GITAM
(DEEMED TO BE UNIVERSITY)
VISAKHAPATNAM • HYDERABAD • BENGALURU

Internship report on
Stepper Motor Control System with Encoder Feedback using Arduino

Submitted By
Gudipati Rohith (32201405026)
GITAM University, Bengaluru
E-mail: rohithgudipati2002@gmail.com

Under the guidance of
Narendra Reddy T
Scientist - D

During the period
15th May 2023 – 6th July 2023

केन्द्रीय विनिर्माणकारी प्रौद्योगिकी संस्थान
CENTRAL MANUFACTURING TECHNOLOGY INSTITUTE
(An Autonomous R&D Institute
Under Ministry of Heavy Industries, Govt. of India)
SINCE 1962

CENTRAL MANUFACTURING TECHNOLOGY INSTITUE
TUMKUR ROAD, BENGALURU-560022, INDIA

INTERNSHIP COMPLETION LETTER

06-07-23

Subject: Request for the Internship completion certificate

Respected Sir/Mam,

I G Rohith (322010405026) third-year Electronics and Communication Engineering student at GITAM University Bengaluru I'm thankful to CMTI for having provided me with the opportunity to carry out my internship on the topic Internship report on 'Stepper Motor Control System with Encoder Feedback using Arduino (Mega)' under the guidance of Mr. Narendra Reddy T, Scientist-D, Additive and Special Manufacturing Process (D-SMPM).

I am grateful for all the experience and knowledge I have received during my internship from

15.05.2023 to 06.07.2023.

I hereby request you kindly issue the internship completion certificate.

Thanking you,

Yours Sincerely,

G Rohith

Intern

C – SMPM

Centre Head (C – SMPM)

Mr. Narendra Reddy T

(Scientist – D)

ACKNOWLEDGEMENT

I would like to thank Central Manufacturing Technology Institute (CMTI) for giving me the opportunity to do an internship within the organization. I also would like thank to all the people that I worked with at Central Manufacturing Technology Institute (CMTI) with their patience and openness they created an enjoyable working environment.

I convey my sincere gratitude and special thanks to Mr. Narendra Reddy T (Scientist-D) (SMPM, CMTI) for guiding me through the internship and to all who guided me in the execution and completion of this internship successfully. Thank you for speaking so favorably of my work ethic. It encourages me to continue to perform at a high standard.

And also acknowledge my deepest gratitude and special thanks to Mr. Vinay Kumar P In - charge & Staff AEAMT for his continuous support and for providing me an opportunity to work in CMTI as an Intern.

Rohith G

TABLE OF CONTENTS

About CMTI

Central Manufacturing Technology Institute, CMTI, is a Research & Development organization focusing on providing Technology Solutions to the manufacturing sector and assisting technological growth in the country. CMTI plays a key role in applied research, design, and development (ROAD), technology forecasting, assimilation, and dissemination of manufacturing technology to Indian industries.

CMTI is a registered Government of India Society, an autonomous institution under the administrative control of the Ministry of Heavy Industries, Government of India, governed by a Governing Council which has representation from the machine tool manufacturing and user industries, the Union Government and the Government of Karnataka, the Governing Council evolves Policies and monitors policy deployment. A Research Advisory Board (RAB), Technical Committee with representatives from industries and academia, assists the institute- on matters relating to technology advancement.

Drishti is an online platform developed by CMTI that provides a synergy between industries and young innovators to solve complex challenges. CMTI, under the SAMARTH Udyog Bharat 4.0 Platform of the Department of Heavy Industry (DHI) is setting up a Smart Manufacturing Demo & Development Cell (SMDDC) as a Common Engineering Facility Centre (CEFC) to develop, propagate and support the process of adoption of smart manufacturing practices, by the rapidly growing Indian manufacturing industry. Skilling and Re-Skilling in Smart Manufacturing are one of the local activities of SMDDC.

VISION:

- Achieve Technological Leadership and Excellence in the Quality of Products and Services.
- Establish a Dynamic, flexible, and result-oriented organizational structure.
- Achieve organizational excellence through a transparent, professional management system.

Stepper Motor Control System with Encoder Feedback using Arduino

Abstract

The Stepper Motor Control System with Encoder Feedback is a project that aims to enhance the control and monitoring capabilities of stepper motors through the use of rotary encoders. Stepper motors are widely utilized in applications requiring precise and controlled movement. This project expands upon the functionality of stepper motors by integrating rotary encoders, which provide accurate position feedback.

The system consists of multiple stepper motors, each equipped with a motor driver and a rotary encoder. The Arduino Mega microcontroller serves as the central control unit, responsible for processing user commands and driving the motors accordingly. The AccelStepper library is utilized to enable smooth acceleration, deceleration, and precise control of the stepper motors.

Rotary encoders are employed to provide real-time feedback on the position of the motors. By monitoring the rotational movements of the motors, the encoders transmit signals to the Arduino, enabling accurate position tracking. The Arduino interprets these signals through interrupt service routines (ISRs), updating the position information accordingly.

The project offers various functionalities for motor control. Users can interact with the system through the Arduino Serial Monitor, issuing commands to start, stop, and change the direction of the motors. The Serial Plotter feature provides a visual representation of the encoder states, facilitating analysis and verification of the motor movements.

The Stepper Motor Control System with Encoder Feedback presents a versatile solution for applications that require precise motor control and position feedback. It finds utility in diverse fields such as robotics, automation, CNC machines, 3D printers, and other projects that demand accurate and controlled motion.

1. Introduction

1.1 Stepper Motor Control System with Encoder Feedback using Arduino

Stepper motors are widely used in various applications that require precise and controlled motion, such as robotics, automation, CNC machines, and 3D printers. The ability to accurately control and monitor the movement of stepper motors is crucial for achieving desired outcomes in these applications. The Stepper Motor Control System with Encoder Feedback is a project designed to enhance the functionality of stepper motors by incorporating rotary encoders for position feedback.

The goal of this project is to develop a system that allows for precise control and real-time monitoring of multiple stepper motors using an Arduino Mega microcontroller. By integrating rotary encoders with the stepper motors, the system can accurately track the position of the motors and ensure their movements align with the desired parameters.

The system consists of two main components: **the hardware setup** and **the software implementation**. The hardware setup includes connecting the stepper motors to motor drivers, as well as integrating rotary encoders with the motors. The Arduino Mega microcontroller serves as the brain of the system, responsible for receiving user commands, controlling the motor movements, and processing the encoder feedback.

To achieve smooth and precise control of the stepper motors, the project utilizes the AccelStepper library, which provides advanced features such as acceleration, deceleration, and speed control. This library allows for precise positioning and enables the motors to move at different speeds and directions as commanded by the user.

The rotary encoders play a crucial role in the system by providing accurate position feedback. They detect the rotational movements of the motors and transmit signals to the Arduino, allowing real-time monitoring of their positions. The Arduino, through interrupt service routines (ISRs), interprets these signals and updates the position information accordingly.

The system provides a user-friendly interface through the Arduino Serial Monitor, allowing users to interact with the system and issue commands for motor control. Additionally, the Serial Plotter feature enables users to visualize the encoder states, aiding in the analysis and verification of the motor movements.

The Stepper Motor Control System with Encoder Feedback offers a versatile and adaptable solution for applications that require precise motor control and position feedback. Its ability to accurately track motor positions in real time enhances the overall control and performance of stepper motors in various applications.

1.2 Hardware used

Stepper motors (Nema 17) - 4

Motor drivers (A4988) – 4

Encoder (1024PPR ABZ 3-phase incremental optical rotary) - 2

Mega Arduino – 1

USB

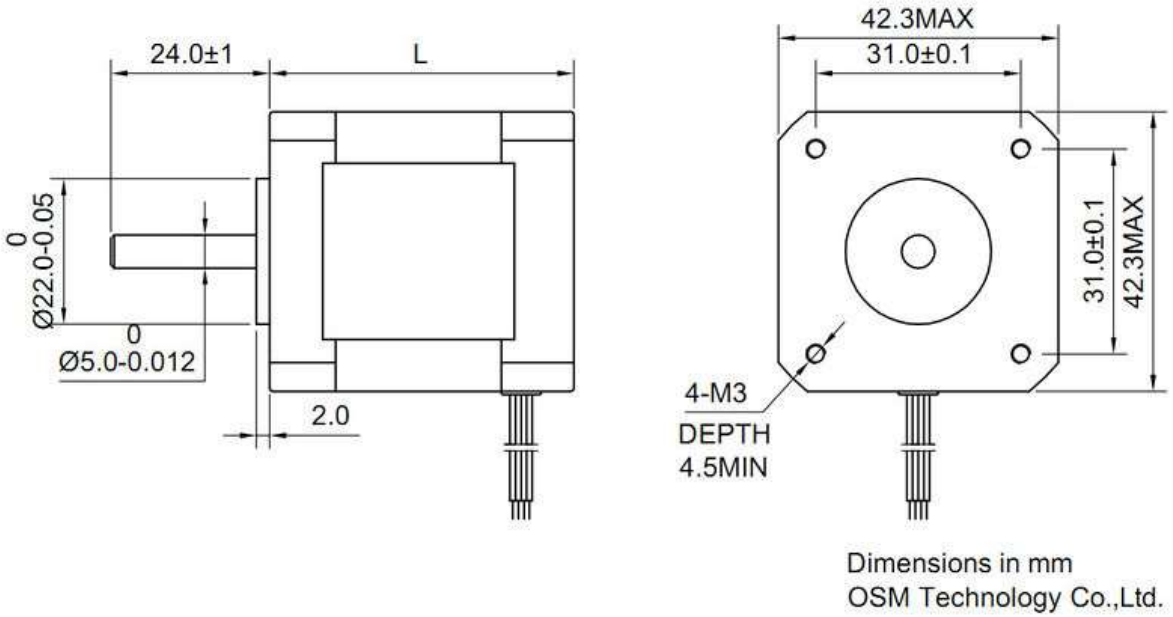
Jumper wires

Stepper motors (NEMA 17)

NEMA 17 stepper motors are known for their torque and reliability and it has a step angle of 1.8 degrees and a frame size of 42mm * 42mm, making it a compact and powerful option for precise motion control, they are widely used in 3D printers and CNC machines.

It works by dividing a full rotation into a number of equal steps, known as steps per revolution (SPR). By energizing its coils in a specific sequence, the motor shaft can be rotated in precise increments, making it an ideal choice for applications requiring precise positioning control, such as 3D printers and CNC machines.

Dimensions:



Dimensions in mm
OSM Technology Co.,Ltd.

Specifications:

- 1.5A to 1.8A current per phase
- 1-4 volts
- 3 to 8 MH inductance per phase
- 44 N·cm (62oz·in, 4.5kg·cm) or more holding torque
- 1.8 or 0.9 degrees per step (200/400 steps/rev respectively)

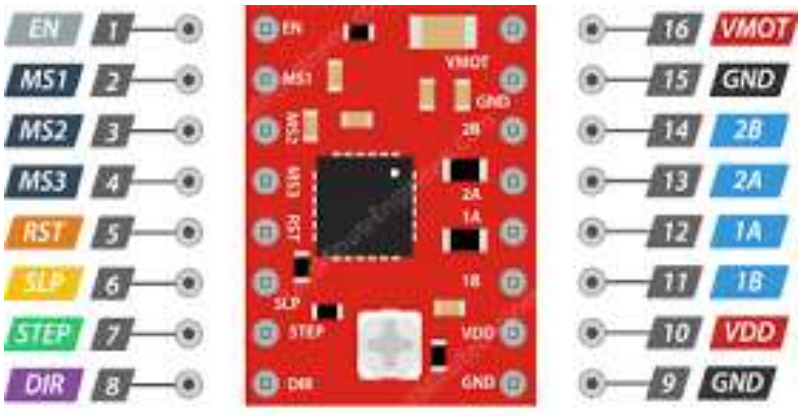
Motor drivers (A4988)

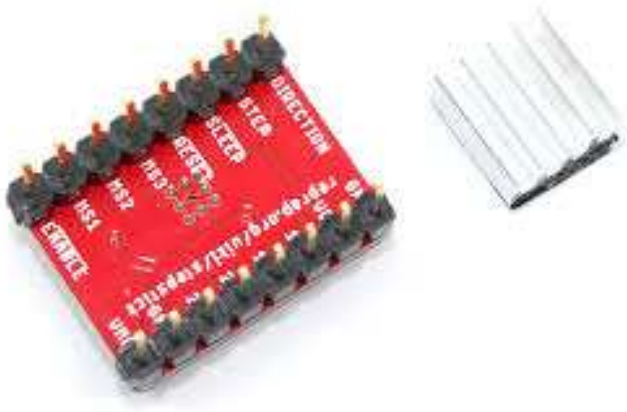
The A4988 driver Stepper Motor Driver is a complete micro-stepping motor driver with a built-in converter, easy to operate. It operates from 8 V to 35 V and can deliver up to approximately 1 A per phase without a heat sink or forced air flow (it is rated for 2 A per coil with sufficient additional cooling).
A4988 driver Stepper Motor Driver includes a fixed off-time current regulator, the regulator can be in slow or mixed decay mode. The converter is the key to the easy implementation of the A4988.
There are no phase sequence tables, high-frequency control interface programming, etc. The application of the A4988 interface is very suitable for a complex microprocessor that is not available or overloaded. In the stepping operation, the chopping control in the A4988 automatically selects the current decay mode (slow or mixed). The mixed decay current control scheme can reduce the audible motor noise, increase step accuracy, and reduce power consumption. Provide internal synchronous rectification control circuitry, in order to improve the pulse width modulation (PWM) power consumption during operation.

Internal circuit protection includes thermal shutdown with hysteresis, under-voltage lockout (UVLO), and crossover current protection. Don't need special power-up sequencing. A4988 uses a surface mount QFN package (ES), the size is 5 mm × 5 mm, nominal overall package height is 0.90 mm, with an exposed thermal pad to enhance the heat dissipation function.

Features:

- ✓ Automatic current decay mode detection/choice.
- ✓ Mixed with slow current decay mode.
- ✓ The low power dissipation of the synchronous rectifier.
- ✓ Internal UVLO (ultra voltage lockout).
- ✓ Crossover current protection.
- ✓ Thermal shutdown circuit.
- ✓ Ground fault protection.
- ✓ Loading and short circuit protection.
- ✓ The optional five-step mode: full, 1/2, 1/4, 1/8 and 1/16.





Specifications:

- ✓ Input Supply Voltage (VDC) - 8 ~ 35
- ✓ Operating Current (A) - 1.0A(No Heatsink), 2.0A (with Heat-Sinking)
- ✓ Logic Input (V) - 3 ~ 5.5
- ✓ Five-Step Mode - full, 1/2, 1/4, 1/8 and 1/16.
- ✓ Shipment Weight - 0.08 kg
- ✓ Shipment Dimensions - 5 × 5 × 4 cm

Encoder (1024PPR ABZ 3-phase incremental optical rotary:

A rotary encoder is a type of position sensor that is used for determining the angular position of a rotating shaft. It generates an electrical signal, either analog or digital, according to the rotational movement.

Orange 1024 PPR Incremental Optical Rotary Encoder is a high-resolution optical encoder with quadrature outputs for increment counting. It will give 4096 transitions per rotation between outputs A and B. whereas the Z phase will produce one transition per rotation. A quadrature decoder is required to convert the pulses to an up count. The Encoder is built to Industrial grade.

The Encoder comes with a Standard 2-meter-long cable which can be extended with extra cable if needed.

Note:

1. Consider Adding Pull up resistor to All A-B-Z phase of the encoder output to avoid the effect of interference in output and get the precise logical output value from the encoder
2. Also adding a pull-up resistor to all phase output lines protects the open collector output-triode from damage due to direct VCC supply short circuit.

Terminal Name	Wire Colour	Description
Phase A	White	Quadrature encoded output A
Phase B	Green	Quadrature encoded output B
Phase Z	Yellow	Quadrature encoded output Z
VCC	Red	VCC should be connected to +ve 5V of supply
GND	Black	The ground should be connected to the negative of the supply
Shield	Golden	The shield should be connected to GND

Power and Input Terminal Assignments:

Shielding is important. At *a minimum*, the cable should be protected either by a foil jacket with a drain wire or by a braided-wire shield that is grounded. For very sensitive applications or high-EMI environments, foil-jacketed wires in combination with an overall braided-wire shield around the cable should be used to avoid interference and precise encoder output.

Features:

- ✓ High cost-efficient advantages.
- ✓ Incremental rotary encoder internally adopts ASIC devices
- ✓ High reliability, long life
- ✓ Anti-jamming performance
- ✓ Small size, lightweight, compact structure
- ✓ Easy installation
- ✓ Stainless steel shaft, High resolution, High quality, line interface with waterproof protection

Specifications:

- ✓ Encoder Type - Incremental
- ✓ Operating Voltage (VDC) - 5 ~ 24
- ✓ Current Consumption (mA) - ≤40
- ✓ Cable Length (Meter) - 2
- ✓ Maximum Speed - 5000 RPM
- ✓ Pulse Per Revolution (PPR) - 1024
- ✓ Counts Per Revolution (CPR) - 4096

- ✓ Operating Temperature Range (°C) -24 to 84
- ✓ Output Phase - A-B-Z
- ✓ Protection Standard - IP54
- ✓ Shaft Type - D-type, Solid
- ✓ Shaft Length (mm) - 12
- ✓ Shaft Diameter (mm) - 6
- ✓ Mounting Hole(mm) - M3
- ✓ Weight (gm) - 160
- ✓ Body Dimensions (Dia. x Length) mm - 40 x 38.40
- ✓ Shipment Weight - 0.19 kg
- ✓ Shipment Dimensions - 10 × 8 × 6 cm

Mega Arduino:

Arduino board is an open-source microcontroller board which is based on Atmega 2560 microcontroller. The growth environment of this board executes the processing or wiring language. These boards have recharged the automation industry with their simple to utilize platform wherever everybody with small otherwise no technical backdrop can start by discovering some necessary skills to program as well as run the Arduino board. These boards are used to extend separate interactive objects otherwise we can connect to software on your PC like MaxMSP, Processing, and Flash. This article discusses an **introduction to Arduino mega 2560 board**, pin diagram and its specifications.

What is an Arduino Mega 2560?

The microcontroller board like "Arduino Mega" depends on the ATmega2560 microcontroller. It includes digital input/output pins-54, where 16 pins are analog inputs, 14 are used like PWM outputs hardware serial ports (UARTs) – 4,

a crystal oscillator-16 MHz, an ICSP header, a power jack, a USB connection, as well as an RST button. This board mainly includes everything which is essential for supporting the microcontroller. So, the power supply of this board can be done by connecting it to a PC using a USB cable, or battery or an AC-DC adapter. This board can be protected from the unexpected electrical discharge by placing a base plate.

Arduino-mega 2560-board

The SCL & SDA pins of Mega 2560 R3 board connects to beside the AREF pin. Additionally, there are two latest pins located near the RST pin. One pin is the IOREF that permit the shields to adjust the voltage offered from the Arduino board. Another pin is not associated & it is kept for upcoming purposes. These boards work with every existing shield although can adjust to latest shields which utilize these extra pins.

Arduino Mega Specifications

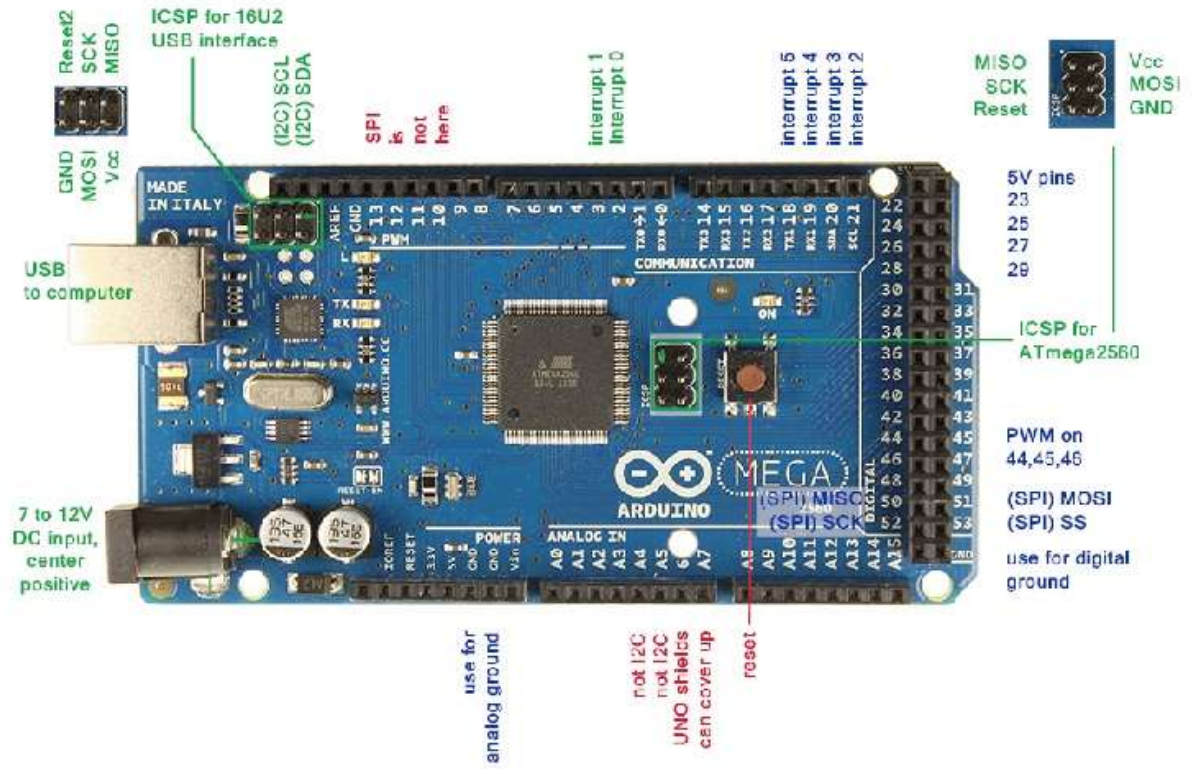
The specifications of Arduino Mega include the following.

- The ATmega2560 is a Microcontroller
- The operating voltage of this microcontroller is 5volts
- The recommended Input Voltage will range from 7volts to 12volts
- The input voltage will range from 6volts to 20volts
- The digital input/output pins are 54 where 15 of these pins will supply PWM o/p.
- Analog Input Pins are 16
- DC Current for each input/output pin is 40 mA
- DC Current used for 3.3V Pin is 50 mA
- Flash Memory like 256 KB where 8 KB of flash memory is used with the help of bootloader
- The static random access memory (SRAM) is 8 KB

- The electrically erasable programmable read-only memory (EEPROM) is 4 KB
- The clock (CLK) speed is 16 MHz
- The USB host chip used in this is MAX3421E
- The length of this board is 101.52 mm
- The width of this board is 53.3 mm
- The weight of this board is 36 g

Arduino Mega Pin Configuration

The pin configuration of this **Arduino mega 2560** board is shown below. Every pin of this board comes by a particular function which is allied with it. All analog pins of this board can be used as digital I/O pins. By using this board, the Arduino mega projected can be designed. These boards offer flexible work memory space is the more & processing power that permits to work with different types of sensors without delay. When we compare with other types of Arduino boards, these boards are physically superior.



Arduino-mega 2560-board-pin-diagram

Pin 3.3V & 5V

These pins are used for providing o/p regulated voltage approximately 5V. This RPS (regulated power supply) provides the power to the microcontroller as well as other components which are used over the Arduino mega board. It can be attained from Vin-pin of the board or one more regulated voltage supply-5V otherwise USB cable, whereas another voltage regulation can be offered by 3.3V0-pin. The max power can be drawn by this is 50mA.

GND Pin
The Arduino mega board includes 5-GND pins where one of these pins can be used whenever the project requires.

Reset (RST) Pin
The RST pin of this board can be used for rearranging the board. The board can be rearranged by setting this pin to low.

Vin Pin
The range of supplied input voltage to the board ranges from 7volts to 20volts. The voltage provided by the power jack can be accessed through this pin. However, the output voltage through this pin to the board will be automatically set up to 5V.

Serial Communication
The serial pins of this board like TXD and RXD are used to transmit & receive the serial data. Tx indicates the transmission of information whereas the RX indicates receive data. The serial pins of this board have four combinations. For serial 0, it includes Tx (1) and Rx (0), for serial 1, it includes Tx(18) & Rx(19), for serial 2 it includes Tx(16) & Rx(17), and finally for serial 3, it includes Tx(14) & Rx(15).

External Interrupts
The external interrupts can be formed by using 6-pins like interrupt 0(0), interrupt 1(3), interrupt 2(21), interrupt 3(20), interrupt 4(19), interrupt 5(18). These pins produce interrupts by a number of ways i.e. Providing LOW value, rising or falling edge or changing the value to the interrupt pins.

LED
This Arduino board includes a LED and that is allied to pin-13 which is named as digital pin 13. This LED can be operated based on the high and low values of the pin. This will give you to modify the programming skills in real time.

AREF
The term AREF stands for Analog Reference Voltage which is a reference voltage for analog inputs

Analog Pins
There are 16-analog pins included on the board which is marked as A0-A15. It is very important to know that all the analog pins on this board can be utilized like digital I/O pins. Every analog pin is accessible with the 10-bit resolution which can gauge from GND to 5 volts. But, the higher value can be altered using AREF pin as well as the function of analog Reference ().

I2C

The I2C communication can be supported by two pins namely 20 & 21 where 20-pin signifies Serial Data Line (SDA) which is used for holding the data & 21-pin signifies Serial Clock Line (SCL) mostly utilized for offering data synchronization among the devices

SPI Communication

The term SPI is a serial peripheral interface which is used to transmit the data among the controller & other components. Four pins like MISO (50), MOSI (51), SCK (52), and SS (53) are utilized for the communication of SPI.

Dimensions

The dimension of Arduino Mega 2560 board mainly includes the length as well as widths like 101.6mm or 4 inch X 53.34 mm or 2.1 inches. It is comparatively superior to other types of boards which are accessible in the marketplace. But, the power jack and USB port are somewhat expanded from the specified measurements.

Shield Compatibility

Arduino Mega is well-suited for most of the guards used in other Arduino boards. Before you propose to utilize a guard, confirm the operating voltage of the guard is well-suited with the voltage of the board. The operating voltage of most of the guards will be 3.3V otherwise 5V. But, guards with high operating voltage can injure the board.

In addition, the distribution header of the shield should vibrate with the distribution pin of the Arduino board. For that, one can connect the shield simply with the Arduino board & make it within a running state.

Programming

The programming of an Arduino Mega 2560 can be done with the help of an IDE (Arduino Software), and it supports C-programming language. Here the sketch is the code in the software which is burned within the software and then moved to the Arduino board using a USB cable.

An Arduino mega board includes a boot loader which eliminates an external burner utilization to burn the program code into the Arduino board. Here, the communication of the boot loader can be done using an STK500 protocol.

When we compile as well as burn the Arduino program, then we can detach the USB cable to remove the power supply from the Arduino board. Whenever you propose to use the Arduino board for your project, the power supply can be provided by a power jack otherwise Vin pin of the board.

Another feature of this is multitasking wherever Arduino mega board comes handy. But, Arduino IDE Software doesn't support multi-tasking however one can utilize additional operating systems namely RTX & FreeRTOS to write C-program for this reason. This is flexible to use in your personal custom build program with the help of an ISP connector.

1.3 Software used

Software **– Arduino IDE**

1. Arduino IDE:

Arduino IDE (Integrated Development Environment) is a software application used for programming Arduino microcontrollers. Arduino is an open-source electronics platform that provides a range of hardware boards and a programming language based on Wiring.

The Arduino IDE allows you to write and upload code to Arduino boards, making it possible to control and interact with various sensors, actuators, and other electronic components. It provides a simplified programming environment with a straightforward syntax and a library of functions for working with Arduino hardware.

The Arduino IDE supports a wide range of Arduino boards, making it accessible to beginners and experienced developers alike. It also has a large community of users and extensive documentation, making it easier to find examples, tutorials, and troubleshooting help.

2. Methodology

Hardware Setup:

- Connect the stepper motors to their respective motor drivers.
- Integrate rotary encoders with the stepper motors to provide position feedback.
- Connect the motor drivers and encoders to the Arduino Mega microcontroller using the specified pins.

Software Implementation:

- Include the AccelStepper library in the Arduino IDE to enable advanced control of the stepper motors.

- Define the necessary variables and constants for pin connections, motor parameters, and encoder states.
- Set up the pin modes and enable internal pull-up resistors for the encoder pins.
- Attach interrupt service routines (ISRs) to the encoder pins to handle changes in encoder states.
- Initialize the AccelStepper objects for each motor, specifying the motor interface type, step pin, and direction pin.
- Configure the initial motor settings such as maximum speed, acceleration, and current position.
- Set up the Serial communication at the desired baud rate to enable user interaction.

Motor Control Logic:

- Read the encoder states of motor1 and motor2 by reading the digital signals from the encoder pins.
- Send the encoder states to the Serial Plotter for visualization and analysis.
- Implement motor control logic using user commands received through the Serial Monitor.
- Based on user commands, move the motors in the desired direction or stop their movement using the move() and stop() functions provided by the AccelStepper library.
- Run the motors using the run() function to continuously update their position and adjust the speed based on the set parameters.
- Implement additional program logic as needed, keeping the loop() function as short and fast as possible.

User Interaction:

- Use the Arduino Serial Monitor as the interface for user commands and feedback.
- Accept user commands to start, stop, and change the direction of the motors.
- Process the received commands and control the motors accordingly.
- Implement commands for additional functionalities such as moving the motors in specific directions or performing specific tasks.

Motor Position Tracking:

- Utilize the interrupt service routines (ISRs) to update the position information of the motors based on the encoder signals.
- Maintain count variables to keep track of the motor positions, incrementing or decrementing the count based on the changes in encoder states.

- Set flags to indicate if the index (Z) pulse of the encoders has been triggered, providing a reference point for position tracking.

Motor Control Functions:

- Define functions for starting and stopping the motors, setting their speeds, and enabling or disabling the motor outputs.
- Implement functions to move the motors in specific directions or perform desired actions based on user commands.

Testing and Optimization:

- Test the system by sending different user commands and verifying the motor movements and position tracking.
- Debug any issues or inconsistencies in motor control or position tracking.
- Optimize the system performance by adjusting motor parameters, speed settings, or acceleration values to achieve the desired motor behavior.

Documentation and Deployment:

- Document the hardware connections, software implementation, and system functionalities for future reference.
- Prepare a user guide explaining how to interact with the system and utilize its features.
- Deploy the system in the intended application, considering any specific requirements or integration needs.

2.1 Algorithm

Initialize the system:

- Set up the required pin connections for the stepper motors, motor drivers, and rotary encoders.
- Configure the initial motor parameters such as maximum speed, acceleration, and current position.
- Enable internal pull-up resistors for the encoder pins.
- Attach interrupt service routines (ISRs) to the encoder pins for position tracking.

Start the Serial communication:

- Set the desired baud rate for communication.
- Begin the Serial communication to establish a connection with the Arduino Serial Monitor.

Enter the main loop:

- Read the encoder states for motor1 and motor2 by monitoring the digital signals from the encoder pins.
- Send the encoder states to the Serial Plotter for visualization and analysis.

Motor Control Logic:

- Check for user commands available in the Serial Monitor.
- Interpret the commands and execute the corresponding motor control actions.
- For motor1, check if the command is 'u' (up), 'd' (down), or 's' (stop).
- Move the motor up or down using the move() function with the appropriate direction and speed.
- Stop the motor using the stop() function.
- For motor2, check if the command is 'r' (right), 'l' (left), or 's' (stop).
- Move the motor right or left using the move() function with the appropriate direction and speed.
- Stop the motor using the stop() function.
- For general commands, check if the command is 'q' (start motors), 'w' (stop motors), or 'e' (move down).
- Execute the corresponding functions startMotors(), stopMotors(), or moveDown() to control motor3 and motor4.
- If none of the specific commands are detected, continue running the motors at their current speed and direction.

Run the motors:

- Check the motorsRunning flag to determine if the motors should be running.
- If motorsRunning is true, use the runSpeed() function to continuously update the positions and speeds of motor3 and motor4.

Additional program logic:

- Implement any additional program logic or functionality required for the specific application.

- Keep the loop() function as short and fast as possible to ensure efficient motor control and real-time response.

Handle Encoder Interrupts:

- Implement interrupt service routines (ISRs) to handle changes in encoder states and index pulses.
- For motor1, handleEncoderInterrupt1() updates the count1 variable based on changes in encoderPinA1 and encoderPinB1.
- For motor1, handleIndexInterrupt1() sets the indexTriggered1 flag when the index pulse on encoderPinZ1 is triggered.
- Similar functions are implemented for motor2 (handleEncoderInterrupt2() and handleIndexInterrupt2()).

Start, Stop, and Move Functions:

- Implement functions to startMotors(), stopMotors(), and moveDown() for controlling motor3 and motor4.
- Set appropriate speeds, directions, and pin inversions as needed.
- Enable and disable motor outputs using the enableOutputs() and disableOutputs() functions.

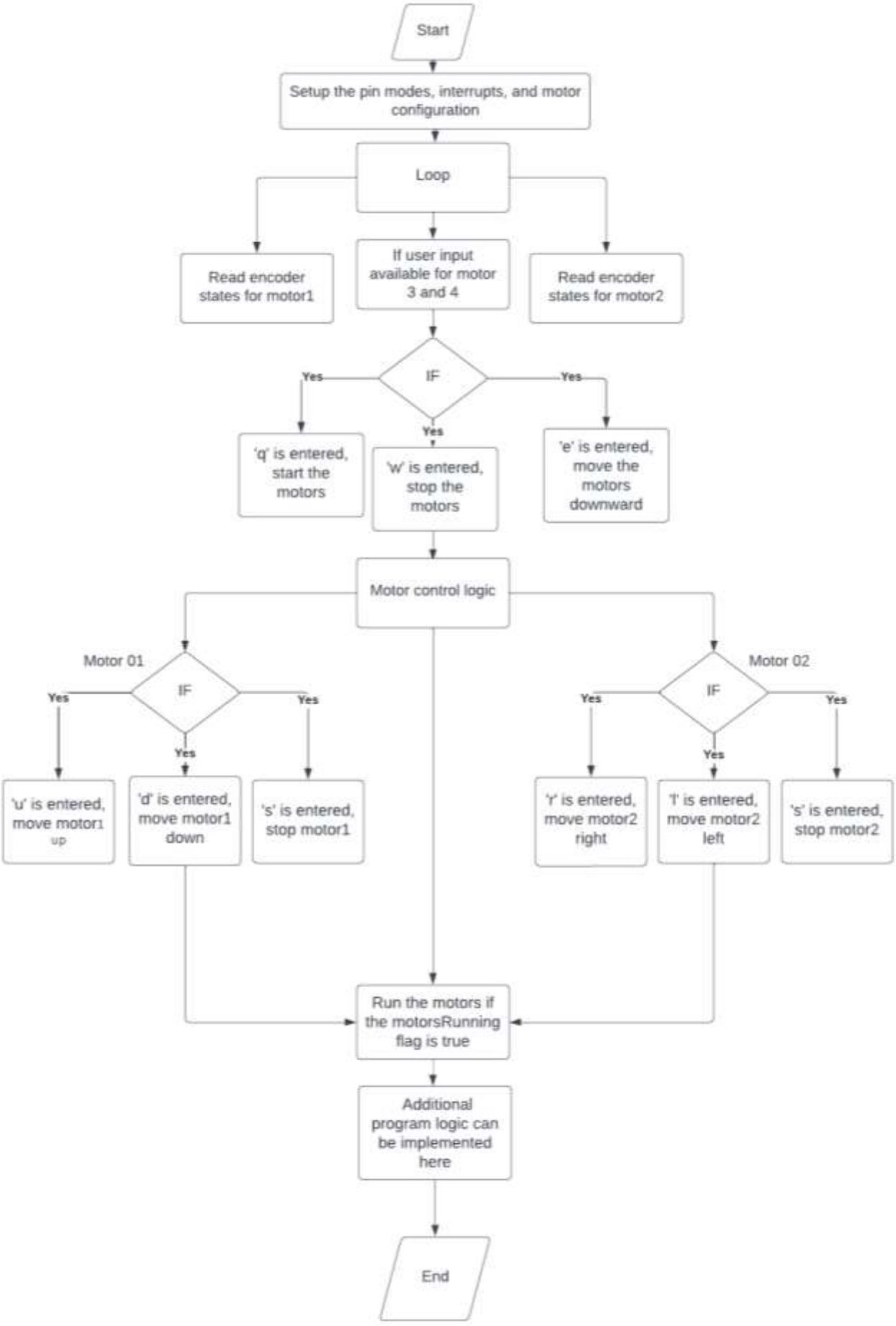
Testing and Optimization:

- Test the system by sending various user commands through the Serial Monitor.
- Verify the motor movements, position tracking, and functionality of the system.
- Debug and optimize the system performance by adjusting motor parameters, speed settings, or acceleration values.

Documentation and Deployment:

- Document the hardware connections, software implementation, and algorithm for future reference.
- Prepare a user guide explaining how to interact with the system and utilize its features.
- Deploy the system in the intended application, considering any specific requirements or integration needs.

2.2 Flow chart



3. Implementation

Gather the required hardware components:

- Arduino Mega or compatible board
- Stepper motors (at least two for continuous rotation and additional motors for other movements)
- Motor drivers (compatible with the stepper motors)
- Rotary encoders (compatible with the motors and capable of providing position feedback)
- Jumper wires and breadboard (for circuit connections)

Set up the circuit:

- Connect the stepper motors to their respective motor drivers, ensuring the correct pin connections for step and direction.
- Connect the motor drivers to the appropriate digital pins of the Arduino board.
- Connect the rotary encoders to the Arduino's digital pins, making sure to connect the encoder A and B channels correctly.
- Connect the index (Z) pulses of the encoders to the designated digital pins.

Install the required libraries:

- Download and install the AccelStepper library, which provides the necessary functions for controlling the stepper motors.
- Open the Arduino IDE, go to Sketch -> Include Library -> Manage Libraries, search for "AccelStepper," and install the library.

Write the Arduino code:

- Open a new sketch in the Arduino IDE.
- Copy and paste the provided code into the sketch.

Customize the code:

- Modify the pin definitions and motor parameters according to your specific hardware setup.
- Adjust the maximum speed, acceleration, and initial position values for each motor based on your requirements.

- Customize the motor control logic and user commands if needed.

Upload the code:

- Connect the Arduino board to your computer using a USB cable.
- Select the appropriate board and port in the Arduino IDE.
- Click on the "Upload" button to compile and upload the code to the Arduino board.

Open the Serial Monitor:

- Once the code is successfully uploaded, open the Serial Monitor in the Arduino IDE.
- Set the baud rate to match the Serial.begin() function in the code (9600 in this case).

Test the system:

- Enter commands in the Serial Monitor to control the motors.
- Verify that the motors respond correctly to the commands, such as moving up, down, left, right, or stopping.
- Observe the encoder states and index pulses displayed in the Serial Plotter.

Optimize and fine-tune:

- Adjust the motor parameters, such as speed and acceleration, to achieve the desired performance.
- Refine the motor control logic and user interface based on your project requirements.
- Perform additional testing and debugging to ensure smooth operation.

Document and deploy:

- Document the hardware connections, pin configurations, and any modifications made to the code.
- Prepare user documentation, including instructions on how to interact with the system and troubleshoot common issues.
- Deploy the system in your intended application, taking into account any necessary housing or enclosure requirements.

3.1CODE

Arduino:

Code:

Full Code:

```
#include <AccelStepper.h>
// Rotary encoder connections for motor1
const int encoderPinA1 = 49;
const int encoderPinB1 = 48;
const int encoderPinZ1 = 43;

// Rotary encoder connections for motor2
const int encoderPinA2 = 22;
const int encoderPinB2 = 26;
const int encoderPinZ2 = 24;
// Stepper motor connections for motor1
const int dirPin1 = 2;
const int stepPin1 = 3;
#define motorInterfaceType1 1
AccelStepper motor1(motorInterfaceType1, stepPin1, dirPin1);

// Stepper motor connections for motor2
const int dirPin2 = 10;
const int stepPin2 = 11;
#define motorInterfaceType2 1
AccelStepper motor2(motorInterfaceType2, stepPin2, dirPin2);

// Define pin connections & motor's steps per revolution for motor3 and motor4
const int dirPin3 = 12;
const int stepPin3 = 13;
const int dirPin4 = 14;
const int stepPin4 = 15;
const int stepsPerRevolution = 200;

bool motorsRunning = false;  // Flag to track if motors are running

AccelStepper motor3(AccelStepper::DRIVER, stepPin3, dirPin3);
AccelStepper motor4(AccelStepper::DRIVER, stepPin4, dirPin4);

// Variables to store encoder state for motor1
volatile int count1 = 0;
volatile bool indexTriggered1 = false;

// Variables to store encoder state for motor2
volatile int count2 = 0;
volatile bool indexTriggered2 = false;
```

```
void setup()
{
  // Set encoder pins as input for motor1
  pinMode(encoderPinA1, INPUT);
  pinMode(encoderPinB1, INPUT);
  pinMode(encoderPinZ1, INPUT);

  // Set encoder pins as input for motor2
  pinMode(encoderPinA2, INPUT);
  pinMode(encoderPinB2, INPUT);
  pinMode(encoderPinZ2, INPUT);

  // Enable internal pull-up resistors for the encoder pins for motor1
  digitalWrite(encoderPinA1, HIGH);
  digitalWrite(encoderPinB1, HIGH);
  digitalWrite(encoderPinZ1, HIGH);
  // Enable internal pull-up resistors for the encoder pins for motor2
  digitalWrite(encoderPinA2, HIGH);
  digitalWrite(encoderPinB2, HIGH);
  digitalWrite(encoderPinZ2, HIGH);

  // Attach interrupts to the encoder pins for motor1
  attachInterrupt(digitalPinToInterrupt(encoderPinA1),
handleEncoderInterrupt1, CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderPinB1),
handleEncoderInterrupt1, CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderPinZ1), handleIndexInterrupt1,
RISING);

  // Attach interrupts to the encoder pins for motor2
  attachInterrupt(digitalPinToInterrupt(encoderPinA2),
handleEncoderInterrupt2, CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderPinB2),
handleEncoderInterrupt2, CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderPinZ2), handleIndexInterrupt2,
RISING);

  // Set motor pin modes for motor1
  pinMode(dirPin1, OUTPUT);
  pinMode(stepPin1, OUTPUT);

  // Set motor pin modes for motor2
  pinMode(dirPin2, OUTPUT);
  pinMode(stepPin2, OUTPUT);

  // Set motor pin modes for motor3 and motor4
  pinMode(dirPin3, OUTPUT);
```

```
  pinMode(stepPin3, OUTPUT);
  pinMode(dirPin4, OUTPUT);
  pinMode(stepPin4, OUTPUT);

  // Set initial motor configuration for motor1
  motor1.setMaxSpeed(10000);
  motor1.setAcceleration(500);
  motor1.setSpeed(1000);
  motor1.setCurrentPosition(0);

  // Set initial motor configuration for motor2
  motor2.setMaxSpeed(10000);
  motor2.setAcceleration(500);
  motor2.setSpeed(1000);
  motor2.setCurrentPosition(0);

  // Set initial motor configuration for motor3
  motor3.setMaxSpeed(2000);
  motor3.setAcceleration(1000);

  // Set initial motor configuration for motor4
  motor4.setMaxSpeed(2000);
  motor4.setAcceleration(1000);

  // Start serial communication
  Serial.begin(9600);
}

void loop()
{
  // Read the encoder state for motor1
  int stateA1 = digitalRead(encoderPinA1);
  int stateB1 = digitalRead(encoderPinB1);
  int stateZ1 = digitalRead(encoderPinZ1);

  // Read the encoder state for motor2
  int stateA2 = digitalRead(encoderPinA2);
  int stateB2 = digitalRead(encoderPinB2);
  int stateZ2 = digitalRead(encoderPinZ2);

  // Send encoder states to Serial Plotter
  Serial.print(stateA1);
  Serial.print("\t");
  Serial.print(stateB1);
  Serial.print("\t");
  Serial.print(stateZ1);
  Serial.print("\t");
  Serial.print(stateA2);
```

```
Serial.print("\t");
Serial.print(stateB2);
Serial.print("\t");
Serial.println(stateZ2);

// Motor control logic for motor1
if (Serial.available() > 0) {
  char command = Serial.read();
  if (command == 'u') { // 'u' for "up"
    motor1.move(1);
  } else if (command == 'd') { // 'd' for "down"
    motor1.move(-1);
  } else if (command == 's') { // 's' for "stop"
    motor1.stop();
  }
}
motor1.run();

// Motor control logic for motor2
if (Serial.available() > 0) {
  char command = Serial.read();
  if (command == 'r') { // 'r' for "right"
    motor2.move(1);
  } else if (command == 'l') { // 'l' for "left"
    motor2.move(-1);
  } else if (command == 's') { // 's' for "stop"
    motor2.stop();
  }
}
motor2.run();

// Check for user input
if (Serial.available() > 0) {
  char command = Serial.read();

  // Start the motors
  if (command == 'q') {
    startMotors();
  }

  // Stop the motors
  if (command == 'w') {
    stopMotors();
  }

  // Move motors downward
  if (command == 'e') {
    moveDown();
```

```
    }
  }

  // Run the motors
  if (motorsRunning) {
    motor3.runSpeed();
    motor4.runSpeed();
  }

  // Additional program logic can go here
  // Note: Keep loop() as short and fast as possible
}

// Function to start the motors
void startMotors()
{
  motorsRunning = true;
  motor3.setSpeed(100);
  motor4.setSpeed(100);
  motor3.setPinsInverted(true);  // Invert the direction for motor3
  motor4.setPinsInverted(true);  // Invert the direction for motor4
  motor3.enableOutputs();
  motor4.enableOutputs();
}

// Function to stop the motors
void stopMotors()
{
  motorsRunning = false;
  motor3.setSpeed(0);
  motor4.setSpeed(0);
  motor3.runSpeedToPosition();
  motor4.runSpeedToPosition();
  motor3.disableOutputs();
  motor4.disableOutputs();
}

// Function to move motors downward
void moveDown()
{
  motorsRunning = true;
  motor3.setSpeed(100);  // Set negative speed for downward movement
  motor4.setSpeed(100);  // Set negative speed for downward movement
  motor3.setPinsInverted(false);  // Restore normal direction for motor3
  motor4.setPinsInverted(false);  // Restore normal direction for motor4
  motor3.enableOutputs();
  motor4.enableOutputs();
}
```

```
// Interrupt service routine for encoder of motor1
void handleEncoderInterrupt1()
{
  if (digitalRead(encoderPinA1) == digitalRead(encoderPinB1)) {
    count1++;
  } else {
    count1--;
  }
}

// Interrupt service routine for index of motor1
void handleIndexInterrupt1()
{
  indexTriggered1 = true;
}

// Interrupt service routine for encoder of motor2
void handleEncoderInterrupt2()
{
  if (digitalRead(encoderPinA2) == digitalRead(encoderPinB2)) {
    count2++;
  } else {
    count2--;
  }
}

// Interrupt service routine for index of motor2
void handleIndexInterrupt2()
{
  indexTriggered2 = true;
}
```

(Part _1)

Code for Two motors Rotating simultaneously:

```
#include <AccelStepper.h>

// Define pin connections & motor's steps per revolution
const int dirPin1 = 12;
const int stepPin1 = 13;
const int dirPin2 = 14;
const int stepPin2 = 15;
const int stepsPerRevolution = 200;

bool motorsRunning = false;  // Flag to track if motors are running
```

```
AccelStepper motor1(AccelStepper::DRIVER, stepPin1, dirPin1);
AccelStepper motor2(AccelStepper::DRIVER, stepPin2, dirPin2);

void setup()
{
  Serial.begin(9600);

  motor1.setMaxSpeed(2000);
  motor1.setAcceleration(1000);
  motor2.setMaxSpeed(2000);
  motor2.setAcceleration(1000);

  stopMotors();  // Initially stop the motors
}

void loop()
{
  // Check for user input
  if (Serial.available() > 0) {
    char command = Serial.read();

    // Start the motors
    if (command == 'q') {
      startMotors();
    }

    // Stop the motors
    if (command == 'w') {
      stopMotors();
    }

    // Move motors downward
    if (command == 'e') {
      moveDown();
    }
  }

  // Run the motors
  motor1.runSpeed();
  motor2.runSpeed();
}

// Function to start the motors
void startMotors()
{
  motorsRunning = true;
  motor1.setSpeed(100);
  motor2.setSpeed(100);
```

```
  motor1.setPinsInverted(true);  // Invert the direction for motor1
  motor2.setPinsInverted(true);  // Invert the direction for motor2
  motor1.enableOutputs();
  motor2.enableOutputs();
}

// Function to stop the motors
void stopMotors()
{
  motorsRunning = false;
  motor1.setSpeed(0);
  motor2.setSpeed(0);
  motor1.runSpeedToPosition();
  motor2.runSpeedToPosition();
  motor1.disableOutputs();
  motor2.disableOutputs();
}

// Function to move motors downward
void moveDown()
{
  motorsRunning = true;
  motor1.setSpeed(100);  // Set negative speed for downward movement
  motor2.setSpeed(100);  // Set negative speed for downward movement
  motor1.setPinsInverted(false);  // Restore normal direction for motor1
  motor2.setPinsInverted(false);  // Restore normal direction for motor2
  motor1.enableOutputs();
  motor2.enableOutputs();
}
```

Part -2

Code for right and left rotation with encoder:

```
#include <AccelStepper.h>
// Rotary encoder connections
const int encoderPinA = 22;
const int encoderPinB = 26;
const int encoderPinZ = 24;

// Stepper motor connections
const int dirPin = 10;
const int stepPin = 11;
#define motorInterfaceType 1
AccelStepper motor2(motorInterfaceType, stepPin, dirPin);

// Variables to store encoder state
volatile int count = 0;
```

```
volatile bool indexTriggered = false;

// Motor control variables
bool isMoving = false;
bool isMovingRight = false;
bool isMovingLeft = false;

void setup() {
  // Set encoder pins as input
  pinMode(encoderPinA, INPUT);
  pinMode(encoderPinB, INPUT);
  pinMode(encoderPinZ, INPUT);

  // Enable internal pull-up resistors for the encoder pins
  digitalWrite(encoderPinA, HIGH);
  digitalWrite(encoderPinB, HIGH);
  digitalWrite(encoderPinZ, HIGH);
  // Attach interrupts to the encoder pins
  attachInterrupt(digitalPinToInterrupt(encoderPinA), handleEncoderInterrupt,
CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderPinB), handleEncoderInterrupt,
CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderPinZ), handleIndexInterrupt,
RISING);

  // Set motor pin modes
  pinMode(dirPin, OUTPUT);
  pinMode(stepPin, OUTPUT);

  // Set initial motor configuration
  motor2.setMaxSpeed(10000);
  motor2.setAcceleration(500);
  motor2.setSpeed(1000);
  motor2.setCurrentPosition(0);

  // Start serial communication
  Serial.begin(9600);
}

void loop() {
  // Read the encoder state
  int stateA = digitalRead(encoderPinA);
  int stateB = digitalRead(encoderPinB);
  int stateZ = digitalRead(encoderPinZ);

  // Send encoder states to Serial Plotter
  Serial.print(stateA);
  Serial.print("\t");
```

```
  Serial.print(stateB);
  Serial.print("\t");
  Serial.println(stateZ);

  // Motor control logic
  if (Serial.available() > 0) {
    char command = Serial.read();
    if (command == 'r') { // 'r' for "right"
      isMovingRight = true;
    } else if (command == 'l') { // 'l' for "left"
      isMovingLeft = true;
    } else if (command == 's') { // 's' for "stop"
      isMovingRight = false;
      isMovingLeft = false;
      motor2.stop();
    }
  }
  if (isMovingRight && !isMovingLeft) {
    if (!motor2.isRunning()) {
      // Move one step right
      motor2.moveTo(motor2.currentPosition() + 1);
    }
    motor2.run();
  } else if (!isMovingRight && isMovingLeft) {
    if (!motor2.isRunning()) {
      // Move one step left
      motor2.moveTo(motor2.currentPosition() - 1);
    }
    motor2.run();
  }

  // Additional program logic can go here
  // Note: Keep loop() as short and fast as possible
}

void handleEncoderInterrupt() {
  int newStateA = digitalRead(encoderPinA);
  int newStateB = digitalRead(encoderPinB);

  if (newStateA != newStateB) {
    count++;
  } else {
    count--;
  }
}

void handleIndexInterrupt() {
  indexTriggered = true;
```

```
  // Additional logic for index pulse handling can be added
```

Part -3

Code for up and down rotation with encoder:

```
#include <AccelStepper.h>
// Rotary encoder connections
const int encoderPinA = 49;
const int encoderPinB = 48;
const int encoderPinZ = 43;

// Stepper motor connections
const int dirPin = 2;
const int stepPin = 3;
#define motorInterfaceType 1
AccelStepper motor1(motorInterfaceType, stepPin, dirPin);
// Variables to store encoder state
volatile int count = 0;
volatile bool indexTriggered = false;

// Motor control variables
bool isMoving = false;
bool isMovingUp = false;
bool isMovingDown = false;

void setup() {
  // Set encoder pins as input
  pinMode(encoderPinA, INPUT);
  pinMode(encoderPinB, INPUT);
  pinMode(encoderPinZ, INPUT);

  // Enable internal pull-up resistors for the encoder pins
  digitalWrite(encoderPinA, HIGH);
  digitalWrite(encoderPinB, HIGH);
  digitalWrite(encoderPinZ, HIGH);

  // Attach interrupts to the encoder pins
  attachInterrupt(digitalPinToInterrupt(encoderPinA), handleEncoderInterrupt,
CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderPinB), handleEncoderInterrupt,
CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderPinZ), handleIndexInterrupt,
RISING);

  // Set motor pin modes
  pinMode(dirPin, OUTPUT);
  pinMode(stepPin, OUTPUT);
```

```
  // Set initial motor configuration
  motor1.setMaxSpeed(10000);
  motor1.setAcceleration(500);
  motor1.setSpeed(1000);
  motor1.setCurrentPosition(0);

  // Start serial communication
  Serial.begin(9600);
}

void loop() {
  // Read the encoder state
  int stateA = digitalRead(encoderPinA);
  int stateB = digitalRead(encoderPinB);
  int stateZ = digitalRead(encoderPinZ);

  // Send encoder states to Serial Plotter
  Serial.print(stateA);
  Serial.print("\t");
  Serial.print(stateB);
  Serial.print("\t");
  Serial.println(stateZ);

  // Motor control logic
  if (Serial.available() > 0) {
    char command = Serial.read();
    if (command == 'u') { // 'u' for "up"
      isMovingUp = true;
    } else if (command == 'd') { // 'd' for "down"
      isMovingDown = true;
    } else if (command == 's') { // 's' for "stop"
      isMovingUp = false;
      isMovingDown = false;
      motor1.stop();
    }
  }

  if (isMovingUp && !isMovingDown) {
    if (!motor1.isRunning()) {
      // Move one step up
      motor1.moveTo(motor1.currentPosition() + 1);
    }
    motor1.run();
  } else if (!isMovingUp && isMovingDown) {
    if (!motor1.isRunning()) {
      // Move one step down
      motor1.moveTo(motor1.currentPosition() - 1);
```

```
    }
    motor1.run();
  }

  // Additional program logic can go here
  // Note: Keep loop() as short and fast as possible
}
void handleEncoderInterrupt() {
  int newStateA = digitalRead(encoderPinA);
  int newStateB = digitalRead(encoderPinB);

  if (newStateA != newStateB) {
    count++;
  } else {
    count--;
  }
}
void handleIndexInterrupt() {
  indexTriggered = true;
  // Additional logic for index pulse handling can be added
```

Part -4

Code for both motors rotation with encoder(rotatory):

```
#include <AccelStepper.h>

// Rotary encoder connections for motor1
const int encoderPinA1 = 30;
const int encoderPinB1 = 32;
const int encoderPinZ1 = 28;

// Rotary encoder connections for motor2
const int encoderPinA2 = 47;
const int encoderPinB2 = 55;
const int encoderPinZ2 = 43;

// Stepper motor connections for motor1
const int dirPin1 = 2;
const int stepPin1 = 3;
#define motorInterfaceType1 1
AccelStepper motor1(motorInterfaceType1, stepPin1, dirPin1);

// Stepper motor connections for motor2
const int dirPin2 = 10;
const int stepPin2 = 11;
```

```
#define motorInterfaceType2 1
AccelStepper motor2(motorInterfaceType2, stepPin2, dirPin2);

// Variables to store encoder state for motor1
volatile int count1 = 0;
volatile bool indexTriggered1 = false;

// Variables to store encoder state for motor2
volatile int count2 = 0;
volatile bool indexTriggered2 = false;

// Motor control variables for motor1
bool isMoving1 = false;
bool isMovingUp1 = false;
bool isMovingDown1 = false;

// Motor control variables for motor2
bool isMoving2 = false;
bool isMovingRight2 = false;
bool isMovingLeft2 = false;

void setup() {
  // Set encoder pins as input for motor1
  pinMode(encoderPinA1, INPUT);
  pinMode(encoderPinB1, INPUT);
  pinMode(encoderPinZ1, INPUT);

  // Set encoder pins as input for motor2
  pinMode(encoderPinA2, INPUT);
  pinMode(encoderPinB2, INPUT);
  pinMode(encoderPinZ2, INPUT);

  // Enable internal pull-up resistors for the encoder pins for motor1
  digitalWrite(encoderPinA1, HIGH);
  digitalWrite(encoderPinB1, HIGH);
  digitalWrite(encoderPinZ1, HIGH);

  // Enable internal pull-up resistors for the encoder pins for motor2
  digitalWrite(encoderPinA2, HIGH);
  digitalWrite(encoderPinB2, HIGH);
  digitalWrite(encoderPinZ2, HIGH);

  // Attach interrupts to the encoder pins for motor1
  attachInterrupt(digitalPinToInterrupt(encoderPinA1),
handleEncoderInterrupt1, CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderPinB1),
handleEncoderInterrupt1, CHANGE);
```

```
  attachInterrupt(digitalPinToInterrupt(encoderPinZ1), handleIndexInterrupt1,
RISING);

  // Attach interrupts to the encoder pins for motor2
  attachInterrupt(digitalPinToInterrupt(encoderPinA2),
handleEncoderInterrupt2, CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderPinB2),
handleEncoderInterrupt2, CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderPinZ2), handleIndexInterrupt2,
RISING);

  // Set motor pin modes for motor1
  pinMode(dirPin1, OUTPUT);
  pinMode(stepPin1, OUTPUT);

  // Set motor pin modes for motor2
  pinMode(dirPin2, OUTPUT);
  pinMode(stepPin2, OUTPUT);

  // Set initial motor configuration for motor1
  motor1.setMaxSpeed(10000);
  motor1.setAcceleration(500);
  motor1.setSpeed(1000);
  motor1.setCurrentPosition(0);

  // Set initial motor configuration for motor2
  motor2.setMaxSpeed(10000);
  motor2.setAcceleration(500);
  motor2.setSpeed(1000);
  motor2.setCurrentPosition(0);

  // Start serial communication
  Serial.begin(9600);
}

void loop() {
  // Read the encoder state for motor1
  int stateA1 = digitalRead(encoderPinA1);
  int stateB1 = digitalRead(encoderPinB1);
  int stateZ1 = digitalRead(encoderPinZ1);

  // Read the encoder state for motor2
  int stateA2 = digitalRead(encoderPinA2);
  int stateB2 = digitalRead(encoderPinB2);
  int stateZ2 = digitalRead(encoderPinZ2);

  // Send encoder states to Serial Plotter
  Serial.print(stateA1);
```

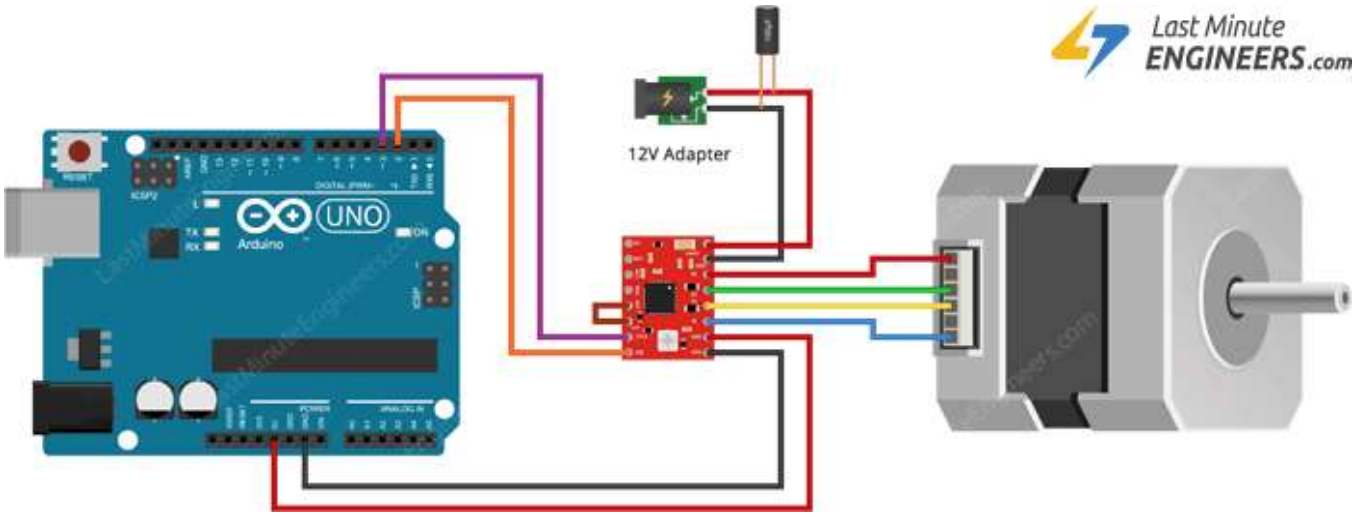
```
Serial.print("\t");
Serial.print(stateB1);
Serial.print("\t");
Serial.print(stateZ1);
Serial.print("\t");
Serial.print(stateA2);
Serial.print("\t");
Serial.print(stateB2);
Serial.print("\t");
Serial.println(stateZ2);

// Motor control logic for motor1
if (Serial.available() > 0) {
  char command = Serial.read();
  if (command == 'u') { // 'u' for "up"
    isMovingUp1 = true;
  } else if (command == 'd') { // 'd' for "down"
    isMovingDown1 = true;
  } else if (command == 's') { // 's' for "stop"
    isMovingUp1 = false;
    isMovingDown1 = false;
    motor1.stop();
  }
}

if (isMovingUp1 && !isMovingDown1) {
  if (!motor1.isRunning()) {
    // Move one step up
    motor1.moveTo(motor1.currentPosition() + 1);
  }
  motor1.run();
} else if (!isMovingUp1 && isMovingDown1) {
  if (!motor1.isRunning()) {
    // Move one step down
    motor1.moveTo(motor1.currentPosition() - 1);
  }
  motor1.run();
}

// Motor control logic for motor2
if (Serial.available() > 0) {
  char command = Serial.read();
  if (command == 'r') { // 'r' for "right"
    isMovingRight2 = true;
  } else if (command == 'l') { // 'l' for "left"
    isMovingLeft2 = true;
  } else if (command == 's') { // 's' for "stop"
    isMovingRight2 = false;
```

```
      isMovingLeft2 = false;
      motor2.stop();
    }
  }

  if (isMovingRight2 && !isMovingLeft2) {
    if (!motor2.isRunning()) {
      // Move one step right
      motor2.moveTo(motor2.currentPosition() + 1);
    }
    motor2.run();
  } else if (!isMovingRight2 && isMovingLeft2) {
    if (!motor2.isRunning()) {
      // Move one step left
      motor2.moveTo(motor2.currentPosition() - 1);
    }
    motor2.run();
  }

  // Additional program logic can go here
  // Note: Keep loop() as short and fast as possible
}

void handleEncoderInterrupt1() {
  int newStateA1 = digitalRead(encoderPinA1);
  int newStateB1 = digitalRead(encoderPinB1);

  if (newStateA1 != newStateB1) {
    count1++;
  } else {
    count1--;
  }
}

void handleIndexInterrupt1() {
  indexTriggered1 = true;
  // Additional logic for index pulse handling can be added here
}

void handleEncoderInterrupt2() {
  int newStateA2 = digitalRead(encoderPinA2);
  int newStateB2 = digitalRead(encoderPinB2);

  if (newStateA2 != newStateB2) {
    count2++;
  } else {
    count2--;
  }
```

```
}

void handleIndexInterrupt2() {
  indexTriggered2 = true;
  // Additional logic for index pulse handling can be added here
}
```

3.2 Circuit diagram:

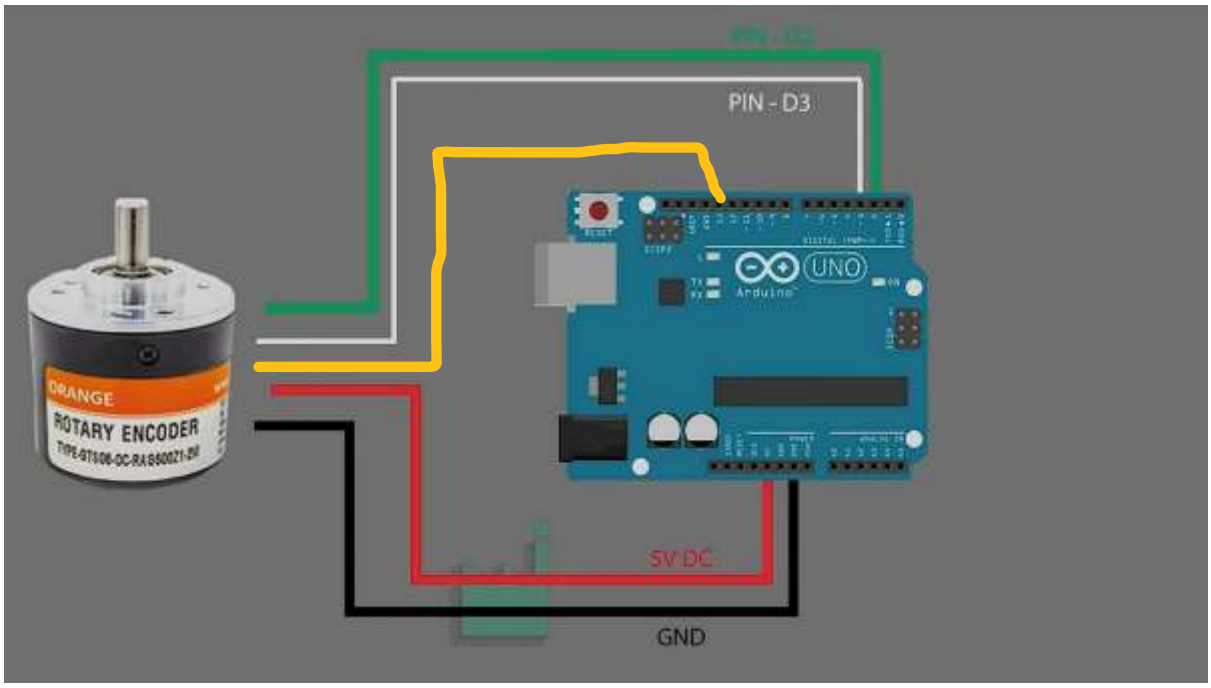
For sinle motor connection:

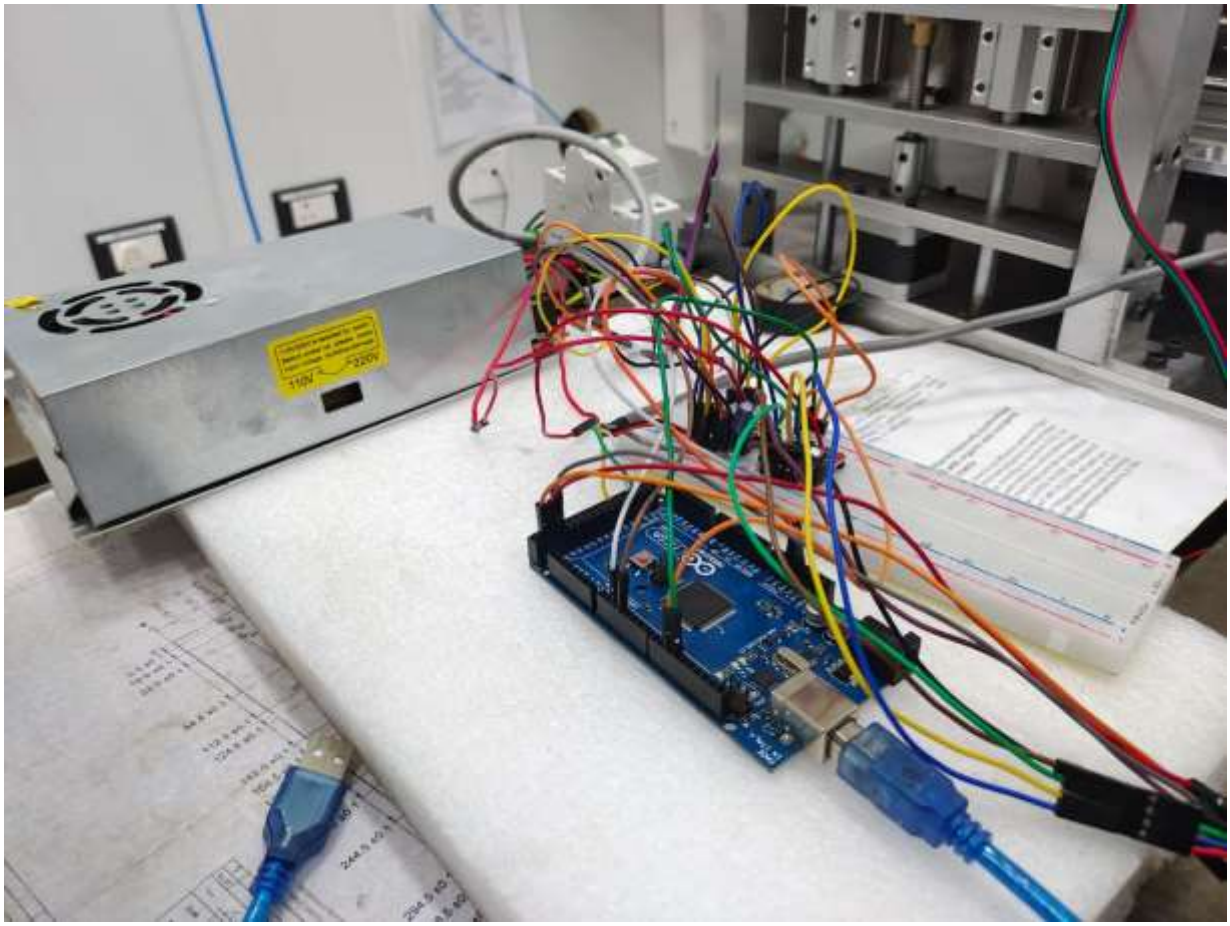


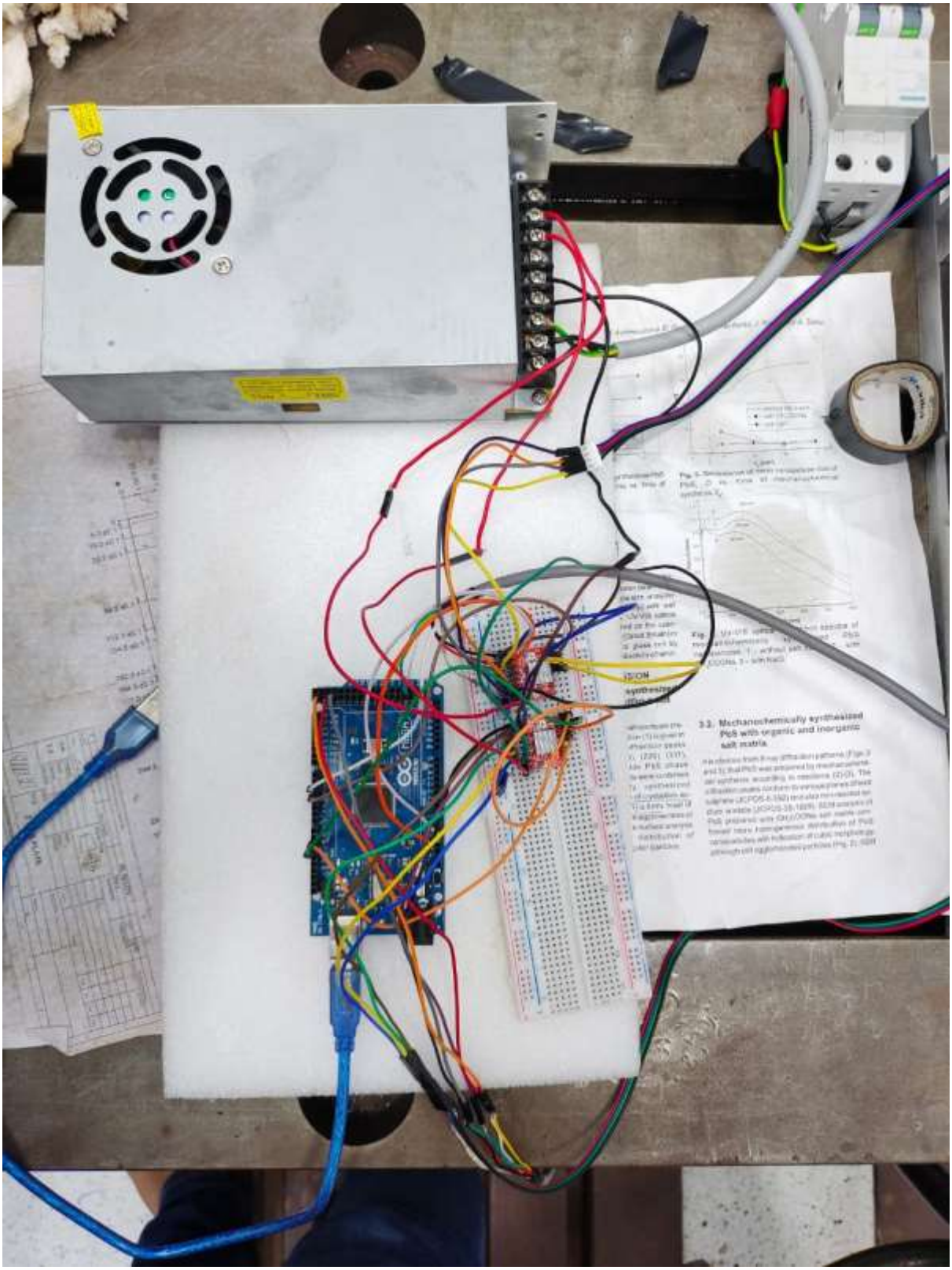
NOTE:

Same the above diagram we can do for all motors by taking each motor driver for it.

Encoder Circuit diagram:






3.2. Mechanochemically synthesized PbS with organic and inorganic salt matrix

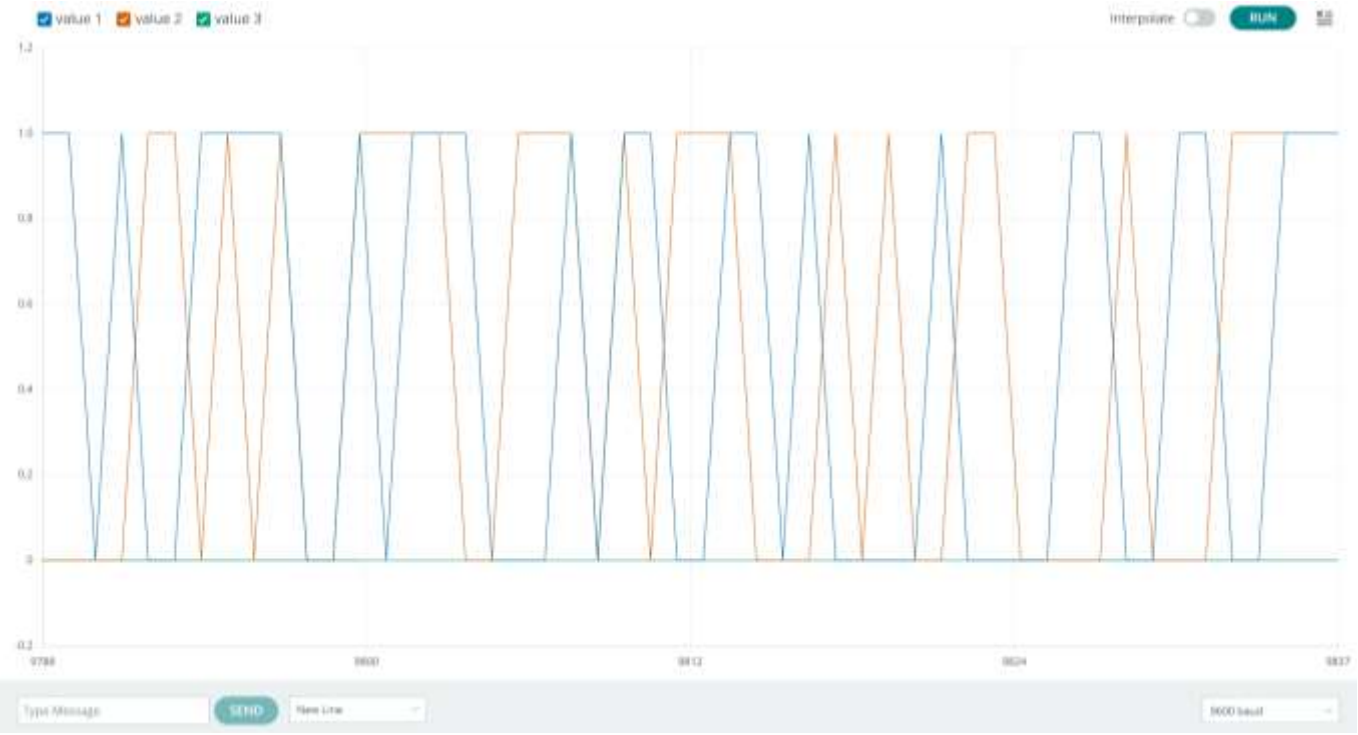
4. Results

Output in Serial monitor and Serial plotter:

Output in the serial monitor Up and down rotating motor with encoder:

```
0       1       0
1       0       0
1       0       0
0       0       0
0       1       0
1       0       0
0       1       0
1       0       0
0       0       0
1       0       0
1       1       0
1       1       0
0       1       0
```

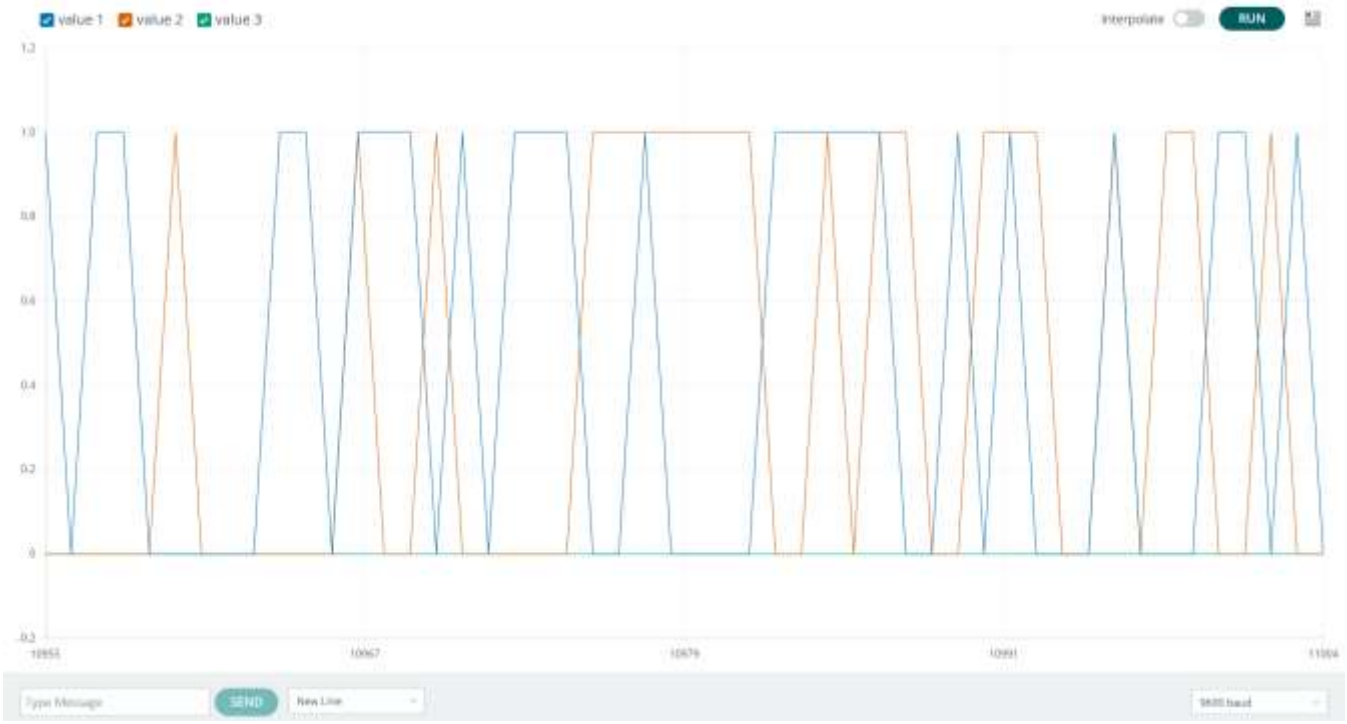
Output in the serial Plotter Up and down rotating motor with encoder:



Output in the serial monitor right and left rotating motor with encoder:

```
1       0       0
0       0       0
1       0       0
1       1       0
1       1       0
0       1       0
0       1       0
1       0       0
1       0       0
0       0       0
0       1       0
1       0       0
0       1       0
```

Output in the serial Plotter right and left rotating motor with encoder:



4.1 Discussions

Benefits of Stepper Motor Control:

- Stepper motors offer precise positioning and control, making them suitable for applications requiring accurate movements.
- The ability to control the motor's position and speed allows for precise and repeatable motion control.
- Stepper motors provide high torque even at low speeds, making them ideal for applications that require high holding torque or torque control.

Encoder Feedback:

- The inclusion of rotary encoders provides feedback on the motor's position, enabling accurate control and monitoring.
- Encoder feedback allows for closed-loop control, where the system can correct any position errors and maintain accuracy.
- By comparing the encoder feedback with the desired position, the system can detect and compensate for any motor slippage or missed steps.

Motor Control Logic:

- The code implements various motor control commands such as moving up, down, left, right, and stopping.
- User commands are received through the Serial Monitor, providing a convenient interface for controlling the motors.
- The use of the AccelStepper library simplifies motor control by providing functions for setting speed, acceleration, and position.

Multi-Motor Control:

- The system supports the control of multiple stepper motors simultaneously.
- Each motor can have its own configuration parameters and control commands.
- The code demonstrates the control of two continuous rotation motors and additional motors for other movements.

Interrupts and Index Pulses:

- Interrupts are used to handle the rotary encoder signals and index pulses.

- When an interrupt is triggered, the corresponding interrupt service routine updates the motor's position count.
- The index pulses provide a reference point for the motor's position and are used to detect a complete rotation or specific positions.

System Optimization:

- The code allows for customization and fine-tuning of motor parameters such as speed, acceleration, and direction.
- Users can adjust these parameters to achieve the desired performance and responsiveness.
- The system can be further optimized by considering factors such as motor resonance, load characteristics, and power requirements.

Application Potential:

- This system can be utilized in various applications such as robotics, CNC machines, 3D printers, automated positioning systems, and camera gimbals.
- The ability to control multiple motors with encoder feedback opens up possibilities for complex and precise motion control tasks.

Future Enhancements:

- The system can be expanded to include additional functionality such as limit switches for detection or sensor integration for environmental feedback.
- The code can be extended to support advanced motion profiles, smooth acceleration/deceleration, or trajectory planning algorithms

5. Conclusion

In conclusion, the Stepper Motor Control System with Encoder Feedback offers a reliable and precise solution for controlling stepper motors in various applications. By incorporating rotary encoders and utilizing interrupt-driven feedback, the system ensures accurate positioning and enables closed-loop control. The ability to control multiple motors simultaneously enhances versatility and expands the system's capabilities.

The implementation of the code, using the AccelStepper library and interrupt handling, simplifies motor control and enhances the user interface through the Serial Monitor. The system allows for the customization of motor parameters, optimizing performance for specific applications.

The discussed system has wide-ranging applications, including robotics, CNC machines, 3D printers, and other automated positioning systems. Its ability to achieve accurate and repeatable motion control opens up possibilities for intricate tasks requiring precise positioning.

Future enhancements may include the integration of additional features such as limit switches for end stop detection, sensor integration for environmental feedback, and advanced motion profiles or trajectory planning algorithms. These improvements would further enhance the system's capabilities and expand its potential for complex motion control applications.

Overall, the Stepper Motor Control System with Encoder Feedback provides a robust and flexible platform for controlling stepper motors with precision, accuracy, and reliability. Its implementation opens up possibilities for a wide range of applications that demand precise motion control and positioning.

6. References

- ✓ **Encoder connections: (Sample example for 3 phase rotary) through Arduino:**

  https://www.youtube.com/watch?v=Y6BjnfwfzKE

- ✓ **Motor connection through Arduino:**

  https://lastminuteengineers.com/a4988-stepper-motor-driver-arduino-tutorial/