"CONTROL OF ROBOTIC ARM AND VIRTUAL OBJECT THROUGH HAPTIC TRANSDUCERS"

By

Kanika Rana(101049)

Under the supervision of

Prof. T.S.Lamba

May-2014

Dissertation submitted in partial fulfilment
Of the requirement for the degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS & COMMUNICATION ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

WAKNAGHAT, SOLAN – 173234, INDIA

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

(Established under the Act 14 of Legislative Assembly of Himachal Pradesh)

Waknaghat, P.O. DomeharBani. Teh. Kandaghat, Distt. Solan- 173234(H.P)

Phone: 01792-245367, 245368,245369

Fax-01792-245362

CERTIFICATE

This is to certify that the work titled **"Control of Robotic Arm and Virtual Object using Haptic Transducer"** submitted by **"Ms. Kanika Rana"** in the partial fulfillment for the award of degree of Bachelor of Technology (ECE) of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other university or institution for the award of this or any other degree or diploma.

[illegible]
27/5/2014

Prof. T.S.Lamba

Dean(Academic and Research)

Department of Electronics and Communication Engineering

Jaypee University of Information Technology (JUIT)

Waknaghat, Solan – 173234, India

(Supervisor)

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

(Established under the Act 14 of Legislative Assembly of Himachal Pradesh)

Waknaghat, P.O. DomeharBani. Teh. Kandaghat, Distt. Solan- 173234(H.P)

Phone: 01792-245367, 245368,245369

Fax-01792-245362

DECLARATION

I hereby declare that the work reported in the B. Tech report entitled **"Control of Robotic Arm and Virtual Object using Haptic Transducers"** submitted by "**Ms. Kanika Rana"** at Jaypee University Of Information Technology, Waknaghat is an authentic record of my work carried out under the supervision of "**Prof. T.S.Lamba"**. This work has not been submitted partially or wholly to any other university or institution for the award of this or any other degree or diploma.

[illegible]

Ms. Kanika Rana

101049

Department of Electronics and Communication Engineering

Jaypee University of Information Technology (JUIT)

Waknaghat, Solan – 173234, India

ACKNOWLEDGEMENT

I take this opportunity to express my profound gratitude and deep regards to my guide (Mentor) **Prof T.S.Lamba** for his exemplary guidance, mentoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall always be with me and take me a long way in the journey of life. I would want to thank Mr. Punit Gupta, CSE Department for all the support and help. I would also thank Mrs Meenakshi Sood for always being there and never saying no to any of the last minute requests for help. I thank every individual who helped me in every way possible to make this project work. Lastly, I thank almighty, my family and friends for their constant encouragement without which this project would not be possible.

Kanika Rana

ABSTRACT

The objective of this project is to make a ‘Haptic Transducer’ for controlling real (the robotic arm) and virtual objects (objects created in MATLAB).

Haptic transducer converts hand movement into a varying voltage level using either sensors or potentiometers. User has the option of wearing either a GLOVE or an EXOSKELETON. The glove is attached with two flex sensors and a Hall Effect sensor which generate two degrees of freedom for virtual environment while the exoskeleton is attached with three potentiometers to generate three degrees of freedom for real environment. The output of the two flex sensors and Hall Effect sensors when fed to a 3 D virtual hand of MATLAB through a micro-controller can control two parameters (hold and lift) of the virtual hand. Alternatively, the output of the three potentiometers when fed to a Robotic arm through a micro controller can control three parameters (claw, wrist and base) of the Real Robotic arm. So this haptic transducer is capable of controlling a virtual object (MATLAB environment) and a real object (Robotic arm) by sensing the actual hand movements.

Research indicates that a considerable population is kinesthetic or tactile learners. Haptics opens the door to an entirely different learning method and style, one that for many students provides the best opportunity to learn. Haptics technology has found its way into a range of commercial video game controllers (Nintendo) including joysticks and steering wheels. This technology has a long way to go and many fields to explore.

TABLE OF CONTENTS

List of Figures

List of Tables

CHAPTER-1
INTRODUCTION TO HAPTICS

1.1 What is Haptics?

Haptics technologies provide force feedback to users about the physical properties and movements of virtual objects, represented by a computer. Historically, human-computer interaction has been visual- words, data or images on a screen. Haptics is the new technology that incorporates both touch (tactile) and motion (Kinesthetic) elements. With the help of haptic interfaces, the user can 'feel' what is happening on the screen. To simulate real physical properties such as friction, weight, momentum, texture, resistance, temperature etc., applications incorporating haptics communicate these properties and let the user 'feel' them as well through the interface.

1.2 How does it work?

Haptics applications use specialized hardware to provide sensory feedback that simulates physical properties and forces. Haptic interfaces can take many forms, the most common configuration uses separate mechanical linkages to connect a person's fingers, sensors then translate these motions into actions on screen and motors transmit feedback through the linkages to the user's fingers.

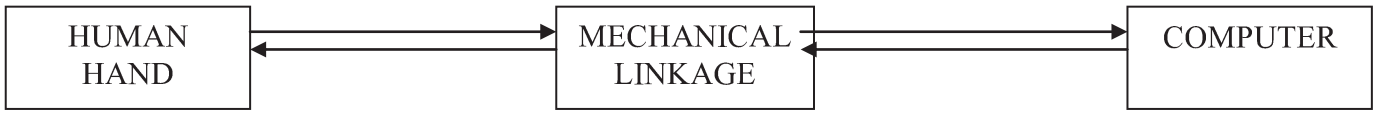


Fig. 1.1 Basic Working

Advantage

Because the object and its environment are purely virtual, the properties can be changed easily and its impact/effect can be seen and felt. In the project, though we plan on implementing the entire cycle, even implementing one side chain would be an achievement.

1.3 Who is using it?

Haptics tools are used in a variety of educational settings, both to teach concepts and to train students in a specific technique.

(a) To teach physics: allow students to interact with experiments that demonstrate gravity, friction, momentum and other fundamental forces.

(b) To teach biology: create virtual models of molecules, and feel their weight, size, shape and understand how they bond

(c) Aviation: Flight simulators combine visual and auditory elements with haptic technology, including resistance and vibrations in controls allowing student pilots to experience the kinds of sensations they will feel when they fly a real plane.

1.4 How is it Significant as a technology?

The interface between humans and computers has been described as **information bottleneck.** Computers store and process vast amounts of data, whereas humans experience through the 5 senses.

But, computers typically only take advantage of one or two sensory channels (sight and sound) to transmit information to people. Haptics promises to open this bottleneck by adding a new channel of communication, using the sense of touch, further expanding the notions of bi-directional communication between humans and computers to include sensory feedback.

It is a known fact that active learning strategies result in stronger comprehension of subjects and Haptics provides that mechanism, putting control and learning literally into the hands of users.

It also plays a vital role in assistive technology for the aid of visually impaired.

1.5 What are the downsides?

Designing and implementing haptic devices can be extremely complex, requiring highly specialized hardware and considerable processing power. Also the costs involved can be considerably high.

Since the object is virtual, a compelling interaction with the device requires that, all of the physical properties and forces involved be programmed into the application.

Generally the devices have fixed installations, not easily portable. Haptics is relatively young and offer somewhat crude experience to users as of now. Gathering raw materials for the device is difficult.

1.6 What are its future prospects?

Development and refining of various kinds of haptic interfaces will continue, providing more lifelike interactions with virtual objects and environment. Researchers will continue to investigate possible avenues for haptics to complement real experiences.
Advantages in hardware will provide opportunities to produce haptic haptic devices in smaller packages and haptic technology will find its way into increasingly common place tools.
Additionally, consumer- grade haptic devices are starting to appear on the market. As access to haptics increases, usage patterns and preferences will inform best best practices and applications – ultimately users will decide which activities are appropriately represented through haptics and which are better left to the real world.

1.7 Implications of Haptics on teaching and learning

Research indicates that a considerable portion of people are kinesthetic or tactile learners. Haptics opens the door to an entirely different learning method and style, one that for many students provides the best opportunity to learn.Haptics technology has found its way into a range of commercial video game controllers (Nintendo) including joysticks and steering wheels.

1.8 Work done by other people

There are a plethora of research projects that have been implemented in the field of Haptics, stating a few of them:

1.8.1 Surround Haptics: Immersive Tactile Experiences

This technology is integrated with a wide variety of entertainment and media contents, such that the contents are not only seen and heard but also felt, simultaneously. The tactile contents are carefully created and synchronized with visual and auditory cues to create effective and immersive experiences and increase the interest of users while playing video games, watching movies, etc. The technology is integrated into theater seats, gaming chairs and vests, rides, gloves, shoes, hand-held devices and controllers, clothes, to create another dimension of sensory feedback. For example, while playing an intense driving simulation game, users feel road conditions, gravel, traction, acceleration, brake, explosions, collisions, etc.

1.8.2 Stroke Sleeve: Spatially Distributed Tactile Feedback for Stroke Rehabilitation

Current therapy methods typically utilize virtual environments, providing patients with visual feedback as they perform repetitive arm movements to improve motor functionality, but patients often struggle to process such information. A therapist also provides hand-over-hand skilled guidance ("shaping") to assist the affected arm in performing functional tasks. These training methods are work-intensive for the therapist and can become boring and laborious for the patient, thus presenting the need to develop alternative therapy methods. Newer rehabilitation therapy benefits from the use of virtual reality and assistive robotic arms. These newer technologies permit the delivery of enhanced feedback and guided practices as well as the option to record user performance for evaluation.

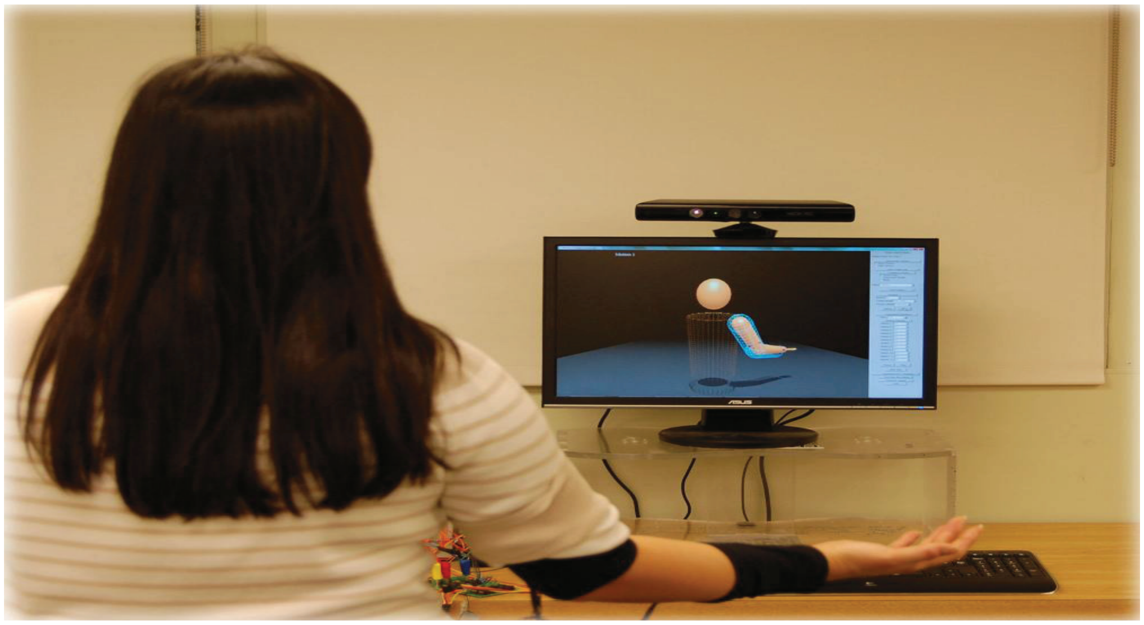


Fig. 1.2 Testing of Stroke Sleeve

A computer monitor displays a graphical representation of the user's arm motions and provides a wireframe overlay of a desired motion for an individual to learn. This helps the user see what the desired path is and make adjustments accordingly. The haptic (vibrotactile) feedback is provided in the form of cuffs that are placed around the bicep and forearm of the user. Stretchable, compression arm sleeves are used to create a tight fit and accomodate a wide variety of individuals. Four, shaftless, eccentric mass motors are evenly distributed around the cuff and attached to the material using rubber coated plastic caps. This ensures that the actuators are close to the skin for good tactile sensation and localization of the vibration actuators

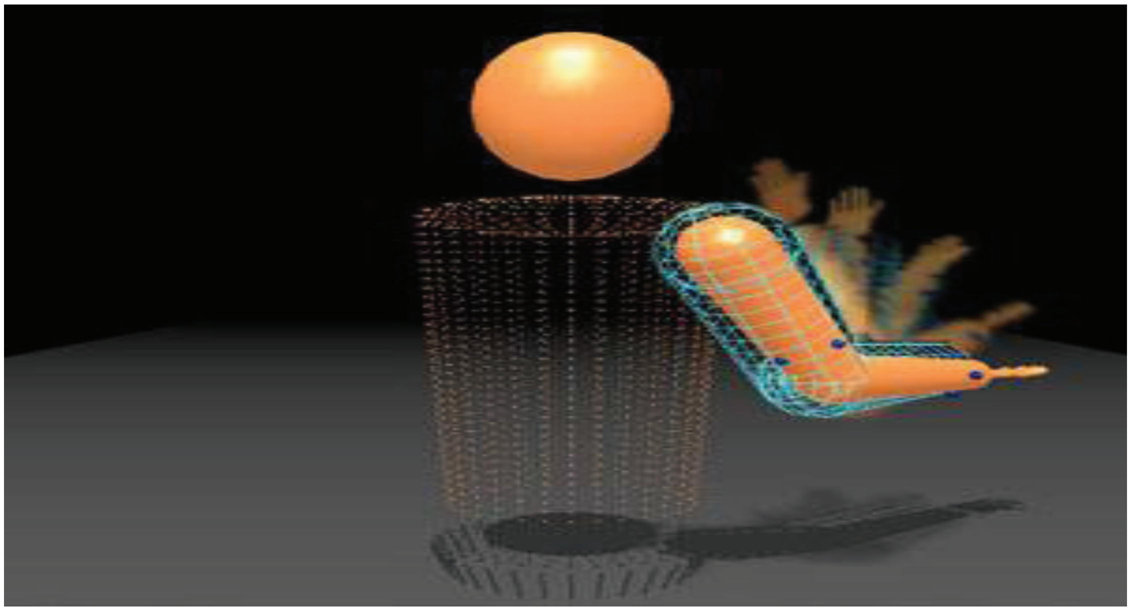


Fig. 1.3 Virtual Model describing the movement of hand

1.8.3 Haptography: Capturing and recreating the rich feel of real surfaces

Haptography, like photography in the visual domain, enables an individual to quickly record the haptic feel of a real object and reproduce it later for others to interact with in a variety of contexts. Particular positive ramifications of establishing the approach of haptography are to let doctors and dentists create haptic records of medical afflictions such as a decayed tooth surface to assist in diagnosis and patient health tracking; to improve the realism and consequent training efficacy of haptic surgical simulators and other computer-based education tools; to allow a wide range of people, such as museum goers and online shoppers, to touch realistic virtual copies of valuable items; to facilitate a haptographic approach to low-bandwidth and time-delayed teleoperation, as found in space exploration; and to enable new insights on human and robot touch capabilities.

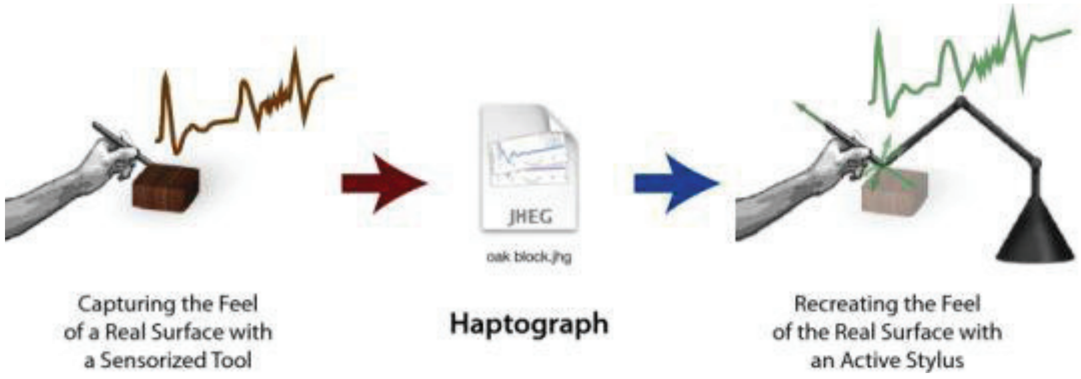

JHEG
oak block.jhg
Capturing the Feel of a Real Surface with a Sensorized Tool
Haptograph
Recreating the Feel of the Real Surface with an Active Stylus

Fig. 1.4 Haptograph

CHAPTER-2
SOFTWARE

2.1 MATLAB

We used **MATLAB** (Matrix Laboratory) for our virtual object simulations and interfacing with the microcontroller. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

MATLAB is a numerical computing environment and fourth-generation programming language. Developed by Math Works, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran.

We worked on MATLAB mainly for two purposes, firstly for creation of 3-D objects like cylinder, cube and sphere. These 3-D figures are the basic building blocks of final graphical interface that would be visible to the user. Adding to it, these are the virtual objects that we'll be working with.

Secondly, we used the **Instrument Control Toolbox** for the serial interfacing of the microprocessor to **MATLAB** and transmission of data from the microcontroller to **MATLAB**. It basically used the COM port 1 of the computer and read the data that was being transmitted by our Microcontroller. Adding to it, there are various other additional features that can be useful for us, like the data read from the Microcontroller can be exported to the workspace of MATLAB, can automatically make a **.mat** file and structures of which the read data is an object.

2.2 Hyper Terminal

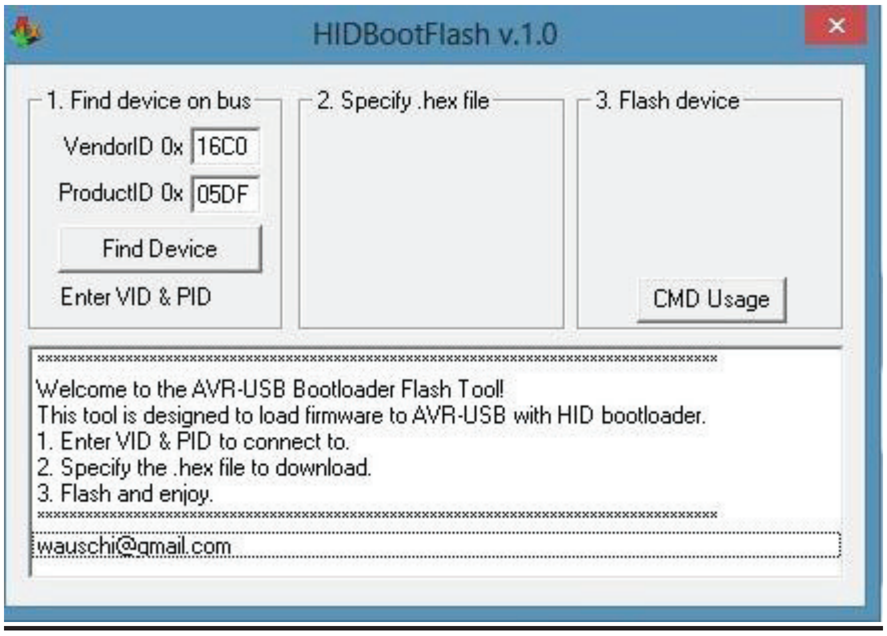
It is basically a *terminal emulator capable of connecting to systems* through TCP/IP Networks, Dial-Up Modems, and COM ports. It is one of the communication utilities provided by Microsoft in their operating systems. We used it for testing serial communication process between our Microcontroller and the computer. This is a default terminal program shipped with Windows xp OS, it not available for other versions of Windows, so we can use other terminal programs like '**PUTTY**'.

2.3 WinAVR

WinAVR is a suite of executable, open source software development tools for the Atmel AVR series of RISC microprocessors hosted on the Windows platform. It includes the GNU GCC compiler for C and C++.We used Programmers Notepad which is a programming environment which allows you to create and edit programs in various languages, and then compile the program into an executable format that the microcontroller will understand.

2.4 HID Bootflash

HID BootFlash is a GUI and command line tool used for transferring the .hex file to the microcontroller. It is an AVR USB Boot equivalent boot loader. In enables connecting to devices and flashing the firmware in a step-by-step approach.

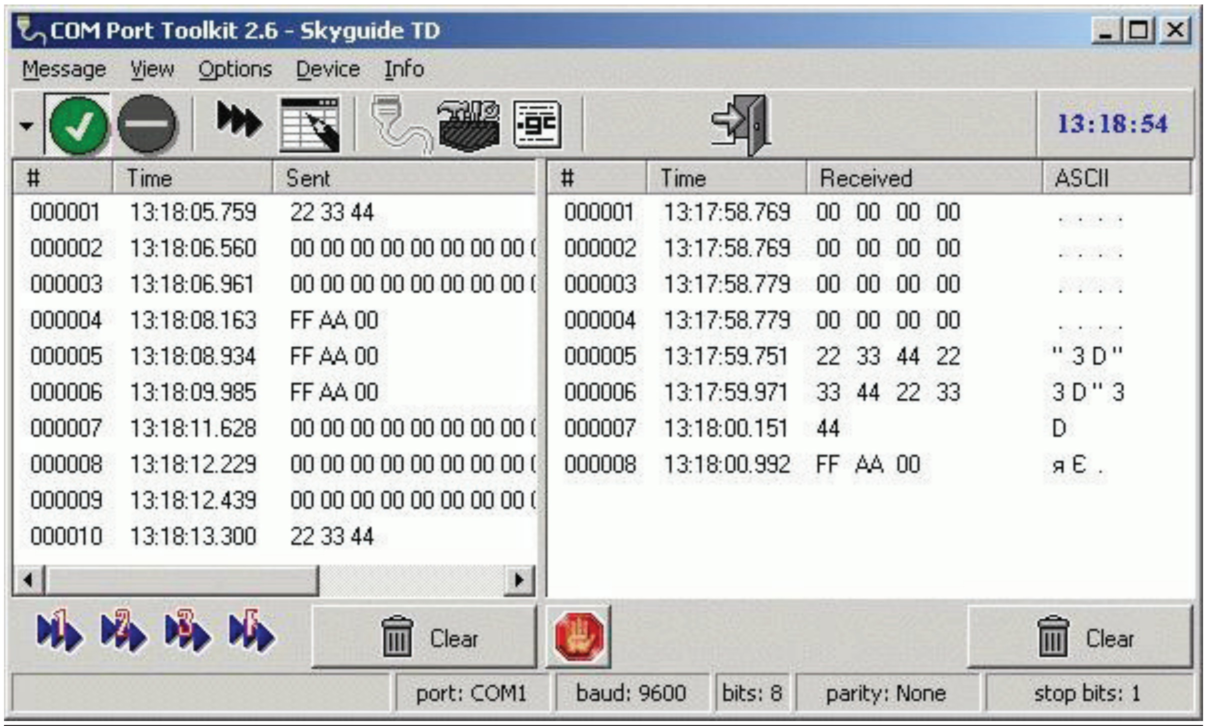


```
********************************************************************************
Welcome to the AVR-USB Bootloader Flash Tool!
This tool is designed to load firmware to AVR-USB with HID bootloader.
1. Enter VID & PID to connect to.
2. Specify the .hex file to download.
3. Flash and enjoy.
********************************************************************************
wauschi@gmail.com
```

Fig.2.1 HID Bootflash main window

2.5 COM Port Toolkit

COM Port toolkit is a serial port monitor with data transmission, data logging, and real- time data capture. It is a protocol, data and timing analyzer designed specifically to isolate problems with serial (RS-232,422,485) data communication control networks. This software can be used for implementing or debugging serial protocol. It can send and capture ASCII and HEX data. Other features of the software include time snipping, multiple devices oriented environment and data export to clipboard or file. The major advantage of this software is that it is in direct synchronization with the serial port and real time data acquisition becomes easy. But my project required real time data acquisition in MATLAB, and the testing times when the code for ADC data acquisition in MATLAB gave no results, and there was doubts about the correctness of code, testing it with COM Port gave results and inspired to go ahead with the code.


COM Port Toolkit 2.6 - Skyguide TD
Message View Options Device Info
13:18:54
Clear
Clear

```
#       Time          Sent                          #       Time          Received       ASCII
000001  13:18:05.759  22 33 44                      000001  13:17:58.769  00 00 00 00    . . . .
000002  13:18:06.560  00 00 00 00 00 00 00 00 (     000002  13:17:58.769  00 00 00 00    . . . .
000003  13:18:06.961  00 00 00 00 00 00 00 00 (     000003  13:17:58.779  00 00 00 00    . . . .
000004  13:18:08.163  FF AA 00                      000004  13:17:58.779  00 00 00 00    . . . .
000005  13:18:08.934  FF AA 00                      000005  13:17:59.751  22 33 44 22    " 3 D "
000006  13:18:09.985  FF AA 00                      000006  13:17:59.971  33 44 22 33    3 D " 3
000007  13:18:11.628  00 00 00 00 00 00 00 00 (     000007  13:18:00.151  44             D
000008  13:18:12.229  00 00 00 00 00 00 00 00 (     000008  13:18:00.992  FF AA 00       я Є .
000009  13:18:12.439  00 00 00 00 00 00 00 00 (
000010  13:18:13.300  22 33 44

port: COM1 | baud: 9600 | bits: 8 | parity: None | stop bits: 1
```

Fig.2.2 Screenshot of COMPORT when in use

2.6 Arduino Module

Arduino is an open-source electronics prototyping platform which is flexible and has easy-to-use hardware and software. It takes input from variety of switches and sensors, and controls a variety of motors, lights and other physical outputs. Projects based on Arduino can be stand-alone; they can communicate with software running on the computer. The Arduino board can be manually assembled or purchased preassembled; the open source IDE can be downloaded for free. The Arduino programming language is nothing but the implementation of wiring. It is similar to the physical computing platform, which is based on the programming needed in multimedia environment.

2.6.1 ARDUINO Uno

The Arduino Uno is a microcontroller board which uses ATmega328. It consists of 14 digital i/o pins. In those 14 pins, 6 pins are used for PMW outputs, another 6 pins are analog inputs. It has 16 MHz ceramic resonator, a USB connection that is use to connect the Arduino to system, a power jack for power supply. It also has an ICSP header, and a reset button which is used to reset the codes stored in the Arduino microcontroller. The board has everything which is needed to support a microcontroller to function. Arduino can be powered up either by connecting it to the computer through USB cable or by using AC-to-DC adapter or battery which is connected to power jack.

2.6.2 FEATURES OF ARDUINO Uno

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by boot loader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Table 2.1 Features Of Arduino Uno

CHAPTER-3
HARDWARE

3.1 ATmega 16 Development Board

We have used an ATmega 16 development board in our project as shown in Fig 3.1 .This development board can be used for interfacing of sensors, motors and LCD. It has switches for boot loading, reset, motors and power. It also has RS232 interface header. The board is compatible with 16×2 and 16x1 alphanumeric LCD. This board contains two L293d IC's which can control 8 unidirectional & 4 bidirectional motors. Four switches are provided. The board also has 4 LED's and a Buzzer for status or debugging purpose.

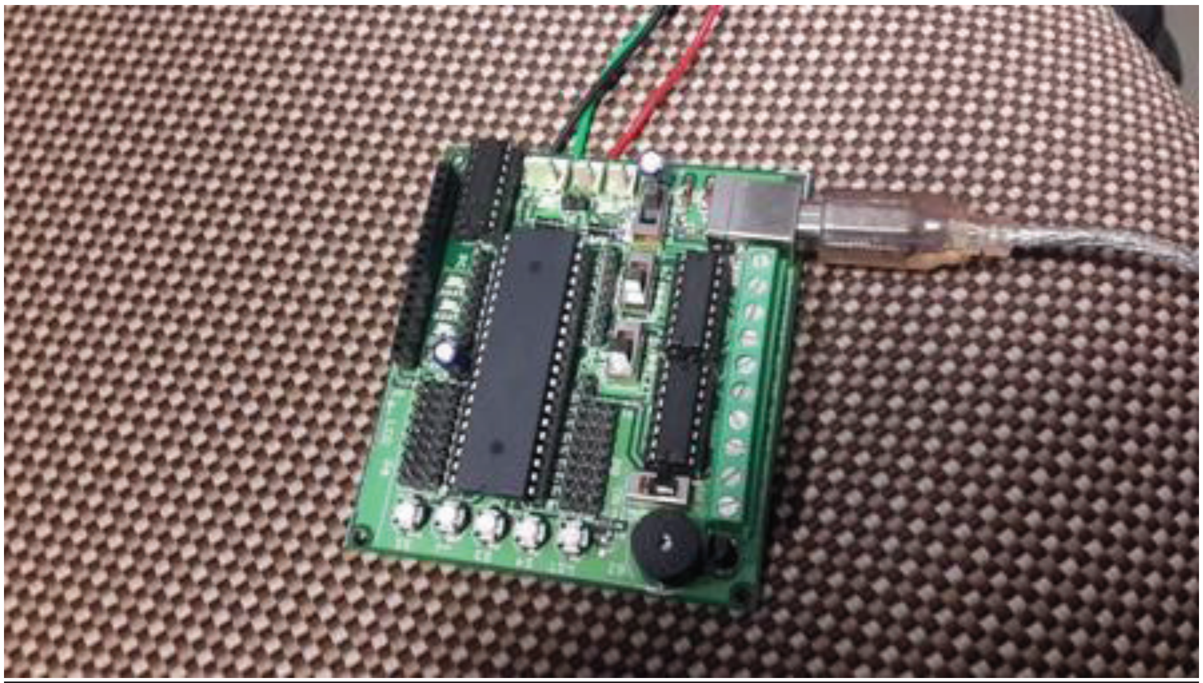


Fig.3.1 Atmega 16 Development Board

The respective ports responsible for controlling of motors, LED's, LCD, sensors and UART communication process are shown below in **Table 3.1**

Table 3.1 Port Description

ADC		**LED interface (Active high)**		**Motor Driver**
PA0 to PA7		LED1- Port C4		M0- Port B0
		LED2- Port C5		M1- Port B1
Switch (Active low)		LED3- Port C6		M2- Port B2
S1- PortA4		LED4- Port C7		M3- Port B3
S2- PortA5				M4- Port D4
S3- PortA6		**LCD interface (16x2 / 16x1)**		M5- Port D5
S4- PortA7		RS - Port C0		M6- Port D6
		RW- Port C1		M7- Port D7
Buzzer (Active High)		E - Port C2		
Port- C3		D5 - Port C4		**RS232/UART**
		D6 - Port C5		R-Receiver – PD0
Sensors		D7 - Port C6		T-Transmitter - PD1
PA0 to PA7		D8 - Port C7		G-Ground
PB0 to PB7				

3.1.1 ATmega16

ATmega16 is an 8-bit high performance microcontroller of Atmel's Mega AVR family with low power consumption. Atmega16 is based on enhanced RISC (Reduced Instruction Set Computing) architecture with 131 powerful instructions. Most of the instructions execute in one machine cycle. Atmega16 can work on a maximum frequency of 16MHz.

It has ***16 KB programmable*** flash memory, static RAM of 1 KB and EEPROM of 512 Bytes. The endurance cycle of flash memory and EEPROM is 10,000 and 100,000, respectively.

It is a ***40 pin microcontroller***. There are 32 I/O (input/output) lines which are divided into four 8-bit ports designated as PORTA, PORTB, PORTC and PORTD.

Moreover, it also has various in-built peripherals like **USART**, **ADC**, Analog Comparator, SPI, JTAG etc. Each I/O pin has an alternative task related to in-built peripherals. The following figure **Fig 3.2** shows the pin description of ATmega16.

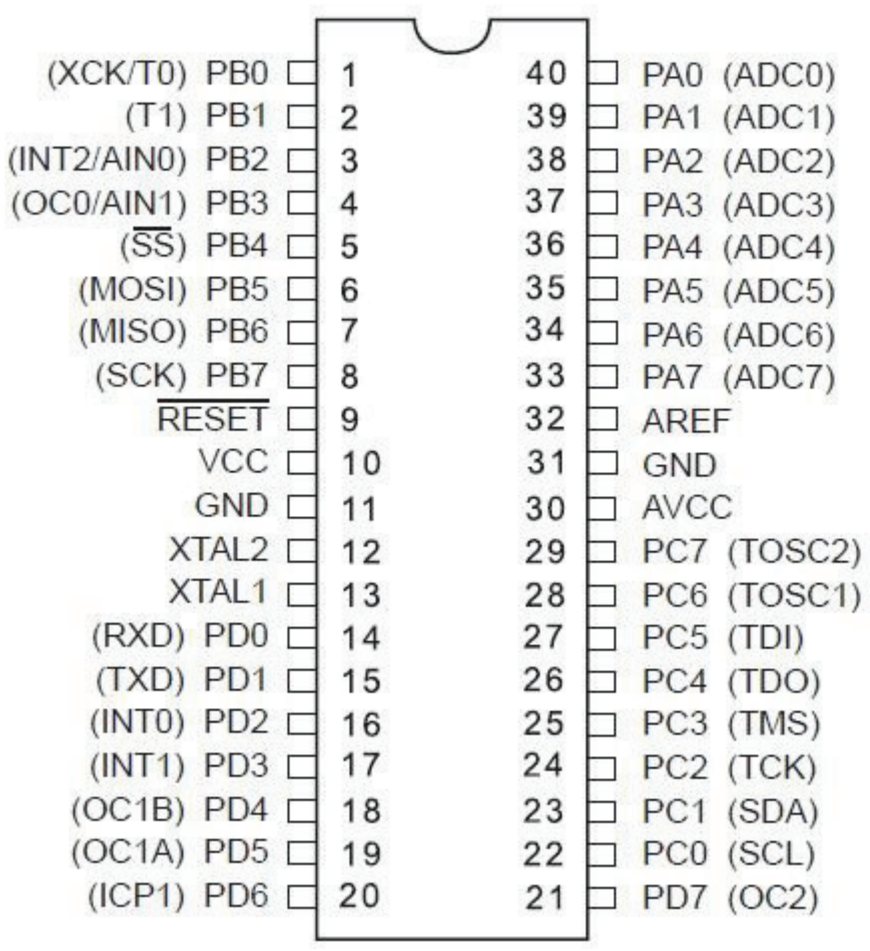


Fig.3.2 ATmega 16 Pin Diagram

3.2 Potentiometers

Potentiometers are transducers, it converts rotary or liner motion from the operator into a change of resistance, and this change is used to control the hand movement of the robotic arm. The potentiometer used is a three legged element, of which two outside terminals act as fixed resistor. A movable contact called wiper (the middle terminal) moves across the resistor, producing a variable resistance between the center terminals and the two sides.

Three potentiometers are used to control the claw, wrist and base movements of the robotic arm.

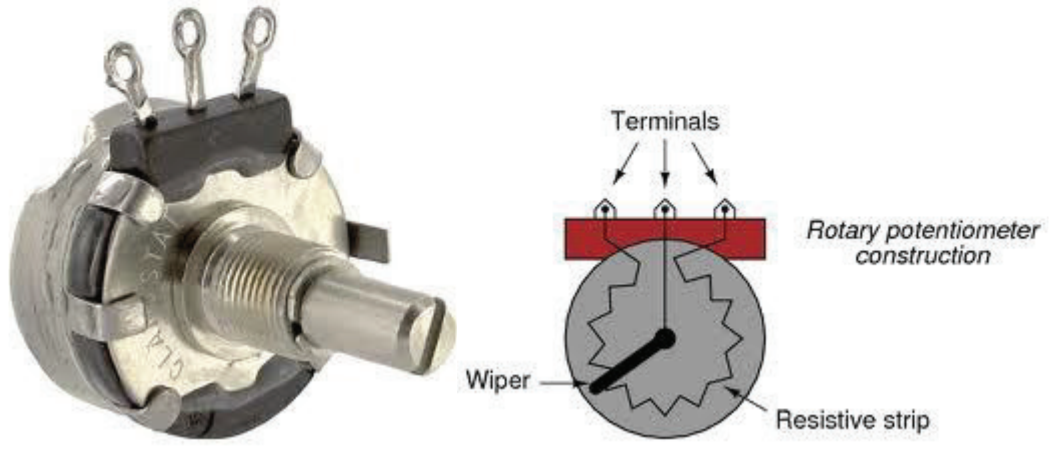


Fig. 3.3 Potentiometer

3.3 Flex Sensors

3.3.1 What are Flex Sensors?

Flex sensors also known as bend sensors are specially made sensors which change their resistance depending on amount they are bent. They convert change in bend into electrical energy. They are available in market in form of thin strips of varying lengths and can be unidirectional or bi-directional, that is whether they sense the bend in one direction or both. They are used not only to detect a bend, but also figure out how much something is bent.

3.3.2 Working

Flex sensors are basically devices that convert physical parameter's to analog electrical signal. Inside the flex sensor is a bunch of conducting particles, which are closest to each other when the sensor is straight or flat. But when it is bent, they spread further away from each other making it less conductive and thus increasing the resistance. To use it we attach Flex sensors to a voltage divider circuit. As we bend these sensors, change in resistance can be measured by checking the changing voltage. It also depends where we bend the sensors from, hence kept the sensor on the finger and marked the three points, where the finger can be bent from(degree of freedom) and measure the possible angles of bent at all the three points(0 to 90 degree with a gap of 10 degree).

3.3.3 Applications of Flex Sensors

Today, there are various applications of Flex sensors some of which are mentioned below.

1. In Robotics, Flex Sensors are used to determine joint movement and placement in various robotics components.
2. They are used in bumper switches and pressure switches which are used for various purposes.
3. These sensors can be used in gaming gloves to make virtual reality possible in gaming.
4. For bio-metrics, the sensor can be placed on a moving joint of athletic equipment to provide an electrical indication of movement or placement.

Flex sensors are also used in auto controls, fitness products, measuring devices, assistive technology, musical instruments, joysticks and many more.

3.4 DB-9 Connector

The term "DB9" refers to a common connector type, one of the D-Subminiature or D-Sub types of connectors. DB9 has the smallest "footprint" of the D-Subminiature connectors, and houses 9 pins (for the male connector) or 9 holes (for the female connector). They are designed to work with the **RS 232** serial interface standard, which determined the function of all nine pins as a standard.The functions of each pin are shown in figure **Fig 3.3.**

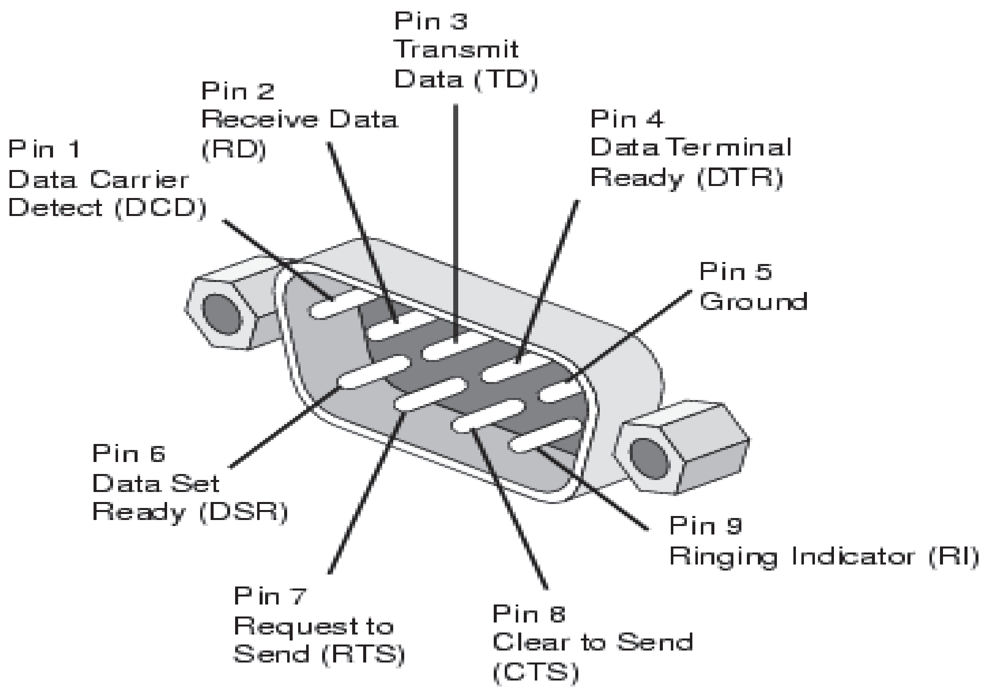


Fig 3.4 DB9 Pin Description

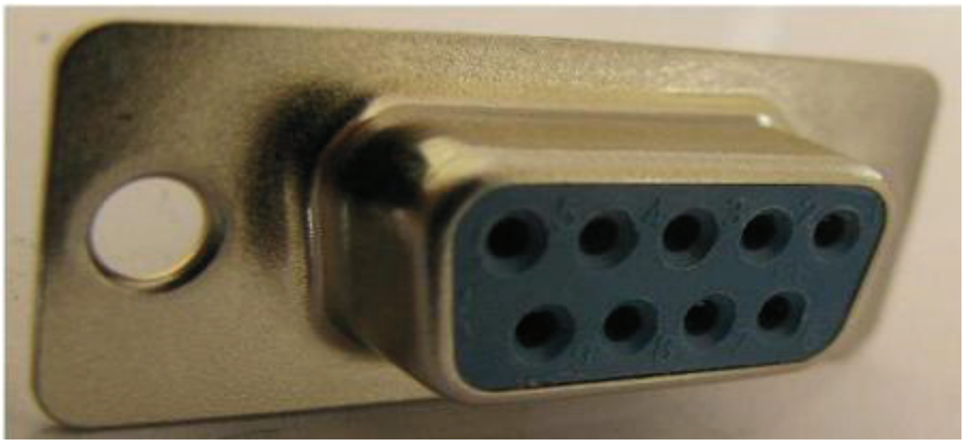


Fig 3.5 DB9 Female Connector

We used 3 pins (Rx, Tx, GND) for serial communication.

3.5 Hall effect sensors

Hall Effect sensors are essentially elements that vary output voltage depending on the changes in the magnetic field. There are essentially four types of Hall Effect sensors:

1. Typical hall effect sensors:

   They basically act as a reed switch and are always in one constant mode and whenever they sense a magnetic field switch, they switch to the other mode. The moment magnetic field is removed; they are back to the original constant mode.

2. Latch type:

   This type of sensor, initially is in a neutral mode, but when senses the activating pole of the magnet (south or north, depending on sensor to sensor), it is switched ON, and it remains in this state now even when the magnetic field is removed. For the output voltage to now switch to low, i.e. sensor to be turned OFF, it needs to be approached by the other pole of the magnet (opposite of the activating pole).

3. Switch type:

   They are initially at the mid-point value of V_{cc} and when approached by the activating pole of the magnet, the output voltage increases to V_{cc} and when approached by the opposite pole, the voltage drops to zero.

4. Linear:

   They produce a hall voltage proportional to the strength of the magnetic field around it.

The Hall Effect sensor used is- WSH134 which is a unipolar Hall Effect switch IC and its features are as follows:

(i) Operates from 2.4V to 26V supply voltage with reverse voltage protection

(ii) Operates with magnetic fields from DC to 15kHz

(iii) On-chip Hall sensor

(iv) On-chip temperature compensation circuitry minimizes shifts in ON and OFF points and hysteresis over temperature and supply voltage.

(v) Ideal sensor for speed measurement, revolution counting, positioning and DC brushless motors.

(vi) ON (low voltage) with south magnetic pole and OFF (high voltage) without magnetic field or with magnetic North Pole.

(vii) Operation over temperature range from -40^{o}C to 125^{o}C

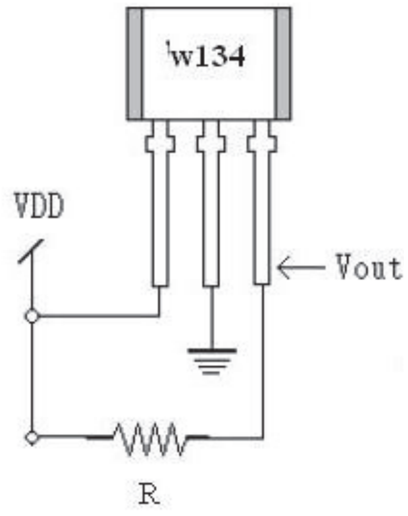


Fig 3.6 Hall Effect Sensor

B- Magnetic flux density, the property of a magnetic field used to determine hall device switch points, unit being Gauss (G) or Tesla (T).

Since B can have a north or south polarity, going by the convention, B is negative for north polarity magnetic fields and positive for south polarity magnetic field. Relative strength of the field is indicated by the absolute value of B and sign just indicates the polarity of the field.

B_{OP} Magnetic operating point; the level of a strengthening magnetic field at which a Hall device switches ON.

B_{RP} Magnetic release point, the level of a weakening magnetic field at which a hall device switches OFF (or for some types of hall devices, the level of a strengthening negative field gives a positive B_{OP}). The resulting state of the device output depends on the individual device electronic design.

B_{HYS} Magnetic switch point hysteresis. The transfer function of a hall device is designed with this offset between the switch points to filter out small fluctuations in the magnetic field that can result from mechanical vibration or electromagnetic noise in the application. $B_{HYS} = |B_{OP} - B_{RP}|$

PULL UP RESISTOR:

A pull up resistor must be connected between the device supply and the output pin. The minimum pull up resistance is a function of the Hall IC maximum current and the supply voltage (V_{cc}/I_{max})

In applications, where current consumption is a concern, the pull-up resistance could be as large as 50 to100kΩ. Caution however is required because large pull up values make it possible to induce external leakage currents to ground. It is not a device problem, rather the leakage occurs in the conductors between pull up resistor and the device output pin. These currents could be high enough to reduce the output voltage, regardless of the state of the magnetic field and device switching state. Taken to the extreme, this can reduce the output voltage, enough to inhibit proper external logic functions. Thus in the project, I have used 220Ω as the pull up resistance.

BYPASS CAPACITORS:

For designs without chopper stabilization-it is recommended that a 0.01uF capacitor be placed between output and ground and between supply and ground pins.

For designs with chopper stabilization- a 0.1uF capacitor be placed between supply and ground pins and a 0.01uF between output and ground pins.

Since my project did not work at conditions that require bypass capacitors and having tested it experimentally, the bypass capacitors made no difference to the output, thus bypass capacitors were not included in the final circuit.

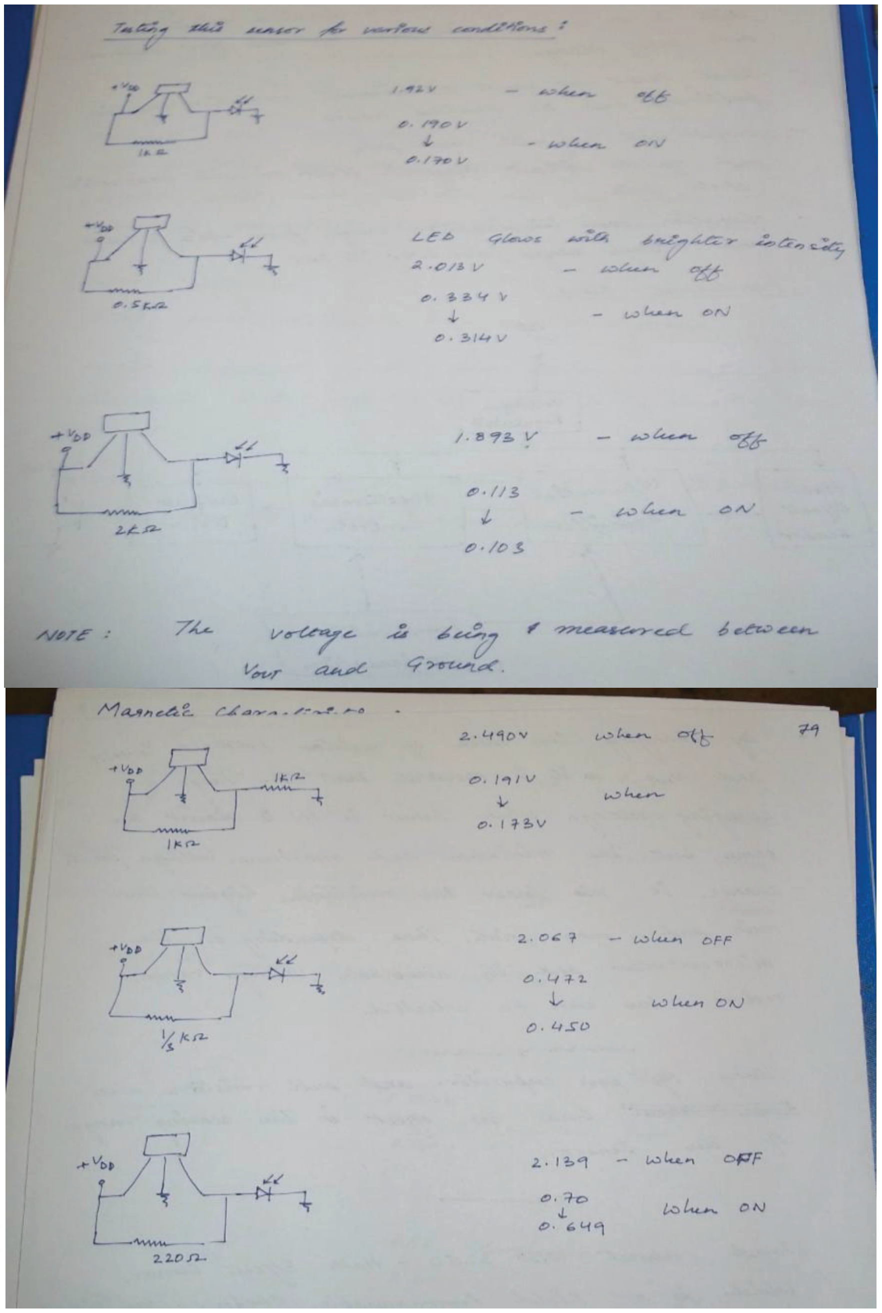
POWER DISSIPATION

Total power dissipation is a sum of two factors:

(a) Power consumed by the Hall device, excluding the power dissipated in the output. This value is V_{cc} times the supply current

(b) Power consumed in the output transistor, this value is $V_{(sat)}$ times the output current (set by pull up resistor).

The Hall sensor was tested for various conditions:


+VDD
1kΩ
+VDD
0.5kΩ
+VDD
2kΩ
+VDD
1kΩ
1kΩ
+VDD
1/3 kΩ
+VDD
220Ω

Testing this sensor for various conditions:

1.92V – when off

0.190V ↓ 0.170V – when ON

LED Glows with brighter intensity

2.013V – when off

0.334V ↓ 0.314V – when ON

1.893V – when off

0.113 ↓ 0.103 – when ON

NOTE: The voltage is being measured between V_{OUT} and Ground.

Magnetic Characteristic.

79

2.490V when off

0.191V ↓ 0.173V when

2.067 – when OFF

0.472 ↓ 0.450 when ON

2.139 – when OFF

0.70 ↓ 0.649 when ON

Fig 3.7 Testing of Hall Effect Sensor

The conclusion reached was to use a 220Ω resistor as it gives a desired range of operation.

3.6 Haptic robotic arm

The haptic robotic arm is constructed using simple plastic materials, screws, gears, worms and motors. There are DC motors connected, one each for controlling the claw, base and the base. When the motor rotates it makes the other parts rotate and as a result perform the desired action. This motion of the robot is controlled through the potentiometers which are fixed into the exoskeleton at the crucial points which detect the movement of the hand and indeed translate it into signals indicating the movement of the arm. The three potentiometers are connected as shown in the picture, where the motion of the fingers translates the movement of the claw (opening and closing). The motion of the wrist translates into the movement of the wrist of the robot (bending up and down) and the motion of arm sideways translates into the rotation of the base of the robot left and right.

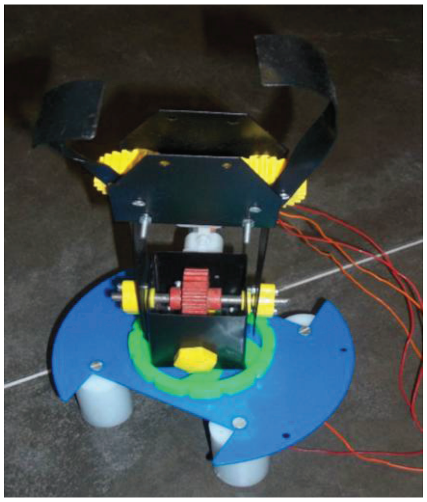


Fig 3.9 Robotic Arm

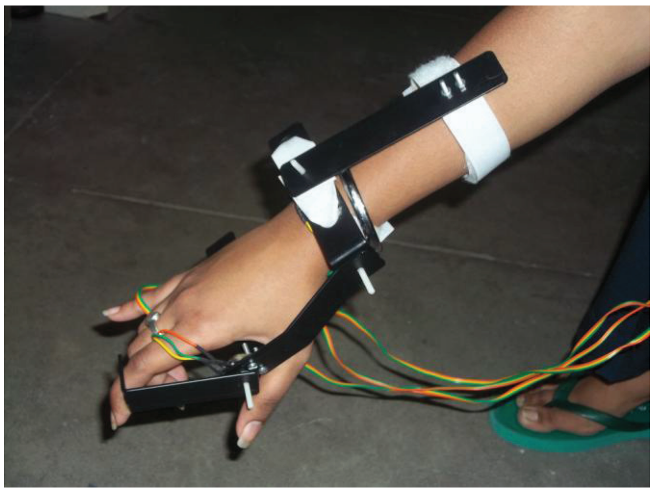


Fig 3.8 Exoskeleton-Robotic arm controller

CHAPTER-4
WORK DONE

The making of 3-D objects in MATLAB with proper lights and material, rotating them, and interfacing of the microcontroller have been accomplished.

4.1 Making 3-D objects using MATLAB

We wrote MATLAB codes for creating 3-D objects like cylinder, cube and sphere. These 3-D figures are the basic building blocks of final graphical interface that would be visible to the user. We also performed different operations on the objects like rotation, highlighting it, adjusting camera angle etc. Adding to it, these are the virtual objects that we'll be working with.
The MATLAB codes can be found in **APPENDIX A.1**.

4.2 Basic ATmega Programs

We wrote a few basic programs like blinking of LED's and working of motors using '**Embedded C'** programming language, just to get familiar with the development board and its working. The codes that we wrote are present in **APPENDIX A.2.**

The steps for writing codes to Microcontroller are :

1. Write the '**C program**' in *Programmers Notepad [WinAVR],* and save it with the name 'main.c'.
2. Now edit the **Makefile** (used by the compiler to understand instructions to compile the C programs).
    a. Write the same name as your C file in the main file option of Makefile.
    b. Set the **F_CPU** to 16000000.
    c. Choose your Microcontroller in the options. In our case it was **ATmega 16a.**
    d. Now save the Makefile in the same folder as your Main file.
3. Now go back to Programmers Notepad and select the option **'MakeAll'.**
4. If your codes compiles successfully then you will see a '**.hex**' file (it contains the code in a language that machine understands) created in the same folder as your main file.

5. Now once you have the ***hex*** file, you can burn this on your Microcontroller using **HID Boot Flash**.
6. Now switch your development board in '**Programming Mode**' and open HID Boot Flash.
7. In HID Boot Flash click on '**Find Device**' option, and if the device is not detected then click on '**Reset**' button on your device.
8. Select the ***hex file*** that is to be burnt on the microcontroller.
9. Then Click on **Flash Device.** Now your microcontroller has been programmed.

4.3 Making Flex Sensors

We made flex sensors in the laboratory by using two different methods, described below.

Method – 1

Given below is the description of materials used and the procedure that we followed for making Flex Sensors.

Materials used

1. Anti Static Bags 10x15cm
2. Masking Tape (2.4 cm and 1.1cm).
3. Jumper Wires

Tools used

1. Pencil
2. Wire Stripper
3. Pen Knife

Preparation

1. Cut 2 pieces of 0.8cm by 15cm, and 1 piece of 1.7cm by 15cm from Anti Static Bag.
2. Take jumper wires and measure 5cm of wire than strip it.
3. Make a loop back down to the base where the insulation starts, and twist the wires together a little so that it stays there.
4. Do this for two jumper wires.
5. Take the thin masking tape (1.1cm in width), measure 17cm and cut it.
6. Cut another strip like this

7. Take the thicker masking tape (2.4cm in width), measure 19cm and cut it.

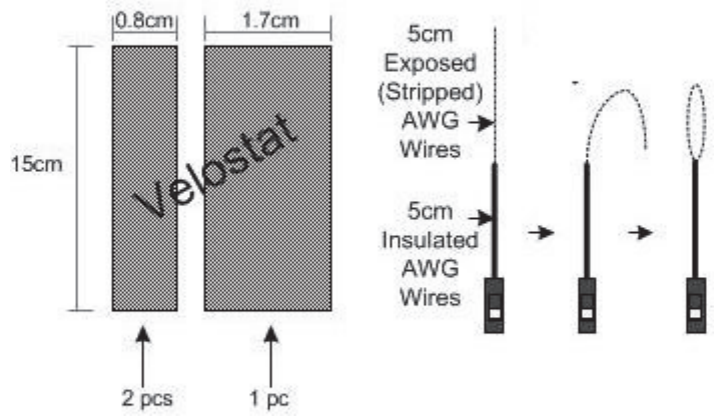


Fig. 4.1 Prepration for Making Flex Sensor

Procedure

1. Using the thinner masking tape (1.1cm) with the sticky side facing up, Place the conductive bag piece (.8cm by 15 cm) that we just cut, right in the middle of it.
2. Make sure there's a border of sticky tape all around. Smooth it out.
3. Take one of the jumper wire connectors and place it slightly off-centre onto the edge as shown.
4. Check that the exposed loop wire should be kept within the black conductive piece.
5. Allow a 0.5cm of insulated wire part to be within the black piece as well.
6. Do the same with another piece.
7. Take the large conductive bag piece that we cut just now (1.7cm by 15 cm), and fold it in half, lengthwise.
8. Lengthwise, align the 2 thin pieces and match the sticky border together
9. Place the large piece that we just folded into half into the sandwich.
10. Wrap this whole thing up.
11. Take the thicker masking tape piece and with the sticky side up, place two unstripped jumper wires right in the middle of it.
12. Place the sensor on top of the wires, right smack in the middle.
13. Fold the sticky edges up of the thicker masking tape onto the sensor.

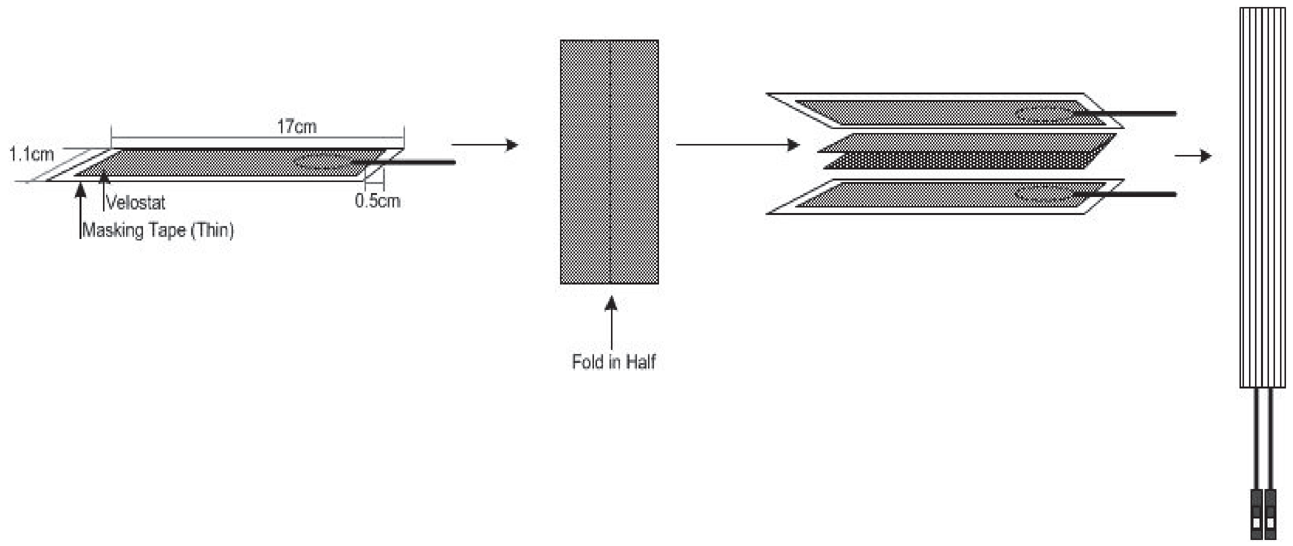


Fig. 4.2 Procedure for Making Flex Sensor

Shown below (**Fig 4.3**) are the flex sensor and intermediate steps in making of flex sensors by us using Method 1.

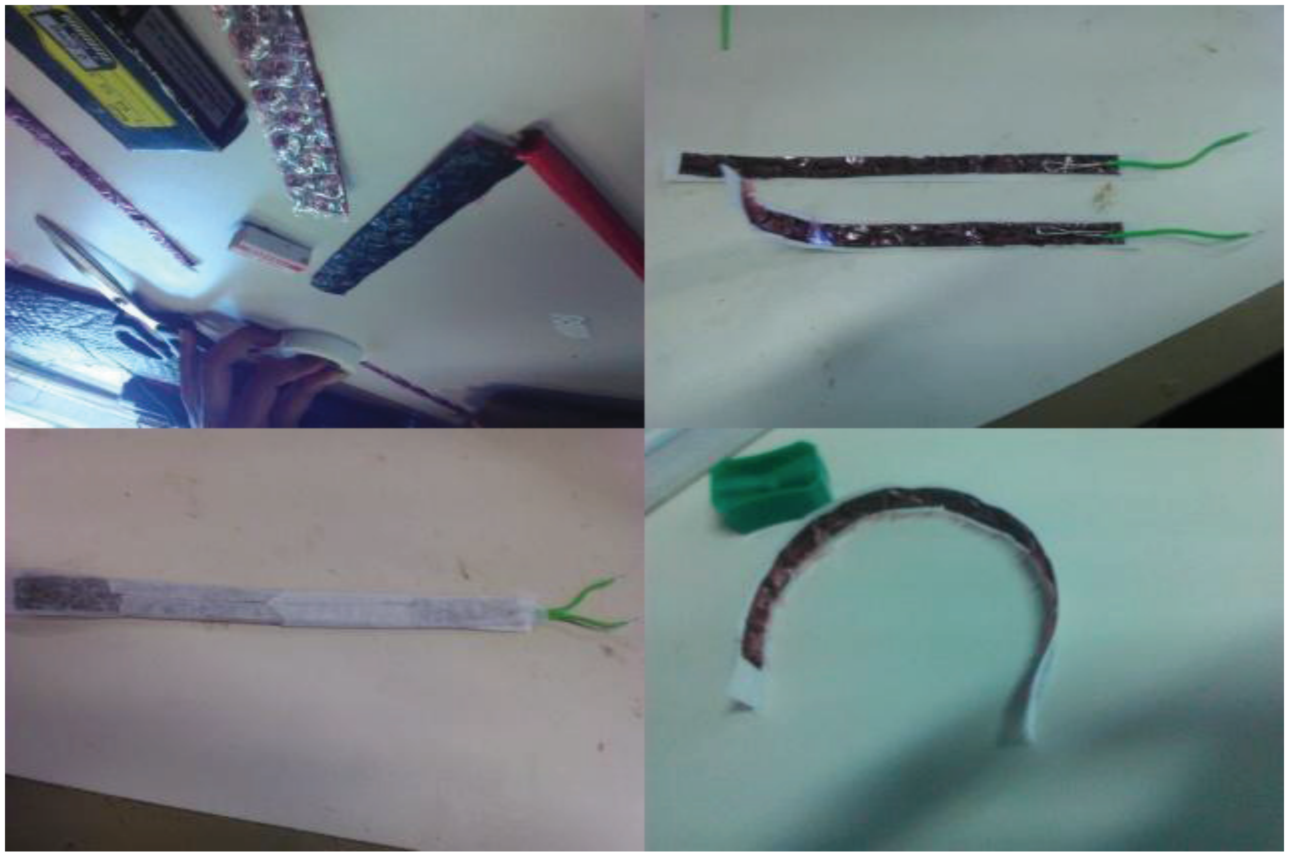


Fig. 4.3 Flex Sensor Using Method 1

Method-2

Given below is the description of materials used and the procedure that we followed for making Flex Sensors using second method.

Materials Required

1. Aluminum foil
2. Resistive Foam
3. Adhesive Tape
4. Connecting wires

Preparation

1. Take Adhesive Tape and cut two strips of 1cm by 8cm each.
2. Cut 2 pieces of 0.8cm by 8cm from Aluminium foil.
3. Take the foam and cut a strip of .8 by 8cm.
4. Take jumper wires and measure 5cm of wire than strip it.
5. Make a loop back down to the base where the insulation starts, and twist the wires together a little so that it stays there.
6. Do this for two jumper wires.

Procedure

1. Using the Adhesive tape with the sticky side facing up, Place the Aluminium foil (.8cmby 8 cm) that we just cut, right in the middle of it.
2. Make sure there's a border of Adhesive tape all around. Smooth it out.
3. Take one of the jumper wire connectors and place it slightly off-centre onto the edge.
4. Check that the exposed loop wire should be kept within the black conductive piece. (No peeking out on the sides!!!).
5. Allow a 0.5cm of insulated wire part to be within the black piece as well.
6. Do the same with another piece.
7. Lengthwise, align the 2 pieces and match the sticky border together
8. Place the conductive foam piece that we just folded into half into the sandwich.
9. Wrap this whole thing up and we are done.

The final results of this method and a glimpse of readings is shown below in **Fig 4.**

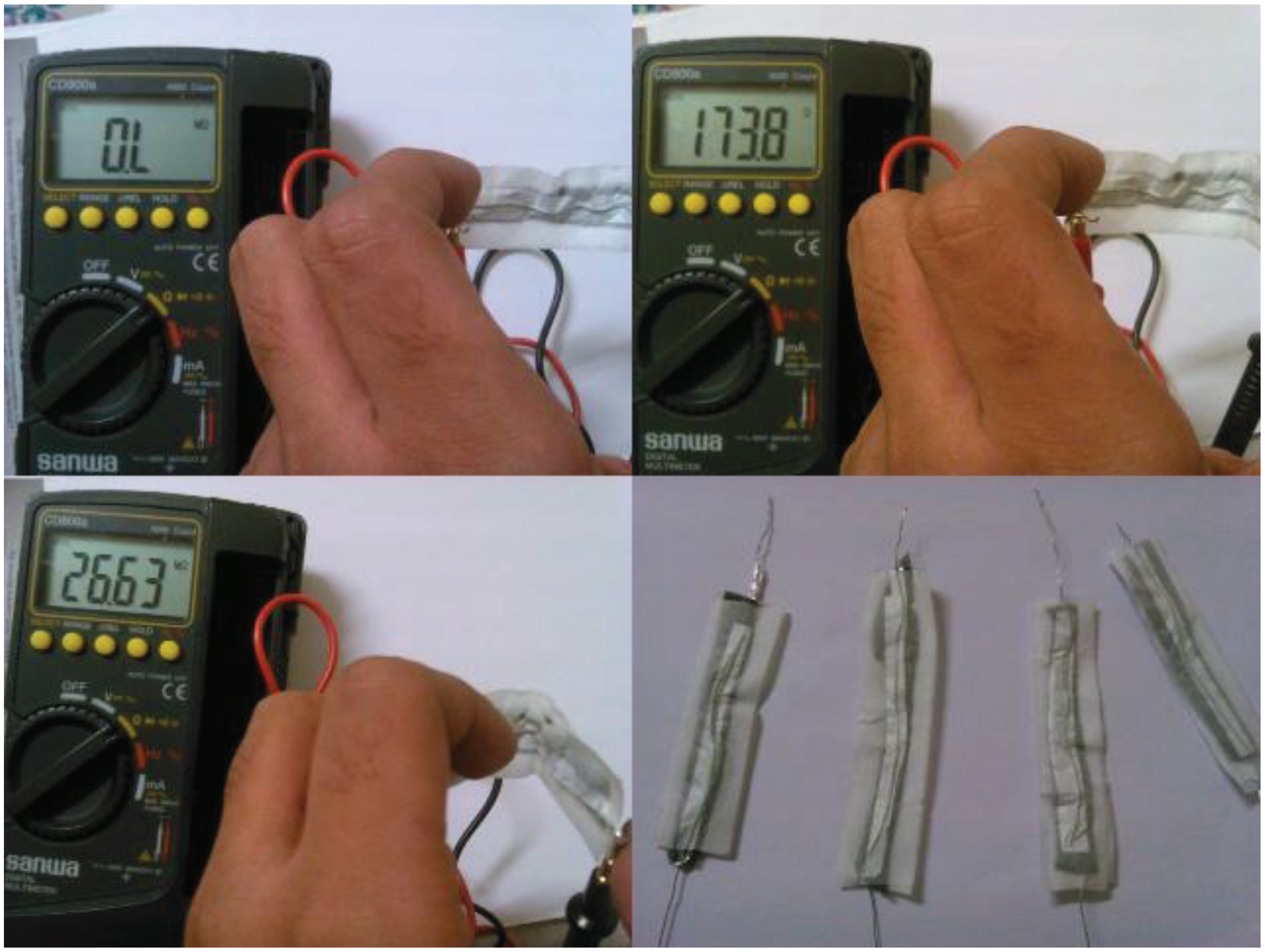


Fig. 4.4 Flex Sensors Using Method 2

4.4 Interfacing Microcontroller board with Computer

4.4.1 Understanding the register structure of USART of AVR

First we need to understand the USART of AVR microcontroller and write the code to initialize the USART and use it to send and receive data.

Like many microcontrollers, AVR also has a dedicated hardware for serial communication this part is called the USART - Universal Synchronous Asynchronous Receiver Transmitter. This special hardware makes life as a programmer easier.

USART automatically senses the start of transmission of RX line and then inputs the whole byte and when it has the byte it informs you (CPU) to read that data from one of its registers. The

USART of AVR is very versatile and can be setup for various different modes as required by our application.

The USART of the AVR is connected to the CPU by the following six registers.

- **UDR** - USART Data Register: Actually this is not one but two register but when you read it you will get the data stored in receive buffer and when you write data to it goes into the transmitter's buffer. This important to remember it.

- **UCSR** - USART Control and status Register: As the name suggests it is used to configure the USART and it also stores some status about the USART. There are three kind of this register: the UCSRA, UCSRB and UCSRC.

- **UBRRH** and **UBRRL**: This is the USART Baud rate register, it is 16BIT wide so UBRRH is the High Byte and UBRRL is Low byte. But as we are using C language it is directly available as UBRR and compiler manages the 16BIT access.

Explaining the registers:

UCSRA: USART Control and Status Register A

Bit No	7	6	5	4	3	2	1	0
Name	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
Initial Val	0	0	1	0	0	0	0	0

RXC this bit is set when the USART has completed receiving a byte from the host (may be your PC) and the program should read it from**UDR**

TXC This bit is set (1) when the USART has completed transmitting a byte to the host and your program can write new data to USART via UDR

UCSRB: USART Control and Status Register B

Bit No	7	6	5	4	3	2	1	0
Name	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
Initial Val	0	0	0	0	0	0	0	0

RXCIE: Receive Complete Interrupt Enable - When this bit is written one the the associated interrupt is enabled.
TXCIE: Transmit Complete Interrupt Enable - When this bit is written one the the associated interrupt is enabled.

RXEN: Receiver Enable - When you write this bit to 1 the USART receiver is enabled. **The normal port functionality of RX pin will be overridden.** So you see that the associated I/O pin now switch to its secondary function, i.e. RX for USART.

TXEN: Transmitter Enable - As the name says!

UCSZ2: USART Character Size

UCSRC: USART Control and Status Register C

Bit No	7	6	5	4	3	2	1	0
Name	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
Initial Val	0	0	0	0	0	0	0	0

IMPORTANT : The UCSRC and the UBRRH (discussed below) register shares same address so to determine which register user want to write is decided with the 7th(last) bit of data if its 1 then the data is written to UCSRC else it goes to UBRRH. This seventh bit is called the

URSEL: USART register select.
UMSEL: USART Mode Select - This bit selects between asynchronous and synchronous mode. As asynchronous mode is more popular with USART we will be using that.

UMSEL	Mode
0	Asynchronous
1	Synchronous

USBS: USART Stop Bit Select - This bit selects the number of stop bits in the data transfer.

USBS	Stop Bit(s)
0	1 BIT
1	2 BIT

UCSZ: USART Character size - These three bits (one in the UCSRB) selects the number of bits of data that is transmitted in each frame. Normally the unit of data in MCU is 8BIT (C type "char") and this is most widely used so we will go for this. Otherwise you can select 5,6,7,8 or 9 bit frames!

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5Bit
0	0	1	6Bit
0	1	0	7Bit
0	***1***	***1***	***8Bit***
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9Bit

UBRR: USART Baud Rate Register:

This is the USART Baud rate register, it is 16BIT wide so **UBRRH** is the High Byte and **UBRRL** is Low byte. But as we are using C language it is directly available as UBRR and compiler manages the 16BIT access. This register is used by the USART to generate the data transmission at specified speed (say 9600Bps.)UBRR value is calculated according to following formula.

$$UBRR = \frac{f_{osc}}{16 \times \text{Baud Rate}} - 1$$

Where fosc is your CPU frequency say 16MHz

Before we start interfacing our device, we need to find out the COM port number of the Serial port to which our avr is connected, since a PC can have several COM ports, each may have some peripheral connected to it like a Modem. Serial Ports on PC are numbered like COM1, COM2 ... COMn etc.

Thus the port number can be found as follows:

1. Right Click on **"My Computer"** icon in Windows Desktop.
2. Select **"Properties"**
3. The System Properties will open up. Go to the **"Hardware"** Tab.

4. In Hardware tab select **"Device Manager"** button. It will open up device manager.
5. In Device Manager Find the Node **"Ports (COM & LPT)"**
6. Expand the port node in device manager and depending on the type of connection we can see the available ports.

In our case, the port is COM1.

4.4.2 Serial interfacing using Hyper Terminal

We basically used Hyper Terminal to see whether our connections are fine and serial communication is possible.

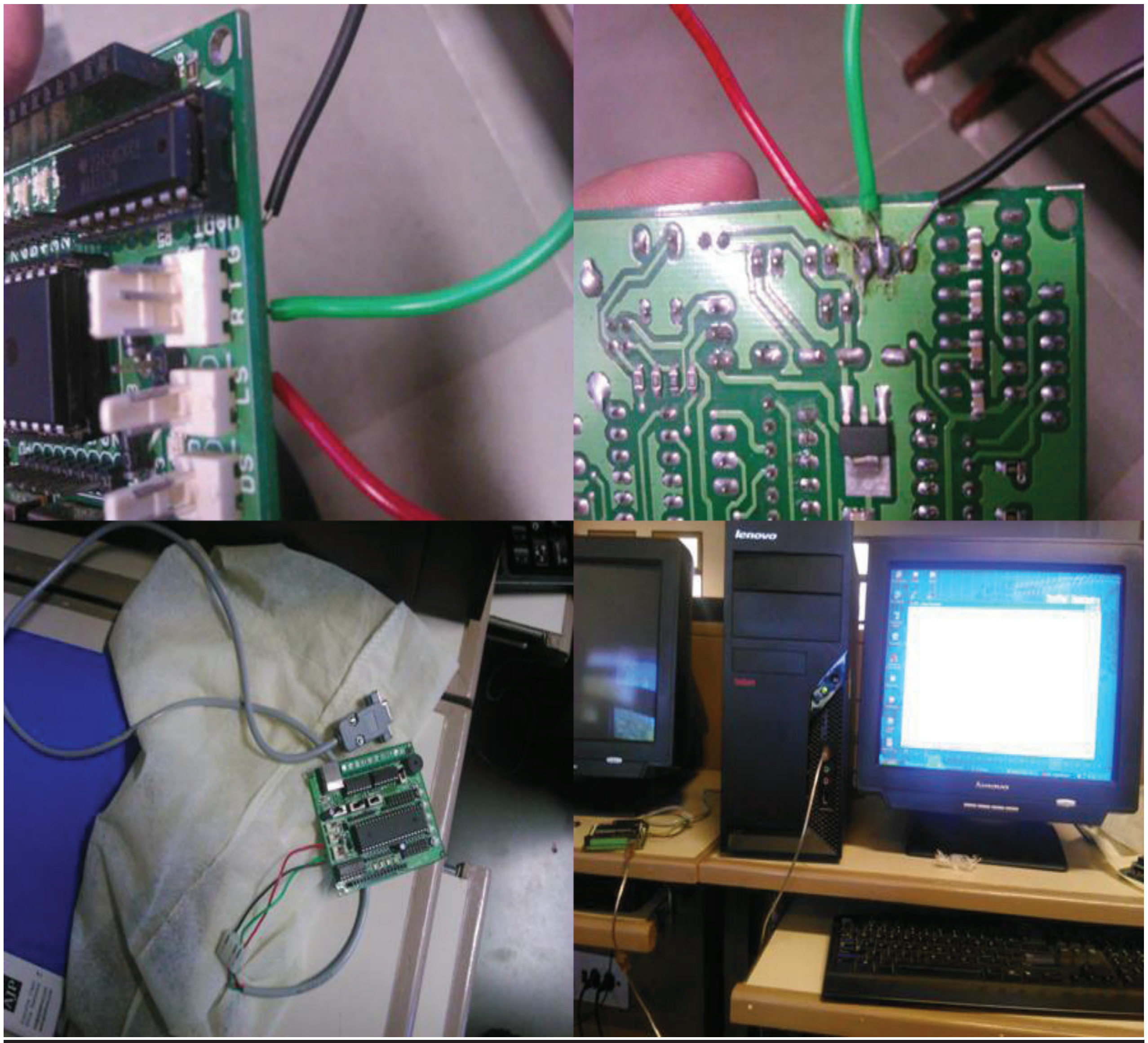


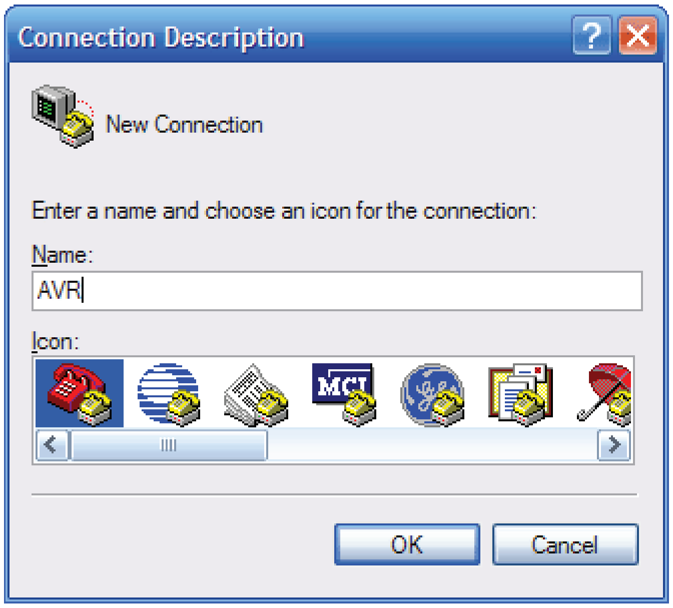
Fig. 4.5 The serial connection setup

Two different terminal programs exchange data with embedded application as follows:

1. Open HyperTerminal from

Start Menu->All Programs->Accessories->Communication>HyperTerminal

2. On startup it will ask for a connection name. Here we will enter **AVR**



Connection Description

New Connection

Enter a name and choose an icon for the connection:

Name: AVR

Fig. 4.6 Hyper Terminal Setup

3. Create new connection by clicking ok. After that select the COM1 as the port we want to use.

4. Now set up the COM port parameters as follows:

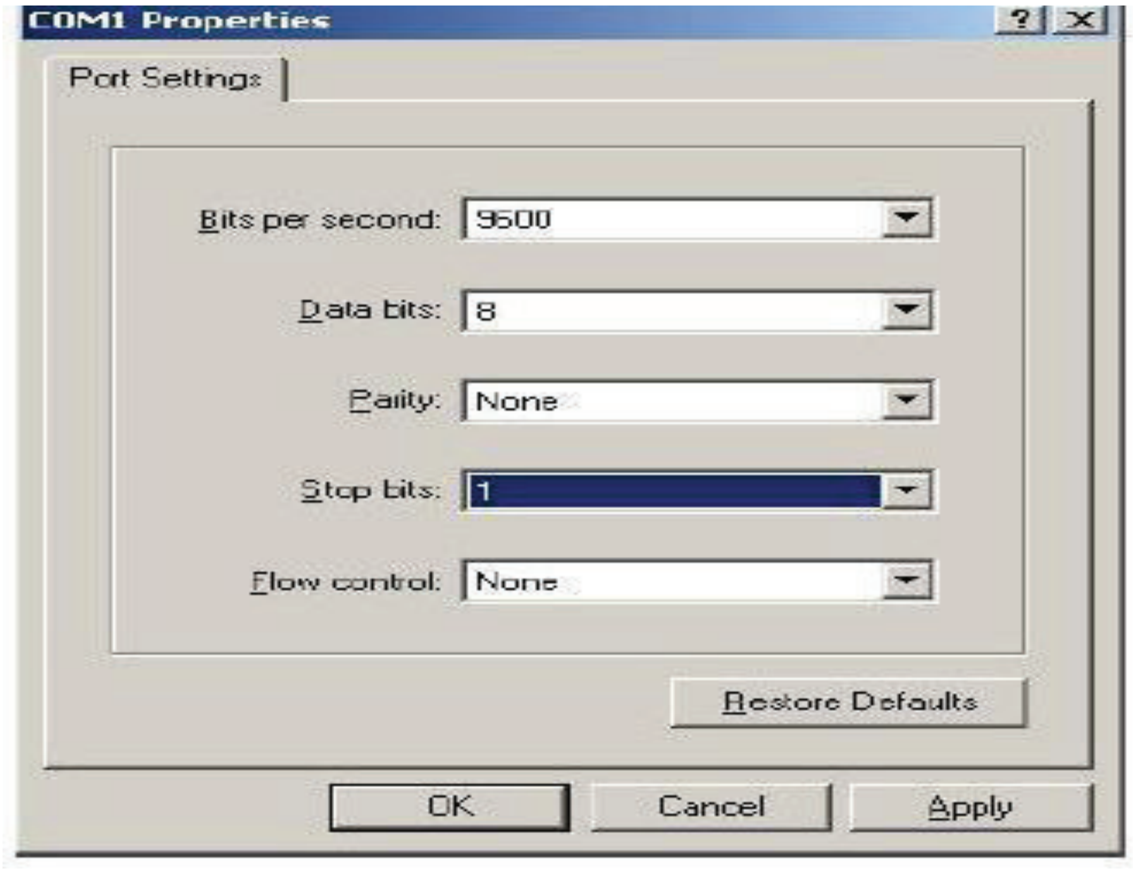
   Baud rate: 9600

   Data bits: 8

   Parity: none

   Stop bits: 1

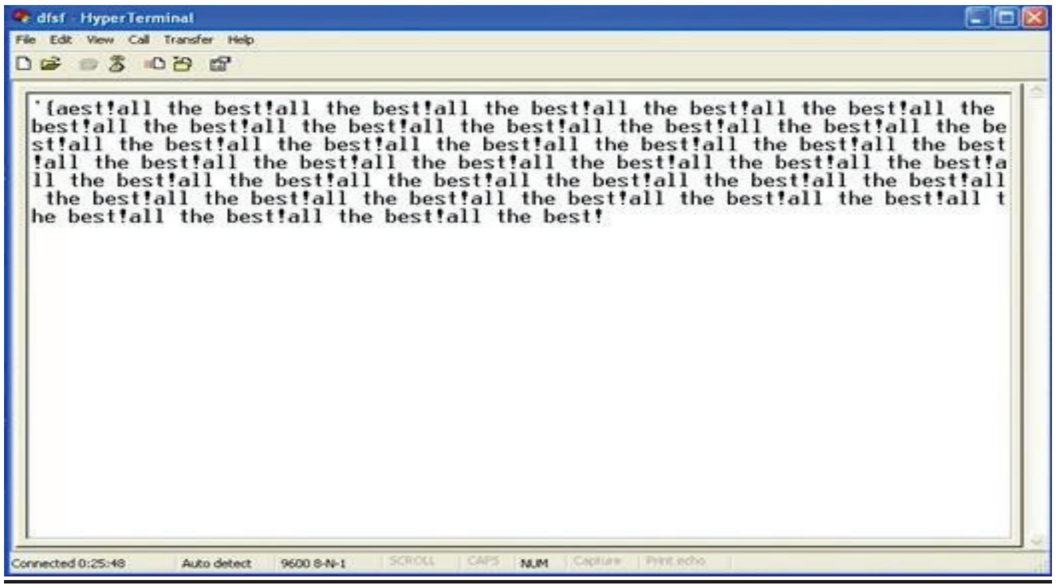
   Flow control: none



Bits per second: 9600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

Fig. 4.7 COM1 Properties

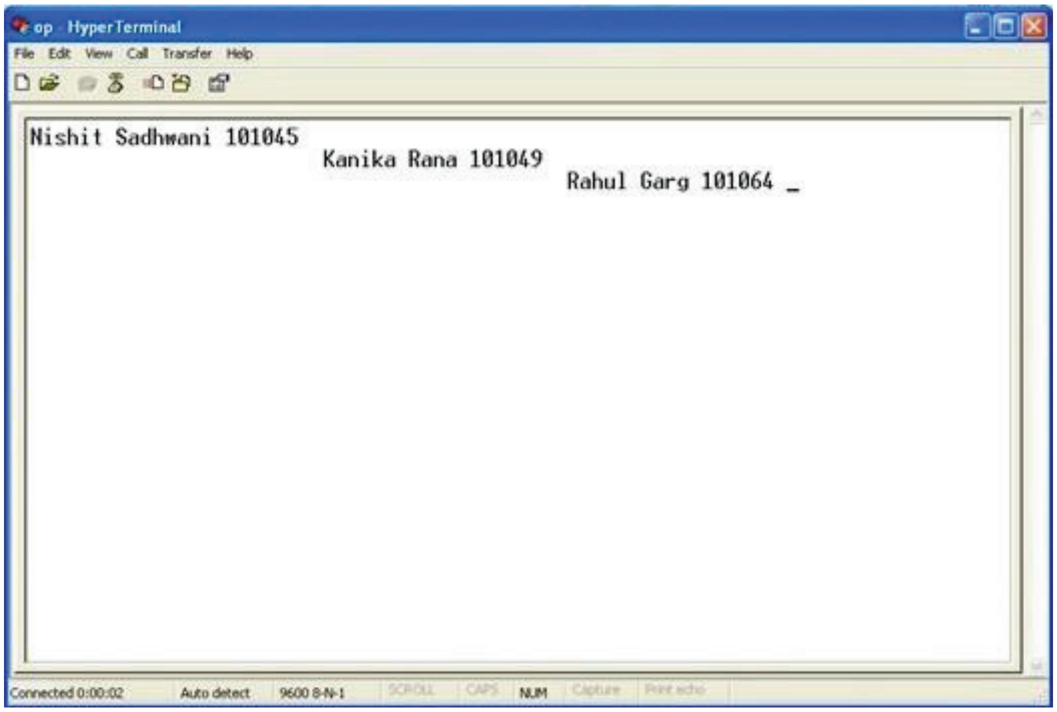
5. Hyper terminal is now ready for communication and if everything is right, the devices communicate.



```
'{aest!all the best!all the best!all the best!all the best!all the best!all the
best!all the best!all the best!all the best!all the best!all the best!all the be
st!all the best!all the best!all the best!all the best!all the best!all the best
!all the best!all the best!all the best!all the best!all the best!all the best!a
ll the best!all the best!all the best!all the best!all the best!all the best!all
 the best!all the best!all the best!all the best!all the best!all the best!all t
he best!all the best!all the best!all the best!
```

```
Connected 0:25:48   Auto detect   9600 8-N-1   SCROLL   CAPS   NUM   Capture   Print echo
```

Fig. 4.8



```
Nishit Sadhwani 101045
                          Kanika Rana 101049
                                                Rahul Garg 101064 _
```

Fig. 4.9 Data transferred from Microcontroller to Hyperterminal

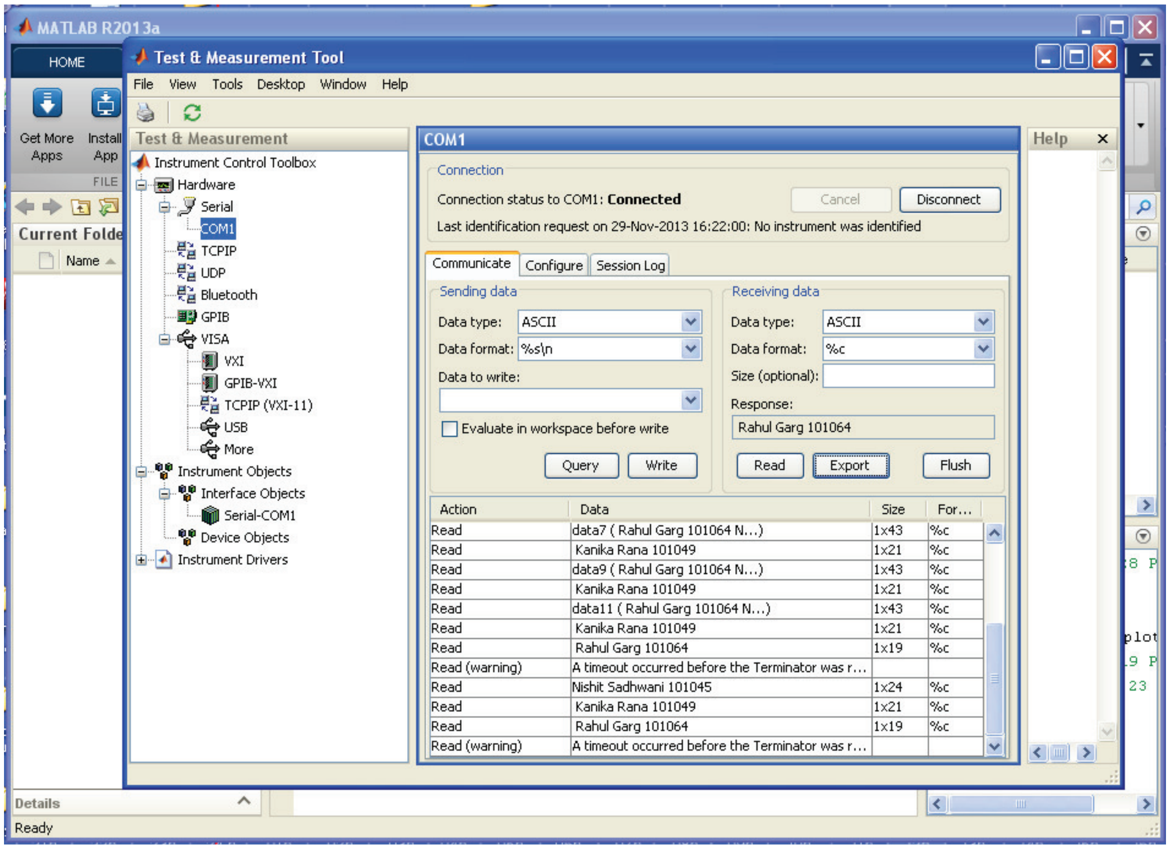
4.4.3 Serial interfacing using MATLAB

With the new control and instrument toolbox/app of MATLAB, now there is more value attached to interfacing. As a lot of operations can be performed with the data read from the controller, obtained via interfacing.

Steps for interfacing with MATLAB:

1. Open the instrument and control toolbox of MATLAB.
2. Then from the “Test and measurement tool” window, click on the “Serial” tab.
3. Then click on COM1
4. Click on the “Connect” tab
5. The click on the configure tab, and set the properties as desired. (as in the hyper terminal)
6. If all the connections and properties are set right, the device will be connected.
7. Go to communicate tab, and depending on whether we are receiving data from microcontroller or sending data to microcontroller, change the reading or writing environment.

8. Now click on read to get data from microcontroller and we can see the data read in the below tab.



COM1

Connection

Connection status to COM1: **Connected**

Last identification request on 29-Nov-2013 16:22:00: No instrument was identified

Sending data

Data type: ASCII

Data format: %s\n

Data to write:

Evaluate in workspace before write

Receiving data

Data type: ASCII

Data format: %c

Size (optional):

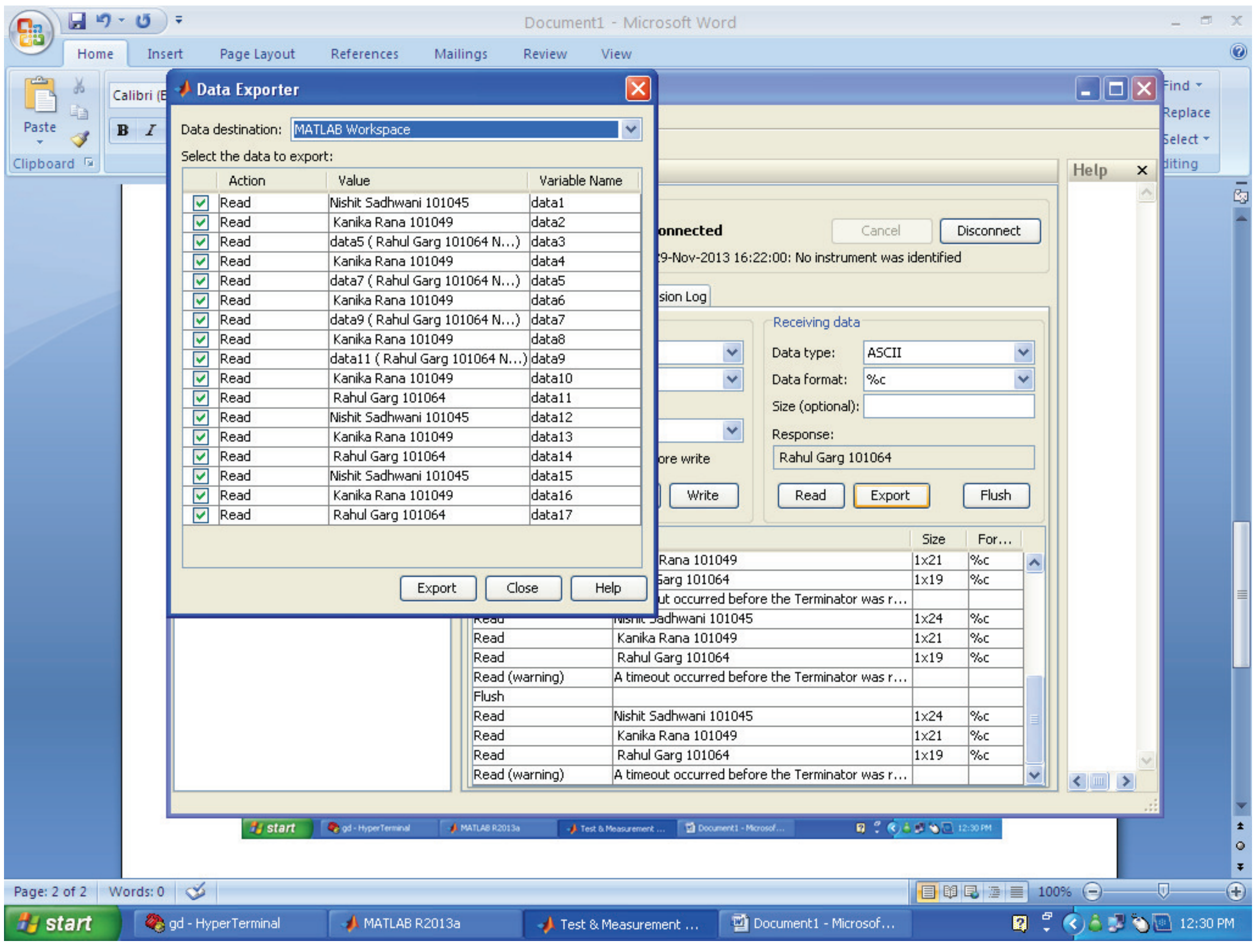
Response: Rahul Garg 101064

Action	Data	Size	For...
Read	data7 (Rahul Garg 101064 N...)	1x43	%c
Read	Kanika Rana 101049	1x21	%c
Read	data9 (Rahul Garg 101064 N...)	1x43	%c
Read	Kanika Rana 101049	1x21	%c
Read	data11 (Rahul Garg 101064 N...)	1x43	%c
Read	Kanika Rana 101049	1x21	%c
Read	Rahul Garg 101064	1x19	%c
Read (warning)	A timeout occurred before the Terminator was r...		
Read	Nishit Sadhwani 101045	1x24	%c
Read	Kanika Rana 101049	1x21	%c
Read	Rahul Garg 101064	1x19	%c
Read (warning)	A timeout occurred before the Terminator was r...		

Fig. 4.10 Data Read in MATLAB

Now, the advantage provided by MATLAB is that, the data read from the microcontroller can now be exported and used by MATLAB for other purposes.

Now click on the export button and a window pops up where you can select what all data needs to be transported.



Data destination: MATLAB Workspace

Select the data to export:

✓	Action	Value	Variable Name
✓	Read	Nishit Sadhwani 101045	data1
✓	Read	Kanika Rana 101049	data2
✓	Read	data5 (Rahul Garg 101064 N...)	data3
✓	Read	Kanika Rana 101049	data4
✓	Read	data7 (Rahul Garg 101064 N...)	data5
✓	Read	Kanika Rana 101049	data6
✓	Read	data9 (Rahul Garg 101064 N...)	data7
✓	Read	Kanika Rana 101049	data8
✓	Read	data11 (Rahul Garg 101064 N...)	data9
✓	Read	Kanika Rana 101049	data10
✓	Read	Rahul Garg 101064	data11
✓	Read	Nishit Sadhwani 101045	data12
✓	Read	Kanika Rana 101049	data13
✓	Read	Rahul Garg 101064	data14
✓	Read	Nishit Sadhwani 101045	data15
✓	Read	Kanika Rana 101049	data16
✓	Read	Rahul Garg 101064	data17

onnected

29-Nov-2013 16:22:00: No instrument was identified

Receiving data

Data type: ASCII

Data format: %c

Size (optional):

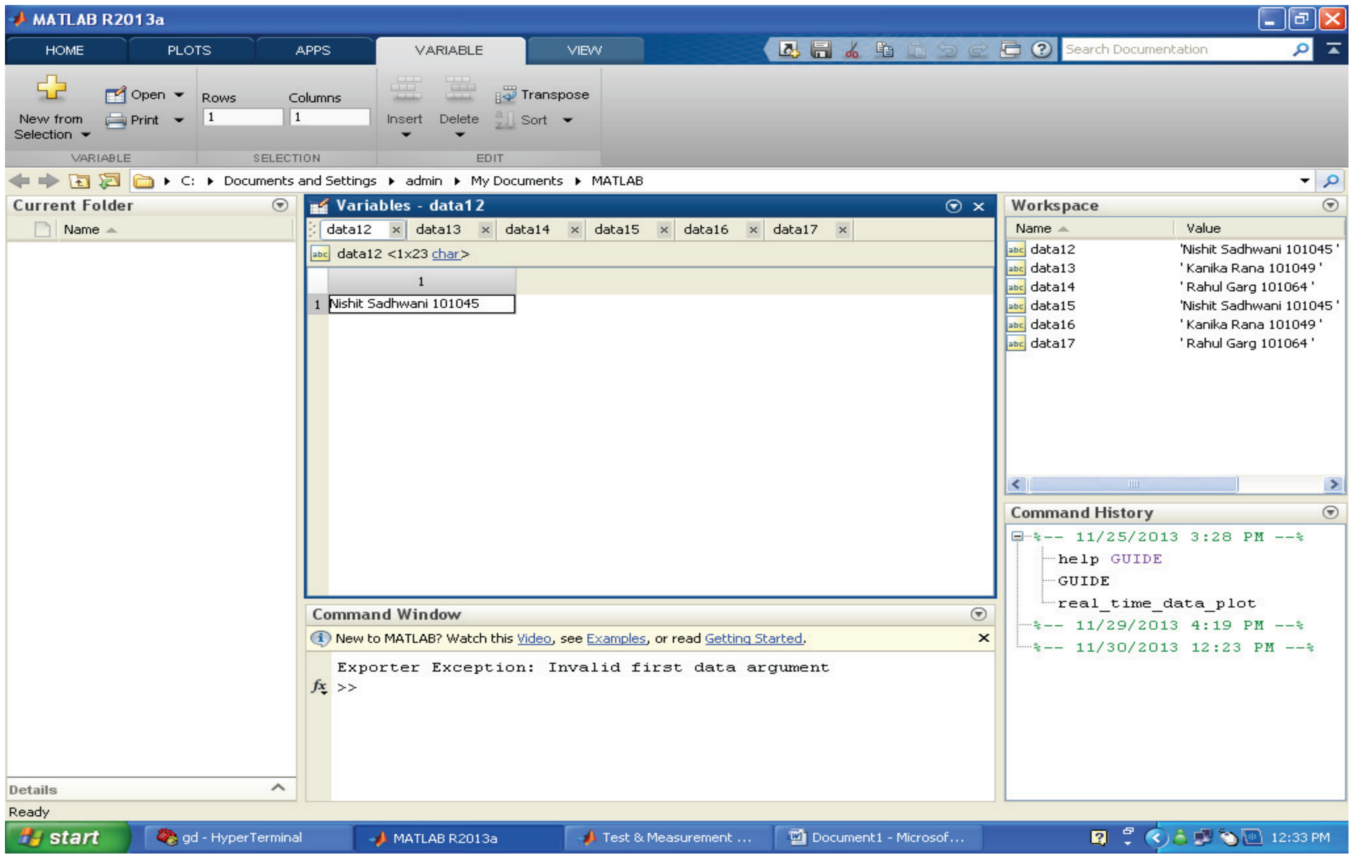
Response: Rahul Garg 101064

		Size	For...
	Rana 101049	1x21	%c
	arg 101064	1x19	%c
	ut occurred before the Terminator was r...		
	adhwani 101045	1x24	%c
Read	Kanika Rana 101049	1x21	%c
Read	Rahul Garg 101064	1x19	%c
Read (warning)	A timeout occurred before the Terminator was r...		
Flush			
Read	Nishit Sadhwani 101045	1x24	%c
Read	Kanika Rana 101049	1x21	%c
Read	Rahul Garg 101064	1x19	%c
Read (warning)	A timeout occurred before the Terminator was r...		

Fig. 4.11

From the drop down menu we can select the data destination as one of the following

1. MATLAB workspace
2. MATLAB variables
3. .mat file
4. Function
5. Structure



data12 <1x23 char>

	1
1	Nishit Sadhwani 101045

Workspace

Name	Value
data12	'Nishit Sadhwani 101045 '
data13	' Kanika Rana 101049 '
data14	' Rahul Garg 101064 '
data15	'Nishit Sadhwani 101045 '
data16	' Kanika Rana 101049 '
data17	' Rahul Garg 101064 '

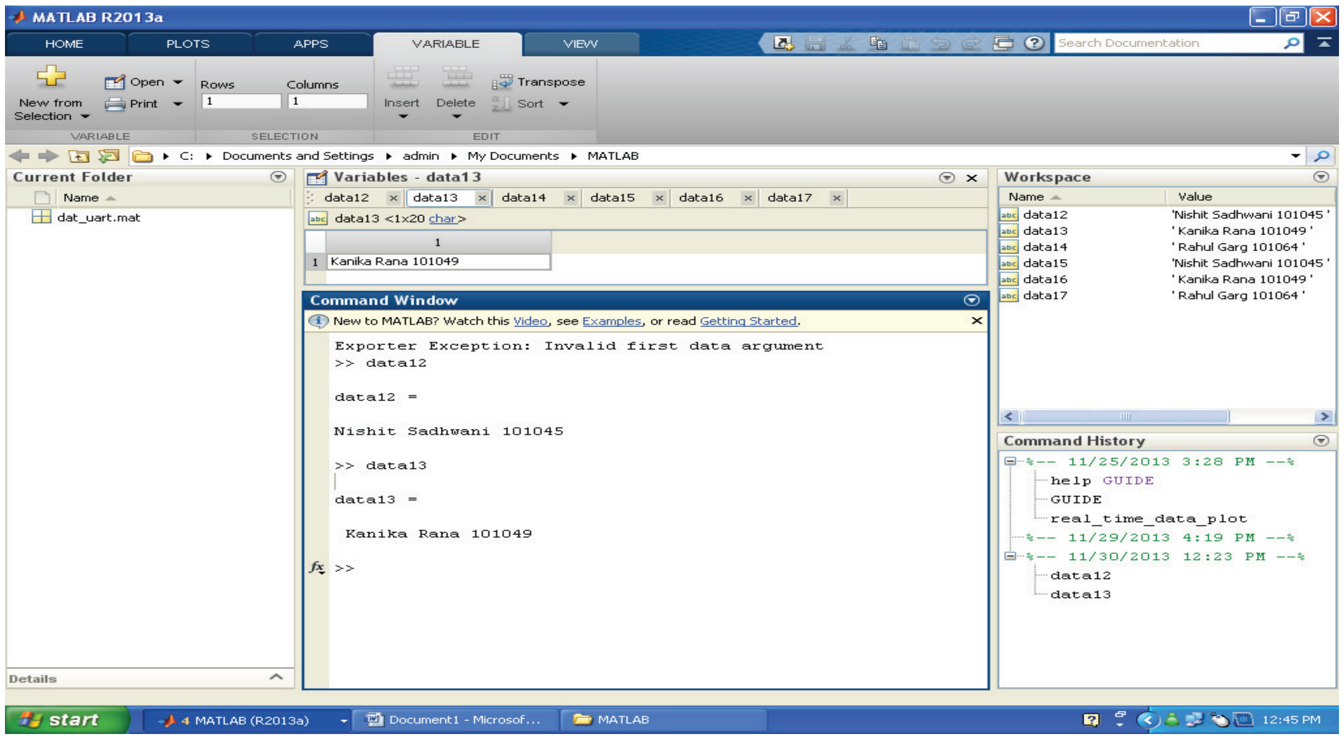
Command History

```
%-- 11/25/2013 3:28 PM --%
    help GUIDE
    GUIDE
    real_time_data_plot
%-- 11/29/2013 4:19 PM --%
%-- 11/30/2013 12:23 PM --%
```

Command Window

New to MATLAB? Watch this Video, see Examples, or read Getting Started.

```
Exporter Exception: Invalid first data argument
>>
```



data13 <1x20 char>

	1
1	Kanika Rana 101049

Workspace

Name	Value
data12	'Nishit Sadhwani 101045 '
data13	' Kanika Rana 101049 '
data14	' Rahul Garg 101064 '
data15	'Nishit Sadhwani 101045 '
data16	' Kanika Rana 101049 '
data17	' Rahul Garg 101064 '

Command History

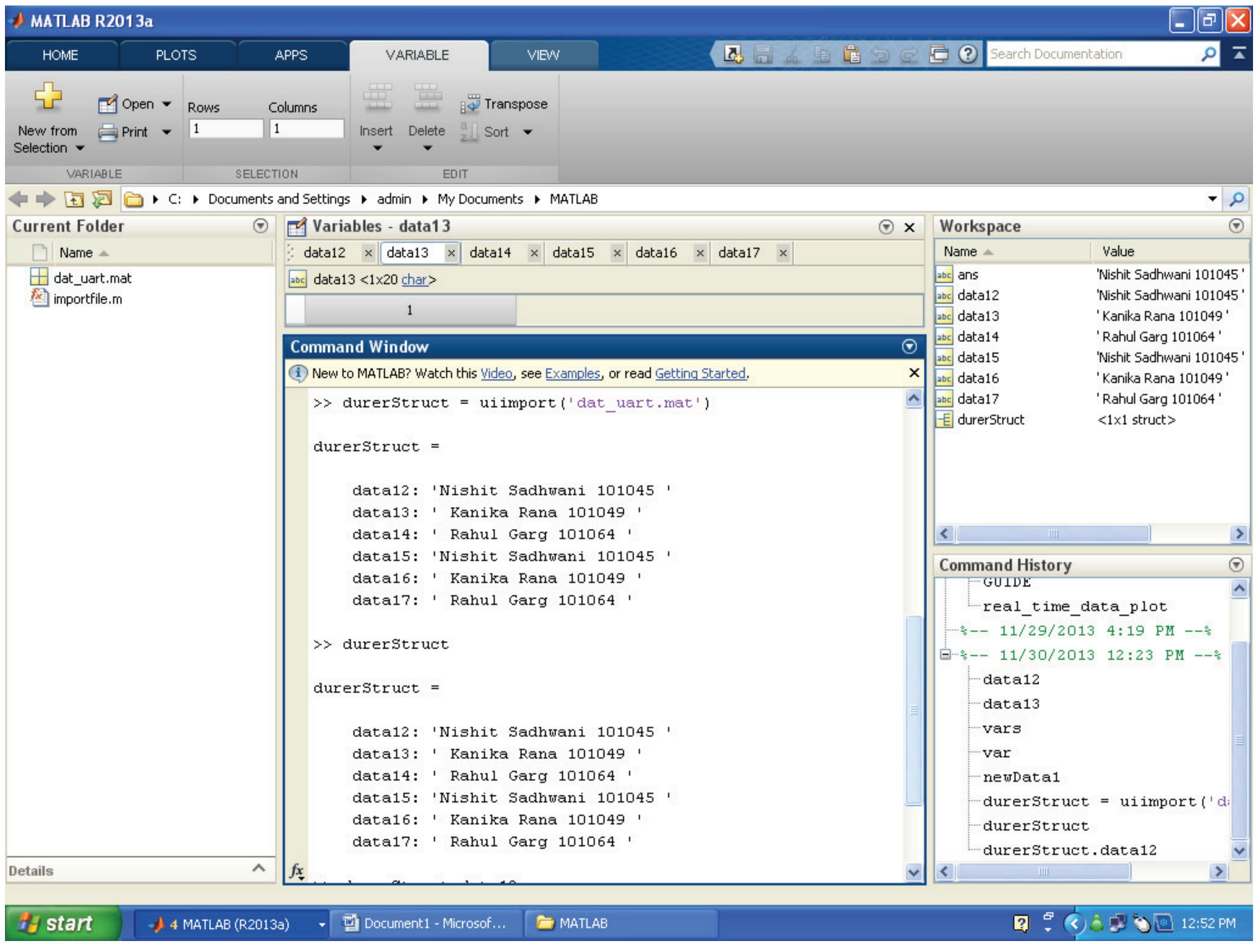
```
%-- 11/25/2013 3:28 PM --%
    help GUIDE
    GUIDE
    real_time_data_plot
%-- 11/29/2013 4:19 PM --%
%-- 11/30/2013 12:23 PM --%
    data12
    data13
```

Command Window

New to MATLAB? Watch this Video, see Examples, or read Getting Started.

```
Exporter Exception: Invalid first data argument
>> data12

data12 =

Nishit Sadhwani 101045

>> data13

data13 =

 Kanika Rana 101049

>>
```

Fig. 4.12



```
>> durerStruct = uiimport('dat_uart.mat')

durerStruct =

    data12: 'Nishit Sadhwani 101045 '
    data13: ' Kanika Rana 101049 '
    data14: ' Rahul Garg 101064 '
    data15: 'Nishit Sadhwani 101045 '
    data16: ' Kanika Rana 101049 '
    data17: ' Rahul Garg 101064 '

>> durerStruct

durerStruct =

    data12: 'Nishit Sadhwani 101045 '
    data13: ' Kanika Rana 101049 '
    data14: ' Rahul Garg 101064 '
    data15: 'Nishit Sadhwani 101045 '
    data16: ' Kanika Rana 101049 '
    data17: ' Rahul Garg 101064 '
```

Name	Value
ans	'Nishit Sadhwani 101045 '
data12	'Nishit Sadhwani 101045 '
data13	' Kanika Rana 101049 '
data14	' Rahul Garg 101064 '
data15	'Nishit Sadhwani 101045 '
data16	' Kanika Rana 101049 '
data17	' Rahul Garg 101064 '
durerStruct	<1x1 struct>

```
GUIDE
real_time_data_plot
%-- 11/29/2013 4:19 PM --%
%-- 11/30/2013 12:23 PM --%
data12
data13
vars
var
newData1
durerStruct = uiimport('d
durerStruct
durerStruct.data12
```

Fig. 4.13 Structure of the imported data using the the “uiimport” command from .mat file

4.5 Getting value from the sensors

Now that the serial communication is setup and tested, data needs to be sent to MATLAB from the sensors, and not just fixed information sent from the microcontroller as just tested above. Since the sensors used are of analog output types, it must be converted to the digital type. For this the sensors, both flex and hall are connected to the inbuilt ADC of Atmega16 (Port A). Thus understanding of the register structure of the ADC is of prime importance.

4.5.1 Understanding the register structure of the ADC of AVR

Atmega16 has an inbuilt 10-bit, 8 channel ADC system. These ADC channels are multiplexed with Port A and use the pins 33 to 40. Pin 32 is AREF, the voltage at this pin acts as the reference voltage for ADC conversion.

ADMUX

ADC Multiplexer and Selection Register

	Bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
Initial value	0	0	0	0	0	0	0	0

Bit number 7 and 6, that is REFS1 and REFS0 are reference select bits, they select the reference voltage for ADC conversion. Bit number 4 to bit number 0 select the ADC channel.

00	AREF
01	V_{acc}
10	reserved
11	2.56V (internal reference voltage)

0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7

The use of ADLAR will be discussed in the next register.

ADCSRA

ADC Control and Status Register

	Bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADSP2	ADSP1	ADSP0
Initial value	0	0	0	0	0	0	0	0

ADEN- ADC Enable Bit

set to 1 for turning on ADC

ADSC- ADC start conversion bit

Set to 1 to start ADC conversion, and as soon as conversion is completed, it is automatically set back to 0 by hardware.

ADATE – ADC Auto Trigger Enable

Set to 1 to enable auto triggering

ADIF – ADC Interrupt flag

This bit is set to 1, when ADC conversion gets complete.

ADIE – ADC Interrupt Enable

Set to 1, if we want to activate the ADC conversion complete interrupt.

ADPS[2:0]- these bits are used to set the ADC clock frequency, i.e- these bits determine the division factor by which the microcontroller clock frequency is divided to get the ADC clock frequency.

ADC clock frequency = crystal frequency/ prescaler

ADCH and ADCL

ADC data registers

When the ADC conversion is complete, the data is stored in these registers. Since 10-bit can be obtained, but only 8-bit registers available, two registers called ADC high and ADC low. The configuration of these registers depend on ADLAR value.

When ADLAR= 0 (data is right adjusted)

ADCH	-	-	-	-	-	-	ADC9	ADC8
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0

When ADLAR=0 (data is left adjusted)

ADCH	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
ADCL	ADC1	ADC0	-	-	-	-	-	-

If both registers are to be read, that is precision is 10-bit then ALWAYS read ADCL before ADCH. Else if 8 bit precision is sufficient then read ADCH after setting ADLAR=1

4.5.2 Interface analog device with AVR Atmega16:

1. To initialize ADC
   - (i) Set value in ADMUX register to select the desired channel (where the sensors are connected) and set the reference voltage.
   - (ii) Set the prescaler bits in ADCSRA register such that we now have an ADC clock frequency
   - (iii) Set the ADEN bit to enable the ADC.

Now ADC is initialized and ready to start converting

2. To read analog value
   - (i) Put the channel value in ADMUX
   - (ii) Start the conversation by setting the ADSC bit
   - (iii) Monitor the ADIF bit for conversion complete.
   - (iv) Clear the conversion bit ADIF
   - (v) Digital converted result is now available in ADCH and ADCL registers.

The ADC conversion for the project was achieved without interrupt and was found to give satisfactory results.

4.5.3 Techniques of ADC of microcontrollers

There are mainly four different types of techniques for performing Analog to Digital Conversion (ADC) for a microcontroller, which are:

(i) Successive Approximation

This techniques using a DAC, a controller and a comparator to perform the ADC process. Starting from the MSB, down to the LSB, the controller turns on each bit at a time and generates an analog signal, with the help of the DAC, to be compared with the original analog input signal. Based on the result comparison, the controller changes or leaves the current bit and turns on the next MSB. The process continues until decision are made for all available bits. Thus the strategy is to narrow down to the answer by partitioning the available range into two equal parts in every turn.

(ii) Integration

This technique uses an integrator, a comparator and a controller to convert the analog signals to digital signals. The sampled analog signal and a fixed reference signal are integrated over a fixed period of time and then compared. When the two integrated values equal, the measured time is converted into a digital encoded value.

(iii) Counter- based conversion

Requires a counter, a DAC and a comparator. The counter starts at 0 and counts up. As the counter counts up, the corresponding value is converted to an analog value and compared with the analog input. As long as the analog input is greater than the signal generated by DAC, the counter keeps counting up. When the DAC generated analog signal is greater than the analog input, the counter value is converted a digital value representing the sampled analog input signal.

(iv) Parallel Conversion

The parallel converter uses a large number of comparators and circuitry to simultaneously measure the input signal and convert it to a digital value. This techniques allows the quickest conversion, but then is very costly as well.

Atmega16 employs successive approximation technique for the Analog to Digital Conversion and thus the little lag observed in the conversion is justified by the use of this technique.

4.5.4 Sensor calibration

Calibration means linking your real world data with the virtual data. When threshold are to be created, such that below a particular value route A is to be followed and above that particular value route B is to be followed, then to define that ‘particular value’ needs to be physically set and that is achieved by calibration.

The sensor is tested for all the conditions and the corresponding ADC values received are recorded, to obtain a function defining the relation between ADC value and the position of the sensor. Comparing these values and analyzing them, we can easily set the threshold.

The result obtained is the equation for obtaining the degree of bent from the voltage received, for 3 different pivots of bend for the finger.

4.6 3-D objects using Constructive Solid Geometry (CSG)

The scheme is to represent each elementary solid with a 3-D field of values; negative on inside and positive on outside. The CSG operations of union, intersection and subtraction then become simple minimum/maximum operations, on the 3D field. All volume-shapes are generated as 3D scalar fields. Since fields are really arrays, all fields which are to be combined must have the same number of elements. The resolution parameter associated with CSG objects sets the number of elements and thus may be the same for all CSG objects that will be combined using operations. After all the operations are performed, the volume object is then converted to surface for rendering. Objects may be scaled after they are converted to surfaces.

CHAPTER-5
RESULTS AND CONCLUSION

First of all, serial communication between microcontroller and computer was established. Then ADC of microcontroller was studied and checked. Then the major part was to figure out the connecting link between ADC and UART communication of the microcontroller; that is, sensors or potentiometers are connected to the ADC port of the microcontroller, and then change in the value of sensors results in change in the analog voltage input, which in turn results in change in the digital value generated by the ADC for the corresponding input voltage. This digital value is stored in the data registers of ADC, and the link for UART communication is established separately. The task now is to take the value from the sensor and transmit it via UART, thus to establish a link between the data registers of UART and ADC, by simply equating the data register of UART (to be sent) to the data register of ADC. This results in successfully getting the sensor value to the computer. Written a code in MATLAB to obtain these values and plot them in real time, or store them as variables or import them as .mat files.

The calibrated value of sensors is analyzed and the values indicating the required position is fed to the microcontroller, such that now when that value of sensor is obtained, the already fed array in microcontroller with a few additional parameters is directly sent to MATLAB and the object is plotted.

The final result is that with the movement of the user’s hand wearing a haptic glove, the movement of holding, holding and lifting, and not holding the object are depicted in MATLAB in real time with a little lag, where the object is virtual 3D object.

On the other hand, since reality is very important, the movement of a Real Robotic arm is controlled by the user wearing an exoskeleton. Which basically translates the values of potentiometers into the driving force for controlling the motor. As the hand moves, the value of potentiometer changes, thus the digital value given by the ADC is used by the microcontroller to drive the motors.

Hence both real (robotic arm) and virtual (MATLAB objects) objects have successfully been controlled by haptic transducer which are worn on the hand of the operator.

CHAPTER-6
CHALLENGES FACED

It is almost impossible that something is put to task and there are no challenges faced or problems encountered. Here are the problems we faced, some we overcame, but some couldn't and hence looked for alternate methods.

1. We found that flex sensors are expensive and would take up the entire cost of the project, and thus thought of making them ourselves. We found a lot of sites showing how to make flex sensors on the cheap, and a common element in them was ESD bags. It was difficult to understand what exactly is an ESD bag, are they truly what we think they are, and then make others, especially vendors understand what it is we are really looking for. Then find where we can find them. When we finally asked for our ESD bags to be couriered, it was found that the presence of bubble wrap over it made its working inappropriate. We did not give up and found another way, this time they worked, but were not accurate and their working wasn't were clear to us. So finally we dropped the idea of making them on our own but purchase them instead, and this time not online.
2. We were almost lost when we were working on the interfacing, as nobody we knew had done it, or the ones that had done it, did it for PIC. We asked almost all teachers of all departments, but did not know what was wrong. We had done everything step-by-step, learned about the AVR registers, learned about db9, cross connection, checked the cable, changed computers, tried new soft wares, everything we thought could possibly be faulty, but we could not find anything and there was still no communication. Finally as of some magic happened, we trial all permutations and combination by hit and trial, and by slight modifications in the code and pressing the reset button, we had our output and that moment joy knew no boundaries. Then we rechecked it to make sure it was not a fluke, and understood the working even more carefully and came out successfully.
3. The combination of flex sensors and Atmega16 was a deadly combination as there were very less references available for the above, but it was a blessing in disguise as the majority of the learning has been because of this only.

CHAPTER-7
FUTURE PROSPECTS

As of now, the haptic glove just controls the motion of the virtual objects with two degree of freedom. This can be extended to give an impression of holding a virtual object, by adding motors or force sensors which execute the feedback path to stop the movement of the hand, such that the user cannot close the hand any further giving the feeling that an object is being held. This gives the dimensionality of ‘feeling’ a virtual object.

The robotic arm is a basic model to see the implementation, a smaller and sophisticated version of this robotic arm can be made to perform accurate movements.

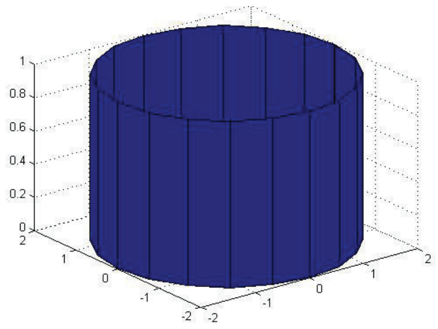
Using a wireless module instead of the wired model, it can be used for remote operation increasing the usability of the arm and importance.

APPENDICES

Appendix A1: MATLAB Codes for creating 3-D objects

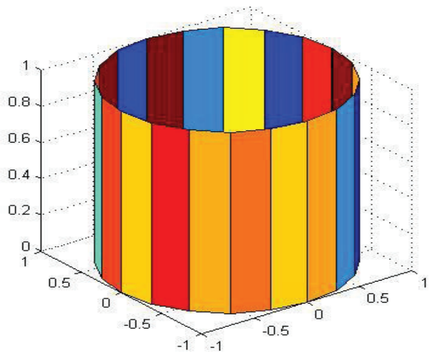
Cylinder

```
clear all
clc
[X,Y,Z] = cylinder(2);
%cylinder(2);
%plot3(X,Y,Z);
surf(X,Y,Z);
```



Multi-Colored Cylinder

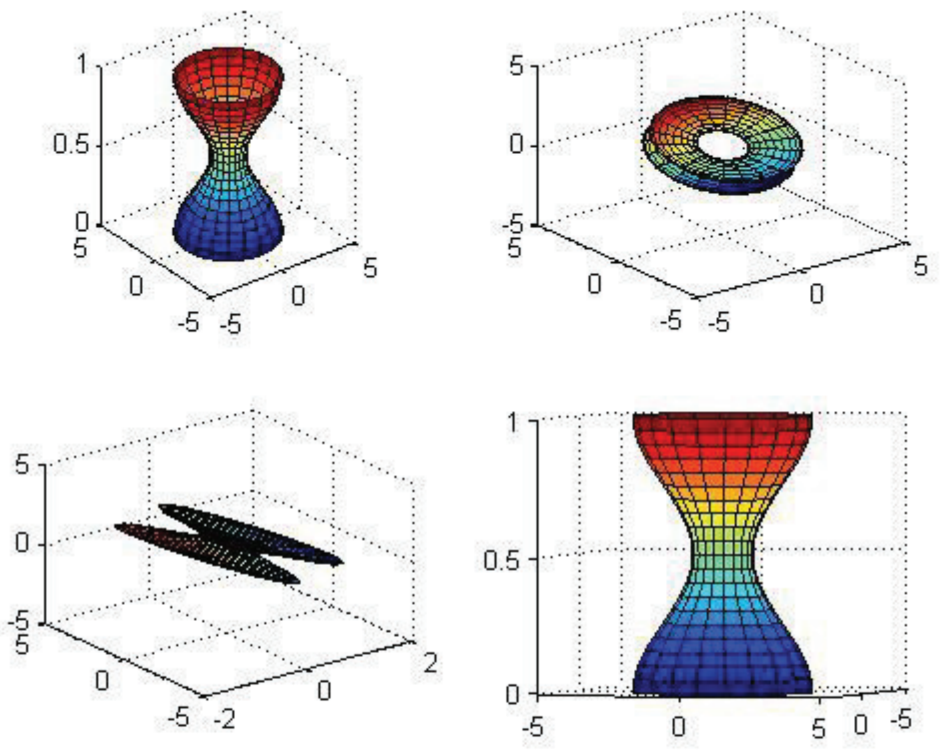
```
clear all
clc
cylinder
axis square
h = findobj('Type','surface');
set(h,'CData',rand(size(get(h,'CData'))))
```



Hour Glass Shape

```
clear all
```

```
clc
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(sin(t))
surf(X,Y,Z)
axis square
```



Cube

```
clc;
formatcompact
set(gcf,'Menubar','none','Name','Cube', ...
'NumberTitle','off','Position',[10,350,300,200], ...
'Color',[0.3 0.1 0.3]);
h(1) = axes('Position',[0.2 0.2 0.6 0.6]);
vert = [1 1 1; 1 2 1; 2 2 1; 2 1 1 ; ...
        1 1 2;1 2 2; 2 2 2;2 1 2];
fac = [1 2 3 4; ...
    2 6 7 3; ...
    4 3 7 8; ...
    1 5 8 4; ...
    1 2 6 5; ...
    5 6 7 8];
patch('Faces',fac,'Vertices',vert,'FaceColor','r');  % patch function
light('Position',[1 3 2]);
light('Position',[-3 -1 3]);
```

```
materialshiny;
alpha('color');
alphamap('rampdown');
camlight(45,45);
lightingphong
view(30,30);
```

Sphere

```
h(3)= axes('Position',[0.17 0.17 0.4 0.4]);

[XsYsZs]=sphere(30);
hs3 = surf(Xs, Ys, Zs);
set(hs3,'EdgeColor','none', ...
'FaceColor','g', ...
'FaceLighting','phong', ...
'AmbientStrength',0.3, ...
'DiffuseStrength',0.8, ...
'SpecularStrength',0.9, ...
'SpecularExponent',25, ...
'BackFaceLighting','lit');
camlightright;
hiddenoff;
axisequal;
set(h,'Visible','off')
```

Appendix A2 : Basic ATmega code

```
#include<avr/io.h>              //This is the header for AVR Microcontroller.
#include <util/delay.h>         //header file to generate time delay.

int main(void)
{
 PORTC=0X00;                              // PortC initialization as all bits low.
 DDRC=0xf8;                     // PortC data direction declaration as output.
 while(1)
 {
 PORTC=0x10;         //LED 1 is on
 _delay_ms(5);       // waits for 5 ms

 PORTC=0x00;    // LED 1 is off
 _delay_ms(5);       //wait 5 ms
 PORTC=0x20;         //LED  2 is on
 _delay_ms(5);       //wait for 5 ms
 PORTC=0x00;         //LED 2 is off
 _delay_ms(5);       //wait for 5 ms
 PORTC=0x40;         //LED 3 is on
 _delay_ms(5);       //wait for 5 ms

 PORTC=0x00;         //LED 3 is off
 _delay_ms(5);       //wait for 5 ms
 PORTC=0x88;         //LED 4 and buzzer is on
 _delay_ms(5);       //wait 5 ms
 PORTC=0x00;         //everything is off
 _delay_ms(5);
 }
 }
```

Appendix A3 : Final MATLAB code for Virtual Object Control

```
Tthresh = 30;

% Setup the background
% 1. Make Table
% 2. Make Hands
% 3. Plot Table and Hands
figure(1)
close
clf

%make a table
table=UnitCube;
table.facecolor=[222/255, 191/255, 150/255]; % A light brown wood-ish color
table.facelighting='flat';
table.edgecolor=[175/255, 124/255, 54/255]; % A darker color for outline
table=scale(table,2,2,.2);
table=translate(table,0,0,-1.2);

%make hand (namely little sticks)
cylHand = UnitCylinder(10);
L1 = 1;
L2 = 1;
radius = 0.03;
arm1 = translate(scale(cylHand,radius,radius,L1/2),-.57,0,L1/2);
arm1 = rotateX(arm1, 90);
arm1 = translate(arm1, -.25, 0, -.4);
arm1.facecolor = 'blue';
arm2 = translate(scale(cylHand,radius,radius,L2/2),.55,0,L2/2);
arm2 = rotateX(arm2, 90);
arm2 = translate(arm2, .25, 0, -.4);
arm2.facecolor = 'green';
hand = combine(arm1, arm2);

%plot table and hands
background = combine(table, hand);
renderpatch(background);
axis off;
axis([-2, 2, -2, 2, -4, 4]);
grid on
daspect([1 1 1])
light('position',[10,-10,10])
%Do a persptective transform
set(gca,'projection','perspective')
set(gca,'CameraViewAngle',6)
%The frame background color
```

```
set(gcf,'color', [183/255, 248/255, 1])
xlabel('x');ylabel('y');zlabel('z');
view(7,20)
drawnow;

disp('Initializing serial...')
initialized=0;
mcu=serial('COM4',...
    'Baudrate',9600,...
    'Stopbits',1,...
    'Parity','none',...
    'FlowControl','none');
fopen(mcu)
  i=0;
try
    while (true)
        disp('Getting serial input...')

        tline = fgetl(mcu);
        result = sscanf(tline, '%f');
        disp(result);

         if numel(result)==8
            objectSelected = result(1);
            dIn = result(2);
            dOut = result(3);
            isSolid = result(4);
            isFull = result(5);
            objectGripped = result(6);
            objectLifted = result(7);
            height = result(8);

            % scale dimensions to fit on the Matlab Screen
            dIn = dIn/20;
            dOut = dOut/20;

            % Foreground with the object and stuff
            % 1. Get input from the MCU
            % 2. Draw object if necessary
            % 3. Move object if necessary
            % 4. Update
            % Parameters inputted: inner diameter, outer diameter, temperature, object
            % selected?  object gripped?  displacement from table, hasHandle, isSolid,
            % isFull, object lifted?
            if (objectSelected)
                res=20;
```

```
% make a closed end cylinder
cyl1=CSGcylinder(0,0,0,dOut,'z',res);
cube1=CSGcube(0,0,-dOut+0.05,dOut+0.05,res);
body=CSGintersection(cyl1,cube1);
if (~isSolid)
   % subtract by a smaller cylinder
   cyl2=CSGcylinder(0,0,0,dIn,'z',res);
   cube2=CSGcube(0,0,-dIn+0.05,dIn+0.05,res);
   hole=CSGintersection(cyl2,cube2);
   body=CSGsubtract(body,hole);
end
object = body;

objectSurface = CSGtoSurface(object, res);

if(isFull)
   cylFull = UnitCylinder(2);
   lengthFull = dOut*0.4;
   radiusFull = dIn - 0.02;
   inContent     =     translate(scale(cylFull,radiusFull,radiusFull,lengthFull-.02),.01,0,-
lengthFull/1.8);
   inContent.facecolor = [6/255, 249/255, 0];
   objectSurface = combine(objectSurface, inContent);
end

if(objectGripped)
   arm1 = translate(arm1, (-dOut - max(arm1.vertices(:,1))), 0, 0);
   arm2 = translate(arm2, (dOut - min(arm2.vertices(:,1))),0, 0);
   objectSurface = combine(objectSurface, arm1, arm2);

   if(objectLifted)
      movingObject = translate(objectSurface, 0, 0, height);
      scene = combine(table, movingObject);
   else
      scene = combine(table, objectSurface);
   end
else
   scene = combine(table, objectSurface);
end

figure(1);
clf
renderpatch(scene);
axis off;
axis([-2, 2, -2, 2, -4, 4]);
grid on
daspect([1 1 1])
```

```
            light('position',[10,-10,10])

            %Do a persptective transform
            set(gca,'projection','perspective')
            set(gca,'CameraViewAngle',6)

            %The frame background color
            set(gcf,'color', [183/255, 248/255, 1])
            xlabel('x');ylabel('y');zlabel('z');
            view(7,20)
            drawnow;

        else
            figure(1)
            clf
            %plot table and hands
            background = combine(table, hand);
            renderpatch(background);
            axis off;
            axis([-2, 2, -2, 2, -4, 4]);
            grid on
            daspect([1 1 1])
            light('position',[10,-10,10])
            %Do a persptective transform
            set(gca,'projection','perspective')
            set(gca,'CameraViewAngle',6)
            %The frame background color
            set(gcf,'color', [183/255, 248/255, 1])
            xlabel('x');ylabel('y');zlabel('z');
            view(7,20)
            drawnow;
        end % if(objectSelected)
      end % if nuel(result) == 6;
   end % while(true)

catch
   fclose(mcu)
   disp('Serial closed')
   disp(lasterror.message)
end
```

References

[1] Haptic- basic knowledge:
http://net.educause.edu/ir/library/pdf/eli7029.pdf

[2] Work Done By Other People:
http://www.disneyresearch.com/project/surround-haptics-immersive-tactile-experiences
strokeSleeve ,Karlin Bark, Frank Tan.
http://haptics.seas.upenn.edu/index.php/Research/TactileFeedbackForRehabilitation
Haptography: Heather Culbertson, Juan Jose Lopez Delgado
http://haptics.seas.upenn.edu/index.php/Research/Haptography

[3] Arduino Module:
http://arduino.cc/en/Main/arduinoBoardUno

[4] 3-D objects and modeling:
http://www.nbb.cornell.edu/neurobio/land/PROJECTS/Hierarchy/

[5] ATmega 16:
http://www.atmel.in/Images/doc8154.pdf

[6] Flex Sensors:
http://mech207.engr.scu.edu/SensorPresentations/Jan%20%20Flex%20Sensor%20Combined.pdf
http://www.youtube.com/watch?v=yOV17hp1Ulw
http://hackaday.com/2012/02/28/building-a-flex-sensor-from-component-packing-materials/
http://www.instructables.com/id/DIY-Bend-Sensor-Using-only-Velostat-and-Masking-T/

[7] RS 232:
http://www.engineersgarage.com/articles/what-is-rs232

[8] USART:
http://www.engineersgarage.com/embedded/avr-microcontroller-projects/serial-communication-atmega16-usart

[9] ADC:
Atmel: AVR Microcontroller Primer: programming and interfacing – google books

[10] Computer Graphics:
Introducing Fundamentals of Computer Graphics Using MATLAB