DEGREE PROJECT IN MECHANICAL ENGINEERING,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2021

Design and control of a 3D printed, 6DoF robot arm

MICHAL GABRIEL SAWCZUK

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

Design and control of a 3D printed, 6DoF robot arm

MICHAL GABRIEL SAWCZUK

Bachlor's Thesis at ITM
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2021:52

Abstract

The purpose of this thesis was to design, construct and control a robotic arm with six degrees of freedom. The arm should be able to do simple tasks such as pick and place with good accuracy and without using external sensors. This thesis investigates the precision and the strength of the constructed robot arm.

The arm was constructed using 3D printed parts and commonly available hardware such as threaded rods, bearings, screws and nuts. Each axis uses a combination of pulleys and belts in order to achieve desired torque. A differential transmission was implemented in four of the axes in order to combine the power of the motors and reduce weight in the upper parts of the arm.

The robot is driven by six stepper motors that are controlled by a combination of RAMPS 1.4 shield and Arduino Mega 2560 microcontroller. The user can manipulate each axis by sending commands to the Arduino through an USB cable. The commands are generated with the help of a simple user interface written in Python.

Experiments have shown that the arm has an average error increase of 0.0289-0.1356 mm for each movement, depending on the chosen speed. The maximum amount of weight that the arm can hold in the worst case scenario is 0.84 kg.

Keywords: Mechatronics, 6DOF Robot Arm, Robotics, Robot, Arduino

Referat

Syftet med denna avhandling var att designa, konstruera och kontrollera en robotarm med sex frihetsgrader. Armen ska kunna utföra enkla uppgifter som pick-and-place med god noggrannhet och utan användning av externa sensorer. Denna avhandling undersöker precisionen och styrkan hos den konstruerade robotarmen.

Armen konstruerades med 3D-printade delar och lättillgänglig hårdvara som gängstänger, lager, skruvar och muttrar. Varje axel använder en kombination av kuggremskivor och kuggremmar för att uppnå önskat moment. En differentialväxel användes i fyra av axlarna för att kombinera motorernas moment och minska vikten i armens övre delar.

Roboten drivs av sex stegmotorer som styrs av en kombination av RAMPS 1.4-shield och Arduino Mega 2560 mikrokontroller. Användaren kan styra varje axel genom att skicka kommandon till Arduinon via en USB-kabel. Kommandona genereras med hjälp av ett enkelt användargränssnitt skrivet i Python.

Experiment har visat att armen har en genomsnittlig felökning på 0,0289-0,1356 mm för varje rörelse, beroende på vald hastighet. Den högsta vikt som armen i värsta fall kan hålla är 0,84 kg.

Nyckelord: Mekatronik, Robotarm, Robotik, Robot, Arduino, Sex frihetsgrader

Acknowledgements

I would like to thank my supervisor Nihad Subasic for his lectures, knowledge and feedback, and Staffan Qvarnström for taking care of my component orders. I would also like to thank all the other students that took their time to give me feedback and answer my questions.

Contents

List of Figures

List of Tables

List of Abbrevations

$3D$	Three Dimensional
CAD	Computer Aided Design
DC	Direct Current
DOF	Degrees of Freedom
FDM	Fused Deposition Modeling
PLA	Polylactic Acid

Chapter 1

Introduction

1.1 Background

Robotic arms are machines that are programmed to do a variety of tasks ranging from welding and machine tending to transporting astronauts during spacewalks. The demand for automation is rising. At the same time, the possibilities of creating robot arms used for less demanding applications are growing. Countless Open-source robot arm projects can be found on the internet and the number of cheap, commercially available robot arms is growing every year. This project is about creating a prototype of a cheap and reliable robot arm using widely available manufacturing methods such as 3D printing and cheap actuators such as stepper motors.

1.2 Purpose

The purpose of this bachelor's thesis is to design, build and control an articulated robot arm with six degrees of freedom. The arm should be accurate and strong enough to able to do different tasks such as pick and place without using external sensors. This report will investigate the following research questions:

- How precise is the robot arm?
- How does the speed of the arm affect the precision?
- How much weight can the robot arm lift?

1.3 Scope

The time frame for the bachelor's degree project in mechatronics at KTH Royal Institute of Technology is a period of around four months. To meet the established time-constraints, the arm will not contain any extra features such as limit switches or motor encoders.

1.4 Method

1.4.1 Design and construction

The first step of designing a robot arm was a design study of the existing arms. Afterward a 3D model of the robot arm was created using Autodesk Fusion 360. The finished construction is shown in Figure 1.1. The model was 3D printed using a modified Creality Ender 3 and an Anet ET5 3D printer using PLA. The 3D printed parts were connected using screws, threaded rods and nuts.

Stepper motors were used to move the axes of the robot arm. To increase the torque, timing belts in combination with 3D printed geared pulleys were used. The amount of weight in the upper parts of the arm was reduced by implementing differential transmission for axis pairs 3,4 and 5,6 (see Figure 1.1). Deep-groove ball bearings were used in every axis to minimize the friction and a thrust(axial) bearing was used in the first axis to support the weight of the arm. A servo motor was used in the gripper of the arm. The gripper itself was bought as a separate component.

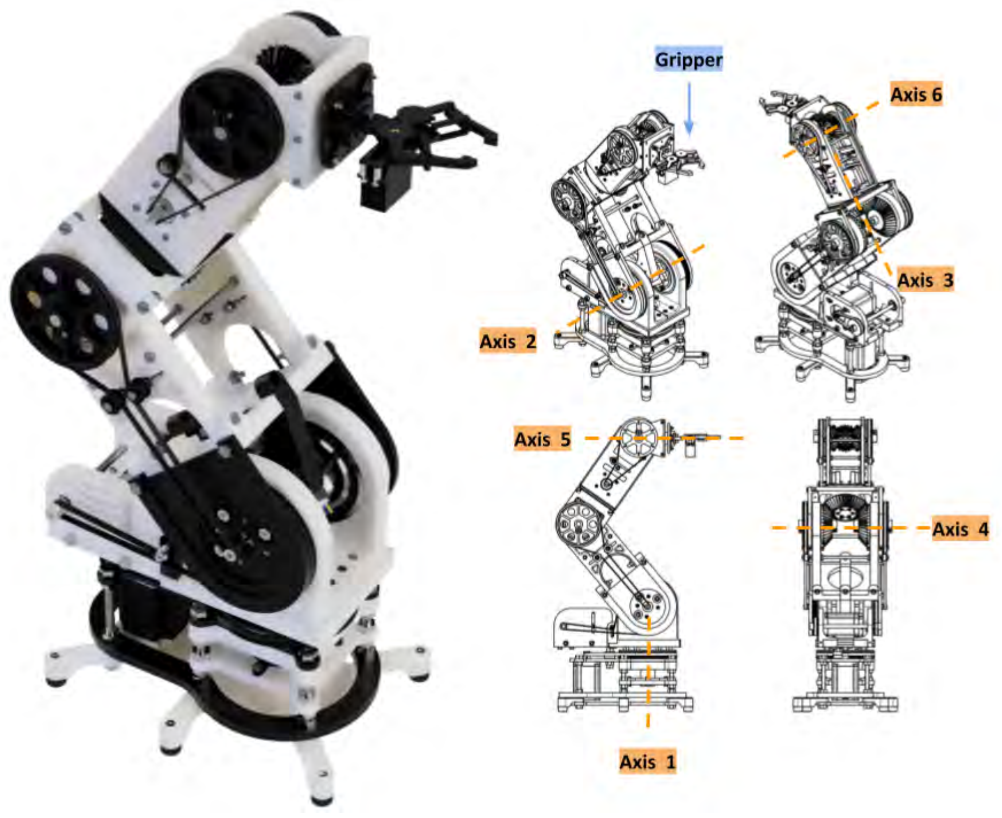


Figure 1.1. CAD model and drawings of the robot arm.

1.4.2 Electronics and control

In order to control the arm, RAMPS 1.4 shield with Arduino Mega 2560 microcontroller was chosen (see section 2.3.1). Six stepper motor drivers that control the motors are connected to the RAMPS and cooled down by a computer fan. The components are powered by a power supply. The servo motor could not be powered directly from the RAMPS, therefore a power converter was used to adjust the voltage. A light emitting diode in combination with a buzzer are used as an indicator. All these components were enclosed in an acrylic cage shown in Figure 1.2. Three different plugs and two power switches are located on the enclosure walls. The USB plug is used to control the Arduino from a computer, the IEC connector (three pins) connects a mains cable with the power supply and the last one connects to the robot arm.

The arm is controlled by sending commands though an USB cable to the Arduino which in turn translates the commands to different stepper positions. A simple user interface was programmed to simplify that procedure.

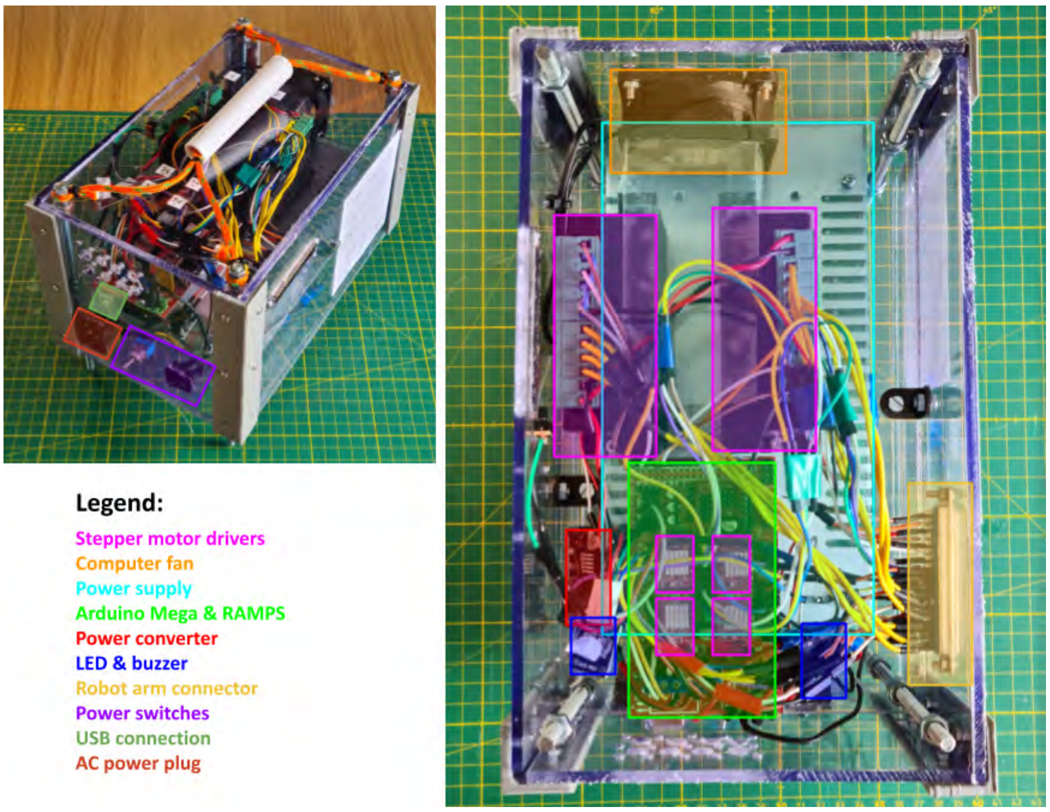


Figure 1.2. Photo of the final electronics enclosure.

1.4.3 Testing

Different tests were performed to answer the research questions defined in section 1.2. The first question about the precision was answered by performing physical experiments. An indicator dial was placed in front of the robot arm and the arm was then repeatably moved to that position. If the arm is precise, value of the indicator dial will stay approximately the same. Two different speeds were used in order to determine how the speed affects the precision. The question about the strength of the arm was answered by loading each axis until it yields.

Chapter 2

Theory

2.1 Construction

2.1.1 Design for FDM printing

Fused deposition modeling (FDM) is one of the most common 3D printing processes. Solid material is melted and extruded through a nozzle onto a build plate. The print head traces out the 3D model layer by layer until the object is created [4]. Different types of material can be used, the most common one is PLA (Polylactic acid). It is easy to print, low cost, stiff and has good strength. The main disadvantage of PLA is the low service temperature of about 57 °C [5].

As a result of the layer by layer printing, it can be hard to determine the exact mechanical properties of a 3D printed part. Furthermore a model can be printed with different settings such as layer height or printing speed. It can also be printed in different orientations and using different nozzles. All these factors are influencing the mechanical properties of a printed part.

As mentioned earlier, the FDM parts are printed layer by layer. Therefore it can be expected that these parts have transversely isotropic[1] properties. According to experiments done by T. Yao, Z. Deng, K. Zhang and S. Li, the tensile strength is 52.29% to 47.46% lower (depending on the layer thickness) when exposed to stress perpendicular to the printing plane [6].

To reduce the weight of a 3D printed part, a property called infill can be used and modified. Infill is the material that exists within a model. It can be printed in different patterns and densities effecting in different mechanical properties. According to research done by C. Lubombo and M. Huneault, a square infill structure gives the best mechanical performance for uniaxial tensile loading. In terms of flexural loading(bending), the hexagonal infill works best. The infill is contained within walls. A higher amount of walls gives a part with better mechanical properties [7].

According to research done by Harshit K. Dave, layer height and printing speed

[1]Transversely isotropic materials have the same properties in all directions within a plane but different properties in the direction perpendicular to the plane

does not have a significant effect on the mechanical properties of the 3D printed part. However it is important to note that the layer height and printing speed is limited to the nozzle size and the 3D printer used [8].

2.1.2 Transmission

Transmission is the essential part of many mechanical systems. It is a device that provides torque and speed transitions between different parts of the machine by using components such as gears, belts and chains [9].

One of the main notions in transmission design is the transmission ratio, it is the ratio of the speed of the input shaft to the speed of the output shaft. It can be calculated as

$$GR = \frac{n_1}{n_2} = \frac{z_1}{z_2} = \frac{d_1}{d_2} \tag{2.1}$$

where n is the speed of the shaft, z is the number of teeth and d is the diameter of the gear[10].

This project uses a combination of belts and 3D printed gears to reduce the speed and increase the torque of the output shafts. Furthermore, a simplified version of a differential is used to transmit the power in Axis 3,4 and 5,6. A differential is most often used in automotive mechanics to distribute the power to both wheels, but allowing them to run at different speeds [11]. In this project it is used to distribute the power to both Axes (3,4 or 5,6, see Figure 1.1) using the same motors. The differential gear used in this project is shown in Figure 2.1. When the side(black) gears rotate in different directions, the top(blue) gear rotates around its own axis (y-axis in the Figure). When they rotate in different directions, the top gear rotates around the side gear's axes instead (x axis in the Figure).

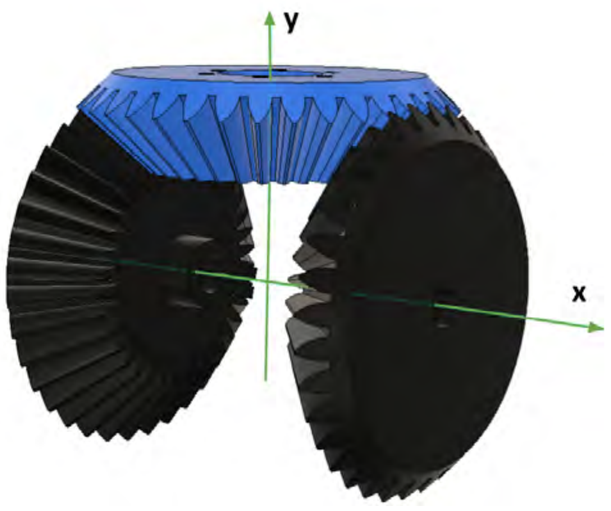


Figure 2.1. Simplified Differential.

2.1.3 Simulation

Before the construction process, it is advantageous to simulate the different forces in order to gain an understanding of the robot dynamics and kinematics. It can be done in many different ways, but in this project, a simulation tool called Acumen was used. Acumen is used for simulating mathematical models and visualizing them in 3D [12].

2.2 Actuators

Actuators are responsible for movement of a machine and are therefore one of the most important components in a robotic arm. Without actuators, the robot cannot move through space. Is important that the actuator is strong, fast and accurate enough for the application. There are many different types of actuators such as stepper motors, brushless motors and servomotors.

2.2.1 Stepper motors

Stepper motors are brushless DC motors that move in fixed increments called steps. Inside a stepper motor there are coils that act as electromagnets when powered. The coils are organized intro different groups called phases. The rotor aligns with the field generated by powering one of the phases, by powering the phases on and off in a sequence, the rotor rotates.

Stepper motors are used for many precision motion applications. The advantages of using stepper motors is the precise positioning, speed control and high torque at lower speeds. The disadvantages of the stepper motors is the low efficiency and low torque at higher speeds. It is important to note that even though the stepper motors are precise, they are not in a closed loop, any steps lost will not be recorded. For that reason it is important that the motor is sized to its application. To prevent accumulation of lost steps, limit switches can be used. Limit switches are switches that are often used in CNC machines and 3D printers to give the machine a reference (home) position. These switches are also good for safety reasons because they limit the allowed angle of rotation [13].

There are a broad selection of different stepper motors. The three main types are:

- Permanent magnet stepper motors (PM)
- Variable reluctance stepper motors (VR)
- Hybrid synchronous stepper motors (HY)

The first type (PM) tend to have a higher torque, but poor precision. The second type (VR) is the opposite, the precision is high and the torque is low. The last type (HY) combines the best of both worlds with good precision and high torque. The disadvantage of the HY stepper motors is the higher price and weight.

Stepper motors can also be divided by the type of winding arrangement of the coils. The two main types are the bipolar and unipolar stepper motors. With the bipolar arrangement, the magnetic pole is reversed by reversing the current in the coil. The unipolar arrangement doesn't need a reverse current to change the magnetic pole. It is generally better to use bipolar stepper motors because of the higher efficiency and torque. Even though the bipolar stepper motors require more complicated drivers, it is not a problem anymore because of the availability of these drivers [14].

Because of the advantages, only bipolar hybrid synchronous stepper motors are used in this project.

Coils in a stepper motor can be energized in different ways that gives different drive methods. The most common ones are the:

- Full-step mode (one phase)
- Full-step (two phases)
- Half-step mode
- Microstep mode

When using one phase full-step mode, the phases are energized one after another. As the name suggests, the rotor turns one step at a time. In the two phase full-step mode two phases are energized at the same time. The rotor is still turning one step at a time but the torque increases by 30-40 %. Unfortunately, energizing two phases at the same time requires double the current. Half-step mode is achieved by combining one- and two phase full-step modes. By alternating between energizing two phases and one phase, the rotor can be aligned between two windings. For that reason, the step angle is reduced by half. The disadvantage of this mode is the reduced torque. Microstepping allows a stepper motor to make even smaller steps by using stepped sine waves. The motor can now turn smoothly but the torque is decreased even more [14].

2.2.2 Servo motors

Servo motors are motors that allows a precise control of the position, speed and acceleration of the output shaft. A common "hobby servo" consists of a DC motor, a gear train, a potentiometer and a circuit board to decode the signals. It is controlled using PWM (Pulse Width Modulation) signals from the microcontroller. A PWM signal can either be high or low. The percentage of the high's is called duty cycle, for instance, a signal that is high for 0.8ms and low for 0.2ms has a duty cycle of 80%. Different duty cycles generate different servo positions [15].

2.3 Control

2.3.1 Computation and control

In order to control the robot arm actuators, Arduino Mega 2560 with a RAMPS 1.4 is used. Arduino Mega 2560 is a microcontroller based on the ATmega2560. It has 54 digital pins, 16 analog inputs and four hardware serial ports (UART) [16]. RAMPS 1.4 is a controller board that can be mounted on top of the Arduino Mega 2560. It is convenient because of the sockets that smaller stepper motor drivers can be mounted into (see section ”Stepper motor drivers”). As a result, it is often used in homemade CNC machines and 3D printers[17].

The arm is controlled by sending commands from a computer to the Arduino through a USB connection. It is done using serial communication. It means that the data is streamed one bit at a time as shown in Figure 2.2. [1]

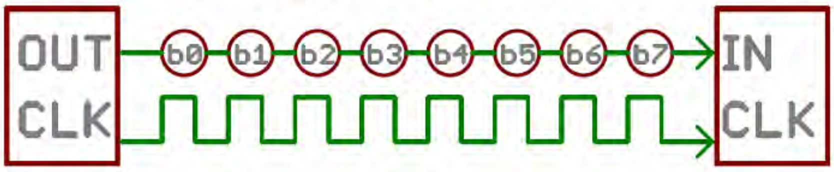


Figure 2.2. How the data is transmitted using serial communication.

2.3.2 Stepper motor drivers

In order to control the stepper motors, stepper motor drivers needs to be used. There are many different drivers on the market and they need to be compatible with the stepper motor to provide good performance. In this project, two different stepper motor drivers are used, A4988 and TB6600. A4988 can handle current up to 2A and allows microstep resolution of up to 1/16[2]. TB6600 is often used for bigger stepper motors that require current up to 4A and has a maximum microstep resolution of 1/32. For a standard 1.8 degree stepper motor, a microstep resolution of 1/16 gives 3200 steps per revolution, and a resolution of 1/32 gives 6400 steps per revolution [3]. Data sheets for the stepper motor drivers used are presented in Appendix A.

Three different signals are used when controlling the stepper motor drivers, DIR, STEP and EN. By sending a STEP signal, the motor moves one step (or microstep depending on the chosen setting) in the direction given by DIR at the time of sending the signal. EN pin is used to reduce the power used by the stepper motor when stationary [14].

2.4 Testing

2.4.1 Accuracy, precision and trueness

There exists different definitions of accuracy, precision and trueness and this projects uses the ISO 5725-1:1994 definition: *"ISO 5725 uses two terms "trueness" and "precision" to describe the accuracy of a measurement method. "Trueness" refers to the closeness of agreement between the arithmetic mean of a large number of test results and the true or accepted reference value. "Precision" refers to the closeness of agreement between test results."* [18].

A simpler way of describing these concepts is shown in Figure 2.3 by using a pistol targets. All the shots on target A are in the middle and close to each other, there is both trueness and precision, therefore the accuracy is also high. If all the shots are in the middle, but not close together as the target B shows, there is trueness but the shots are not precise. On the other hand in C, the shots are precise but lack trueness. Lastly, when the shots are not close together and not in the middle as in target D, there is no precision and no trueness.

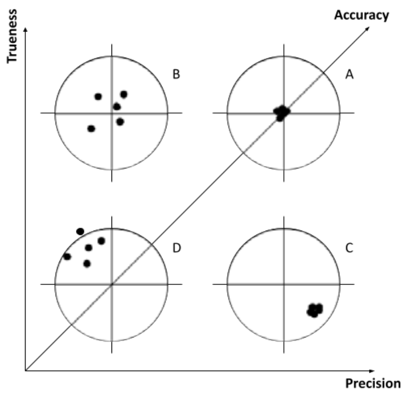


Figure 2.3. Accuracy, trueness and precision.

2.4.2 Errors

Errors can be divided into two different types, systematic and random. Systematic errors can be predicted because they always affect the measurement in the same way. It also means that they often can be eliminated if the cause of the error is found. Random errors are unpredictable and often are normally distributed [19].

Chapter 3

Prototype

3.1 Construction

The robot arm was designed using Fusion 360. The different axes were constructed one at a time, starting with the first axis and ending with the sixth. This approach enabled testing of each axis directly after it was assembled. Exploded views of each axis are presented in Appendix B. Additionally, a simple simulation was created in Acumen in order to simulate the different forces. The simulation is presented in Appendix C.

Axis 1

The first Axis is located at the base of the robot arm. It is therefore necessary to make it as stable and durable as possible so it doesn't tip or break under the weight of the arm. Two different designs were constructed as shown in figure 3.1. The frame of the first iteration consisted of only two parts that were connected together to form an enclosure. It had many issues such as low strength and very long printing time. It was quickly abandoned and replaced with the second version.

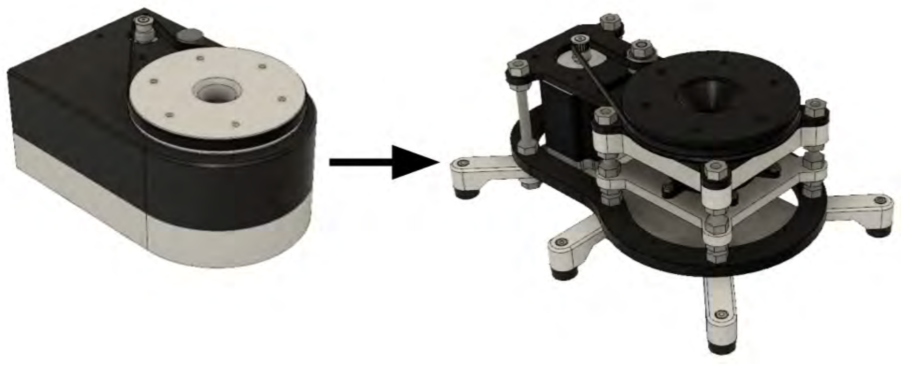


Figure 3.1. Iterations of Axis 1.

A CAD drawing of the final version is presented in Figure 3.2. The final version consists of two main groups of parts, the frame (blue pointers in the figure) and the drive train (red pointers). A more throughout description of Axis 1 follows down below.

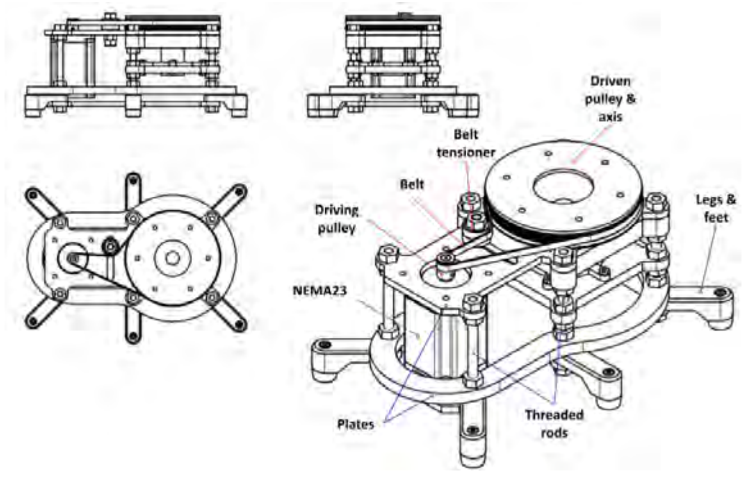


Figure 3.2. CAD drawing of the first axis.

The frame of the first axis consists of several 3D printed plates printed horizontally to maximize strength. These are secured to each other with six 8 mm threaded rods and nuts in order to ensure rigidity of the construction. An additional advantage of using plates and threaded rods is that it reduces the need for precise 3D printed parts. By tightening the nuts at different heights, the different parts of the frame can be adjusted as needed. That solution also reduces the material cost and the printing time, making it cheaper and easier to tweak the design if needed. Six legs pointing outwards are attached to the threaded rods to make the robot stable on the surface it stands on. Rubber feet are attached on the bottom of the legs to increase the friction between the robot arm and the surface and to dampen the vibrations.

The drive train of the first axis consists of a NEMA 23 stepper motor mounted on the top plate of the frame and an axis driven by that stepper motor. These two are connected to each other via a 500 mm belt. The motor drives the belt through a small geared pulley with a diameter of 12 mm. The belt drives a bigger 120 mm pulley that the driven axis is integrated into. The size difference of the pulleys results in a gear ratio of 1/10. A simple belt tensioner consisting of two bearings, nuts and a screw, is mounted on the top plate of the frame and can be adjusted to ensure the optimal belt tension. In order to secure the driven axis to the frame, a series of bearings is used as shown as a cross section view in Figure 3.3. The topmost bearing, located between the bigger pulley and the top plate is a thrust bearing that takes the weight of the remaining robot arm parts. The two other bearings are standard deep-groove ball bearings that support the axis and increase the rigidity of the arm. Wires coming from the rest of the robot arm are placed inside the driven axis, reducing the risk of wire entanglement.

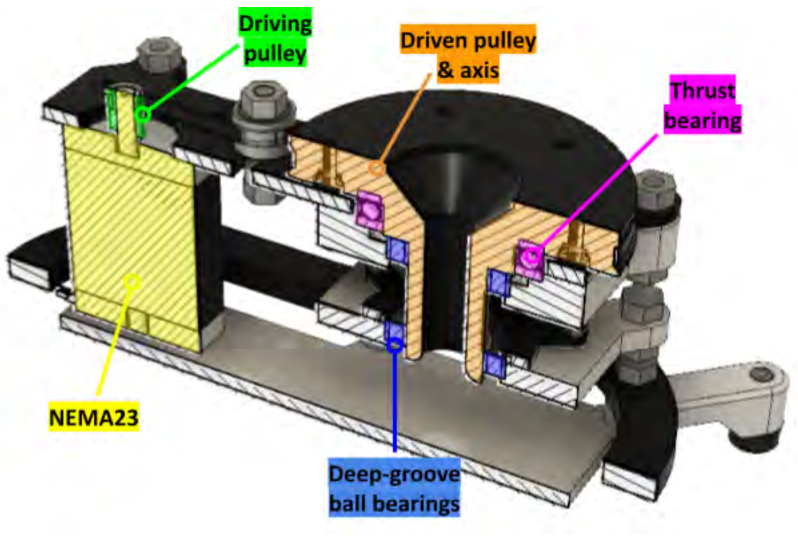


Figure 3.3. Cross section view of the first axis.

Axis 2

The second axis is mounted on the driven pulley of the first axis. The most important aspect of this axis is that it needs to move the remaining parts of the arm overcoming the gravity (unlike the first axis that moves the arm in the horizontal plane where gravity doesn't matter that much). A strong actuator or transmission and a proper belt tension is therefore required. Two different iterations were constructed as presented in Figure 3.4. Initially, one stage reduction was used. It was good enough for smaller loads but it could not manage the weight of the Axis 3 and 4 because of the belt slippage. A two stage reduction was implemented instead, which gave much better results. The final version of the second axis is presented in Figure 3.5.

The final construction of the second axis consists of a frame and a drive train. The frame is made up of several 3D printed parts so the printing orientation can be optimized for maximum strength. The parts are attached to each other using screws and nuts. The frame can be further divided into two main parts, the static one and the rotating one (relative to the pulley of the first axis). The static one is where the drive train is mounted. It also houses two big bearings that the rotating part is attached to. The rotating part consists of the main pulley (Pulley 4 in the Figure 3.5) and a mirrored copy of that pulley on the other side. The pulleys have integrated axes that are placed inside the bearings, and some support material to add rigidity to the construction. The axes have a cavity where stepper motors used in the third and fourth axes are mounted. The main pulleys are connected to each

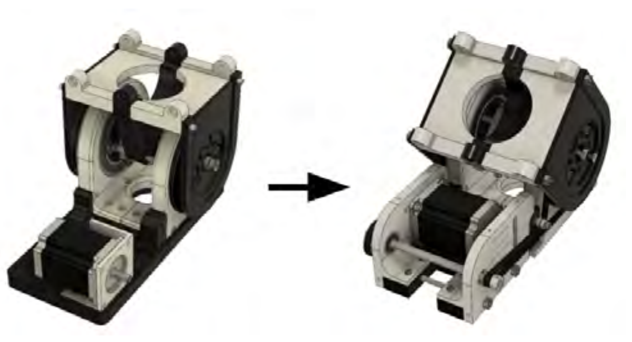


Figure 3.4. Iterations of the Axis 2.

other using threaded rods and 3D printed support parts that also provide mounting locations for the rest of the robot arm.

The drive train of the second axis consists of a NEMA 23 stepper motor attached horizontally to the frame. It drives, through a 12 mm geared pulley (Pulley 1), a 200 mm belt which in turn drives a bigger 36 mm pulley (Pulley 2) giving a reduction ratio of 1/3. The bigger pulley is mounted on a 8 mm steel axis that is secured to the frame using two deep-groove ball bearings. On the other side of the axis, a 12 mm pulley (Pulley 3) is mounted. It drives, though a 640 mm belt, the earlier mentioned 120 mm pulley (Pulley 4). The two stage reduction gives a gear ratio of 1/30 which is needed to lift the remaining parts of the robot arm. To transfer that much torque, the belts needs to be properly tensioned. The smaller 200 mm belt is tensioned by moving the stepper motor, and the longer belt is tensioned using an idler mounted to the frame. It is constructed using a screw, nuts and bearings. By placing it close to the Pulley 4, the contact area between the pulley and the belt is increased, reducing the risk of slippage.

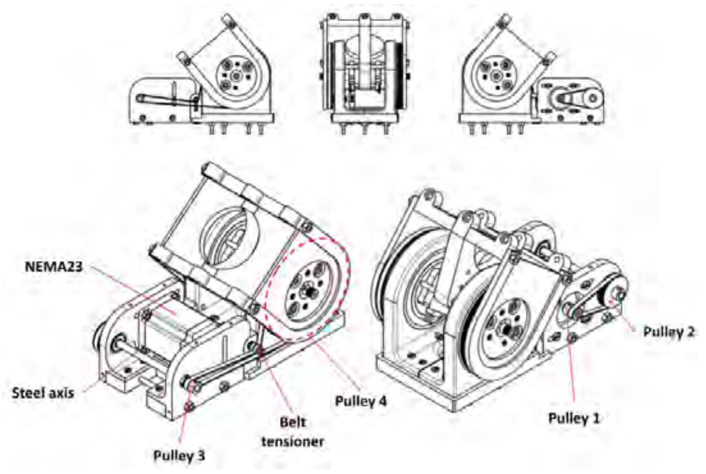


Figure 3.5. CAD drawing of the second axis.

Axis 3 and 4

The third and fourth axis is attached to the second axis. The main concern with this part of the arm was the balance between weight and stiffness. Two different iterations were constructed as shown in Figure 3.6. The only change is the material removed to reduce the weight. The final version is presented in Figure 3.7 as a CAD drawing. A description of the final version follows down below together with a cross section view of the differential transmission in Figure 3.8.

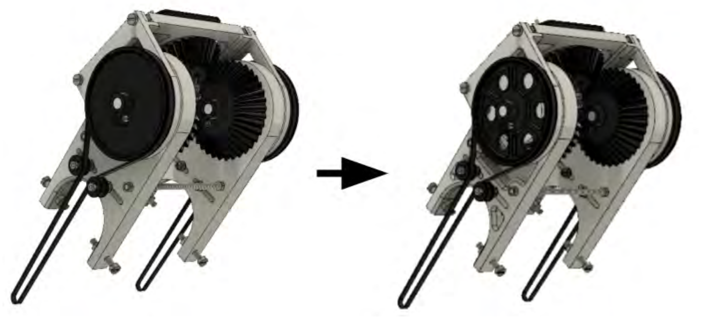


Figure 3.6. Iterations of the Axis 3 and 4.

The third and fourth axes are both driven with the same NEMA 17 stepper motors combined. This is accomplished by implementing a differential consisting of three bevel gears. Two of these gears are connected with 10 mm axes to 96 mm geared pulleys which in turn, through a 640 mm belt, are connected to the 9.6 mm pulleys mounted on the NEMA 17 motors. That gives a reduction ratio of 1/10. The steppers are attached inside of the axes of the Axis 2, as mentioned earlier. It is advantageous to place the steppers so far down on the robot arm because the force caused by the weight of the steppers can then be neglected (because the line of action between the force and the second axis equals zero). The plate that the rest of the arm (Axis 5 & 6) is attached to is connected through a 3D printed axis to the third bevel gear. Idlers are implemented to ensure proper belt tension. They are constructed in the same way as the idlers used on the other axes, using screws, nuts and bearings.

The frame of the third and fourth axis is attached to the second axis using screws and nuts. It consists of two main plates that the rest of the components are mounted onto. These plates are designed in such a way that reduces the weight of the axes and are 3D printed horizontally for optimal strength. Rigidity is increased by connecting the plates with two threaded rods that are also used to adjust the tightness of the bevel gears. It is important because 3D printed gears are not as precise as machined gears therefore giving room for too much backlash (bevel gears too loose) or too much friction (bevel gears too tight). The rotating part that houses the third bevel has integrated, hollow axes placed inside bearings mounted into the main plates. These axes are hollow to give room for the 10 mm axes that the bevel gears and pulleys are connected with. Placing two small bearings inside the hollow axes enables rotation of the 10 mm axes.

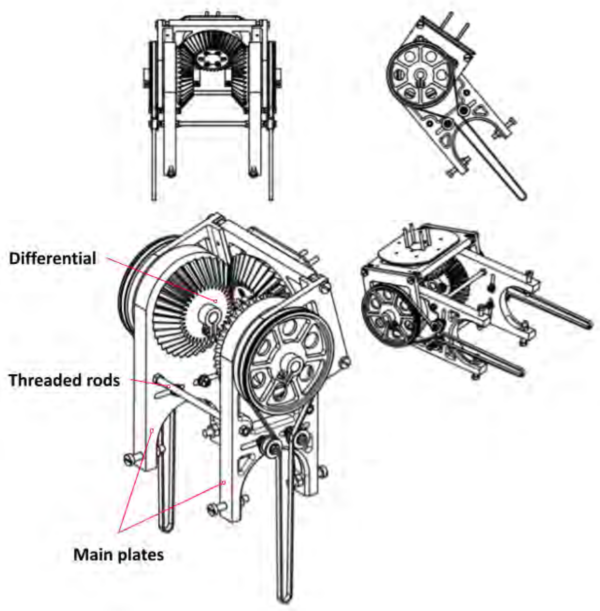


Figure 3.7. CAD drawing of the axis 3 and 4.

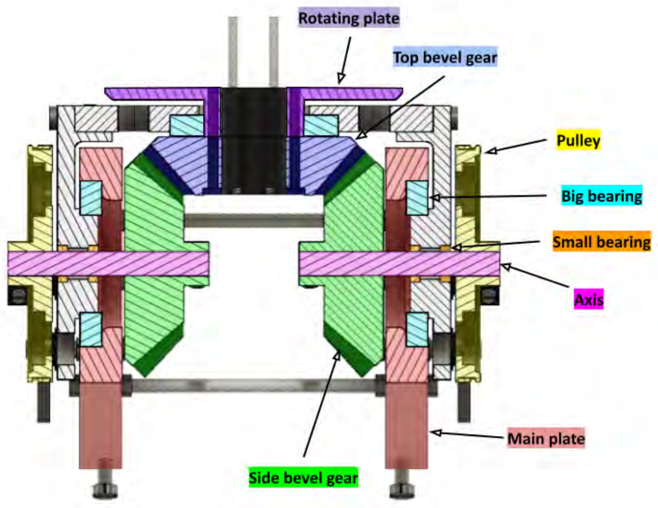


Figure 3.8. Cross section view of the axis 3 and 4.

Axis 5,6 and the end-effector

The last two axes are mounted onto the rotating plate of the third and fourth axis. A CAD drawing of the axes is shown in Figure 3.9. This part of the robot arm is constructed in the same way as Axis 3 and 4. The only difference is that it is shorter and the stepper motors are mounted on the main plates. The main plates are much thinner than the main plates of the previous axes. The gear ratio is now 1.2/10 because of the smaller pulley and the belt is tensioned from only one side.

The last part is the end effector and it can be chosen arbitrarily. In this case, a simple aluminium gripper driven by a servo motor was chosen.

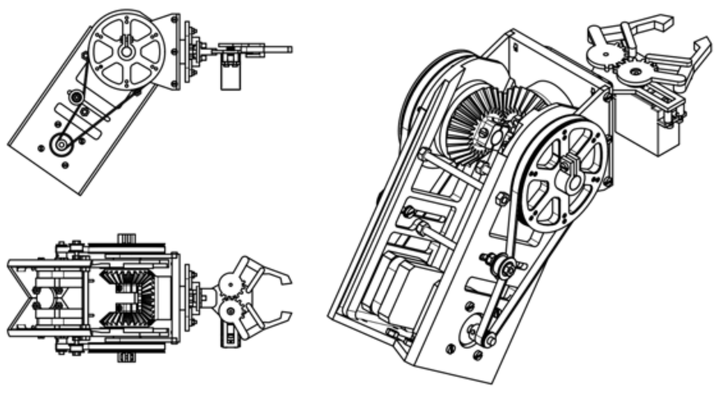


Figure 3.9. A CAD drawing of the fifth and sixth axes.

3.2 Actuators

Robot arm constructed in this project uses six stepper motors in total and a sevo motor for the gripper. Datasheets for the actuators are presented in Appendix D.

The first axis uses a 57HS76-3004B08-A2323 NEMA 23 stepper motor. It has a holding torque of 1.5 Nm. With a gear ratio of 1/10, the output shaft has therefore an ideal holding torque of 15 Nm. It is driven by a TB6600 stepper motor driver using a microstep setting of 1/16. The stepper motor has a standard step angle of 1.8°giving 3200 steps per revolution.

For the second axis, a 23H2A8425 NEMA 23 stepper motor is used. It has a holding torque of 1.8 Nm and step angle of 1.8°. With a gear ratio of 1/30, the output shaft has an ideal holding torque of 54 Nm. It is driven with a TB6600 driver with a microstep resolution of 1/16, it can therefore turn 3200 steps per one revolution.

The third and fourth axis uses two 42BYGHM809 NEMA 17 stepper motors. It has a holding torque of 0.48 Nm and a step angle of 0.9°. With a reduction ratio of 1/10, every pulley has an ideal holding torque of 4.8 Nm. The power is combined as mentioned earlier, giving a total holding torque of 9.6 Nm.

The last two axes uses two smaller NEMA 17 stepper motors that produce 0.2 Nm of holding torque. A smaller size of NEMA 17 was chosen to reduce the force caused by their weight. With a gear ratio of 1.2/10, it gives a ideal holding torque of around 1.67 Nm. The torque is doubled using a differential, giving a total ideal holding torque of 3.33 Nm.

The NEMA 17 stepper motors are driven by A4988 stepper drivers with a microstep setting of 1/16. With a step angle of 0.9°and 1.8°, it gives 3200 and 6400 steps per one revolution.

To control the gripper, a MG996R metal gear servo is used. It has a stalling torque of 10 kg and a rotating range of 120°which is enough for the gripper used.

3.3 Control

Electronics

The robot arm is controlled using a RAMPS 1.4 shield mounted on top of an Arduino Mega 2560. Four A4899 stepper motor drivers are mounted onto the RAMPS to control the NEMA 17 stepper motors. TB6600 stepper drivers controlling the NEMA 23 steppers are connected using the digital pins. Power is supplied to the RAMPS using a 12 V, 10 A power supply, which is just enough to drive all the motors. The RAMPS cannot supply enough current from the regular pins to drive the servo motor. Power output ports on the RAMPS are used instead. However, the voltage from the ports is too high for the servo motors. To solve that problem, a LM2596 power converter was connected between the port and the motor. A LED lamp and a buzzer were added to act as indicators of different states of the arm. The schematic of the electronics is shown in Figure 3.6. Note that the RAMPS shield is not shown on the schematic.

Software

Arduino is connected with an USB-cable to a computer. User can control the robot arm by sending commands through serial communication. A simple user interface was created to simplify that procedure. The serial commands are then translated in the arduino which in turn forwards the commands to the stepper drivers.

User interface The user interface was created using a Python package called PySimpleGUI. It is a package that simplifies user interface creation[20]. A screenshot of the interface is shown in Figure 3.11. The topmost bar consists of buttons used for different actions. The ”Connect” button establishes the serial connection to the Arduino. The ”Send command” button sends the data that is visible in the lower part of the window (”Command” frame). The ”Save command” and ”Load command” buttons saves/loads that data, and the ”Settings” button opens a settings menu. The command is created using the ”Axis” and ”Gripper” frames. Every axis has its own frame that contains a slider, five buttons and information about

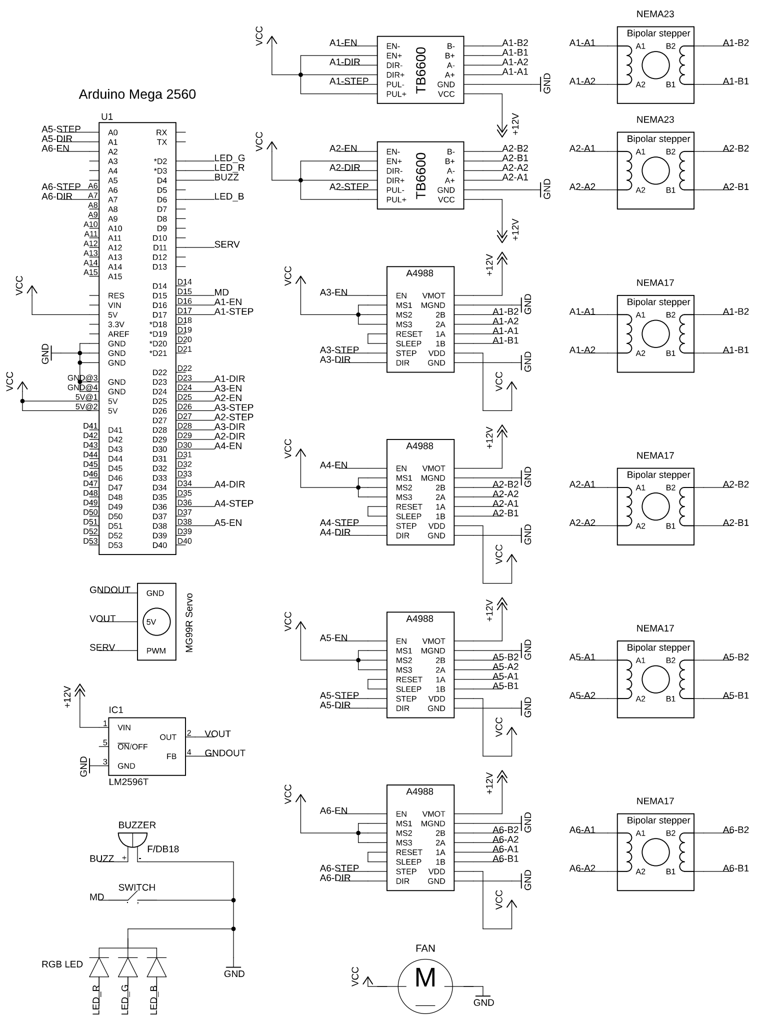
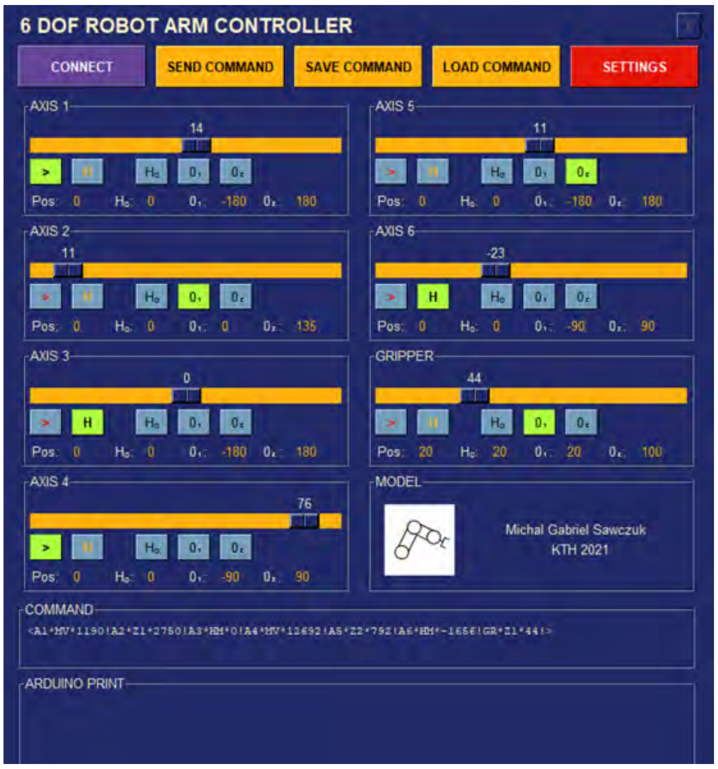


Figure 3.10. Circuit diagram of the robot arm electronics.

the axis. The arrow button creates a ”Move” command that moves the axis to the angle selected through the slider. The ”H” button moves the axis to a preselected, ”home” position. The other three switches are used to set the value of the ”home”, and the limit positions. The undermost frame, ”Arduino print” shows the data sent from the Arduino to the computer. Lastly, the ”Model” frame shows a simplified model of the arm that shows the current axis when hovering the mouse pointer over corresponding axis frame. The Python code for the user interface can be found in Appendix E.

The command sent by the interface has to be constructed in a certain way so



AXIS 1: 14 — Pos: 0 H_0: 0 0_1: -180 0_2: 180

AXIS 2: 11 — Pos: 0 H_0: 0 0_1: 0 0_2: 135

AXIS 3: 0 — Pos: 0 H_0: 0 0_1: -180 0_2: 180

AXIS 4: 76 — Pos: 0 H_0: 0 0_1: -90 0_2: 90

AXIS 5: 11 — Pos: 0 H_0: 0 0_1: -180 0_2: 180

AXIS 6: -23 — Pos: 0 H_0: 0 0_1: -90 0_2: 90

GRIPPER: 44 — Pos: 20 H_0: 20 0_1: 20 0_2: 100

COMMAND

```
<A1,MV,1190!A2,Z1,2750!A3,HM,0!A4,MV,12692!A5,Z2,792!A6,HM,-1656!GR,Z1,44!>
```

ARDUINO PRINT

Figure 3.11. Screenshot of the user interface.

the Arduino can interpret it. A valid command should look like this

<TARGET,TASK,VALUE!TARGET,TASK,VALUE ... !>

Where "<", ">" denote respectively a start, end of the data transfer. Between every command an exclamation mark "!" is placed. Each command is further divided into three different parts, target of the command (TARGET), the task to be performed by the target (TASK) and an eventual value of that task (VALUE). Each part is divided using a comma ",". The allowed values of the different parts are presented in Table 3.1.

Arduino The arduino code was written and uploaded with Arduino IDE and can be found in Appendix F. It consists of two main parts. The first part reads and translates the data received by serial connection. The second part uses that data to control the robot arm using stepper motor drivers. AccelStepper library was used to control the stepper motors [21]. It is better than the standard stepper library because it supports acceleration and usage of several stepper motors at the same time. However, it is important to note that acceleration control is not available when using several stepper motors concurrently. User can switch between

Table 3.1. Allowed values of the different command-parts sent to the Arduino.

Part	Allowed value	Explanation
TARGET	A1	Axis 1
	A2	Axis 2
	A3	Axis 3
	A4	Axis 4
	A5	Axis 5
	A6	Axis 6
	GR	Gripper
TASK	MV	Move to VALUE
	HM	Move to home
	H0	Set new home position
	Z1	Set new zero position (1)
	Z2	Set new zero position (2)

the ”Acceleration control” mode, and the ”Move all” mode using a physical button. Standard servo library was used to control the servo motor in the gripper.

Chapter 4

Results

Three different tests were performed in order to answer the research questions. The first two tests concerned the precision of each individual axis and the whole robot arm using two different stepper motor speeds. The third test measured the maximum torque that each drive train produced.

4.1 Precision tests

In order to measure the error, an indicator dial with an accuracy of 0.01mm was used. It was mounted together with the robot arm onto a wooden board to make sure they doesn't move relative to each other during testing. The wooden board was clamped in place to further increase stability. Then, each axis was repeatedly moved to a chosen position ten times and the indicator dial values were recorded each time. After the value for each axis was recorded, the test was repeated for the whole arm (all axes), for two different positions.

The different speeds that were used are presented in Table 4.1 and the measurements are presented in Tables 4.2 and 4.3.

Table 4.1. The different stepper motor speeds used for precision tests.

Axis	Speed A [Steps/s]	Speed B [Steps/s]
1	2000	4000
2	2000	4000
3,4	1000	2000
5,6	1000	2000

Table 4.2. Accumulated error, Speed A [mm].

Test	Axis 1	Axis 2	Axis 3	Axis 4	Axis 5	Axis 6	All 1	All 2
1	0	0	0	0	0	0	0	0
2	0.05	0.02	0.08	-0.01	-0.02	0.05	0.06	-0.75
3	0.08	0.05	0.1	0	0.09	0.21	0.1	-0.57
4	0.1	0.11	0.1	0	0.2	0.35	0.12	-0.5
5	0.07	0.16	0.12	0.02	0.45	0.57	0.13	-0.35
6	0.16	0.21	0.14	0.02	0.6	0.92	0.27	-0.16
7	0.38	0.23	0.15	0.03	0.98	1.22	0.07	-0.15
8	0.43	0.24	0.17	0.04	1.3	1.37	0.13	0.26
9	0.42	0.25	0.17	0.05	1.61	1.75	0.05	0.33
10	0.48	0.27	0.17	0.05	2.03	2.17	-0.02	0.54

Table 4.3. Accumulated error, Speed B [mm].

Test	Axis 1	Axis 2	Axis 3	Axis 4	Axis 5	Axis 6	All 1	All 2
1	0	0	0	0	0	0	0	0
2	-0.07	0.03	0.01	0.12	0.31	0.09	0.4	0.18
3	0.07	0.08	0.01	0.28	0.9	0.39	0.52	0.38
4	0.09	0.1	0.01	0.52	1.63	1.01	0.62	0.53
5	0.11	0.15	0.02	0.72	2.74	1.75	0.69	0.58
6	0.23	0.2	0.04	1.02	4.09	2.91	0.62	0.8
7	0.39	0.24	0.02	1.24	5.39	3.98	0.94	0.99
8	0.77	0.26	0.03	1.42	6.74	4.81	0.96	1.14
9	0.97	0.3	0.03	1.77	7.18	6.85	1.04	1.18
10	1.21	0.32	0.03	2.12	8.41	8.31	1.12	1.32

4.2 Strength tests

In order to measure the strength, each drive train was locked and loaded until it yielded. The measured holding torque and the point of failure for each drive train is presented in Table 4.4.

Table 4.4. Measured holding torque of each drive train.

Axis	torque[Nm]	Point of failure
1	10.6	Belt slippage
2	5.7	Belt slippage
3,4	3.2	Belt slippage
5,6	2.0	Stepper motor yielded

Chapter 5

Discussion

5.1 Precision tests results

The precision tests have shown that the error is fairly low for both speeds, but it increases each time the axes are moved. Error change between each test was calculated and plotted in Figure 5.1 and presented in tables in Appendix G. The solid lines represent Speed A (slower) and the dashed lines represent Speed B (faster). It seems that most of the error is systematic, it changes by approximately the same amount between different tests, though one exception is found. The error change for Axis 5 and 6, when using speed B seem to be completely random. The reason for that could be attributed to the poor design. The construction is not stiff enough and the bevel gears tend to bind with each other. The most unexpected result is that the errors from each individual axis doesn't lead to a bigger error when testing all axes at once. It is unknown why that happens. It could be that the errors from the individual axes cancel each other.

When comparing the speeds, it becomes clear that the slower speed (Speed A) gives less error. However, for unknown reasons, Axis 3 gave less error when Speed B was used. For Axis 2, the speed change didn't make much of a difference, the total accumulated error was only 0.05mm higher when using Speed B.

Because of the systematic nature of the errors, it is not possible to quantify the precision of the robot arm. Instead, the average increase in error when moving all axes once is calculated and presented in Table 5.1. Note that only two different positions were tested (All 1 and All 2) and that the arm was not loaded during the tests. The results are therefore not a good measure of the robot arm precision.

Table 5.1. Average change in error [mm].

Speed	Average change in error
A	0.0289 mm
B	0.1356 mm

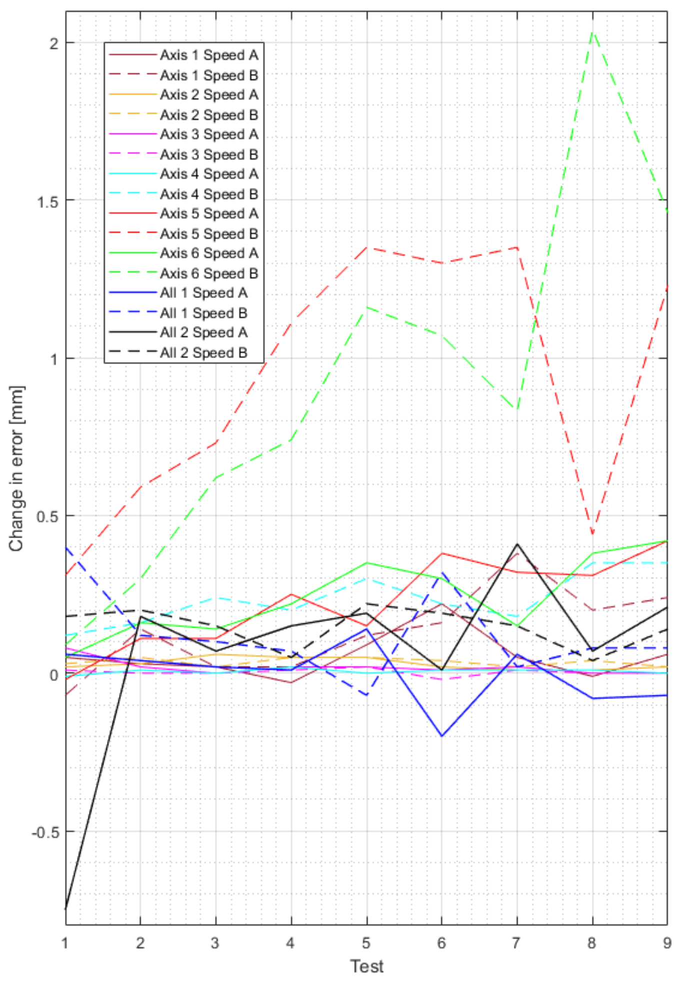


Figure 5.1. Change in error.

5.2 Strength test results

The strength tests have shown that the drive trains are disappointingly inefficient:

- Axis 1 reached 70.7% of the theoretic holding torque (10.6/15 Nm) before the belt skipped.
- Axis 2 reached only 10.6% of the theoretic holding torque (5.7/54 Nm) before the belt skipped.
- Axis 3 and 4 reached 34.3% of the theoretic holding torque (3.2/9.6 Nm) before the belt skipped.
- Axis 5 and 6 reached 60% of the theoretic holding torque (2/3.33 Nm) before the stepper motor yielded.

This means that the maximum weight that the arm can hold is 0.84 kg in the worst case scenario with all axes extended outwards. Note that the dynamic torque is even lower, and was not measured in this project. The belts currently used are often used in hobby CNC machines and 3D printers, they are not designed to transfer that much torque. By redesigning the belt tensioners and changing the belts, a much stronger robot arm could be made without replacing the stepper motors.

5.3 Future work

The current construction have shown that a strong and accurate, 3D printed robot arm should be feasible. In order to improve the current design, following issues needs to be fixed:

- The belts should be replaced and the belt tensioners redesigned in order to reach the stepper motors full potential and make the arm much stronger.
- The frame of Axis 5 and 6 should be made stiffer in order to increase the precision of these axes.
- The parts in contact with the NEMA17 stepper motors should be printed with a more heat resistant material. The frame of Axis 5 and 6 is getting softer after a few minutes of use.
- The systematic errors occurring in every axis should be reduced. It might be enough to add compensation in the software.
- The current software doesn't allow acceleration control when moving the stepper motors simultaneously. It should be changed in order to make the arm more practical.
- Forward and inverse kinematics should be calculated in order to make the gripper's exact position known.

5.4 Conclusions

The goal of designing, constructing and controlling an articulated robot arm with six degrees of freedom was achieved. The arm is able to pick and place different objects without using external sensors. The research questions are answered as follows:

- How precise is the robot arm?
  Because of the large amount of systematic errors, the question could not be answered. Instead, an average increase in error after moving all axes (once) was calculated to be 0.0289-0.1356 mm.

- How does the speed of the arm affect the precision?
  A slower speed leads to better precision for all axes except Axis 3. Testing all axes at once resulted in the same conclusion.

- How much weight can the robot arm lift?
  The maximum amount of weight that the arm can hold is 0.84 kg (in the worst case scenario).

Bibliography

[1] Sparkfun, "*Serial Communication*," Available at https://learn.sparkfun.com/tutorials/serial-communication/all (Accesssed on: 07/04/2021).

[2] Components101, "*A4988 Stepper Motor Driver Module*," Available at https://components101.com/modules/a4988-stepper-motor-driver-module (Accesssed on: 5/02/2021).

[3] Soratec, "*Analog Driver Model TB6600*," Available at https://www.mcielectronics.cl/website_MCI/static/documents/TB6600_data_sheet.pdf (Accesssed on: 5/02/2021).

[4] C. Cameron, *3D Printing*, 1st ed. New York: Penguin Group (USA) Inc., 2015.

[5] MatWeb, "*Overview of materials for Polylactic Acid (PLA) Biopolymer*," Available at http://www.matweb.com/search/DataSheet.aspx?MatGUID=ab96a4c0655c4018a8785ac4031b9278&ckck=1 (Accesssed on: 12/02/2021).

[6] T. Yao, Z. Deng, K. Zhang, and S. Li, "A method to predict the ultimate tensile strength of 3d printing polylactic acid (pla) materials with different printing orientations," *Composites Part B: Engineering*, vol. 163, pp. 393–402, 2019.

[7] C. Lubombo and M. A. Huneault, "Effect of infill patterns on the mechanical performance of lightweight 3d-printed cellular pla parts," *Materials Today Communications*, vol. 17, pp. 214–228, 2018.

[8] H. Dave, S. Rajpurohit, N. Patadiya, S. Dave, K. Sharma, S. Thambad, V. Srinivasn, and K. Sheth, "Compressive strength of pla based scaffolds: Effect of layer height, infill density and print speed," *International Journal of Modern Manufacturing Technologies*, vol. 11, 2019.

[9] E. Britannica, "*Transmission*," Available at https://www.britannica.com/technology/transmission-engineering (Accesssed on: 05/04/2021).

[10] W. K. G. W. P. Kosky, R. Balmer, *Exploring Engineering, An Introduction to Engineering and Design*, 5th ed. United Kingdom: Academic Press, 2020.

[11] E. Britannica, "*Differential gear*," Available at https://www.britannica.com/technology/differential-gear (Accesssed on: 05/04/2021).

[12] W. Taha, "*Acumen*," Available at http://www.acumen-language.org/2016/04/about.html (Accesssed on: 21/05/2021).

[13] B. Earl, "*All About Stepper Motors*," Available at https://cdn-learn.adafruit.com/downloads/pdf/all-about-stepper-motors.pdf (Accesssed on: 04/02/2021).

[14] M. Scarpino, *Motors for Makers: A Guide to Steppers, Servos, and Other Electrical Machines.* United States of America: Pearson Education, 2015.

[15] Sparkfun, "*SERVOS EXPLAINED*," Available at https://www.sparkfun.com/servos (Accesssed on: 05/04/2021).

[16] Arduino, "*ARDUINO MEGA 2560 REV3*," Available at https://store.arduino.cc/arduino-mega-2560-rev3 (Accesssed on: 6/02/2021).

[17] RepRap, "*RAMPS 1.4*," Available at https://reprap.org/wiki/RAMPS_1.4 (Accesssed on: 6/02/2021).

[18] I. O. for Standardization, "*ISO 5725-1:1994(en) Accuracy (trueness and precision) of measurement methods and results*," Available at https://www.iso.org/obp/ui/#iso:std:iso:5725:-1:ed-1:v1:en (Accesssed on: 10/02/2021).

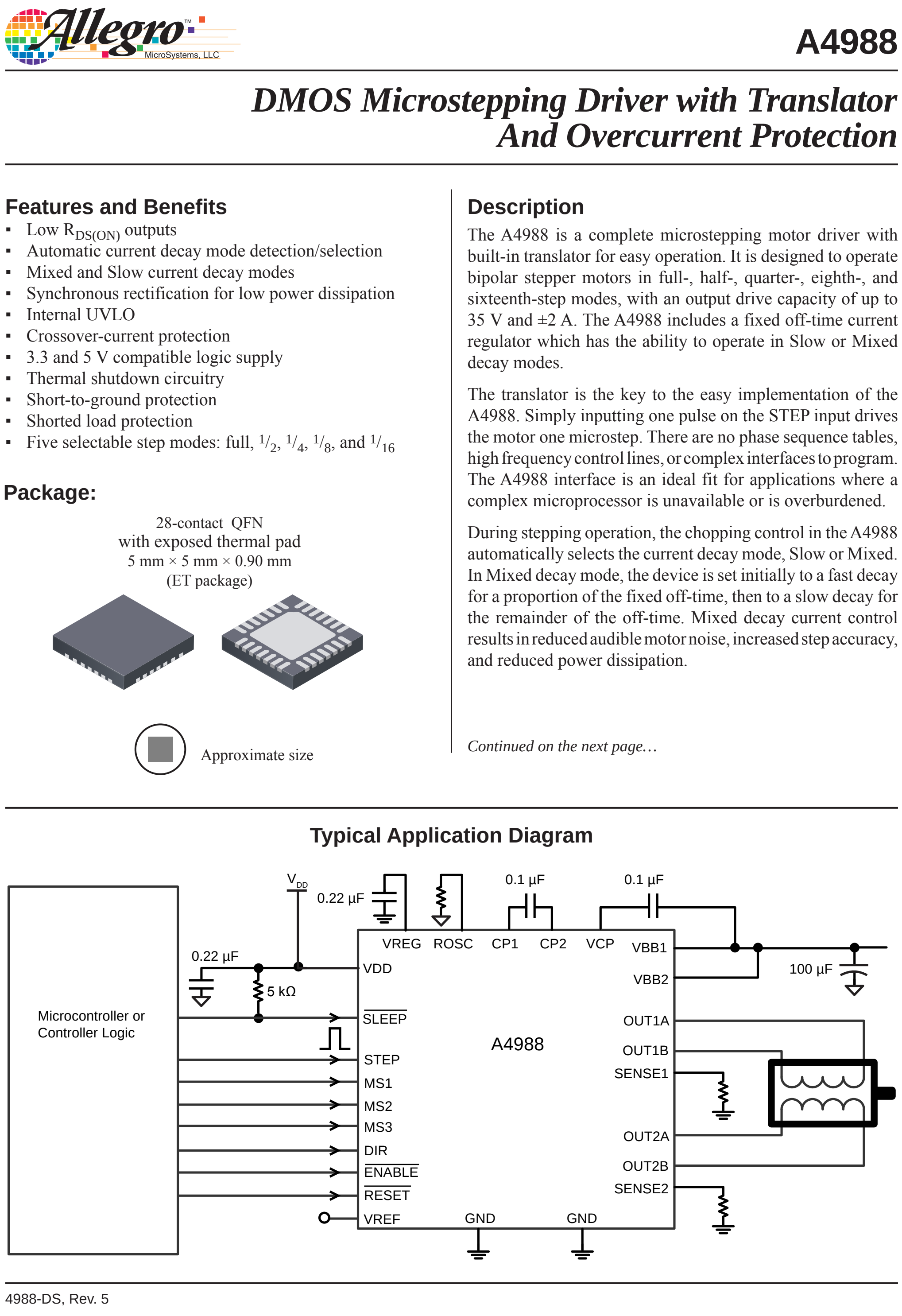
[19] J. Taylor, *Introduction To Error Analysis: The Study of Uncertainties in Physical Measurements.* Sausalito: University Science Books, 1997.

[20] PySimpleGUI, "*PySimpleGUI*," Available at https://pypi.org/project/PySimpleGUI/#:~:text=PySimpleGUI%20is%20a%20Python%20package,%22Elements%22%20in%20PySimpleGUI). (Accesssed on: 11/04/2021).

[21] M. McCauley, "*AccelStepper library for Arduino*," Available at https://www.airspayce.com/mikem/arduino/AccelStepper/ (Accesssed on: 11/04/2021).

Appendix A

Data sheets of the stepper motor drivers.



Allegro MicroSystems, LLC

A4988

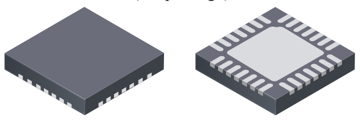
DMOS Microstepping Driver with Translator And Overcurrent Protection

Features and Benefits

- Low $R_{DS(ON)}$ outputs
- Automatic current decay mode detection/selection
- Mixed and Slow current decay modes
- Synchronous rectification for low power dissipation
- Internal UVLO
- Crossover-current protection
- 3.3 and 5 V compatible logic supply
- Thermal shutdown circuitry
- Short-to-ground protection
- Shorted load protection
- Five selectable step modes: full, $^1/_2$, $^1/_4$, $^1/_8$, and $^1/_{16}$

Package:

28-contact QFN
with exposed thermal pad
5 mm × 5 mm × 0.90 mm
(ET package)



Approximate size

Description

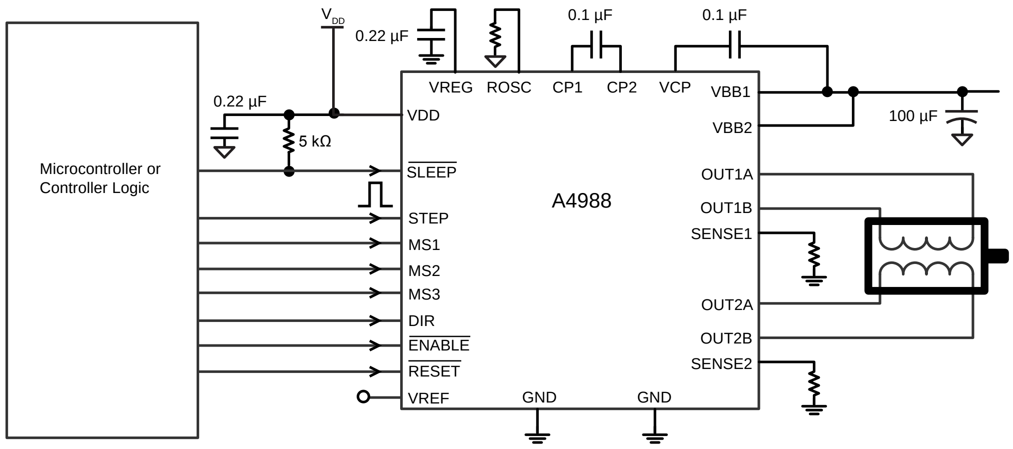
The A4988 is a complete microstepping motor driver with built-in translator for easy operation. It is designed to operate bipolar stepper motors in full-, half-, quarter-, eighth-, and sixteenth-step modes, with an output drive capacity of up to 35 V and ±2 A. The A4988 includes a fixed off-time current regulator which has the ability to operate in Slow or Mixed decay modes.

The translator is the key to the easy implementation of the A4988. Simply inputting one pulse on the STEP input drives the motor one microstep. There are no phase sequence tables, high frequency control lines, or complex interfaces to program. The A4988 interface is an ideal fit for applications where a complex microprocessor is unavailable or is overburdened.

During stepping operation, the chopping control in the A4988 automatically selects the current decay mode, Slow or Mixed. In Mixed decay mode, the device is set initially to a fast decay for a proportion of the fixed off-time, then to a slow decay for the remainder of the off-time. Mixed decay current control results in reduced audible motor noise, increased step accuracy, and reduced power dissipation.

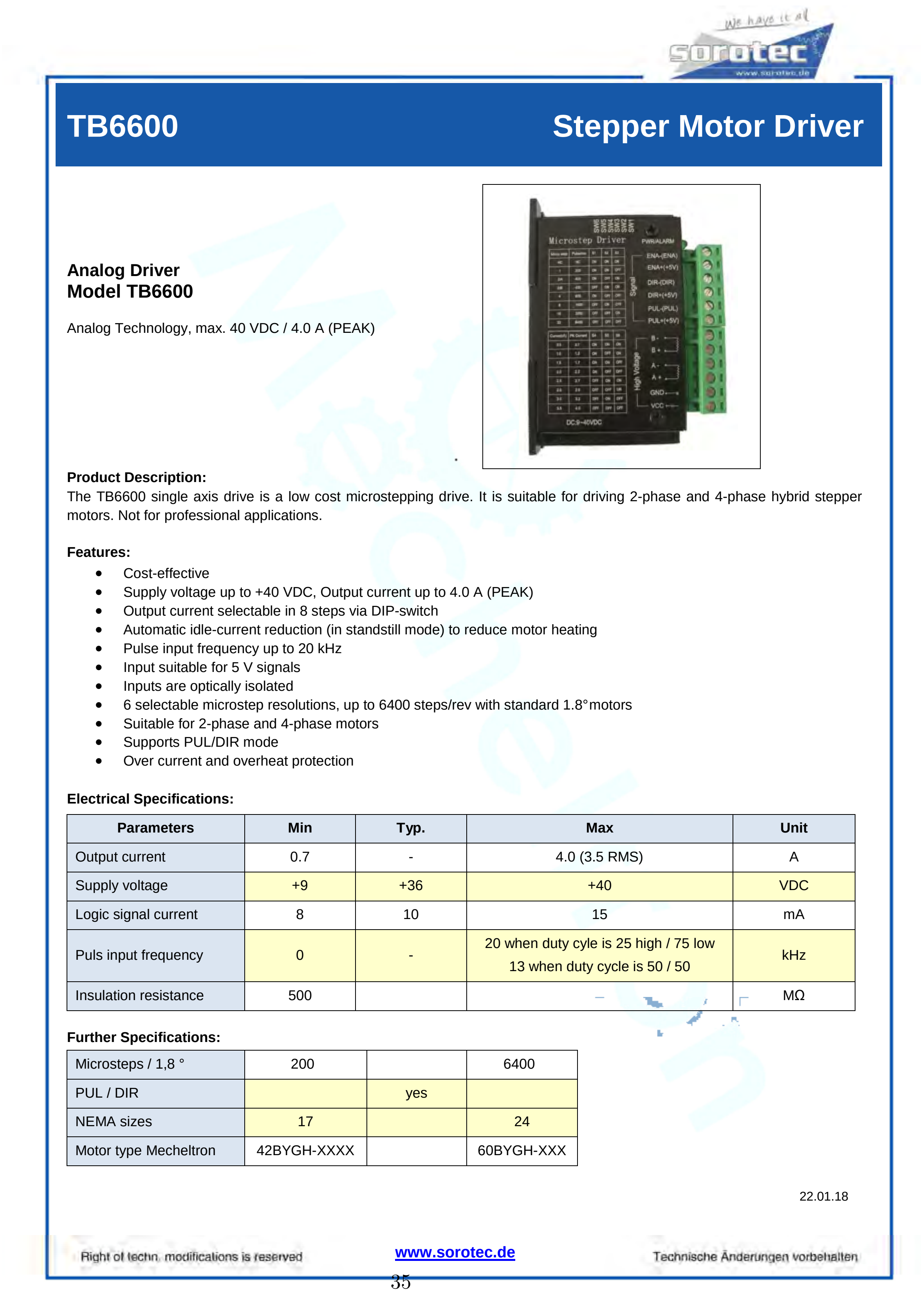
Continued on the next page…

Typical Application Diagram



4988-DS, Rev. 5

Figure A.1. Data sheet of the A4988 stepper motor driver.



TB6600 Stepper Motor Driver

Analog Driver
Model TB6600

Analog Technology, max. 40 VDC / 4.0 A (PEAK)

Product Description:
The TB6600 single axis drive is a low cost microstepping drive. It is suitable for driving 2-phase and 4-phase hybrid stepper motors. Not for professional applications.

Features:

- Cost-effective
- Supply voltage up to +40 VDC, Output current up to 4.0 A (PEAK)
- Output current selectable in 8 steps via DIP-switch
- Automatic idle-current reduction (in standstill mode) to reduce motor heating
- Pulse input frequency up to 20 kHz
- Input suitable for 5 V signals
- Inputs are optically isolated
- 6 selectable microstep resolutions, up to 6400 steps/rev with standard 1.8°motors
- Suitable for 2-phase and 4-phase motors
- Supports PUL/DIR mode
- Over current and overheat protection

Electrical Specifications:

Parameters	Min	Typ.	Max	Unit
Output current	0.7	-	4.0 (3.5 RMS)	A
Supply voltage	+9	+36	+40	VDC
Logic signal current	8	10	15	mA
Puls input frequency	0	-	20 when duty cyle is 25 high / 75 low 13 when duty cycle is 50 / 50	kHz
Insulation resistance	500			MΩ

Further Specifications:

Microsteps / 1,8 °	200		6400
PUL / DIR		yes	
NEMA sizes	17		24
Motor type Mecheltron	42BYGH-XXXX		60BYGH-XXX

22.01.18

Right of techn. modifications is reserved www.sorotec.de Technische Änderungen vorbehalten

Figure A.2. Data sheet of the TB6600 stepper motor driver.

Appendix B

Exploded views of the robot arm

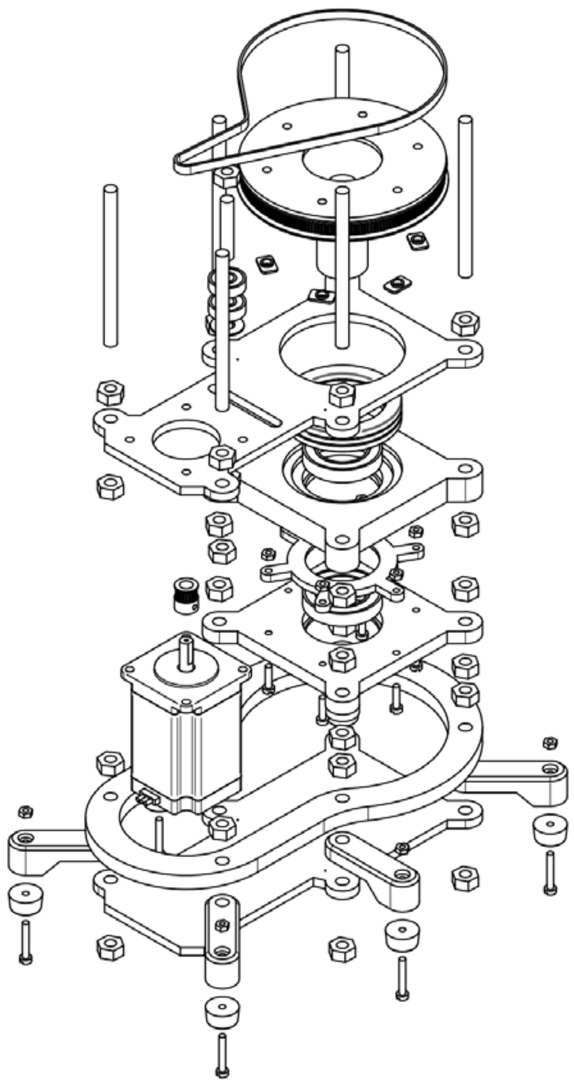


Figure B.1. Exploded view of the first axis.

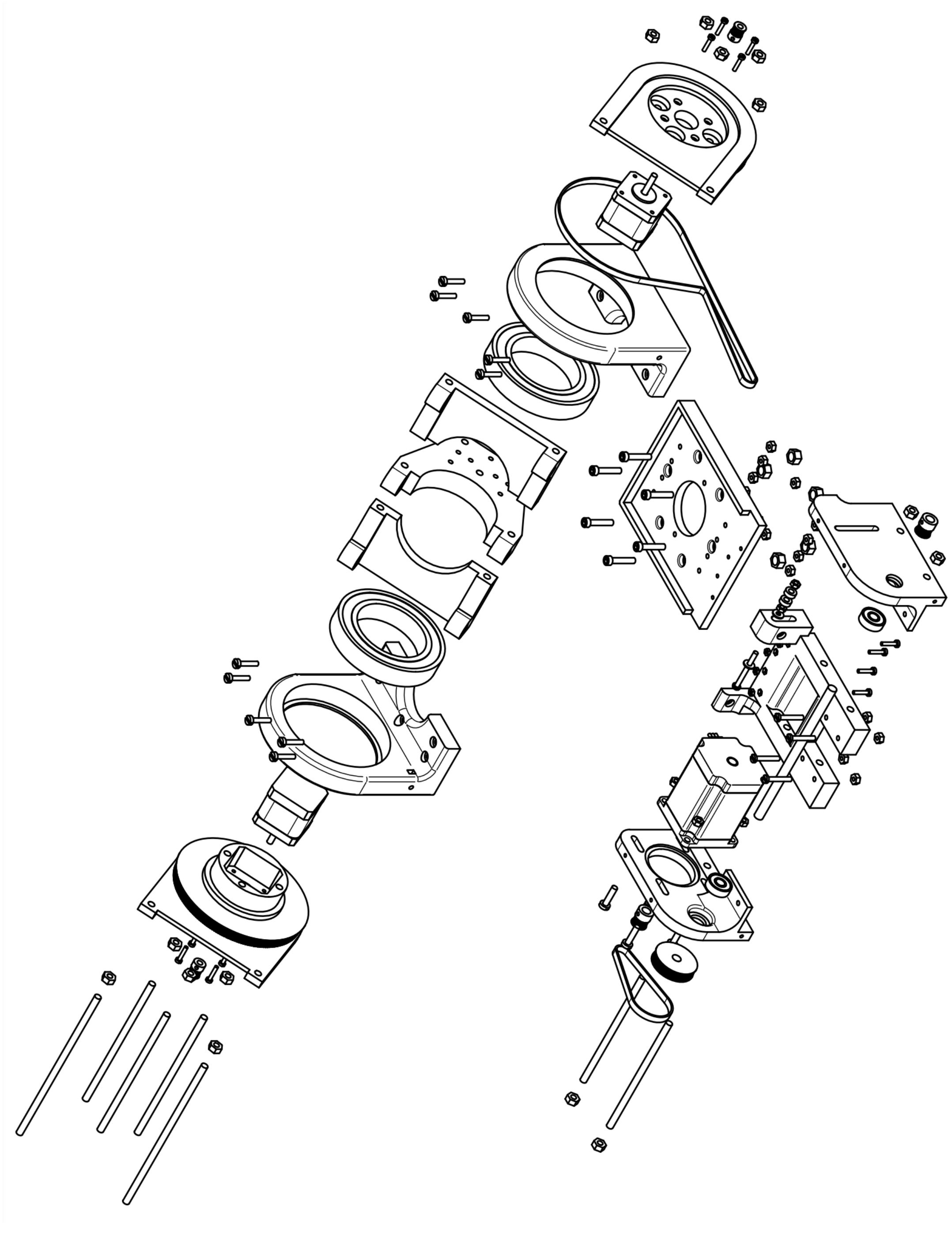


Figure B.2. Exploded view of the second axis.

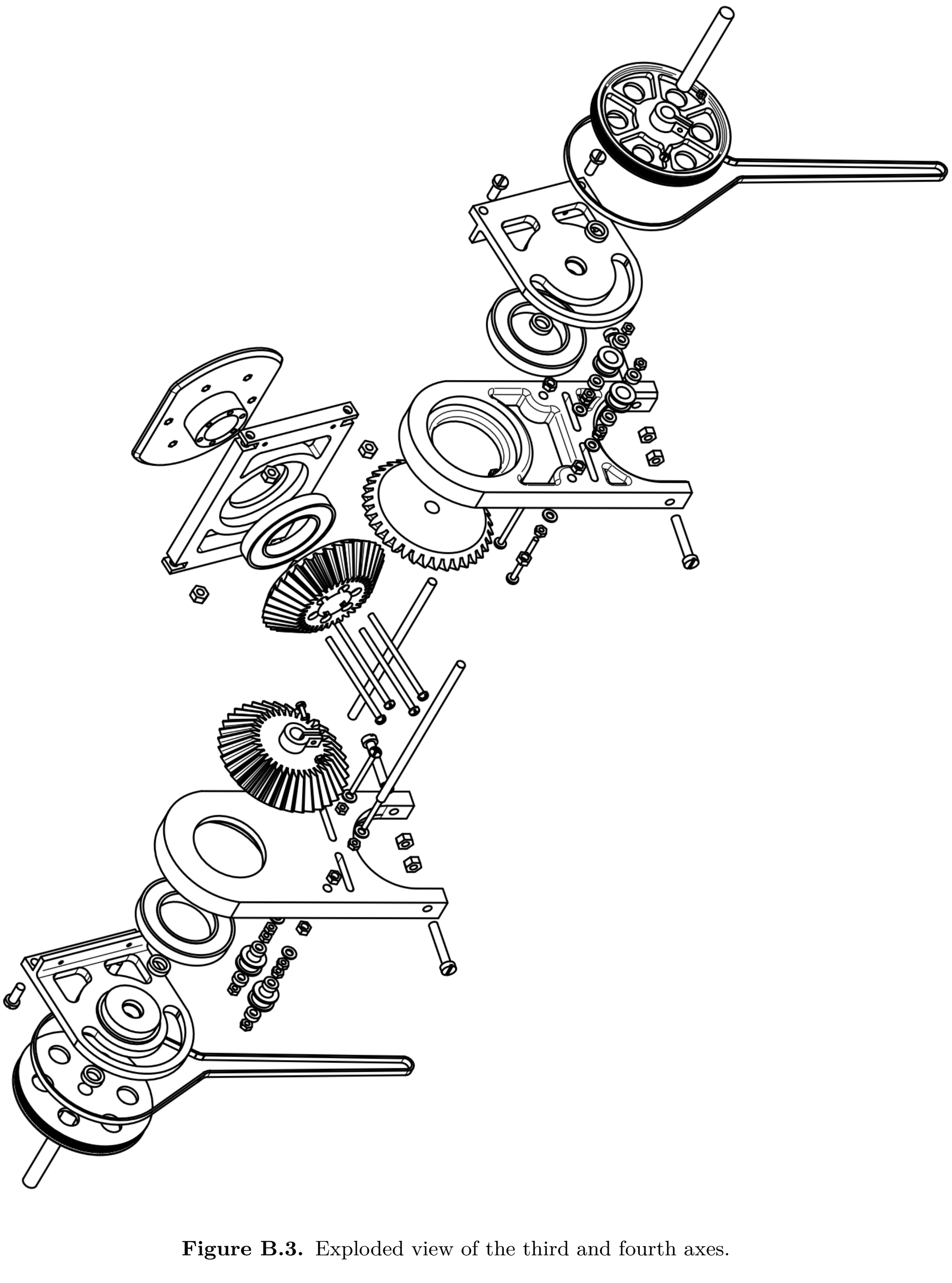


Figure B.3. Exploded view of the third and fourth axes.

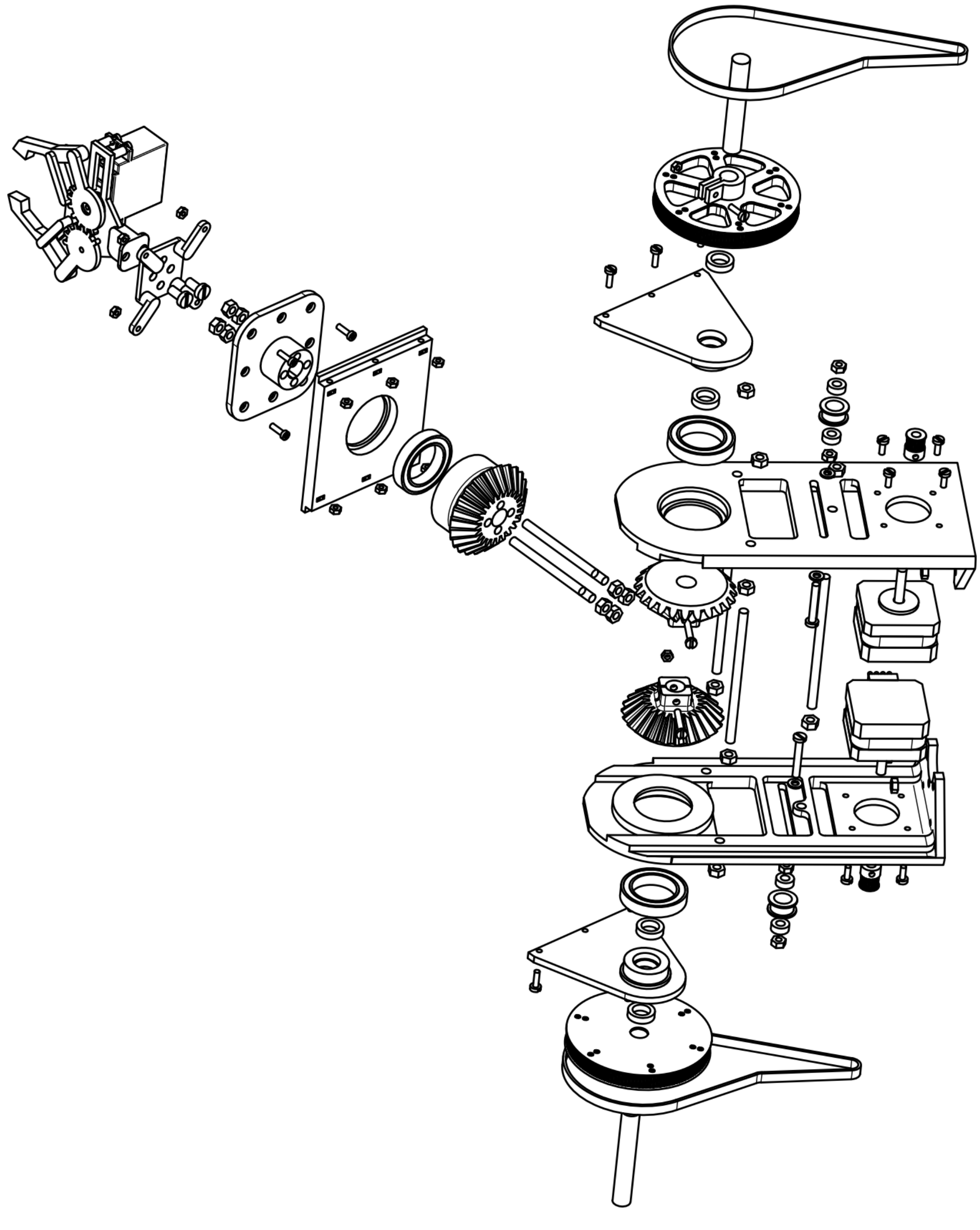


Figure B.4. Exploded view of the fifth and sixth axes.

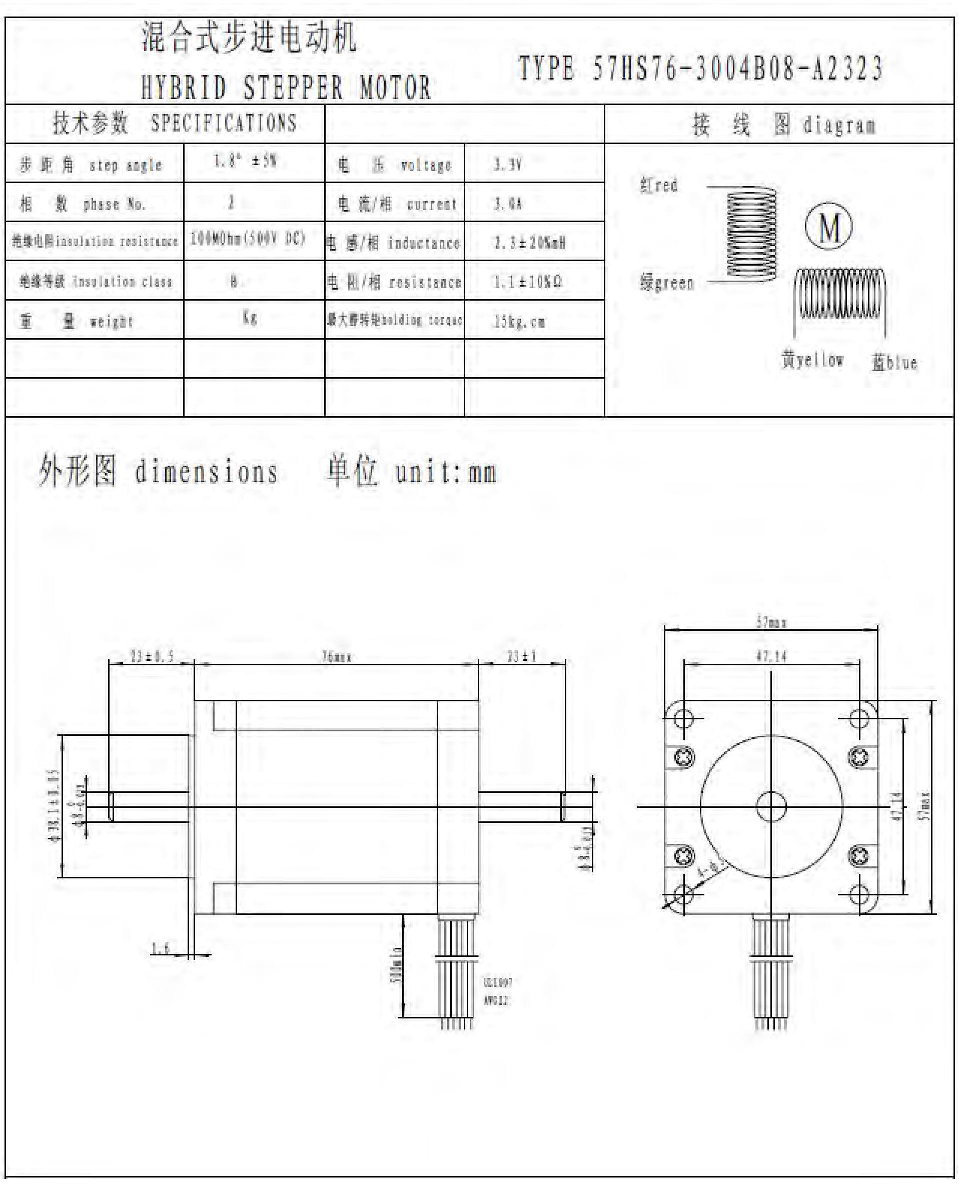
Appendix C

Acumen

```
//Michal Gabriel Sawczuk 28/03/2021
//KTH MF133X

model Main(simulator) =
initially
a = 0, a' = 0, a'' = 0, arate = 0.5, //Axis 1 angle
b = 0, b' = 0, b'' = 0, brate = 0.7, //Axis 2 angle
la = 2, lb = 1.5, //Lengths
amassa = 1, bmassa = 0.6, //Mass
Fga = 0, Fgb = 0, //Gravity
Faa = 0, Fab = 0, //Acceleration forces
Ma = 0, Mb = 0, //Torque
p = (0,0,0), //Center position axis 1 (link)
p2 = (0,0,0), //Center position axis 2
p3 = (0,0,0), //Center position axis 2 (link)
_3D = ()
always
a'' = arate, b'' = brate, //acceleration
Fga = -9.81*amassa, Fgb = -9.81*bmassa,
Faa = -a''*0.5*la*amassa, Fab = -b''*0.5*lb*bmassa,
Mb = (Fgb*0.5*lb*cos(b) + Fab)*lb*0.5,
Ma = (Fga*0.5*la*cos(a) + Faa)*la*0.5+Mb,

p = (0.5*la*sin(a),0,0.5*la*cos(a)), //Kinematics
p2 = (2*sin(a),0,2*cos(a)),
p3 = (la*sin(a)+0.5*lb*sin(b),0,la*cos(a)+0.5*lb*cos(b)),

_3D = (Cylinder center = (0,0,0) rotation = (0,a,0) size = (1,0.3) color = green
    //Movement
        Box center = p rotation = (0,a,0) size=(0.5,0.5,la)
        Box center = p3 rotation = (0,b,0) size=(0.5,0.5,lb)
        Cylinder center = p2 rotation = (0,b,0) size = (1,0.3) color = green
),
 if (a>pi/4) then //reverse movement
arate+ = -0.5,
brate+ = -0.7
noelse,
 if (a<0) then
arate+ = 0.5,
brate+ = 0.7
noelse
```

Appendix D

Data sheets of the actuators



混合式步进电动机
HYBRID STEPPER MOTOR

TYPE 57HS76-3004B08-A2323

技术参数 SPECIFICATIONS

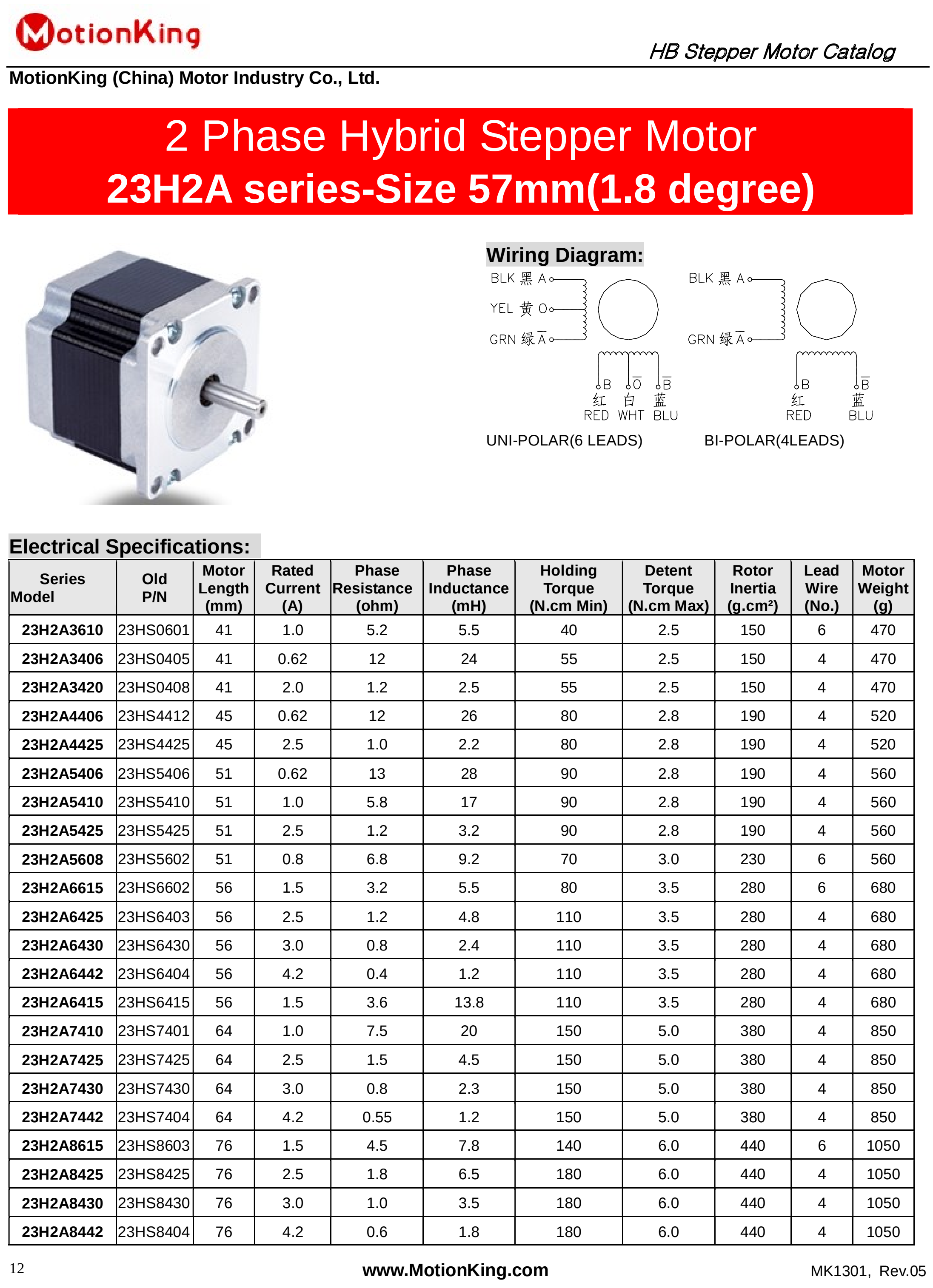
步 距 角 step angle	1.8° ±5%	电 压 voltage	3.3V
相 数 phase No.	2	电 流/相 current	3.0A
绝缘电阻insulation resistance	100MOhm(500V DC)	电 感/相 inductance	2.3±20%mH
绝缘等级 insulation class	B	电 阻/相 resistance	1.1±10%Ω
重 量 weight	Kg	最大静转矩holding torque	15kg.cm

接 线 图 diagram

外形图 dimensions 单位 unit:mm

红red
绿green
M
黄yellow
蓝blue
23±0.5
76max
23±1
Φ38.1±0.05
Φ8-0.013
Φ8-0.013
1.6
500min
UL1007
AWG22
57max
47.14
47.14
57max
4-Φ5

Figure D.1. Data sheet for the stepper motor used in the first axis.



MotionKing
BLK 黑 A
YEL 黄 O
GRN 绿 $\overline{A}$
B
$\overline{O}$
$\overline{B}$
红 RED
白 WHT
蓝 BLU
UNI-POLAR(6 LEADS)
BLK 黑 A
GRN 绿 $\overline{A}$
B
$\overline{B}$
红 RED
蓝 BLU
BI-POLAR(4LEADS)

HB Stepper Motor Catalog

MotionKing (China) Motor Industry Co., Ltd.

2 Phase Hybrid Stepper Motor

23H2A series-Size 57mm(1.8 degree)

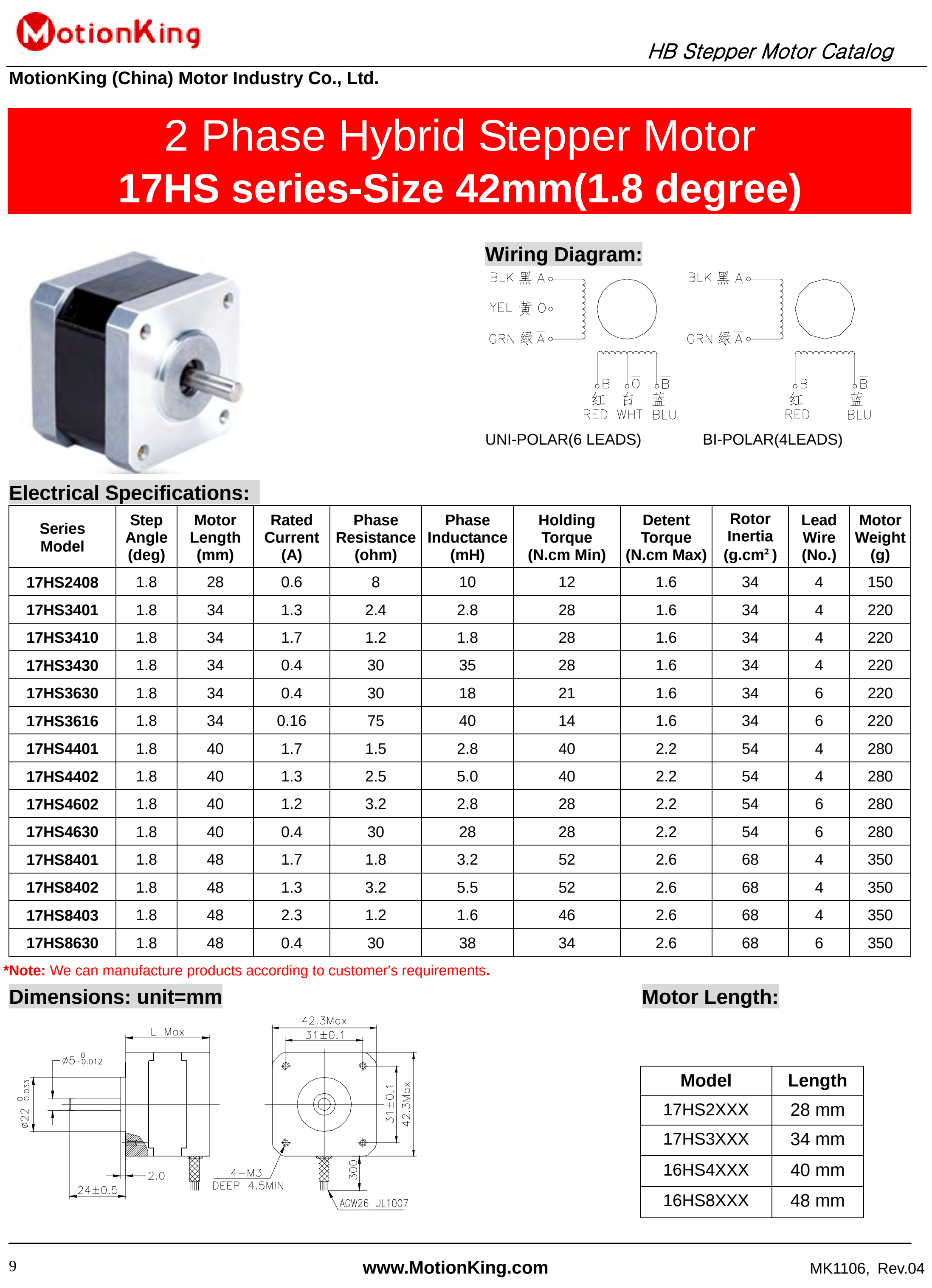
Wiring Diagram:

Electrical Specifications:

Series Model	Old P/N	Motor Length (mm)	Rated Current (A)	Phase Resistance (ohm)	Phase Inductance (mH)	Holding Torque (N.cm Min)	Detent Torque (N.cm Max)	Rotor Inertia (g.cm²)	Lead Wire (No.)	Motor Weight (g)
23H2A3610	23HS0601	41	1.0	5.2	5.5	40	2.5	150	6	470
23H2A3406	23HS0405	41	0.62	12	24	55	2.5	150	4	470
23H2A3420	23HS0408	41	2.0	1.2	2.5	55	2.5	150	4	470
23H2A4406	23HS4412	45	0.62	12	26	80	2.8	190	4	520
23H2A4425	23HS4425	45	2.5	1.0	2.2	80	2.8	190	4	520
23H2A5406	23HS5406	51	0.62	13	28	90	2.8	190	4	560
23H2A5410	23HS5410	51	1.0	5.8	17	90	2.8	190	4	560
23H2A5425	23HS5425	51	2.5	1.2	3.2	90	2.8	190	4	560
23H2A5608	23HS5602	51	0.8	6.8	9.2	70	3.0	230	6	560
23H2A6615	23HS6602	56	1.5	3.2	5.5	80	3.5	280	6	680
23H2A6425	23HS6403	56	2.5	1.2	4.8	110	3.5	280	4	680
23H2A6430	23HS6430	56	3.0	0.8	2.4	110	3.5	280	4	680
23H2A6442	23HS6404	56	4.2	0.4	1.2	110	3.5	280	4	680
23H2A6415	23HS6415	56	1.5	3.6	13.8	110	3.5	280	4	680
23H2A7410	23HS7401	64	1.0	7.5	20	150	5.0	380	4	850
23H2A7425	23HS7425	64	2.5	1.5	4.5	150	5.0	380	4	850
23H2A7430	23HS7430	64	3.0	0.8	2.3	150	5.0	380	4	850
23H2A7442	23HS7404	64	4.2	0.55	1.2	150	5.0	380	4	850
23H2A8615	23HS8603	76	1.5	4.5	7.8	140	6.0	440	6	1050
23H2A8425	23HS8425	76	2.5	1.8	6.5	180	6.0	440	4	1050
23H2A8430	23HS8430	76	3.0	1.0	3.5	180	6.0	440	4	1050
23H2A8442	23HS8404	76	4.2	0.6	1.8	180	6.0	440	4	1050

12 **www.MotionKing.com** MK1301, Rev.05

Figure D.2. Data sheet for the stepper motor used in the second axis.



MotionKing

HB Stepper Motor Catalog

MotionKing (China) Motor Industry Co., Ltd.

2 Phase Hybrid Stepper Motor

17HS series-Size 42mm(1.8 degree)

Wiring Diagram:

BLK 黑 A
YEL 黄 O
GRN 绿 $\bar{A}$
B 红 RED
O 白 WHT
$\bar{B}$ 蓝 BLU
UNI-POLAR(6 LEADS)
BLK 黑 A
GRN 绿 $\bar{A}$
B 红 RED
$\bar{B}$ 蓝 BLU
BI-POLAR(4LEADS)

Electrical Specifications:

Series Model	Step Angle (deg)	Motor Length (mm)	Rated Current (A)	Phase Resistance (ohm)	Phase Inductance (mH)	Holding Torque (N.cm Min)	Detent Torque (N.cm Max)	Rotor Inertia (g.cm^2)	Lead Wire (No.)	Motor Weight (g)
17HS2408	1.8	28	0.6	8	10	12	1.6	34	4	150
17HS3401	1.8	34	1.3	2.4	2.8	28	1.6	34	4	220
17HS3410	1.8	34	1.7	1.2	1.8	28	1.6	34	4	220
17HS3430	1.8	34	0.4	30	35	28	1.6	34	4	220
17HS3630	1.8	34	0.4	30	18	21	1.6	34	6	220
17HS3616	1.8	34	0.16	75	40	14	1.6	34	6	220
17HS4401	1.8	40	1.7	1.5	2.8	40	2.2	54	4	280
17HS4402	1.8	40	1.3	2.5	5.0	40	2.2	54	4	280
17HS4602	1.8	40	1.2	3.2	2.8	28	2.2	54	6	280
17HS4630	1.8	40	0.4	30	28	28	2.2	54	6	280
17HS8401	1.8	48	1.7	1.8	3.2	52	2.6	68	4	350
17HS8402	1.8	48	1.3	3.2	5.5	52	2.6	68	4	350
17HS8403	1.8	48	2.3	1.2	1.6	46	2.6	68	4	350
17HS8630	1.8	48	0.4	30	38	34	2.6	68	6	350

***Note:** We can manufacture products according to customer's requirements.

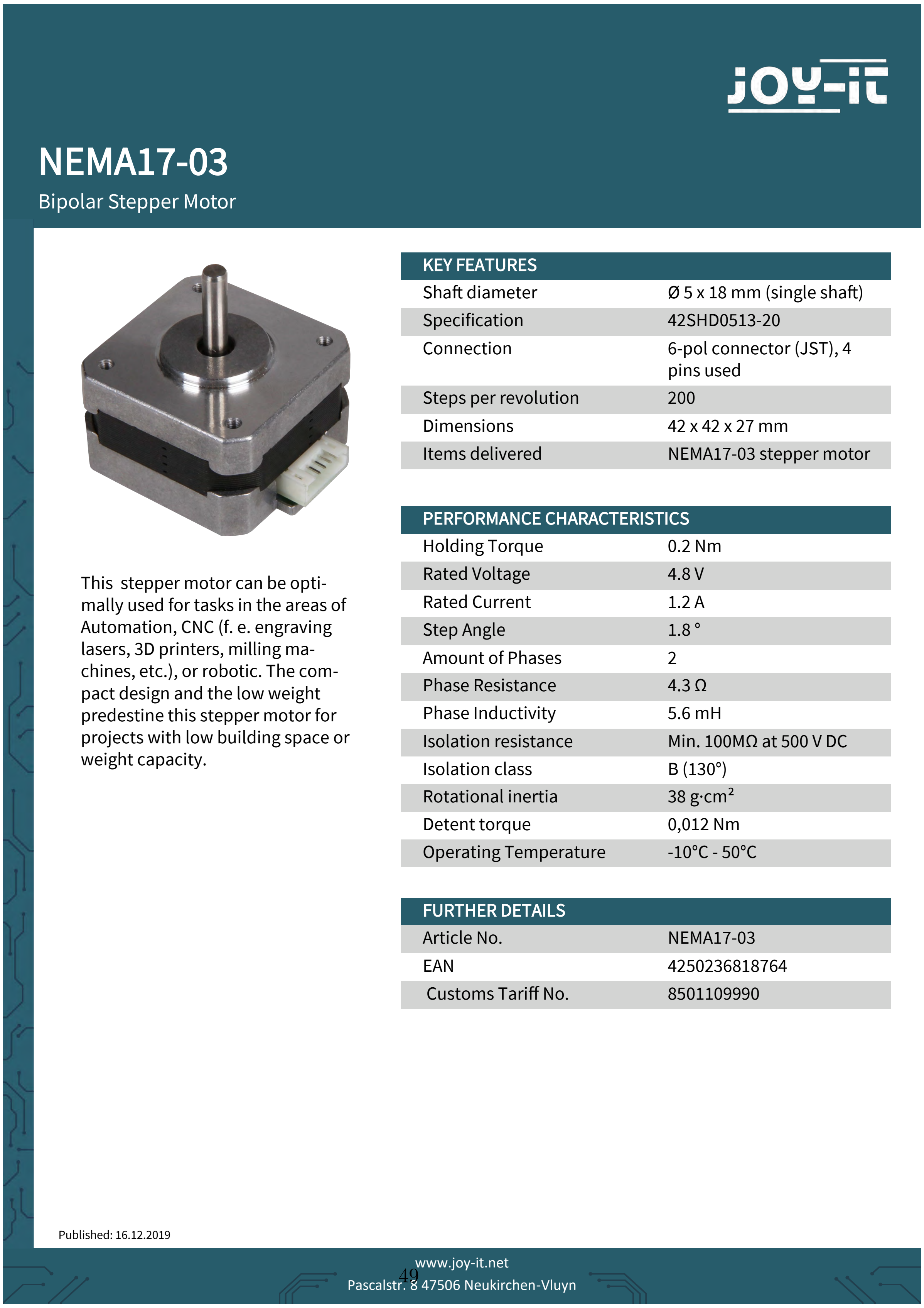
Dimensions: unit=mm

Motor Length:

Model	Length
17HS2XXX	28 mm
17HS3XXX	34 mm
16HS4XXX	40 mm
16HS8XXX	48 mm

9 www.MotionKing.com MK1106, Rev.04

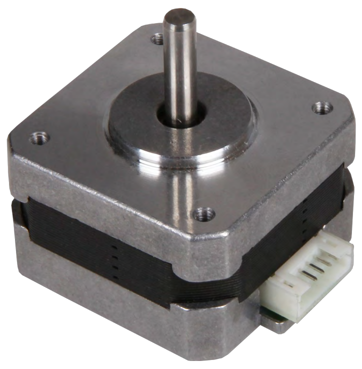
Figure D.3. Data sheet for the stepper motor used in the third and fourth axis.



JOY-iT

NEMA17-03

Bipolar Stepper Motor



This stepper motor can be optimally used for tasks in the areas of Automation, CNC (f. e. engraving lasers, 3D printers, milling machines, etc.), or robotic. The compact design and the low weight predestine this stepper motor for projects with low building space or weight capacity.

KEY FEATURES	
Shaft diameter	Ø 5 x 18 mm (single shaft)
Specification	42SHD0513-20
Connection	6-pol connector (JST), 4 pins used
Steps per revolution	200
Dimensions	42 x 42 x 27 mm
Items delivered	NEMA17-03 stepper motor

PERFORMANCE CHARACTERISTICS	
Holding Torque	0.2 Nm
Rated Voltage	4.8 V
Rated Current	1.2 A
Step Angle	1.8 °
Amount of Phases	2
Phase Resistance	4.3 Ω
Phase Inductivity	5.6 mH
Isolation resistance	Min. 100MΩ at 500 V DC
Isolation class	B (130°)
Rotational inertia	38 g·cm^2
Detent torque	0,012 Nm
Operating Temperature	-10°C - 50°C

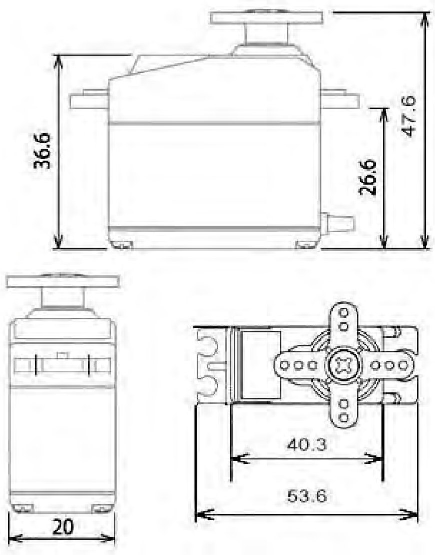
FURTHER DETAILS	
Article No.	NEMA17-03
EAN	4250236818764
Customs Tariff No.	8501109990

Published: 16.12.2019

www.joy-it.net
Pascalstr. 8 47506 Neukirchen-Vluyn

Figure D.4. Data sheet for the stepper motor used in the fifth and sixth axis.

MG996R High Torque Metal Gear Dual Ball Bearing Servo



This High-Torque MG996R Digital Servo features metal gearing resulting in extra high 10kg stalling torque in a tiny package. The MG996R is essentially an upgraded version of the famous MG995 servo, and features upgraded shock-proofing and a redesigned PCB and IC control system that make it much more accurate than its predecessor. The gearing and motor have also been upgraded to improve dead bandwith and centering. The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.

This high-torque standard servo can rotate approximately 120 degrees (60 in each direction). You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG996R Metal Gear Servo also comes with a selection of arms and hardware to get you set up nice and fast!

Specifications

- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 9.4 kgf·cm (4.8 V), 11 kgf·cm (6 V)
- Operating speed: 0.17 s/60º (4.8 V), 0.14 s/60º (6 V)

Figure D.5. Data sheet for the servo motor used in the gripper.

Appendix E

Python code of the user interface

main.py:

```
##############################################################
#                                                            #
#              A 6 axis robot arm controller                 #
#                                                            #
#              Michal Gabriel Sawczuk                        #
#              KTH MF133X                                    #
#              04/2021                                       #
#                                                            #
##############################################################

import PySimpleGUI as sg
import serial
import re

import GUI_images as im
import GUI_settings as settings
from GUI_ax import Axis


##############################################################
#                                                            #
#              GLOBAL VARIABLES AND SETTINGS                 #
#                                                            #
##############################################################

# Settings
port = settings.port  # Serial port
baud = settings.baud  # Baud rate
savedcommands = settings.savedcommands  # saved positions

# Special variables
special = False  # Special command that can overwrite positions that are not valid
connected = False  # Serial
ardu_prev_saved = {1: "", 2: "", 3: "", 4: "", 5: "", 6: "", 7: "", 8: "", 9: "",
     10: "", 11: "", 12: ""}

# Create axes
AXIS_1 = Axis("A1", "A1", 0, 0, -180, 180, settings.AXIS_1_ratio)  # NAME, COMMAND
     NAME, RL_POS, H0_POS, Z1_POS, Z2_POS, RATIO
AXIS_2 = Axis("A2", "A2", 0, 0, 0, 135, settings.AXIS_2_ratio)
AXIS_3 = Axis("A3", "A3", 0, 0, -180, 180, settings.AXIS_3_ratio)
AXIS_4 = Axis("A4", "A4", 0, 0, -90, 90, settings.AXIS_4_ratio)
AXIS_5 = Axis("A5", "A5", 0, 0, -180, 180, settings.AXIS_5_ratio)
AXIS_6 = Axis("A6", "A6", 0, 0, -90, 90, settings.AXIS_6_ratio)
AXIS_7 = Axis("A7", "GR", 20, 20, 20, 100, settings.AXIS_7_ratio)  # Gripper
ALL_AXIS = [AXIS_1, AXIS_2, AXIS_3, AXIS_4, AXIS_5, AXIS_6, AXIS_7]

# Themes
theme = settings.theme
sg.theme_add_new('Theme', theme)
```

```
theme2 = settings.theme2
sg.theme_add_new('Theme2', theme2)


############################################################
#                                                          #
#                  LAYOUT - MAIN WINDOW                    #
#                                                          #
############################################################

# Theme
sg.theme('Theme')

# Top bar
L_main_topbar = [sg.Text('6 DOF ROBOT ARM CONTROLLER' + 52 * ' ', font=('helvetica
    ', 15, 'bold')),
                 sg.Button('X', font=('helvetica', 10), button_color=('grey40',
                     '#222A68'), size=(2, 1), key='EXIT_MAIN')]

# Main buttons
L_main_buttonbar = [sg.Button('CONNECT', font=('helvetica', 10, 'bold'), size=(15,
     2), button_color=('white', '#654597'), key='-CONNECT_DISCONNECT-'),
                    sg.Button('SEND COMMAND', font=('helvetica', 10, 'bold'), size
                        =(15, 2), button_color=('black', '#FFB30F'), key='-
                        SEND_COMMAND-'),
                    sg.Button('SAVE COMMAND', font=('helvetica', 10, 'bold'), size
                        =(15, 2), button_color=('black', '#FFB30F'), key='-
                        SAVE_COMMAND-'),
                    sg.Button('LOAD COMMAND', font=('helvetica', 10, 'bold'), size
                        =(15, 2), button_color=('black', '#FFB30F'), key='-
                        LOAD_COMMAND-'),
                    sg.Button('SETTINGS', font=('helvetica', 10, 'bold'), size
                        =(15, 2), button_color=('white', '#E3170A'), key='-
                        SETTINGS-')]

# Model, info
L_main_model = [sg.Frame(layout=[[
    sg.Image(data=im.image_tom_base64, key='-IMAGE-', background_color="white",
        pad=(14, 14)),
    sg.Column([
        [sg.Text('Michal Gabriel Sawczuk', pad=(30, 0))],
        [sg.Text('KTH 2021', pad=(76, 0))]
    ])]], title='MODEL')]

# Axis frames, column 1  (see GUI_ax.py)
L_main_col_1 = [sg.Column([
    AXIS_1.createCommandWindow("AXIS 1"),
    AXIS_2.createCommandWindow("AXIS 2"),
    AXIS_3.createCommandWindow("AXIS 3"),
    AXIS_4.createCommandWindow("AXIS 4")])]

# Axis and model frames, column 2 (see GUI_ax.py)
L_main_col_2 = [sg.Column([
    AXIS_5.createCommandWindow("AXIS 5"),
    AXIS_6.createCommandWindow("AXIS 6"),
    AXIS_7.createCommandWindow("GRIPPER"),
    L_main_model])]

# Preview of the current and the last command
L_main_preview = [sg.Frame(layout=[[sg.Text('', font=('courier', 8), size=(94, 1),
     key='-COMMAND_PREVIEW-')],
                                   [sg.Text('', font=('courier', 8), size=(94, 1),
                                       key='-COMMAND_PREVIEW_LAST-', text_color
                                      =('#FFB30F'))]], title="COMMAND")]

# Arduino prints
L_main_arduino_frame = []
for i in range(12, 0, -1):
    L_main_arduino_frame.append([sg.Text('', font=('courier', 8), size=(94, 1),
        key='-ARDUINO_PREVIEW_' + str(i) + '-', text_color=('#FFB30F'))])
L_main_arduino = [sg.Frame(layout=L_main_arduino_frame, title="ARDUINO PRINT")]

# Main window layout
layout = [
    L_main_topbar,
    L_main_buttonbar,
    [*L_main_col_1,
     *L_main_col_2],
    L_main_preview,
    L_main_arduino]

# Create main window
window = sg.Window('6 DOF ROBOT ARM CONTROLLER', layout, default_element_size=(40,
     1), grab_anywhere=False, finalize=True, no_titlebar=True)

# Bind frames to mouse
window.bind('<FocusOut>', '+FOCUS OUT+')
AXIS_1.bindFrame(window)
AXIS_2.bindFrame(window)
AXIS_3.bindFrame(window)
AXIS_4.bindFrame(window)
AXIS_5.bindFrame(window)
AXIS_6.bindFrame(window)
AXIS_7.bindFrame(window)


############################################################
#                                                          #
#                       LOAD WINDOW                        #
#                                                          #
############################################################


def load_commands():
    sg.theme('Theme2')

    # Read file
    file = open(savedcommands, "r")
    file_vec = file.read().splitlines()

    # Topbar
    L_load_topbar = [sg.Text('LOAD COMMANDS', size=(50, 1), font=('helvetica', 15,
         'bold')),
                    sg.Button('X', font=('helvetica', 10), button_color=('grey40
                        ', '#39A9DB'), size=(2, 1), key='Exit_commands')]

    # Part that shows file content
    L_load_file = []
    for j, i in enumerate(file_vec):
        if (j % 2) != 0:
            L_load_file.append([sg.Radio(str(i), 'command', font=('helvetica', 10)
                , enable_events=True, default=True, key=("-CMDR_" + str(j) + "-"))
                ])
        else:
            L_load_file.append([sg.Text(str(i), font=('courier', 8), enable_events
               =True, key=("-CMD_" + str(j) + "-"))])

    # Load Button
    L_load_button = [sg.Button("Load", key='-LOAD-'),
                    sg.Button("Load Special", button_color=('black', '#AFFC41'),
                        key='-LOAD_SPECIAL-')]

    # Layout of the load window
    layout = [
        L_load_topbar,
        *L_load_file,
        L_load_button]

    # Create window
    window_commands = sg.Window('LOAD COMMAND', layout, finalize=True, no_titlebar
        =True, grab_anywhere=True)

    # Event loop
    holder = ""
    while True:
        # Read events
        event_c, values_c = window_commands.read()
        # Close window if X
        if event_c in (sg.WIN_CLOSED, 'Exit_commands'):
            break

        # Load command
        if event_c == '-LOAD-':
            for i in range(1, len(L_load_file), 2):
                if values_c["-CMDR_" + str(i) + "-"] == True:
                    holder = file_vec[i + 1]
                    break
            break

        # Load special command
        if event_c == '-LOAD_SPECIAL-':
            for i in range(1, len(L_load_file), 2):
                if values_c["-CMDR_" + str(i) + "-"] == True:
                    holder = file_vec[i + 1]
                    holder = "SPEC" + holder
                    break
            break

    # Close window and return loaded command
    window_commands.close()
    if holder != "":
        return holder
    else:
        return


##############################################################
#                                                            #
#                      SETTING WINDOW                        #
#                                                            #
##############################################################


def settingWindow():
    sg.theme('Theme2')

    # Topbar
    L_setting_topbar = [sg.Text('SETTINGS', size=(26, 1), font=('helvetica', 15, '
        bold')),
                        sg.Button('X', font=('helvetica', 10), button_color=('
                            grey40', '#39A9DB'), size=(2, 1), key='Exit_settings')
                            ]

    # Ratios bar
    L_setting_ratios = []
    for i in range(1, 7):
        L_setting_ratios.append([sg.Text("Ratio Axis " + str(i) + ": ", font=('
            courier', 10)),
                                 sg.Text(round(globals()["AXIS_" + str(i)].RATIO),
                                     size=(5, 1), key="RATIO" + str(i)),  #
                                     UPDATE KEY
                                 sg.InputText("steps", size=(8, 1), key="A" + str(
                                     i) + "_steps"),
                                 sg.InputText("angle", size=(8, 1), key="A" + str(
                                     i) + "_angle"),
                                 sg.Button("Save", key="A" + str(i) + "_ratio")])
    L_setting_ratios.append([sg.Text("Ratio Gripper:", font=('courier', 10)), sg.
        Text(round(globals()["AXIS_7"].RATIO), size=(5, 1), key="RATIO7"), sg.
        InputText("steps", size=(8, 1), key="A7_steps"), sg.InputText("angle",
        size=(8, 1), key="A7_angle"), sg.Button("Save", key="A7_ratio")])

    # Layout
    layout = [L_setting_topbar, *L_setting_ratios]

    window_settings = sg.Window('SETTINGS', layout, finalize=True, no_titlebar=
        True, grab_anywhere=True)

    # Event loop
    while True:
        # Read events
        event_s, values_s = window_settings.read()
        # Update ratios
        ratio_reg = re.findall("((A[0-9])_ratio)", event_s)
        if ratio_reg:
            steps = values_s[ratio_reg[0][1] + "_steps"]
            angle = values_s[ratio_reg[0][1] + "_angle"]
            ratio = round(int(steps) / int(angle))
            try:
                globals()["AXIS_" + ratio_reg[0][1][1]].RATIO = ratio
                window_settings['RATIO' + str(ratio_reg[0][1][1])].update(ratio)
                with open("GUI_settings.py", "r") as file:
                    setting_list = file.readlines()
                settings_reg = [[k, i] for k, i in enumerate(setting_list) if re.
                    search("(AXIS_" + ratio_reg[0][1][1] + "_ratio = [0-9]*)", i)
                    is not None]
                setting_list[settings_reg[0][0]] = 'AXIS_' + str(ratio_reg
                    [0][1][1]) + '_ratio = ' + str(ratio) + "\n"
                with open("GUI_settings.py", "w") as filew:
                    filew.writelines(setting_list)
            except:
                sg.popup_ok("invalid ratios")

        # Close window if X
        if event_s in (sg.WIN_CLOSED, 'Exit_settings'):
            break

    window_settings.close()

##############################################################
#                                                            #
#                        FUNCTIONS                           #
#                                                            #
##############################################################


def connectDisconnect():
    # Updates connect/disconnect button and connects/disconnects from the robot
        arm
    global serial_connection
    if connected == False:
        window['-CONNECT_DISCONNECT-'].update('...', button_color=('white',
            '#654597'))
        try:
            serial_connection = serial.Serial(port, baud)
            window['-CONNECT_DISCONNECT-'].update('DISCONNECT', button_color=('
                black', '#AFFC41'))
            return True
        except:
            sg.popup_ok('Failed :c')
            window['-CONNECT_DISCONNECT-'].update('CONNECT', button_color=('white
                ', '#654597'))
            return False
    elif connected == True:
        window['-CONNECT_DISCONNECT-'].update('CONNECT', button_color=('white',
            '#654597'))
        serial_connection.close()
        return False


def updateArduinoPrev(new_messege):
    for i in range(12, 1, -1):
        ardu_prev_saved[i] = ardu_prev_saved[i - 1]
    ardu_prev_saved[1] = new_messege
    for i in range(1, 13):
        window['-ARDUINO_PREVIEW_' + str(i) + '-'].update(ardu_prev_saved[i])


def updateCommandPreview():
    command_vec = []
    command_txt = ""
    command_vec.append(AXIS_1.getCommand())
    command_vec.append(AXIS_2.getCommand())
    command_vec.append(AXIS_3.getCommand())
    command_vec.append(AXIS_4.getCommand())
    command_vec.append(AXIS_5.getCommand())
    command_vec.append(AXIS_6.getCommand())
    command_vec.append(AXIS_7.getCommand())
    command_vec.insert(0, "<")
    command_vec.insert(len(command_vec), ">")
    command_txt = ''.join([str(data) for data in command_vec])
    window['-COMMAND_PREVIEW-'].update(command_txt)
    return command_txt


##############################################################
#                                                            #
#                      MAIN EVENT LOOP                       #
#                                                            #
##############################################################


while True:
    # Read events and values
    event, value = window.read()

    # Close window if X
    if event in (sg.WIN_CLOSED, 'EXIT_MAIN'):
        break

    # Connect/Disconnect from robot arm if Connect button activated
    if event == '-CONNECT_DISCONNECT-':
        connected = connectDisconnect()

    # Update arduino preview
    if connected:
        while serial_connection.in_waiting:
            updateArduinoPrev(serial_connection.readline().decode())

    # Open settings window
    if event == '-SETTINGS-':
        settingWindow()

    # Update model picture
    frame_event_reg = re.findall("([A-Z][0-9])(FRAME\+MOUSE OVER\+)", event)
    if frame_event_reg:
        img = "image_axis" + frame_event_reg[0][0][1] + "_base64"
        window['-IMAGE-'].update(data=im.all_b64[img])
    else:
        window['-IMAGE-'].update(data=im.image_tom_base64)

    # Read and update buttons, update TASK
    button_event_reg = re.findall("([A-Z][0-9](?!FRAME|SLIDER))([A-Z](?:[A-Z
        ]|[0-9]))", event)
    if button_event_reg:
        TARGET_reg = "AXIS_" + button_event_reg[0][0][1]
        globals()[TARGET_reg].TASK = button_event_reg[0][1]
        globals()[TARGET_reg].updateButtonColors(window)
        updateCommandPreview()

    # Read sliders and update VALUE
    slider_event_reg = re.findall("([A-Z][0-9]SLIDER)", event)
    if slider_event_reg:
        TARGET_reg = "AXIS_" + slider_event_reg[0][1]
        globals()[TARGET_reg].VALUE = int(value[slider_event_reg[0]])
        updateCommandPreview()

    # Load position and update TASK,VALUE
    if event == '-LOAD_COMMAND-':
        loaded = load_commands()
        if loaded != None:
            special = True if re.search("(SPEC)", loaded) != None else False
            list_reg = re.findall("([A|G](?:[1-6]|R)\*(?:[A-Z]*[0-9]*)\*\-*[0-9]*)
                ", loaded)
            for i, k in enumerate(list_reg):
                target_reg = re.findall("([A-Z](?:[A-Z]|[0-9])(?=\*[A-Z]))(?:\*([A
                    -Z](?:[A-Z]|[0-9]))\*)(-*[0-9]+)", k)
                target = "AXIS_" + target_reg[0][0][1] if target_reg[0][0] != "GR"
                     else "AXIS_7"
                target_reg_list = list(target_reg[0])
                target_reg_list[2] = str(round((int(target_reg_list[2])) / (
                    globals()[target].RATIO)))

                globals()[target].TASK = target_reg_list[1]
                globals()[target].updateButtonColors(window)
                globals()[target].VALUE = int(target_reg_list[2])
                window['-COMMAND_PREVIEW-'].update(loaded)
                globals()[target].updateSlider(window)
    # Save choosen position
    if event == '-SAVE_COMMAND-':
        description = sg.popup_get_text('Describe this command:')
        file = open(savedcommands, "a")
        file.write('\n' + description)
        file.write('\n' + updateCommandPreview())
        file.close()

    # Send position to arduino
    if event == '-SEND_COMMAND-' and connected:
        valid = False

        # Check if command is valid
        if special == False:
            for AX in ALL_AXIS:
                valid = AX.testCommand()
                if not valid:
                    sg.popup_ok('Command not valid')
                    break
        else:
            valid = True

        # Send command and update values if valid
        if valid:
            serial_connection.write(updateCommandPreview().encode())
            for AX in ALL_AXIS:
                AX.updateCommand(window)
                AX.updateSlider(window)
            window['-COMMAND_PREVIEW_LAST-'].update(updateCommandPreview())

    # Pop up if arm not connected
    elif event == '-SEND_COMMAND-' and not connected:
        sg.popup_ok('Robot arm not connected!')

window.close()
SSSSS
```

GUI_images.py: (Note that the image files are needed to run the code)

```
import base64

# Import model images from ./img folder
image_tom = './img/tom.png'
image_tom_64 = open(image_tom, 'rb+')
image_axis1 = './img/axis1.png'
image_axis1_64 = open(image_axis1, 'rb+')
image_axis2 = './img/axis2.png'
image_axis2_64 = open(image_axis2, 'rb+')
image_axis3 = './img/axis3.png'
image_axis3_64 = open(image_axis3, 'rb+')
image_axis4 = './img/axis4.png'
image_axis4_64 = open(image_axis4, 'rb+')
image_axis5 = './img/axis5.png'
image_axis5_64 = open(image_axis5, 'rb+')
image_axis6 = './img/axis6.png'
image_axis6_64 = open(image_axis6, 'rb+')
image_axis7 = './img/gripper.png'
image_axis7_64 = open(image_axis7, 'rb+')

# Convert images to base64
image_tom_base64 = base64.b64encode(image_tom_64.read())
image_axis1_base64 = base64.b64encode(image_axis1_64.read())
image_axis2_base64 = base64.b64encode(image_axis2_64.read())
image_axis3_base64 = base64.b64encode(image_axis3_64.read())
image_axis4_base64 = base64.b64encode(image_axis4_64.read())
image_axis5_base64 = base64.b64encode(image_axis5_64.read())
image_axis6_base64 = base64.b64encode(image_axis6_64.read())
image_axis7_base64 = base64.b64encode(image_axis7_64.read())

all_b64 = {"image_axis1_base64": image_axis1_base64,
           "image_axis2_base64": image_axis2_base64,
           "image_axis3_base64": image_axis3_base64,
           "image_axis4_base64": image_axis4_base64,
           "image_axis5_base64": image_axis5_base64,
           "image_axis6_base64": image_axis6_base64,
           "image_axis7_base64": image_axis7_base64}
```

GUI_ax.py:

```
import PySimpleGUI as sg


class Axis:
    def __init__(self, NAME, CNAME, RL_POS, H0_POS, Z1_POS, Z2_POS, RATIO):
        self.NAME = NAME
        self.CNAME = CNAME
        self.RL_POS = RL_POS  # Actual position
        self.H0_POS = H0_POS  # Home position
        self.Z1_POS = Z1_POS  # Limit position (1)
        self.Z2_POS = Z2_POS  # Limit position (2)
        self.RATIO = RATIO  # steps per angle
        self.TASK = "no"
        self.VALUE = 0
        self.KEYS = {
            "Slider": (self.NAME + 'SLIDER'),
            "Frame": (self.NAME + 'FRAME'),
            "Move button": (self.NAME + 'MV'),
            "Home button": (self.NAME + 'HM'),
            "Home 0 button": (self.NAME + 'H0'),
            "Zero 1 button": (self.NAME + 'Z1'),
            "Zero 2 button": (self.NAME + 'Z2'),
            "Position": (self.NAME + 'POS'),
            "Home 0 position": (self.NAME + 'H0POS'),
            "Zero 1 position": (self.NAME + 'Z1POS'),
            "Zero 2 position": (self.NAME + 'Z2POS')
        }

    def getSteps(self, angle):
        # calculate the amount of steps required to reach the target angle
        return round(self.RATIO * angle)

    def getCommand(self):
        steps = round(self.RATIO * self.VALUE)
        return self.CNAME + '*' + self.TASK + '*' + str(steps) + '!'

    def getCommandReverse(self):
        steps = round(1 / self.RATIO * self.VALUE)
        return self.CNAME + '*' + self.TASK + '*' + str(steps) + '!'

    def createCommandWindow(self, title):
        frame = [
            sg.Frame(layout=[
                [sg.Slider(range=(self.Z1_POS, self.Z2_POS), orientation='h', size
                    =(34.5, 15), default_value=self.H0_POS, enable_events=True,
                    key=self.KEYS["Slider"])],
                [sg.Button(' > ', button_color=('#E3170A', '#759FBC'), font=('
                    helvetica', 10, 'bold'), key=self.KEYS["Move button"], size
                    =(3, 1)),
                 sg.Button(' H ', button_color=('#FFB30F', '#759FBC'), font=('
                     helvetica', 10, 'bold'), key=self.KEYS["Home button"], size
                     =(3, 1)),
                 sg.Text('   '),
                 sg.Button('  H    ', button_color=('black', '#759FBC'), font=('
                     helvetica', 10), key=self.KEYS["Home 0 button"], size=(3, 1))
                     ,
                 sg.Button(' 0    ', button_color=('black', '#759FBC'), font=('
                     helvetica', 10), key=self.KEYS["Zero 1 button"], size=(3, 1))
                     ,
                 sg.Button(' 0    ', button_color=('black', '#759FBC'), font=('
                     helvetica', 10), key=self.KEYS["Zero 2 button"], size=(3, 1))
                     ,
                 sg.Text('   ')],
                [sg.Text('Pos:'),
                 sg.Text(self.RL_POS, text_color=('#FFB30F'), size=(3, 1), key=
                     self.KEYS["Position"]),
                 sg.Text('  H  :'),
                 sg.Text(self.H0_POS, text_color=('#FFB30F'), size=(3, 1), key=
                     self.KEYS["Home 0 position"]),
                 sg.Text(' 0    :'),
                 sg.Text(self.Z1_POS, text_color=('#FFB30F'), size=(3, 1), key=
                     self.KEYS["Zero 1 position"]),
                sg.Text(' 0    :'),
                sg.Text(self.Z2_POS, text_color=('#FFB30F'), size=(3, 1), key=
                    self.KEYS["Zero 2 position"])]
            ], title=title, key=self.KEYS["Frame"])
        ]
        return frame

    def bindFrame(self, window):
        window[self.KEYS["Frame"]].bind('<Enter>', '+MOUSE OVER+')
        window[self.KEYS["Frame"]].bind('<Leave>', '+MOUSE AWAY+')

    def updateButtonColors(self, window):
        on = ('black', '#AFFC41')
        off1 = ('#E3170A', '#759FBC')
        off2 = ('#FFB30F', '#759FBC')
        off3 = ('black', '#759FBC')
        if self.TASK == 'MV':
            window[self.KEYS["Move button"]].update(button_color=on)
            window[self.KEYS["Home button"]].update(button_color=off2)
            window[self.KEYS["Home 0 button"]].update(button_color=off3)
            window[self.KEYS["Zero 1 button"]].update(button_color=off3)
            window[self.KEYS["Zero 2 button"]].update(button_color=off3)
        elif self.TASK == 'HM':
            window[self.KEYS["Move button"]].update(button_color=off1)
            window[self.KEYS["Home button"]].update(button_color=on)
            window[self.KEYS["Home 0 button"]].update(button_color=off3)
            window[self.KEYS["Zero 1 button"]].update(button_color=off3)
            window[self.KEYS["Zero 2 button"]].update(button_color=off3)
        elif self.TASK == 'H0':
            window[self.KEYS["Move button"]].update(button_color=off1)
            window[self.KEYS["Home button"]].update(button_color=off2)
            window[self.KEYS["Home 0 button"]].update(button_color=on)
            window[self.KEYS["Zero 1 button"]].update(button_color=off3)
            window[self.KEYS["Zero 2 button"]].update(button_color=off3)
        elif self.TASK == 'Z1':
            window[self.KEYS["Move button"]].update(button_color=off1)
            window[self.KEYS["Home button"]].update(button_color=off2)
            window[self.KEYS["Home 0 button"]].update(button_color=off3)
            window[self.KEYS["Zero 1 button"]].update(button_color=on)
            window[self.KEYS["Zero 2 button"]].update(button_color=off3)
        elif self.TASK == 'Z2':
            window[self.KEYS["Move button"]].update(button_color=off1)
            window[self.KEYS["Home button"]].update(button_color=off2)
            window[self.KEYS["Home 0 button"]].update(button_color=off3)
            window[self.KEYS["Zero 1 button"]].update(button_color=off3)
            window[self.KEYS["Zero 2 button"]].update(button_color=on)

    def updateSlider(self, window):
        window[self.KEYS["Slider"]].update(self.VALUE)

    def testCommand(self):
        if self.TASK == "Z1" and (self.VALUE >= self.Z2_POS or self.VALUE >= self.
            H0_POS):
            return False
        elif self.TASK == "Z2" and (self.VALUE <= self.Z1_POS or self.VALUE <=
            self.H0_POS):
            return False
        elif self.TASK in ["H0", "MV", "HM"] and (self.VALUE < self.Z1_POS or self
            .VALUE > self.Z2_POS):
            return False
        elif self.TASK not in ["H0", "MV", "HM", "Z1", "Z2"]:
            return False
        else:
            return True

    def updateCommand(self, window):
        if self.TASK == "Z1":
            self.Z1_POS = self.VALUE
            window[self.KEYS["Zero 1 position"]].update(self.Z1_POS)
        elif self.TASK == "Z2":
            self.Z2_POS = self.VALUE
            window[self.KEYS["Zero 2 position"]].update(self.Z2_POS)
        elif self.TASK == "H0":
            self.H0_POS = self.VALUE
    window[self.KEYS["Home 0 position"]].update(self.H0_POS)
elif self.TASK == "MV":
    self.RL_POS = self.VALUE
    window[self.KEYS["Position"]].update(self.RL_POS)
elif self.TASK == "HM":
    self.RL_POS = self.H0_POS
    self.VALUE = self.H0_POS
    window[self.KEYS["Position"]].update(self.RL_POS)
```

Appendix F

Arduino code

```
/*
 *  Arduino code that reads serial data received from the serial
 *  monitor/external user interface and controls the robot actuators
 *
 *  Michal Gabriel Sawczuk
 *  MF133X KTH 2021-04-15
 */

#include <AccelStepper.h>
#include <MultiStepper.h>
#include <Servo.h>
#include <Buzzer.h>

//------------PINS
//RAMPS individual pins:
#define PWR_3 10
#define BUZZ 4
#define SER_PIN 11
#define LED_R 3
#define LED_G 2
#define LED_B 6
#define MODE 15
#define A1_EN 16
#define A1_DIR 23
#define A1_STEP 17
#define A2_EN 25
#define A2_DIR 29
#define A2_STEP 27
//RAMPS E0/E1 pins
#define A3_DIR 28
#define A3_STEP 26
#define A3_EN 24
#define A4_DIR 34
#define A4_STEP 36
#define A4_EN 30
//RAMPS X/Y pins
#define A5_DIR A1
#define A5_STEP A0
#define A5_EN 38
#define A6_DIR A7
#define A6_STEP A6
#define A6_EN A2
//RAMPS power output
#define PWR_1 8
#define PWR_2 9

//------------VARIABLES USED FOR THE SERIAL DATA TRANSFER
bool newCommands = false; //new commands available/not available
char DATA[91]; //Data received is stored here
char TARGET[7][3]; //Target of every command stored here
char TASK[7][3]; //Task of each command stored here
char VALUE[7][8]; //Value of each command stored here
bool newExtracted = false; //new extracted data available
long stepperPositions[6] = {0,0,0,0,0,0}; //received stepper positions
long chDIFF_3[4] = {0,0,0,0}; //Differential positions holder
long DIFF_3[4] = {0,0,0,0}; //Differential positions holder
long DI_3[4] = {0,0,0,0}; //Differential positions holder

long servoPosition; //received servo position
long stepperHomePositions[6] = {0,0,0,0,0,0};
long servoHomePosition = 0;
long stepperZero1Positions[6] = {-100000,-100000,-100000,-100000,-100000,-100000};
long servoZero1Position = 0;
long stepperZero2Positions[6] = {100000,100000,100000,100000,100000,100000};
long servoZero2Position = 0;
bool newPositions = false; //new positions ready to run available
int mode; //acceleration on/off

//-----------STEPPER,SERVO AND BUZZER INSTANCES
AccelStepper AX1(1,A1_STEP,A1_DIR);
AccelStepper AX2(1,A2_STEP,A2_DIR);
AccelStepper AX3(1,A3_STEP,A3_DIR);
AccelStepper AX4(1,A4_STEP,A4_DIR);
AccelStepper AX5(1,A5_STEP,A5_DIR);
AccelStepper AX6(1,A6_STEP,A6_DIR);
MultiStepper ALL; //all axes
MultiStepper DIFF; //differential
Servo SER; //gripper
Buzzer buzzer(BUZZ);

void setup() {
    Serial.begin(9600); //Start serial communication
    Serial.println("Arduino setup..."); //startup messenge (GUI)

    pinMode(PWR_1,OUTPUT); digitalWrite(PWR_1,HIGH); //turn on TB6600 stepper
        drivers
    pinMode(PWR_2,OUTPUT); digitalWrite(PWR_2,HIGH); //turn on servo
    pinMode(PWR_3,OUTPUT); digitalWrite(PWR_3,HIGH); //turn on fan

    pinMode(LED_R, OUTPUT); pinMode(LED_G, OUTPUT); pinMode(LED_B, OUTPUT); //LEDs
    RGBled(255,0,0);delay(1000);RGBled(0,255,0);delay(1000);RGBled(0,0,255); //
        startup blink

    pinMode(MODE,INPUT_PULLUP); //Mode switch

    SER.attach(SER_PIN); SER.write(75);delay(1000); SER.write(10); //setup servo
        connection

    pinMode(A3_EN, OUTPUT); digitalWrite(A3_EN, LOW); //turn on A4899 stepper
        drivers
    pinMode(A4_EN, OUTPUT); digitalWrite(A4_EN, LOW);
    pinMode(A5_EN, OUTPUT); digitalWrite(A5_EN, LOW);
    pinMode(A6_EN, OUTPUT); digitalWrite(A6_EN, LOW);

    AX1.setMaxSpeed(3000); AX1.setAcceleration(800); //setup speed and
        accelerations
    AX2.setMaxSpeed(3000); AX2.setAcceleration(800);
    AX3.setMaxSpeed(1000); AX3.setAcceleration(800);
    AX4.setMaxSpeed(1000); AX4.setAcceleration(800);
    AX5.setMaxSpeed(5000); AX5.setAcceleration(800);
    AX6.setMaxSpeed(5000); AX6.setAcceleration(800);

    ALL.addStepper(AX1); ALL.addStepper(AX2); ALL.addStepper(AX3); ALL.addStepper(
        AX4); ALL.addStepper(AX5); ALL.addStepper(AX6); //group steppers
    DIFF.addStepper(AX3);DIFF.addStepper(AX4);DIFF.addStepper(AX5);DIFF.addStepper
        (AX6);

    Serial.println("...complete"); //startup messenge (GUI)
    buzzer.begin(10);buzzer.sound(NOTE_B4, 100);buzzer.sound(NOTE_A4, 100);buzzer.
        sound(NOTE_B4, 100);buzzer.end(300); //startup sound
}

void loop() {
    readSerial(); //read data from the serial port
    translateToCommand(); //extract different parts of the data
    commandToPosition(); //extract the position vector from the command
    moveArm(); //move the robot arm
}


void readSerial() {
    /*
     *  Function that reads data from the serial port
     */
    static bool dataTransfering = false; //true if data transfer in progress
    static byte i = 0; //index
    char rc; //Received character

    char startChar = '<'; //Messege start
    char endChar = '>'; //Messege end

    while (Serial.available() > 0 && newCommands == false) {
        rc = Serial.read(); //Reads one char of the data
        if (rc == startChar) { //Start data transfer if startChar found
            dataTransfering = true;
        }
        else if (dataTransfering == true) {
            if (rc != endChar) { //Transfer data as long as endChar not found
                DATA[i] = rc; //Save data
                i++;
            }
            else { //Stop data transfer if endChar found
                DATA[i] = '\0'; //End the string
                buzzer.begin(10);buzzer.sound(NOTE_B4, 50);buzzer.sound(NOTE_A5,
                    100); buzzer.end(10);
                Serial.println("Received data:");
                Serial.println(DATA);
                newCommands = true; //New data available
                i = 0; //Reset the index
                dataTransfering = false; //Stop data transfer
            }
        }
    }
}

void translateToCommand(){
    /*
    *  Function that extracts the target, task and value from DATA
    */
    int j = 0; //Command index
    int k = 0; //Command part index
    int l = 0; //Command part char index

    if (newCommands == true){ //Start translation if new data available
        for (int i = 0; i <= sizeof(DATA); i++){
            if (DATA[i] == '\0'){ //If data end, break
                break;
            }
            else if (DATA[i] == '!'){ //If command end, increase command index and
                 reset command part, command part char index
                j++;
                k = 0;
                l = 0;
            }
            else if (DATA[i] == '*'){ //If command part end, increase command part
                 index and reset command part char index
                k++;
                l = 0;
                }
            else{
                if (k == 0){ //extract target
                    TARGET[j][l] = DATA[i];
                }
                else if (k == 1){ //extract task
                    TASK[j][l] = DATA[i];
                }
                else if (k == 2){ //extract value
                    VALUE[j][l] = DATA[i];
                    if (DATA[i+1] == ',' || DATA[i+1] == '!' ){ //terminate string
```

```
                        if last char in value found
                        VALUE[j][l+1] = '\0';
                    }
                }
                l++; //increase command part index
            }
        }
        newExtracted = true;
        Serial.println("That gives following values:");
        Serial.print("TG[0]: ");Serial.print(TARGET[0]);Serial.print(" | TS[0]: ")
            ;Serial.print(TASK[0]);Serial.print(" | VL[0]: ");Serial.print(VALUE
            [0]);
        Serial.print(" ::: TG[1]: ");Serial.print(TARGET[1]);Serial.print(" | TS
            [1]: ");Serial.print(TASK[1]);Serial.print(" | VL[1]: ");Serial.
            println(VALUE[1]);
        Serial.print("TG[2]: ");Serial.print(TARGET[2]);Serial.print(" | TS[2]: ")
            ;Serial.print(TASK[2]);Serial.print(" | VL[2]: ");Serial.print(VALUE
            [2]);
        Serial.print(" ::: TG[3]: ");Serial.print(TARGET[3]);Serial.print(" | TS
            [3]: ");Serial.print(TASK[3]);Serial.print(" | VL[3]: ");Serial.
            println(VALUE[3]);
        Serial.print("TG[4]: ");Serial.print(TARGET[4]);Serial.print(" | TS[4]: ")
            ;Serial.print(TASK[4]);Serial.print(" | VL[4]: ");Serial.print(VALUE
            [4]);
        Serial.print(" ::: TG[5]: ");Serial.print(TARGET[5]);Serial.print(" | TS
            [5]: ");Serial.print(TASK[5]);Serial.print(" | VL[5]: ");Serial.
            println(VALUE[5]);
        Serial.print("TG[6]: ");Serial.print(TARGET[6]);Serial.print(" | TS[6]: ")
            ;Serial.print(TASK[6]);Serial.print(" | VL[6]: ");Serial.println(VALUE
            [6]);
    }
    newCommands = false;
}

void commandToPosition(){
    /*
    *  Function that translates the data from translateToCommand() to stepper
        motor and servo positions
    */
    if (newExtracted == true){
        for (int i = 0;i<7;i++){
            if (strcmp(TARGET[i],"A1")==0){ //Axis 1
                stepperPositions[0] = commandMatch(0,0);
            }
            else if (strcmp(TARGET[i],"A2")==0){ //Axis 2
                stepperPositions[1] = commandMatch(1,0);
            }
            else if (strcmp(TARGET[i],"A3")==0){ //Axis 3 (Note that stepper 3 and
                 4 work together)
                chDIFF_3[0] = commandMatch(2,0) - DI_3[0];
                chDIFF_3[1] = commandMatch(2,0) - DI_3[1];
                DI_3[0] = commandMatch(2,0);
                DI_3[1] = commandMatch(2,0);
                DIFF_3[0] = DIFF_3[2]+chDIFF_3[0]; //Stage 1 stepper 3
                DIFF_3[1] = DIFF_3[3]+chDIFF_3[1]; //Stage 1 stepper 4
                stepperPositions[2] = commandMatch(2,0);
                stepperPositions[3] = commandMatch(2,0);
            }
            else if (strcmp(TARGET[i],"A4")==0){ //Axis 4 (Note that stepper 3 and
                 4 work together)
                chDIFF_3[2] = -commandMatch(3,0) - DI_3[2];
                chDIFF_3[3] = commandMatch(3,0) - DI_3[3];
                DI_3[2] = -commandMatch(3,0);
                DI_3[3] = commandMatch(3,0);
                DIFF_3[2] = DIFF_3[0]+chDIFF_3[2]; //Stage 2 step 3
                DIFF_3[3] = DIFF_3[1]+chDIFF_3[3]; //Stage 2 step 4
                stepperPositions[2] = -commandMatch(3,0)+stepperPositions[2];
                stepperPositions[3] = commandMatch(3,0)+stepperPositions[3];
            }
            else if (strcmp(TARGET[i],"A5")==0){ //Axis 5 (Note that stepper 5 and
                 6 work together)
                stepperPositions[4] = commandMatch(4,0);
                stepperPositions[5] = commandMatch(4,0);
            }
            else if (strcmp(TARGET[i],"A6")==0){ //Axis 6 (Note that stepper 5 and
                 6 work together)
                stepperPositions[5] = stepperPositions[5];//commandMatch(5,0);
                stepperPositions[4] = stepperPositions[4];//commandMatch(5,1);
            }
            else if (strcmp(TARGET[i],"GR")==0){ //Gripper
                servoPosition = commandMatch(6,0);
            }
        }
        newExtracted = false;
        newPositions = true;
    }
}

long commandMatch(int i, int other){
    /*
    *  Function that analyzes the task and returns a relevant position
    */
    long valueholder;
    sscanf(VALUE[i], "%ld", &valueholder);

    if (strcmp(TASK[i],"MV")==0){ //Move robot arm
        if(valueholder < stepperZero2Positions[i] || valueholder >
            stepperZero1Positions[i]){
            return valueholder;
        }
        else{
            return stepperPositions[i];
        }
    }
    else if (strcmp(TASK[i],"HM")==0){ //Move robot arm to home position
        if(valueholder < stepperZero2Positions[i] || valueholder >
            stepperZero1Positions[i]){
            return stepperHomePositions[i];
        }
        else{
            return stepperPositions[i];
        }
    }
    else if (strcmp(TASK[i],"H0")==0){ //Set home position
        if(valueholder < stepperZero2Positions[i] || valueholder >
            stepperZero1Positions[i]){
            stepperHomePositions[i] = valueholder;
        }
        return stepperPositions[i];
    }
    else if (strcmp(TASK[i],"Z1")==0){ //Set first limit position
        stepperZero1Positions[i] = valueholder;
        return stepperPositions[i];
    }
    else if (strcmp(TASK[i],"Z2")==0){ //Set second limit position
        stepperZero2Positions[i] = valueholder;
        return stepperPositions[i];
    }
    else{
      return stepperPositions[i];
    }
}

void moveArm() {
    /*
     *  Function that sends the data to the stepper motor drivers and to the servo
         motor
     */
    mode = digitalRead(MODE);
    switch(mode){
        //case 0: no acceleration control, move all axis at the same time
        case 0:
            RGBled(0,255,0); //change led to green
            if (newPositions == true){
                ALL.moveTo(stepperPositions);
                ALL.runSpeedToPosition();
                SER.write(servoPosition);
                newPositions = false;
            }
            break;
        //case 1: acceleration control, move one axis after another
        case 1:
            RGBled(0,0,255); //change led to blue
            if (newPositions == true){
                //Move Axis 1
                AX1.moveTo(stepperPositions[0]);
                while (AX1.distanceToGo() != 0) {
                    AX1.run();
                }
                //Move Axis 2
                AX2.moveTo(stepperPositions[1]);
                while (AX2.distanceToGo() != 0) {
                    AX2.run();
                }
                //Move Axis 3
                AX3.moveTo(DIFF_3[0]);
                AX4.moveTo(DIFF_3[1]);
                while (AX3.distanceToGo() != 0 && AX4.distanceToGo() != 0) {
                    AX3.run();
                    AX4.run();
                }
                //Move Axis 4
                AX3.moveTo(DIFF_3[2]);
                AX4.moveTo(DIFF_3[3]);
                while (AX3.distanceToGo() != 0 && AX4.distanceToGo() != 0) {
                    AX3.run();
                    AX4.run();
                }
                //Move servo
                SER.write(servoPosition);


                newPositions = false;
            }
            break;
    }
}

void RGBled(int red, int green, int blue){
    /*
    *  Function that controls the LED
    */
    analogWrite(LED_R,red);
    analogWrite(LED_G,green);
    analogWrite(LED_B,blue);
}
```

Appendix G

Change in error (Precision tests).

Table G.1. Rate of change low speed.

Test	Axis 1	Axis 2	Axis 3	Axis 4	Axis 5	Axis 6	All 1	All 2
1	-	-	-	-	-	-	-	-
2	0.05	0.02	0.08	-0.01	-0.02	0.05	0.06	-0.75
3	0.03	0.03	0.02	0.01	0.11	0.16	0.04	0.18
4	0.02	0.06	0	0	0.11	0.14	0.02	0.07
5	-0.03	0.05	0.02	0.02	0.25	0.22	0.01	0.15
6	0.09	0.05	0.02	0	0.15	0.35	0.14	0.19
7	0.22	0.02	0.01	0.01	0.38	0.3	-0.2	0.01
8	0.05	0.01	0.02	0.01	0.32	0.15	0.06	0.41
9	-0.01	0.01	0	0.01	0.31	0.38	-0.08	0.07
10	0.06	0.02	0	0	0.42	0.42	-0.07	0.21
Average	0.0533	0.03	0.0189	0.0056	0.2256	0.2411	-0.0022	0.06

Table G.2. Rate of change high speed.

Test	Axis 1	Axis 2	Axis 3	Axis 4	Axis 5	Axis 6	All 1	All 2
1	-	-	-	-	-	-	-	-
2	-0.07	0.03	0.01	0.12	0.31	0.09	0.4	0.18
3	0.14	0.05	0	0.16	0.59	0.3	0.12	0.2
4	0.02	0.02	0	0.24	0.73	0.62	0.1	0.15
5	0.02	0.05	0.01	0.2	1.11	0.74	0.07	0.05
6	0.12	0.05	0.02	0.3	1.35	1.16	-0.07	0.22
7	0.16	0.04	-0.02	0.22	1.3	1.07	0.32	0.19
8	0.38	0.02	0.01	0.18	1.35	0.83	0.02	0.15
9	0.2	0.04	0	0.35	0.44	2.04	0.08	0.04
10	0.24	0.02	0	0.35	1.23	1.46	0.08	0.14
Average	0.1344	0.0356	0.0033	0.2356	0.9344	0.9233	0.1244	0.1467

TRITA -ITM-EX 2021:52

www.kth.se