GESTURE CONTROLLED ROBOTIC ARM

CHEAH KAH HOU

A project report submitted in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering (Hons) Electronics Engineering

Faculty of Engineering and Green Technology
Universiti Tunku Abdul Rahman

January 2018

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : ____________________________

Name : ____________________________

ID No. : ____________________________

Date : ____________________________

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"GESTURE CONTROLLED ROBOTIC ARM"** was prepared by **CHEAH KAH HOU** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons) Electronics Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : ____________________________

Supervisor : Dr. Teh Peh Chiong

Date : ____________________________

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Teh Peh Chiong for his invaluable advice, guidance and his enormous patience throughout the development of the research.

GESTURE CONTROLLED ROBOTIC ARM

ABSTRACT

Robotic arm is widely used in various fields such as industry, medical and even military due to its abilities of high accuracy, efficiency and repeatability. Some special application in military such as bomb disposing by bomb disposal robot, human control is needed for the robotic arm on top of it. However, the bomb disposal robot typically controlled by special design joystick or gaming console such as Playstation controller, the control of those controller for the robotic arm is complicated and unintuitive. Thus, the goal of this project is to build a 6-axis robotic arm and implement gesture control into it. Leap motion sensor technology is part of the study since the sensor was used for capturing the gestures as the control of the robotic arm. The captured gesture data was then extracted and processed in the Processing software and sent to Linkit board by serial communication, the Linkit board was responsible to disassemble the data to drive the servo motors. The 6-axis robotic arm was built and gesture control with the Leap motion sensor was successfully implemented into it as the results.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

NUI	natural user interface
VR	virtual reality
USB	universal serial bus
SDK	software development kit
ROS	robot operating system
API	application programming interface
GPIO	general-purpose input/output
CAT	computer assisted translation
DOF	degree of freedom
GUI	graphic user interface
PWM	pulse width modulation

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

Robotic arm is a programmable manipulator, it comprises of linear and rotary joints to allow for controlled movements (Robots.com, 2017). It widely used in various field such as industry, medical and military for many years due to its high repeatability, accuracy and efficiency. The flexibility or dexterity of an articulated robotic arm is proportional to the number of axes of it. For the industrial articulated robotic arm, it ranging from different sizes depending on different application, for instance, the big heavy duty articulated arm perform automotive assembly while application such as electronics assembly is performed by smaller articulated arm.

In military, articulated robotic arm is used in bomb disposal robot. According to (Ray Allison, 2016), bomb disposal robot is used to disable explosive ordnance over years and saved countless lives. The robot is remotely controlled by bomb expert at a safe distance to examine explosive devices closely without putting themselves in danger. Thus the robot usually equipped with cameras as the "eyes" of the robot to provide the vision of the surrounding situation so that the operator can control the robotic arm through cameras to examine and dispose explosive devices. The robot also equipped with pairs of caterpillar tracks or wheels to allow it to traverse rough terrain such as climbing stairs and tools can be attached on the robotic arm such as wire cutter in order to bypass fences. The body itself also armed with explosive detectors and X-ray devices for the detection of explosives, not only bomb but unexploded munitions and landmines as well. Bomb disposal robot technically is not a "robot", more accurately as a drone, as human control still needed since bomb experts' experience and decision is crucial in explosive disposal operation.

Figure 1.1: Industrial articulated robotic arm

Figure 1.2: Bomb disposal robot (iRobot 510 PackBot)

Image taken from: Army Technology. (2014). *Detect and diffuse – The top 5 military robots for explosive ordnance disposal*. [online] Available at: http://www.army-technology.com/features/featuredetect-and-diffuse-the-top-5-military-robots-for-explosive-ordnance-disposal-4372678/ [Accessed 19 Aug. 2017].

1.2 Problem statement

The number of terrorist attack increasing recently, terrorist like ISIS they used to plant explosive devices at crowded areas such as shopping mall or roadside in order to murder. Remote controlled bomb disposal robot will do the job to disable the explosive devices as it lowers down the chances of injury if the bomb expert examines and disposes the bomb manually, but the problem is the robot is typically controlled by a specially designed joystick controller with buttons and knobs, some with computer based controller or even gaming console. The bomb expert need to be trained on how to use the controller since it is difficult to be used. The bomb disposal operation is slow since the control is not intuitive, a more intuitive way in controlling the robotic arm is needed.

1.3 Aims and objectives

The objectives of the project are shown as following:

- To build a 6-axis robotic arm.

- To implement gesture control on the robotic arm.

- To familiar with Leap motion sensor technology.

Gesture control method will be adopted in this project. With this gesture controlled robotic arm, the bomb disposal operation will be higher efficiency as the robot can be operate in faster and more intuitive way and no training is needed.

CHAPTER 2

LITERATURE REVIEW

2.1 Background of study

2.1.1 Gesture control

"Gesture control is the ability to recognize and interpret movements of the human body in order to interact with and control a computer system without direct physical contact." The interface of these system is known as natural user interface (NUI), generally lack of intermediate devices between the system and the user. (Gartner IT Glossary, 2017)

Gesture control is much more effortless and intuitive compared to pressing switches, tweaking knobs, manipulating mouse and touching screens. It especially contributes to ease the interaction between user and devices, replacing or reducing the need for keyboard, mouse or buttons. Human language will be more understandable by the computers and that will create a better user experience through gestures when face recognition and voice commands such advanced user interface technologies combined together. (dipert et al., 2013) Gesture control is being used in medical applications, alternative computer interfaces, entertainment applications and automation systems. In medical applications, life threatening conditions can be recognized with advanced gesture recognition robotic systems. For the entertainment applications, gesture control can provide more intuitive control environment that can immerse the player in the game like never before while in the automation systems in vehicles, homes and offices, gesture control can be incorporated to reduce the necessity of primary and secondary input systems such as buttons in the car entertainment systems as that will distract the drivers when the driver wants to control it while driving. (Gondane, 2017)

Gesture control can be separated into two categories by the processing methods:

Table 2.1: Comparison between offline and online gestures. Source from: (Sziládi, Ujbányi and Katona, 2016)

Offline gestures	Controlling methods such as menu or function activation, which the processing starts after the user's interaction. (not real time)
Online gestures	Kinect and Virtual Reality (VR) systems, the processing begins when the user starts an interaction, the controlling results is instant with minimal lateness. (real time)

2.2 Journals review

2.2.1 Review on Journal: *Gesture Control of Drone Using a Motion Controller* by Sarkar, a. et al.

This study presenting the implementation of motion sensor to control a drone by hand gestures. Leap motion sensor is used as the motion sensor or controller and the Parrot AR DRONE 2.0 as well in the implementation. This study shows a simple gesture controller can make the task of piloting much easier. LEAP motion sensor is used due to its wide range of infrared tracking, large field of view and high frame rate of tracking of gestures. The Parrot AR DRONE 2.0 is an off the shelf quadcopter with built in WIFI. The drone is connected to the ground station through the built in WIFI and

LEAP motion sensor also connected to the ground station via USB cable when operating. The gestures are captured by the LEAP motion sensor and then relayed to the ground station, thus the drone receives the signal from the ground station that captured by the LEAP and perform different movements such as yaw, pitch and roll. The method of retrieving of gesture raw data is using LEAP software development kit (SDK). Robot Operating System (ROS) is running in the ground station in Linux operating system, which is the platform for this entire system. Python programming language is used to interact between LEAP motion sensor and the AR DRONE, to interpret the hand gestures into the AR DRONE motions. With the aid of LEAP, the drone can perform various task or more since the "LEAP can be taught to recognize more hand gestures and movements by altering the python scripts and adding more functionality to it." In other words, by input certain gestures to the LEAP, SDK is then used to recognize the gestures which the gestures can be predefined or set by the user, some commands also need to be added into the python codes to interpret the new gestures which predefined in the SDK. Thus, the system can recognize those new gestures when it is made to the LEAP as this study added the "flip" gesture to carry out the flipping movement of the drone, it is an extension of the previous project.

In conclusion, this study provided an idea on how a gesture control system is formed. Which is shown in the Figure 2.1 below:

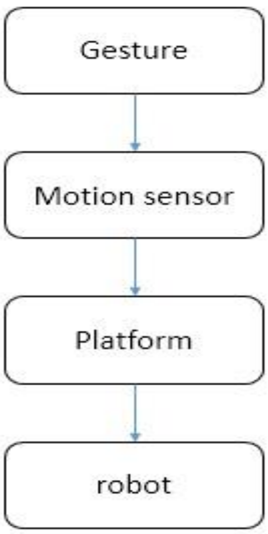

Figure 2.1: the block diagram of the system architecture.

The system basically constituted by gesture input, motion sensor, a platform and the robot. The motion sensor act as the controller as it replaces those typical joystick or console controller since it provided a more intuitive way of control. The platform is a "middle man", an interface between input and the output that interprets the gesture to the robot, usually is an operating system.

2.2.2 Review on Journal: ***Development of a Robotic Arm and implementation of a control strategy for gesture recognition through Leap Motion device*** by J.S., A. and J.L., M.

The objective of this study is to develop an articulated robotic arm prototype with implementing the gesture recognition control strategy through Leap motion sensor which focus on detection of hand gesture. This kind of control techniques in robotics introduce a complement in obligatory basic teachings in education which can provide a more entertaining and fun form, showing that there are always with additional motivation to the students when they develop and control robotic elements remotely or locally.

The robotic arm is made with obsolete materials such as some shafts and gears from the old printer, some are 3D printed out in order to stay the cost low. The relationship between the user and the robotic arm is enhanced by the incorporated gesture control through Leap motion sensor since it has a better control ability. This robotic arm prototype is capable to simulate the human forearm which can operate in 3D that follows the degree of freedom of human forearm. The reason to adopt gesture control is applications can be generated in a very intuitive way, this is the most effective gestures since the software adapts to human, not human adapts to the software. Human moves by instinct, without the need of thinking in most cases. Thus the best is computer understands human, that would make our life better.

The entire control system is integrated with several systems and new technology such as Leap motion device, implementation of hardware constituted by microcontroller and 3D printing technology. The control is performed by an Arduino controller that receives commands from the computer via serial communication to transmit the data to drives servo motors. Arduino is a platform that interface between the computer and the robotic arm, that translate the gestures into motion of the robotic arm.

In this study, several kinds of motion sensor such as Microsoft Kinect, Asus Xtion sensor, Wii remote, MYO and Leap motion are discussed and compared. These motion devices mostly used in video games to enrich the gaming experience. However, in this study shows that those devices can also be used in robotic development. Leap motion sensor is chosen due to its highest accuracy compared to other devices, which has the accuracy of 0.2mm and capturing frame can reach up to 200 frames per second, high portability with small size, easy to use and low cost. The SDK of the Leap transform the data captured into intuitive vectors such as position of fingers and hands and gestures like pinching and touching that facilitates the programmer works. There are application programming interfaces (APIs) inside the Leap, data of hands and fingers can be sent to the program that designed by programmer through the software. A few microcontroller boards are compared, the most important aspects to be considered is the connectivity, versatility and accessibility. For instance, the GPIO, the board has a large number output ports that will increase the accessibility of it since different types of peripherals can be accessed to. The Arduino One board is chosen because it met the necessary requirements and properties.

2.2.3 Review on Journal: ***Robot-arm Control System Using LEAP Motion Controller*** by Y., P. et al.

The objective of this study is to design and construct the control system for the robotic arm with Leap motion controller. The principle of Leap motion sensor and servo motors control adapted in this research. Leap motion sensor again is used in this research due to its specification is better than other motion device in this application. Basically the design of the control system is composed by three parts which is detection, data processing and signal control using microcontroller Arduino UNO R3 and robotic arm with a mini gripper. The system is tested and successfully function as it can control the robotic arm movements. The program is written in Javascript, leapjs and johnny-five package these two libraries need to be installed in computer in order to get raw data from Leap motion controller and send to microcontroller to execute.

The coordinates of the hand model inside Leap data have to map with the real coordinates of the robotic arm, the angle of the servo motors at each joint have to calculated with the captured X, Y and Z axes data. The X-axis is for the base rotation of the robotic arm, two fingers for the end-effector gripper, and the Y and Z axes are for the rest of the joint between base and gripper. Inverse kinematics equations are used in calculation for the servo motors angle between base and gripper. The robotic arm is 3D printed, the prototype is first designed by using CAT software. The results are satisfactory as the distance between hand and Leap motion versus angle of servo motor are similar for all the three axes, meaning that the robotic arm with this system that using inverse kinematics calculation can mimic the user's hand movement with high accuracy. The only limitation for this robotic arm is it cannot mimic the human hand movements completely as it only had 4 degree of freedoms (DOF), a higher DOF robotic arm have a better dexterity.

CHAPTER 3

METHODOLOGY

3.1 System working principle

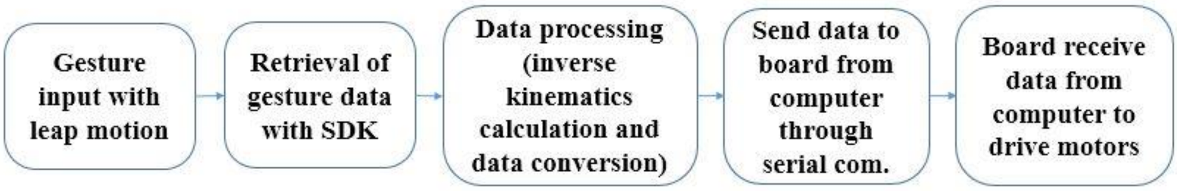

Figure 3.1.1: flow chart of the system.

The Figure 3.1 above showing how the system works. First, the gesture is captured by the Leap Motion sensor as the gesture input data, this gesture is then simulated by the Leap software development kit (SDK) in the form of 3D hand model which can be visualized by the Leap Motion Visualizer. Next, data processing such as inverse kinematics calculation and data conversion is done in the Processing software by coding in Java language so the angle or position of the servo motors are determined. After data processing is done, the processed data is then sent to the Linkit Smart 7688 duo board through serial communication to drive the servo motors so that the desired movement of the robotic arm can be rendered out.

3.2 Project management

The schedule of the project is shown in the Gantt chart below:

Table 3.2.1: Gantt chart for FYP 1

Activity/week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Title selection														
Idea discussion														
Literature review														
Methodology research														
Component selection and procurement														

Table 3.2.2: Gantt chart for FYP 2

Activity/week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Hardware design and build														
Gesture data extraction and implementation														
Program writing														
Testing and improvement														
Report writing														

3.3 Leap motion control panel

Device status can be shown in the troubleshooting tab in Leap motion control panel window such as service, device, calibration, tracking, bandwidth, lighting and smudge statuses to ensure whether the sensor in connected to the computer, whether the software development kit is tracking the hand and streaming to the visualizer and whether there is smudge on the sensor. Recalibration of the sensor also can be done in this window, it should be done if the sensor is out of its initial alignment, otherwise it will cause the tracking range become poorer, discontinues in the tracking data and jumpiness. There is an avoid poor performance option which tracking will pause when bad conditions on sensor is detected. This option was chosen since smooth gesture tracking data is needed, in order words, to avoid discontinuities occur in the control of the robotic arm. Sensor was connected to computer by USB port, before the sensor is connected, the device status is shown as in the Figure 3.3.1 below while Figure 3.3.2 is when the sensor is connected.

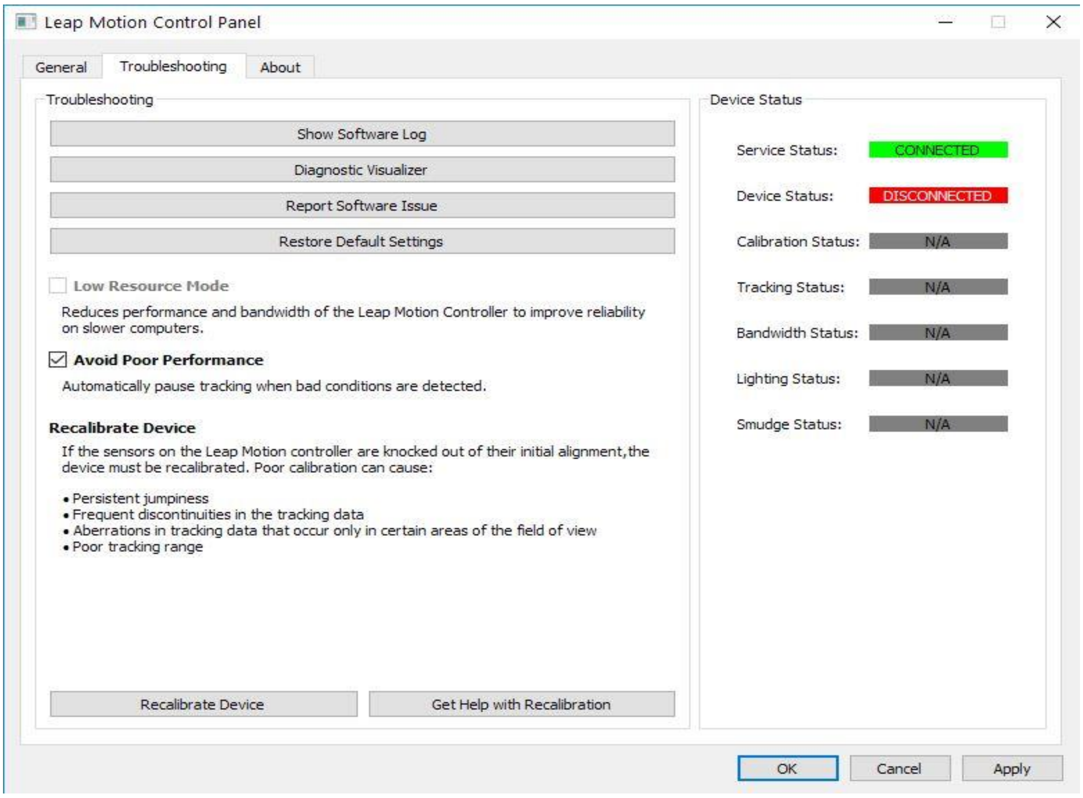

Figure 3.3.1: Leap motion control panel window with sensor disconnected.

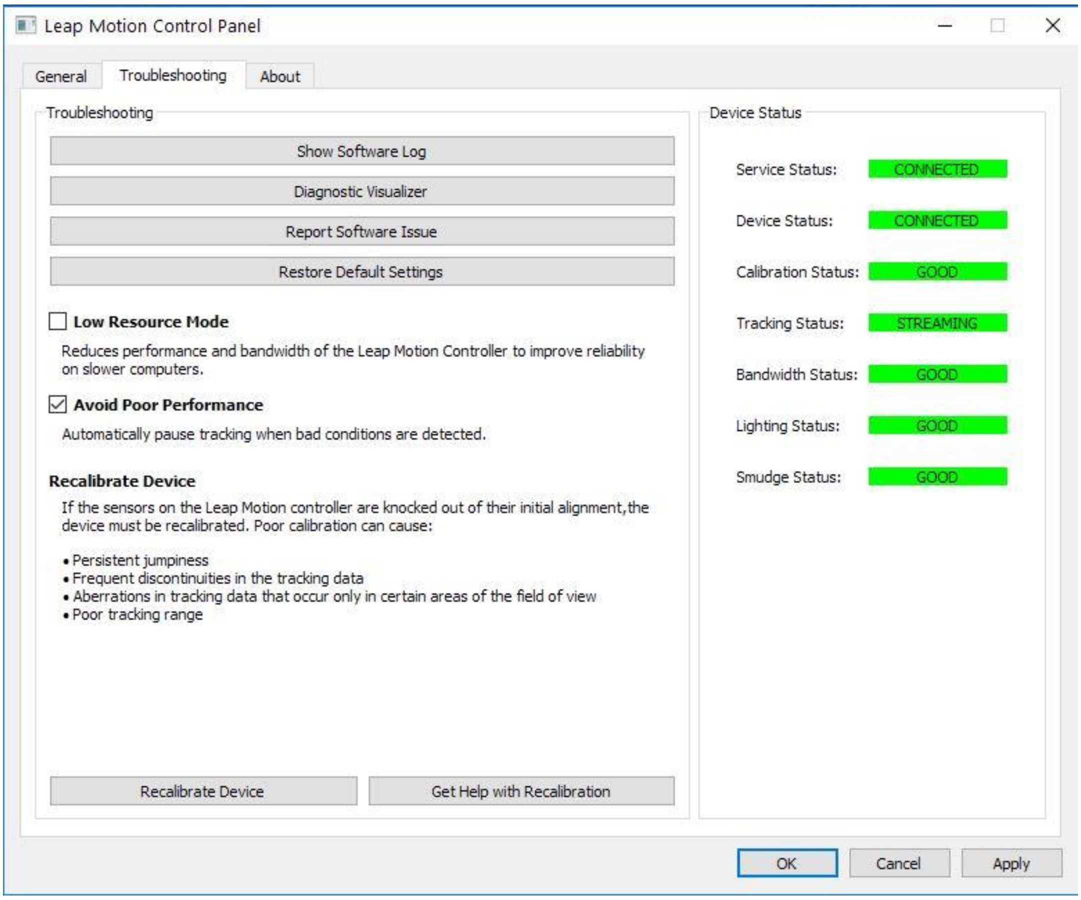

Figure 3.3.2: Leap motion control panel window with sensor connected.

3.4 Leap motion visualizer

Motion tracking data generated by the sensor can be displayed in the Leap motion visualizer in the form of modelled 3D hand as shown in the Figure 3.4.1 below. Various fps (frame per second) such as render fps, data fps and device fps were shown at the top right corner of the window. By referring to the specification of the Leap motion sensor, which claimed the tracking can have a fps of up to 120, which is very accurate. The basic three axes coordinate of the hand also been tracked and displayed here as well as the speed. Thus, the Leap motion visualizer was used to ensure the hand or gesture is being well tracked by the sensor and also visualization can be done to get a better understanding of how the gesture looks like.

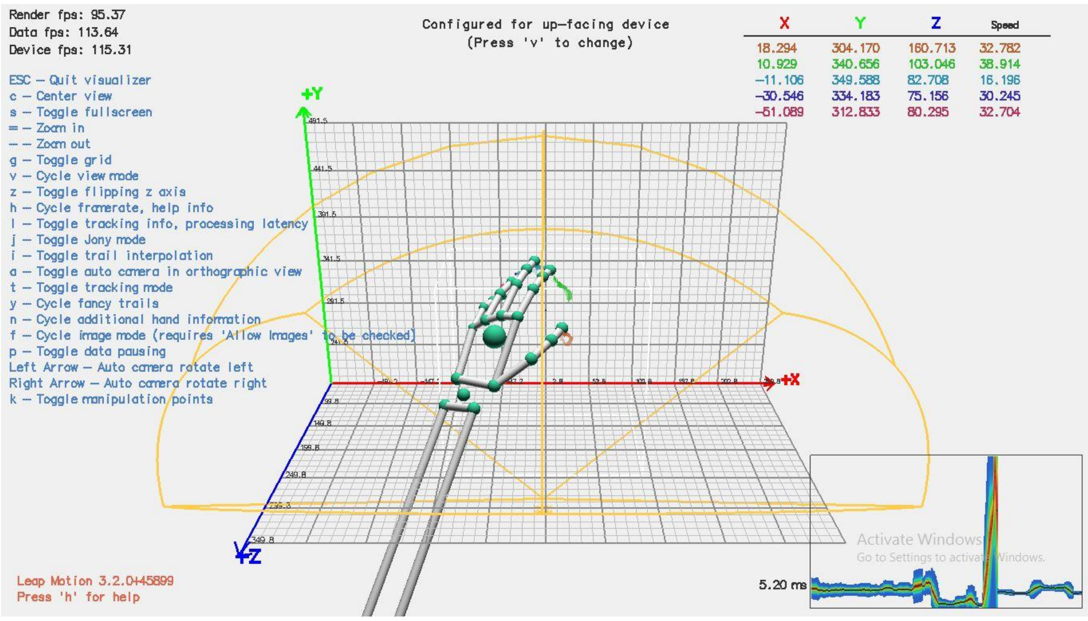

Figure 3.4.1: Leap motion visualizer window showing data generated from the sensor.

3.5 Coding in Processing software

3.5.1 Importing the libraries

Importing libraries is a very crucial step in code writing, since there are some important data such as configuration data, classes and parameter types that were predefined inside, those data need to be imported when the programmer needs it for program development. There are few libraries were imported in the code by writing "import" in front of the library name as shown in Figure 3.5.1.1 below, those libraries are: "*de.voidplus.leapmotion.**" for extracting the motion tracking data for the gesture captured by the Leap motion sensor, "*controlP5.**" and "*g4p_controls.**" for creating the GUI (graphic user interface) and the last one is "*processing.serial.**" to enable the serial communication function of Processing software in order to send data between Processing software and Linkit smart 7688 duo board.

```
import de.voidplus.leapmotion.*;   // lib : leap motion for processing
import controlP5.*;                 // lib for GUI
LeapMotion leap;                   // var. leap, LeapMotion type.
import g4p_controls.*;             // lib for GUI
Serial myPort;                     // declare the port as serial type
import processing.serial.*;        // serial lib
```

Figure 3.5.1.1: code of importing libraries.

3.5.2 Creation of knobs for indication

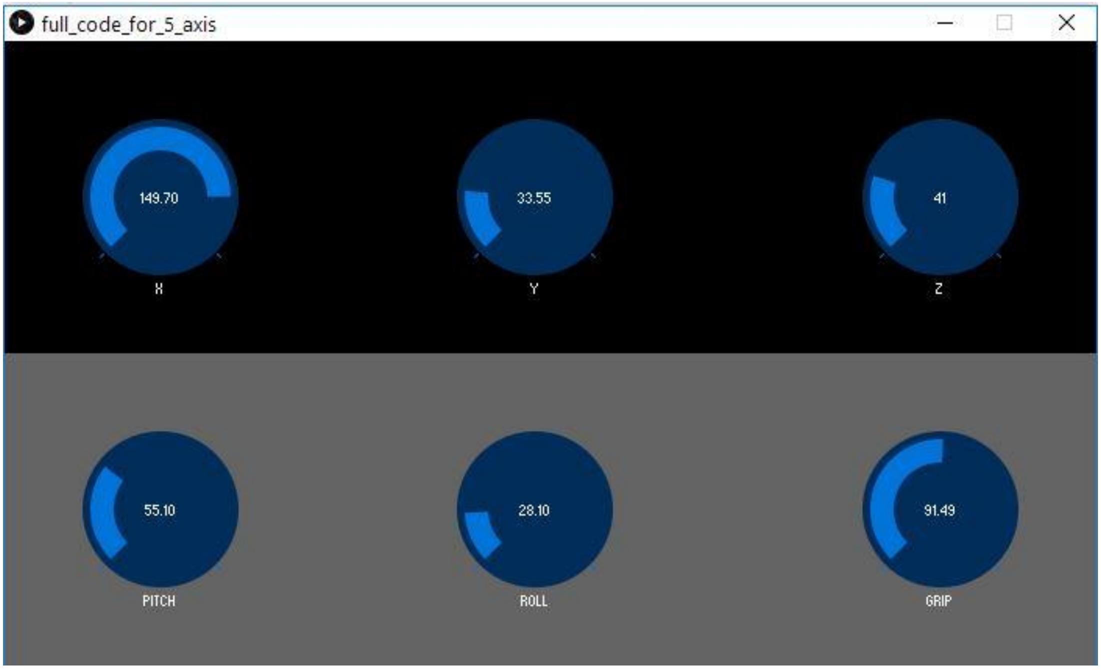

Figure 3.5.2.1: knobs created for indicating the gesture parameters.

Processing software is powerful in creation of graphic user interface, as shown in Figure 3.5.2.1, knobs were created for monitoring the gesture parameters extracted from sensor such as X, Y, and Z direction of hand, degree of pitch, degree of roll and the palm size for gripping during the control of the robotic arm. In figure 3.5.2.2 below, knobs were created by first declaring the name in "Knob" type, this "Knob" type can be created because the graphic user interface library of the Processing software was imported in the previous step. Next, the knob parameters were set as shown in Figure 3.5.2.3 below, by "*myKnobA = cp5.addknob("X")*" at the first line, which means the knob named "X" was created. Second line " *.setRange(0, 180)*" meaning that set the range of the knob set to from 0 to 180. Third line " *.setValue(50)*" is setting the initial value of the knob. Fourth line " *.setPosition(50, 50)*" is setting the position of the knob in the knob window which the first value is the horizontal coordinate and second value is the vertical coordinate. Fifth line " *.setRadius(50)*" was setting the radius of the knob and the last line " *.setDragDirection(Knob.VERTICAL)*" was allowed the knob to be scrolled in vertical direction for changing the value.

```
int myColorBackground = color(0, 0, 0);        // set the background color of the knobs window
int knobValue = 100;                           // the intensity of the background of the knobs window

Knob myKnobA;    // declaring knob names, knob class type, this class from GUI lib.    //for x
Knob myKnobB;    //for y
Knob myKnobC;    //for z
Knob myKnobD;    //for pitch
Knob myKnobE;    //for roll
Knob myKnobF;    //for grip
```

Figure 3.5.2.2: knobs declaring code.

```
myKnobA = cp5.addKnob("X")
  .setRange(0, 180)
    .setValue(50)
      .setPosition(50, 50)
        .setRadius(50)
          .setDragDirection(Knob.VERTICAL)
            ;
```

Figure 3.5.2.3: knobs creating code.

3.5.3 Data extraction and conversion

This section is the main part of the entire system that shows how the tracked gesture data being extracted and processed by the codes. The main 6 data of the hand to control the robotic arm were extracted by accessing the "*hand1*" class that created earlier. The statements were extracting the position of the hand, stabilized position of the hand which includes the x, y and z coordinates that need to be access later, roll of the palm in degree that used to control the wrist of the robotic arm, the time for the hand visible on the sensor that used to confirm whether the hand is appeared on the sensor, the pitch of the palm in degree that used to control the wrist of the robotic arm too and lastly, the sphere radius of the palm or palm size that used to control the gripper of the robotic arm.

```
PVector hand1_position    = hand1.getPosition();
PVector hand1_stabilized  = hand1.getStabilizedPosition();
float   hand1_roll        = hand1.getRoll();
float   hand1_time        = hand1.getTimeVisible();
float   hand1_pitch       = hand1.getPitch();
float   hand1_grip        = hand1.getSphereRadius();
```

Figure 3.5.3.1: Data extraction code.

For the data conversion as shown in Figure 3.5.3.2, before sending the data out to the board, some process or conversion must be done first. For using the data to drive the servo motor, those raw data must be mapped, the range of the extracted data must be converted to match the operating range of the servo motor which is 0 to 180 degree. The operating range of the servo motor not necessary to be fully utilized to render the movement of the hand. The "*map(hand1_stabilized.x, 570, 30, 40, 180)*" is the mapping statement, this mapped the hand stabilized position of x coordinate from the range of 30 – 570 to 40 – 180. After that, this mapped value was sent to the respective knob that created earlier to indicate this mapped data during the control of the robotic arm, in this case is the myKnobA for x coordinate. This value is then converted to integer type first from float type since the servo motor only accept integer value to drive. "*PosX(X)*" is the function header that used to send this processed data to the Linkit board through serial communication. The serial communication function will be explained later. The last statement "*delay(1)*" was added in the code to give some time to the Linkit board to respond for the signal that send out from the Processing software, the control of the robotic arm will be crashed without this delay function.

```
float transHand1PosX = map(hand1_stabilized.x, 570, 30, 40, 180);
myKnobA.setValue(transHand1PosX);
int X = int(transHand1PosX);
posX(X);
delay(1);
```

Figure 3.5.3.2: Data conversion code.

For the gripper, an additional step was done for better control and smoother by making the gripper to close only when the palm size bigger smaller than a certain value. Figure 3.5.3.3 shows that the gripper closes only when the palm size is bigger than 120, it is inverted because the configuration of the motor on the robotic arm structure was inverted. At the beginning, the gripper opens when the palm closes, with it invert mapping, now the gripper closes when the palm closes. Why the value was set to 120 because it is the most suitable value that match various palm size when different users take control.

```
float transHand1Grip = map(hand1_grip, 150, 10, 10, 150);
myKnobF.setValue(transHand1Grip);
int grip = int(transHand1Grip);
if (grip>120){
posgrip(140);
delay(1);
}
else {
posgrip(90);
delay(1);
}
```

Figure 3.5.3.3: setting the range for the various palm size.

3.5.4 Inverse Kinematics calculation

Inverse Kinematics is the process realizing the desired end effector pose by finding the value of the joint variables. (De Luca, 2018) The angle of each joint of the robotic arm were found by using the Inverse kinematics calculation, the servo motor can be driven by the value that found in the calculation to realize the desired motion. This calculation only applied to two servo motors at middle, which is "shoulder" and "elbow". Remaining servo motors were controlled separately and individually by respective data. 2D inverse kinematics were used since the desired movement only realized in x and y axes. This 2D inverse kinematics calculation can be done by using trigonometry equations as shown below.

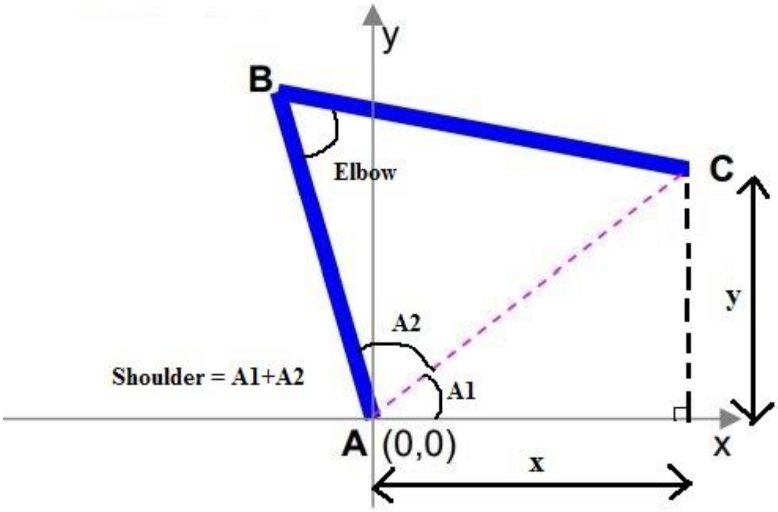

Figure 3.5.4.1: 2D inverse kinematics derivation.

$$c^2 = a^2 + b^2 - 2ab\ cos\ \Upsilon \tag{3.1}$$

$$\Upsilon = \cos^{-1}\left[\frac{a^2+b^2-c^2}{2ab}\right] \tag{3.2}$$

$$Elbow = \cos^{-1}\left[\frac{AB^2+BC^2-AC^2}{2\times AB\times BC}\right] \tag{3.3}$$

$$A2 = \cos^{-1}\left[\frac{AB^2-BC^2+AC^2}{2\times AB\times BC}\right] \tag{3.4}$$

$$AC = \sqrt{y^2 + x^2} \tag{3.5}$$

$$A1 = \tan^{-1}(\frac{y}{x}) \tag{3.6}$$

$$Shoulder = A1 + A2 \tag{3.7}$$

The angle of Elbow and Shoulder have to be found for realizing the robotic arm desired pose, which is the "Elbow" and "A1" + "A2" for shoulder. Equation 3.6 was used to calculate the angle of A1 while A2 was found by using the equation 3.4 which derived from the law of cosine as shown in equation 3.1 and equation 3.2. The "AC" from equation 3.3 was found by the equation 3.5 which is the Pythagoras theorem. The "x" and the "y" is the x coordinate and y coordinate of the hand respectively. All the equation was converted to code in Processing software as shown in Figure 3.5.4.2 below.

```
float M = sqrt((y*y)+(x*x));
if (M <= 0)
  return 1;
float A1 = atan(y/x);
float A2 = acos((A*A-B*B+M*M)/((A*2)*M));
float Elbow = acos((A*A+B*B-M*M)/((A*2)*B));
float Shoulder = A1 + A2;
Elbow = Elbow * rtod;
Shoulder = Shoulder * rtod;


while ( (int)Elbow <= 0 || (int)Shoulder <= 0)
  return 1;
float Elbowmapped = map(Elbow, 10, 200, 30, 100);
float Shouldermapped = map(Shoulder, 80, 30, 10, 70);
posY(int(Shouldermapped));
delay(1);
posZ(int(Elbowmapped));
delay(1);
println(" IE, Elbow: "+Elbow+"  Shoulder: "+Shoulder);
```

Figure 3.5.4.2: 2D inverse kinematics equations in Java code.

3.5.5 Serial communication to Linkit board

For sending data between Processing software and Linkit board, serial communication was established by first declaring serial port named "*myPort*" as shown in Figure 3.5.5.1 below. The data was sent with the serial function as shown in Figure 3.5.5.2.

X-axis coordinate sending function was the example in this explanation. The serial port must be cleared before sending data through, writing data after clearing that ensures the data send out was not mixed with other data. The first number in the bracket is the servo motor number that assigned in the programme in the Linkit board. The value is the servo motor angle that processed out from the data conversion. The "*\n*" is the "next line" command which used to indicate the serial data was ended here. These three things were concatenated together for faster serial transfer.

```
myPort = new Serial(this, Serial.list()[0], 9600);
myPort.bufferUntil('\n');
```

Figure 3.5.5.1: Enabling serial communication in Processing.

```
public void posX(int theValue) {
  myPort.clear();
  myPort.write("0,"+theValue+"\n");
 }
```

Figure 3.5.5.2: Send processed data through serial communication.

3.6 Arduino IDE (Integrated Development Environment)

3.6.1 Setting up servo motors

The task of the Linkit board is interfacing servo motors and gesture data from Processing software. Certain pins must be reserved for the servo motors in order for sending PWM (pulse width modulation) signal from the Linkit board to servo motors. Figure 3.6.1.1 shows that how the pins were assigned for the servo motors. For instance, the pin 3 was assigned to servo 0 which is for the x coordinate, meaning that this servo 0 is responsible to turn left and right for the robotic arm. For the Figure 3.6.1.2, which is the pre-setting the angle for the servo motors, this pre-setting is for the robotic arm starting pose.

```
servo[0].attach(3);   //x
//delay(100);
servo[1].attach(4);   //for z
//delay(100);
servo[2].attach(5);   //for y
//delay(100);
servo[3].attach(6);   //for roll
//delay(100);
servo[4].attach(7);   //for pitch
//delay(100);
servo[5].attach(8);   //for gripper
//delay(100);
```

Figure 3.6.1.1: setting up pins for servo motors.

```
servo[0].write(110);  //for x
//delay(100);
servo[1].write(50);    //for z
//delay(100);
servo[2].write(45);   //for y
//delay(100);
servo[3].write(90);   //for roll
//delay(100);
servo[4].write(0);   //for pitch
//delay(100);
servo[5].write(90);   //for gripper
//delay(100);
```

Figure 3.6.1.2: pre-setting the angle for the servo motors.

3.6.2 Receive data from Processing software

When the data sent out from the Processing software through serial communication, it has to be read and decoded in the Linkit board. In Figure 3.6.2.1 and Figure 3.6.2.2 showing the code that how to read the string from serial and how the string data parsing down respectively. The "serialEvent()" containing a while loop that keep checking the serial is it available, meaning that as long as the signal sending in from the serial, the loop will keep looping for reading data. Since the serial is transferring the signal byte by byte, so the new byte must keep concatenating with the previous byte to form the complete signal. How to know whether the signal is completed was by checking whether the "*\n*" or "next line" is read from the serial. When this "next line" is read, the flag "*stringComplete*" will be raised, become true and this flag signal will be sent to the next function "*parseCommand()*" to do the parsing task as shown in Figure 3.6.2.2.

```
void serialEvent() {
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    inputString += inChar;
    Serial.println(inputString);


    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

Figure 3.6.2.1: reading string from serial.

The "*parseCommand()*" function will first checking whether the "*stringComplete*" flag is true, if yes, this function only starts to do the task. It will start to trim the leading and tailing whitespaces in the string to remove unnecessary signal. Next, the servo number was extracted from the string by taking the value before the comma (,). With using the command of "*indexOf(',')*", the comma (,) position was found, this position was used as the separator index to separate the servo number and the servo angle. Thus the servo angle was retrieved by looking at the value after the separator index. Recap that the signal was sent out by this format from the Processing software as shown in Figure 3.5.5.2, so the signal has to be parsed in this way to retrieve back the actual signal. Lastly, all the strings buffer and the flag must be cleared for the next parsing.

```
void parseCommand(){

  if (stringComplete) {
    inputString.trim();
    Serial.println(inputString);
    int separatorIndex = inputString.indexOf(',');
    servoString=inputString.substring(0,1);
    commandString=inputString.substring(separatorIndex+1);
    int servoNum = servoString.toInt();
    int servoPos = commandString.toInt();

    servo[servoNum].write(servoPos);

    // clear the string:
    inputString = "";
    servoString = "";
    commandString = "";
    stringComplete = false;
  }
}
```

Figure 3.6.2.2: Parsing function.

3.7 Equipment and Cost analysis

Table 3.7.1: Equipment and component list with price.

No.	Component	Quantity	Price (RM)	Remarks:
1.	Leap motion sensor	1	418	From Taobao
2.	Linkit smart 7688 duo	1	-	Acquired from lab, with cable
3.	Servo motor	6	113.4	RM 18.9 each
4.	Gripper	1	36.9	From lelong
5.	Acrylic board	1	-	Acquired from senior
6.	PCB stand	4	-	Acquired from senior
7.	U shape servo motor bracket holder	3	20.7	From lelong, RM 6.9 each
8.	Multipurpose servo motor bracket holder	5	54.5	From lelong, RM 10.9 each
9.	Screw	54	10.8	From lelong, RM 0.2 each
10.	Nut	54	10.8	From lelong, RM 0.2 each
11.	Computer	1	-	Own laptop with Processing software and Arduino IDE installed
12.	Power bank	1	60	With cable
13.	Breadboard	1	-	Acquired from lab
14.	Jumper wire		-	Acquired from lab

The total cost is RM 725.1

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Robotic arm outlook

The robotic arm was built with six servo motors, servo motor bracket holders and a gripper. It powered up by power bank with 5 Volts and 2.1 Ampere output. There are 3 pins in the cable of servo motor which positive, negative and signal pin. The positive and negative pin of the servo motor connected to the positive and negative wire that separated out from the USB to micro USB cable of power bank. The servo motor signal pin connected to the assigned pin on the Linkit board for receiving signal.

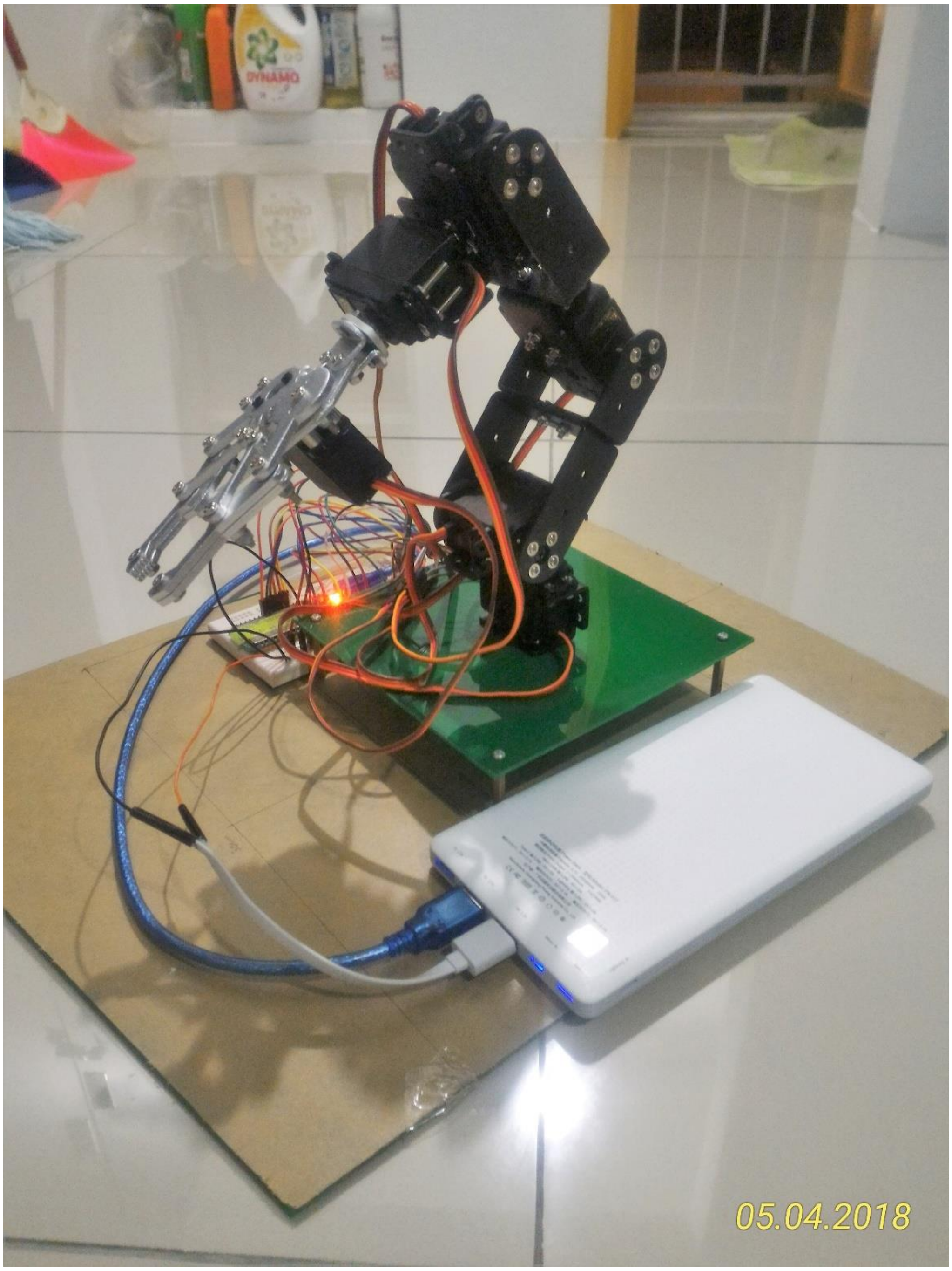

Figure 4.1.1: The 6-axis robotic arm.

4.2 Testing

The robotic arm was able to follow the hand gesture such as grip, left, right, roll in counter-clockwise, roll in clockwise, pitch-up, pitch-down, downward, upward, forward and backward as shown in the figures below. All these movements can be done simultaneously.

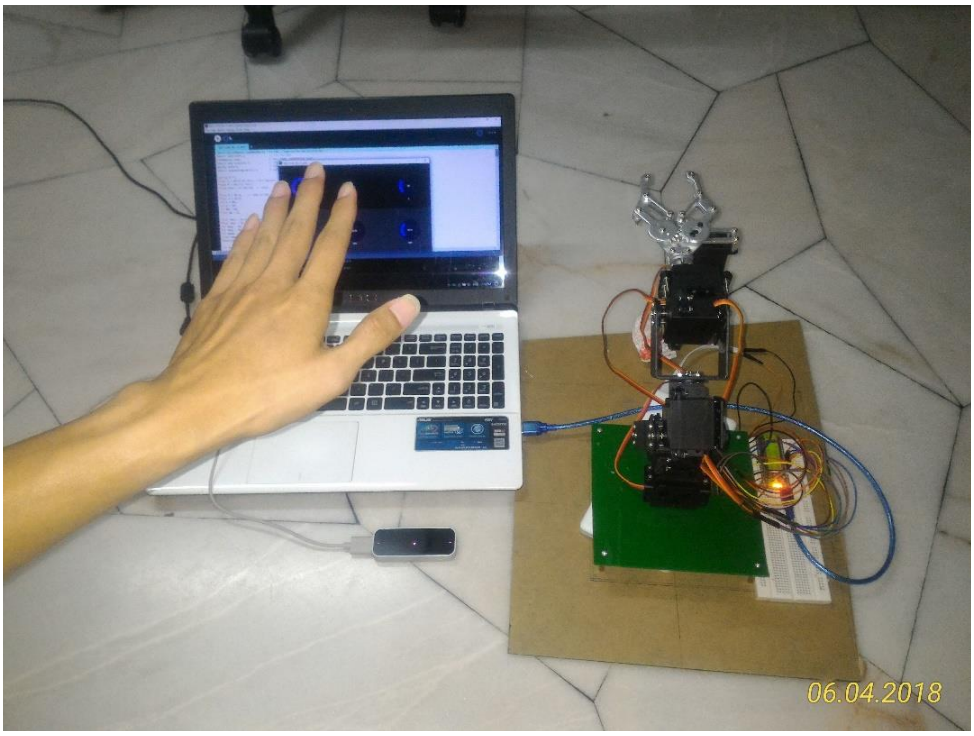

Figure 4.2.1: starting position at centre point.

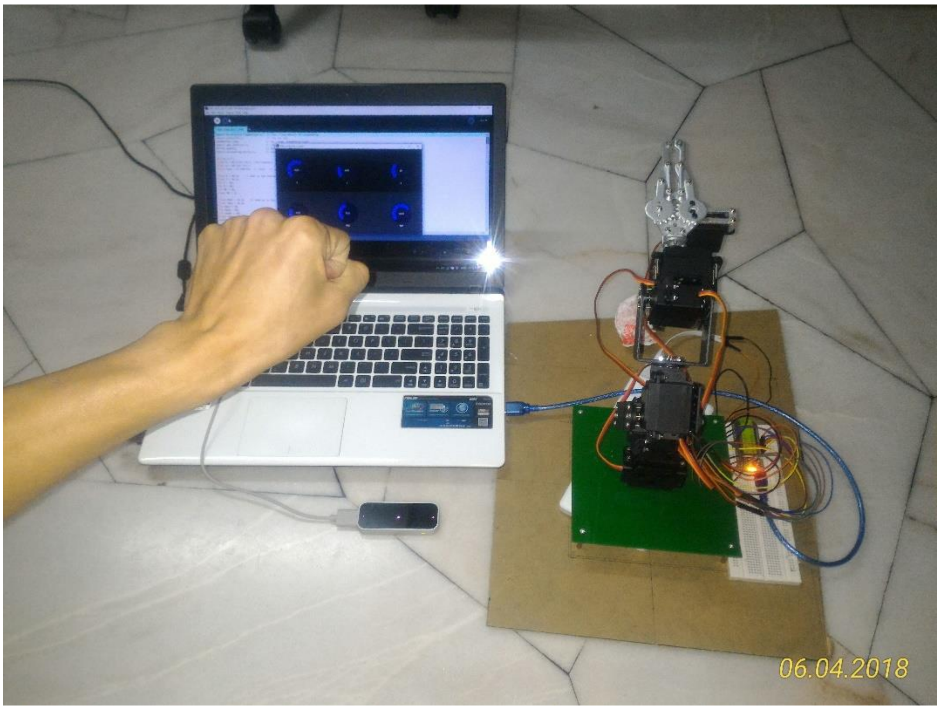

Figure 4.2.2: gripping

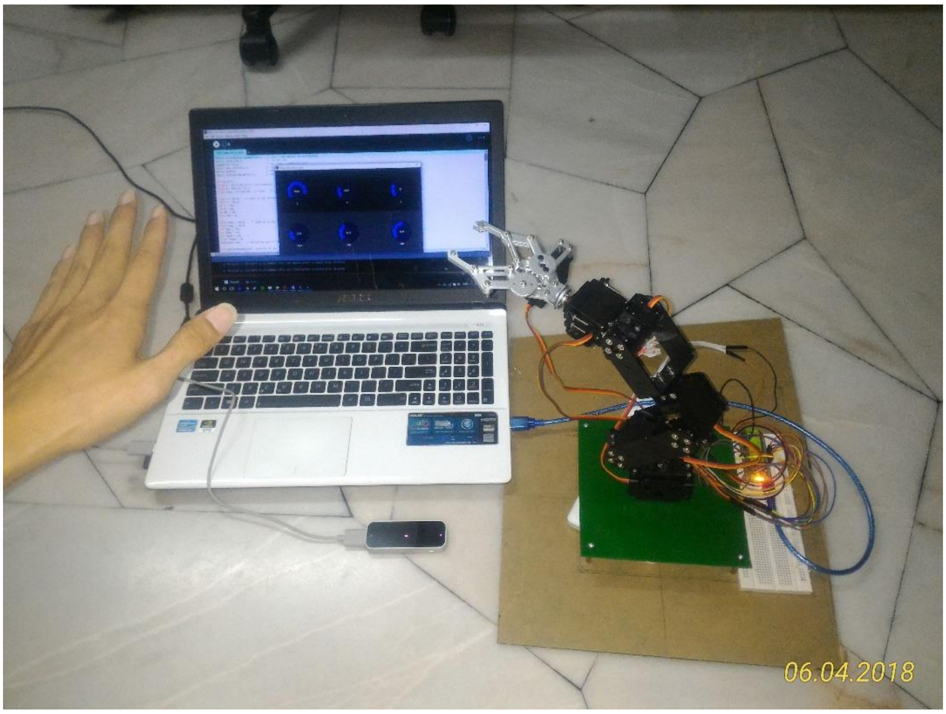

Figure 4.2.3: move to left.

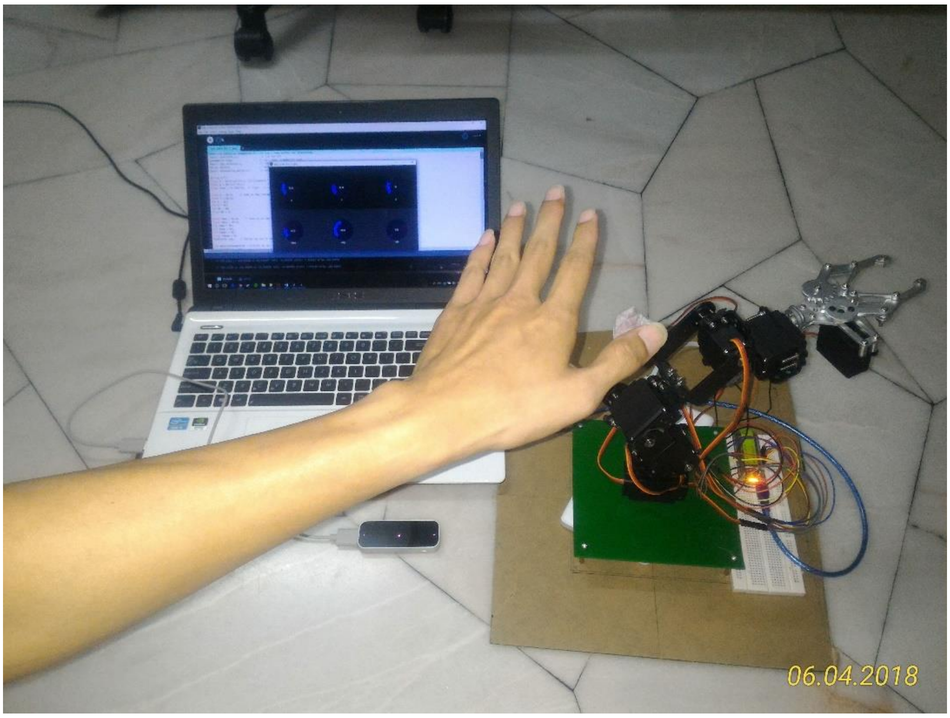

Figure 4.2.4: move to right.

Figure 4.2.5: no roll.

Figure 4.2.6: roll in counter-clockwise.

Figure 4.2.7: roll in clockwise.

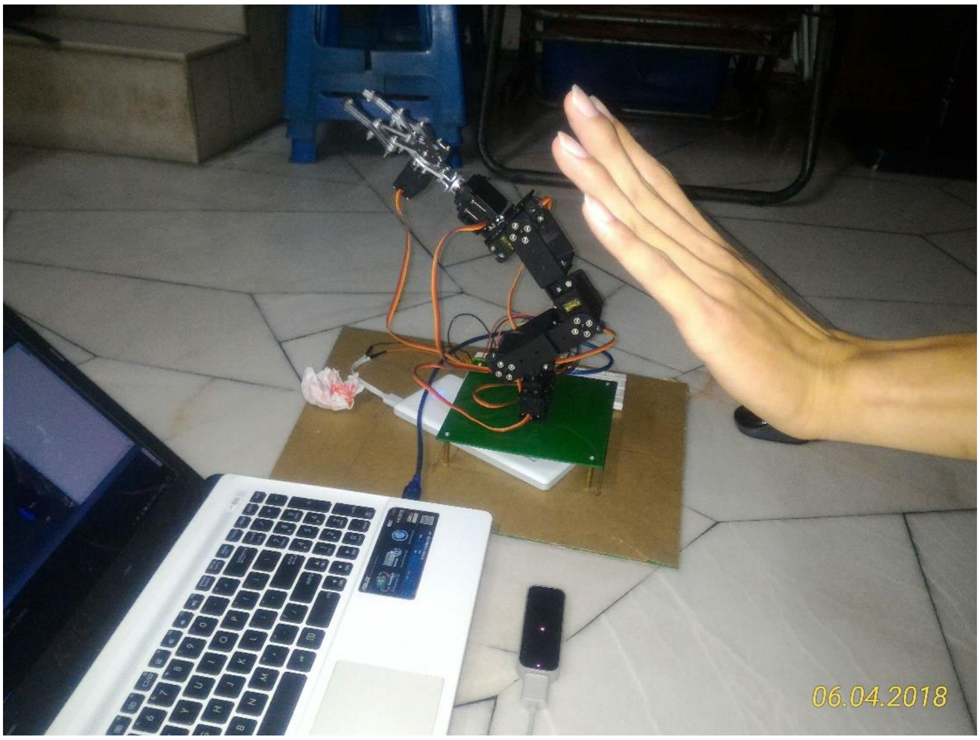

Figure 4.2.8: pitch-up

Figure 4.2.9: pitch-down

Figure 4.2.10: back to centre point.

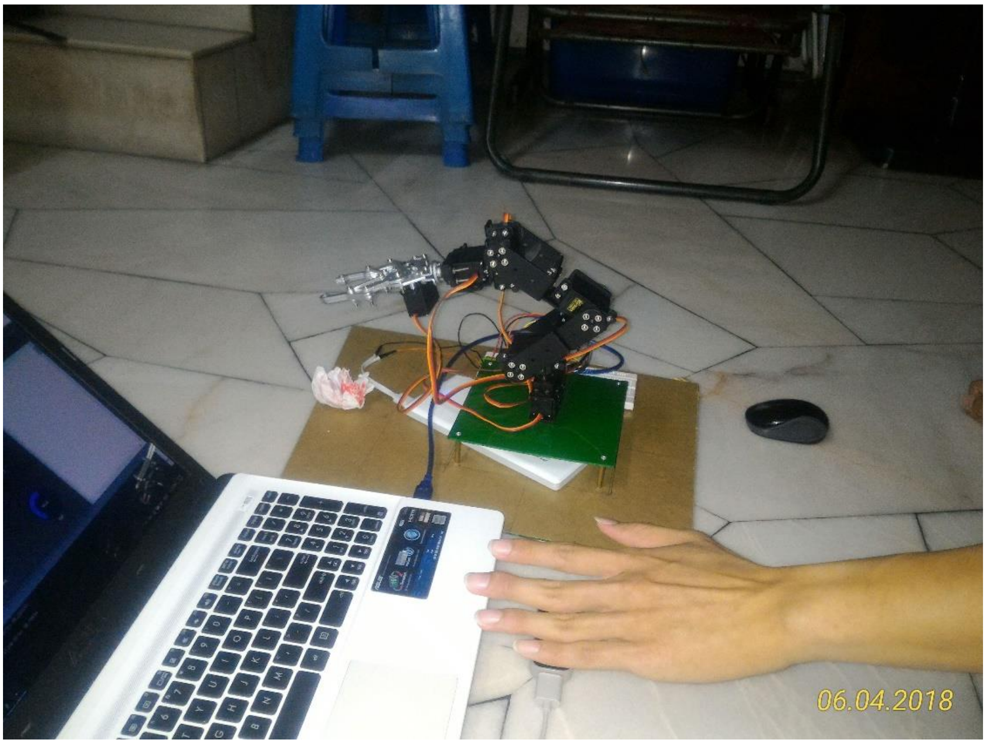

Figure 4.2.11: move downward

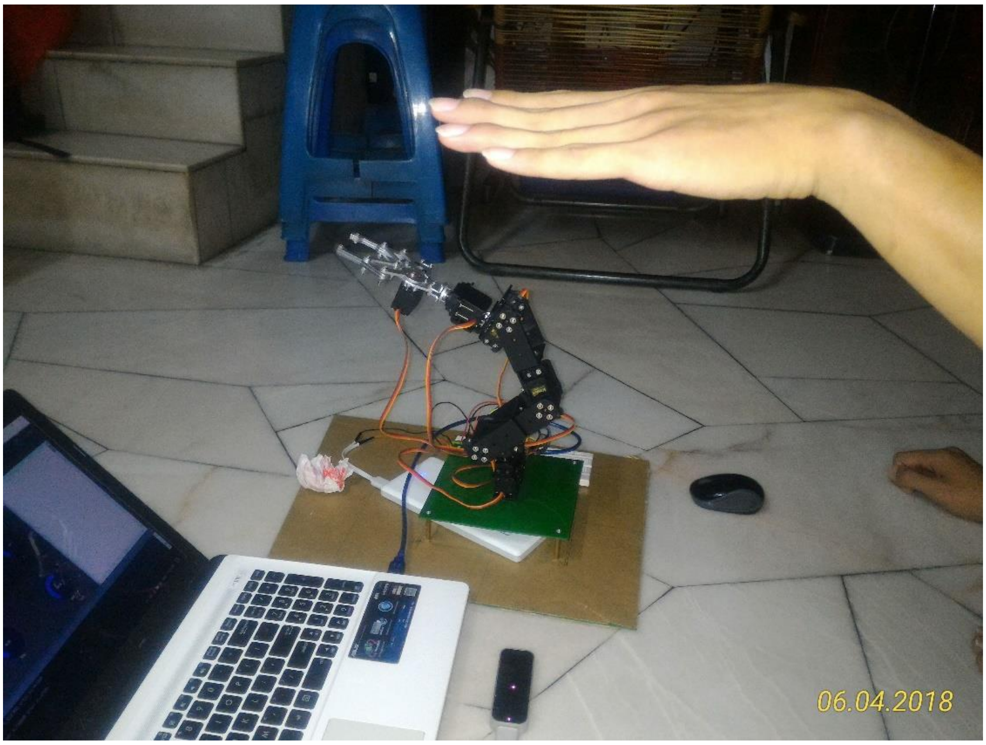

Figure 4.2.12: move upward

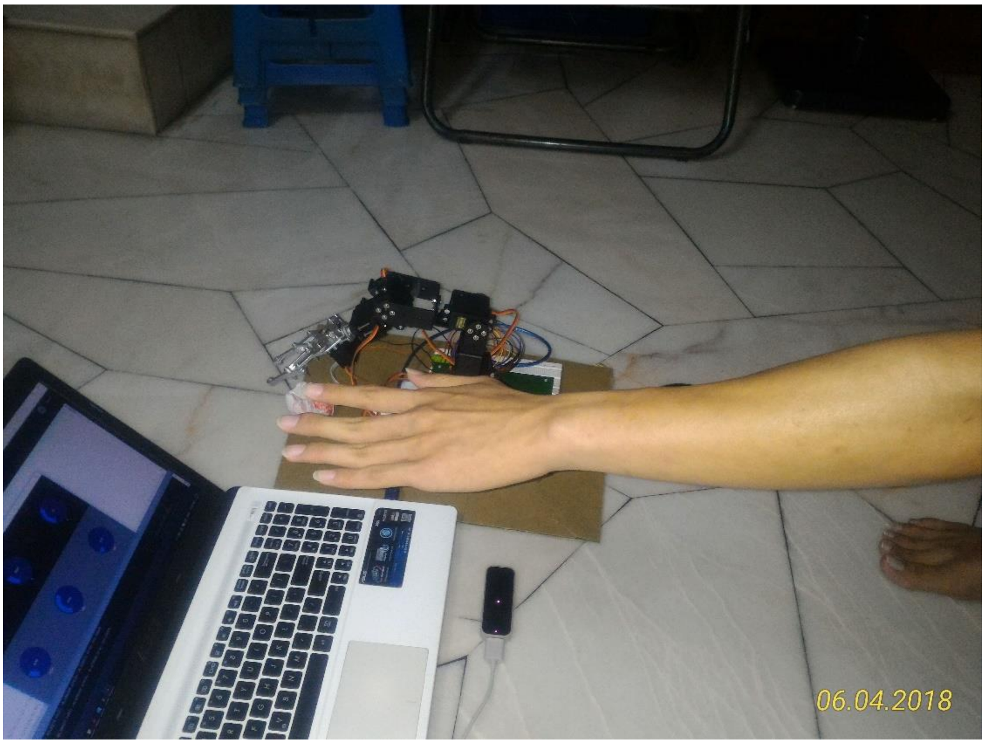

Figure 4.2.13: move forward

Figure 4.2.14: move backward

Table 4.1: Voltage and current comparison between power supply and power bank.

	Power supply	Power bank
Stationary voltage	6.592 V	4.92 V
Stationary current	300 mA	330 mA
Moving voltage (min.)	5.812 V	4.333 V
Moving current (max.)	1471 mA	1507 mA

The optimum voltage for the servo motor is 6.6 V, however the power bank only can output 5 V. For the portability, power bank was used to power up the robotic arm. Overall performance of the robotic arm powered up by power bank was satisfied. If power supply were used to power up, the robotic arm can produce more energetic, smoother, more stable and responsive movement since with the optimum power source. Use Linkit as the power source was avoided, it can result in burn up the Linkit board since six servo motors will draw too much current from the board.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

In conclusion, the 6-axis robotic arm was built and it is able follow the gesture of the user. Six (6) for the axis number of a robotic arm is consider high, 6-axis was chosen to build is because when the number of axis higher meaning that higher the dexterity of the robotic arm is. Next, with the technology of Leap motion, gesture control is successfully implemented on the robotic arm, meaning that the robotic arm is now able to follow the user hand gesture movement, controlling the robotic arm intuitively. This project showed how convenient is the robotic arm controlled by this pendrive-size Leap motion sensor with a computer. This addressed the problem of using typical controller of the bomb disposal robot.

Leap motion sensor can capture the gesture with high accuracy, according to the its manufacturer, the Leap motion sensor can capture the gesture up to 200 frame per second by two infrared cameras with accuracy up to 0.2 mm. (J.S., A, 2016) This technology can be applied to many applications especially the one that required human control such as medical surgery, controlling the robotic arm remotely in operation with high accuracy to avoid infection.

One of the limitation of this project is the robotic arm is bonded to the computer for serial communication. Wi-Fi technology can be implemented into this project in future improvement work for better portability of the robotic arm and user can remotely control the robotic arm anywhere for better convenience.

REFERENCES

Army Technology. (2014). *Detect and diffuse – The top 5 military robots for explosive ordnance disposal*. [online] Available at: http://www.army-technology.com/features/featuredetect-and-diffuse-the-top-5-military-robots-for-explosive-ordnance-disposal-4372678/ [Accessed 19 Aug. 2017].

De Luca, A. (2018). *Inverse kinematics*. [online] Diag.uniroma1.it. Available at: http://www.diag.uniroma1.it/~deluca/rob1_en/10_InverseKinematics.pdf [Accessed 9 Apr. 2018].

dipert, b., siegel, y., morris, s., rostock, l. and kutliroff, g. (2013). *The gesture interface: A compelling competitive advantage In the technology race | EE Times*. [online] EETimes. Available at: http://www.eetimes.com/document.asp?doc_id=1280756 [Accessed 22 Aug. 2017].

Gartner IT Glossary. (2017). *Gesture Control - Gartner IT Glossary*. [online] Available at: http://www.gartner.com/it-glossary/gesture-control/ [Accessed 22 Aug. 2017].

Gondane, R. (2017). *Future with Gesture Recognition Technology Control and Applications*. [online] Edgefx Kits Official Blog. Available at: http://www.edgefxkits.com/blog/what-is-gesture-recognition-technology-glimpse-of-the-future/ [Accessed 22 Aug. 2017].

J.S., A. and J.L., M. (2016). *Development of a Robotic Arm and implementation of a control strategy for gesture recognition through Leap Motion device*. [online] Available at: http://ieeexplore.ieee.org.libezp.utar.edu.my/document/7528373/ [Accessed 13 Aug. 2017].

Ray Allison, P. (2016). *What does a bomb disposal robot actually do?*. [online] Bbc.com. Available at: http://www.bbc.com/future/story/20160714-what-does-a-bomb-disposal-robot-actually-do [Accessed 19 Aug. 2017].

Robots.com. (2017). *What is a robot arm?*. [online] Available at: https://www.robots.com/faq/show/what-is-a-robot-arm [Accessed 19 Aug. 2017].

sarkar, a., Arvindbhai Patel, k., Ram R.K., G. and Krishna Capoor, G. (2016). *Gesture Control of Drone Using a Motion Controller*. [online] IEEE. Available at: http://ieeexplore.ieee.org.libezp.utar.edu.my/stamp/stamp.jsp?arnumber=7462401 [Accessed 6 Aug. 2017].

Sziládi, G., Ujbányi, T. and Katona, J. (2016). *Cost-effective hand gesture computer control interface - IEEE Xplore Document*. [online] Ieeexplore.ieee.org. Available at: http://ieeexplore.ieee.org/document/7804555/ [Accessed 22 Aug. 2017].

Y., P., P., C., N., T., K., K. and Ch., P. (2016). *Robot-arm control system using LEAP motion controller*. [online] IEEE Xplore. Available at: http://ieeexplore.ieee.org.libezp.utar.edu.my/document/7782091/ [Accessed 13 Aug. 2017].

APPENDICES

APPENDIX A: Source code of Processing software

```
import de.voidplus.leapmotion.*;   // library for leap motion in processing
import controlP5.*;                // library for GUI
LeapMotion leap;
import g4p_controls.*;            // library for GUI
Serial myPort;                    // declare the port as serial type
import processing.serial.*;       // serial library

String s="";
float A = 83.5; //millimeters   // joint to joint length
float B = 98;
float rtod = 57.295779;


ControlP5 cp5;    // declaring cp5 is ControlP5 type.

int myColorBackground = color(0, 0, 0);        // set the background color of the knobs window
int knobValue = 100;                           // the intensity of the background of the knobs window

Knob myKnobA;    // declaring knob names for x
Knob myKnobB;    //for y
Knob myKnobC;    //for z
Knob myKnobD;    //for pitch
Knob myKnobE;    //for roll
Knob myKnobF;    //for grip
```

```
void setup() {

  myPort = new Serial(this, Serial.list()[0], 9600);    // enable serial communication
  myPort.bufferUntil('\n');

  size(700, 400);      // declaring the window size.  700 = build-in var. "height", 400 = build-in var. "width"

  smooth();                        // smooth the edges.
  noStroke();                      // no outline on the edges.
  leap = new LeapMotion(this).withGestures();
  cp5 = new ControlP5(this);

  myKnobA = cp5.addKnob("X")                    // add knob from cp5 GUI library, creating knobs
    .setRange(0, 180)                           // set the range from 0 to 180
      .setValue(50)                             // set the knob initial value
        .setPosition(50, 50)                    // set the knob position in the window
          .setRadius(50)                        // set the radius of the knob
            .setDragDirection(Knob.VERTICAL)    // set the drag direction to change the value, scroll can be used too.
              ;
```

```
myKnobB = cp5.addKnob("Y")
  .setRange(0, 180)
    .setValue(50)
      .setPosition(290, 50)
        .setRadius(50)
          .setDragDirection(Knob.VERTICAL)
            ;
myKnobC = cp5.addKnob("Z")
  .setRange(0, 180)
    .setValue(50)
      .setPosition(550, 50)
        .setRadius(50)
          .setDragDirection(Knob.VERTICAL)
            ;
myKnobD = cp5.addKnob("pitch")
  .setRange(0, 180)
    .setValue(50)
      .setPosition(50, 250)
        .setRadius(50)
          .setDragDirection(Knob.VERTICAL)
            ;

myKnobE = cp5.addKnob("roll")
  .setRange(0, 180)
    .setValue(50)
      .setPosition(290, 250)
        .setRadius(50)
          .setDragDirection(Knob.VERTICAL)
            ;

myKnobF = cp5.addKnob("grip")
  .setRange(0, 180)
    .setValue(50)
      .setPosition(550, 250)
        .setRadius(50)
          .setDragDirection(Knob.VERTICAL)
            ;       }
```

```
void draw() {
  background(myColorBackground);              // set background to no color
  fill(knobValue);                            // fill with intensity, color can be set with 3 variables.
  rect(0, height/2, width, height/2);         // rectangle, rect (x,y,width,height)
  fill(0, 100);                               // fill (gray,opacity if the background)
  int fps = leap.getFrameRate();              // straight away declare fps., get the frame rate from leap library
  ArrayList hands = leap.getHands();          // an Arraylist created named "hands" , and get the info from leap.gethands()


  if (!hands.isEmpty()) {                     // make sure hand is detected
    Hand hand1 = (Hand) hands.toArray()[0];

    PVector hand1_position    = hand1.getPosition();             // get the position of the hand
    PVector hand1_stabilized  = hand1.getStabilizedPosition();   // the stabilized centre position of the palm in mm.
    float   hand1_roll        = hand1.getRoll();                 // get the roll angle
    float   hand1_time        = hand1.getTimeVisible();          // the time duration that the hand visible on the sensor.
    float   hand1_pitch       = hand1.getPitch();                 // get the pitch angle
    float   hand1_grip        = hand1.getSphereRadius();          // get the palm size


    if (hand1_time>1.0) {                                // if the hand visible more than 1 second, then do..
      println(" x: " +hand1_stabilized.x+" y: "+hand1_stabilized.y+" z: "+hand1_stabilized.z+" roll:
         "+hand1_roll+" pitch: "+hand1_pitch+" Grip: "+hand1_grip+"\n");

      float transHand1PosX = map(hand1_stabilized.x, 570, 30, 40, 180);       // mapping data range to servo motor range
      myKnobA.setValue(transHand1PosX);                                        // set the knob value
      int X = int(transHand1PosX);
      posX(X);
      delay(1);


      float transHand1PosY = map(hand1_stabilized.y, 300, 100, 25, 50);
      myKnobB.setValue(transHand1PosY);                                        //  for y-direction (up/down)
      int Y = int(transHand1PosY);
      posY(Y);
      delay(1);



      float transHand1PosZ = map(hand1_position.z, 50, 80, 60, 90);
      myKnobC.setValue(transHand1PosZ);                                        //  for z-direction (front/back)
      int Z = int(transHand1PosZ);
      posZ(Z);
      delay(1);
      //IE(hand1_stabilized.y,hand1_position.z);


      float transHand1Pitch = map(hand1_pitch, -70, 70, 30, 100);
      myKnobD.setValue(transHand1Pitch);                                       //  for pitch angle
      int pitch = int(transHand1Pitch);
      pospitch(pitch);
      delay(1);

      float transHand1Roll = map(hand1_roll,-90,90, 0, 180);
      myKnobE.setValue(transHand1Roll);                                        //  for roll angle
      int roll = int(transHand1Roll);
      posroll(roll);
      delay(1);



      float transHand1Grip = map(hand1_grip, 150, 10, 10, 150);
      myKnobF.setValue(transHand1Grip);                                        //  for grip
      int grip = int(transHand1Grip);
      if (grip>120){
      posgrip(140);
      delay(1);
      }
      else {
      posgrip(70);
      delay(1);
      }

    }
  }
}
```

```
public int IE(float y, float x) {

                                        // inverse kinematics equation (below)

        float M = sqrt((y*y)+(x*x));    // pythygoras theorem
        if (M <= 0)                     // make sure M more than 0 only go for calculation.
          return 1;
        float A1 = atan(y/x);
        float A2 = acos((A*A-B*B+M*M)/((A*2)*M));
        float Elbow = acos((A*A+B*B-M*M)/((A*2)*B));
        float Shoulder = A1 + A2;
        Elbow = Elbow * rtod;
        Shoulder = Shoulder * rtod;


        while ( (int)Elbow <= 0 || (int)Shoulder <= 0)      //make sure more than 0 only calculate.
          return 1;
        float Elbowmapped = map(Elbow, 10, 200, 30, 100);
        float Shouldermapped = map(Shoulder, 80, 30, 10, 70);
        posY(int(Shouldermapped));
        delay(1);
        posZ(int(Elbowmapped));
        delay(1);
        println(" IE, Elbow: "+Elbow+"  Shoulder: "+Shoulder);

        return 0;
      }
```

```
public void posX(int theValue) {      // send processed data to Linkit board
  myPort.clear();
  myPort.write("0,"+theValue+"\n");
 }

 public void posZ(int theValue) {
  myPort.clear();
  myPort.write("1,"+theValue+"\n");
 }

 public void posY(int theValue) {
  myPort.clear();
  myPort.write("2,"+theValue+"\n");
 }

public void posroll(int theValue) {
  myPort.clear();
  myPort.write("3,"+theValue+"\n");
 }

 public void pospitch(int theValue) {
  myPort.clear();
  myPort.write("4,"+theValue+"\n");
 }

 public void posgrip(int theValue) {
  myPort.clear();
  myPort.write("5,"+theValue+"\n");
 }
```

APPENDIX B: Source code of Arduino IDE

```
#include <Servo.h>                              // servo motor library

Servo servo[6];                                 // declare servo motor array

String inputString = "";                        // clear string buffer and flag
String servoString = "";
String commandString = "";
boolean stringComplete = false;

void setup(){
  Serial.begin(9600);                           // start serial communication
  Serial.println("Go...");

  inputString.reserve(10);                      // reserve spaces for the string
  servoString.reserve(10);
  commandString.reserve(10);

  servo[0].attach(3);                             //x
  servo[1].attach(4);                             //for z
  servo[2].attach(5);                             //for y
  servo[3].attach(6);                             //for roll
  servo[4].attach(7);                             //for pitch
  servo[5].attach(8);                             //for gripper


  servo[0].write(110);                           //set the motor angle for starting position
  servo[1].write(50);
  servo[2].write(45);
  servo[3].write(90);
  servo[4].write(0);
  servo[5].write(90);
}
```

```
void loop(){

  parseCommand();                // main functions
  serialEvent();
}

void serialEvent() {                     // combine all data in one string
  while (Serial.available()) {
    char inChar = (char)Serial.read();  // read the new byte
    inputString += inChar;              // add it to the inputString, keep combining to form input string.
    Serial.println(inputString);
    if (inChar == '\n') {
      stringComplete = true;            // set the flag to run the parseCommand.
    }
  }
}

void parseCommand(){                    //disassemble the string into array to drive motors.

  if (stringComplete) {
    inputString.trim();                 //trim is remove leading or trailing whitespace in the string.
    Serial.println(inputString);
    int separatorIndex = inputString.indexOf(',');            //locates the comma (,)
    servoString=inputString.substring(0,1);                   //find substring from after 0 until 1.
    commandString=inputString.substring(separatorIndex+1);    // take the string start after "," till the end.
    int servoNum = servoString.toInt();                       //toInt() converts string to integer
    int servoPos = commandString.toInt();

    servo[servoNum].write(servoPos);    // drive the servo motor.

    inputString = "";           // clear the string:
    servoString = "";
    commandString = "";
    stringComplete = false;
  }
}
```

APPENDIX C: Pinout of Linkit smart 7688 duo board