

**TWO DIMENSIONAL (2D) STEPPER MOTOR CONTROLLER USING
JAVA PROGRAMMING**

ANITH KHAIRUNNISA BINTI GHAZALI

UNIVERSITI TEKNOLOGI MALAYSIA

UNIVERSITI TEKNOLOGI MALAYSIA

DECLARATION OF THESIS / UNDERGRADUATE PROJECT PAPER AND COPYRIGHT

Author's full name : ANITH KHAIRUNNISA BINTI GHAZALI

Date of birth : 28TH OCTOBER 1992

Title : TWO DIMENSIONAL (2D) STEPPER MOTOR CONTROLLER USING JAVA PROGRAMMING

Academic Session : **2014/2015**

I declare that this thesis is classified as :

- CONFIDENTIAL** (Contains confidential information under the Official Secret Act 1972)*
- RESTRICTED** (Contains restricted information as specified by the organisation where research was done)*
- OPEN ACCESS** I agree that my thesis to be published as online open access (full text)

I acknowledged that Universiti Teknologi Malaysia reserves the right as follows :

1. The thesis is the property of Universiti Teknologi Malaysia.
2. The Library of Universiti Teknologi Malaysia has the right to make copies for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by :

SIGNATURE

921028-14-5268
(NEW IC NO. /PASSPORT NO.)

Date : 7TH JUNE 2015

SIGNATURE OF SUPERVISOR

DR ABD RAHMAN BIN TAMURI
NAME OF SUPERVISOR

Date : 7TH JUNE 2015

**TWO DIMENSIONAL (2D) STEPPER MOTOR CONTROLLER USING
JAVA PROGRAMMING**

ANITH KHAIRUNNISA BINTI GHAZALI

A report submitted in partial fulfilment of the
requirements for the award of the degree of
Bachelor of Science (Industrial Physics)

Faculty of Science

Universiti Teknologi Malaysia

2015

“I here would like to state that I have read this thesis and in my opinion this thesis fulfils the scope and quality to be awarded a degree of Bachelor of Science (Industrial Physics).”

Signature : _____

Supervisor : DR ABD RAHMAN BIN TAMURI

Date : 28TH JUNE 2015

“I declare that this report entitled “Two Dimensional (2D) Stepper Motor controller using Java Programming” is the result of my own research except as cited in the references. The report has not been accepted for any degree and is not currently submitted in candidature of any other degree.”

Signature : _____

Name : ANITH KHAIRUNNISA BINTI GHAZALI

Date : 28TH JUNE 2015

To my mother, father, brothers and friends, thank you for your support.

“I love all of you”

ACKNOWLEDGEMENTS

Alhamdulillah and praise to Allah (S.W.T) who has guided me to complete the thesis. I would like to express my appreciation to my supervisor, Dr Abd Rahman Bin Tamuri for his professional advice, endless support and guidance during the process to develop this project. I would like to thanks all lecturers who have directly or indirectly help and share their knowledge with me to complete the project. I'm also very thankful to my friends who shared idea and opinion about this research. Lastly, special thanks to my beloved parents Encik Ghazali Bin Othman and Puan Che Norlida Binti Ismail for their prayers and moral support to make this project successful.

ABSTRACT

A 2D stepper motor controller based on Java programming was successfully designed to control two bipolar stepper motors. A Java program was developed to control a 3A driver board TB3-axis HY-TB3DV-N by using PIC 18F14K50. The designed Java program can control the forward and reverse movement of stepper motor for full step mode by setting the individual bits to zero as output and one as input. In full step mode, two phases are always 'on' and 'off' alternately to produce maximum rated torque. The controller was set to move the stepper motor in range from (0,0) to (540,0) for x axis and from (0,0) to (0,400) for y-axis. For positioning, Java will save the current position and calculate the step for next position using loop method and for reset action, two micro switches were placed at origin as initial point to keep the equipment operating safely. When physical contact occurs between stepper motor and micro switch, the logic 1 (5V) is set which will cause the circuit open and current position is set to (0,0). From the result, the frequency of stepper motor is 50Hz and the resolution is 1mm/pulse for the grid size 54 cm x 40 cm. The output square waveform will be seen on the display of oscilloscope and several information such as period, and duty cycle were analysed. For positioning, the result in experiment will be compared to from the actual measurement and the uncertainty was recorded. Minimum uncertainty for x-axis is -2mm and minimum uncertainty for y-axis is -3mm.

ABSTRAK

Pengawal motor pelangkah 2D telah berjaya direka menggunakan pengaturcaraan Java adalah reka bentuk untuk mengawal dua buah motor pelangkah bipolar. Pengaturcaraan Java telah dibangunkan untuk mengawal 3A papan penggerak TB3 HY-TB3DV-N dengan menggunakan PIC 18F14K50. Program Java yang direka boleh mengawal pergerakan motor pelangkah ke hadapan dan belakang untuk mod langkah penuh dengan menetapkan bit individu kepada sifar sebagai output dan satu sebagai input. Dalam mod langkah penuh, dua fasa buka dan tutup sentiasa berselang seli untuk menghasilkan tork yang maksimum. Pengawal ini telah direka untuk menggerakkan motor pelangkah dalam julat dari (0,0) ke (540,0) untuk paksi x dan dari (0,0) ke (0,400) bagi y paksi. Untuk mengawal kedudukan, Java akan menyimpan kedudukan semasa dan mengira langkah untuk kedudukan seterusnya menggunakan kaedah loop. Untuk set semula, kedudukan motor pelangkah dua suis mikro diletakkan di titik permulaan untuk memastikan peralatan beroperasi dengan selamat. Apabila sentuhan fizikal berlaku antara motor pelangkah dan suis mikro, logik 1 (5V) adalah sebagai penentu untuk memutuskan litar t dan set kedudukan semasa kepada (0,0). Dari hasil kajian, frekuensi motor pelangkah adalah 50Hz dan resolusi adalah 1mm/nadi untuk saiz grid 54 cm x 40 cm. Bentuk gelombang output persegi dapat dilihat pada paparan osiloskop dan beberapa maklumat seperti tempoh, dan kitar tugas telah dianalisis. Untuk kawalan kedudukan, hasil dalam eksperimen akan dibandingkan dengan ukuran sebenar dan ketidaktentuan direkodkan. Ketidaktentuan untuk paksi x ialah -2mm dan untuk paksi y ialah 3mm.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
	DECLARATION	ii
	ACKNOWLEDGEMENTS	v
	ABSTRACT	vi
	TABLES OF CONTENTS	vii
	LIST OF TABLE	x
	LIST OF FIGURES	xi
	LIST OF ABBREVIATIONS	xii
	LIST OF SYMBOLS	xiv
1	INTRODUCTION	
	1.1 Background of Research	1
	1.2 Problem Statement	2
	1.3 Objective of the Research	2
	1.4 Scope of the Research	3
	1.5 Significance of the Research	3
2	LITERATURE REVIEW	
	2.1 Introduction	4
	2.2 Stepper Motor	4
	2.3 Purpose of Controller	5
	2.4 Stepper Motor Driver	6

2.5	Types of Driver	7
2.6	Requirements of Driver	8
2.7	Full Stepping and Half Stepping	9
2.8	Micro Stepping	9
2.9	Unipolar and Bipolar	10
2.10	Stepper Motor System	11
2.11	Motor Characteristics	12
2.12	Advantages of Stepper Motor	13
3	RESEARCH METHODOLOGY	
3.1	Introduction	14
3.2	Experimental Setup	15
3.3	Instruments	16
3.4	Circuit Design	20
3.5	Java Programming	23
4	RESULTS AND DISCUSSION	
4.1	Results	29
4.2	Discussion	33
5	CONCLUSION	
5.1	Conclusion	38
5.2	Recommendation	39
	LIST OF REFERENCES	40-41
	APPENDICES	42-48

LIST OF TABLES

TABLE NO.	TITLE	PAGE
2.1	Electrical part for one cycle.	8
3.1	The definition of 1-PIN 25 of Parallel Interface	18
3.2	The connection between 1-PIN 25 of Parallel Interface with PIC18F14K50	20
4.1	Steps and average distance x	30
4.2	Steps and average distance y	31
4.3	Truth table	33
4.4	Comparison between Java Programming and hardware	36

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE
2.1	General arrangement of drive system	8
2.2	Unipolar stepper motor	11
2.3	Bipolar stepper motor	11
3.1	Block diagram of experimental setup	15
3.2	Micro switch	16
3.3	Stepper motor driver	17
3.4	Bipolar stepper motor	18
3.5	PIC18F14K50	19
3.6	Oscilloscope	19
3.7	Circuit block diagram	21
3.8	Computer, microcontroller and parallel interface block diagram	22
3.9	Flow chart of Java Programming	23
3.10	Coding for forward button	24
3.11	Coding for reverse button	25
3.12	Coding for reset button	26

3.13	Coding for check changes	27
3.14	Coding for positive changes	27
3.15	Coding for negative changes	27
4.1	Steps vs average distance x	30
4.2	Steps vs average distance y	31
4.3	Output for 10 ms	33
4.4	Output for 20 ms.	36

LIST OF ABBREVIATIONS

2D	-	Two dimensional
AC	-	Analog
ADC	-	Analog-to-Digital converter
DC	-	Direct current
Hz	-	Hertz
PIC	-	Programmable Intelligent Computer
Std Dev	-	Standard Deviation

LIST OF SYMBOLS

A	-	Ampere
f	-	Frequency
mm	-	Milimeter
ms	-	Milisecond
T	-	Time
V	-	Voltage

CHAPTER 1

INTRODUCTION

1.1 Background of Research

The stepper motor is a brushless motor that converts the digital pulses into mechanical shaft rotations. The number of rotation will be divided into a group number of steps. A separate pulse for each step will be sent by stepper motor.

Besides that, stepper motor is one type of the DC motor that moves in discrete steps. The motor will rotate one step at a time and at a same length. This will happen by energizing multiple coil called phase.

By using computer controlled stepping, precise positioning and speed control can be achieved. If the frequency of the digital pulse increase, a continuous rotation will convert the stepping movement with the velocity of the rotation directly proportional to control pulse frequency. The stepper motors are widely used in machines and devices due to some characteristics such as low cost, accuracy, compact size and high torque at low speeds.

1.2 Problem Statement

A two dimensional (2D) stepper motors are very important in order to get high precision reading of measurement. In this research, the Java program was used to develop a simple stepper motor controller that capable to control and positioning the stepper motor using microcontroller PIC18F14K50 and stepper motor driver TB3-axis HY-TB3DV.

1.3 Objective of the Research

The objectives of this study are:

- a. To control and positioning 2D stepper motor by develop a Java programming software system that can be used to control and display motion of stepper motor.
- b. To characterize the performance of the developed system.

1.4 Scope of the Research

The scope of this project will cover the concepts of DC stepper motor operation and the system to control the stepper motor by comparing with the Java programming. The Java programming will control the position and movement of stepper motor on the 540mm x 400mm grid. The microcontroller PIC18F14K50 and 3A of TB3-axis HY-TB3DV stepper motor driver with USB interface circuit are used during the development of the electronic circuit.

1.5 Significance of the Research

The small movement of stepper motor can be monitored by the developed software program. The performance can be characterized by monitoring the movement using Java program. By designing simple stepper motor driver and circuit, the movement of stepper motor can be verified.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter will discuss in details on the concepts of stepper motor driver and the system of the operation. In addition, the concepts of steps modes will be also deliberated.

2.2 Stepper Motor

According Jeff Keljik (2007), stepper motor create incremented step of motion rather than a smooth unbroken rotation. Basic concept of stepper motor explained using permanent magnet on rotor with two sets of poles. After DC pulses energize

the stator, the permanent magnet will be repelled or attracted to line up with the stator magnetic poles.

The important feature of the stepper motor is the way it revolve. The motor will revolve through a fixed angle for each pulse applied to the logic sequencer. When receive the step command pulse, logic sequence will determine either the phase to be energized or de-energized. Then, it send signals to stepper motor driver. If the output channel from logic sequencer is 'HIGH' the power works to excite the corresponding phase of the winding. At the phase of the same number will not excite or tuned off if the output is 'LOW'.

2.3 Purpose of Controller

According to Herman and Alerich (1990), some factor need to be considered when selecting and installing motor controller for use with machine or system. First factor when the motor stated by connecting directly across the source voltage. Slow and steady starting required to protect the machine and ensure the current inrush is not too large amount for the power system. Besides that, some driven machines might be damaged if they are started with rapid turning effort.

Next factor that need to be consider is during stopping. Usually, controllers allows stepper motor to glide to a standstill motor when stop. There are some controller force the braking action when the machine stop immediately. Speedy stopping is important function of the controller for emergency stops. Controllers help the stopping action by delaying the motion of stepper motor and lowering operation of pulley.

Another factor for selecting and installing motor controller are reversing and running occurrence. During reversing, controllers require to change the rotation of stepper motor automatically or by operator command at a control panel. For reversing, the main purpose and function of controller to maintain the speed of operation and their characteristics. Controller will keep the motors, machine and instrument safe while running.

2.4 Stepper Motor Driver

The function of the stepper motor driver is to control the movement of the stepper motor. The stepper motor driver will receive steps and signals of directions from a control system using computer. According to Giorgos Lazaridis (2010), electrical power will be converted into mechanical power by using stepper motor. The difference between stepper motor and other motors are the way they work. Stepper motor does not rotate continuously and each step is a fraction of a full circle. Every step of the stepper motor shaft needs a pulse. Usually, a standard 200 steps motor will need 200 steps to complete one revolution of the stepping motor shaft in full step. The stepper motor shaft rotation and speed is directly proportional to the frequency of the pulse.

From the stepper motor driver, current will flow to the winding, so, the speed and torque of a stepper motor can be determined. The inductance will reduce the time taken by current during energizing the winding process. Basically, stepper motor

driver circuit is designed to supply large amount of voltage rather than the stepper motor rated voltage. If the stepper motor driver gives higher output voltage, this will lead to higher level of torque versus speed. The output voltage from stepper motor driver also known as a bus voltage. Besides that, it should be rated five to ten times higher than the stepper motor voltage rating. The stepper motor controller current should be limited to the stepper motor current rating in order to protect the stepper motor.

2.5 Types of Driver

According to Austin Hughes (1990), there are three types of driver that are commonly used in industrial sector such as constant voltage drive, current forced drive and chopper drive. All types of driver need a transistor which act as switches either turned fully on or being cut off. Constant voltage driver is the simplest type of driver. The driver provides a reasonably good approximation to a rectangular current waveform at low stepping rate. At higher frequency, the 'on' period is short compared to the winding time constant and the current waveform will degenerates. When the pull out torque speed drop rapidly, it means that the motor is limited to low speed operation. For the type of current-forced driver, they need a higher supply voltage to increase the rate of rise of current during switch-on state. An additional resistor has to be added in series with the winding to prevent the current from exceeding the rated value. Chopper driver used a high power voltage with two switching transistor attached to it at every phase. First transistor is turned on for the whole period during which current is required. Then, the second transistor will turn on when actual current drop below the threshold.

2.6 Requirements of Driver

Basically, the complete driver functions as a converter to convert command input signals into appropriate patterns of current in motor windings.

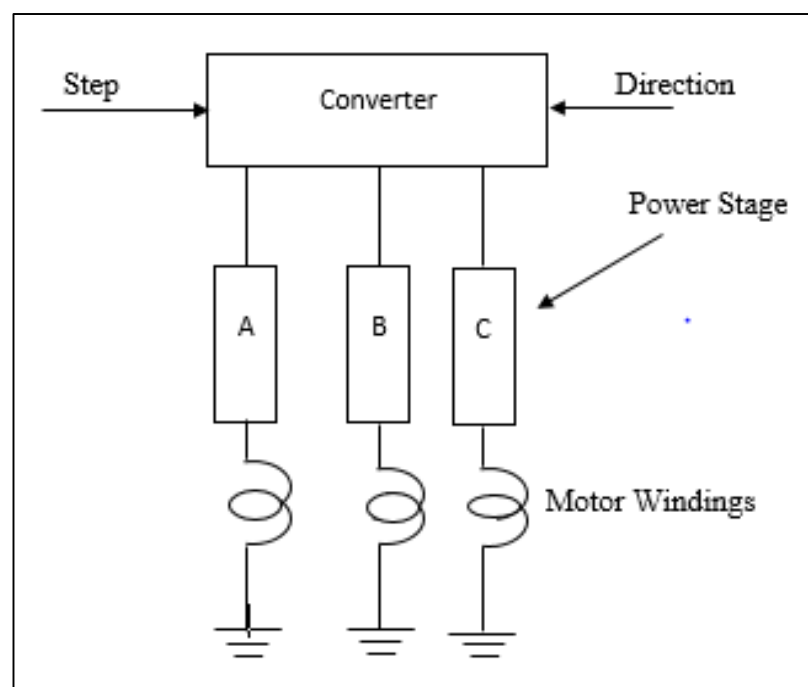


Figure 2.1: General arrangement of drive system

The converter changes the step command pulses into 'on' and 'off' signal at each of three power stage. During this stages, current will supply to the windings. When converter energised a phase, the current should established immediately. Then current will maintain constant during the 'on' period. When the converter calls to turn off, it should be reduced to zero immediately (Austin Hughes,1990).

2.7 Full Stepping and Half Stepping

Table 2.1: Electrical part for one cycle.

Electric polarity	Winding A	+	+		-	-	-		+
	Winding B		+	+	+		-	-	-
Electrical cycle part	Full-stepping	1		2		3		4	
	Half-stepping	1	2	3	4	5	6	7	8

The Table 2.1 shows that the full stepping mode powers one winding at one time. Takashi Kenjo (2001) states that this situation has cause four different possible position during full stepping. If both of the winding were powered, the stepper motor will be trapped between the positions and this event is known as half stepping. Besides that, by powering winding simultaneously gives 8 position as shown in ‘Half stepping row’. This action will also cause the torque 1.4 times higher than powering one winding but the cause power consumption are twice.

2.8 Micro Stepping

According to Bob Parente (2011), the function of a driver is to drive the stepper motor in micro step motion. Each phase will receive different amounts of currents to force the motor to make a step. One pulse for full stepping is 1.8° of rotation while for half stepping is 0.9° of rotation. Micro stepping can be calculated by dividing the natural full step of motors with the smaller increment. The formula is shown as below:

$$\text{Micro steps} = \text{Pulse/ seconds}$$

The micro stepping will cause the motor to run as smooth as possible. For a smooth motor, the driver will send two sine waves at 90 degrees at a phase. The results of motor rotation will be quite perfect if two step coil can be made to follow these sine waves. The two waves are functioning to keep the motor in a smooth transition from pole to pole. If the current decreases in one coil, it will increase in another coil and this will lead to an advance smooth step and output torque at each position are passed continuously.

2.9 Unipolar and Bipolar

A stepper motor that operates in one winding and a centre tap per phase was called as unipolar type of motor. Every single section of the winding will be switched on for each direction of the magnetic field. The winding will be switched on for each direction of the magnetic pole that can reverse the direction of current without switching it.

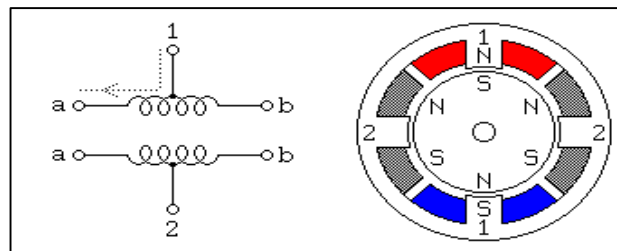


Figure 2.2: Unipolar stepper motor

M. Fauzi (2010) states that wired unipolar motor is as shown in Figure 2.2 above with a center tap of two windings. Positive supply wire is connected to the center tap and the two ends of each winding alternately grounded to reverse the direction of the field provided by windings.

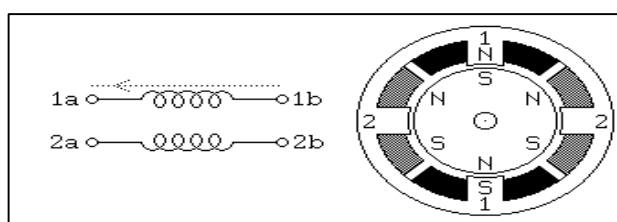


Figure 2.3: Bipolar stepper motor

According to William Yeadon (2001), a bipolar stepper motor is exactly the same as a unipolar stepper except the two windings are wired more simply and there are no center taps. Usually a bipolar motor is wired as shown in the schematic diagram. The driving circuit will be more complex to switch the magnetic pole for converting the current in the winding. A bipolar motor has two leads per phase and it is quite difficult to operate compared to a unipolar motor because the feature of wire is twice in the same space.

2.10 Stepper Motor System

A stepper motor can be controlled by a computer, microprocessor, or programmable controllers. The output shaft rotates in a series of discrete angular intervals or steps. When one step is taken at a time, a command pulse is received. After

a definite number of pulse has been supplied, the shaft will turned through a known angle, and this makes the motor suits ideally for an open loop position control. Each step can be completed in a few milliseconds. When a large number of steps is set for the step, command pulse will deliver rapidly, sometimes as fast as several thousand steps per second. At a high stepping rate, the shaft rotation becomes smooth and works like an ordinary motor (Austin Hughes, 1990).

2.11 Motor Characteristics

Herman and Alerich (1990) state that two phase synchronous motor have the ability to virtually start, stop or reverse direction of rotation instantly. The motor starts about one and half cycle of the applied voltage and stop within 5 to 25 millisecond. There are no induced current in rotor and no high inrush of current occurs when motor is started because the rotor is a permanent magnet. The amount current applied during starting and running are same. The motor did hold the received torque when it is turned off due to the permanent magnet structure. If more holding torque is needed, DC voltage can be used to one or both windings when motor is turned off. If DC applied only one windings, holding torque will be applied at approximately 20% greater than rated torque. While, if applied to both windings, holding torque will be one and half time greater than rated torque.

2.12 Advantages of Stepper Motor

Stepper motor works as a digital electromechanical device where each electrical pulse input resulted from the movement of the rotor by a discrete angle called step-angle of motor. Alexandru Morar (2013) states that there are several advantages of stepper motor. Firstly, the relationship between rotations angles of the motor is proportional to the input pulse. Secondly, the motor has full torque at standard, even when the winding are energized. Next, stepper motors have precise positioning and memorize the position. Lastly, the motor is simple and require less cost to control because it is compatible with digital technique.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This chapter is about the clarification of research methodology that ensures this research will complete and function very well. As stated before, the main purpose of this research is to design a stepper motor controller by using Java programming. The proper procedures were applied in order to achieve the objectives of the research and generate good results. In this chapter the instruments used, programming and the information of the controller is also elaborated and explained. Data were collected throughout the whole process of the research by applying procedures given.

3.2 Experimental Setup

Before the experiment is carried out, a proper setup is required to minimize the difficulties and mistakes. The sequence of setup is shown in the figure below.

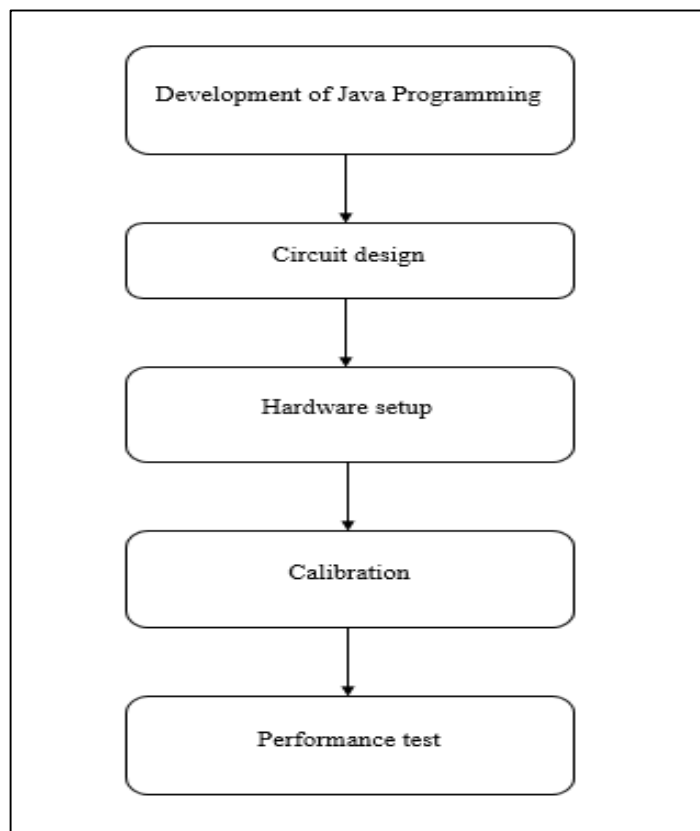


Figure 3.1: Block diagram of experimental setup

Figure 3.1 clarifies the sequence of action to be taken into consideration before the experiment was carried out. The first step is the development of Java programming that serves as the controller of the stepper motor for forward, reverse, positioning and reset motion. The second step is the designing of electrical circuit by using computer, microcontroller (PIC18F14K50), driver (HY-TB3DV) and stepper motor. The third

step is the hardware setup. At this stage, the size of grid, maximum and minimum movement of stepper motor for y-axis and x-axis were checked. The next step in the experimental setup is the calibration stage. During this stage, the initial position of stepper motor was set at (0,0) on the grid and also in the software. Besides that, the scale on the hardware and software are calibrated. The last step to be taken is to test the performance of the stepper motor by checking the step, direction and distance moved by the stepper motor.

3.3 Instruments

Several instruments was used in this experiment such as micro switch, stepper motor driver, bipolar stepper motor, PIC18F14K50, and oscilloscope. The details about instrument will discuss in this subtopic.



Figure 3.2: Micro switch

Figure 3.2 shows the limit micro switch that was used in this research. The micro switch consists of one electromechanical actuator that detect any contact. When the stepper motor meets the actuator, the micro switch triggers to break electrical connection. Besides that, micro switch also functioned as an emergency device to prevent machinery from being malfunctioned.

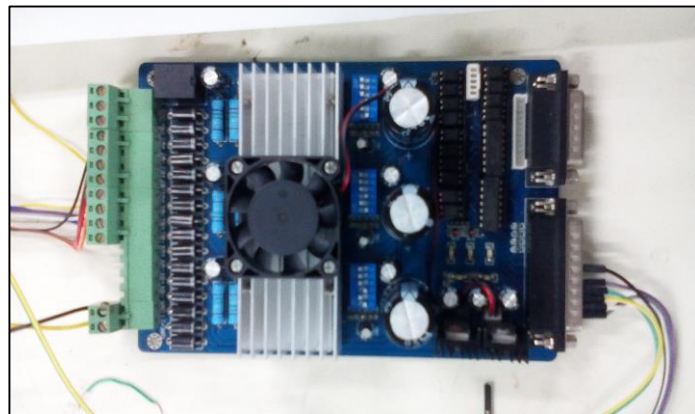


Figure 3.3: Stepper motor driver

The 3A driver board TB3-axis HY-TB3DV-N in Figure 3.3 was installed with Toshiba TB6560 Stepper Motor Driver PIC and this component was used in this research as part of the stepper motor driver. This device is a PWM chopper-type stepping motor driver PIC that was designed for controlling the micro step sinusoidal-input for bipolar stepping motor. TB6560 can be used for 2-phase, 1-2 phase, 2 winding 1-2 phase and 4 windings 1-2 phase excitation modes. Besides that, it is also capable to produce low vibration, high performance reverse and forward driving of two phase bipolar stepping motor by using clock signal. During handling TB6560, environment must be protected against electrostatic discharge because these ICs are highly sensitive to electrostatic discharge. Other component such as micro switches were used in this research to detect the presence and absence of stepper motor by physical contact. These devices are good in applications because they require simple on and off actions, so, they were used to reset position of the stepper motor.

Table 3.1: The definition of 1-PIN 25 of Parallel Interface

PIN 9	PIN 14	PIN 7	PIN 1	PIN 2	PIN 3	PIN 8	PIN 6	PIN 4	PIN 5	PIN 16	PIN 17
Spindle motor	X Enable	X Dir	X Step	Y Enable	Y Dir	Y Step	Z Enable	Z Dir	Z Step	Expand output1	Expand output2

Table 3.1 shows the definition of 1-PIN 25 Parallel Interface. This Parallel Interface was attached to the 3A driver board TB3-axis HY-TB3DV-N stepper motor driver.

**Figure 3.4:** Bipolar stepper motor

Two bipolar stepper motor in Figure 3.4 with 1.8° of rotation were used for x-axis and y-axis in this research. During operation, the current flow for x-axis motor is 0.65A and the current flow for y axis is 1A. The stepper motor will rotate continuously when 12V DC voltage is applied and the input pulse is converted into shaft increment. Every single pulse will rotate the shaft at specific angle. Besides that, PIC 18F14K50 will energize the electromagnet to make the shaft turn.

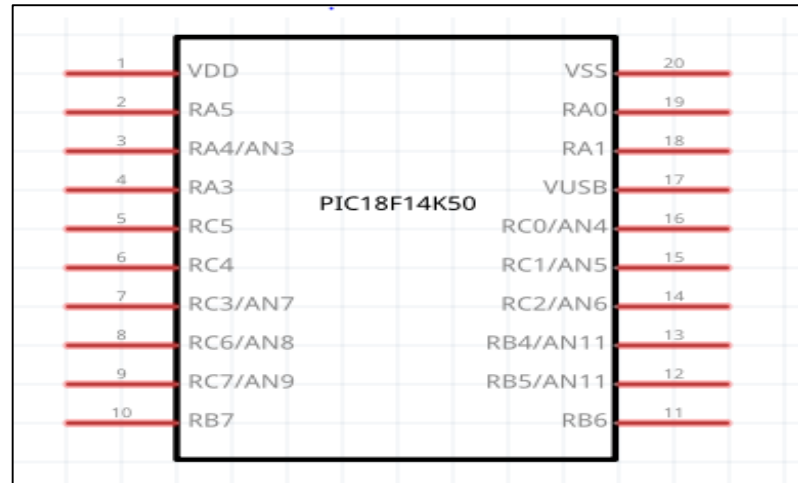


Figure 3.5: PIC18F14K50

The ‘Peripheral Interface Controller’ (PIC) in Figure 3.5 can be programmed to carry out a wide range of task in electrical field. Usually, the PIC is found in various electronic devices such as alarm system and phones. A compact microcomputer was designed to direct the operation in machines. Basically, a microcontroller consists of processor, memory and peripheral. PIC18F14K50 has three digital PORT A, B and C. RB 4 – RB 7 from PORT B is set for y-axis and RC0-RC7 from PORT C is set for x-axis. The microcontroller will connect to 25-PIN Parallel Interface on the drive for Enable, Direction and Step signals. The direction of stepper motor is determined when the signal at logic ‘HIGH’ (5V). The rotation clockwise or counter-clockwise will depend on how the motor phase are wired.

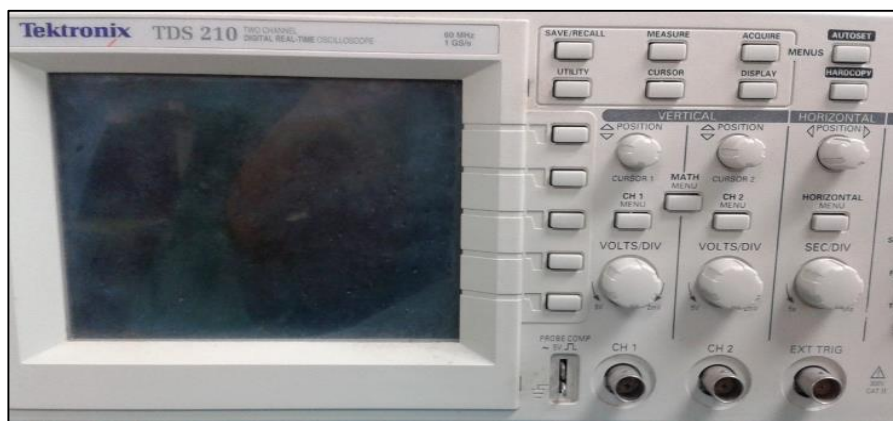


Figure 3.6: Oscilloscope

The oscilloscope in Figure 3.6 was used to obtain the output results. Most oscilloscope produce a two dimensional graph. Time parameter will represent on x-axis. While, voltage parameters represent on the y-axis. In addition, oscilloscope can detect the malfunction of circuit by determining the frequency and amplitude. Furthermore, oscilloscope also can identify the noise and the shape of waveform. In this project, oscilloscope was used to observe output signal produce when the stepper motors running.

3.4 Circuit Design

The connection between computer, microcontroller and stepper motor driver was determine before design the circuit.

Table 3.2: The connection between 1-PIN 25 of Parallel Interface with PIC18F14K50

	Signals	Parallel Interface	PIC18F14K50
x-axis	Enable	PIN 14	PC 0
	Direction	PIN 7	PC 1
	Steps	PIN 1	PC 2
y axis	Enable	PIN 2	PB 4
	Direction	PIN 3	PB 5
	Steps	PIN 8	PB 6

Table 3.2 shows the Pin involved between PIC18F14K50 and Parallel Interface. The PIC18F14K50 that attached to the driver board TB3-axis HY-TB3DV was programmed with Java programming to make it work.

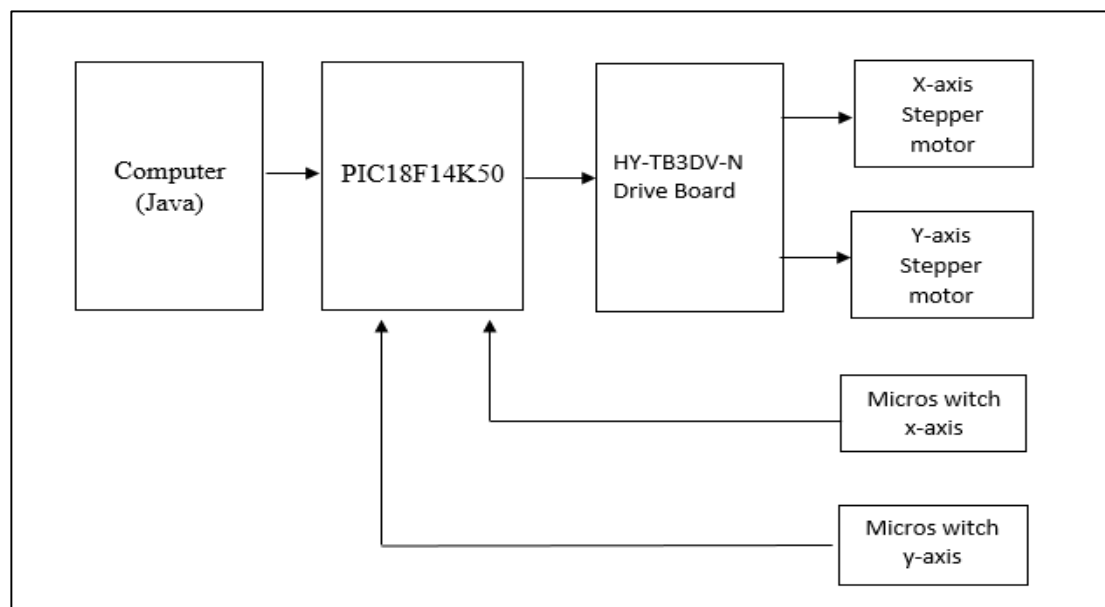


Figure 3.7: Circuit block diagram

Figure 3.7 shows how the stepper motor controller circuit was connected. From the Java Program, the signal will be sent to the PIC18F14K50 microcontroller. The selected pin for PIC18F14K50 was set for stepper motor of x-axis and y-axis. Then, the PIC18F14K50 will be connected to the driver board TB3-axis HY-TB3DV by Parallel Interface. Two stepper motor for y-axis and x-axis were connected directly to the driver and two micro switch were connected to the PIC18F14K50.

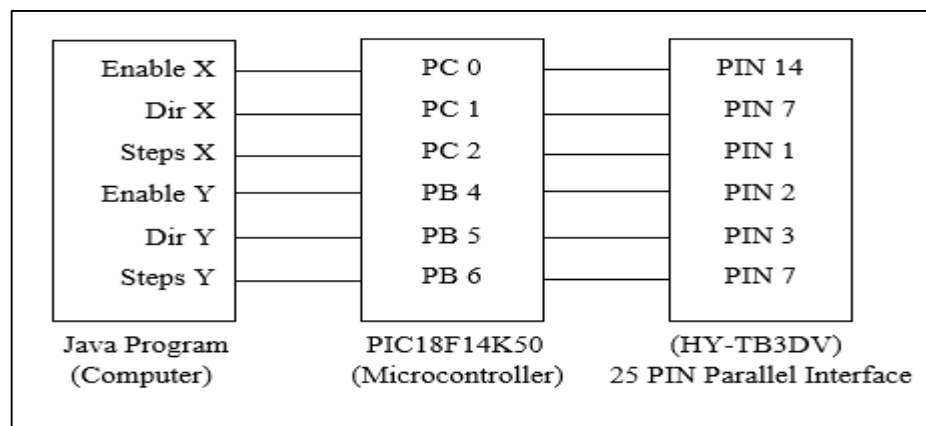


Figure 3.8: Computer, microcontroller and parallel interface block diagram

The connection between computer, microcontroller and parallel interface was shown in Figure 3.8. Java programming will give command through PORT C for x-axis and PORT B for y-axis. For x-axis stepper motor, PC0 was set to Enable signal, PC1 was set to Direction signal and PC2 was set to Step signal. While, for y-axis, PB4 was set to Enable, PB5 to Direction signal and PB6 was set to Step signal. Next, the PIC18F14K50 will connect to Parallel Interface according to their pin in the definition of 1-PIN 25 Parallel Interface.

3.5 Java Programming

Java programming is used to construct the command to control the stepper motor for forward and backward direction. This program is also able to reset and position the stepper motor to the selected point. The sample command will be shown below and the complete command can be referred in APPENDIX A. Java programming will display the position of stepper on the interface. The sequence to control the stepper motor was planned before developed the coding.

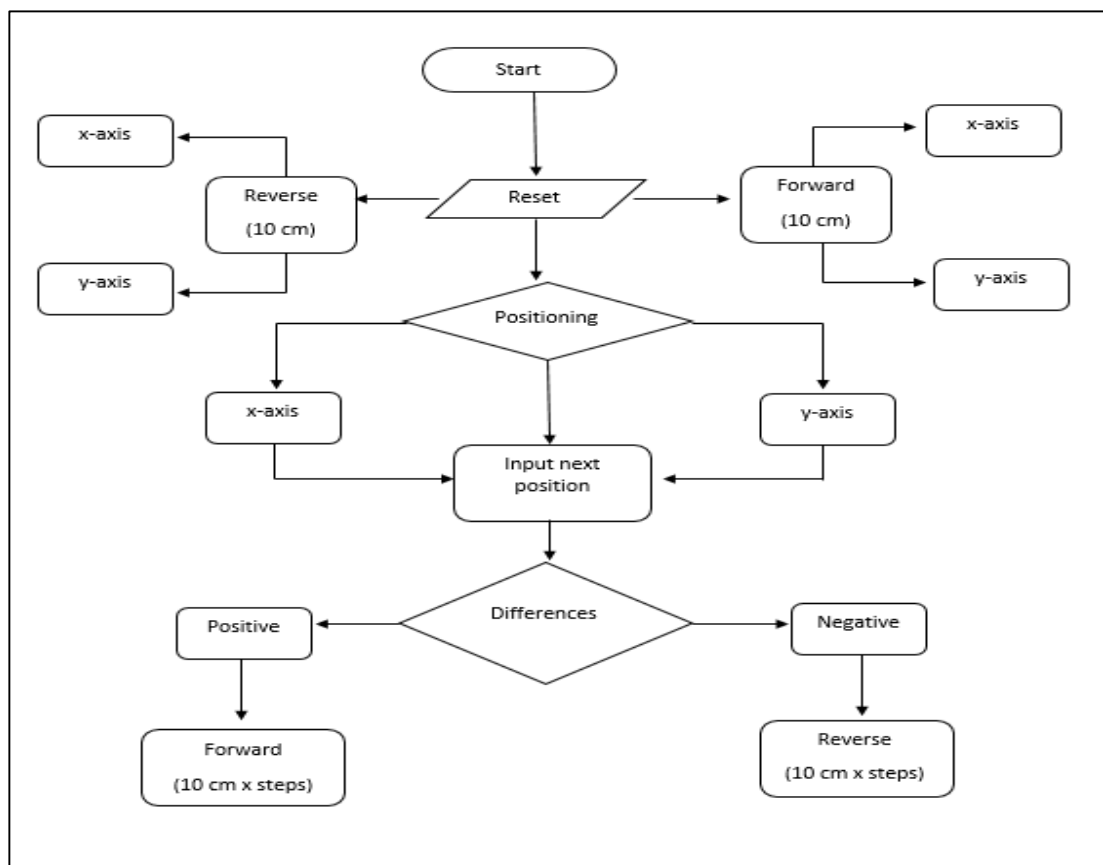


Figure 3.9: Flow chart of Java Programming

Figure 3.9 shows the flow of sequence for Java Programming. Before the stepper motor operate it reset to initial point. Then operator may choose either to run positioning mode or not. The direction of stepper motor during this mode will be determine by the input given of next position. If operator choose the positioning, the axis of stepper motor is selected and the operator need to put the next position. If the operator does not choose the positioning mode, they may choose the stepper motor of axis x or y and the direction of stepper motor.

```

if (currentpostX <=540){ //Maximum X position
try{
int i;
com.sendString ("setpc0\r",0);//High Enable
com.sendString ("respc1\r",0);// Low Direction

for (i =0; i<100; i++){//100 pulse (10 cm)
com.sendString ("setpc2\r",0);
Thread.sleep(10);
com.sendString ("respc2\r",0);//Low Steps
Thread.sleep(10);
}
currentpostX=currentpostX +i/10; //Add up with current position
X_location.setText (""+currentpostX);
repaint();
}catch (Exception e){System.out.println("Device not found");}
}

```

Figure 3.10: Coding for forward button

The forward program as shown in Figure 3.10 above will perform the command if the current position is less or equal to 540. PORT C is used for x-axis and 3 pins were used for Enable, Step and Direction signals. The PC0 is set to Enable pin and is always set to 'HIGH' signal, while PC1 is for Direction pin and is set to 'LOW' signal. Step pin was set at PC2 and will be always 'on' and 'off' alternatingly for every 10 ms. For y-axis, the program will perform if current position less or equal to 400 and

the Enable pin is connected to PB 4 and is always set to 'HIGH' signal. Direction pin will be connected to PB5 and gives 'LOW' signal. The Step pin for y-axis is connected to PB6 and the 'on' and 'off' signal will be also set alternatingly. Both axis will add up the current position to get the next position.

```

if (currentpostX >=10 ){//Minimum X position
try{
  repaint();
  int i;
  com.sendString ("setpc0\r",0);//High Enable
  com.sendString ("setpc1\r",0);// High Direction

  for (i =0; i<100 ;i++){//100 pulse (10 cm)
    com.sendString ("setpc2\r",0);
    Thread.sleep(10);
    com.sendString ("respc2\r",0);//Low Steps
    Thread.sleep(10);
  }
  currentpostX=currentpostX -i/10;//Minus with current position
  X_location.setText(""+currentpostX);
  repaint();
}
catch (Exception e){System.out.println("Device not found");}
}

```

Figure 3.11: Coding for reverse button

Figure 3.11 above shows the coding for reverse action that will be executed if the current position is more or equal to 10 for both axis. The difference between forward and reverse is the direction signal which is PC1 for x-axis PB5 for y-axis. In reverse command, PC1 and PB5 is set to 'HIGH' and this will rotates the motor counter clockwise. To get to the next position, pulse will be divided by ten and will be subtracted with the current position.

```

//-----X RESET-----
try{

com.sendString ("setpc0\r",0);//High Enable
com.sendString ("setpc1\r",0);//High Direction
    repaint();
int data ;

do {
    com.sendString ("setpc2\r",0);//High Step
    Thread.sleep(10);
    repaint();
    com.sendString ("respc2\r",0);//Low Step
    Thread.sleep(10);
    com.sendString("RD8\r",0);//X switch connection
    data = Integer.parseInt(com.receiveToString('\n', 0));//ADC

    data++;|
    repaint();
} while (data < 1000);//5V
    repaint();
{ com.sendString ("respc0\r",0);} ;//Low Enable
currentpostX = 0;
    repaint();
X_location.setText(""+ currentpostX );
    repaint();
}catch (Exception e){System.out.println("Device not found");}

```

Figure 3.12: Coding for reset button

The reset command was designed as in Figure 3.12 above in order to set the current position of stepper motor to the initial point (0,0). Two micro switches will detect the presence of stepper motor for x-axis and y-axis. The switch was connected to PIC18F14K50 microcontroller at RD6 for x-axis and RD7 for y-axis. The program will read the analog signal input and convert into digital signal when stepper motor triggered the micro switch.

```

if ((position==1)){//Calculate different
int postX= stpInpX -currentpostX;
int postY= stpInpY -currentpostY;
repaint();
}

```

Figure 3.13: Coding for check changes

Figure 3.13 shows the command to check the different position to determine either forward or reverse movement. The positive differences will give forward movement and the negative differences will give reverse motion.

```

//-----Positive X-----
if ((postX> 0) &(currentpostX <=540)){//Check different and position

repaint();

try{
int i;
com.sendString ("setpc0\r",0);//High Enable
com.sendString ("respc1\r",0);//Low Direction
repaint();
for (i =0; i<10*postX; i++){//Time with Step
com.sendString ("setpc2\r",0);// High Step
Thread.sleep(10);
repaint();
com.sendString ("respc2\r",0);//Low Step
Thread.sleep(10);
repaint();
}
currentpostX=currentpostX +i/10; //Add up with current position
X_location.setText (""+currentpostX);
repaint();
}catch (Exception e){System.out.println("Device not found");}
}

```

Figure 3.14: Coding for positive changes

After calculating the differences, Java programming will solve the forward command as shown in Figure 3.14 when the changes is positive. It will checked the current position and do the command. The differences will be multiplied by ten to convert into pulse.

```

//-----Negative X-----
    repaint();
if ((postX< 0)&(currentpostX >=10)){ //Check different and position
    repaint();
    try{

        int i;
        com.sendString ("setpc0\r",0);//High Enable
        com.sendString ("setpc1\r",0);//High Direction
        repaint();
        for (i =0; i<(-10)*postX ;i++){//Time with Step (negative steps)
            com.sendString ("setpc2\r",0);// High Step
            Thread.sleep(10);
            com.sendString ("respc2\r",0);// Low Step
            Thread.sleep(10);
            repaint();
        }
        currentpostX=currentpostX -i/10; //Minus with current position
        X_location.setText(""+currentpostX);
        repaint();
    }
    catch (Exception e){System.out.println("Device not found");}
}
    repaint();

```

Figure 3.15: Coding for negative changes

Coding in Figure 3.15 will be accomplished if the differences were negative and the current position has exceed or equal to ten, so the reverse command will be functional. As in forward position, the differences will be also multiplied with negative 10 to convert into pulse and this will change the pulse to a positive value.

CHAPTER 4

RESULT AND DISCUSSION

4.1 Results

In this experiment, the full step mode was set and the results will be discussed in this chapter. One pulse will rotate 1.8° for full stepping of rotation. To find step per revolution, calculation is formulated as follows:

$$1 \text{ pulse} = 1.8^\circ$$

$$1 \text{ rev} = 360^\circ$$

$$\frac{\text{Pulse}}{\text{revolution}} = \frac{360^\circ}{1.8^\circ} = 200 \text{ pulse/rev}$$

In this research, nine readings of stepper motor position were taken and each reading was repeated for three times to find the average reading. The table below shows the steps given and the distance moved by the stepper motor. The results were recorded and the standard deviation were determined. From the research, 100 pulse will move stepper motor 100 mm forward or reverse direction and single pulse will move 1 mm. The maximum pulse needed to move the stepper motor is 5400 pulse for x-axis and 4000 pulse for y-axis. For the first reading five images were taken as a proof and the error was recorded.

Table 4.1: Steps and average distance x

Steps	Distance_x1 (± 0.1) mm	Distance_x2 (± 0.1) mm	Distance_x3 (± 0.1) mm	Average (± 0.1) mm	Std. Dev
500	51.0	52.0	53.0	52.0	0.8
750	74.0	74.0	74.0	74.0	0.2
1000	102.0	102.0	103.0	102.0	0.4
2000	200.0	201.0	201.0	201.0	0.6
2250	227.0	227.0	228.0	228.0	0.6
2500	255.0	255.0	256.0	255.0	0.5
3000	304.0	305.0	305.0	305.0	0.6
4000	396.0	397.0	398.0	397.0	0.8
5400	544.0	545.0	545.0	545.0	0.7

Table 4.2: Steps and average distance y

Steps	Distance_y1 (± 0.1) mm	Distance_y2 (± 0.1) mm	Distance_y3 (± 0.1) mm	Average (± 0.1) mm	Std. Dev
0	1.0	2.0	0.0	1.0	0.8
500	48.0	48.0	49.0	48.0	0.6
1000	100.0	101.0	101.0	101.0	0.7
1500	148.0	148.0	148.0	148.0	0.2
1850	187.0	188.0	188.0	188.0	0.6
2000	199.0	200.0	201.0	200.0	0.8
2000	201.0	201.0	202.0	201.0	0.4
3000	298.0	298.0	298.0	298.0	0.2
3500	346.0	346.0	347.0	346.0	0.4
4000	404.0	405.0	405.0	405.0	0.5

Table 4.2 and Table 4.1 above show nine data that was obtained for y-axis and y-axis after the experiment was carried out. The average distances and the standard deviation are determined and the graph to show the relationships between distance and pulses are plotted in Figure 4.1 for x-axis and 4.2 for y-axis. The error for x-axis is in the range of -3mm to 5mm while the error for y-axis is between -2mm to 5mm.

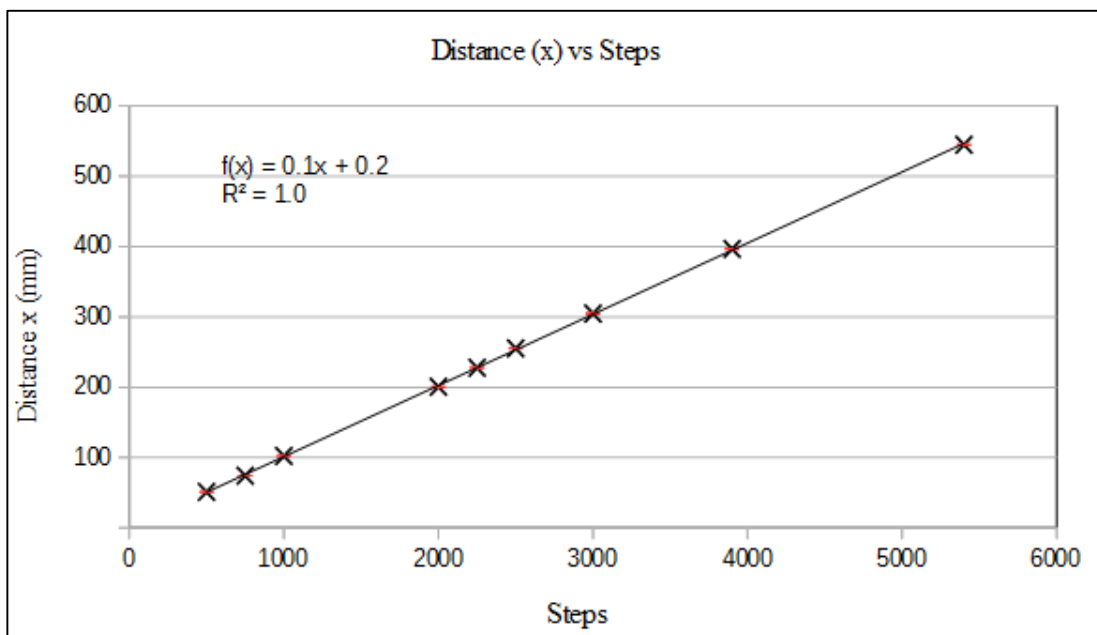


Figure 4.1: Steps vs average distance x

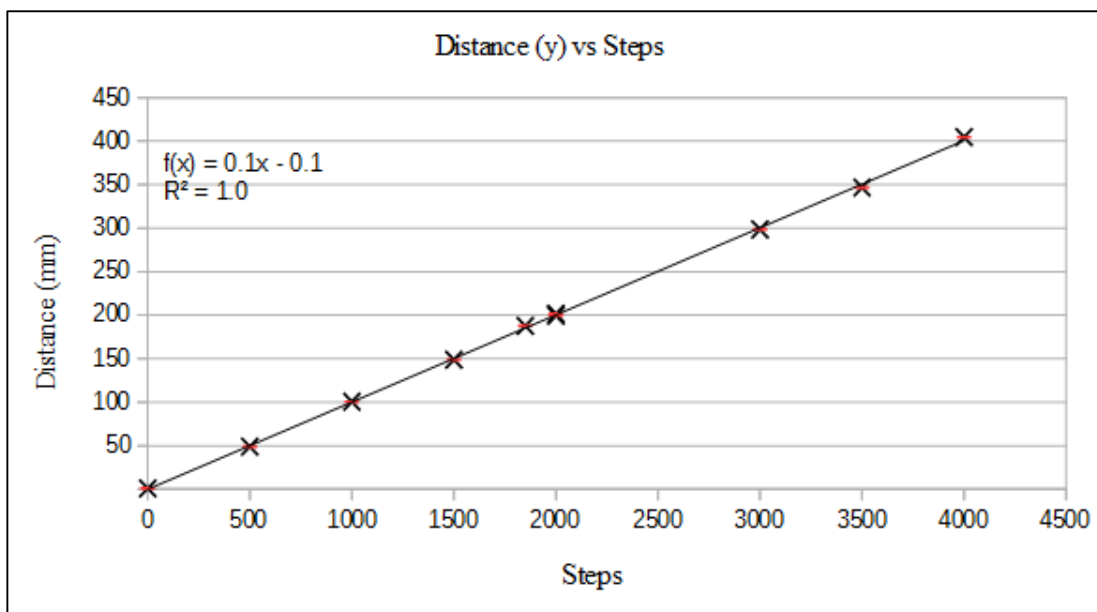


Figure 4.2: Steps vs average distance y

Both Figure 4.1 and Figure 4.2 above show the relationships between distance and pulses for x-axis and y-axis. Coefficient of determination (R^2) indicates the percent of data closest to the best fit line. It is calculated by dividing the explained of variation with the total variation. Value of R^2 represents the variance explained in percent. From both graph we can see that the value of R^2 is 1 which is equals to 100%. These values indicate the variability of data around its means.

$$R^2 = \frac{\text{Explained variation}}{\text{Total variation}}$$

The Y error bars was set with red colour for Figure 4.1 and Figure 4.2. Both graphs show a respectively small error of the data.

4.2 Discussion

During doing this experiment, the truth table form TB6560 data sheet was studied to understand the operation of the ICs.

Table 4.3: Truth table

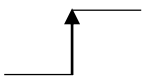
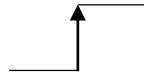
INPUT				OUTPUT MODE
CLK/STEP	DIRECTION	RESET	ENABLE	
	LOW	HIGH	HIGH	CW
	HIGH	HIGH	HIGH	CCW

Table 4.3 above shows how the direction of stepper motor was determined. For x-axis, PORT C was used for the connection between the driver and the computer. Pin PC0 was set for Enable pin, PC1 was set for Direction pin and PC2 was set for Step pin. While y-axis used PORT B to drive the motor. PB4 was set for Enable pin, PB5 was set for Direction pin and PB6 was set for Step pin.

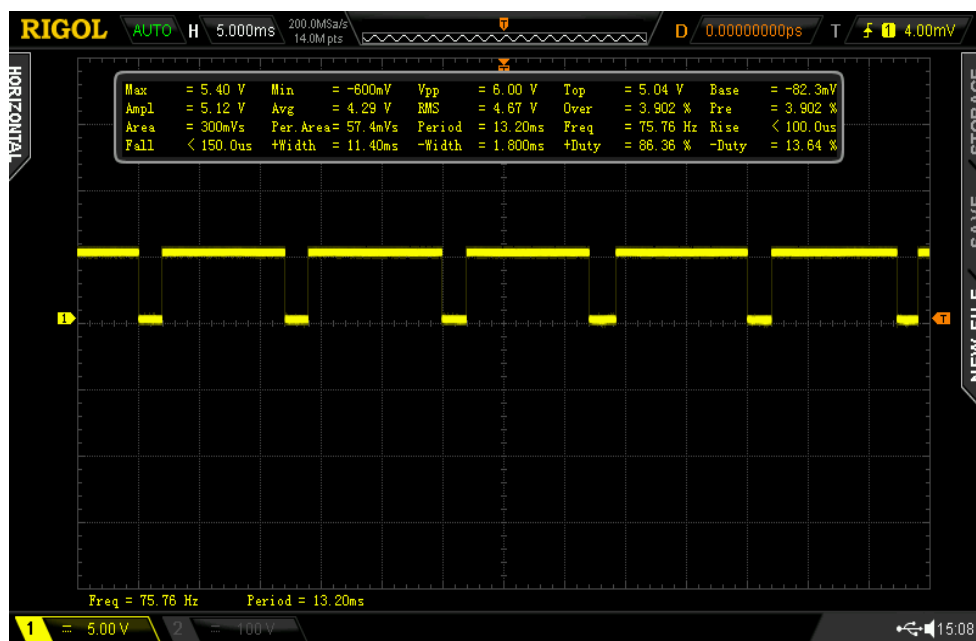


Figure 4.3: Output for 10 ms

The oscilloscope was set to Channel 1 and used for observing the output of the stepper motor. From the truth table shown, the directions of the stepper motor move clockwise (forward) or counter clockwise (reverse) were determined by the input given either 'LOW' or 'HIGH' for Direction input and Enable pin must always set to 'HIGH' signal. If the Direction signal is set to 'LOW' signal, it rotates to clockwise direction and if it is to 'HIGH' signal, it rotates to counter clockwise. The reset pin is automatically controlled by the driver at 'HIGH' signal.

From the oscilloscope shown in Figure 4.3, the frequency for x forward is 75.76 per 100 pulse at 13.20 ms. 10 ms was programmed but the oscilloscope shows 13.20 ms due to some delay time taken by motor to convert from mechanical signal to electrical signal. The calculation to obtain the frequency is shown below.

$$\begin{aligned}
 f &= 1/T \\
 &= 1/13.20 \text{ ms} \\
 &= 75.76 \text{ Hz}
 \end{aligned}$$

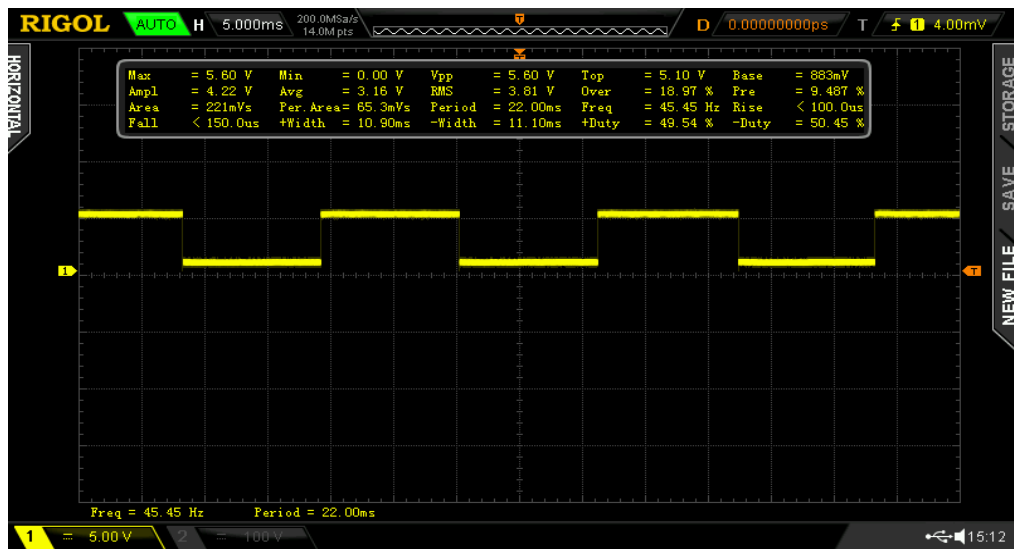
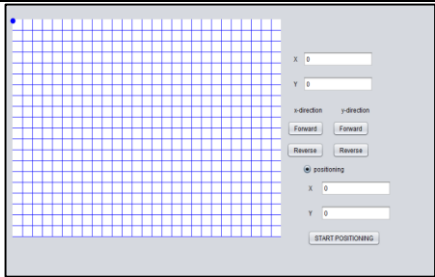
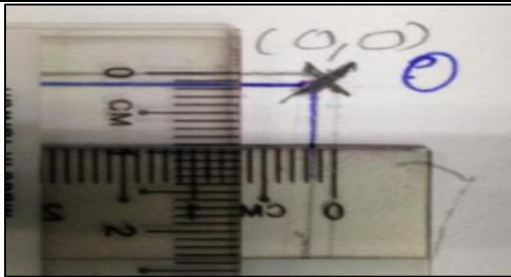
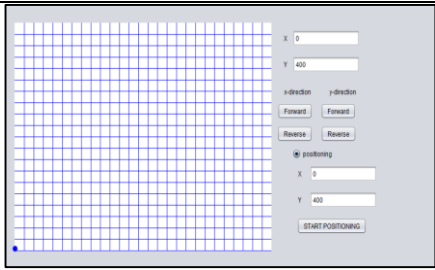
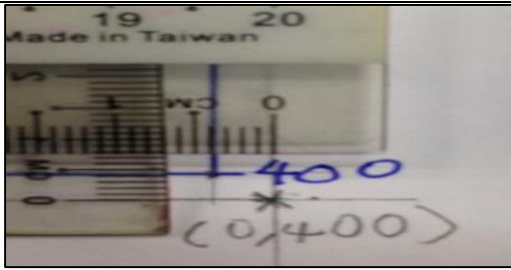
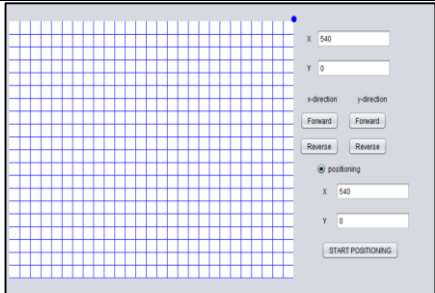
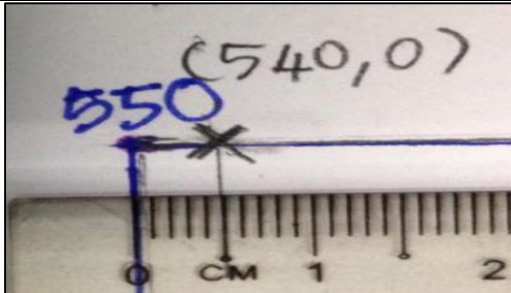
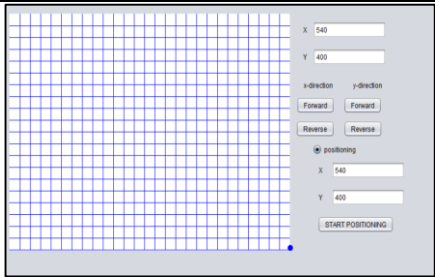
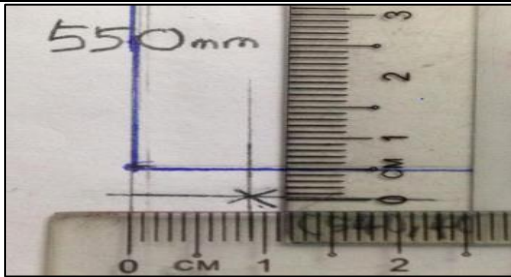
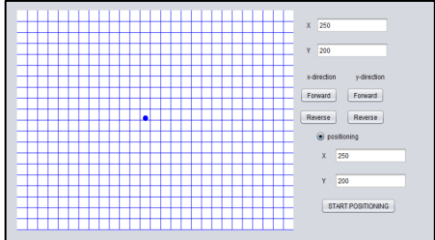
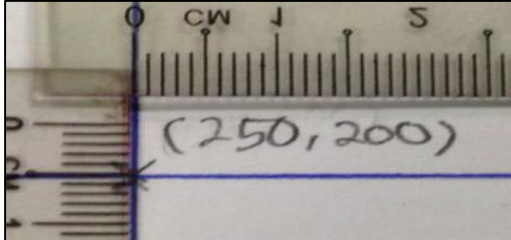


Figure 4.4: Output for 20 ms.

The delay time was changed to 20 ms per 100 pulse in Java programming. From the output result as shown in Figure 4.4, the frequency is 45.45 Hz per 100 pulse. This frequency is acquired from the calculation below.

$$\begin{aligned}
 f &= 1/T \\
 &= 1/22.00 \text{ ms} \\
 &= 74.46 \text{ Hz}
 \end{aligned}$$

Table 4.4: Comparison between Java Programming and hardware

	Java Programming	Hardware
A	 <p>Origin position (0,0)</p>	
B	 <p>Position (0,400)</p>	
C	 <p>Position (540,0)</p>	
D	 <p>Position (540,400)</p>	
	 <p>Position (250,200)</p>	

From the Table 4.4, the images shown that some point does not accurately hit the selected point. Point A, B, C, D and E was positioned at (0,0), (0,400), (540,0), (540,400) and (250,200). Basically, there are no physical quantity can be measured with perfect certainty. There are always error in measurements. Two factors that cause uncertainties are random errors and systematic errors. For systematic error, the initial point of stepper motor does not exactly start at (0,0) point. Random errors are caused from electronic noise in the circuit, electrical instrument and mechanical torque. For example, the mechanical movement of stepper motor need enough delay period given by the software to convert into electrical signal. When a stepper motor moves from one point to another, rotor does not immediately stop, it actually passes to the next point. The rotor is drawn back to the position in opposite direction. It occurs when every single step the motor take, the momentum cause the rotor passes its stop position and bounces forward and backward until it rests.

Another factor causes the error is the motor linearity. From motor technical specification, it should move 1.8° every single steps and 0.18° for every 10 micro steps. All stepper motor exhibit some non-linearity which means the micro steps bunch together in the full step mode. Two effects due this problem is the motor position is not at optimum position and dynamic speed of resonance low.

CHAPTER 5

CONCLUSION

5.1 Conclusion

As the conclusion, the project has been successfully developed a two dimensional (2D) stepper motor controller. The relationship between the stepper motor distance movement and the given pulse were studied. From the experiment that was conducted, the controller is capable to move the stepper motor to their specific distance and position at certain pulse. In addition, 360° shaft rotation need 200 pulse will move the stepper motor to 200mm. The error for x-axis is in the range of -3mm to 5mm while the error for y-axis is between -2mm to 5mm. Some safety factor need to be considered in order to protect the stepper motors and other instruments from damage.

5.2 Recommendation

It is recommended that identical research should be carried out by using different software program, types of driver and types of stepper motor to compare the performance of the stepper motor. In addition, the speed control for stepper motor can be design to govern the stepper motor motion and frequency can be varied.

REFERENCES

1. Douglas W. J (1998). *Control of Stepping Motors*. University of IOWA, United States.
2. Herman, S., & Alerich, W. (1990). *Industrial motor control* (2nd ed.). Albany, N.Y.: Delmar.
3. Hughes, A. (2006). *Electric motors and drives fundamentals, types, and applications* (3rd ed.). Amsterdam: Elsevier/Newnes.
4. Keljik, J. (2007). *Electric Motors and Motor Controls* (2nd ed.). United States of America: Delmar.
5. Lazaridis, G. (2010). *How Stepper Motors Work*,.Electronic Workbench PCB heaven.
6. M. M Fauzi (2010). *Stepper Motor Controller*. Undergraduates Project Papers, Faculty of Electrical and Electronic Engineering, Universiti Malaysia Pahang, Malaysia.
7. Morar, A. (2013). *A Study of Development of a Dedicated Control IC for a Five Phase Stepper Motor Driver*. Procedia Technology, 83-89.
8. Parente, B. (2011). *Stepper motor basics: Half and Micro stepping*. Retrieved June 5, 2015, from <http://motion.schneider-electric.com/technology-blog/stepper-motor-basics-half-and-micro-stepping/>

9. Takashi, K (1984). *Stepping motors and their microprocessor controls*. Kanagawa: Clarendon Press ;.
10. Yeadon, W. (2001). *Handbook of small electric motors*. New York: McGraw-Hill.

APPENDIX A

Java Coding For Stepper Motor Controller

```

/*
 *This program are about stepper motor controller for x-axis and y axis. This program for start, reset, forward,
reverse
 *and positioning the stepper motor.
 *@author Anith Khairunnisa Binti Ghazali
 * 4 SSCF 2014/15
 * 2D Stepper Motor Controller
 */

import ComInt.*;
import javax.swing.*;
import java.awt.event.*;
import ComInt.Com;
import ComInt.Parameters;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.MouseInfo;
import java.awt.Point;
import java.awt.PointerInfo;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.Timer;

public class mouseUI extends javax.swing.JFrame implements ActionListener {
*****VARIABLES DECLARATION*****
Graphics2D g1,g2,g2d,g3;
Timer timer;
static Parameters param;
static Com com;
int msec;//milisec
int data,RB,RC, RC7,RC6,stpInpX,stpInpY,position,X;
String strRC;
int ONOFF;
private Point previousPoint = new Point();
private Point nextPoint = new Point();
private boolean drawing;
double x ;
double y ;
int currentpostX=0;
int currentpostY=0;
/**
 * Creates new form mouseUI
 */
public mouseUI() {

    initComponents();

    msec= 500;
    timer= new Timer(msec, this);
    timer.start();

    try
    {param=new Parameters();
    param.setPort("COM2");
    com=new Com(param);
    com.sendString("RESET\r",1);
    com.sendString("CPB0\rCPC11111000",0);
    }catch (Exception e){System.out.append("Device not Found");}
}

```

```

@SuppressWarnings("unchecked")
public void actionPerformed(ActionEvent ee){
    try {
        stpInpX = Integer.parseInt(X_steps.getText());
        stpInpY = Integer.parseInt(Y_steps.getText());
    }
    catch(Exception e){System.out.println("Do not left empty");}
}
*****BUTTON*****

STOP.setText("STOP");
STOP.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        STOPActionPerformed(evt);
    }
});

RESET1.setText("RESET");
RESET1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        RESET1ActionPerformed(evt);
    }
});

start.setText("START POSITIONING");
start.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        startActionPerformed(evt);
    }
});

private void RevYActionPerformed(java.awt.event.ActionEvent evt) {
*****REVERSE Y*****
if (currentpostY >=10){ //Minimum Y position
try{
    int i;
    com.sendString ("setpb4\r",0);//High Enable
    com.sendString ("setpb5\r",0);//High Direction

    for ( i =0; i<100 ;i++){//100 pulse (10 cm)
        com.sendString ("setpb6\r",0);//High Steps
        Thread.sleep(10);
        com.sendString ("respb6\r",0);//Low Steps
        Thread.sleep(10);
    }
    currentpostY=currentpostY -i/10;//Minus with current position
    Y_location.setText(""+currentpostY);
    repaint();
}
catch (Exception e){System.out.println("Device not found");}
}
*****FORWARD Y*****
private void fwdYActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:

if (currentpostY <=400){ //Maximum Y position
try{
    int i;

    com.sendString ("setpb4\r",0);//High Enable
    com.sendString ("respb5\r",0)// Low Direction

    for ( i =0; i<100; i++){//100 pulse (10 cm)
        com.sendString ("setpb6\r",0);//High Steps
        Thread.sleep(10);
        com.sendString ("respb6\r",0);//Low Steps
        Thread.sleep(10);
    }
    currentpostY=currentpostY +i/10;//Add up with current position
    Y_location.setText(""+currentpostY);
    repaint();
}
}
}

```

```

    }catch (Exception e){System.out.println("Device not found");
    }
}
*****REVERSE X*****

private void revXActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:

if (currentpostX >=10 ){//Minimum X position
try{
repaint();
int i;
com.sendString ("setpc0\r",0);//High Enable
com.sendString ("setpc1\r",0);// High Direction

for (i =0; i<100 ;i++){//100 pulse (10 cm)
com.sendString ("setpc2\r",0);
Thread.sleep(10);
com.sendString ("respc2\r",0);//Low Steps
Thread.sleep(10);
}
currentpostX=currentpostX -i/10;//Minus with current position
X_location.setText(""+currentpostX);
repaint();
}
catch (Exception e){System.out.println("Device not found");}

}

}

private void FwdXActionPerformed(java.awt.event.ActionEvent evt) {
*****Forward X*****
if (currentpostX <=540){ //Maximum X position

try{
int i;
com.sendString ("setpc0\r",0);//High Enable
com.sendString ("respc1\r",0);// Low Direction

for (i =0; i<100; i++){//100 pulse (10 cm)
com.sendString ("setpc2\r",0);
Thread.sleep(10);
com.sendString ("respc2\r",0);//Low Steps
Thread.sleep(10);
}
currentpostX=currentpostX +i/10; //Add up with current position
X_location.setText(""+currentpostX);
repaint();
}catch (Exception e){System.out.println("Device not found");}

}

}
*****STOP*****

private void STOPActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
fwdY.setEnabled(true);
RevY.setEnabled(true);
FwdX.setEnabled(true);
revX.setEnabled(true);
STOP.setEnabled(false);
position=0;
}

private void RESET1ActionPerformed(java.awt.event.ActionEvent evt) {
*****RESET*****

//-----X RESET-----
try{

com.sendString ("setpc0\r",0);//High Enable
com.sendString ("setpc1\r",0);//High Direction
repaint();
}
}

```

```

int data ;

do {
    com.sendString ("setpc2\r",0);//High Step
    Thread.sleep(10);
    repaint();
    com.sendString ("respc2\r",0);//Low Step
    Thread.sleep(10);
    com.sendString("RD8\r",0);//X switch connection
    data = Integer.parseInt(com.receiveToString('\n', 0));//ADC

    data++;
    repaint();
} while (data < 1000);//5V
    repaint();
{ com.sendString ("respc0\r",0); ; //Low Enable
currentpostX = 0;
repaint();
X_location.setText(""+ currentpostX );
repaint();
}catch (Exception e){System.out.println("Device not found");}

//-----Y RESET-----
try{

    Thread.sleep(500); //safety delay
    com.sendString ("setpb4\r",0);//High Enable
    com.sendString ("setpb5\r",0);//High Direction
    repaint();
    int data ;
    repaint();
    do {
        com.sendString ("setpb6\r",0);//High Step
        Thread.sleep(10);
        repaint();
        com.sendString ("respb6\r",0);//Low Step
        Thread.sleep(10);
        com.sendString("RD9\r",0);//Y switch connection
        data = Integer.parseInt(com.receiveToString('\n', 0));//ADC
        data++;
        repaint();
    } while (data < 1000);//5V
        repaint();
{ com.sendString ("respb4\r",0); ; //Low Enable
currentpostY = 0;

Y_location.setText(""+ currentpostY);
repaint();
}catch (Exception e){System.out.println("Device not found");}

}
*****SELECTION BUTTON*****
private void X_stepsActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    X_steps.setEnabled(false);
    position=1;

}

private void postActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    X_steps.setEnabled(true);
    Y_steps.setEnabled(true);
    position=1;

}

private void startActionPerformed(java.awt.event.ActionEvent evt) {
*****POSITIONING*****

    if ((position==1)){//Calculate different
    int postX= stpInpX -currentpostX;
    int postY= stpInpY -currentpostY;

```

```

repaint();

//-----Positive X-----
if ((postX > 0) & (currentpostX <= 540)) { // Check different and position

repaint();

    try{
int i;
    com.sendString ("setpc0\r",0); // High Enable
    com.sendString ("respc1\r",0); // Low Direction
    repaint();
    for (i = 0; i < 10 * postX; i++) { // Time with Step
    com.sendString ("setpc2\r",0); // High Step
    Thread.sleep(10);
    repaint();
    com.sendString ("respc2\r",0); // Low Step
    Thread.sleep(10);
    repaint();
    }
    currentpostX = currentpostX + i / 10; // Add up with current position
    X_location.setText("" + currentpostX);
    repaint();
} catch (Exception e) { System.out.println("Device not found");
}

//-----Negative X-----
repaint();
if ((postX < 0) & (currentpostX >= 10)) { // Check different and position
repaint();
    try{

int i;
    com.sendString ("setpc0\r",0); // High Enable
    com.sendString ("setpc1\r",0); // High Direction
    repaint();
    for (i = 0; i < (-10) * postX; i++) { // Time with Step (negative steps)
    com.sendString ("setpc2\r",0); // High Step
    Thread.sleep(10);
    com.sendString ("respc2\r",0); // Low Step
    Thread.sleep(10);
    repaint();
    }
    currentpostX = currentpostX - i / 10; // Minus with current position
    X_location.setText("" + currentpostX);
    repaint();
    }
    catch (Exception e) { System.out.println("Device not found");
}
}
repaint();

//-----Positive Y-----
if ((postY > 0) & (currentpostY <= 400)) { // Check different and position

    try{
int i;
repaint();
    com.sendString ("setpb4\r",0); // High Enable
    com.sendString ("respb5\r",0); // Low Direction

repaint();
    for (i = 0; i < 10 * postY; i++) { // Time with Step
    com.sendString ("setpb6\r",0); // High Step
    Thread.sleep(10);
    repaint();
    com.sendString ("respb6\r",0); // Low Step
    repaint();
    }
    currentpostY = currentpostY + i / 10; // Add up with current position
    Y_location.setText("" + currentpostY);
    Thread.sleep(10);
    repaint();
} catch (Exception e) { System.out.println("Device not found");
}
}

```

```

    }

//-----Negative Y-----
    repaint();
    if ((postX< 0)&&(currentpostY >=10)){
    try{ repaint();
        int i;
        com.sendString ("setpb4\r",0);//High Enable
        com.sendString ("setpb5\r",0);//High Direction
        repaint();
        for ( i =0; i<(-10)*postX ;i++){//Time with Step (negative steps)
        com.sendString ("setpb6\r",0);// High Step
        Thread.sleep(10);
        repaint();
        com.sendString ("respb6\r",0);// Low Step
        Thread.sleep(10);
        repaint();

        }
        currentpostY=currentpostY -i/10; //Minus with current position
        Y_location.setText(""+currentpostY);
        repaint();
    }
    catch (Exception e){System.out.println("Device not found");}}

}
}

private void Y_stepsActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Y_steps.setEnabled(false);
    position=1;

}
@Override
*****GRID COMMAND*****

public void paint(Graphics g) {

    g2 =(Graphics2D)g;
    g1 =(Graphics2D)g;
    g2d =(Graphics2D)g;
    g3 =(Graphics2D)g;
    super.paint(g2);
    super.paint(g1);
    super.paint(g3);
    g1=(Graphics2D) mouseUI.getGraphics();
    g2=(Graphics2D) mouseUI.getGraphics();
    g1.setColor(Color.blue);
    g2.setColor(Color.BLUE);

//-----X Grid-----
    g2.drawLine(0, 20, 540, 20);
    g2.drawLine(0, 40, 540, 40);
    g2.drawLine(0, 60, 540, 60);
    g2.drawLine(0, 80, 540, 80);
    g2.drawLine(0, 100, 542, 100);
    g2.drawLine(0, 120, 542, 120);
    g2.drawLine(0, 140, 542, 140);
    g2.drawLine(0, 160, 542, 160);
    g2.drawLine(0, 180, 542, 180);
    g2.drawLine(0, 200, 542, 200);
    g2.drawLine(0, 220, 542, 220);
    g2.drawLine(0, 240, 542, 240);
    g2.drawLine(0, 260, 542, 260);
    g2.drawLine(0, 280, 542, 280);
    g2.drawLine(0, 300, 542, 300);
    g2.drawLine(0, 320, 542, 320);
    g2.drawLine(0, 340, 542, 340);
    g2.drawLine(0, 360, 542, 360);
    g2.drawLine(0, 380, 542, 380);
    g2.drawLine(0, 400, 542, 400);

```

```

g2.drawLine(0, 420, 542, 420);
g2.drawLine(0, 440, 542, 440);
g2.drawLine(0, 460, 542, 460);
g2.drawLine(0, 480, 542, 480);
g2.drawLine(0, 500, 542, 500);
g2.drawLine(0, 520, 542, 520);
g2.drawLine(0, 540, 542, 540);
g2.drawLine(0, 542, 542, 542);

//-----Y Grid-----

g1.drawLine(20, 0, 20, 400);
g1.drawLine(40, 0, 40, 400);
g1.drawLine(60, 0, 60, 400);
g1.drawLine(80, 0, 80, 400);
g1.drawLine(100, 0, 100, 400);
g1.drawLine(120, 0, 120, 400);
g1.drawLine(140, 0, 140, 400);
g1.drawLine(160, 0, 160, 400);
g1.drawLine(180, 0, 180, 400);
g1.drawLine(200, 0, 200, 400);
g1.drawLine(220, 0, 220, 400);
g1.drawLine(240, 0, 240, 400);
g1.drawLine(260, 0, 260, 400);
g1.drawLine(280, 0, 280, 400);
g1.drawLine(300, 0, 300, 400);
g1.drawLine(320, 0, 320, 400);
g1.drawLine(340, 0, 340, 400);
g1.drawLine(360, 0, 360, 400);
g1.drawLine(380, 0, 380, 400);
g1.drawLine(400, 0, 400, 400);
g1.drawLine(420, 0, 420, 400);
g1.drawLine(440, 0, 440, 400);
g1.drawLine(460, 0, 460, 400);
g1.drawLine(480, 0, 480, 400);
g1.drawLine(500, 0, 500, 400);
g1.drawLine(520, 0, 520, 400);

g2d.setColor(Color.blue);
g2d.setRenderingHint(
RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
g2d.setStroke(new BasicStroke(8,BasicStroke.CAP_ROUND, BasicStroke.JOIN_BEVEL));
g3.drawLine(previousPoint.x, previousPoint.y, nextPoint.x, nextPoint.y);
g2d.fillOval((currentpostX+64), (currentpostY+88), 10, 10);
}

```