ROBOTIC ARM CONTROL THROUGH HUMAN ARM MOVEMENT USING ACCELEROMETERS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS & INSTRUMENTATION ENGINEERING

by

ASHUTOSH PATTNAIK, 109EI0297

RAJIV RANJAN, 109EI0339

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING,

NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA

ROBOTIC ARM CONTROL THROUGH HUMAN ARM MOVEMENT USING ACCELEROMETERS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS & INSTRUMENTATION ENGINEERING

by

ASHUTOSH PATTNAIK, 109EI0297

RAJIV RANJAN, 109EI0339

UNDER THE SUPERVISION OF

PROF. SANTOS KUMAR DAS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING,

NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA

NATIONAL INSTITUTE OF TECHNOLOGY,

ROURKELA

CERTIFICATE

This is to certify that the project report titled " **Robotic Arm Control Through Human Arm Movement using Accelerometers** " submitted by **Ashutosh Pattnaik** (Roll No: 109EI0297) and **Rajiv Ranjan** (Roll No: 109EI0339) in the partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Electronics and Instrumentation Engineering during session 2009-2013 at National Institute of Technology, Rourkela and is an authentic work carried out by them under my supervision and guidance.

Date:

Prof. Santos Kumar Das

Department of Electronics & Communication Engineering,

National Institute of Technology, Rourkela

ACKNOWLEDGEMENT

We would like to express my deep sense of gratitude and respect to my supervisor Prof. Santos Kumar Das for his excellent guidance, suggestions and support. I consider myself extremely lucky to be able to work under the guidance of such a dynamic personality. We would like to render heartiest thanks to my friend who’s ever helping nature and suggestion has helped us to complete this present work.

We would like to thank all faculty members and staff of the Department of Electronics and Communication Engineering, N.I.T. Rourkela for their extreme help throughout course. An assemblage of this nature could never have been attempted without reference to and inspiration from the works of others whose details are mentioned in the reference section. We acknowledge our indebtedness to all of them.

Ashutosh Pattnaik (109EI0297)

Rajiv Ranjan (109EI0339)

ABSTRACT

In today's world there is an increasing need to create artificial arms for different inhuman situations where human interaction is difficult or impossible. They may involve taking readings from an active volcano to diffusing a bomb. Here we propose to build a robotic arm controlled by natural human arm movements whose data is acquired through the use of accelerometers. For proper control mechanism and to reduce the amount of noise coming in from the sensors, proper averaging algorithm is used for smoothening the output of the accelerometer. The development of this arm is based on ATmega32 and ATmega640 platform along with a personal computer for signal processing, which will all be interfaced with each other using serial communication. Finally, this prototype of the arm may be expected to overcome the problem such as placing or picking hazardous objects or non-hazardous objects that are far away from the user.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER - 1

INTRODUCTION

1.1 Introduction

Nowadays, robots are increasingly being integrated into working tasks to replace humans especially to perform the repetitive task. In general, robotics can be divided into two areas, industrial and service robotics. International Federation of Robotics (IFR) defines a service robot as a robot which operates semi- or fully autonomously to perform services useful to the well-being of humans and equipment, excluding manufacturing operations. These robots are currently used in many fields of applications including office, military tasks, hospital operations, dangerous environment and agriculture. Besides, it might be difficult or dangerous for humans to do some specific tasks like picking up explosive chemicals, defusing bombs or in worst case scenario to pick and place the bomb somewhere for containment and for repeated pick and place action in industries. Therefore a robot can be replaced human to do work.

1.2 Robotic arm definition

A robotic arm is a robot manipulator, usually programmable, with similar functions to a human arm. The links of such a manipulator are connected by joints allowing either rotational motion (such as in an articulated robot) or translational (linear) displacement. The links of the manipulator can be considered to form a kinematic chain. The business end of the kinematic chain of the manipulator is called the end effectors and it is analogous to the human hand. The end effectors can be designed to perform any desired task such as welding, gripping, spinning etc., depending on the application. The robot arms can be autonomous or controlled manually and can be used to perform a variety of tasks with great accuracy. The robotic arm can be fixed or mobile (i.e. wheeled) and can be designed for industrial or home applications. [1][2]

This report deals with a robotic arm whose objective is to imitate the movements of a human arm using accelerometers as sensors for the data acquisition of the natural arm movements. This method of control allows greater flexibility in controlling the robotic arm rather than using a controller where each actuator is controlled separately. The processing unit takes care of each actuator's control signal according to the inputs from accelerometer, in order to replicate the movements of the human arm. Figure 1 shows the block diagram representation of the system to be designed and implemented.

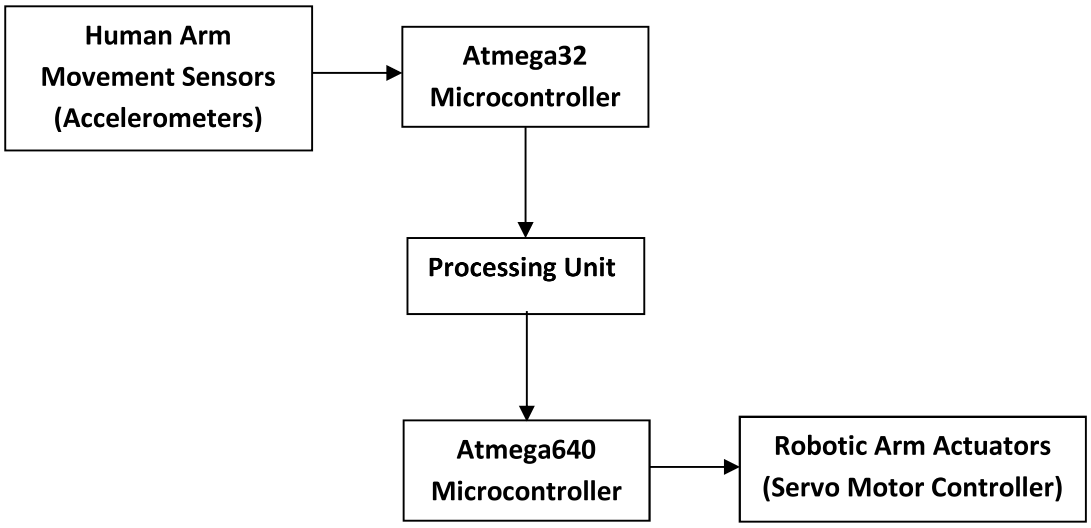

Human Arm Movement Sensors (Accelerometers)
Atmega32 Microcontroller
Processing Unit
Atmega640 Microcontroller
Robotic Arm Actuators (Servo Motor Controller)

Fig1. Block Diagram Representation of the Proposed Robotic Arm System

1.3 Literature Review

There are various ways in which a robotic arm may be controlled. In the past there have been many researchers working to control robotic arm through computer terminals, Joysticks, even interfacing them with the internet so they can be controlled from anywhere in the world. [1][2] Usually most of the robotic arms are controlled by a central controller which makes uses of values taken in from the terminal that are entered by the user at the terminal to move the arm to a particular coordinates in space. This makes the control very difficult as the control values of the motors are very difficult to predict to achieve a particular movement. This is easily achieved by our project.

This Project represents a simple accelerometer controlled robotic arm using Atmega32/640 powered embedded system as the core of this robot and also a Computer to interface the robot with the sensors. The robot does not require training because the robotic arm is fully controlled by the user. This interfacing is done using wired communication but it can easily be switched to wireless with ease.

1.4 Project Overview

In this Project, the hardware and software function are combined to make the system reliable. The ATmega32 and ATmega640 will be interfacing the robot with the sensor i.e. 3 axis accelerometer and the actuators i.e. servo motors which will control the movement of the robot respectively.

The chapter that follows describe the hardware (Chapter 2), which is followed by the description of the software being used (Chapter 3) Chapter 4 describes the implementation of the project and Chapter 5 concludes the discussion followed by the future scope of the project.

CHAPTER – 2

HARDWARE DESIGN AND DESCRIPTION

This chapter describes the hardware that is being used in the project.

2.1 Hardware Requirements

1. Accelerometers (Sensor)
2. Servo Motors (Actuator)
3. ATmega32 (Data Acquisition)
4. ATmega640 (Controller)
5. 16x2 LCD Module (Display)

2.2 Accelerometer

Here we use an accelerometer designed by Freescale Semiconductors. The MMA7361L is a low power, low profile capacitive micro-machined accelerometer featuring signal conditioning, a 1-pole low pass filter, temperature compensation, self-test, 0g-Detect which detects linear free fall, and g-Select which allows for the selection between 2 sensitivities. [11]

Salient Features of MMA7361L

- 3mm x 5mm x 1.0mm LGA-14 Package
- Low Current Consumption: 400µA
- Sleep Mode: 3µA
- Low Voltage Operation: 2.2 V – 3.6 V
- High Sensitivity (800 mV/g @ 1.5g)
- Selectable Sensitivity (±1.5g, ±6g)
- Fast Turn On Time (0.5ms Enable Response Time)

- Self-Test for Free-fall Detect Diagnosis
- 0g-Detect for free fall Protection
- Signal Conditioning with Low Pass Filter
- Robust Design, High Shocks Survivability
- RoHS Compliant
- Environmentally Preferred Product
- Low Cost

Typical applications of this include

- Robotics: Motion Sensing
- 3D Gaming: Tilt and Motion Sensing, Event Recorder
- HDD MP3 Player: free-fall Detection
- Laptop PC: free-fall Detection, Anti-Theft
- Cell Phone: Image Stability, Text Scroll, Motion Dialing, E-Compass
- Pedometer: Motion Sensing
- PDA: Text Scroll
- Navigation and Dead Reckoning: E-Compass Tilt Compensation

The device consists of a surface micro-machined capacitive sensing cell (g-cell) and a signal conditioning ASIC contained in a single package. The sensing element is sealed hermetically at the wafer level using a bulk micro-machined cap wafer. The g-cell is a mechanical structure formed from semiconductor materials (poly-silicon) using semiconductor processes (masking and etching). It can be modeled as a set of beams attached to a movable central mass that move between fixed beams. The movable beams can be deflected from their rest position by subjecting the system to acceleration. As the beams attached to the central mass move, the distance from them to the fixed beams on one side will increase by the same amount that the distance to the fixed beams on the other side decreases. The change in distance is a measure of acceleration. The g-cell beams form two back-to-back capacitors. As the center beam moves with acceleration, the distance between the beams changes and each capacitor's value will change, ($C = A\varepsilon/D$). Where A is the area of the beam, ε is the dielectric constant, and D is the distance between the beams.

The ASIC uses switched capacitor techniques to measure the g-cell capacitors and extract the acceleration data from the difference between the two capacitors. The ASIC also signal conditions and filters (switched capacitor) the signal, providing a high level output voltage that is ratio metric and proportional to acceleration. Figure 2 shows the simplified functional block diagram of an accelerometer. [11]

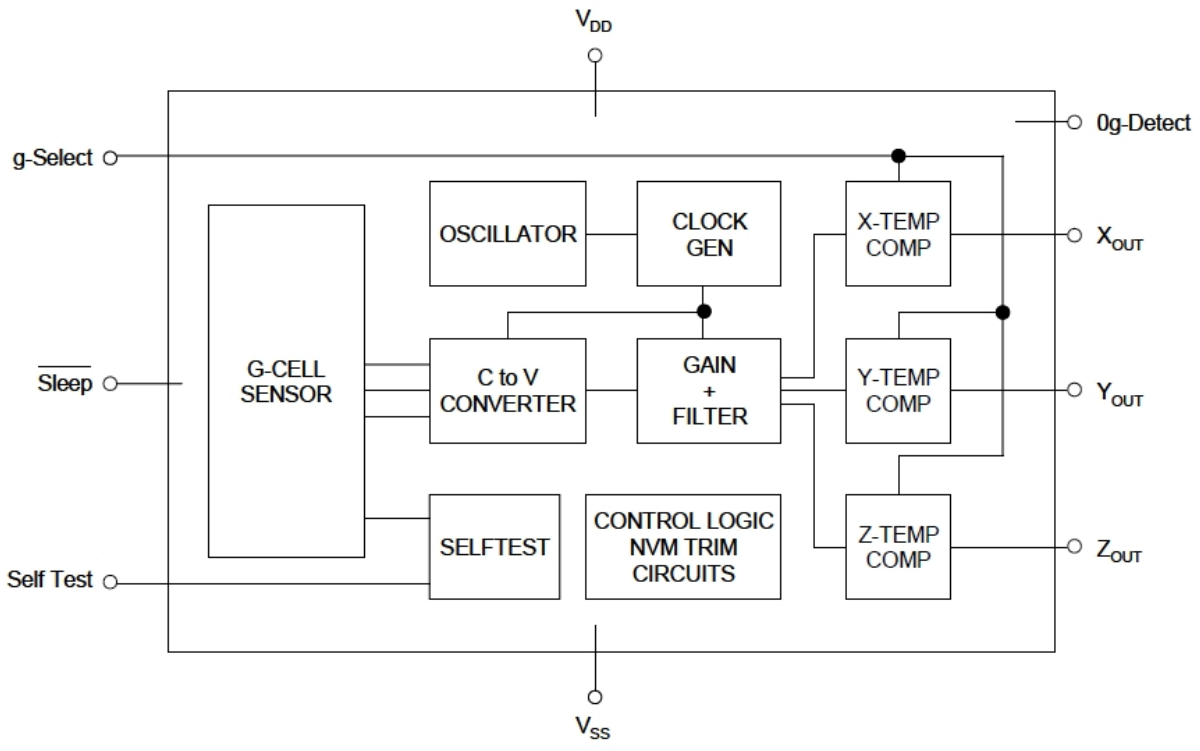


Fig2. Simplified Accelerometer Functional Block Diagram

2.3 Servo Motors

Servo motors are a type of electromechanical actuators that do not rotate continuously like DC/AC or stepper motors; rather, they are used to position and hold some object. They are used where continuous rotation is not required so they are not used to drive wheels (unless a servo is modified). In contrast they are used where something is needed to move to particular position and then stopped and hold there. Most common use is to position the rudder of aircrafts and boats etc. The servo can be commanded to rotate to a particular angle (say 30) and then hold its position there. Servos also employ a feedback mechanism, so it can sense an error in its positioning and correct it. This is called servomechanism. Say if you ask servo to go and lock itself to 30 degrees and then try to rotate it with your hand, the servo will try hard and its best to overcome the force and keep servo locked in its specified angle. Controlling a servo is easy by using a microcontroller, no external driver like h-bridge etc. are required. Just a control signal is needed to be feed to the servo to position it in any specified angle. The frequency of the control signal is 50 Hz (i.e. the period is 20ms) and the width of positive pulse controls the angle. We can use the AVR microcontrollers PWM feature to control servo motors. In this way the PWM with automatically generate signals to lock servo and the CPU is free to do other tasks.

Here we use AVR Timer1 Module which is a 16bit timer and has two PWM channels (A and B). The CPU frequency is 8 MHz; this frequency is the maximum frequency that most AVRs are capable of running. And so it is used in most development board like Low Cost AVR Development Boards. We chose the pre-scaler as 64. So the timer will get 8MHz/64 = 125 kHz (8uS period). We setup Timer Mode as Mode 14. [3]

Timer Mode 14 features

- FAST PWM Mode
- TOP Value = ICR1

So the timer will count from 0 to ICR1 (TOP Value).

2.4 ATmega32 Microcontroller

The ATmega32 is a part of the AVR family. It uses on-chip flash memory for program storage, as opposed to one-time programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time. Flash, EEPROM, and SRAM are all integrated onto a single chip, removing the need for external memory in most applications [8]. Some of the Salient Features and Specifications of ATmega32 are:

- High-performance, Low-power 8-bit Microcontroller
- Advanced RISC Architecture
- 131 Powerful Instructions
- 32 × 8 General Purpose Working Registers
- 32Kbytes of In-System Self-programmable Flash program memory
- 1024Bytes EEPROM
- Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
- Data retention: 20 years at 85°C/100 years at 25°C(1)
- Two 8-bit Timer/Counters with Separate Pre-scalers and Compare Modes
- One 16-bit Timer/Counter with Separate Pre-scaler, Compare Mode, and Capture Mode
- Four PWM Channels

- 8-channel, 10-bit ADC
- 8 Single-ended Channels
- 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
- Programmable Serial USART
- External and Internal Interrupt Sources
- 32 Programmable I/O Lines
- Operating Voltages: 4.5V - 5.5V for ATmega32
- Power Consumption at 1MHz, 3V, 25°C

The Pin configuration of ATmega is shown in Figure 3.

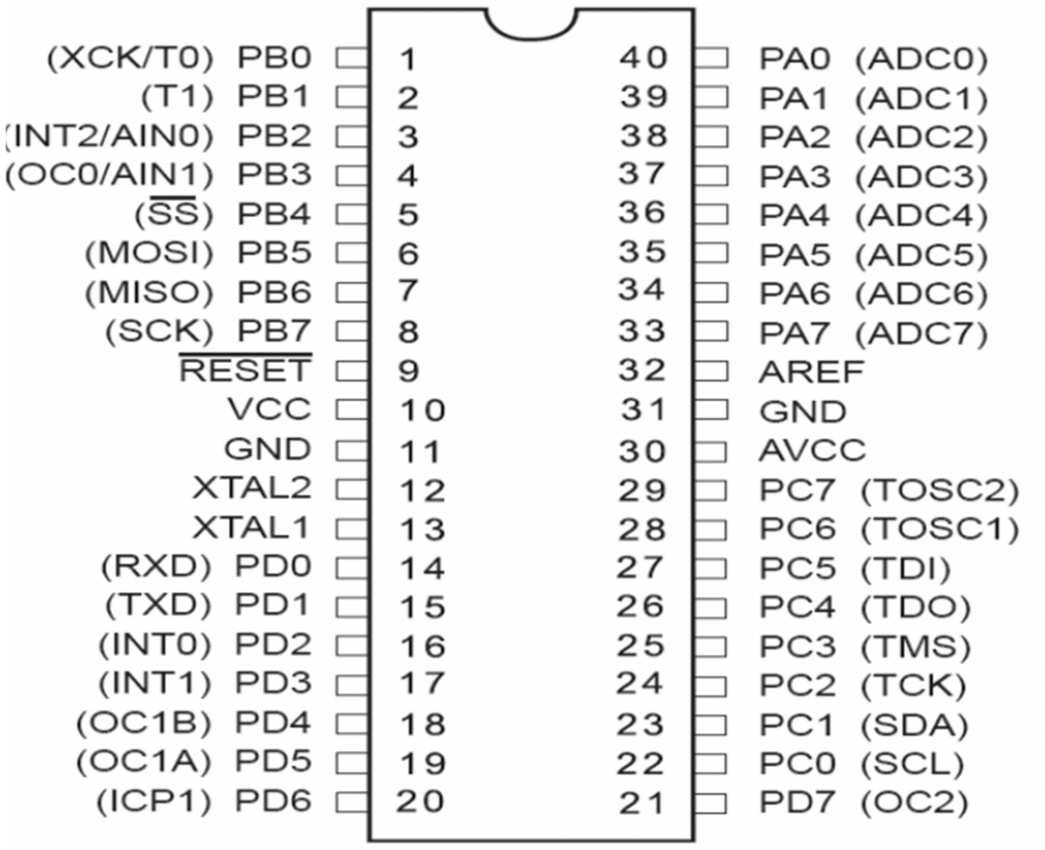


Fig3. Pin Configuration of ATmega32

The main features of ATmega32 that we are using in this project are

1. USART Interface
2. Analog to Digital Converter (ADC)

2.4.1 USART Interface

ATmega32 Development board also has a dedicated hardware for serial communication this part is called the USART - Universal Synchronous Asynchronous Receiver Transmitter. Here we just have to supply the data (in this case the ADC output) need to transmit and it will do the rest. The advantage of hardware USART is that we just need to write the data to one of the registers of USART and we are free to do other things while USART is transmitting the byte.

Also the USART automatically senses the start of transmission of RX line and then inputs the whole byte and when it has the byte it informs through an interrupt(CPU) to read that data from one of its registers. [7]

We are using USART in our project to transfer ADC output of the ATmega32 microcontroller to a Computer. The Computer then receives the byte and processes the data accordingly.

2.4.2 Analog to Digital Converter (ADC)

Most of the physical quantities around us are continuous. By continuous we mean that the quantity can take any value between two extreme. If an electrical quantity is made to vary directly in proportion to the physical quantity (that needs to be measured) then what we have is an analog signal. Now we have we have brought a physical quantity into electrical domain. The electrical quantity in most case is voltage. To bring this quantity into digital domain we have to convert this into digital form. For this an ADC or analog to digital converter is used. ATmega32 has an ADC on chip. An ADC converts an input voltage into a number. An ADC has a resolution of 10bits. A 10 Bit ADC has a range of 0-1023. (2^10=1024) The ADC also has a Reference voltage (ARef). When input voltage is GND the output is 0 and when input voltage is equal to ARef the output is 1023. So the input range is 0-ARef and digital output is 0-1023.

We are using ADC in our project to acquire data from the 3-axis accelerometer which provides us with an analog voltage signal to convert this signal into digital domain for further processing. This needs to be done because the ATmega32 microcontroller can only work in digital domain. [6]

2.5 ATmega640 Microcontroller

The ATmega640 is a part of the AVR family. The ATmega640/1280/1281/2560/2561 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega640/1280/1281/2560/2561 achieves throughputs approaching 1 MIPS per MHz allowing the system designed to be optimized for power consumption versus processing speed. Some of the Salient Features and Specifications of ATmega640 are [9]:

- High Performance, Low Power Microcontroller
- Advanced RISC Architecture
- 135 Powerful Instructions – Most Single Clock Cycle Execution
- 32 x 8 General Purpose Working Registers

- Up to 16 MIPS Throughput at 16 MHz
- On-Chip 2-cycle Multiplier
- Non-volatile Program and Data Memories
- 64K/128K/256K Bytes of In-System Self-Programmable Flash
- Endurance: 10,000 Write/Erase Cycles
- Optional Boot Code Section with Independent Lock Bits
- 4K Bytes EEPROM
- Endurance: 100,000 Write/Erase Cycles
- 8K Bytes Internal SRAM
- Up to 64K Bytes Optional External Memory Space
- JTAG (IEEE std. 1149.1 compliant) Interface
- Extensive On-chip Debug Support
- Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Two 8-bit Timer/Counters with Separate Pre-scaler and Compare Mode
- Four 16-bit Timer/Counter with Separate Pre-scaler, Compare- and Capture Mode
- Real Time Counter with Separate Oscillator
- Four 8-bit PWM Channels
- Six/Twelve PWM Channels with Programmable Resolution from 2 to 16 Bits (ATmega1281/2561, ATmega640/1280/2560)
- 8/16-channel, 10-bit ADC
- Two/Four Programmable Serial USART (ATmega1281/2561,ATmega640/1280/2560)
- Master/Slave SPI Serial Interface
- Byte Oriented 2-wire Serial Interface

- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator
- Interrupt and Wake-up on Pin Change

The Pin configuration of ATmega640 is shown in Figure 4.

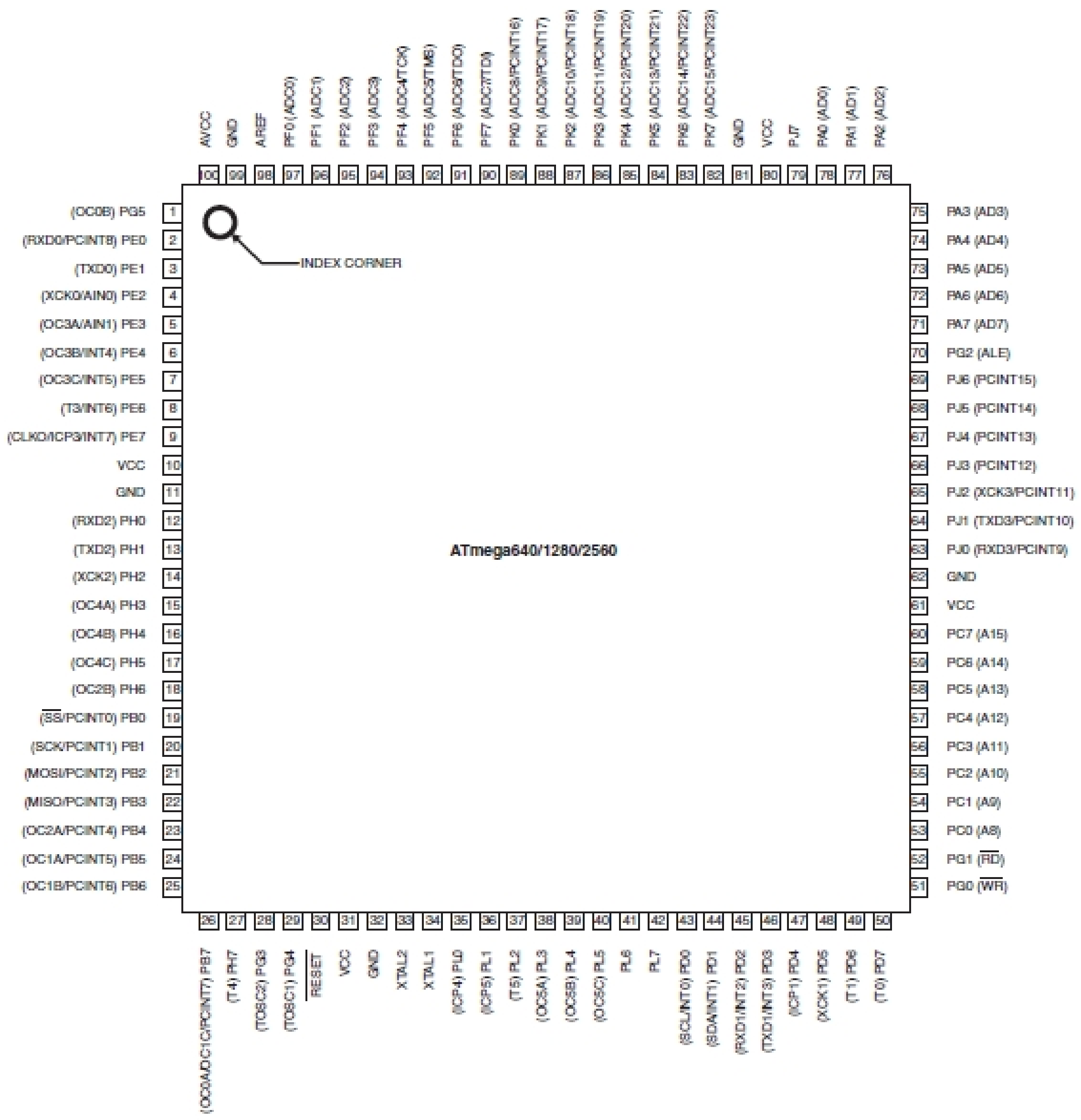


Fig4. Pin Configuration of ATmega640

The main features of ATmega32 that we are using in this project are

1. Timer
2. USART Interface

2.5.1 Timer

A timer in simplest term is a register. Timers generally have a resolution of 8 or 16 Bits. So an 8 bit timer is 8 Bits wide so capable of holding value within 0-255. But this register has a property that its value increases/decreases automatically at a predefined rate (supplied by user). This is the timer clock. And this operation does not need CPU's intervention.

ATmega640 has six different timers of which the simplest is TIMER0. Its resolution is 8 bits i.e. it can count from 0 to 255. The Pre-scaler is a mechanism for generating clock for timer by the CPU clock. As we know that CPU has a clock source such as an external crystal of internal oscillator. Normally these have the frequency like 1 MHz, 8 MHz, 12 MHz or 16MHz (MAX). The Pre-scaler is used to divide this clock frequency and produce a clock for TIMER. The Pre-scaler can be used to get the following clock for timer; No Clock (Timer Stop), No Pre-scaling (Clock = FCPU), FCPU/8, FCPU/64, FCPU/256, FCPU/1024. Timers can also be externally clocked

Timer is being used in our project to generate the PWM signal of required pulse width in order to control the servo motor's position. By varying the value of the registers of the timer we can change the pulse width of the control signal thus controlling the robotic arm itself.

2.5.2 USART Interface

ATmega640 Development board consists of dedicated hardware for serial communication this part is called the USART - Universal Synchronous Asynchronous Receiver Transmitter. Here we just have to supply the data (in this case the ADC output) need to transmit and it will do the rest. We are using USART with ATmega640 in our project to transfer control values for the servo motors from the Computer. The ATmega640 then receives the bytes which are then fed to respective timers to generate the required PWM signals for individual servo motors.

2.6 16x2 LCD Module

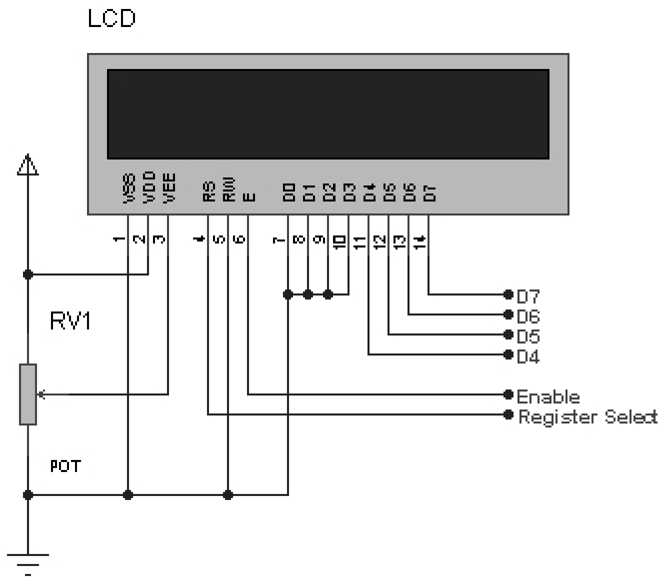


Fig5. LCD Pin Configuration and Connections

LCD Modules can present textual information to user. They come in various types. The most popular one that we use here can display 2 lines of 16 characters. [5]

LCD on ATmega32 Development is being used to display the value of the ADC output which takes the accelerometer as the input and the LCD on ATmega640 is used to display the value of the corresponding OCR Register value which provides the changes in PWM signal to control the robotic arm.

2.7 Hardware Design

The schematic diagram for the data acquisition from the sensor via ATmega32 is as shown in Figure 6, followed by the schematic of the servo motor control via the ATmega640 in Figure 7. As can be seen from Figure 6, the two accelerometer's individual X, Y and Z channel inputs are given to the ADC0-ADC5 pins which are multiplexed with PORTA. The common Vcc and GND power the accelerometer as well as the ATmega32. A serial transmission is interfaced between the ATmega640 and the digital computer via the Rx-Tx of the ATmega32 which are PIN14 & PIN15.

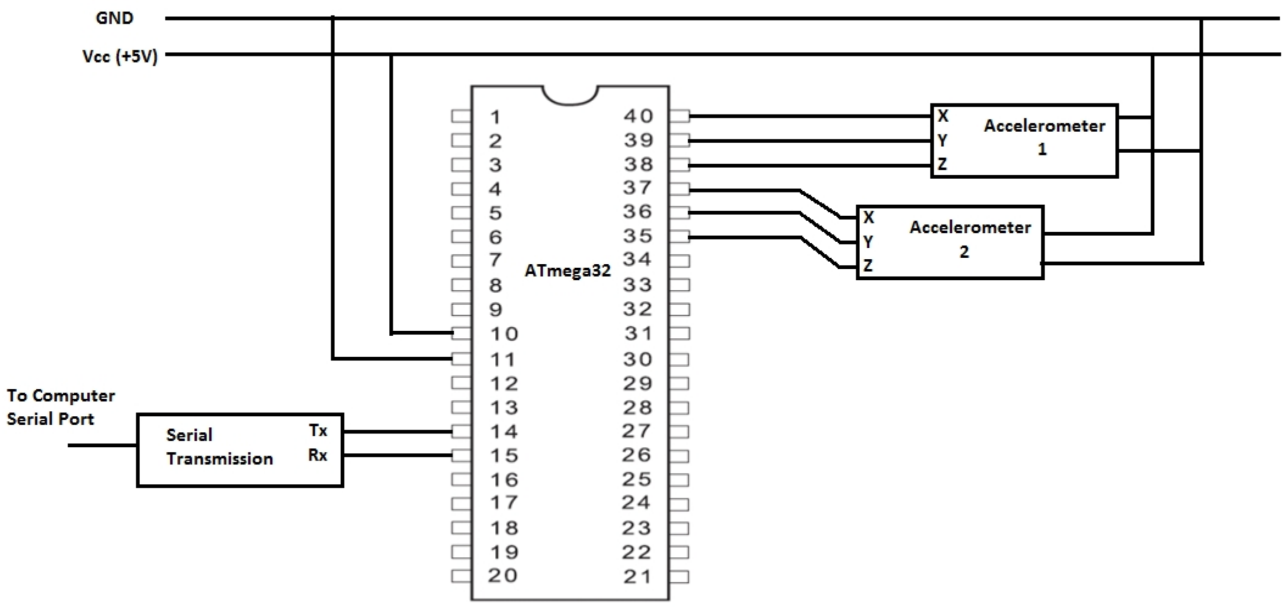


Fig6. Circuit diagram for the data acquisition from the sensor via ATmega32

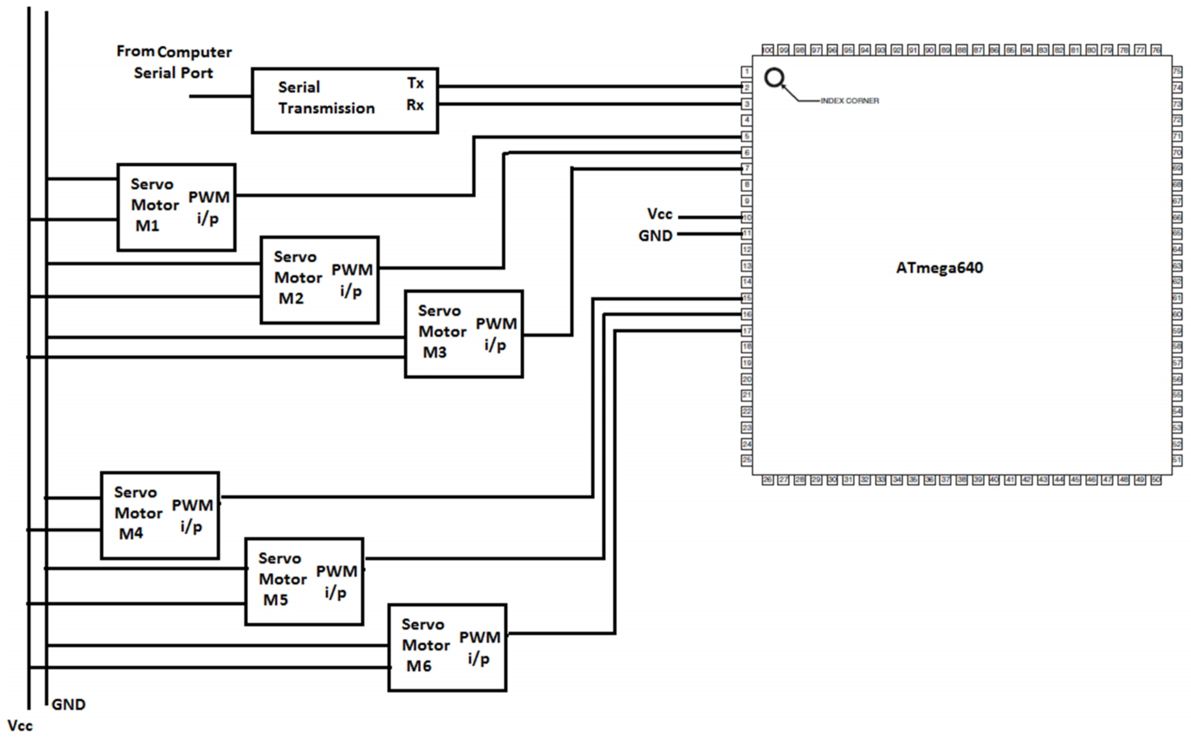


Fig7. Circuit diagram for the control of servo motors via ATmega640

In Figure 7, it is seen that the servo motors PWM input signal is coming from the ATmega640 timer outputs, viz. OCR3A, OCR3B, OCR3C, OCR4A, OCR4B which control the motors M1, M2, M3, M4 & M5 respectively. The servo motors also share the same power supply as the ATmega640. A serial transmission is interfaced between the ATmega640 and the digital computer via the Rx-Tx of the ATmega640 which are PIN1 & PIN2.

The following chapter describes the software that is being used in the project.

CHAPTER – 3

SOFTWARE DESIGN AND DESCRIPTION

This chapter describes the software that is being used in the project.

3.1 Software Requirements

1. AVR Studio 4.18
2. WinAVR – Build 20090313
3. SinaProg 2.0
4. MATLAB v7.6 (R2012a)

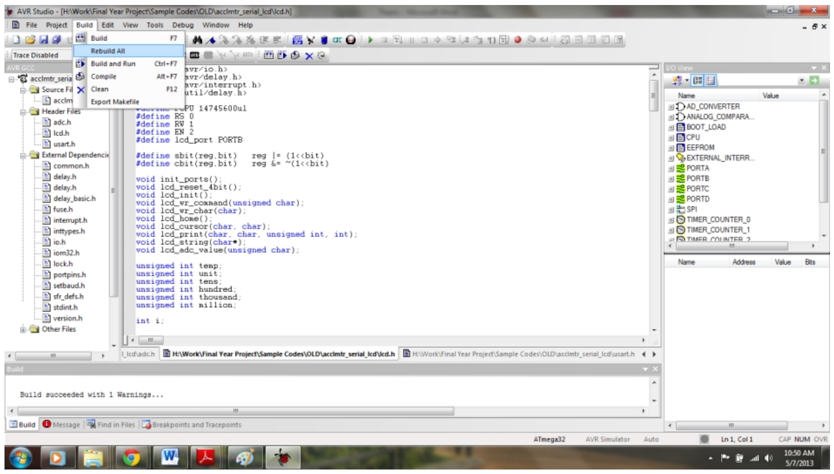
3.2 Software for programming and dumping on the microcontrollers

We use the following software to program and dump onto the ATmega32 and ATmega640 Microcontrollers.

1. AVR Studio 4 (Programming the Microcontroller)
2. SinaProg 2.0 (Dumping HEX file on Microcontroller)
3. WinAVR – Build 20090313 (Used in backend for compiler support)

The steps that need to be followed are:

a. Start "AVR Studio" from Start Menu->All programs->Atmel AVR Tools-> AVR Studio 4. Fig8. Shows the AVR Studio IDE being used.



```
avr/io.h>
avr/delay.h>
avr/interrupt.h>
util/delay.h>

      PU 14745600ul
#define RS 0
#define RW 1
#define EN 2
#define lcd_port PORTB

#define sbit(reg,bit)   reg |= (1<<bit)
#define cbit(reg,bit)   reg &= ~(1<<bit)

void init_ports();
void lcd_reset_4bit();
void lcd_init();
void lcd_wr_command(unsigned char);
void lcd_wr_char(char);
void lcd_home();
void lcd_cursor(char, char);
void lcd_print(char, char, unsigned int, int);
void lcd_string(char*);
void lcd_adc_value(unsigned char);

unsigned int temp;
unsigned int unit;
unsigned int tens;
unsigned int hundred;
unsigned int thousand;
unsigned int million;

int i;
```

```
Build succeeded with 1 Warnings...
```

Fig8. Screenshot of AVR Studio 4 running on Windows 7 platform

b. Select AVR GCC in Project type then enter a suitable project name say "hello" and select a location in the hard disk. Then click next. Make sure that "Create initial file" and "Create folder" option is checked.

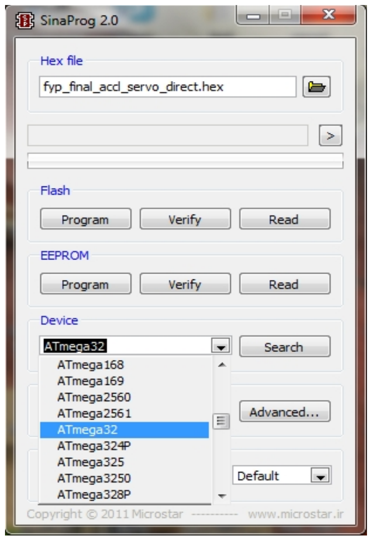
c. In next dialog box select AVR Simulator in "Debug Platform" list and select the ATmega32/640 in Device list. Click finish. After that you will be presented with an Integrated Development Environment-IDE.

d. Here create a new .c file and write the code as desired.

e. Then go to Menu->Build->Rebuild All. This will compile and generate all the required files. The .hex file which is needed for burning will also be generated in this step in the root folder.

f. Once the ".hex" file is generated it is dumped on the ATmega32/640 using "SinaProg".

g. Start "SinaProg" -> Open HEX file to dump -> Choose the type of Microcontroller being programmed -> Click on Program Command. The HEX file will be successfully dumped when the message "Programming OK" message appears on "SinaProg". Fig 9. Shows the SinaProg being used.


SinaProg 2.0

Hex file: fyp_final_accl_servo_direct.hex

Device: ATmega32

Default

Copyright © 2011 Microstar ---------- www.microstar.ir

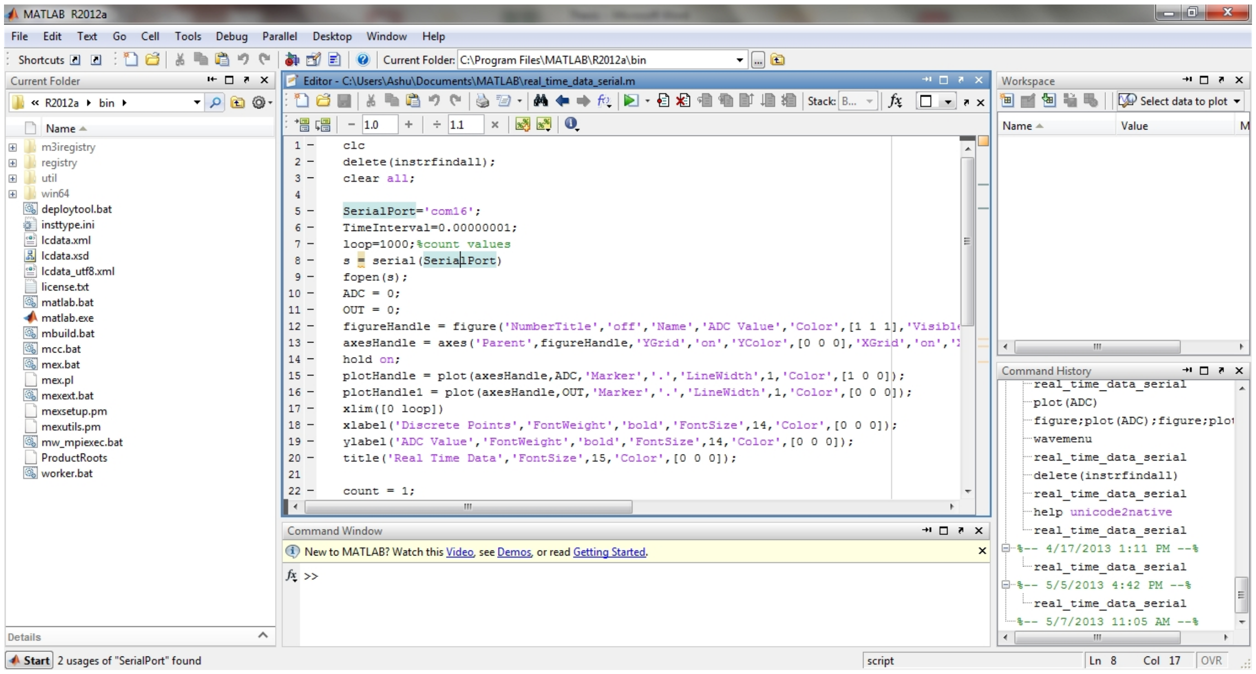
Fig9. Screenshot of SinaProg 2.0 running on Windows 7 platform

3.3 Software for Signal Processing and Actuation of the servo motors

We use the following software to interface the sensors and the servo motors via the ATmega32 and ATmega640 microcontrollers.

1. MATLAB v7.6 (R2012a)

Also signal processing is done in this software to reduce noise coming in from the sensors. Figure 10 shows a screenshot of MATLAB being used.


MATLAB R2012a
File Edit Text Go Cell Tools Debug Parallel Desktop Window Help
Current Folder: C:\Program Files\MATLAB\R2012a\bin
Editor - C:\Users\Ashu\Documents\MATLAB\real_time_data_serial.m
Workspace
Command History
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
2 usages of "SerialPort" found
script
Ln 8 Col 17 OVR

```
clc
delete(instrfindall);
clear all;

SerialPort='com16';
TimeInterval=0.00000001;
loop=1000;%count values
s = serial(SerialPort)
fopen(s);
ADC = 0;
OUT = 0;
figureHandle = figure('NumberTitle','off','Name','ADC Value','Color',[1 1 1],'Visible
axesHandle = axes('Parent',figureHandle,'YGrid','on','YColor',[0 0 0],'XGrid','on','
hold on;
plotHandle = plot(axesHandle,ADC,'Marker','.','LineWidth',1,'Color',[1 0 0]);
plotHandle1 = plot(axesHandle,OUT,'Marker','.','LineWidth',1,'Color',[0 0 0]);
xlim([0 loop])
xlabel('Discrete Points','FontWeight','bold','FontSize',14,'Color',[0 0 0]);
ylabel('ADC Value','FontWeight','bold','FontSize',14,'Color',[0 0 0]);
title('Real Time Data','FontSize',15,'Color',[0 0 0]);

count = 1;
```

```
real_time_data_serial
plot(ADC)
figure;plot(ADC);figure;plo
wavemenu
real_time_data_serial
delete(instrfindall)
real_time_data_serial
help unicode2native
real_time_data_serial
%-- 4/17/2013 1:11 PM --%
real_time_data_serial
%-- 5/5/2013 4:42 PM --%
real_time_data_serial
%-- 5/7/2013 11:05 AM --%
```

```
>>
```

Fig10. Screenshot of MATLAB v7.6 (R2012a) running on Windows 7 platform

Interfacing between the microcontrollers and the computer is done using serial communication that is being controlled by MATLAB.

3.4 Software Design

The software is designed to achieve the required objective. There are three software modules which make up the project are:

1. Software development for ATmega32: To receive data from the accelerometers and convert the real time analog signal to digital signals and transmits these digital signals to a computer via serial communication.
2. Software development for the Computer (Processing Unit): To receive data via serial communication from the ATmega32 and smoothens the data and calculate the appropriate values of the timer register contents for servo control and transmit these control words to ATmega640 via serial communication again.
3. Software development for ATmega640: To receive data from the Computer and store them in their respective timer registers and generate corresponding PWM signal for servo motor actuation.

The block diagram in Figure 9 shows the intermediate work/input entering the individual blocks.

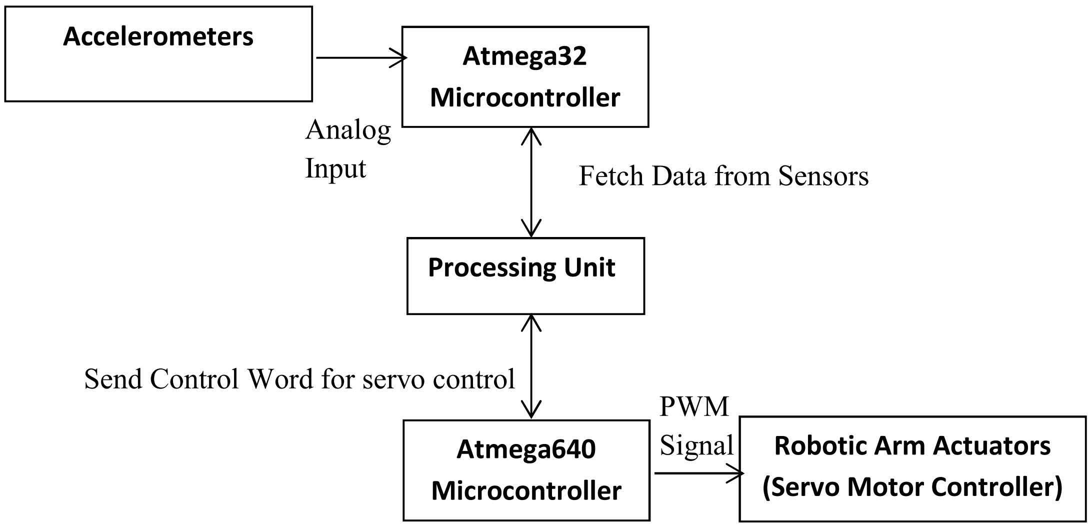

Accelerometers
Atmega32 Microcontroller
Analog Input
Fetch Data from Sensors
Processing Unit
Send Control Word for servo control
PWM Signal
Atmega640 Microcontroller
Robotic Arm Actuators (Servo Motor Controller)

Fig11. Block Diagram of the implemented system with signal information

The following chapter describes the implementation of the hardware and software of the robotic arm.

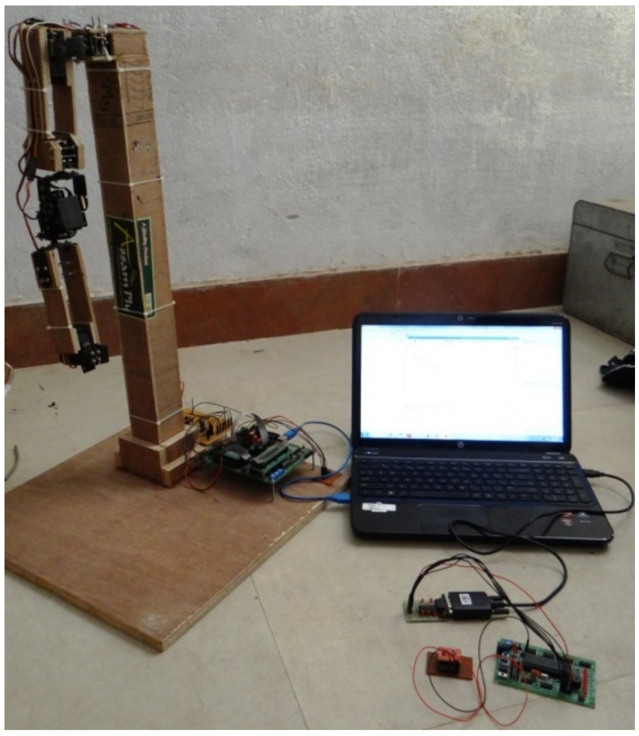
CHAPTER – 4

IMPLEMENTATION

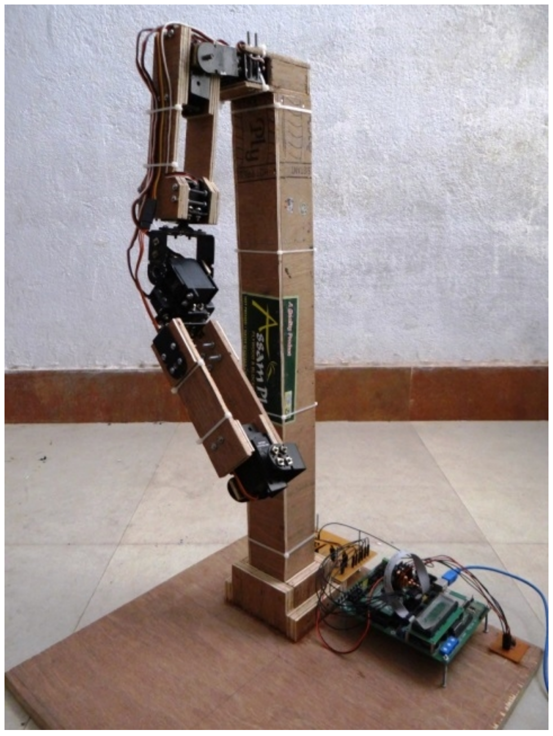
4.1 Implementation

The accelerometers are connected to the ATmega32 development board which is then connected to the Computer via serial communication. Now the data received by the computer is processed to remove as much noise as possible. Again the ATmega640 development board is connected with the computer through another serial communication channel.

The design of the software modules and hardware modules can be referred back to section 2.7 and section 3.5 respectively. Figure 12 shows the physical implementation of the system.



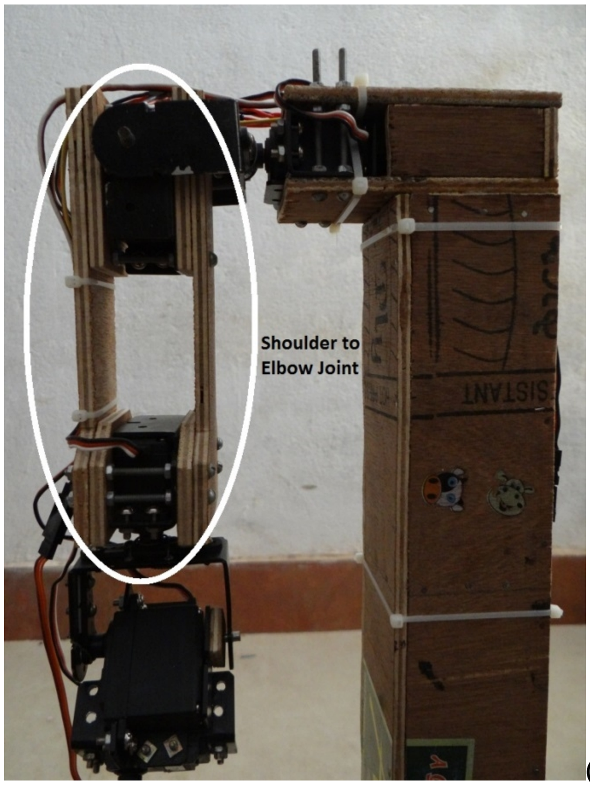
(a)



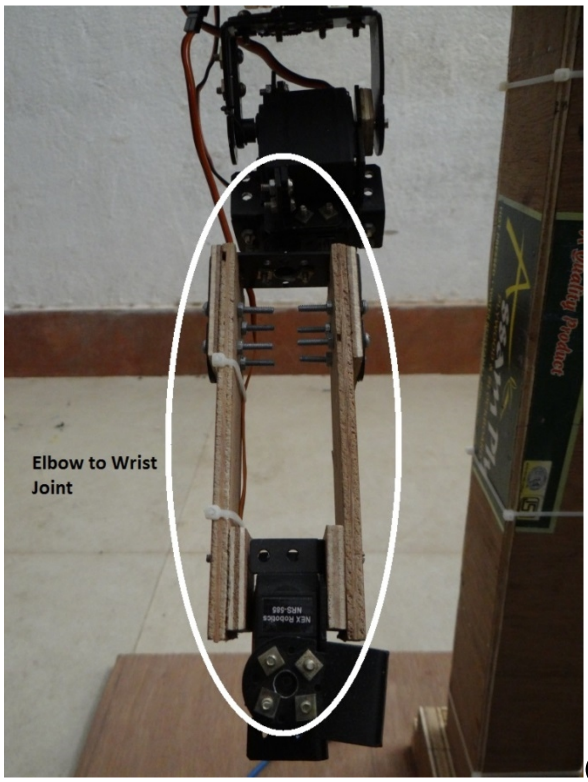
(b)

Fig.12. (a) Physical Implementation of the system; (b) Robotic Arm Only

The various joints of the robotic arm are shown in Figure 13.



(a)



(b)

Fig.13. (a) Implementation of the Shoulder to Elbow Joint; (b) Implementation of the Elbow to Wrist Joint

Each motor moves the arm in one plane. As we have implemented two motors at the shoulder joint as can be seen from Figure 14, M1 is to move the arm in Y-Z plane and M2 is for the movement along the X-Z plane. In this way the two motors provide the shoulder joint to be moved in any direction in space. From Figure 15, it can be seen that we have implemented three motors at this joint. The Motor M3 is for the movement of the arm along the Z-axis in the X-Y plane. The Motor M4 is used for the bending motion of the elbow and the Motor M5 is for the rotation/twisting of the elbow to wrist portion.

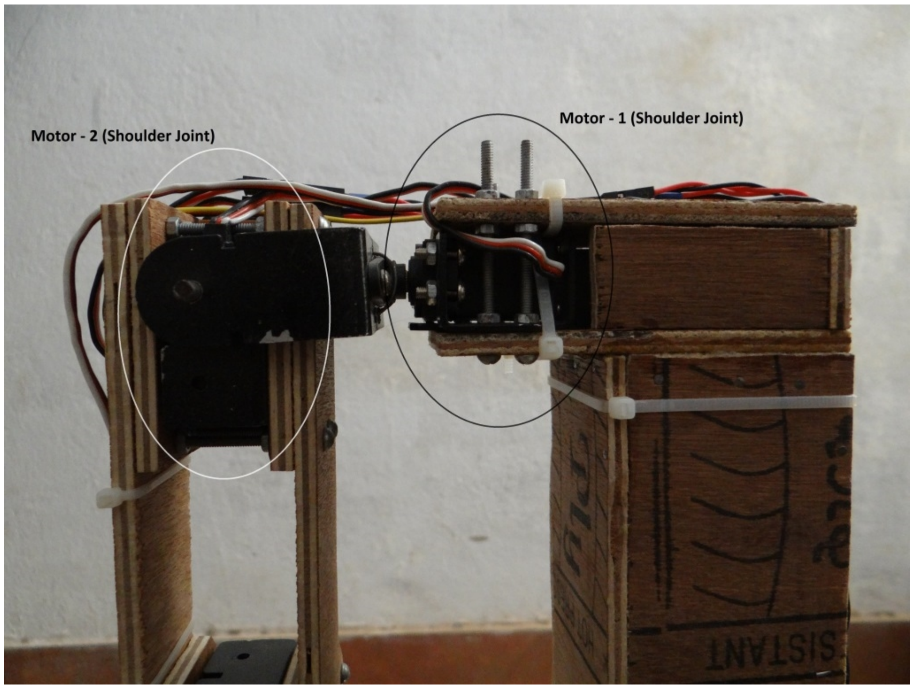


Fig.14. Shoulder Joint Motors (M1 and M2)

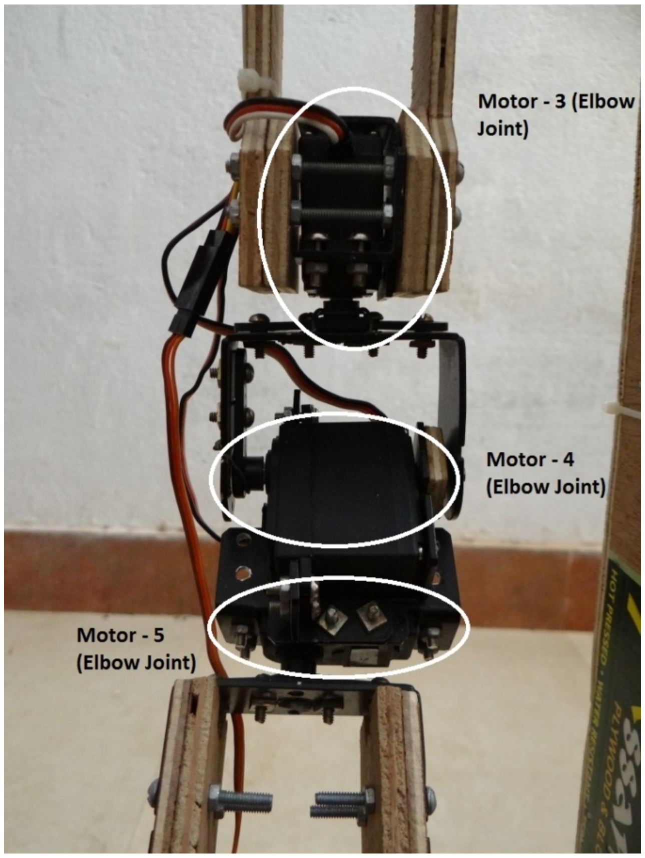
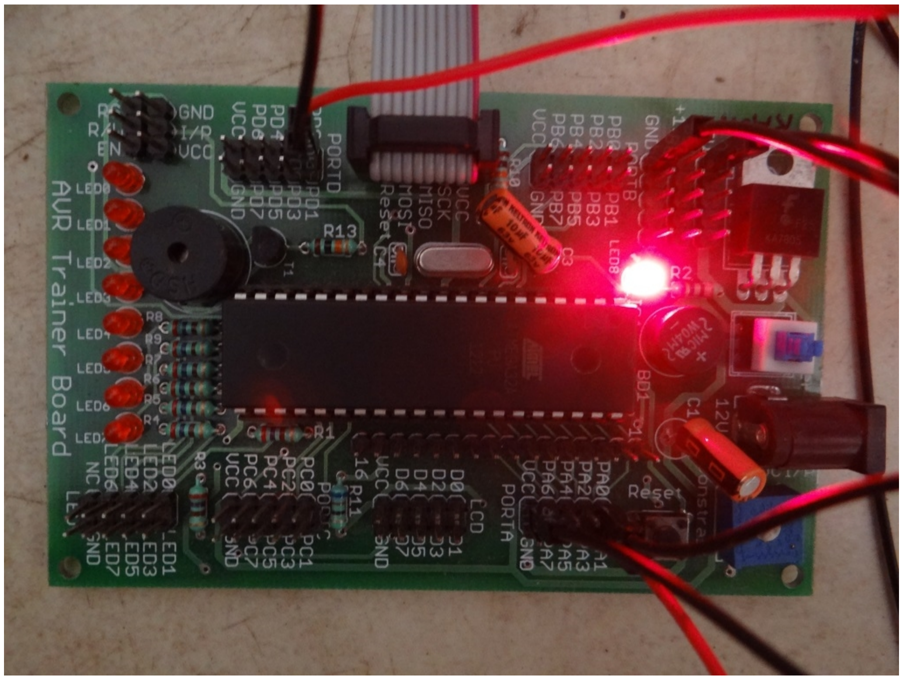
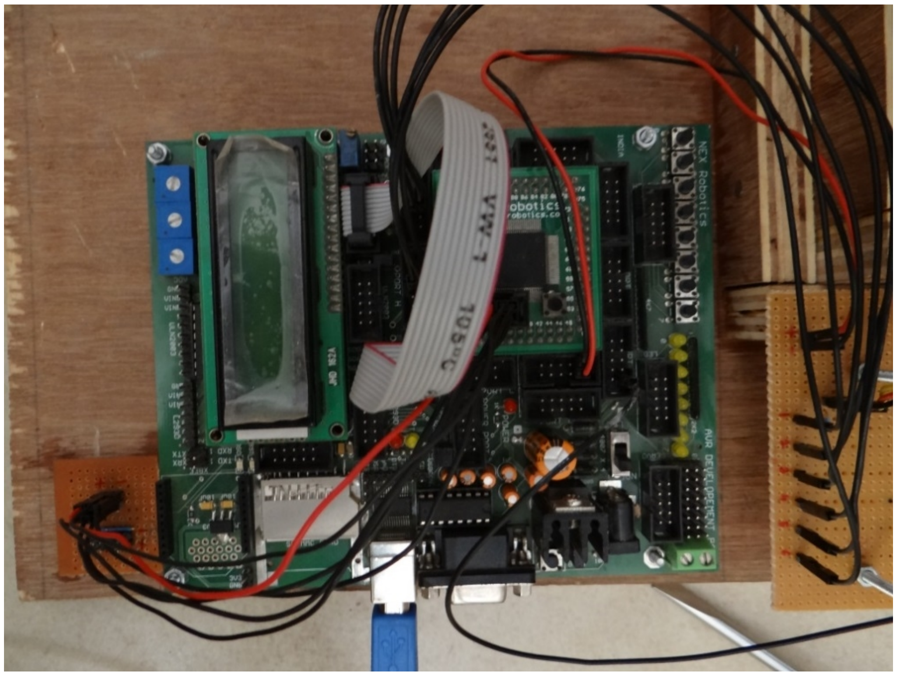


Fig.15. Elbow Joint Motors (M3, M4 and M5)

The ATmega32 and ATmega640 development boards are shown in Figure 13.



(a)



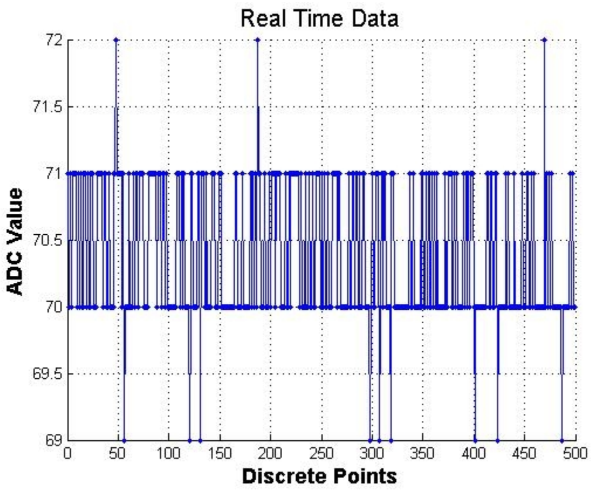
(b)

Fig16. (a) ATmega32 (b) ATmega640 Development Board

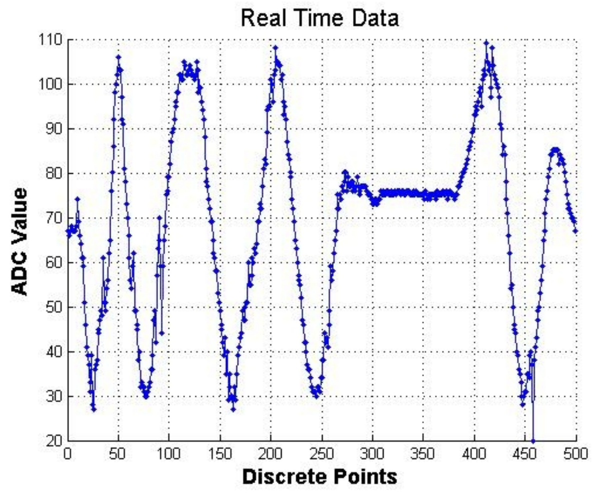
4.2 Data Acquisition, Processing and Calibration

4.2.1 Data Acquisition

The data from the accelerometer is shown in Figure 12 which is plotted using MATLAB. (ATmega32 sends the output of the accelerometer after A-D conversion through the serial port). Figure 17(a) and 17(b) show the real time plots of the serial data. Further processing will be carried out at a later stage of our project.



(a)



(b)

Fig17. (a) Real time plot when accelerometer is kept constant; (b) Real time plot when accelerometer is in rotation in both anti clockwise and clockwise direction.

The two plots show that there are shot and spikes in the input. Due to these sharp and sudden changes in the accelerometer output, the arm has sudden jerks in the movement. To minimize this jerking effect we use signal processing algorithms. Also, as MATLAB is being used to send control words to the ATmega640 for the control of the servo motors, the data acquisition of the accelerometer output and the processing of this output all have to be executed in real time in MATLAB. Therefore the following tasks are implemented in MATLAB:

1. Data acquisition from the ATmega32 via serial communication over “COMx”.
2. Processing of this acquired data to smoothen the data set.
    a. For better analysis of the signal we plot the input as well as the processed signal in real time.
3. Using the processed dataset to send appropriate control words to the ATmega640 via serial communication over “COMy”.

4.2.2 Data Processing

Here in our project we use a simple moving average algorithm to smoothen the output of the accelerometer.

Moving Average Algorithm: Given a series of numbers and a fixed subset size, the first element of the moving average is obtained by taking the average of the initial fixed subset of the number series. Then the subset is modified by "shifting forward"; that is, excluding the first number of the series and including the next number following the original subset in the series. This creates a new subset of numbers, which is averaged. This process is repeated over the entire data series.

The real time plot of the processed output that is drawn by MATLAB is achieved by connecting all the (fixed) averages in the moving average. This gives us the smoothened output.

Now the appropriate number of data points to be accounted for in moving average are found out by experimentation. Figure 18 shows the original input followed by the smoothened outputs when N = 5, 10, 15 respectively shown by Figure 19, 20 & 21.

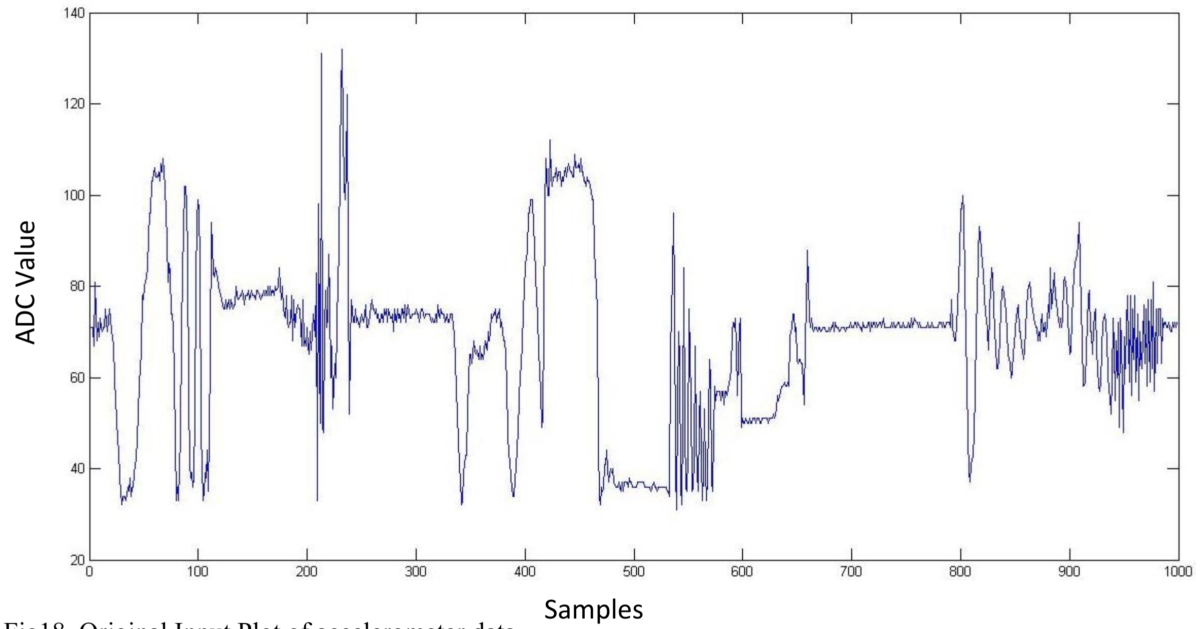


Fig18. Original Input Plot of accelerometer data

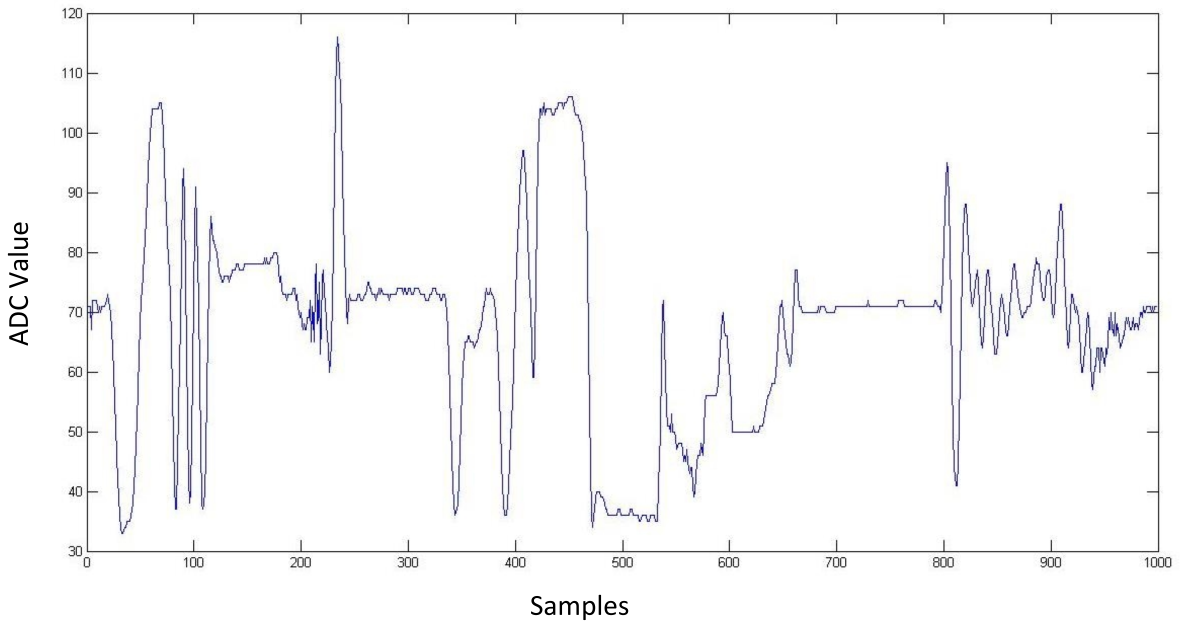


Fig19. Smoothened Output Plot of accelerometer data when N = 5

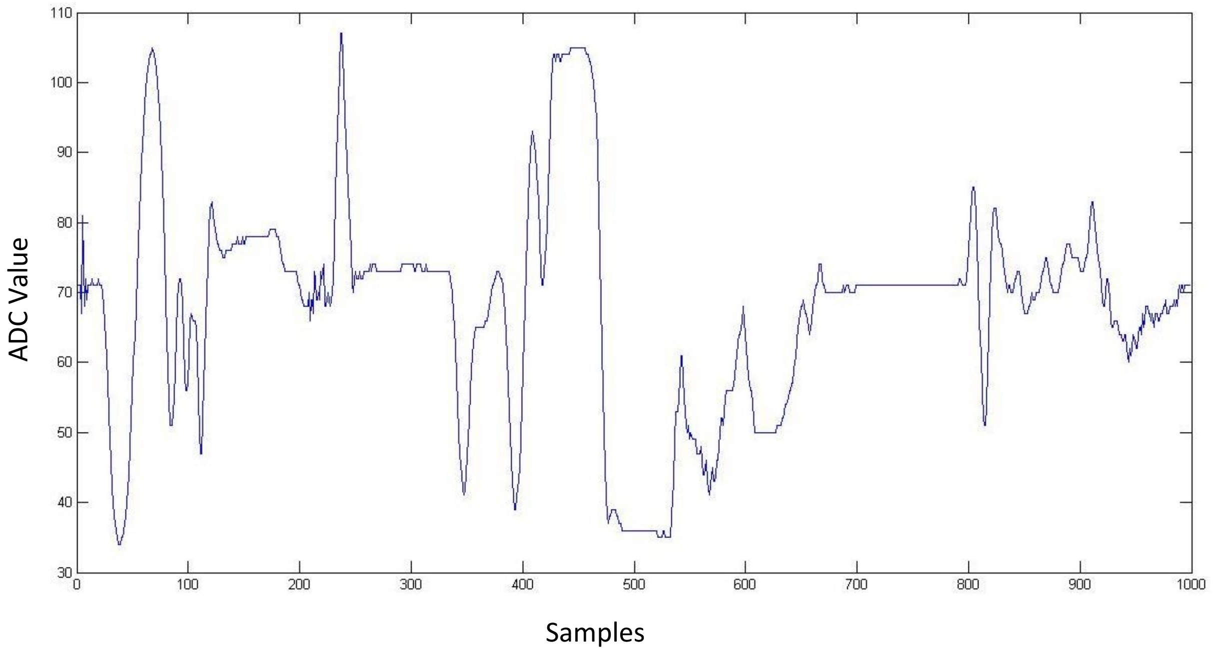


Fig20. Smoothened Output Plot of accelerometer data when N = 10

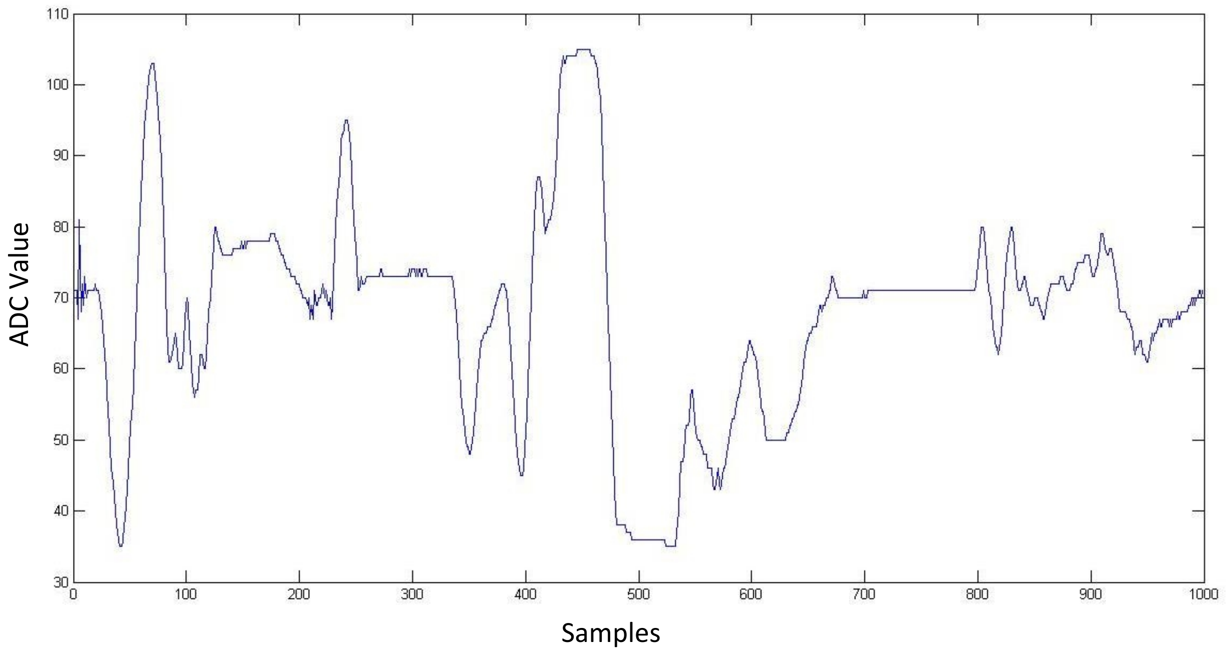


Fig21. Smoothened Output Plot of accelerometer data when N = 15

As can be seen from the processed outputs, choosing N = 5 proves to be of not much use. The output still is jerky. When N = 15 is chosen, the output becomes a lot smoother but it is smoothing out even the required movements that are actually not noises but rather movements that should be tracked by the accelerometer. Therefore, N = 10 is the chosen value for the number of samples to be taken for the moving average as it provides sufficient smoothing while still keeping all the necessary information intact.

4.2.3 Calibration of the Servo Motors

As mentioned above, we are using MATLAB to send the control words to the ATmega640 for the register values of the timer that will generate the PWM signal for their respective servo motors. To figure out the control word for the servo motor for a given value of the accelerometer output, it needs to be calibrated first.

The motors are kept at reference positions whose timer values are noted and their corresponding accelerometer outputs are noted as well. This gives us the value of the timer and value of the accelerometer for a particular position. This is done for all the motors separately along their direction of movement.

Three separate positions are taken for each motor and the intermediate values of the timer register values for the intermediate accelerometer output is interpolated using two point method which makes the system piece wise linear. Here we assume that the values given by accelerometer are approximately linear for the ranges taken into consideration.

Table 1 contains the respective values of the timer's register for the position of each individual motor.

Motor	**Position 1**	**Timer Register Value**	**Position 2**	**Timer Register Value**	**Position 3**	**Timer Register Value**
M1	0°	398	90°	188	-	-
M2	0°	220	90°	396	-	-
M3	-90°	441	0°	305	90°	220
M4	0°	253	90°	459	165°	545
M5	-90°	166	0°	345	90°	532

Table1. Table containing the Timer Register Values for the reference positions taken during calibration.

After calibration is done, then the MATLAB code is implemented which takes the accelerometer data coming in from the serial port, smoothen it and calculates the corresponding timer register value for the ATmega640 to be sent over another serial port.
This completes the implementation of the robotic arm.

The next section gives the conclusion and the future scope of the project.

CHAPTER – 5

CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

The objectives of this project has been achieved which was developing the hardware and software for an accelerometer controlled robotic arm. From observation that has been made, it clearly shows that its movement is precise, accurate, and is easy to control and user friendly to use. The robotic arm has been developed successfully as the movement of the robot can be controlled precisely. This robotic arm control method is expected to overcome the problem such as placing or picking object that away from the user, pick and place hazardous object in a very fast and easy manner.

5.2 Future Scope

The project is built on a wired model. It could further be developed to work on wireless communication, thus allowing the user to move in an even easier unrestricted manner. A clamper can be connected on the motor M6 which will allow the movements of the palm and allow picking and placing of objects. Currently the accelerometer signal is being processed via a digital computer; this could be eliminated by using a fast microprocessor such as ARMv7, etc. It could also be possible to eliminate the ATmega32 altogether when ARMv7 is being used. The microprocessor could take the input from the accelerometer and smoothen it and then generate the corresponding PWM signal itself to actuate the servo motors.

BIBLIOGRAPHY

[1] Mohd Ashiq Kamaril Yusoffa, Reza Ezuan Saminb, Babul Salam Kader Ibrahimc, “Wireless Mobile Robotic Arm”, International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012), July 2012

[2] Wan Muhamad Hanif Wan Kadir, Reza Ezuan Samin, Babul Salam Kader Ibrahim, “Internet Controller Robotic Arm”. International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012), July 2012

[3] Avinash Jain, “Servo Motor Control by Using AVR ATmega32 Microcontroller”, http://extr emeelectronics.co.in/avr-tutorials/servo-motor-control-by-using-avr-atmega32-microcontroller/, June 2010

[4] Paul Smith, “Programming with AVRDUDE”, http://www.ladyada.net/learn/avr/ avrdude .html/, April 2012

[5] Avinash, “Using LCD Modules with AVR”, http://extremeelectronics.co.in/avrtutorials/using -lcd-module-with-avrs/, July 2008

[6] Avinash, “Using ADC on AVR”, http://extremeelectronics.co.in/avr-tutorials/using-the-analog-to-digital-converter/, September 2008

[7] Avinash, “Using the USART of Microcontrollers”, http://extremeelectronics.co.in/avr-tutorials/using-the-usart-of-avr-microcontrollers/, December 2008

[8] Atmel ATmega32 Datasheet, AVR Corporation, Feb 2011

[9] Atmel ATmega640 Datasheet, AVR Corporation, April 2012

[10] ATmega640 Development Board Manual, Nex Robotics, Oct 2010

[11] MMA7361L Datasheet, Freescale Semiconductors, Apr 2008