" I hereby declare that I have read through this report entitled "DESIGN AND DEVELOPMENT OF LEAD-THROUGH PROGRAMMING METHOD USING LOW COST INCREMENTAL ENCODER FEEDBACK" and found that it has complied the partial fulfillment for awarding the degree of Bachelor of Electrical Engineering (Mechatronics)

Signature: .....

Supervisor's Name: DR. MUHAMMAD FAHMI BIN MISKON

Date: 24th/6/2015

DESIGN AND DEVELOPMENT OF LEAD-THROUGH PROGRAMMING METHOD USING LOW COST INCREMENTAL ENCODER FEEDBACK

SAMEH MOHSEN OMER KANZAL

A report submitted as a partial fulfillment of the requirements for the degree of Mechatronics Engineering

Faculty of Electrical Engineering

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

June 2015

I declare that this report entitled "DESIGN AND DEVELOPMENT OF LEAD-THROUGH PROGRAMMING METHOD USING LOW COST INCREMENTAL ENCODER FEEDBACK" 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 concurrently submitted in candidature of any other degree.

Signature:……

Name: Sameh Mohsen Omer Kanzal

Date: 24st/June/2015

DEDICATION

I would like to express my gratitude to my supervisor: DR. MUHAMMAD FAHMI BIN MISKON for his sincere guidance along my project. I would also like to thank my panels and lecturers for their continuous contributions that made this project possible.

Lastly, I would like to thank the special gifts have been given to me by Allah, my Father, Mother and Sisters, for their prayers and support along the journey of my study. I would never be able to compensate them for whatever they have been doing to me along my bachelor degree journey.

ABSTRACT

Recently robots are widely used in various fields particularly in industry. Despite this fact, robots still require an undeniable amount of knowledge from the operators or workers who deal with them. As a result, robots cannot be easily programmed if the operator or the worker is not well-experienced in robotics' field. One of the programming methods that has been introduced to make programming task user-friendly is lead-through robot programming. However, the existing lead-through programming methods still require an amount of knowledge that is not available for most of the operators and workers. The main objective of this project is to design a lead through programming method for point-to-point robots' programming using inexpensive incremental encoder feedback, which can record, save and playback the robots' motion while considering the accuracy and precision of the motion. To validate this method, an experiment was conducted in this project, where an operator manually moves a two DOF (degree of freedom) robotic arm on a white board while the encoder feedback was recorded and later the same motion was played back by the robot. Then both recorded and playback trajectories were compared and analyzed. The results show that the played back accuracy is 96.17% for motor 1 and 97.86% for motor 2 with a standard deviation of 0.9593 for motor 1 and 2.33583 for motor 2.

ABSTRAK

Dewasa ini robot digunakan dalam banyak aktiviti manusia terutamanya di industri. Walaupun robot banyak digunakan, ia masih memerlukan operator dan juruteknik berkemahiran tinggi untuk digunakan. Kesannya, robot sukar di programkan. Salah satu usaha yang memudahkan program robot ialah dengan kaedah lead-through. Bagaimanapun, kaedah ini masih memerlukan tenaga mahir dan kosnya tinggi dengan sensor dan alatan tambahan yang mahal. Justeru, objektif projek ini adalah untuk mereka kaedah program lead-through menggunakan incremental encoder, yan gboleh rekod, simpan, dan main semula pergerakan robot. Untuk tujuan validasi, eksperimen dijalankan dengan seorang operator menggerakkan 2 DOF robot di atas sekeping papan putih dengan bacaan enkoder direkod dan dimain semula. Trajektori yang direkod dan yang dimainkan di bandingkan dan dianalisa. Hasil kajian menunjukan ketepatan motor 1 dan motor 2 ialah 96.17% dan 97.86% dengan standard deviation sebanyak 0.9593 dan 2.33583 untuk motor 1 dan 2.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.0 Overview

This chapter includes motivation regarding the continuous evolution of robotics existence in a diverse of fields, occurrence of lead-through programming method, Problem statement, objectives of this paper and the scope along with the expected limitations of the project.

1.1 Motivation

In the last two decades, robots have gained enough technological concern and public acceptance to shift from revolutionary concept to an evolutionary development that remarkably attracts developers' and operators' attention [1], Figure 1.1 shows the world annual supply of the industrial training and how it has dramatically increased.

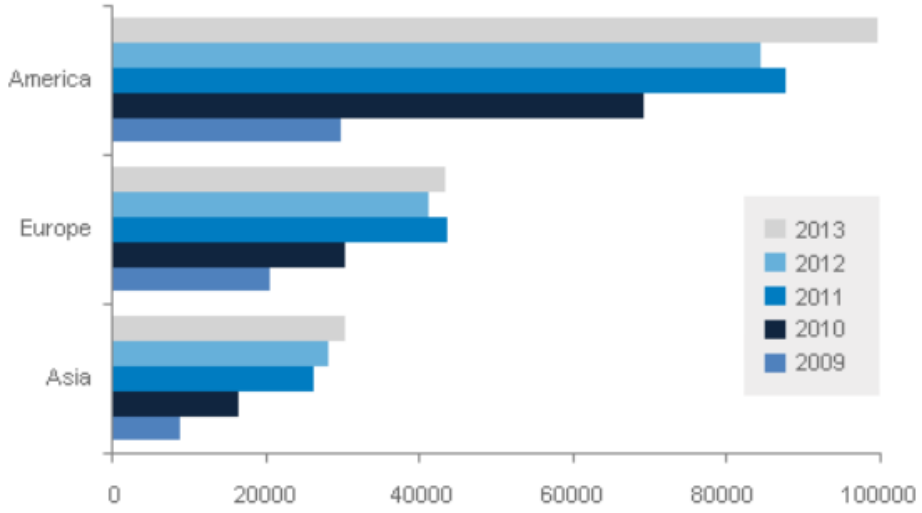

Figure 1.1: World Annual Supply of Industrial Training by Region 2009-2013 [2]

Motion planning problem is the main concept that hooks robots developers' concern, this type of planning is known as a trajectory generation. Recently, a diversity of human-friendly robots and partner robots have been developed for the aim of interaction between human and robots in various fields. These robots require intelligent capabilities to support the human-robot interactions [3].

On the other hand, many people are afraid that robots are replacing the human being jobs. But in fact they are relieving humans from various tedious, routine and even dangerous jobs. One of the widely spreading jobs that are being taken over by robots is welding process, as robots have recently replaced human in such jobs, as they are considered extremely hazardous, in terms of noise, intense generated heat and ultraviolet light form the welding torch [4]. As a consequence of the above mentioned various applications used, where robots are implemented in, robots and their motion planning, termed as a trajectory generation, have been given a remarkable attention and undetached part of human being daily life.

Robots Programming can be complicated and time consuming in terms of their motion planning, thus the process of simplifying robots' motion programming has been a top-priority for robotics' industry since the inception of the first industrial robot [13]. Consequently, making robots affordable to everyone, including those who are not well-

experienced with robotic systems' basic knowledge has gained a non-deniable concern. Based on that need, a new trajectory generation method was proposed on 12th of August 1994 by Timothy L. Graf, lead-through teaching method [14]. It relied on the concept that the operator moves the robot and meanwhile it records the motion data and then saves it for a further playback of the same motion applied by the operator. By applying this method to the field of robotic systems trajectory generation industries, the affordability, feasibility and even efficiency will be ensured. Moreover, it gave the robots a sense of human as they detect the motion and then play it back without any effort from the operator during the playing back mode.

1.2 Problem Statement

Currently, robots' trajectory generation is planned and designed by engineers and designers. In other words, manipulating robots' trajectories using a joystick or keypad on a teach-pendant is not easy for a limited skills and experiences operator [9]. For example, if an industrial company needs to change the position and orientation information of the robots used in its industry it will have to contact the manufacturer of its robots to adjust this information using software or whatever method that is used to program their robots. Such procedures make it a bit burdensome for SMEs (Small and Medium Enterprises) to handle and deal with, especially when changes are needed more frequently. As a result, the need for an easier trajectory generation concept, which can be handled with a wider range of workers and operators became vital. Despite the fact that a lead-through programming method, using a teach-pendant, is able to give an operator or a worker the ability to handle the generation of a robot's trajectory, the initial position (Q_0), final position (Q_f) and the time required to achieve the trajectory (t) are still knowledge-demanding variables and require a certain level of robotics knowledge [10]. Moreover, the accuracy of the lead-through programming method became very critical when the robotic arm is required to pick and place an object, as any inaccurate recording information may result in a failure for the robotic arm to grab the object and place it to its exact final position. Consequently, a lead-through programming method that requires only a physical effort from the operator, which eliminates the need for a teach-pendant, can solve the knowledge limitation of

SMEs' operators. i.e. the operator in such a programming method is required to only deal with simple switches and physical movement of the end effector.

1.3 Objectives

The objectives of this project is to:

i. Design and develop a lead-through programming method for a robotic arm that can record the initial and final positions, save them and then repeat them as accurately as possible.

ii. Analyze the performance parameters of the lead-through programming systems in terms of error, accuracy, encoders' pulses detection-speed and precision.

1.4 Scope

This project develops a trajectory generation using a lead-through programming method for robotic systems used in SMEs (Small and Medium Enterprises). The project aims to produce a prototype of a robotic arm with two DOF (degree of freedom) that is able to record the initial and final position of the end effector as moved by the operator and then play it back when required. The performance of the designed system is discussed in terms of error, accuracy, encoders' pulses detection-speed and precision. For the experiment and analysis, an Arduino DUE controller is used to interface the developed system and control the trajectory of the arm based on the motion of the operator. The results of this project is a robotic arm moved manually to a desired final position (Q_{rec}) and then repeat the same motion by itself to the same final position (Q_{played}).

CHAPTER 2

LITERATURE REVIEW

2.0 Overview

This chapter contains theoretical background of the trajectory generation, methods of trajectory generation, lead-through programming method and its problems along with the available proposed solutions and the summary of the solution related to the proposed idea in this project.

2.1 Theoretical Background

Figure 2.1 shows a schematic of a robotic system in general.

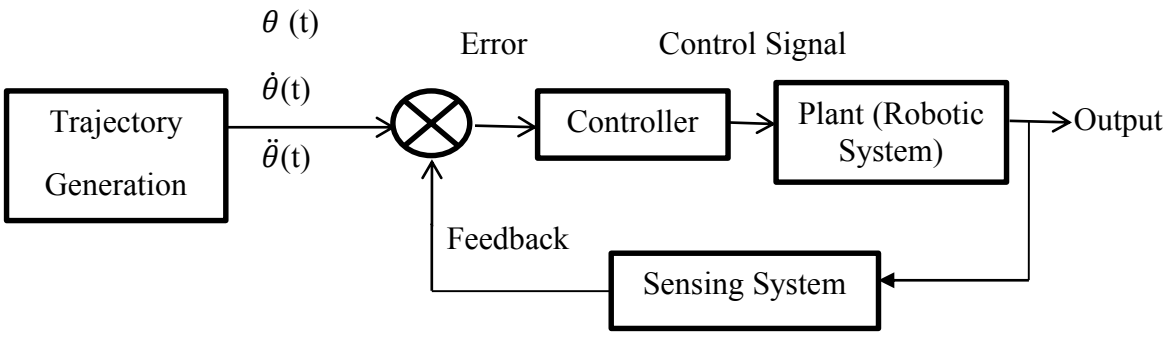

Figure 2.1: Robotic Systems Block Diagram

Robots programming has gained an undeniable concern for the few past years, due to their daily corporative and interactive applications offered to their users and operators.

According to [5], trajectory refers to a time history of position, velocity and acceleration for each degree of freedom. The term trajectory generation is not only generating a path for a tool frame to be located within a tool frame, but also includes the human interface issue with the robot' path specification [5]. For example, if an operator wants to change the location of the robot within a specific space then he may want to be able to specify nothing more than the location and orientation of the end effector and then let the system decide the other information required for that motion, such as duration, velocity and other details.

By assuming the motion of the manipulator is considered as a tool frame, T, and its space is the station frame, S, then the trajectory generation is, in general, changing the position and orientation of the tool frame from an initial value, $T_{initial}$, to an end value, T_{final}, relative to the station frame [5].

In some applications, it is vital to specify the motion of the tool frame in more details. For example specifying the sequence of the desired via point (intermediate points between the initial and final position). These via points are considered as a set of intermediate points carry out the position and orientation information of the tool-frame relative to the station-frame [5].

For further elaboration, most of robotic systems have a common block diagram, shown in figure 2.1, which explains and illustrates the system general input and output and then the sub-blocks which include the processes involved in both input and output.

As mentioned above robots programming has been given much attention. Recently robots have been involved in most of nowadays activities, such as industrial, human services, and even rehabilitation systems. For these reasons a pathway for robots is vital to be studied and determined as well as the human interface issue which indicates how the robotic system does receive its pathway from human.

Trajectory generation is a very wide term that includes many problems which need to be studied independently, not to deny that spatial, time and smoothness are the most relevant problems to trajectory generation.

- **Spatial**: the orientation and location of the end effectors, and how accurately they reach their destination.
- **Time**: how long the end effectors take to reach their desired destination.
- **Smoothness**: identifies whether the robotic system vibrates while moving from the initial to the final station-frame. And how smooth its motion is.

For the purpose of solving such relevant issues many studies have been introduced to contribute to this field. Below are some of these studies, introduced in details.

2.2 Methods to Generate a Trajectory

As shown in Figure 2.2, in [6] robots programming methods are divided into three main categories, despite the fact that over 90% of the robots are programmed using the first method, teach method, lead method and off-line programming.

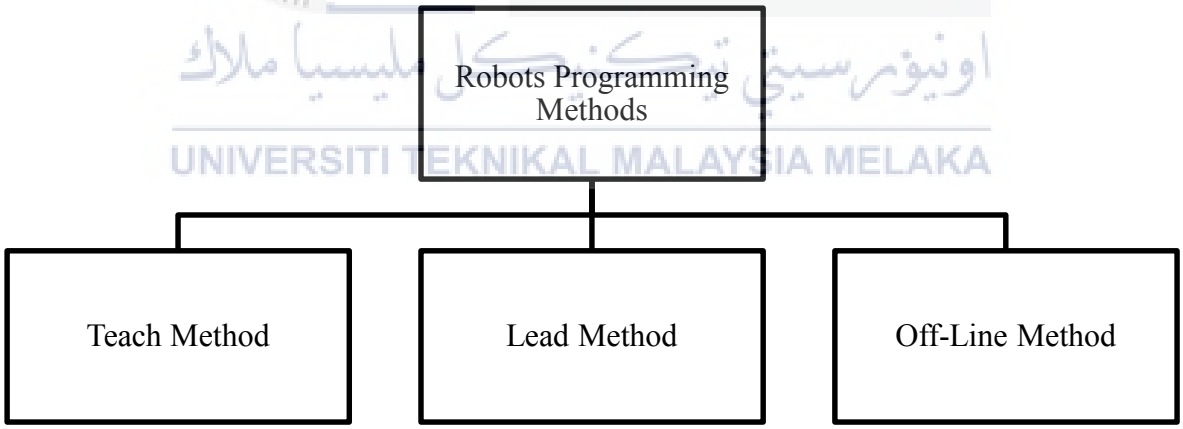

Figure 2.2: Robots Programming Methods [6]

- **Teach Method, On-Line**

The program is generated using either a menu-based system or a text editor. The main characteristic of this method is that the robot is thought how to change its position and/or orientation in a number of different co-ordinate systems to a desired location. This method of programming is simple to be implemented when simple movements are required, but its main disadvantage is that the robot will be out of service during the programming session. Example of teach method is Tiji trajectory generation [7].

- **Off-Line Programming Method**

This method is similar to the teach method in terms of the program build up, except that there are additional tools used to process the CAD (Computer Aided Design) data of the components and generate a sequence of information to be processed. The advantages of this method over the other methods are as follows:

i. Reduce the programming time.
ii. Makes the programming easier.
iii. Enables concurrent engineering and reduces product lead time.
iv. Allows process optimization.

An example of a trajectory generation using off-line programming is in [8].

- **Lead Method**

This method is a physical movement of the robot itself by the operator, during that movement the robot records the movement of its joint and then plays it back. This method is limited to small and medium size robots only, as it is difficult to physically move a large-size robot [8].

2.3 Lead-Through Programming Method

Figure 2.3 illustrates the process of recording data where the operator moves the robot manually by either using one of the interface devices, mentioned below in the problems section, or moving it physically. During the robotic system movement, the transducers attached to the system record the movement's data and store it in either RAM or external memory card for further processing. As a final stage, the data recorded will be processed for the trajectory generation.

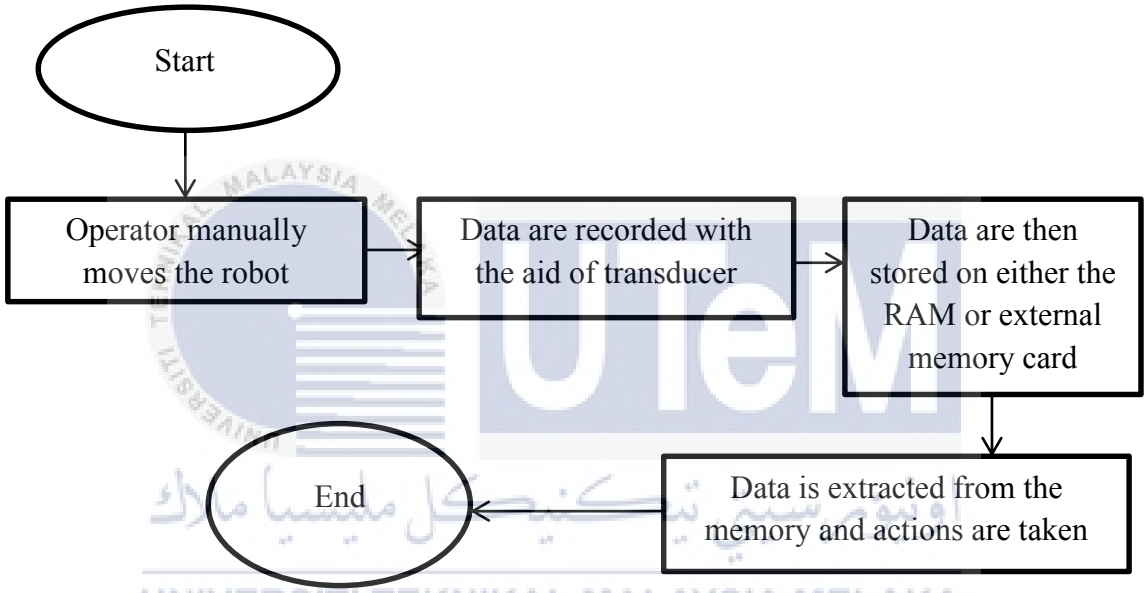

Figure 2.3: Flow Chart of the Data Record Process

2.3.1 Lead-Through Programming Problems

There are four known problems with the lead through method which are (1) affordability, (2) intuitiveness and teaching accuracy of the teach-pendant interface as a human machine interface (HMI), (3) feasibility of the on-line programming due to the great number of the teaching points and (4) the confidentiality and intellectuality of the sensor-less systems especially when path precision is taken into consideration.

The first problem can be described in terms of changing the robotic arms' location and orientation. It is desirable to move the robotic arm's tool frame rather than moving the space frame itself, for such changes in locations and orientations maneuvering robots using a keypad of joystick on the pendant is not easy and affordable to all operators, as it requires a non-deniable amount of skills and experiences [9].

The second problem is regarding the teach-pendant which is one of the most common ways for programming a robot as well as a common human machine interface (HMI) [10]. Yet to program a robot using a teach pendant, the operator should set up the robot's jogging conditions, frame and motion mode, only then he can use the joystick of the teach pendant to move the robot [6]. In comparison with the off-line programming methods, programming a robot with a teach pendant does not need a PC, which is an advantage in terms of cost. Yet a teach pendant programming method is not intuitive and has a low teaching accuracy which requires rounds and rounds of trails and errors, hence it is a time consuming and requires a certain level of robotics knowledge to deal with a teach pendant [10].

On the other hand, the third problem is based on a robotic machining perspective, where there are two types of machining processes whose motion are governed by complex work-piece [11]. Cleaning and deburring machines are typically the first type, which have a very complex 3D curved surface path, crucial cycle time requirements and relatively low surface accuracy. Most of the deburring operations are done manually in extremely noisy, dusty and unhealthy environmental conditions, therefore an automation for these operations is highly desirable [11]. On the other hand, milling machines are the second type of machining processes in which robots move in a simpler path with a lower feeding speed (20-30 mm/s) [11]. One of the most difficulties these machines encounter is generation of the robot motion. Despite the fact that teach pendant is the most carried out conventional method to fulfill a robot on-line programming, it is not feasible for machining processes especially for deburring processes as it has a great number of teaching points and high accuracy is needed for positioning purposes [11]. Moreover, offline programming method, which extracts the CAD data of the work-piece, is more accurate and flexible but it is cost-effective for large batch sizes and still requires additional calibration procedures for higher accuracy demands [11].

Finally yet importantly, the fourth problem is involved in robots that have direct contact with objects they manipulate are called robot force control. With the force control, robots gain one more step towards human nature (feeling or touching). Trajectory generation by the lead-through teaching for force control robots is quite time-consuming process if path precision is considered [12].

According to [12], methods of programming robot paths can be categorized as CAD based and non-CAD based method. CAD-based system is a method where the operator specifies the geometrical entities such as the surface or the edge of the geometry from a CAD model, and then the system will automatically simulate and generate the path in the virtual world. Despite the beneficial features of the CAD drawings, in reality they are neither confidential nor intellectual especially in the foundry industry [12]. As robots in certain situations have to be able to effectively capture the geometrical information of the area or the object they are acting on.

2.3.2 Available Solutions

Based on the previously listed problems, there are four relevant solutions to these problems respectively. (i) Is the usage of the ISD (industrial steering device) which is known as the jogging mouse, (ii) is a 6 DOF (degree of freedom) wire-based programming device, (iii) is an effective teaching method referred as programming by demonstrating (PbD) and (iv) is the addition of a sensing system to the robotic hand.

The first solution was proposed in [9], a commercially available 6 degrees-of-freedom steering device ISD (Industrial Steering Device) from space control was used on a welding arm. For the purpose of accessibility the mouse jogging device is attachable to various locations on the robot. Both the mouse jog and the robot axes were calibrated so that the operator could not jog the robot if the mouse was re-mounted from one location to another without a calibration done on the device. After the mouse jogging device was mounted and calibrated, as shown in Figure 2.4, a graphical user interface (GUI) was used to aid the process of the lead through teaching method, as shown in Figure 2.5.

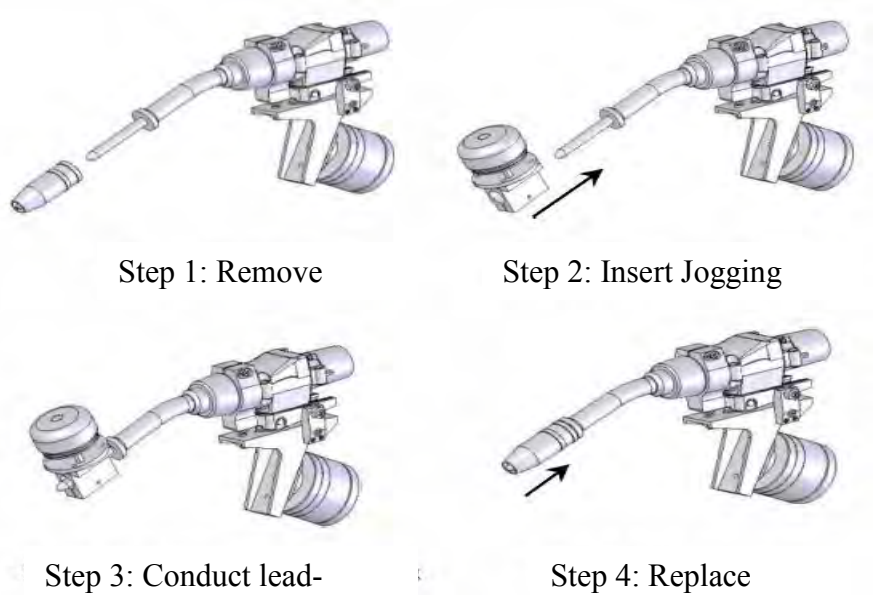

Figure 2.4: Four Major Operational Sequences for the Lead-Through Teaching [9]

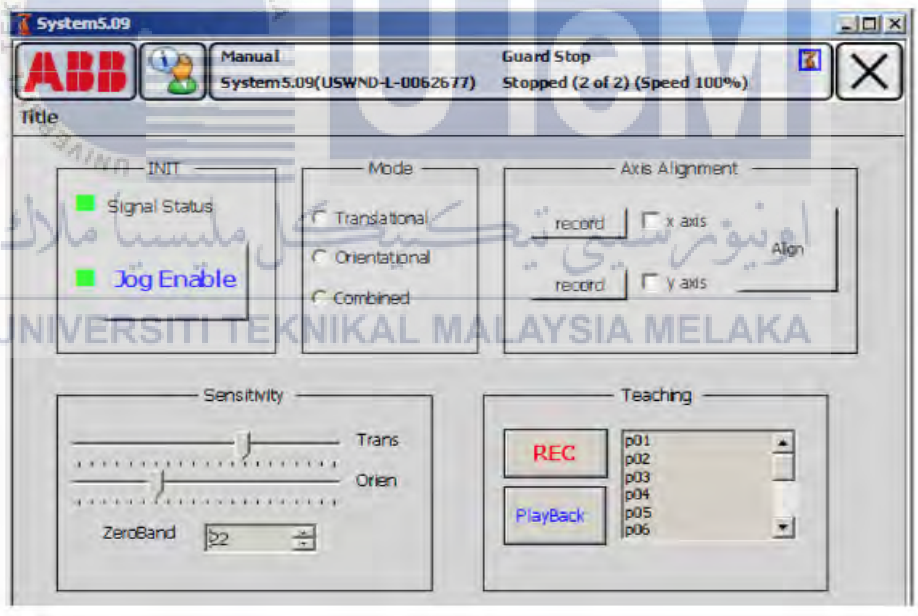

Figure 2.5: Graphical User Interface on Teaching Pendant to Assist Jogging [9]

The second solution was proposed in [10] for the second problem. Even though a 6-DOF mouse is an intuitive technology and demands low physical efforts, it was not a simple solution since the calibration between the 6-DOF mouse and robot coordinate system was required. Consequently, a new device to program a robot was introduced, a

Robot-Puppet. A Robot-Puppet was a 6 DOF wire-based programming device that can detect and measure the motion in 3-DOF rotation and 3-DOF translation. Robot-Puppet programming device thought the robot by the lead-through method, by attaching it to the end effector of the robot and then it was moved by the operator and generates incremental movement information as shown in Figure 2.6. This information was then further processed and a repeatable robot program was generated.

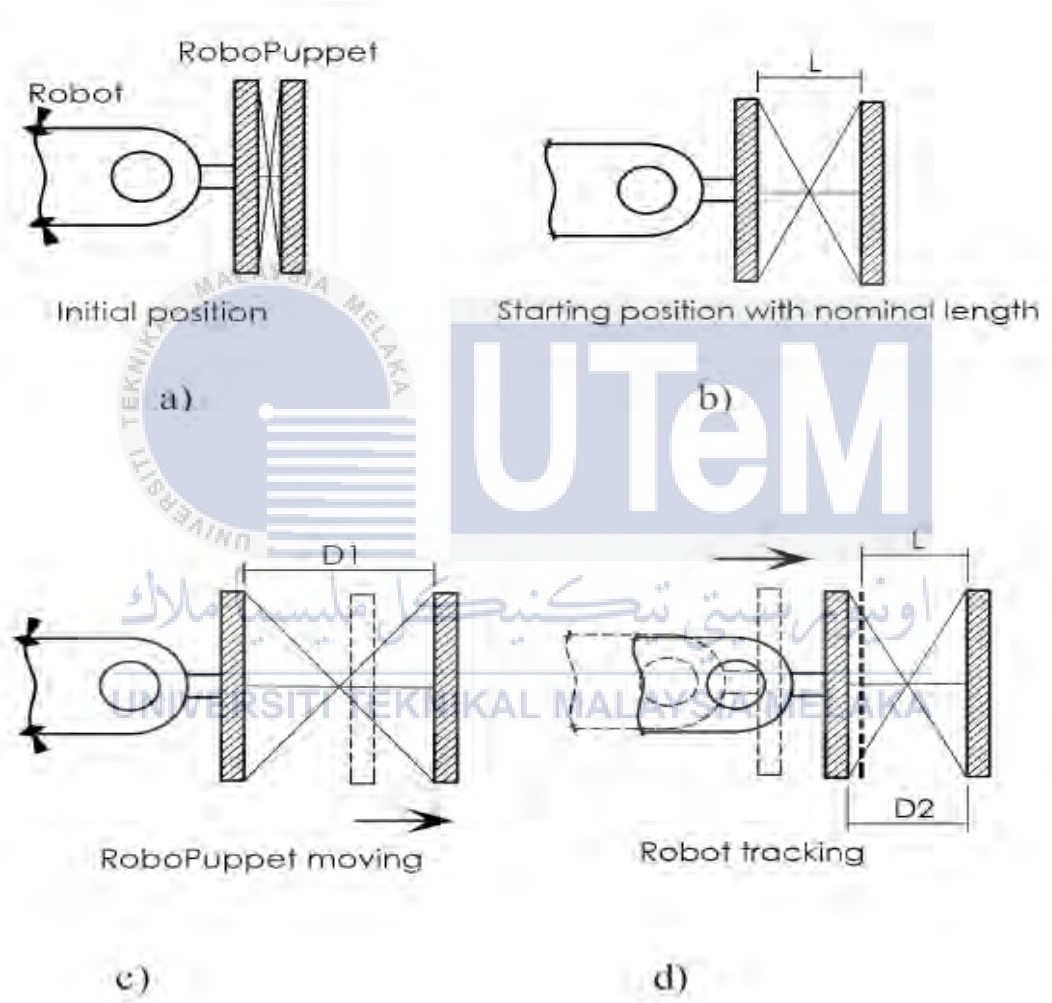

Figure 2.6: General Process of Teaching Robot with Robot-Puppet [10]

The third solution in [11] introduced another effective lead-through teaching method, Programming by demonstrating (PbD). A programming by demonstrating (PbD), aims to solve the problems regarding teaching robots in foundry industries. It consists of three stages:

- Lead-through stage: is the only step that requires the human interaction through the entire stages. As shown in Figure 2.7 the operator identifies few gross guiding points, which are used in the second stage.

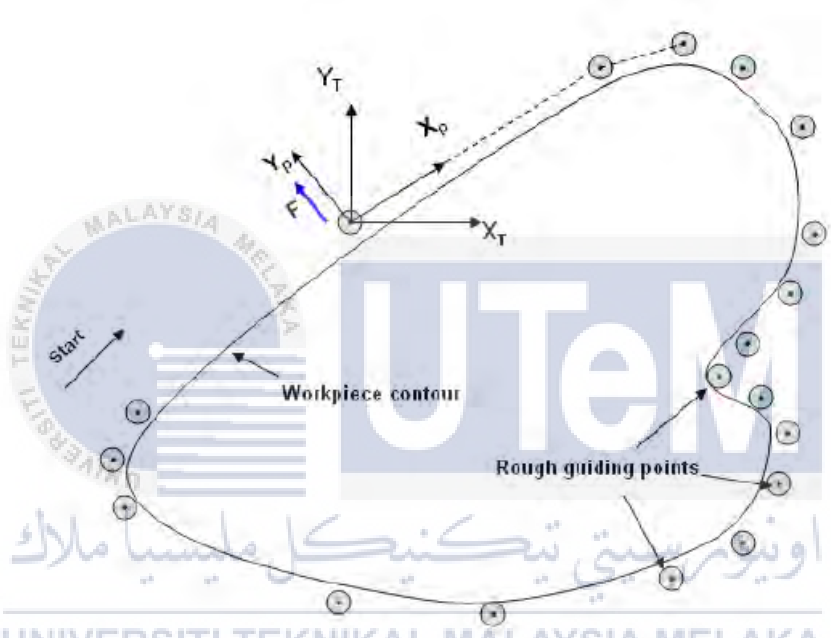

Figure 2.7: Lead-Through and Path Learning

- Automatic path-learning: a robot program based on the point drawn previously by the operator in the first stage. Figure 2.8 shows the path learning results [11]

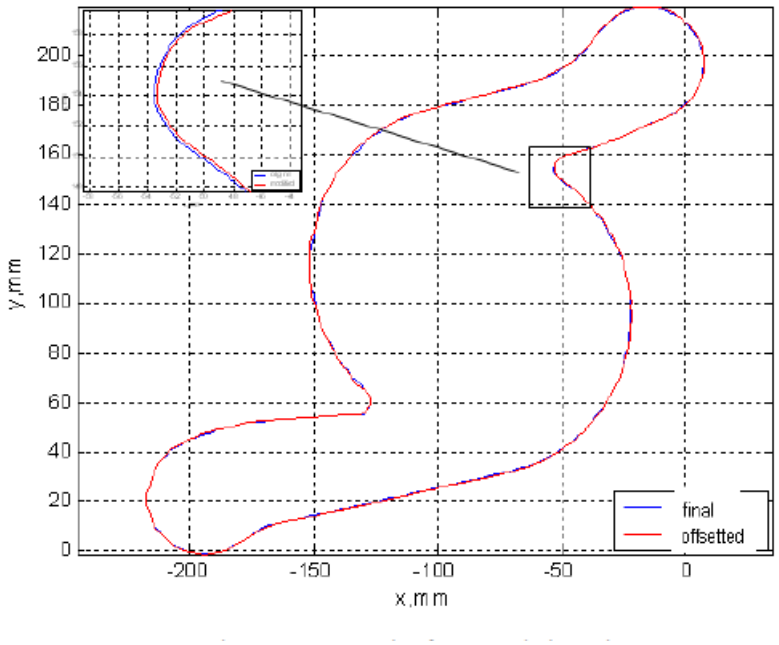

Figure 2.8: Results from Path-Learning [11]

- Post processing: it includes the position data filtration and reduction by the controller.

The last solution was proposed in [12-13], the existence of external sensors and devices was introduced for a more confidential and intellectual properties to be applied on robotic systems. The basic idea is to let the operator teach the robot a few approximated positions along the desired trajectory using a force control lead through, and then the robot executes a force feedback. Then post processing algorithms are applied to make further adjustments to the recorded path. Figure 2.9 shows the reduction of the guiding points learned for the purpose of avoiding two target points to be closed together. In addition Figures 2.10 and 2.11 show the learned path and the contacted force during the force control, respectively.

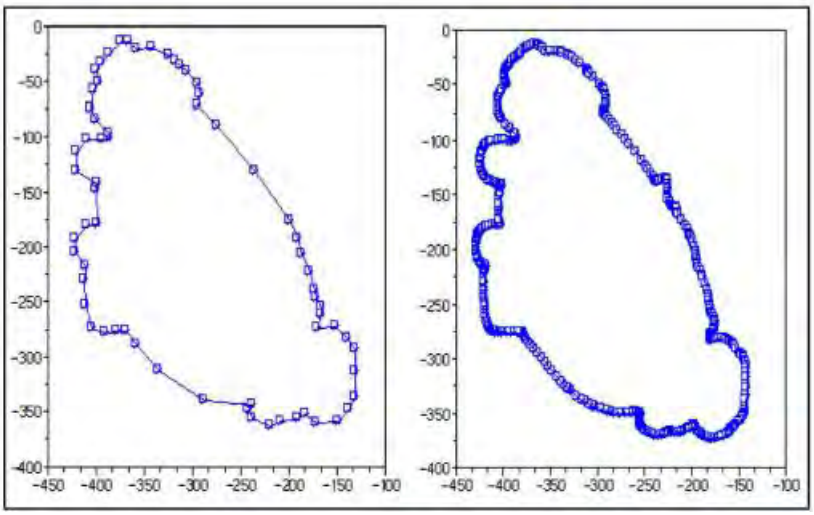

Figure 2.9: Learned Path Recorded (right) and Post Processed Path (left) [13]

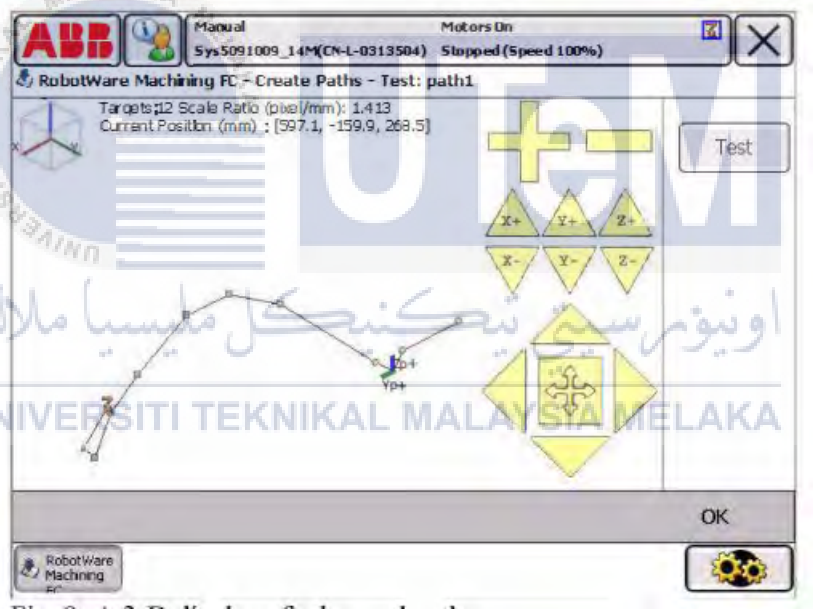

Figure 2.10: A 3-D Display of a Learned Path [12]

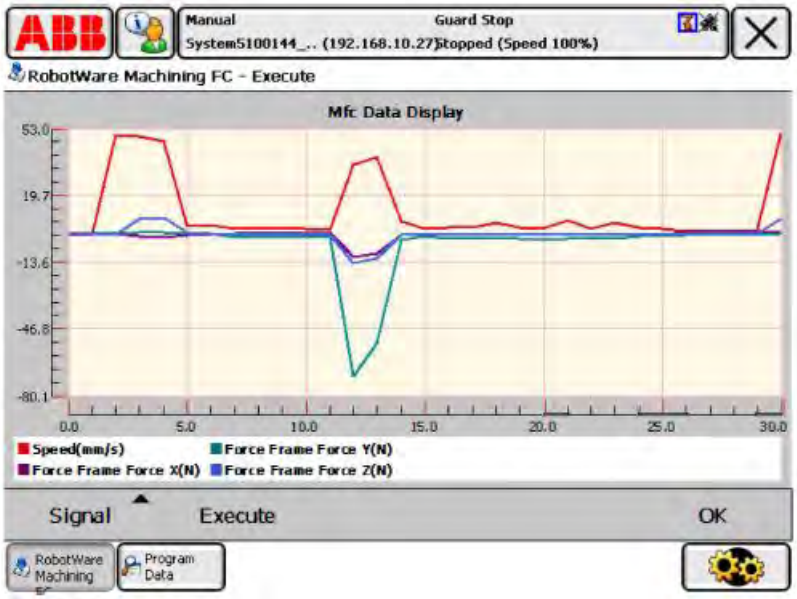

Figure 2.11: Display of the Contacting Force during the Controlled Motion [12]

2.4 Summary and Conclusion

Table 2.1 summarizes the available solution to the above mentioned problems.

Table 2.1: Comparison of the Available Solutions in the Lead-Through Programming Method

Available Solutions	Drawbacks of the design
A) Mouse Jog	It raises high demand to robot motion control, as it is attached to the robot arm rigidly, furthermore it a bit complicated for unskilled operators to calibrate a 6-DOF mouse and the robot coordinate system.
B) Robot-Puppet	As the robot follows the robot-puppet for the lead-through programming, its speed is constrained and then it can be overwritten with higher value. Meaning that it needs minimum robotics understanding. Additionally, it is matched to low accuracy application such as painting and spraying.
C) Programming by Demonstration	It must satisfy the requirements for potential robot operators who have knowledge about machining, basic robotic operations such as jogging and writing a simple robot program.
D) Path learning through a GUI and teach pendant	In order to program a robot with a teach pendant, an operator should setup the jogging condition, frame, motion mode, steps. Additionally, it involves experimental results and simulations on dummy doll before applying it on a virtual world application. Moreover, it requires the operator's knowledge to understand, analyze and interpret the obtained information on the Graphical user interface.

Most of the solutions proposed and recently applied to the industrial fields require operators' robotics knowledge. Additionally, they need to be calibrated by the operator each time the space-frame is changed. As a result, lead-through teaching method using external interfacing devices are accurate and more sophisticated in terms of data screening and analysis but still robotics are desired in SME (small and medium enterprises). Not to forget that a small or medium enterprise cannot offer operators who are acquainted with the robotics fields. Despite the fact that such a solution will trade off the accuracy and sophistication of the robotic field, it will greatly increase the easiness of dealing with robotics and make them human-friendly more than ever.

For this reason, in this project, low cost incremental encoders' feedback is proposed for a lead-through programming method. It is hypothesized that implementing such encoders for the lead-through programming method will fulfill the desired trajectory generation with amount of error which makes it applicable for low-accuracy demanding applications.

CHAPTER 3

METHODOLOGY

3.0 Overview

This chapter includes the experimental objectives of this project, the equipment used to conduct the experiment along with their respective parameters, the experimental setup, the procedures followed to fulfill the experiment' objectives and finally the precautions.

3.1 Lead-Through Robot Programming Method

A lead-through programming method is a term used to indicate the ability of the robot to physically learn its designed trajectory path. The system used in this experiment uses one controller (Arduino DUE) and two motors to be controlled alternatively, both of the two motors used were attached with rotary encoders. This system was designed to detect the number of pulses given by the attached encoders and then compare them to a saved number of pulses that were saved during the process of recording the trajectory path. i.e. the controller stops supplying either of the motors as soon as the number of the pulses is equal to the number of the saved pulses.

In Figure 3.1, the outline of the experiment is shown in details. The controller is connected to three switches record, home and play-back switches which send the signal to the controller. A feedback is given from the incremental encoders, attached to the motors, to the controller. And then the controller sends a signal to the motors driver based on the feedback received.

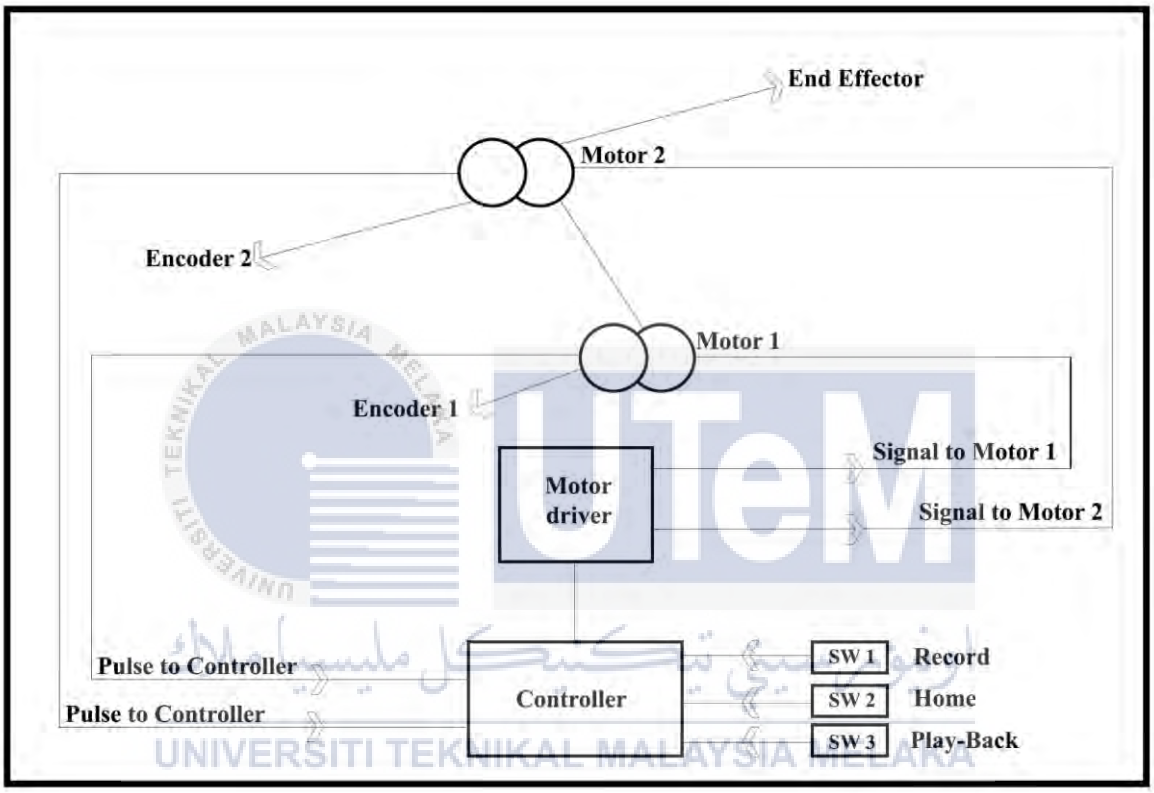

Figure 3.1: Lead-Through Schematic System Diagram

The flow chart in Figure 3.2 summarizes the lead through programming process proposed in this project.

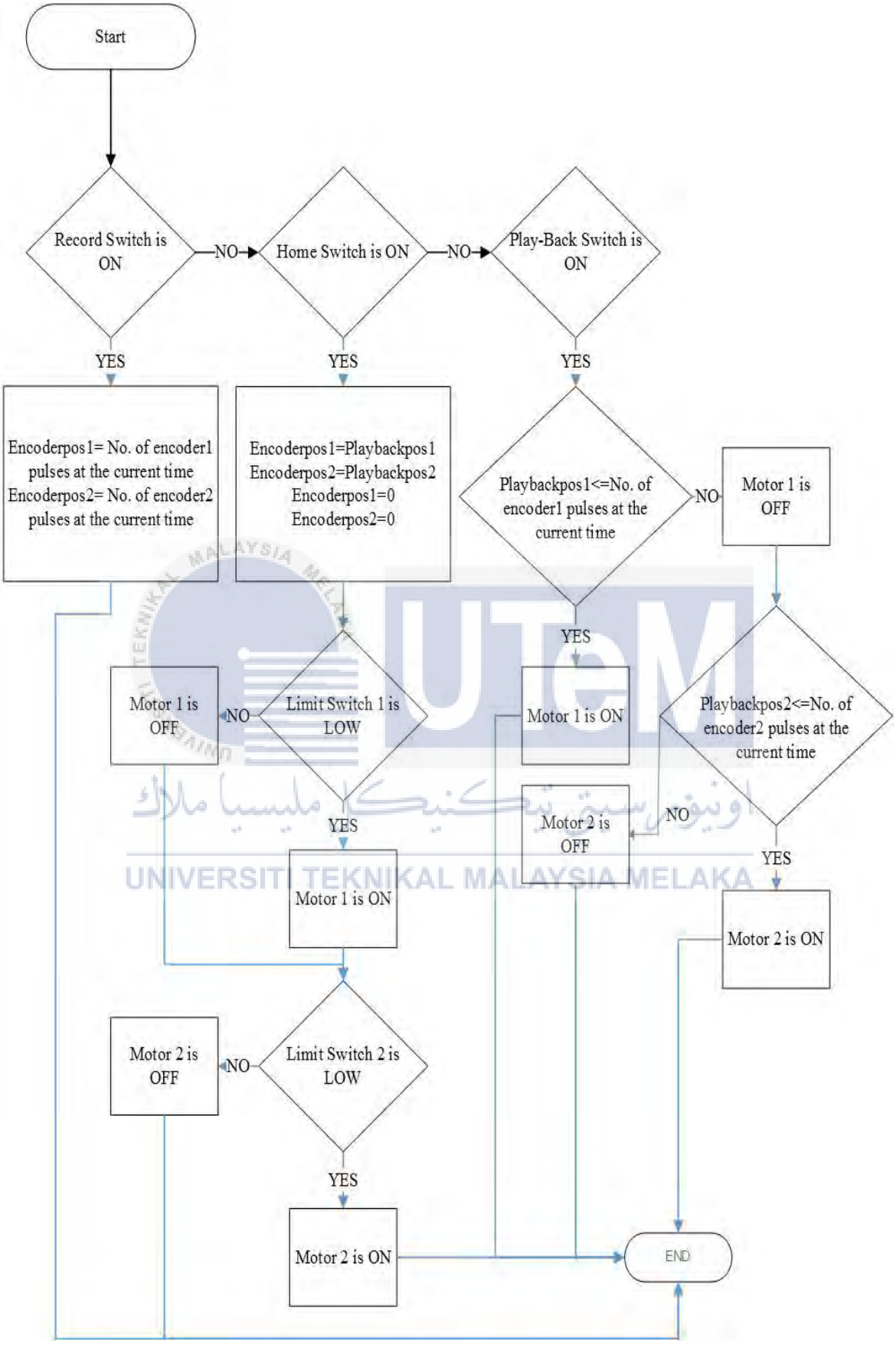

Figure 3.2: Lead-Through System Design

3.2 Experiments

The main objective of this experiment is to examine the performance parameters of the system, which are the error between the recorded and played-back positions, and the accuracy of the generated trajectory by comparing the X-Y coordinates of both points. Moreover is to test the speed of the processor by which it detects the pulses given by the rotary encoders of both motors and the precision of the trajectory generation as well.

3.2.1 Experimental Equipment and Parameters

Follows is the list of the experimental equipment used to conduct the experiment:

1) Arduino board, Due
2) Dual Channel 10A DC Motor Driver, Cytron MDD10A
3) 2 Geared DC motors
4) 12V External Power Supply
5) 2 5V Quadrature Hall Effect Encoders
6) 3 Switches
7) 3 LEDs
8) Breadboard
9) Male Wire Jumpers

3.2.2 Experimental Set Up

In order to test the accuracy, pulses detection speed, error and precision several types of data needs to be collected from the system and analyzed. X-Y coordinates (cm) of

both recorded and played-back trajectories were taken, by taking reached values on the gridded white-board as shown in Figure 3.3. Additionally, the pulses (pulse/time) given by both encoders of both motors were taken using the controller interrupts functions as shown in the schematic diagram in Figure 3.1. The position of both links (in degree) was obtained from the number of pulses and tabulated as well, by placing a compass at the center of the joints shown in Figure 3.3.

As shown in Figure 3.3 a 91cmX62cm white-board was used to fix the arm on and draw the final position to be reached. Additionally, two links of 30cm length were used to link the two motors and form the two degree of freedom arm. The points (22cm, 50cm) and (77cm, 50cm) on the white-board were chosen as the initial and final positions respectively. Moreover, the links were fixed to the motor's rear-shaft with two screws, one for each link, to reduce the amount of the mechanical loose encountered, despite the fact that there was still a mechanical loose due to the loose in the gearing system of the geared DC motor itself.

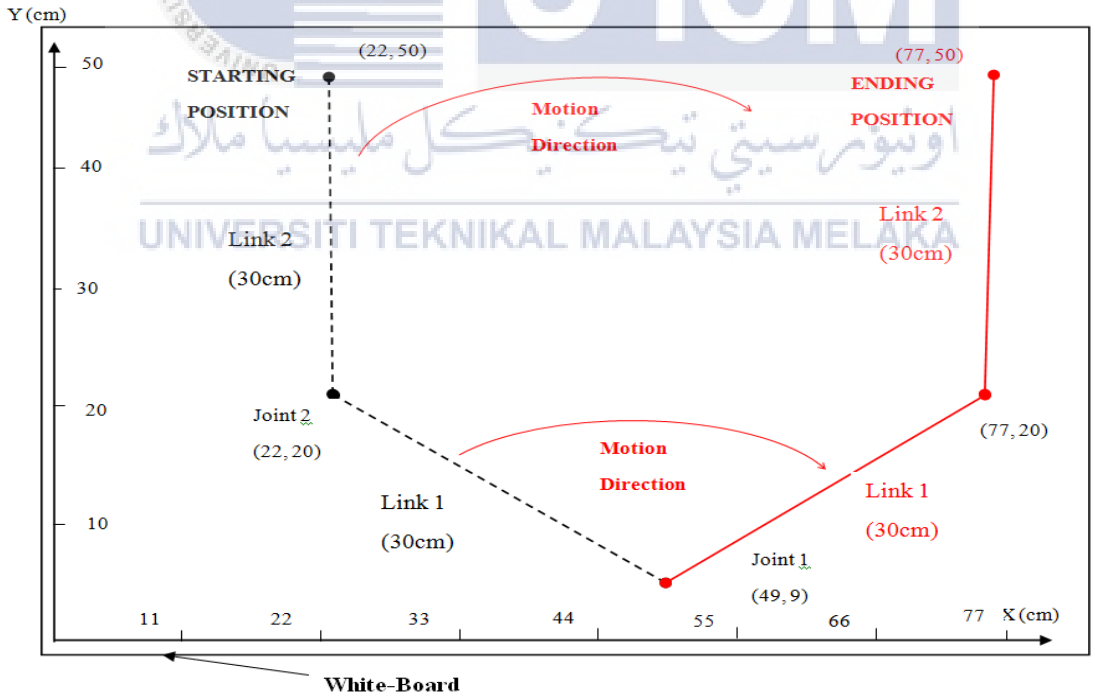

Figure 3.3: Planned Experimental Setup

Figure 3.4 shows the real experimental setup conducted as per the setup planned on Figure 3.3.

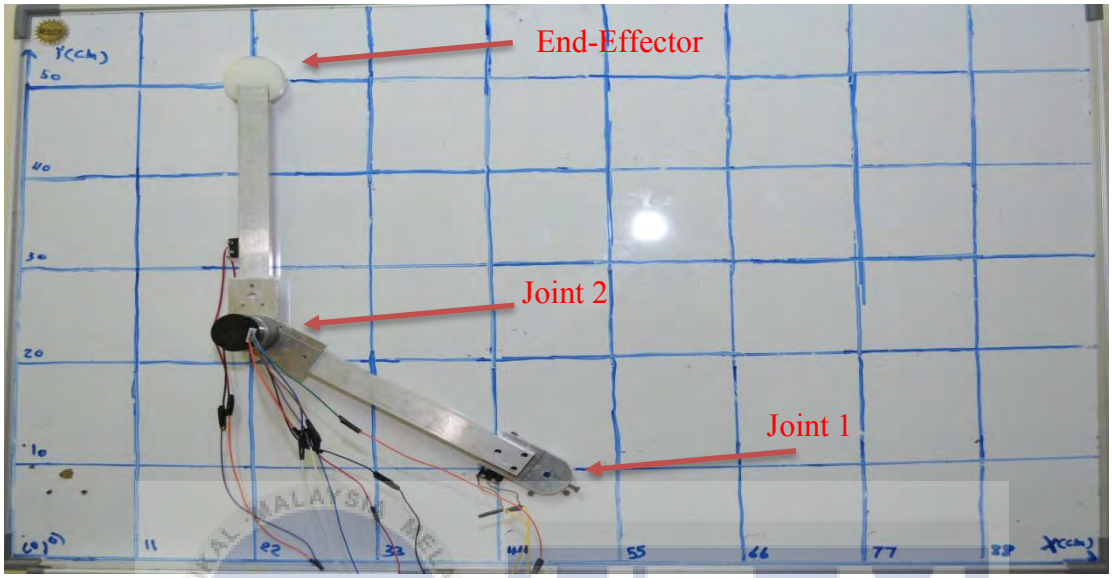

Figure 3.4: Real Experimental Setup

3.2.3 Procedures

Figure 3.5 illustrates the constructed circuit for the lead-through robots' programming method indicating the three buttons used for the three different trajectory generation stages which are record, play-back and home buttons.

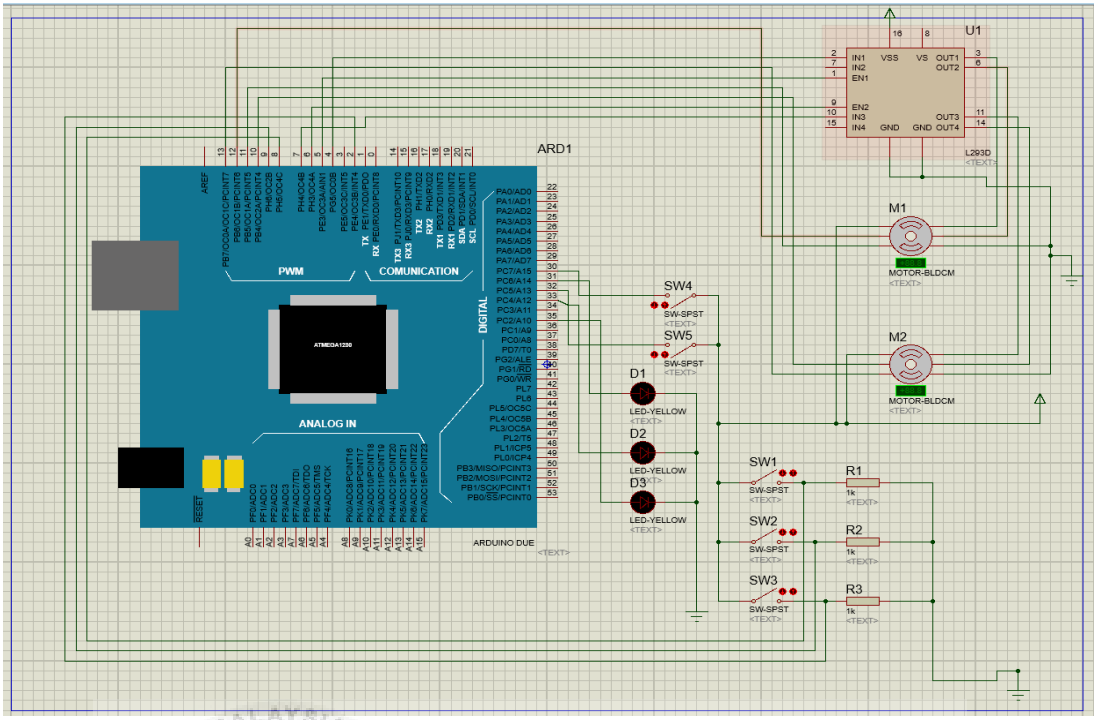

Figure 3.5: Constructed Circuit

The controller used in this experiment is Arduino DUE board with the specifications indicated in Table 3.1.

Table 3.1: Arduino DUE Specification

Item/Paramete	Specification
Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O pins	54 (12 provide PWM)
Analog Input Pins	12
Analog Outputs Pins	2 (DAC)
Total DC Output Current	130mA
DC Current for 3.3V Pin	800mA
DC Current for 5V Pin	800mA

Flash Memory	512KB
SRAM	9KB
Clock Speed	84MHz
Length	101.52mm
Width	53.3mm
Weight	36g

The two motors were used are of the specifications indicated in Table 3.2.

Table 3.2: Geared DC Motor Specifications

Parameter	SPG30-300
Rated Voltage	12VDC
No Load Speed	7000rpm
No Load Current	70mA
Rated Torque	1176
Rated Current	410mA
Rated Speed	12
Stall Torque	23.5mN.m
Stall Current	1.8A
Gear Ration	270:1
Encoder Resolution	3/rear shaft revolution
Encoder Pulses/Main shaft revolution	810

For the two motors a 10 Amperes driver was used with the below specifications indicated in Table 3.3.

Table 3.3: MDD10A Motor Driver Specification

Parameters	**Min**	**Typical**	**Max**	**Unit**
Power Input Voltage	5	-	25	V
I_{MAX} (Maximum Continuous Motor Current)	-	-	10	A
I_{PEAK} – (Peak Motor Current) *	-	-	30	A
V_{IOH} (Logic Input – High Level)	3	-	5.5	V
V_{IOL} (Logic Input – Low Level)	0	0	0.5	V
Maximum PWM Frequency	-	-	20	KHz

The three stages of the experiment are as follows:

1. **Record Stage**

This stage is initiated by a push button, shown as SW1 in Figure 3.3. The first stage is where the operator switches on the SW1 and manually moves the end-effector of the robotic arm. On the Cartesian space drawn on the white-board the final position was indicated by (22cm, 50cm) and (77cm, 50cm) was indicated as the final position.

Before starting this stage, the experiment was setup as shown in Figures 3.1 and 3.2, including the position of the arm and the power connection to both Arduino board and the motor driver, additionally the Arduino was connected to a laptop for the pulses of the encoder to be monitored.

Then the operator moves the end-effector of the robotic arm from the initial to the final position and the processor automatically records the movement data during the manual generation of the trajectory. When the final position is reached the operator should switch off the SW1 and move to stage two.

2. Home Stage, Initial Position

In this stage the Home button is switched ON, labeled as SW2 on Figure 3.3, and the robotic arm automatically goes back to its initial position, the movement was ceased whenever the limit switches are turned ON by having the arm's links hitting them. In this stage the operator's only required action is to press the home button, and the arm goes back by itself to the initial position. After the robotic arm reaches its initial position the operator should depress the home button and starts the next stage

3. Play-Back Stage

In this stage, the operator switches on the play-back button, labeled as SW3 in Figure 3.3, then the robotic arm repeats the same motion made by the operator. In this stage the operator should only press on the play-back button and everything recorded is repeated by the arm automatically, i.e. the operator's action is not required on the robotic arm to repeat the motion, it is all done automatically.

Figure 3.4 is a flow chart of the process's procedures involved in generating a trajectory of the robotic arm using a lead-through programming method.

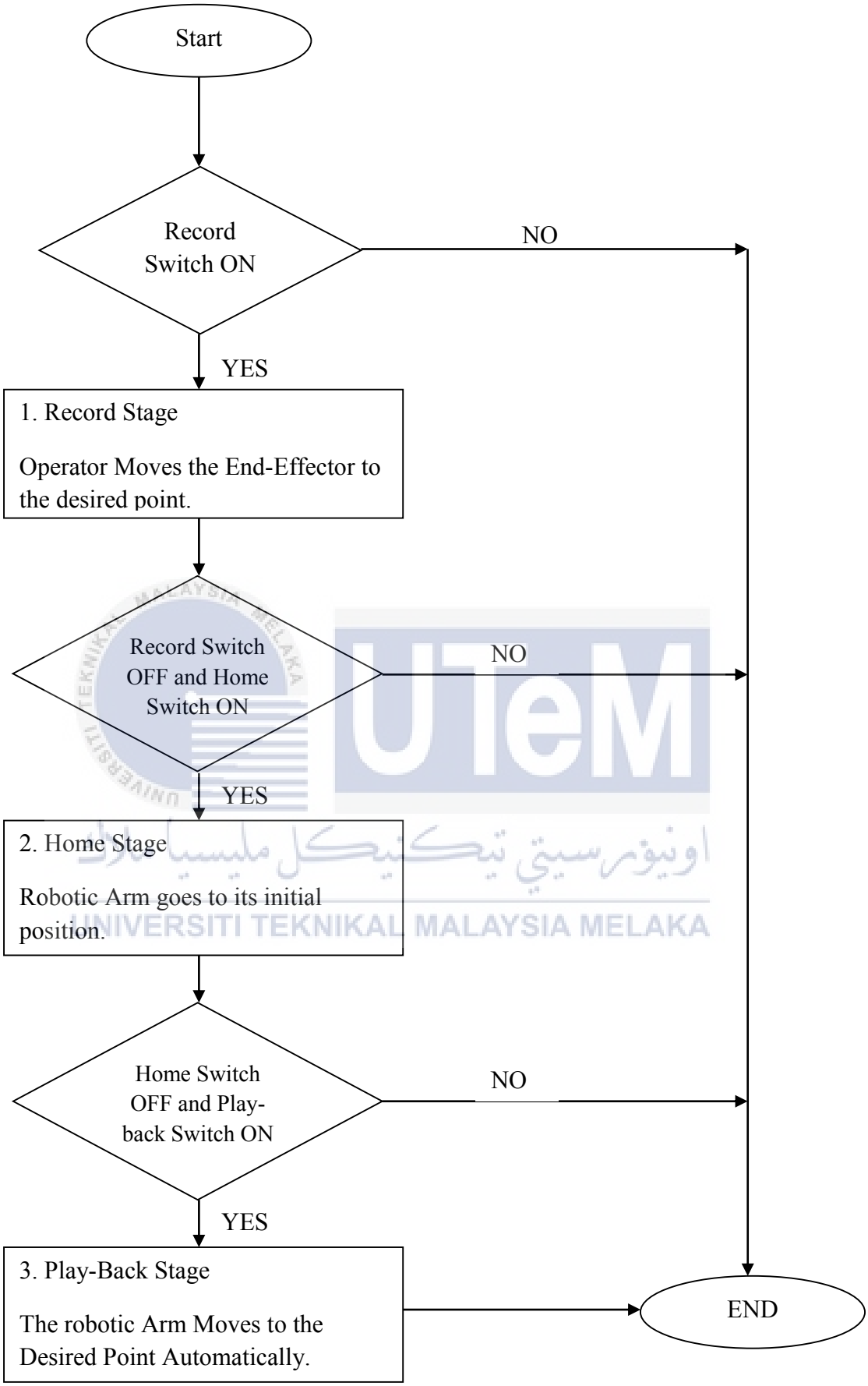

Figure 3.6: Experimental Procedures

3.2.4 Safety Precautions

The three buttons should not be ON at the same time, as no action will be taken if such a case has taken place. Additionally, for a new record of new final position the RESET button on the Arduino DUE board itself needs to be pressed when the robotic arm is at its initial position. Moreover, the operator needs to be careful regarding the space between him and the robotic arm itself. On the other hand, the robotic arm trajectory path must be out of any possible obstacles as they may cause damage to the gearing system of the geared DC-motors used.

3.2.4 Precautions on validity issues

The system designed is proposed to be used for applications that do not require a high level of accuracy as it uses inexpensive incremental encoders for the purpose of the cost-reduction, which may result in a systematic error that reduces the accuracy level of the proposed system. Additionally, a gross error may occur as well due to the dependency of the trajectory generated on the operator himself. Lastly, the replicability of the proposed system may show a deviation in some of the repeated trajectories due to the usage of inexpensive geared DC motors.

3.3 Method of Analysis

In order to evaluate the performance parameters from the obtained data several methods were followed to analyze the data and compare them. First of all, error was calculated using equation 3.1, where Q_{rec} is the recorded position and Q_{played} is the played-back position.

$$\% \, Error = \frac{Q_{rec} - Q_{played}}{Q_{rec}} \times 100\% \quad (3.1)$$

Moreover, Accuracy was considered and tabulated for each motor trajectory using equation 3.2

$$\% \, Accuracy = 1 - \ Error \ \times 100\% \quad (3.2)$$

The position of the links (in degrees) was obtained from equation 3.3

$$Q = \frac{360 \times No.of \ Pulses}{3240} \quad (3.3)$$

Precision of the system was examined by repeating the same recorded position (119.3^o) for fifty times and check the consistency of the system through-out the fifty trails.

CHAPTER 4

RESULTS AND DISCUSSION

This chapter includes the obtained results and the discussion of the respective results.

4.1 Record and play-back stages comparison

After the experimental setup was prepared an experiment was conducted and the pulses given by both encoders for the two different stages of the trajectory generation, record and play-back, along with their respective positions in respect with time were recorded and tabulated as shown in the appendix.

Figure 4.1 shows the plotted graph from the data taken from Tables 4.1 and 4.2 in the appendix, which compares the trajectory generated during the record stage and the play-back stage in motor 1.

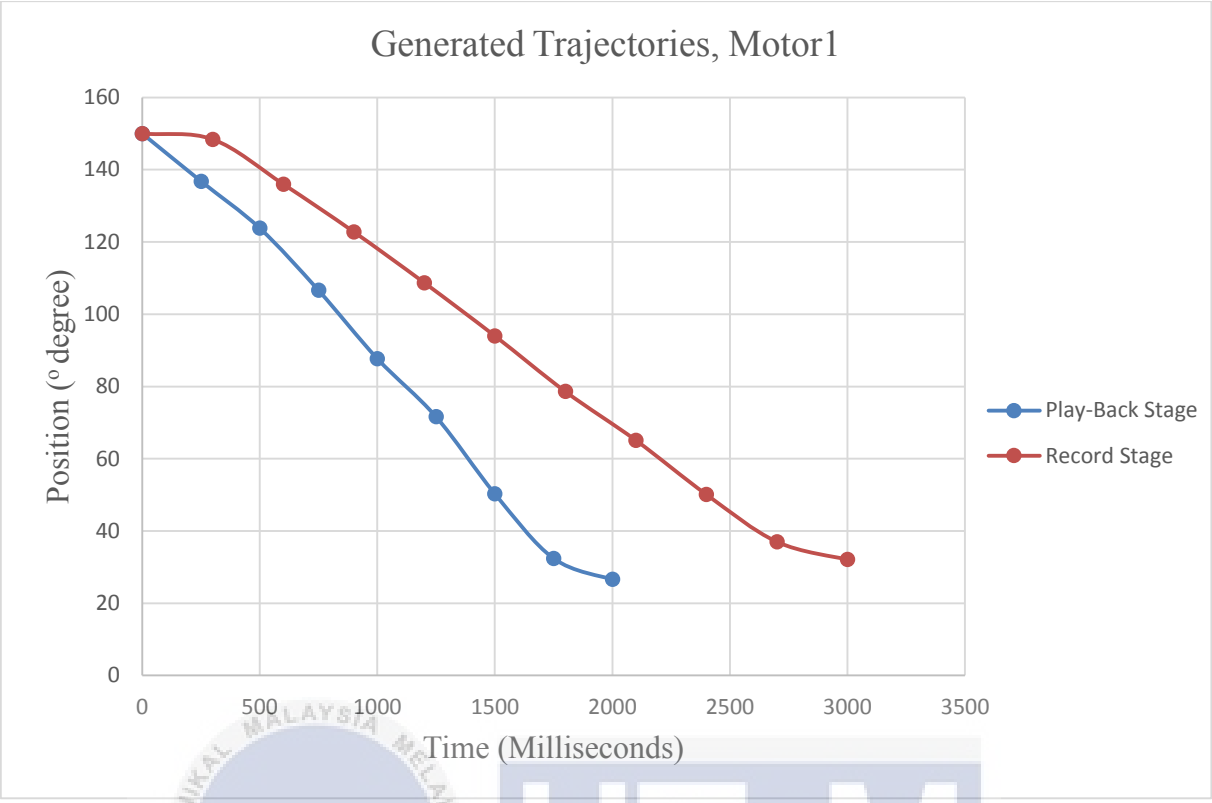

Figure 4.1: Generated Trajectories Comparison, Motor 1

As can be seen from Figure 4.1 the time taken for the record stage is 3000 milliseconds depending on the speed of the operator's hand moving the robotic arm. In this experiment the speed of the operator's hand applied on motor 1 can be calculated as in Equation 4.1;

$$Speed_{rec,\ motor1} = \frac{Degree\ traveled}{Time\ taken} = \frac{117.9^0}{3sec} = 39.3^0/sec \quad (4.1)$$

On the other hand the the time taken for the play-back stage is only 2000 millisecond as the motor is given only 150 pulse width modulation analog input as the full speed gives 12 rpm which gives a speed as in Equation 4.2;

$$12\ rpm \times 360^0 = 4320^0/min \times 60 = 72^0/sec \quad (4.2)$$

For this reason only 150 PWM is given to the motor which produces the following speed shown in Equation 4.3;

$$Speed_{played,\ motor1} = \frac{72^0/sec \times 150}{255} = 42.4^0/sec \quad (4.3)$$

Based on Equations 4.1 and 4.3 the time difference causes such a deviation in the record and play-back graphs drawn with respect to time.

As can be seen from the graph drawn in figure 4.1 in addition to the time gap there are slight fluctuations in the trajectory generated during the play-back stage. These fluctuations are caused as a result of the mechanical loose that results from the teeth slip of the geared DC motor.

Figure 4.2 shows the Plotted graph from the data taken from Tables 4.3 and 4.4 in the appendix, which compares the trajectory generated during the record stage and the play-back stage in motor 2.

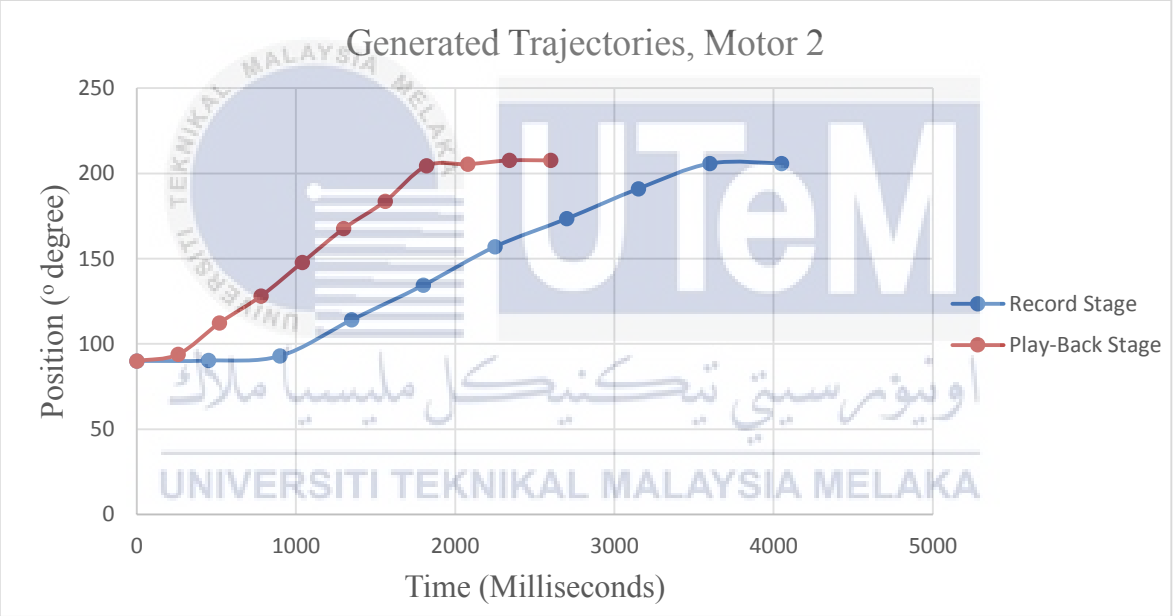

Figure 4.2: Generated Trajectories Comparison, Motor 2

As can be seen from Figure 4.2 the time taken for the record stage is 4050 milliseconds depending on the speed of the operator's hand moving the robotic arm. In this experiment the speed of the operator's hand applied on motor 2 can be calculated as in Equation 4.4

$$Speed_{rec,\ motor2} = \frac{Degree\ traveled}{Time\ taken} = \frac{115.8^0}{4.05sec} = 28.6^0/sec \quad (4.4)$$

On the other hand the time taken for the play-back stage is only 2600 millisecond as the motor is given only 150 pulse width modulation analog input as the full speed gives 12 rpm which gives a speed as in Equation 4.2;

$$12\ rpm \times 360^0 = 4320^0/min \times 60 = 72^0/sec \tag{4.5}$$

For this reason only 150 PWM is given to the motor, which produces the following speed shown in Equation 4.5;

$$Speed_{played,\ motor2} = \frac{72^0/sec \times 150}{255} = 42.4^0/sec \tag{4.6}$$

Based on equations 4.4 and 4.4 the time difference causes such a deviation in the record and play-back graphs drawn with respect to time.

Moreover, it is obvious that in motor 2 the same fluctuations occurs for the mechanical loose occurred in motor 1.

For further visualization, Figures 4.3 and 4.4 show the real experiment results of both the recorded trajectory and the played-back trajectory respectively.

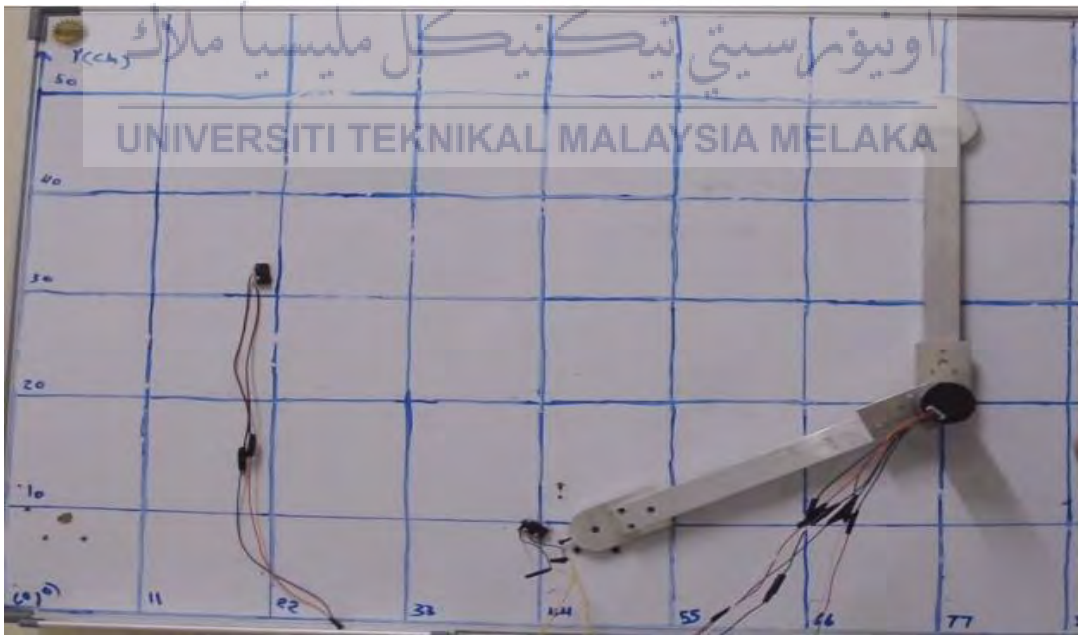

Figure 4.3: Recorded Trajectory

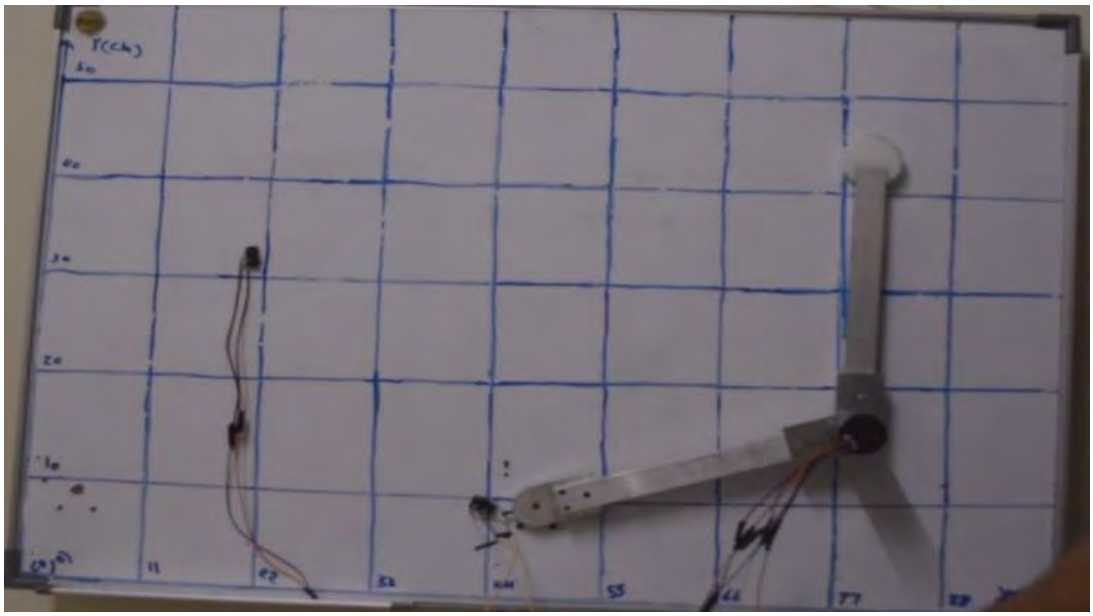

Figure 4.4: Played-Back Trajectory

Figures 4.5-4.9 illustrate the robotic arm undergoing both record and play-back stages. These instantaneous positions were taken and captured with an interval of one second as shown in the figures below.

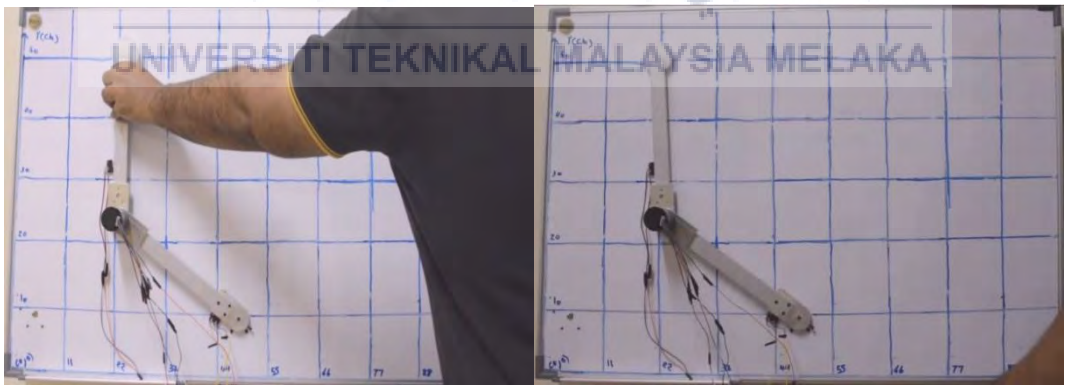

Figure 4.5: Record Stage, to the Left, and Play-Back Stage, to the Right, at T=1sec

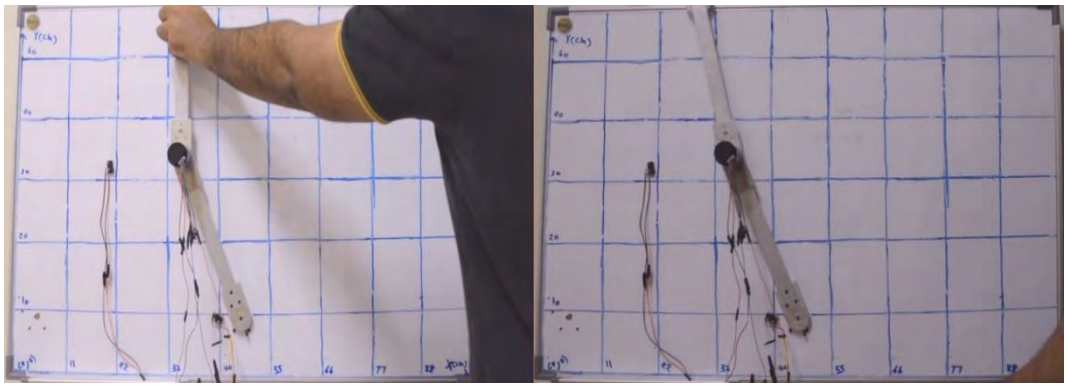

Figure 4.6: Record Stage, to the Left, and Play-Back Stage, to the Right, at T=2sec

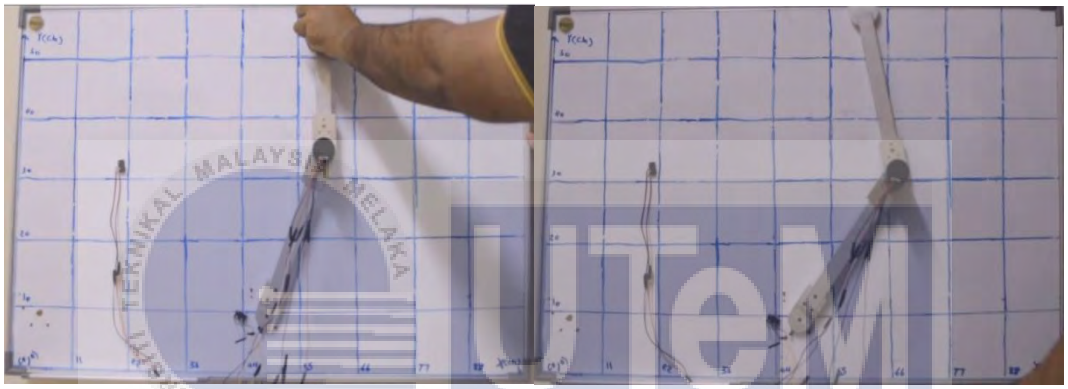

Figure 4.7: Record Stage, to the Left, and Play-Back Stage, to the Right, at T=3sec

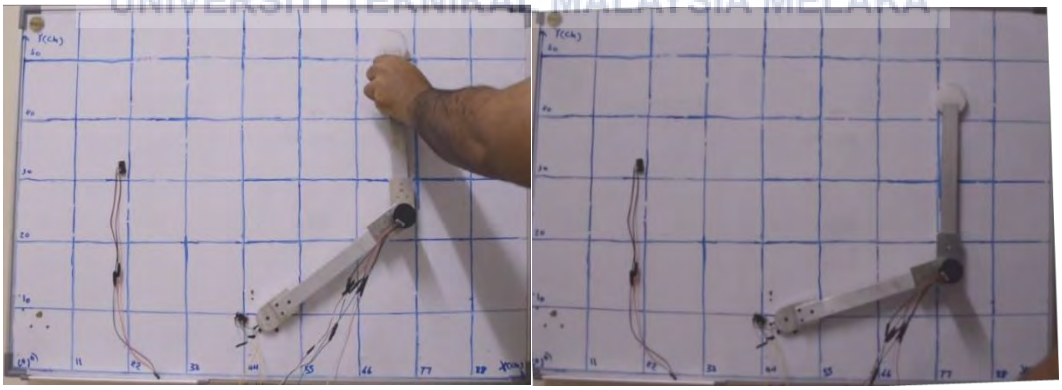

Figure 4.8: Record Stage, to the Left, and Play-Back Stage, to the Right, at T=4sec

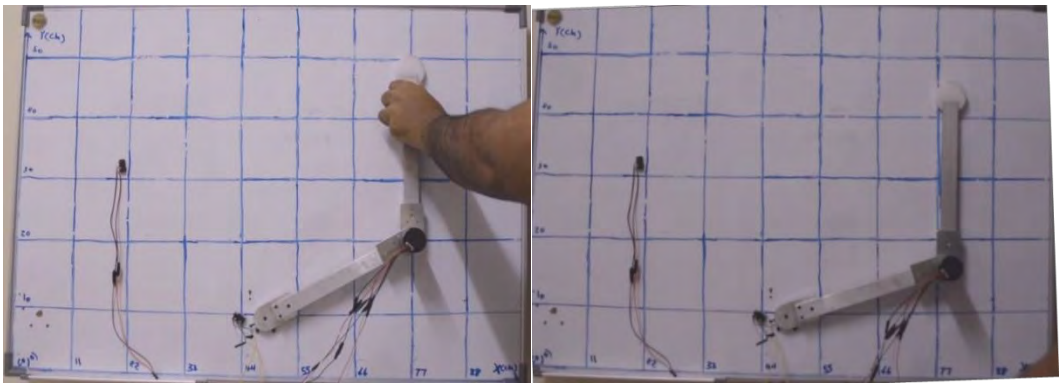

Figure 4.9: Record Stage, to the Left, and Play-Back Stage, to the Right, at T=5sec

As can be seen from Figures, 4.5-4.9, both recorded and played-back trajectories are following the same trajectory paths from second 1-4. On the other hand, at the T=5 the played-back trajectory is already at its final position unlike the recorded trajectory that entirely depends on the operator's hand-speed. However, the speed of the played-back trajectory can be manipulated using the pulse width modulation given by the controller to the motors. In this experiment 150 of 255 pulse width modulation was applied to both motors.

4.2 Errors and Accuracy

Errors and accuracy are determined based on Equations 3.1 and 3.2 respectively. Each hall-effect sensor of each encoder gives three pulses per rear shaft revolution. Gear ratio of each motor is 270:1, so 810 pulses are given per one main shaft revolution. Both of the sensors output states, positive and negative states, are considered, so $810\ \text{pulses} \times 2\text{states} = 1620\ \text{pulses/main shaft revolution}$. Since two hall effect sensors exist for each motor then the total number of pulses given is $1620 \times 2 = 3240\ \text{pulses/main shaft rev}$.

Tables, 4.1 and 4.2 show the errors and accuracy of both motors.

Table 4.1: Motor One Error and Accuracy.

Trial	Motor 1					
	$Pulses_{rec}$	$Pulses_{played}$	$Q1_{rec}$ (^{o} degree)	$Q1_{played}$ (^{o} degree)	%Error	%Accuracy
1	1064	1118	118.2	124.2	5%	95%
2	1064	1097	118.2	121.9	3.1%	96.9%
3	1064	1097	118.2	121.9	3.1%	96.9%
4	1064	1099	118.2	122.1	3.3%	96.7%
5	1064	1105	118.2	122.8	3.9%	96.1%
6	1064	1110	118.2	123.3	4.3%	95.7%
7	1064	1099	118.2	122.1	3.3%	96.7%
8	1064	1103	118.2	122.6	3.7%	96.3%
9	1064	1105	118.2	122.8	3.9%	96.1%
10	1064	1113	118.2	123.7	4.7%	95.3%
Mean	-	-	-	-	3.83%	96.17%

As can be seen from Table 4.1 there is a small deviation between the recorded position $Q1_{rec}$ and the played-back position $Q1_{played}$ that gave an average error of 96.17% with an accuracy of 3.83%.

Table 4.2: Motor Two Error and Accuracy.

Trial	Motor 2					
	$Pulses_{rec}$	$Pulses_{played}$	$Q2_{rec}$ (^{o} degree)	$Q2_{played}$ (^{o} degree)	%Error	%Accuracy
1	1087	1120	120.8	124.4	2.9%	97.1%
2	1087	1129	120.8	125.4	3.8%	96.2%
3	1087	1102	120.8	122.4	1.3%	98.7%
4	1087	1106	120.8	122.9	1.7%	98.3%
5	1087	1111	120.8	123.4	2.6%	97.4%
6	1087	1105	120.8	122.8	1.7%	98.3%
7	1087	1106	120.8	122.9	1.7%	98.3%
8	1087	1105	120.8	122.8	1.7%	98.3%
9	1087	1110	120.8	123.3	2.1%	97.9%
10	1087	1108	120.8	123.1	1.9%	98.1%
Mean	-	-	-	-	2.14%	97.86%

As can be seen from the Table 4.2 there is a small deviation between the recorded position $Q2_{rec}$ and the played-back position $Q2_{played}$ which gives an average error of 97.86% with an accuracy of 2.14%.

Based on Tables 4.1 and 4.2 it is noticeable that the error occurs in motor one is a bit higher than motor two due to two main reasons. The first reason is the higher torque applied to motor 1 as the distance between the center of motor 1 and the end-effector is 60cm, 30cm length of both links, which is double the distance between the center of motor 2 and the end-effector, 30cm of the second link only. Regardless of the amount of force applied to the end-effector the error will still be higher on motor one as the torque is calculated by equation 4.7;

$$Torque = force\ applied\ (F) \times perpendicular\ distance\ (d) \quad (4.7)$$

The second reason is that the weight carried by motor 1 is higher than the weight carried by motor 2. The first motor, carries both links of the robotic arm along with the second motor as shown in Figure 3.3, highlighted as joint 1. For this reason the mechanical

loose exists as a result of the gear slip in the first DC geared motor will noticeably affect the first motor more than the second motor.

4.3 Precision

After the consistency test of fifty trials was conducted a normal distribution curve, normal bell curve, was drawn for both motor 1 and motor 2 as shown in Figures 4.10 and 4.11 respectively.

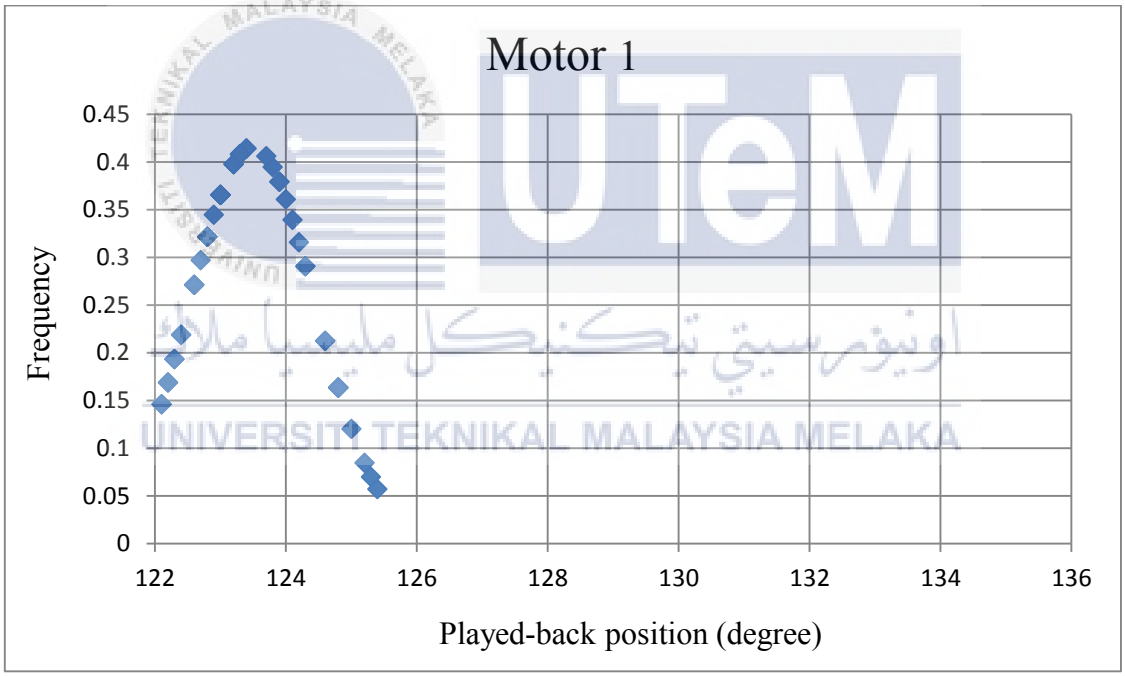

Figure 4.10: Normal Bell Curve, Motor One

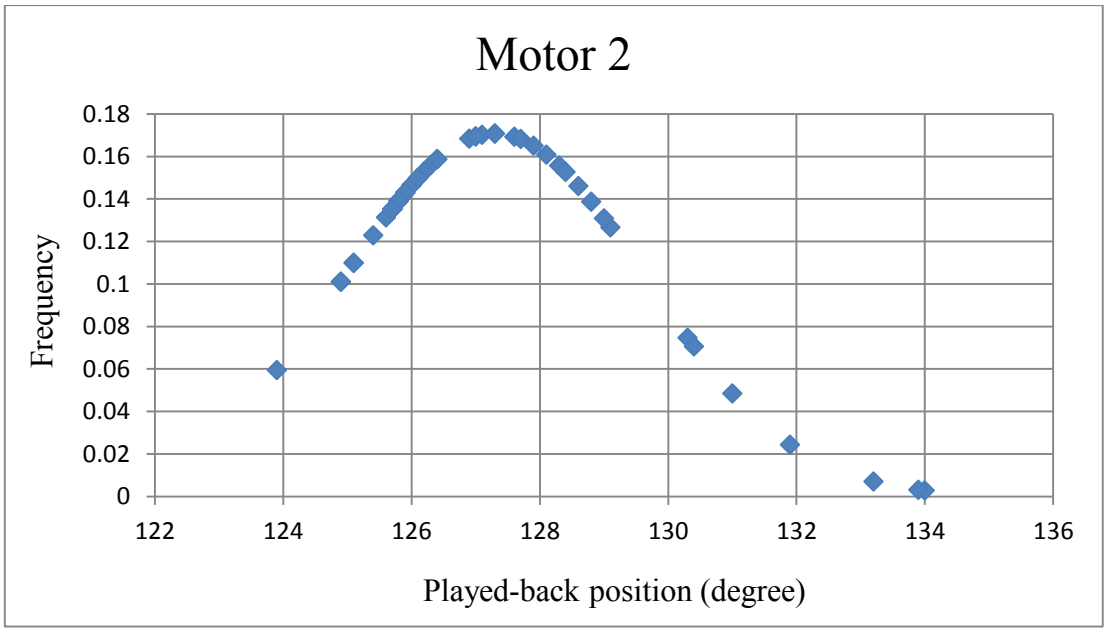

Figure 4.11: Normal Bell Curve, Motor Two

Based on Figure 4.10 shown, the values obtained from the fifty trails are distributed evenly from 122^0 to 125.5^0 with a standard deviation of 0.9593 and a mean of 123.488^0. Due to the error indicated above in table 4.1, the values obtained are not distributed in a high portion in one part of the graph, which indicates that every time the experiment is conducted a slightly different value is obtained. On the other hand in figure 4.11 shown above the values obtained are intensively distributed within the range of 125^0 to 129.5^0 with a standard deviation of 2.33583 and a mean of 127.294^0. The values of the lowest occurrence frequency are fallen to the far right and left areas of the graph in both 4.10 and 4.11.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

This chapter includes the conclusion and the suggested future work.

5.1 Conclusion

In this project a lead-through programming method is developed for offering workers and operators who lack the basic knowledge and experience in robotic systems to easily generate a trajectory for a robotic system in a human-friendly manner. Error between the recorded and played-back trajectory, accuracy, precision and repeatability are tested and results are tabulated and compared for both recorded and played-back trajectories. Based on the obtained results, it is clear that a lead-through programming method can be implemented in SMEs (Small and Medium Enterprises) as the designed system shows reliability in repeating the same manually recorded trajectory with the accuracy of 96.17% and 97.86% for motor 1 and 2, respectively. Validity of the designed system can be observed in the deviation of the actual position from the desired position as it gives an accepted amount of error of 3.83% and 2.14% for motor one and two, respectively.

A prototype of the robotic arm is built with two geared DC motors which are attached to two links that forms a robotic arm with two DOF (degree of freedom). Then,

the experiment is conducted, by generating a trajectory to a specified desired position and then the same trajectory is repeated by the controller to analyze the performance parameters stated in the objectives of the thesis. The proposed idea is illustrated in terms of measuring the angel (in degree) traveled by the robotic arm in both recorded and played-back trajectories. Then both traveled angles are drawn in the same graph for both motors and relevant data is retrieved and analyzed from the drawn graphs.

One of the limitations of the designed system is that it lacks in accuracy in comparison with the systems that use interface devices such as teach pendants, but on the other hand it increases the application and usefulness of the robotics implementation in SMEs.

5.2 Future Work and Recommendation

The designed system will be built with a wider range of applications in industrial applications by increasing the degree of freedom to up to four DOF. Moreover, optical sensors will be implemented to fulfill a full rotation of the installed geared DC motors, which will give the system the ability to fully rotate a 360^0 for each joint, as the limit switch limits the rotation of the motor to a specified angle as soon as the link hits the limit switch. Lastly, the system will be designed in fully 3D motion instead of fixing it on a white board which produces only a 2D motion.

REFERENCES

[1] E. Kahale, P. Castillo, and Y. Bestaoui, "Minimum time reference trajectory generation for an autonomous quadrotor," *2014 Int. Conf. Unmanned Aircr. Syst.*, pp. 126–133, May 2014.

[2] International Federation of Robotics, Automation and robotics. Available at: https://www.fidelityworldwideinvestment.com/middle-east/news-insight/21-century-themes/automation-and-robotics.page [accessed 27 November 2014]

[3] N. Kubota, Y. Nojima, I. Adji, and F. Kojima, "Interactive Trajectory Generation using Evolutionary Programming for A Partner Robot," pp. 335–340, 2003.

[4] M. XRC, M. DX100 and F. RJ3iB, 'Robots are taking over -- the really dangerous jobs', *Robots.com*, 2014. [Online]. Available: http://www.robots.com/articles/viewing/robots-are-taking-over-the-really-dangerous-jobs. [Accessed: 11- Nov- 2014].

[5] John J. Craig, Introduction to Robotics Mechanical and control. 3rd edition. United States of America: Pearson Prentice Hall, 2005.

[6] Bara.org.uk, 'Robot Programming Methods | BARA', 2014. [Online]. Available: http://www.bara.org.uk/robots/robot-programming-methods.html. [Accessed: 11- Nov- 2014].

[7] V. Delsart and T. Fraichard, "Tiji, a generic trajectory generation tool for motion planning and control," *2010 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pp. 1439–1444, Oct. 2010.

[8] L. Pa and T. H. Speeter, Transformation of Human Hand Positions for Robotic Hand Control. pp. 1758-1763, 1989.

[9] W. Eakins, G. Rossano, and T. Fuhlbrigge, "Lead-through robot teaching," *2013 IEEE Conf. Technol. Pract. Robot Appl.*, no. c, pp. 1–4, Apr. 2013.

[10] L. Qi, D. Zhang, J. Zhang, and J. Li, "A lead-through robot programming approach using a 6-DOF wire-based motion tracking device," *2009 IEEE Int. Conf. Robot. Biomimetics*, pp. 1773–1777, Dec. 2009.

[11] Z. Pan, "Robotic machining from programming to process control," *2008 7th World Congr. Intell. Control Autom.*, pp. 553–558, 2008.

[12] G. Zhang, J. Wang, and J. Ge, "Robotic path learning with Graphical User Interface," *Ieee Isr 2013*, pp. 1–4, Oct. 2013.

[13] G. Zhang, "A force control assisted robot path generation system," *2008 IEEE Int. Conf. Autom. Sci. Eng.*, pp. 528–533, Aug. 2008.

[14] L. G. Timothy, Lake Elmo and Minn, "Lead-Through Robot Programming System," U.S. Patent 5,880,956, March 9, 1999

Appendix A

Table 1: Motor one, Record Stage

Time (Milliseconds)	Motor 1	
	$Pulses_{rec}$	Q_{rec} (^{o} degree)
0	0	150
300	14	148.4
600	126	136
900	245	122.8
1200	372	108.7
1500	504	94
1800	643	78.6
2100	764	65.1
2400	899	50.1
2700	1017	37
3000	1061	32.1

Figure 1 below illustrates the plotted graph of position versus time from table 1, for the trajectory generated manually by the operator.

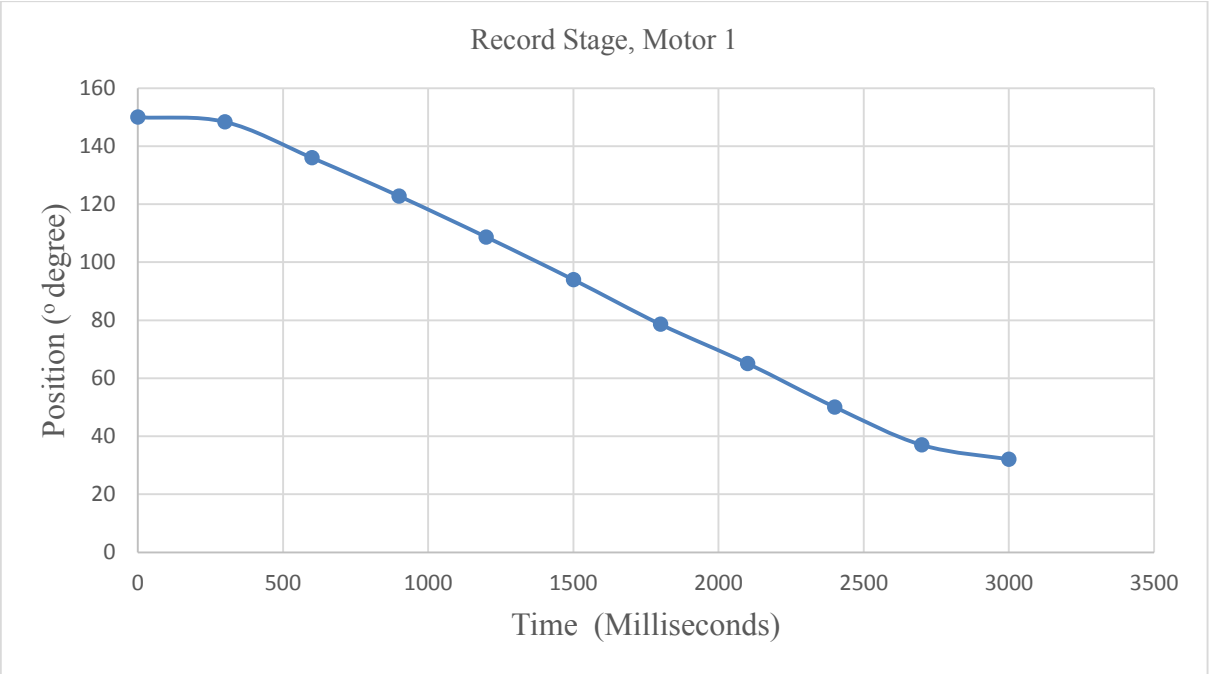

Figure 1: Trajectory Generated during Record Stage, Motor one

Table 2: Motor one, Play-Back Stage

Time (Milliseconds)	Motor 1	
	$Pulses_{played}$	Q_{Played} (Degree)
0	0	150
250	119	136.8
500	236	123.8
750	391	106.6
1000	561	87.7
1250	706	71.6
1500	897	50.3
1750	1058	32.4
2000	1111	26.6

Figure 2 below illustrates the plotted graph of position versus time from table 2, for the trajectory generated automatically by the controller.

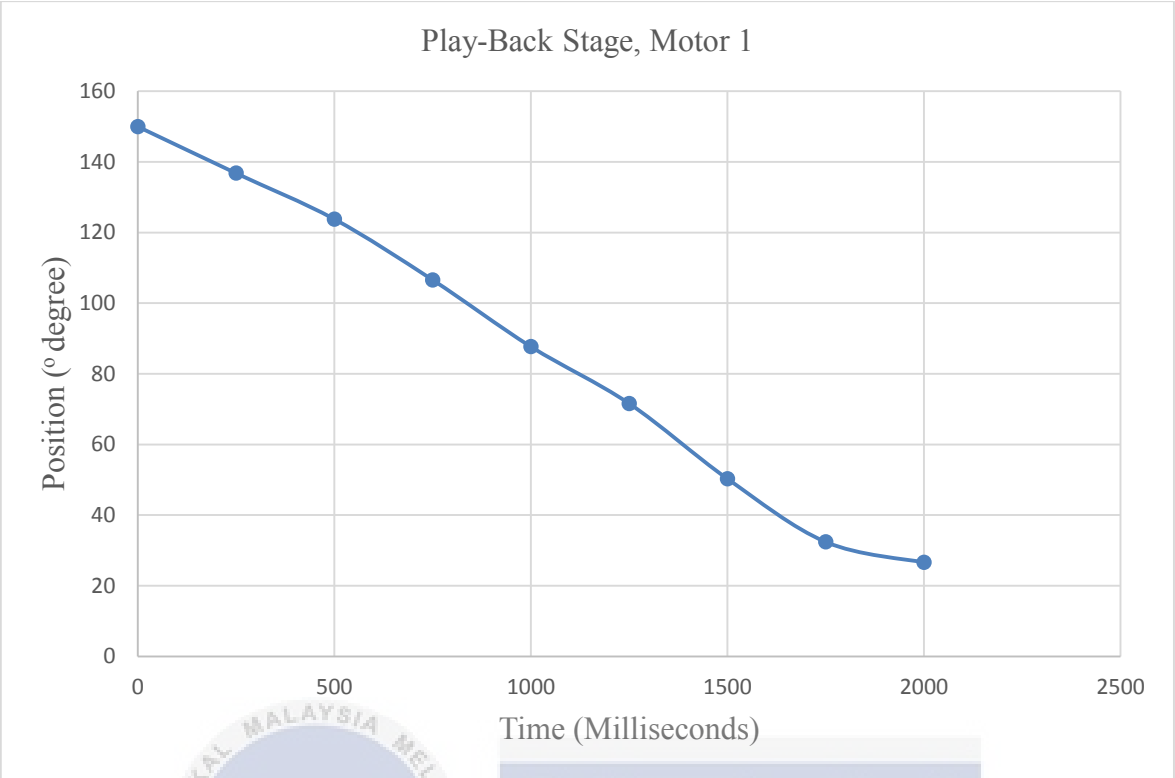

Figure 2: Trajectory Generated during Play-Back Stage, Motor one

Table 3: Motor two, Record Stage

Time (Milliseconds)	Motor 2	
	$Pulses_{rec}$	Q_{rec} (° degree)
0	0	0
450	2	0.2
900	26	2.9
1350	217	24.1
1800	400	44.4
2250	603	67
2700	751	83.4
3150	909	101
3600	1042	115.8
4050	1042	115.8

Figure 3 below illustrates the plotted graph of position versus time from table 3, for the trajectory generated manually by the operator.

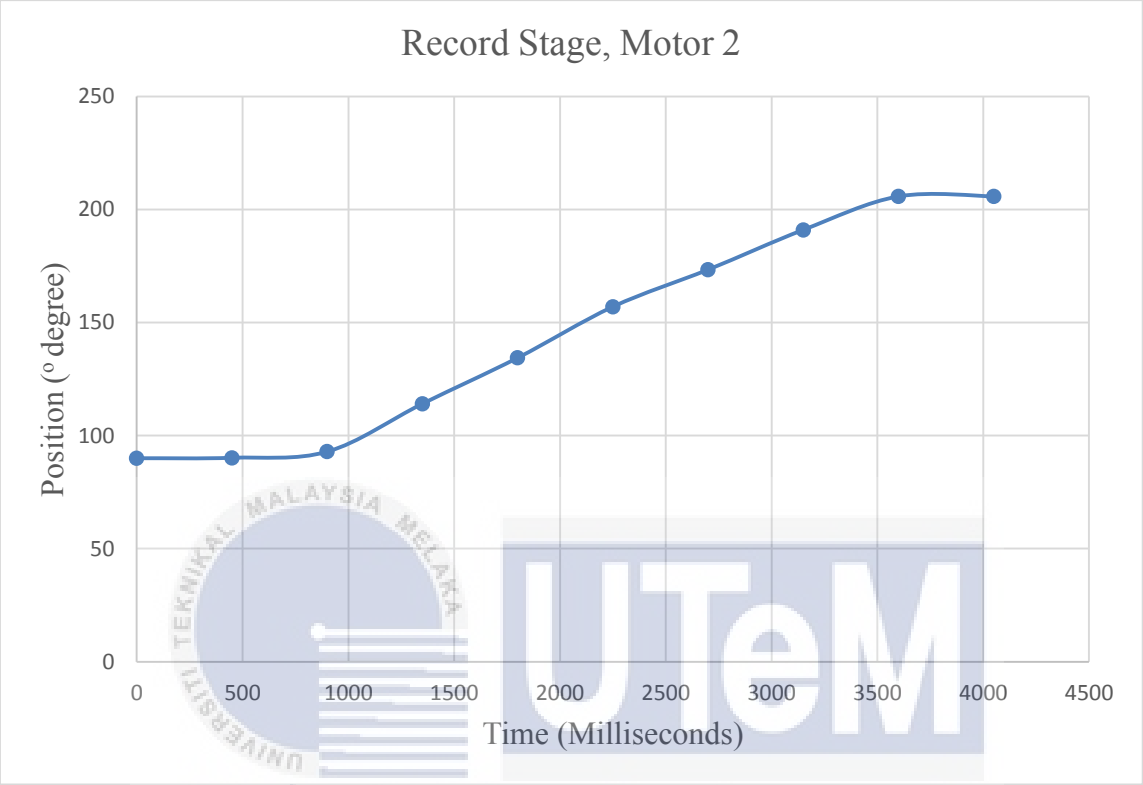

Figure 3: Trajectory Generated during Record Stage, Motor two

Table 4: Motor two, Play-Back Stage

Time (Milliseconds)	Motor 2	
	$Pulses_{played}$	Q_{Played} (Degree)
0	0	0
260	35	3.9
520	201	22.3
780	342	38
1040	520	57.8
1300	698	77.6
1560	842	93.6
1820	1030	114.4

2080	1039	115.4
2340	1058	117.6
2600	1058	117.6

Figure 4 below illustrates the plotted graph of position versus time from table 4, for the trajectory generated automatically by the controller.

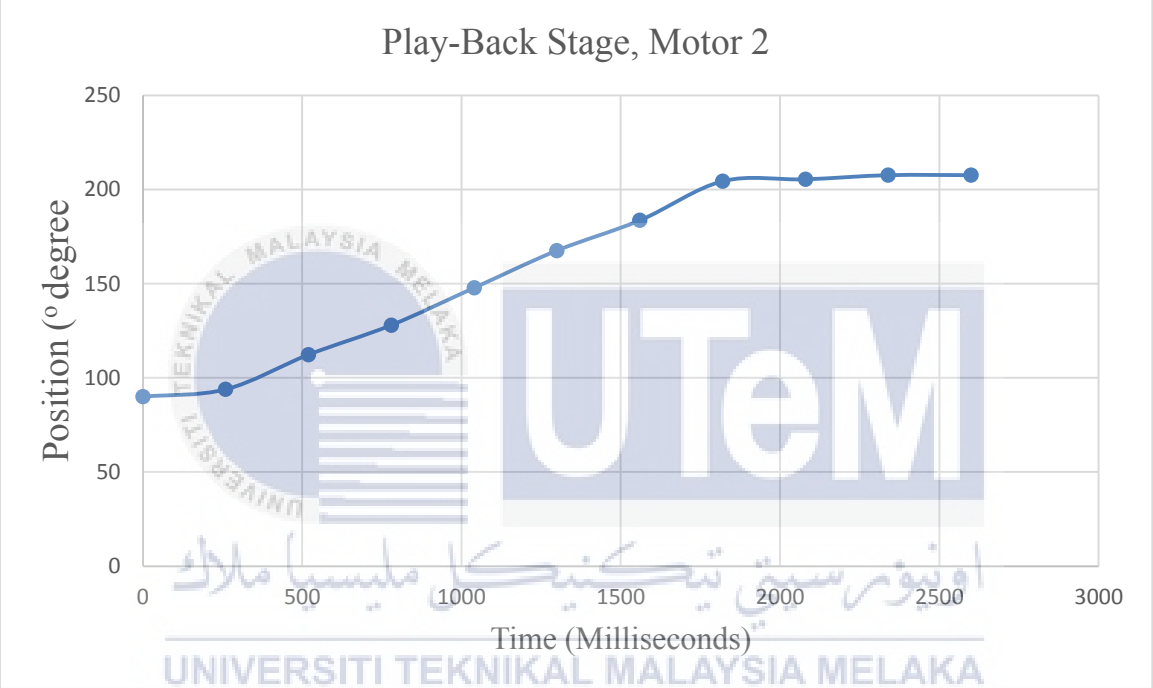

Figure 4: Trajectory Generated during Play-Back Stage, Motor two

Appendix B

```
#include <Time.h>

int encoderPinA1 = 12;

int encoderPinB1 = 11;

int encoderPinA2 = 10;

int encoderPinB2 = 13;

int recordpin = 8;

int playbackpin = 9;

int motoren1 = 5;

int motordir1 = 4;

int motoren2 = 6;

int motordir2 = 7;

int Home = 2;

int lims1 = 30;

int lims2 = 32;

int led1 = 31;

int led2 = 33;

int led3 = 35;

unsigned long time;
```

```
volatile int encoderPos1 = 0;

int lastReportedPos1 = 1;

int playbackpos1 = 0;

boolean A_set1 = false;

boolean B_set1 = false;


volatile int encoderPos2 = 0;

int lastReportedPos2 = 1;

int playbackpos2 = 0;

boolean A_set2 = false;

boolean B_set2 = false;


void setup() {

  pinMode(encoderPinA1, INPUT);

  pinMode(encoderPinB1, INPUT);

  pinMode(encoderPinA2, INPUT);

  pinMode(encoderPinB2, INPUT);

  pinMode(recordpin, INPUT);

  pinMode(playbackpin, INPUT);

  pinMode(Home, INPUT);

  pinMode(led1, OUTPUT);
```

```
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(motoren1, OUTPUT);
  pinMode(motordir1, OUTPUT);
  pinMode(motoren2, OUTPUT);
  pinMode(motordir2, OUTPUT);
  digitalWrite(encoderPinA1, HIGH);  // Internal pullup resistor
  digitalWrite(encoderPinB1, HIGH);  // Internal pullup resistor
  digitalWrite(encoderPinA2, HIGH);  // Internal pullup resistor
  digitalWrite(encoderPinB2, HIGH);  // Internal pullup resistor

  Serial.begin(9600);
}
```



```
void loop(){
  if (digitalRead(recordpin)==HIGH)
  {
    digitalWrite(led1, HIGH);
    attachInterrupt(11, doEncoderA1, CHANGE);
    attachInterrupt(12, doEncoderB1, CHANGE);
    attachInterrupt(13, doEncoderA2, CHANGE);
    attachInterrupt(10, doEncoderB2, CHANGE);
```

```
if (lastReportedPos1 != encoderPos1) {
  Serial.print("Motor1: ");
  Serial.print(encoderPos1);
  Serial.println();
  Serial.print("Time:");
  time = millis();
  Serial.println(time);
  lastReportedPos1 = encoderPos1;
  playbackpos1 = encoderPos1;

}
if (lastReportedPos2 != encoderPos2) {
  Serial.print("Motor2: ");
  Serial.print(encoderPos2);
  Serial.println();
  Serial.print("Time:");
  time = millis();
  Serial.println(time);
  lastReportedPos2 = encoderPos2;
  playbackpos2 = encoderPos2;
```

```
}
else
{
  digitalWrite(motoren1, LOW);
  digitalWrite(motordir1, LOW);
  digitalWrite(motoren2, LOW);
  digitalWrite(motordir2, LOW);
}
}
if (digitalRead(Home)==HIGH)
{
  digitalWrite(led2, HIGH);
  encoderPos1 = 0;
  encoderPos2 = 0;
  Serial.print("encoderPos1 is ");
  Serial.println(encoderPos1);
  Serial.print("playbackpos1 is ");
  Serial.println(playbackpos1);
  Serial.print("encoderPos2 is ");
  Serial.println(encoderPos2);
  Serial.print("playbackpos2 is ");
  Serial.println(playbackpos2);
  Serial.println("_ _ _ _ _ _");
```

```
if(digitalRead(lims2)==HIGH) {
  digitalWrite(motoren2, LOW);
  digitalWrite(motordir2, LOW);
}

else if (digitalRead(lims2)==LOW) {
  digitalWrite(motoren2, HIGH);
  digitalWrite(motordir2, LOW);
}

if(digitalRead(lims1)==HIGH) {
  digitalWrite(motoren1, LOW);
  digitalWrite(motordir1, LOW);
}

else if (digitalRead(lims1)==LOW) {
  digitalWrite(motoren1, HIGH);
  digitalWrite(motordir1, HIGH);
}
else {
  digitalWrite(motoren1, LOW);
  digitalWrite(motordir1, LOW);
```

```
    digitalWrite(motoren2, LOW);
    digitalWrite(motordir2, LOW);
  }

}

if (digitalRead(playbackpin)==HIGH)
{
  digitalWrite(led3, HIGH);
  attachInterrupt(11, doEncoderA1, CHANGE);
  attachInterrupt(12, doEncoderB1, CHANGE);
  attachInterrupt(13, doEncoderA2, CHANGE);
  attachInterrupt(10, doEncoderB2, CHANGE);
  Serial.print("encoderPos1 is ");
  Serial.println(encoderPos1);
  Serial.print("encoderPos2 is ");
  Serial.println(encoderPos2);
  Serial.print("Time:");
  time = millis();
  Serial.println(time);

  if (encoderPos1 < playbackpos1 && encoderPos2 < playbackpos2) {
    analogWrite(motoren1, 150);
```

```
  digitalWrite(motordir1, HIGH);
  analogWrite(motoren2, 150);
  digitalWrite(motordir2, HIGH);
}

else if (encoderPos1 < playbackpos1 && encoderPos2 >= playbackpos2) {
  analogWrite(motoren1, 150);
  digitalWrite(motordir1, HIGH);
  analogWrite(motoren2, 0);
  digitalWrite(motordir2, HIGH);
}

else if (encoderPos1 >= playbackpos1 && encoderPos2 < playbackpos2) {
  analogWrite(motoren1, 0);
  digitalWrite(motordir1, HIGH);
  analogWrite(motoren2, 150);
  digitalWrite(motordir2, HIGH);
}

else if (encoderPos1 >= playbackpos1 && encoderPos2 >= playbackpos2) {
  analogWrite(motoren1, 0);
  digitalWrite(motordir1, HIGH);
  analogWrite(motoren2, 0);
```

```
    digitalWrite(motordir2, HIGH);
  }


  else
  {
  analogWrite(motoren1, 0);
  analogWrite(motordir1, 0);
  analogWrite(motoren2, 0);
  analogWrite(motordir2, 0);
}


}

else
{
  analogWrite(motoren1, 0);
  analogWrite(motordir1, 0);
  analogWrite(motoren2, 0);
  analogWrite(motordir2, 0);
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
}
```

```
}



// Interrupt on A1 changing state

void doEncoderA1(){

  // Test transition

  A_set1 = digitalRead(encoderPinA1) == HIGH;

  // and adjust counter + if A leads B

  encoderPos1 += (A_set1 != B_set1) ? +1 : -1;

}


// Interrupt on B1 changing state

void doEncoderB1(){

  // Test transition

  B_set1 = digitalRead(encoderPinB1) == HIGH;

  // and adjust counter + if B follows A

  encoderPos1 += (A_set1 == B_set1) ? +1 : -1;

}


// Interrupt on A2 changing state

void doEncoderA2(){

  // Test transition

  A_set2 = digitalRead(encoderPinA2) == HIGH;
```

```
  // and adjust counter + if A leads B
  encoderPos2 += (A_set2 != B_set2) ? +1 : -1;
}

// Interrupt on B changing state
void doEncoderB2(){
  // Test transition
  B_set2 = digitalRead(encoderPinB2) == HIGH;
  // and adjust counter + if B follows A
  encoderPos2 += (A_set2 == B_set2) ? +1 : -1;

}
```