Differential Drive Motor Control for 3D-printed Robots

In service of the GEMS Erasmus Project: Graceful Equalising of Mechatronics Students

EE3L11: Bachelor Afstudeer Project

Robin Appel
Joshua van der Geize

Delft University of Technology

TU Delft

Differential Drive Motor Control for 3D-printed Robots

In service of the GEMS Erasmus Project: Graceful Equalising of Mechatronics Students

by

Robin Appel
Joshua van der Geize

to obtain the degree of Bachelor of Science
at the Delft University of Technology,

Student number:	5454042 (R. Appel)	
	5462290 (J. van der Geize)	
Project duration:	April 21, 2025 – June 27, 2025	
Thesis committee:	Dr. J. Dong,	TU Delft, supervisor
	ir. S. Yadav,	TU Delft, supervisor
	prof. dr. P. French	TU Delft, jury
	dr. ir. F. A. Muñoz	TU Delft, jury

This thesis is confidential and cannot be made public until December 31, 2025.

TU Delft

Abstract

In this thesis, a motor controller was designed, implemented on a Printed Circuit Board (PCB), and validated. The motor controller was designed to drive a motor as part of a differential drive configuration, it does this based on external signals which are generated by a Finite State Machine (FSM). The PCB was specifically designed to implement the following features: driving the motor, communication with other subsystems via a CAN interface, determining the current draw of the motor, and measuring the position and speed of the wheels. A Proportional-Integral (PI) feedback controller was designed and tuned to determine the duty cycle of the motors based on speed measurements. To validate the design, the PI controller response was measured and documented, and compared against the theoretical model. The PI controller proved to be an effective solution and was congruent with the theoretical model, albeit with some non-linearities which came with the physical implementation. Furthermore, the CAN communication protocol was verified by measuring a handshaking signal.

Contents

Abstract i

1 Introduction 1

2 Programme of Requirements 3
2.1 Overarching Programme of Requirements . . . 3
2.2 Unit Programme of Requirements . . . 3
2.2.1 Functional Requirements . . . 3
2.2.2 Non-functional Requirements . . . 4

3 Background Knowledge and Existing Designs 6
3.1 System Architecture Overview . . . 6
3.2 H-Bridge . . . 6
3.3 Kinematic Model . . . 8
3.4 Motor Drive . . . 9
3.4.1 Flux Control . . . 9
3.4.2 Armature Control . . . 10
3.4.3 Voltage Control . . . 10
3.4.4 Pulse Width Modulation (PWM) . . . 10
3.5 Current Measurement . . . 10
3.5.1 Shunt Resistor . . . 10
3.5.2 Hall Effect Sensor . . . 11

4 Design 12
4.1 ESP32-S3: Microcontroller . . . 12
4.2 MOT2N: Motor . . . 13
4.3 DRV8411A: H-Bridge . . . 13
4.4 ACS71240: Current Sensor . . . 14
4.5 Control System Representation . . . 15
4.6 Actuator: Motor Drive (MCPWM) . . . 15
4.7 Feedback: Quadrature Encoder . . . 17
4.7.1 Hall Sensors . . . 17
4.7.2 Quadrature Encoding . . . 17
4.8 PI Control . . . 19
4.8.1 Modelling the System: Controller, Plant, Sensor . . . 19
4.8.2 Closed-loop Transfer Function . . . 20
4.8.3 Bode analysis . . . 21
4.9 Position Tracking . . . 23

5 Testing Protocol 24
5.1 Testing Setup . . . 24
5.2 Features to be Tested . . . 25
5.2.1 Motor Functionality . . . 25
5.2.2 PI Controller . . . 25
5.2.3 CAN Communication . . . 26

6 Implementation and Results 27
6.1 Motor Functionality . . . 27
6.2 PI Controller: Initial Implementation . . . 27
6.3 PI Controller: Final Tuning . . . 29
6.4 CAN Communication . . . 31

7 Discussion and Conclusion **32**
7.1 Difficulties . . . 32
7.2 Missed Targets . . . 32
7.3 Improving GEMS . . . 33
7.4 Design Alterations . . . 33

References **34**

A Source Code **36**

B Further Derivations **40**
B.1 Plant Transfer Function . . . 40

C Schematic **41**

1

Introduction

The need for spatially distributed data arises when information in an area cannot be adequately represented by a single point of measurement. In many applications — from robotics and autonomous vehicles to Urban structural integrity monitoring — understanding variations in data based on location is crucial for making accurate decisions and optimizing performance. Spatially distributed data lets us capture a complete picture of a system's condition, whether it's the stress along a bridge, the temperature within a data centre, or the pressure across a pipeline. The dense and distributed nature of the acquired information enables more sophisticated control strategies, helps identify problems before they become points of failure, and guides designers toward smarter, more robust solutions.

Currently, there are limited solutions available for collecting spatially distributed data. Fixed sensor stations often fail to capture the spatial variability of environmental parameters, and randomly distributed sensor systems often fail to achieve complete coverage of a surveillance region [17]. The proposed design offers a low cost, scalable, and flexible solution in the form of a mobile sensing station. The goal of this project is to use existing GEMS architecture, as can be seen in figure 1.1, in order to create a mobile sensor station which is capable of collecting spatially distributed data, whilst simultaneously validating and improving upon the existing GEMS architecture [19].

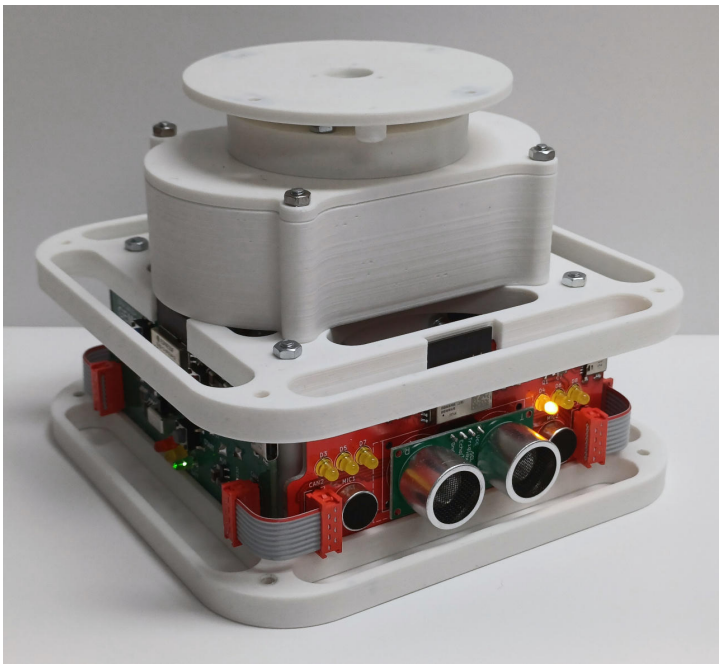

Figure 1.1: GEMS architecture

The GEMS architecture is a 3D printed drivetrain which has four different modules: wireless communication, power supply, sensor node and the motor controller. Each of these modules work in tandem in order to realize the final goal. By combining two of these drivetrain modules a mobile robot can be created. This can be seen in figure 1.2.

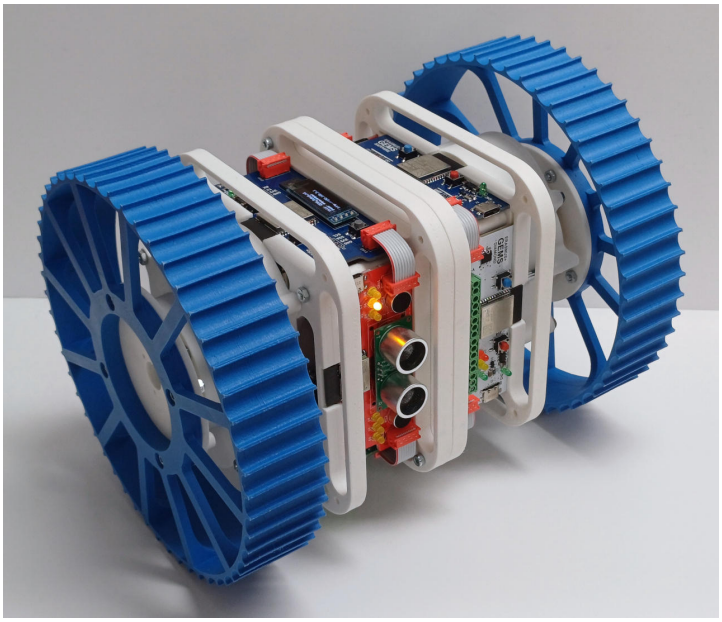

Figure 1.2: Mobile robot using GEMS architecture

This thesis focuses on the design, implementation, and evaluation of a motor control module. The main objective is to provide a solution that can regulate motor speed and position while offering robustness against disturbances and parameter variations. To achieve this, a controller is integrated with sensors, and a microcontroller platform.

The motor control module presented in this thesis is designed to be versatile, cost-effective, and easy to implement. It strives to combine high performance with simplicity and robustness, making it a valuable tool for both education and industry. This work covers the theoretical background, hardware design, controller algorithm, and implementation details, alongside a comprehensive evaluation of its performance through simulation and experimental testing.

2

Programme of Requirements

2.1. Overarching Programme of Requirements

This section contains the programme of requirements as defined for the greater whole, i.e. the entire robot with all other involved units working together.

1. Functional requirements
   (a) **The system must start in a stopped state when power is initially supplied.**
   (b) **The system drives the motors with a 3-state protocol**
   (c) **The inter-module communication of the system must be implemented through the CAN protocol**
   (d) **The system must be able to communicate wirelessly with a host device**
   (e) **The input voltage provided by either the battery or USB-C must be between 1.5 V and 6 V, while short bursts are allowed up to 7 V.**
   (f) **Input voltage must be converted to 3.3 V**
   (g) **Overcurrent protection should kick in when the overcurrent threshold has been reached or exceeded.**
   (h) **Over-temperature protection should kick in when a battery temperature of 60 degrees Celsius has been measured.**
2. Non-functional requirements
   (a) **Data will be stored in CSV files**
   (b) **The robot should be able to operate for at least half an hour on battery power**
   (c) **The user must be able to turn on the robot by pressing the enable / start button on the power supply PCB whenever sufficiently charged batteries are correctly placed in the battery holder.**
   (d) **The user must be able to flip a physical switch in order to connect or disconnect the battery.**

2.2. Unit Programme of Requirements

The following is a programme of requirements of the motor control unit. The goal of the motor control unit is to convert input signals to robot movement by driving the motors in a differential-drive configuration.

2.2.1. Functional Requirements

The functional requirements specify how the system must behave. This includes the tasks and functions the system must be able to execute. The following is a list of such requirements.

1. **The system must start in a stopped state when power is initially supplied.**

   It would be undesirable for the robot to start moving at random the moment it is turned on. Therefore it must wait for input of some kind before it starts to move. The default mode of the robot is ”stopped”.

2. **The system must respond to input received over CAN**

   In order to facilitate cohesion with other modules mounted on the robot, the module will drive the motor based on input received over the CAN communication channel.

3. **The system must calibrate its own wheel position based on a mounted Hall sensor**

   The design must include a mounted Hall sensor to detect a known reference point once per full rotation. Given possible noise factors such as gear backlash, encoder noise, or sensor glitches, the Hall sensor provides an error correction mechanism over longer distances to keep the measurement error within 5% of the wheel circumference. The system must calibrate its own wheel position based on a mounted Hall sensor

4. **The system drives the motors with a 3-state protocol**

   Since one module steers one wheel, and given that the robot uses a differential-drive configuration, each wheel needs only 3 states to facilitate any kind of movement in the 2D plane. At any given time, the motor is driven in one of the following states:

   - Forward
   - Backward
   - Stop

5. **The motor must be able to track a reference speed in m/s using PI control.**

   The motor control module shall receive a reference speed value (in meters per second) along with a direction indicator (forward or backward). A PI controller will be used to minimize the error between the commanded and actual wheel speed. The control system must maintain the *measured speed* within ± 0.1 m/s of the reference under nominal operating conditions.

6. **The system must respond independently of CAN signals in emergency situations.**

   When the front-facing distance sensors detect an obstacle within a time-to-collision (TTC) of less than 500 milliseconds , the motor control module shall override incoming CAN commands and execute an immediate stop. This local safety response must take effect within 200 milliseconds of detection to prevent collision. Normal CAN-based operation may resume only after the obstacle has been cleared for at least 500 milliseconds and a reset condition is satisfied.

7. **The system must implement a Stop-and-Go protocol for pointwise measurements**

   The general purpose of the robot is to generate spatial sensor data, i.e. sensor readouts associated with spatial coordinates. It is vital that the coordinates of the measurements are accurate. To facilitate this, the robot must temporarily come to a halt when sensor measurements are being processed. When a measurement trigger is received, the robot must come to a complete stop within 1 cm Euclidean distance of the desired coordinate. The system will remain stationary until sensor readouts have been stored, whereafter it will move to the next designated measurement coordinate.

8. **The system must track its own position in a static reference frame**

   After defining a reference point (0,0), the system must keep track of its movements with respect to this point, so that sensor data can be associated with specific coordinates.

2.2.2. Non-functional Requirements

1. **The system must include overcurrent protection to ensure hardware safety**

   The overcurrent is defined at 4.0 A, in alignment with the stall current of the motor which is 4.11 A.

2. **The system is mathematically controllable and observable**

   Reality will introduce certain non-linearities which are impractical to model. Nevertheless, the system should be built on a sound theoretical base. The linearized system obeys the Kalman controllability and observability criteria: if the system has ν state variables, then both the row rank of the controllability matrix and the column rank of the observability matrix have to be equal to ν.
3. **The PI controller shall achieve a rise time of at most 1000 milliseconds.**
4. **The PI controller shall achieve a settling time of at most 2000 milliseconds.**
5. **The PI controller shall limit overshoot to less than 1% of the reference speed.**
6. **The PI controller shall achieve a steady-state error of less than 5%.**
7. **The static reference frame localisation error is less than 1 cm**

3

Background Knowledge and Existing Designs

3.1. System Architecture Overview

Figure 3.1 shows a block diagram of a typical DC drive system.

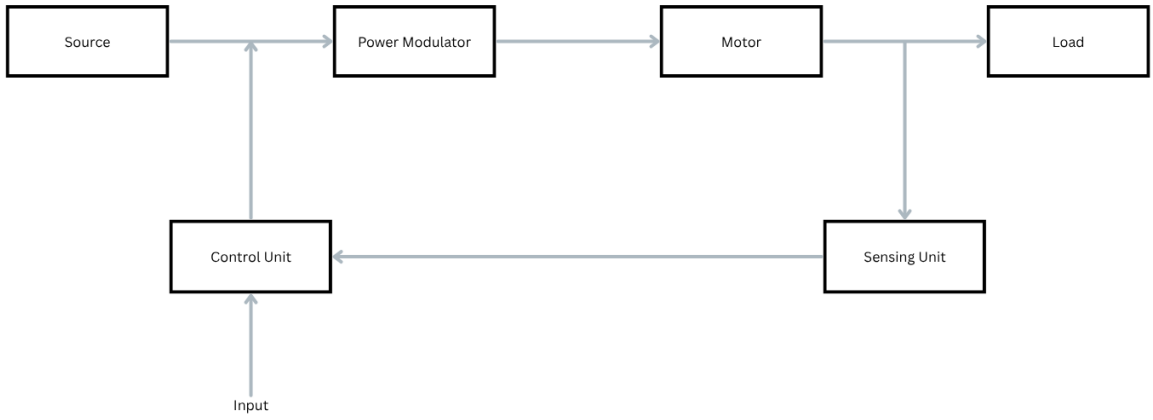

Figure 3.1: Block diagram of a DC drive system

Any DC drive system needs a source. This is the power source which is needed to power the entire system. Subsequently, the power modulator regulates the power flow from the source to the motor, to enable the motor to develop the torque and speed characteristics required by the load. A sensing unit is then used to measure certain variables which can later be used by the control unit. Typically, these variables are motor speed and rotor position. A control unit takes in the desired output and the measured output in order to determine the required input to get the desired output [13]. Finally, the load is typically a mechanical component which is designed to accomplish a specific task, such as a wheel

3.2. H-Bridge

In order to get bidirectional capability in a DC motor the polarity of the applied voltage must be reversed. The simplest method to achieve this is by implementing an H-bridge. An H-bridge switches the polarity of a voltage applied to a load [1]. Thus, making it ideal for controlling the polarity and thus the direction of a DC motor. Figure 3.2 shows the H-bridge topology.

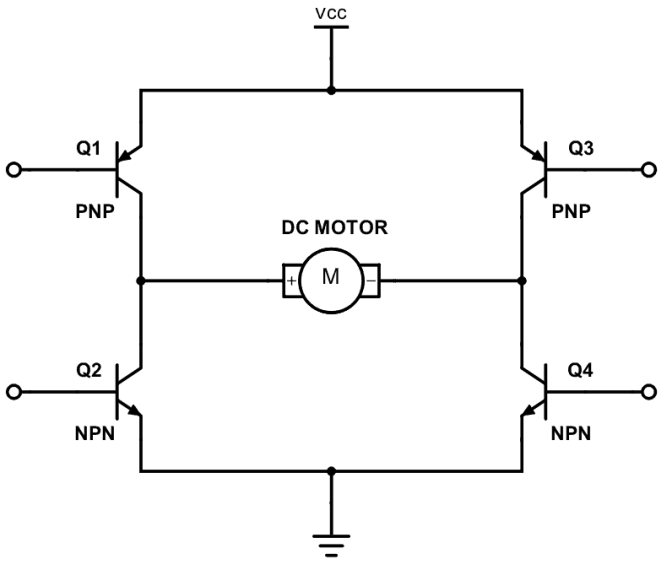

Figure 3.2: H-bridge [3]

3.3. Kinematic Model

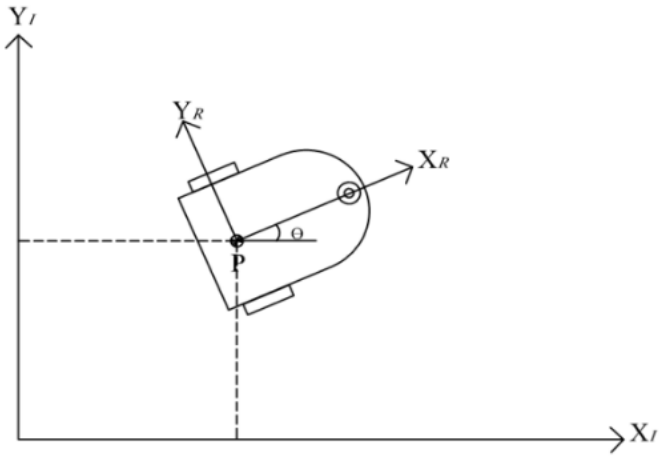

Figure 3.3: Coordinate definitions in inertial and relative reference frames [9]

For effective control of the differential drive robot, it is necessary to utilize a kinematic model. First, the state configuration is defined as the vector $q = \begin{bmatrix} x & y & \theta \end{bmatrix}$, where x and y represent the coordinates of the centre point of the robot in an **inertial reference frame**, and θ represents the angle the robot makes with the x-axis. See Figure 3.3. For these sorts of robots, it is conventional to assume that the robot cannot slide, and therefore does not move laterally. This can be expressed by the non-holonomic constraint: the velocity vector is perpendicular to the lateral vector:

$$\begin{bmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{bmatrix} \begin{bmatrix} \sin\theta \\ -\cos\theta \\ 0 \end{bmatrix} = 0 \tag{3.1}$$

Additionally, equations can be derived which govern the motion of the robot in relation to the angular velocities of the left and right motor ω_L and ω_R respectively. The linear velocity is related to the angular velocity and wheel diameter R_W according to Equation 3.2 and Equation 3.3.

$$v_R = \omega_R R_W \tag{3.2}$$

$$v_L = \omega_L R_W \tag{3.3}$$

Using these equations, and that the velocity of the middle point of the wheel axis is $\frac{1}{2}(v_R + v_L)$ [10], the magnitude of the linear velocity of the middle point of the wheel axis can be written in terms of the angular velocities of both wheels (Equation 3.4).

$$v = \frac{(\omega_R + \omega_L)R}{2} \tag{3.4}$$

Given the angle θ with respect to the inertial reference frame, $\dot{x}$ and $\dot{y}$ can be computed as in eq 3.5.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \frac{(\omega_R + \omega_L)R}{2} \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \tag{3.5}$$

Lastly, the change in θ can be derived as well. This process is a bit more involved, as it requires relating the movement to the Instantaneous Center of Curvature (ICC). Equation 3.6 shows the relation of $\dot{\theta}$ to ω_R and ω_L with the wheel axis length l. [9]

$$\dot{\theta} = \frac{(\omega_R - \omega_L)R}{l} \tag{3.6}$$

Equations 3.5 and 3.6 can be combined to form the total (non-linear) kinematic model of a non-holonomic differential drive robot, given in Equation 3.7.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R_W}{2}\cos(\theta) & \frac{R_W}{2}\cos(\theta) \\ \frac{R_W}{2}\sin(\theta) & \frac{R_W}{2}\sin(\theta) \\ \frac{R_W}{l} & -\frac{R_W}{l} \end{bmatrix} \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} \tag{3.7}$$

3.4. Motor Drive

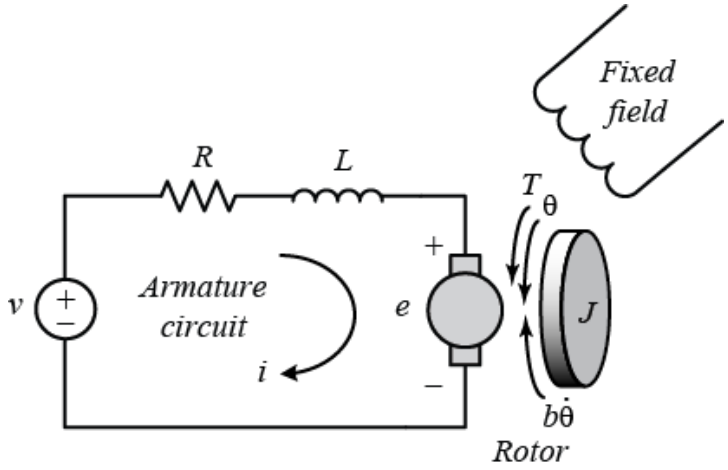

Figure 3.4: Armature current driven DC motor model.

Figure 3.4 depicts an equivalent circuit of a DC motor. The equivalent circuit of a DC motor models its electrical behaviour by representing its main components with electrical elements. At its core, a DC motor consists of an armature resistance (R) in series with an armature emf or back emf (e), which is proportional to its speed. This back emf reflects the voltage generated by the rotation of the armature in a magnetic field. Additionally, there is a field circuit typically represented by a field resistance (R_{field}) in series with a field voltage, which controls the flux and, consequently, the motor's torque [2]. The armature current is influenced by the voltage applied across the armature and the voltage drop due to its resistance, while the field current is governed by its own voltage and resistance. Overall, this equivalent circuit forms a convenient way to analyse the motor's performance, predict its response under different conditions, and design appropriate control strategies for desirable operation [18].

There are three ways of controlling the speed of a DC motor. These being: flux control, armature control and voltage control. This can be seen in the following formulas which depict the speed of a DC motor [22]:

$$N \propto \frac{E_b}{\phi} \tag{3.8}$$

$$N = K\frac{V - I_a R}{\phi} \tag{3.9}$$

3.4.1. Flux Control

As seen in equation 3.8 a change in flux will result in a change in motor speed. In order to achieve this a variable resistor can be placed in series with the shunt field winding. This reduces the shunt field current and therefore the flux [22].

3.4.2. Armature Control
By varying the voltage across the armature the back e.m.f can be changed and therefore the motor speed can be changed. This can be done by inserting a variable resistor in series with the armature. The voltage drop in the controller resistance results in a reduction in the back e.m.f [22].

3.4.3. Voltage Control
For voltage control the voltage source supplying the field current is different from the one supplying the armature. By doing this the disadvantages of poor speed regulation and low efficiency can be avoided [22].

3.4.4. Pulse Width Modulation (PWM)
The simplest method of controlling the speed of the motor is through altering the driving voltage. However, if simple voltage regulation is utilized there will be a loss of power. Therefore, pulse width modulation (PWM) is utilized. The rapid rising and falling edges ensure that all devices are turned on and off as fast as possible in order to minimize switching losses [14]. By altering the duty cycle of a PWM signal the motor speed can be controlled as the average voltage is affected as shown in Equation 3.11. The duty cycle can be calculated with the formula given by Equation 3.10.

$$D = \frac{t_{ON}}{t_{ON} + t_{OFF}} \tag{3.10}$$

$$V_{avg} = V * D \tag{3.11}$$

From equation 3.10 it can be seen that altering t_{ON} will lead to a change in the motor speed. Therefore, if t_{ON} is at it's maximum then the motor speed will also be at it's maximum [16]. When it comes to the frequency of the PWM signal it is heavily dependent on the motor. The frequency must be higher than the time constant, τ, of the motor.

3.5. Current Measurement
There are two main methods of measuring the current flow through an area in a circuit. These being the shunt resistor and Hall effect sensor. Both methods have their own inherent advantages and disadvantages,

3.5.1. Shunt Resistor
A shunt resistor is a simple but effective way to measure current by exploiting Ohm's Law. When current flows through the resistor, it generates a small voltage drop that is directly proportional to the current's magnitude — following the formula $V = IR$. Because the resistor's resistance is precisely known and typically very small, this voltage drop can be safely measured without significantly affecting the main current flow. This voltage is then fed into an analog-to-digital converter or a monitoring circuit, allowing the current to be accurately calculated in real time [15].

Advantages [7]:

- Simple and cost-effective — shunts are low-cost components that are easy to implement.
- Accurate and reliable — with proper calibration, a shunt resistor can provide a high degree of accuracy in current sensing.
- Small size and low profile — their compact form makes them convenient for space-limited applications.
- Linear response — the voltage drop across the resistor is directly proportional to current, simplifying the mathematical relation.

Disadvantages [20]:

- Power dissipation — the resistor converts some power into heat, which can affect both its own resistance and nearby components.
- Insertion resistance — adding resistance to the main current path can produce a small voltage drop and reduce the voltage available to the load.

- Heating effects — high currents can cause significant heating, affecting resistance stability and adding error to the measurement
- Low-side configuration — shunts are often placed on the low side (ground path), which can be less convenient for some high-side or high voltage applications.
- Possible interference — the resistor's voltage drop may be influenced by electrical noise, requiring careful filtering or shielding in sensitive circuits.

3.5.2. Hall Effect Sensor

Hall effect sensors measure current by detecting the magnetic field that flows around a current-carrying conductor. When current passes through a wire or traces nearby, it generates a magnetic field proportional to its magnitude — a phenomenon described by Ampère's circuital law. The Hall sensor, typically placed close to or encircling the current path, converts this magnetic field into a small voltage signal. This output is directly related to the current's strength, allowing for accurate and non-invasive current measurements. Because Hall effect sensors provide electric isolation and do not interrupt the flow of current, they are especially useful in high-current applications, motor control, and battery management systems where safety and robustness are a top priority.

Advantages [8]:

- Non-invasive and isolated — Hall sensors do not interrupt or connect directly to the current path, which helps avoid additional resistance and power dissipation.
- Safe and versatile — they provide electrical isolation, making them suitable for high-current, high-voltage, or hazardous applications.
- AC and DC current sensing — Hall sensors can measure both direct current (DC) and alternating current (AC) with high accuracy.
- Fast response and low latency — their response time is typically faster than many other methods, allowing for real-time current monitoring.
- Robust and reliable — Hall sensors are less prone to wear and are typically reliable under a range of environmental conditions.

Disadvantages [21]:

- Higher cost — Hall sensors are usually more expensive than simple shunt resistor solutions.
- Sensitivity to external fields — nearby magnets or strong electromagnetic fields can sometimes affect their accuracy if proper shielding isn't in place
- Temperature drift — their output may vary with temperature, requiring additional calibration or compensation.
- General complexity — they typically require additional conditioning circuits and power, adding to the overall complexity of the design.

4

Design

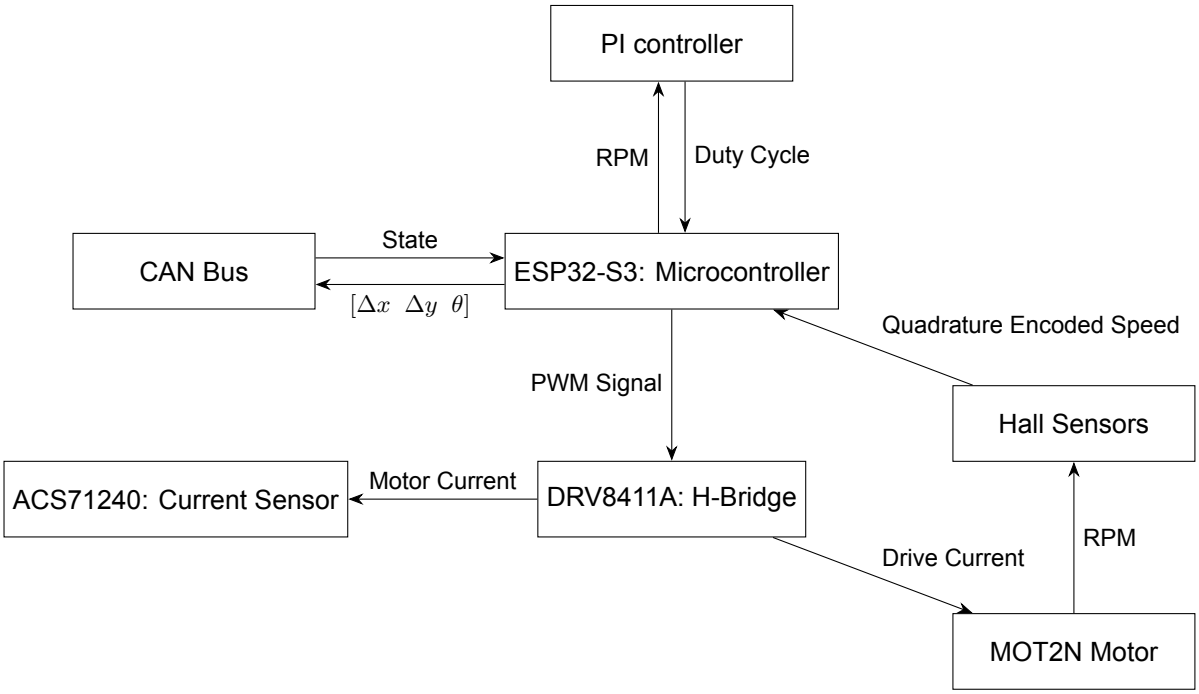

Figure 4.1: High-level block diagram of the system

Figure 4.1 shows a high-level block diagram of the system. At the heart of the system is the **microcontroller**, its function is to run calculations with the given inputs and subsequently generate control signals which define the system's behaviour. This chapter will discuss the seven main components displayed and how they functionally work together to implement the desired system behaviour.

4.1. ESP32-S3: Microcontroller

The ESP32-S3-Wroom-1 [4] was chosen to serve as the microcontroller of the system mainly for its affordable price point, and to maintain a homogeneous design as the external Wi-Fi & sensing modules also make use of the same chip for its integrated Wi-Fi and Bluetooth capabilities. Figure 4.2 shows how the microcontroller is integrated on the PCB with pin numbers and labels. It connects to the CAN bus via $\mathrm{CAN\_RX}$, and $\mathrm{CAN\_TX}$, the motor driver with $\mathrm{PWM\_A}$, $\mathrm{PWM\_B}$, and $\mathrm{MOT\_I}$, and the Hall sensors via $\mathrm{E\_A}$, $\mathrm{E\_B}$, and $\mathrm{E\_C}$.

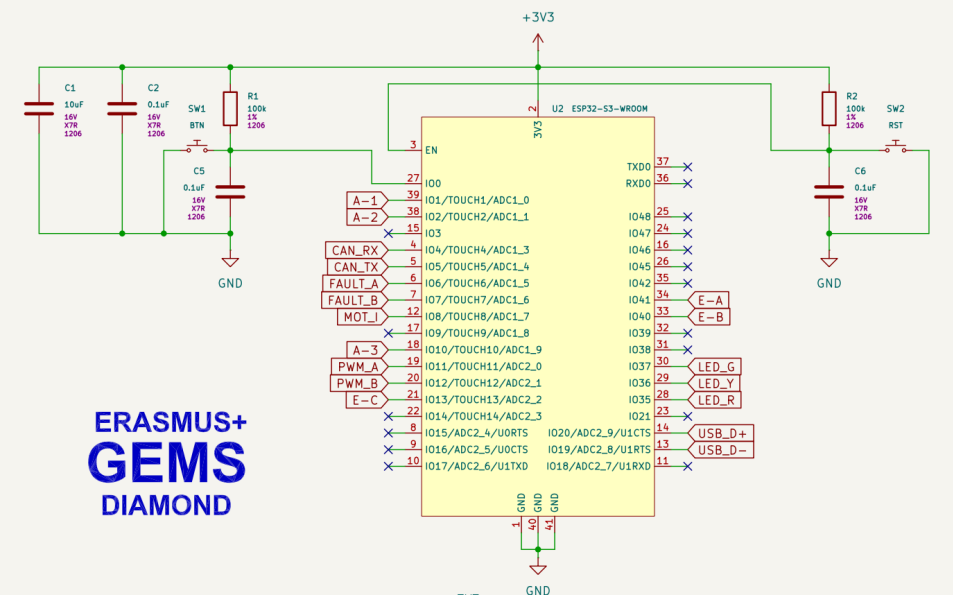

Figure 4.2: Microcontroller schematic

4.2. MOT2N: Motor

The motor that will be used is the Velleman MOT2N. It is a compact brushed, high speed DC motor that takes an input voltage between $2.5 - 6\ V$. This is ideal as the motor is supplied by the battery, which has a voltage level between $3.2 - 3.7\ V$. Additionally, the motor is very affordable. In order to be able to accurately model the motor its parameters must be determined.

First the motor resistance will be found. This is done by holding the rotor still and applying a low voltage. The current draw can then be measured and with it the motor resistance can be calculated through Ohm's law, $V = IR$. At $1.5\ V$ the motor drew $1.57\ A$. Therefore, the motor resistance is $0.955\ \Omega$.

Next the no load current will be determined. In order to measure the no load current the rotor should spin freely with no load. Meanwhile, the voltage will be ramped up to the maximum rated voltage of $6\ V$. The then measured current will be the no load current. This was measured to be $0.22\ A$.

Knowing both the no load current and the motor resistance the motor velocity constant can be calculated with Equation 4.1. Doing so gives: $K_v = 2504.36 \frac{RPM}{V} = 262.26 \frac{rad}{s} \frac{1}{V}$.

$$K_v = \frac{RPM}{V - i_o R} \tag{4.1}$$

The motor velocity constant allows for the calculation of the motor torque constant, K_T, with Equation 4.2. This provides a motor torque constant of: $K_T = 3.81 * 10^{-3} \frac{Nm}{A}$.

$$K_T = \frac{\tau}{I_a} = \frac{1}{K_{v(SI)}} \tag{4.2}$$

Subsequently, the motor constant, K_m, can be calculated by using the motor torque constant with Equation 4.3. This gives a motor constant of: $K_m = 3.90 * 10^{-3} \frac{Nm}{\sqrt{W}}$.

$$K_m = \frac{K_T}{\sqrt{R}} \tag{4.3}$$

4.3. DRV8411A: H-Bridge

The chosen H-bridge is the DRV8411A dual H-bridge. A dual H-bridge is used to take advantage of the inherent properties of an H-bridge: the current carrying capacity of an H-bridge is usually limited

by the power dissipation, $I^2R_{DS(ON)}$. By connecting $\mathrm{IN1}$ and $\mathrm{IN2}$ together, the two internal H-bridges operate in parallel. As a result, $R_{DS(ON)}$ is reduced to half of its original value, which in turn halves the power dissipation [12]. This configuration allows a single DC motor to draw more current than if a single H-bridge is used [6].

The DRV8411A also has built-in overcurrent protection that triggers when the motor draws more than $4\ A$. This is ideal as the stall current of the motor is $4.11\ A$.

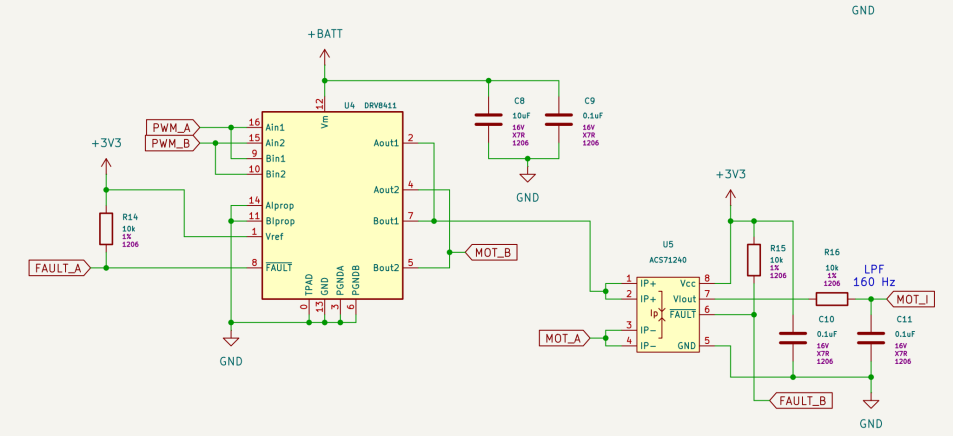

Figure 4.3: H-bridge and current sensor schematic

Figure 4.3 shows how the H-bridge is connected on the PCB. Two PWM signals generated by the microcontroller serve as inputs to the parallel H-bridge. The H-bridge produces two outputs: one that is directly connected to the motor pin, MOT_B, and another that passes through the current sensor before connecting to the other motor pin, MOT_A.

4.4. ACS71240: Current Sensor

The current sensor measures the current draw of the motor in real time for monitoring and protection purposes. The ACS71240 was chosen as the current sensor as it offers accurate, real-time current sensing with built-in electrical isolation. Unlike simple shunt resistors, the ACS71240 does not require direct electrical contact between the sensing circuit and the high-power motor line, which eliminates power loss because there is no direct interference by the sensor [15]. Its bidirectional sensing capability allows for the detection of both forward and reverse currents, which is important for determining motor direction and potential fault conditions. Additionally, the ACS71240's analog output is directly compatible with the built-in ADC of the ESP32-S3 [11]. Figure 4.3 shows how it is implemented on the PCB: it measures the H-bridge output on pins $\mathrm{Aout1}$ and $\mathrm{Bout1}$ and provides an analog signal to the microcontroller.

4.5. Control System Representation

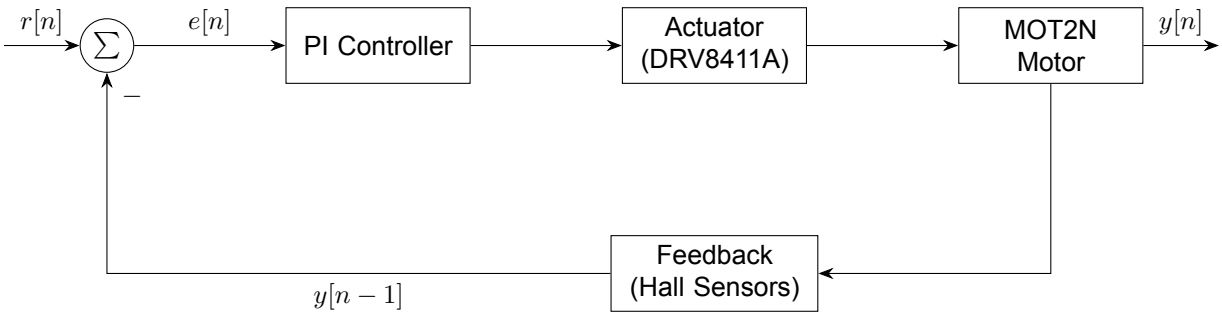

Figure 4.4: High-level representation of the control system scheme

Figure 4.4 shows a high-level representation of the control system. $r[n]$ represents the reference given by the microcontroller in RPM. $e[n]$ represents the error, which is the signed difference between the reference and measured signal $y[n-1]$. The PI controller indirectly controls the actuator by outputting a duty cycle which is converted to a PWM signal in the microcontroller as can be seen in Figure 4.1. The actuator converts the PWM signal to an analog voltage which drives the MOT2N motor. As can also be seen in Figure 4.1, the motor RPM is measured by a Hall sensor setup, which will be elaborated on in section 4.7.

4.6. Actuator: Motor Drive (MCPWM)

The actuator consists of two parts: first, the PI controller output needs to be converted to a PWM signal, secondly, the PWM signal is fed to the DRV8411A H-bridge chip which drives the motor. The ESP32's dedicated peripheral MCPWM library was chosen as the foundation of the PWM signal generation logic. The library offers built-in compatibility with the ESP32 architecture, alongside useful additional features. These include fault handling, centre-aligned complementary PWM generation, and dead-time control [5]. These features allow for more precise control out of the box, compared to more basic PWM solutions like Arduino's built-in *analogWrite* function, or the alternative ESP32 peripheral LEDC library.

Centre-aligned PWM, also known as symmetric PWM has a number of advantages compared to edge-aligned PWM because it reduces transient harmonics thereby lowering voltage and current ripple, and consequently, torque ripple. Complementary PWM involves two inverse switching signals for the high and low side switches on each half-bridge. Dead-time insertion is managed inside the DRV8411A chip to ensure that two connected high and low switches are never turned on simultaneously, preventing a short circuit.

Referring to the schematic (Figure 4.2), $\mathrm{ADC2\_0}$ and $\mathrm{ADC2\_1}$ corresponding to pins $\mathrm{IO11}$ and $\mathrm{IO12}$ output the PWM signals generated by the MCPWM peripheral in the microcontroller. These are fed to the DRV8411A half-bridge to drive the motor.

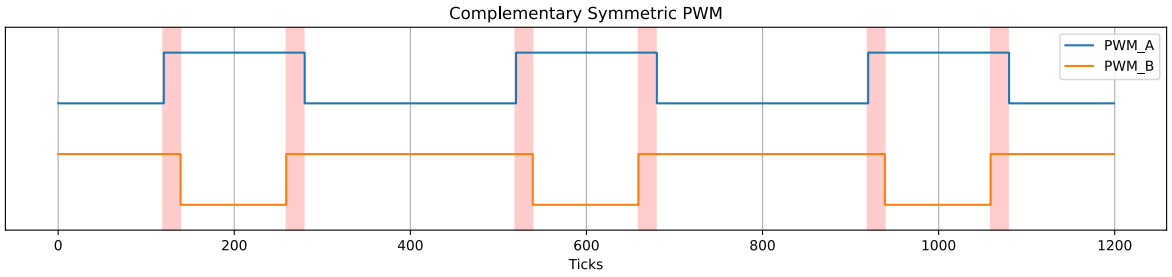

Figure 4.5: The figure shows an example of complementary symmetric PWM. Dead time is inserted to prevent simultaneous switching on the high and low side of the bridge, highlighted in red

Four key components define the PWM signal generation scheme. The first is the **timer**, a hardware

counter that increments or decrements every $T_{timer} = 100ns$ nanoseconds. As a result, the timer frequency is defined as the inverse of the period at $f_{timer} = 10MHz$. The second component is the **generator**, a component which sets the connected pin to high or low. The final two components are the **comparator**, which triggers an *event* when a certain condition is met by the timer (e.g. $\text{timer\_count} > 50$ and the **operator** which handles the event trigger and controls the generator based on a defined logic sequence. At the end of one PWM cycle, the timer count is reset to 0. The timer reset condition determines the period of one PWM cycle (Equation 4.4).

$$T_{cycle} = T_{timer} \times n_{reset} \tag{4.4}$$

The motor is driven at a PWM frequency of $25kHz$. Meaning, the duty cycle is set or updated every $40\mu s$. The timer frequency is set at $10MHz$. As a result, every PWM cycle has 400 ticks. It is necessary for the PWM frequency to be defined such that every PWM cycle has an integer number of ticks for clean PWM generation, i.e. the fraction in Equation 4.5 has to be whole.

$$\frac{f_{\text{timer}}}{f_{\text{PWM}}} = \frac{10\,000kHz}{25kHz} = 400\text{ticks} \tag{4.5}$$

The duty cycle resolution is defined as the smallest increment with which the duty cycle can be adjusted. This corresponds to one tick of the timer. Every tick contributes equally to the duty cycle, then the duty cycle resolution, given by Equation 4.6, is equal to 0.25%.

$$\text{duty cycle resolution} = \frac{f_{timer}}{f_{PWM}} \times 100\% = \frac{T_{timer}}{T_{cycle}} \times 100\% = 0.25\% \text{ per tick} \tag{4.6}$$

The duty cycle is typically clamped between -98% and 98% to prevent damage to the H-bridge. However, the DRV8411A supports 100% duty cycle operation [6, Section 8.4.1], so in this specific implementation it is not necessary. Even though it is not necessary, 100% duty cycle operation is not a required feature and thus the duty cycle will be clamped regardless.

4.7. Feedback: Quadrature Encoder

The design features a quadrature encoder using two Hall effect sensors mounted at an offset of 90 degrees. Figure 4.6 shows schematically the physical relation of the Hall sensors to the motor when installed in the robot frame.

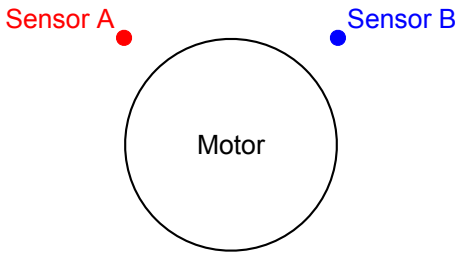

Figure 4.6: The two Hall sensors are shown at a 90 degree angle to each other. This corresponds with the physical arrangement of the sensors.

4.7.1. Hall Sensors

The design utilizes a total of three Hall sensors per driven wheel. Two Hall sensors are used to measure the speed and direction. The final sensor is used to calibrate wheel position and correct for measurement errors. The sensors work by detecting the alternating magnetic poles of the rotor magnets in the motor. As the motor turns the wheel, the magnets within the motor pass by the Hall sensor. Each Hall sensor essentially functions as a digital latch: it switches state when exposed to a North pole and resets when exposed to a South pole. This produces a binary alternating signal which results in a square-wave output. The square wave is processed by the quadrature encoder.

4.7.2. Quadrature Encoding

A quadrature encoder is used instead of rising-edge counting because counting both rising and falling edges in a quadrature configuration effectively quadruples the resolution. A high measurement resolution is crucial for feedback control, especially in low speed and high precision applications. Control accuracy directly relates to measurement accuracy, as inaccurate measurements will yield an inaccurate response from the controller. The two offset Hall sensors accordingly generate two phase-shifted square-wave signals. The direction the wheel is turning can be derived by detecting which signal leads and which one lags.

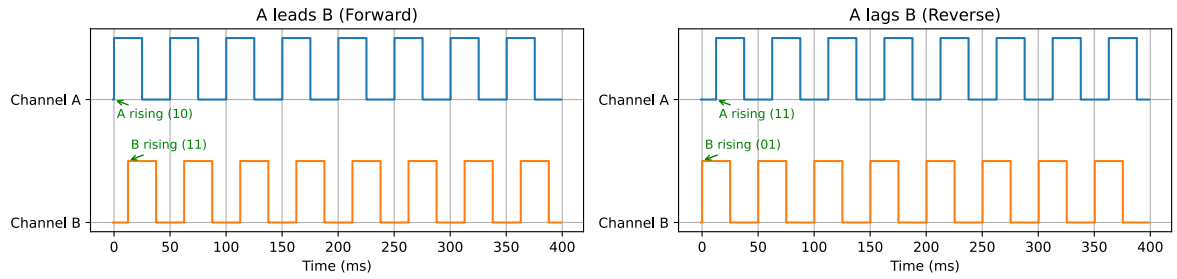

Figure 4.7: Forward movement (left) is measured when A leads B: the signal AB goes from 00 to 10. Backward movement (right) is measured when A lags B: the signal AB transitions from 01 to 11.

The measured signal logic is implemented using the PCNT (Pulse Counter) peripheral, which is built-in with the ESP microcontroller, just like the MCPWM peripheral. An advantage of the PCNT peripheral is that it handles the quadrature decoding entirely in hardware, leaving the CPU free to do other computations. The encoder features a glitch filter set to filter any pulses shorter than $t_{glitch} = 10\mu s$. This is because if a $10\mu s$ pulse were to be measured, it would correspond to an angular velocity of $1.5 \times 10^6\ RPM$, which is well outside of the operating range of the motor. The quadrupled resolution is achieved by counting all four events per cycle: A rises, B rises, A falls, and B falls. Whether A or B rises first, determines the direction of the motor. The angular velocity of the motor is derived by dividing

the count by four and multiplying it by the measurement frequency:

$$\dot{\theta}_{motor} = \omega_{motor} = \frac{\text{PCNT}}{4 \times T_{s,encoder}} \times 60 \text{ [RPM]} \tag{4.7}$$

Previous State AB	New State AB	Increment
00	10	+1
10	11	+1
11	01	+1
01	00	+1
00	01	-1
01	11	-1
11	10	-1
10	00	-1

Table 4.1: State transition table for PCNT

Table 4.1 shows which state transitions correspond to incrementing or decrementing the PCNT variable.

4.8. PI Control

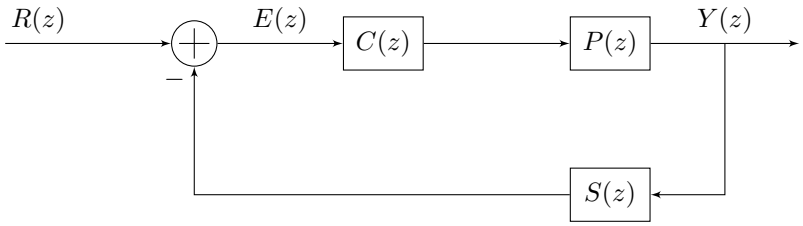

Figure 4.8: Closed-loop block diagram of the PI control system. $R(z)$ and $E(z)$ represent the z-domain reference and error signals respectively, both in rad/s. $Y(z)$ denotes the plant output signal, which is the motor speed in rad/s.

The system is controlled by a Proportional-Integral (PI) feedback controller. Figure 4.8 shows the closed loop diagram featuring three components and three key signals given by their Z-transform. Many parallels can be drawn with Figure 4.4, where $R(z)$, $E(z)$, and $Y(z)$ are simply the Z-transforms of $r[n]$, $e[n]$, and $y[n]$ respectively. The actuator and motor have been combined to a single Plant block, $P(z)$. The Hall sensors are represented by $S(z)$ and the PI controller is represented by $C(z)$. PI control is defined by two coefficients as its name suggests: the proportional gain and integral gain. The proportional gain coefficient, K_p, determines the linear response of the controller to the input error $E(z)$. The integral gain coefficient, K_i, determines the contribution of the accumulated error over time (integrating behaviour).

4.8.1. Modelling the System: Controller, Plant, Sensor

A solid theoretical base is vital for proper tuning of the controller parameters. Therefore, the plant transfer function is derived and subsequently used to simulate the step response with a PI controller. Tuning of the parameters is performed with targets defined by the system bandwidth and gain and phase margins, which are determined from the Bode plot.

The plant transfer function can be derived based on the armature current driven DC motor model as shown in Figure 3.4. A detailed derivation can be found in Appendix B.1. The plant transfer function in Laplace domain is:

$$P(s) = \frac{\dot{\Theta}(s)}{V(s)} = \frac{K_m}{(Js+b)(Ls+R) + {K_m}^2} \tag{4.8}$$

The design does not implement current control, only speed control. Therefore, electrical dynamics were neglected as the armature inductance time constant,

$$\tau_a = \frac{L_a}{R_a} \approx 0.2ms$$

is typically orders of magnitude smaller than the mechanical time constant for a typical iron-core brushed DC motor:

$$\tau_m = \frac{J}{b} \approx 1s$$

This simplification eliminates the higher-order dynamics from the plant transfer function, which is an advantage of controlling only the speed and not the armature current. The simplified transfer function is given by Equation 4.9.

$$P(s) = \frac{K_m}{Js + b + {K_m}^2} \tag{4.9}$$

The Laplace domain transfer function of a PI controller with coefficients K_p and K_i, is given by:

$$C(s) = \frac{K_p s + K_i}{s} \tag{4.10}$$

The PI controller is implemented digitally and updates the actuator signal with a period of T_s. T_s is initially defined at $T_s = 100\ ms$ to ensure the controller can respond to the mechanical dynamics.

The bandwidth will be derived in subsection 4.8.3, the controller frequency must be well above this bandwidth, typically $f_{controller} > 5 \times Bandwidth$. The sampling behaviour of the controller must be represented in the model; therefore, the transfer function is approximated to the Z-domain using the forward Euler method, according to the following condition:

$$s = \frac{z-1}{T_s} \tag{4.11}$$

Equation 4.11 works as an approximation because T_s is expected to be sufficiently small that the discretization error does not significantly affect the representation of the system. Deriving the exact transfer functions is possible but given the relatively small difference in accuracy, and the fact that the Euler approximation avoids introducing exponentials in the transfer function, it was deemed not necessary. This is backed up by the results compared to the theoretical model in section 6.2. Converting both $P(s)$ and $C(s)$ to Z-domain gives:

$$P(z) = \left(\frac{K_m T_s}{J(z-1) + (b + {K_m}^2)T_s} \right) \tag{4.12}$$

$$C(z) = \left(\frac{K_p(z-1) + T_s K_i}{z-1} \right) \tag{4.13}$$

The feedback loop includes the Hall sensor quadrature encoder (section 4.7), represented by $S(z)$. The sensing system introduces no (significant) dynamics. However, because the measurement is sampled and used in the *next control cycle*, $S(z)$ is modelled as a unit delay: $s[n] = y[n-1]$. The Z-domain equation for $S(z)$ logically follows and is given in Equation 4.14, z^{-1} represents a delay of one sample, which is defined by T_s.

$$S(z) = z^{-1} \tag{4.14}$$

This concludes the modelling of the system components. The following section explores how these components interact in order to model the system as a coherent whole.

4.8.2. Closed-loop Transfer Function

With all of the components modelled by their transfer functions, we now take a step back to look at the system as a whole (Figure 4.8). We want to determine the response of the system, $Y(z)$, in relation to the reference signal, $R(z)$. Deriving the transfer function, $\frac{Y(z)}{R(z)}$, directly from the block diagram:

$$E(z) = R(z) - Y(z)S(z) = R(z) - Y(z)z^{-1} \tag{4.15}$$

$$Y(z) = U(z)P(z) = E(z)C(z)P(z) \tag{4.16}$$

Combining the equations for $Y(z)$ and $E(z)$ gives:

$$Y(z) = \left(R(z) - Y(z)z^{-1}\right) C(z)P(z) \tag{4.17}$$

$$\frac{Y(z)}{R(z)} = \frac{C(z)P(z)}{1 + C(z)P(z)z^{-1}} \tag{4.18}$$

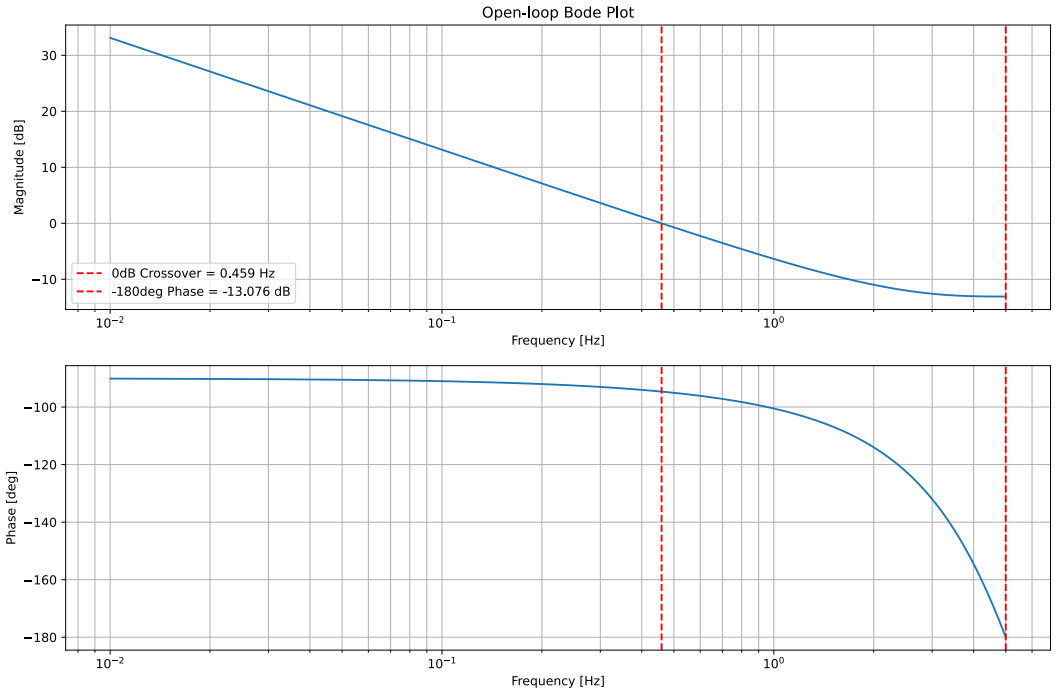

Figure 4.9: The open-loop Bode plot shows the frequency response of the system.

Parameter	Value
K_p	0.00045
K_i	0.004
T_s	$100ms$

Table 4.2: PI tune parameters

4.8.3. Bode analysis

To ensure system stability, it is important to ensure appropriate gain and phase margins. For these purposes, the **open-loop** Bode plot is graphed in Figure 4.9 using system parameters and the PI tune given by Table 4.2.

The gain margin is defined as the difference with respect to the $0dB$ gain when the phase is at $-180°$. Figure 4.9 shows that the gain margin is equal to $-13.076dB$, which means any not accounted for factors in the physical implementation can introduce a gain of up to $13.076dB$ without destabilizing the system. The same principle is applied in reverse to determine the phase margin. The 0dB crossover frequency of the magnitude plot is shown to be $0.459Hz$, where the phase equals $-94.62°$. The phase margin is then given by the same condition, as the absolute phase shift that would cause the $(0\text{dB}, -180°)$ instability criterion:

$$PM = \phi_{0dB} + 180° = 85.38° \tag{4.19}$$

The gain and phase margin give the system's resilience to tolerances and non-linearities in practical implementation. Generally, a system should have a phase margin of at least $30°$ to $45°$ and a gain margin of > 3dB to ensure stability. This means the given system configuration is somewhat conservative in both PM and GM.

Since the practical implementation makes use of 3D printed parts and comes with certain limitations, such as loose tolerances and resulting gear backlash, the phase and gain margins were not tightened to ensure stability under worst-case conditions.

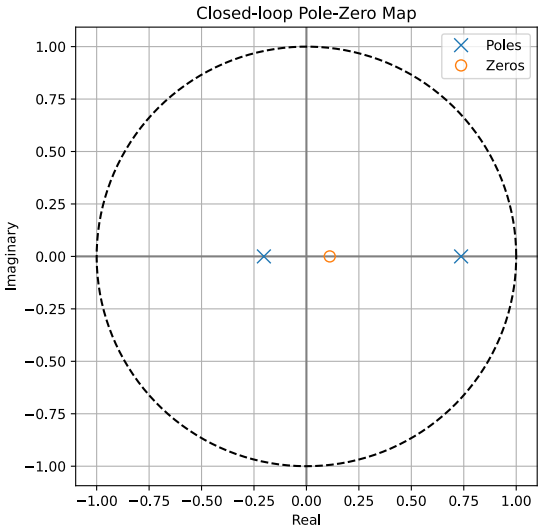

Figure 4.10: The poles and zeros lie well within the unit circle, which is the pole zero stability criterion in the z domain.

Figure 4.10 shows the poles and zeros of the closed-loop system in the Z-domain. The poles and zeros lie well within the unit circle, indicating a stable closed-loop system.

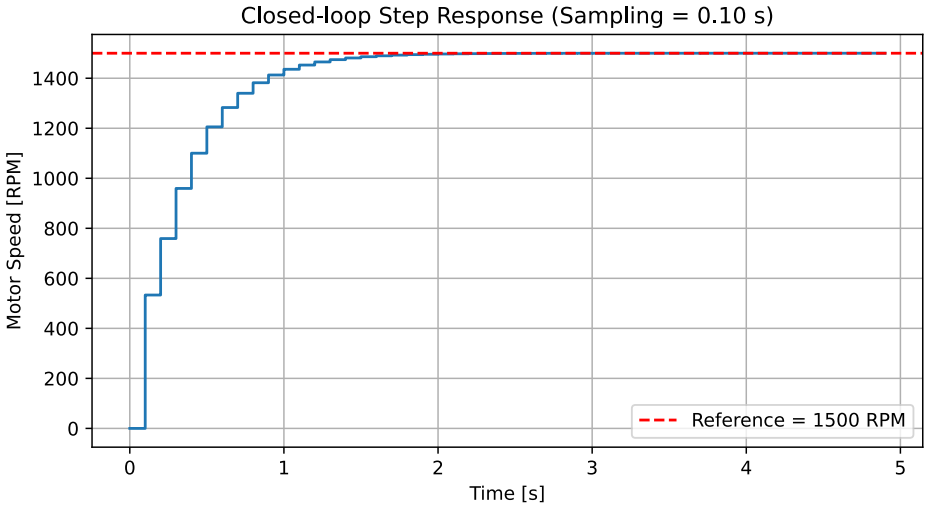

Figure 4.11: The system has a clean step response, settling at the reference within approximately 2 seconds, which is in accordance with the $0.459Hz$ 0dB-crossover frequency of the Bode plot.

Figure 4.11 shows the closed-loop step response of the system, with a sampling rate of 0.10 s. The settling time is approximately 2 seconds, which is in accordance with the program of requirements.

4.9. Position Tracking

At every update interval, the system approximates the covered distance by trapezoidal integration: The previous and current RPM measurements are averaged and multiplied by the sampling period and wheel diameter. This algorithm is demonstrated in Equation 4.20

$$\Delta s = \frac{1}{2}(RPM[n] + RPM[n-1]) \times T_s \times \frac{2\pi R_{wheel}}{60}\,[m] \tag{4.20}$$

The position tracking FSM runs on the external communications module, as it interfaces directly with the user PC. The module expects three parameters at each update interval: the covered distance and absolute rotation with respect to the static reference frame: $\Delta x, \Delta y,$ and θ. By tracking θ internally, Δx and Δy can be calculated at every interval as shown in Equation 4.21.

$$\Delta x = \Delta s \times \cos(\theta)\ , \Delta y = \Delta s \times \sin(\theta) \tag{4.21}$$

5

Testing Protocol

5.1. Testing Setup

In order to test the system external components must be connected to the PCB. These connections can be seen in figure 5.1. The motor is connected to pins 1 and 2. Pins 3-4-5, 6-7-8 and 9-10-11 are used for the three Hall sensors. Pins 14, 15 and 16 are available for other analog signals. Once all external components are connected it is possible to test the different features of the system.

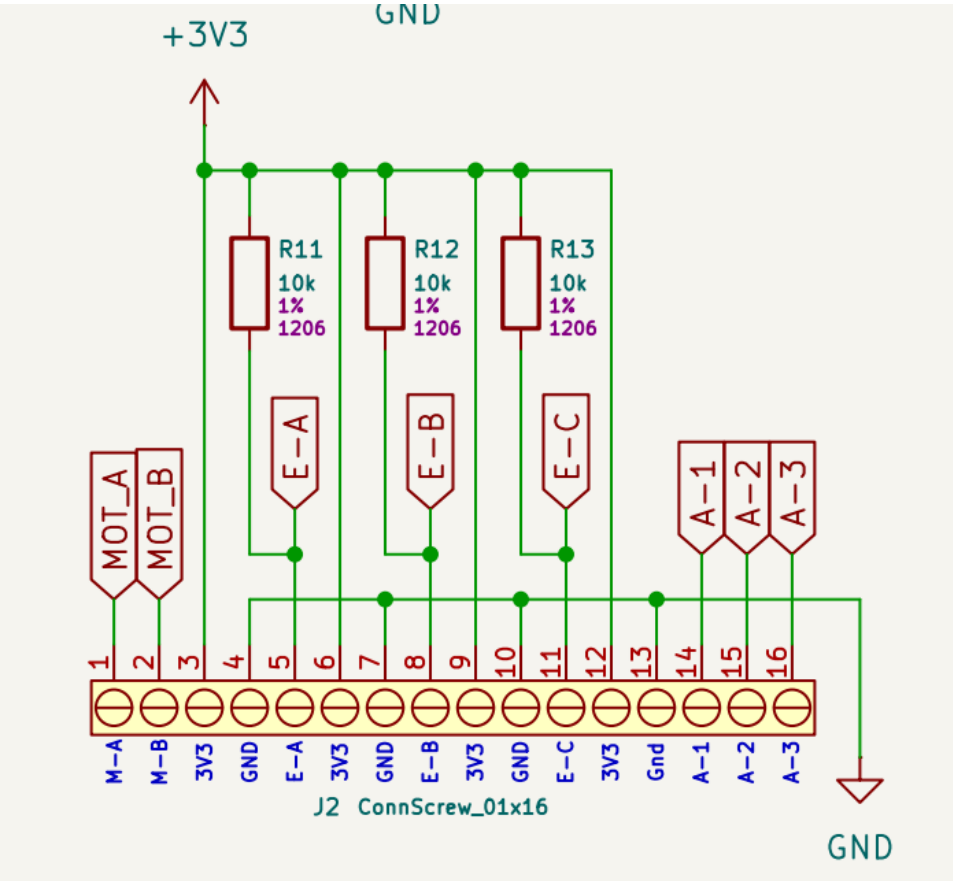

Figure 5.1: This image shows how all external elements are connected to the PCB

5.2. Features to be Tested

The following is a checklist for reference during testing to ensure that the requirements of the system are met.

- **Motor functionality:**
  - Is the PWM duty cycle adjustable in real time?
  - Does adjusting the duty cycle put unnecessary strain on the system?
  - Do the 3 motor states function as intended?
  - Does the system limit current draw and engage overcurrent protection when the current exceeds $4.0\ A$?
- **PI controller:**
  - Is the response stable?
  - Are the system parameters within requirements?
    - Settling time $< 2000\ ms$;
    - overshoot $< 1\ \%$;
    - steady state error $< 5\ \%$.
  - Does the controller deal well with physical disturbances?
  - Is the system response consistent across the operating range? I.e. does the system respond the same to a reference of 1000 RPM as it does to 3000 RPM?
- **CAN communication:**
  - Can signals be sent to other modules?
  - Can signals be received from other modules?
- **System integration:**
  - Does the system override the CAN signal and stop when collision is imminent?
  - Does the system initialise in a stopped state when no command is given?

5.2.1. Motor Functionality

In order to test the functionality of the motor a PWM signal must be sent to the H-bridge, which in turn sends a PWM signal to the motor. The duty cycle of this PWM signal must be able to be changed in real time. A potentiometer will be attached to pin 14. This in turn generates an analog signal which can be interpreted by the microcontroller in order to adjust the duty cycle.

Once it is confirmed that the motor can be controlled through a PWM signal, the duty cycle can be changed and communication is possible over CAN, the 3 motor states can be tested. This is done by waiting for the communications module to send the wanted state. Once this is received, the system will go into the chosen state and do the corresponding actions.

5.2.2. PI Controller

The PI Controller is tested and tuned based on the theory as discussed in section 4.8. Using the $\mathrm{pyserial}$ library to read the serial bus, the Hall sensor measurement data and PI parameters can be stored in $.\mathrm{csv}$ format for analysis and documentation. Tuning the PI controller involves an iterative process:

1. Set $T_s = 100ms$, and $K_p = 0.00045$ as determined by the theoretical model. Set $K_i = 0$ for the time being;
2. Run the motor at the desired reference speed, in this case $R = 1500$ RPM;
3. Check whether the chosen K_p results in a stable response, ignoring the steady-state error as long as the system settles at an arbitrary point;
4. If the response is oscillatory or too slow, adjust K_p:

- If the system oscillates, decrease K_p;
- If the system takes too long to settle, increase K_p;

5. When K_p has been tuned, adjust K_i starting at $K_i = 0.004$ in accordance with the theoretical model;
6. Run the motor once again at the same reference speed;
7. Tune K_i until the response is within specification:
   - If the system takes too long to hit the desired reference speed, increase K_i
   - If the system overshoots, oscillates hysterically, or overcompensates in another undesired fashion, decrease K_i.
8. Check the acquired tune against the theoretical model to evaluate possible instabilities which by chance did not manifest in testing.
9. Verify the tune against other reference signals, e.g. 1000 RPM, 2000 RPM, 3000 RPM.
10. Document the settling time, overshoot, and steady state error for all cases. These must be within the requirements as defined in subsection 2.2.2.
11. Disturb the physical process by hindering the wheel from turning and monitor the system response.

5.2.3. CAN Communication

To test CAN communication, interfacing with another module is necessary. The module to be interfaced with is the communications module. A signal will be sent over the CAN bus to the communications module, which will interpret it and send a response to verify two-way communication.

6

Implementation and Results

6.1. Motor Functionality

The functionality of the motor was tested as described in subsection 5.2.1. It was confirmed that the motor could be controlled by the given PWM signal and that changing the duty cycle was possible. Additionally, a test was run in order to determine if the overcurrent protection held. The rotor was locked into place in order to force the motor to draw more current. Soon after the overcurrent protection triggered and the motor stopped receiving any input signals.

6.2. PI Controller: Initial Implementation

The PI controller was originally implemented with a higher sampling time to ensure a stable response and guarantee useful initial measurements. The following parameters were used:

Parameter	Value
K_p	0.00045
K_i	0.0004
T_s	220ms

Table 6.1: Implemented PI controller parameters

The given parameters yielded the following system responses when tracking a known reference:

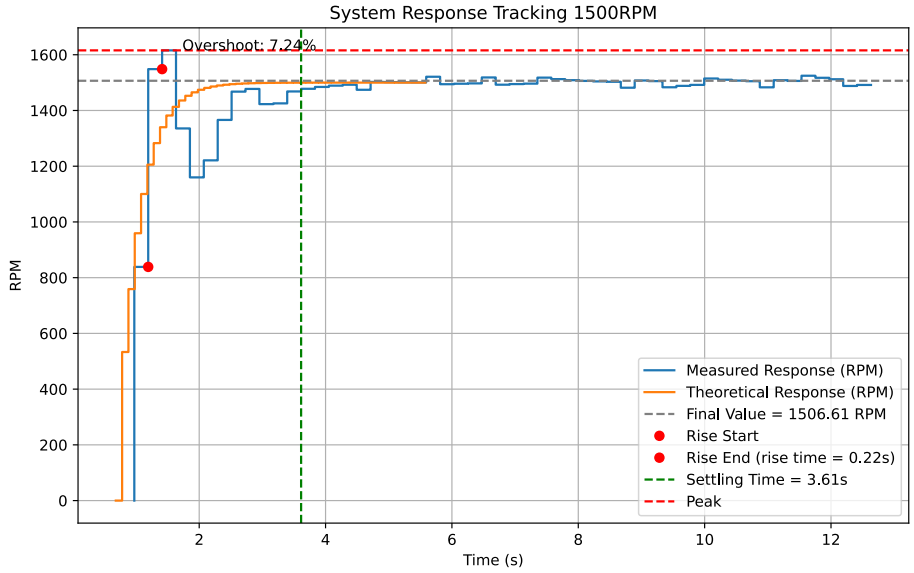

Figure 6.1: The 1500 RPM response of the system.

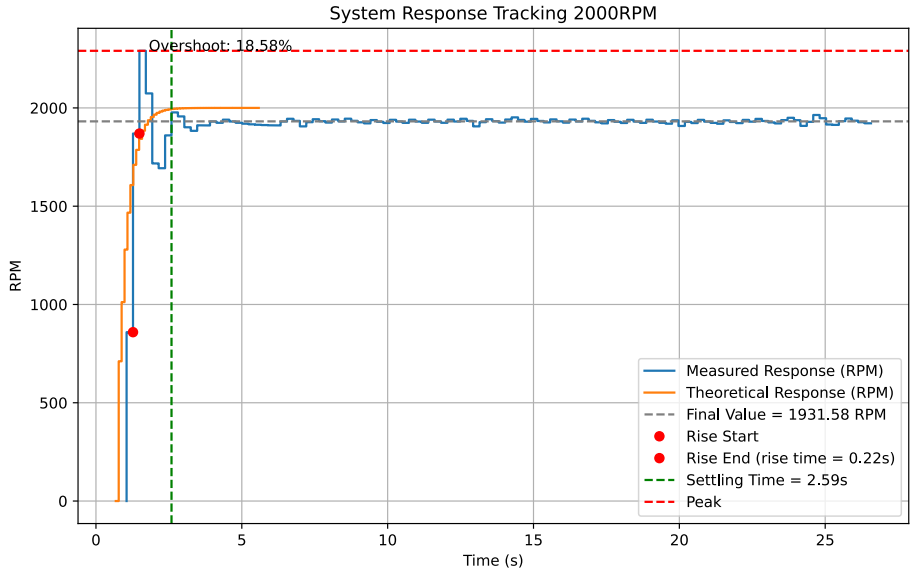

Figure 6.2: The 2000 RPM response of the system.

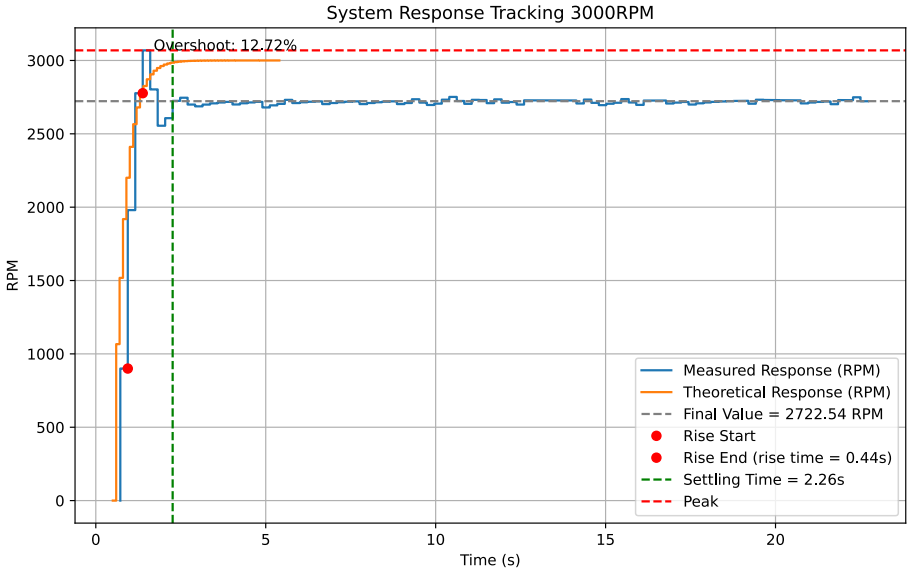

Figure 6.3: The 3000 RPM response of the system..

As shown in Figure 6.1, for a reference of 1500 RPM the system has a relatively smooth rise with an overshoot of $7.24\ \%$ and settles within approximately $3.6\ s$. At low speeds, the chosen gains provide adequate damping and integral action to remove steady state error. However, the overshoot is too much.

Figure 6.2 shows a considerably higher overshoot of $18.6\ \%$ and steady state error of $e_{ss} \approx 70$ RPM. The system cannot adequately compensate the load torque of the motor, which is highlighted by the response at higher speeds. Figure 6.3 shows a lower overshoot than the response at 2000 RPM, however, the steady state error increases further to approximately 280 RPM. This shows that the controller is not robust enough to track different reference signals in a satisfactory way. Interesting to note is that the settling time gets better at higher reference signals, this may be due to the large initial proportional term as a higher reference value yields a higher error when the motor is stationary.

6.3. PI Controller: Final Tuning

After a number of physical adjustments, the PI controller was re-tuned and it was found that $T_s = 100\ ms$ was achievable without causing instability.

Parameter	Value
K_p	0.00028
K_i	0.0004
T_s	100 ms

Table 6.2: Implemented PI controller parameters

Figure 6.4, Figure 6.5, and Figure 6.6 show the newly attained system responses. It can be seen that the measured response more closely matches the theoretical response. This seems obvious enough when considering that the theoretical model uses $T_s = 100\ ms$. However, it is interesting to note how much of a difference it makes. Overshoot has been reduced to 2.38% for 1500RPM and 1.73% for 3000RPM.

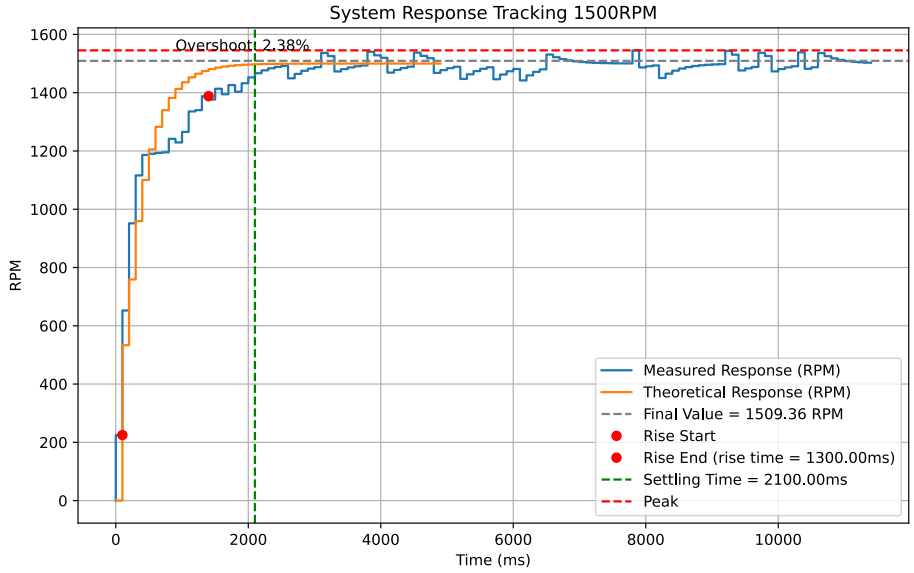

Figure 6.4: The 1500 RPM response of the system at $T_s = 100\ ms$.

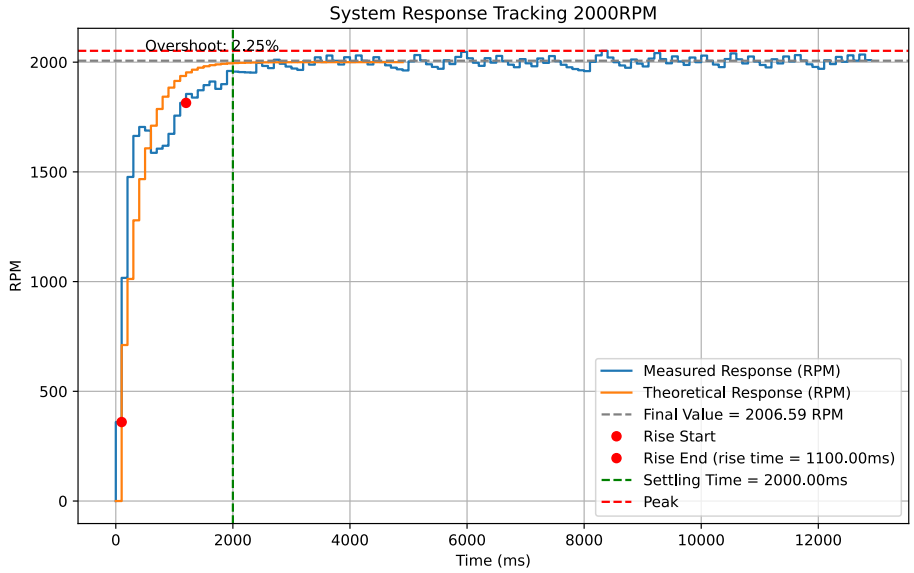

Figure 6.5: The 2000 RPM response of the system at $T_s = 100\ ms$.

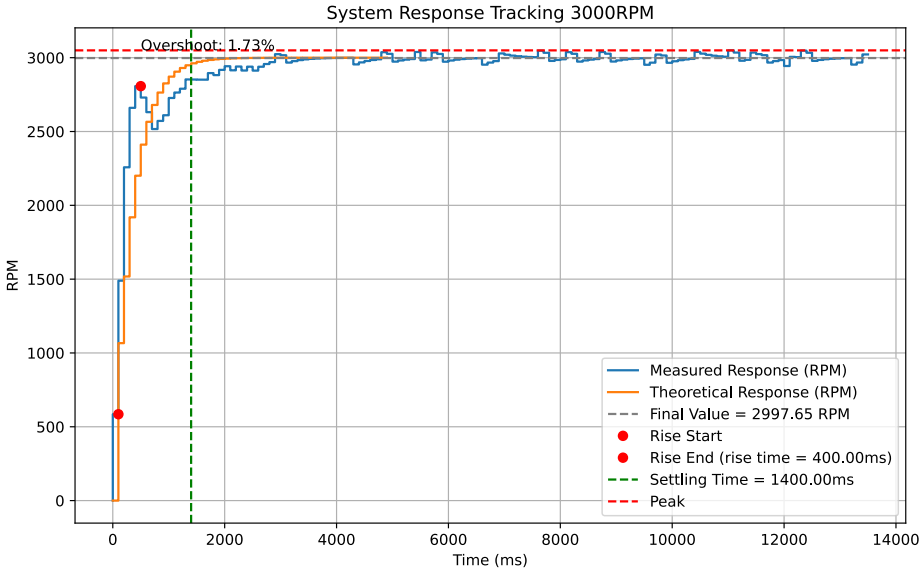

Figure 6.6: The 3000 RPM response of the system at $T_s = 100\ ms$.

6.4. CAN Communication

Interfacing with the communications module allowed for the CAN communication to be tested. As described in subsection 5.2.3 a signal was sent over the CAN bus to the communications module. This signal was subsequently interpreted and relayed back to the motor control module, thereby verifying two-way communication.

7

Discussion and Conclusion

7.1. Difficulties

The implementation and testing phases of the motor control module faced several challenges that significantly influenced the overall progress of the project.

One early hurdle was stemmed from previously functional software and hardware that were no longer operational at the outset of the project. This regression meant that additional time was required to debug and repair components before proceeding with further development. Consequently, this delayed subsequent phases and forced a reevaluation of the project timeline. The dead time was made useful by spending more time on high-level system design. For example, it was difficult to test features such as speed control without a housing for the Hall sensors.

Initially, it was assumed that the previously functional hardware had deteriorated and consequently stopped working. As a result, it was decided that a new PCB should be manufactured, this led to additional delays caused by the logistics involved. It was later determined that the original PCB did not have faulty hardware, and there was a problem with the software instead. This was not an improbable outcome due to the inherent difficulty of diagnosing errors in embedded systems code.

Overall, these issues highlight the importance of robustness, redundancy, and careful forward planning in both hardware and software design. Proper contingencies for potential failures and delivery delays can aid in mitigating their impact and enable faster recovery when problems arise. Despite these challenges, progress was eventually made, and valuable lessons were learned for future control system design and implementation.

7.2. Missed Targets

While the main goal of the project was to design and build a robot within the given framework with relatively simple functionality, ambitions were accounted for if the need for them arose halfway through the project.

In the early design stage, the groundwork was laid for advanced manoeuvring using inverse kinematics to track a point or a path as given by the user. This was not implemented due to time constraints.

Full integration of all modules also proved challenging within the given time constraint, in part due to the difficulties discussed in section 7.1. While this is not a catastrophic outcome, it is definitely beneficial to have the opportunity to test the device in its intended environment as opposed to a lab environment.

Whilst the implementation and testing of the PI controller was certainly successful, the aforementioned failure to reach full system integration hindered testing interaction with the external navigation FSM which runs on the communications module.

7.3. Improving GEMS

One of the goals of the project was to evaluate the effectiveness of the GEMS framework and to investigate where improvements could be made to make designing within the framework accessible and powerful for mechatronics students[19]. With this in mind, an additional report will be made outlining our perspective and experience with the framework, alongside suggestions for improvement.

7.4. Design Alterations

When developing a control application, PID control is usually the first consideration, as was the case here. It turns out that PI control works broadly for the purposes of the project. However, as can be seen in section 6.2, there is room for improvement, mainly in the transient response and steady state error at higher target RPMs. These issues can be alleviated partly through more extensive parameter tuning, but there are also other options. It may be worthwhile to investigate different control schemes such as adding derivative feedback or a lead compensator, or maybe even the more advanced Model Predictive Control (MPC) or Adaptive PI methods for compensating the loose tolerances of the system. These would likely function more as research experiments rather than actual significant improvements over the model though, as for example MPC is typically viewed as overkill for a simple brushed DC motor speed loop.

subsection 4.8.1 mentions that current control was not implemented in favour of only controlling the system based on the speed. Whilst speed control certainly is powerful enough for the purposes of this project, an underlying reason for underutilising the current sensor is the trickiness of calibration. Ideally, the PCB design would implement some kind of interactive ADC calibration method/peripheral, which may be worth investigation for future development.

$$T(s) = \frac{G(s)}{1 + G(s)} \tag{7.1}$$

At $(0\ dB, -180°) \implies G(s) = -1$

Then:

$$T(s) = \frac{-1}{1 - 1} \tag{7.2}$$

References

[1] E. Armando et al. “H-Bridge Converter as Power Electronics Workbench: An Effective Teaching Case of Learning by Doing”. In: *2019 21st European Conference on Power Electronics and Applications (EPE ’19 ECCE Europe)*. 2019, P.1–P.10. DOI: 10.23919/EPE.2019.8915071.

[2] Stephen Chapman. *Electric Machinery Fundamentals*. en. 4th ed. McGraw-Hill Science/Engineering/Math, 2003. Chap. 2, 3, 5. ISBN: 978-0072465235.

[3] Øyvind Nydal Dahl. “What Is an H-Bridge?” In: *Build Electronic Circuits* (2018). URL: https://www.build-electronic-circuits.com/h-bridge/.

[4] *ESP32-S3-WROOM-01 ESP32-S3-WROOM-01U*. ESP32-S3-WROOM-01. Espressif. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf.

[5] Ltd. Espressif Systems (Shanghai) Co. *ESP-IDF Programming Guide: Motor Control Pulse Width Modulator (MCPWM)*. 2025. URL: https://docs.espressif.com/projects/esp-idf/en/v5.4.1/esp32s3/api-reference/peripherals/mcpwm.html.

[6] Texas Industries. *DRV8411A Dual H-Bridge Motor Driver with Current Regulation*. 2021. URL: https://www.ti.com/lit/ds/symlink/drv8411a.pdf?ts=1703168298370&ref_url=https%253A%252F%252Fwww.ti.com%252Fsitesearch%252Fen-us%252Fdocs%252Funiversalsearch.tsp%253FlangPref%253Den-US%2526searchTerm%253DDRV8411APWPR%2526nr%253D3.

[7] Isabellenhutte. *Shunt Advantages and Challenges for EV Current Measurement*. 2024. URL: https://blog.isabellenhuetteusa.com/shunt-advantages-and-challenges-for-ev-current-measurement/.

[8] LEM. *Hall Effect Current Sensors*. 2024. URL: https://www.lem.com/en/hall-effect-current-sensors.

[9] Sandeep Kumar Malu and Jharna Majumdar. “Kinematics, Localization and Control of Differential Drive Mobile Robot”. In: *Global Journals of Research in Engineering* 14.H1 (Jan. 2014), pp. 1–7. URL: https://engineeringresearch.org/index.php/GJRE/article/view/1233.

[10] Eka Maulana, M. Aziz Muslim, and Akhmad Zainuri. “Inverse kinematics of a two-wheeled differential drive an autonomous mobile robot”. In: *2014 Electrical Power, Electronics, Communicatons, Control and Informatics Seminar (EECCIS)* (2014). DOI: 10.1109/EECCIS.2014.7003726.

[11] ALLEGRO microsystems. *Automotive Current Sensor IC with Fast Overcurrent Fault and Common-Mode Rejection*. 2025. URL: https://www.allegromicro.com/-/media/files/datasheets/acs71240-datasheet.pdf.

[12] NXP. *Parallel Configuration of H-Bridges*. 2014. URL: https://www.nxp.com/docs/en/application-note/AN4833.pdf.

[13] Badiryah Obaid, Ameer L.Saleh, and Abbas Kadhim. “Resolving of optimal fractional PID controller for DC motor drive based on anti-windup by invasive weed optimization technique”. In: *Indonesian Journal of Electrical Engineering and Computer Science* 15 (July 2019), pp. 95–103. DOI: 10.11591/ijeecs.v15.i1.pp95-103.

[14] Nirav D. Mehta Pathik A. Kapadiya Ishan R. Shah. “STUDY AND IMPLEMENTATION OF DC DRIVE USING PIC16F877A MICROCONTROLLER”. In: *International Journal of Research in Engineering and Technology* 3 (5 2014), pp. 621–625. URL: https://ijret.org/volumes/2014v03/i05/IJRET20140305114.pdf.

[15] David Pereles. *Measuring Current using Shunt Resistors*. 2024. URL: https://www.tek.com/en/blog/measuring-current-using-shunt-resistors.

[16] Livinti Petru and Ghandour Mazen. “PWM Control of a DC Motor Used to Drive a Conveyor Belt”. In: *Procedia Engineering* 100 (2015). 25th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2014, pp. 299–304. ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2015.01.371. URL: https://www.sciencedirect.com/science/article/pii/S1877705815003987.

[17] Errol G. Rowe and Thomas A. Wettergren. “Coverage and Reliability of Randomly Distributed Sensor Systems with Heterogeneous Detection Range”. In: *International Journal of Distributed Sensor Networks* 5.4 (2009), pp. 303–320. DOI: 10.1080/15501320802299853. eprint: https://doi.org/10.1080/15501320802299853. URL: https://doi.org/10.1080/15501320802299853.

[18] Mulukutla S. Sarma. *Electric Machines: Steady-State Theory and Dynamic Performance*. en. 2nd ed. Cengage Learning, 1997. Chap. 5, 9. ISBN: 978-0534938437.

[19] Podržaj Škulj G. “Modular Design for Inclusive and Accessible Mechatronic Educational Equipment”. In: *33rd International Electrotechnical and Computer Science Conference* (2024), pp. 635–639. URL: https://erk.fe.uni-lj.si/2024/papers/skulj(modular_design).pdf.

[20] Matthias Spang and Niklas Hofstoetter. “Evaluation of Current Measurement Accuracy for a Power Module with Integrated Shunt Resistors”. In: *PCIM Europe 2017; International Exhibition and Conference for Power Electronics, Intelligent Motion, Renewable Energy and Energy Management*. 2017, pp. 1–8.

[21] Kyle R. Stone. *Simplifying high-voltage sensing with Hall-effect current sensors*. 2024. URL: https://www.ti.com/lit/ta/ssztcy8/ssztcy8.pdf.

[22] *Textbook of Electrical Technology*. en. 1st ed. New Dheli: Chand (S.) & Co Ltd ,India, 2008. ISBN: 978-8121924412. URL: https://nit-edu.org/wp-content/uploads/2021/09/ch-30-Speed-control-of-dc-motors.pdf.

A

Source Code

Code for the PI controller

```

#include "Arduino.h"
#include "driver/mcpwm_prelude.h"
#include "driver/mcpwm_types.h"
#include "driver/pulse_cnt.h"
#include <stdio.h>
#include <stdint.h>
#include "esp_check.h"
#include "esp_adc/adc_oneshot.h"

// The following macro defines the PWM channels
#define BDC_MCPWM_TIMER_RESOLUTION_HZ 10000000 // 10MHz, 1 tick = 0.1us
#define BDC_MCPWM_FREQ_HZ             25000    // 25KHz PWM
#define BDC_MCPWM_PERIOD_TICKS        BDC_MCPWM_TIMER_RESOLUTION_HZ / BDC_MCPWM_FREQ_HZ
#define BDC_MCPWM_DUTY_TICK_MAX       BDC_MCPWM_PERIOD_TICKS/2
#define BDC_MCPWM_GPIO_A              11
#define BDC_MCPWM_GPIO_B              12
#define BDC_MCPWM_GROUP_ID            0
#define BDC_ADC_UNIT                  ADC_UNIT_1
#define BDC_ADC_CHAN                  ADC_CHANNEL_9

#define BDC_ENCODER_GPIO_A            41
#define BDC_ENCODER_GPIO_B            40
#define BDC_ENCODER_PCNT_HIGH_LIMIT   1000
#define BDC_ENCODER_PCNT_LOW_LIMIT    -1000

#define BDC_SPEED_LOOP_PERIOD_MS      220   // control motor speed every 220ms
#define BDC_ENCODER_LINES             1

// useful macros for controller implementation
#define limit_var_up(var, value) if(var > value){var=value;}
#define limit_var_down(var, value) if(var < value){var=value;}

#define Kp_s 0.00045
#define Ki_s 0.0004

mcpwm_timer_handle_t timer;
mcpwm_oper_handle_t opr;
mcpwm_cmpr_handle_t cmpa;
mcpwm_cmpr_handle_t cmpb;
mcpwm_gen_handle_t gena;
mcpwm_gen_handle_t genb;
adc_oneshot_unit_handle_t adc1_handle;

float ia = 0; // sensed armature current in A
float rpm_ref = 3000;
float duty = 0;
```

```
48 float cur_rpm_err = 0;
49
50 TaskHandle_t pwm_control_task_handle;
51
52
53 pcnt_unit_handle_t encoder_unit = NULL;
54
55 typedef struct {
56     pcnt_unit_handle_t encoder;
57     float rpm;
58 } speed_measure_t;
59
60 static speed_measure_t speed_measure_ctx = {
61     .encoder = NULL,
62     .rpm     = 0,
63 };
64
65 void ADC_init() {
66   adc_oneshot_unit_init_cfg_t adc_config;
67   adc_config.unit_id = BDC_ADC_UNIT;
68   adc_config.ulp_mode = ADC_ULP_MODE_DISABLE;
69   ESP_ERROR_CHECK(adc_oneshot_new_unit(&adc_config, &adc1_handle));
70
71   adc_oneshot_chan_cfg_t adc_chan_config;
72   adc_chan_config.bitwidth = ADC_BITWIDTH_DEFAULT;
73   adc_chan_config.atten = ADC_ATTEN_DB_12;
74
75   ESP_ERROR_CHECK(adc_oneshot_config_channel(adc1_handle, BDC_ADC_CHAN, &adc_chan_config));
76 }
77
78 void MCPWM_init() {
79   mcpwm_timer_config_t timer_config;
80   mcpwm_operator_config_t operator_config;
81   mcpwm_comparator_config_t comparator_config;
82   mcpwm_generator_config_t generator_config;
83
84   timer_config.group_id = BDC_MCPWM_GROUP_ID; // there are two independent MCPWM peripherals,
         use 0 here
85   timer_config.intr_priority = 1;
86   timer_config.clk_src = MCPWM_TIMER_CLK_SRC_DEFAULT;
87   timer_config.resolution_hz = BDC_MCPWM_TIMER_RESOLUTION_HZ; // pwm resolution
88   timer_config.period_ticks = BDC_MCPWM_PERIOD_TICKS;
89   timer_config.count_mode = MCPWM_TIMER_COUNT_MODE_UP_DOWN; // symmetric complemetary
90
91   operator_config.group_id = BDC_MCPWM_GROUP_ID;
92   operator_config.intr_priority = 1;
93   comparator_config.intr_priority = 1;
94   comparator_config.flags.update_cmp_on_tez = true;
95   generator_config.gen_gpio_num = BDC_MCPWM_GPIO_A;
96
97   ESP_ERROR_CHECK(mcpwm_new_timer(&timer_config, &timer));
98   ESP_ERROR_CHECK(mcpwm_new_operator(&operator_config, &opr));
99   ESP_ERROR_CHECK(mcpwm_operator_connect_timer(opr, timer));
100  ESP_ERROR_CHECK(mcpwm_new_comparator(opr, &comparator_config, &cmpa));
101  ESP_ERROR_CHECK(mcpwm_new_comparator(opr, &comparator_config, &cmpb));
102  // set the initial compare value for both comparators
103  ESP_ERROR_CHECK(mcpwm_comparator_set_compare_value(cmpa, 0));
104  ESP_ERROR_CHECK(mcpwm_comparator_set_compare_value(cmpb, BDC_MCPWM_DUTY_TICK_MAX));
105  ESP_ERROR_CHECK(mcpwm_new_generator(opr, &generator_config, &gena));
106   generator_config.gen_gpio_num = BDC_MCPWM_GPIO_B;
107  ESP_ERROR_CHECK(mcpwm_new_generator(opr, &generator_config, &genb));
108
109    // dual edge, complementary PWM
110  ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena,
111                  MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,
                        MCPWM_GEN_ACTION_HIGH),
112                  MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN, cmpa,
                        MCPWM_GEN_ACTION_LOW),
113                  MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
114  ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(genb,
```

```
115                     MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb,
                            MCPWM_GEN_ACTION_LOW),
116                     MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN, cmpb,
                            MCPWM_GEN_ACTION_HIGH),
117                     MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
118 }
119
120 void encoder_init()
121 {
122   pcnt_unit_config_t unit_config = {};
123   unit_config.high_limit = BDC_ENCODER_PCNT_HIGH_LIMIT;
124   unit_config.low_limit = BDC_ENCODER_PCNT_LOW_LIMIT;
125   unit_config.flags.accum_count = true; // enable counter accumulation
126
127   esp_err_t err = pcnt_new_unit(&unit_config, &encoder_unit);
128   ESP_ERROR_CHECK(err);
129
130   pcnt_glitch_filter_config_t filter_config = {};
131   filter_config.max_glitch_ns = 10000;
132
133   ESP_ERROR_CHECK(pcnt_unit_set_glitch_filter(encoder_unit, &filter_config));
134   pcnt_chan_config_t chan_a_config = {};
135   chan_a_config.edge_gpio_num = BDC_ENCODER_GPIO_A;
136   chan_a_config.level_gpio_num = BDC_ENCODER_GPIO_B;
137
138   pcnt_channel_handle_t pcnt_chan_a = NULL;
139   ESP_ERROR_CHECK(pcnt_new_channel(encoder_unit, &chan_a_config, &pcnt_chan_a));
140   pcnt_chan_config_t chan_b_config = {};
141   chan_b_config.edge_gpio_num = BDC_ENCODER_GPIO_B;
142   chan_b_config.level_gpio_num = BDC_ENCODER_GPIO_A;
143
144   pcnt_channel_handle_t pcnt_chan_b = NULL;
145   ESP_ERROR_CHECK(pcnt_new_channel(encoder_unit, &chan_b_config, &pcnt_chan_b));
146   ESP_ERROR_CHECK(pcnt_channel_set_edge_action(pcnt_chan_a, PCNT_CHANNEL_EDGE_ACTION_DECREASE
          , PCNT_CHANNEL_EDGE_ACTION_INCREASE));
147   ESP_ERROR_CHECK(pcnt_channel_set_level_action(pcnt_chan_a, PCNT_CHANNEL_LEVEL_ACTION_KEEP,
          PCNT_CHANNEL_LEVEL_ACTION_INVERSE));
148   ESP_ERROR_CHECK(pcnt_channel_set_edge_action(pcnt_chan_b, PCNT_CHANNEL_EDGE_ACTION_INCREASE
          , PCNT_CHANNEL_EDGE_ACTION_DECREASE));
149   ESP_ERROR_CHECK(pcnt_channel_set_level_action(pcnt_chan_b, PCNT_CHANNEL_LEVEL_ACTION_KEEP,
          PCNT_CHANNEL_LEVEL_ACTION_INVERSE));
150   ESP_ERROR_CHECK(pcnt_unit_add_watch_point(encoder_unit, BDC_ENCODER_PCNT_HIGH_LIMIT));
151   ESP_ERROR_CHECK(pcnt_unit_add_watch_point(encoder_unit, BDC_ENCODER_PCNT_LOW_LIMIT));
152   ESP_ERROR_CHECK(pcnt_unit_enable(encoder_unit));
153   ESP_ERROR_CHECK(pcnt_unit_clear_count(encoder_unit));
154   ESP_ERROR_CHECK(pcnt_unit_start(encoder_unit));
155 }
156
157 void speed_loop_cb(void *args)
158 {
159     static int last_pulse_count = 0;
160     speed_measure_t *ctx = (speed_measure_t *)args;
161
162     // Measure speed from encoder
163     int cur_pulse_count = 0;
164     pcnt_unit_get_count(ctx->encoder, &cur_pulse_count);
165     int real_pulses = cur_pulse_count - last_pulse_count;
166     last_pulse_count = cur_pulse_count;
167
168     float new_rpm = (float)real_pulses / (BDC_ENCODER_LINES * 4) * 60000.0f /
          BDC_SPEED_LOOP_PERIOD_MS;
169
170     // Apply EMA filtering to smooth out noise
171     float alpha = 0.3f;
172     ctx->rpm = alpha * new_rpm + (1.0f - alpha) * ctx->rpm;
173
174     // Proportional controller
175     static float old_rpm_err = 0;
176     float error = rpm_ref - ctx->rpm;
177     float Ts = BDC_SPEED_LOOP_PERIOD_MS / 1000.0f; // Scale the I-term according to the
          sampling period
```

```
178
179     // Ignore small errors
180     float err_log = error;
181     if (fabs(error) < 10.0f) error = 0.0f;
182
183     // Proportional
184     float p_term = Kp_s * error;
185
186     // Integral
187     static float i_term = 0;
188     i_term += Ki_s * Ts * old_rpm_err;
189     limit_var_up(i_term, 0.3f);
190     limit_var_down(i_term, -0.3f);
191
192     old_rpm_err = error;
193
194     // Apply saturation
195     float duty_clamped = p_term + i_term;
196     limit_var_up(duty_clamped, 0.7f);
197     limit_var_down(duty_clamped, -0.7f); // Assuming forward only
198
199     duty = duty_clamped;
200
201     // Log results over serial bus
202     Serial.printf("%lu,%.2f,%.2f,%.2f,%.2f\n", millis(), ctx->rpm, duty, p_term, i_term);
203
204 }
205
206 void setup() {
207   Serial.begin(115200);
208   while(!Serial){}
209
210   duty = 0;
211   ADC_init();
212   MCPWM_init();
213
214   mcpwm_timer_enable(timer);
215   mcpwm_timer_start_stop(timer, MCPWM_TIMER_START_NO_STOP);
216
217   delay(100);
218   encoder_init();
219   speed_measure_ctx.encoder = encoder_unit;
220   delay(50);
221   //Serial.println("Motor Control ready");
222   Serial.println("time,rpm,duty,p_term,i_term");
223
224
225 }
226
227 unsigned long last_speed_loop = 0;
228
229 void loop() {
230   unsigned long now = millis();
231
232   // Run speed loop every 100 ms
233   if (now - last_speed_loop >= BDC_SPEED_LOOP_PERIOD_MS) {
234     speed_loop_cb(&speed_measure_ctx);
235
236     // Update MCPWM duty
237     ESP_ERROR_CHECK(mcpwm_comparator_set_compare_value(cmpa, (int)((1 + duty) *
          BDC_MCPWM_DUTY_TICK_MAX / 2)));
238     ESP_ERROR_CHECK(mcpwm_comparator_set_compare_value(cmpb, (int)((1 + duty) *
          BDC_MCPWM_DUTY_TICK_MAX / 2)));
239
240     last_speed_loop = now;
241   }
242 }
```

B

Further Derivations

B.1. Plant Transfer Function

The motor torque, T_m, and back EMF e_{emf} are related to the armature current i_a and angular velocity $\dot{\theta}$ by:

$$\begin{cases} T_m = K_v i_a \\ e = K_v \dot{\theta} \end{cases} \tag{B.1}$$

where K_v is the motor velocity constant.

The following governing equations can be derived based on Newton's 2nd law and Kirchhoff's voltage law:

$$\begin{cases} J\ddot{\theta} + b\dot{\theta} = K_v i_a \\ L\dfrac{di}{dt} + Ri_a = V - K_v \dot{\theta} \end{cases} \tag{B.2}$$

Applying the Laplace transform gives:

$$\begin{cases} s(Js+b)\Theta(s) = K_v I(s) \\ (Ls+R)I(s) = V(s) - K_v s\Theta(s) \end{cases} \tag{B.3}$$

Both equations can be written with respect to $I(s)$ as follows:

$$\begin{cases} I(s) = \dfrac{s(Js+b)\Theta(s)}{K_v} \\ I(s) = \dfrac{V(s) - K_v s\Theta(s)}{(Ls+R)} \end{cases} \tag{B.4}$$

Now $I(s)$ is eliminated by equating the expressions, whereafter solving for $\frac{\Theta(s)}{V(s)}$ gives:

$$\frac{s(Js+b)\Theta(s)}{K_v} = \frac{V(s) - K_v s\Theta(s)}{Ls+R} \tag{B.5}$$

$$\frac{s(Js+b)(Ls+R)\Theta(s)}{K_v} + K_v s\Theta(s) = V(s) \tag{B.6}$$

$$P(s) = \frac{\Theta(s)}{V(s)} = \frac{K_v}{(Js+b)(Ls+R) + K_v^2} \tag{B.7}$$

C

Schematic

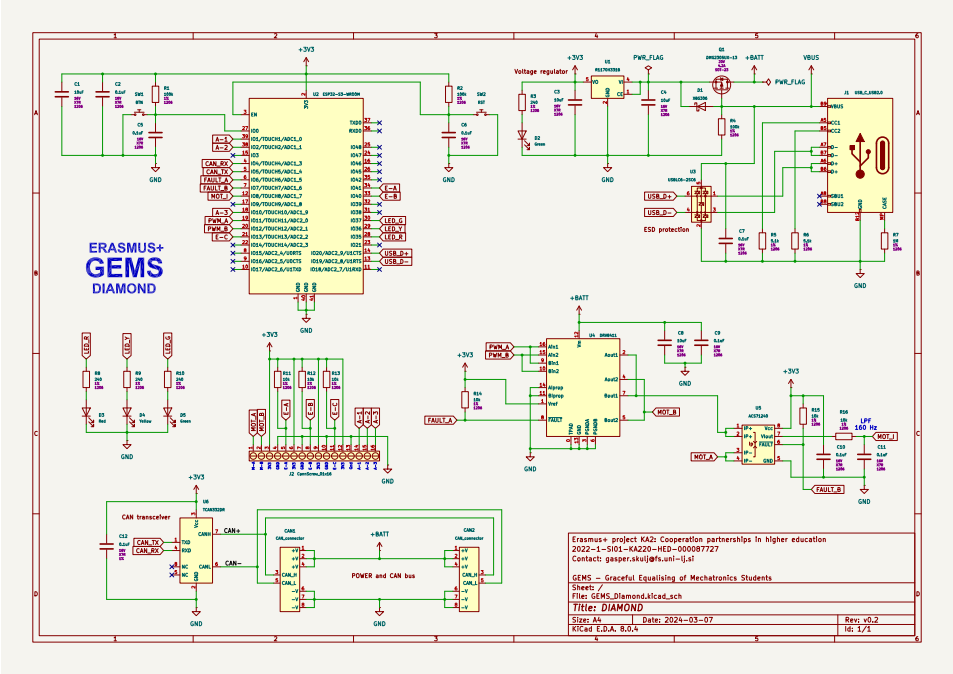

Figure C.1: PCB Schematic