MUSIC PLAYING ROBOTIC ARM

Ashif Mohammad

Bachelor of Engineering
Mechatronic

Department of Electronic Engineering
Macquarie University

November 13, 2017

Supervisor: Professor Subhas Mukhopadhyay

ACKNOWLEDGMENTS

I would like to acknowledge and express my heartfelt thanks to my academic supervisor Subhas Mukhopadhyay for his continual guidance and assistance throughout the course of my thesis, in teaching me and helping me understand in every way he could.

I would also like to thank my co-supervisor Anindya Nag, who is also a Ph.D student at Macquarie University and like an elder brotherly figure to me, for giving me advice and mentoring me, as well as looking out for me whenever I required any sort of help.

STATEMENT OF CANDIDATE

I, Ashif Mohammad, declare that this report, submitted as part of the requirement for the award of Bachelor of Engineering in the Department of Mechatronic Engineering, Macquarie University, is entirely my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualification or assessment an any academic institution.

Student's Name: Ashif Mohammad

Student's Signature: Ashif

Date: 13/11/17

ABSTRACT

The precision and accuracy of the movement of the robotic arm is a vital criterion for design in applications such as high-precision medical surgeries, vending retrieval system and industrial automation. While normal servos can be used for resolutions up to 255 bits in five degrees of freedom, actuators provide much better control and less steady-state error. This thesis highlights the manipulation of a single robotic arm to play musical notes on a synthesiser piano with the help of Arduino. The calibration of the arm was done using RoboPlus and LabView software, and the error percentages between the theoretical and experimental precision values were measured and analysed. The robotic arm was deployed to work at various speeds, and the difference in completion times of musical notes between the robotic arm and human arm was explored. This project is hoped to open the door for further improvements of the model in the near future and act as a reference for more advanced automation systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The importance of robots and more particularly of the robotic arm, is growing steadily in manufacturing industries as well as in daily applications. Humans and manual labour are being increasingly replaced by robots in a range of fields, including office, military tasks, hospital operations, dangerous environment and agriculture. In order to make the most out of the execution of these uses, many studies on robotic programming and manipulation have been based on giving a higher level of cognitive intelligence.

The most important reason in this is that they outperform humans by a considerable margin when it comes to jobs that require pinpoint accuracy and precision, speed, endurance and reliability. Differences in the functions of human and robotic arms are characterised by attributes such as independent control of joints and fingers, motion range, output forces, speed and weight. Another distinguishing characteristic in the design criteria is the anthropomorphic configuration in which tasks like holding objects in multiple degrees of freedom are tailor-made for human hands. However, the issue of a reliable gripper poses no problems in a situation where the gripper acts as a finger for pressing a button on a keyboard or a synthesiser.

The aim of this thesis project is to manipulate a robotic arm to play a musical note on a piano or a synthesiser when a certain command is given, using MATLAB and/or LABVIEW. Another aim is to precisely mimic a human arm in response to when any button in a keyboard is pressed, via wires or wirelessly.

Chapter 2

Background and Related Work

2.1 Literature Review

2.1.1 Wireless Robotic Arm

Wireless robotic arms are useful in applications where it might be difficult and possibly dangerous for people at work to put their hands in, picking and placing something that can affect itself, or where it's a long way from the user. For example, things like toxic/corrosive chemicals that cannot be picked by human arms or by the military when bombs are needed to be defused. Thus, a locomotive robot can replace a human to do its job.

One of the wireless techniques that can be used in this regard is the RF technology, which uses variations of radio frequency and requires the robot to be suitably compatible with RF. Another platform of wireless control would be through Bluetooth, where the robot has remote Bluetooth connectivity for movements in all directions (forward, reverse, right, left) and is powered by the Arduino Mega and controlled by pulses generated using PWM.

System Architecture

Module	Specification
Interface	Arduino Mega
Controller	Sony PS2 Wireless Controller
Programming Language	Arduino
Actuator	Servo Motor

Table 2.1: Wireless motor arm specs

Design and Development

The mechanical design and assembly of the robot consists of three stages: (1) Main structure arm robot (2) Robot arm design (3) Locomotion (joints and wheels) for carrying the arm. For the arm, brackets and mount are made of aluminium because of their light weight. The arm itself is made of acrylic due to its strength, flexibility and ability to bear the motor weight.

The software used during the development phase are:

- Arduino IDE: Open source software used for coding and debugging.
- Proteus 7 Professional Edition: Used to design the motor driver circuit for the robotic arm.
- Google SketchUp: Designing and simulating the robot.
- PCB Wizard: For PCB design and schematics.

Results and Discussion

Number	Power Supply	Time Taken	Velocity
1	9	4.83	0.20
2	8	5.83	0.17
3	7	23.8	0.04
4	6	N/A	-

Table 2.2: Time taken for various power supply voltages

From the table, it can be seen that for lesser voltages, it takes longer for the robotic arm to reach a specific distance, in this case, 1 metre. Thus, speed of the actuator arm is proportional to the power supply.

2.1.2 Dual Arm Manipulation

There is increasing interest on the topic that life-sized anthropomorphic robotic arms can be used to replace human labour without massive changes to the workplace. The capacity to interchange human and robotic workers in the workers stem from the aim of low-cost and flexible automation systems. There are many independent causes that enable dual arm setup to succeed:

- Resembles the operator: Allows the operator's inherent tendency to perform bimanual tasks to be smoothly transferred to the remote working site.
- Flexible and stiff: Using two robotic arms can combine the strength and stiffness of a parallel manipulator with the adeptness of a serial link manipulator.
- Easy to manipulate: Easy to control tasks that required both hands in domestic and industrial applications, such as dishwashing and offloading deliveries.
- Form factor: Takes up less space and time than a pair of human arms, with losing of the properties of a real arm.

Modelling and Control

Modelling is based on closed-loop system characteristics such as grasp matrix, internal and external forces and points of contact, restraint properties, distribution of stress and force, intrinsic redundancies, implications of singular and dual arm configurations, etc.

There are two main types of cooperative manipulator tasks: (1) Holding with fixed gripping points, where no relative motion occurs between the object and the gripping point and complete interaction is enabled and (2) Holding by contact points or contact areas, where relative motion between the object and gripping area is enabled to some extent and constraints are unilateral. In dual arm configurations repetitiveness may arise even when none when single configurations were used separately. These are useful in facilitating the agility at which the robotic arms work together, and improves the coordination. The modelling gets more complicated if the robotic arm is mounted on free-moving or mobile platforms in multiple degrees of freedom.

The issue of trajectory tracking has traditionally been dealt using linearization models, where the structure of the robotic system is known beforehand to estimate the input-output steps. Non-linear models can be used to solve and linearize the dynamic equations of the closed-loop control systems. This enables the normal and tangential forces to be controlled without the integration of inputs, but also increases the singularity of the decoupling matrix. A good technique in dual-arm manipulation control uses a hybrid

force/position and impedance, which keeps internal forces at a desirable level by kineto-static filtering and keeps track of velocity changes. This can be extended to satisfy a central control loop for internal forces under both rigid and flexible constraints.

Future Work

In order to enable the automation systems to work in an environment without restrictions, it is necessary to provide them with the means to adapt to their surroundings. This makes economic sense, as manually programming all the aspects of a robot's behaviour is painstaking and requires a lot of time and effort, as well as the fact that it is almost impossible to predict all the possible situations that the robot might encounter. Learning by AI (Artificial Intelligence) is one such promising development that can solve these problems and enhance performance to a great extent.

Another possibility in future research of robotic systems, and indeed the dual-arm manipulation systems would be towards integrating all the elements from systems engineering using tools from cognitive procedures.

2.2 Piano and Tunings

Piano is an acoustic, stringed musical instrument invented in the early 1700s in Italy. The word itself is the shortened form of *pianoforte*, which in Italian means soft and loud. Made out of a wooden case which surrounds the soundboard and metal strings, pressing a key causes the strung metal, which is already under high tension, to be vibrated at the resonant frequency. The vibrations are transmitted through to the soundboard where the frequency is amplified by converting the mechanical energy of the air molecules into sound. Unlike some musical instruments such as the pipe organ and the harp, piano can produce variations in volume and pitch of a tune according to how forcefully the key is pressed.

2.2.1 Components and Construction

Every piano is made up of six basic structural aspects, which are made out of materials that has strength and longevity:

- Keyboard
- Hammers
- Dampers
- Bridge
- Soundboard
- Metal strings

The massive size serves the purpose of being an immobile object where a flexible soundboard is in the best situation to vibrate. This conserves the vibrational energy within the strings rather than dissipate and be lost in the air as heat, improving the efficiency. The piano wires are made out of carbon steel to withstand long periods of extreme tension.

Every modern keyboard is made up of 52 white keys and 36 black keys totalling to 88 keys, which comprise of seven octaves and a minor third. Some have an extended range of eight octaves and 97 keys. These extra ones prevent visual disorientation for performers and give increases resonance from the related strings.

2.2.2 Mechanism and Theory

When a note is struck, a chain reaction produces the sound. Firstly, key raises the 'wippen' mechanism, which presses the jack against the hammer and also raises the damper. After the hammer strikes the string it falls back immediately, making a sound. When the key is released, the damper falls back onto the strings, preventing resonance and thus stopping the sound. The non-uniform shape of the soundboard and bridge makes sure that vibration occurs strongly at all frequencies. The three factors influencing the pitch of a string are:

- Wire length: The smaller the length, the higher the pitch.
- Mass per unit length of the wire: The thicker the wire, the lower the pitch.
- Tension: The tighter the wire, the higher the pitch.

The frequency of the string tune can be nicely summed by the following equations:

$$v = f\lambda \tag{2.1}$$

where v is the speed of the wave, f is the frequency and λ is the wavelength. Superposition occurs on the strings, resulting in a reflected wave pattern of $\lambda =$

$$\frac{2L}{n}$$

Hitting the key with greater intensity increases the wave amplitude and thus the volume.

2.3 Robotic Arm

According to RIA, a robot can be defined as a re-programmable, multifunctional manipulating system that can move materials, parts, tools or specialised devices through variable programmed motion for a wide range of tasks. A typical robotic arm may contain all or some of the following accessories:

- Manipulator/rover: Main body, which consists of joints, links and other structural elements.
- End effector: Usually the part that handles objects, makes inter-machine connections or performs specified tasks. Varies in terms of complexity and size.
- Actuators: Responsible for the movement of the manipulators, such as servos and stepper motors.

Figure 2.1: Piano

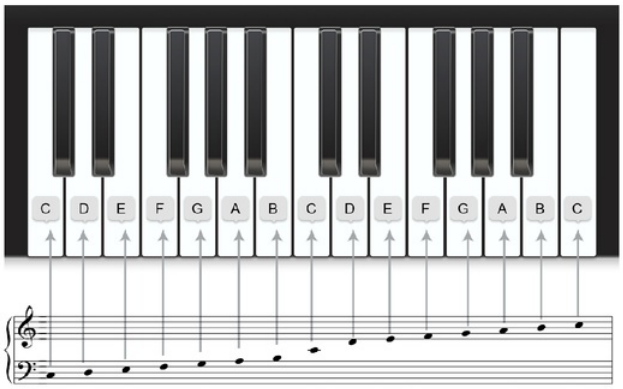


Figure 2.2: Piano keyboard

- Sensors: Gathers data and information about the internal perimeters and variables of the robot to and from the external environment.
- Controller: Controls and coordinates the behaviour of the actuators with feedback received from a computer.

The joints in robotic arms can be Some of the common configurations used in robotic arm applications are given in the following table:

Configuration	Prismatic joints	Revolute joints
Cartesian/Rectangular	3	0
Cylindrical	2	1
Spherical	1	2
Articulated/Anthropomorphic	0	3
SCARA*	1	2

Table 2.3: Joint configuration in robotic arms

2.4 Kinematics

Kinematics refer to the study of motion/kinematics of robots and its components. It consists of the relationships between position, velocity and acceleration of the joints and links of the manipulator, which can be an arm, finger or a leg, without taking into account the external forces causing the motion. Thus, the inter-relationships between motion, and its related forces and torques make up the dynamics of the robotic system.

Kinematics considers the aspects of repetitiveness, avoiding collisions and singularity avoidance. While analysing, each part of the robot is assigned a frame of reference, which could be either a world reference frame, joint reference frame or tool reference frame. This means that each section of the robot individual frame given to its mobile parts.

2.4.1 Forward and Inverse Kinematics

When analysing manipulator position, the kinematics problems can be divided into two categories: forward and inverse kinematics.

Forward kinematics is also known as direct kinematics. In this, the length of each link and the angle of each joint is provided. Based from these information, it is necessary to figure out the position of any point of the robot in Cartesian (x-y-z) coordinate system. The opposite is true for inverse kinematics. In this, the length of each link and the position of a specific point of the robot in Cartesian (x-y-z) coordinate system is given. Based from this, it is necessary to calculate the angles required for each joint/link to reach that position.

In the case of serial chain forward kinematics, there is always a unique solution to the problem since a joint position vector results in only one end effector point. This can be done using simple geometrical methods such as trigonometric laws or, by using transformation matrices. For parallel chains, computation gets more complex but is possible using algebraic geometry in closed loop system.

If the lengths of link a_1 and a_2 are already known, and angles θ_1 and θ_2 are given, the position of the end coordinates X and Y can be found using the following equations:

$$X = a_1 cos\theta_1 + a_2 cos\theta_1 + \theta_2 \tag{2.2}$$

$$Y = a_1 sin\theta_1 + a_2 cos\theta_1 + \theta_2 \tag{2.3}$$

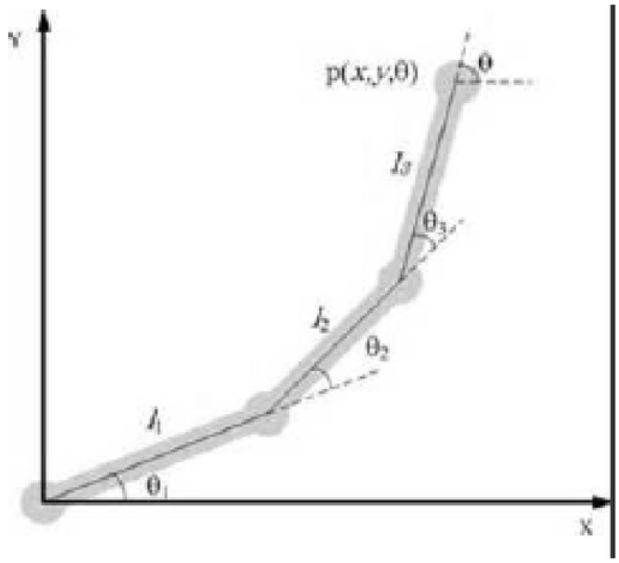


Figure 2.3: Forward Kinematics

Inverse kinematics works in exactly the opposite principle to forward kinematics, and is also known as indirect kinematics. In this, the length of each link and the end position of some point on the motor is given in Cartesian coordinates. Based from these information, it is necessary to calculate the angles of each joint required to attain that desired position.

Inverse kinematics is much more complex and more complicated to solve than its forward counterpart. It can be solved using either the analytic method, where trigonometric laws were used or the inverse Jacobian method. The latter is used when there are more than two linkages are involved and matrix equations need to be applied, preferably on MATLAB.

The equations used to solve a 2-D planar manipulator are as follows:

$$x^2 + y^2 = a_1^2 + a_2^2 - 2a_1a_2cos\pi - \theta_2 \tag{2.4}$$

$$cos\theta_2 = \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1a_2} \tag{2.5}$$

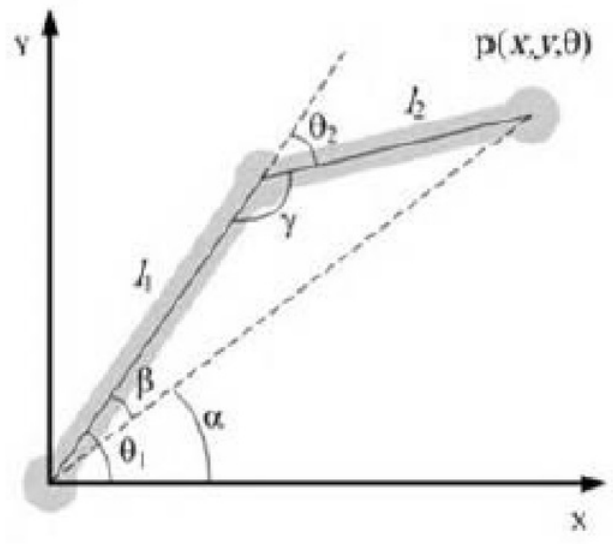


Figure 2.4: Inverse Kinematics

2.5 Degrees of Freedom

In a mechanical context, degrees of freedom are a specific pathway in which a robotic arm system can move, or rotate. Each degree of freedom corresponds to the number of independent displacement routes or field of motion. However, degree of freedom is not the same as dimension. Because a robotic arm may move in three dimensions, but can have different degrees of freedom depending on how it was assembled.

Typically, a robot with three mobile joints will have three axis and degrees of freedom, a robot with four mobile joints will have four axis and degrees of freedom and so on, and so forth.

Figure 2.5: Primary Degrees of Freedom

Some of the basic functions of a robotic arm with various degrees of freedom are given below:

- One axis: pick an object up and move in a straight line
- Two axis: pick an object up and move it horizontally or vertically
- Three axis: pick an object up and move it around the xyz coordinate system without changing the object's orientation
- Four axis: pick an object up and move it around the xyz coordinate system while changing the orientation along one axis
- Six axis: pick an object up and move it around the xyz coordinate system while changing the orientation along two axis
- Seven axis: pick an object up and move it around the xyz coordinate system while changing the orientation along three axis

2.5.1 Transformation and Rotation

Rotation matrices for single axis can be given by:
R(x, θ) =

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix}$$

R(y, θ) =

$$\begin{bmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix}$$

R(z, θ) =

$$\begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A point P can be translated to P' using the following general equation: $\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} =$

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Positive rotation angles move in an anticlockwise direction about an axis, while negative angles move in a clockwise direction. The three orientations of rotation can be outlined as:

- Roll - rotation around front-to-back axis
- Pitch - rotation around side-to-side axis
- Yaw - rotation around vertical axis

2.5.2 D-H Principle

Denavit-Hartenberg parameters can be defined as the length of the perpendicular between the joint axes, $Z_{(i)}$ and $Z_{(i-1)}$. These two axes can be thought as imaginary lines in 3D space, and the common perpendicular is the smallest distance between the two axes as well as being perpendicular to the Z axes itself.

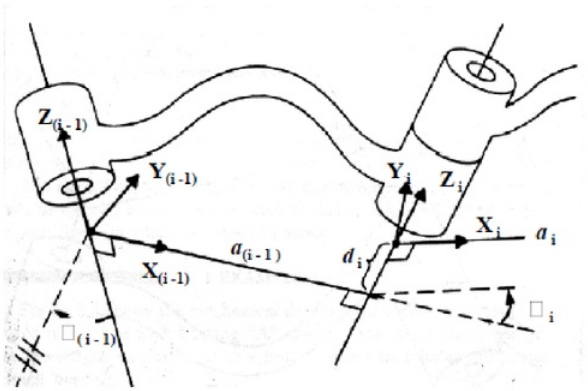


Figure 2.6: D-H parameters

The final Denavit-Hartenberg matrix can be deduced using the following matrix:

$$\begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & a_{i-1} \\ sin(\theta_i)cos(\alpha_{i-1}) & cos(\theta_i)cos(\alpha_{i-1}) & -sin(\alpha_{i-1}) & -sin(\alpha_{i-1})d_i \\ sin(\theta_i)sin(\alpha_{i-1}) & cos(\theta_i)cos(\alpha_{i-1}) & cos(\alpha_{i-1}) & cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The D-H matrix is similar to the homogeneous rotation matrices in that it transforms a matrix from one coordinate to the other. The four DH parameters used in tables denote the following:

- d: offset from reference normal to the new z
- θ: angle about the reference z, from previous x to new x
- r: length of the common normal
- α: angle of the common normal

Chapter 3

Hardware and Software

3.1 Dynamixel AX-12A Robotic Arm

3.1.1 Dynamixel AX-12A

Schematics

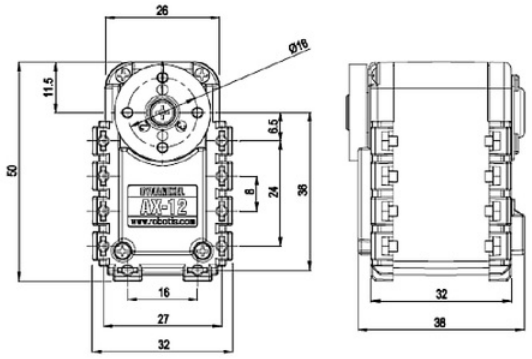


Figure 3.1: AX-12A physical dimensions

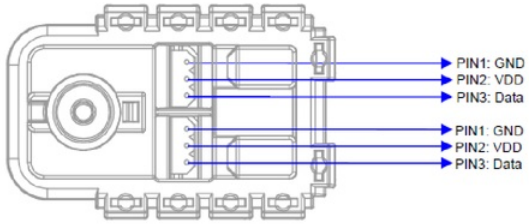


Figure 3.2: Dynamixel AX-12A wiring

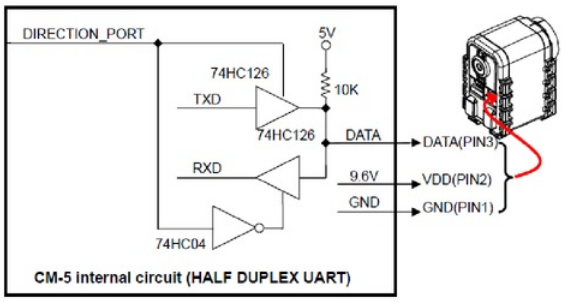


Figure 3.3: Connection to UART

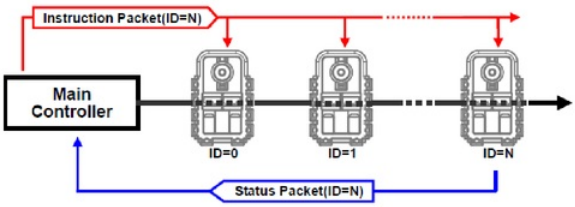


Figure 3.4: Communication protocol

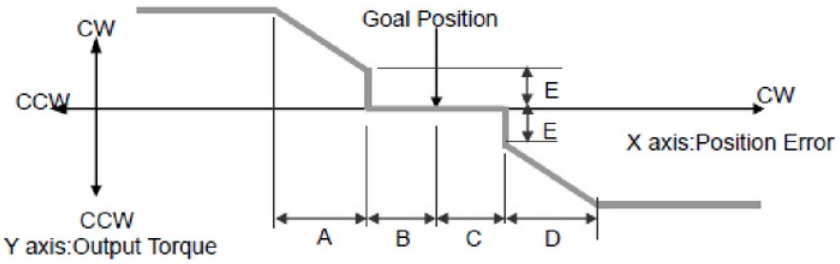


Figure 3.5: Compliance margin

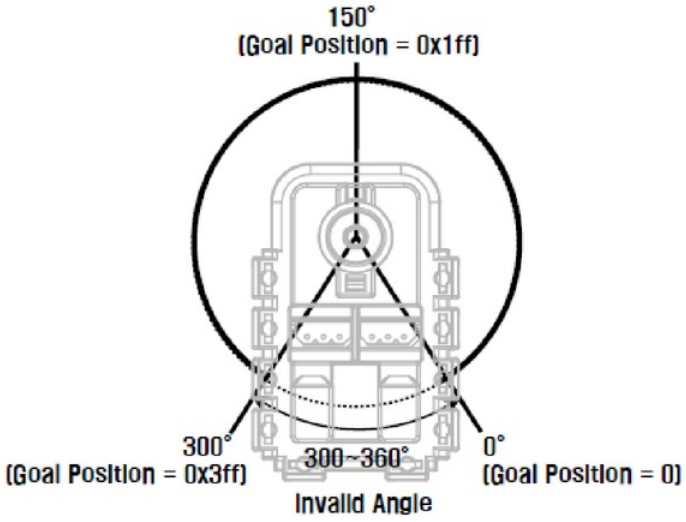


Figure 3.6: AX-12A position constraints

Specifications

AX-12A Specs	
Operating voltage	**12V**
Stall torque	15.3 kg.cm
No-load speed	59 rpm
Weight	55g
Size	32x50x40 mm
Resolution	0.29°
Reduction ratio	1/254
Operating angle	300°
Operating voltage	9-12V
Maximum current	900mA
Standby current	50mA
Internal operating temperature	-5°C - 70°C
Protocol	TTL half duplex asynchronous serial
Module limit	254 valid addresses
Com speed	7343-1 Mbps
Position feedback	Yes
Temperature feedback	Yes
Load voltage feedback	Yes
Input voltage feedback	Yes
Compliance	Yes
Material	Plastic gears and body
Motor	Cored motor
Manual	*See appendix for the AX-12A manual*

Table 3.1: AX-12A actuator specifications

3.1.2 Robotic Arm Assembly

Components

The components required to assemble the AX-12 Robotic arm can be classified into several categories:

Electronics/Software:

- 12V, 5A power supply (optional)
- AX-12A/AX-18A servos

Parts:

- Pivot bpx
- Turntable
- Turntable braces (upper and lower)
- Main beam
- Servo brackets (brace and short rotator)
- 90° braces
- Servo forearm
- Palm bracket
- Grippers (dual hands)
- Flat and gear braces
- Servo gear brace
- Sensor bracket

Tools:

- Phillips head screwdriver
- White grease

- Thread locker, such as Nylock
- Needle nose pliers

Miscellaneous:

- AX-12 Smart Robotic Arm Manual
- Cable ties
- 60 tooth gear sets
- 12" extended servo cables
- Gripper padding
- Turntable bearings
- Spacer (turntable and gripper gear)
- AX-12A pivot spacer sets
- Nuts and lock nuts
- Washers and lock washers
- Screws and servo pivot screws
- Nylon spacers

Pre-Assembly Tips

- Workspace should preferably have a clean environment and lots of space.
- All the components and parts should be well organised and within easy reach.
- The robotic arm kit requires all parts to be assembled in exactly the same order as illustrated in the manual, so it is essential to take enough time and not rush through.
- The figures and instructions should be referred to thoroughly before starting any part of the construction.
- The orientation and direction of screws and the parts should always be noted, as well as which side of the robotic arm is being assembled.

Procedure

1. Installing the nuts to the servos: The nuts needed to be placed into slots, sitting straight up and down. Place the needle nose pliers until the nut clicks in place.

2. Screw installation and nut replacement: Every screw was installed using the "Loctite" thread locker coating. The nuts can be replaced by removing the set screw and servo horn with a small flathead screwdriver. Replacement nuts are found within the box.

3. Preparing the servos: Each servo was installed according to its address. Each address was assigned a specific location, and this unique address was set up using RoboPlus.

4. Connecting the servos: AX-12A servos have two sides, where left port is the input and right is the output. The servos were connected sequentially.

5. Arm assembly:

   - Second sets of nuts were installed onto the second slots of the servo.
   - Address 1 servo was installed into the base of the turntable top.
   - The carbon steel bearings were attached at the top end of the base.
   - The lower turntable braces were placed on the turntable by aligning the holes.
   - The turntable spacer was installed on the other side of the assembly using 12mm screws.
   - The turntable assembly was installed onto the first servo by aligning the turntable holes with that of the servo, and then making an even press fit.
   - The upper turntable brace was then fitted to the lower one, and a tight fit was ensured to make the assembly stiff.
   - Nuts were installed on the bases of servos 2-5, with servo brackets and nylon spacers attached.
   - The short rotator servo brackets were installed on each of the servos using 5mm screws.
   - The servo wires were connected to the output port of each of the servos serially.
   - The main beam was installed to the shorter servo brackets, and the main beam was centered before tightening the screws.
   - Servos 4 and 5 were attached on the opposite sides of the main beam.

- Nuts were installed onto the slots of servo 6 and 7.
- The two servos were oriented at 90 degrees to each other to facilitate installation of short and long wires, with the servo brackets installed.
- The same procedure is repeated for the last servo.
- The gripper pads were assembled and rubber attached to the inner side to reduce friction.
- The gripper assembly was then attached to the main beam to complete the procedure.

6. Turning the gripper assembly: The assembly lock nuts were tuned for better performance. The lock nuts are tightened to the point that it's neither too loose nor too tight.
7. Checking gripper alignment: The gripper alignment was checked by making sure the ends of the gripper touch together. The unevenness was resolved by tightening or loosening the appropriate screws.
8. Setting initial parameters of the robotic arm: The initial angle of the base is set around 60 degrees, secured in position using lock nuts and screws.
9. Wiring the servos: The servos were wired serially to each other in a way that does not interfere with their movement and were set at their maximum constraints before wiring was done.
10. Mounting the servos: The robotic arm was mounted on a secure spot and clamped on using fasteners and sticky tape, and the distance between the user and the robotic arm was preferably 3 feet away.
11. Programming the arm: The programming of the robotic arm could be done using various programs such as LabView, MATLAB or Arduino.
12. Powering the arm: The arm was powered using a separate power supply delivering 12V and 5A of current.
13. Connecting the arm: The robotic arm was connected using a USB2Dynamixel, which interfaced the arm to the computer.

Please check the AX-12A manual for detailed description of assembly. The manual for the robotic arm can be found at here

3.2 Arduino

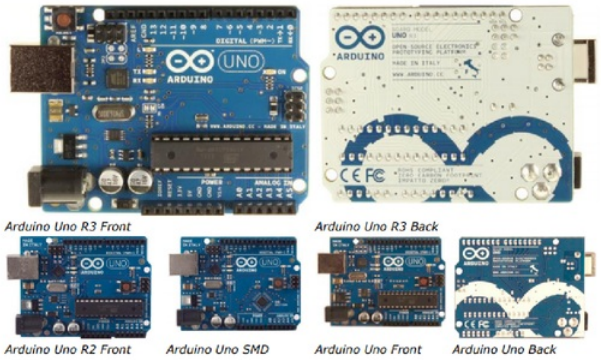


Figure 3.7: Arduino Uno R3

Arduino Uno is based on the ATmega328 series of microcontroller boards, with 14 digital input and output pins, USB connection line, a power jack, 6 analog inputs and PWM outputs, an ICSP header and a 16MHz ceramic resonator. It has built-in self sustaining system for the microcontroller and only needs to be connected to a computer to get started.

The R3 version was used in this project, which differs from the older versions of Uno in that:

- SDA and SCL pins added to the AREF pin, which allows shields to be compatible with the microcontroller, as well as AVR and Arduino Due.
- Better Reset circuit.
- 8U2 replaced by Atmega 16U2

3.2.1 I/O

The outline of the I/O pins for the Arduino are given as follows:

1. VIN: External input power supply pin to the Arduino

2. 5V: Output pin for a regulated 5V power supply on the board
3. 3V3: 3.3V supply pin generated by the on-board regulator
4. GND: Ground pin
5. Serial(RX and TX): Receive RX data and transmit TTL data serially
6. External interrupts (2 and 3): Can be configured to form interrupts on rising or falling edge of signals
7. PWM (3,5,6,9,10,11): 8-bit output with the analogWrite() function
8. SPI: support SPI communcation via the libraries
9. LED: When the pin is HIGH, the LED turns on, and vice versa.
10. TWI: Supports communication via the wire library
11. AREF: Reference voltage for analog inputs
12. Reset: Resets the entire microcontroller

Communication

There are several ways through which Arduino can communicate to a computer. The UART TTL communication channel is available for digital pins 1 and 0. Atmega16U2 interfaces this communcation which appears as a COM port on the software. Arduino also enables communication via a serial monitor, thus enabling multiples commands to work without actually tinkering with the original code. The SoftWareSerial library allows communication with the PWM pins, while the I2C bus allows the built-in libraries to be called as functions.

3.2.2 Specifications

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage	7-12V
Input Voltage (limits)	6-20V
Digital I/O pins	14
Analog Input Pins	6
DC Current per I/O pin	40mA
DC CUrrent per 3.3V pin	50mA
Flash memory	32kB (0.5kB used by bootloader)
SRAM	2 kB
EEPROM	1 kB
Clock speed	16MHz

Table 3.2: Arduino Uno R3 specs

3.2.3 Schematics and Pinout

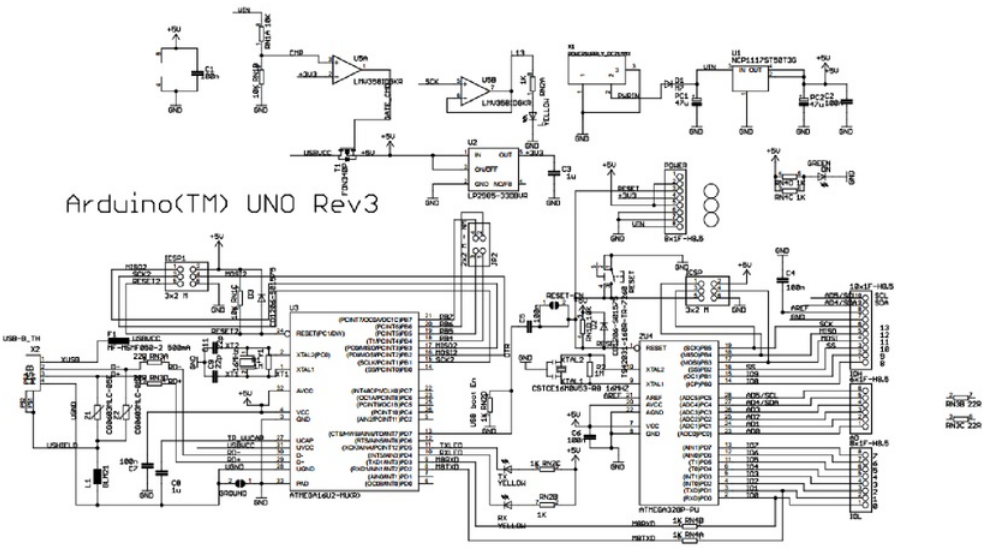


Figure 3.8: Arduino Schematic

For detailed references on Arduino and Atmega microcontrollers, please see here

3.2.4 Servo Shield

The main reason for the necessity of a servo shield is that the Dynamixel AX-12A actuators have a half-duplex serial and the servo shield integrates a half-duplex circuit inside, using the concept of Whetstone rectifying bridge. The shield makes it easier to multiple servos with daisy chain connection to Arduino Uno, making it the perfect auxiliary addition to this project.

Figure 3.9: Dynamixel AX-12A servo shield

The transmitting wire from UART is connected to all the actuators, which means that if a command is sent without the mention of a servo ID, it is relayed to all the servos. If the command has a specific servo address, then only the servo with the matching ID will receive the command prompt. This hierarchical communication protocol allows upto 200 servos to be connected together, each its own velocity, torque and current.

Pinout and Schematics

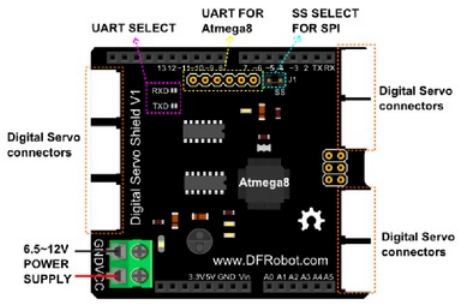


Figure 3.10: Servo Shield Pinout Diagram

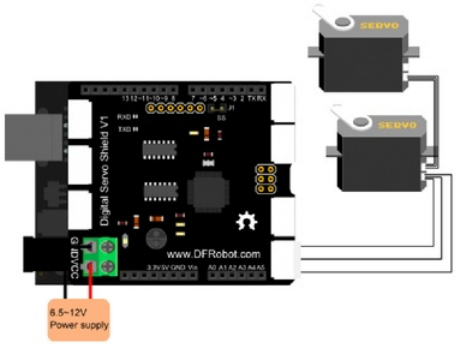


Figure 3.11: Shield Connection

m

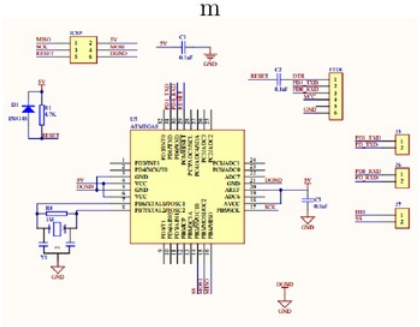


Figure 3.12: Servo shield Schematic: MCU

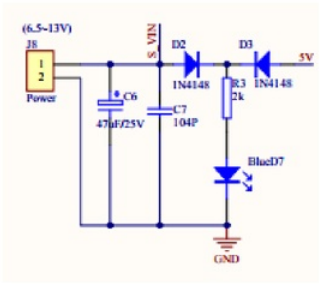


Figure 3.13: Servo shield Schematic: Power

For more details on the servo shield user manual, click here

3.3 Piano: Launchkey Mini

The piano chosen to be used in this project is the LaunchKey Mini synthesiser, because it is small, portable, can be connected via a computer and thus can be played in parallel to the robotic arm, and the dimensions fall within the constraints of the arm movement, since a generic sized piano keyboard would be too big for the robotic arm to operate on effectively.

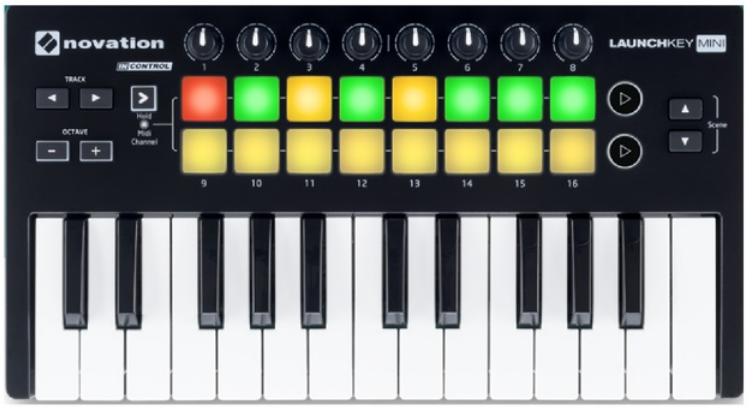


Figure 3.14: Novation LaunchKey Mini synthesiser

3.3.1 Technical Specifications

- ☐ 25-note velocity sensitive keyboard
- ☐ 16 velocity sensitive drum pads
- ☐ 8 knobs, 2 octave buttons, track buttons and navigation buttons
- ☐ inControl button
- ☐ Plug-and-play technology
- ☐ MicroUSB socket; USB bus powered system
- ☐ Kensington security slot
- ☐ Ableton Live software included
- ☐ Dimensions: 325mm x 175mm x 43mm
- ☐ Mac OSX, Windows and iOS

Chapter 4

Experimenting and Setup

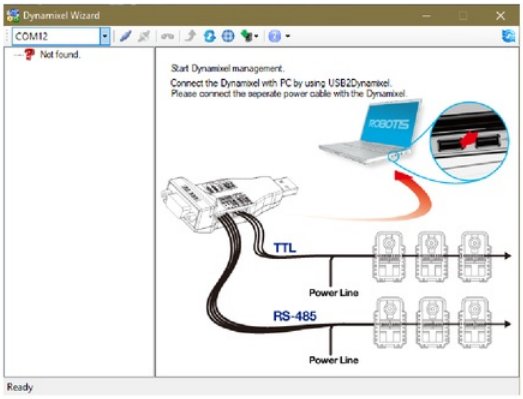
Firstly, the Dynamixel AX-12A servos will be initially set up by the RoboPlus program. Then the initial calibration of the keyboard of the synthesiser will be done using LabView, and later by programming via Arduino. The coding done using Arduino will be responsible for the final movement of the robotic arm to play notes on the synthesiser, and the serial monitor will be used to execute the commands. The musical notes used in this project under investigation is the intro of 'In the End' played by Linkin Park and the first few notes of the theme song of Game of Thrones. Various speeds (in rpm) were tried on the robotic arm to compare the differences between the time taken to complete the notes, in contrast to the actual time taken by human hands. Then the results were compared and qualitative and quantitative analysis made.

4.1 Setting Up the Motor IDs by RoboPlus

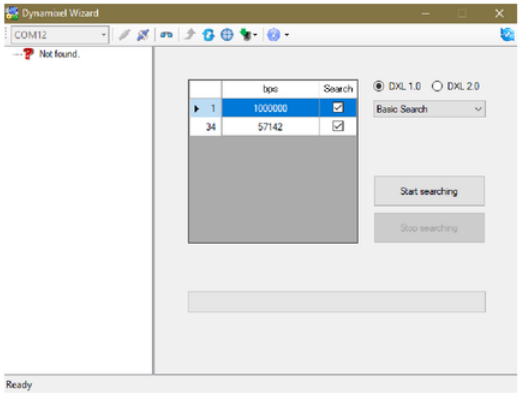
Set up the Dynamixel robotic arm by making sure the USB2Dynamixel cable is set onto TTL mode and connected to a PC, and the robotic arm is be given its separate power supply.

Open the RoboPlus program, and click on the Dynamizel Wizard pop-up icon.

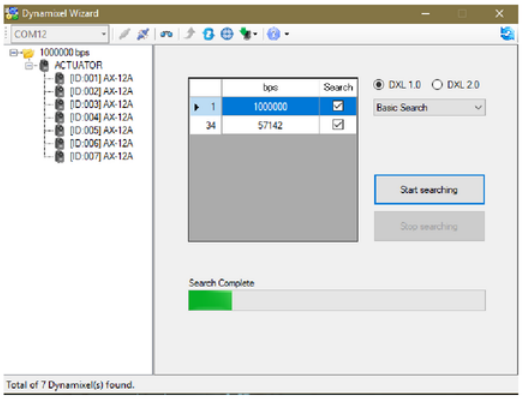
Now click on the 'Open Port' icon to turn on the connections. Make sure the COM port is recognized.



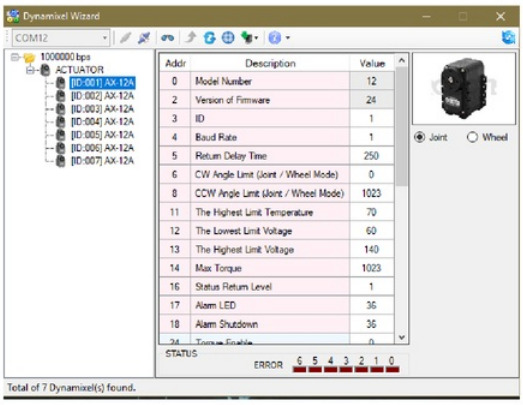
Click on DXL 1.0 and perform a Basic Search to find the Dynamixel actuators, based on the baud rate they are set at. If basic search doesn't show the actuators, a Custom Search or an Advanced Search can be performed.



7 actuators show up at the baud rate of 1000000 bits per second (bps). If they are set at 57142 bps, they will need to be changed to 1000000 bps to function better for Arduino.

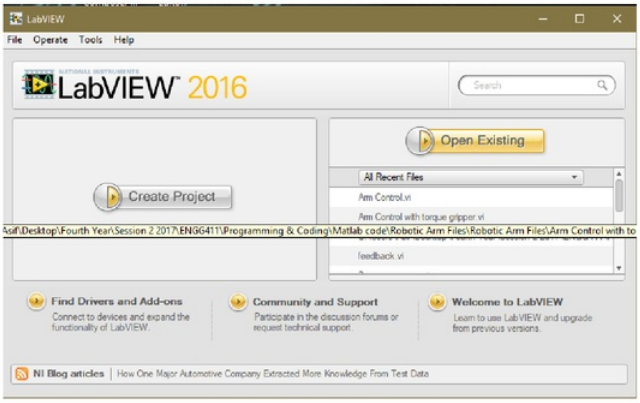


If all this procedure works accordingly, the servo IDs will have been set and the programming is good to go.

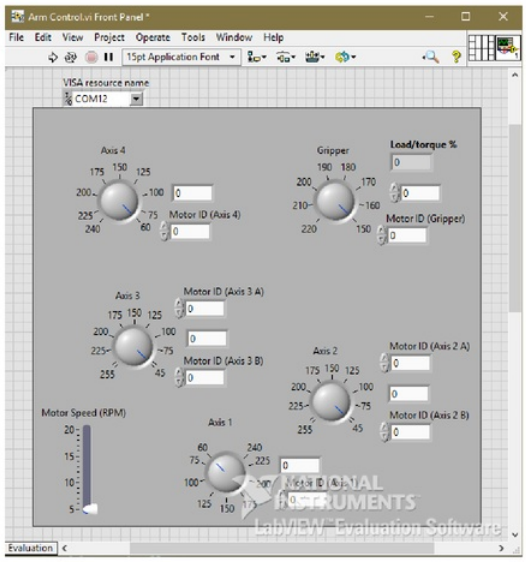


4.2 Calibrating using LabView

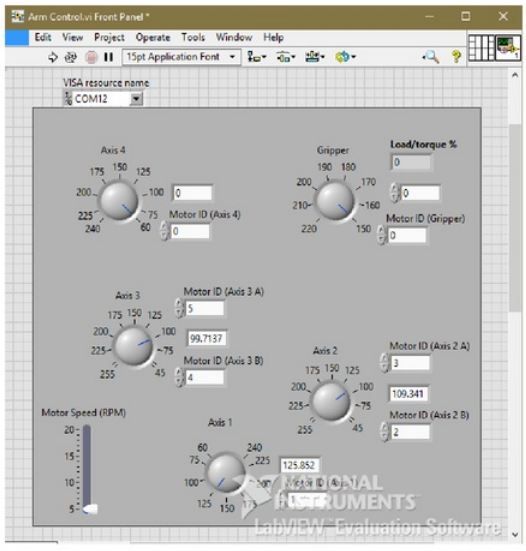
Open up the LabView software, and click the Arm Control program.



Set up the COM port to make the program ready for execution. Make sure the correct NI-VISA Wizard version is installed so that the COM port can be recognized and interfaced with the computer.

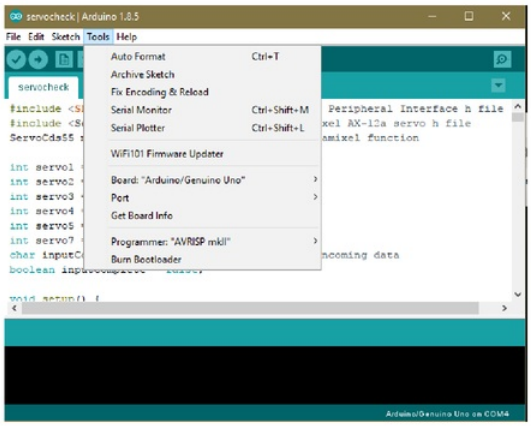


Set the servos IDs 1-5, and rotate the knobs until they are on a satisfactory position touching the first reference key on the keyboard/synthesiser. Repeat the procedure for every keyboard, and the results can be seen in the LabView Calibration section in Results.



4.3 Calibrating and Programming using Arduino

Open the Arduino software, and set up the COM port to the suitable number (should be around 1-15) and the board to Arduino Uno.



SPI, and the *ServoCds55* libraries were added; the SPI is a built-in function at Arduino, whereas the ServoCds55 file was imported externally as a .ZIP file. The global variables for the servo IDs were initialized.

```
#include <SPI.h>              // including Serial Peripheral Interface h file
#include <ServoCds55.h>       // including Dynamixel AX-12a servo h file
ServoCds55 myservo;           // defining the dynamixel function

int servo1 = 1;   // Servo ID:1
int servo2 = 2;   // Servo ID:2
int servo3 = 3;   // Servo ID:3
int servo4 = 4;   // Servo ID:4
int servo5 = 5;   // Servo ID:5
int servo7 = 7;   // Servo ID:7
char inputCommand ;           // string to hold incoming data
boolean inputComplete = false;
```

The *void setup* function initialises the whole code, and the baud rate is set to 1000000 bps via **serial.begin**.

```
void setup() {
  Serial.begin(1000000);     //baud rate:1000000
  myservo.begin();
}
```

The *void loop* function calls up two other functions, *serialEvent* and *controlServo* within itself as nested functions, as it continuously iterates the program until it is stopped or the power switched off.

```
void loop() {
  serialEvent();
  if (inputComplete) {
    Serial.println("Press any key to move the arm to the desired position");
    Serial.print("Your command is: ");
    Serial.println(inputCommand);
    Serial.println("");
    controlServo(inputCommand);

    // Clearing the command:
    inputCommand = 0;
    inputComplete = false;
  }
}
```

The *serialEvent* function sets the boolean condition that if a suitable character is not pressed, the program will not work. The *controlServo* function serves as the core part of the function, which makes the robotic arms move and can be controlled via the serial monitor.

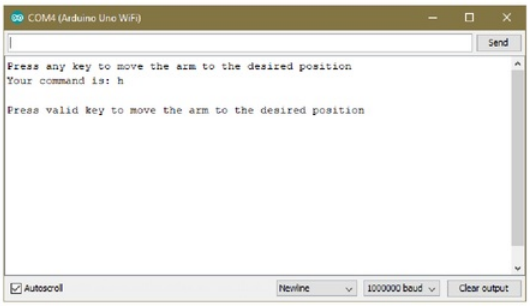
```
void serialEvent() {
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    if (inChar == '\n') {
      inputComplete = true;
      break;
    }
    inputCommand += inChar;
  }
}

void controlServo(char val) {
  switch (val) {
```

The characters from A-J denote the various octaves of key notes from the piano. The calibration of the key notes are shown in the Arduino Calibration section in Results. The following diagram shows the code for pressing on the note J.

```
case 'J':
  myservo.setVelocity(35);
  myservo.write (servo1, 30);
  delay(1000);
  myservo.write (servo4, 218);
  myservo.write (servo5, 218);
  delay(1000);
  myservo.write (servo2, 250);
  myservo.write (servo3, 250);
  delay(1500);
  myservo.write(servo2, 225);
  myservo.write(servo3, 225);
  delay(1000);
  myservo.write(servo4, 225);
  myservo.write(servo5, 225);
  delay(1000);
  myservo.write(servo1, 60);
  delay(1000);
  break;
```

The Dynamixel servo shield is attached to the Arduino Uno, and connected to the computer. This allows the serial monitor to be utilized. The program is now to ready to be executed.



The full Arduino code can be found on the Appendix chapter.

Chapter 5

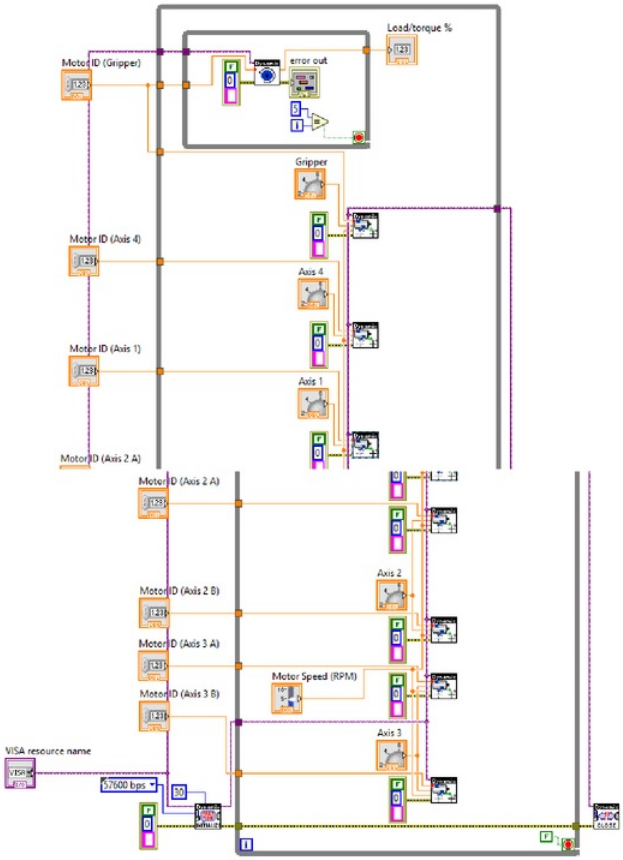
Results and Analysis

5.1 Results

5.1.1 Arduino Program

The entire Arduino code used to program the robotic arm can be found in Appendix

5.1.2 LabView Program



5.1.3 Arduino Calibration

Note	Equivalent key	Motor ID:1	Motor IDs:2 and 3	Motor IDs:4 and 5
C0	K	90	255	220
D0	L	86	255	220
E0	M	82	255	208
F0	N	77	255	208
G0	O	73	250	218
A1	A	69	250	218
B1	B	65	250	218
C1	C	60	250	218
D1	D	55	250	218
E1	E	51	250	218
F1	F	47	250	218
G1	G	43	255	208
A2	H	38	255	208
B2	I	34	255	220
C2	J	30	255	220
E11	P*	54	250	208

Table 5.1: Piano Keyboard Calibration using Arduino

5.1.4 Music Notes Time Comparison

Note 1: In the End

Notes	Speed(rpm)	Time taken T_1	Time taken T_2	Time taken T_3	Average
Actual time	N/A	22	22	22	22
Manually	N/A	31	28	25	28
Robotic arm	5	567	538	551	552
-	10	475	446	457	459.33
-	20	411	395	423	409.67
-	30	343	328	229	300
-	40	278	256	262	265.33
-	50	226	198	201	208.33

Table 5.2: Time taken to complete the piano notes for 'In the End'

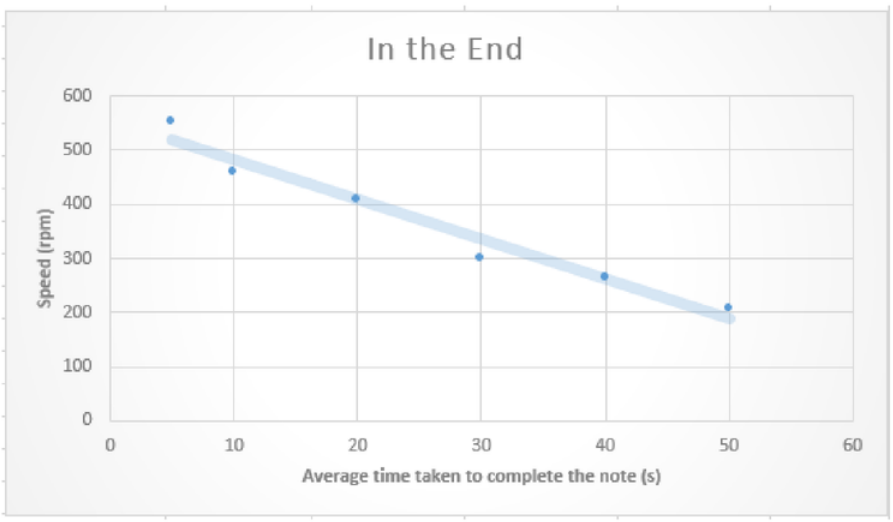


Figure 5.1: Time taken to complete the piano note of the intro of 'In the End

Note 2: Game of Thrones Theme Song

Notes	Speed(rpm)	Time taken T_1	Time taken T_2	Time taken T_3	Average
Actual time	N/A	18	18	18	18
Manually	N/A	26	28	25	26.33
Robotic arm	5	634	617	622	624.33
-	10	576	556	564	565.33
-	20	499	514	503	505.33
-	30	411	427	422	420
-	40	334	345	367	348.67
-	50	256	267	244	255.67

Table 5.3: Time taken to complete the piano notes for GoT theme

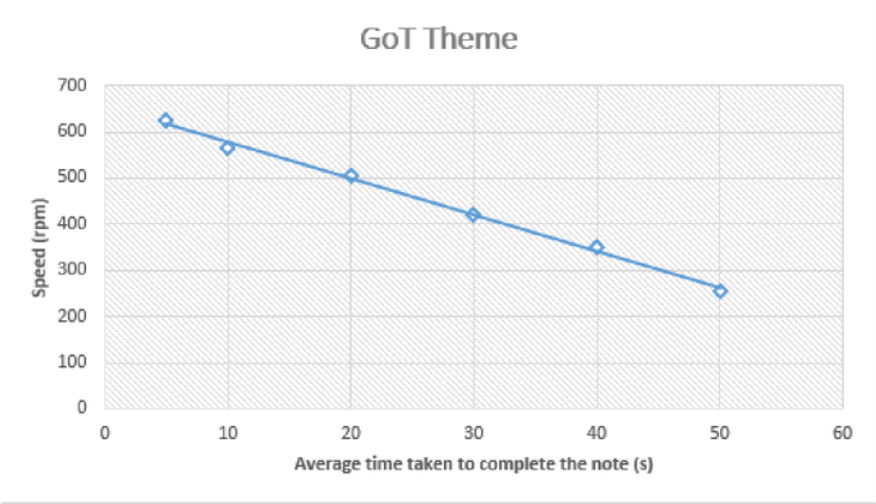


Figure 5.2: Time taken to complete the piano note of GoT theme song

5.1.5 LabView Calibration

Goal Position	Actual Position	Difference	Error Percentage
90	89.7361	0.2639	0.293222
180	179.472	0.528	0.293333
60	59.824	0.176	0.293333
150	149.56	0.44	0.293333
180	179.472	0.528	0.293333
150	150.147	0.147	0.098
255	254.252	0.748	0.293333
45	44.5748	0.4252	0.944889
150	149.56	0.44	0.293333
150	149.56	0.44	0.293333
0	0.29325	0.29325	0.29325

Table 5.4: Results of robotic arm actuator ID: A1(1)

Goal Position	Actual Position	Difference	Error Percentage
90	89.7361	0.2639	0.293222
180	179.472	0.528	0.293333
60	59.824	0.176	0.293333
150	149.56	0.44	0.293333
180	179.472	0.528	0.293333
150	150.147	0.147	0.098
255	254.252	0.748	0.293333
45	44.5748	0.4252	0.944889
150	149.56	0.44	0.293333
150	149.56	0.44	0.293333
0	0.29325	0.29325	0.29325

Table 5.5: Results of robotic arm actuator ID: A2(a)

Goal Position	Actual Position	Difference	Error Percentage
90	89.7361	0.2639	0.293222
180	180.059	0.059	0.032778
60	59.5308	0.4692	0.782
150	150.147	0.147	0.098
180	180.059	0.059	0.032778
150	149.56	0.44	0.29333
255	255.132	0.132	0.051765
45	44.2815	0.7185	1.596667
150	150.147	0.147	0.098
150	150.147	0.147	0.098
0	0.29325	0.29325	0.29325

Table 5.6: Results of robotic arm actuator ID: A1(a) with speed=10 rpm

Goal Position	Actual Position	Difference	Error Percentage
90	89.7361	0.2639	0.293222
180	179.472	0.528	0.293333
60	59.824	0.176	0.293333
150	149.56	0.44	0.293333
180	179.472	0.528	0.293333
150	150.147	0.147	0.098
255	254.252	0.748	0.293333
45	44.5748	0.4252	0.944889
150	149.56	0.44	0.293333
150	149.56	0.44	0.293333
0	0.29325	0.29325	0.29325

Table 5.7: Results of robotic arm actuator ID: A1(a) with speed=20 rpm

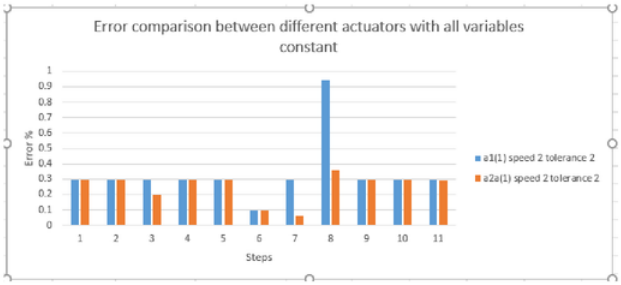


Figure 5.3: Comparing error percentage between two robot arm actuators with all parameters constant

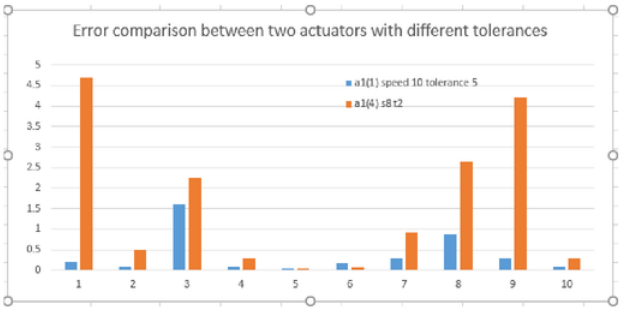


Figure 5.4: Comparing error percentage between two robot arm actuators with different tolerances

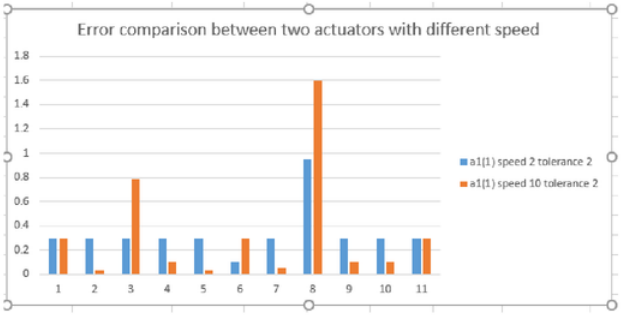


Figure 5.5: Comparing error percentage between two robot arm actuators with different speeds

Angles (in degrees)					
Letter	**Motor ID:1**	**Motor ID:2**	**Motor ID:3**	**Motor ID:4**	**Motor ID:5**
Q	N/A	N/A	N/A	N/A	N/A
W	N/A	N/A	N/A	N/A	N/A
E	89	255	255	218	218
R	84	255	255	220	220
T	80.7	255	255	224	224
Y	75.85	250	250	229	229
U	73	251.5	251.5	228.75	228.75
I	67	249	249	233	233
O	63	250	250	233	233
P	60	249	249	233	233
A	96	255	255	218	218
S	92.5	255	255	222	222
D	88	254	254	225.7	225.7
F	84	251.8	251.8	230	230
G	81	250	250	233	233
H	75.8	250	250	237	237
J	72.7	250	250	237	237
K	68.5	250	250	239.5	239.5
L	62	247.5	247.5	241	241
Z	96	255	255	222	222
X	92.6	252	252	230	230
C	89	250	250	235	235
V	85	246	246	240	240
B	80	247	247	240	240
N	76	245	245	243	243
M	72	244	244	247	247

Table 5.8: Keyboard calibration

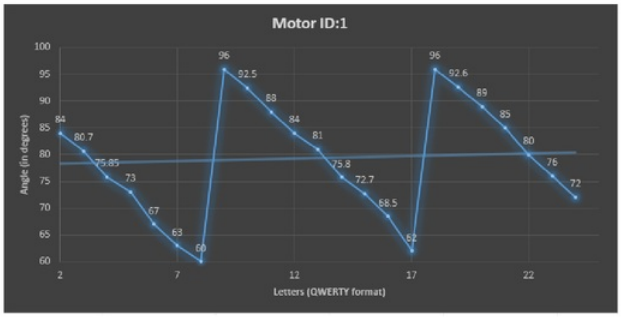


Figure 5.6: Motor ID:1

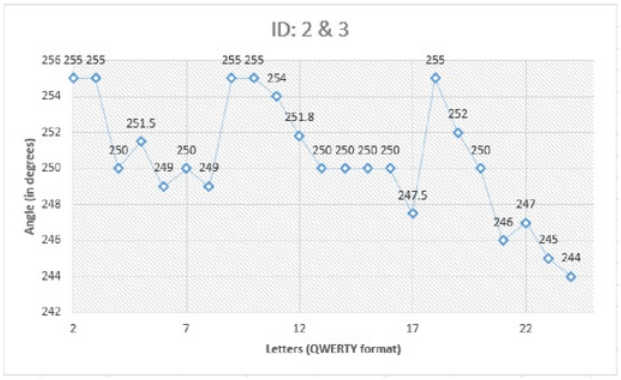


Figure 5.7: Motor ID:2 and 3

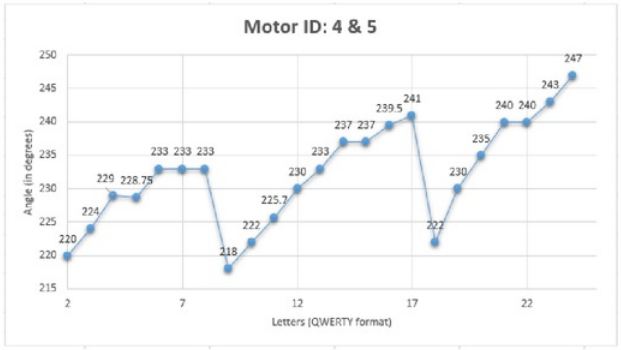


Figure 5.8: Motor ID: 4 and 5

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The primary aims of this thesis was the manipulation of a Dynamixel robotic arm to be able to play musical notes on a piano or an online keyboard synthesiser accurately with, and without human intervention, which can essentially be classed as a forward kinematics problem. This was done mainly with the help of Arduino, with a bit of LabView programming for auxiliary cross-checking calibration purposes.

The literature review conducted for this report covers a range of topics and concepts that were utilized, at least partially, over the course of the project. It was necessary to adjust the scope of the project based on the limitations of the hardware and/or software that was available, and slight modifications had to be made in order for the sake of relevance.

Based on the results obtained and its analysis, it was found that for increasing the speeds at which the actuators moved, the response times reduced. However, faster movements affected the precision of the joint movements and a compromise between speed and accuracy was needed to be reached.

All in all, it can be concluded that Arduino provides an effective platform for microcontroller-based robotic arm movement, while the actuators provide much better stability and accuracy to the entire robotic system than generic servos. Given the nature of the project, this provides the foundation and essential reference for more complex robotic mobility systems in the near future.

6.2 Future Work and Improvements

The first room for improvement lies on the project itself. This could be extended to include the inverse problem, which could cover the part where the robotic arm mimics the human arm in playing the piano, and the accuracy and precision of the robotic arm can be investigated. This can be done with the help of sensors either wirelessly via remote control or through wired connections.

To improve the steady-state error and lessen the overshoot which occurred frequenly at high speeds, PID control can be integrated into the Arduino code, or Simulink within MATLAB for smoother running of the actuators. Multiple robotic arms can be programmed to work together with coordinated delays and interrupt signals to reduce the time taken to finish the musical notes to a considerable extent.

This concept can be extended to perform more complicated automation tasks, such as vending machines or even a model of an automated retrieval system deployed in Macquarie University library.

Chapter 7

Abbreviations

PWM	Pulse Width Modulation
RF	Radio Frequency
RIA	Robotics Institute of America
SCARA	Selective Compliance Assembly Robot Arm
D-H	Denavit-Hartenberg
SPI	Serial Peripheral Interface
I2C /¿ Interface 2 Computer	

Chapter 8

Appendix

8.1 Arduino Code

Here is the full Arduino code used to move the robotic arm:

```
#include <SPI.h>              // including Serial Peripheral Interface h file
#include <ServoCds55.h>       // including Dynamixel AX-12a servo h file
ServoCds55 myservo;           // defining the dynamixel function

int servo1 = 1;   // Servo ID:1
int servo2 = 2;   // Servo ID:2
int servo3 = 3;   // Servo ID:3
int servo4 = 4;   // Servo ID:4
int servo5 = 5;   // Servo ID:5
int servo7 = 7;   // Servo ID:7
char inputCommand ;           // string to hold incoming data
boolean inputComplete = false;

void setup() {
  Serial.begin(1000000);    //baud rate:115200
  myservo.begin();
}

void loop() {
  serialEvent();
  if (inputComplete) {
```

```
    Serial.println("Press any key to move the arm to the desired position");
    Serial.print("Your command is: ");
    Serial.println(inputCommand);
    Serial.println("");
    controlServo(inputCommand);

    // Clearing the command:
    inputCommand = 0;
    inputComplete = false;
  }
}

void serialEvent() {
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    if (inChar == '\n') {
      inputComplete = true;
      break;
    }
    inputCommand += inChar;
  }
}

void controlServo(char val) {
  switch (val) {

    case 'J':
      myservo.setVelocity(35);
      myservo.write (servo1, 30);
      delay(1000);
      myservo.write (servo4, 218);
      myservo.write (servo5, 218);
      delay(1000);
      myservo.write (servo2, 250);
      myservo.write (servo3, 250);
      delay(1500);
      myservo.write(servo2, 225);
      myservo.write(servo3, 225);
      delay(1000);
      myservo.write(servo4, 225);
      myservo.write(servo5, 225);
      delay(1000);
      myservo.write(servo1, 60);
```

```
    delay(1000);
    break;

  case 'I':
    myservo.setVelocity(35);
    myservo.write(servo1, 34);
    delay(1000);
    myservo.write(servo4, 220);
    myservo.write(servo5, 220);
    delay(1000);
    myservo.write(servo2, 255);
    myservo.write(servo3, 255);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    break;

  case 'H':
    myservo.setVelocity(35);
    myservo.write(servo1, 38);
    delay(1000);
    myservo.write(servo4, 208);
    myservo.write(servo5, 208);
    delay(1000);
    myservo.write(servo2, 255);
    myservo.write(servo3, 255);
    delay(1500);
    myservo.write(servo2, 222);
    myservo.write(servo3, 222);
    delay(1500);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    break;
```

```
case 'G':
  myservo.setVelocity(35);
  myservo.write(servo1, 43);
  delay(1000);
  myservo.write(servo4, 208);
  myservo.write(servo5, 208);
  delay(1000);
  myservo.write(servo2, 255);
  myservo.write(servo3, 255);
  delay(1500);
  myservo.write(servo2, 225);
  myservo.write(servo3, 225);
  delay(1000);
  myservo.write(servo4, 225);
  myservo.write(servo5, 225);
  delay(1000);
  myservo.write(servo1, 60);
  delay(1000);
  break;

case 'F':
  myservo.setVelocity(35);
  myservo.write(servo1, 47);
  delay(1000);
  myservo.write(servo4, 218);
  myservo.write(servo5, 218);
  delay(1000);
  myservo.write(servo2, 250);
  myservo.write(servo3, 250);
  delay(1500);
  myservo.write(servo2, 225);
  myservo.write(servo3, 225);
  delay(1000);
  myservo.write(servo4, 225);
  myservo.write(servo5, 225);
  delay(1000);
  myservo.write(servo1, 60);
  delay(1000);
  break;

case 'E':
  myservo.setVelocity(35);
  myservo.write(servo1, 51);
```

```
    delay(1000);
    myservo.write(servo4, 218);
    myservo.write(servo5, 218);
    delay(1000);
    myservo.write(servo2, 255);
    myservo.write(servo3, 255);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    break;

  case 'D':
    myservo.setVelocity(35);
    myservo.write(servo1, 55);
    delay(1000);
    myservo.write(servo4, 218);
    myservo.write(servo5, 218);
    delay(1000);
    myservo.write(servo2, 250);
    myservo.write(servo3, 250);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    break;

  case 'K':
    myservo.setVelocity(35);
    myservo.write(servo1, 90);
    delay(1000);
    myservo.write(servo4, 218);
    myservo.write(servo5, 218);
```

```
    delay(1000);
    myservo.write(servo2, 250);
    myservo.write(servo3, 250);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    break;

  case 'L':
    myservo.setVelocity(35);
    myservo.write(servo1, 86);
    delay(1000);
    myservo.write(servo4, 220);
    myservo.write(servo5, 220);
    delay(1000);
    myservo.write(servo2, 255);
    myservo.write(servo3, 255);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    break;

  case 'M':
    myservo.setVelocity(35);
    myservo.write(servo1, 82);
    delay(1000);
    myservo.write(servo4, 208);
    myservo.write(servo5, 208);
    delay(1000);
    myservo.write(servo2, 255);
    myservo.write(servo3, 255);
```

```
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    break;

  case 'N':
    myservo.setVelocity(35);
    myservo.write(servo1, 77);
    delay(1000);
    myservo.write(servo4, 208);
    myservo.write(servo5, 208);
    delay(1000);
    myservo.write(servo2, 255);
    myservo.write(servo3, 255);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    break;

  case 'O':
    myservo.setVelocity(35);
    myservo.write(servo1, 73);
    delay(1000);
    myservo.write(servo4, 218);
    myservo.write(servo5, 218);
    delay(1000);
    myservo.write(servo2, 250);
    myservo.write(servo3, 250);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
```

```
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    break;

  case 'A':
    myservo.setVelocity(35);
    myservo.write(servo1, 69);
    delay(1000);
    myservo.write(servo4, 218);
    myservo.write(servo5, 218);
    delay(1000);
    myservo.write(servo2, 250);
    myservo.write(servo3, 250);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    break;

  case 'B':
    myservo.setVelocity(35);
    myservo.write(servo1, 65);
    delay(1000);
    myservo.write(servo4, 218);
    myservo.write(servo5, 218);
    delay(1000);
    myservo.write(servo2, 250);
    myservo.write(servo3, 250);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
```

```
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    break;

  case 'C':
    myservo.setVelocity(35);
    myservo.write(servo1, 60);
    delay(1000);
    myservo.write(servo4, 218);
    myservo.write(servo5, 218);
    delay(1000);
    myservo.write(servo2, 250);
    myservo.write(servo3, 250);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    break;

  case 'P':
    myservo.setVelocity(35);
    myservo.write(servo1, 55);
    delay(1000);
    myservo.write(servo4, 205);
    myservo.write(servo5, 205);
    delay(1000);
    myservo.write(servo2, 252);
    myservo.write(servo3, 252);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
```

```
      break;

    default:
      Serial.println("Press valid key to move the arm to the desired position");
  }
}
```

Here is the full Arduino code to manipulate the robotic arm to play the tune:

```
#include <SPI.h>              // including Serial Peripheral Interface h file
#include <ServoCds55.h>       // including Dynamixel AX-12a servo h file
ServoCds55 myservo;           // defining the dynamixel function

int servo1 = 1;   // Servo ID:1
int servo2 = 2;   // Servo ID:2
int servo3 = 3;   // Servo ID:3
int servo4 = 4;   // Servo ID:4
int servo5 = 5;   // Servo ID:5
int servo7 = 7;   // Servo ID:7
char inputCommand ;           // string to hold incoming data
boolean inputComplete = false;

void setup() {
  Serial.begin(1000000);    //baud rate:115200
  myservo.begin();
}

void loop() {
  serialEvent();
  if (inputComplete) {
    Serial.println("Press any key to move the arm to the desired position");
    Serial.print("Your command is: ");
    Serial.println(inputCommand);
    Serial.println("");
    controlServo(inputCommand);

    // Clearing the command:
    inputCommand = 0;
    inputComplete = false;
  }
```

```
}

void serialEvent() {
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    if (inChar == '\n') {
      inputComplete = true;
      break;
    }
    inputCommand += inChar;
  }
}

void controlServo(char val) {
  switch (val) {

    case 'S':
      myservo.setVelocity(35);
      //C1
      myservo.write(servo1, 60);
      delay(1000);
      myservo.write(servo4, 218);
      myservo.write(servo5, 218);
      delay(1000);
      myservo.write(servo2, 250);
      myservo.write(servo3, 250);
      delay(1500);
      myservo.write(servo2, 225);
      myservo.write(servo3, 225);
      delay(1000);
      myservo.write(servo4, 225);
      myservo.write(servo5, 225);
      delay(1000);
      myservo.write(servo1, 60);
      delay(1000);
      //G1
      myservo.write(servo1, 43);
      delay(1000);
      myservo.write(servo4, 208);
      myservo.write(servo5, 208);
      delay(1000);
      myservo.write(servo2, 255);
      myservo.write(servo3, 255);
```

```
 delay(1500);
 myservo.write(servo2, 225);
 myservo.write(servo3, 225);
 delay(1000);
 myservo.write(servo4, 225);
 myservo.write(servo5, 225);
 delay(1000);
 myservo.write(servo1, 60);
 delay(1000);
 //G1
 myservo.write(servo1, 43);
 delay(1000);
 myservo.write(servo4, 208);
 myservo.write(servo5, 208);
 delay(1000);
 myservo.write(servo2, 255);
 myservo.write(servo3, 255);
 delay(1500);
 myservo.write(servo2, 225);
 myservo.write(servo3, 225);
 delay(1000);
 myservo.write(servo4, 225);
 myservo.write(servo5, 225);
 delay(1000);
 myservo.write(servo1, 60);
 delay(1000);
//E#
 myservo.write(servo1, 55);
 delay(1000);
 myservo.write(servo4, 205);
 myservo.write(servo5, 205);
 delay(1000);
 myservo.write(servo2, 252);
 myservo.write(servo3, 252);
 delay(1500);
 myservo.write(servo2, 225);
 myservo.write(servo3, 225);
 delay(1000);
 myservo.write(servo4, 225);
 myservo.write(servo5, 225);
 delay(1000);
 myservo.write(servo1, 60);
 delay(1000);
```

```
//D1
myservo.write(servo1, 55);
delay(1000);
myservo.write(servo4, 218);
myservo.write(servo5, 218);
delay(1000);
myservo.write(servo2, 250);
myservo.write(servo3, 250);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//D1
myservo.write(servo1, 55);
delay(1000);
myservo.write(servo4, 218);
myservo.write(servo5, 218);
delay(1000);
myservo.write(servo2, 250);
myservo.write(servo3, 250);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//D1
myservo.write(servo1, 55);
delay(1000);
myservo.write(servo4, 218);
myservo.write(servo5, 218);
delay(1000);
myservo.write(servo2, 250);
myservo.write(servo3, 250);
delay(1500);
```

```
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//D1
myservo.write(servo1, 55);
delay(1000);
myservo.write(servo4, 218);
myservo.write(servo5, 218);
delay(1000);
myservo.write(servo2, 250);
myservo.write(servo3, 250);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//E#
myservo.write(servo1, 55);
delay(1000);
myservo.write(servo4, 205);
myservo.write(servo5, 205);
delay(1000);
myservo.write(servo2, 252);
myservo.write(servo3, 252);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//C1
```

```
myservo.write(servo1, 60);
delay(1000);
myservo.write(servo4, 218);
myservo.write(servo5, 218);
delay(1000);
myservo.write(servo2, 250);
myservo.write(servo3, 250);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//G1
myservo.write(servo1, 43);
delay(1000);
myservo.write(servo4, 208);
myservo.write(servo5, 208);
delay(1000);
myservo.write(servo2, 255);
myservo.write(servo3, 255);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//G1
myservo.write(servo1, 43);
delay(1000);
myservo.write(servo4, 208);
myservo.write(servo5, 208);
delay(1000);
myservo.write(servo2, 255);
myservo.write(servo3, 255);
delay(1500);
myservo.write(servo2, 225);
```

```
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//E#
myservo.write(servo1, 55);
delay(1000);
myservo.write(servo4, 205);
myservo.write(servo5, 205);
delay(1000);
myservo.write(servo2, 252);
myservo.write(servo3, 252);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//D1
myservo.write(servo1, 55);
delay(1000);
myservo.write(servo4, 218);
myservo.write(servo5, 218);
delay(1000);
myservo.write(servo2, 250);
myservo.write(servo3, 250);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//D1
myservo.write(servo1, 55);
```

```
delay(1000);
myservo.write(servo4, 218);
myservo.write(servo5, 218);
delay(1000);
myservo.write(servo2, 250);
myservo.write(servo3, 250);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//D1
myservo.write(servo1, 55);
delay(1000);
myservo.write(servo4, 218);
myservo.write(servo5, 218);
delay(1000);
myservo.write(servo2, 250);
myservo.write(servo3, 250);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//D1
myservo.write(servo1, 55);
delay(1000);
myservo.write(servo4, 218);
myservo.write(servo5, 218);
delay(1000);
myservo.write(servo2, 250);
myservo.write(servo3, 250);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
```

```
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//E#
myservo.write(servo1, 55);
delay(1000);
myservo.write(servo4, 205);
myservo.write(servo5, 205);
delay(1000);
myservo.write(servo2, 252);
myservo.write(servo3, 252);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//C1
myservo.write(servo1, 60);
delay(1000);
myservo.write(servo4, 218);
myservo.write(servo5, 218);
delay(1000);
myservo.write(servo2, 250);
myservo.write(servo3, 250);
delay(1500);
myservo.write(servo2, 225);
myservo.write(servo3, 225);
delay(1000);
myservo.write(servo4, 225);
myservo.write(servo5, 225);
delay(1000);
myservo.write(servo1, 60);
delay(1000);
//G1
myservo.write(servo1, 43);
delay(1000);
```

```
    myservo.write(servo4, 208);
    myservo.write(servo5, 208);
    delay(1000);
    myservo.write(servo2, 255);
    myservo.write(servo3, 255);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
    //G1
    myservo.write(servo1, 43);
    delay(1000);
    myservo.write(servo4, 208);
    myservo.write(servo5, 208);
    delay(1000);
    myservo.write(servo2, 255);
    myservo.write(servo3, 255);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
    myservo.write(servo4, 225);
    myservo.write(servo5, 225);
    delay(1000);
    myservo.write(servo1, 60);
    delay(1000);
   //E#
    myservo.write(servo1, 55);
    delay(1000);
    myservo.write(servo4, 205);
    myservo.write(servo5, 205);
    delay(1000);
    myservo.write(servo2, 252);
    myservo.write(servo3, 252);
    delay(1500);
    myservo.write(servo2, 225);
    myservo.write(servo3, 225);
    delay(1000);
```

```
      myservo.write(servo4, 225);
      myservo.write(servo5, 225);
      delay(1000);
      myservo.write(servo1, 60);
      delay(1000);
      //D1
      myservo.write(servo1, 55);
      delay(1000);
      myservo.write(servo4, 218);
      myservo.write(servo5, 218);
      delay(1000);
      myservo.write(servo2, 250);
      myservo.write(servo3, 250);
      delay(1500);
      myservo.write(servo2, 225);
      myservo.write(servo3, 225);
      delay(1000);
      myservo.write(servo4, 225);
      myservo.write(servo5, 225);
      delay(1000);
      myservo.write(servo1, 60);
      delay(1000);
      break;

      default:
      Serial.println("Press valid key to move the arm to the desired position");
  }
}
```

Chapter 9

Bibliography

1. C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. Dimarogonas and D. Kragic, "Dual arm manipulation—A survey", Robotics and Autonomous Systems, vol. 60, no. 10, pp. 1340-1353, 2012.
2. M. Yusoff, R. Samin and B. Ibrahim, "Wireless Mobile Robotic Arm", Procedia Engineering, vol. 41, pp. 1072-1078, 2012.
3. C. Su, S. Rakheja and H. Liu, in Intelligent Robotics and Applications, Montreal, QC, Canada, 2012.
4. Y. Lu, Z. Dai, N. Ye and P. Wang, "Kinematics/statics analysis of a novel serial-parallel robotic arm with hand", Journal of Mechanical Science and Technology, vol. 29, no. 10, pp. 4407-4416, 2015.
5. H. Al-Qahtani, A. Mohammed and M. Sunar, "Dynamics and Control of a Robotic Arm Having Four Links", Arabian Journal for Science and Engineering, vol. 42, no. 5, pp. 1841-1852, 2016.
6. D. Lan, "The Application of Intelligent Industrial Robotic Control System Based on PLC in Mechanical Automation", Advanced Materials Research, vol. 738, pp. 272-275, 2013.
7. Keerti S Nair and B. Prasannakumari, "Kinematic Modelling and Analysis of a 5 Axis Articulated Robot Arm Model VRT-502", International Journal of Engineering Research and, vol. 4, no. 07, 2015.
8. "LabView Interface with Arduino Robotic ARM", International Journal of Science and Research (IJSR), vol. 4, no. 11, pp. 2423-2426, 2015.
9. A. Sharma, K. Lewis, V. Ansari and V. Noronha, "Design And Implementation Of Anthropomorphic Robotic Arm", International Journal of Engineering Research and Applications, vol. 4, no. 1, pp. 73-79, 2014.

10. Y. Liu and H. Yu, "Visual Servo Robotic Arm Control System Based on LabVIEW", Applied Mechanics and Materials, vol. 602-605, pp. 844-847, 2014.

11. D. Xu, C. Calderon, J. Gan, H. Hu and M. Tan, "An Analysis of the Inverse Kinematics for a 5-DOF Manipulator", International Journal of Automation and Computing, vol. 2, pp. 114-124, 2005.

12. K. Dharmendra, K. Ramachandra and S. Sartaj, "Kinematic Modeling and Hardware Development of 5-DoF Robot Manipulator", Applied Mechanics and Materials, vol. 612, pp. 51-58, 2014.