UNIVERSITY OF AGDER

Active Slip Ring Friction Compensation for 3 DOF Quadcopter Lab

by

Torgrim Solvang Stensrud

Supervisor

Kristian Muri Knausgård, UiA

This bachelor's thesis is carried out as a part of the education at the University of Agder and is therefore approved as a part of this education. However, this does not imply that the University answers for the methods that are used or the conclusions that are drawn.

University of Agder, 2018

Faculty of Technology and Science

Department of Engineering

Abstract

This project report covers the designing and construction of an active friction compensation system for a slip ring connection, with the purpose of being used for quadcopter lab exercises. The considered USB slip ring intended for the lab generates more friction than desired, as multi-channel slip rings often do, which is the reason why friction compensation is considered as a solution. This marks the project foundation. The system uses a strain gauge in order to detect shear strain caused by drone rotation. A PI controller algorithm is used to process the strain samples and outputs the signal to a three phase motor driver and brushless DC motor actuator. The control system is then tested with different PI and filter parameters by applying a load cycle from a motor. The Ziegler-Nichols approach is also used to estimate desirable gain.

The results show that the compensation system works as intended. The control loop actively reduces the shaft strain and hence the experienced friction. The circuit components communicate as desired and the PI controller effectively accumulates the offset errors. However, the results also depicts a certain performance reduction due to undesirable external disturbance. Vibrations during the load tests and shaft skewness are the main contributors. Additionally, the current system is not capable of coping with bending strain that occurs when the quadcopter rests on the joint in a tilted position. The results from running the Ziegler-Nichols method show that the PI controller parameters can not be determined since the system response does not exhibit sustained oscillations for increased proportional controller action. Improvements to the system such as shaft straightening/replacement and bending strain compensation by absorbing the bending torque with a bearing-support, are suggested.

Abbreviations

BLDC	Brushless Direct Current Motor
MCU	Microcontroller Unit
ISR	Interrupt Service Routine
IC	Integrated circuit
QC	Quadcopter

Contents

Chapter 1

Introduction

In most control applications the aim is to control a plant towards the desired response and sometimes cope with potential disturbances that may affect the system [1]. This is a fundamental brick of control theory, and real-world implementations can be found almost everywhere. For instance, the control of room temperature done by a radiator. The radiator will heat up if the temperature setting is changed by the user, but will also compensate for disturbance in temperature caused by; e.g. a draft from an open window or cool air from an air conditioner.

One physical phenomenon that technology has tried to compensate for during all of history is *friction*. On one side, friction realised the possibility of generating fire by rubbing coarse materials together, while on the other it made transporting timber and stone over greater distances a nightmare for our predecessors. When the first ball bearing was patented by inventor Philip Vaughan in 1794, it revolutionised the industrial sector [2]. Prior to this, machinery required much maintenance and parts needed to be changed regularly. Additionally, the efficiency loss due to heat generation in drives was quite substantial. Bearings could *passively* compensate for friction and increase the efficiency of industrial applications by a considerable amount.

Friction is present in all mechanical systems to a certain degree and is perhaps one of the most important aspects to consider in dynamic systems. For control engineers, it is especially important to keep in mind during the design phase. Friction can be quite challenging to deal with due to its highly non-linear nature. More precisely, a static system will experience different friction forces than a dynamic one. For instance, a rotating drive shaft will be subjected to less friction torque than the friction subjected to the static beam just before rotation [3] [4].

This particular project focuses on actively compensating for the friction generated in a *slip ring* by a *quadcopter* (QC). Or more precisely; actively compensating for the *experienced* friction/resistance by the QC. The slip ring [5] is connected to a stationary base that transmits power and data, essentially connecting the base with the QC that operates at the end of a rotary shaft. Since a slip ring fundamentally works by transmitting signals over two surfaces in contact, one stationary and one rotary, by slipping "brushes", friction is generated at the points of contact [6]. By measuring the friction-caused strain between the slip ring and QC, a motor-actuator could be used to actively compensate for this strain, thus reducing the mechanical resistance experienced by the QC. I.e., the control loop aims to make the QC "perceive" as it is operating in a frictionless environment. It is important to note that the friction torque generated by the slip ring remains

the same, i.e. there is no actual reduction in friction, and the loss of efficiency caused by the slip ring stays unchanged [4].

1.1 Problem Statement

The project will focus on the implementation of a slip ring connection between a stationary base and a QC, and a control loop that actively compensates for the friction experienced by the QC. The connection needs to fulfil mechanical criteria so that the drone is allowed to operate with 3 DoF. More precisely; tilt < 45 degrees about the horizontal axes and continuous rotation about the vertical axis. The QC will not be allowed translational movement. Additionally, the designing of a stationary base that houses control components and effectively spreads the air flow generated by the QC propellers is required.

The following QC boundary conditions are given:

1. QC mass $0.25\,\text{kg} \leq m_{\text{quad}} < 1.0\,\text{kg}$
2. QC maximum bidirectional angular velocity $|\omega_{\text{quad,max}}| = 360.0\,\frac{\text{deg}}{\text{s}}$

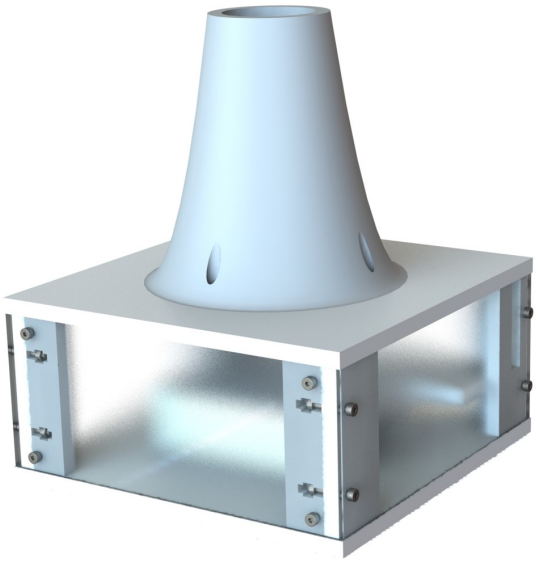


Figure 1.1: Rendered CAD model of the base.

Chapter 2

Theory

The following chapter covers the theoretical foundation for the project.

2.1 Torque and Shear Strain of Beams

The initial system consists primarily of a drone attached to a slip ring by a cylindrical shaft. The drone can rotate freely about the vertical axis, but do not translate in any direction. The illustration in Fig. 2.1 shows the simplified system.

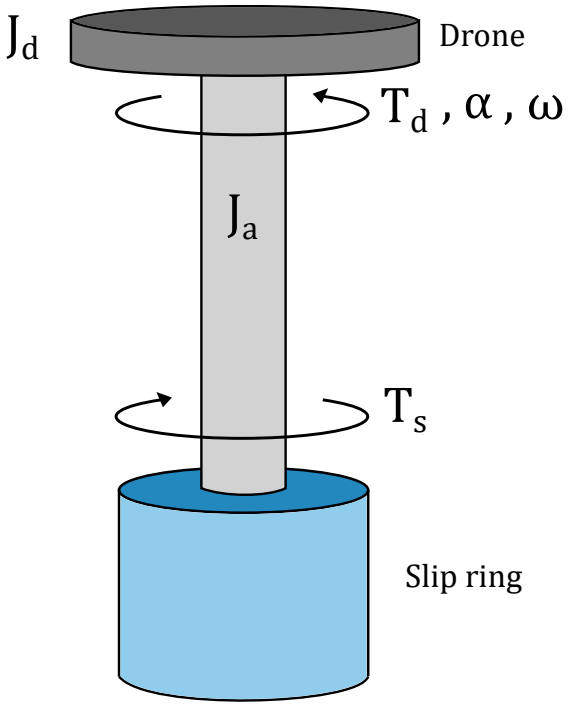


Figure 2.1: Simplified slip ring setup.

Here, T_d (Nm) is the torque subjected on the shaft by a drone with some moment of inertia J_d ($\mathrm{kg} \cdot \mathrm{m}^2$). The torque subjected on the shaft caused by slip ring resistance is denoted T_s. The shaft moment of inertia is denoted J_a and the angular speed and

acceleration of the rotating parts, i.e. drone and shaft, are denoted ω ($\frac{\text{rad}}{\text{s}}$) and α ($\frac{\text{rad}}{\text{s}^2}$) respectively.

By introducing Newton's 2nd law for rotational motion [4, 7] the following equation is achieved (2.1):

$$T_d - T_s = (J_a + J_d) \cdot \alpha \tag{2.1}$$

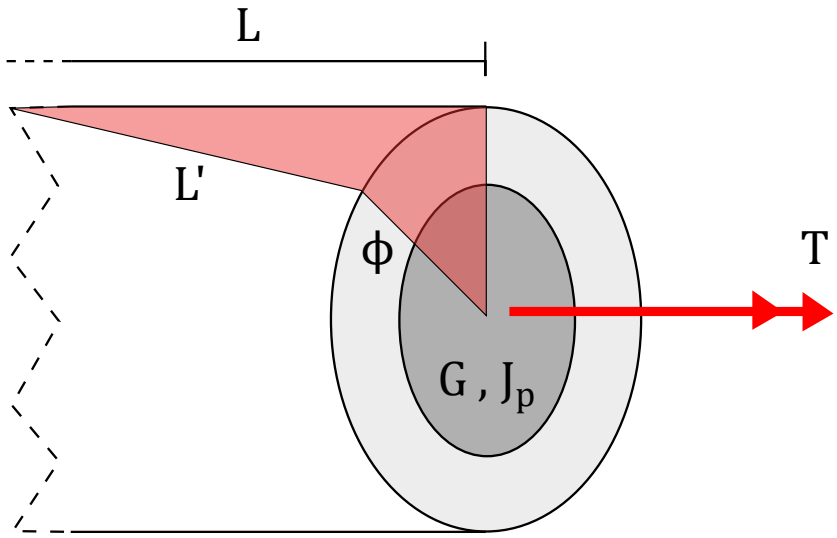


Figure 2.2: Pipe displacement caused by torque.

Consider the case of a static pipe, i.e. $\alpha = 0$. The torque generated by the drone T_d is not able to overcome the static friction torque in the slip ring T_s, which yields $T_d = T_s$ for every cross section of the pipe. The subjected torque will result in shear stress and strain throughout the pipe, which results in a displacement angle at the end of the pipe. Note that the shear related torque is the torque applied to the shaft ends with opposite signs. The relation between shear angular displacement φ and subjected torque T for cylindrical shafts can be derived from *Hookes law in shear* [8, 7] (2.2):

$$\varphi = \frac{T \cdot L}{G \cdot J_p} \tag{2.2}$$

Where L (m) denotes length and J_p is the shaft polar moment of inertia (m^4). G is the *shear modulus of rigidy* (Pa), which denotes the ratio of shear stress per unit shear strain of a material. From (2.2), it can be noted that L, G and J_p are constants for the given case, which yields a linear relationship between torque and shear displacement. Finally, the shear strain γ subjected on each cross section of the pipe is defined by the following equation (2.3) [7]:

$$\gamma = \frac{\varphi \cdot r}{L} \tag{2.3}$$

Where r is the radius from the shaft centre.

2.1.1 Shaft Strain Estimation

By using the equations introduced in the previous section, the theoretical strain of the static shaft can be estimated. Initially, some assumptions are made:

- The shaft is in static equilibrium, i.e. $\sum T = 0\,\text{Nm}$
- The length from slip ring mount to shaft end is $L = 250.0\,\text{mm}$, outer shaft radius is $r_o = 6.0\,\text{mm}$ and inner is $r_i = 4.0\,\text{mm}$
- The shaft material is carbon steel

Since the shaft is in static equilibrium the torque generated by the drone must be equal to the opposite friction torque generated from the slip ring, i.e. $T = T_d = T_s$. The SNU11 datasheet [5] lists the following friction torque for their slip rings (2.4):

$$\text{Torque} = 0.1\,\text{Nm} + \frac{x}{6} \cdot 0.03\,\text{Nm} \tag{2.4}$$

Where x denotes the amount of circuits, i.e. lines, through the slip ring. Since the SNU11-0410-04S slip ring used for this project consists of $x = 8$ circuits, the resulting slip ring torque yields $T_s = 0.14\,\text{Nm}$. Whether this is maximum static or dynamic friction is not mentioned, but lets assume its the peak static friction torque.

The polar moment of inertia J_p can then be calculated to (2.5) [7, 8]:

$$J_p = \frac{\pi}{2} \cdot (r_o^4 - r_i^4) \approx 1.63363 \cdot 10^{-9}\ \ m^4 \tag{2.5}$$

The Modulus of Rigidity G for the material can be found from look-up tables [7, 9]. For normal carbon steel the shear modulus equals $G \approx 77\,\text{GPa}$ and is therefore used in this regard. The displacement angle at the end of the shaft φ can then be calculated from (2.2), yielding (2.6):

$$\varphi = \frac{T \cdot L}{G \cdot J_p} = \frac{0.14\,\text{Nm} \cdot 0.25\,\text{m}}{77 \cdot 10^9\ \ \text{Pa} \cdot 1.63363 \cdot 10^{-9}\ \ \text{m}^4} \approx 2.78243 \cdot 10^{-4}\ \ \text{rad} \tag{2.6}$$

This results in the following shear strain of the outer shaft surface per unit of length (m) by the formula (2.3), thus (2.7) [4]:

$$\gamma = \frac{\varphi \cdot r_o}{L} \approx 6.67783 \cdot 10^{-6} \tag{2.7}$$

2.2 Feedback Control

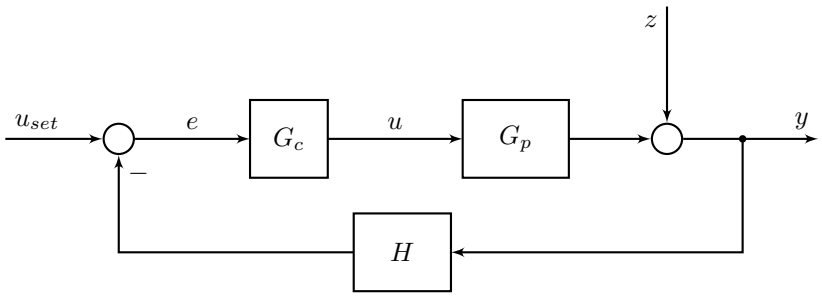


Figure 2.3: Block diagram of a feedback system.

A feedback system is fundamentally a system where the plant input is a function of the desired output signal, the *setpoint*, and the measured actual system output, the *process variable*. In other words, a system that actively adapts its output based on information about the difference between where it is and where it desires to be [1]. A block diagram of a generic feedback system is shown in Fig. 2.3.

Here, u_{set} is the setpoint, y denotes the process variable, z is some disturbance from an external source and e is the error function. G_c and G_p are the system controller transfer function and the plant transfer function respectively, i.e. the controller and what we want to control. H is some feedback transfer function that converts the measured process variable y to a unit that is comparable with the setpoint u_{set}, e.g. a strain gauge that relates strain to voltage. In the Laplace domain, the system output y, i.e. the process variable, can be expressed as the sum of the setpoint input response $y_{u_{set}}$ and the disturbance input response y_z [1]:

$$y(s) = y_{u_{set}}(s) + y_z(s) \longrightarrow \frac{G_c(s) \cdot G_p(s) \cdot u_{set}(s) + z(s)}{1 + G_c(s) \cdot G_p(s) \cdot H(s)} \tag{2.8}$$

Further on, the error function e is the specific difference we want to measure, i.e. how far from the setpoint output currently is. Simplified, this difference can be seen as:

$$\text{error} = \text{setpoint} - \text{process variable}$$

Mathematically, the function can be expressed as Eq. (2.9) [1]:

$$e(s) = u_{set}(s) - H(s)y(s) \tag{2.9a}$$
$$e(s) = u_{set}(s) - H(s)(z(s) + G_p(s)G_c(s)e(s)) \tag{2.9b}$$
$$\rightarrow e(s) = \frac{u_{set}(s) - H(s)z(s)}{1 + H(s)G_c(s)G_p(s)} \tag{2.9c}$$

2.3 Brushless DC Modelling

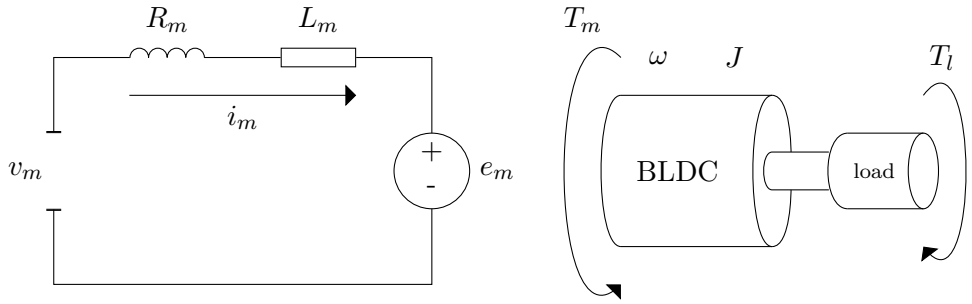


Figure 2.4: BLDC model [10].

The system actuator is essentially a BLDC motor. Mathematically, a BLDC can be expressed in terms of electrical and mechanical equations. The same fundamental equations that apply to a brushed DC motor also applies to a BLDC, in a simplified manner. The generic electrical and mechanical model is displayed in Fig. 2.4 [11].

In electrical terms, a BLDC can be expressed as a circuit. *Kirchoffs 2nd law* applied to the electrical circuit (on the left) yields (2.10) [7, 10]:

$$v_m = R_m \cdot i_m + L_m \frac{d}{dt} \cdot i_m + e_m \tag{2.10}$$

Table 2.1: Description of circuit symbols.

Symbol	**Description**	**SI**
i_m	Motor current	A
v_m	Motor voltage	V
e_m	Back-EMF voltage	V
R_m	Motor resistance	Ω
L_m	Motor inductance	H

In practice, the circuit equation in (2.10) applies to each of the armatures of a BLDC, e.g. for a three-phase BLDC this would yield $\sum v_k = v_a + v_b + v_c$ and would vary depending on position. For simplicity, we assume that the BLDC is one big coil [11].

By applying Newton's 2nd Law of rotational motion (2.1) on the right model, we get (2.11) [4]:

$$T_m = J \cdot \frac{d}{dt} \cdot \omega + B \cdot \omega + T_l \tag{2.11}$$

The relationship between back-emf voltage and motor speed is given by (2.12) [10]:

Table 2.2: Description of mechanical model symbols.

Symbol	Description	SI
T_m	Torque generated by BLDC	Nm
T_l	Load torque	Nm
J	Moment of inertia	$\mathrm{kg} \cdot \mathrm{m}^2$
ω	Angular velocity	$\frac{\mathrm{rad}}{\mathrm{s}}$
B	Friction coefficient	$\mathrm{Nm} \cdot \mathrm{s}$

$$e_m = k_e \cdot \omega \tag{2.12}$$

While the motor torque can be expressed in terms of (2.13) [10]:

$$T_m = k_m \cdot i_m \tag{2.13}$$

The back-emf constant k_e and motor torque constant k_m is here assumed equal, i.e. $K = k_e = k_m$ [11]. The equations can be simplified by assuming that there is no external load applied, i.e. $T_l = 0$. Instead, let J denote all potential inertia moments on the motor. After Laplace transforming both equations (2.10) and (2.11), the transfer function $G_{p,1}$, defining ω as plant output and v_m as plant input, can thus be obtained (2.14)[10, 1]:

$$G_{p,1} = \frac{\omega(s)}{v_m(s)} = \frac{1}{s^2 + (\frac{R_m}{L_m} + \frac{B}{J}) \cdot s + \frac{B \cdot R_m + K^2}{L_m \cdot J}} \tag{2.14}$$

It can be seen that the resulting transfer function is of 2nd-order. However, in an *ideal* motor it can be assumed that the viscous friction is small (especially in a BLDC due to no brush contact between rotor and stator) and therefore negligible, i.e. $B \approx 0\,\mathrm{Nm} \cdot \mathrm{s}$ [10]. Additionally, in most motors it can also be assumed that the electric circuit applies its output i_m to the motor much faster than the motor applies its speed to the load. This can be expressed as $\tau_{el} \gg \tau_m$, where $\tau_{el} = \frac{L_m}{R_m}$ and $\tau_m = \frac{JR_m}{K^2}$ are the electrical and mechanical time constants respectively [1]. In this case, the motor transfer function in (2.14) can be reduced to (2.15):

$$G_{p,1} = \frac{\frac{1}{K}}{\frac{JR_m}{K^2} \cdot s + 1} = \frac{k_1}{\tau_m \cdot s + 1} \tag{2.15}$$

Where k_1 is the DC gain. Further on, the transfer function can be expressed in terms of T_m as output by using the relation $\omega = \frac{T_m}{J \cdot s}$ [7], thus (2.16):

$$G_{p,2} = \frac{T_m(s)}{v_m(s)} = \frac{\frac{J}{K} \cdot s}{\tau_m \cdot s + 1} = \frac{k_1 k_2 \cdot s}{\tau_m \cdot s + 1} \tag{2.16}$$

Where $k_2 = J$. Now, by including the strain-torque relation from (2.2) and assuming that the shaft is restrained, i.e. fixed, at one end, the resulting transfer function $G_{p,3}$ relating shaft strain $\gamma = \frac{\varphi \cdot r_o}{L}$ and motor voltage v_m can be expressed as (2.17):

$$G_{p,3} = \frac{\gamma(s)}{v_m(s)} = \frac{\frac{J \cdot r_o}{K} \cdot s}{J_p \cdot G(\tau_m \cdot s + 1)} = \frac{k_1 k_2 k_3 \cdot s}{\tau_m \cdot s + 1} \tag{2.17}$$

The new gain value k_3 yields $\frac{r_o}{J_p G}$. The resulting open-loop gain now becomes $k_1 k_2 k_3 = \frac{J r_o}{K J_p G}$. Note that J now also includes the shaft moment of inertia, i.e. its "rotational mass" [4, 1].

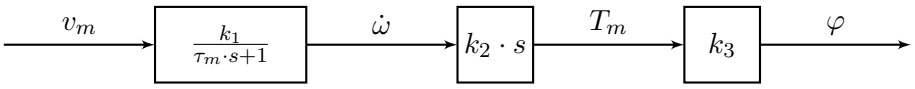


Figure 2.5: Estimated open-loop plant block diagram.

The block diagram for the transfer function $G_{p,3}$ can be seen in Fig. 2.5. It can be noted that the plant ultimately consists of a 1st-order lag element in series with a derivator. Since there are $N = 0$ free integrators in the transfer function the plant is of type-0 [1].

2.4 System Model

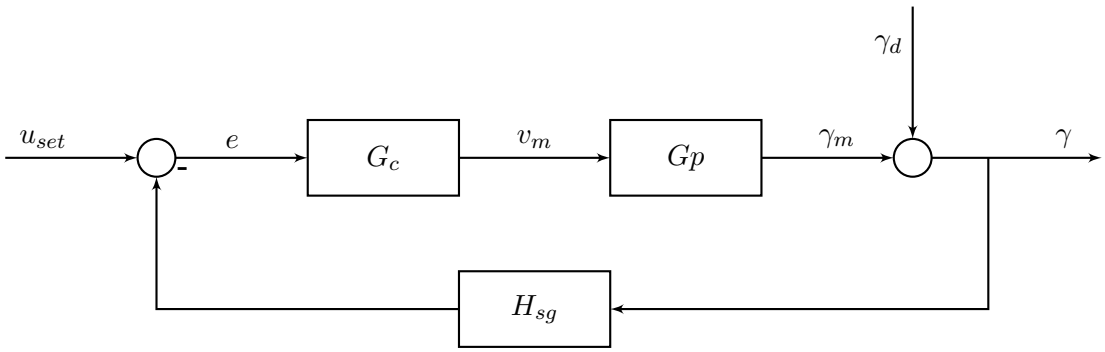


Figure 2.6: Block diagram of system feedback loop.

Now that the plant transfer function $G_p = G_{p,3}$ has been derived from section 2.3, a system model can be estimated. The block diagram in Fig. 2.6 displays the transfer function G_p applied to the feedback scheme from section 2.2. Here, γ_d denotes the disturbance strain. The process value γ now contains the sum of motor generated strain and load strain, i.e. $\gamma = \gamma_m + \gamma_d$. The sensor transfer function H_{sg} is the strain gauge transfer function, and relates strain as input and voltage as output.

2.4.1 Type-0 Error Offset

The theoretical error in steady state for a type-0 system by applying a step input can be expressed as follows (2.18) [1]:

$$e_{ss}(s) = \lim_{s \to 0} \frac{u}{1 + G_{ol}(s)} \tag{2.18}$$

Where u is the step value and $G_{ol}(s)$ is the open-loop tranfer function. By looking at Fig. 2.6 and assuming the derived plant transfer function $G_p = G_{ol}$, i.e. $H_{sg} = G_c = 1$ and no disturbance $\gamma_d = 0$ the following error equation is obtained for step input $u_{set} = \frac{u}{s}$ (2.19):

$$e_{ss}(s) = \frac{u}{1 + G_p(0)} = \frac{u}{1 + k_1 k_2 k_3} \tag{2.19}$$

It can be noted that since the system is of type-0 it will exhibit steady state offset errors for step inputs [1].

2.4.2 Non-Linear Friction Considerations

It is critical to note that the plant model derived in section 2.3 is simplified, and is valid for a BLDC torque applied to a fixed shaft. So far the only case considered is the case when the fixed shaft-end torque is static and has the same magnitude as the BLDC torque applied at the other end, i.e. $T_m = T_d$. However, the dynamic characteristics and the friction caused by the slip ring has not yet been considered. By revisiting the BLDC torque equation (2.11), let us rewrite it by adding the slip ring friction torque:

$$T_m = J \cdot \frac{d}{dt} \cdot \omega + B_m \cdot \omega + B_{sr} \cdot \omega \tag{2.20}$$

Here, the slip ring friction torque is now embedded in the BLDC as $B_{sr} \cdot \omega$, with B_{sr} as the slip ring friction. It was assumed earlier that the internal friction of the BLDC could be neglected, i.e. $B_m \approx 0$ However, this is not the case with the slip ring friction B_{sr} [5, 4].

The friction caused by the slip ring is quite substantial and is therefore not negligible. Additionally, the friction coefficient B_{sr} is not constant and instead highly dependent on the velocity ω, which means $B_{sr}(\omega)$. The slip ring friction torque $B_{sr}(\omega) \cdot \omega$ therefore results in non-linear plant characteristics, which makes a precise mathematical system model challenging to derive [1].

Chapter 3

Method

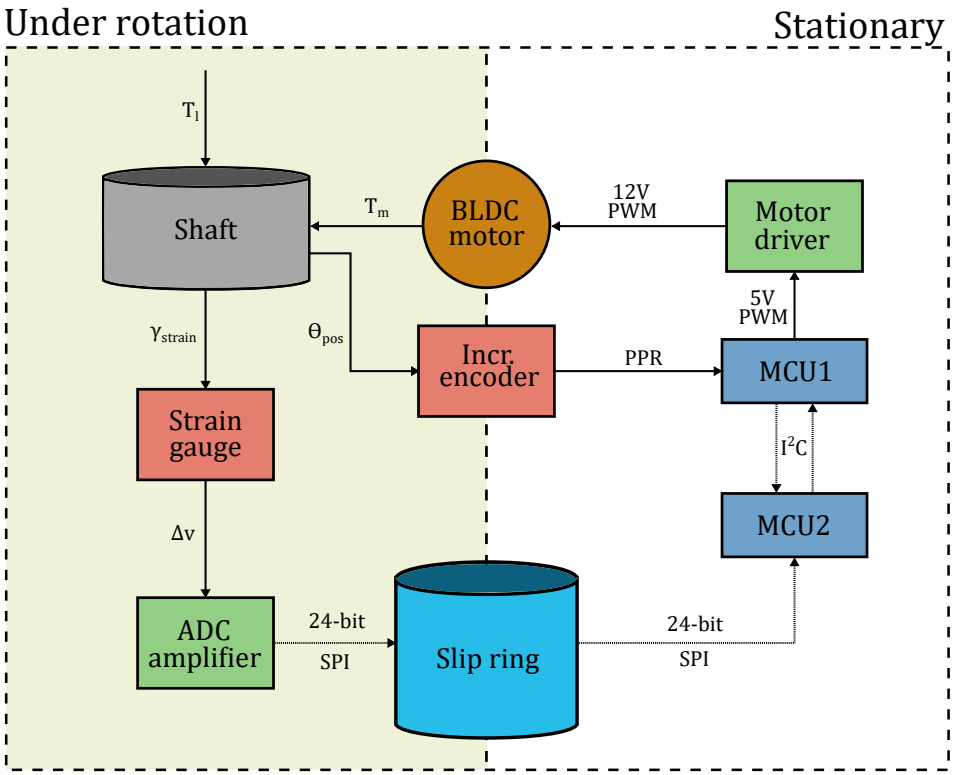


Figure 3.1: System overview flowchart.

In this chapter, the project approach and the methods implemented will be covered. A flowchart displaying the simplified system setup is shown in Fig. 3.1.

3.1 Strain Gauge Signal Amplification

In order to measure the torque differential between the rotational and stationary parts; strain gauges are implemented. Strain gauges are used in a variety of industrial applications as strain sensors due to their low cost and high sensitivity [12, 13]. For our application, it can be used to measure the pipe angular displacement φ ((2.2)) covered in section 2.1 and hence the applied torque.

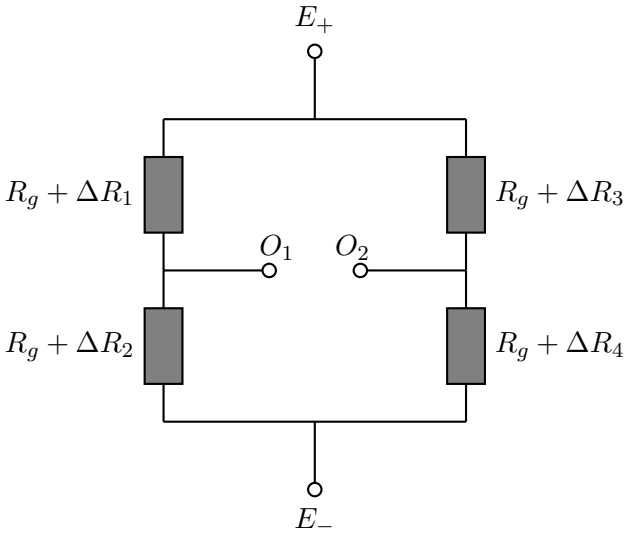


Figure 3.2: Strain gauges in Wheatstone bridge configuration.

Fundamentally, a strain gauge consists of a conductive material that changes its electrical resistance when subjected to strain, i.e. $R_n(\varepsilon) = \Delta R_n(\varepsilon) + R_g$, where ΔR_n is the change in resistance caused by strain ε and R_g is the un-deformed gauge resistance [14]. Strain gauges in a full Wheatstone bridge configuration can be seen in Fig. 3.2. These circuits are particularly useful due to their high accuracy and differential characteristics, e.g. their ability to cope with temperature changes and other disturbances that affect the circuit in total [12]. The 250US full bridge strain gauge from Micro Measurements was used for the project, as displayed in Fig. 3.3 [15]. These gauges are ideal for torque strain measuring due to the 45° gauge offset, which is approximately where the shear strain is most significant (see Fig. 2.2) [8].

Since the strain gauge output differential voltage $\Delta O = O_1 - O_2$ is quite small due to minuscule changes in strain gauge resistance, an *integrated circuit* (IC) that amplifies the signal is needed [16]. The *Sparkfun Load Cell Amplifier* board, based on the HX711 24-bit analog-to-digital (AD) converter, was used for this purpose [17, 18]. Essentially the board reads the output voltage pads of the gauge (GRN and WHT), amplifies the differential using some gain and transmits the readings to the data pin (DAT) at a rate proportional to the input clock pin (CLK). By default, the amplification gain is set to 128, which results in a detection resolution of ±40 mV given 5 V supply voltage.

From the HX711 datasheet [17], the output data rate for the board when using an external clock source has 2 main modes. The modes are toggled by setting the input to the RATE pin either equal to VCC (MODE2) or pull it LOW (MODE1), and this can be done by soldering a jumper underneath the board. By default, the RATE pin is pulled low. In this mode the output data rate is given by the following function:

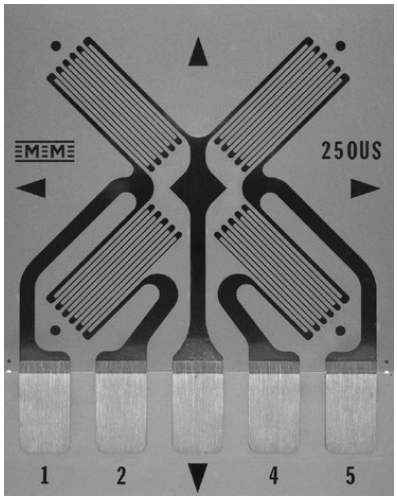


Figure 3.3: 250US from Micro Measurements [15].

$$f_{out} = \frac{f_{clk}}{1105920} \text{ Hz}$$

Since the clock oscillator frequency for our Arduino is $f_{clk} \approx 16$ MHz the maximum output data rate yields $f_{out} \approx 14.46$ Hz [19]. The input noise for this mode equals to typically 50 nV(rms). The other mode, i.e. when RATE pin equals VCC, has the following output data rate function:

$$f_{out} = \frac{f_{clk}}{138240} \text{ Hz}$$

It can be noted that this allows for a much higher rate, as much as $f_{out} \approx 115.74$ Hz for our oscillator. On the other hand, this mode also increases the input noise to about 90 nV on average [17]. Therefore, a compromise between noise and output rate needs to be made. Both modes will therefore be tested and covered in section 4.1 to see which is most desirable.

3.2 L6234 Driver and Three-Phase BLDC Control

For actuation in the system, a *brushless direct current* (BLDC) motor was implemented. More precisely, a Turnigy HD 5208 Gimbal BLDC with 12 cogs and 7 pole pairs [20]. BLDC motors have become a popular choice for motor control due their high efficiency and low maintenance [11]. For the purpose of this project, a BLDC was used due to its small *dead space*, as compared to regular brushed DC motors [10]. This section will cover the basic working principle of a BLDC motor and the project implementation.

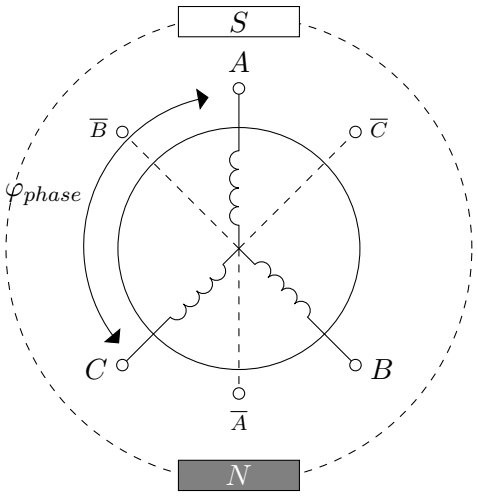


Figure 3.4: Three-phase BLDC setup.

Fundamentally, a BLDC consist of a stationary and a rotary part, i.e. the *stator* and *rotor* respectively. The stator consist of several windings of coils that generates a magnetic flux when current is applied. The rotor consist of permanent magnets with alternating polarity. Figure 3.4 displays one of the most common BLDC arrangements: the *three-phase* BLDC. Here, 3 electro magnets (A, B and C) are seperated by a phase of $\varphi_{phase} = 120\ ^\circ$. By applying a positive voltage to all coils (A, B and C) and letting the current exit at the opposite end of the coils (marked as $\overline{A}$, $\overline{B}$ and $\overline{C}$), torque is generated on the rotor by the stator. By manually rotating the rotor poles (N and S) 360 electrical degrees, the torque acting on the rotor varies with position. Figure 3.6 shows the theoretical rotor torque from each coil set (A, B and C) with respect to angular position φ. Note that this is the case when current is constant (i = const.) for all coils and that the torque sum, $\sum T$, for all steps is not constant. In order to get more consistent torque, the current needs to be varied. The right plots in Fig. 3.7 shows the current waveforms applied in order to achieve more or less constant torque characteristics, i.e. the torque sum $\sum T$ each step is constant, shown in the left plot. It can be seen that the current waveforms for B and C are the same as A, only each have a added phase of $\varphi_{phase} = 120\ ^\circ$ [21].

The SMT L6234 three-phase motor driver is used in order to control the BLDC [22]. The Arduino MCU itself cannot handle currents $> 50\,\text{mA}$, which makes motor current control directly from the Arduino challenging [19]. However, the L6234 IC allows voltages up to 52 V and 5 A bidirectional current which makes it viable for a range of BLDC applications [22]. The chip used for this project is embedded in a open-license circuit board designed by Michael Anton [23]. The board includes features such as pin headers and over-current

protection. Essentially, the chip works by connecting a supply voltage, i.e. the voltage to the motor, and controlling the voltage applied to each phase by a *pulse-width modulated* (PWM) signal from the MCU to the IN1, IN2 and IN3 pins on the chip [24].

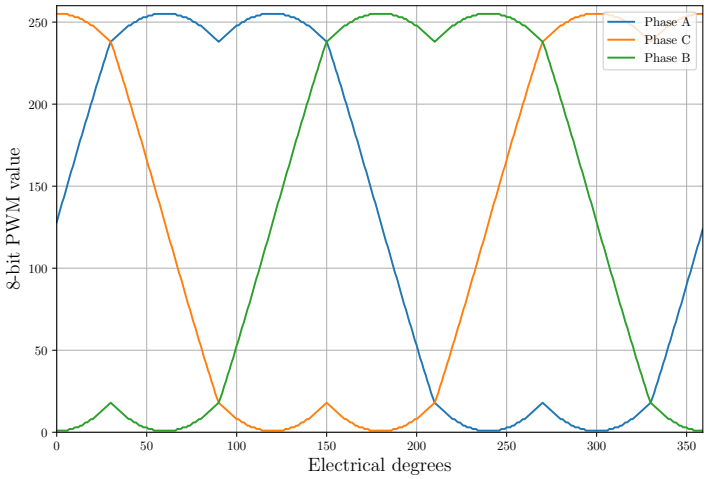


Figure 3.5: State vector PWM waveforms [25].

In order to achieve the current waveforms seen in Fig. 3.7 the L6234 Application Note suggests a 6-step sequence (per 360 electrical degrees) consisting of HIGH (5 V) and LOW (0 V) rectangular voltage pulses [24]. However, since the 6-step sequence may be more suited for high-speed applications due to "choppy" steps, the sequence can be smoothed and divided into smaller steps by adjusting the PWM *duty cycle*. Therefore, a 360-step PWM sequence will be used instead, derived from *state vector modulation* (SVM) that minimizes voltage drops. The sequence is based on look-up tables provided by BerryJam-user Ignas Gramba [25, 26]. Since the BLDC motor used has 7 pole pairs, each mechanical revolution consists of $7 \cdot 360 = 2520$ steps, i.e. ≈ 0.14 mechanical degrees per step, which should ensure no oscillating ("choppy") BLDC speed.

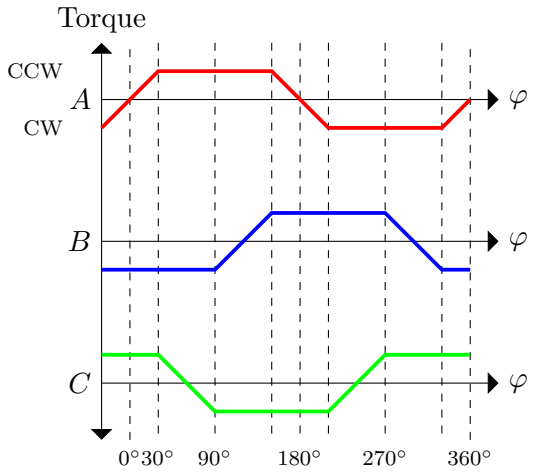


Figure 3.6: 3-phase BLDC torque waveforms when $i = \text{const.}$ [21].

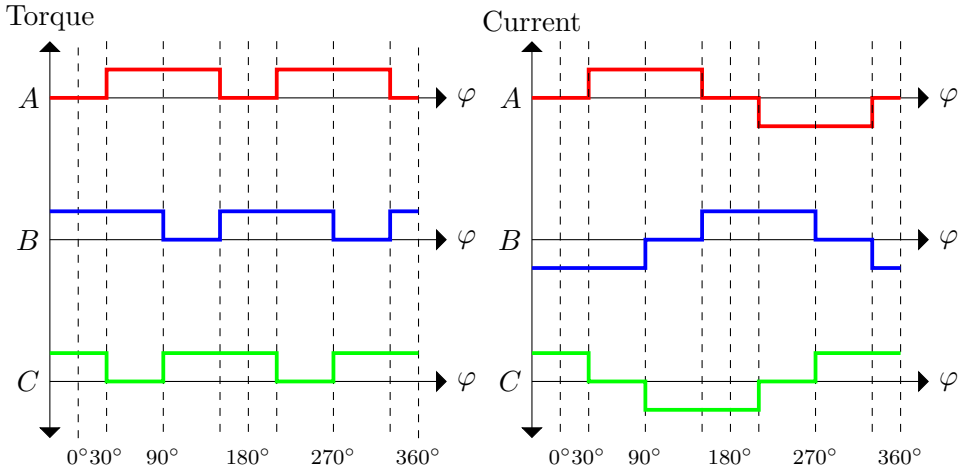


Figure 3.7: 3-phase BLDC torque waveforms with varying current [21].

3.2.1 Commutation by Encoder Feedback

In order to control the commutation of the BLDC, i.e. when it is time to rotate the magnetic field vector, some knowledge about rotor position is needed. There are several ways of acquiring position feedback, e.g. hall-effect sensors and back-EMF measurement, but for this project an encoder was used [24, 27]. The encoder is the AMT20 from CUI that allows both incremental and absolute encoder measurements, with resolutions of 1024 *pulses per revolution* (PPR) and 12-bit (4096) respectively [28]. This yields $360/1024 \approx 0.35$ degrees per step. Additionally, the incremental data outputs on 2 channels, A and B, which allows direction detection and 4 times higher resolution ($4 \cdot 1024 = 4096$). The signal protocols for channel A and B are displayed in Fig. 3.8. For the purpose of this project, 1024 PPR should be sufficient resolution for incremental readings.

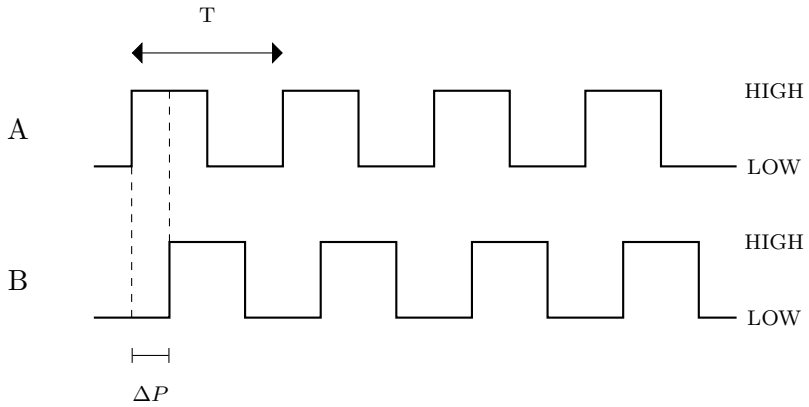


Figure 3.8: CCW quadrature signal protocol for line A and B from the AMT20 datasheet [28].

By using the position data provided by the encoder, the BLDC *commutation*, i.e. when the magnetic field should rotate to the next step, can be derived. A commonly used reference that draws similarities to BLDC control is the "horse and the carrot"-metaphor. The magnetic field (the carrot) is constantly being chased by the rotor (the horse), but

will always be one step ahead. The same principle applies for both applications. One question that arises in this regard is: What should be the magnetic field position relative to the rotor position? In other words; what is the optimal commutation angle φ_{com}? By inspecting the torque plot by coil A in Fig. 3.6 it can be noted that max CCW torque $T_{A,max}$ occurs in the range $[30°, 150°]$. This is the case for an ideal motor, and in reality the torque in this range will vary but usually peak at $\approx 90°$. Therefore, the commutation angle for optimal torque is placed at $\varphi_{com} \approx 90°$ [29]. Note that this is not mechanical degrees, but rather electrical. Since the PWM sequence already has 360 steps, i.e. one per electrical degree, the sequence position for A can be set using the following equation:

$$\text{Step A} = \text{El deg per pulse} \cdot \text{Pulses} + 90$$

3.3 Stationary Base Design

The following section contains the description of the stationary base sketching and construction. The base in its entirety can be seen as a rendered model in Fig. 3.9. Initially, the general idea was to build a solid base using easily accessible materials and tools. The initial conditions were to make a base that could support the weight and thrust generated by a quadcopter, and that the base itself would not exceed the 300 mm × 300 mm range on the horizontal plane. All the electronics and ICs would then be contained within the base.

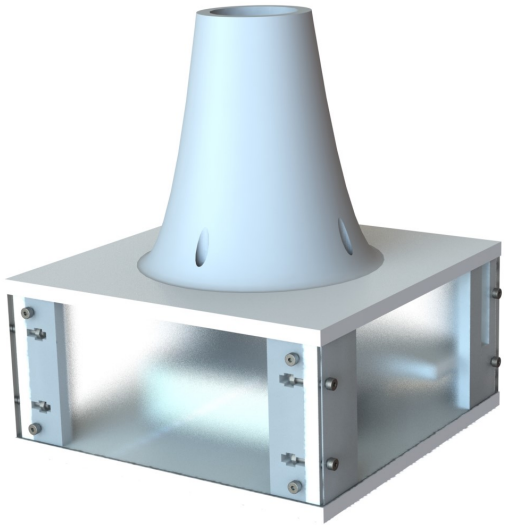


Figure 3.9: Rendered CAD model of the base.

In the initial phase, the idea was to use processed plates of 18 mm wood as the material for the top and bottom plates. The plates were to be cut out using a laser cutter available at the university. However, initial testing showed that the plates were to thick for the laser to cut through. Therefore, it was decided to use plates of 4 mm plexi glas instead. Plexi yielded much more precise and clean cuts, and layers of 3 plates ($3 \cdot 4$ mm = 12 mm)

were used to make sure the parts became rigid enough and could support the load. The base interior was made to fit all necessary components, and screw holes allowed locking the parts in place. Locking nuts were used instead of regular nuts in order to withstand vibrations without loosening.

3.3.1 Air-Thrust Considerations

Figure 3.10: CAD-model of conic pipe for air-thrust redirection.

As mentioned in the problem statement in section 1.1, a way to effectively redirect the air thrust generated by the quadcopter needed to be implemented. The air could exert a substantial force on the base top plate if the air flow was allowed a perpendicular contact angle, and potentially disturb the quadcopter operation as well [4]. The cone shaped part shown in Fig. 3.10 is the component used in order to spread the air flow. The sides of the “cone” is shaped using a parabolic function.

3.4 Microcontroller Logic

Fundamentally, the controller logic is responsible for processing all the sensor inputs and assigning output signals to the BLDC driver. The processing workload is divided between two *micro controller units* (MCUs); 1 Sparkfun Pro Micro and 1 Arduino Micro [19, 30]. This is done in order to ensure that the controllers are able to process the required calculations, especially since the BLDC commutation control depends on pin-change interrupts from the encoder. Therefore, one MCU is assigned with controlling the speed and commutation of the BLDC (MCU1), while the other is assigned with reading strain measurements and the PI controller algorithm for calculating the next speed output (MCU2). The I^2C bus works as the communication platform between the MCUs and is covered in subsection 3.4.5.

3.4.1 L6234 and BLDC Control

In order to control the L6234 motor driver, the MCU1 is assigned with the task of encoder interpretation, commutation and motor voltage control. The MCU1 code in its entirety can be found in Appendix A. In order to implement the sequences shown in Fig. 3.5 in code all the 360 PWM values for phase A was assigned to a global array `pwmArray[360]`. Indices for each phase (A, B and C) are then declared, shifted 120 degrees apart.

In order to acquire encoder data, a *Pin Change Interrupt* (PCINT) is assigned to the encoder output pin A [31]. By inspecting the CCW signal protocol from Fig. 3.8 it can be seen that the A line goes HIGH prior to B with $\Delta P = 90\,^\circ$ of the cycle T. Line B goes first if the direction is reversed. With this information, both direction and position of the shaft can be found. The code snippet in Listing 3.1 shows the function (ISR) that triggers whenever line A goes from LOW to HIGH, i.e. when A meets a rising edge, and by checking the state of line B can determine whether the position value `encoderCounter` should increment or decrement.

```
1  void newPulseA() {
2    if (digitalRead(encoderPinB) == LOW)
3    {
4      encoderCounter--;
5    }
6    else
7    {
8      encoderCounter++;
9    }
10 }
```

Listing 3.1: PCINT ISR attached to channel A on rising edge.

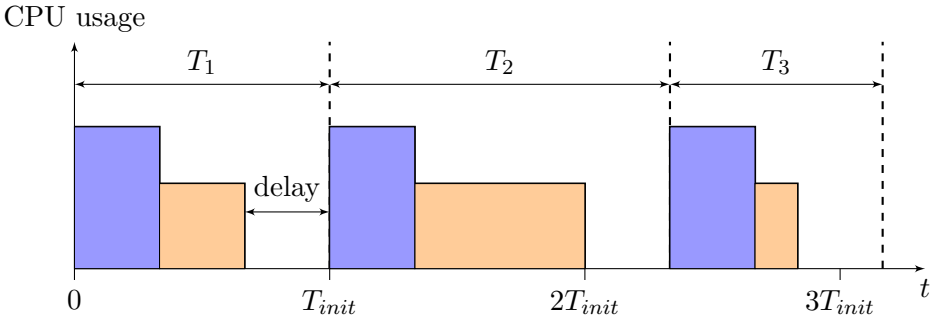
Since the incremental mode of the encoder is used there is no information about absolute position available, i.e. we can only find the relative position from where it was initialized but not the position at $t = 0\,\mathrm{s}$ [10]. Therefore, a way to define a starting point needs to be implemented. A way to do this is by positioning and holding the motor in a fixed part of the 360-degree PWM sequence and use this as the relative starting position [27]. A function `bldcInitializePosition()` is therefore called in in `setup()` and executes the steps mentioned above. The full function is shown is Listing 3.2.

```
void bldcInitializePosition() {
  indexA = 0; // Set A index
  indexB = phaseShift; // Set B relative to A
  indexC = indexB + phaseShift; // Set C relative to B
  digitalWrite(en1, HIGH);  // Enable PWM for A
  digitalWrite(en2, HIGH);  // Enable PWM for B
  digitalWrite(en3, HIGH);  // Enable PWM for C
  analogWrite(in1, pwmArray[indexA]); // Set PWM voltage value for A
  analogWrite(in2, pwmArray[indexB]); // Set PWM voltage value for B
  analogWrite(in3, pwmArray[indexC]); // Set PWM voltage value for C
  delay(2000);  // Let the motor reach the flux vector position and hold it there
  encoderCounter = 0; // Make sure encoderCounter is reset
}
```

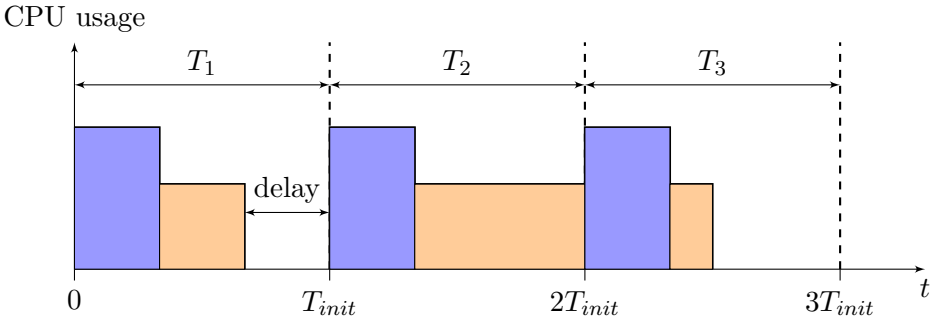
Listing 3.2: Initialization of encoder and BLDC start position.

3.4.2 Timer Interrupts and Consistency

In order to run the code at a consistent rate, the main functions in MCU2 is executed by a timer interrupt. Timer interrupts are essentially used to trigger an event, i.e. an *interrupt service routine* (ISR), whenever a counter reaches a certain threshold value [32]. The ISR will trigger as long as interrupts are enabled, no matter which part of the code is currently being executed. The difference between a code iteration executed after a delay and code executed as an ISR can be seen in Fig. 3.11. Note that the periods in the upper plot is not consistent, i.e. $T_1 \neq T_2 \neq T_3$, while the lower plot yields consistent iteration periods despite variable computation time.



(a) Main loop iterations with time delay.



(b) Iterations from a timer ISR.

Figure 3.11: Comparison of time-delay and ISR iterations.

The on-board timers embedded in MCU2's ATmega32U4 chip can be configured to allow such interrupts [19]. The code snippet in Listing 3.3 assigns a timer interrupt to 16-bit

Timer/Counter3 that triggers an ISR with a frequency of 10 Hz, i.e. once every 0.1 s. Here, the `initializeTimerInterrupt()` function is called only once in `setup()`, which enables the `ISR(TIMER_COMPA_vect)` routine. The frequency depends on the master clock frequency f_{clk} and can be modified by changing the prescaler value in `TCCR3B` and the value of the `OCR3A` register [19]. The following function can be used to calculate the ISR frequency f_{isr}:

$$f_{isr} = \frac{\text{master clock frequency}}{\text{counter} \cdot \text{prescaler}} = \frac{16\ \text{MHz}}{6250 \cdot 256} = 10\,\text{Hz} \tag{3.1}$$

```
1  void initializeTimerInterrupt() {
2    TCCR3A = 0;           // Clear register TCCR3A
3    TCCR3B = 0;           // Clear register TCCR3B
4    TCCR3B |= (1 << WGM32);   // Set the "clear timer on compare match" (CTC) mode
5    TCCR3B |= (1 << CS32);    // Set prescaler to 256
6    TIMSK3 |= (1 << OCIE3A);  // Enable interrupt on match
7    TCNT3 = 0;            // Reset
8    OCR3A = 6250;           // Counter value. (6250 for 10 Hz, 625 for 100 Hz etc.)
9  }
10
11 ISR(TIMER3_COMPA_vect) {
12                 // Add the ISR executions here
13 }
```

Listing 3.3: Initializing 10 Hz ISR on Timer/Counter3 for ATmega32U4.

3.4.3 HX711 Data Sampling

In order to read the data from the HX711 ADC the code snippet in Listing 3.4 was used. The `readInput()` function receives and assigns the readings to the variable `strainInput` and compensates for the offset by moving it to zero-range. The `strainGaugeADC.read()` function derives from the `<Q2HX711.h>` public library from GitHub user "*queuetue*" [33].

```
1  void readInput() {
2    strainInput = strainGaugeADC.read() ;
3    strainInput -= rawSetpoint;
4    if (strainInput < HIGHER_ERROR_OFFSET && strainInput > LOWER_ERROR_OFFSET) {
5      strainInput = strainSetpoint;
6    }
7  }
```

Listing 3.4: Function for reading HX711 data.

3.4.4 PI Controller Algorithm

A PI controller algorithm is used to process the strain gauge inputs. The PI controller variant is chosen due to the estimated offset error of type-0 systems, as covered in subsection 2.4.1. PI controllers, i.e. variants of the *proportional-integral-derivative* (PID) controller, are used extensively in a range of applications [1]. Fundamentally, the PI controller consists of two terms: A proportional term that scales the input signal and an integral term that accumulates the input signals each iteration. The following function depicts the output of a PI controller in time domain [1].

$$u(t) = u_p(t) + u_i(t) = K_p \cdot e(t) + K_i \int_0^t e(\tau)d\tau \tag{3.2}$$

Where $u(t)$ is the output and K_p and K_i are the proportional and integral gains respectively. In the Laplace domain, the PI controller transfer function can be derived as:

$$U(s) = K_p \cdot E(s) + \frac{K_i}{s} \cdot E(s) \tag{3.3a}$$

$$\longrightarrow G_c(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} \tag{3.3b}$$

Essentially, the PI controller acts on both current and previous errors. The P-term output depends entirely on the current error and the gain value K_p, while the I-term accumulates previous errors based on time, error and the gain value K_i. Since the MCU can not output signals of continuous time, the function in (3.2) needs to be adapted to the discrete time domain. This can be achieved by applying *Riemann sums* and *finite differences* [34]:

$$u(t) \approx K_p \cdot e_n + K_i \sum^{n} e_n \cdot \delta t \tag{3.4}$$

Here, the *timestep*, i.e. the time between each iteration, is denoted as δt. The code snippet in Listing 3.5 shows the PI controller algorithm uploaded to the MCU2 [35].

```
1  void PIcontrol() {
2    errorSignal = strainSetpoint - strainInput; // Calculate the error
3    pTerm = Kp * errorSignal;                   // Calculate the P-term
4    iTerm += Ki * errorSignal * TIMESTEP;       // Accumulate the I-term
5    if (iTerm > MAX_OUTPUT) {                   // Check for I-term overflow
6      iTerm = MAX_OUTPUT;
7    }
8    else if (iTerm < MIN_OUTPUT) {
9      iTerm = MIN_OUTPUT;
10   }
11   controllerOutput = pTerm + iTerm;           // Calculate PI controller output
12   if (controllerOutput > MAX_OUTPUT) {        // Check for output overflow
13     controllerOutput = MAX_OUTPUT;
14   }
15   else if (controllerOutput < MIN_OUTPUT) {
16     controllerOutput = MIN_OUTPUT;
17   }
18 }
```

Listing 3.5: PI controller algorithm

3.4.5 I²C Arduino Communication

As mentioned in the start of the section, the controller logic was divided between 2 MCUs due to the need for consistent data rates. In order to allow the Arduinos to exchange data with one another, a communication bus needs to be implemented. The data that is desired to exchange between the units are encoder data (from MCU1 to MCU2) and BLDC speed command (MCU2 to MCU1). There are several accepted serial buses embedded on the ATmega chips, e.g. SPI and UART, but the *Inter-Integrated Circuit* (I^2C) bus is chosen for this project [36].

One of the advantages with I^2C is the combination of synchronous high data rate and few lines. The typical data speed range is 100.0 $\frac{kb}{s}$ or 400.0 $\frac{kb}{s}$, which is much more then needed for our application which lies approximately in the 3.6 $\frac{kb}{s}$ range [1]. Additionally, only 2 signal lines are needed, one data *signal line* (SDA) and one *clock line* (SCL). For simplicity, the protocol essentially works by the master sending an 7-bit address, corresponding to one of the slave devices, followed by an *acknowledgement-bit* that indicates whether the slave is ready to receive data or not [36]. Our application only consists of one master (MCU2) and one slave (MCU1) with a given address of `SLAVE_ADDRESS = 0x32` (50). The code used by the master and the slave are displayed in Listing 3.6 and Listing 3.7, respectively [37].

```
1  void sendOutput() {
2    uint8_t speedCommand = controllerOutput + 0.5;  // Round up output
3    Wire.beginTransmission(SLAVE_ADDRESS);  // Notify slave of incoming data
4    Wire.write(speedCommand); // Send 8-bit speed command
5    Wire.endTransmission();   // End transmission
6  }
7
8  void requestEncoderInput() {
9    uint8_t n = Wire.requestFrom(SLAVE_ADDRESS, 2); // Request 2 bytes from slave
10   uint8_t encoderBuffer[2]; // Declare 8-bit buffer array with 2 indecies
11   for( int i = 0; i < n; i++) { // Append 1 byte at a time to buffer from data
         line
12     encoderBuffer[i] = Wire.read();
13     }
14   encoderPosition &= 0x00;  // Clear encoderPosition by setting all bits to 0
15   encoderPosition |= encoderBuffer[1];  // Add the 1st byte to buffer
16   encoderPosition |= (encoderBuffer[0] << 8);   // Shift the buffer by 1 byte and
         add the remaining bits from data line
17 }
```

Listing 3.6: I^2C implementation for master device MCU2.

[1] Approximated value

```
void receiveSpeedCommand() {
  uint8_t tempBuffer = Wire.read(); // Declare buffer and add byte from master
  speedControl = map(tempBuffer, 0, 255, -100, 100);  // Map the value to correct
      range
  if (speedControl < 0.0) { // Assign BLDC direction
    bldcDirection = false;
    speedControl *= -1.0;
  }
  else {
    bldcDirection = true;
  }
  speedControl /= 100.0;
  }

void sendEncoderValue() {
  uint16_t tempEncoderValue = encoderCounter; // Assign encoderCounter to a 16-bit
       buffer in case of PCINT during transmission
  encoderBytes[0] = tempEncoderValue >> 8;  // Add the 10th and 9th bit (MSB) to
      first position
  tempEncoderValue &= 0xFF; // Remove 9th and 10th bit from int
  encoderBytes[1] = tempEncoderValue; // Add the 2nd byte (LSB) to 2nd position
  Wire.write(encoderBytes, 2);  // Transfer to master
}
```

Listing 3.7: I^2C implementation for slave device MCU1.

3.4.6 Recursive Signal Filtering

Since the raw samples from the HX711 ADC could contain noise, a way to potentially filter the readings were implemented on the controller MCU2. There are several good data filtering algorithms. However, since the Arduino filters in real-time, many continuously running filters also come with a substantial amount of phase-lag [38]. In order to avoid this obstacle, a simple recursive filter was implemented. The filter can be expressed using the following function (3.5):

$$y[n] = \alpha \cdot x[n] + (1 - \alpha) \cdot y[n - 1] \tag{3.5}$$

Here, $y[n]$ and $x[n]$ denotes the filter output and sample input respectively, at some timestep n. The coefficient $\alpha \in [0.0 \;\; 1.0]$ determines the filter strength, i.e. the weight of the filter. For high values of α the filtering is small, while low values of α yields greater amounts of filtering. The code used can be seen in Listing 3.8. The function `filterInput()` is called after strain sampling, where `rawStrainInput` is assigned with the raw sample value [38, 39].

```
void filterInput() {
filteredInput = alpha * rawStrainInput + (1 - alpha) * lastFilteredOutput;
lastFilteredOutput = filteredInput;
}
```

Listing 3.8: Simple recursive LP signal filter for MCU2.

3.5 The Ziegler-Nichols Method

The *Ziegler-Nichols* (ZN) method is an approach for the tuning of different variants of the PID controller and estimating desirable gain values. The method is especially useful since it can be used as an experimental approach when gain values are not acquired from mathematical models [1]. The method used for estimating tuning parameters in this project is known as the 2nd ZN method, and the results are covered and discussed in section 4.3 and 5.2, respectively.

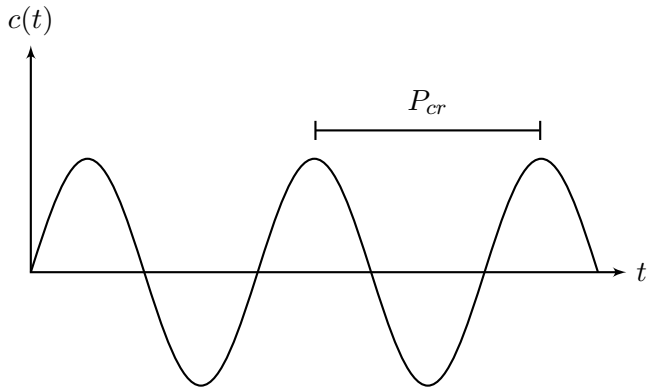


Figure 3.12: Sustained oscillations in transient response.

The method is implemented by using the proportional (P) controller only. From the feedback block diagram in Fig. 2.6, this yields $G_c(s) = K_p \cdot E(s)$. The next step is to increase the gain value K_p from 0 to some critical gain K_{cr} where the system first becomes marginally stable. This is characterized by sustained oscillations in the transient response, i.e. oscillations that neither increase in amplitude (instability) or phases out (stability) over time. The oscillation period P_{cr}, i.e. the critical period, is the period of the marginally stable system response [1]. If the system exhibits this behaviour at some critical gain K_{cr}, the PI controller parameters can be determined as listed in Table 3.1:

Table 3.1: Gain values determined from the 2nd ZN method.

PI parameters	**Formula**
K_p	$0.45 \cdot K_{cr}$
K_i	$0.54 \cdot \frac{K_{cr}}{P_{cr}}$

If applicable to the system, the method can be useful as a guideline for PI controller parameters.

Chapter 4

Results

The following chapter contains the results from testing the compensation system in correlation with the method chapter. Fundamentally, the test were carried out with the system assembled in place, as shown in Fig. 4.1. The load cycle experimentation were undertaken by constructing a stationary configuration that allowed an load actuator, i.e. motor, to be placed concentric above the shaft. The following boundary conditions were followed in line with the problem statement QC specifications:

- Peak rotational speed of $\omega_d = \pm 360.0 \, \frac{\text{deg}}{\text{s}} = \pm 60.0 \, \text{rpm}$
- Peak acceleration of $\alpha_d = \pm 360.0 \, \frac{\text{deg}}{\text{s}^2}$

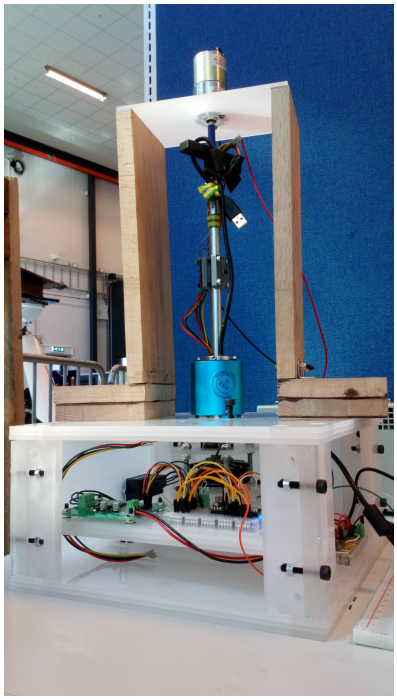
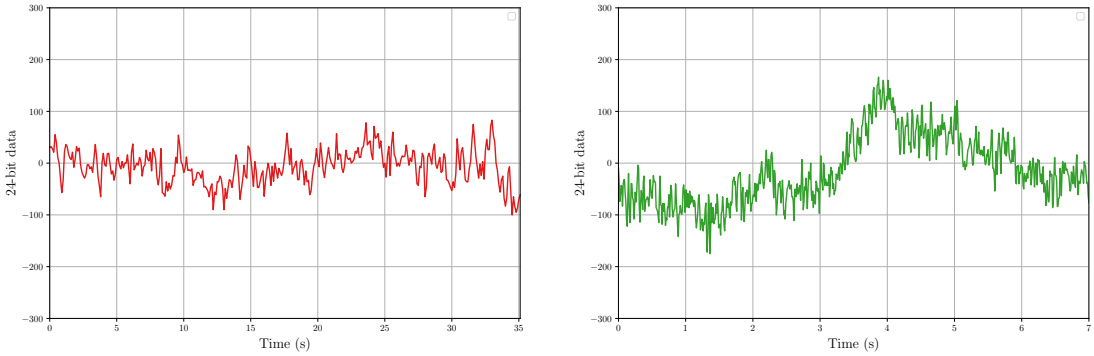


Figure 4.1: Test setup with geared DC motor on top.

However, it should be noted that the load actuators available were not capable of reaching the peak velocity. Instead, velocities of $\approx$ 30 rpm were achievable [40]. Additionally, the tests were carried out without purposely inflicting normal bending strain that could occur when the drones centre of mass is not coincident with the shaft vertical axis, i.e. when a bending moment acts on the shaft. Therefore, the system response to such loads has not been tested.

4.1 Strain Gauge Sampling

As discussed in section 3.1, the HX711 ADC contains 2 modes for output data rate, *MODE1* with up to 14.46 Hz (low noise) and *MODE2* with up to 115.74 Hz (additional noise). The 2nd mode can be accessed manually by cutting the RATE pin jumper on the underside of the load cell amplifier board. Additionally, a switch connector was soldered between the open connections allowing for easy toggling between the modes. The modes were tested using an timer ISR executed at 10 Hz and 100 Hz, for MODE1 and MODE2 respectively. It should also be noted that the HX711 chip used the maximum value of sampling gain, i.e. 128. This section will cover the results from testing the raw output data through the slip ring for both modes, with no rotation of the shaft, i.e. static and no torque load.



(a) Samples at 10 SPS.

(b) Samples at 100 SPS.

Figure 4.2: Sampling mode comparison.

From testing of the 1st mode, the results can be seen as plotted left in Fig. 4.2. As mentioned, this test was conducted using a ISR executed every 100 ms, which yields 10 readings each second. The plot values are added with a constant offset of 7552025 in order to move the data to the 0-range and make it more representable. It can be seen that the samples oscillates with a peak differential of $\Delta P_{10} \approx 200.0$ and does not exceed this limit for the 35 seconds of test duration.

Further on, the results from the 2nd test is displayed on the right in Fig. 4.2, which was done using an ISR executed in 10 ms intervals. It can be noted from the plot that there is an increased amount of noise present, as much as $\Delta P_{100} \approx 370.0$ at most. It can also be noted that the values not only oscillates, but also changes over multiple samples in longer periods of time, e.g. from $t \approx 3.0$ s to $t \approx 4.0$ s, and then drops again. This

behaviour also occured in different test runs with longer time durations, but similarly the peak differential never exceeded that of 400.0.

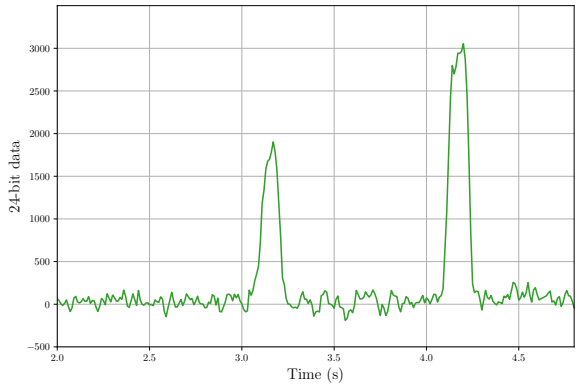


Figure 4.3: Strain readings at 100 SPS with small torque pulses.

In order to decide which mode is most suitable for the application, or more precisely; decide if the noise generated during MODE2 is small enough to sample reliable strain readings, the strain during small stationary load pulses was tested. By applying small torque loads that does not exceed the stationary friction torque, i.e. keeps the shaft stationary, the detection resolution during MODE2 can be detected. The results are displayed in Fig. 4.3 with 100 SPS (MODE2), where the small torque pulses (at $t_1 \approx 3.1\,\mathrm{s}$ and $t_2 \approx 4.2\,\mathrm{s}$) were applied by manually twisting the shaft. It can be noted that the data samples that occur whenever a load is applied exceeds the noise range by a great amount. Therefore, MODE2 seems like the best choice for this application since it is capable of coping with noise and due to the desire for the lowest amount of system phase lag, which immediately increases with lower sampling rate (MODE1).

4.2 Filtering of HX711 Samples

The filtering of the HX711 MODE2 signals could provide improvements to the overall system performance and was therefore tested. As covered in section 3.4.6, a simple recursive filtering algorithm were implemented. This section will cover the results from testing the filtered samples with different values of α.

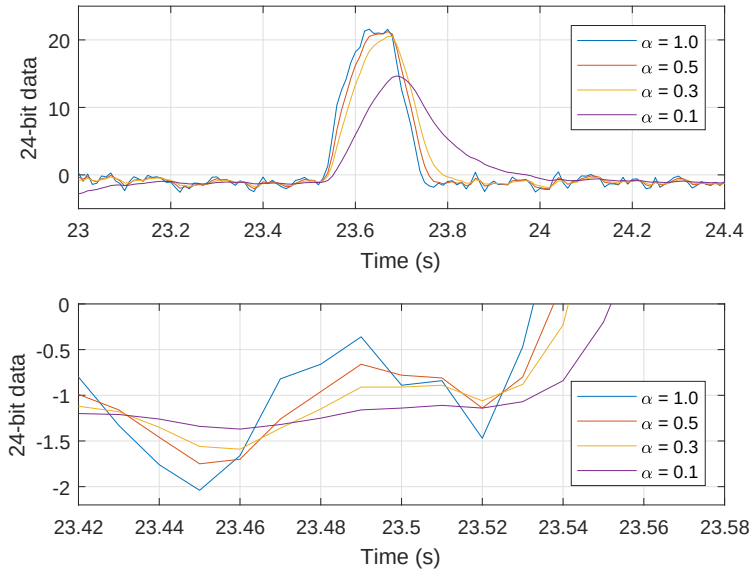
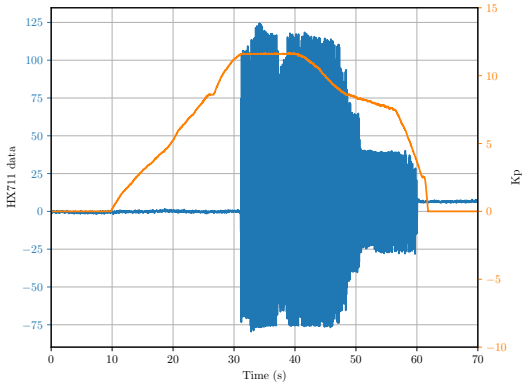


Figure 4.4: Strain filtering with different values of α.

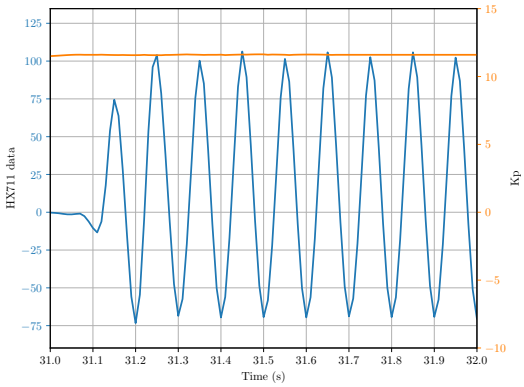
The results from testing the filter with different values of α can be seen in Fig. 4.4. The upper diagram contains samples from $t = 23.0\,\text{s}$ to $t = 24.4\,\text{s}$, while the lower diagram is a magnified portion of the upper plot, from $t = 23.42\,\text{s}$ to $23.58\,\text{s}$. The raw strain is displayed by the blue plot, when $\alpha = 1.0$. At $t \approx 23.53\,\text{s}$ a torque pulse is applied.

4.3 Ziegler-Nichols and Critical Gain Determination

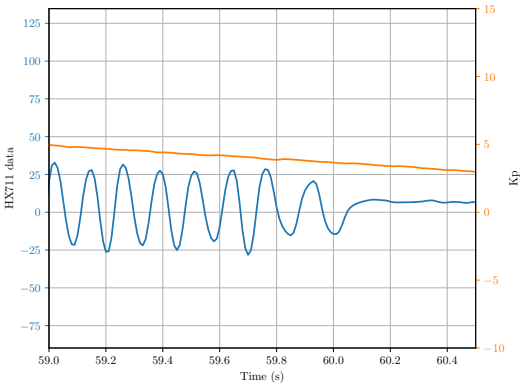
A systematic approach in order to determine optimal controller gain was tested, as covered in section 3.5. The results from testing the 2nd ZN approach are shown in Fig. 4.5. Here, the shaft is restrained at the upper end and the response in sampled as K_p is increased. Notice that oscillations begin suddenly at $t \approx 31.2\,\text{s}$ with $K_p \approx 12.0$, while ending at $t \approx 60\,\text{s}$.



(a) ZN: Response to increasing K_p.



(b) Transition to oscillation phase.



(c) End of oscillation phase.

Figure 4.5: Plot of Ziegler-Nichols results.

4.4 Load Cycle with Stepper Motor

This section covers the testing of the system with a load applied by a stepper motor and sampling strain results. As mentioned in the beginning of the chapter, the motor is placed directly above the shaft and has 200 steps per revolution which ran with $\frac{1}{8}$th micro stepping results in 1600 steps per revolution. This should provide smooth velocity and torque characteristics of the load. The test was conducted by applying a trapezoidal cycle profile for the stepper which is displayed by the orange plot in Fig. 4.6. The motor accelerates to $\approx -200.0\,\frac{\text{deg}}{\text{s}}$ at $t \approx 2.0\,\text{s}$, stays at constant velocity for $\approx 8\,\text{s}$, de-accelerates to $0.0\,\frac{\text{deg}}{\text{s}}$ and then repeats the same cycle the other direction. The middle figure in Fig. 4.6 shows the post-filtered strain readings with no BLDC actuation, i.e. $K_p = K_i = 0.0$, while the lower shows the samples with $K_p = 0.0$ and $K_i = 10.0$. The upper plot shows the unfiltered samples.

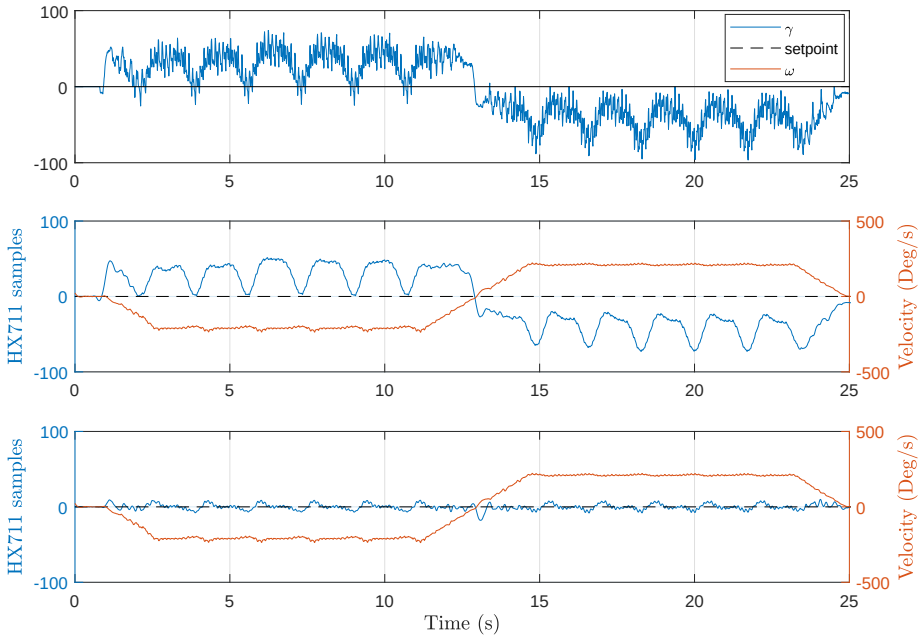


Figure 4.6: Stepper load response with no actuation, raw (upper), post-filtered (middle) and with actuation ($K_p = 0.0$ and $K_i = 10.0$) (lower).

4.5 Load Cycle with DC Motor

In order to improve the load cycle test and minimize vibration disturbances, the stepper motor was replaced with a geared high-torque DC motor [40]. The DC motor gear configuration stated a peak rotational velocity of 30 rpm which meant the test could not reach the QC peak velocity of 60 rpm, as given by the boundary conditions. However, the same trapezoidal cycle was implemented. The first test consisted of running the cycle without any BLDC actuation. The results are plotted in Fig. 4.7.

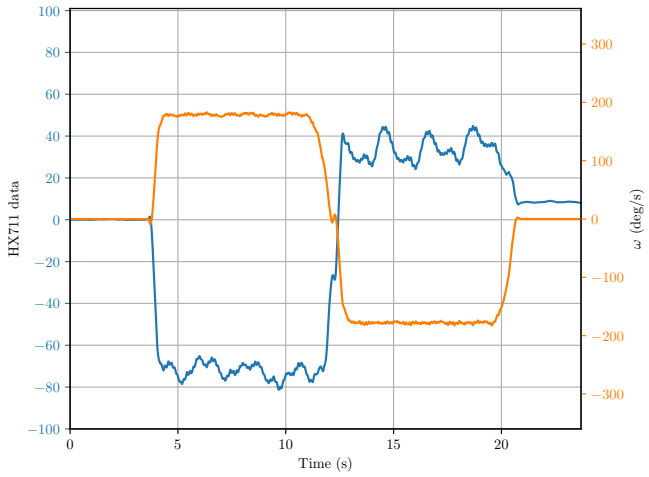
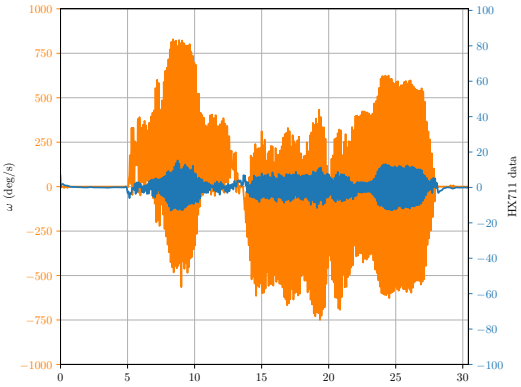
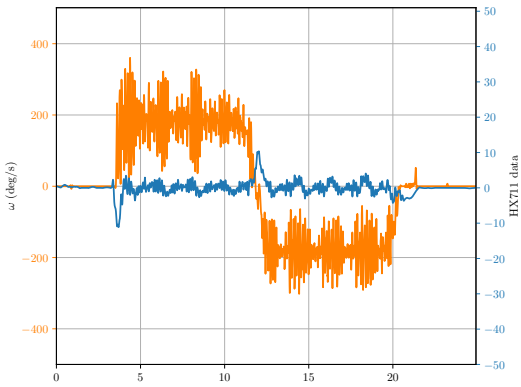


Figure 4.7: Strain readings (blue) and velocity (orange) with no actuation ($K_p = K_i = 0$) during DC load cycle.

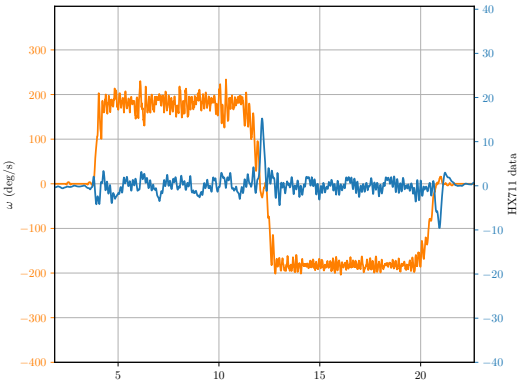
The following test consisted of testing different gain values K_i and K_p with some relation to the values estimated by the ZN method. However, the actual ZN values could not be used as a valid approach, which will be discussed in section 5.2. It should be noted that running the test with $K_p > 1.5$ and $K_i > 22.0$ resulted in highly unstable behaviour, as can be depicted from plot (a) in Fig. 4.8, and resulted in unsuccessful test runs. The data from these runs are therefore not included. The most relevant results are displayed on the next page, in Fig. 4.8.



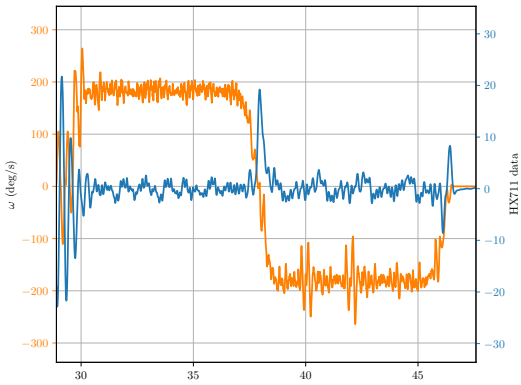
(a) DC cycle: $K_p = 1.25$ and $K_i = 20.0$.



(b) DC cycle: $K_p = 1.0$ and $K_i = 15.0$.



(c) DC cycle: $K_p = 0.0$ and $K_i = 15.0$.



(d) DC cycle: $K_p = 0.0$ and $K_i = 12.5$.

Figure 4.8: Results from the DC load test.

Chapter 5

Discussion and analysis

After acquiring the results data there is a lot to inspect. The following chapter will therefore cover the discussion and analysis of the results, but most importantly the main findings.

5.1 Desired Filter Value α

After testing the strain readings for different values of the filter parameter α the plots in Fig. 4.4 displays the results. It is desired that the filter reduces the noise that occurs when there is no actual strain on the shaft, but at the same time allows reacting to sudden applied loads. However, it can be seen that for increasing amount of filtering, an increasing amount of phase lag occurs. When $\alpha = 0.1$ the samples lag behind $\approx 20.0\,\text{ms}$ from the actual raw input, which is substantial when we want a fast transient response. On the other hand, the samples when $\alpha = 0.5 \wedge 0.3$ results in decent filtering, as well as an acceptable amount of phase lag [1].

5.2 Analysing the Ziegler-Nichols Results

By inspecting results from running the ZN method, where the critical gain K_{cr} is tried to be determined, it can be seen that the response starts to oscillate at $t \approx 32\,\text{s}$ when $K_p \approx 12.0$. However, it can be noted that the response immediately reaches a threshold limit. The results data revealed that the controller output immediately reaches the `MAX_OUTPUT` and `MIN_OUTPUT` threshold values, and keeps oscillating between them. In other words, the response goes directly from 0 to the limit of what the BLDC actuator can provide. Since the critical gain K_{cr} is defined as the gain where the system first exhibits steady oscillations, i.e. where the system is marginally stable, and the results shows oscillations that points towards instability, the value of K_{cr} can not be reliably determined from the results [1].

Additionally, it can be seen that the oscillations dampens out at $t \approx 60\,\text{s}$ when $K_p \approx 3$. Since $3 < 12$, the plot shows the difference between the static and dynamic system response. The excited system will experience lower friction than the static one, and therefore have different plant characteristics [1].

5.3 Stepper and DC Test Analysis

In this section, the general results from the load tests will be discussed. From Fig. 4.6 it can be noted that the strain seems to overshoot and oscillate for the unfiltered samples, i.e. when the stepper load is applied ($t \approx 1.0\,\mathrm{s}$) there is a great increase in generated noise. If these oscillations were the case only for the lower plot with high K_i value, it would be as anticipated due to the 2nd order nature of the I-controlled system [1]. However, it seems that this behaviour occurs when there is no actuation as well (upper plot). A possible reason for this lies in the characteristics of the stepper motor. A stepper motor moves fundamentally in incremental steps, and despite 1600 steps per revolution it was noted from further inspection that the stepper motor vibrated substantially [41]. These vibrations causes strain that most likely are picked up by the sensitive strain gauge. The stepper motor needs to be replaced by an actuator that delivers more continuous velocity curves and lower vibration.

The next step was to carry out the load tests with a DC motor. By initial comparison of Fig. 4.6 and Fig. 4.7 it can already be noted that the vibrations are greatly reduced. On the other hand, variable strain during each rotation is reoccurring, but this topic will be discussed in the next section. The PI controller system-response to the DC load cycle can be seen in Fig. 4.8. Plot (a) shows the system response when the gain values are too high, especially K_p. The BLDC actuator torque starts oscillating back and forth, creating steap peaks of shaft acceleration. This is reduced with lower gain values, such as in (c) and (d). It can be noted that the strain is greatest when the load is accelerating/de-accelerating, e.g. at $t \approx 12.0\,\mathrm{s}$ in plot (c), which is as anticipated. The controller generally manages to react fast and keep the response close to setpoint. However, the strain overshoot is reoccurring in all tests. This is most probably caused by test conditions. The strain gauge is highly sensitive and picks up the slightest vibrations. Since the geared DC motor would also generate vibration noise, this was added to the process variable [1].

5.4 Implications of a Non-Perpendicular Shaft

As mentioned in the previous section, it was noted another undesired response that occurred in the test results. The strain seems to increase at certain angle intervals during each rotation, even when there is no BLDC actuation. For instance in Fig. 4.6, the constant load strain when $5\,\mathrm{s} < t < 10\,\mathrm{s}$. This phenomenon is especially apparent in the middle plot. The cause for this seems most likely to be a skew shaft, i.e. non-perpendicular shaft angle relative to the horizontal plane. By further inspection, it turned out that the shaft itself was a little curved. Since the DC and stepper motor were fixed concentric with respect to the BLDC actuator, the shaft would experience varying axial strain caused by bending with respect to position angle [4]. Unfortunately, it was not possible to straighten the shaft at this point in the project.

5.5 Bending Strain

One advantage with Wheatstone bridges is that they can cope with uniform axial strain, i.e. strain that is constant for each point of the cross-section and affects all 4 gauges simultaneously [12]. However, strain caused by bending will cause elongation on one side of the shaft and compression on the other, and is therefore not constant at each point of the cross-section. This would be the case when the QC is tilted and rests on the joint side. Figure 5.1 shows the shaft subjected to a bending moment when the QC mass is unaligned with the shaft centre by a distance x. This will cause compression strain on the left side of the shaft and elongation strain on the right. From the Wheatstone bridge in Fig. 3.2 from section 3.1, this will cause negative values of ΔR_1 and ΔR_2, while ΔR_3 and ΔR_4 will increase in resistance. In the current sensor setup, the algorithm can not differ between shear strain caused by torque or axial strain caused by bending. Therefore, bending strain will be interpreted as shear strain by the HX711 and thus result in a false increased error magnitude [4, 13].

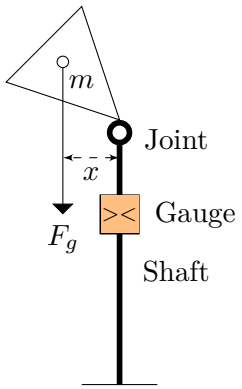


Figure 5.1: Bending strain when the QC is tilted and rests on the joint sides.

Chapter 6

Conclusion

After project completion there are several important topics to cover. In terms of project scope, the time management proved challenging. The practical part of the project required a substantial amount of time, e.g. base building, correctly commutating the BLDC and implementing the circuit. Additionally, essential components such as the L6234 circuit board needed to be milled and shipped from China, which meant additional delays. This resulted in less time available for tuning and improving of the actual control system.

After running the load cycle experiments it can be seen that the system actively reduces the shaft strain. It seemed that $10.0 < K_i < 20.0$ yielded acceptable responses from I control, and fine-tuned values probably exist in this range. The recursive filter with α in the ≤ 0.5-range also proved to tackle noise obstacles quite well. However, it seemed that the response started oscillating as soon as some proportional gain K_p was applied. For instance, when manually testing for $K_p = 1.25$ and $K_i = 15.0$ the BLDC started to vibrate, but as soon as the proportional term was removed the vibrations stopped.

One of the main challenges with current system is the skewness of the shaft. Since the QC will not be fixed in space by anything other than the shaft it may not yield the same problems as the fixed DC motor from the load tests. However, the skewness will make the mass un-centred which causes additional bending strain on the shaft.

An important aspect that is quite apparent in the realised control system is the impact of the non-linear friction torque caused by the slip ring. When the system switches between the static and the rotational states, it is noted that the I controller accumulates the error during static friction and then overshoots as soon as the shaft starts to rotate. Additionally, this was also highly apparent in the ZN test results. The excited, rotating system will oscillate for much smaller values of K_p as compared to the gain needed to make the static system begin oscillating [3, 1].

Conclusively, the compensation system works as intended, to a certain degree. It actively compensates for the friction strain by accumulating the errors, and by manually twisting of the shaft there is considerably less resistance. However, it can be stated that the setup used for testing were not optimal due to vibrations, bending strain and other undesirable disturbance. It is not certain whether the disturbances would reoccur during QC operation. The final section will suggest some further improvements.

6.1 Considerations for Further Steps

In order to make further improvements to the control system, a way to cope with strain caused by bending needs to be implemented. One solution could be to add a support fixed to the base that is connected to the shaft top by bearings. The goal is to let the support absorb any bending torque that may occur from the QC, and not the shaft itself [4].

Improvements to the actual load test is also highly recommended. This would require a motor actuator setup that generates a minimal amount of vibrations and has smooth torque waveforms. Further on, as covered in the discussion chapter the current state of the system involves a skew shaft. The shaft either needs to be straightened or replaced, even though this would require the installation of a new strain gauge.

Improvements to the control algorithm could be made by running tests and building a precise model of the actual system. For instance, a Kalman filter could be implemented given a good estimation of physical parameters [42].

Bibliography

[1] K. Ogata, *Modern Control Engineering (5th Edition)*. Pearson, 2009. [Online]. Available: https://www.amazon.com/Modern-Control-Engineering-Katsuhiko-Ogata/dp/0136156738?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0136156738

[2] P. Nie, "The history of bearing," Nov. 2015. [Online]. Available: https://www.linkedin.com/pulse/history-bearing-pan-nie

[3] H. Olsson, "Control systems with friction," 1996. [Online]. Available: http://www.control.lth.se/documents/1996/olsh96dis.pdf

[4] J. L. Meriam, *Engineering Mechanics (Engineering Mechanics V. 1 1)*. John Wiley & Sons Inc, 2012. [Online]. Available: https://www.amazon.com/Engineering-Mechanics-V/dp/1118164997?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1118164997

[5] S. E. Co., "Snu11 series usb slip ring," datasheet. [Online]. Available: http://www.senring.com/data/upload/pdf/SNU/snu11.pdf

[6] S. Wanda, "How does a slip ring work," UEA blog, Aug. 2016. [Online]. Available: http://www.uea-inc.com/resources/blog/how-does-a-slip-ring-work/

[7] J. Haugan, *Formler og tabeller*. NKI Forlaget, 1992.

[8] B. University, "Mechanics of materials: Torsion." [Online]. Available: http://www.bu.edu/moss/mechanics-of-materials-torsion/

[9] T. E. Toolbox, "Modulus of rigidity." [Online]. Available: https://www.engineeringtoolbox.com/modulus-rigidity-d_946.html

[10] M. Ottestad, "Mas220: Servoteknikk," 2016, notes from the "MAS220 Servoteknikk" curriculum tought at UiA.

[11] A. F. N. Azam, A. Jidin, M. A. Said, H. Jopri, and M. Manap, "High performance torque control of bldc motor," in *2013 International Conference on Electrical Machines and Systems (ICEMS)*, Oct 2013, pp. 1093–1098.

[12] M. Ottestad, "Mas200: Mekatronikk," 2016, notes from the "MAS200 Mekatronikk" curriculum tought at UiA.

[13] A. Bighashdel, H. Zare, S. H. Pourtakdoust, and A. A. Sheikhy, "An analytical approach in dynamic calibration of strain gauge balances for aerodynamic measurements," *IEEE Sensors Journal*, vol. 18, no. 9, pp. 3572–3579, May 2018.

[14] N. Instruments, "Measuring strain with strain gages," May 2016. [Online]. Available: http://www.ni.com/white-paper/3642/en/

[15] M. Measurements, "250us: General purpose strain gages - shear/ torque pattern," Jul. 2014, datasheet. [Online]. Available: http://www.vishaypg.com/docs/11334/250us.pdf

[16] S. Al-Mutlaq, "Getting started with load cells," sparkfun. [Online]. Available: https://learn.sparkfun.com/tutorials/getting-started-with-load-cells

[17] A. Semiconductor, "Hx711: 24-bit adc for weigh scales," datasheet. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf

[18] S. Al-Mutlaq, "Load cell amplifier hx711 breakout hookup guide," sparkfun. [Online]. Available: https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide

[19] A. Corporation, "Atmega32u4," Sep. 2010, datasheet. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf

[20] HobbyKing.com, "Turnigy hd 5208 brushless gimbal motor (bldc)," 2018, webstore. [Online]. Available: https://hobbyking.com/en_us/turnigy-hd-5208-brushless-gimbal-motor-bldc.html?____store=en_us

[21] W. Brown, "An857: Brushless dc motor control made easy," 2002, microchip Technology Inc. [Online]. Available: http://ww1.microchip.com/downloads/en/appnotes/00857a.pdf

[22] STMicroelectronics, "L6234: Three-phase motor driver," Mar. 2017, datasheet. [Online]. Available: http://www.st.com/content/ccc/resource/technical/document/datasheet/6d/24/36/da/ab/59/43/ba/CD00000046.pdf/files/CD00000046.pdf/jcr:content/translations/en.CD00000046.pdf

[23] M. Anton, "Bldc motor driver," 2009, open-licensed. [Online]. Available: http://manton.ca/open:bldc-motor-driver

[24] STMicroelectronics, "An1088: L6234 three phase motor driver," Apr. 2001, application note. [Online]. Available: http://www.st.com/content/ccc/resource/technical/document/application_note/78/44/47/d5/a8/63/4a/8e/CD00004062.pdf/files/CD00004062.pdf/jcr:content/translations/en.CD00004062.pdf

[25] I. Gramba, "Bldc svpwm lookup tables," Apr. 2015, spreadsheet. [Online]. Available: http://www.berryjam.eu/wp-content/uploads/2015/04/BLDC_SPWM_Lookup_tables.ods

[26] ——, "Spining bldc(gimbal) motors at super slooooooow speeds with arduino and l6234," Apr. 2015, blogpost. [Online]. Available: http://www.berryjam.eu/2015/04/driving-bldc-gimbals-at-super-slow-speeds-with-arduino/

[27] M. P. Milan Brejl, "An2957: Bldc motor with quadrature encoder and speed closed loop, driven by etpu on mcf523x," Jun. 2006. [Online]. Available: https://www.nxp.com/docs/en/application-note/AN2957.pdf

[28] C. Inc., "Amt20: Modular absolute encoder," May 2016, datasheet. [Online]. Available: https://www.cui.com/product/resource/amt20.pdf

[29] V. N. Kumar, A. Syed, D. Kuruganti, A. Egoor, and S. Vemuri, "Measurement of position (angle) information of bldc motor for commutation used for e-bike," in *2013 International Conference on Advanced Electronic Systems (ICAES)*, Sept 2013, pp. 316–318.

[30] A. Corporation, "Atmega328/p:avr microcontroller with picopower technology," 2018, datasheet. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

[31] Arduino, "attachinterrupt()," function documentation. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

[32] B. Earl, "Timer interrupts," May 2015. [Online]. Available: https://learn.adafruit.com/multi-tasking-the-arduino-part-2/timers

[33] S. Russell, "Simple arduino driver for the hx711 adc," 2015, github repository. [Online]. Available: https://github.com/queuetue/Q2-HX711-Arduino-Library

[34] J. Fessler, "Discrete-time signals and systems," May 2004. [Online]. Available: http://web.eecs.umich.edu/~fessler/course/451/l/pdf/c2.pdf

[35] B. Beauregard, "Improving the beginner's pid," Apr. 2011, blog-post series. [Online]. Available: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

[36] "SFUptownMaker", "I2c," learn.sparkfun. [Online]. Available: https://learn.sparkfun.com/tutorials/i2c

[37] Arduino, "Wire library." [Online]. Available: https://www.arduino.cc/en/Reference/Wire

[38] P. Martinsen, "Three methods to filter noisy arduino measurements." [Online]. Available: https://www.megunolink.com/articles/3-methods-filter-noisy-arduino-measurements/

[39] K. Chatterjee, "A simple digital low-pass filter in c," Nov. 2014, blog-post. [Online]. Available: https://kiritchatterjee.wordpress.com/2014/11/10/a-simple-digital-low-pass-filter-in-c/

[40] Z. Electromotor, "Zga37rg." [Online]. Available: http://wzh001.gotoip55.com/upload/file/ZGA37RG%20%26%20ZGB37RG.pdf

[41] R. Repas, "How to take vibration out of stepmotors," Nov. 2008. [Online]. Available: http://www.machinedesign.com/motorsdrives/how-take-vibration-out-stepmotors

[42] T. Lacey, "Tutorial: The kalman filter," miT edu. [Online]. Available: http://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf

List of Figures

Appendix A

A.1 List of Components

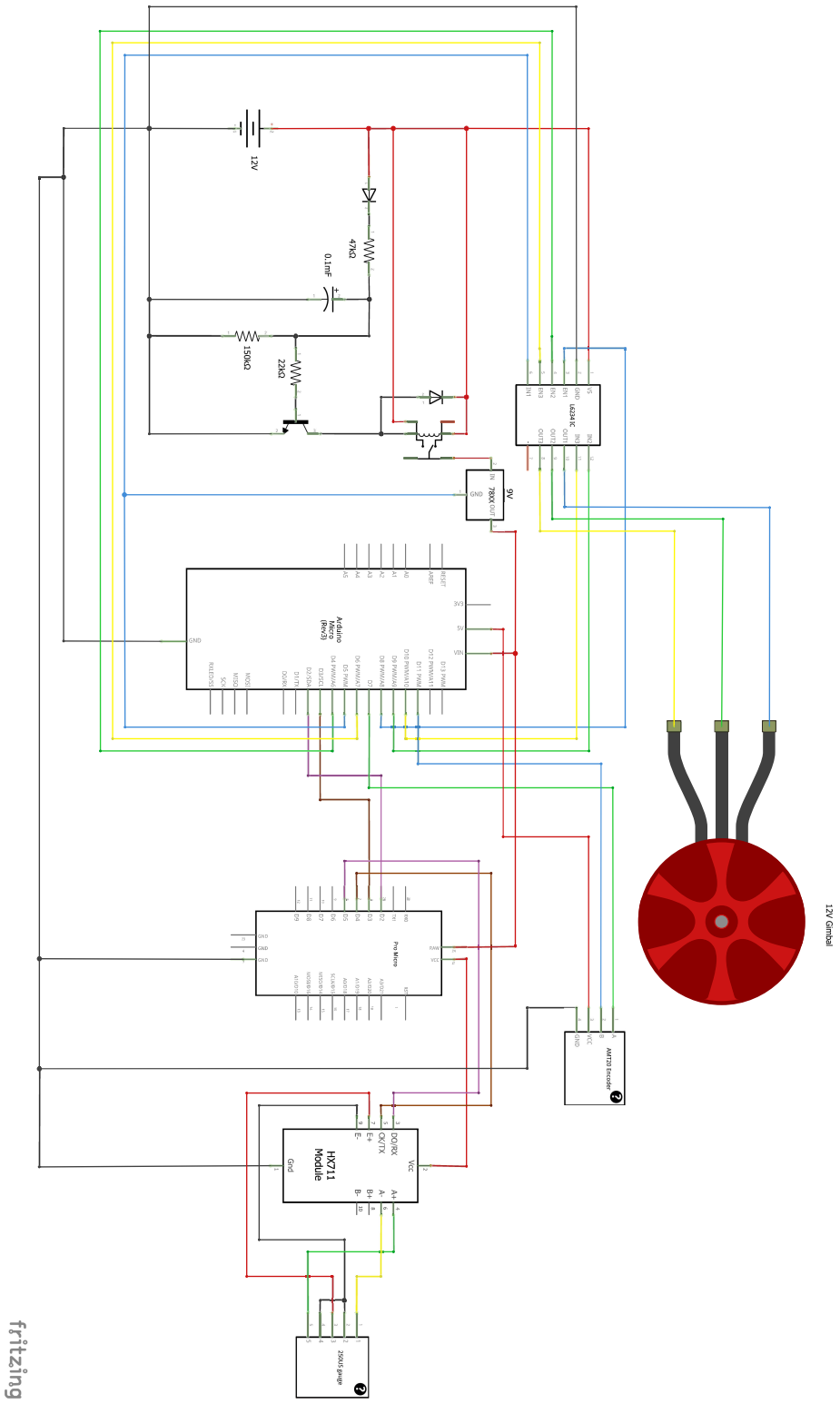
Table A.1: List of building materials.

Qty.	Description	Dimensions
4	Plexi plates	600x600 mm
1	Steel pipe	Ø12mm x 300 mm (inner Ø8mm)
1	3D-printed thrust redirector	-
1	3D-printed BLDC-Pipe adapter	-
1	3D-printed HX711 encasing	-
-	Clear Plexi adhesive	-
-	Machine Screws	M6, M5, M4, M3, M2.5
-	Locking nuts	M6, M5, M4, M3, M2.5

Table A.2: List of circuit components.

Qty.	**Description**	**Features**
1	L6234 three phase driver	
1	L6234 circuit board	
1	AMT20 encoder	PPR: 1024
1	Arduino Micro	
1	Sparkfun Pro Micro	
1	Power supply	Out: 12 V
1	Turnigy HD5208 Gimbal BLDC	
1	Sparkfun Load Cell Amp	HX711 chip
1	250US Shear Strain Gauge	
1	Senring SNU11-0410-04S	
2	Clear breadboards	
-	Jumper cables / Wires	
-	Pin headers	
1	Voltage regulator	Out: 9 V
-	Resistors	
-	Electrolytic capacitors	
1	Relay switch	12 V
-	Fast-switching diodes	

A.2 Circuit Diagram



A.3 Arduino code

The full code used by MCU1 (Arduino Micro) and MCU2 (Sparkfun Pro Micro) is listed below.

A.3.1 MCU1

```
#include <Wire.h>
#include <math.h>

// I2C variables
const uint8_t SLAVE_ADDRESS = 0x32; // 50
uint8_t encoderBytes[2];

// IO
const uint8_t en1 = 8;
const uint8_t en2 = 4;
const uint8_t en3 = 6;

const uint8_t in1 = 5;
const uint8_t in2 = 9;
const uint8_t in3 = 10;

const uint8_t encoderPinA = 7;
const uint8_t encoderPinB = 11;

// Commutation variables
uint16_t indexA;
uint16_t indexB;
uint16_t indexC;

const uint8_t phaseShift = 120;
const uint8_t commutationShift = 90;
const uint8_t pwmArray[] = {128, 132, 136, 140, 143, 147, 151, 155, 159, 162, 166,
     170, 174, 178, 181, 185, 189, 192, 196, 200, 203, 207, 211, 214, 218, 221,
    225, 228, 232, 235, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248,
    248, 249, 250, 250, 251, 252, 252, 253, 253, 253, 254, 254, 254, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 254, 254, 254, 253,
    253, 253, 252, 252, 251, 250, 250, 249, 248, 248, 247, 246, 245, 244, 243,
    242, 241, 240, 239, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248,
    248, 249, 250, 250, 251, 252, 252, 253, 253, 253, 254, 254, 254, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 254, 254, 254, 253,
    253, 253, 252, 252, 251, 250, 250, 249, 248, 248, 247, 246, 245, 244, 243,
    242, 241, 240, 239, 238, 235, 232, 228, 225, 221, 218, 214, 211, 207, 203,
    200, 196, 192, 189, 185, 181, 178, 174, 170, 166, 162, 159, 155, 151, 147,
    143, 140, 136, 132, 128, 124, 120, 116, 113, 109, 105, 101, 97, 94, 90, 86,
    82, 78, 75, 71, 67, 64, 60, 56, 53, 49, 45, 42, 38, 35, 31, 28, 24, 21, 18,
    17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 8, 7, 6, 6, 5, 4, 4, 3, 3, 3, 2, 2, 2,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 5, 6, 6, 7, 8,
    8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9,
    8, 8, 7, 6, 6, 5, 4, 4, 3, 3, 3, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 2, 2, 2, 3, 3, 3, 4, 4, 5, 6, 6, 7, 8, 8, 9, 10, 11, 12, 13, 14, 15, 16,
    17, 18, 21, 24, 28, 31, 35, 38, 42, 45, 49, 53, 56, 60, 64, 67, 71, 75, 78,
    82, 86, 90, 94, 97, 101, 105, 109, 113, 116, 120, 124};

// BLDC speed and direction
float torqueControl = 0.00;
uint8_t bldcDirection = 0; // 1 = CW, 0 = CCW

// Encoder variables
volatile int16_t encoderCounter = 0;
const float elDegPerPulse = 2.4609375;
double elDegrees;


void setup() {
```

```
40    Wire.begin(SLAVE_ADDRESS);
41    Wire.onReceive(receiveSpeedCommand);
42    Wire.onRequest(sendEncoderValue);
43    pinMode(en1, OUTPUT);
44    pinMode(en2, OUTPUT);
45    pinMode(en3, OUTPUT);
46
47    pinMode(in1, OUTPUT);
48    pinMode(in2, OUTPUT);
49    pinMode(in3, OUTPUT);
50
51    pinMode(A2, INPUT);
52
53    pinMode(encoderPinA, INPUT);
54    pinMode(encoderPinB, INPUT);
55
56    TCCR1B = TCCR1B & 0b11111000 | 0x01;
57    TCCR3B = TCCR3B & 0b11111000 | 0x01;
58
59    bldcInitializePosition();
60    attachInterrupt(digitalPinToInterrupt(encoderPinA), newPulseA, RISING);
61  }
62
63  void loop() {
64    setBldcTorque();
65    bldcCommutation();
66  }
67
68  void newPulseA() {
69    if (digitalRead(encoderPinB) == LOW) {
70      encoderCounter--;
71    }
72    else {
73      encoderCounter++;
74    }
75  }
76
77
78  void bldcInitializePosition() {
79    indexA = 0;
80    indexB = phaseShift;
81    indexC = indexB + phaseShift;
82    digitalWrite(en1, HIGH);
83    digitalWrite(en2, HIGH);
84    digitalWrite(en3, HIGH);
85    analogWrite(in1, pwmArray[indexA]);
86    analogWrite(in2, pwmArray[indexB]);
87    analogWrite(in3, pwmArray[indexC]);
88    delay(2000);
89    encoderCounter = 0;
90  }
91
92  void bldcCommutation() {
93    if (encoderCounter > 1024) {
94      encoderCounter = encoderCounter % 1024;
95    }
96    else if (encoderCounter < 0) {
97      encoderCounter = 1024 + encoderCounter;
98    }
99    elDegrees = fmod(encoderCounter * elDegPerPulse, 360.0);
100   if (bldcDirection == true) {
101     indexA = elDegrees + commutationShift + 0.5;
102   }
103   else {
104     indexA = elDegrees + commutationShift + 180.5;
105   }
106   if (indexA > 360) {
107     indexA = indexA - 360;
108   }
109   else if (indexA < 0) {
```

```
110     indexA = 360 - indexA;
111   }
112   indexB = indexA + phaseShift;
113   if (indexB > 360) {
114     indexB = indexB - 360;
115   }
116   else if (indexB < 0) {
117     indexB = 360 - indexB;
118   }
119   indexC = indexB + phaseShift;
120   if (indexC > 360) {
121     indexC = indexC - 360;
122   }
123   else if (indexC < 0) {
124     indexC = 360 - indexC;
125   }
126 }
127
128 void setBldcTorque() {
129   analogWrite(in1, torqueControl * pwmArray[indexA]);
130   analogWrite(in2, torqueControl * pwmArray[indexB]);
131   analogWrite(in3, torqueControl * pwmArray[indexC]);
132 }
133
134 void receiveSpeedCommand() {
135   uint8_t tempBuffer = Wire.read();
136   torqueControl = map(tempBuffer, 0, 255, -100, 100);
137   if (torqueControl < 0.0) {
138     bldcDirection = false;
139     torqueControl *= -1.0;
140   }
141   else {
142     bldcDirection = true;
143   }
144   torqueControl /= 100.0;
145 }
146
147 void sendEncoderValue() {
148   uint16_t tempEncoderValue = encoderCounter;
149   encoderBytes[0] = tempEncoderValue >> 8;
150   tempEncoderValue &= 0xFF;
151   encoderBytes[1] = tempEncoderValue;
152   Wire.write(encoderBytes, 2);
153 }
```

Listing A.1: Complete Arduino code used by MCU1.

A.3.2 MCU2

```
#include <Q2HX711.h>
#include <Wire.h>

//Interrupt flag
volatile byte ISRflag = 0;

// I2C
uint8_t SLAVE_ADDRESS = 0x32;
uint16_t encoderPosition;

//HX711 variables
const uint8_t HX711_DATAPIN = 5;
const uint8_t HX711_CLKPIN = 4;
Q2HX711 strainGaugeADC(HX711_DATAPIN, HX711_CLKPIN);

//Input, output and filter parameters
double rawStrainInput;
double filteredInput;
double lastFilteredOutput;
double alpha = 0.10;
double rawSetpoint;

//PI controller variables
double Kp = 0.0;
double pTerm;
double Ki = 12.5;
double iTerm;
double errorSignal;
double strainSetpoint = 0.0;
double controllerOutput;
float integratorOutput = 127.5;
const float TIMESTEP = 0.01;
const float MAX_OUTPUT = 127.5;
const float MIN_OUTPUT = -127.5;
const float HIGHER_ERROR_OFFSET = 2.0;
const float LOWER_ERROR_OFFSET = -2.0;

void setup() {
  delay(2000);
  Wire.begin();
  findSetpoint();
  initializeTimerInterrupt();
}

void loop() {
if (ISRflag == 1) {
  ISRflag = 0;
  readInput();
  filterInput();
  PIcontrol();
  sendOutput();
  requestEncoderInput();
  }
}

ISR(TIMER3_COMPA_vect) {
  ISRflag = 1;
}

void PIcontrol() {
  errorSignal = strainSetpoint - filteredInput; // Calculate the error
  pTerm = Kp * errorSignal;                     // Calculate the P-term
  iTerm += Ki * errorSignal * TIMESTEP;         // Accumulate the I-term
  if (iTerm > MAX_OUTPUT) {                     // Check for I-term overflow
    iTerm = MAX_OUTPUT;
  }
  else if (iTerm < MIN_OUTPUT) {
```

```
68      iTerm = MIN_OUTPUT;
69    }
70    controllerOutput = pTerm + iTerm;           // Calculate total PI controller
          output
71    if (controllerOutput > MAX_OUTPUT) {        // Check for output overflow
72      controllerOutput = MAX_OUTPUT;
73    }
74    else if (controllerOutput < MIN_OUTPUT) {
75      controllerOutput = MIN_OUTPUT;
76    }
77  }
78
79  void readInput() {
80    rawStrainInput = strainGaugeADC.read() / 100.0;
81    rawStrainInput -= rawSetpoint;
82    if (rawStrainInput < HIGHER_ERROR_OFFSET && rawStrainInput > LOWER_ERROR_OFFSET)
          {
83      rawStrainInput = strainSetpoint;
84    }
85  }
86
87  void sendOutput() {
88    uint8_t speedCommand = controllerOutput + 127.5 + 0.5;
89    Wire.beginTransmission(SLAVE_ADDRESS);
90    Wire.write(speedCommand);
91    Wire.endTransmission();
92  }
93
94  void requestEncoderInput() {
95    uint8_t n = Wire.requestFrom(SLAVE_ADDRESS, 2);
96    uint8_t encoderBuffer[2];
97    for( int i = 0; i<n; i++) {
98      encoderBuffer[i] = Wire.read();
99    }
100   encoderPosition &= 0x00;
101   encoderPosition |= encoderBuffer[1];
102   encoderPosition |= (encoderBuffer[0] << 8);
103 }
104
105 void findSetpoint() {
106   uint8_t numberOfReadings;
107   double total = 0;
108   for (numberOfReadings = 0; numberOfReadings < 200; numberOfReadings++) {
109     total += (strainGaugeADC.read() / 100.0);
110     delay(10);
111   }
112   rawSetpoint = total / 200.0;
113 }
114
115 void initializeTimerInterrupt() {
116   TCCR3A = 0; // Clear register TCCR3A
117   TCCR3B = 0; // Clear register TCCR3B
118   TCCR3B |= (1 << WGM32); // set the "clear timer on compare match" (CTC) mode
119   TCCR3B |= (1 << CS32); // Set prescaler to 256
120   TIMSK3 |= (1 << OCIE3A); // Enable interrupt on match
121   TCNT3 = 0; // Reset
122   OCR3A = 625; // Count value
123 }
124
125
126 void filterInput() {
127   filteredInput = alpha * rawStrainInput + (1 - alpha) * lastFilteredOutput;
128   lastFilteredOutput = filteredInput;
129 }
```

Listing A.2: Complete Arduino code used by MCU1.