UPPSALA UNIVERSITET

UPTEC IT 21033
Examensarbete 30 hp
June 2021

Developing a Graphical Application to Control Stepper Motors with Sensorless Load Detection

Mattias Adolfsson

Civilingenjörsprogrammet i informationsteknologi

Master Programme in Computer and Information Engineering

UPPSALA
UNIVERSITET

**Department of
Information Technology**

Visiting Address:
ITC, Polacksbacken
Lägerhyddsvägen 2

Post Address:
Box 337
751 05 Uppsala

Web Page:
http://www.it.uu.se

Abstract

Developing a Graphical Application to Control Stepper Motors with Sensorless Load Detection

Mattias Adolfsson

For positioning of linear stages in absolute coordinates, there is a general need to find a reference position since the initial one is unknown. This is commonly called *homing*. To reduce costs and facilitate assembly, homing can be performed without additional sensors, known as *sensorless* homing. This thesis delves into sensorless homing, specifically with respect to stepper motors, and develops a graphical application for control of them. The commercial technology StallGuard is applied in conjunction with exploration into how it – and sensorless load detection in general – functions.
The resulting graphical application can be used to configure the stepper motors, perform homing using StallGuard, and automatically tune StallGuard to work despite varying conditions. In addition, rudimentary sensorless load detection independent from StallGuard is developed, demonstrating how it could work in practice. Testing shows homing with StallGuard resulting in a position within a ± 5 µm window in 94% of cases, less than 1/7 the width of an average strand of human hair. Additionally, homing is easily performed with a single button press from the graphical interface, with optional additional configuration available.

External supervisor: August Tynong, Etteplan Sweden
Reviewer: Ping Wu
Examiner: Lars-Åke Nordén
ISSN 1401-5749, UPTEC IT 21033

Populärvetenskaplig sammanfattning

Som en del av att bekämpa tarmmaskar (eng: *soil-transmitted helminths*) utvecklas ett mikroskop hos Etteplan i Uppsala som själv kan skanna avföringsprover och räkna ägg med hjälp av bildigenkänning. Det automatiserade mikroskopet använder sig av linjära ställdon för att positionera provet och kunna undersöka olika delar av det. Dessa använder en plattform monterad på en gängad stång som kan roteras med hjälp av en stegmotor. Eftersom motorn har väldefinierade steg kan plattformen – och därmed provet – positioneras med hög precision relativt till den nuvarande positionen. Dock är den initiala positionen är inte känd. För att lösa detta behöver plattformen förflyttas till en känd referensposition, något som på engelska kallas *homing*. I det nuvarande automatiserade mikroskopet görs detta med optiska sensorer som känner av när plattformen når änden på stången, s.k. ändstoppssensorer. Detta projekt handlar om att göra samma sak utan sensorerna, vilket kallas *sensorlös* homing. Vidare är ett mål också att skapa en grafisk applikation för att enkelt kunna konfigurera de linjära ställdonen och utföra sensorlös homing. Motivationen bakom att göra sig av med sensorerna är att det reducerar den totala kostnaden och att det förenklar tillverkning och montering.

Sensorlös homing bygger på att lasten blir stor när ändpunkten nås och vidare rörelse är blockerad. En kommersiell teknik för detta är *StallGuard* från tyska Trinamic Motion Control. Denna mäter den inducerade spänningen (elektro-motoriska kraften) som uppstår då motorn roterar och uppskattar utifrån det lasten på motorn. Från relaterat arbete ses att detta inte är en unik eller ny teknik och som en del av att undersöka hur StallGuard fungerar implementeras en grundläggande version av tekniken i detta arbete med hjälp av ett digitalt oscilloskop.

Huvuddelen av arbetet består av skapandet av en grafisk applikation skapas som kan användas tillsammans med StallGuard för att utföra sensorlös homing. Denna applikation inkluderar även annan funktionalitet, exempelvis manuell stegvis motorkontroll, automatisk inställning av känsligheten hos StallGuard, och konfigurering av parametrar som hastighet, acceleration, ström, etc. Applikationens användningsområde är som ett test- och utvecklingsverktyg internt hos intressenten. Den körs på valfri dator och kommunicerar över ett nätverk (exempelvis Wi-Fi) med en Raspberry Pi som styr motorerna och därmed ställdonen.

Som en del i att förenkla och att till stor del automatisera sensorlös homing har automatisk inställning av känsligheten hos StallGuard utvecklats. StallGuard har en känslighetsparameter som avgör vid vilken last ett motorstopp detekteras. Vid för låg känslighet kan ändpunkten nås utan att detta upptäcks, vilket kan leda till skadad motor eller mekanik som följd. Vid för hög känslighet kan motorstopp felaktigt detekteras, exempelvis på grund av små laster från smuts eller ojämn mekanisk konstruktion. Generellt spelar den exakta känsligheten mindre roll ju högre hastigheten är. Den automatiska inställningen av känslighet bygger därför på att använda låg hastighet och sedan testa

lägre och lägre känslighet tills inget stopp detekteras: alltså, tills det felaktigt detekterade motorstoppet försvinner.

Resultaten visar att den automatiska inställningen av StallGuards känslighet är framgångsrik för en rad olika nivåer av ström, hastighet, och spänning. Vidare visar tester att precisionen vid sensorlös homing är god. Utav 50 testfall kan plattformen stanna på en position inom ett område av ± 5 µm i 47 fall. Den högsta avvikelsen är 20 µm, vilket kan jämföras med att ett genomsnittligt mänskligt hårstrå är ungefär 75 µm i diameter. Den ungefärliga av intressenten eftersökta precisionen är ungefär 100 µm, vilket uppnås med god marginal. Hela processen av homing sker också automatiskt via ett knapptryck i den grafiska applikationen.

Acronyms

BEMF	Back-EMF
CS	Chip-Select
EMF	Electromotive Force
GPIO	General-Purpose Input/Output
GUI	Graphical User Interface
JSON	JavaScript Object Notation
NTD	Neglected Tropical Disease
PM	Permanent Magnet
RPM	Revolutions Per Minute
RPS	Revolutions Per Second
SDFT	Sliding Discrete Fourier Transform
SDK	Software Development Kit
SGT	StallGuard Threshold
SPI	Serial Peripheral Interface
STH	Soil-Transmitted Helminth

Contents

1 Introduction

1.1 Background

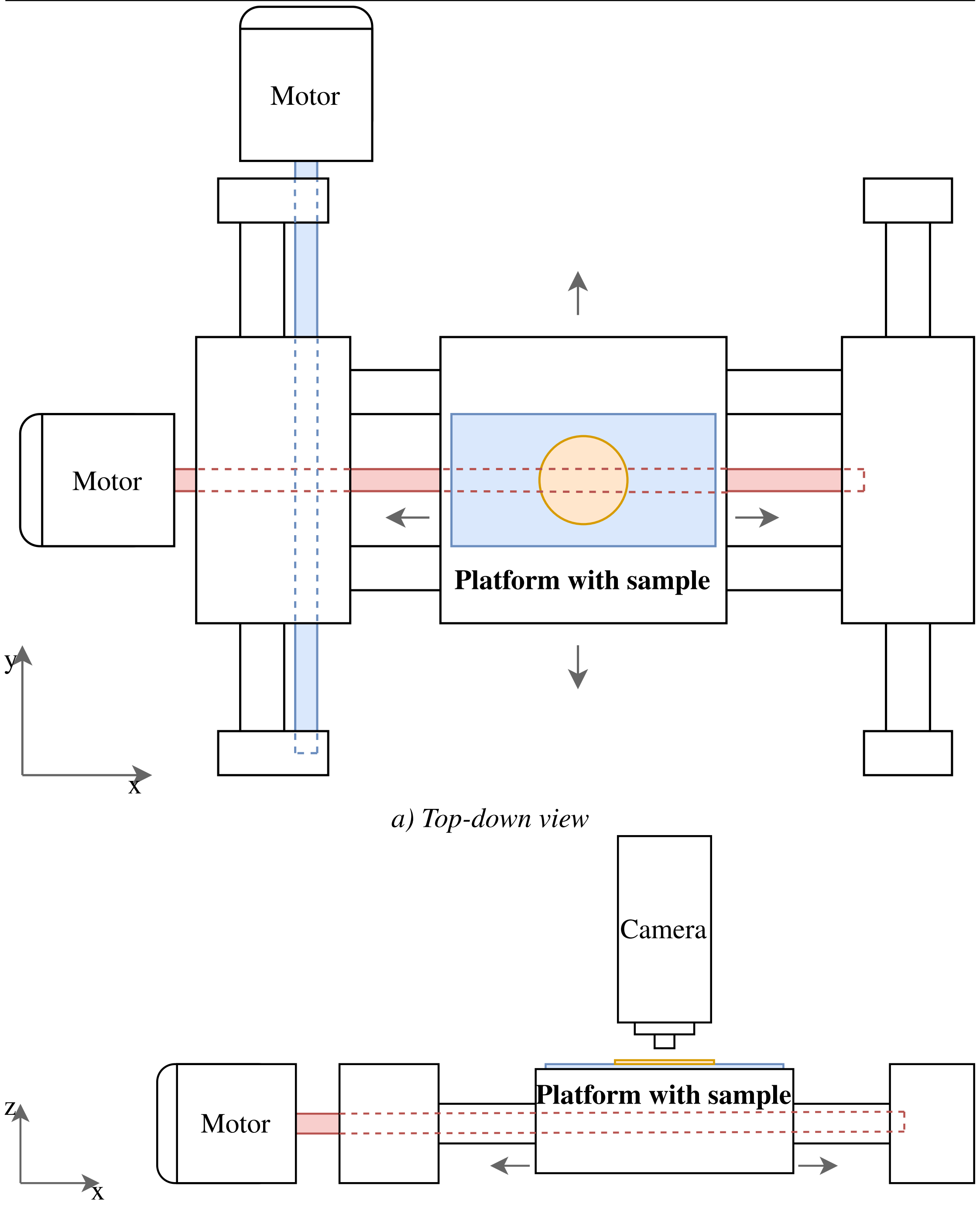
Neglected Tropical Diseases (NTDs) are a set of diseases mainly present in tropical and sub-tropical regions of the world, affecting more than a billion people across the globe [1]. Almost exclusively affecting poor regions, many of the diseases can be prevented or even completely eradicated given proper access to existing technologies and tools. However, affected population are often of low socio-economic status and of low priority in public health efforts [2].

One particular class of NTDs are Soil-Transmitted Helminths (STHs): Ascaris, whipworm and hookworm. These are intestinal, parasitic worms transmitted through contaminated soil. Each type of STH is estimated to affect between 500 million to 1 billion people on a global scale [3]. As intestinal worms, the eggs of STHs are spread through the feces of the infected.

1.1.1 Automating diagnosis

Part of the effort to detect STH infections has been to look for parasite eggs under a microscope. Specifically, stool samples are examined, counting eggs to determine infection status. However, due to long training time of staff and high turnover rate, an automated solution could prove to be much more efficient. Such a solution is under development at Etteplan Sweden, the main stakeholder in this thesis project. The solution consists of motorized linear stages, a digital microscope camera and an image processing unit. An overview of the arrangement is shown in Fig. 1.1. The sample is mounted on an XY linear stage moving in the horizontal plane, while the camera is mounted on a separate linear stage moving vertically. The image processing unit is used to process the sample's image from the camera, automatically detecting and reporting parasite eggs.

This thesis project concerns the motorized linear stages in Fig. 1.1 used to scan across stool samples with the microscope. They consist of stepper motors connected to lead screws which move a platform when rotated. The platform's relative position from startup can be known by counting the number of steps moved in each direction. However, to be able to position it absolutely, the positions of the linear stages need to be determined relative to some known reference point. This is commonly called *homing*. The scope of this thesis pertains specifically to *sensorless* homing of the linear stages of the microscope, which is homing without the use of external sensors.



a) Top-down view

b) Side view

Figure 1.1 An illustration of the automated STH egg microscope in development at Etteplan Sweden. The stool sample is fixed on a platform of an XY linear stage, enabling it to move in the horizontal plane. A separate linear can adjust the vertical position of the camera.

1.1.2 Motivation

One specific commercial tool for sensorless homing is *StallGuard* from Trinamic Motion Control [4]. It is a sensorless load torque estimator present in many of their stepper motor drivers. The stakeholder's project currently uses such stepper drivers from Trinamic and as such, it is of interest to investigate feasibility of StallGuard for sensorless homing.

In the stakeholder's project, homing is currently done by using optical switches at the ends of each axis, which can determine when the platforms have reached their respective shaft endpoints. However, this incurs an extra cost both in components and in labour to assemble the device. As such, it is desirable to perform sensorless axis-homing, without the optical switches.

The motivation behind investigating how StallGuard functions, specifically, is two-fold: firstly, to gain general insight into sensorless load detection and secondly to possibly automate some parts of configuration of StallGuard. In addition, it is desirable to know the limits of StallGuard's capabilities to see if it can be used in the stakeholder's project. Currently, the desired precision of homing is approximately 100 µm, which StallGuard has to fulfill to be considered a viable for homing.

1.2 Purpose and Goals

The purpose of this project is to delve into sensorless load estimation of stepper motors, as well as applying it practically to detect motor stall for use in sensorless homing. It both explores literature and previous work on the subject and applies a working product (Trinamic StallGuard) in a real-world setting. In terms of concrete goals, the project aims to achieve the following:

- Present theoretically how stepper motors work and methods of sensorless load estimation.
- Investigate feasibility of – and implement – sensorless homing using Trinamic StallGuard.
- Investigate factors affecting performance of StallGuard's load detection and use results to:
  - Automate tuning of StallGuard sensitivity.
  - Present how StallGuard presumably functions.
- Develop a small graphical utility to configure various settings of the stepper driver, both manually and automatically. Specifically, the goal is to automatically tune Trinamic StallGuard, and to be able to perform calibration, such as axis-homing, based on other parameters.

1.3 Tasks and Scope

Based on the purpose and goals, the following are a list of concrete tasks of the project:

1. Research technical background:
   (a) Previous and related work
   (b) Hardware: Linear stages and stepper motors
   (c) Sensorless load detection
2. Choose and assemble hardware. Set up development environment and software.
3. Develop software interface to control hardware, e.g. the stepper drivers.
4. Utilise the above interface to:
   - Investigate StallGuard by applying varying conditions, e.g. motor current. Based on the findings, develop automated tuning of StallGuard.
   - Use StallGuard to perform sensorless homing. Evaluate homing with respect to success rate and precision.
   - Perform live measurements of motor current and motor voltage to find characteristics of how StallGuard functions.
   - Develop graphical front-end to the functions of the hardware interface.

The scope of the project does not include implementing sensorless homing or load detection in the stakeholder's product. Instead, the project is an investigation, prototype, and proof-of-concept of sensorless load detection and sensorless homing, in particular with respect to StallGuard. Additionally, the project includes a graphical interface because it is of interest as a means of tuning and testing various linear stages with stepper motors.

While the end application – the stakeholder's project – uses linear stages connected together for multi-dimensional movement, the prototyping and development is done on 3 completely separate linear stages instead of the arrangement in Fig. 1.1. However, the results are also applicable to the end application.

1.4 Related Work

The concept of sensorless stall detection of stepper motors is nothing new. For example, stall detection by means of measuring Back-EMF (BEMF) can be seen in an almost 20 year old Siemens patent. Part of contemporary work on the problem seems to be focused on 3D-printers, where sensorless homing is of interest. This can for example be seen in open-source 3D-printer software such as Klipper and Marlin, both of which provide support for sensorless homing, specifically using StallGuard from Trinamic's stepper drivers.

1.4.1 Stall detection methods in patents

A patent particularly similar to StallGuard is from Siemens in 2003 [5]. It is explicitly based on measuring BEMF during phase off-time, similar to how this work finds StallGuard to behave. By cutting current briefly and giving it time to decay to zero, BEMF is measured over the leads of the motor and compared using a comparator circuit. If the BEMF is above a predetermined threshold, a stall is detected. The BEMF threshold is analogous to StallGuard Threshold (SGT) in StallGuard, tuning at what load torque a stall is detected. However, StallGuard also incorporates other features, for example a load measurement, filtering, automatic current scaling, etc.

In addition to the Siemens patent, one can find several later inventions describing attempts at sensorless stall detection [6], [7]. This shows that it is both a common problem and one that has been in development over some time. For example, a 2007 Texas Instruments patent describes attempts to overcome some problems with measuring BEMF by instead measuring phase current directly [6].

1.4.2 Load angle estimation in practice

Another example of sensorless load detection applied in a master thesis is [8]. It explores sensorless control of a hybrid stepper motor, both with respect to speed and load detection. The application is low-speed robotics, where other motors (e.g. BLDC motors) are unsuitable due to requiring high speed to produce large torque. To avoid a constantly high current, sensorless load angle estimation using a Sliding Discrete Fourier Transform (SDFT) is developed for use in variable current control based on load torque.

Compared to this work, the method used for load angle estimation is different. However, both methods are based on estimating BEMF. Additionally, both works are examples of cost- and complexity reduction in stepper motor systems by eliminating extra sensors.

1.4.3 Ystruder

One example of sensorless load detection applied practically in an academic context is *Ystruder* [9]. Developed at Aalto University, Finland, it is an open-source hardware project of a syringe pump, which can deliver precise volumes of liquid at specific flow rates. It features a 3-axis linear stage to position the syringe and the stepper motors are driven using Trinamic TMC2130 stepper drivers.

Notably, Ystruder does not include limit switches. Instead, it utilizes the StallGuard functionality of its stepper drivers for sensorless homing. As such, both the complexity of assembly and the cost of hardware is reduced. As an open-source hardware project, these could be key aspects in facilitating its adoption and usage by others.

1.5 Outline

The report is structured as follows. Section 1 present background of the project followed by purpose, goals, and motivation. Section 2 presents technical background and theory, mainly on stepper motors, necessary for subsequent sections. Section 3 provides an overview of the implemented system, followed by detailed descriptions in terms of the following key areas:

1. (Automatic) StallGuard tuning
2. Homing
3. BEMF measurements
4. Graphical application

Following the implementation, Section 4 presents results and evaluates the system with respect to the goals. Finally, Section 5 concludes the work, highlighting key findings, and presents possibilities for future work.

2 Technical Background and Theory

This section presents relevant technical background and theory about stepper motors and stall detection. It briefly describes the functionality of linear stages and stepper motors, details what the load-angle is, how it can be measured, and finally how it can be used for stall detection. This is necessary to later understand the experiments with a commercial stall detection system (Trinamic's StallGuard) and to find out how it functions in practice.

2.1 Linear Stages

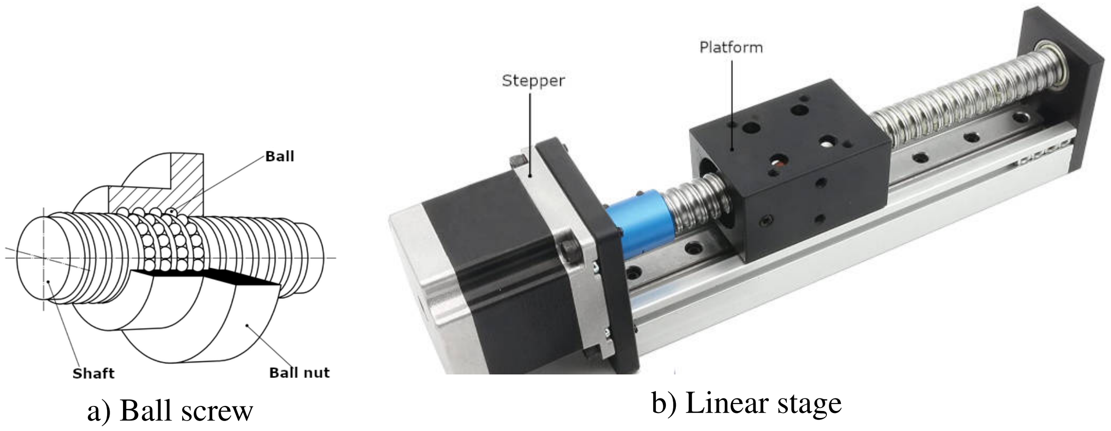


Figure 2.1 A sketch of a ball screw's internals and an example of a motorized linear stage. The ball screw (a) moves along the shaft by keeping the nut's rotation fixed while rotating the shaft. The linear stage (b) uses a stepper motor along with a ball screw to move a platform with high precision along a single axis.

Using a ball screw as depicted in Fig. 2.1, an object (e.g. a stool sample under a microscope) can be put on the platform and moved along an axis by spinning the shaft. When in addition using a high-precision motor, such as a stepper, the sample can be positioned with high precision. For example, using a screw with a pitch of 2 mm, and a stepper with 200 steps per revolution, the theoretical step size is $\frac{2\text{mm}}{200} = 10\mu\text{m}$. Further precision can also be attained by moving partial steps (micro-stepping). For example, using 256 microsteps per full step theoretically leads to a step size of $0.04\mu\text{m}$. While this assumes perfect mechanical construction and operation, it nevertheless hints at how good the precision can get. As a comparison, Soil-Transmitted Helminth (STH) eggs are approximately 50-60 µm in diameter at their longest points, more than 1000 times the theoretical step size.

Since all linear stages used in this project are motorized, the term "linear stage" is used with the implication that it is motorized using a stepper motor, unless otherwise specified.

2.1.1 Sensorless homing

For each linear stage, the relative position from startup can be determined by counting the step commands sent to the stepper-motors. As such, the position of the platform can be tracked in an open loop, without any feedback from a sensor. However, to get the absolute position along the shaft, the initial position still needs to be determined. This is commonly referred to as *homing* of the axis. For example, it could be used to position the platform at a specific distance from the start of the shaft, getting it directly under the microscope.

One way to home an axis is to move the motor in a single direction until it hits the physical endpoint of the shaft, where it is unable to move any further. This is exemplified in Fig. 2.2. Then, the absolute position is known to be the end of the axis and an initial absolute position has been acquired. To detect the endpoint, a limit switch can be used, e.g. an optical sensor or physical switch. Alternatively, endpoint detection can be done without any external sensors. This is called *sensorless* homing, where for example measurements on the current and voltage of the motor coils are used instead.

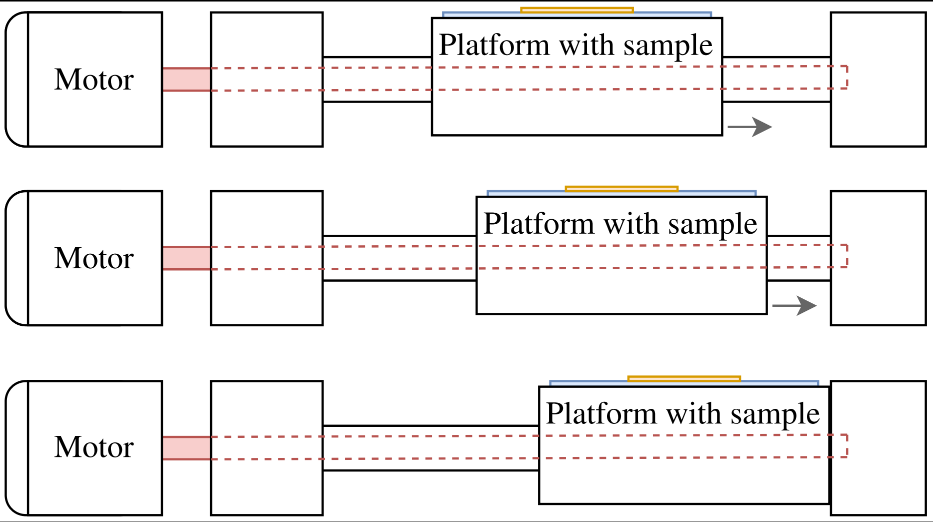


Figure 2.2 An illustration of linear stage homing by moving the platform to the physical endpoint of the shaft.

2.1.2 Trinamic StallGuard

A specific commercial implementation of sensorless endpoint detection is *StallGuard* from Trinamic Motion Control [4]. It is a sensorless load estimator present in many of their stepper drivers. It can determine the approximate load torque on the motor without external sensors, measuring up to 1024 discrete levels of load. The project at Etteplan currently uses stepper drivers from Trinamic, meaning StallGuard is available at hand for stall detection in this thesis project.

2.2 Stepper Motor

2.2.1 Operating principle

The operating principle of a stepper motor is, similar to many other motors, to apply a magnetic field to a rotor. An illustration of this is shown in Fig. 2.3. Specifically, it is showing a Permanent Magnet (PM) stepper motor, which is characterized by a central PM (rotor) which is moved by the magnetic field of the coils (stator) [10]. By moving the magnetic field ahead of the rotor, the rotor follows and a rotating motion, a *step*, is achieved. By keeping the magnetic field constant, a fixed position can be held indefinitely. Additionally, the relative position of the rotor can be known by keeping track of how many steps in each direction it has moved.

In a practical setting, the rotor often has more than two poles, instead using a set of teeth of alternating magnetic polarity. This is called a *hybrid* stepper motor. An example of this is shown in Fig. 2.4. Such a construction achieves a small rotation between steps, typically 1.8° [10], and less distance between the magnetic poles of the stator and the rotor. The motor is constructed such that the teeth of the rotor align with at most the teeth of one phase at a time (coil 1 and 3 *or* 2 and 4). As such, actuating the other phase instead will move the rotor one step to align with those teeth. The operating principle remains the same as shown in Fig. 2.3: actuating alternating stator coil pairs (phases) to move the rotor. The step size is simply smaller and torque higher.

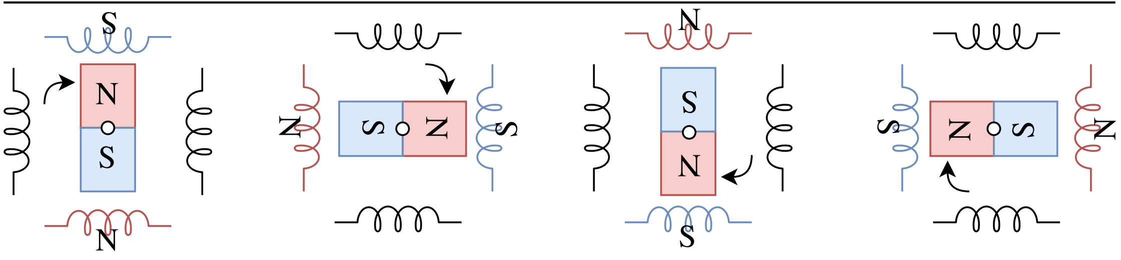


Figure 2.3 An illustration of the principle of operation of a 2-phase PM motor. The stator consists of two pairs of coils, each pair being one phase.

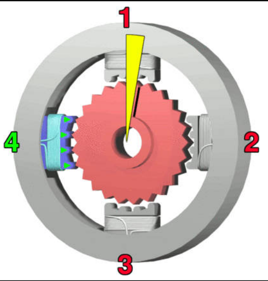


Figure 2.4 An illustration of a 2-phase stepper motor with teeth on the rotor and stator.

2.2.2 Micro-stepping

Micro-stepping is a technique to divide each step into several smaller steps, an illustration of which is shown in Fig. 2.5. By using more current levels than coils just being on or off, a smoother motion is achieved. A step is divided into a fixed number of micro-steps, commonly up to 256, where the current levels of the micro-steps together form a sinusoidal shape. The rotor can also be held at a fixed micro step position. For example, actuating neighbouring coils both with the same current, the rotor will be pulled to a position precisely in between them.

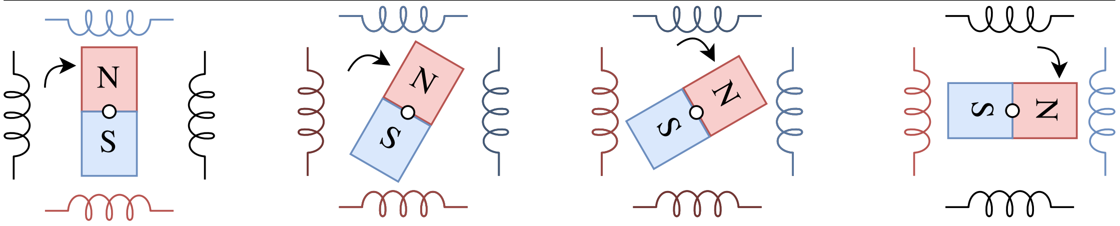


Figure 2.5 An illustration of micro-stepping, dividing a full step into 3 micro-steps

2.3 Load Angle

The load angle of the motor is the angle between the magnetic fields of the rotor and the stator. It is also a measurement of the mechanical load being driven by the motor, meaning it is an indication of the force counteracting the rotor's current position. As the load angle increases, the rotor will increasingly lag behind the magnetic field of the stator.

If there is no load, the rotor will align itself with the stator such that all magnetic forces are aligned with the rotor normal and cancel out, thus producing no torque. This is shown to the left in Fig. 2.6 and Fig. 2.7. By rotating the rotor (e.g. by applying a load), the rotor will no longer be aligned with the stator's magnetic field. The effect is that the magnetic forces produce a net torque counteracting the rotation. This is shown in the middle of Fig. 2.6 and Fig. 2.7. To the right is shown the maximum load angle in a hybrid stepper motor: 90°. Any increase in load angle at this point will cause the rotor teeth to align with the previous ones of the stator, moving backwards with respect to the stator, which is called a step loss.

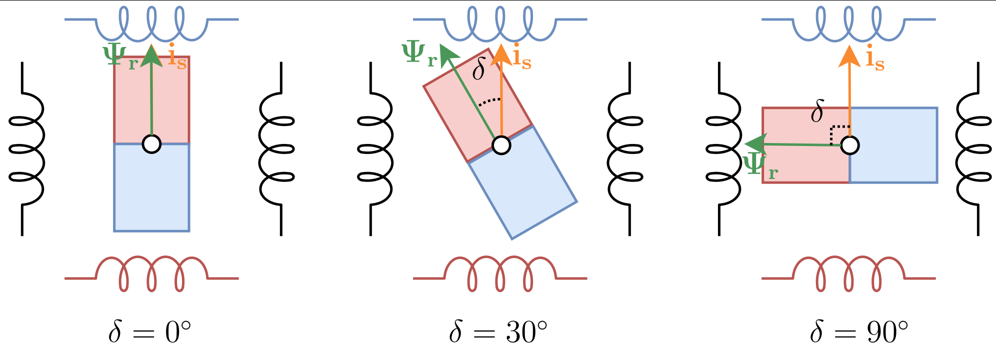


Figure 2.6 An illustration of load angle δ on a PM stepper motor. Vectors Ψ_r and i_s show the rotor magnetic flux and stator current resultant respectively.

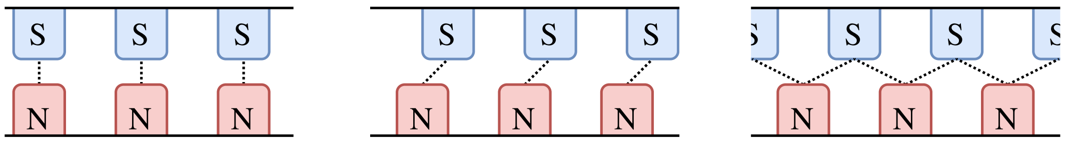


Figure 2.7 An illustration of load angle on a toothed stepper motor, with dotted lines indicating magnetic attraction. From left to right: 0, 45, and 90 degrees.

As shown by [11], for a hybrid stepper motor the produced torque T depends on the resulting stator current i_s, rotor flux Ψ_r, and load angle δ by the following equation:

$$T = \Psi_r i_s \sin(\delta) + \frac{L_d - L_q}{2} i_s^2 \sin(2\delta) \tag{2.1}$$

where L_d and L_q respectively are the resulting inductance along Ψ_r and perpendicular to it. As further demonstrated, the first term is the dominant component, and thus the produced torque is roughly proportional to the sine of the load angle. Additionally, the load angle increases with load torque until reaching an equilibrium point where the produced torque (due to increased load angle) precisely counteracts to the load torque. That is, the steady-state load angle is roughly proportional to the sine of the load torque.

2.3.1 Use-cases

The idea of sensorless load angle estimation is to, without any external sensors, estimate the load angle of the motor during operation of varying load. It has both been covered in research, e.g. by [12], and is available in commercial products such as Trinamic Motion Control's StallGuard [4]. Since, as shown, the load angle is roughly proportional to the sine of the load torque, load angle estimation is in effect also load torque estimation. This has several potential use cases, such as:

- Reducing current for low load, saving energy and producing less heat.
- Detecting step-loss by the load angle becoming too large.

Specifically of interest to this work is detection of step-loss, since this is what happens during stalling of the motor; the load torque becomes too large for the produced torque to keep the rotor in sync with the magnetic field of the stator. By estimating the load angle, a motor step-loss can be detected by the load angle becoming large.

2.4 Sensorless Load-Angle Estimation From BEMF

When the rotor is rotating, its magnetic poles are moving and applying a varying magnetic field to the stator's coils, essentially acting as a generator. By Faraday's law, this induces an electro-motive force in the coils, also known as Back-EMF (BEMF). For a given stator phase, the BEMF will be a sinusoidal voltage over that coil. As shown by [12], the load angle can be effectively estimated with knowledge of the phase difference between BEMF and the coil current (also sinusoidal).

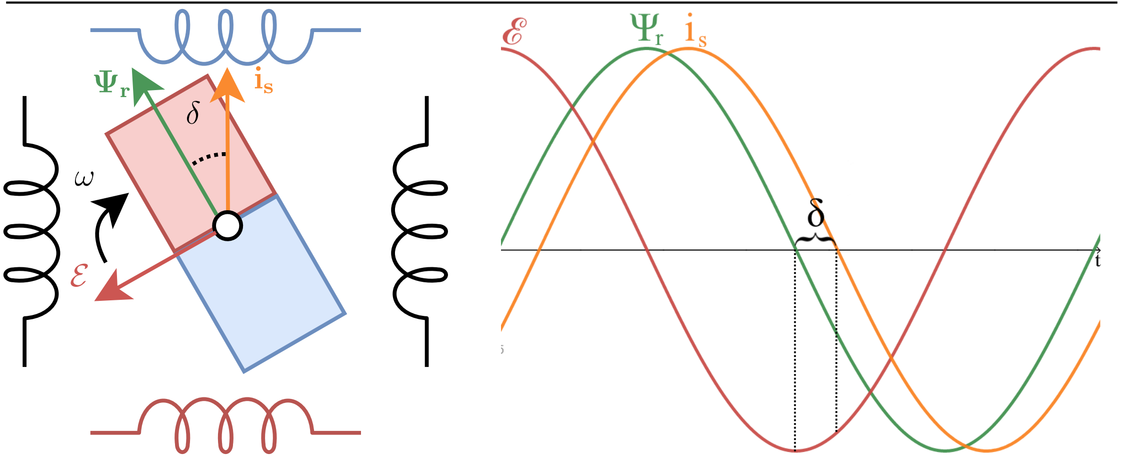


Figure 2.8 Illustration of how phase difference between coil current and relates to load angle δ, as presented by [12]. Graph shows BEMF $\mathcal{E}$, flux Ψ_r and current i_s for a single coil over time. The rotor rotates with angular velocity ω and the dotted lines highlight value of BEMF when flux and current cross zero respectively.

An illustration of how the phase difference between coil current and BEMF relates to the load angle is shown in Fig. 2.8. The phase of the BEMF ($\angle\mathcal{E}$) is 90° ahead of the phase rotor flux ($\angle\Psi_r$), which is fixed to the actual rotor rotation. The phase of the coil current ($\angle i_s$) is fixed to the stator's magnetic field. Thus, the phase difference between the two magnetic fields (the load angle δ) can be determined from the phase difference between the BEMF and the coil current as follows:

$$\delta = \angle i_s - \angle\Psi_r = \angle i_s + 90^\circ - \angle\mathcal{E} \tag{2.2}$$

To determine $\mathcal{E}$, the voltage u over a stator coil can be modelled as [13]:

$$u = Ri_S + L\frac{di_s}{dt} + \mathcal{E} \tag{2.3}$$

where i_s is current, R is coil resistance, L coil inductance, and $\mathcal{E}$ is BEMF, the variable of interest. Measuring u and i_s is trivial, but as suggested by [11], simply solving for $\mathcal{E}$ does not work in practice due to taking the derivative $\frac{di_s}{dt}$ of a noisy signal. Additionally, R and L are possibly unknown for the used motor or may change significantly with temperature. To overcome these problems, several methods have been presented, two of which are detailed in the following sections. One measures $\mathcal{E}$ at the precise moments when $\frac{di_s}{dt} = i_s = 0$ and one uses a Fourier transform to extract the fundamental sinusoidal component of $\mathcal{E}$ without noise.

2.4.1 Current zero-crossing method

Measuring BEMF $\mathcal{E}$ during current zero-crossings has been suggested by [12] and [14]. Once every full step one of the phases will change direction of current, passing zero. Giving the current time to settle (the coil's magnetic field collapsing), the current and its rate of change is zero: $\frac{di_s}{dt} = i_s = 0$. This simplifies Eq 2.3 to $u = \mathcal{E}$, meaning the BEMF can be measured directly as the voltage over the phase coil.

Measuring $\mathcal{E}$ during current zero-crossing does not directly include information about the phase and thus load angle as per Eq 2.2. However, it can be calculated given knowledge of the motors BEMF constant K_e. For a two-phase hybrid stepper motor, the exact BEMF that will be measured for the first phase is [12]:

$$\mathcal{E} = K_e \cdot \omega \cdot \sin(-p \cdot \theta) \tag{2.4}$$

where p is number of pole pairs (typically 50), θ is rotational angle, and ω is rotational speed. For a given current zero-crossing of the first phase, the angle θ is every odd full step, with an offset (lag) equal to load angle δ:

$$\theta\ [\text{deg}] = \frac{(2n+1)90 - \delta}{p}, \qquad n = 0, 1, 2, 3, ... \tag{2.5}$$

which gives the expected BEMF $\mathcal{E}$ for a given load angle δ as:

$$\mathcal{E} = -K_e \cdot \omega \cdot \sin(180n + 90 - \delta) \tag{2.6}$$

or simplified as the absolute value regardless of n:

$$|\mathcal{E}| = K_e \cdot \omega \cdot \sin(90 - \delta) \tag{2.7}$$

and conversely load angle as a function of BEMF:

$$\delta \text{ [deg]} = 90 - \sin^{-1}\left(\frac{|\mathcal{E}|}{K_e \cdot \omega}\right) \tag{2.8}$$

A plot of Eq 2.7 is shown in Fig. 2.9. As load increases, the measured BEMF decreases from $K_e \cdot \omega$ at no load ($\delta = 0$) to 0 at maximum load ($\delta = 90$). Except for low load, there is a close to linear relationship. The same can be seen previously in Fig. 2.8, where the point at which current $i_s = 0$ (rightmost dotted line) is offset to the right based on load angle δ: the farther to the right (up to load angle $\delta = 90$), the smaller the magnitude of BEMF.

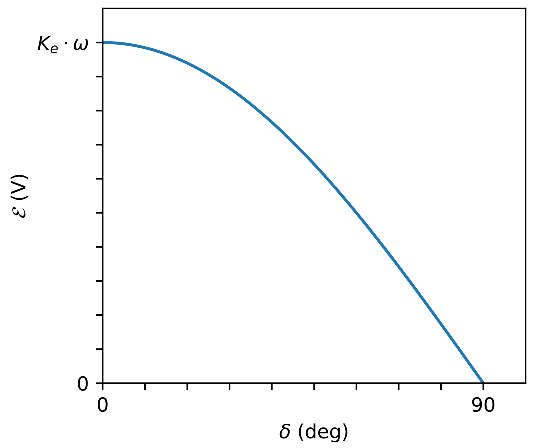


Figure 2.9 Theoretical BEMF $\mathcal{E}$ from load angle δ when coil current crosses zero. The steady-state load angle is roughly proportional to the sine of the load torque, meaning BEMF decreases with increasing load torque.

2.4.2 SDFT method

An alternative method, as suggested by [11], is to use a Sliding Discrete Fourier Transform (SDFT) to continually estimate the fundamental sinusoidal components of the BEMF and current, thus finding their phase difference. The advantage of this method is that load angle can be estimated at a higher frequency than once per full step and without additional information about the motor and load. However, it is more complex and incurs a higher computational cost.

3 Method and Implementation

This section outlines the overarching approach taken. The overall method consists of experiments and research on the chosen stepper motor driver, in particular its stall-detection feature (StallGuard). Included is also implementation of a graphical application used to interface with the driver. Detailed in following sections are the chosen hardware, the approach taken to automatic and manual StallGuard tuning, automatic homing, and methods of measurements on StallGuard in determining how it functions.

3.1 System Overview

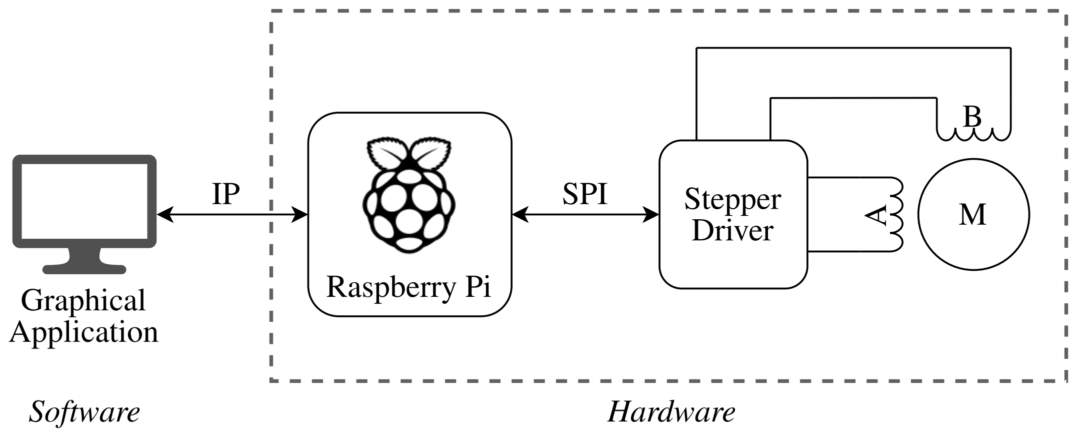


Figure 3.1 An overview of the developed system: A Raspberry Pi interfacing with a stepper driver over SPI, driving stepper motor M with two phases A and B. The software runs as a graphical application on a separate computer, e.g. a laptop.

An overview of the system is illustrated in Fig. 3.1. It consists of a Raspberry Pi connected over Serial Peripheral Interface (SPI) to a stepper driver which in turn is connected to a stepper motor. The stepper motor is attached to a small platform on a ball screw (a linear stage), thus being able to move the platform back and forth by rotation of the motor.

A Graphical User Interface (GUI), developed as part of the project, is used to configure and move the motor according to user input. By using the library *pigpio*, control over SPI on the Raspberry Pi is networked (here denoted *IP*, e.g. over Wi-Fi) and the graphical application can remotely access the stepper driver. This enables the graphical application to run on a separate host, e.g. a laptop, should that be desired.

3.2 Hardware

The hardware consists of a Raspberry Pi, a Trinamic TMC5160 stepper driver [15] and hybrid stepper motor from Casun Motor (model 42SHD0034-226NTP) [16]. The client running the graphical application is a Windows 10 x86-64 computer, but it can be any host capable of running Python programs and the Tk GUI library.

3.2.1 Stepper motor

The stepper motor is a NEMA 17 stepper motor (i.e. 43mm frame diameter) with 200 steps per revolution. It has the following electrical and mechanical properties [16]:

- Motor phases: 2
- Holding torque $T = 0.3$ N/m
- Motor peak current $I_{peak} = 1.28$ A
- Phase resistance $R = 5.0\ \Omega \pm 10\%$
- Phase inductance $L = 6.2\ mH \pm 20\%$

3.2.2 Stepper drivers

Chip	**TMC2130**	**TMC2209**	**TMC5160**
Voltage (V)	5.5-46	5.5-36	10-35
Current (A)	0.96	1.7	3.0
Interface	SPI	UART	SPI
StealthChop	No	Yes	No
Step generator	No	Velocity target	Position target
Price	\$12.73	\$15.58	\$15.00

Table 3.1 A high-level comparison of StallGuard-capable Trinamic SilentStepstick stepper drivers

The driver is required to be capable of Trinamic StallGuard, since it is part of the purpose of this work to investigate it. The SilentStepStick series [17] from Trinamic is chosen for its plug-and-play capability between different models and due to not requiring any extra components. Table 3.1 shows a comparison between the StallGuard-capable SilentStepStick models. All feature acceptable voltage and current capabilities. The Raspberry Pi features both SPI and UART, and as such the interface does not matter. Neither does *StealthChop*, which is a silent operating mode, but not a crucial feature. Lastly, the price difference is small. The only relevant difference is the step generator.

It is internal driver circuitry capable of generating step pulses, where TMC2209 can generate pulses accelerating the motor to a velocity target, and TMC5160 can generate pulses to smoothly move to a target position. To avoid implementation of such a step pulse generator, the TMC5160 is the chosen driver for the system.

3.2.3 Linear stages

Two separate linear stages are used: one with mostly metal construction and one plastic. For reference, photos of the stages are shown in Appendix A in Fig. A.1 and Fig. A.2 respectively. The difference in mechanical construction is relevant in some in testing of the system. However, due to the metal stage being more rigid and closer to the linear stage used in the main project, the metal stage is the one used if not otherwise specified.

The lead screw (shaft) pitch of the linear stages are the following:

Metal 2mm (10µm per step)

Plastic 1mm (5µm per step)

3.2.4 Raspberry Pi

Raspberry Pi is a series of single board computers developed by the Raspberry Pi foundation based on SoCs (system-on-a-chip) from Broadcom. In particular, the Raspberry Pi 4 model B, which is used in this project, uses Broadcom BCM2711 and has a quad-core A72-Cortex ARM processor and 1 to 8 GB of RAM [18]. The Raspberry Pi 4 is chosen mainly due to also being used in the main microscope project. Additionally, it enables remote access to General-Purpose Input/Output (GPIO) pins and SPI using the library pigpio.

3.3 Software and Development Tools

The main software and development tools used are the programming language Python, in version 3.8.6 of which the GUI application is written, and Raspberry Pi OS, which is the Linux distribution running on the Raspberry Pi.

3.3.1 Raspberry Pi OS

Raspberry Pi OS (previously Raspbian) is a linux distribution and the officially supported operating system of the Raspberry Pi [19]. It is released as three versions: with desktop and recommended software, with desktop, and *lite*, which is a minimal version without recommended included software or a graphical desktop environment. In this project, the version with graphical desktop and recommended software is used. However, the only requirement on it is support for interfacing with the GPIO pins using pigpio.

3.3.2 Python

Python is an interpreted, dynamically and strongly typed, object-oriented programming language published by the non-profit Python Software Foundation [20]. It is a general-purpose language with a large standard library, of which *asyncio* and *tkinter* are of particular importance to this project. Additionally, third party libraries can be installed. Following are short descriptions of the main libraries used:

tkinter is a Python library providing an interface to Tcl/Tk, which is not strictly part of python [21]. Tcl is a separate programming language of which Tk is the default GUI toolkit [22]. Both Tcl/Tk and *tkinter* are available on most unix and windows platforms and thus together provide a highly cross-platform compatible GUI toolkit in Python.

asyncio is a Python library for running asynchronous tasks and concurrent code in a single thread using cooperative multitasking [23]. Using an async/await syntax, control can be deferred during long-running tasks such as disk IO. However, the important feature for this project is the ability to create background *tasks* which can be cancelled.

pigpio is a third-party Python library with an accompanying system service to interface with the GPIO and SPI of the Raspberry Pi [24]. The library is included by default in the Raspberry Pi OS version used in this project. It runs a background daemon on the Raspberry Pi, which a Python program can connect to over a network socket (using the corresponding Python library) and issue commands. For example, it could be transferring data over SPI or configuring GPIO pins. As such, `pigpio` enables the graphical application to be run on a different device from the Raspberry Pi.

3.4 Stepper Motor Control

A schematic of the complete hardware setup is shown in Fig. 3.3, while a photo of it can be found in Fig. 3.2. The drivers (TMC5160) are communicating with the Raspberry Pi using daisy-chained SPI. The motors are powered from an external 18V power supply, whereas the Raspberry Pi directly supplies IO voltage (VIO) to the drivers.

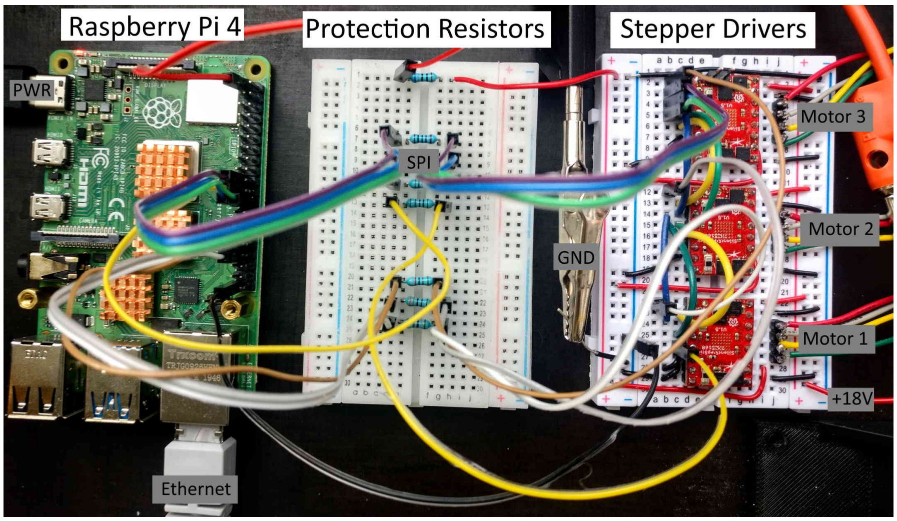


Figure 3.2 Labeled photo of physical setup of Raspberry Pi and stepper drivers. Motors are not included in the picture, but the four wires to each motor can be seen to the far right.

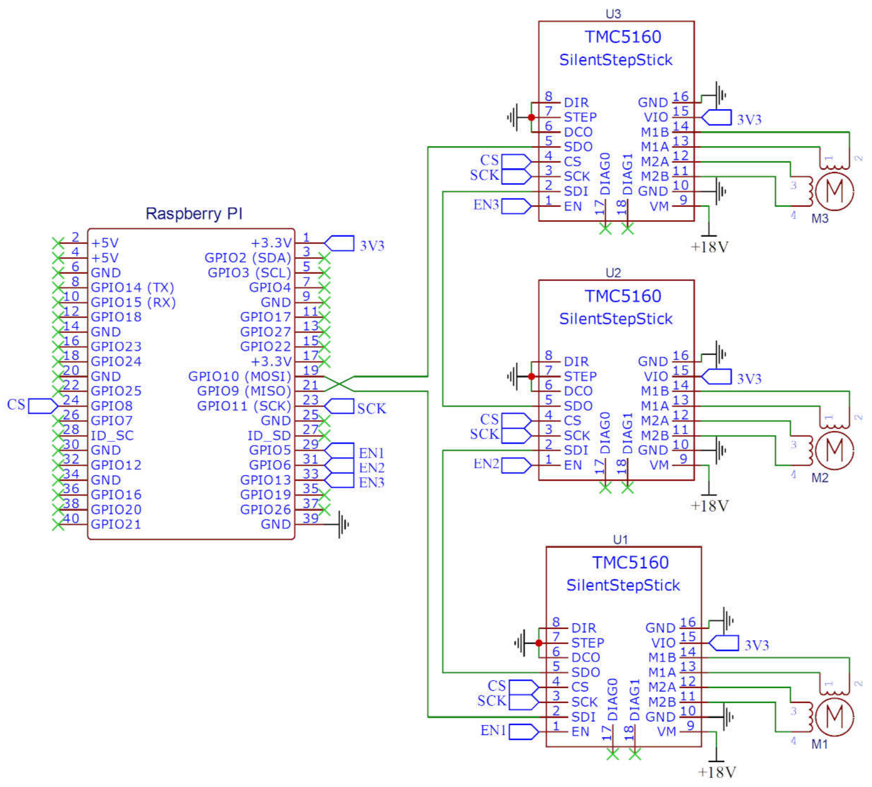


Figure 3.3 Schematic of the hardware setup of Raspberry pi and stepper drivers.

3.4.1 Daisy chaining SPI

To simplify wiring and to reduce the number of SPI channels needed, the stepper drivers are daisy chained. The procedure consists of connecting data output (SDO) of each driver to data input (SDI) on the next driver [15]. The first driver in the chain receives data directly from the master (Raspberry Pi) and the last driver in the chain outputs data directly to the master. Data is moved along the chain by sending empty data, which pushes data currently in the chain along to the next drivers and finally the master.

3.4.2 Accessing driver registers over SPI

An SPI datagram of the TMC5160 is 40 bits, where the first 8 bits are address and the rest 32 bits are data. In general, registers are integers (signed or unsigned) or 1-bit flags. Register access is performed by sending a datagram where the first 8 bits are set to the register address as given by the datasheet. Setting the most significant bit of the address to 1 writes the datagram data to the register, whereas if it is 0 the driver will ignore the datagram data and send the register contents as a response.

Data on the SDI pin (slave data in) is shifted out on SDO (slave data out) using an internal 40-bit shift register. As such, to reach e.g. the second device, first the 40 bit datagram is sent and then the clock (SCK) is pulsed 40 times to shift the datagram from the first device to the second. During this whole process, Chip-Select (CS) must be held low. Otherwise, the devices earlier in the chain will act on the datagram as if it is intended for them. Finally, when CS is released, the drivers receive and act on the data currently in their respective shift registers.

The SPI communication is implemented using the library `pigpio`. It does not provide an option to manually control the CS pin. Instead, to force it to hold CS low while shifting data through the daisy chain, datagrams are padded with dummy bits (zeroes) after its data. A padding of $40 \cdot k$ bits is used to make data reach the *k*th device in the chain. Response from the *k*th device, if required, is subsequently read by sending $40(n - k + 1)$ dummy bits for a chain of n total devices.

Sign extension While datagrams have 32 bits of data, some registers are less than 32 bits in size. For unsigned integers, this poses no problem since the data can be unpacked as a 32-bit unsigned integer, zeroing the extra upper bits. However, some registers contain signed integers (in two's complement). To allow the datagrams to be generically unpacked as 32-bit unsigned integers, regardless of underlying register, sign extension is performed in the special cases of signed integers.

In two's complement, if the most significant bit is set, the number is negative. If it is not negative, the binary representation is the same as for an unsigned number and no sign extension is needed. The implementation is done in Python, which does not have fixed bit-length or explicit signedness of its integers. To overcome this limitation, the sign extension is performed in the following way:

1. Negate by flipping bits and adding 1, obtaining positive counterpart.
2. Negate using '-' operator, making Python understand that it is a negative number.

In a language with fixed size and defined signedness of integers, sign extension can instead be done by simply casting to a signed type, padding the extra upper bits with the value of the most significant bit of the value to be sign-extended.

3.5 Tuning StallGuard

As part of the goals, sensorless stall detection using Trinamic StallGuard is demonstrated and automatic tuning of StallGuard's sensitivity is developed. During operation, StallGuard can measure up to 1024 discrete levels of load, where 1023 is minimum load and 0 is maximum load, i.e. motor stall. StallGuard's sensitivity is tuned by a single value: StallGuard Threshold (SGT). This works as an offset for the result, where lower SGT gives a lower StallGuard value for the same load. In other words, StallGuard is more sensitive the lower the value of SGT. If SGT is set too low, the StallGuard result will always be 0 (maximum load). If it is set too high, the result will never reach 0 and detect a stall, even with the motor being overloaded and skipping steps.

In order to develop understanding of stall detection and in particular tuning of SGT, it is of interest to find the factors affecting StallGuard sensitivity. This is both to aid in developing automatic tuning of SGT and to extrapolate knowledge of stall detection in general.

As per the datasheet, optimal SGT is affected by the following factors [15]:

- Back-EMF constant of the motor (K_e)
- Motor coil inductance
- Motor coil resistance
- Load angle
- Angular velocity
- Current
- Supply voltage

The first 3 factors are constants and can not be varied to examine their effect. The load angle is variable, but is difficult to precisely control. As such, angular velocity, current, and supply voltage are examined. More precisely, StallGuard's load measurement is recorded for a range of values of these 3 factors.

Supply voltage is commonly set to at least 3 times the rated motor voltage [10], which for the selected motors is $3 \cdot 6.4 = 19.2$V. This is thus used as a starting point in the tests of supply voltage. The motor is rated for at most 1.28A, which is used as the upper limit for tests of motor current. Lastly, StallGuard performs poorly at too low or high velocities [4]. As the driver datasheet suggests, best homing performance is achieved at 1 to 5 revolutions per second, which is used as a basis for angular velocity tests.

3.5.1 SGT tuning background

Trinamic provides two suggestions for tuning of StallGuard via SGT [15]. It can be tuned manually by first choosing fixed values for current and speed and then changing SGT until the StallGuard result is close to 0 with load close to stall. Additionally, an automatic method is suggested, which finds the highest value of SGT (least sensitivity) at low speed where StallGuard result remains at 0. The high-level algorithm is as follows:

1. Move at a low speed, at most 10 Revolutions Per Minute (RPM).
2. Sample StallGuard result for some amount of time.
3. If StallGuard result is consistently 0 (maximum load), increase SGT and go to 2.
4. Otherwise, set SGT to current SGT - 1.

The automatic algorithm is used as basis for developing automatic tuning with the chosen components. To evaluate the automatic tuning, it is compared with respect to manual tuning. In particular, the range of SGT values for which StallGuard can detect increased load are manually found. The range is defined as follows:

Maximum The value of SGT where maximum load without stall is required to trigger StallGuard.

Minimum The value of SGT where only a minuscule increase in load is enough to trigger StallGuard.

If the automatically tuned SGT is within this range, StallGuard will work to successfully detect increasing load and stop before a stall. If SGT is too high, StallGuard will never (or unreliably) detect a stall. If SGT is too low, StallGuard will detect false positives.

3.5.2 Implementing automatic SGT tuning

This section aims to provide concrete implementation details of the automatic tuning of SGT. The implementation is based on the high-level algorithm (see Section 3.5.1). However, a few modifications are made and certain values are decided on, e.g. sampling rate. The result is the following algorithm, which finds a valid value of SGT for a target operational current i:

1. Set SGT to 0, current to i, velocity to v and acceleration to any positive value.
2. To reduce effect of noise, activate **sfilt** option on the stepper driver, which is a low pass filter of StallGuard result.
3. Wait until motor has reached target speed.
4. Sample StallGuard result 4 times with t_{step} interval.
5. If any sample is 0:
   (a) Increase SGT by 1
   (b) Wait $4 \cdot t_{\text{step}}$
   (c) Go to 4.
6. Otherwise, current SGT is the tuning result.

Time per full step is $t_{\text{step}} = \frac{1}{v \cdot n}$ for n steps per revolution (usually 200). Since StallGuard result is only updated once per full step [15], a delay equal to the step time is introduced between samples to avoid sampling the same value multiple times. Additionally, since StallGuard result is filtered over 4 full steps with **sfilt**, and 4 full steps is a full period of the (sinusoidal) coil current, StallGuard result is sampled 4 times. This reduces effect of noise and other random variables, increasing consistency of tuning.

The tuning procedure can be sped up by reducing delay in 5b). However, doing so decreases tuning consistency and erroneously increases resulting SGT, drastically so for low delays. The delay is needed between changing SGT and sampling StallGuard result. Waiting for a full filter/current period (4 steps) appears to give good results.

Although the suggested high-level algorithm states that SGT should finally be decreased by 1, this is not done in the implementation. Generally, adding the negative offset on SGT produces values that are too low to work on low velocities. As such, it is omitted in the implemented tuning algorithm. However, if it should be kept or not is possibly dependent on mechanical construction.

Choosing StallGuard velocity threshold The driver includes the ability to set a velocity threshold above which StallGuard is active. That is, the motor will only be stopped if StallGuard result becomes 0 while the motor velocity is above a pre-defined value. This is especially useful to avoid false positives at low velocities, where BEMF is low and StallGuard works less reliably. Additionally, load torque is higher while initially accelerating from standstill since static friction must be overcome.

Generally, from testing, it seems that increasing the velocity threshold allows a more sensitive StallGuard tuning (lower SGT) without false positives, thus making it easier to tune. However, if StallGuard velocity threshold is chosen too high, the motor could spend considerable time moving into e.g. the axis endpoint before StallGuard becomes active and stops it. As a middle ground, the threshold is set to half the operating velocity.

Choosing tuning velocity The velocity v in the tuning algorithm is neither specified by the datasheet nor the high-level algorithm described as part of the method. The recommended speed is at most 10 RPM. To be able to see the effect of different speeds, a tool is built to automatically sweep through a range between 1 to 10 RPM, recording the average out of 5 automatically tuned SGT values for each speed. Since the tuning algorithm has delays proportional to speed v, a higher value gives faster run time. The lower limit of the range is chosen based on lower speeds causing the tuning to take excessively long time.

The range of values of v are tested at 18V and between 150mA to 500mA. An excerpt of the results is shown in Table 3.2 together with a comparison to manually found values of SGT at low speed (0.35 Revolutions Per Second (RPS)). Based on the result, a higher tuning velocity generally produces lower tuning values, especially so for high velocity. Additionally, a lower tuning velocity results in more accurate values of SGT as compared to the viable range found manually. As a trade-off between fast tuning run time and tuning accuracy, $v = 2.5$ RPM is chosen.

Current (mA)	Automatic				Manual	
	1.5 RPM	2.5 RPM	5 RPM	10 RPM	Min	Max
500	11	11	11	10.4	11	12
450	10	10.2	10	9	10	11
400	9	9.2	9	8	9	10
350	8	8	7.8	7	8	9
300	7	7	6.8	6	7	8
250	6	6	6	5	6	7
200	5	5	5	4.6	5	6
175	4	5	4	4	4	6
150	4	4	4	3.4	4	5

Table 3.2 Automatic tuning of SGT for different tuning velocities, average of 5 iterations. Manual minimum and maximum of SGT are determined at 0.35 revolutions per second.

3.6 Sensorless Homing

Once StallGuard has been configured to successfully detect motor stall, it is ready to be used in homing. For this purpose, the following driver registers/flags are configured:

sg_stop Setting to 1 to cause driver to stop motor movement when detecting a stall (when StallGuard result **sg_result** = 0)

en_softstop Setting to 0 to cause driver to perform a hard, immediate stop when stopping due to stall detection.

tcoolthrs Sets a lower velocity threshold, below which StallGuard is disabled.

thigh Sets an upper velocity threshold, above which StallGuard is disabled. Set to maximum such that StallGuard is always active above lower threshold.

With **sg_stop** active, once a stall is detected and velocity is within configured thresholds, the driver stops the motor and sets a flag **event_stop_sg**. The motor remains stopped until the flag is cleared, and thus the program interfacing with the driver does not need to immediately react to the motor stalling.

To home the axis, once stall detection is set up, the following sequence is performed:

1. Tune SGT for a given homing current and homing speed
2. Move in one direction until StallGuard triggers a motor stop
3. Set current position to 0
4. Move in other direction until StallGuard triggers a motor stop

5. Set current position to maximum position (axis length)

The driver's step pulse generator (also called *ramp generator*) is used to move the motor. It provides automatic movement based on desired position, velocity and acceleration, removing the need to accurately generate step pulses on the Raspberry Pi. To keep track of position, the driver includes a register with a signed value counting the number of microsteps moved. By setting this to a known value (e.g. 0), it enables accurately knowing exact relative position at which StallGuard stopped the motor, even without immediately reacting to the stall event. For example, this is used to move from one end of the axis to the other, measuring the length of the axis.

3.6.1 Selecting velocity and current

The two main parameters needed for homing, outside of StallGuard configuration, are motor current and velocity. If current is low, there is a risk of torque being too low and motor stalling prematurely. A high current ensures enough torque is available, but can make the motor movement stiff and jerky, possibly causing damage if stall detection malfunctions. If velocity is too low, StallGuard does not work reliably [4]. However, too high velocity comes with potential problems: The high momentum can cause damage when hitting the axis end and friction increases, thus also increasing amount of needed torque and current.

For best homing results, the Trinamic suggests using 30% to 70% of nominal motor current (for the used motor: 272mA to 634mA RMS) and velocity 1 to 5 RPS [15]. For the metal stage, friction is low and current can be kept low. A current of 300mA is chosen, at it provides enough torque while still being in the lower range of the recommended current. Homing velocity chosen is 3 RPS, which is in the middle of the recommended range and provides fast homing without requiring unnecessarily high current.

3.6.2 Relevance of mechanical construction

From testing during implementation, it is apparent that mechanical construction of the linear stage affects precision of homing. If non-rigid, bendy material is used (e.g. plastic), the stop when the axis hits the physical endpoint is more gradual and soft. The load torque increases more slowly, leading to a slower decline in StallGuard result and thus the exact point at which StallGuard detects a stall is less precise. The time and distance between endpoint contact and motor stop is also increased, leading to lower accuracy.

3.7 BEMF Measurements

In an effort to gain insight into how StallGuard functions, measurements are taken on the voltage and current of the motor phases and correlated with known methods of sensorless load detection. Given that the driver takes one StallGuard sample per full step

[15], and that StallGuard is explicitly stated to be based on measuring BEMF [4], the hypothesis is that the method used is that of measuring BEMF during zero-current crossing (as detailed in Section 2.4.1). The phase voltage during zero current is examined for indications of such measurements. For example, as shown by [12], then there would be phase voltage (BEMF) when current is zero, the magnitude of which would decrease with increasing load.

3.7.1 Measuring current

The phase voltage is measured trivially as the voltage over the two wires of one of the motor's two phases. The current could be measured by connecting a small resistor in series with the phase. However, since the driver already includes a current sense resistor, for simplicity it is used instead. Due to the way the resistor is connected and the way the driver uses the H-bridge, shown in Fig. 3.4, the phase current does not always go through the resistor (i.e. during slow decay). Additionally, the direction of current through the phase is not consistent with direction of current through the sense resistor. Nevertheless, it can still be used to measure magnitude of current and when it is zero.

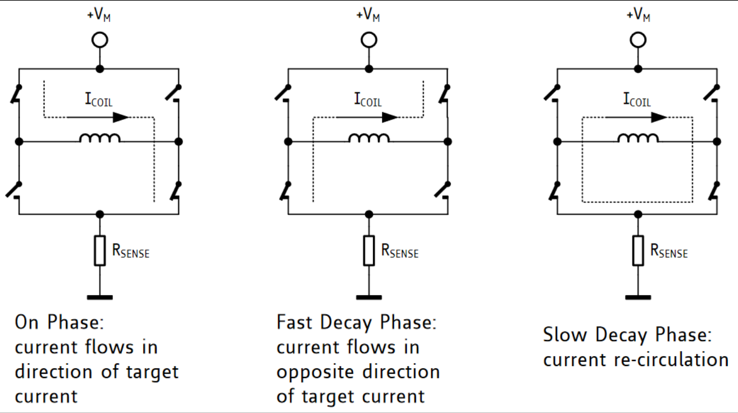

+V_M
I_COIL
R_SENSE
On Phase: current flows in direction of target current
Fast Decay Phase: current flows in opposite direction of target current
Slow Decay Phase: current re-circulation

Figure 3.4 H-bridge and current sense resistor R_{SENSE} of the stepper driver for one motor phase, along with the three stages of the chopper, as shown in TMC5160/A Datasheet [15]. The chopper is used to keep current through the motor constant, regardless of supply voltage, BEMF, etc.

3.7.2 Theoretical BEMF during current zero-crossing

The motor's BEMF constant K_e is equal to torque constant K_t, which is given as follows:

$$K_e = K_t = \frac{T}{I_{RMS}} = \frac{0.3}{\frac{1.28}{\sqrt{2}}} \approx 0.33146 \qquad \left[\frac{V}{\text{rad}/s}\right] \tag{3.1}$$

To find the expected BEMF when the current crosses zero, Eq 2.8 is used with the calculated value for K_e. If there is no load on the motor, load angle $\delta = 0$ and the BEMF is maximized: $\mathcal{E} = K_e\omega$. As the load angle increases, the BEMF is expected to fall to 0 along a sinusoidal shape.

3.7.3 Practical measurements of BEMF during current zero-crossing

In addition to theoretical derivation, practical measurements of BEMF are also done. The measurements are done in part to verify the theoretical BEMF of the unloaded motor and in part to correlate it with simultaneous samples of StallGuard result. A digital oscilloscope (Digilent Analog Discovery 2) is used along with its Software Development Kit (SDK) to automatically measure phase voltage and find BEMF.

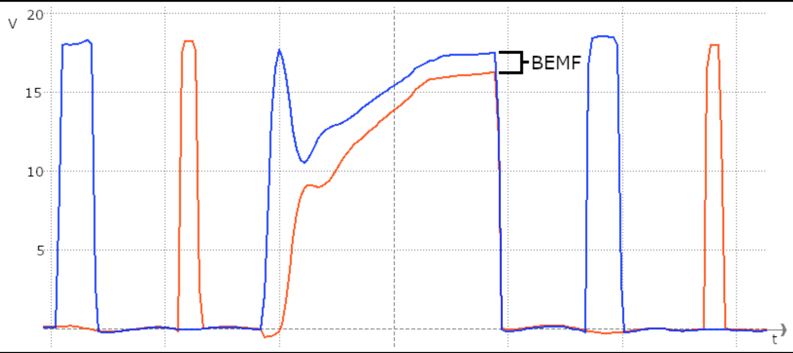


Figure 3.5 An example of the voltages in measured at each side of a phase (in relation to ground) when the current crosses zero. The two pulses to the left and right showcase non-zero current operation: one side of the phase active at a time, with the other being grounded.

Probes are attached to each side of one of the motor phases. Fig. 3.5 shows an example of the probe measurements before, during, and after the phase current crossing zero. The pulses to the left and right show the driver regulating a non-zero current (as per the chopper states shown in Fig. 3.4). The central pattern, where both probes measure non-zero voltage, is assumed to be the driver floating both sides of the coil to be able to measure BEMF. All states of the chopper (current regulation) described in the datasheet have at at least one side of the phase connected to ground.

To automatically measure BEMF, the central pattern in Fig. 3.5 must be found. The oscilloscope can capture a set of samples based on a trigger, but only supports triggering on one of the channels rising or falling. To solve this problem in software, samples are instead continuously captured and examined using the oscilloscope SDK. Based on the figure, to get a BEMF measurement from a set of samples, the objective is to find point at which the lower voltage of the two channels is maximized while both channels are non-zero (floating). Here, the lower channel is given by the red curve, but they alternate since the coil polarity alternates. The following algorithm is used:

1. Set $v_{1,\text{max}} = 0$ and $v_{2,\text{max}} = 0$
2. Capture a set of voltage samples using oscilloscope
3. For each sample pair (v_1, v_2) for channel 1 and 2 respectively:
   - For a threshold v_t, if $v_1 > v_t$ and $v_2 > v_t$: Sample is valid
   - If sample is valid and $\min(v_1, v_2) > \min(v_{1,\text{max}}, v_{2,\text{max}})$:
     Set $v_{1,\text{max}} = v_1$ and $v_{2,\text{max}} = v_2$
4. If at least n valid samples: BEMF $= |v_{1,\text{max}} - v_{2,\text{max}}|$

The voltage threshold v_t is used to discard samples outside of the BEMF measurement window: the central pattern in the figure. By counting the number of valid samples, a BEMF measurement can be discarded if not at least a given portion of the measurement window was captured. For example, sampling could start or end in the middle of the window.

3.8 Graphical Application

The graphical application is developed using Python since it is the language used in the main microscope project. This facilitates merging parts of the work with the main codebase further on, should that be desired. To interface with the GPIO and SPI of the Raspberry Pi, the library `pigpio` is used, which is included by default in the Raspberry Pi Linux distribution Raspberry Pi OS. It runs a daemon as a systemd service on the Raspberry Pi, which a Python program can connect to over a network socket (using the corresponding Python library) and issue commands. For example, it could be transferring data over SPI or configuring GPIO pins. As such, `pigpio` enables the graphical application to be run on a different device from the Raspberry Pi.

To be able to use tuning results in other programs, it is optionally written to a file. For portability and since it is the format used in configuration of the main project, JavaScript Object Notation (JSON) is used to store tuning results. For example, one tuning result is SGT.

3.8.1 Program structure

An overview of the structure of the graphical tuning application is shown in Fig. 3.6. The main goal of the application structure is to reduce coupling between the GUI and hardware by defining an intermediary hardware interface. A low-level interface handles

SPI data transfer and register access, while a higher-lever interface provides e.g. calibration routines and motor movement. Some parts of the GUI need only access specific registers, for example to read motor velocity, and are thus interfacing with low-level functions directly. However, everything beyond simple SPI data transfer and register configuration is done using the high-level functions, keeping GUI logic relatively simple.

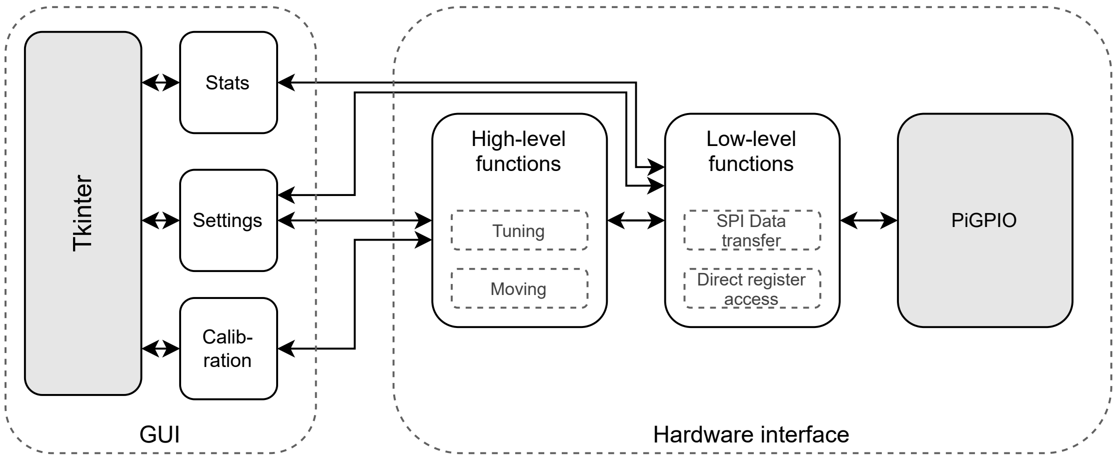


Figure 3.6 Overview of the structure of the graphical tuning software. Generally, it is divided into a front-end (using TKinter) and a hardware interface (using pigpio), which in turn is divided into a high-level and a low-level one.

3.8.2 Scaling to multiple drivers

The low-level interface, as well as most GUI components, are generalised to be able to be instantiated multiple times: one time for each axis. This allows per-axis control without much extra work in implementation. Additionally, axes can be enabled and disabled from a configuration file before without any code changes. At startup, the program simply checks which axes are enabled and instantiates GUI widgets and hardware interface objects for the given axes.

There is some interdependence between the drivers that needs to be considered. Most of it stems from hardware configuration, for example daisy chaining SPI. In order for any SPI communication to work, a driver interface instance most know how many other drivers there are, and in which position it is in the SPI chain. Additionally, since SPI communication is a two-part process (sending datagram, receiving response), care must be taken to avoid any other driver interface instances accessing the SPI bus in the meantime. In the implementation, this is simply done with a global lock (mutex) for any GPIO or SPI access. While more granular locking could be used, for example allowing concurrent GPIO access on different pins, a single lock is used for simplicity as the performance is not needed.

3.8.3 Driver event system for GUI reconfiguration

An architectural problem encountered in development was updating GUI based on driver state without too much coupling. In several cases where the stepper driver's state changes, multiple parts of the GUI need to be updated. For example, many buttons and inputs are disabled until user connects to the driver and at the end of calibration routines, several configuration inputs need to refresh their values. Most of the time, the driver state changes are from user action, but keeping track of what other GUI components are affected by an action is not feasible without high coupling.

The solution used is to introduce a central event system in the driver interface layer ("hardware interface" in Fig. 3.6). All actions on the driver go through the hardware interface and as such, it works as a central point at which the actions can be observed. Events are subsequently dispatched based on actions taken. For example, when starting a calibration routine (such as homing the axis) an event is dispatched that calibration has started. The GUI components themselves listen to the event take action, in this case (most of them) disabling themselves until the event is dispatched that calibration has ended.

3.8.4 Cancellable calibration routines

Calibration routines are long-running tasks, calibrating or tuning one or more values of the stepper driver. The following are implemented:

- Automatic tuning of StallGuard Threshold (SGT)
- Homing axis
- Measuring axis length (finding both endpoints)
- Full calibration with JSON output:
  - SGT for normal operation
  - SGT for homing
  - Axis length

While most other actions simply access one or more register values, the calibration routines move the motor and thus generally take noticeable time (10 to 60 seconds). Due to the time taken and since moving the motor possibly has undesired consequences, all calibration routines are explicitly made cancellable. This means that at any point during calibration, the user can cancel it, returning the driver to a valid, predictable state.

The calibration routines are implemented and made cancellable using Python's library asyncio. An asyncio coroutine (here: the calibration routine) is used to create an asyncio task, which when cancelled throws a `CancelledError` exception in the coroutine. Thus, to make the calibration functions into cancellable asyncio coroutines, the following is done:

- Use asyncio equivalents for IO, delays (`time.sleep`), etc. These are the cancellation points at which `CancelledError` can be thrown.
- When modifying driver settings, catch `CancelledError` to restore driver state on cancellation, then re-raise it to propagate cancellation upwards in the stack.

For most coroutines implemented, restoring driver state simply involves restoring some previously saved values to a set of driver registers. Some coroutines also set the motor in motion. When cancelled, these coroutines stop the motor and wait until the driver has decelerated the motor to standstill before re-raising the `CancelledError`. Effectively, this incorporates motor stopping in cancellation using a feature of asyncio's tasks: that coroutines can delay or even suppress cancellation by catching the corresponding generated exception.

4 Results and Discussion

4.1 Prototype

The resulting prototype setup is shown in Fig. 4.1. The graphical application is connected over a local network to a Raspberry Pi, which in turn controls 3 TMC5160 drivers on a breadboard. One of the 3 stepper motors are showed above, connected to a metal linear stage. A detailed view of the raspberry pi and drivers are shown earlier in Fig. 3.2. 3 separate linear stages are connected simultaneously.

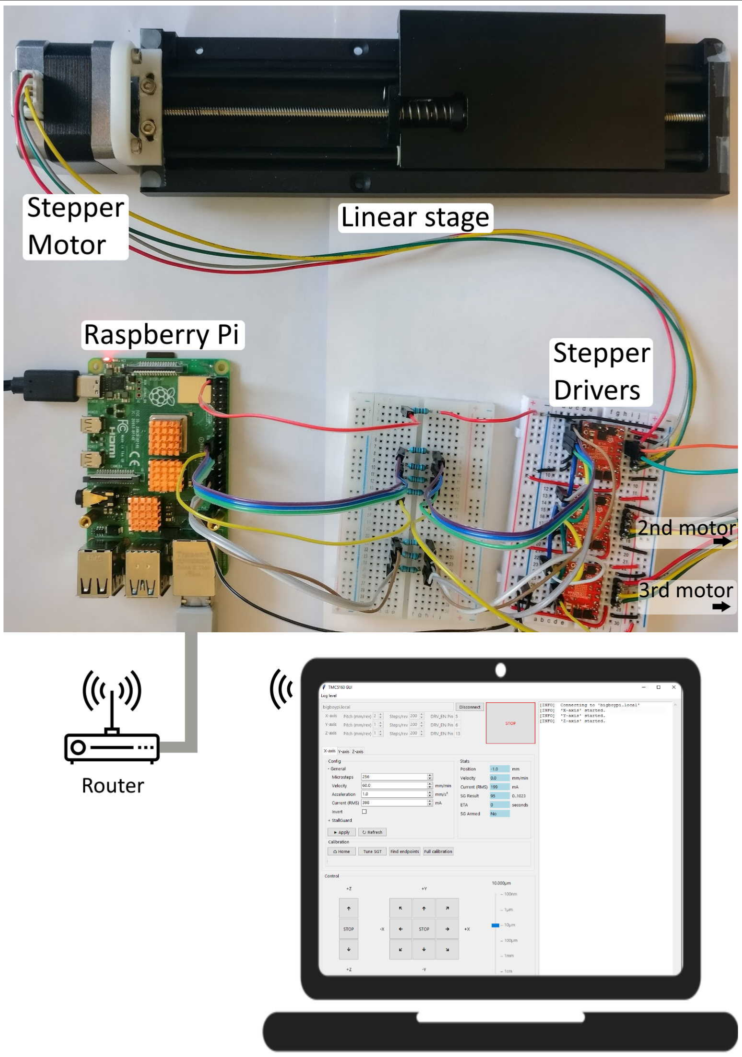


Figure 4.1 An illustration of the complete prototype. Shown is the GUI together with a labelled photo of the hardware. The router symbolises any configuration of a local area network.

4.2 Homing Precision

To evaluate performance of the sensorless homing with respect to precision, it is performed several times while recording the resulting position. The result of 50 samples is shown in Table 4.1 and illustrated in Fig. 4.2, where distance is measured with 5µm precision. Number of microsteps is recorded as reported by the stepper driver. From the results, it can be seen that the sensorless homing has high precision. The resulting position is the same in 94% of cases and the highest recorded offset is 20 µm. As a comparison, the approximate desired precision as per the stakeholder is 100 µm. However, it should be noted that this does not imply good accuracy, which could also be desirable. It is possible that the time and distance from the point of contact to stopping the motor is large.

µm	microsteps	Count	Relative
0	0	47	94%
20	1024	1	2%
-10	-509	1	2%
-5	-253	1	2%
	Total	50	100%

Table 4.1 Resulting position, in both µm and microsteps, from sensorless homing of the metal linear stage. Position is relative and zero point is chosen as the most frequent result. Positive direction is toward the axis endpoint. Homing was done at 200mA and 3 RPS with automatic tuning of SGT each time.

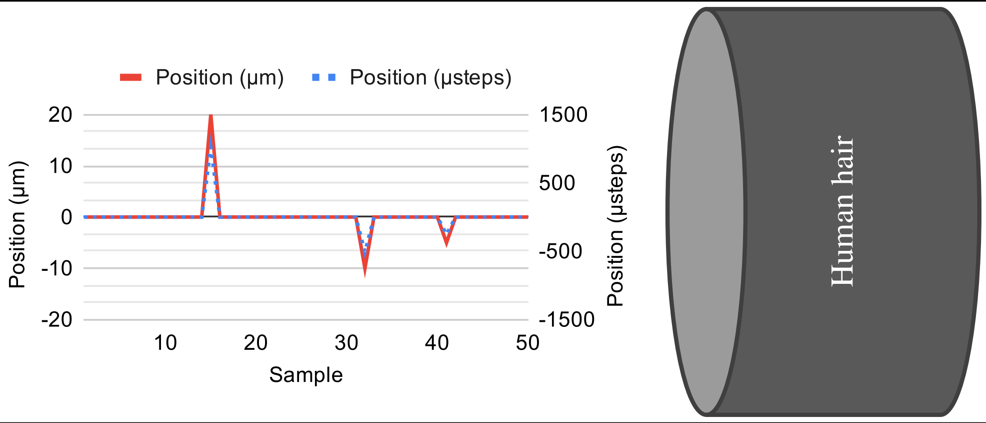


Figure 4.2 Resulting sensorless homing position from Table 4.1 shown as a graph with a to-scale average strand of human hair (75µm diameter) for reference.

The microstep count at each homing position is close to a multiple of 256. During testing, the stepper driver was configured to use 256 microsteps per full step. This shows that, as is already known, StallGuard can only detect a stall each full step, which with the given motor is 200 per revolution.

With a pitch of 2mm and 200 steps per revolution, each full step is theoretically 10μm. This is confirmed to be the case in practice, except when the linear stage platform is close to the endpoint. As can be seen from the homing results, each full step offset (256 microsteps) produces approximately 5μm real offset. The reason is possibly that the linear stage platform makes uneven contact with the endpoint, as can be seen in Figure 4.3. This leads to slower increase in load torque after contact, increasing the time it takes until StallGuard stops the motor. Additionally, it is possible that during this time, softer parts flex and bend, such as the plastic connector between the stepper and the linear stage.

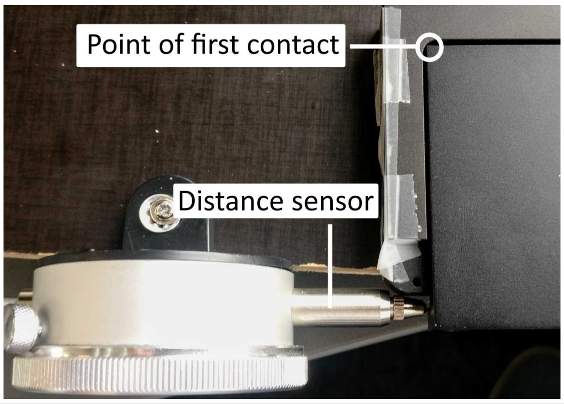


Figure 4.3 Photo of distance sensor and end of linear stage. Due to a rounded corner, the platform makes uneven contact with the linear stage endpoint, leading to a slightly softer stop.

4.3 Automatic Tuning of SGT

The results of the developed automatic tuning of SGT are shown in Fig. 4.4 and Fig. 4.5. The value of SGT found automatically is compared to SGT found manually, demonstrating the viability of the automatically tuned value. Minimum and maximum are the limits defined as per the method in Section 3.5.1.

4.3.1 Effect of velocity

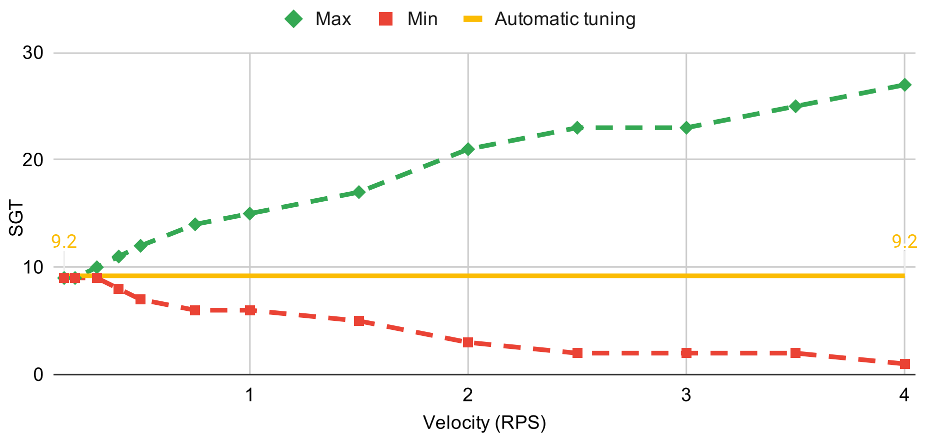


Figure 4.4 Comparison of automatic vs manual tuning for fixed current and different velocities. *Max* is the highest SGT at which a stall can at all be detected. *Min* is the lowest SGT before false positive stalls are detected. An acceptable SGT must be between these values. Motor current is 400mA.

Fig. 4.4 shows dependency on velocity for the range of viable values of SGT, where the motor is run at a fixed velocity in a single direction. Since the automatic SGT is found using a fixed velocity during tuning, it is constant for the whole range of velocity values. From the figure, it can be seen that the automatic tuning finds SGT close to the minimum for low velocities. It is also known from testing during implementation that increasing the velocity during tuning decreases tuned SGT. Both of these observations are to be expected from the way the tuning works: moving at a low velocity and finding the highest SGT (lowest sensitivity) where StallGuard no longer thinks the motor is stalled. In other words, the automatic tuning finds the minimum SGT needed to not trigger a false-positive stall, which by definition is the minimum manual SGT.

As the velocity increases, the range of acceptable values for SGT increases. The figure shows that an acceptable value of SGT for low velocities is also acceptable for the higher velocities tested. The increased range of acceptable values could, in part, be due to the generated BEMF increasing in magnitude, increasing range of BEMF values and thus also being less sensitive to noise. The maximum value shows a close to linear increase with velocity, similar to BEMF. Although not shown in the figure, the minimum value is mostly affected by the StallGuard velocity threshold, below which StallGuard is disabled. As determined in the implementation, it is half the operational velocity (x-axis in figure).

The automatic tuning finds a single value of SGT that works for a wide range of velocities. However, it can still be manually tuned for more control over how sensitive the stall detection is. For example, to avoid damage it is possibly desirable to stop the motor on slightest increase in load. This would not be achieved with the automatic tuning of SGT, but instead require some manual offset or adjusted tuning process.

4.3.2 Effect of current

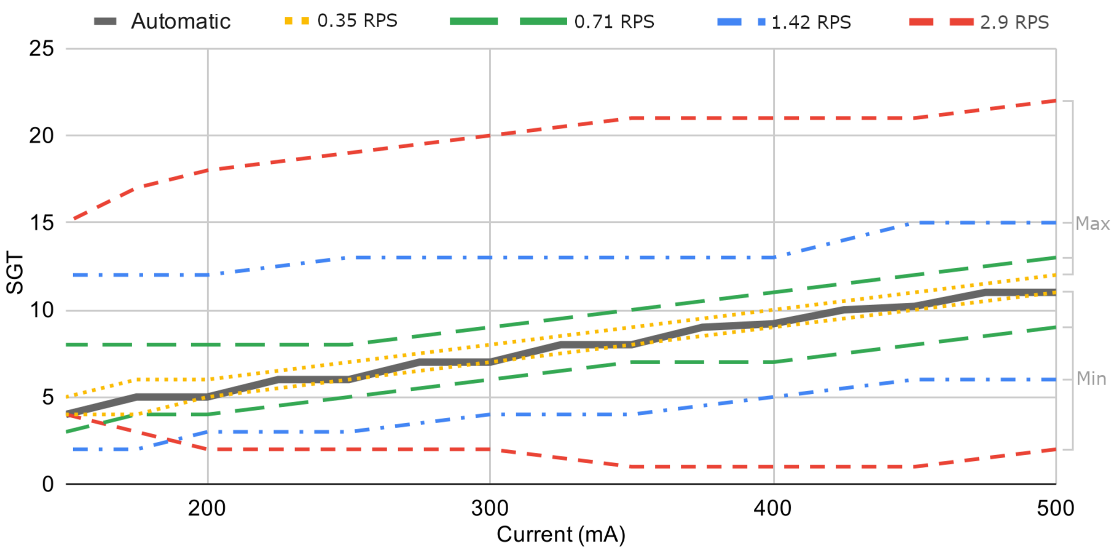


Figure 4.5 Comparison of automatic vs manual tuning of SGT for different velocities at 400mA. *Max* is the highest SGT at which a stall can at all be detected. *Min* is the lowest SGT before false-positive stalls are detected. An acceptable SGT must be between these values.

Fig. 4.5 shows dependency on chosen motor current for both automatic and manual tuning of SGT. In general, it shows that the automatic tuning of SGT produces an acceptable value for a wide range of currents. It also shows that the margin for error is much larger for higher velocities, whereas the current does not affect the range of acceptable values significantly.

Both the lower and upper limit increase with current, especially for lower velocities. One possible reason is that the motor movement is more "stiff" the higher the current and the lower the velocity. Magnetic forces from the coils are higher and the rotor reaches the position of the next step faster. This produces more friction (load torque), higher acceleration, more variance in velocity, and thus more variance in BEMF. Decreasing the current reduces the forces on the rotor, reducing acceleration and velocity of a step. When continually stepping, as is done in testing, it produces a smoother motion instead of a step-wise one. However, consideration must be taken that it also reduces motor torque. At the highest velocity, low torque issues can be seen at less than 200mA, where SGT needs to be increased to avoid false-positives.

4.3.3 Effect of supply voltage

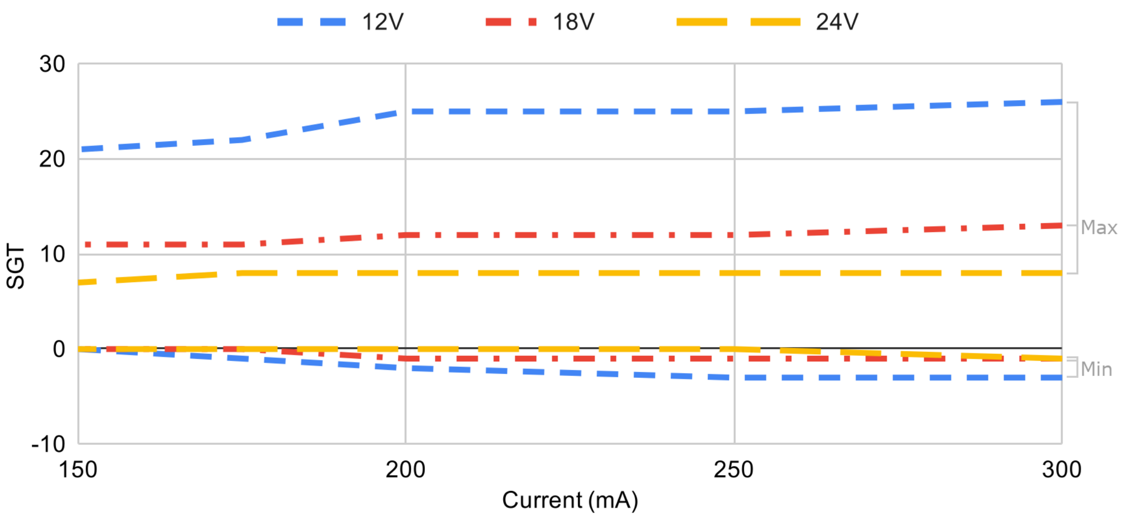


Figure 4.6 Comparison of maximum and minimum values of SGT for different supply voltages and motor currents, using the plastic linear stage. *Max* is the highest SGT at which a stall can at all be detected. *Min* is the lowest SGT before false-positive stalls are detected. An acceptable SGT must be between these values. Motor velocity is 1.42 RPS.

Fig. 4.6 shows dependency on supply voltage for the range of acceptable values of SGT. The figure results are from the plastic linear stage. Thus, they do not directly show the effect of supply voltage for StallGuard when using the metal stage. However, the difference is mainly in friction and rigidity of construction and the results therefore are assumed to be indicative of that of the metal stage as well.

Generally, the range increases with decreasing supply voltage, mostly affecting the upper limit (lowest viable sensitivity). Thus, it is likely easier to tune StallGuard for lower supply voltage. Current has a negligible effect compared to supply voltage for all tested levels. However, the range of tested currents is relatively small.

Although lower supply voltage gives better performance for StallGuard, 12V is possibly too low. Commonly, 3 to 25 times the rated motor voltage (here: 6.4V) is used as supply voltage for a stepper driver [10]. The higher the motor speed, the faster the driver needs to be able to change the phase current. Furthermore, the higher the applied voltage, the faster the current through the motor coils rises or falls. If the motor is run slowly, a lower supply voltage is possibly feasible, but StallGuard will also perform worse due to the lower speed. As such, supply voltage must be chosen on a case-by-case basis, taking into account the inductance of the motor used and the required speed.

4.4 BEMF Measurements

As part of understanding StallGuard, measurements with the motor active are taken on voltage and current of one of the motor phases. Measured values as the current crosses zero are shown in Fig. 4.8. While the current measurement is noisy and inexact (as detailed in Section 3.7), the magnitude still corresponds to magnitude of phase current. It can be seen that, as is expected from the driver chopper phases, the phase voltage fluctuates between positive, negative, and zero. However, as the current reaches zero, a special pattern emerges: the phase voltage slowly decays (demagnetizes) and reaches a stable point. Subsequently, normal chopper operation resumes and current rises again.

The demagnetizing pattern is similar in shape to that of previous research [12], where BEMF was measured as current crossed zero. Given that StallGuard and the method in research measures load once per full step [15], which is the same frequency at which current crosses zero, it is likely that StallGuard is implemented using a similar method. If that is the case, the point of lowest phase voltage before normal chopper operation resumes is the motor BEMF, which is confirmed by correlating measurements with theoretical BEMF in Fig. 4.7.

Fig. 4.7 shows BEMF as a function of velocity at no load, using measurements from oscilloscope SDK as per Section 3.7.3. For each velocity tested, an average over several samples is automatically taken. As can be seen, the measured values closely match the theoretical BEMF of the motor, indicating that the measured values are indeed the motor BEMF. The slight decrease at higher velocities is possibly caused by higher load torque due to friction.

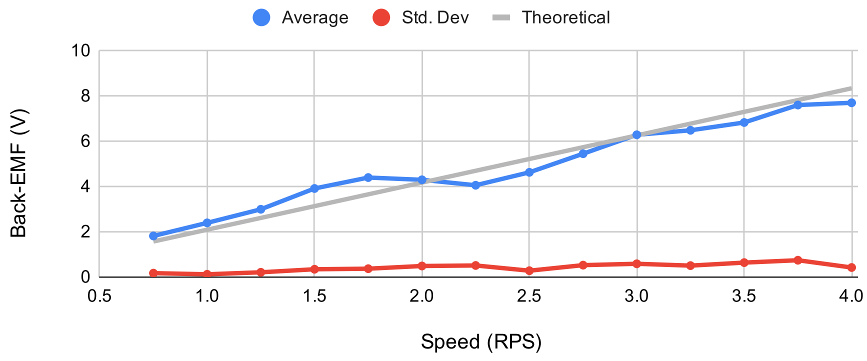


Figure 4.7 Measured BEMF in relation to rotational speed. Theoretical value shown is from linear relationship using BEMF constant $K_e = 2.0826 \, \frac{V}{1/s}$

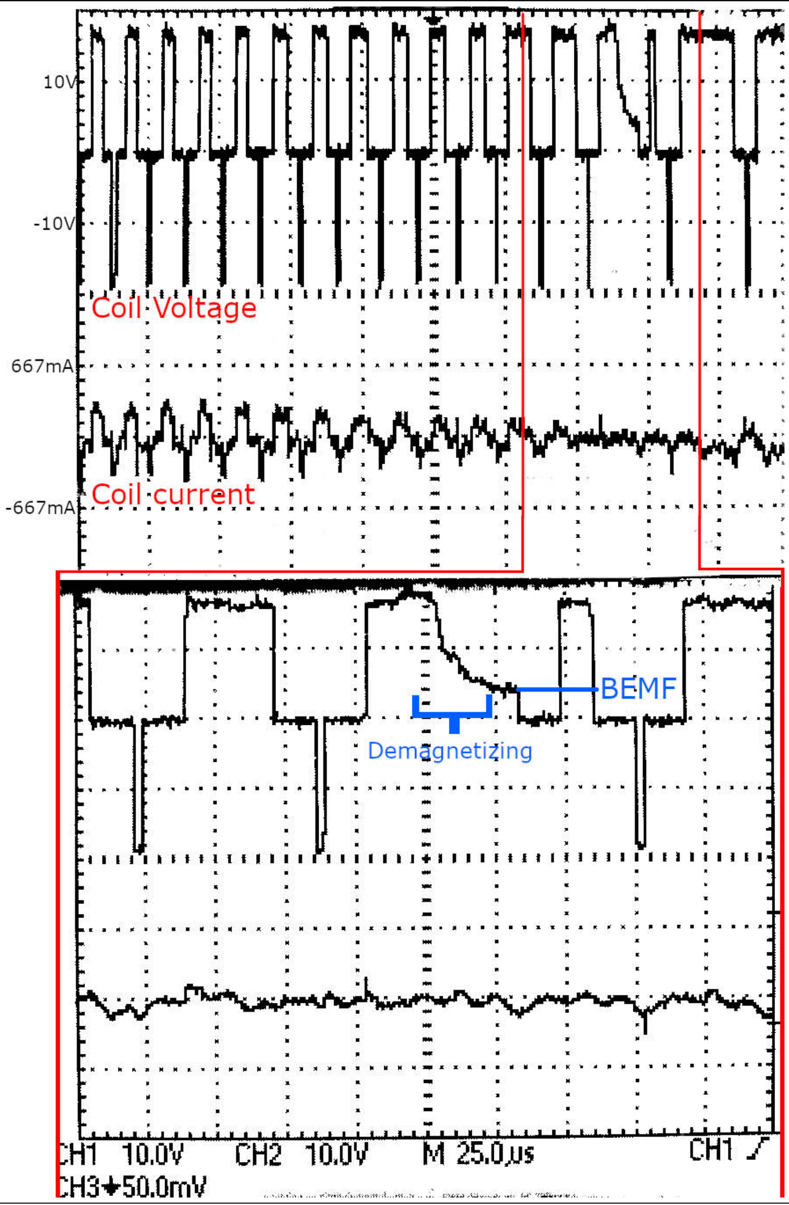


Figure 4.8 Oscilloscope measurements of the voltage and current of one motor phase during current zero-crossing. Current is measured as passing through driver's sense resistor, and polarity thus does not reflect actual direction of coil current. Additionally, it is noisy as it only shows high side of resistor. *BEMF* marks the point at which BEMF is measured as the voltage over the coil.

4.4.1 Correlating BEMF and StallGuard result

Fig. 4.9 shows a comparison of StallGuard result and BEMF measured at current zero-crossing. Except during acceleration and motor stall, there is a close correlation between StallGuard result and measured BEMF. Assuming StallGuard is accurate in measuring load torque, this shows that the measured BEMF, too, is indicative of the load on the motor. Further, it shows that even without the StallGuard feature on the motor driver, reasonable estimates of load torque (and thus stall detection) can be done with the developed program and the digital oscilloscope. In effect, it is a standalone, external StallGuard. However, to measure BEMF, it relies on the stepper driver floating the phase terminals as the current crosses zero. It should also be noted that the BEMF shown in the figure is filtered using a moving average, and thus not as accurate or quick as the StallGuard result. This can especially be seen during stall where the motor vibrates and StallGuard results vary quickly between full load and no load.

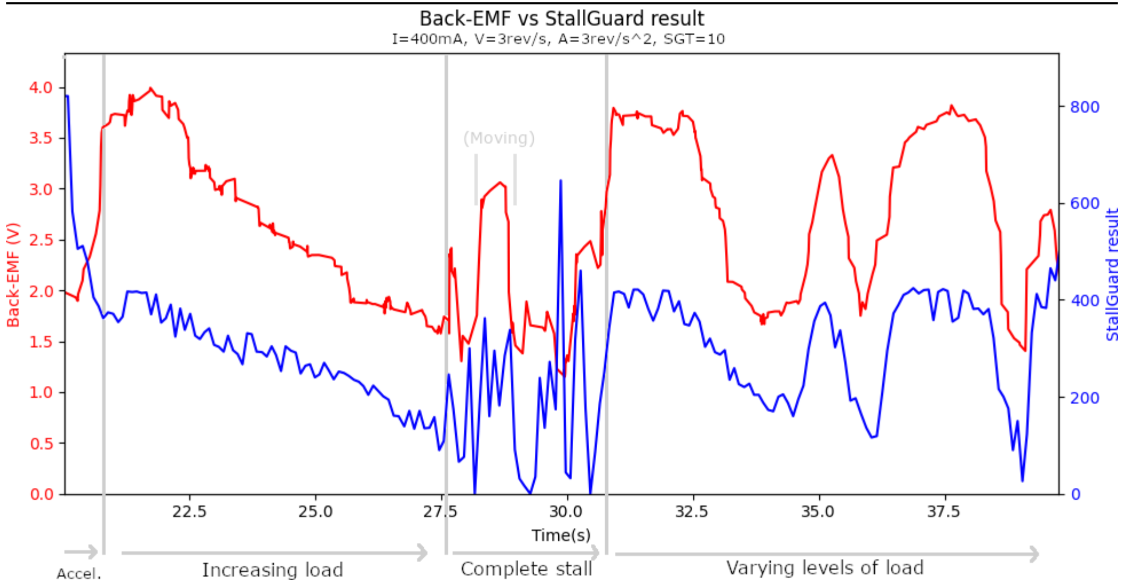


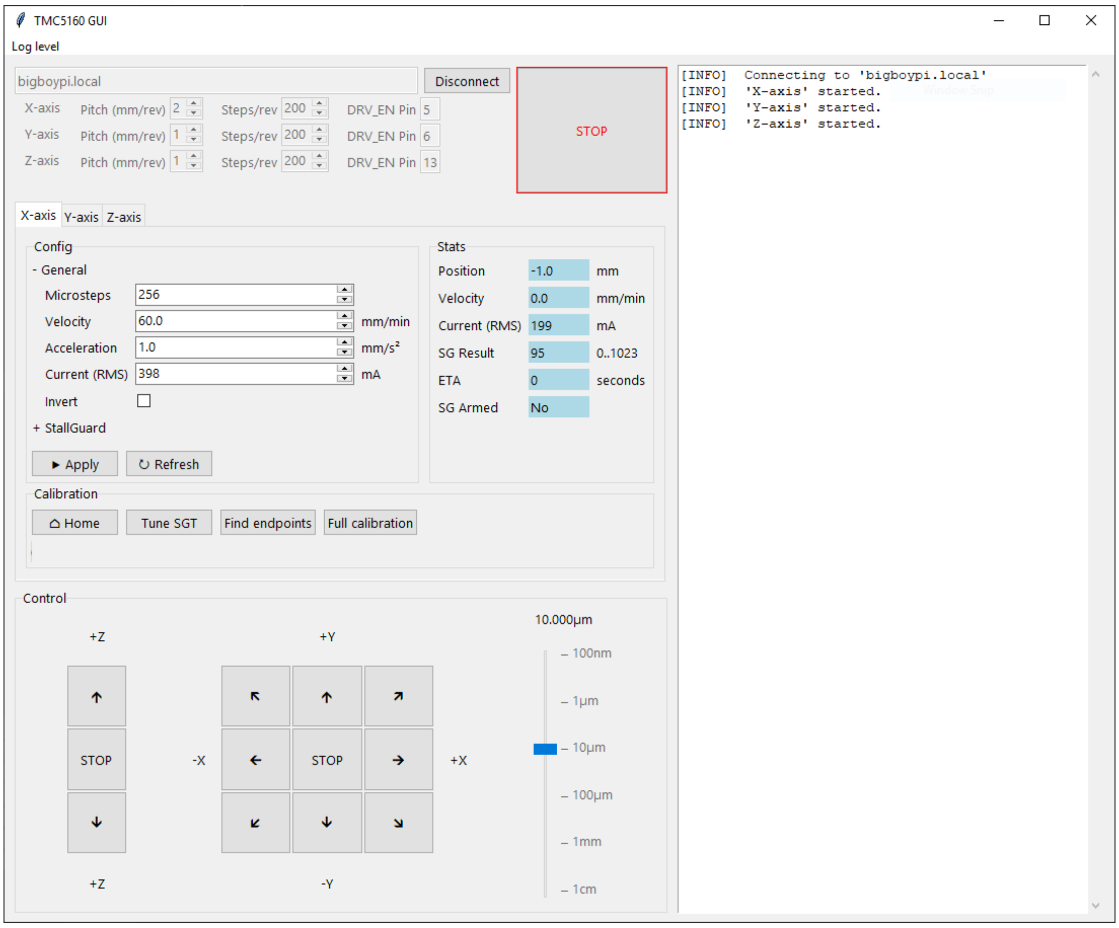
Figure 4.9 StallGuard result and BEMF as measured concurrently over a period of time. After an initial acceleration period, the motor shaft is subjected to increasing load until it stalls. Lastly, varying levels of load are applied. Back-EMF values are filtered using a 6-sample moving average.

4.5 Graphical Application

The resulting graphical utility developed as part of the project is a GUI featuring:

- 3-axis joystick for movement control
- Real-time status of motor and driver
- Configuration of useful registers, some of which are:
  - Current
  - Velocity
  - StallGuard threshold
- Several automatic calibration routines, some of which are:
  - Finding axis length
  - Homing axis
  - StallGuard tuning

A screenshot of the application is shown in Fig. 4.10. In terms of deliverable goals, the application fulfills the requirements as set out by the project aims. The desired features are present, and it works as a graphical interface to configure and tune various settings of the stepper driver. Additionally, it is a useful tool in performing the other tests on the system and gathering data, e.g. for development of StallGuard tuning.



bigboypi.local

X-axis	Pitch (mm/rev)	2	Steps/rev	200	DRV_EN Pin	5
Y-axis	Pitch (mm/rev)	1	Steps/rev	200	DRV_EN Pin	6
Z-axis	Pitch (mm/rev)	1	Steps/rev	200	DRV_EN Pin	13

Config

- General

Microsteps	256	
Velocity	60.0	mm/min
Acceleration	1.0	mm/s²
Current (RMS)	398	mA
Invert	☐	

+ StallGuard

Stats

Position	-1.0	mm
Velocity	0.0	mm/min
Current (RMS)	199	mA
SG Result	95	0..1023
ETA	0	seconds
SG Armed	No	

```
[INFO]  Connecting to 'bigboypi.local'
[INFO]  'X-axis' started.
[INFO]  'Y-axis' started.
[INFO]  'Z-axis' started.
```

Figure 4.10 Screenshot of the graphical tuning application.

5 Conclusions and Future Work

5.1 Conclusions

In this thesis, the aim has been to delve into sensorless load estimation of stepper motors and its use in sensorless homing for cost and complexity reduction of linear stages. The end result is a graphical application able to simultaneously control one or more stepper motors, automatically tune sensitivity of load estimation and, using Trinamic StallGuard, home the linear stages. Evaluation shows high precision of sensorless homing. In 94% of cases, the resulting position is within a ± 5 µm interval, demonstrating sensorless load detection as a promising alternative to limit-switches. The maximum deviation was 20 µm out of 50 attempts, which compared to the stakeholder's desired precision – 100 µm – is acceptable by a large margin.

Additionally, general technical background on sensorless load estimation has been presented, coupled with a practical demonstration of one such method. From measurement of motor Back-EMF (BEMF) using a digital oscilloscope, a custom-made sensorless load detection is able to be realised with characteristics similar to that of StallGuard.

A large part of the work has involved investigation into what conditions are optimal for StallGuard and how it can best be tuned for a given motor current. It has been found that increasing velocity generally has little to no downside with respect to detecting motor stall with StallGuard. Using an algorithm from the stepper driver manufacturer as a basis, StallGuard is able to be automatically tuned to successfully detect motor stall for a wide range of conditions. The result is a sensorless homing that can be performed automatically and precisely with little to no manual input or configuration. Additionally, through the graphical interface, the process is easily performed and monitored by a user.

5.2 Future Work

Promising results have been shown both with respect to the automatic StallGuard tuning and using BEMF measurements as sensorless load detection. It would be interesting to develop these areas more and this section presents some ideas for potential further work.

5.2.1 Load detection using microcontroller with ADC

Measuring BEMF using a digital oscilloscope and a software-based trigger has been shown in this work to deliver promising sensorless load detection. A potential further development could be to use a microcontroller with an analog-to-digital converter (ADC) to, using the same method, sample BEMF and detect load. A microcontroller could react more quickly and possibly sample faster than the oscilloscope-based solution. However, it is still dependent on the stepper driver disconnecting the phase terminals to allow measurement of BEMF. It can not sample load without the correct stepper driver configured in a specific way. Nevertheless, such a system could be interesting, especially so if it could be made to work with stepper drivers incapable of load detection natively.

Alternatively, additional circuitry (e.g. MOSFETs) could be added to enable the microcontroller to on its own periodically disconnect the phase terminals and measure BEMF. It would then be interesting to see if this could be developed to be an add-on for sensorless load detection to existing stepper drivers without such functionality.

5.2.2 Applying controllable and measurable load

As part of an improved tuning algorithm for StallGuard, a way of applying a controllable and measurable load could be incorporated into the system. This would possibly enable tuning it to detect stalling at a specific load torque. By applying a constant torque, StallGuard Threshold (SGT) could be varied until a StallGuard reliably signals a stall.

Additionally, if the controllable load included a sensor of the shaft rotation, it would enable automatically finding the limits of viable values of SGT. It would provide a ground truth of whether the motor is stalled or not, which could then be compared to StallGuard's result to tell if it is too sensitive or not sensitive enough.

5.2.3 Finding area of motion and path finding

A potential further development of homing is to apply it to *all* linear stages to automatically find an area of motion. For example, one could find the linear stage endpoints and put them together into a 3-dimensional cuboid-shaped area. Further, it could then be possible to map physical obstacles inside the cuboid in the way of the linear stage platform. This would enable developing automatic path finding to avoid the obstacles.

References

[1] Center for Disease Control and Prevention. (2021). "NTD Factsheet 2021", Neglected Tropical Diseases, [Online]. Available: https://www.cdc.gov/globalhealth/ntd/resources/ntd_factsheet_2021.pdf [Accessed 04/30/2021].

[2] World Health Organization. (Jan. 17, 2012). "Neglected tropical diseases". Press Release, [Online]. Available: https://www.who.int/news-room/q-a-detail/neglected-tropical-diseases [Accessed 04/30/2021].

[3] Center for Disease Control and Prevention. (Oct. 27, 2020). "Soil-transmitted Helminths", [Online]. Available: https://www.cdc.gov/parasites/sth/index.html [Accessed 04/30/2021].

[4] Trinamic Motion Control, *Parameterization of stallGuard2 and coolStep*, Appl. Note AN002, V1.04, Hamburg, Germany, Nov. 3, 2014.

[5] C. Dragoi and R. Vanwyck, "Stall protection based on back EMF detection", U.S. Patent 20030210011A1, Nov. 13, 2003. [Online]. Available: https://patents.google.com/patent/US20030210011A1/en [Accessed 04/30/2021].

[6] M. S. Arefeen and J. D. Childers, "Method of stall detection for stepper motor system", U.S. Patent 7224140B2, May 29, 2007. [Online]. Available: https://patents.google.com/patent/US7224140B2/en?oq=us+7224140 [Accessed 04/30/2021].

[7] R. P. Russ and J. Turner, "Method and apparatus for stepper motor stall detection", U.S. Patent 7880423B2, Feb. 1, 2011. [Online]. Available: https://patents.google.com/patent/US7880423/en [Accessed 04/30/2021].

[8] L. Karlsson, "Sensorless Control of a Hybrid Stepper Motor", Linköping University, Vehicular Systems, SE-581 83 Linköping, LiTH-ISY-EX–16/4962–SE, 2016, p. 66.

[9] V. Klar, J. M. Pearce, P. Kärki, and P. Kuosmanen, "Ystruder: Open source multifunction extruder with sensing and monitoring capabilities", *HardwareX*, vol. 6, e00080, Oct. 1, 2019, ISSN: 2468-0672. DOI: 10.1016/j.ohx.2019.e00080. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2468067219300471 [Accessed 02/02/2021].

[10] Advanced Micro Systems. (2018). "Stepper Motor Control Basics", [Online]. Available: http://stepcontrol.com/stepping101/ [Accessed 03/02/2021].

[11] S. Derammelaere, C. Debruyne, F. Belie, K. Stockman, and L. Vandevelde, "Load angle estimation for two-phase hybrid stepping motors", *IET Electric Power Applications*, vol. 8, pp. 257–266, Aug. 1, 2014. DOI: 10.1049/iet-epa.2013.0333.

[12] S. Derammelaere, L. Carlier, B. Vervisch, C. Debruyne, K. Stockman, P. Cox, L. Vandevelde, and G. V. d. Abeele, ”The opportunities of two-phase hybrid stepping motor back EMF sampling”, in *2011 IEEE Energy Conversion Congress and Exposition*, ISSN: 2329-3748, Sep. 2011, pp. 83–87. DOI: 10.1109/ECCE.2011.6063752.

[13] S. Derammelaere, B. Vervisch, F. Belie, J. Cottyn, G. Van den Abeele, P. Cox, K. Stockman, and L. Vandevelde, ”A nonlinear and linear model of a hybrid stepping motor”, Jun. 6, 2011.

[14] P. Cox and B. De Cock, ”Output Contact for Feedback in Integrated Circuit Motor Driver”, pat. 20 080 218 113, Sep. 11, 2008. [Online]. Available: https://www.freepatentsonline.com/y2008/0218113.html [Accessed 02/25/2021].

[15] Trinamic Motion Control, *Power driver for stepper motors*, TMC5160/A Datasheet, Rev. 1.14, Hamburg, Germany, May 19, 2020.

[16] Casun Motor, *Hybrid Stepping Motor*, Type 42SHD0034-226NTP, Rev. A, Guangzhou, China, 2020.

[17] Trinamic Motion Control. (2021). ”SilentStepStick”, [Online]. Available: https://www.trinamic.com/support/eval-kits/details/silentstepstick/ [Accessed 03/30/2021].

[18] Raspberry Pi Trading Ltd. (Jan. 2021). ”Raspberry Pi 4 Product Brief”, [Online]. Available: https://datasheets.raspberrypi.org/rpi4/raspberry-pi-4-product-brief.pdf [Accessed 05/12/2021].

[19] Raspberry Pi Foundation. (2021). ”Operating System Images - Raspberry Pi”, [Online]. Available: https://www.raspberrypi.org/software/operating-systems/ [Accessed 05/13/2021].

[20] Python Software Foundation. (2021). ”General Python FAQ”, Python 3.9.5 Documentation, [Online]. Available: https://docs.python.org/3/faq/general.html [Accessed 05/13/2021].

[21] Python Software Foundation, *tkinter — Python interface to Tcl/Tk*, Python 3.8 Standard Library, 2020. [Online]. Available: https://docs.python.org/3.8/library/tkinter.html [Accessed 05/04/2021].

[22] Tcl Developer Xchange. (2021). ”Tcl Developer Site”, [Online]. Available: https://www.tcl.tk/ [Accessed 05/13/2021].

[23] Python Software Foundation, *asyncio — Asynchronous I/O*, Python 3.8 Standard Library, 2020. [Online]. Available: https://docs.python.org/3.8/library/asyncio.html [Accessed 05/04/2021].

[24] Joan2937, *pigpio library*, Github Repository, 2021. [Online]. Available: https://github.com/joan2937/pigpio [Accessed 05/13/2021].

Appendix A Physical test setup

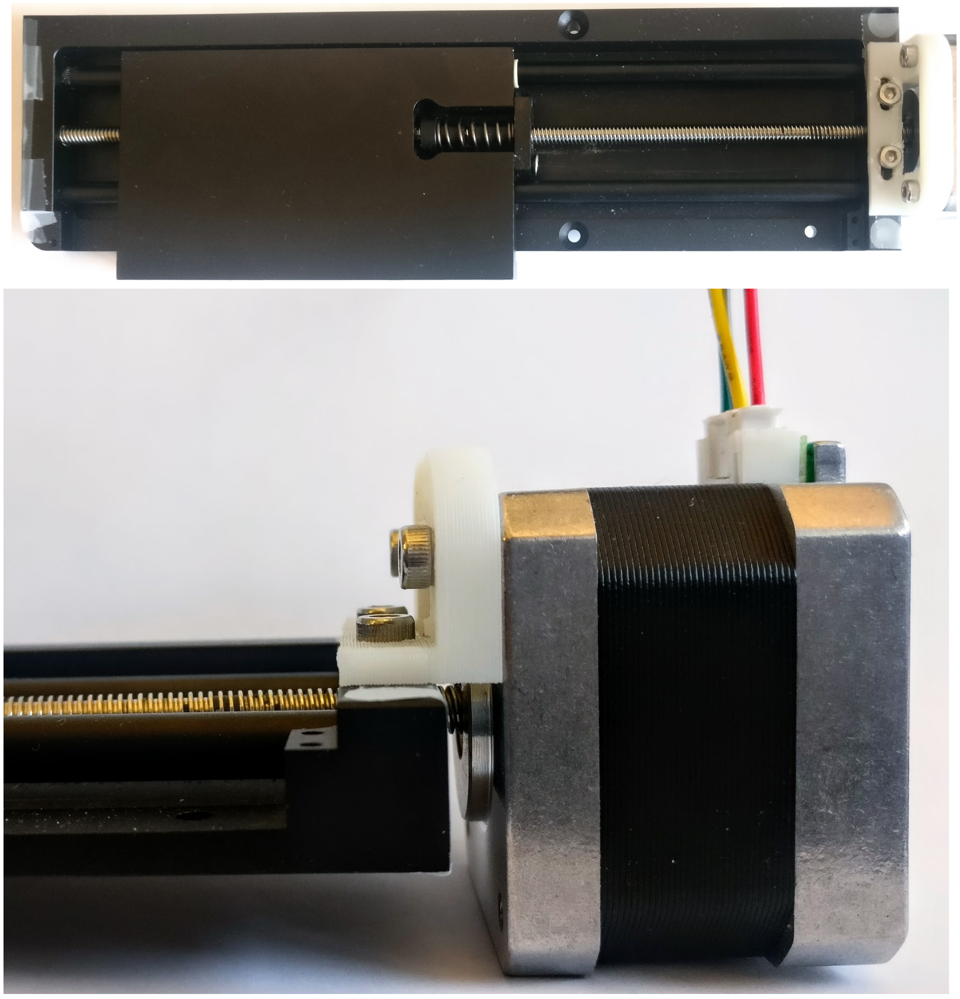


Figure A.1 Photos of the metal stage from the side and above. Everything is metal except the white angle bracket, which is 3D printed plastic, and the black central part of the motor.

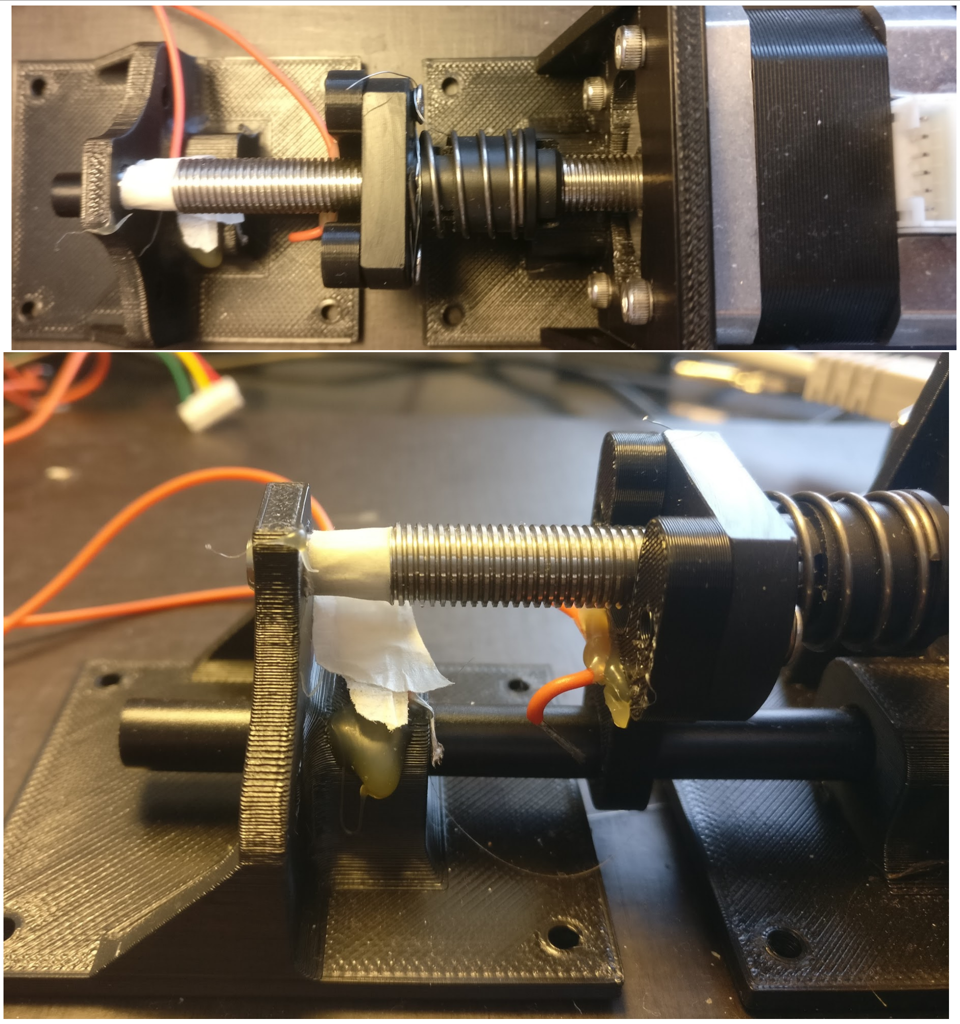


Figure A.2 Photos of one of the plastic stages from the side and above. Most parts are 3D printed plastic. The hot-glued orange wire is an old limit switch not used in this project.