A Servo Controller for Brushed DC Motor

by

BASHIR TAWFIG BASHIR ABUGHARSA

1181102921

Session 2023/2024

The project report is prepared for

Faculty of Engineering

Multimedia University

in partial fulfilment for

Bachelor of Engineering (Hons) Electronics

majoring in Telecommunications

FACULTY OF ENGINEERING

MULTIMEDIA UNIVERSITY

April 2016

DECLARATION

I hereby declare that this work has been done by myself and no portion of the work contained in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

I also declare that pursuant to the provisions of the Copyright Act 1987, I have not engaged in any unauthorised act of copying or reproducing or attempt to copy / reproduce or cause to copy / reproduce or permit the copying / reproducing or the sharing and / or downloading of any copyrighted material or an attempt to do so whether by use of the University's facilities or outside networks / facilities whether in hard copy or soft copy format, of any material protected under the provisions of sections 3 and 7 of the Act whether for payment or otherwise save as specifically provided for therein. This shall include but not be limited to any lecture notes, course packs, thesis, text books, exam questions, any works of authorship fixed in any tangible medium of expression whether provided by the University or otherwise.

I hereby further declare that in the event of any infringement of the provisions of the Act whether knowingly or unknowingly the University shall not be liable for the same in any manner whatsoever and undertakes to indemnify and keep indemnified the University against all such claims and actions.

Signature: [illegible]

Name: BASHIR TAWFIG BASHIR ABUGHARSA

Student ID: 1181102921

Date: 11-06-2024

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my supervisor, Dr. Lo Yew Chiong, for his invaluable guidance, support, and encouragement throughout the duration of this project. His expertise in Robotics and Embedded Systems, as well as IoT, has been instrumental in shaping the direction and success of this research. His insightful advice and continuous motivation have been crucial in overcoming the challenges faced during the project.

I am also profoundly grateful to my moderator, Dr. Cham Chin Leei, for his insightful feedback and constructive criticism. His extensive knowledge in Control and Identification and Microcontroller Applications has provided a solid foundation for the evaluation and enhancement of this project. Dr. Cham's detailed and thoughtful comments have significantly improved the quality and depth of my work.

Special thanks to the Faculty of Engineering at Multimedia University for providing the necessary resources and support. The facilities and academic environment have been conducive to conducting my research effectively. I also appreciate the assistance provided by the technical staff, whose expertise ensured the smooth operation of the equipment used in this project.

I would like to extend my appreciation to my peers and colleagues who have provided me with valuable discussions, suggestions, and moral support throughout this journey. Their camaraderie and collaboration have made this experience enjoyable and fulfilling.

Lastly, I would like to thank my family and friends for their unwavering support and encouragement throughout my academic journey. Their belief in me has been a constant source of inspiration and strength, enabling me to persevere and achieve my goals.

ABSTRACT

This project focuses on the design and implementation of a servo controller for a Brushed DC motor, aimed at achieving precise position control. The motor driver circuit, developed using an H-bridge configuration, is controlled by PWM signals generated from an STM32 microcontroller. A PID controller is integrated to ensure accurate positioning, utilizing feedback from an optical encoder and following a user-defined trapezoidal motion profile, including parameters such as position, speed, acceleration, and deceleration.

The motor driver circuit was evaluated for its ability to maintain consistent PWM signals and reliable gate drive voltages, demonstrating smooth and efficient operation at different duty cycles. The PID controller was tested under various load conditions, showing effectiveness in closely following the theoretical motion profile with minimal errors. Key findings indicate that the designed servo controller meets the precision requirements for Brushed DC motor control, offering reliable and accurate performance.

The project results show that the motor driver circuit and PID controller work effectively together to provide smooth and accurate motor control. However, some limitations were identified, such as the inability to dynamically adjust the motion profile during operation and challenges in maintaining smooth motion at very low speeds. Future research could focus on auto-tuning methods, developing dynamic motion profile adjustments, and improving low-speed control to further enhance the system's performance and adaptability. The findings highlight the system's robustness and suitability for high-precision applications.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1
1.1 Overview 1
1.2 Problem Statements 2
1.3 Project Scope 3
1.4 Report Outline 3

CHAPTER 2 LITERATURE REVIEW 5
2.1 Brushed DC Motor Driver Circuit Design 6
2.1.1 Efficiency and Reliability 9
2.2 PID Control in DC Motor Position Systems 10

CHAPTER 3 DETAILS OF THE DESIGN 13
3.1 Main Components: 13
3.1.1 IRF3205 N-channel MOSFET: 14
3.1.2 IR2110 MOSFET Gate Driver: 15
3.1.3 LM7812 12V Voltage Regulator 16
3.1.4 LM7805 5V Voltage Regulator 17
3.1.5 1N5819 Schottky Diode 18
3.1.6 STM32f103 Microcontroller 19
3.1.7 12V 5A AC/DC Adapter 20
3.1.8 MD36NP2724V Brushed DC Motor 20
3.2 Hardware Tools 21

3.2.1 Hantek2000 Oscilloscope / Multimeter ... 21

3.2.2 ST-LINK/V2 Debugger and Programmer ... 22

3.2.3 USB to TTL Module (CP2102) ... 23

3.3 Software Tools ... 24

3.3.1 STM32CubeIDE ... 24

3.3.2 Arduino IDE 2 ... 25

3.3.3 Autodesk Fusion 360 ... 26

3.3.4 OpenOCD ... 27

3.4 Brushed DC Motor Driver Circuit Design ... 27

3.4.1 Driver Circuit Schematic Design ... 29

3.4.2 High Side MOSFET Bootstrap Circuit ... 32

3.4.3 MOSFET Gate Discharge Diode ... 34

3.5 PID Controller ... 35

3.5.1 PID Controller Implementation ... 35

3.6 Motion Profile ... 39

3.7 Brushed DC Motor Optical Encoder ... 42

3.8 User Interface via UART ... 45

3.9 System Flowchart ... 47

CHAPTER 4 DATA PRESENTATION AND DISCUSSION OF FINDINGS ... 49

4.1 Brushed DC Motor Driver Circuit ... 50

4.1.1 Driver Circuit at PWM 40% Duty Cycle ... 50

4.1.2 Driver Circuit at PWM 80% Duty Cycle 55

4.2 PID Controller 59

4.2.1 PID Controller Performance under No Load 59

4.2.2 PID Controller Performance under Constant Load 61

4.2.3 PID Controller Performance under Varying Load 63

4.3 Discussion of Findings 64

4.3.1 Brushed DC Motor Driver Circuit 64

4.3.2 PID Controller Performance 65

4.3.3 Limitations 65

CHAPTER 5 CONCLUSIONS 67

5.1 Summary and Conclusions 67

5.2 Areas of Future Research 68

5.2.1 Auto-Tuning Methods: 68

5.2.2 Dynamic Adjustment of Motion Profile: 68

5.2.3 Enhanced Low-Speed Control: 68

5.2.4 PCB Design for Motor Driver: 69

LIST OF FIGURES

Figure 2.1 H-Bridge DC Motor Control Circuit [7] 7
Figure 2.2 Motor Driver L293D [8] 8
Figure 2.3 Bi-directional rotation using a full bridge [9] 8
Figure 2.4 H- Bridge motor driver circuit using BJTs [10] 9
Figure 2.5 Block diagram for the DC motor model [11] 10
Figure 2.6 Block diagram for DC motor position control [12] 11
Figure 3.1 IRF3205 N-channel MOSFET [15] 14
Figure 3.2 IR2110 MOSFET gate driver [16] 15
Figure 3.3 LM7812 12V voltage regulator IC [17] 16
Figure 3.4 LM7805 5V voltage regulator [18] 17
Figure 3.5 1N5819 Schottky Diode [19] 18
Figure 3.6 STM32f103 microcontroller [21] 19
Figure 3.7 12V 5A AC/DC Adapter 20
Figure 3.8 MD36NP2724V brushed DC motor [22] 21
Figure 3.9 Hantek2000 Handheld Oscilloscope [23] 22
Figure 3.10 ST-LINK/V2 in-circuit debugger/programmer [24] 23
Figure 3.11 3.2.3 USB to TTL Module (CP2102) [25] 24
Figure 3.12 STM32CubeIDE user interface 25
Figure 3.13 Arduino IDE 2 user interface 26
Figure 3.14 Autodesk Fusion 360 user interface 27
Figure 3.15 Basic H-bridge circuit [30] 28
Figure 3.16 Brushed DC motor driver circuit schematic design 29
Figure 3.17 Motor direction flowchart 30
Figure 3.18 Bootstrap diodes and capacitors 32
Figure 3.19 MOSFET Gate Discharge Diode 34
Figure 3.20 PID controller block diagram [33] 36
Figure 3.21 Control output $\boldsymbol{ut}$ flowchart 38
Figure 3.22 Trapezoidal motion profiles 39
Figure 3.23 Motor optical encoder flowchart 43
Figure 3.24 Encoder output A and B [39] 44

Figure 3.25 Arduino IDE serial monitor 45
Figure 3.26 User input flowchart 46
Figure 3.27 System flowchart 47
Figure 4.1 Brushed DC motor driver 49
Figure 4.2 Clockwise 40% Duty cycle PWM signal 50
Figure 4.3 Counter-Clockwise 40% Duty cycle PWM signal 51
Figure 4.4 Clockwise 40% duty cycle high side MOSFET gate voltage 51
Figure 4.5 Counter-Clockwise 40% duty cycle high side MOSFET gate voltage 52
Figure 4.6 Clockwise 40% duty cycle low side MOSFET gate to source voltage 52
Figure 4.7 Counter-Clockwise 40% duty cycle low side MOSFET gate to source voltage 53
Figure 4.8 Motor clockwise 40% duty cycle 53
Figure 4.9 Motor Counter-Clockwise 40% duty cycle 54
Figure 4.10 Clockwise 80% duty cycle PWM signal 55
Figure 4.11 Counter-Clockwise 80% duty cycle PWM signal 55
Figure 4.12 Clockwise 80% duty cycle high side MOSFET gate to source voltage 56
Figure 4.13 Counter-Clockwise 80% duty cycle high side MOSFET gate to source voltage 56
Figure 4.14 Clockwise 80% duty cycle low side MOSFET gate to source voltage 57
Figure 4.15 Counter-Clockwise 80% duty cycle low side MOSFET gate to source voltage 57
Figure 4.16 Motor clockwise 80% duty cycle 58
Figure 4.17 Motor Counter-Clockwise 80% duty cycle 58
Figure 4.18 Motor speed graph 0 degrees to 1200 degrees under no load 59
Figure 4.19 Serial monitor motor position 60
Figure 4.20 Motor speed graph 0 degrees to 120 degrees under no load 60
Figure 4.21 Serial monitor motor position 61
Figure 4.22 Motor speed graph 0 degrees to -3000 degrees under contant load 62
Figure 4.23 Serial monitor motor position 62
Figure 4.24 Motor speed graph 0 degrees to 4000 degrees under varying load 63
Figure 4.25 Serial monitor motor position 63

LIST OF TABLES

Table 3.1 IRF3205 N-channel MOSFET specifications [15] 14

Table 3.2 IR2110 MOSFET gate driver specifications [16] 15

Table 3.3 LM7812 12V voltage regulator specification [17] 16

Table 3.4 LM7805 5V voltage regulator specification [18] 18

Table 3.5 1N5819 Schottky diode specification [19] 19

LIST OF ABBREVIATIONS

FYP	Final Year Project
DC	Direct Current
AC	Alternating Current
PID	Proportional-Integral-Derivative
PWM	Pulse Width Modulation
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
BJT	Bipolar Junction Transistor
OpenOCD	Open On-Chip Debugger
UART	Universal Asynchronous Receiver Transmitter
HAL	Hardware Abstraction Layer
GPIO	General Purpose Input Output
IC	Integrated Circuit
VCC	Voltage Common Collector

CHAPTER 1 INTRODUCTION

1.1 Overview

The advent of electric motors has been pivotal in the evolution of various mechanical systems, and among the spectrum of motors utilized, the Brushed DC motor is known for its straightforward architecture and control [1]. This type of motor is ubiquitous across multiple sectors due to its operational simplicity and cost-effectiveness, making it a preferred choice for mass-produced goods. Brushed DC motors are characterized by their direct compatibility with DC power sources, a feature that has solidified their position in applications where easy power access is a prerequisite. However, with the advent of more sophisticated technological demands, there is a pressing need for precision in motor operations. Precision, a non-negotiable quality in contemporary applications such as automated precision machining, unmanned aerial vehicles, and sophisticated navigational systems, requires an advanced degree of control that surpasses the capabilities of conventional open-loop controllers [2].

Considering these requirements, the domain of servo controllers has gained popularity, offering the potential for refined control and enhanced operational efficiency of Brushed DC motors. These controllers employ feedback mechanisms, principally through encoders, to furnish a continuous stream of data regarding motor position and velocity, facilitating an immediate corrective response via closed-loop control systems. The implementation of such feedback loops is fundamental to the servo control methodology, enabling the system to counteract any deviations from predefined motor performance criteria. Nonetheless, the task of engineering a servo controller that is both precise and efficient is fraught with challenges [3]. It necessitates meticulous signal processing, effective power management, and a resilient design that can withstand the exigencies of operation. As the application spectrum of Brushed DC motors broadens to more demanding tasks, the controller technology must concurrently advance, incorporating sophisticated control algorithms. This project aims to forge a controller that focuses on performance, costefficiency, and minimal complexity, thereby extending the functional envelope of Brushed DC motors.

1.2 Problem Statements

The inherent mechanical properties of Brushed DC motors limit their capacity for precision control when relying solely on conventional driver circuits. These standard circuits lack the sophistication to finely tune the motor’s speed and positioning, which is a critical deficiency for applications that necessitate exact movements and strict adherence to motion profiles. The driver circuit alone is not equipped to account for the dynamic variables that impact motor performance, such as external loads and power supply irregularities. To achieve the high level of precision required in advanced technological applications, it is essential to go beyond the basic control that driver circuits offer.

The necessity for precise motor control becomes evident in applications like automated precision machining, unmanned aerial vehicles, and sophisticated navigational systems, where even minor deviations can lead to significant errors or failures. Conventional open-loop control systems are insufficient in these scenarios as they cannot provide real-time feedback and correction, resulting in inaccuracies and inefficiencies.

Moreover, as the complexity and demand for higher performance in technological systems increase, there is a pressing need for a robust control solution that can dynamically adjust to varying conditions and maintain optimal motor performance. This challenge is further compounded by the need for a cost-effective and minimally complex solution that can be easily implemented and maintained.

Thus, the primary problem addressed by this project is the development of a servo controller for Brushed DC motors that can provide precise and efficient control. This involves designing a system that integrates feedback mechanisms and advanced control algorithms to ensure accurate motor operation despite external disturbances and variations in operating conditions. By addressing these challenges, the project aims to enhance the functionality and applicability of Brushed DC motors in high-precision and demanding environments.

1.3 Project Scope

The objective of this project is to engineer a servo controller tailored for Brushed DC motors, with a focus on significantly enhancing their precision in terms of positioning control. A critical part of the project involves designing and building a driver circuit. This circuit will be controlled by a Pulse Width Modulation (PWM) signal originating from a microcontroller [4]. The driver circuit's role is pivotal as it acts as the primary mechanism for controlling the motor.

The project will incorporate a Proportional-Integral-Derivative (PID) closed-loop control system [5]. This system will be integrated seamlessly with the driver circuit and microcontroller, establishing the PID controller as the core unit responsible for continuously monitoring and fine-tuning the motor's output. An optical encoder will be used in the feedback loop to provide real-time data on the motor's position and velocity. Through this dynamic regulation, the system will be able to maintain strict adherence to the predefined motion profiles, effectively handling external influences such as variations in load. This comprehensive approach aims to elevate the performance of Brushed DC motors to meet the demanding precision standards of modern applications, ensuring they operate efficiently and accurately under a wide array of conditions.

1.4 Report Outline

This report is organized into five chapters, each detailing different aspects of the design, implementation, and analysis of the servo controller for a Brushed DC motor.

The first chapter, Introduction, provides an overview of the project, including its objectives, significance, and scope. It introduces the key concepts and the motivation behind developing a servo controller for Brushed DC motors.

The second chapter, Literature Review, presents a comprehensive review of existing literature related to Brushed DC motor driver circuit design and PID controller design. It compares the methodologies and findings from various studies, highlighting their relevance and application to the current project.

The third chapter, Details of the Design, outlines the methodology and design of the servo controller. It covers the initial design of the motor driver circuit using an H-bridge configuration, the integration of the PID control system, and the practical implementation. The chapter also describes the components used, their configuration, and the overall system architecture.

The fourth chapter, Data Presentation and Discussion of Findings, presents the collected data from the implementation and testing of the motor control system. It discusses the performance metrics of the motor driver circuit and PID controller, analysing their effectiveness in achieving precise motor control. The chapter also highlights any limitations and provides a detailed discussion of the findings.

The final chapter, Summary and Conclusions, summarizes the project, discussing the key findings and their implications. It also outlines the areas for future research, suggesting potential improvements and further development to enhance the system's performance and adaptability.

CHAPTER 2 LITERATURE REVIEW

This chapter provides a comprehensive overview of the existing literature related to the development and implementation of servo controllers for Brushed DC motors. It outlines the various works that have been conducted in this area, emphasizing different methodologies used to enhance motor control precision and efficiency. More specifically, it highlights the approaches that integrate advanced control techniques, including Pulse Width Modulation (PWM) and Proportional-Integral-Derivative (PID) controllers, in motor control systems.

Particular attention is given to the design and application of H-bridge circuits and their role in controlling the speed and direction of Brushed DC motors. The review also delves into the integration of feedback mechanisms, such as optical encoders, which provide real-time data essential for closed-loop control systems. These feedback systems are crucial for maintaining strict adherence to predefined motion profiles, ensuring accurate and efficient motor performance.

This chapter critically evaluates the contributions of various methodologies, comparing their strengths and weaknesses in relation to the overall research context. The investigated research domains include real-time control techniques and the implementation of microcontrollers, specifically STM32, in embedded motor control applications. The adaptation of these advanced control systems to handle external influences, such as load variations, is also discussed.

The findings of this review will offer an in-depth analysis of the current state of research in the field of Brushed DC motor control. Additionally, it will highlight novel strategies and advancements in control algorithms and feedback mechanisms that contribute to the enhanced precision and efficiency of servo controllers for Brushed DC motors.

2.1 Brushed DC Motor Driver Circuit Design

The design of driver circuits for Brushed DC motors is crucial in achieving precise control of motor speed and direction. Various methods and components are utilized to enhance the performance and efficiency of these driver circuits. This section reviews the key methodologies and components, focusing on the use of Pulse Width Modulation (PWM) and H-bridge circuits.

The H-bridge configuration is a fundamental circuit for driving DC motors, allowing them to operate in both forward and reverse directions. An H-bridge consists of four switching elements, typically transistors or MOSFETs, arranged in a configuration that can control the direction of the current flow through the motor, thus controlling its rotation direction [6].

Research has shown that designing an H-bridge DC motor driver using a microcontroller that generates high-frequency PWM signals can effectively control motor speed and direction. One study [7] implemented an H-bridge driver circuit utilizing NPN and PNP MOSFETs driven by TLP250 MOSFET drivers. The use of PWM signals allowed for efficient control of motor speed by adjusting the duty cycle, which directly influenced the motor terminal voltage.

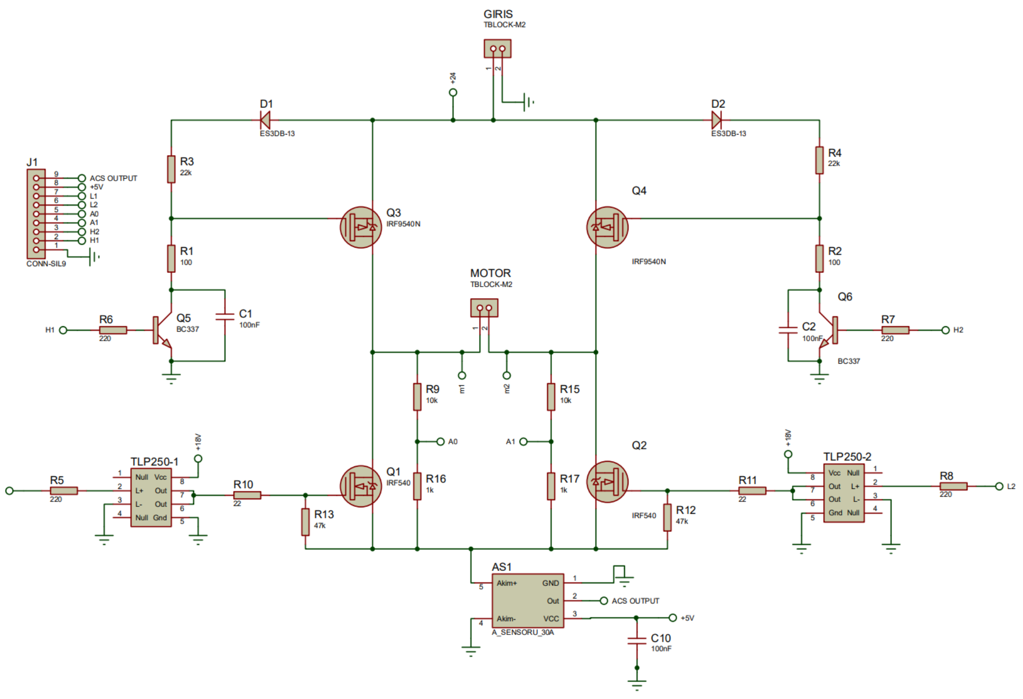


Figure 2.1 H-Bridge DC Motor Control Circuit [7]

The design included a current sensor to monitor the motor current, protecting the motor from high current conditions such as short circuits or overloading. This integration ensured the longevity and reliability of the motor and the driver circuit [7].

Another research [8] explores the use of Pulse Width Modulation (PWM) for controlling DC motor speed, employing an AT89S52 microcontroller and L293D IC. This method excels in providing precise control over small DC motors in a cost-effective manner. However, the reliance on L293D IC limits its applicability to small motors, posing a challenge for more complex operational contexts.

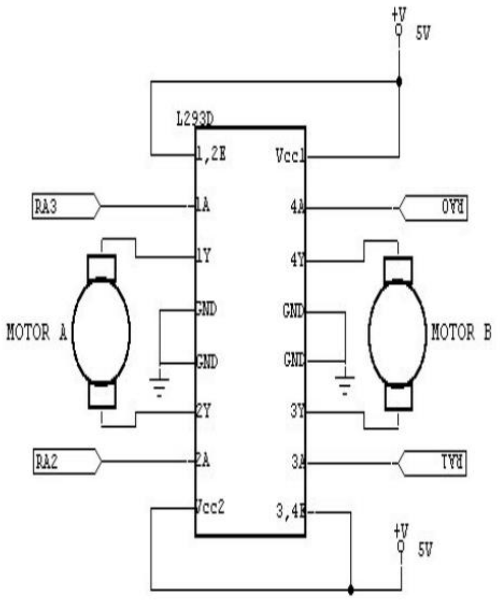


Figure 2.2 Motor Driver L293D [8]

Other research [9] implements a PWM-based motor control circuit using an LM324 operational amplifier and four MOSFET to form the H-Bridge circuit. This design is lauded for its efficiency and suitability for small scale applications. Nonetheless, the LM324's limitations in bandwidth and response accuracy may hinder performance in high-speed applications.

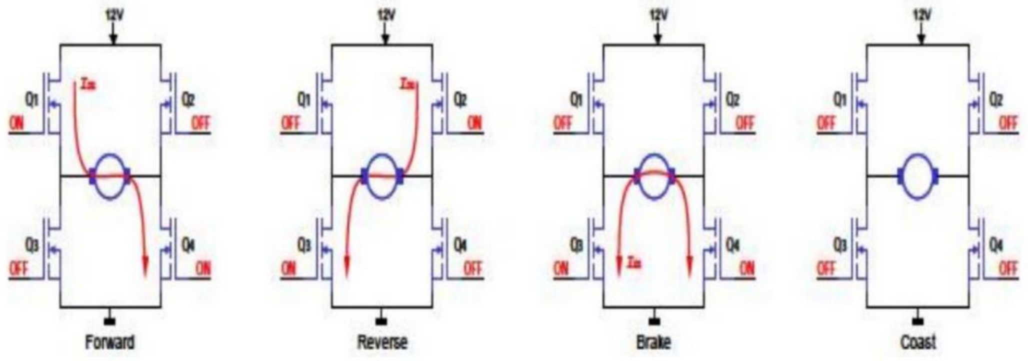


Figure 2.3 Bi-directional rotation using a full bridge [9]

Another research [10] discussed the design of wheeled mobile robots that use H-bridge motor driver circuit, stands out for its detailed analysis of circuit designs under varying load conditions. The use of BJTs in this design provides a cost-effective solution, but it falls short in terms of efficiency and power management compared to MOSFETs, which could affect its performance in more demanding robotic applications.

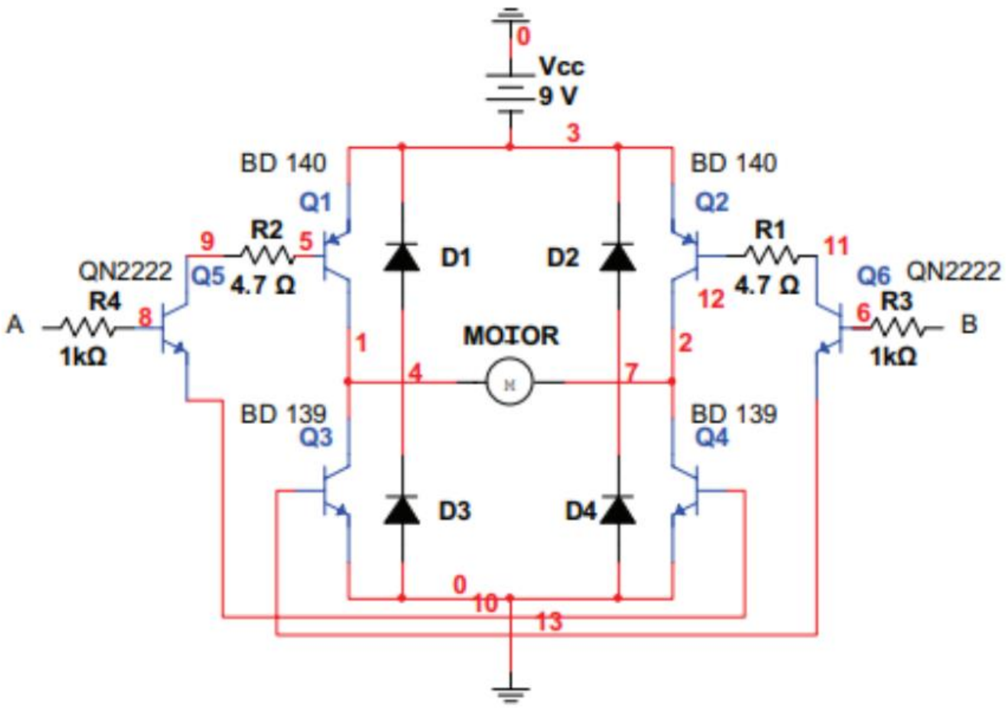


Figure 2.4 H- Bridge motor driver circuit using BJTs [10]

2.1.1 Efficiency and Reliability

The efficiency of an H-bridge driver circuit is significantly enhanced by using PWM control. Traditional methods, such as using a variable resistor, result in considerable power loss due to heat dissipation. PWM, on the other hand, minimizes these losses by rapidly switching the supply voltage, reducing the time the transistors spend in the transition states where power loss is highest.

In their implementation, researchers emphasized the cost-effectiveness and reliability of their design, noting that it allowed for precise speed control without the need for expensive components. The use of microcontrollers for generating PWM signals further enhanced flexibility and control accuracy, making the design suitable for various industrial applications.

2.2 PID Control in DC Motor Position Systems

The proportional-integral-derivative (PID) controller is a widely used control algorithm in industrial applications due to its simplicity and effectiveness in handling a variety of control problems [10]. For brushed DC motors, PID controllers are particularly valuable in achieving precise speed and position control, which is critical in applications requiring exact movements and strict adherence to motion profiles.

In one study [11], a PID controller was applied to control the angular position of a DC motor connected to a valve of a hydraulic pump. The system was implemented using an ATmega16 microcontroller, and the PID parameters were carefully tuned to optimize performance. The experimental results demonstrated that the PID controller effectively suppressed oscillations caused by system and sensor nonlinearities, maintaining precise control of the motor's position.

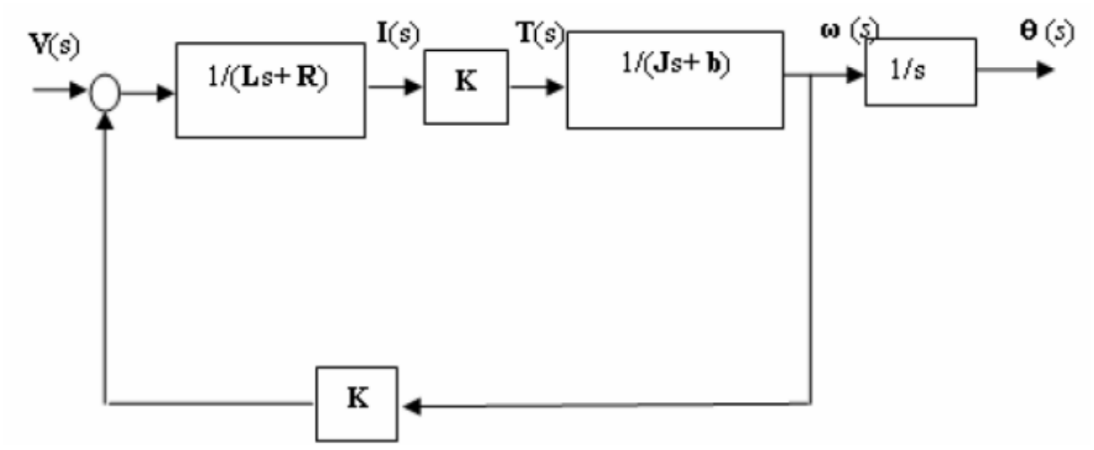


Figure 2.5 Block diagram for the DC motor model [11]

Another research [12] focused on controlling the angular position of a DC geared motor using a PID controller with friction compensation. The study utilized an Arduino microcontroller and an L298N dual H-bridge motor driver to execute PWM signals. The PID controller successfully minimized errors and oscillations, achieving a high level of precision in the motor's position control. The inclusion of friction compensation further improved the system's performance, making it suitable for applications like robotic arm position control.

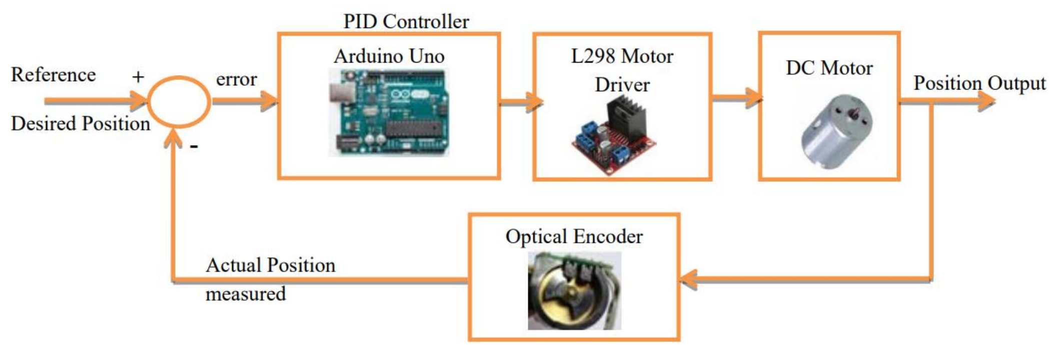


Figure 2.6 Block diagram for DC motor position control [12]

Implementing a PID controller for DC motors involves several challenges, including tuning the PID parameters and handling system nonlinearities. The process of tuning involves setting the proportional gain (Kp), integral gain (Ki), and derivative gain (Kd) to achieve the desired control performance. Incorrect tuning can lead to issues like overshoot, oscillations, and steady-state errors.

A study [13] on DC motor speed control using PWM highlighted the importance of precise tuning of PID parameters. The researchers used an Arduino microcontroller to generate PWM signals and control the motor speed. They emphasized the need for iterative tuning and testing to find the optimal PID settings that balance responsiveness and stability.

Another study [14] investigated the application of PID controllers for DC motor angular position control, highlighting the advantages of PID controllers in terms of stability and precision. The study utilized a genetic algorithm for tuning the PID parameters, achieving superior performance compared to the Ziegler-Nichols method. This approach provided a more refined tuning process, reducing the error between the actual and desired responses, and enhancing the system's robustness against disturbances

The primary advantage of PID controllers lies in their ability to provide stable and accurate control with relatively simple implementation. They are capable of handling a wide range of operating conditions and can be easily adjusted to meet specific performance requirements. The use of PID controllers in brushed DC motor systems enhances their applicability in high-precision environments, such as automated machinery, robotics, and navigation systems.

CHAPTER 3 DETAILS OF THE DESIGN

This chapter provides a comprehensive overview of the methodology employed in the creation of a servo controller for a Brushed DC motor. It covers the entire design process, from the initial development of the H-bridge driver circuit to the integration of the PID control system. Emphasis is placed on both the practical implementation and the theoretical foundations necessary to achieve precise motor control.

The design and construction of the H-bridge driver circuit are detailed with schematics and component descriptions, addressing the selection of components and the challenges encountered. The integration of the PID control system is then explored, highlighting the principles of Proportional-Integral-Derivative control and the tuning methods used to optimize performance. Additionally, the chapter discusses the feedback mechanism involving an optical encoder, which provides real-time data on the motor's position and velocity, crucial for maintaining accurate control and responding to changes in load and other external factors. By detailing each step from concept to implementation, this chapter aims to provide a clear understanding of the design and execution involved in developing a robust servo control system for Brushed DC motors.

3.1 Main Components:

In designing the Brushed DC motor driver circuit, several key components are essential for achieving precise control over the motor's speed and direction. This section provides an overview of the main components used in the circuit. By understanding the function and role of each component, we can better appreciate how they are integrated to create an efficient and reliable motor control system. The primary components discussed include the H-bridge circuit, MOSFETs, the microcontroller, and the PWM signals.

3.1.1 IRF3205 N-channel MOSFET:

This power MOSFET is known for its high current (110A) and voltage (55V) handling capabilities, making it suitable for heavy-duty applications. Its low on-resistance and fast switching speed are advantageous for efficient power management in motor control circuits [15].

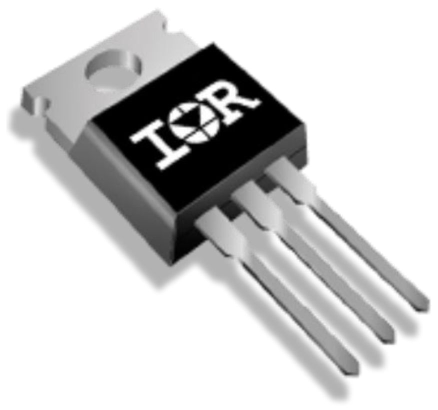


Figure 3.1 IRF3205 N-channel MOSFET [15]

Table 3.1 IRF3205 N-channel MOSFET specifications [15]

Specification	**IRF3205**
ID (@25°C) max	110 A
Mounting	THT
Operating Temperature min max	-55 °C / 175 °C
Ptot max	150 W
Package	TO-220
Polarity	N
QG (typ @10V)	97.3 nC
Qgd	36 nC
RDS (on) (@10V) max	8 mΩ
RthJC max	1 K/W
Tj max	175 °C
VDS max	55 V
VGS(th) min max	3 V / 2 V / 4 V
VGS max	20 V

3.1.2 IR2110 MOSFET Gate Driver:

The IR2110 is a high voltage (up to 500V), high-speed driver specifically designed for MOSFETs and IGBTs. It features independent high and low side referenced output channels, crucial for precise and rapid switching in H-bridge configurations [16].

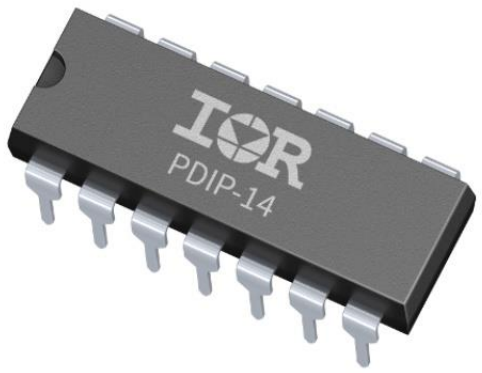


Figure 3.2 IR2110 MOSFET gate driver [16]

Table 3.2 IR2110 MOSFET gate driver specifications [16]

Specification	IR2110
Channels	2
Configuration	High-side and low-side
Input Vcc min max	10 V / 20 V
Isolation Type	Functional levelshift
Output Current (Source)	2.5 A
Output Current (Sink)	2.5 A
Qualification	Industrial
Turn Off Propagation Delay	94 ns
Turn On Propagation Delay	120 ns
VBS UVLO (On)	8.6 V
VBS UVLO (Off)	8.2 V
VCC UVLO (On)	8.5 V
VCC UVLO (Off)	8.2 V
Voltage Class	500 V

3.1.3 LM7812 12V Voltage Regulator

The LM7812 is a three-terminal positive voltage regulator IC that provides a stable 12V output from a higher voltage input, typically ranging from 14.5V to 35V. It is widely used in electronic circuits to ensure a consistent voltage supply, protecting components from voltage fluctuations. The LM7812 is capable of delivering up to 1.5A of output current and features internal thermal overload protection, short-circuit protection, and safe area protection. This makes it a reliable choice for a variety of applications requiring a regulated 12V power supply [17].

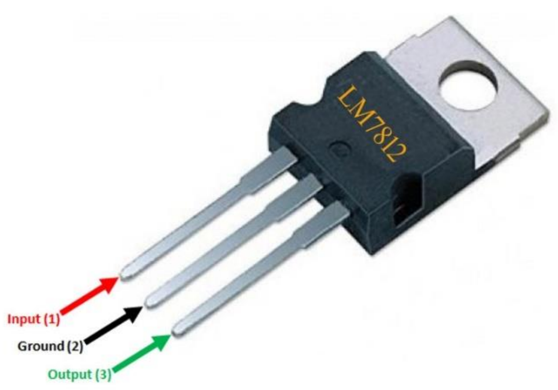


Figure 3.3 LM7812 12V voltage regulator IC [17]

Table 3.3 LM7812 12V voltage regulator specification [17]

Specification	**LM7812**
Output Voltage	12V
Input Voltage Range	14.5V to 35V
Output Current (max)	1.5A
Dropout Voltage (typ)	2V
Line Regulation	3mV
Load Regulation	15mV

Quiescent Current (typ)	8mA
Operating Temperature Range	0°C to 125°C
Thermal Overload Protection	Yes
Short-Circuit Protection	Yes
Package Types	TO-220, TO-3, D2PAK

3.1.4 LM7805 5V Voltage Regulator

The LM7805 is a three-terminal positive voltage regulator IC that provides a fixed 5V output from a higher voltage input, typically ranging from 7V to 35V. It is commonly used in a variety of electronic devices to ensure a stable and consistent 5V power supply. The LM7805 can deliver up to 1.5A of output current and includes features such as internal thermal overload protection, short-circuit protection, and safe area protection. These features make the LM7805 a reliable and widely used component for powering low-voltage digital circuits and other applications requiring a regulated 5V supply [18].

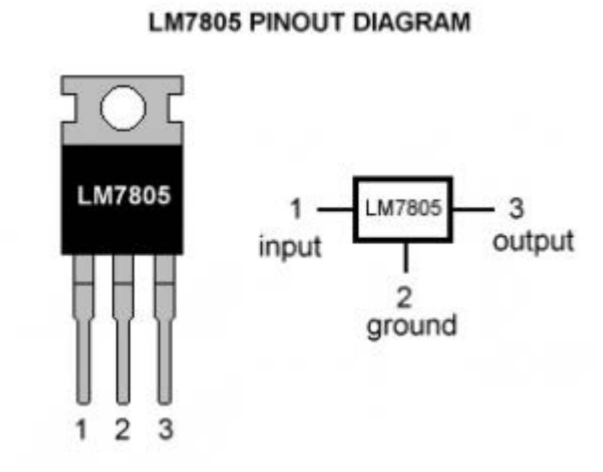


Figure 3.4 LM7805 5V voltage regulator [18]

Table 3.4 LM7805 5V voltage regulator specification [18]

Specification	LM7805
Output Voltage	5V
Input Voltage Range	7V to 35V
Output Current (max)	1.5A
Dropout Voltage (typ)	2V
Line Regulation	3mV
Load Regulation	15mV
Quiescent Current (typ)	8mA
Operating Temperature Range	0°C to 125°C
Thermal Overload Protection	Yes
Short-Circuit Protection	Yes
Package Types	TO-220, TO-3, D2PAK

3.1.5 1N5819 Schottky Diode

The 1N5819 is a Schottky diode known for its low forward voltage drop and fast switching speed. It is widely used in power supply circuits and other applications where efficiency and performance are crucial. The 1N5819 can handle a maximum reverse voltage of 40V and a forward current of up to 1A. Its low forward voltage drop, typically around 0.2V to 0.45V, makes it ideal for use in low-voltage, high-efficiency applications. Additionally, the fast recovery time of the 1N5819 enhances its performance in high-speed switching applications [19].

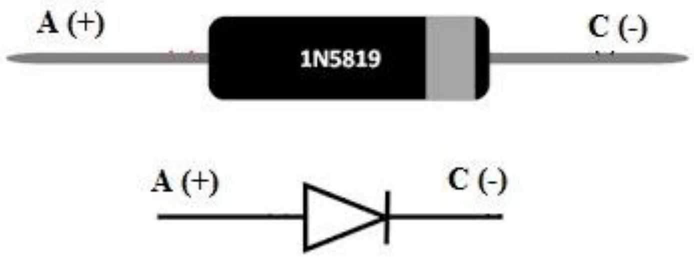


Figure 3.5 1N5819 Schottky Diode [19]

Table 3.5 1N5819 Schottky diode specification [19]

Specification	**1N5819**
Maximum Reverse Voltage (VR)	40V
Forward Current (IF)	1A
Forward Voltage Drop (VF)	0.2V to 0.45V
Reverse Current (IR)	1mA at 40V
Operating Temperature Range	-65°C to 125°C
Package Type	DO-41

3.1.6 STM32f103 Microcontroller

This microcontroller board offers robust features like a 72 MHz ARM Cortex-M3 processor, 20KB SRAM, and 64KB Flash memory. It's capable of complex tasks like generating accurate PWM signals, essential for intricate motor control applications like speed and directional adjustments [20].

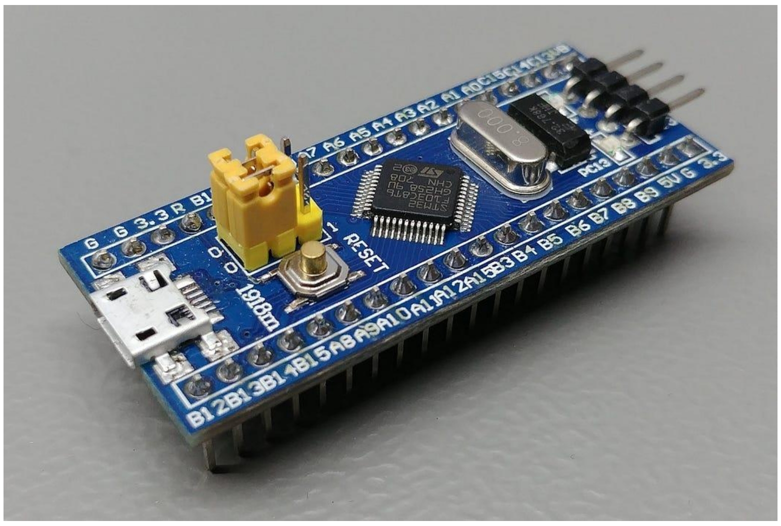


Figure 3.6 STM32f103 microcontroller [21]

3.1.7 12V 5A AC/DC Adapter

The AC to DC Power Adapter converts an input of 100-240V AC to a stable 12V DC output, capable of delivering up to 5A of current. This adapter is well-suited for powering motor driver circuits, providing the necessary DC voltage to operate components such as the H-bridge and control electronics. The adapter's standard DC barrel connector ensures compatibility with common motor driver setups. Additionally, it includes built-in safety features such as overvoltage, overcurrent, and short-circuit protection, ensuring reliable and safe operation of the motor driver circuit. Its compact design facilitates easy integration into various setups, making it a practical choice for powering Brushed DC motor control systems.

Figure 3.7 12V 5A AC/DC Adapter

3.1.8 MD36NP2724V Brushed DC Motor

This Brushed DC Motor, available on AliExpress, operates at 24V and is equipped with an optical encoder that provides 500 pulses per revolution. The motor features a gear ratio of 3249:121, allowing for precise control of its speed and position. The inclusion of the optical encoder enhances its performance in applications requiring accurate feedback and fine-tuned control. This makes it ideal for integration into systems where precise motor control is essential, such as in servo mechanisms and automated machinery [22].

Figure 3.8 MD36NP2724V brushed DC motor [22]

3.2 Hardware Tools

This section outlines the various hardware tools utilized in the design and implementation of the servo controller for the Brushed DC motor. These tools are essential for developing, testing, and refining the motor control system. By providing detailed descriptions of each hardware tool, we aim to give a comprehensive understanding of their roles and importance in the project. The hardware tools discussed include the microcontroller, development boards, power supplies, oscilloscopes, and other essential equipment used throughout the design and testing phases.

3.2.1 Hantek2000 Oscilloscope / Multimeter

The Hantek2000 Handheld Oscilloscope is a versatile and portable diagnostic tool designed for capturing and analyzing electrical signals in various applications. It features a compact design with a user-friendly interface, making it ideal for fieldwork and on-the-go troubleshooting. The oscilloscope provides real-time waveform viewing with high resolution and accuracy, capable of measuring a wide range of signal frequencies and amplitudes. It includes multiple measurement modes, such as single-shot and continuous sampling, as well as built-in functions for signal analysis. The Hantek2000 is equipped with a rechargeable battery, ensuring extended operational time without the need for constant power sources, and its robust design ensures durability in various working environments [23].

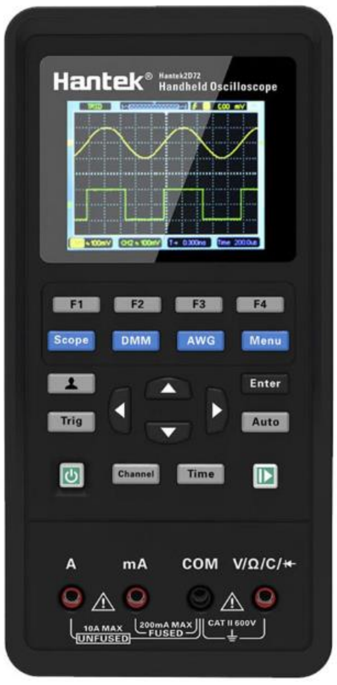


Figure 3.9 Hantek2000 Handheld Oscilloscope [23]

3.2.2 ST-LINK/V2 Debugger and Programmer

The ST-LINK/V2 is an in-circuit debugger and programmer for the STM8 and STM32 microcontroller families. It provides a straightforward interface for programming and debugging these microcontrollers, making it an essential tool for development and troubleshooting. The ST-LINK/V2 supports JTAG/SWD and

SWIM communication protocols, ensuring compatibility with a wide range of STMicroelectronics devices. It connects to a host computer via USB, offering a reliable and high-speed connection for firmware uploads and debugging sessions. The programmer is compact and portable, equipped with multiple connectors for easy integration into various development environments. Its robust design and comprehensive support make it a valuable tool for both professional and hobbyist embedded systems developers [24].

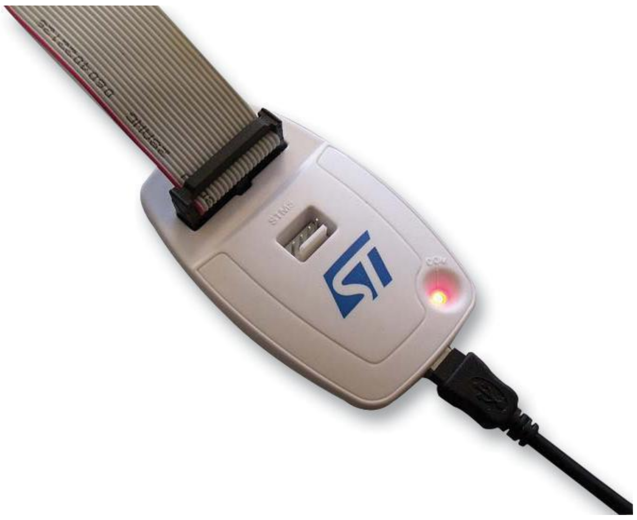


Figure 3.10 ST-LINK/V2 in-circuit debugger/programmer [24]

3.2.3 USB to TTL Module (CP2102)

The USB to TTL Module (CP2102) is a compact and efficient adapter that facilitates communication between a computer's USB port and TTL (Transistor-Transistor Logic) serial devices. Based on the CP2102 chipset, this module provides reliable and fast data transfer rates, making it ideal for a wide range of applications, including microcontroller programming, debugging, and serial communication with other TTL devices. It features easy-to-use pin headers for direct connection to the target device and supports various baud rates to accommodate different communication needs. The module is widely compatible with major operating systems, including Windows, MacOS, and Linux, ensuring seamless integration into diverse development environments. Its small form factor and robust performance make it an essential tool for embedded systems developers and hobbyists alike [25].

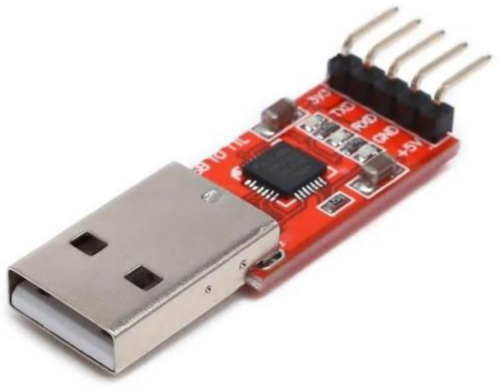


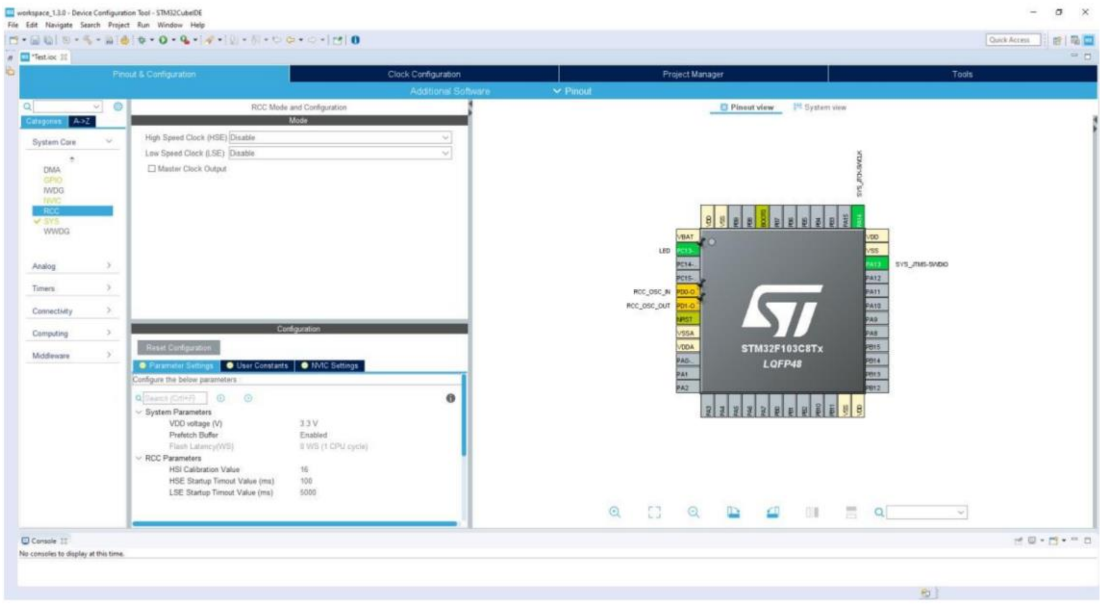
Figure 3.11 3.2.3 USB to TTL Module (CP2102) [25]

3.3 Software Tools

This section outlines the various software tools utilized in the development and implementation of the servo controller for the Brushed DC motor. These tools play a crucial role in programming, debugging, simulation, and overall project management. By providing detailed descriptions of each software tool, we aim to give a comprehensive understanding of their roles and importance in the project. The software tools discussed include integrated development environments (IDEs), debugging tools, and other essential software used throughout the design and testing phases.

3.3.1 STM32CubeIDE

STM32CubeIDE is an integrated development environment (IDE) developed by STMicroelectronics specifically for STM32 microcontrollers. It combines the capabilities of the STM32CubeMX graphical configurator with the Eclipse-based development environment, providing a comprehensive tool for configuring, coding, debugging, and optimizing STM32 applications. STM32CubeIDE supports C/C++ programming languages and offers advanced debugging features, including support for SWD, JTAG, and ST-LINK interfaces. The IDE integrates seamlessly with STM32CubeMX, allowing users to generate initialization code and set up peripheral configurations easily. With its extensive libraries and middleware, STM32CubeIDE simplifies the development process, making it an essential tool for embedded systems developers working with STM32 microcontrollers [26].



High Speed Clock (HSE) Disable
Low Speed Clock (LSE) Disable
☐ Master Clock Output

Configure the below parameters

System Parameters

VDD voltage (V)	3.3 V
Prefetch Buffer	Enabled
Flash Latency(WS)	0 WS (1 CPU cycle)

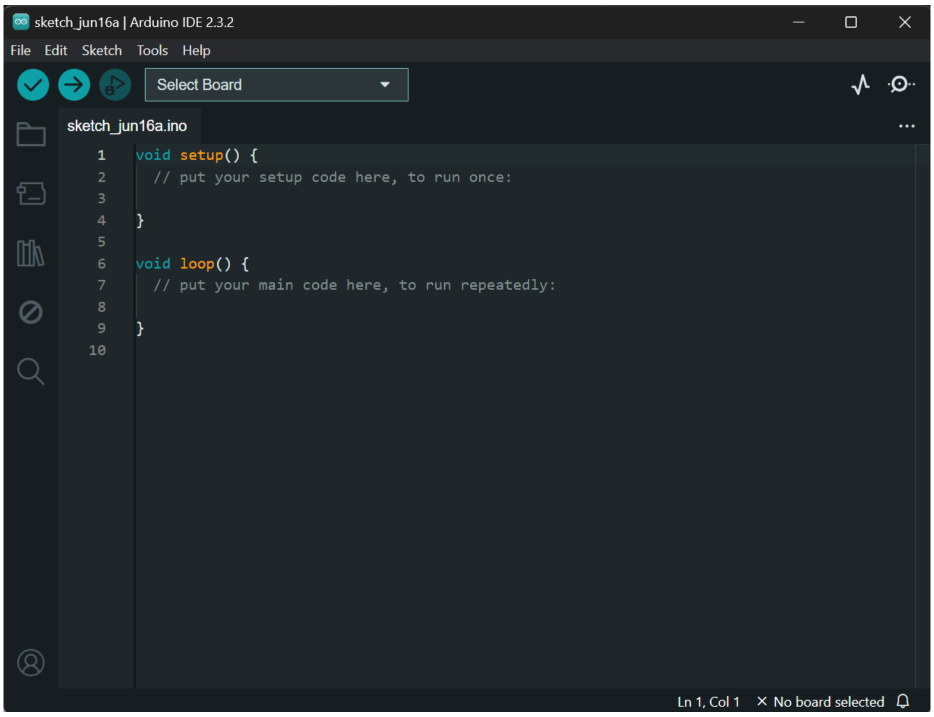
RCC Parameters

HSI Calibration Value	16
HSE Startup Timout Value (ms)	100
LSE Startup Timout Value (ms)	5000

Figure 3.12 STM32CubeIDE user interface

3.3.2 Arduino IDE 2

The Arduino Integrated Development Environment (IDE) is a versatile software tool used primarily for programming Arduino microcontrollers. It provides a user-friendly interface for writing, compiling, and uploading code to Arduino boards. In this project, the Arduino IDE was utilized specifically for its serial monitor and serial plotter features. These tools facilitate real-time data visualization and debugging by allowing users to monitor and plot serial data transmitted from the microcontroller. The serial monitor displays text data sent from the microcontroller, while the serial plotter provides graphical representation of numerical data, making it easier to analyze the system's performance and troubleshoot issues [27].



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}

```

Figure 3.13 Arduino IDE 2 user interface

3.3.3 Autodesk Fusion 360

Fusion 360 is a comprehensive 3D CAD, CAM, and CAE tool developed by Autodesk. It integrates various design and engineering capabilities into a single platform, making it a powerful tool for product development. In this project, Fusion 360 was used specifically for schematic design. The software's robust schematic capture and PCB design tools facilitated the creation of detailed and accurate circuit schematics, ensuring a reliable foundation for the hardware implementation of the servo controller. Fusion 360's intuitive interface and extensive library of electronic components made it an essential tool for designing and documenting the electrical aspects of the project [28].

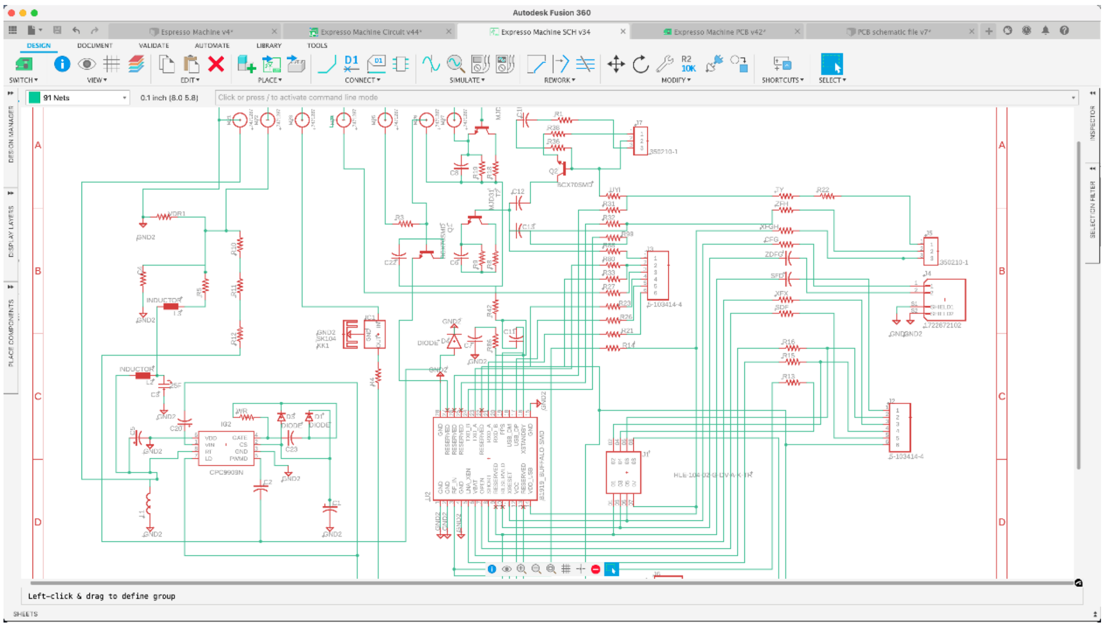


Figure 3.14 Autodesk Fusion 360 user interface

3.3.4 OpenOCD

OpenOCD (Open On-Chip Debugger) is an open-source software tool that provides debugging, in-system programming, and boundary-scan testing for embedded target devices. It supports a wide range of microcontrollers and architectures, offering capabilities such as flash programming and hardware debugging. In this project, OpenOCD was used for debugging the STM32 microcontroller, enabling detailed inspection and control over the microcontroller's operation. Its support for various debug adapters, including ST-LINK, made it an essential tool for developing and troubleshooting embedded systems. OpenOCD's flexibility and extensive feature set make it a valuable resource for embedded developers seeking to ensure the reliability and correctness of their hardware and software integrations [29].

3.4 Brushed DC Motor Driver Circuit Design

The Brushed DC motor driver circuit is designed using an H-bridge configuration to control the speed and direction of the motor. The H-bridge circuit allows for the bidirectional control of the motor by changing the polarity of the voltage applied to it. The speed of the motor is controlled using Pulse Width Modulation (PWM) signals generated from a microcontroller.

The H-bridge circuit consists of four MOSFETs arranged in a configuration that enables the motor to be driven in both forward and reverse directions. By switching the appropriate pairs of MOSFETs, the polarity of the voltage applied to the motor is reversed, allowing for the control of the motor's rotational direction [6]. The PWM signals, which control the switching of the MOSFETs, are generated by the microcontroller. The duty cycle of these PWM signals determines the average voltage applied to the motor, thus controlling its speed.

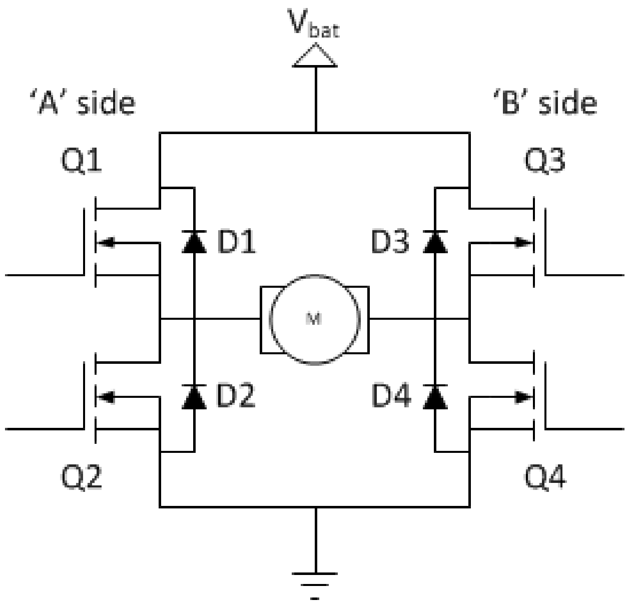


Figure 3.15 Basic H-bridge circuit [30]

3.4.1 Driver Circuit Schematic Design

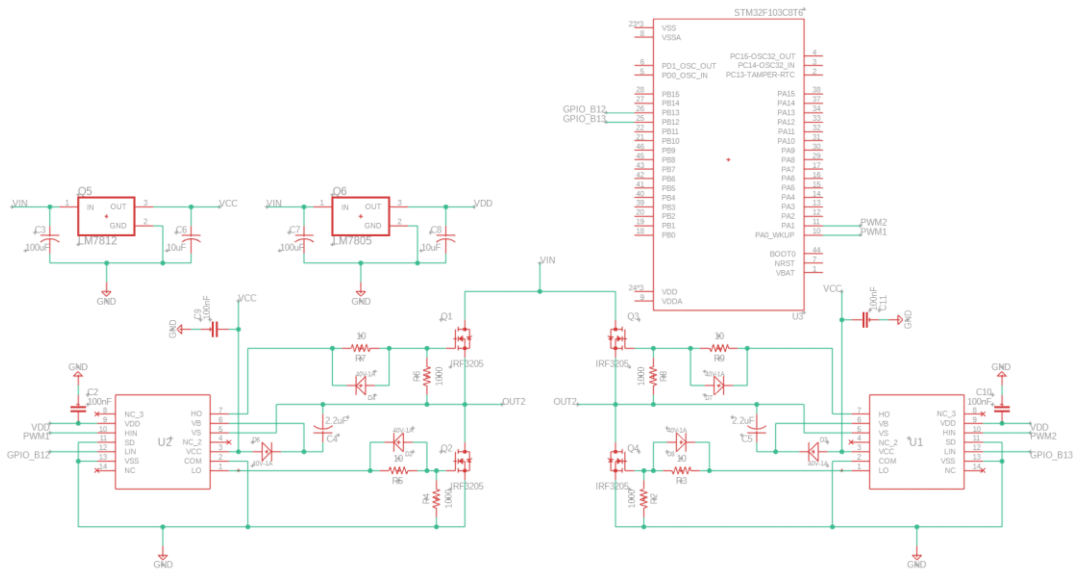


Figure 3.16 Brushed DC motor driver circuit schematic design

In the motor driver circuit, the IR2110 is utilized as a MOSFET gate driver, while IRF3205 N-channel MOSFETs are employed to construct the H-bridge configuration. This H-bridge is critical for controlling both the direction and speed of the motor. The H-bridge allows the motor to run in both forward and reverse directions by changing the polarity of the voltage applied to the motor.

The IR2110 gate driver is responsible for providing the necessary drive signals to the MOSFETs, ensuring efficient switching and minimizing losses. In the circuit design, two IR2110 MOSFET gate drivers are used to control both sides of the H-bridge [31]. The LM7812 voltage regulator maintains a steady 12V output, which is suitable for powering the IR2110 gate drivers and the motor.

The IR2110 is a high and low-side driver designed for driving N-channel MOSFETs in a half-bridge configuration. In an H-bridge setup, two IR2110 drivers are typically used to control the four MOSFETs that constitute the H-bridge. The LIN (Low-Side Input) pin on the IR2110 is crucial for controlling the low-side MOSFET in the half-bridge.

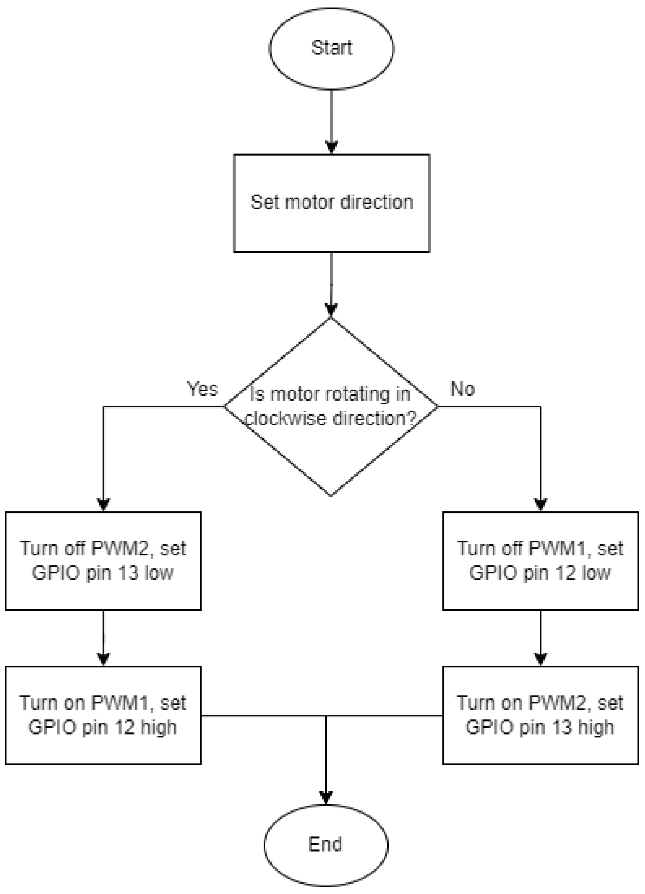

Start
Set motor direction
Is motor rotating in clockwise direction?
Yes
No
Turn off PWM2, set GPIO pin 13 low
Turn off PWM1, set GPIO pin 12 low
Turn on PWM1, set GPIO pin 12 high
Turn on PWM2, set GPIO pin 13 high
End

Figure 3.17 Motor direction flowchart

GPIO Pin 12 is connected to the LIN pin of the IR2110 on the left side of the H-bridge, and GPIO Pin 13 is connected to the LIN pin of the IR2110 on the right side. This setup allows precise control of the motor's direction by toggling these GPIO pins.

For forward motor rotation, GPIO Pin 12 is set to off (low), and GPIO Pin 13 is set to on (high). This configuration turns off the low-side MOSFET on the left side of the H-bridge and turns on the low-side MOSFET on the right side. As a result, current flows through the motor in one direction, causing it to rotate forward.

Conversely, for reverse motor rotation, GPIO Pin 12 is set to on (high), and GPIO Pin 13 is set to off (low). This configuration turns on the low-side MOSFET on the left side of the H-bridge and turns off the low-side MOSFET on the right side. The reversed current flow through the motor causes it to rotate in the opposite direction.

By toggling the states of GPIO Pin 12 and GPIO Pin 13, the direction of the motor's rotation can be controlled. While these GPIO pins manage the motor's direction, the PWM signals applied to the high-side inputs (HIN) of the IR2110 drivers modulate the motor's speed. This combined approach enables precise and flexible control of the motor's operation.

The choice of IRF3205 MOSFETs, known for their low on-resistance and high current handling capability, is essential for managing the substantial power requirements of the motor driver circuit. These MOSFETs can handle high currents up to 110A, making them suitable for high-power motor applications.

The LM7805 5V voltage regulator is used to maintain a steady 5V output, which is utilized to power the VDD of the IR2110 gate drivers and the optical encoder of the DC motor. Additionally, this 5V supply can also be used to power the microcontroller, ensuring stable and reliable operation of the control electronics.

Pulse Width Modulation (PWM) signals, which are necessary for modulating the motor speed, are generated by the STM32F103 microcontroller. The STM32F103, with its robust performance and integrated peripherals, generates precise PWM signals that control the duty cycle applied to the MOSFETs. By adjusting the duty cycle of the PWM signals, the average voltage supplied to the motor is controlled, thus regulating its speed.

The motor operates at 12V, supplied by the LM7812 voltage regulator. The MOSFETs Q1 and Q4, driven by PWM1, enable forward motor movement, while Q2 and Q3, controlled by PWM2, reverse the direction. These PWM signals are generated by the microcontroller, with adjustable duty cycles ensuring precise directional control of the motor.

3.4.2 High Side MOSFET Bootstrap Circuit

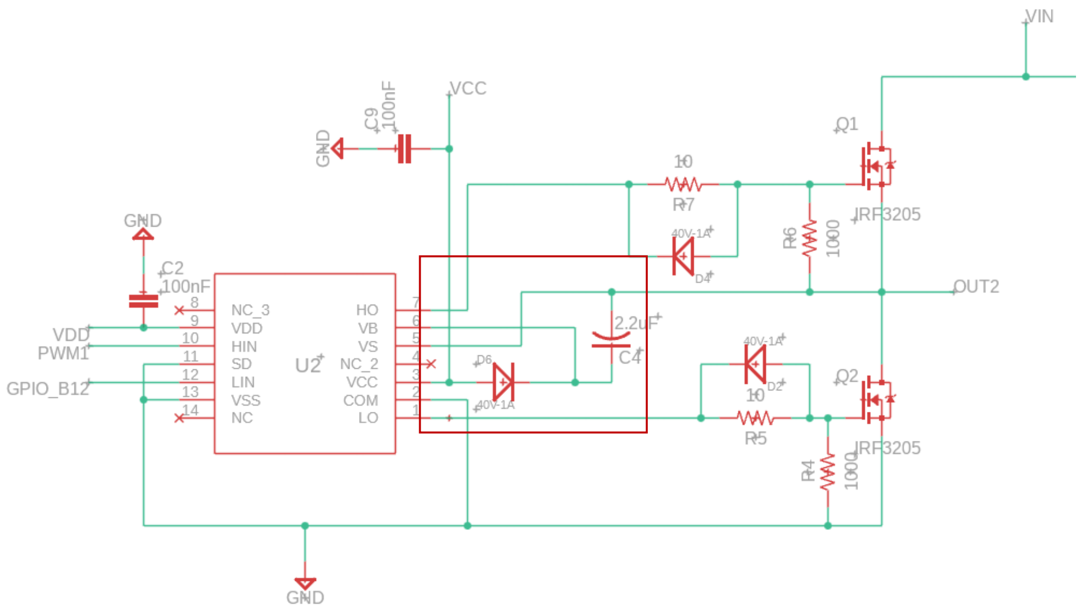


Figure 3.18 Bootstrap diodes and capacitors

Bootstrap diodes and capacitors (highlighted in figure 3.17) are included in the design to facilitate the operation of the high-side MOSFETs in the H-bridge. These components are crucial for generating the necessary gate drive voltage for the high-side MOSFETs, which is higher than the supply voltage. The bootstrap diode allows the capacitor to charge when the low-side MOSFET is on, and this stored charge is then used to drive the high-side MOSFET when it needs to turn on.

Minimum bootstrap capacitor (C_{BS}) value is calculated using the following formula:

$$C_{BS(min)} = \frac{Q_g + (I_{QBS} \times T_{on})}{Voltage\ Ripple} \tag{3.1}$$

Q_g: Total gate charge needed to fully turn on the MOSFET.

Q_g value of IRF3205 N-channel MOSFET = 146nC

I_{QBS}: Current required by the gate driver to maintain the bootstrap voltage

I_{QBS} value of IRF3205 N-channel MOSFET = 230µA

T_{on}: on-time of a switching device (MOSFET) in a PWM signal.

$$T_{on} = \frac{1}{PWM\ Frequency} \times 0.95 \tag{3.2}$$

PWM Frequency = 30KHz

0.95 is the maximum duty cycle.

$$T_{on} = \frac{1}{30K} \times 0.95 = 3.167 \times 10^{-5}\ \text{second}$$

Voltage Ripple: The allowable ripple voltage on the bootstrap capacitor.

Voltage Ripple = 1V

$$C_{BS(MIN)} = \frac{146n + (230\mu \times 3.167 \times 10^{-5})}{1} = 0.153\mu\text{F}$$

The bootstrap capacitor should be 10× the minimum value calculated [32].

$$C_{BS} = 10 \times 0.153\mu\text{F} = 1.53\mu\text{F}$$

Capacitor of value 2.2µF is selected for reliability and availability in standard values.

3.4.3 MOSFET Gate Discharge Diode

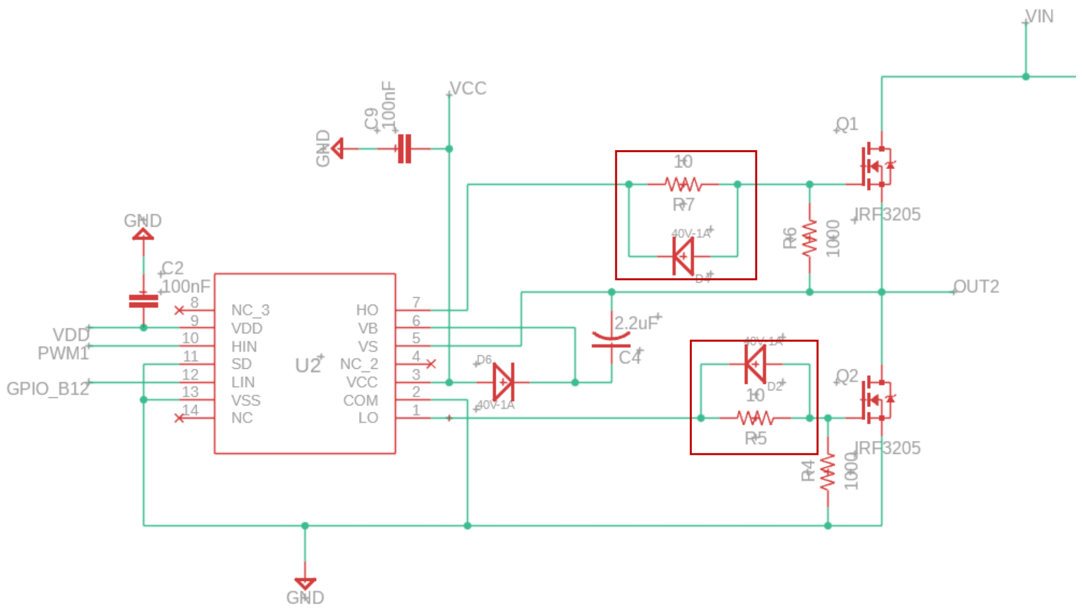


Figure 3.19 MOSFET Gate Discharge Diode

In the schematic shown in Figure 3.18, a diode is placed at the gate of each MOSFET. The primary purpose of this diode is to enhance the discharge time of the MOSFET gate. When a MOSFET is switched off, the charge stored at the gate must be rapidly discharged to ensure quick switching. This rapid discharge is crucial for achieving efficient and high-speed switching performance, which is essential in applications such as motor control where precise timing is critical.

The presence of the diode facilitates a faster discharge path for the gate capacitance. When the gate voltage is pulled low to turn off the MOSFET, the diode provides a low-impedance path for the gate charge to flow to ground, thereby speeding up the discharge process. This improvement in discharge time helps to minimize the switching losses and reduces the potential for cross-conduction, where both the high-side and low-side MOSFETs might be on simultaneously, leading to shoot-through current.

3.5 PID Controller

The Proportional-Integral-Derivative (PID) controller is a widely used feedback control system in industrial and engineering applications, including motor control. It provides a robust and efficient method for achieving precise control over dynamic systems by continuously adjusting the control inputs to minimize the error between the desired setpoint and the actual output. The PID controller operates by calculating three distinct terms: the proportional term, which responds to the current error; the integral term, which accounts for the accumulation of past errors; and the derivative term, which predicts future errors based on the rate of change. This section details the design and implementation of the PID controller for the Brushed DC motor, highlighting its role in enhancing the system's responsiveness and stability. The integration of the PID controller into the motor driver circuit ensures accurate position control, making it an essential component in the servo control system.

3.5.1 PID Controller Implementation

The Proportional-Integral-Derivative (PID) controller is a widely used control algorithm that combines three control actions proportional, integral, and derivative to provide robust and accurate control of dynamic systems. The PID controller continuously calculates an error value as the difference between a desired setpoint and a measured process variable, and it applies a correction based on proportional, integral, and derivative terms.

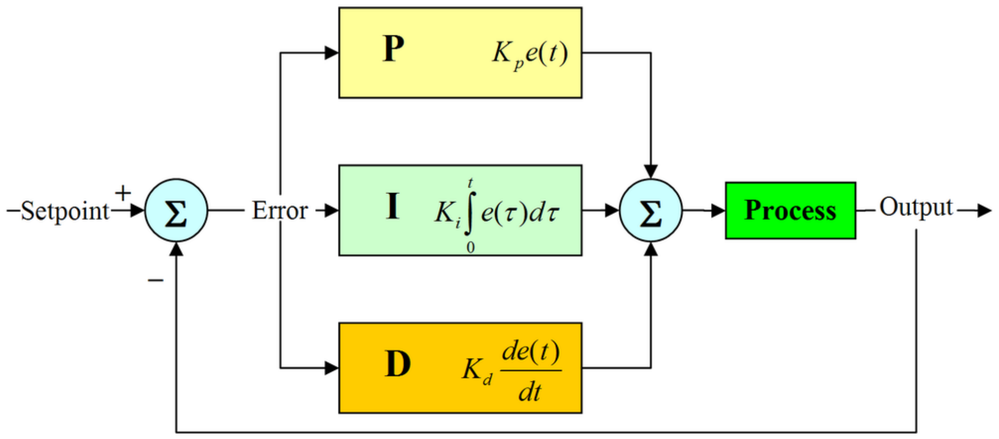


Figure 3.20 PID controller block diagram [33]

PID control formula [34] can be expressed as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \qquad (3.3)$$

where:

$u(t)$: is the control output,

$e(\tau)$: is the error between the setpoint and the process variable,

K_p: is the proportional gain,

K_i is the integral gain,

K_d: is the derivative gain.

Proportional Term

The proportional term produces an output that is proportional to the current error value:

$$P(t) = K_p e(t) \qquad (3.4)$$

The proportional response can be adjusted by changing the proportional gain K_p. A larger K_p will increase the response to the current error, but if K_p is too large, the system can become unstable.

Integral Term

The integral term produces an output that is proportional to the cumulative sum of the error over time:

$$I(t) = K_i \int_0^t e(\tau) d\tau \qquad (3.5)$$

The integral response corrects past errors by integrating the error over time. This term helps eliminate the residual steady-state error that can occur with a pure proportional controller.

Derivative Term

The derivative term produces an output that is proportional to the rate of change of the error:

$$D(t) = K_d \frac{de(t)}{dt} \qquad (3.6)$$

The derivative response predicts the future trend of the error, thus providing a damping effect that reduces the overshoot and improves the system's stability.

To tune the PID controller, the trial-and-error method [35] was employed to determine the optimal values for the proportional, integral, and derivative gains. Through iterative adjustments and testing, it was found that the best performance for the motor control system was achieved with K_p=2.0, K_i=0.1, and K_d=0.1. These values provided a good balance between responsiveness and stability, ensuring precise control of the motor's speed and position according to the specified motion profile.

The PID controller's error is calculated as the difference between the motor's current position and the theoretical position at a given time according to the motion profile. By continuously adjusting the motor's speed to minimize this error, the PID controller ensures that the motor closely follows the desired motion trajectory, achieving precise and accurate positioning throughout the entire motion sequence.

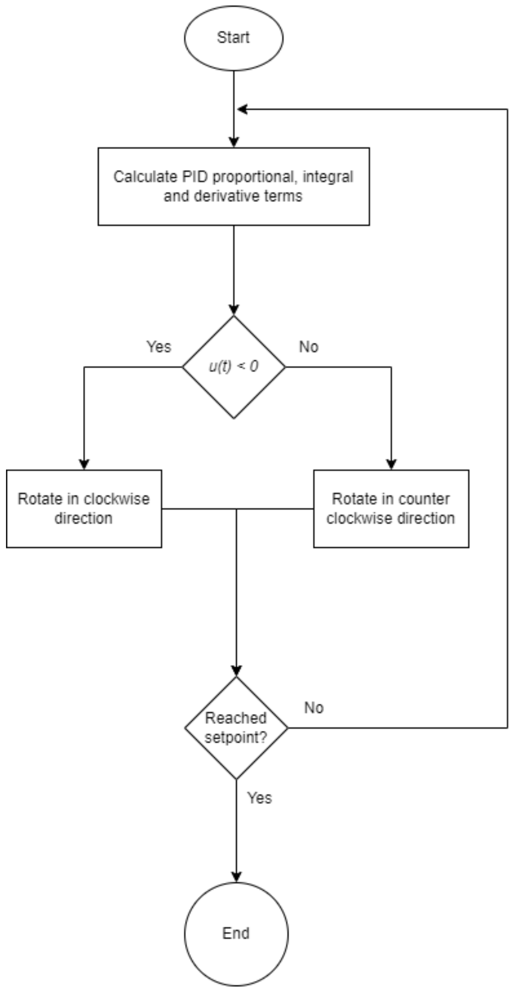

Start
Calculate PID proportional, integral and derivative terms
Yes
u(t) < 0
No
Rotate in clockwise direction
Rotate in counter clockwise direction
Reached setpoint?
No
Yes
End

Figure 3.21 Control output $u(t)$ flowchart

Based on the control output $u(t)$, the direction of the motor is determined. If the control output is negative, the motor moves in the clockwise direction; otherwise, it moves in the counterclockwise direction. The control output is also used as the duty cycle of the PWM signal. To ensure smooth and continuous motor operation, the value of the control output is capped at 205, corresponding to an 80% duty cycle.

This limitation helps to prevent abrupt stops and maintain consistent motor movement.

3.6 Motion Profile

For the motion profile [36], a trapezoidal motion profile was chosen. In this profile, the user specifies the end position, speed, acceleration, and deceleration of the motor. The software algorithms then calculate the motion profile, ensuring that the motor follows the desired position at any given time.

In a trapezoidal motion profile, the motor accelerates to the specified speed, maintains that speed for a period, and then decelerates to the end position. This approach provides smooth and controlled motion, preventing sudden changes in speed that could lead to mechanical stress or instability.

The most effective way to analyze a trapezoidal motion profile is to divide it into three segments: two right triangles representing the acceleration and deceleration phases, and a rectangle representing the constant velocity phase [37].

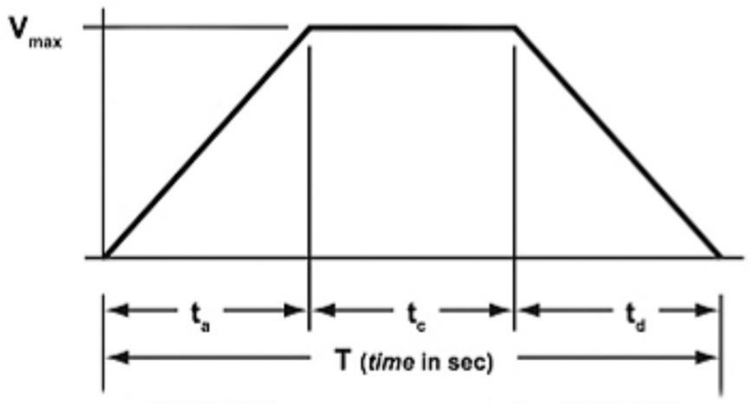


Figure 3.22 Trapezoidal motion profiles

Acceleration Phase:

To calculate the acceleration time:

$$ta = \frac{V_{max} - V_0}{a} \qquad (3.7)$$

Where:

ta: acceleration time.

V_{max}: maximum velocity.

V_0: initial velocity.

a: acceleration.

To calculate the distance at acceleration phase:

$$da = \frac{1}{2} \times a \times ta^2 \qquad (3.8)$$

Where:

da: acceleration distance.

a: acceleration.

ta: acceleration time.

Deceleration Phase:

To calculate acceleration time:

$$td = \frac{V - V_0}{d} \qquad (3.9)$$

Where:

td: deceleration time.

V: final velocity which is zero here.

V_0: initial velocity which is V_{max}.

d: deceleration.

To calculate distance at deceleration phase:

$$d_{dec} = -\frac{1}{2} \times d \times td^2 \tag{3.10}$$

Where:

d_{dec}: deceleration distance.

d: deceleration.

td: deceleration time.

Constant Velocity Phase:

To calculate the total distance:

$$d_{total} = P - P_0 \tag{3.11}$$

Where:

d_{total}: total distance.

P: target position.

P: initial position.

To calculate the distance at constant velocity phase:

$$d_c = d_{total} - (d_a + d_{dec}) \tag{3.12}$$

d_c: constant velocity distance.

d_{total}: total distance.

d_a: acceleration distance.

d_{dec}: deceleration distance.

To calculate time at constant velocity phase:

$$tc = \frac{dc}{V_{max}} \tag{3.13}$$

tc: time at constant velocity phase.

d_c: constant velocity distance.

V_{max}: max velocity.

3.7 Brushed DC Motor Optical Encoder

The optical encoder [38] used in this setup has two outputs, A and B. These outputs are used to determine both the position and direction of the motor's rotation. Output A is connected to GPIO pin 7, and output B is connected to GPIO pin 6, which is configured to trigger an interrupt.

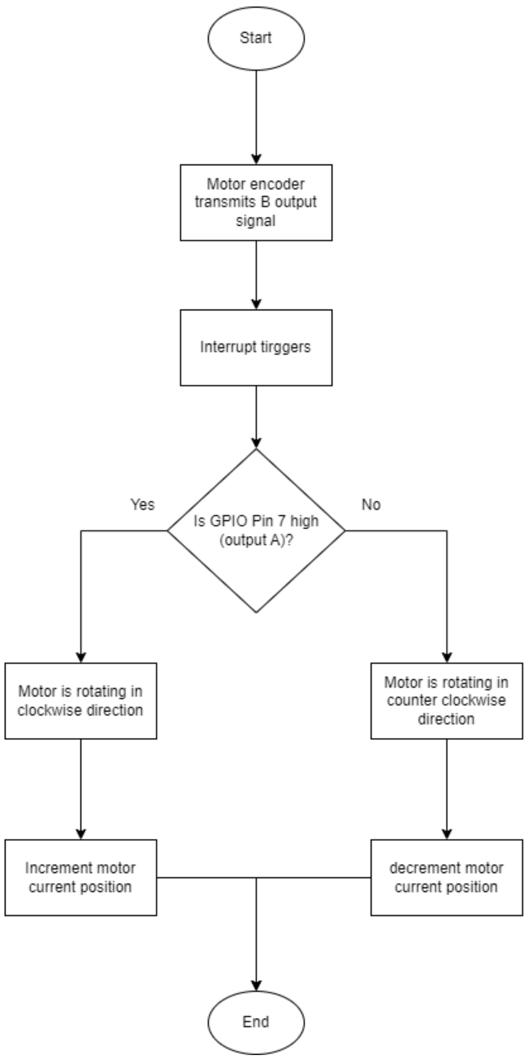

Start
Motor encoder transmits B output signal
Interrupt tirggers
Is GPIO Pin 7 high (output A)?
Yes
No
Motor is rotating in clockwise direction
Motor is rotating in counter clockwise direction
Increment motor current position
decrement motor current position
End

Figure 3.23 Motor optical encoder flowchart

When a state change (either a rising or falling edge) from output B is detected on GPIO pin 6, an interrupt is triggered. This interrupt calls a callback function, which processes the encoder signals to update the motor's position.

The direction of the motor's rotation is determined by the relative states of outputs A and B. If output A (GPIO pin 7) is leading output B (GPIO pin 6), the motor is moving in a clockwise direction. Conversely, if output B is leading output A, the motor is moving in a counterclockwise direction.

During the interrupt service routine, the state of GPIO pin 7 is read to determine the direction. If GPIO pin 7 is logic high (1) when the interrupt occurs, it indicates that the motor is moving clockwise. If GPIO pin 7 is logic low (0), it indicates that the motor is moving counterclockwise [39].

The motor's current position is then updated by incrementing or decrementing a position variable based on the direction of rotation. This variable stores the motor's position in terms of encoder pulses. To convert this position into degrees or another unit of measurement, an appropriate scaling factor, based on the encoder's resolution and the gear ratio, is applied.

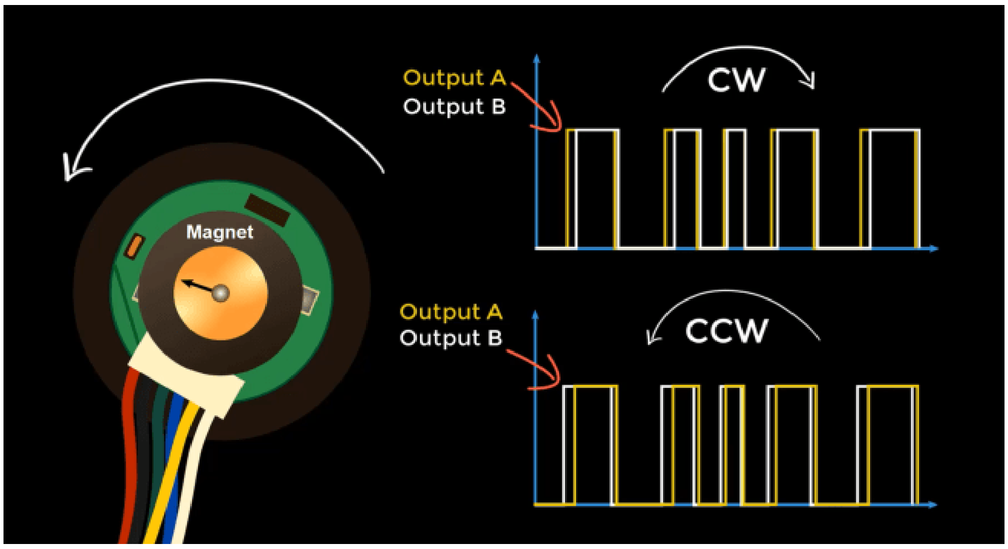


Figure 3.24 Encoder output A and B [39]

The encoder pulses, combined with the motor's gear ratio, are converted into degrees to simplify user interpretation. Using degrees allows for more intuitive understanding and control of the motor's position. Additionally, representing the motor's position in degrees facilitates the integration of the current position into the motion profile calculations, ensuring precise alignment with the desired motion trajectory. This approach enhances both usability and accuracy, enabling more effective control and monitoring of the motor's movements.

To convert the motor position from encoder pulses to degrees [40]:

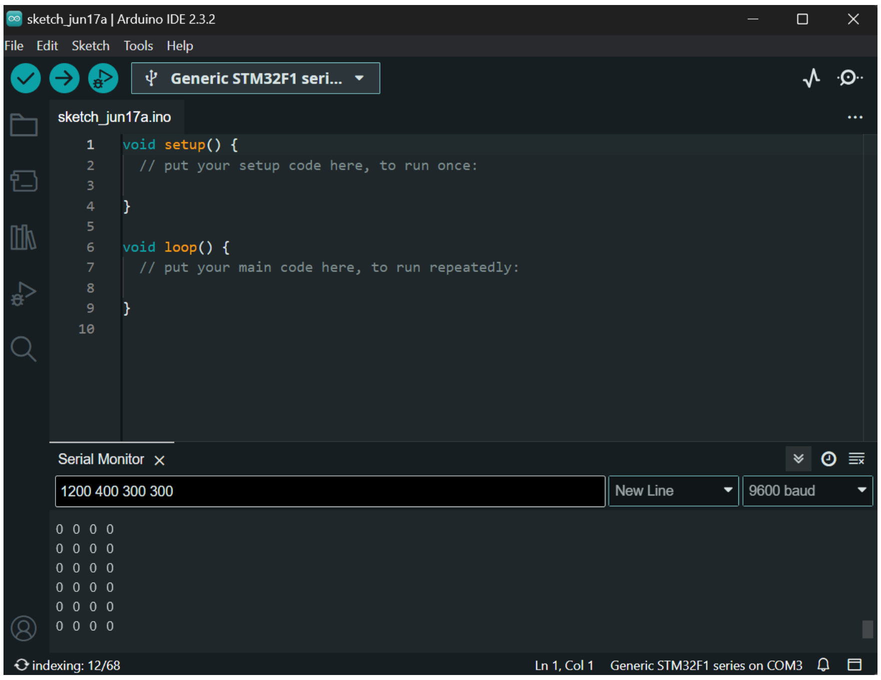
$$Position\ (Degrees) = \frac{360 \times Position\ (Encoder\ Pulses)}{Motor\ Gear\ Ratio \times Encoder\ Pulses\ Per\ Revolution} \quad (3.14)$$

The gear ratio of MD32NP2724V brushed DC motor is 3249:121 and the encoder pulses per revolution is 500 pulses.

The motor angular speed [41] is calculated by determining the difference between the current motor position and the previous motor position, and then dividing this difference by the time interval between these two position measurements. This method allows for an accurate assessment of the motor's speed by using the change in position over a known period of time. By continuously updating the current and previous positions, the system can provide real-time speed measurements, which are crucial for precise motor control and maintaining the desired motion profile.

3.8 User Interface via UART

In this project, UART (Universal Asynchronous Receiver-Transmitter) [42] is utilized for the user to send input data to the system. The Arduino IDE is employed for the serial monitor [43], which facilitates easy communication and data entry.

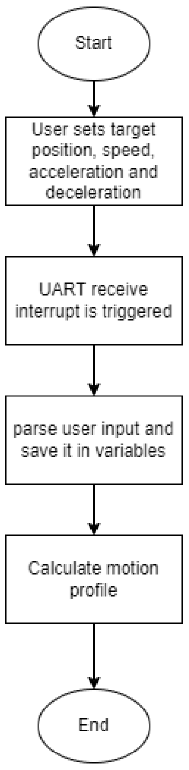


```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
1200 400 300 300
```

```
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
```

Figure 3.25 Arduino IDE serial monitor



Start
User sets target position, speed, acceleration and deceleration
UART receive interrupt is triggered
parse user input and save it in variables
Calculate motion profile
End

Figure 3.26 User input flowchart

The user inputs data in the following order: position in degrees, speed in degrees per second, acceleration in degrees per second squared, and deceleration in degrees per second squared. Once these parameters are received, the system processes them to define the motor's motion profile. The motor then starts moving according to this specified motion profile, ensuring precise control based on the provided inputs.

3.9 System Flowchart

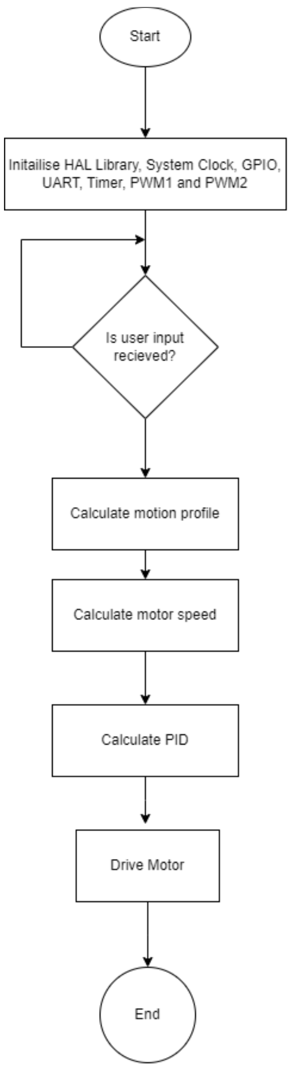


Figure 3.27 System flowchart

The system flowchart outlines the sequential steps involved in the operation of the motor control system. The process begins with the initialization of the Hardware Abstraction Layer (HAL) [44] library, which provides a uniform interface for hardware components. The system clock is set up to ensure precise timing and synchronization of operations. General Purpose Input/Output (GPIO) pins are configured for various functions, such as reading encoder signals and controlling the motor. The Universal Asynchronous Receiver-Transmitter (UART) interface is initialized for user communication, and timers are configured to generate PWM signals for motor control (PWM1 and PWM2).

The system then waits for user input, which is sent via the UART interface. The input data includes the desired position, speed, acceleration, and deceleration. Once the user input is received, the system calculates the motion profile based on the specified parameters. A trapezoidal motion profile is used, where the motor accelerates to the specified speed, maintains that speed, and then decelerates to the end position. This profile ensures smooth and controlled motion.

Next, the motor speed is calculated by determining the difference between the current motor position and the previous motor position, divided by the time interval between these two position measurements. This real-time speed measurement is crucial for precise motor control. The PID controller then calculates the control output *u(t)* based on the error between the desired position (setpoint) and the actual position. The PID parameters were determined using the trial and error method. The control output is used as the duty cycle of the PWM signal to adjust the motor's speed and direction, and it is capped at 205 (80% duty cycle) to ensure smooth motor operation.

Based on the control output *u(t)*, the direction and speed of the motor are determined. If the control output is negative, the motor moves in the clockwise direction; otherwise, it moves in the counterclockwise direction. The motor is driven according to the calculated motion profile and PID control to achieve the desired position and motion characteristics. The system reaches the end of its operation sequence after executing these steps.

CHAPTER 4 DATA PRESENTATION AND DISCUSSION OF FINDINGS

This chapter presents the collected data and discusses the findings from the implementation and testing of the motor control system. The data includes various performance metrics such as motor speed, position accuracy, and response to user inputs, as well as results from the motor driver circuit to assess its efficiency and effectiveness in controlling the motor's speed and direction. The analysis aims to evaluate the effectiveness of the PID controller, the accuracy of the motion profile, and the overall system performance. By interpreting the results, this chapter seeks to provide insights into the strengths and limitations of the designed system, as well as potential areas for improvement.

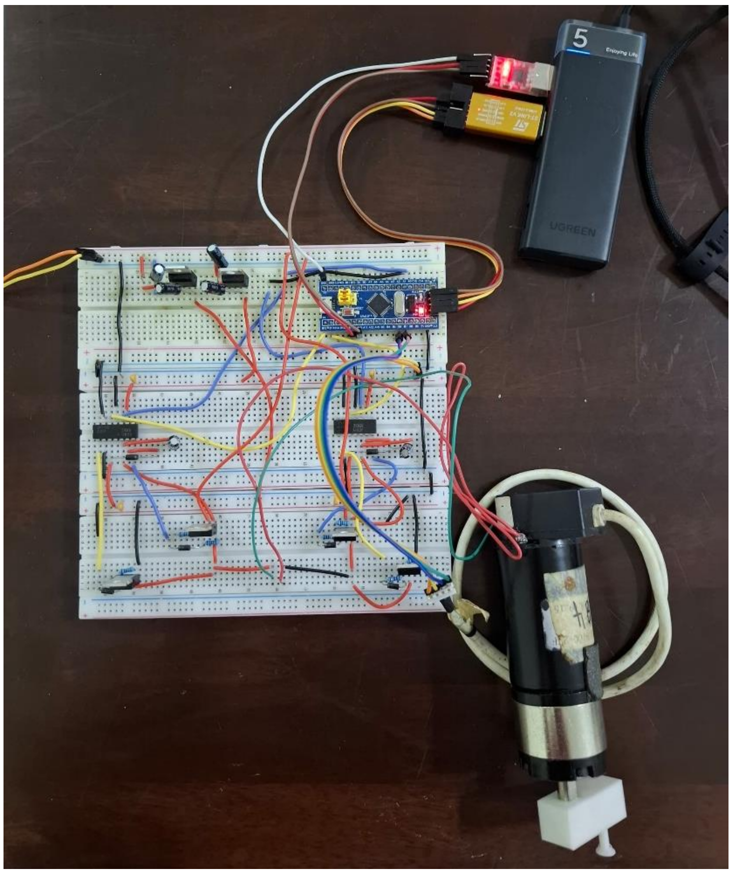


Figure 4.1 Brushed DC motor driver

4.1 Brushed DC Motor Driver Circuit

The performance of the Brushed DC Motor Driver Circuit was thoroughly evaluated to determine its effectiveness in controlling the motor's speed and direction with high precision and efficiency. This involved a series of tests and measurements designed to assess various aspects of the circuit's functionality. Key performance indicators included the accuracy of speed regulation, the responsiveness to control inputs, the stability of direction changes, and the overall efficiency of power management. By systematically analysing these parameters, we aimed to ensure that the motor driver circuit meets the stringent requirements necessary for reliable operation in practical applications.

4.1.1 Driver Circuit at PWM 40% Duty Cycle

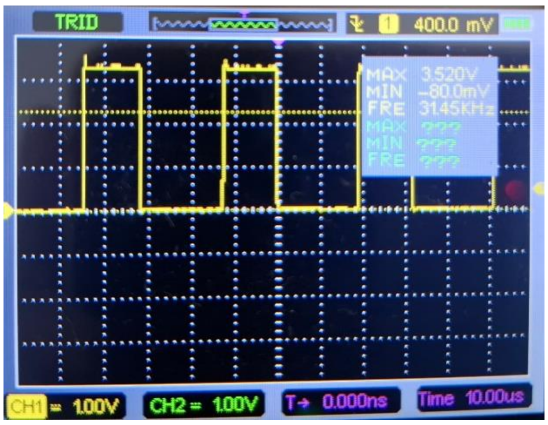


Figure 4.2 Clockwise 40% Duty cycle PWM signal

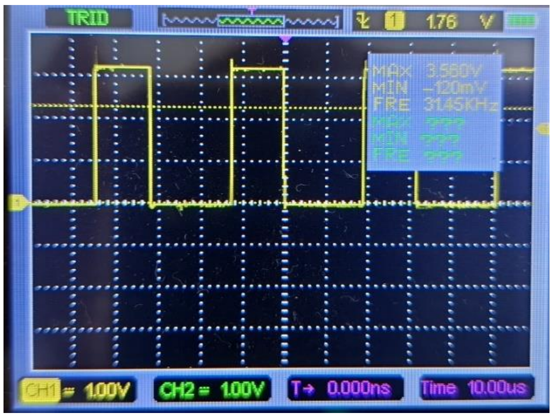


Figure 4.3 Counter-Clockwise 40% Duty cycle PWM signal

As observed from the graphs, the PWM signal generated by the STM32 microcontroller exhibits a duty cycle of 40% in both the clockwise and counterclockwise directions. This consistent duty cycle demonstrates the microcontroller's ability to maintain uniform control over the motor's speed regardless of the direction of rotation. The stability of the PWM signal ensures that the motor operates smoothly and efficiently in both directions, highlighting the effectiveness of the control algorithm implemented in the microcontroller. Additionally, the precise regulation of the duty cycle contributes to the overall accuracy of the motor's performance, ensuring reliable and predictable behavior in various operational scenarios.

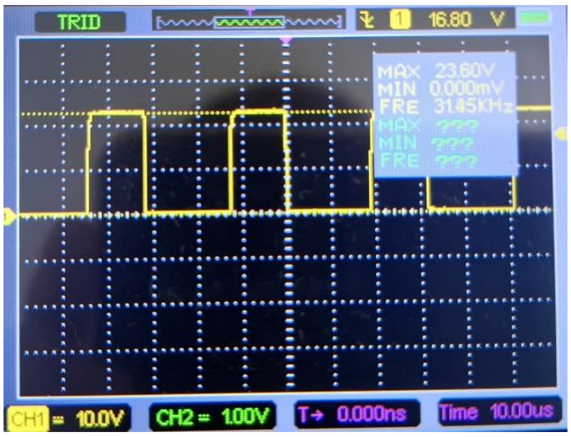


Figure 4.4 Clockwise 40% duty cycle high side MOSFET gate voltage

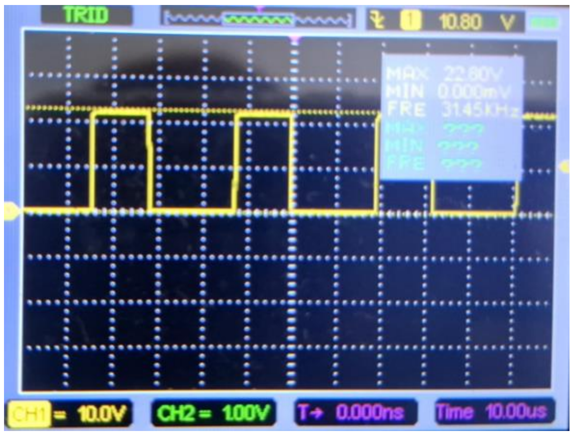


Figure 4.5 Counter-Clockwise 40% duty cycle high side MOSFET gate voltage

It is evident from the data that the input signal to the high-side MOSFET gate consistently measures approximately 24V. This indicates that the bootstrap capacitor is functioning correctly, providing the necessary gate drive voltage for the high-side MOSFETs. The consistent voltage level confirms the reliability of the bootstrap circuit in both the clockwise and counterclockwise directions of motor rotation. The effective operation of the bootstrap capacitor ensures that the high-side MOSFETs are fully turned on, minimizing conduction losses and improving overall efficiency. This performance is crucial for maintaining the stability and responsiveness of the motor driver circuit under various load conditions.

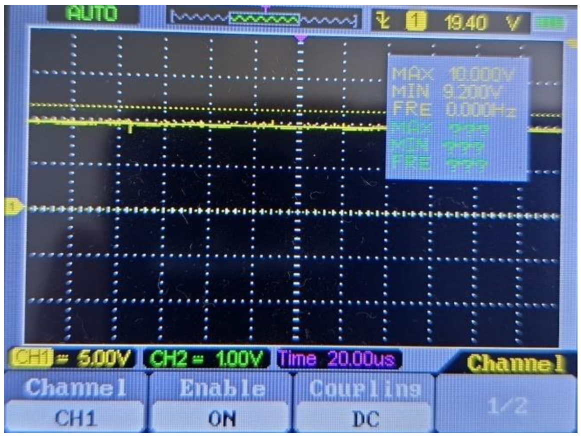


Figure 4.6 Clockwise 40% duty cycle low side MOSFET gate to source voltage

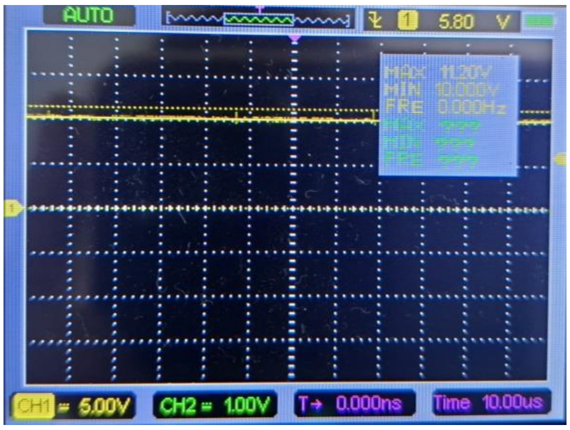


Figure 4.7 Counter-Clockwise 40% duty cycle low side MOSFET gate to source voltage

The maximum voltage at the gate of the low-side MOSFET is approximately 11V, closely matching the VCC supplied to the IR2110 MOSFET driver IC. This consistency, observed in both clockwise and counterclockwise directions, indicates effective regulation by the IR2110 driver. The proximity of the gate voltage to the VCC supply ensures full enhancement of the MOSFETs, minimizing conduction losses and ensuring efficient operation. This reliable gate drive voltage is crucial for stable and precise motor control, enabling rapid and efficient switching in response to PWM signals from the microcontroller.

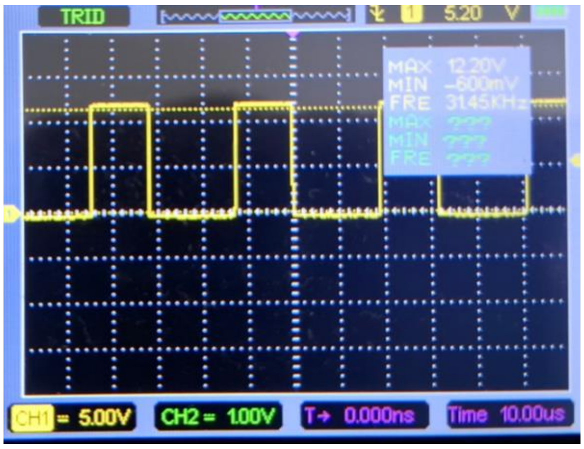


Figure 4.8 Motor clockwise 40% duty cycle

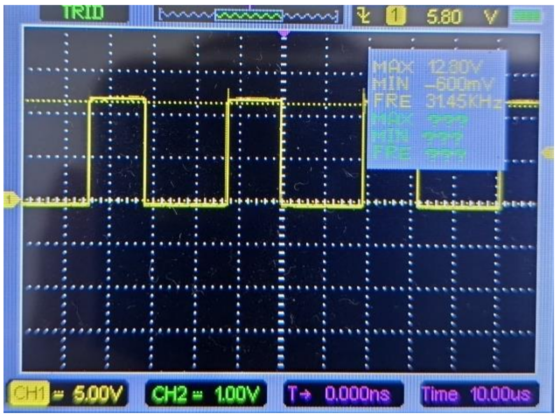


Figure 4.9 Motor Counter-Clockwise 40% duty cycle

As illustrated by the graphs, the motor operates with a duty cycle of 40%, and it functions properly in both clockwise and counterclockwise directions. This indicates that the control system is effectively regulating the PWM signal to maintain consistent motor performance. The stability of the 40% duty cycle ensures that the motor receives a uniform amount of power, facilitating smooth and efficient operation. Additionally, the motor's reliable performance in both rotational directions underscore the robustness of the motor driver circuit and the accuracy of the implemented control algorithms. This consistency is crucial for applications requiring precise bidirectional control, demonstrating that the system can handle varying operational demands without compromising performance.

4.1.2 Driver Circuit at PWM 80% Duty Cycle

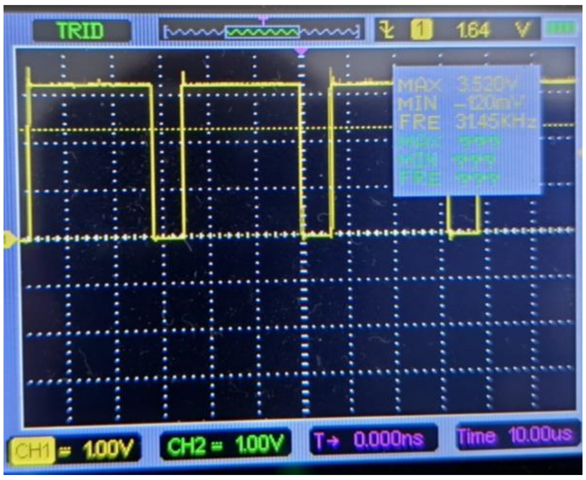


Figure 4.10 Clockwise 80% duty cycle PWM signal

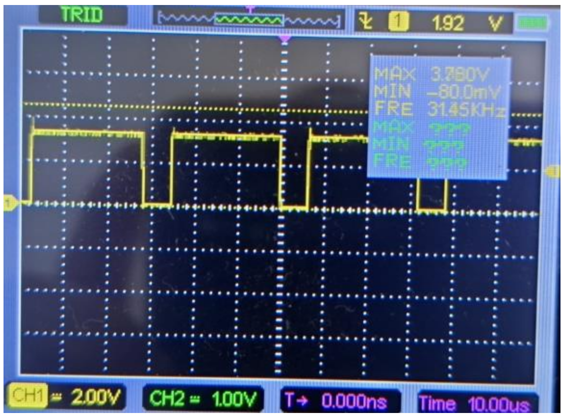


Figure 4.11 Counter-Clockwise 80% duty cycle PWM signal

As observed from the graphs, the PWM signal generated by the STM32 microcontroller exhibits a duty cycle of 80% in both the clockwise and counterclockwise directions. This consistent duty cycle demonstrates the microcontroller's ability to maintain uniform control over the motor's speed regardless of the direction of rotation. The stability of the PWM signal ensures that the motor operates smoothly and efficiently in both directions, highlighting the effectiveness of the control algorithm implemented in the microcontroller. Additionally, the precise regulation of the duty cycle contributes to the overall accuracy of the motor's performance, ensuring reliable and predictable behaviour in various operational scenarios.

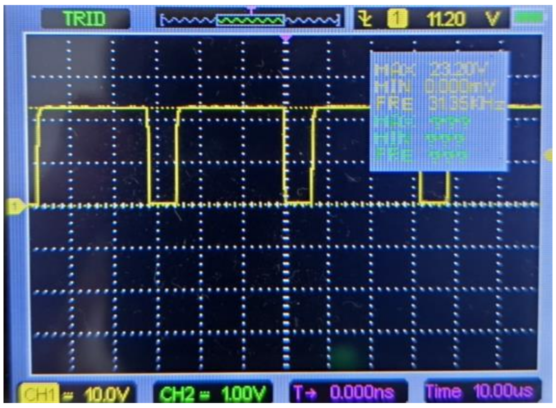


Figure 4.12 Clockwise 80% duty cycle high side MOSFET gate to source voltage

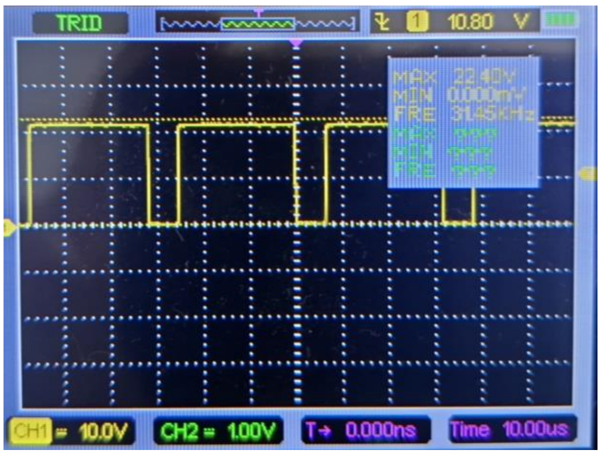


Figure 4.13 Counter-Clockwise 80% duty cycle high side MOSFET gate to source voltage

It is evident from the data that the input signal to the high-side MOSFET gate consistently measures approximately 24V. This indicates that the bootstrap capacitor is functioning correctly, providing the necessary gate drive voltage for the high-side MOSFETs. The consistent voltage level confirms the reliability of the bootstrap circuit in both the clockwise and counterclockwise directions of motor rotation. The effective operation of the bootstrap capacitor ensures that the high-side MOSFETs are fully turned on, minimizing conduction losses and improving overall efficiency.

This performance is crucial for maintaining the stability and responsiveness of the motor driver circuit under various load conditions.

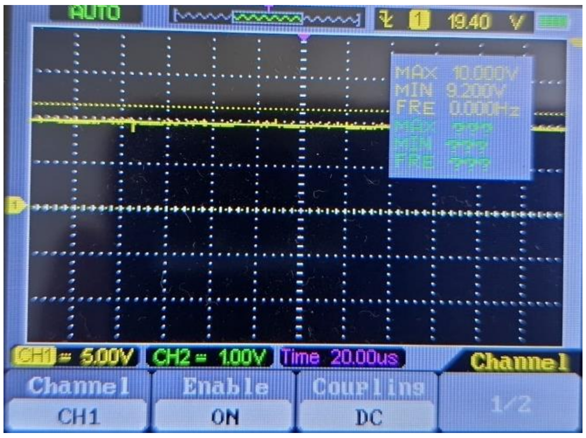


Figure 4.14 Clockwise 80% duty cycle low side MOSFET gate to source voltage

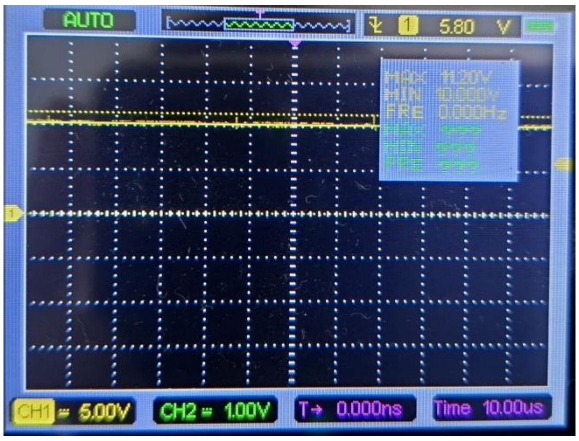


Figure 4.15 Counter-Clockwise 80% duty cycle low side MOSFET gate to source voltage

The maximum voltage at the gate of the low-side MOSFET is approximately 11V, closely matching the VCC supplied to the IR2110 MOSFET driver IC. This consistency, observed in both clockwise and counterclockwise directions, indicates effective regulation by the IR2110 driver. The proximity of the gate voltage to the VCC supply ensures full enhancement of the MOSFETs, minimizing conduction losses and ensuring efficient operation. This reliable gate drive voltage is crucial for stable and precise motor control, enabling rapid and efficient switching in response to PWM signals from the microcontroller.

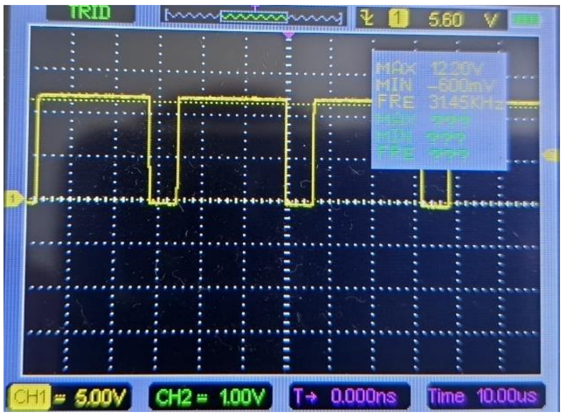


Figure 4.16 Motor clockwise 80% duty cycle

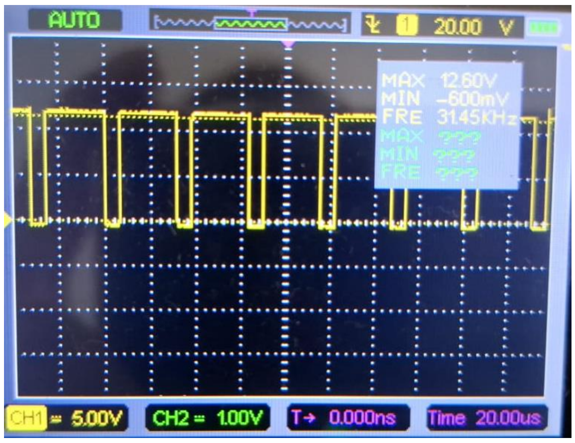


Figure 4.17 Motor Counter-Clockwise 80% duty cycle

As illustrated by the graphs, the motor operates with a duty cycle of 80%, and it functions properly in both clockwise and counterclockwise directions. This indicates that the control system is effectively regulating the PWM signal to maintain consistent motor performance. The stability of the 80% duty cycle ensures that the motor receives a substantial amount of power, facilitating smooth and efficient operation. Additionally, the motor's reliable performance in both rotational directions underscore the robustness of the motor driver circuit and the accuracy of the implemented control algorithms. Furthermore, the circuit is capable of driving the motor at various duty cycles, demonstrating its versatility and adaptability to different operational demands. This consistency and flexibility are crucial for applications requiring precise bidirectional control, ensuring the system can handle varying conditions without compromising performance.

4.2 PID Controller

This section presents the results from the implementation and testing of the PID controller used in the motor control system. The primary objective is to evaluate the performance of the PID controller in achieving precise control over the motor's speed and position. Key metrics such as response time, accuracy in following the motion profile, and stability under various operating conditions are analysed. The effectiveness of the PID tuning, conducted through the trial and error method, is also assessed to determine its impact on the overall system performance. By interpreting these results, we aim to provide insights into the strengths and potential areas for improvement of the PID control implementation.

4.2.1 PID Controller Performance under No Load

Configuring the servo controller to move from 0 to 1200 degrees with an angular speed of 450 degrees per second, an acceleration of 360 degrees per second squared, and a deceleration of 360 degrees per second squared.

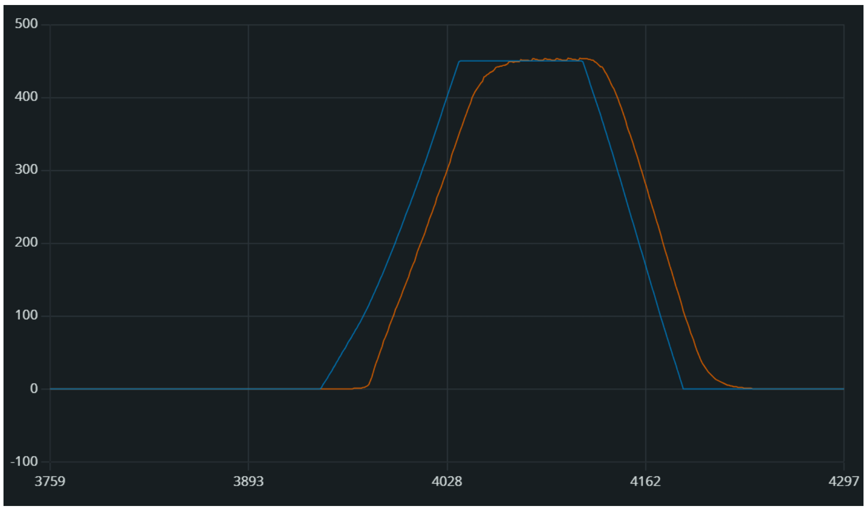
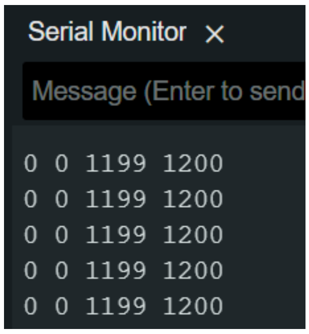


Figure 4.18 Motor speed graph 0 degrees to 1200 degrees under no load



```
0  0  1199  1200
0  0  1199  1200
0  0  1199  1200
0  0  1199  1200
0  0  1199  1200
```

Figure 4.19 Serial monitor motor position

As shown in Figure 4.17, the blue graph represents the theoretical motion profile, while the orange graph represents the motor following the motion profile. It is evident that the motor attempts to follow the theoretical motion profile closely. In Figure 4.18, the final position of the motor is 1199 degrees, which is just one degree short of the target position.

Configuring the servo controller to move from 0 to 120 degrees with an angular speed of 50 degrees per second, an acceleration of 200 degrees per second squared, and a deceleration of 200 degrees per second squared.

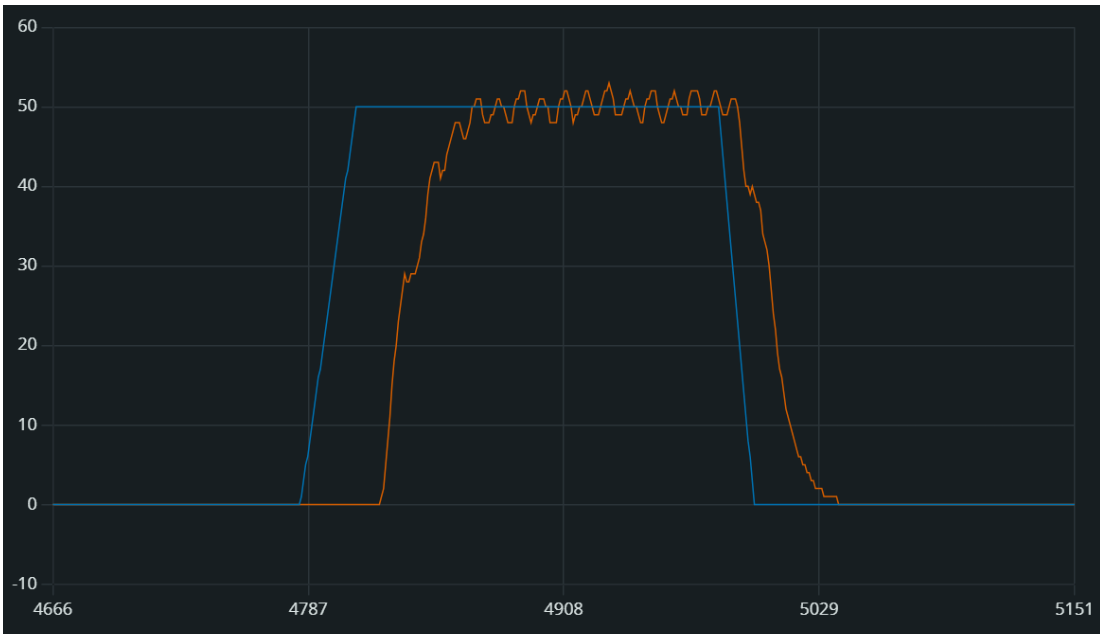
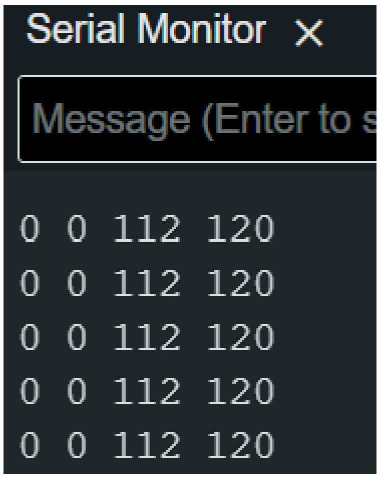


Figure 4.20 Motor speed graph 0 degrees to 120 degrees under no load



```
0 0 112 120
0 0 112 120
0 0 112 120
0 0 112 120
0 0 112 120
```

Figure 4.21 Serial monitor motor position

As shown in Figure 4.19, the orange graph represents the motor speed. It is evident that the motor speed oscillates significantly at low speeds and ultimately stops at 112 degrees, which is quite far from the target position.

4.2.2 PID Controller Performance under Constant Load

Configuring the servo controller to move from 0 to -3000 degrees with an angular speed of 360 degrees per second, an acceleration of 200 degrees per second squared, and a deceleration of 200 degrees per second squared.

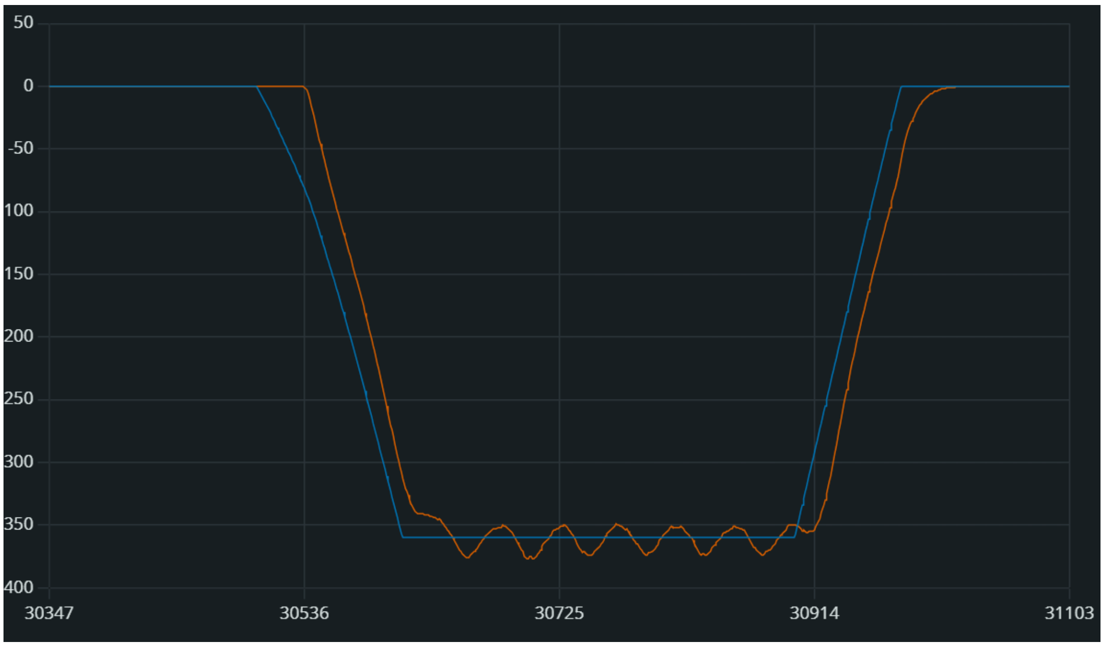
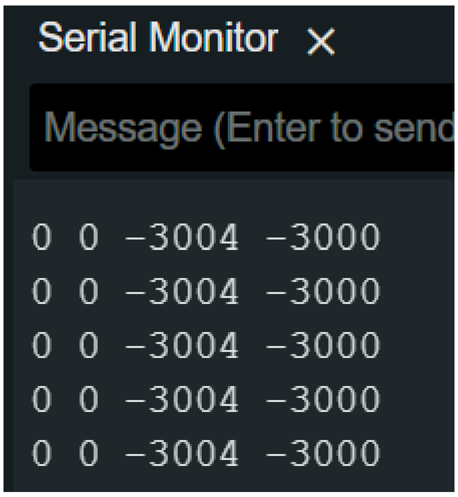


Figure 4.22 Motor speed graph 0 degrees to -3000 degrees under contant load



```
0  0  -3004  -3000
0  0  -3004  -3000
0  0  -3004  -3000
0  0  -3004  -3000
0  0  -3004  -3000
```

Figure 4.23 Serial monitor motor position

When the motor is under a load of 150g, it can be observed that the motor speed oscillates slightly around the maximum speed value. However, the motor is still able to reach the target speed with a small error of only 4 degrees.

4.2.3 PID Controller Performance under Varying Load

Configuring the servo controller to move from 0 to 4000 degrees with an angular speed of 500 degrees per second, an acceleration of 300 degrees per second squared, and a deceleration of 100 degrees per second squared

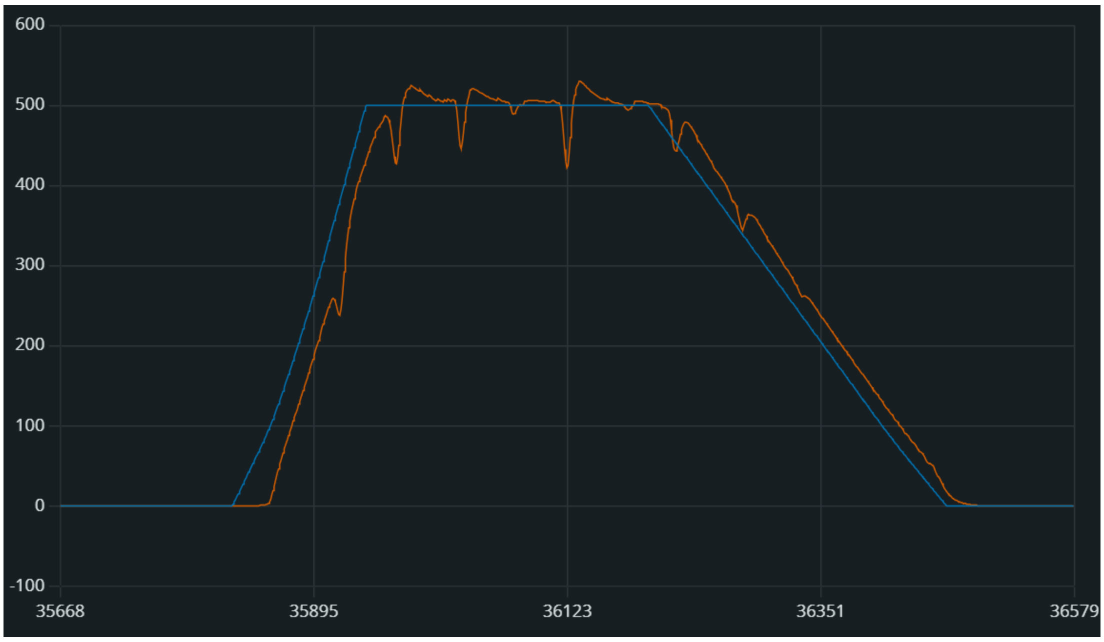
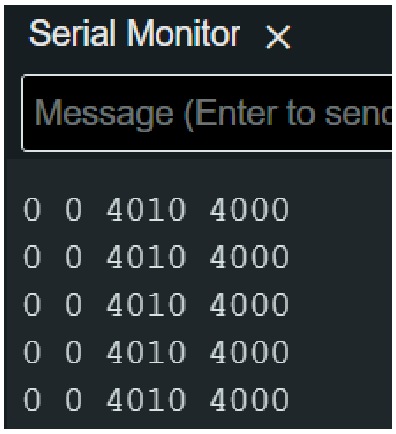


Figure 4.24 Motor speed graph 0 degrees to 4000 degrees under varying load



```
0 0 4010 4000
0 0 4010 4000
0 0 4010 4000
0 0 4010 4000
0 0 4010 4000
```

Figure 4.25 Serial monitor motor position

As illustrated in Figure 4.23, the orange graph shows the motor speed fluctuating under the changing load. Despite these variations, it is evident that the servo controller attempts to adjust accordingly. Additionally, Figure 4.24 demonstrates that the final position is very close to the target, with an error of only 10 degrees.

4.3 Discussion of Findings

This section interprets and explains the results obtained from the implementation and testing of the servo controller for the Brushed DC motor. The findings are discussed in the context of the literature and existing knowledge, with a focus on the performance of the motor driver circuit and the PID controller.

4.3.1 Brushed DC Motor Driver Circuit

The Brushed DC Motor Driver Circuit was evaluated for its ability to control the motor's speed and direction accurately and efficiently. The results indicated that the PWM signal generated by the STM32 microcontroller maintained a consistent duty cycle in both clockwise and counterclockwise directions. This consistency ensured that the motor received a uniform amount of power, facilitating smooth and efficient operation. The stability of the PWM signal and the precise regulation of the duty cycle contributed to the overall accuracy of the motor's performance.

The input signal to the high-side MOSFET gate consistently measured approximately 24V, indicating that the bootstrap capacitor functioned correctly. This confirmed the reliability of the bootstrap circuit in both directions of motor rotation, ensuring that the high-side MOSFETs were fully turned on and minimizing conduction losses.

The maximum voltage at the gate of the low-side MOSFET was approximately 11V, closely matching the VCC supplied to the IR2110 MOSFET driver IC. This effective regulation by the IR2110 driver ensured full enhancement of the MOSFETs, minimizing conduction losses and enabling efficient operation. The reliable gate drive voltage was crucial for stable and precise motor control, allowing rapid and efficient switching in response to PWM signals from the microcontroller.

The motor's performance at different duty cycles demonstrated the robustness and versatility of the motor driver circuit. At both 40% and 80% duty cycles, the motor operated smoothly and efficiently in both directions. The circuit's capability to drive the motor at various duty cycles highlighted its adaptability to different operational demands, making it suitable for applications requiring precise bidirectional control.

4.3.2 PID Controller Performance

The PID controller was tested for its performance in achieving precise control over the motor's speed and position. The results under no load conditions showed that the motor closely followed the theoretical motion profile, with only a minor deviation of one degree from the target position. This indicated that the PID controller was effective in minimizing the error and maintaining accurate control of the motor's position.

Under constant load conditions, the motor's speed oscillated slightly around the maximum speed value, but it was still able to reach the target speed with a small error of only 4 degrees. This demonstrated the PID controller's capability to adapt to external disturbances and maintain accurate control.

When tested under varying load conditions, the motor speed fluctuated, but the servo controller adjusted accordingly. The final position was very close to the target, with an error of only 10 degrees. This highlighted the PID controller's robustness and ability to handle changing load conditions while maintaining precise control over the motor's position.

4.3.3 Limitations

While the system demonstrates significant strengths, there are some limitations. One notable limitation is that the motion profile cannot be changed dynamically while the motor is running. Once the motion profile parameters are set and the motor starts its operation, any adjustments to the motion profile require stopping the motor and reconfiguring the parameters. This limitation can impact applications where real-time adjustments to the motion profile are necessary to respond to dynamic operational conditions.

Another limitation is that the servo controller cannot run the motor at very small angular speeds, resulting in a stuttering effect. This issue arises due to the minimum effective duty cycle that the system can maintain, which impacts the smoothness of the motor's operation at low speeds. This limitation could affect applications requiring precise low-speed control, as the motor may not run as smoothly and could experience intermittent motion.

CHAPTER 5 CONCLUSIONS

5.1 Summary and Conclusions

This project aimed to design and implement a servo controller for a Brushed DC motor, focusing on achieving precise control over the motor's speed and position. The development process included designing a motor driver circuit, implementing a PID controller, and integrating a motion profile to guide the motor's movements. The STM32 microcontroller was used for generating PWM signals, while the IR2110 and IRF3205 components were selected for the H-bridge configuration to manage the motor's speed and direction.

Key findings from the evaluation of the motor driver circuit showed that the PWM signals maintained consistent duty cycles in both clockwise and counterclockwise directions, ensuring smooth and efficient motor operation. The bootstrap capacitor and IR2110 gate driver functioned effectively, providing reliable gate drive voltages and minimizing conduction losses.

The PID controller's performance was analysed under no load, constant load, and varying load conditions. The results indicated that the PID controller could closely follow the theoretical motion profile, maintaining accurate control over the motor's position with minimal errors. However, some limitations were identified, including the inability to change the motion profile dynamically during operation and difficulties in maintaining smooth motion at very low speeds.

The findings of this project demonstrate that the designed servo controller effectively meets the precision requirements for controlling a Brushed DC motor. The motor driver circuit and PID controller work in tandem to provide reliable and accurate control, ensuring the motor operates smoothly and efficiently under various conditions.

5.2 Areas of Future Research

Considering the findings and limitations identified in this project, several areas of future research are suggested to enhance the performance and versatility of the servo controller for the Brushed DC motor.

5.2.1 Auto-Tuning Methods:

Exploring and implementing auto-tuning methods such as fuzzy logic or Ziegler-Nichols could significantly improve the efficiency of the PID controller. These methods can automate the tuning process, optimizing the PID parameters in real-time and adapting to changing conditions without manual intervention. This could enhance the system's responsiveness and stability, particularly in dynamic environments.

5.2.2 Dynamic Adjustment of Motion Profile:

Developing methods to dynamically adjust the motion profile in real-time is crucial for applications requiring adaptability to changing operational conditions. Implementing algorithms that allow for on-the-fly adjustments without stopping the motor would increase the system's flexibility and efficiency. This capability is particularly beneficial in scenarios where real-time response to external stimuli is essential.

5.2.3 Enhanced Low-Speed Control:

Improving the control algorithm to handle low-speed operation more effectively is another critical area for future research. Reducing stuttering and ensuring smoother motion at minimal angular speeds would expand the system's applicability to tasks requiring precise low-speed control. Techniques such as advanced PWM modulation and refined PID tuning could be explored to achieve this goal.

5.2.4 PCB Design for Motor Driver:

Designing a dedicated PCB for the motor driver would enhance the overall system integration and reliability. A custom PCB could incorporate optimized layouts for the H-bridge, voltage regulators, and other components, reducing signal interference and improving thermal management. This design step would also facilitate easier assembly and maintenance, contributing to a more robust and user-friendly system.

By addressing these areas of future research, the performance, adaptability, and reliability of the servo controller for the Brushed DC motor can be significantly enhanced. These improvements would make the system more suitable for a broader range of high-precision and dynamic applications.

REFERENCES

[1] D. Steefo, "A comprehensive guide on DC motor applications," *Medium*, Jan. 15, 2023. [Online]. Available: https://medium.com/@dm.steefo/a-comprehensive-guide-on-dc-motor-applications-ecbbf417a5f8#:~:text=DC%20motors%20have%20many%20advantages,appliances%20to%20large%20industrial%20machinery. [Accessed: Jun. 15, 2024].

[2] "Introduction to Servo Motors," Fuji Electric, [Online]. Available: https://www.fujielectric.com/about/column/detail/servo_01.html#01. [Accessed: Jun. 15, 2024].

[3] "Servo Motors: Advantages," RealPars, [Online]. Available: https://www.realpars.com/blog/servo-motors-advantages. [Accessed: Jun. 15, 2024].

[4] J. Upadhyay, "What is a PWM Signal?," CircuitBread. [Online]. Available: https://www.circuitbread.com/ee-faq/what-is-a-pwm-signal. [Accessed: Jun. 15, 2024].

[5] T. Mortenson, "PID Controller Explained," RealPars, Dec. 14, 2022. [Online]. Available: https://www.realpars.com/blog/pid-controller. [Accessed: Jun. 15, 2024].

[6] J. Shah, "How does an H-Bridge work?," CircuitBread. [Online]. Available: https://www.circuitbread.com/ee-faq/how-does-an-h-bridge-work. [Accessed: Jun. 15, 2024].

[7] T. Özer, S. Kıvrak, and Y. Oğuz, "H Bridge DC Motor Driver Design and Implementation with Using dsPIC30f4011," International Journal of Innovative Research in Science, Engineering and Technology, vol. 6, Special Issue 10, May 2017. [Online]. Available: https://www.researchgate.net/publication/317225711_H_Bridge_DC_Motor_Driver_Design_and_Implementation_with_Using_dsPIC30f4011. [Accessed: Jun. 15, 2024].

[8] K. Priyanka and A. Mariyammal, "DC Motor Speed Control Using PWM," International Journal of Innovative Science and Research Technology, vol. 3, no. 2, Feb. 2018. [Online]. Available: https://www.researchgate.net/publication/338116979_DC_Motor_Speed_Control_Using_PWM. [Accessed: Jun. 15, 2024].

[9] J. A.-K. Mohammed, "Pulse Width Modulation for DC Motor Control Based on LM324," Engineering and Technology Journal, vol. 31, Part (A), no. 10, pp. 1882-1896, Mar. 2013. [Online]. Available: https://www.researchgate.net/publication/338389782_Pulse_Width_Modulation_for_DC_Motor_Control_Based_on_LM324. [Accessed: Jun. 15, 2024].

[10] U. Waseem, "PID Loops: A Comprehensive Guide to Understanding and Implementation," Wevolver, Dec. 12, 2022. [Online]. Available: https://www.wevolver.com/article/pid-loops-a-comprehensive-guide-to-understanding-and-implementation. [Accessed: Jun. 15, 2024]

[11] Khaled Sailan and K.-D. Kuhnert, "DC Motor Angular Position Control Using PID Controller for the Purpose of Controlling the Hydraulic Pump," International Conference on Control, Engineering & Information Technology (CEIT'13), 2013.

[12] M. M. Maung, M. M. Latt, and C. M. Nwe, "DC Motor Angular Position Control Using PID Controller with Friction Compensation," International Journal of Scientific and Research Publications, vol. 8, no. 11, Nov. 2018.

[13] M. Saad, A. Amhed, and M. Al Sharqawi, "Real Time DC Motor Position Control Using PID Controller in LabVIEW," Journal of Robotics and Control (JRC), vol. 2, no. 5, Sep. 2021. [Online]. Available: https://www.researchgate.net/publication/348151106_Real_Time_DC_Motor_Position_Control_Using_PID_Controller_in_LabVIEW. [Accessed: Jun. 15, 2024].

[14] R. Awar, Y. K. Al Ali, and Y. Al Jrab, "Speed and Position Control of a DC Motor," American University of Beirut, Beirut, Lebanon, 2021. [Online]. Available:
https://www.researchgate.net/publication/348151106_Real_Time_DC_Motor

_Position_Control_Using_PID_Controller_in_LabVIEW. [Accessed: Jun. 15, 2024].

[15] Infineon Technologies, "IRF3205," Infineon, [Online]. Available: https://www.infineon.com/cms/en/product/power/mosfet/n-channel/irf3205/. [Accessed: Jun. 11, 2024].

[16] Infineon Technologies, "IR2110," Infineon, [Online]. Available: https://www.infineon.com/cms/en/product/power/gate-driver-ics/ir2110/. [Accessed: Jun. 11, 2024].

[17] Components101, "LM7812 Voltage Regulator IC," [Online]. Available: https://components101.com/ics/lm7812-voltage-regulator-ic-pinout-datasheet-circuit-specifications. [Accessed: Jun. 11, 2024].

[18] "7805 IC Voltage Regulator," Electronics For You, [Online]. Available: https://www.electronicsforu.com/technology-trends/learn-electronics/7805-ic-voltage-regulator. [Accessed: Jun. 11, 2024].

[19] "1N5819 Schottky Diode," ElProCus, [Online]. Available: https://www.elprocus.com/1n5819-schottky-diode/. [Accessed: Jun. 11, 2024].

[20] STMicroelectronics, "STM32F103 Microcontrollers," [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f103.html. [Accessed: Jun. 11, 2024].

[21] J. Benas, "Power Management and RTC Alarms in STM32F103," Medium, Oct. 10, 2020. [Online]. Available: https://medium.com/@jobenas_25464/power-management-and-rtc-alarms-in-stm32f103-d144a214cc40. [Accessed: Jun. 11, 2024].

[22] "24V Brushed DC Motor with Optical Encoder," AliExpress, [Online]. Available: https://vi.aliexpress.com/i/1005001798024541.html. [Accessed: Jun. 11, 2024].

[23] "Hantek2000 Handheld Oscilloscope," Hantek, [Online]. Available: https://hantek.com/products/detail/13174. [Accessed: Jun. 11, 2024].

[24] "ST-LINK/V2 In-circuit debugger/programmer for STM8 and STM32," STMicroelectronics, [Online]. Available: https://www.st.com/en/development-tools/st-link-v2.html. [Accessed: Jun. 11, 2024].

[25] "CP2102 USB to UART Bridge VCP Drivers," Silicon Labs, [Online]. Available: https://www.silabs.com/interface/usb-bridges/classic/device.cp2102?tab=specs. [Accessed: Jun. 11, 2024].

[26] "STM32CubeIDE," STMicroelectronics, [Online]. Available: https://www.st.com/en/development-tools/stm32cubeide.html. [Accessed: Jun. 11, 2024].

[27] "Arduino IDE," Arduino Documentation, [Online]. Available: https://docs.arduino.cc/software/ide/. [Accessed: Jun. 11, 2024].

[28] "Fusion 360," Autodesk, [Online]. Available: https://www.autodesk.com/asean/products/fusion-360/overview?term=1-YEAR&tab=subscription. [Accessed: Jun. 11, 2024].

[29] "OpenOCD: Open On-Chip Debugger," OpenOCD, [Online]. Available: https://openocd.org/. [Accessed: Jun. 11, 2024].

[30] A. Tantos, "H-bridges: The Basics," Modular Circuits, May 20, 2010. [Online]. Available: https://www.modularcircuits.com/blog/articles/h-bridge-secrets/h-bridges-the-basics/. [Accessed: Jun. 11, 2024].

[31] T. M. Tahmid, "Using the high-low side driver IR2110 with motor drives and power MOSFETs," Tahmid's blog, Jan. 10, 2013. [Online]. Available: https://tahmidmc.blogspot.com/2013/01/using-high-low-side-driver-ir2110-with.html. [Accessed: Jun. 11, 2024].

[32] Texas Instruments, "Bootstrap Circuit Design for MOSFETs," Application Report SLUA887A, Aug. 2018. [Online]. Available: https://www.ti.com/lit/an/slua887a/slua887a.pdf?ts=1718551562248&ref_url=https%253A%252F%252Fwww.google.com%252F. [Accessed: Jun. 11, 2024].

[33] M. N. "Speed Control of DC Motor Using PID Algorithm STM32," Instructables, May 23, 2021. [Online]. Available: https://www.instructables.com/Speed-Control-of-DC-Motor-Using-PID-Algorithm-STM3/. [Accessed: Jun. 11, 2024].

[34] "Proportional Integral Derivative Control," APMonitor, [Online]. Available: https://apmonitor.com/pdc/index.php/Main/ProportionalIntegralDerivative. [Accessed: Jun. 11, 2024].

[35] "PID Tuning Methods," InControl, [Online]. Available: https://www.incatools.com/pid-tuning/pid-tuning-methods/. [Accessed: Jun. 11, 2024].

[36] D. Collins, "What is a motion profile?" Motion Control Tips, [Online]. Available: https://www.motioncontroltips.com/what-is-a-motion-profile/. [Accessed: Jun. 11, 2024].

[37] D. Collins, "How to calculate velocity," Linear Motion Tips, [Online]. Available: https://www.linearmotiontips.com/how-to-calculate-velocity/. [Accessed: Jun. 11, 2024].

[38] "What is an optical encoder?" Encoder Products Company, [Online]. Available: https://www.encoder.com/article-what-is-an-optical-encoder. [Accessed: Jun. 11, 2024].

[39] "Position Control," Curiores, [Online]. Available: https://curiores.com/positioncontrol. [Accessed: Jun. 11, 2024].

[40] "Using an Optical Encoder to Measure Angles," Quantum Devices Inc., [Online]. Available: https://www.quantumdev.com/using-an-optical-encoder-to-measure-angles/. [Accessed: Jun. 11, 2024].

[41] "How are encoders used for speed measurement?" Motion Control Tips, [Online]. Available: https://www.motioncontroltips.com/how-are-encoders-used-for-speed-measurement/. [Accessed: Jun. 11, 2024].

[42] E. Peňa and M. G. Legaspi, "UART: A Hardware Communication Protocol," Analog Dialogue, Analog Devices, [Online]. Available:

https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html. [Accessed: Jun. 11, 2024].

[43] "Serial Monitor," Arduino Documentation, [Online]. Available: https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-serial-monitor/. [Accessed: Jun. 11, 2024].

[44] STMicroelectronics, "Description of STM32F4 HAL and Low-layer Drivers," User Manual UM1725, [Online]. Available: https://www.st.com/resource/en/user_manual/um1725-description-of-stm32f4-hal-and-lowlayer-drivers-stmicroelectronics.pdf. [Accessed: Jun. 11, 2024].

APPENDIX A

main.cpp file

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>

#include <math.h>

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/

/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/

/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/

/* USER CODE BEGIN PM */

#define RECIEVED_BUFFER_SIZE 4

#define PWM_CLOCKWISE_CHANNEL TIM_CHANNEL_1

#define GPIO_CLOCKWISE_PIN GPIO_PIN_12 // port B

#define GPIO_COUNTER_CLOCKWISE_PIN GPIO_PIN_13 // Port B

#define PWM_COUNTER_CLOCKWISE_CHANNEL TIM_CHANNEL_2

#define PWM_DEADTIME_DELAY 1

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

TIM_HandleTypeDef htim2;

TIM_HandleTypeDef htim3;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */

typedef enum MotorDirection

{

	ClockWise = 0,
		CounterClockWise,
  none
}MotorDirection;
typedef struct ConfigData
{
		float speed;
		float position;
		float acceleration;
		float deceleration;
		float accelerationTime;
		float decelerationTime;
		float encoderPulses;
		float gearRatio;
		float ki;
		float kp;
		float kd;
		uint8_t send;
} ConfigData;
MotorDirection motorDirection = ClockWise;
MotorDirection prevMotorDirection = none;
float filt_speed_1 = 0.001f;
float motor_speed_1 = 0.001f;
uint32_t PWM_CurrentChannel = PWM_CLOCKWISE_CHANNEL;
uint32_t PWM_countingDutyCycle = 0;
uint8_t rx_buffer[1];
uint8_t buffer[256];
```

```
uint8_t bufferIndex = 0;

ConfigData data;

float motor_current_position;

float motor_prev_position;

float motor_current_speed = 1.0f;

float motor_current_acceleration;

float _time;

// position PID

uint32_t currentTime;

float deltaTime;

uint32_t prevTime;

float errorValue;

float prevErrorValue;

float derivative;

float integral;

float controlSignal = 0.0f;

static float fs = 0.0f;

//Motion Profile

typedef struct MotionProfile

{

        float max_acceleration;

        float max_deceleration;

        float total_distance;

        float distance_to_max_velocity;

        float distance_to_stop;

        float distance_at_max_velocity;
```

```
	float time_to_max_velocity;
	float time_at_max_velocity;
	float time_to_stop;
	float total_time;
	float max_velocity;
  float initial_velocity;
  float initial_position;
  MotorDirection dir;
} MotionProfile;
MotionProfile motionProfile;
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
/* USER CODE BEGIN PFP */
float PulsestoDegrees(float pulses)
{
	return ((pulses * 360.0f) / (data.gearRatio * data.encoderPulses));
}
void CalculateMotionProfile()
{
	memset(&motionProfile, 0, sizeof(motionProfile));
  if(data.position < PulsestoDegrees(motor_current_position))
  motionProfile.dir = ClockWise;

 else

  motionProfile.dir = CounterClockWise;

       motionProfile.max_acceleration = data.acceleration;

       motionProfile.max_deceleration = data.deceleration;

  motionProfile.initial_velocity = fs;

  motionProfile.initial_position = PulsestoDegrees(motor_current_position);

       motionProfile.total_distance = fabs((data.position) - (PulsestoDegrees(motor_current_position)));

       motionProfile.max_velocity = motionProfile.dir == ClockWise? -data.speed : data.speed;

       if(motionProfile.max_acceleration == 0.0f || motionProfile.max_deceleration == 0.0f || motionProfile.max_velocity == 0.0f)

              return;

       motionProfile.time_to_max_velocity = fabs((motionProfile.max_velocity - motionProfile.initial_velocity) / motionProfile.max_acceleration);

       motionProfile.time_to_stop = fabs((motionProfile.max_velocity) / -motionProfile.max_deceleration);



       motionProfile.distance_to_max_velocity = 0.5 * motionProfile.max_acceleration * (float)pow(motionProfile.time_to_max_velocity, 2);

       motionProfile.distance_to_stop = 0.5 * motionProfile.max_deceleration * (float)pow(motionProfile.time_to_stop, 2);

       motionProfile.distance_at_max_velocity = motionProfile.total_distance - (motionProfile.distance_to_max_velocity + motionProfile.distance_to_stop);



       motionProfile.time_at_max_velocity = fabs(motionProfile.distance_at_max_velocity / motionProfile.max_velocity);

  motionProfile.total_time = motionProfile.time_at_max_velocity + motionProfile.time_to_max_velocity + motionProfile.time_to_stop;



       motionProfile.max_deceleration *= -1;
```

```
        if(motionProfile.dir == ClockWise)

        {

        motionProfile.distance_at_max_velocity *= -1;

        motionProfile.distance_to_max_velocity *= -1;

        motionProfile.distance_to_stop *= -1;

        motionProfile.total_distance *= -1;

        motionProfile.max_acceleration *= -1;

        motionProfile.max_deceleration *= -1;

        }

}

float GetPostionAtTime()

{

  float currentPos = 0.0f;

  if (_time <= motionProfile.time_to_max_velocity)

  {

    currentPos = motionProfile.initial_position + (motionProfile.initial_velocity * _time + 0.5f *
( motionProfile.max_acceleration * pow(_time, 2)));

  }

  else if(_time <= motionProfile.time_to_max_velocity + motionProfile.time_at_max_velocity)

  {

    currentPos = motionProfile.initial_position + motionProfile.distance_to_max_velocity +
(motionProfile.max_velocity * (_time - motionProfile.time_to_max_velocity));

  }

  else if(_time <= motionProfile.total_time)

  {

    float v = ((motionProfile.max_velocity * (_time - (motionProfile.time_to_max_velocity +
motionProfile.time_at_max_velocity))) + 0.5f * motionProfile.max_deceleration * pow((_time -
(motionProfile.time_to_max_velocity + motionProfile.time_at_max_velocity)), 2));
```

```
    currentPos = motionProfile.initial_position + motionProfile.distance_to_max_velocity +
motionProfile.distance_at_max_velocity + v;

  }

  else

  {

    currentPos = motionProfile.initial_position + motionProfile.distance_to_max_velocity +
motionProfile.distance_at_max_velocity + motionProfile.distance_to_stop;

  }

  return currentPos;

}

float GetVelocityAtTime()

{

              float velocity;

              //Acceleration Phase

              if(_time < motionProfile.time_to_max_velocity)

              {

    if(motionProfile.initial_velocity > motionProfile.max_velocity)

                             velocity = motionProfile.initial_velocity + motionProfile.max_acceleration * _time;

    else

      velocity = motionProfile.initial_velocity + motionProfile.max_acceleration * _time;

              }

              //Max Velocity Phase

             else if(_time < (motionProfile.time_at_max_velocity + motionProfile.time_to_max_velocity))

              {

                            velocity = motionProfile.max_velocity;

              }

              //Deceleration Phase
```

```
        else if (_time < (motionProfile.total_time))
        {
  if(motionProfile.initial_velocity > motionProfile.max_velocity)
                velocity = motionProfile.max_velocity + (motionProfile.max_deceleration * (_time - motionProfile.time_at_max_velocity - motionProfile.time_to_max_velocity));
  else
   velocity = motionProfile.max_velocity + (motionProfile.max_deceleration * (_time - motionProfile.time_at_max_velocity - motionProfile.time_to_max_velocity));
        }
        else
        {
                velocity = 0.0f;
        }
 return velocity;
}
void CalculateSpeed()
{
        motor_current_speed = (PulsestoDegrees(motor_current_position) - PulsestoDegrees(motor_prev_position)) / deltaTime;
        motor_prev_position = motor_current_position;
}
void CalculatePID()
{
        currentTime = HAL_GetTick();
        deltaTime = (float)((float)(currentTime) - (float)(prevTime)) / 1000.0f;
        _time = _time + deltaTime;
 if (!isnanf(motor_current_speed) && !isinf(motor_current_speed))
 {
  fs = 0.1f * motor_current_speed + (1.0f - 0.1f) * fs;

 }

        errorValue = GetPostionAtTime() - PulsestoDegrees(motor_current_position);

        derivative = (errorValue - prevErrorValue) / deltaTime;

        integral = integral + errorValue * deltaTime;

        controlSignal = data.kp * errorValue + data.ki * integral + data.kd * derivative;

        char msg[255];

        sprintf(msg,    "%d    %d    %d    %d\n\0",    (int)GetVelocityAtTime(),    (int)fs,
(int)PulsestoDegrees(motor_current_position), (int)GetPostionAtTime());

        HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);

        prevTime = currentTime;

        prevErrorValue = errorValue;

}

void DriveMotor()

{

        if(controlSignal < 0)

        {

                motorDirection = ClockWise;

        }

        else

        {

                motorDirection = CounterClockWise;

        }



        if(prevMotorDirection != motorDirection)

        {

                switch(motorDirection)
```

```
{
case ClockWise:
{
  PWM_CurrentChannel = PWM_CLOCKWISE_CHANNEL;
  __HAL_TIM_SET_COMPARE(&htim2,
PWM_COUNTER_CLOCKWISE_CHANNEL, 0);
  __HAL_TIM_SET_COMPARE(&htim2, PWM_CLOCKWISE_CHANNEL, 0);
  //delay here
  //HAL_Delay(PWM_DEADTIME_DELAY);
  __HAL_TIM_SET_COMPARE(&htim2, PWM_CurrentChannel, 0);
  HAL_GPIO_WritePin(GPIOB,          GPIO_COUNTER_CLOCKWISE_PIN,
GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_CLOCKWISE_PIN, GPIO_PIN_RESET);

}break;
case CounterClockWise:
{
  PWM_CurrentChannel = PWM_COUNTER_CLOCKWISE_CHANNEL;
  __HAL_TIM_SET_COMPARE(&htim2, PWM_CLOCKWISE_CHANNEL, 0);
  __HAL_TIM_SET_COMPARE(&htim2,
PWM_COUNTER_CLOCKWISE_CHANNEL, 0);
  //delay here
 // HAL_Delay(PWM_DEADTIME_DELAY);
  __HAL_TIM_SET_COMPARE(&htim2, PWM_CurrentChannel, 0);
  HAL_GPIO_WritePin(GPIOB, GPIO_CLOCKWISE_PIN, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB,          GPIO_COUNTER_CLOCKWISE_PIN,
GPIO_PIN_RESET);
}break;
}
```

```
        }
        //setting PWM value
        PWM_countingDutyCycle = (uint32_t)fabs(controlSignal);
        if(PWM_countingDutyCycle > 205)
        {
                PWM_countingDutyCycle = 205;
        }
  __HAL_TIM_SET_COMPARE(&htim2, PWM_CurrentChannel, PWM_countingDutyCycle);


        prevMotorDirection = motorDirection;
}
/* USER CODE END PFP */
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
        HAL_UART_Receive_IT(&huart1, rx_buffer, 1);
        if(rx_buffer[0] == '\n')
        {
                int p, s, a, d;
                sscanf(buffer, "%d %d %d %d", &p, &s, &a, &d);
                memset(buffer, 0, 256);
                data.position = p;
                data.speed = s;
                data.acceleration = a;
                data.deceleration = d;
                bufferIndex = 0;
                _time = 0.0f;
                integral = 0.0f;
                controlSignal = 0.0f;
                CalculateMotionProfile();
                return;
        }
        buffer[bufferIndex] = rx_buffer[0];
        bufferIndex++;
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if(GPIO_Pin == GPIO_PIN_6)
  {
         int32_t inc = 0;
         if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_7))
         {
                 inc = 1;
         }
         else
         {
                 inc = -1;
         }
         motor_current_position += (float)inc;
  }
}
```

```
/* USER CODE END 0 */

/**

  * @brief  The application entry point.

  * @retval int

  */

int main(void)

{

  /* USER CODE BEGIN 1 */



  /* USER CODE END 1 */



  /* MCU Configuration--------------------------------------------------------*/



  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */

  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */

  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */

  MX_GPIO_Init();

  MX_USART1_UART_Init();

  MX_TIM2_Init();

  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
  HAL_UART_Receive_IT(&huart1, rx_buffer, 1);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
  HAL_TIM_Base_Start_IT(&htim3);
  data.gearRatio = 3249.0f / 121.0f;
  data.encoderPulses = 500.0f;
  data.kp = 2.00f;
  data.ki = 0.1f;
  data.kd = 0.1f;
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
		CalculateSpeed();
		CalculatePID();
		DriveMotor();
  }
  /* USER CODE END 3 */
}
/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL16;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)

  {

    Error_Handler();

  }

}

/**

  * @brief TIM2 Initialization Function

  * @param None

  * @retval None

  */

static void MX_TIM2_Init(void)

{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */

  htim2.Instance = TIM2;

  htim2.Init.Prescaler = 0;

  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;

  htim2.Init.Period = 255;

  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;

  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
```

```
if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
{
  Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
{
  Error_Handler();
}
if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
{
  Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
  Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 125;
sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
  Error_Handler();
  }
  sConfigOC.Pulse = 0;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */
  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);
}
/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{
  /* USER CODE BEGIN TIM3_Init 0 */
  /* USER CODE END TIM3_Init 0 */
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  /* USER CODE BEGIN TIM3_Init 1 */
  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 128 - 1;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 65535 - 1;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */
  /* USER CODE END TIM3_Init 2 */
}
/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
```

```
static void MX_USART1_UART_Init(void)
{
  /* USER CODE BEGIN USART1_Init 0 */
  /* USER CODE END USART1_Init 0 */
  /* USER CODE BEGIN USART1_Init 1 */
  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */
  /* USER CODE END USART1_Init 2 */
}
/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
```

```
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12|GPIO_PIN_13, GPIO_PIN_RESET);
  /*Configure GPIO pins : PB12 PB13 */
  GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  /*Configure GPIO pin : PB6 */
  GPIO_InitStruct.Pin = GPIO_PIN_6;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  /*Configure GPIO pin : PB7 */
  GPIO_InitStruct.Pin = GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
/* USER CODE BEGIN 4 */
/* USER CODE END 4 */
/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name

  * @param  line: assert_param error line source number

  * @retval None

  */

void assert_failed(uint8_t *file, uint32_t line)

{

  /* USER CODE BEGIN 6 */

  /* User can add his own implementation to report the file name and line number,

     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

  /* USER CODE END 6 */

}

#endif /* USE_FULL_ASSERT */
```