter further describes the algorithms utilized for receiving and interpreting encoder signals, controlling motor direction, and regulating motor speed.

By integrating both the hardware and software aspects, Chapter 2 provides a comprehensive insight into the methodology used for system modeling and control, encompassing the derivation of mathematical models, the design of control algorithms, and the software implementation necessary for their execution.

Chapter 3: results and discussions

This Chapter presents the results obtained by the implemented system and provides a detailed discussion and analysis of these results, which are then analyzed and compared with the expected behavior based on the system models and control algorithms developed in chapter 2. These results are then analyzed and compared with the expected behavior based on the system models and control algorithms developed in Chapter 2. Any discrepancies or deviations from the expected results are examined and discussed in detail, taking into account factors such as sensor accuracy, system dynamics and control performance. In addition, the chapter explores the implications and significance of the results obtained in relation to the overall objectives of the project. It examines the effectiveness of the control system implemented to achieve precise motor positioning and speed regulation. Overall, Chapter 3 provides a comprehensive assessment of the results obtained, their interpretation and implications, forming the basis for the conclusions and recommendations set out in the final chapters of the project.

Finally, we'll end this dissertation with a general conclusion and outlook that will summarize the interest of our study.

# Chapter I:

## Project Hardware

## I.1Introduction

An electrical device known as a DC motor transforms electrical energy into mechanical energy.

A current-carrying wire put in a magnetic field receives a force that causes it to rotate this is how electromagnetic induction works DC motors are widely used in a variety of applications because they offer a number of advantages over other types of motors. One major advantage of DC motors is their simplicity, which makes them relatively easy to design, operate, and maintain. Additionally, DC motors are highly efficient, meaning they can convert electrical energy into mechanical energy with minimal loss, which helps to conserve energy and reduce operating costs.

Another key advantage of DC motors is their control capabilities. By varying the voltage applied to the motor, it is possible to control the speed and torque output of the motor, which makes DC motors highly adaptable to a wide range of applications. DC motors can also be configured to provide precise and accurate control, making them an ideal choice for use in robotics, automation, and other applications where precise control is critical.

## I.2 Description of a motor

DC motors are electromechanical transducers:

DC machine is equipped with an electromechanical energy converter reversible. It is capable of transforming electrical energy into mechanical energy (operation in engine), or vice versa; transform mechanical energy into electrical energy (operation in generator) [1].

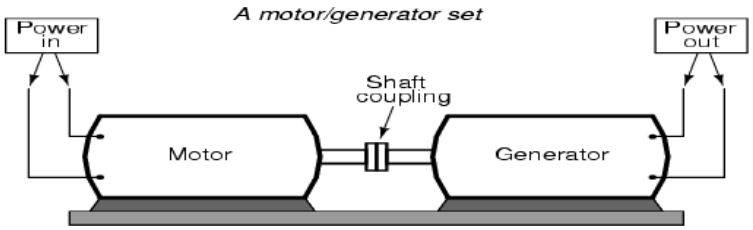



**Figure I.1**motor and generator

## I.3 The different modes of excitation

DC motors differ in the way in which excitation current is supplied. The different possible cases are:

### *I.3.1 The separately excited motor*

Is a type of DC motor where the excitation current is provided by an excitation winding separate from the motor itself. In this case, the motor has two main windings: the armature winding and the excitation winding.

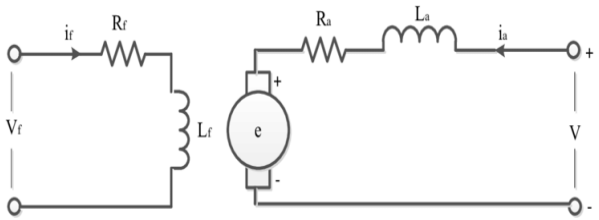
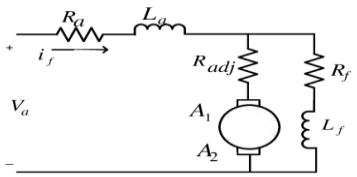



**Figure I.2** Electrical modeling of a separately excited motor

### *I.3.2 Shunt excitation motor (Derivative)*

The shunt excitation motor, also known as the derivative excitation motor, is a type of DC motor in which the excitation winding is connected in bypass with respect to the armature winding. In this case, the excitation current and the charging current pass through separate windings




**FigureI.3**Shunt excitation motor (Derivative)

### *I.3.3 Series excitation motor*

This motor is characterized by the series connection of the windings of the armature and the inductor. The inversion of direction of rotation sinbtenue either by the inversion of the polarity supply of the armature to the inducteur [2]

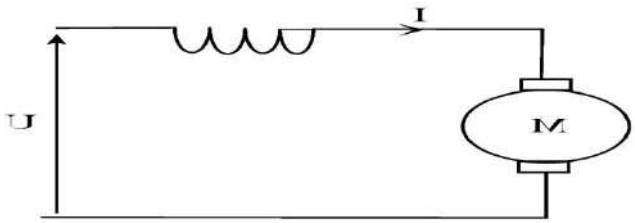



**Figure I.4** Electrical modeling of a series excitation motor

### *I.3.4 Compound excitation motor (compound)*

This motor is distinguished by the fact that it comprises two inductor circuits as well as the connection of the inductor circuits will combine the structure of the shunt excitation motor and the series excitation motor in order to retain the advantages of its two motor technologies. [2].

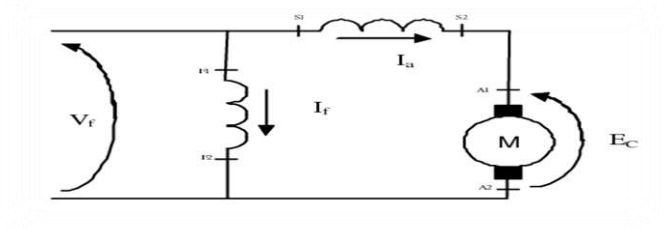



**Figure I.5** Electrical modeling of a compound excitation motor

## I.4 Operation of a DC motor

The DC motor is driven at constant voltage and rotates quickly angular proportional to the supply voltage, The standard voltage is 5 volts and therefore the motor runs at high speed. The problem is That because the engine is small, it is much less efficient to produce Couples. That is why it makes sense to put the gearbox (traingears) on the motor shaft to provide good torque. [3] The DC motor of a stator in the following is how it works.

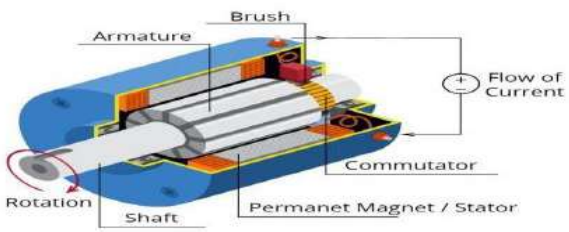



**Figure I.6** DC motor In detail

### *I.4.1 The stator*

Is a fixed element, whose role is to create a magnetic flux. Its function is ensured either by a permanent magnet or by an electric current circulating in a winding.

### *I.4.2 The rotor*

also known as an armature, is composed of a metal frame comprising notches on which windings are placed.


**Figure I.7** DC motor stator and rotor

## I.5 How to Specify A Motor Encoder

When specifying a motor encoder, there are several key factors to consider:

### *I.5.1 Output type*

Depending on the kind of signal it generates, a motor encoder might have many output kinds. With a motor encoder, the following outputs are the most typical:

### *I.5.2 An incremental encoder*

Generates output signals that show the amplitude and direction of movement.

It generates a string of pulses that are counted in order to detect the orientation and rotational velocity of the motor shaft. The resolution of the encoder is determined by the number of pulses per revolution (PPR).

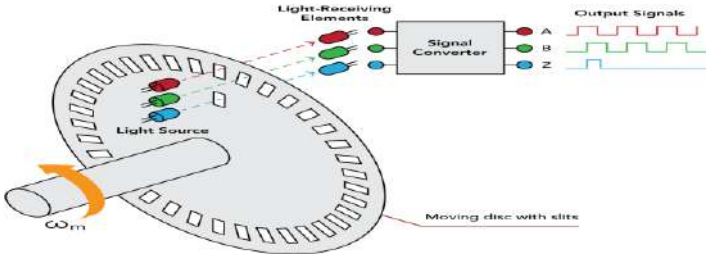



**Figure I.8** An incremental encoder

### I.5.3 Absolute encoder

At each position of the motor shaft, an absolute encoder gives a distinct digital code. An absolute encoder does not need a reference point since its output is direction-independent. Even after a power failure or system restart, it may still give precise position input.

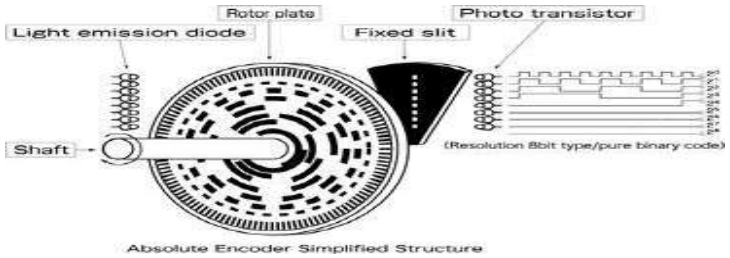



**Figure I.9** Absolute encoder

### I.5.4 Linear encoder

As opposed to rotational motion, a linear encoder gives position feedback for linear motion. For applications requiring precise position feedback for linear positioning, it can be installed on a linear actuator.

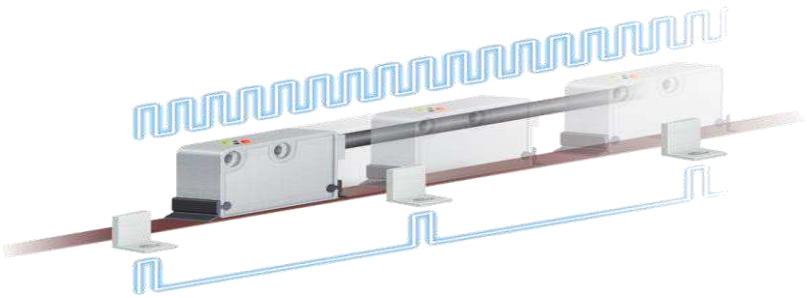


**Figure I.10** Linear encoder

Based on the needs of the application, the motor encoder's output type should be chosen. For instance, a speed control application would benefit from an incremental encoder, but a high-precision positioning application might require an absolute

encoder.

Applications requiring linear motion, such as linear actuators or stages, may make use of linear encoders.

### I.5.5 Resolution

The quantity of pulses per revolution (PPR) that an encoder can generate determines its resolution. The measurement of the motor's location and speed is more exact the greater the resolution.

Absolutely, it is accurate. The number of pulses per revolution (PPR) that an encoder can generate, which represents the accuracy of the measurement of the motor's position and speed, is referred to as resolution. The better the resolution and the more accurate the measurement, the more pulses an encoder generates every revolution. This is crucial in applications like robotics,

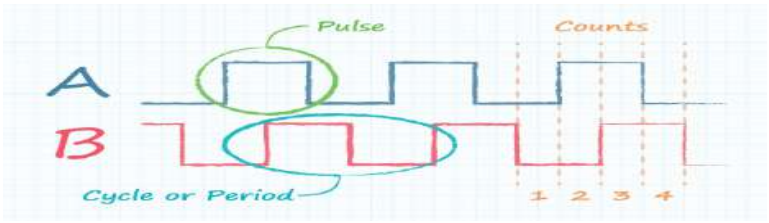



**Figure I.11** The drawing illustrates the several ways encoder resolution can be defined

### I.5.6 Environmental factors

The encoder must be able to function in the same environment that the motor is being operated in. Included in this are elements like temperature, humidity, and vibration.

The precision of the encoder may be impacted by thermal expansion if it is utilized in a high-temperature environment. A high-vibration environment can also cause mechanical shock to the encoder, which can harm its components and impair its function.

### I.5.7 Price

Depending on an encoder's features and performance, the price might vary significantly. It's crucial to strike a balance between performance expectations and available funding.

When choosing an encoder, it's critical to strike a compromise between the performance needs and the available budget. For some applications, a high-performance encoder could be required, yet in other situations, a less expensive encoder might be adequate.

## I.6 Types of Motor Encoder

Motor encoders of various varieties are utilized with various types of motors. Here are several examples:

### I.6.1 Servo Motor Encoders

A servo motor encoder is a tool that gauges the speed and location of a servo motor's rotation. It is made up of a rotating disc with parallel, uniformly spaced slots or lines that block a light beam and generate electrical signals that represent the location and speed of the motor.

Servo motor encoders come in two primary categories: incremental and absolute. Absolute encoders offer encoders provide information on changes in position and speed with respect to a reference point information on the precise location of the motor shaft, whereas incremental In conclusion, servo motor encoders are crucial parts for precise positioning and control of servo motors in a variety of applications, including CNC machines, robotics, and automation.[4]

**Figure I.12** Servo Motor Encoders

In conclusion, servo motor encoders are crucial parts for precise positioning and control of servo motors in a variety of applications, including CNC machines, robotics, and automation.

### I.6.2 Stepper Motor Encoders

Devices called stepper motor encoders are employed to gauge the speed and position of a stepper motor. As stepper motors lack internal encoders, unlike servo motors, external encoders must be used to give exact feedback and closed-loop control.

By giving feedback on the location, speed, and direction of the stepper motor, stepper motor encoders enable precise control of its motions. They are frequently utilized in systems for automation, machine tools, and robotics.

It's vital to remember that using stepper motor encoders makes the control system more complex and expensive. Yet, they can offer important advantages in terms of performance, accuracy, and dependability, particularly in high-precision applications.

**Figure I.13**Stepper Motor Encoders

## I.7 Outils d'expérience

- Arduino uno
- L298
- Motor Encoder

### I.7.1 Arduino uno

It was explained in the second chapter

### I.8 L298

Robotics and motor control applications frequently employ the twin H-bridge motor driver IC (integrated circuit) L298. It can manage up to 2 amps of current per channel and can operate two DC motors or one bipolar stepper motor. The L298 may

be controlled by TTL logic-level signals or PWM signals for speed control. It includes built-in protective features including heat shutdown and overcurrent protection.

### I.8.1 Interface L298 DC Motor Driver Module with Arduino

The L298 motor driver is what you'll likely need to start with even though you'll eventually need to learn how to operate DC motors in order to create your own robot. It has the ability

to regulate the two DC motors' rotational speed and direction. Moreover it has the ability to manage a bipolar stepper motor.[5]

**Figure I.14** L298 Motor Driver

### I.8.2 Controlling a DC Motor

A DC motor can only be completely under our control if we have complete control over its speed and direction of rotation. Using these two methods makes it feasible to do this.

PWM – to control speed

H-Bridge – to control the spinning direction

### I.8.3 PWM – to control speed

The input voltage can be changed to change the speed of a DC motor. One strategy that is widely employed to do this is pulse width modulation (PWM).

PWM is a technique that modifies the average value of the input voltage by sending a series of ON-OFF pulses. The pulse width affects the duty cycle, which is also sometimes referred to as the average voltage.

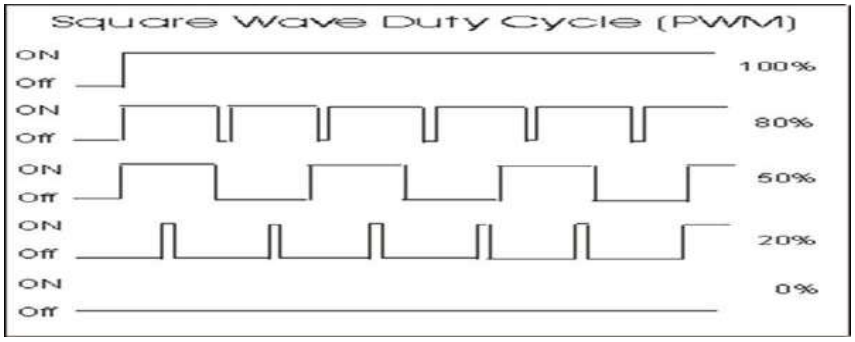



**Figure I.15** PWM – to control speed

### I.8.4 H-Bridge to control the spinning direction

By altering the polarity of a DC motor's input voltage, the direction in which it spins may be changed. Using an H-bridge is a common method for doing this.

Four switches are stacked in a H shape, with the motor at the middle, to form an H-bridge circuit.

Reversing the polarity of the electricity provided to the motor requires simultaneously closing two specified switches. As a result, the motor's direction of rotation changes

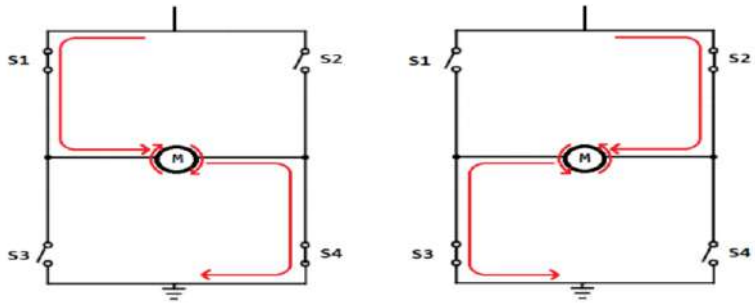



**Figure I.16** H-Bridge

### I.8.5Power Pins

**VS** and VSS are the L298d motor driver's two input power pins

**Vss** The motors are powered via pin, which powers the internal H-Bridge of the IC. The input voltage range for this pin is 5 to 12V.

**Vs** is used to power the logic circuitry within the L298 IC, and can range between 5V and 7V.

**GND** is the common ground pin.

## I.9 The potentiometer

The potentiometer is connected to the gear train via a toothed wheel, to pull the angle and the real speed of the motor, that is to say the instantaneous real position of the motor, it acts as a sensor rudder position. It delivers a voltage proportional to the angular position, the voltage used by the electronic card in the loop. [6]

**Figure I.17** the potentiometer

## I.10.Conclusion

Encoders can be added to DC motors to track the location and speed of their revolution. While the motor shaft rotates, encoders generate pulses that may be counted and utilized to determine the speed and location of the motor. The use of encoders with DC and Arduino.

# Chapter II:

## Methodology for System Modeling and Control

## II.1 Introduction

This chapter explores the integration of Arduino, Proteus, and COM0COM in modeling and controlling DC motors. These tools have revolutionized electronic prototyping and simulation.

Arduino is an open-source microcontroller platform for interactive electronic projects, while Proteus enables virtual testing of electronic circuits. COM0COM facilitates seamless communication between software and hardware interfaces.

By combining Arduino, Proteus, and COM0COM, researchers and practitioners can efficiently prototype and simulate complex motor control algorithms.

Throughout this chapter, practical examples demonstrate the potential of this integrated approach in real-world applications.

In summary, this chapter provides an overview of Arduino, Proteus, and COM0COM integration in DC motor modeling and control, driving advancements in the field.

## II.2 Arduino history

The first ever Arduino board was born in the classrooms of the Interactive Design Institute in I vrea, Italy in 2005. It is an Open Source microcontroller based development board that has opened the doors of electronics to a number of designers and creative engineers.

## II.3 About the Arduino

The new prototype board, the Arduino, created by Massimo Banzi and other founders, is a low cost microcontroller board that allows even a novice to do great things in electronics. An Arduino can be connected to all kind of lights, motors, sensors and other devices; easy-to-learn programming language can be used to program how the new creation behaves. Using the Arduino, you can build an interactive display or a mobile robot or anything that you can imagine.[7]

## II.4 Arduino Nano

Arduino Nano is a small breadboard-friendly version of Arduino UNO. It has more or less functionality of the Arduino UNO but in a small form factor. The only major differences from UNO are the lack of a DC power jack, the usage of a Mini USB port instead of a USB B port, and the USB-TTL converter chip. Nano uses an FT232, a dedicated USB-UART bridge chip from FTDI instead of an **ATMega16U2**. It is also a very popular choice among the developers just like UNO because of its small size and cheap price.[8]

## II.5 Arduino Mega2560 Rev3

Arduino Mega 2560 is the biggest of all the boards we have discussed so far. It is designed for applications where a lot of I/O or peripherals are needed. It is powered by a bigger and more capable processor the **ATMega2560**. This board has the greatest number of I/O than most other boards, 54 I/O pins (of which 15 can be used as PWM outputs), 16 analog inputs, and 4 UARTs. It has more flash storage and SRAM than most other basic Arduino boards. It is most popular with the open source CNC and 3D printer community as well as the open source PLC community.[8]

## II.6 Arduino UNO R3

**Arduino UNO** is the most popular and widely used development board. It is powered by an **ATMega328P microcontroller**. It is the most popular choice among the community because it’s, cheap, easy to learn and use, and also a variety of premade modules are available for this which makes it easier for developing new projects or prototypes. It consists of 14Digital I/O out of which 6 pins are 8bit PWM pins, 6 pins are10-bit Analog inputs, and basic communication ports like SPI, I2C, and UART.[8]

Now, there are many **different types of Arduino UNO** boards available across the global market, but most of these boards are the clone or copy version of the original UNO board that you see above. Hence the color or the appearance of the board might be different than what is shown above.[8]

There are many other types in Arduino, but we used the latter in the study of these, which we will learn more details about.

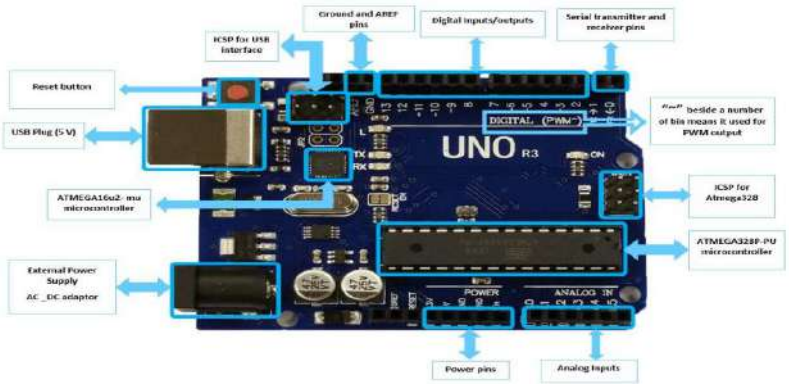



**Figure II.2.**Arduino UNO port

## II.7 The process of developing with Arduino

The development process for Arduino involves using the Arduino development environment, which is a user-friendly software application designed to facilitate the writing and uploading of code to an Arduino board. The environment is made up of various tools that work in tandem to simplify the process, making it accessible to individuals with varying levels of experience.

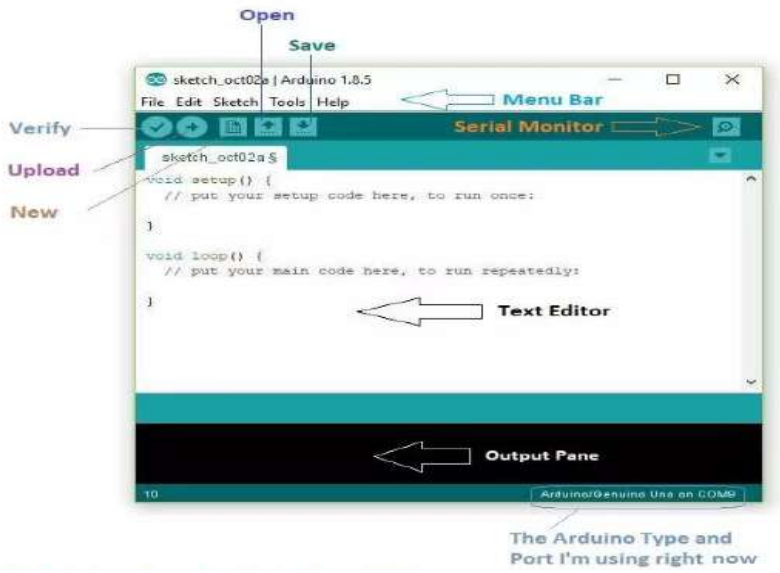



**Figure II.3** Software application Arduino

## II.8 Serial Monitor

The Serial Monitor is a useful tool in the Arduino development environment that can help with debugging by showing any serial information being transmitted by the Arduino board. When the Serial Monitor is opened, a new window will display real-time data being sent and received by the board.

This feature is particularly useful for identifying and troubleshooting any issues with your code or circuitry during the development process. The Serial Monitor can be used to view sensor readings, debug communication protocols, and troubleshoot any other issues that may arise.

## II.9Proteus 8 Professional

In the realm of electronics and circuit design, software programs play a crucial role in simulating and testing circuits before they are physically implemented. Among the numerous software solutions available, Proteus stands as a premier tool that offers unparalleled capabilities for electronic circuit simulation and design. This software sets itself apart from other programs in its class by combining a comprehensive set of features, user-friendly interface, and accurate simulation results.

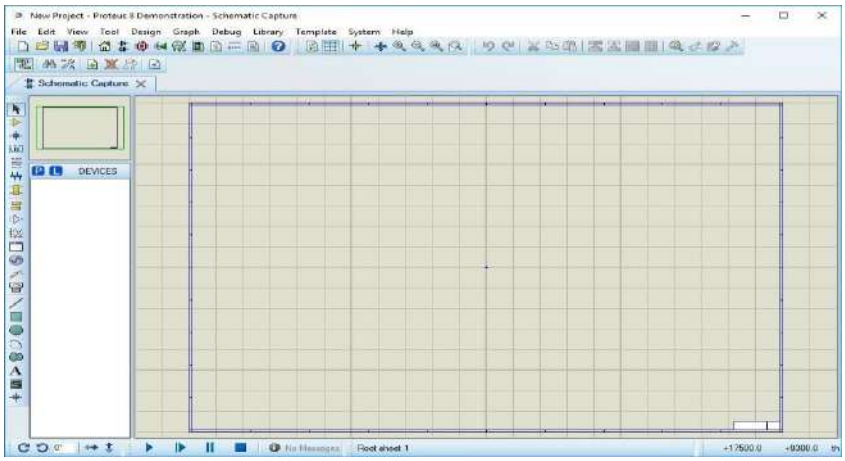



**Figure II.4** Proteus interface

## II.10.Null-modem emulator (com0com)

Analyzing, Visualizing, and interacting with your data is one of the coolest things an engineer can achieve The main purpose of this project is to visualize and analyze our generated data directly from Proteus 8 professional to your Arduino IDE. This can b done by connecting your Arduino IDE to Proteus 8 professional via a

virtual serial port. The virtual serial port virtually connects the Arduino IDE to Proteus 8 professional using two inactive COM ports.


**Figure II.5** Interfacing between Arduino and Proteus using COM0COM

To achieve this idea of a project, we need to use a virtual serial port that acts as a bridge between the two software, that is the Arduino IE a Proteus 8 Professional. Below is the software nee to accomplish this job.

- Com0com
- Arduino IDE
- Proteus 8 Professional

### II.10.1 Com0com

The Null-modem emulator is a free GPL-licensed open source kernel-mode virtual serial port driver for Windows. You may construct an endless number of virtual COM port pairs with the Null-modem emulator and use any pair to link one COM port-based application to another. Two COM ports are provided by each COM port pair. One port's output is the other port's input and vice versa. Device emulators can use the Null-modem emulator to offer a serial interface.

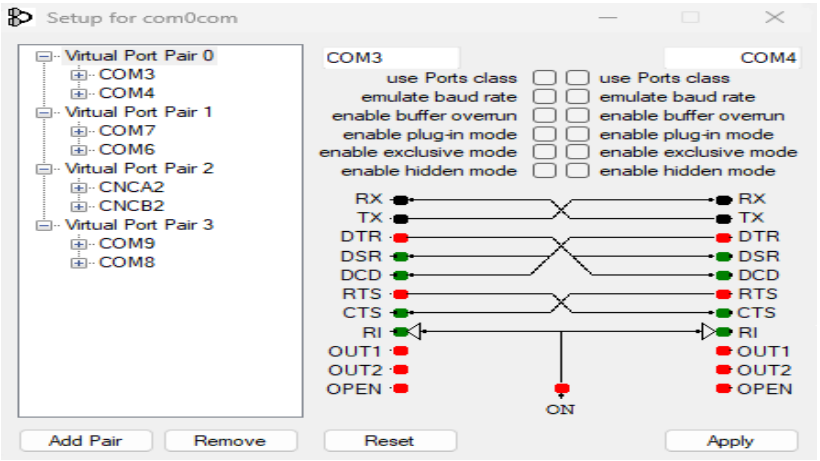



**Figure II.6** Null-modem emulator (com0com)

In this example, the device emulation program uses one of the pair's ports, while the other can be utilized by a Windows or DOS application that needs to interface with the device via a COM port. For example, to send/receive faxes over IP, use a virtual COM port pair to connect Windows Fax to t38modem (T38FAX Pseudo Modem, part of the t38modem project). For COM port redirectors, the Null-modem emulator can be used to offer a serial interface. You can communicate with TCP/IP servers through a serial interface using the com2tcp (COM port to TCP redirector, part of the com0com project).

If you think com2tcp is what you're looking for but can't locate some key features (like RFC 2217 support), try hub4com instead. It is possible to handle data and signals from a single serial device by a number of different programs using the hub4com (HUB for communications, part of the com0com project) (for example, several applications can share data from one GPS device). It's also feasible to use the remote computer's real serial ports as if they were on a local computer.[9]

## II.11 Modeling a DC motor

Modeling a direct current motor (DC) involves establishing the relationship between its rotational speed and the voltage applied to its terminal.

The DC motor equations are given below:

$$u(t) = R.\,i(t) + L\frac{di(t)}{dt} + e(t) \qquad (1)$$

$$e(t) = Ke\Omega(t) \qquad (2)$$

$$J\frac{d\Omega(t)}{dt} = Cu(t) - Cr(t) - f\Omega(t) \qquad (3)$$

$$Cu = Kc.\,i(t)$$

u(t) : voltage applied to motor terminals [V]

e (t) : electromotive force [V]

i(t) : current [A]

Cu(t) : engine torque [N.m]

Cr (t) : the resistant couple [N.m]

$\Omega(t)$ : the resistant couple [N.m]

R : the strength of the motor armatures [Ω]

L : inductance of motor armatures [H]

J : engine inertia [kg.m2 ]

f : coefficient of friction [N.m.s]

Kc : engin torque constant [N.m/A]

Ke : electromotive force constant [V.s/rad]

**II.11.1 Motor transfer function**

$$\mathrm{Ri(p)} + \mathrm{Li(p)} + \mathrm{E} = \mathrm{U(p)} \qquad (4)$$
$$\mathrm{E} = \mathrm{Ke}\Omega\mathrm{(P)} \qquad (5)$$
$$\mathrm{Jp}\Omega\mathrm{(p)} = \mathrm{Cu} - \mathrm{Cr} \qquad (6)$$

By combining (4) and (5) we obtain:

$$\mathrm{U(p)} - \mathrm{Ri(p)} + \mathrm{Lpi(p)} + \mathrm{Ke}\Omega\mathrm{(p)}$$

By modifying (6) we have:

$$\mathrm{Jp}\Omega\mathrm{(p)} = \mathrm{Kci(p)} - \mathrm{Cp} - \mathrm{f}\Omega\mathrm{(p)}$$

We deduce the expression of Ω(p):

$$\Omega\mathrm{(p)} = \frac{Kci(p) - Cp}{f + IP}$$

$$\mathrm{I(p)} = \frac{\mathrm{f} + \mathrm{JP}}{Kc}\left(\Omega\mathrm{(p)} + \frac{\mathrm{Cp}}{\mathrm{f} + \mathrm{JP}}\right)$$

It is now injected into (4):

$$\mathrm{U(p)} = \Omega\mathrm{(p)}\left(\frac{(\mathrm{R} + \mathrm{LP})(\mathrm{f} + \mathrm{JP})}{\mathrm{Kc}} + Ke\right) + \frac{\mathrm{R} + \mathrm{LP}}{\mathrm{f} + \mathrm{JP}}\mathrm{Cp}$$

In order to simplify the system, we can take Cp to be null and void as it is thought that the moment of loss cutting (which is viewed as a disturbance) is insignificant in comparison to the moment of the electromagnetic torque (Kci (t)).

$$U(p) - \Omega(p)\left(\frac{(R + LP)(f + JP)}{Kc} + Ke\right)$$

The desired transfer function H(p) is between the voltage entering the motor U(p) and the output speed Ω(p). [5], [6]

$$H(p) = \frac{\Omega(p)}{U(p)} = \frac{Kc}{(R + LP)(f + JP) + KcKe}$$

The DC system of equations is a system of coupled differential equations, difficult to solve in this form.

But by applying a Laplace transform to them, these equations

become algebraic and the system linear.

This system of equations can easily be associated with a block diagram which will be the basis of the Simulink digital model.

### II.11.2The Simulink model of the DC motor

#### *II.11.2.1block diagram:*

#### *II.11.2.2 engine parameters*

Ra = 12 OHM;

La = 10*10^(-3 ) H ;

Ki = 0.7 N.m/A;

Kb = 0.8598 V.s/rad;

Jm = 6.8*10^(-4) kg.m2 ;

Bm = 0.02 N.m.s/rad;

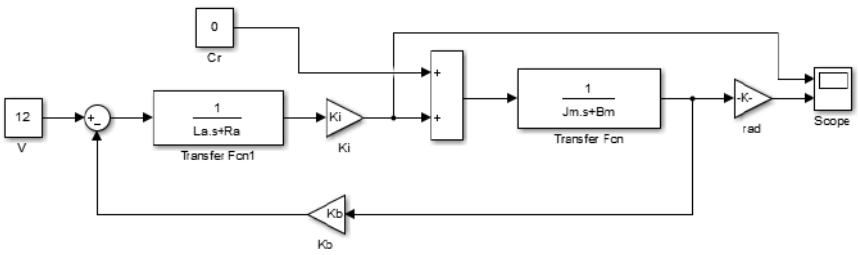



**Figure II.7** DC motor block diagram

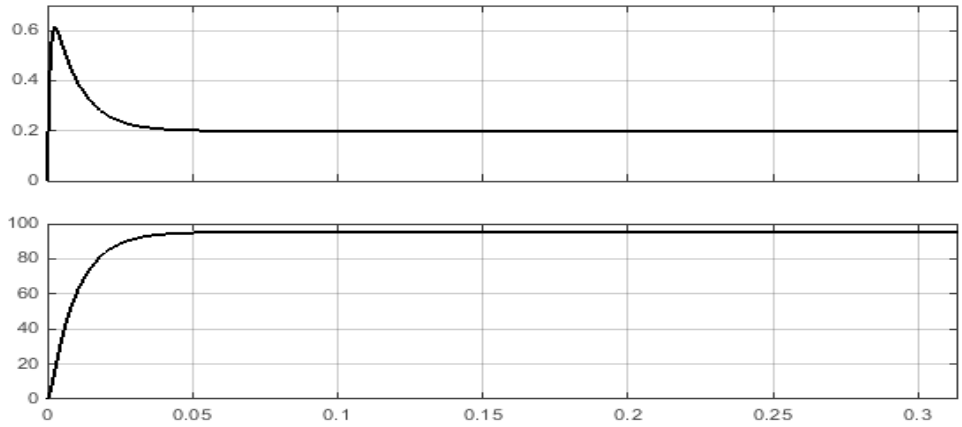



**Figure II.8** Engine speed and torque

### II.11.3 DC motor control

There are different ways of controlling a DC motor. **Speed-control** and **position-control** are two common examples, but **torque-control** is possible as well. You can even make a temperature- or a noise-control if you wish. Logically enough, an appropriate sensor will be needed in each case i.e. a speed control will work well with a speed sensor.

You will use a PID controller.

#### II.11.4.1 PID control

A PID controller is a good example of motor loop control (though it can be used with various different things, like a kitchen oven or a space-exploration rocket), and widely used in robotic applications.

But what really is a PID programming code? The term PID stands for **proportional**, **integral**, **derivative**. The term PID stands for proportional, integral, derivative. They are the three main components of the code (associated to three respective constants that I will talk about later).

Highly simplified expression of a PID command, with e as the error between the set-point and the actual measured value.

Note that this kind of command can be sometimes simplified into a PI, PD, or even a P command. It depends on the system.

### II.11.4.2 We have constants related to the PID

Kp: relative constant, determines the speed of movement of the motor towards its new position;

Ki: the integral constant, specifying how the motor will position itself at the end of its movement. The positioning angle is not quite equal to the set value. This results in an error called static error. This is a fixed error that will minimize the integration term.

Kd: A derived constant, when the motor is moving towards a desired position it will overshoot it, possibly oscillating, before leveling off. It is therefore this excess that will reduce the derived constant. [11]

The choice of these three constants is not due to chance, a bad choice can lead to unexpected results. The assembly may begin to wobble (the motor shaft will come and go on its own without ever settling), for example. There are ways to determine these constants: using graphs or other numerical tools, and we will rely in our study on MATLAB.

### II.11.5.3 Simulink model of a PID-controlled DC motor

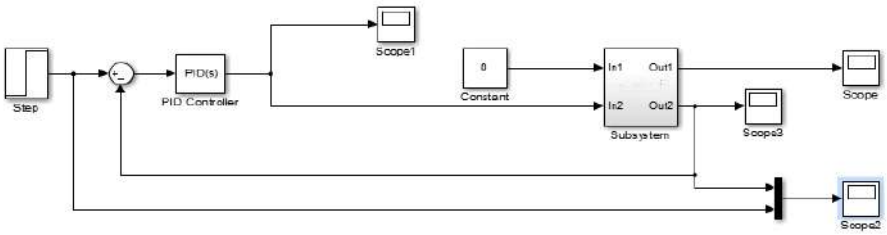



**Figure II.9**DC motor control diagram

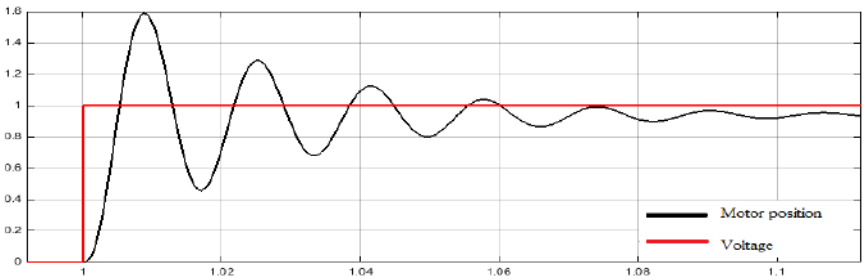



**Figure II.10** DC Perform output before controlling the PID values

Before modifying the values of Kp, Ki, and Kd, we notice instability in the system. We modify the values and observe the result again.

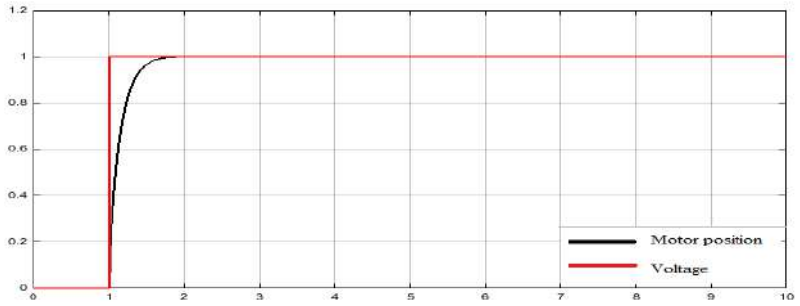



**Figure II.11**Output performance after controlling the PID values

After adjusting the PID values in MATLAB, the following values for Kp, Ki, and Kd were obtained:

Kp = 8.1, Ki = 0.095, Kd = 0.095

After getting the PID values from MATLAB it can now be applied to controlling the motors using Arduino.

## II.12 Encoder modeling

Full rotation (360 degrees) one pulse of width is equal to a quarter of a step angle (90 degrees). The position of the column corresponding to this pulse can be considered as a reference position Conversion managers A are usually respectively named B, the reference is called the product of the "mark" Z.

Incremental encoder modeling

The input signal of the incremental encoder is the angular position θ of its column Regarding a fixed reference axis. Output signals are pulses that have turned By a quarter of an angle step A (θ) and B (θ), respectively the sign Z (θ). If θ pis the angular step of the encoder,

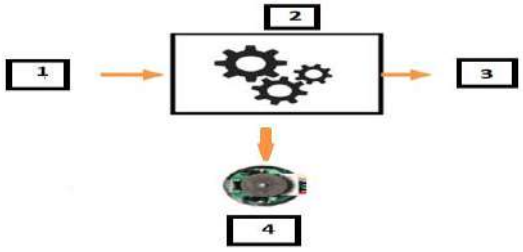



**Figure II.12**Kc Encoder modeling

1: Angle (θ)

2: Kc Encoder modeling

3: Signal PWM

4: Encoder

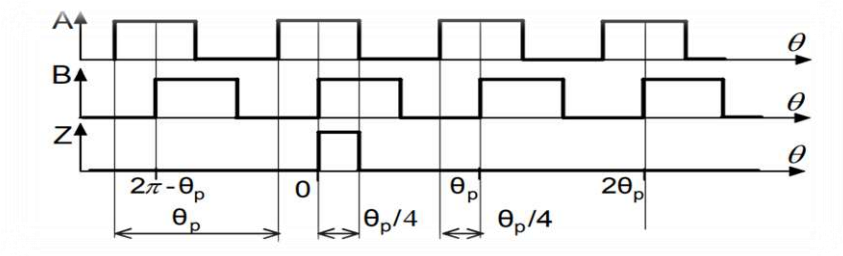



**Figure II.13** A signal that is read from the encoder

Diagram of the output signals generated by the incremental encoder

and through the angle of rotation of the shaft is equal to the angle Graduation step θp There are four switching events in the output pulses, and Therefore, the minimum increase in the rotation angle that can be detected by the encoder is$\frac{(\theta p)}{4}$

The number of pulses generated by the encoder will be rotation

$$\mathrm{Nr} = \frac{2\pi}{\theta p}$$

Angular step θp of Encryption = Θp

which is equal to the number of angular steps of graduation on the ring The track is on the rotor. Number of pulses per rotation with angular The speed of the column will determine the frequency of the encoder output signals .

$$Kc = \frac{\omega * \mathrm{Nr}}{2\pi}$$

Angle speed ω = 314 Rad / S

Nr = Number of pulses per rotation

$$\Theta = \frac{360}{Nr}$$

Full rotation (360 degrees) one pulse of width is equal to a quarter360 =

Nr = Number of pulses per rotation

### II.12.1 Encoder-based Position Computation

In an incremental cryptography-based system, the angular position θ indicates a constant

The reference axis is obtained

Θ=number Encryption pulses generated* Angular step θp of Encryption

In order to obtain the angular position θ

$$\theta = \frac{N}{\theta p}$$

N= number of pulses according to the position of the column

Angular step θp

## II.13Conclusion

The integration of Arduino, Proteus, and COM0COM in the modeling and control of DC motors has proven to be a powerful and effective approach in the field of electronic prototyping. These tools have provided researchers and practitioners with cost-effective and time-efficient solutions for designing, simulating, and testing complex motor control algorithms.

By leveraging the capabilities of Arduino, users can easily program and integrate various components to create interactive electronic projects. Proteus Design Suite complements this by offering a virtual environment for simulating and validating electronic circuits, eliminating the need for physical prototyping. Additionally, COM0COM serves as a bridge between software applications and emulated hardware interfaces, facilitating seamless communication.

In conclusion, the integration of Arduino, Proteus, and COM0COM has revolutionized the field of electronic prototyping and simulation, particularly in the realm of DC motor modeling and control. This integrated approach offers immense possibilities for advancing the capabilities of motor control systems and driving innovation in the field

of electronics and robotics

# Chapter III:

## Results and discussions

## III.1 Introduction

DC motors have the advantage of rotating faster with a short response time at the expense of position accuracy. However, if we have a sensor that allows us to know the angular position of the DC motor shaft at any time, the situation changes. It becomes possible to eliminate these errors and precisely control the motor in terms of position or speed. This is the principle of DC motors, which can be found in robots. In this project, we will explain this issue in detail.

## III.2 Principle of operation of the assembly

### III.2.1 Schematic diagram

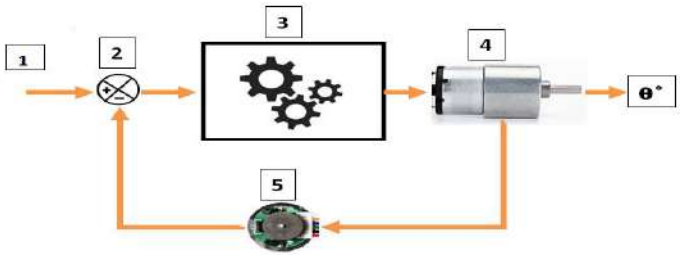



**Figure III.1** Schematic diagram [11]

1: reference or set point

2: Differentiator

3: Control device - command

4: DC Motor 12V

5: Hall effect magnetic position sensor

## III.3 How it works

The desired angle set point will be fixed using the potentiometer (1); the control - command system (3) sends an electrical command to the motor (4) which will move its axis to the desired position θ. A position sensor (5) indicates the position of the motor shaft to the differentiator (2), which calculates the difference between the actual position of this shaft and the set position. The error calculated by the differentiator is taken into account by the control system,

which can then send a new command to the motor (if this error is not zero) to return its axis to the set position.

In this way, the circuit operates in a closed loop. The position error is continuously evaluated, and as soon as it differs from zero, the control system will send a command to correct the motor position proportionally.

Theoretical approach:

In this case, we can say that we're dealing with a control loop; and the control device (3) is the controller, which in theory is a PID controller.[11]

Such a controller can be modeled as follows:

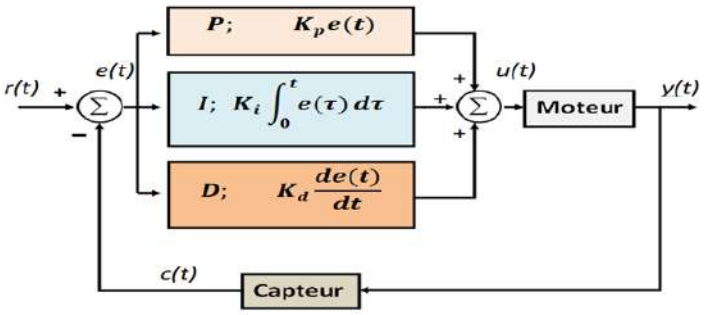



**Figure III.2** PID controller [11]

P: represents the Proportional term to the current error;

I: represents the Integral term, taking into account past values of the error;

D: represents the Derivative term, estimating the future direction of the error.

Regarding the signals,

r(t): is the desired set point or position;

c(t): the representation of the motor shaft position;

e(t): the position error;

u(t): the control signal, derived from the regulator;

y(t): the angle of displacement of the motor shaft.

The error is determined by equation (1): e(t) = r(t) - c(t)

The role of the regulator is to minimize this error as much as possible. Thus, the correction applied takes the form of a control signal u(t), given by equation (2):

$$u(t) = Kpe(t) + Ki\int_{t0}^{t} e(t)dt + Kd\frac{de(t)}{dt}$$

These two equations will be used in the Arduino board program.

In equation (2), we have the following constants:

Kp: the proportional constant, determining the speed of the motor's movement towards the new position; Ki: the integral constant, determining how the motor will settle at the end of its movement. The positioning angle may not exactly match the set value, resulting in a static error. The integral term minimizes this static error as much as possible. Kd: the derivative constant, when the motor moves towards the requested position, it may overshoot and possibly oscillate before stabilizing. The derivative term reduces this overshoot.

The choice of these three constants is not random, as a poor choice can lead to unexpected results. For example, the system may start oscillating (the motor shaft moves back and forth without stabilizing). There are methods to determine these constants, such as using charts or digital tools.

## III.4 Practical implementation of the model

The set point voltage from the 10KΩ-1turn potentiometer can vary from 0 to 5V. At 0V, the motor shaft will be at 0°, and at 5V, it will be at 360°.

The sensor used is a magnetic incremental encoder directly attached to the motor shaft. This encoder has a disc with magnetic poles and two inductive sensors mounted in quadrature, generating square signals shifted by a quarter of a period as the disc rotates.

The detection of rising and/or falling edges allows determining the number of pulses per revolution or the motor's rotation direction.

The calculation of the error and the control voltage will be done by the Arduino board. To drive the 12V DC motor, a power interface board in the form of an H-bridge, built around an L298, will be used.[11]

## III.5 Tools used in the project

DC motor encoder

Arduino Uno

L298 driver

Potentiometer

COMPIM

## III.6The general electrical circuit of the project

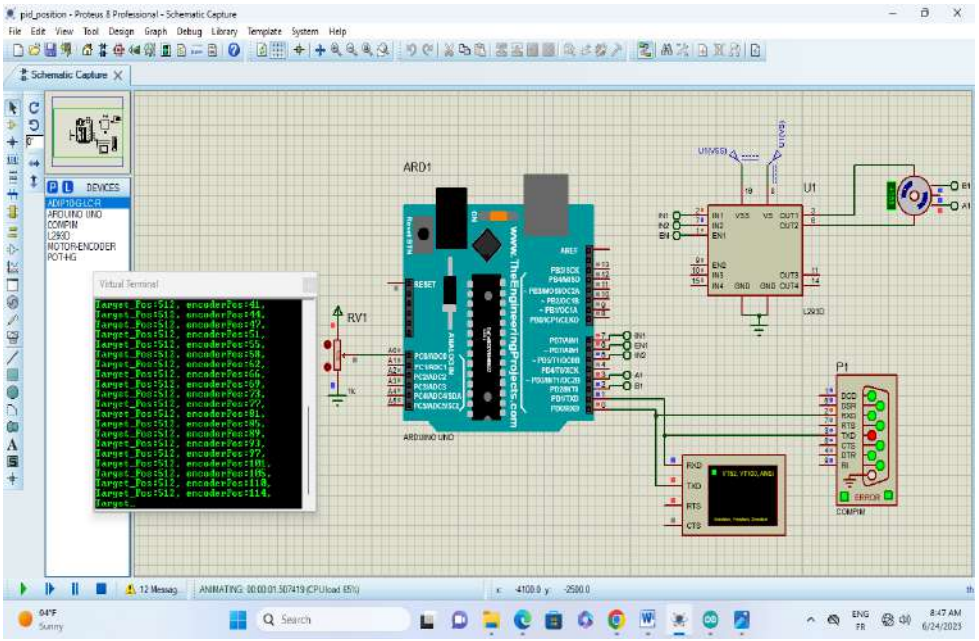



**Figure III.3** DC motor position control circuit

## III.7A motor, having the following characteristics

voltage: 12V;

Ra = 12OHM;
La = 10*10^(-3) H;
Ki = 0.7 N.m/A;
Kb = 0.8598V.s/rad;
Jm = 6.8*10^(-4) kg.m2;
Bm = 0.02N.m.s/rad;

## III.8DC motor operation by L298 driver

In this experiment, our objective is to manually control the speed and direction of rotation of the motor using the L298 motor driver (as explained in the first chapter) and the encoded DC motor. We will need two logic states for this, where the Encoder output A and Encoder output B of the encoder motor are connected to Out1 and Out2 of the L298 motor driver.

We will also connect the In1 and In2 pins of the L298 motor driver to logic state 1 and logic state 2, respectively. Additionally, we will connect the +5V pin from the power supply to the H bridge and establish a connection between the stable power supply's +12V pin and the GND pin of the H bridge.

By setting up this configuration, we will be able to control the motor's speed and direction effectively**.**

| ENA | In1 | In2 | Function (DC motor) |
|---|---|---|---|
| 1 | 1 | 1 | Stop |
| 1 | 0 | 0 | Stop |
| 1 | 1 | 1 | Cw |
| 1 | 0 | 1 | Ccw |

**Tables :**Determine the direction of rotation of the motor

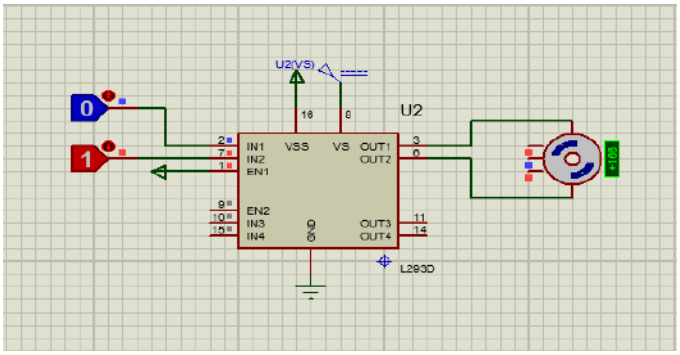



**Figure III.4** DC direction control clockwise and counterclockwise

## III.9DC Motor with Encoder and Arduino

In this experiment, we aim to determine the direction of rotation of the motor using the encoding outputs. Without the need for an L298 engine. Instead, we will use an Arduino Uno board and an encrypted DC motor.

The encoder works by observing changes in the magnetic field generated by a magnet attached to the motor shaft. The engine rotates the encoder outputs periodically. If we select encoder Pin A before encoder Pin B, the encoded DC motor will rotate clockwise (CW). On the other hand, if we select the Pin B encoder before the Pin A encoder, the encoded DC motor will rotate in a counterclockwise direction (CCW).

The ground pin of the engine connects to Arduino GND.

The engine coder A connects to pin 2 of Arduino. Pin 2 from Arduino will record each time there is an ascending digital signal from Encoder A.

The B encoder connects to pin 3 of Arduino. The signal that is read from pin 3 on Arduino. .

The VCC pin of the motor connects to the 5V pin in Arduino. This pin is responsible for providing power to the encoder program.

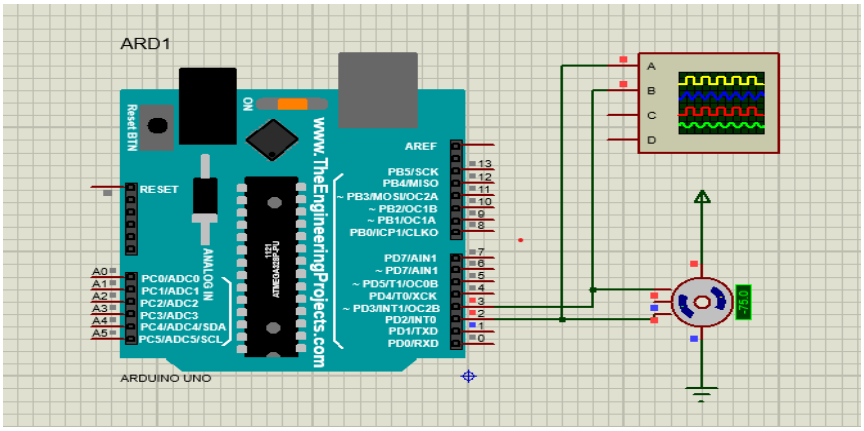



**Figure III.5** Determine the direction of rotation of the motor using the Arduino encoder

In these two images, we can see a clear difference in the direction of rotation of the motor. In the first picture of the cathode oscilloscope, we notice that the Arduino reads from output A first, corresponding to the blue curve with the value of HIGH, and after a small time difference it becomes yellow HIGH also, the motor rotates clockwise as shown in the first picture. In the second picture we see the opposite scenario where the Arduino reads from output B first, which causes the motor to rotate counterclockwise.

Or in the third picture, we notice an increase in the number of pulses that I read from the encoder by Arduino

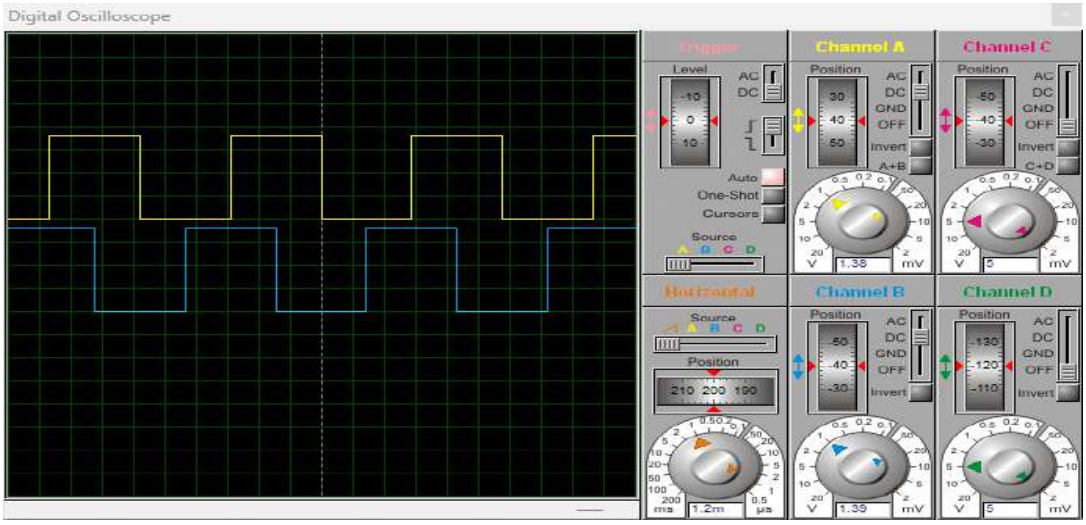



**Figure III.6** Rotate the motor clockwise from oscilloscope

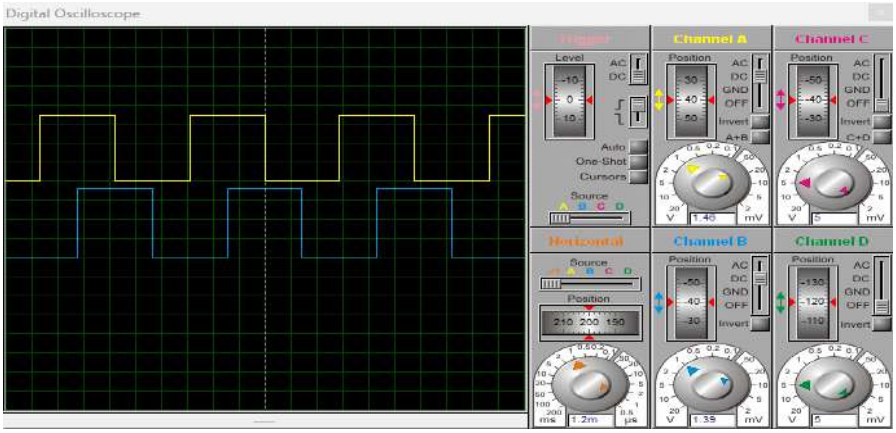



**Figure III.7** Rotate the motor counterclockwise from oscilloscope

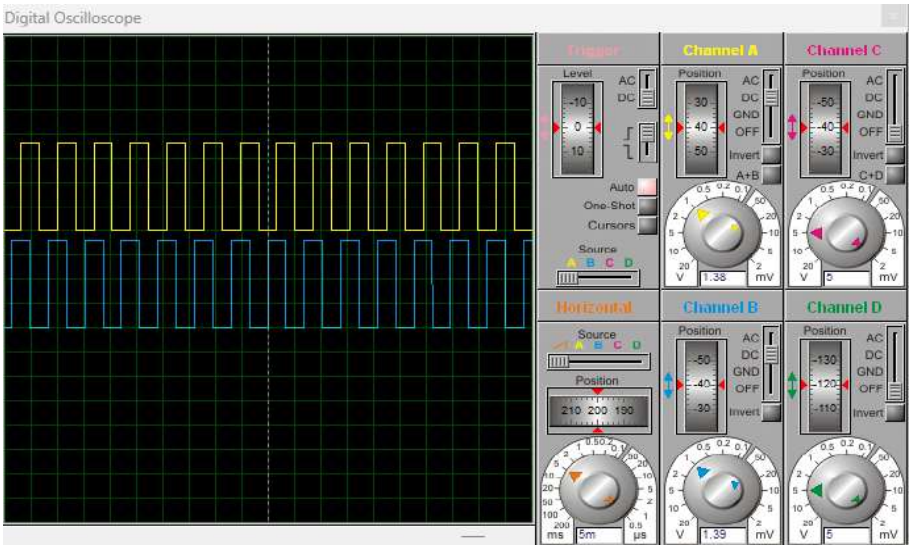



**Figure III.8** Increase the number of pulses read

## III.10Position control of a DC motor using an encoder and Arduino

Now that we have a complete understanding of how the encoder works and how pulses are generated, it's time to automate the process and control the position of the motor.

### III.10.1DC motor position control results before switching PID values.

#### *III.10.1.1 Position control by setting a specific target*

The following results were obtained from the experiment by showing it in a serial plotter

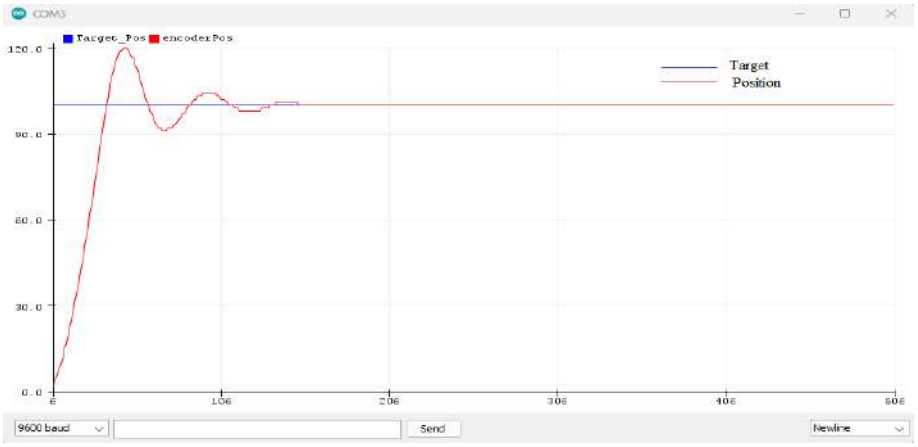



**Figure III.9** DC motor position control target

### *III.10.1.2 Position control via variable target setting (potentiometer)*

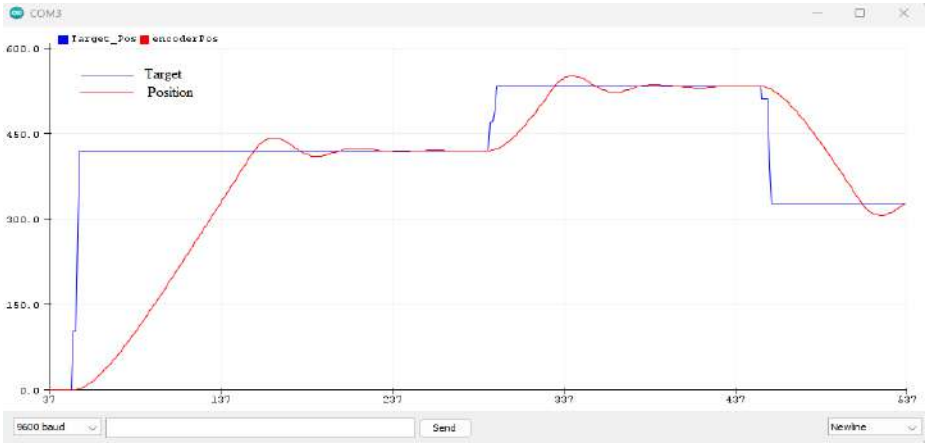



**Figure III.10** DC motor position change

From these previously shown results, it is clear that the system is not very stable. We note both overshoot and under-target, suggesting that improvements can be made in this system. By fine-tuning the PID parameters and optimizing the control algorithm, we can achieve a more precise and stable position control. The goal is to minimize overshoot, reduce settling time, and improve overall accuracy. With these improvements, the system will be able to track the desired position more effectively, resulting in enhanced performance and reliability.

### *III.10.1.3 The design a system for the following specifications*

Static error = 0

Response time at 5% = 0.9 s

Exit without overshoot = 0.01

## III.10.2DC motor position control results after setting the PID values.

### *III.10.2.1 Position control by setting a specific target*

The following results were obtained from the experiment by displaying them in a sequence diagram after controlling for the values of Kp and Ki, Kd

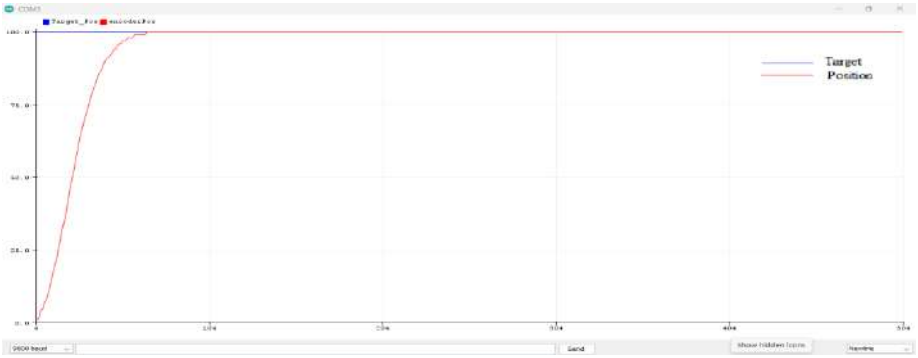



**Figure III.11** DC motor position control target

### *III.10.2.2 Position control via variable target setting (potentiometer)*

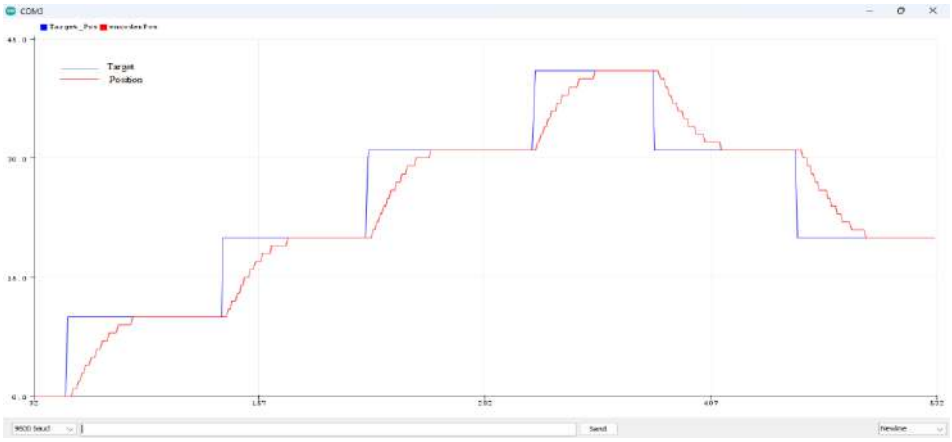



**Figure III.12** DC motor position change

From these latest results, we note an improvement over previous results and the required specifications have been achieved. The fine-tuning of the PID parameters and optimization of the control algorithm have significantly enhanced the system's stability and accuracy. The overshoot and under-target issues have been minimized, and the system now tracks the desired position more effectively. These improvements demonstrate the effectiveness of the implemented control strategy and validate its ability to meet the desired performance specifications. With these successful outcomes, we can conclude that the position control via variable target setting has been successfully accomplished.

## III.11 DC motor speed control using PID

Controlling the Speed of a DC Motor using PWM allows precise adjustment for optimal performance. Feedback mechanisms, such as encoders, ensure desired speed is maintained by comparing it with the actual speed.

### III.11.1 Single DC Motor Speed Control

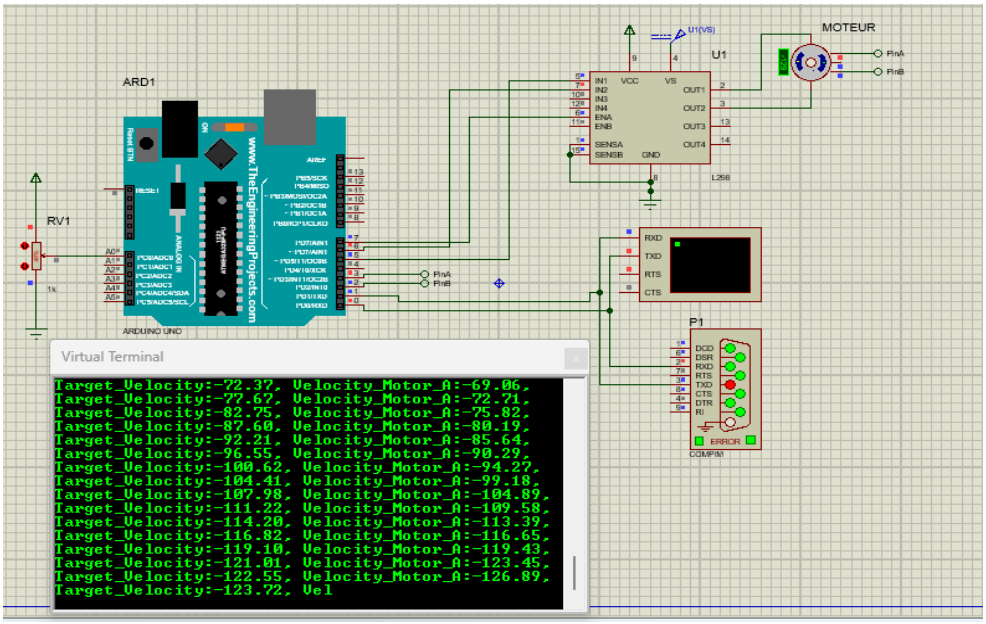



**Figure III.13** Single DC motor speed control circuit

Before entering the values of Kp (proportional gain), Ki (integral gain), and Kd (derivative gain), it is important to consider the desired performance and characteristics of the motor. These

parameters determine how the controller responds to changes in speed, errors, and disturbances. By tuning these values appropriately, the motor's speed control system can achieve stability, responsiveness, and optimal performance.

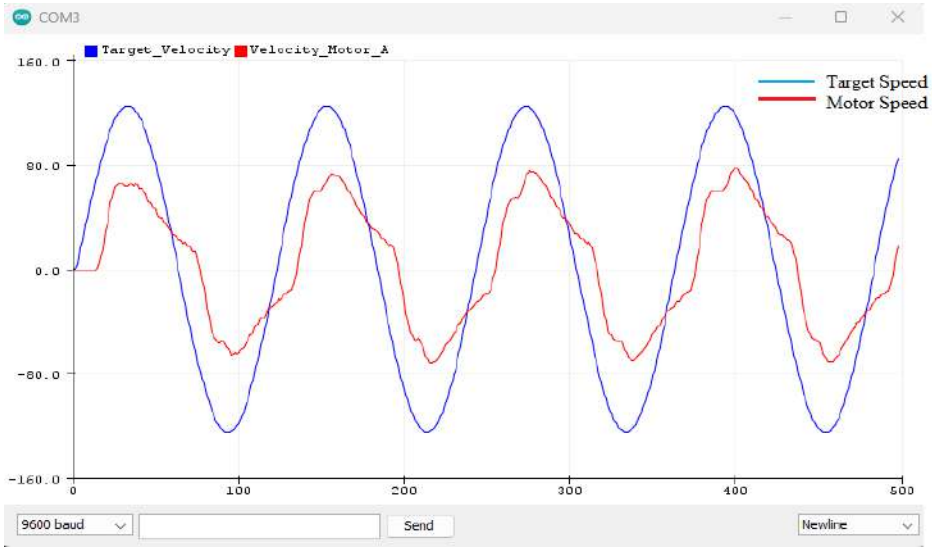



**Figure III.14** Engine control result before entering PID values

By observing the motor's speed control system without PID parameters, we can gather valuable insights such as response time, steady-state error, and sensitivity to disturbances. This information serves as a baseline for evaluating the effectiveness of the PID tuning process.

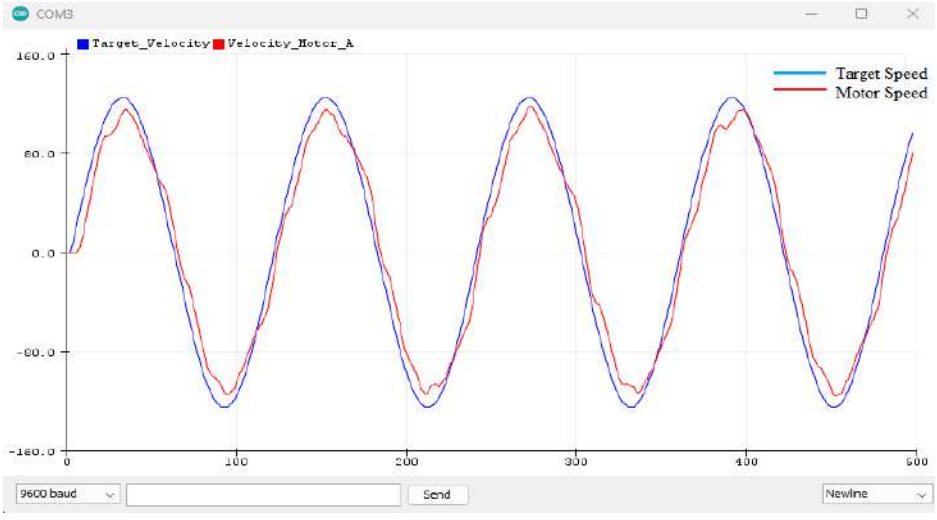



**Figure III.15** Result One motor speed control after inputting the PID values

Before entering PID values, the engine speed control system presents some limitations. I experienced slower response time, higher steady state error, and difficulty maintaining the desired speed under various conditions. However, after entering appropriate PID values (Kp, Ki, and Kd), remarkable improvements in system performance can be observed.

### III.11.2 Speed control of two DC motors

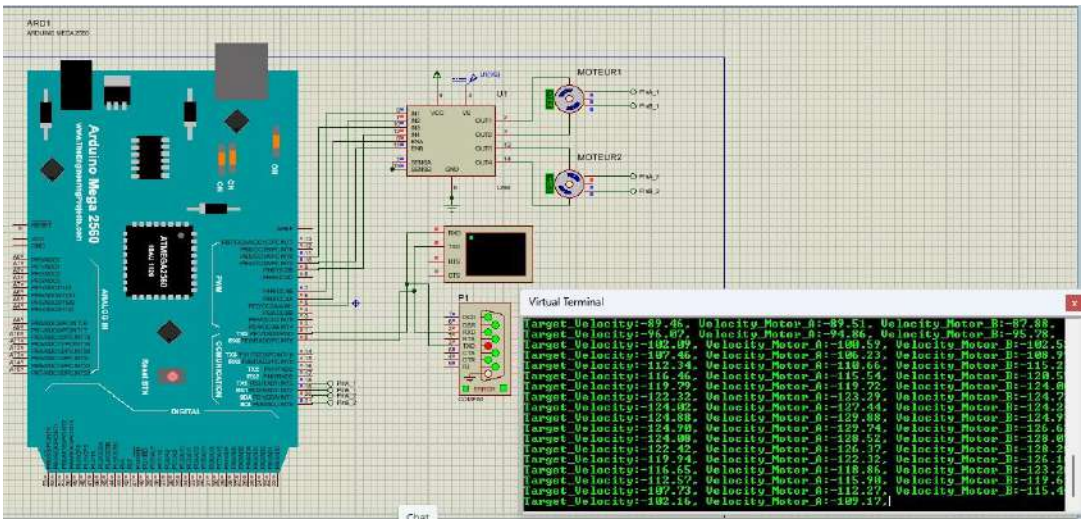



**Figure III.16** A speed control circuit for two DC motors

Prior to inputting the values of Kp (proportional gain), Ki (integral gain), and Kd (derivative gain), it is crucial to take into account the intended performance and characteristics of the motors. These parameters dictate the controller's response to speed variations, errors, and disturbances. By appropriately adjusting these values, the speed control system of the motor can attain stability, responsiveness, and peak performance.

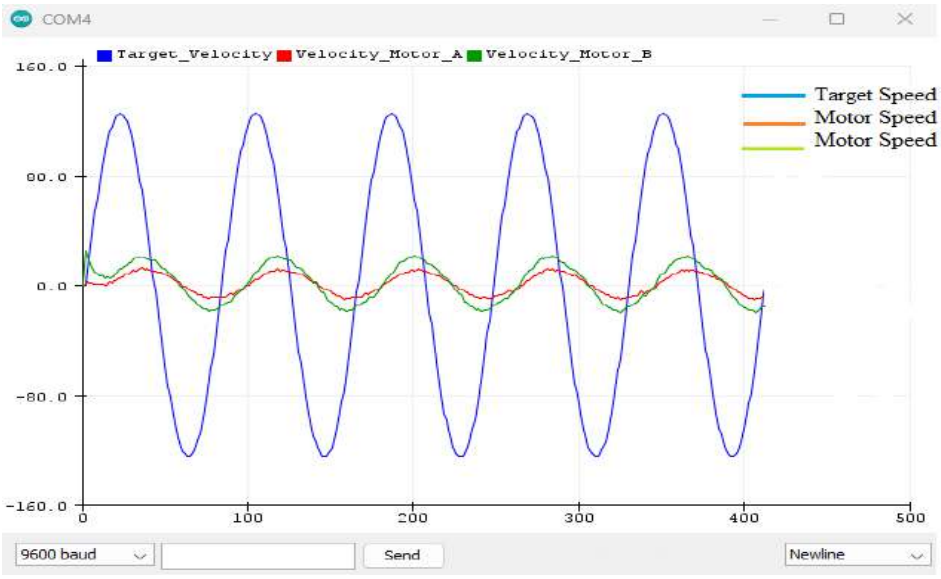



**Figure III.17** Engine control result before entering PID values

By observing the motor's speed control system using two motors instead of one, we can gather valuable insights such as response time, steady-state error, and sensitivity to disturbances. This information serves as a baseline for evaluating the effectiveness of the PID tuning process.

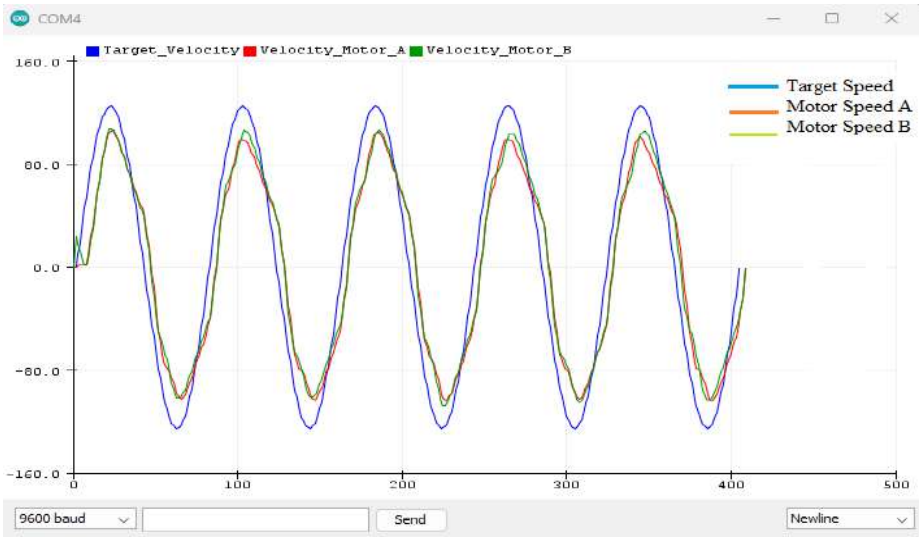



**Figure III.18** The result is a speed control of the two motors after entering the PID values

Comparing the speed control results of two motors before and after entering PID values reveals significant improvements in system performance. Prior to inputting the PID values, the motors may exhibit limitations such as slower response time, higher steady-state error, and difficulty

maintaining the desired speed under varying conditions. However, after inputting appropriate PID values (Kp, Ki, and Kd), the system shows remarkable enhancements.

After tuning the PID parameters, the motors demonstrate faster response time, reduced steady-state error, and improved ability to maintain the required speed under different conditions. The control system becomes more robust and adaptive, effectively compensating for disturbances and achieving better stability.

## III.12 The importance of PID to control the position and speed of DC motors

PID control is vital for controlling the position and speed of DC motors. It ensures accuracy, dynamic response, disturbance rejection, and adaptability. Implementing PID control allows for precise motor control and improved performance.

## III.13 Conclusion

The implementation of PID control of DC motor speed and position control brings great benefits. Through fine tuning of PID parameters, the engine performance is greatly improved. It achieves precise speed control, accurate position tracking, and improved response to turbulence. The PID controller allows the motor to adapt to different conditions, ensuring stability and optimum performance. Overall, PID control is a critical tool in maximizing the capabilities of DC motors, resulting in efficient and reliable operation in various applications.

# General conclusion

# General conclusion

In conclusion, this project has successfully tackled the position and speed control of a DC motor equipped with an encoder, using the L298 driver and Arduino. Thanks to the methodology described in the previous chapters, we have achieved significant results and made valuable contributions to the field of automation and robotics.

The project began with a complete understanding of the hardware components involved, including DC motors, encoders and the L298 driver. By integrating these elements with Arduino, we were able to develop a robust control system capable of precise, reproducible movements.

The modeling and control techniques implemented in this project proved effective in achieving the desired performance. The mathematical models derived in Chapter 2 provided a solid basis for the development of the control algorithms. Through careful tuning and testing, we were able to achieve precise position control and accurate speed regulation.

The results presented in Chapter 3 confirm the success of our implementation. Recent simulations have demonstrated the ability of our control system to position the motor accurately and maintain stable speed control.

This project paves the way for various applications in the fields of automation and robotics. Precise control of motor position and speed can be exploited in industrial automation, robotic arm control, numerically controlled machines, etc. The control system developed can be enhanced and adapted to different motor configurations and applications.

In conclusion, this project has provided a complete solution for controlling the position and speed of a DC motor with an encoder using the L298 driver and Arduino.

This project opens up several promising avenues for future research and development in the field of DC motor position and speed control with encoders using the L298 driver and Arduino. Here are some potential avenues to explore:

- Advanced control techniques: Further research can be carried out into advanced control techniques such as adaptive control, neural networks or machine learning algorithms to

improve system performance, robustness and adaptability to different operating conditions.

- Sensor fusion integration: Extending the integration of sensor fusion techniques by combining data from multiple sensors such as encoders, accelerometers and gyroscopes can improve motion estimation, position accuracy and overall system reliability

The results obtained, combined with the improvements proposed above, pave the way for future advances in motor control and automation. This work contributes to the ever-evolving field of robotics, and lays a solid foundation for future research and development.

# Bibliographic references

# Bibliographic references

[1] : Luc Lasne, Notions de base et machine électrique, Dunod, Pairs, 2005

[2] : Gérard Guihéneuf, “Electric motors explained to electronics engineers (starting, speed variation, braking)”, Second expanded edition, 2012, Publitronic_Elektor International Media, Netherlands

[3] : M. Amira, "Design and realization of a manipulator arm controlled by the arduino2560," Master's thesis, M'hamedBougara University of Boumerdes, Faculty of Sciences ofthe engineer, Department of Mechanical Engineering, Modeling and Simulation in Mechanics,2016/2017.

[4] : B. B. and Eddie Smigiel, "Discover the constitution of servomotors," Strasbourg, 2019

[5] : Motor Control: Fundamentals and Applications" rry Ross, Muhammad Ali El.

[6] : FG. Jean-Pierre, FrédéricGenevey& Jean-Pierre Dulex, February, Arduino at school: Free educational resource edition, 201

[7] : Invention Story and History of Development of Arduino (circuitstoday.com)

[8] : Different Types of Arduino Boards Quick Comparisonon Specification & Features (circuitdigest.com)

[9] : https://www.afronicsblog.com/2021/12/connect-proteus-to-arduino.html

[10] : An introduction to PID control with DC motor | by Simon BDY | luos | Medium

[11] : https://www.informatique-et-electronique.fr/index.php/electronique-tutos/asservissement-de-position-d-un-moteur-a-courant-continu

# Annex

**Table showing the difference between the Arduino mega and the Arduino uno:**

In our study, we have used both Arduino Mega and Arduino Uno. Through this table that shows the difference between them, we notice that Arduino Mega is more durable than Arduino Uno.

| Feature | Arduino Uno | Arduino Mega |
|---|---|---|
| Microcontroller | ATmega328P | ATmega2560 |
| Operating Voltage | 5V | 5V |
| Input Voltage (recommended) | 7-12V | 7-12V |
| Input Voltage (limit) | 6-20V | 6-20V |
| Digital I/O Pins | 14 (of which 6 provide PWM output) | 54 (of which 15 provide PWM output) |
| Analog Input Pins | 6 | 16 |
| DC Current per I/O Pin | 20 mA | 40 mA |
| DC Current for 3.3V Pin | 50 mA | 50 mA |
| Flash Memory | 32 KB (ATmega328P) of which 0.5 KB used by boot loader | 256 KB (ATmega2560) of which 8 KB used by boot loader |
| SRAM | 2 KB (ATmega328P) | 8 KB (ATmega2560) |
| EEPROM | 1 KB (ATmega328P) | 4 KB (ATmega2560) |
| Clock Speed | 16 MHz | 16 MHz |

## Control code in motor position :

*************single motor position control ***************

```
// Motor direction constants
const int CW = 0;
const int CCW = 1;

// Motor control pins
const int motorDirPin = 5;
const int motorPWMPin = 6;
const int enablePin = 7;

// Encoder pins
const int encoderPinA = 2;
const int encoderPinB = 3;

// Motor and PID variables
volatile int encoderPos = 0;
float Kp = 8.1;
float Ki = 0.095;
float Kd = 0.095;
int targetPos;
float integral = 0.0;
float derivative = 0.0;
float prevError = 0.0;
float dt = 0.01; // 10 milliseconds
```

```
void doEncoderA() {
  if (digitalRead(encoderPinA) == digitalRead(encoderPinB)) {
    encoderPos--;
  } else {
    encoderPos++;
  }
}
void setup() {
  // Initialize encoder pins
  pinMode(encoderPinA, INPUT_PULLUP);
  pinMode(encoderPinB, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(encoderPinA), doEncoderA, CHANGE);

  // Initialize motor control pins
  pinMode(motorDirPin, OUTPUT);
  pinMode(motorPWMPin, OUTPUT);
  pinMode(enablePin, OUTPUT);
  digitalWrite(enablePin, HIGH);

  // Initialize serial communication for debugging purposes
  Serial.begin(9600);
}
void loop() {
  // Read target position from analog input
  targetPos = analogRead(A0);
```

```
// Calculate error
int error = targetPos - encoderPos;

// Calculation of PID terms
integral += error * dt;
derivative = (error - prevError) / dt;

// PID control signal calculation
float controlSignal = Kp * error + Ki * integral + Kd * derivative;

// Limit control signal in the range -255 to 255
int velocity = constrain(controlSignal, -255, 255);

// Set motor direction
if (velocity >= 0) {
  digitalWrite(motorDirPin, CW);
} else {
  digitalWrite(motorDirPin, CCW);
  velocity = -velocity;
}

// Apply PWM signal to control motor speed
analogWrite(motorPWMPin, velocity);

// Display debugging information
Serial.print("Target_Pos: ");
```

```
  Serial.print(targetPos);
  Serial.print(", encoderPos: ");
  Serial.println(encoderPos);

  // Update PID variables
  prevError = error;

  // Wait for time interval dt before next iteration
  delay(dt * 1000);
}
```

**********  speed controller for two motors **********

```
// Pin definitions for Motor A
#define ENCA_A 18 // Encoder A pin for Motor A
#define ENCB_A 19 // Encoder B pin for Motor A
#define PWM_A 6 // Output to control speed ENA of Motor A
#define IN1_A 7 // Output to control rotation direction IN1 of Motor A
#define IN2_A 5 // Output to control rotation direction IN2 of Motor A

// Pin definitions for Motor B
#define ENCA_B 20 // Encoder A pin for Motor B
#define ENCB_B 21 // Encoder B pin for Motor B
#define PWM_B 10 // Output to control speed ENB of Motor B
#define IN1_B 11 // Output to control rotation direction IN3 of Motor B
#define IN2_B 9 // Output to control rotation direction IN4 of Motor B
```

```
// Global variables
long previousTime; // Previous time in microseconds
float previousError_A; // Previous error on Motor A speed
float previousError_B; // Previous error on Motor B speed
int previousPos_A; // Previous position of Motor A
int previousPos_B; // Previous position of Motor B

// Volatile variables used in interrupts
volatile int pos_A; // Current position count for Motor A
volatile int pos_B; // Current position count for Motor B
float filteredSpeed_A; // Filtered speed for Motor A
float previousSpeed_A; // Previous filtered speed for Motor A
float filteredSpeed_B; // Filtered speed for Motor B
float previousSpeed_B; // Previous filtered speed for Motor B
float integral_A; // Integral term of PID for Motor A
float integral_B; // Integral term of PID for Motor B

void setup() {

  Serial.begin(9600);

  // Pin configuration
  pinMode(ENCA_A, INPUT);
  pinMode(ENCB_A, INPUT);
  pinMode(PWM_A, OUTPUT);
```

```
  pinMode(IN1_A, OUTPUT);
  pinMode(IN2_A, OUTPUT);

  pinMode(ENCA_B, INPUT);
  pinMode(ENCB_B, INPUT);
  pinMode(PWM_B, OUTPUT);
  pinMode(IN1_B, OUTPUT);
  pinMode(IN2_B, OUTPUT);

  // Attach interrupts on encoder channels A
  attachInterrupt(digitalPinToInterrupt(ENCA_A), readEncoder_A, RISING);
  attachInterrupt(digitalPinToInterrupt(ENCA_B), readEncoder_B, RISING);

}

void loop() {

  // Speed setpoint
  float targetSpeed = 125 * sin(previousTime / 1e6);

  // Read motor positions
  int pos_A;
  int pos_B;

  noInterrupts(); // Disable interrupts during read
  pos_A = pos_A;
```

```
pos_B = pos_B;
interrupts(); // Re-enable interrupts

// Calculate motor speeds
long currentTime = micros();
float deltaT = (currentTime - previousTime) / 1e6;
float speed_A = (pos_A - previousPos_A) / deltaT;
float speed_B = (pos_B - previousPos_B) / deltaT;

previousPos_A = pos_A;
previousPos_B = pos_B;
previousTime = currentTime;

// Convert count/s to RPM
float velocity_A = speed_A * 60 / 48;
float velocity_B = speed_B * 60 / 48;

// 25 Hz low-pass filtering
filteredSpeed_A = 0.854 * filteredSpeed_A + 0.0728 * velocity_A + 0.0728 * previousSpeed_A;
previousSpeed_A = velocity_A;

filteredSpeed_B = 0.854 * filteredSpeed_B + 0.0728 * velocity_B + 0.0728 * previousSpeed_B;
previousSpeed_B = velocity_B;

// Compute PID control
float kp = 22.50;
```

```
float kd = 4.75;
float ki = 36.75;

float error_A = targetSpeed - filteredSpeed_A;
float dError_A = (error_A - previousError_A) / deltaT;
integral_A += error_A * deltaT;
float control_A = kp * error_A + kd * dError_A + ki * integral_A;

float error_B = targetSpeed - filteredSpeed_B;
float dError_B = (error_B - previousError_B) / deltaT;
integral_B += error_B * deltaT;
float control_B = kp * error_B + kd * dError_B + ki * integral_B;

// Set direction and speed Motor A
int sens_A = 1;
if (commande_A < 0) sens_A = -1;
int puissance_A = abs(control_A);
if (puissance_A > 255) puissance_A = 255;
// Set direction and speed Motor B
 int sens_B = 1;
if (control_B < 0) sens_B = -1;

int puissance_B = abs(control_B);
if (puissance_B > 255) puissance_B = 255;
// Voltage Motor A
float tension_A = (float) puissance_A * sens_A * (12.0 / 255.0);
```

```
  // Voltage Motor B
  float tension_B = (float) puissance_B * sens_B * (12.0 / 255.0);

  // Store previous errors
  previousError_A = Error_A;
  previousError_B = Error_B;
  setMotor_A(sens_A, puissance_A, PWM_A, IN1_A, IN2_A);
  setMotor_B(sens_B, puissance_B, PWM_B, IN1_B, IN2_B);

  // Display
  Serial.print("Target speed:");
  Serial.print(TargetSpeed);
  Serial.print("Motor speed A:");
  Serial.print(speedFilter_A);
  Serial.print("Motor speed B:");
  Serial.println(speedFilter_B);
  Delay(1);

}

// Motor control functions
void setMotor_A(int sens, int puissance, int pwm, int in1, int in2) {

  analogWrite(pwm, power);

  if (sens == 1) {
```

```
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
  }
  else if (sens == -1) {
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
  }
  else {
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
  }

}
void setMotor_B(int sens, int puissance, int pwm, int in1, int in2) {

  analogWrite(pwm, puissance) ;

  if (sens == 1) {
    digitalWrite(in1, HIGH) ;
    digitalWrite(in2, LOW) ;
  }
  else if (sens == -1) {
    digitalWrite(in1, LOW) ;
    digitalWrite(in2, HIGH) ;
  }
  else {
```

```
    digitalWrite(in1, LOW) ;
    digitalWrite(in2, LOW) ;
  }

}
// Encoder interrupt service routines
void EncoderRead_A() {

  increment int ;

  if (digitalRead(ENCB_A))
    increment = 1 ;
  otherwise
    increment = -1 ;

  pos_A += increment ;

}
// Interrupt on encoder channel B
void EncoderRead_B() {

  int increment;

  if (digitalRead(ENCB_B))
    increment = 1;
  else
```

```
    increment = -1;

  pos_B += increment;

}

// End of program
```