SIMULATION OF VIRTUAL SENSORS OVER INTERNET

By

Salman Mahmood

Wasif Arshad

Omar Ali

Imran Khan

Supervisor:

Major Dr. Adil Masood

Submitted to the Faculty of Department of Electrical Engineering,

Military College of Signals, National University of Sciences and Technology, Islamabad in

partial fulfillment for the requirements of a B.E. Degree in Telecom Engineering

JUNE 2012

ABSTRACT

SIMULATION OF VIRTUAL SENSORS OVER INTERNET

At present, systems employing movement replication are limited to surgical purposes. The DaVinci system by Intuitive Surgical USA and ZEUS Robotic Surgical System by Computer Motion are two such examples. However, these systems are very costly and cannot be used for other purposes due to their large size and weight. The principle goal of the project is to provide a general purpose product which can be used in multiple situations, not just for medical purposes.

Movement replication via simulation of virtual sensors over the internet allows the movement of a human arm to be replicated on a remote site using internet as a medium of communication through a robotic counterpart.

Such a solution can help soldiers stay away from unnecessary exposure to dangers yet performing the given tasks, medical supervision available to countless patients via remote assistance, and radioactive material handled by non-human counterparts where needed.

The project uses a potentiometer based exo-skeleton that translates the movements of a human arm into potential differences. These potential differences are digitized using a PIC based ADC and then sent to a computer via USB. This digital sequence is transmitted to the remote system using internet, which when received, is transformed into corresponding PWM signals. These PWM signals are fed to the corresponding servos of the robotic arm to perform the movement.

There could be numerous dangerous situations in which human presence can be replicated by robots to perform specific tasks thus saving precious lives. Examples of these situations include battlefields, bomb disposals and areas with radioactive and poisonous gas leaks where some kind of operation is essential. This is a partial list of situations where human presence is required but it could be either too dangerous for them to be there or it may be altogether impossible. Hence an alternate system is required with which experts can operate in such areas without having to be physically present there.

DECLARATION

No portion of the work presented in this dissertation has been submitted in support of another award or qualification either at this institution or elsewhere.

DEDICATION

In the name of Allah, the Most Merciful, the Most Beneficent

To our parents, without whose support and continuous cooperation, a work of this magnitude would not have been possible.

ACKNOWLEDGEMENT

We are grateful to ALLAH, who has given us guidance, strength and enabled us to accomplish this task. The journey on this road of success had not been possible without His will, and henceforth, all this success and accomplishment, we owe it to Him, in totality.

We are also grateful to our parents and family and well-wishers for their untiring support. We would like to thank our supervisor Major Dr. Adil Masood, for his help and motivation throughout the course of our project. Without his help we would have not been able to accomplish anything. Under his guidance, we were able to achieve our goals.

Table of Contents

List of Tables

Table of Figures

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Introduction

This chapter describes the project overview, problem statement, project objectives and organization of this document.

The project is to develop a system that will allow the simulation of a sensor in real time on a remote system located in any part of the world. The sensor will translate the motion of the user and transmit it over the internet.

There could be numerous dangerous situations in which human presence can be replicated by robots to perform specific tasks thus saving precious lives. Examples of these situations include battlefields, bomb disposals and areas with radioactive and poisonous gas leaks where some kind of operation is essential. This is a partial list of situations where human presence is required but it could be either too dangerous for them to be there or it may be altogether impossible. Hence an alternate system is required with which experts can operate in such areas without having to be physically present there.

Such a solution can help soldiers stay away from unnecessary exposure to dangers yet performing the given tasks, medical supervision available to countless patients via remote assistance, and radioactive material handled by non-human counterparts where needed.

1.2 Objectives

This section provides details on the targets of this project. It discusses in detail the type of the project and previous work done in this field in addition to the advantages of this project over the existing systems.

1.2.1 Type of Project

This project is basically an implementation project. The system this project implements has been previously developed by many organizations. However, all of these systems either lack the communication methods to use such a system over very long ranges or are simply too expensive to be used for general purpose applications like preventive maintenance of an offshore site of a multinational firm. The project proposes to use internet as the method of communication, thus making it possible to use this system remotely in any part of the world in addition to simple and less expensive alternative sensors for motion capturing and translation in order to minimize the cost of the entire system.

1.2.2 Previous Work

As already mentioned above, this is not a research based project and has seen various implementations previously. Some of the implementations that we got hold of were Da Vinci Surgical System by Intuitive Surgical and ZEUS Robotic Surgical System by Computer Motion. These systems are either too expensive or have a limited range, both of which are not suitable for a general purpose system which can be used for anything that involves human movement. The system proposed in this project addresses both aspects and provides a system that is low in cost but has a large range.

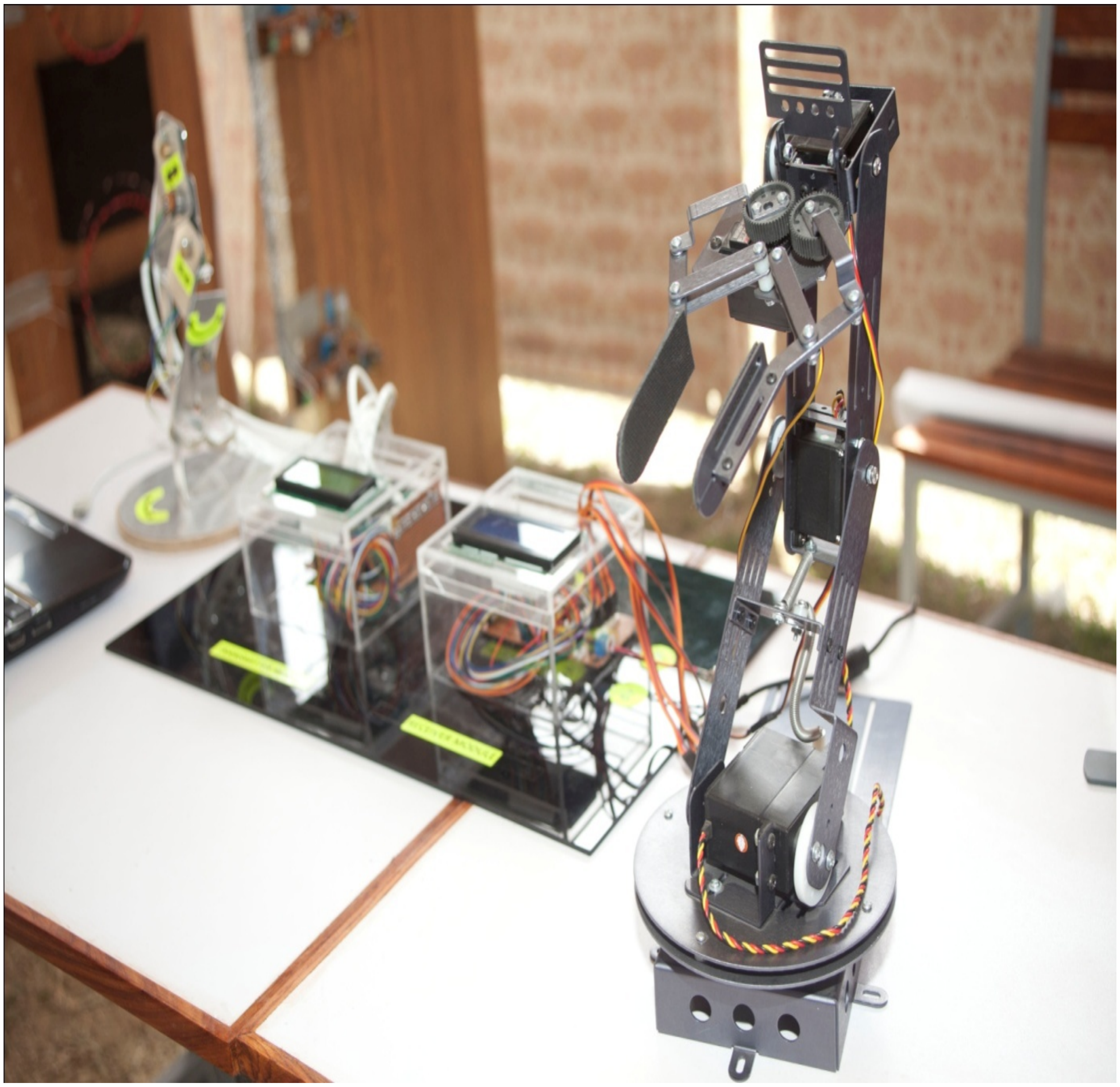


Figure 1-1: A Glimpse of the Project

Figure 1 illustrates the complete setup of the project including the robotic arm, the transmitter unit, the motor control module, the sensor and the camera.

1.3 Block Diagram

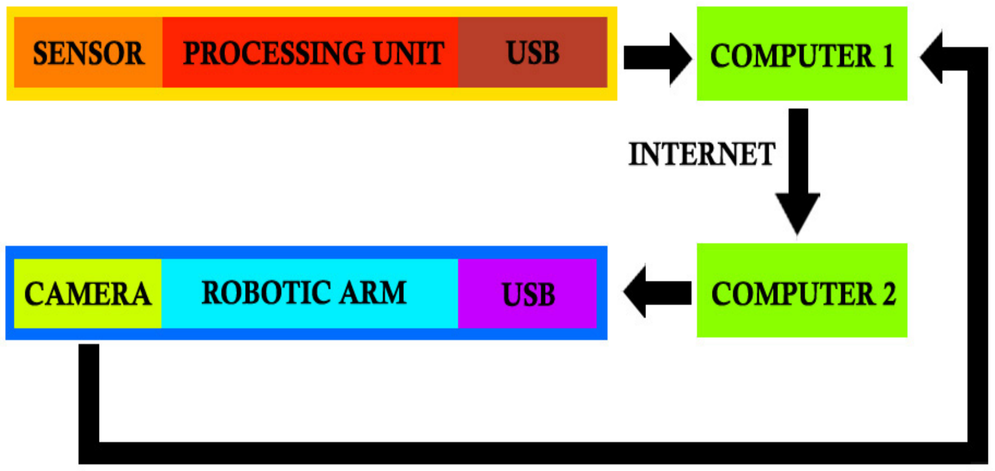


Figure 1-2: Block Diagram

Figure 2 shows the block diagram for the project. The sensor sends analog signals to the microcontroller which digitizes them and sends them to the computer via USB. These signals are transmitted over the internet to another system where they are forwarded to a motor control module attached to the PC via USB which converts them to appropriate PWM signals and sends them to the robotic arm. A camera is attached for feedback as well which sends the video of the replicating end to the transmitting PC.

1.4 Background

Robotics industry leaders point out that advances in military, transportation, medical, and other non-manufacturing robotics applications, where research and development investments are justified by dramatic potential benefits, will provide the methods to improve next generations of robots for manufacturing applications.

The end effectors can be designed to perform any desired task such as remote bomb disposal unit, remote assistance and trouble shooting, security at check points, medicine and utilization of skilled workforce over the internet for an economy boost

1.5 Teleoperation of Robotic Arm

Teleoperation capabilities or the ability for a project is to operate and control a robotic arm remotely from a safe location through internet. There could be numerous dangerous situations in which human presence can be replicated by robots to perform specific tasks thus saving precious lives. Examples of these situations include battlefields, bomb disposals and areas with radioactive and poisonous gas leaks where some kind of operation is essential. This is a partial list of situations where human presence is required but it could be either too dangerous for them to be there or it may be altogether impossible.

Teleoperation capabilities are essential to the warfighter because they enable remote operations and thereby eliminate the presence of the operator in hostile environment, such as minefields and in areas of potential explosive hazards.

1.6 Organization of Document

Chapter 2 deals with the components used in the project. Chapter 3 deals in the detailed discussion of each module of the project. Chapter 4 deals with the applications and future work in this field. Appendix A deals with the code for the transmitter end. Appendix B deals with the code for the receiver end. Appendix C deals with the code for the transmitter desktop application. Appendix D deals with the code for the receiver desktop application.

CHAPTER 2

COMPONENTS OF THE PROJECT

2.1 Components used in the project

The main components of the system are Exo-Skeleton, sensor, motor control module, servo motors and power supply.

2.1.1 Exo– Skeleton

A metallic structure for supporting the potentiometer network and accurately positioning the potentiometers against the joints of a human arm would also be made. The project uses Aluminum strips due to their durability and lesser weight. The exo skeleton contains six individual strips which are connected through potentiometers as shown in the figure.

2.1.2 Sensor

The project uses a network of potentiometers knitted into a metallic exoskeleton for capturing and translating the motion of the user. The sensor comprises of the potentiometer network and PIC184550 based board.

2.1.3 Potentiometer Network

All the potentiometers would have a reference, (50% resistance) at which the output will be taken as zero. The potentiometers would be interfaced with the ADC of a PIC microcontroller that would convert the analog voltage provided as input into a stream of bits. The bits would correspond to the movement at the potentiometer.

2.1.4 PIC 18F4550

The project uses PIC 18F4550 to be the brain of this entire project. The reasons for our selection of the device are low cost of PIC as compared to other microcontrollers with the same capabilities, greater efficiency than other interfacing modules and simplicity of application, predefined methods and functions in order to enable serial or parallel communication between potentiometer network and PC, greater capacity, processing speed and response as compared to 8051 microcontroller, flexibility in control programming and interrupt handling and internal USB function is available and can be easily accessed.

2.1.4 .1 Resistors

A resistor is a passive terminal electrical that implements electrical resistance as a circuit element. Current through a resistor is directly proportional to the potential applied across its terminals. Thus, the ratio of the voltage applied across a resistor's terminals to the current through the circuit is called resistance. The project uses resistors to limit the current for the MCLR pin of the microcontroller and to set the contrast on the LCD.

2.1.4 .2 Capacitors

A capacitor is a passive terminal electrical used to store energy in an electric field. Capacitors are basically two conductors separated by a dielectric. The project uses capacitors in the system with the crystal as decoupling capacitors.

2.1.4 .3 Crystal

A crystal is a device that uses pressure sensitive crystals/materials to produce precise frequencies. The project uses 20MHz crystal to provide clock to the microcontroller.

2.1.4 .4 LCD

A liquid crystal display (LCD) is a device used to display messages by using liquid crystals. The project uses the alphanumeric LCD for this purpose.

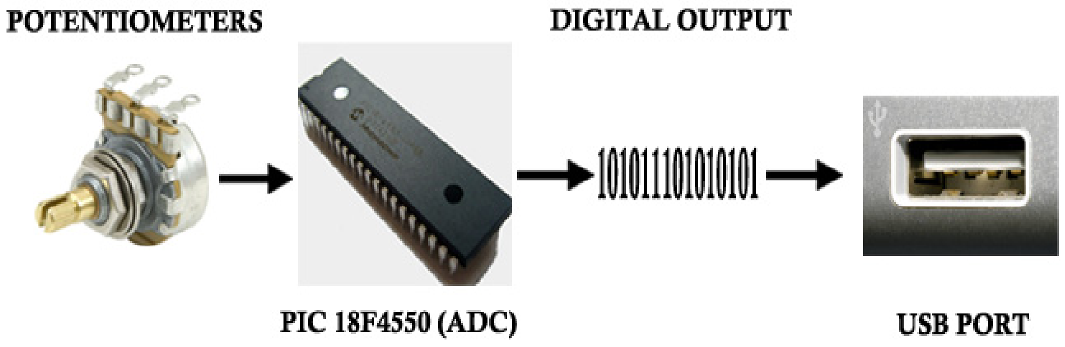


Figure 2-1: Sensor Block Diagram

2.1.5 Motor Control Module

The motor control module is at the receiving end and is used to control the motors by producing the corresponding PWM signals. The components used in the motor control module are PIC18F4550 microcontroller, 20MHz crystal, Capacitors, Resistors and LCD.

2.1.5 .1 Servo Motors

The servo motors are used in the robotic arm for replication of the movement. There are three types of servo motors that the project uses. The specifications of these motors are listed in table 2-1.

Model Number	**Torque**
HS-485HB	4.8V: 72.0 oz-in (5.18 kg-cm) 6.0V: 89.0 oz-in (6.41 kg-cm)
HS-645MG	4.8V:106.9 oz-in (7.70 kg-cm) 6.0V:133.3 oz-in (9.60 kg-cm)
HS-805BB	4.8V:275.0 oz-in (19.80 kg-cm) 6.0V:343.0 oz-in (24.70 kg-cm)

Table 2-1: Servo Model Specifications

2.1.5 .2 Power Supply

A power supply is used to power the servo motors. This is a 5V, 4A power supply which takes 220V AC as input. The power supply has been connected to the module by using the terminal blocks.

CHAPTER 3

SYSTEM DESIGN

3.1 System Modules

System modules which are used in the project are explained individually in great detail in this section.

3.1.1 Block Diagram

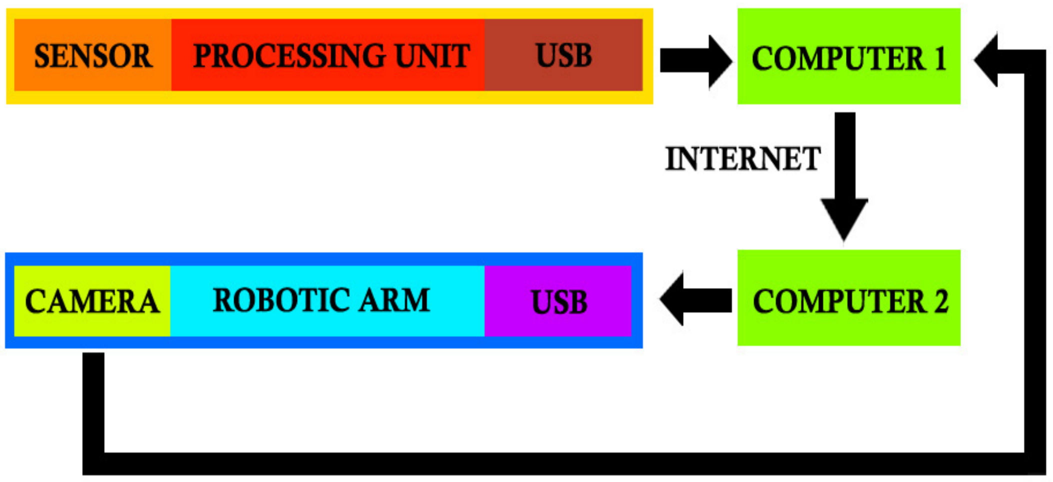


Figure 3-1: Block Diagram

3. 2 Transmitting End

The transmitting end consists of the sensor, the processing unit for converting analog signals to digital signals and the USB interface for transmitting these signals to the PC.

3.2.1 Sensor

The sensor is basically a network of potentiometers knitted into a metallic exoskeleton for capturing and translating the motion of the user. The sensor comprises of potentiometer network, Exo skeleton.

All the potentiometers have a reference, (50% resistance) at which the output will be taken as zero. The potentiometers are interfaced with the ADC of a PIC microcontroller that would convert the analog voltage provided as input into a stream of bits. The bits correspond to the movement at the potentiometer.

3.2.2 Exoskeleton

The exoskeleton is made of aluminum strips. The exoskeleton is made in such a manner that when assembled, it has the same movement as that of the robotic arm at the replicating end. The number of joints represents the degrees of freedom or in other words, the number of motors in the robotic arm.

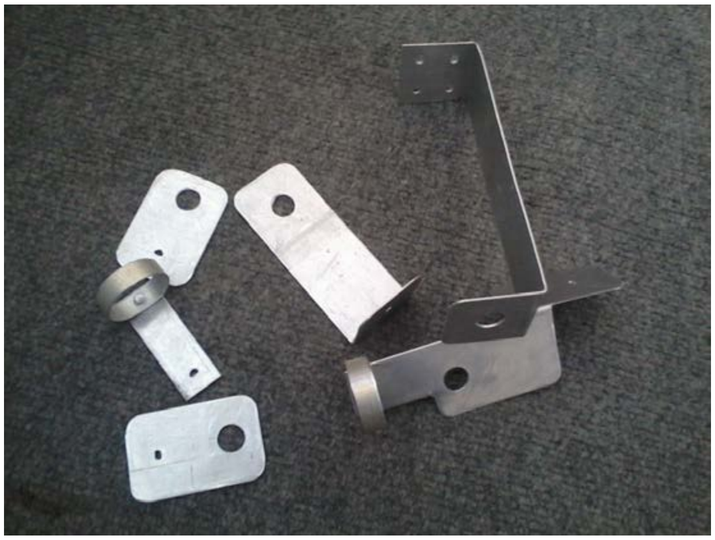


Figure 3-2: Exoskeleton Strips

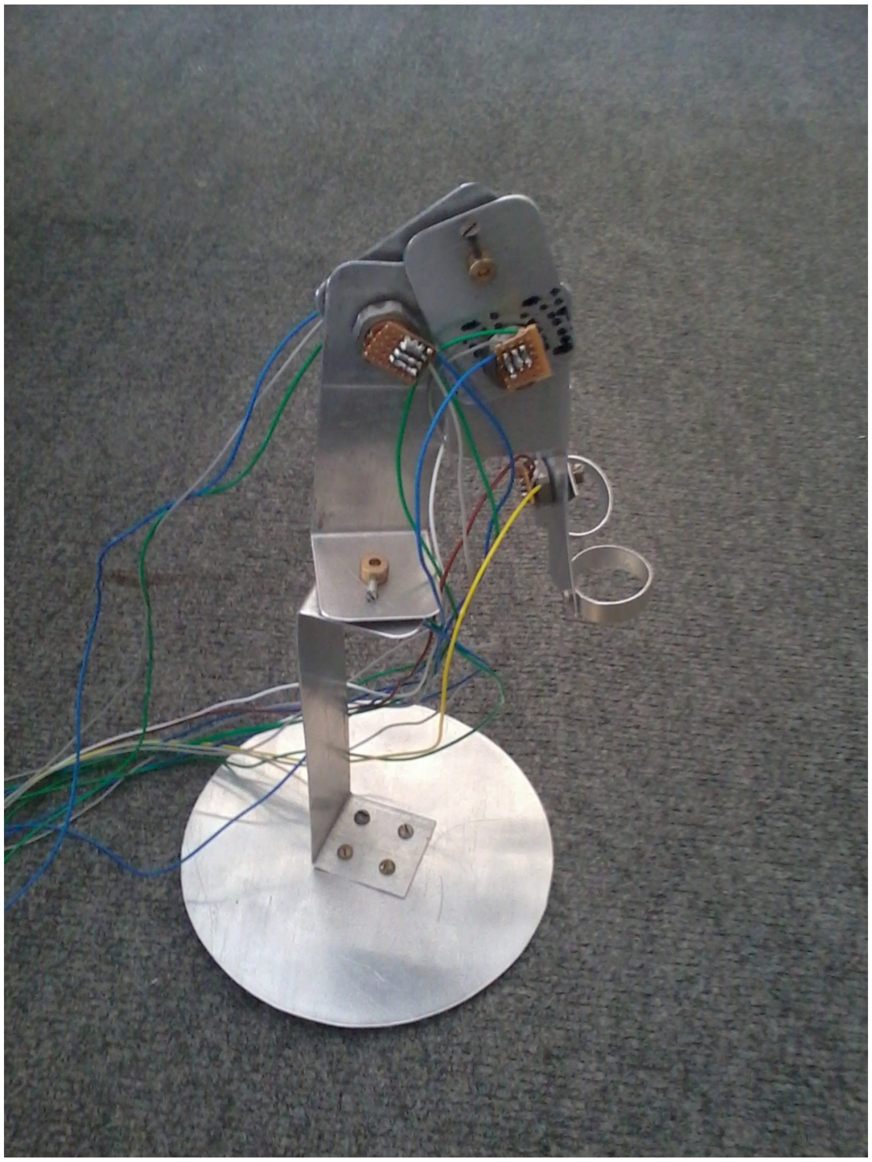


Figure 3-3 : Fully Assembled Exoskeleton

3.2.3 Potentiometer Network

Five potentiometers are attached to the joints of the exoskeleton. The maximum resistance of each potentiometer is 5K ohm. All the potentiometers are set at 50% resistance in while assembling the exoskeleton so that both upwards and downwards, left and right movement can be captured.

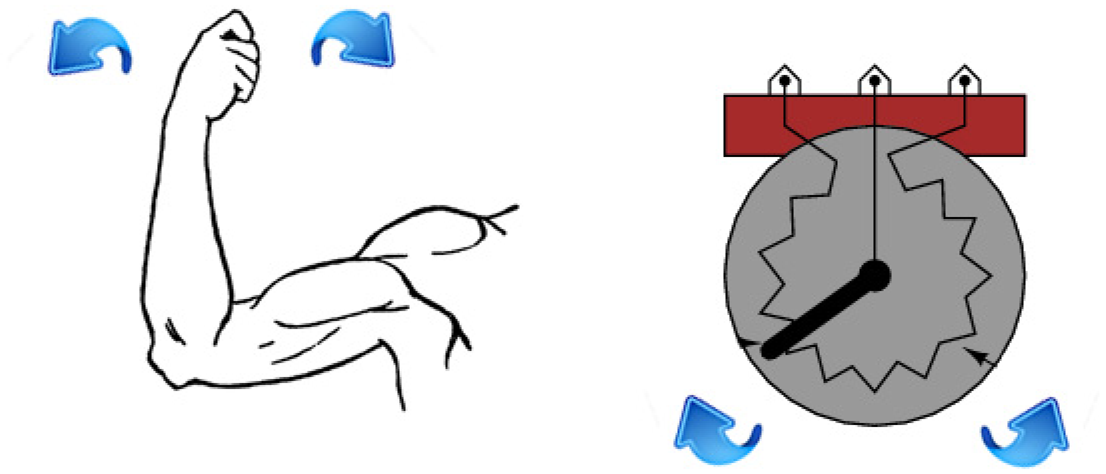


Figure 3-4: 50% Reference for Potentiometers

When the potentiometers are turned clock wise, the resistance of the potentiometer decreases and when the potentiometer is turned anti clock wise, the resistance increases. Hence when the hand is moved upwards, the resistance increases and vice versa for the downwards movement.

Figure 3-5: Potentiometers Used in the Exoskeleton

3.3 Processing Unit of the Transmitter

The function of the transmitter processing unit is to digitize the analog potential differences produced by the potentiometers due to the movement of the exoskeleton. The heart of the processing unit is the PIC 18F4550 microcontroller which has an inbuilt 10 bit ADC. The potential differences are fed to the microcontroller's ADC which provides the digitized sequences.

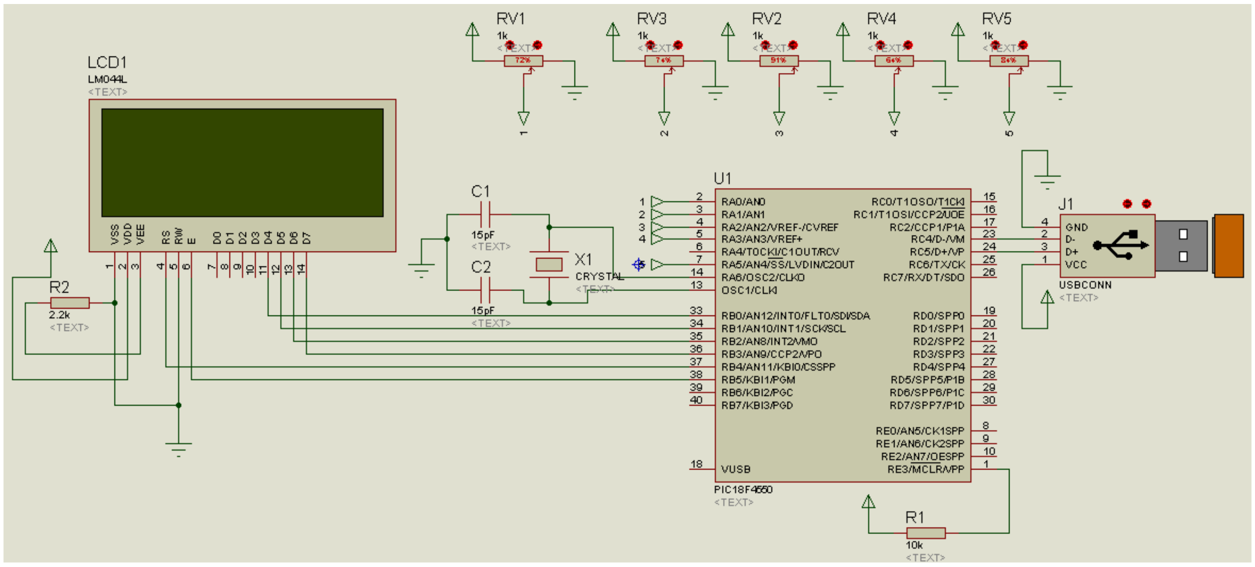


Figure 3-6: Transmitter Schematic

3.4 PIC18F4550 Microcontroller

PIC 18F4550 microcontroller is a very powerful microcontroller with an inbuilt 10bit ADC and a USB 2.0 peripheral. The transmitter module uses its ADC and USB modules. Both the sections are discussed in detail.

3.4.1 PIC18F4550 as an ADC

The Analog-to-Digital (A/D) converter module has 13 inputs. This module allows conversion of an analog input signal to a corresponding 10-bit digital number. The module has five registers.

3.4.1 .1 A/D Result High Register (ADRESH)

This is an eight bit register used for saving the eight most significant bits of the result. This result is later sent to the PC for transmission over the internet. It basically represents the higher half of the analog value.

3.4.1 .2 A/D Result Low Register (ADRESL)

This is an eight bit register used for saving the eight most significant bits of the result. This result is later sent to the PC for transmission over the internet. It basically represents the lower half of the analog value.

3.4.1 .3 A/D Control Register 0 (ADCON0)

ADCON0 register is the most important control register of the PIC18F4550 ADC. It includes the configuration bits for selecting the channels for analog input.

This is basically done by configuring CHS3:CHS0 bits. In addition to this ADON and GO/!DONE are used for enabling the ADC and for initializing the ADC for conversion respectively.

The value of ADCON0 register is set to 00010000 or 0x10H. This means that the first five analog channels from AN0 to AN4 have been selected.

3.4.1 .4 A/D Control Register 1 (ADCON1)

This register is used for configuring the references for ADC i.e. VREF+ and VREF-. In addition to this, the last four bits of this register i.e. PCFG3: PCFG0 are used to configure which of the thirteen pins dedicated for the ADC can be used as analog pins and which can be used as digital I/O pins.

Set the value of ADCON1 register to 00001010 or 0X0AH. This means that the VREF+ is +5V, VREF- is ground, and only pins 0,1,2,3,4 and 5 of port A are selected as analog inputs, the rest are configured as digital input/output pins.

3.4.1 .5 A/D Control Register 2 (ADCON2)

This register is used for configuring the clock settings and acquisition time for the ADC. In addition to this, there is a bit reserved for justifying the result to the left or to the right.

Set the value of ADCON2 register to 10001010 or 0x8AH. This means that the result is justified to right and selected ADC clock as 1/32 times the oscillator frequency.

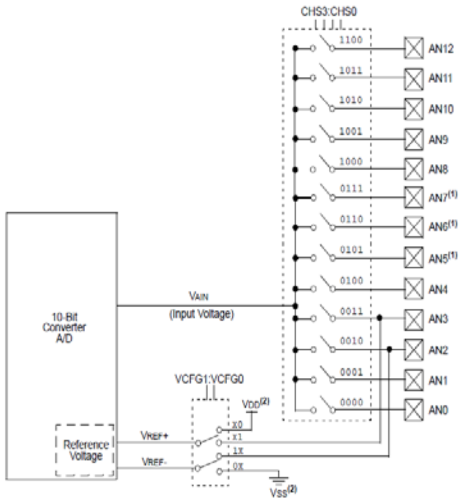


Figure 3-7: ADC Block Diagram

3.5 Transmitter USB Module

Once the ADC has digitized the potential differences from the exoskeleton, the digitized sequences are transmitted to the PC using USB. PIC18F4550 microcontroller has an inbuilt USB peripheral that can be used as a Human Interface Device (HID) as well as a storage device. The project uses USB peripheral as a HID device. The USB peripheral is 2.0 version capable of data transmission at 480Mbps. For this data rate, the frequency of operation for USB module is 48MHz.

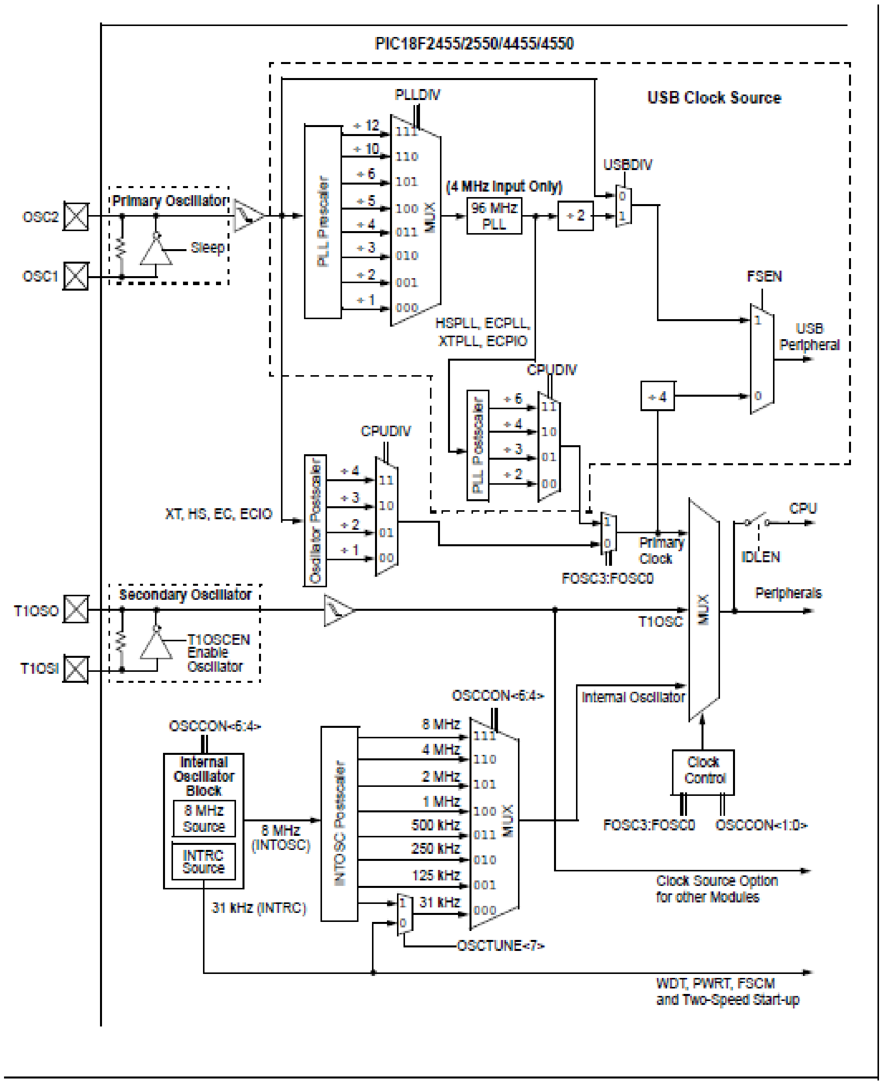


Figure 3-8: Clock Structure for PIC18F4550

The vendor ID (VID) for our transmitter device is 1234 and the product ID (PID) is 0001. The USB memory bank is shown in the figure 3-19. The buffer descriptors and USB data take the memory space from 400h to 4FFh. The User data for write buffer is allocated space from 500h to 7FFh. There are two user data buffers each of three bytes used in the code for the transmitter USB module.

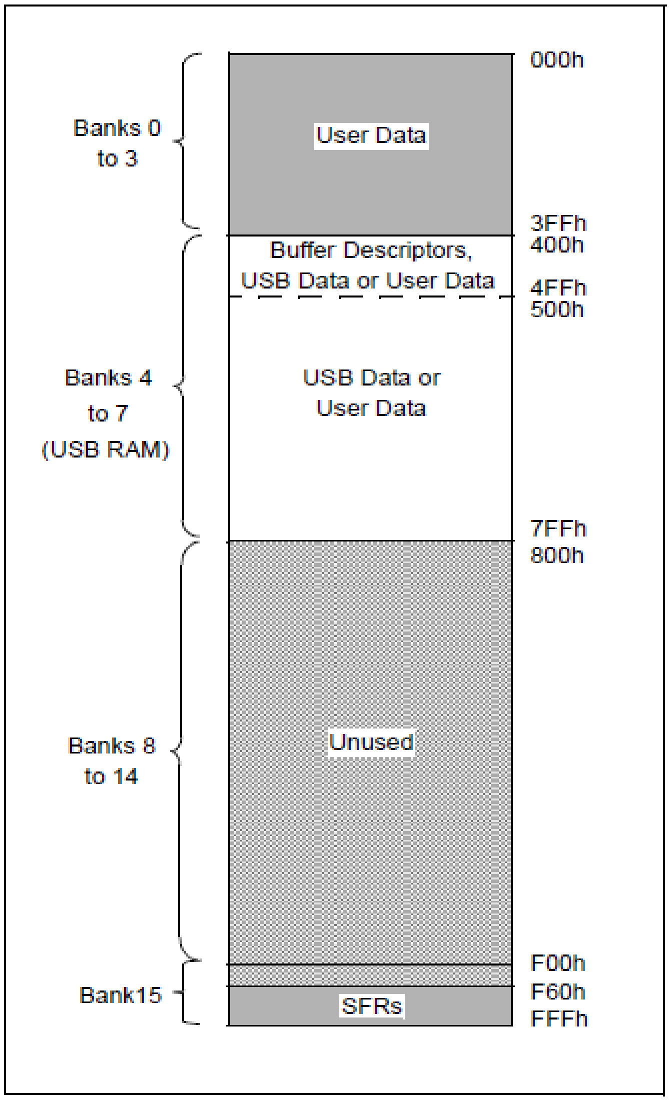


Figure 3-9 : PIC18F4550 Memory Structure

3.5.1 Read Buff

This is the read buffer of the USB module. All the data that is to be read from the PC is stored in this buffer. Basically this is an array with address from 500h to 539h.

3.5.2 Write Buff

This is the write buffer of the USB module. All the data that is to be sent to the PC is stored in this buffer. Actually this is an array with address from 540h to 7FFh. The size of array is three bytes. The array data is explained as follows:

First entry of the write buffer array is the analog channel number. This is an important piece of data as it tells us which potentiometer had a resistance change due to the arm's movement. The second entry of the write buffer array is the two bits of the digitized sequence. These are actually the contents of the ADRESH register which contains the most significant two bits of the ADC result. The third entry of the write buffer array is the last eight bits of the digitized sequence. These are the contents of the ADRESL register which contains the last eight bits of the ADC result.

Once the write buffer array is filled with the data, a HIDwrite function is called to send this data to the PC. The HIDwrite function sends each entry of the write buffer array to the PC. The HIDWrite function is called inside a loop which sends all the entries sequentially to the PC's application where a ReadBuffer stores all these entries for further processing.

The entries of Write buffer are updated repeatedly at a frequency many times larger than the frequency of arm movement so there are no delays in relaying the movement to the PC hence there is no need for a queue.

3.6 Receiving End

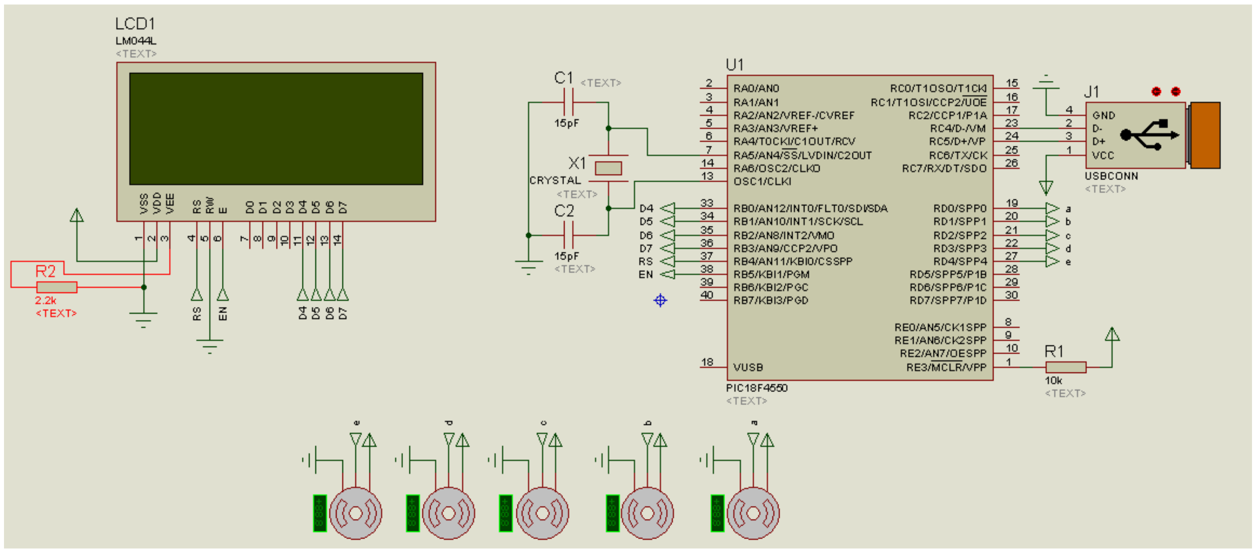


Figure 3-10: Receiver Schematic

3.6.1 USB Module

On the receiving end, the data is retrieved from the PC using the PIC microcontroller. The vendor ID (VID) for our transmitter device is 1234 and the product ID (PID) is 0001.

Here the project uses the read buffer array for storing the data. There are three entries of the read buffer. The first entry is the channel number, which is used to identify the motor for which the PWM signal is intended. The second entry is the two most significant bits of the ADC result. The third entry is the last eight bits of the ADC result.

When the data is received, the second and third entries are concatenated and the used as one value while the first entry decides which motor to drive with the PWM. The PWM generated drives the servo motor.

3.6.2 Operation of Servo Motor

A servomotor (servo) is an electromechanical device in which an electrical signal determines the angle of the servo's shaft. They are widely used in toys, airplanes as well as robots. Figure 3-11 shows servo motor.

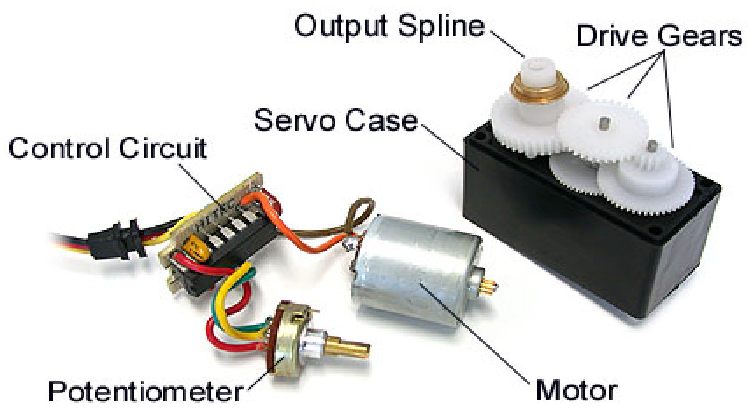


Figure 3-11: Internal Structure of a Servo Motor

Servo motor has three inputs. One of the wires is ground, one for high power (5V) and one for PWM control signal.   Figure 4.8 shows pin out of servo.

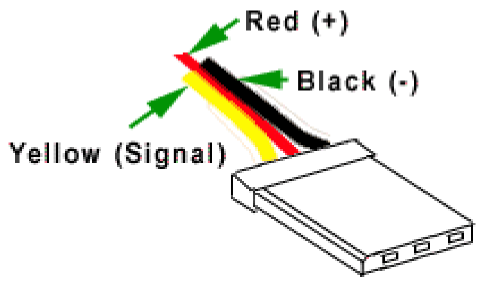


Figure 3-3: Pin Out of Servo Motor

The power and ground wires are hooked directly up to whatever battery or power supply you are using to power the servos. The signal wire will be hooked up to the microcontroller used to control the servo, in our case the PIC18f4550. A noticeable first impression, the servo only requires 1 pin from the PIC. The signal that we need to create in order to control the servos is called a Pulse Width Modulation signal or PWM for short. An example is shown in Figure 3-13.

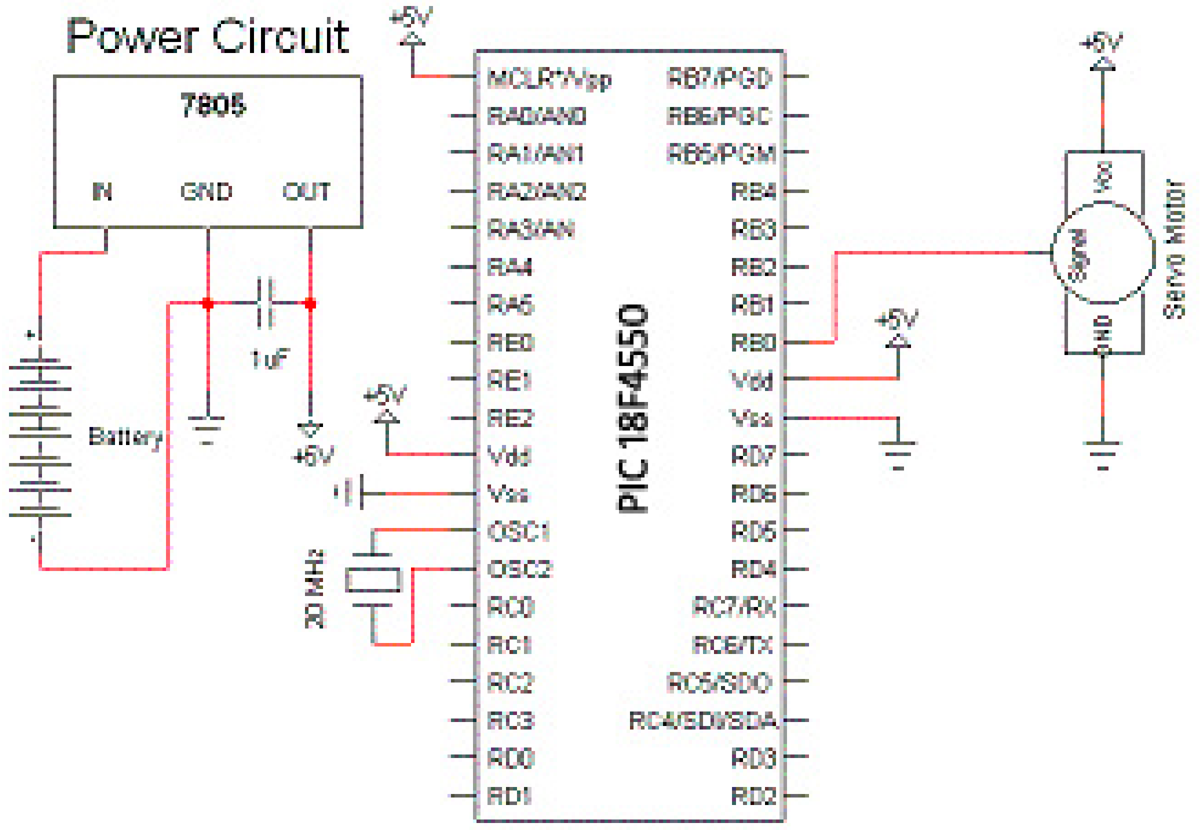


Figure 3-43: Servo Motor Connected to PIC18F4550

The regulator is used for regulating the voltage so that the circuit can operate with a standard 9 volt battery.

3.6.3 Servo Control

As shown by the figure 3-12, the yellow wire of the servo motor needs to be connected with the microcontroller for the control signals. For the servo motors the control signals are PWM signals.

Servos come in different brands and their overall pulse structure for control may vary but almost all the brands have a neutral position at 1.5ms for the on time.

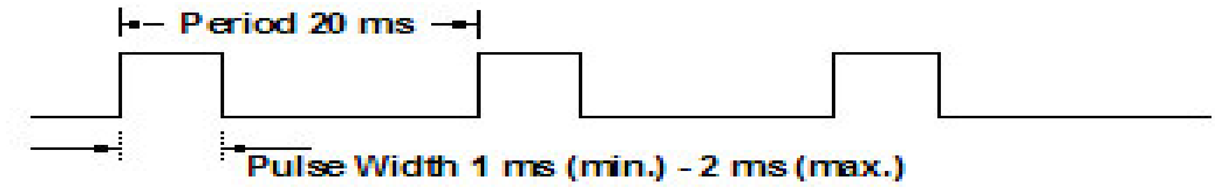


Figure 3-54: PWM Signal

The angle is a result of the pulse provided on the control pin. This is called Pulse width Modulation. The servo expects to see a pulse every 20 ms. The width of the pulse will decide how far the motor turns in terms of angle. For example, a 1.5 ms wide pulse will turn the motor to the 90 degree position (neutral position) from both ends.

When servo motors are signaled to move they will turn to that angle and hold that position. While the servo is holding a specific angle, the servo will resist from moving out of that position if an external force pushes against the servo.

When the pulse width of the control signal sent to a servo is less than 1.5 ms the servo rotates to an angle and holds its output shaft some number of degrees anticlockwise from the neutral point (90 degree reference). When the pulse width is greater than 1.5 ms the reciprocal occurs. In the same way, varying the pulse width from minimum value to the neutral value will cause the motor to move its shaft from zero to 90 degree. The same is true when the pulse width is varied from neutral position value to the maximum value. Here it must be mentioned that different servos from different or even the same brands have different maximum and minimum on time for the pulse width. The servos we use have a minimum on time of 600us and a maximum on time of 2400us.

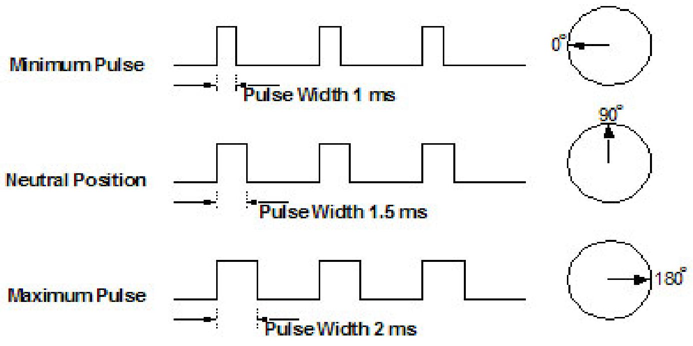
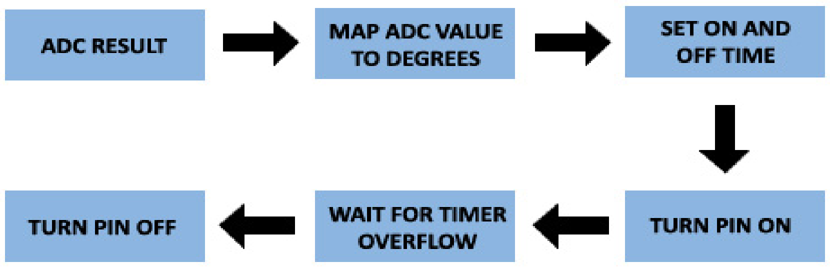


Figure 3-65: Variation of Servo Motor With Variation in Width of Servo-Motor

The servos we have used are from HiTec and have a maximum pulse high time of 2400us and a minimum high time of 600us.

3.6.4 PWM Generation

We have used timers for the purpose of generation of PWM signals. The algorithm that we have devised for this purpose is explained by figure 3-16



.

Figure 3-16: Algorithm for generation of PWM signals

What happens is that the ADC value is translated into a corresponding PWM signal proportional to the angle of movement of the human arm or in other terms, the resistance change of the potentiometer. In the first step, the ADC result is translated into degrees of movement. As the servos are capable of 180 degree motion, we have a maximum range of motion of 180 degrees for each joint and hence there have to be 180 distinct PWM signals for each motor. Thus for five motors, we have a total of 900 PWM signals.

3.6.5 Our Time Unit (OTU)

We have used a custom time unit named Our Time Unit (OTU) for specifying the high pulse time and the low pulse time. Its relation to second (s) is described as follows:

There are 1024 levels of the ADC hence 1024 different values can represent the ADC result. The maximum voltage at any potentiometer is 5V. This means that the factor which has to be multiplied to any digital sequence to convert it back to analog equivalent is:

5 / 1024 = 0.004883

0.004883 x 1000 = 4.883 (milivolt scale)

As the pulse high time is from 600us to 2400us, the total pulse high time is:

Total pulse high time = 2400 – 600 = 1800us

Now, the minimum delay generated by the timers is 50us, this implies that in terms of 50us, the pulse high time is:

1800/50 = 36

As we have used 5V as maximum voltage (5000mV) the mapping factor for servos and ADC result would be:

Servo mapping factor = 36 / 5000 = 0.0072

The equivalent of the above steps can be summarized into one factor as:

0.0072 * 4.883 = 0. 0351576

The resultant number is multiplied to the ADC result and then 12 are added to it. The number 12 is added because the result of the above process is 12 less than the required value for the PWM high time.

The result of the above calculation is in OTU. The minimum delay that we have used for PWM generation is 50us. The product of OTU with 50us gives the exact pulse high time.

Pulse low time is given as follows:

Pulse low time = 20000us – Pulse high time

3.7 Timer

The project uses TIMER0 of PIC18F4550 for generation of PWM signals. The control register of TMR0 is TCON0 register.

TCON0 register consists of the timer settings which allow the timer to be initialized properly by setting the prescaler value, the clock source, the mode of the timer, the interrupt mode and the initial values of the timer itself.

There are three registers of TMR0 TCON0 register (has the control parameters), TMR0H register (Overflow value most significant bits), TMR0L register (Overflow value least significant bits). The timer has been used to generate a delay in this project. It counts from a specific value to a specific value specified by the TMR0H and TMR0L registers. When the count reaches the specified value, an interrupt is generated and the counter is reset to the specified value. In this way, timers can be used to generate very precise delays.

3.8 Robotic Arm

The servo motors are embedded in a metallic body which functions as a robotic arm. As the PWM signals are fed to these servo motors, they move to respective angles replicating the movement of the user. The body of the robot is made of aluminum and is very durable. There are five stages of the robot which include the base, the shoulder, the elbow, the wrist and the jaw. In addition, for mounting the sensors or the camera, an additional platfrom is attached to the head of the robotic arm. The base of the robot also provides ample room for any onboard circuitry that may be mounted on the robotic arm.

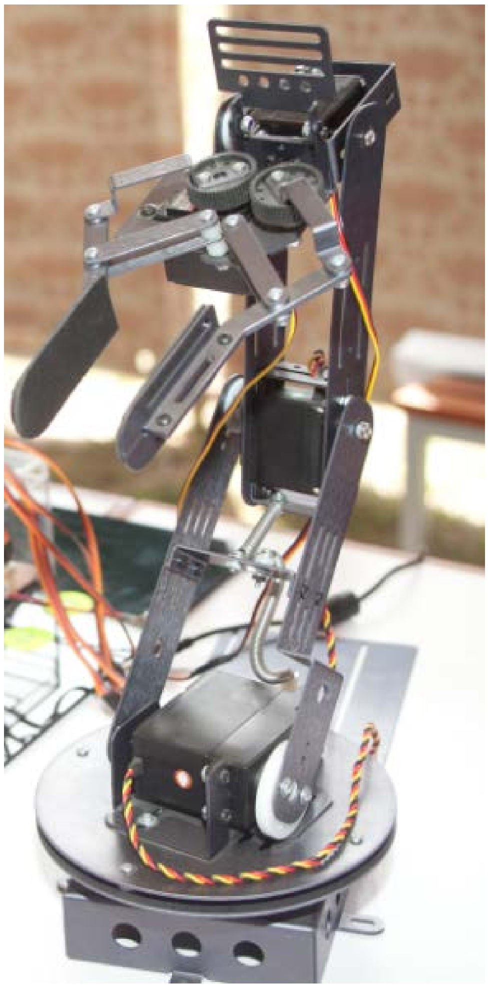


Figure 3-17: The Robotic Arm

3.9 Camera feedback

There is a camera mounted on top of the robotic arm. This camera is connected to the PC and sends feedback in the form of video to the sender so that the user can see where the robotic arm is moving. The project uses a regular webcam for this purpose. The webcam is mounted on a platform above the head of the robotic arm for a clear and wide view of the field.

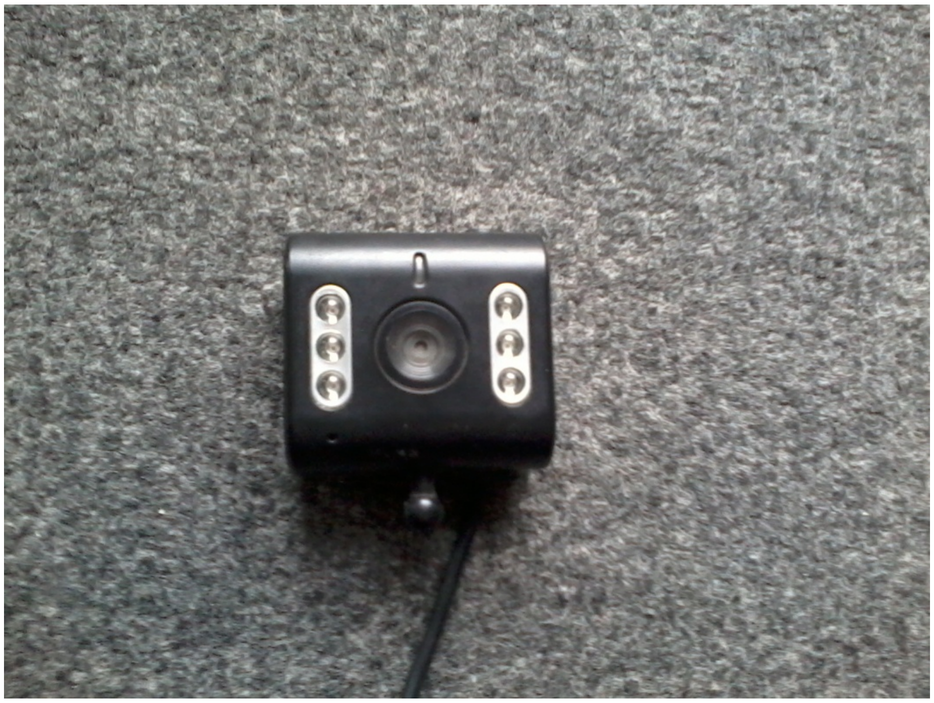


Figure 3-18: Camera Used for Feedback

3.10 Method of communication (computer application)

The project uses a custom made application for data transmission over the internet. A user friendly interface for transmitting the data from the sensor in real time is used to establish link over the internet. This ensures that the range of the system can be extended to any place where internet can be accessed. The language that the project uses for this purpose is Visual Basic.

3.11 Visual Basic NET Introduction

Visual Basic 2008 is one of the best choices for building windows and PC applications. In modern software development, however, the language is just one of the ingredients that are used for build applications. The most important part is the .NET Framework, which the most essential ingredient of any application. It can be viewed as a collection of library functions.

All the functionality of the operating system and its availability to the application is the responsibility of the framework. Two components for building an application are the framework and the language.

The third component in application development is integrated development environment, or IDE. This is the part which speeds up application making by providing a very user friendly environment. Visual Studio 2008 provides all the three components and in addition to this, it also provides methods for manipulating databases. This makes it the ideal tool for application development.

3.12 Socket Introduction

A communications connection endpoint that can be named and addressed in a network is called a socket. The processes that employ the socket can be on the very same system, on different systems or even on different networks. Sockets enable us to exchange information between processes on the same system or across a network, distribute work to the most efficient system and allow access to centralized data easily.

3.13 Characteristics of Sockets

Sockets have many characteristics. Some of them are an integer represents a socket, socket descriptor is that integer, a socket exists as long as the process maintains an open link to the socket. A socket can be named and used it to communicate with other sockets in a communication domain.

3.13.1 How do sockets work?

There is a typical flow of events of the socket. No connection is established by connectionless sockets for data transfer. Rather, the application running on the server specifies its name where a client can send requests**.** User Datagram Protocol (UDP) instead of TCP/IP is used by Connectionless sockets.

The following figure shows client/server relationship from the socket APIs used for the coding of connectionless sockets.

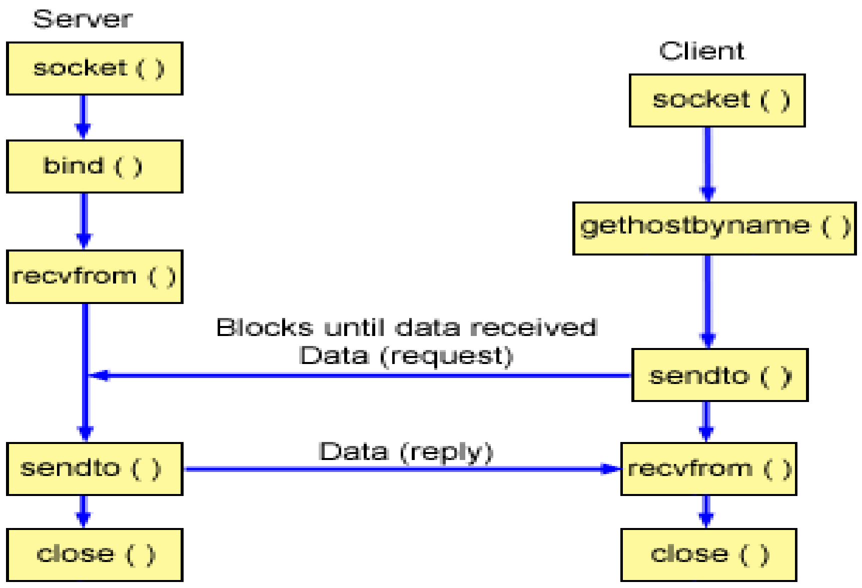


Figure 3-19: Socket Programming Algorithm

3.13.2 Socket flow of events: Connectionless server

The relationship between the server and client application in a connectionless design is described by the socket flow of events. There are usage notes on specific APIs present in each set of flows. The socket() function represents an endpoint by returning a socket descriptor. The usage of INET

(Internet Protocol) address family with the UDP transport (SOCK_DGRAM) is also identified by this statement. After the socket descriptor is created, unique name for the socket is produced by the bind( ) function. The recvfrom( ) function is used by the server for receiving that data. The recvfrom( ) function waits for indefinite time for data to arrive. The sendto( ) function sends the data back to the client. Any open socket descriptors are closed by the close( ) function.

3.13.3 Socket flow of events: Connectionless client:

It uses the sequence of function calls as the socket() function represents an endpoint by returning a socket descriptor. The usage of INET (Internet Protocol) address family with the UDP transport (SOCK_DGRAM) is also identified by this statement. IP address of the server is retrieved by the gethostbyname() function. For sending the data to the server the sendto() function is used. For receiving the data back from the server the recvfrom() function is used. For any open socket descriptors to be ended the close() function is used.

3.14 UDP Socket Programming Flowchart

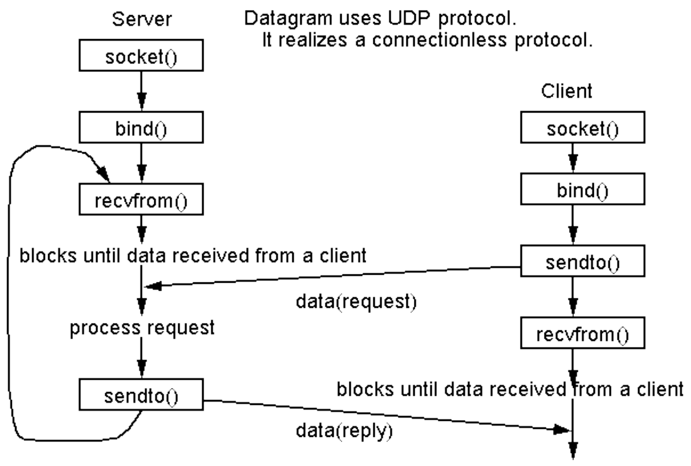

Server
Datagram uses UDP protocol.
It realizes a connectionless protocol.
socket()
bind()
recvfrom()
blocks until data received from a client
process request
sendto()
Client
socket()
bind()
sendto()
data(request)
recvfrom()
blocks until data received from a client
data(reply)

Figure 3-20: UDP Socket Programming Flowchart

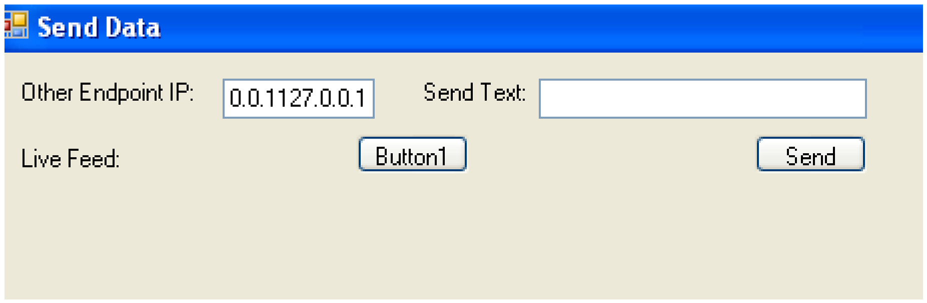


Figure 3-21: Screenshot of the Transmitter Application

Figure 24 shows a screenshot of the transmitter application. The other point IP field has the IP address of the receiving end and live feed is used to monitor the receiver video.

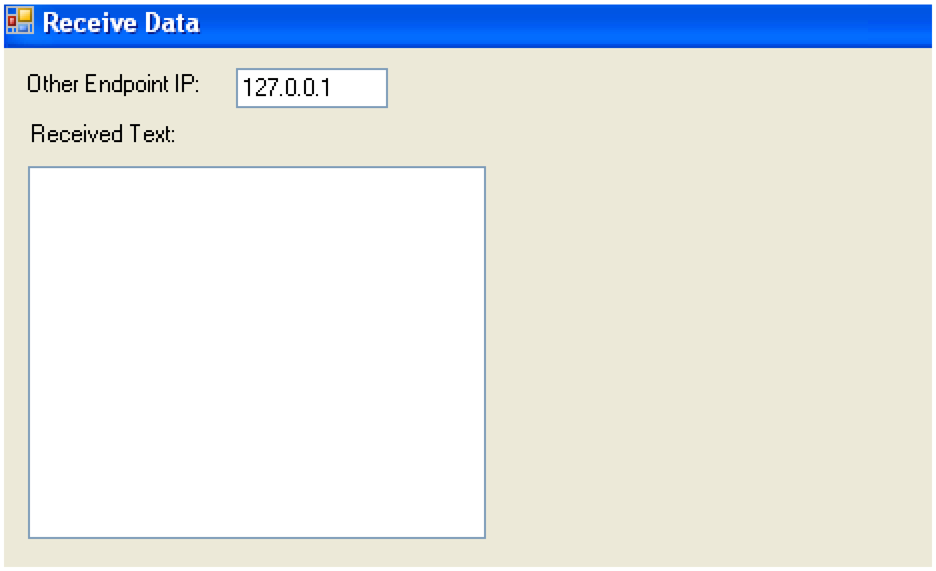


Figure 3-22: Screenshot of the Receiver Application

The receiver application is used for sending the camera feedback and also for receiving the data from the transmitter end.

CHAPTER 4

FUTURE ENHANCEMENTS AND APPLICATIONS

4.1 Development in this system

There is much development that can be made in this system. Some of the proposed improvements are mounting the system on a movable platform and making the system independent of the receiver computer by integrating a Wifi module to the receiver end.

4.2 Applications of system

The end product can be designed to perform any desired task such as remote bomb disposal unit, can be effectively used as part of bomb disposal unit. While sitting at back end using web cam on the robotic arm it can be send to diffuse the bomb. Instead of sending the person on the site a robot can be used for this purpose, remote assistance and trouble shooting, security at check points, medicine, utilization of skilled workforce over the internet for an economy boost and handling of dangerous material.

APPENDIX A

Code for microcontroller for the transmitter module:

```
#ifndef __LCDCONFIG_H

#define __LCDCONFIG_H

sbit LCD_RS at RB4_bit;

sbit LCD_EN at RB5_bit;

sbit LCD_D4 at RB0_bit;

sbit LCD_D5 at RB1_bit;

sbit LCD_D6 at RB2_bit;

sbit LCD_D7 at RB3_bit;


sbit LCD_RS_Direction at TRISB4_bit;

sbit LCD_EN_Direction at TRISB5_bit;

sbit LCD_D4_Direction at TRISB0_bit;

sbit LCD_D5_Direction at TRISB1_bit;

sbit LCD_D6_Direction at TRISB2_bit;

sbit LCD_D7_Direction at TRISB3_bit;

#endif


#include "lcdConfig.h"

unsigned char readbuff[2] absolute 0x500;

unsigned char writebuff[2] absolute 0x540;

unsigned int i = 0;

unsigned int adcResult1 = 0;

unsigned char adcResult1_high = 0;

unsigned char adcResult1_low = 0;

unsigned char adcResult1_new = 0;

unsigned char adcResult2_high = 0;

unsigned char adcResult2_low = 0;
unsigned char adcResult3_high = 0;
unsigned char adcResult3_low = 0;
unsigned char adcResult4_high = 0;
unsigned char adcResult4_low = 0;
unsigned char adcResult5_high = 0;
unsigned char adcResult5_low = 0;
unsigned int adcResult2 = 0;
unsigned int adcResult3 = 0;
unsigned int adcResult4 = 0;
unsigned int adcResult5 = 0;
unsigned int adcResult2_new = 0;
unsigned int adcResult3_new = 0;
unsigned int adcResult4_new = 0;
unsigned int adcResult5_new = 0;

void interrupt()
{
     if(USBIF_bit)
     {
       USBIF_bit = 0;
       USB_Interrupt_Proc();
     }
}
void adcInit(void) {
    ADCON1  = 0X0A;
    ADCON0  = 0X10 LCD)
    ADCON2.ADFM  = 1;
    ADCON2       = 0b10001010;
    ADCON0.ADON  = 1;
}
```

```
void init(){
    CMCON   = 0x07;
    TRISA   = 0xFF;
    TRISB   = 0x00;
    TRISD   = 0x00;
    TRISE   = 0;
    PORTD   = 0;
    PORTE   = 0;
    adcInit();
    Lcd_Init();
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Cmd(_LCD_CURSOR_OFF);
    Lcd_Out(1, 1, "Power ON");
    Delay_ms(1000);
}

void main(void){
  init();
  RB6_bit = 1;
  HID_Enable(&readbuff,&writebuff);
  while(1) {

       Lcd_Cmd(_LCD_CLEAR);

       ADCON0.GO    = 1;
       while(ADCON0.DONE);

       adcResult1    = ADC_Get_Sample(0);
```

```
adcResult2    = ADC_Get_Sample(1);
adcResult3    = ADC_Get_Sample(2);
adcResult4    = ADC_Get_Sample(3);
adcResult5    = ADC_Get_Sample(4);

if (adcResult1 != adcResult1_new){
   Lcd_Out(1,1,"the value is new 1");
   Delay_ms(200);
   adcResult1_low = ADRESL;
   adcResult1_high = ADRESH;
   writebuff[0] = 0x01;
   writebuff[1] = adcResult1_low;
   writebuff[2] = adcResult1_high;
   adcResult1_new = adcResult1;
   HID_Write(&writebuff,2);
}

if (adcResult2 != adcResult2_new){
   Lcd_Out(1,1,"the value is new 2");
   Delay_ms(200);
   writebuff[0] = 0x02;
   writebuff[1] = adcResult2_low;
   writebuff[2] = adcResult3_high;
   adcResult2_new = adcResult2;
   while(!HID_Write(&writebuff,2));
}

if (adcResult3 != adcResult3_new){
   Lcd_Out(1,1,"the value is new 3");
   Delay_ms(200);
```

```
    writebuff[0] = 0x03;
    writebuff[1] = adcResult3_low;
    writebuff[2] = adcResult3_high;
    adcResult3_new = adcResult3;
    while(!HID_Write(&writebuff,2));
  }

  if (adcResult4 != adcResult4_new){
    Lcd_Out(1,1,"the value is new 4");
    Delay_ms(200);
    writebuff[0] = 0x04;
    writebuff[1] = adcResult4_low;
    writebuff[2] = adcResult4_high;
    adcResult4_new = adcResult4;
    while(!HID_Write(&writebuff,2));
  }

  if (adcResult5 != adcResult5_new){
    Lcd_Out(1,1,"the value is new 5");
    Delay_ms(200);
    writebuff[0] = 0x05;
    writebuff[1] = adcResult5_low;
    writebuff[2] = adcResult5_high;
    adcResult5_new = adcResult5;
    while(!HID_Write(&writebuff,2));
  }
}
```

APPENDIX B

Code for the microcontroller of receiver end:

```
at RB0_bit;

#ifndef __LCDCONFIG_H

#define __LCDCONFIG_H

sbit LCD_RS at RB4_bit;

sbit LCD_EN at RB5_bit;

sbit LCD_D4

sbit LCD_D5 at RB1_bit;

sbit LCD_D6 at RB2_bit;

sbit LCD_D7 at RB3_bit;


sbit LCD_RS_Direction at TRISB4_bit;

sbit LCD_EN_Direction at TRISB5_bit;

sbit LCD_D4_Direction at TRISB0_bit;

sbit LCD_D5_Direction at TRISB1_bit;

sbit LCD_D6_Direction at TRISB2_bit;

sbit LCD_D7_Direction at TRISB3_bit;

#endif



#include "lcdConfig.h"

unsigned char readbuff[2] absolute 0x500;

unsigned char writebuff[2]  absolute 0x540;


unsigned int currentTime1 = 0;

unsigned int currentTime2 = 0;

unsigned int currentTime3 = 0;

unsigned int currentTime4 = 0;
unsigned int currentTime5 = 0;

unsigned int pulsePeriod = 0;

unsigned int onTime1     = 0;

unsigned int onTime2     = 0;

unsigned int onTime3     = 0;

unsigned int onTime4     = 0;

unsigned int onTime5     = 0;


void interrupt(){

    if(T0IF_bit){

        currentTime1++;

        currentTime2++;

        currentTime3++;

        currentTime4++;

        currentTime5++;


        if(currentTime1 >= onTime1)

            RD0_bit = 0;

        if(currentTime2 >= onTime2)

            RD1_bit = 0;

        if(currentTime3 >= onTime3)

            RD2_bit = 0;

        if(currentTime4 >= onTime4)

            RD3_bit = 0;

        if(currentTime5 >= onTime5)

            RD4_bit = 0;


        if(currentTime1 >= pulsePeriod){

            RD0_bit = 1;

            currentTime1 = 0;
        }
        if(currentTime2 >= pulsePeriod){
            RD1_bit = 1;
            currentTime2 = 0;
        }
        if(currentTime3 >= pulsePeriod){
            RD2_bit = 1;
            currentTime3 = 0;
        }
        if(currentTime4 >= pulsePeriod){
            RD3_bit = 1;
            currentTime4 = 0;
        }
        if(currentTime5 >= pulsePeriod){
            RD4_bit = 1;
            currentTime5 = 0;
        }

        TMR0H = 0xFF;
        TMR0L = 0x22;
        T0IF_bit = 0;
    }
}

void interrupt1(){
    USB_Interrupt_Proc();
}
void initTimer0(){
      T0CON.TMR0ON = 1;
      T0CON.T08BIT = 0;
      T0CON.T0CS   = 0;
      T0CON.T0SE   = 0;
      T0CON.PSA    = 1;
      T0CON.T0PS2  = 0;
      T0CON.T0PS1  = 0;
      T0CON.T0PS0  = 0;
      TMR0H = 0xFD;
      TMR0L = 0xA8;
}
void init(){
    TRISD = 0;
    RD0_bit = 0;
    TRISB = 0;
    adcInit();
    initTimer0();
    T0IE_bit = 1;
    PEIE_bit = 1;
    GIE_bit  = 1;
    Lcd_Init();
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Cmd(_LCD_CURSOR_OFF);
    Lcd_Out(1, 1, "Power ON");
    HID_Enable(&readbuff, &writebuff);
}
void main() {
    float input1 = 0;
    float input2 = 0;
    float input3 = 0;
    float input4 = 0;
    float input5 = 0;
```

```
int   input  = 0;


unsigned int pot1 = 0;

unsigned int pot2 = 0;

unsigned int pot3 = 0;

unsigned int pot4 = 0;

unsigned int pot5 = 0;



currentTime1 = 0;

currentTime2 = 0;

pulsePeriod = 400;

onTime1 = 30;

onTime2 = 30;

onTime3 = 30;

onTime4 = 30;

onTime5 = 30;


init();

while(1){


    while(HID_Read())

    {

    writebuff[1] = inputL;

    writebuff[2] = inputH;

    }


    if (writebuff[0] == 0x01)

    input1 = input;
    if (writebuff[0] == 0x02)
    input2 = input;

    if (writebuff[0] == 0x03)
    input3 = input;

    if (writebuff[0] == 0x04)
    input4 = input;

    if (writebuff[0] == 0x05)
    input5 = input;



    input1 *= 0.0351576;
    input2 *= 0.0351576;
    input3 *= 0.0351576;
    input4 *= 0.0351576;
    input5 *= 0.0351576;

    pot1 = ceil(input1) + 12;
    pot2 = ceil(input2) + 12;
    pot3 = ceil(input3) + 12;
    pot4 = ceil(input4) + 12;
    pot5 = ceil(input5) + 12;

    onTime1 = pot1;
    onTime2 = pot2;
    onTime3 = pot3;
    onTime4 = pot4;
    onTime5 = pot5; }}
```

APPENDIX C

Code for UDP socket:

```
' This is a Buffer class incorperating the UDP transport protocol from system.net
' i used a timer to check the socket for available data..

Imports System.Net
Imports System.Text

Public Class UDPSocket_Buffer
    Private UDP_Sock As Sockets.UdpClient
    Private WithEvents C_Timer As System.Windows.Forms.Timer
    Private WithEvents I_Timer As System.Windows.Forms.Timer
    ' Thanks to ChaosTheEternal From CodeGuru for sorting out the Timer Threading problem..
    'Private WithEvents t_Timer As Timers.Timer - In IDE Starts a new processor thread
    Private B_Data() As String
    Private B_Sender() As IPAddress
    Private RecievedBytes() As Byte
    Private RemoteIpEndPoint As New IPEndPoint(IPAddress.IPv6Any, 0)
    Private Tot_Array As Integer
    Private Tmp_loop_1 As Integer
    Public Event Data_Arrival(ByVal Data As String, ByVal Source As IPAddress)
    Public Event Socket_Error(ByVal Err As Exception)

    Public Sub New()
        C_Timer = New System.Windows.Forms.Timer
        I_Timer = New System.Windows.Forms.Timer
        C_Timer.Interval = 100
        I_Timer.Interval = 1000
        ReDim B_Data(0)
        ReDim B_Sender(0)
    End Sub

    Public Function Init(ByVal Port As Integer) As Boolean
        Try
            UDP_Sock = New Sockets.UdpClient(Port)
            UDP_Sock.EnableBroadcast = True
            C_Timer.Enabled = True
            Init = True
        Catch ex As Exception
            Init = False
            RaiseEvent Socket_Error(ex)
        End Try
    End Function
    Public Function Close() As Boolean
        Try
            UDP_Sock.Close()
            UDP_Sock = Nothing
            C_Timer.Enabled = False
            Close = True
        Catch ex As Exception
            Close = False
        End Try
    End Function
```

```
Public Function Send(ByVal Data As String, ByVal Address As IPEndPoint) As Boolean
    Try
        UDP_Sock.Send(Encoding.ASCII.GetBytes(Data), Len(Data), Address)
        Send = True
    Catch ex As Exception
        Send = False
        RaiseEvent Socket_Error(ex)
    End Try
End Function

Private Sub C_Timer_Elapsed(ByVal sender As Object, ByVal e As System.EventArgs) Handles C_Timer.Tick
    If UDP_Sock.Available > 0 Then
        Try
            RecievedBytes = UDP_Sock.Receive(RemoteIpEndPoint)
            Tot_Array = UBound(B_Data) + 1
            ReDim B_Data(Tot_Array)
            ReDim B_Sender(Tot_Array)
            B_Data(Tot_Array) = Trim(Encoding.ASCII.GetString(RecievedBytes))
            B_Sender(Tot_Array) = RemoteIpEndPoint.Address
            I_Timer.Enabled = True
        Catch ex As Exception
            RaiseEvent Socket_Error(ex)
        End Try
    End If
End Sub

Private Sub I_Timer_Elapsed(ByVal sender As Object, ByVal e As System.EventArgs) Handles I_Timer.Tick
    Tot_Array = UBound(B_Data)
    If Tot_Array > 0 Then
        For Tmp_loop_1 = 0 To Tot_Array - 1
            B_Data(Tmp_loop_1) = B_Data(Tmp_loop_1 + 1)
            B_Sender(Tmp_loop_1) = B_Sender(Tmp_loop_1 + 1)
        Next
        ReDim Preserve B_Data(Tot_Array - 1)
        ReDim Preserve B_Sender(Tot_Array - 1)
        If B_Data(0) <> "" Then
            RaiseEvent Data_Arrival(B_Data(0), B_Sender(0))
        End If
    Else
        I_Timer.Enabled = False
    End If
End Sub

Protected Overrides Sub Finalize()
    Try
        UDP_Sock.Close()
        UDP_Sock = Nothing
    Catch ex As Exception
        ' Socket already closed or never initiated..
        ' since we closing we can ignore this error..
    End Try
    C_Timer = Nothing
    I_Timer = Nothing
    MyBase.Finalize()
End Sub
End Class
```

Code for UDP sender :

```
Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class Sender
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.

    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.IPBox = New System.Windows.Forms.TextBox()
        Me.PictureBox1 = New System.Windows.Forms.PictureBox()
        Me.Label1 = New System.Windows.Forms.Label()
        Me.connectLabel = New System.Windows.Forms.Label()
        Me.Label2 = New System.Windows.Forms.Label()
        Me.ListBox1 = New System.Windows.Forms.ListBox()
        Me.Label3 = New System.Windows.Forms.Label()
        Me.Button1 = New System.Windows.Forms.Button()
        Me.clearBtn = New System.Windows.Forms.Button()
        Me.Label4 = New System.Windows.Forms.Label()
        Me.sendBox = New System.Windows.Forms.TextBox()
        Me.sendBtn = New System.Windows.Forms.Button()
        CType(Me.PictureBox1, System.ComponentModel.ISupportInitialize).BeginInit()
        Me.SuspendLayout()
        '
        'IPBox
        '
        Me.IPBox.Location = New System.Drawing.Point(152, 16)
        Me.IPBox.Name = "IPBox"
        Me.IPBox.Size = New System.Drawing.Size(100, 22)
        Me.IPBox.TabIndex = 0
        Me.IPBox.Text = "127.0.0.1"
        '
        'PictureBox1
        '
        Me.PictureBox1.Location = New System.Drawing.Point(16, 82)
        Me.PictureBox1.Name = "PictureBox1"
Me.PictureBox1.Size = New System.Drawing.Size(300, 245)
Me.PictureBox1.SizeMode = System.Windows.Forms.PictureBoxSizeMode.AutoSize
Me.PictureBox1.TabIndex = 3
Me.PictureBox1.TabStop = False
'
'Label1
'
Me.Label1.AutoSize = True
Me.Label1.Location = New System.Drawing.Point(13, 16)
Me.Label1.Name = "Label1"
Me.Label1.Size = New System.Drawing.Size(124, 17)
Me.Label1.TabIndex = 4
Me.Label1.Text = "Other Endpoint IP:"
'
'connectLabel
'
Me.connectLabel.AutoSize = True
Me.connectLabel.Location = New System.Drawing.Point(281, 16)
Me.connectLabel.Name = "connectLabel"
Me.connectLabel.Size = New System.Drawing.Size(0, 17)
Me.connectLabel.TabIndex = 5
'
'Label2
'
Me.Label2.AutoSize = True
Me.Label2.Location = New System.Drawing.Point(360, 56)
Me.Label2.Name = "Label2"
Me.Label2.Size = New System.Drawing.Size(75, 17)
Me.Label2.TabIndex = 8
Me.Label2.Text = "Sent Data:"
Me.Label2.Visible = False
'
'ListBox1
'
Me.ListBox1.FormattingEnabled = True
Me.ListBox1.ItemHeight = 16
Me.ListBox1.Location = New System.Drawing.Point(363, 82)
Me.ListBox1.Name = "ListBox1"
Me.ListBox1.Size = New System.Drawing.Size(120, 228)
Me.ListBox1.TabIndex = 7
Me.ListBox1.Visible = False
'
'Label3
'
Me.Label3.AutoSize = True
Me.Label3.Location = New System.Drawing.Point(13, 56)
Me.Label3.Name = "Label3"
Me.Label3.Size = New System.Drawing.Size(74, 17)
Me.Label3.TabIndex = 9
Me.Label3.Text = "Live Feed:"
'
'Button1
'
Me.Button1.Location = New System.Drawing.Point(241, 50)
Me.Button1.Name = "Button1"
Me.Button1.Size = New System.Drawing.Size(75, 23)
Me.Button1.TabIndex = 10
Me.Button1.Text = "Button1"
Me.Button1.UseVisualStyleBackColor = True
'
'clearBtn
'
Me.clearBtn.Location = New System.Drawing.Point(363, 316)
Me.clearBtn.Name = "clearBtn"
Me.clearBtn.Size = New System.Drawing.Size(75, 23)
Me.clearBtn.TabIndex = 11
Me.clearBtn.Text = "Clear"
Me.clearBtn.UseVisualStyleBackColor = True
Me.clearBtn.Visible = False
'
'Label4
'
Me.Label4.AutoSize = True
Me.Label4.Location = New System.Drawing.Point(281, 16)
Me.Label4.Name = "Label4"
Me.Label4.Size = New System.Drawing.Size(76, 17)
Me.Label4.TabIndex = 13
Me.Label4.Text = "Send Text:"
'
'sendBox
'
Me.sendBox.Location = New System.Drawing.Point(363, 16)
Me.sendBox.Name = "sendBox"
Me.sendBox.Size = New System.Drawing.Size(218, 22)
Me.sendBox.TabIndex = 12
'
'sendBtn
'
Me.sendBtn.Location = New System.Drawing.Point(506, 50)
Me.sendBtn.Name = "sendBtn"
Me.sendBtn.Size = New System.Drawing.Size(75, 23)
Me.sendBtn.TabIndex = 14
Me.sendBtn.Text = "Send"
Me.sendBtn.UseVisualStyleBackColor = True
'
'Sender
'
Me.AutoScaleDimensions = New System.Drawing.SizeF(8.0!, 16.0!)
Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
Me.ClientSize = New System.Drawing.Size(605, 366)
Me.Controls.Add(Me.sendBtn)
Me.Controls.Add(Me.Label4)
Me.Controls.Add(Me.sendBox)
Me.Controls.Add(Me.clearBtn)
Me.Controls.Add(Me.Button1)
Me.Controls.Add(Me.Label3)
Me.Controls.Add(Me.Label2)
Me.Controls.Add(Me.ListBox1)
Me.Controls.Add(Me.connectLabel)
Me.Controls.Add(Me.Label1)
Me.Controls.Add(Me.PictureBox1)
Me.Controls.Add(Me.IPBox)
```

```
        Me.Name = "Sender"
        Me.Text = "Send Data"
        CType(Me.PictureBox1, System.ComponentModel.ISupportInitialize).EndInit()
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub
    Friend WithEvents IPBox As System.Windows.Forms.TextBox
    Friend WithEvents PictureBox1 As System.Windows.Forms.PictureBox
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents connectLabel As System.Windows.Forms.Label
    Friend WithEvents Label2 As System.Windows.Forms.Label
    Friend WithEvents ListBox1 As System.Windows.Forms.ListBox
    Friend WithEvents Label3 As System.Windows.Forms.Label
    Friend WithEvents Button1 As System.Windows.Forms.Button
    Friend WithEvents clearBtn As System.Windows.Forms.Button
    Friend WithEvents Label4 As System.Windows.Forms.Label
    Friend WithEvents sendBox As System.Windows.Forms.TextBox
    Friend WithEvents sendBtn As System.Windows.Forms.Button

End Class
```

APPENDIX D

Code for UDP receiver:

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class Receiver
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.

    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.IPBox = New System.Windows.Forms.TextBox()
        Me.Label1 = New System.Windows.Forms.Label()
        Me.connectLabel = New System.Windows.Forms.Label()
        Me.ListBox1 = New System.Windows.Forms.ListBox()
        Me.Label2 = New System.Windows.Forms.Label()
        Me.clearBtn = New System.Windows.Forms.Button()
        Me.ListBox2 = New System.Windows.Forms.ListBox()
        Me.Label3 = New System.Windows.Forms.Label()
        Me.SuspendLayout()
        '
        'IPBox
        '
        Me.IPBox.Location = New System.Drawing.Point(157, 12)
        Me.IPBox.Name = "IPBox"
        Me.IPBox.Size = New System.Drawing.Size(100, 22)
        Me.IPBox.TabIndex = 2
        Me.IPBox.Text = "127.0.0.1"
        '
        'Label1
        '
        Me.Label1.AutoSize = True
        Me.Label1.Location = New System.Drawing.Point(13, 13)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(124, 17)
        Me.Label1.TabIndex = 3
        Me.Label1.Text = "Other Endpoint IP:"
        '
        'connectLabel
        '
        Me.connectLabel.AutoSize = True
        Me.connectLabel.Location = New System.Drawing.Point(16, 44)
        Me.connectLabel.Name = "connectLabel"
        Me.connectLabel.Size = New System.Drawing.Size(0, 17)
        Me.connectLabel.TabIndex = 4
        '
        'ListBox1
        '
        Me.ListBox1.FormattingEnabled = True
        Me.ListBox1.ItemHeight = 16
        Me.ListBox1.Location = New System.Drawing.Point(343, 73)
        Me.ListBox1.Name = "ListBox1"
        Me.ListBox1.Size = New System.Drawing.Size(120, 228)
        Me.ListBox1.TabIndex = 5
        Me.ListBox1.Visible = False
        '
        'Label2
        '
        Me.Label2.AutoSize = True
        Me.Label2.Location = New System.Drawing.Point(340, 44)
        Me.Label2.Name = "Label2"
        Me.Label2.Size = New System.Drawing.Size(105, 17)
        Me.Label2.TabIndex = 6
        Me.Label2.Text = "Received Data:"
        Me.Label2.Visible = False
        '
        'clearBtn
        '
        Me.clearBtn.Location = New System.Drawing.Point(346, 304)
        Me.clearBtn.Name = "clearBtn"
        Me.clearBtn.Size = New System.Drawing.Size(75, 23)
        Me.clearBtn.TabIndex = 7
        Me.clearBtn.Text = "Clear"
        Me.clearBtn.UseVisualStyleBackColor = True
        Me.clearBtn.Visible = False
        '
        'ListBox2
        '
        Me.ListBox2.FormattingEnabled = True
        Me.ListBox2.ItemHeight = 16
        Me.ListBox2.Location = New System.Drawing.Point(19, 73)
        Me.ListBox2.Name = "ListBox2"
        Me.ListBox2.Size = New System.Drawing.Size(304, 228)
        Me.ListBox2.TabIndex = 8
        '
        'Label3
        '
        Me.Label3.AutoSize = True
        Me.Label3.Location = New System.Drawing.Point(16, 44)
        Me.Label3.Name = "Label3"
        Me.Label3.Size = New System.Drawing.Size(102, 17)
        Me.Label3.TabIndex = 9
```

```
        Me.Label3.Text = "Received Text:"
        '
        'Receiver
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(8.0!, 16.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(489, 339)
        Me.Controls.Add(Me.Label3)
        Me.Controls.Add(Me.ListBox2)
        Me.Controls.Add(Me.clearBtn)
        Me.Controls.Add(Me.Label2)
        Me.Controls.Add(Me.ListBox1)
        Me.Controls.Add(Me.connectLabel)
        Me.Controls.Add(Me.Label1)
        Me.Controls.Add(Me.IPBox)
        Me.Name = "Receiver"
        Me.Text = "Receive Data"
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub
    Friend WithEvents IPBox As System.Windows.Forms.TextBox
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents connectLabel As System.Windows.Forms.Label
    Friend WithEvents ListBox1 As System.Windows.Forms.ListBox
    Friend WithEvents Label2 As System.Windows.Forms.Label
    Friend WithEvents clearBtn As System.Windows.Forms.Button
    Friend WithEvents ListBox2 As System.Windows.Forms.ListBox
    Friend WithEvents Label3 As System.Windows.Forms.Label

End Class
```

Code for General Single UDP socket :

```
Imports System.Net.Sockets
Imports System.Text
Imports System.Net.EndPoint
Imports System.Net

Module Module1

    Sub Main()
        ' This constructor arbitrarily assigns the local port number.
        Dim udpClient As New UdpClient(11000)
        Try
            udpClient.Connect("192.168.137.20", 11000)

            ' Sends a message to the host to which you have connected.
            Dim sendBytes As [Byte]() = Encoding.ASCII.GetBytes("hello?")

            udpClient.Send(sendBytes, sendBytes.Length)

            Console.ReadKey()
            ' IPEndPoint object will allow us to read datagrams sent from any source.
            Dim RemoteIpEndPoint As New IPEndPoint(IPAddress.Any, 0)

            ' UdpClient.Receive blocks until a message is received from a remote host.
            Dim receiveBytes As [Byte]() = udpClient.Receive(RemoteIpEndPoint)
            Dim returnData As String = Encoding.ASCII.GetString(receiveBytes)

            ' Which one of these two hosts responded?


            Console.WriteLine(("This is the message you received " + _
                                     returnData.ToString()))
            Console.WriteLine(("This message was sent from " + _
                                    RemoteIpEndPoint.Address.ToString() + _
                                    " on their port number " + _
                                    RemoteIpEndPoint.Port.ToString()))
            udpClient.Close()


        Catch e As Exception
            Console.WriteLine(e.ToString())
        End Try
        Console.ReadKey()

    End Sub


End Module
```

Bibliography

[1] Carlos A. Acosta Claderon, Housheng Hu, *Robot imitation from human body movements*, 2005.

[2] MargaChiri, *Joystick control for TinyOS* , 2002.

[3] Ali Mazidi, *Programming PIC18F in C and assembly*, Second Edition. Jonh Wiley & Sons, 2005.