

Haptic Glove

Project Report submitted in partial fulfillment of the requirement for the
degree of

Bachelor of Technology.

in

Electronics and Communication Engineering

under the Supervision of

Prof. T.S.Lamba

By

Rahul Garg 101064

to



Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled “HAPTIC GLOVE”, submitted by Rahul Garg in partial fulfillment for the award of degree of Bachelor of Technology in Electronics and Communication Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date:

Supervisor’s Name: Prof. T.S. Lamba

Designation: Dean (Academic&Research)

Acknowledgement

We take this opportunity to express our profound gratitude and deep regards to our guide (Mentor) **Prof T.S.Lamba** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall carry us a long way in the journey of life on which we are about to embark. We are obliged to all our faculty members of JUIT, for the valuable information provided by them in their respective fields. We are grateful for their cooperation during the period of our project. Lastly, we thank almighty, our parents, brothers, sisters and friends for their constant encouragement without which this project would not be possible.

Date: 31st July 31, 2014

Name of students:

Rahul Garg

Table of Content

<i>S.No</i>	<i>Topic</i>	<i>Page No.</i>
	Abstract	6
1.	Chapter-1 : Introduction to Haptic	7
1.1	Haptics	7
1.2	Current application of Haptics	7
1.3	General Working of Haptics	8
1.4	Significance of Haptics as Technology	8
1.5	Problems in Haptics	9
1.6	Future of Haptics	9
1.7	Implications of Haptics on teaching and learning	9
2.	Chapter-2 : Software	11
2.1	MATLAB	11
2.2	Arduino IDE	11
3.	Chapter- 3 :Hardware	13
3.1	Arduino UNO R	13
3.1.1	ATmega 16	14
3.2	Flex Sensors	16
3.3	Hall Effect Sensor	17
4.	Chapter-4 : Work Done	19
4.1	Making 3-D objects using MATLAB	19
4.2	Basic Arduino programs	20
4.3	Making Flex Sensors	20
4.4	Interfacing Microcontroller with computer	28
4.4.1	Understanding the register structure of USART of AVR	28
4.4.2	Serial Interfacing using Matlab	32
4.4.3	Serial Interfacing using MATLAB Alternate	36
5.	Chapter-5 : Conclusions and Challenges Faced	37
5.1	Conclusion	37
5.2	Problems Faced	37
5.3	Future Prospects of Haptics	38
8.	Appendices	39
A1	MATLAB Codes for creating 3-D objects	39
9.	References	51

List of Figures

<i>S.No</i>	<i>Title</i>	<i>Page No.</i>
1.	Chapter-1 : Introduction	
1.1	Basic Working	7
2.	Chapter-2 : Software	
2.1	Arduino IDE	11
3.	Chapter- 3 :Hardware	
3.1	Arduino UNO R	13
3.2	Atmega-16 Pin Diagram	15
4.	Chapter-4 : Work Done	
4.1	Working in Matlab	19
4.2	Preparation for making Flex sensor	21
4.3	Procedure of making Flex sensors	22
4.4	Flex sensors using Method 1	23
4.5	Flex sensors and readings using Method 2	25
4.6	Flex Sensor	26
4.7	Taking Reading From Flex Sensors	27
4.8	Readings	28
4.9	Data read in Matlab	33
4.10	Showing serial interface	34
4.11	COM6 Properties	35
4.12	Data Transmitted From Microcontroller to Hyper Terminal	36

Abstract

The objective of our project is to make a “HAPTIC GLOVE” which is a wearable device that will provide tactile sensation through feedback of virtual objects.

Elaborating it further, it is a normal wearable glove, which has a few sensors (primarily flex and force sensors) connected to the microprocessor, such that the movement of the hand reflected in the changing values of the sensors. The 3-D virtual objects are generated through MATLAB. Now the main aim is that when we now try and hold this virtual object, we can actually “feel” the object, which is not present in reality. If the object we are holding is say a sphere, after enclosing the sphere in the hand, we will not be able to close the hand completely, hence giving the effect of holding a real object.

Although many people don't realize it, tactile feedback is as important as any of the other four major senses. It's not just about feeling a soft cloth or a furry pet, but the tactile sensations help us accomplish everyday tasks. Ever try tying your shoelaces while your hands are numb? Your hands haven't lost their dexterity, but without the tactile sensation, the task suddenly becomes a lot more difficult.

We divided the project into three parts: software, hardware and interfacing. The MATLAB part consists of an MCU serial interface (to obtain object properties) and the 3-D objects created. To draw objects from basic 3D shapes as well as animate them in various ways. We use this as the visual feedback of our system where the user can see the object that he/she is holding, get information about the shape of the object and see it be picked up.

CHAPTER-1 : Introduction to Haptics

1.1 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 on the other hand incorporates both touch(tactile) and motion (Kinesthetic) elements.

For applications that simulate real physical properties such as weight, momentum, friction, texture or resistance, haptics communicates these properties through interfaces that let users ‘feel’ what is happening on the screen.

1.2 Current application of Haptics

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.

1.3 General working of Haptic Devices

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 users fingers.



Fig 1.1
Basic Working

Advantage : Because properties can be changed easily, the 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.

1.4 Significance of Haptics 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 Problems In haptics technology

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 Future of Haptic Technology

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 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.

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.

In our project we are using **MATLAB** for basically one purpose that is to provide Graphical Interface to person. It is being used to create various 3D objects that we need for our project. All these 3D objects need to be dynamic so they can be moved.

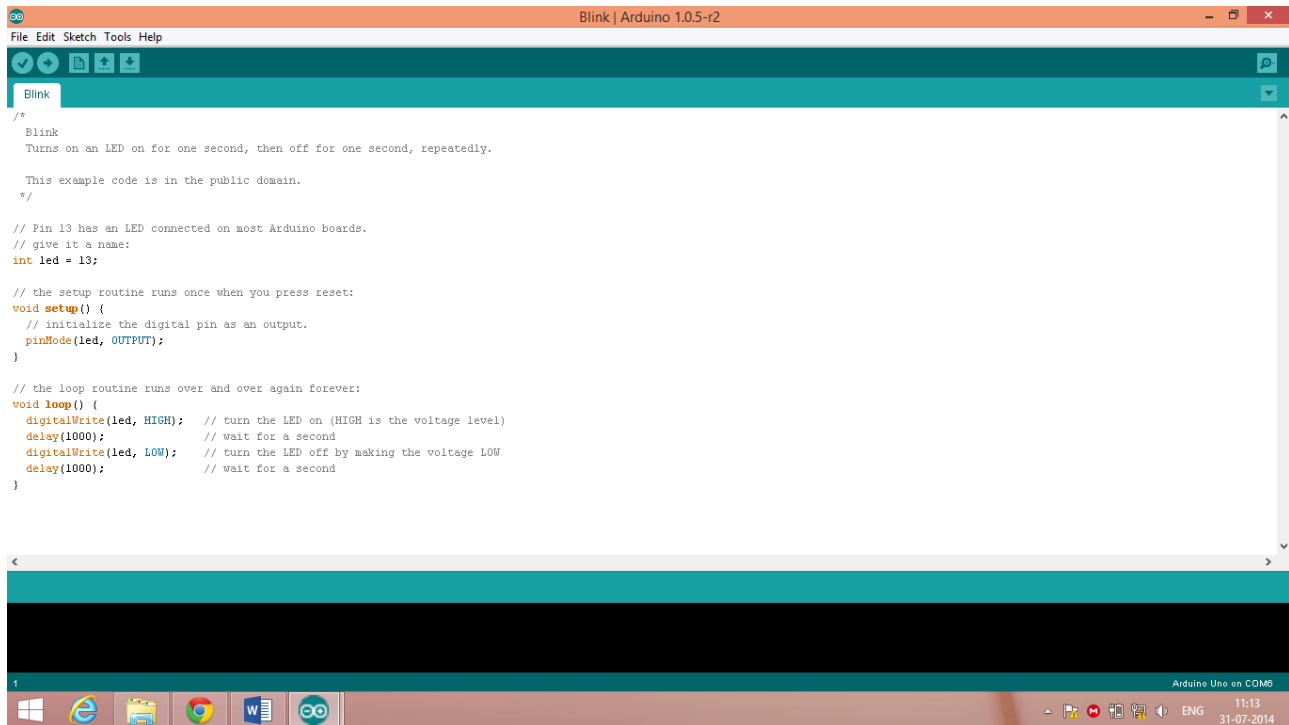
Another thing which **MATLAB** is doing is receiving Signal from our microprocessor, and according to that signal it moves our object in its GUI.

2.2 Arduino IDE

Arduino IDE is a suite of executable, open source software development tools for the Arduino uno R microprocessor hosted on the Windows platform. It includes the GNU GCC compiler for C and C++.

We can directly compile ,create and upload program on our board using IDE and a usb.

We are using Arduino IDE for programming our microprocessor. It tells our our Microprocessor how to behave. In Arduino IDE we directly write a program in it and it is uploaded on our microprocessor using a usb.

The image shows a screenshot of the Arduino IDE software. The window title is "Blink | Arduino 1.0.5-r2". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for opening files, saving, and running. The main text area contains the following code:

```
/*  
 * Blink  
 * Turns on an LED on for one second, then off for one second, repeatedly.  
 *  
 * This example code is in the public domain.  
 */  
  
// Pin 13 has an LED connected on most Arduino boards.  
// give it a name:  
int led = 13;  
  
// the setup routine runs once when you press reset:  
void setup() {  
  // initialize the digital pin as an output.  
  pinMode(led, OUTPUT);  
}  
  
// the loop routine runs over and over again forever:  
void loop() {  
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)  
  delay(1000); // wait for a second  
  digitalWrite(led, LOW); // turn the LED off by making the voltage LOW  
  delay(1000); // wait for a second  
}
```

The bottom of the window shows a Windows taskbar with icons for Start, Internet Explorer, File Explorer, Google Chrome, Microsoft Word, and the Arduino IDE. The system tray on the right shows the date and time as "11:13 31-07-2014" and the language as "ENG".

Fig 2.1 Arduino Ide

CHAPTER-3: Hardware

3.1 Arduino UNO R

We have used an Arduino uno r 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. One switches is provided. The board also has 1 LED's

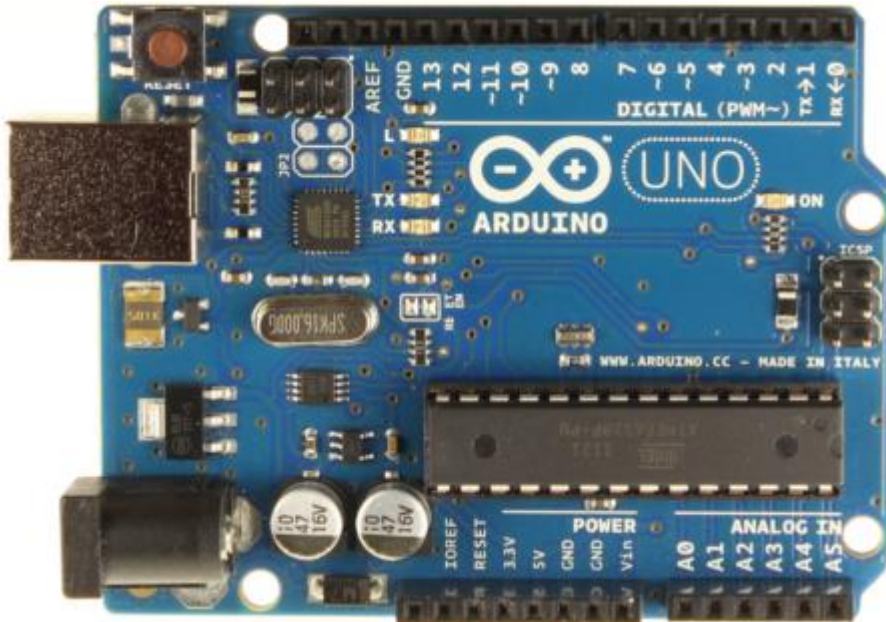


Fig 3.1
Arduino UNO R Development Board

This Board is used for connecting our sensors with our pc. All the signals which are generated in MALAB from pc is processed through this board and is transmitted to the sensors, also the signals produced by sensors are processed on this board and then is sent to MATLAB in our PC.

3.1.1 ATmega -16

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.

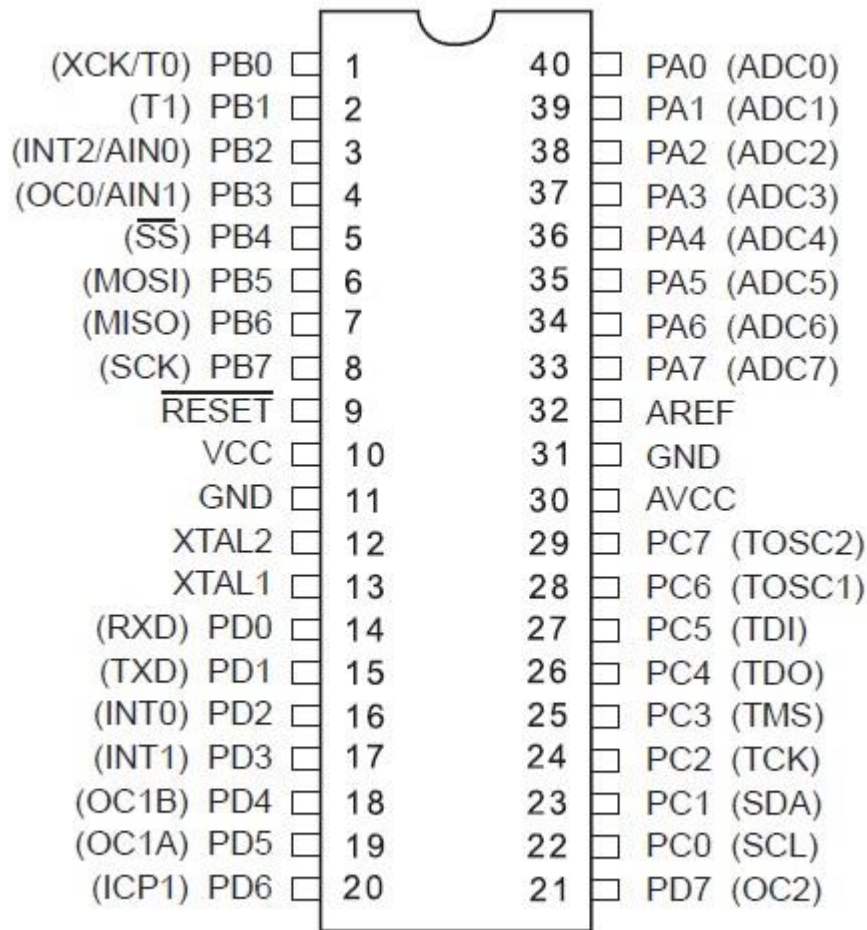


Fig 3.2
Atmega-16 Pin Diagram

It is placed on our Arduino UNO Board. Everything which we processing on Board is being processed here. To use it we have to program it. It is programed through Arduino Ide. It contains a small flash memory where the program is stored. It works according to the program which we program into it. It is very important part of the board. Every input and output of our board is connected to it

3.2 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. The more you bend the sensor the more resistance they provide. They are available in market in form of thin strips of varying lengths. They can be made unidirectional or multi directional.

Flex sensors are basically devices that convert physical parameter's to analog electrical signal. They are analog resistors and work as analog voltage dividers. There are carbon resistive elements within a thin flexible substrate. When this substrate is bent the sensor produces output according to this bent radius. 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.

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.

In our project we are using Flex Sensors to determine the diameter of the object in our Glove. The change in resistance as we are moving or finger is being projected through this sensor. As we move our finger this sensor is being bent. Due to this bend the resistance is changed. Due to change in resistance the voltage changes. This change in voltage is being given to our microprocessor which is further sent to our MATLAB. Then according to this signal our MATLAB can know the diameter our hand and it works accordingly.

3.3 Hall Effect sensor

A Hall Effect sensor is a transducer that varies its output voltage in response to a magnetic field. Hall Effect sensors are used for proximity switching, positioning, speed detection, and current sensing applications.

In its simplest form, the sensor operates as an analog transducer, directly returning a voltage. With a known magnetic field, its distance from the Hall plate can be determined. Using groups of sensors, the relative position of the magnet can be deduced.

Frequently, a Hall sensor is combined with circuitry that allows the device to act in a digital (on/off) mode, and may be called a switch in this configuration. Commonly seen in industrial applications such as the pictured pneumatic cylinder, they are also used in consumer equipment; for example some computer printers use them to detect missing paper and open covers.

When a beam of charged particles passes through a magnetic field, forces act on the particles and the beam is deflected from its straight line path. The beam of charged particles refers to the electrons flowing through a conductor. When a current carrying conductor is placed in a magnetic field perpendicular to the path of the electrons, the electrons are deflected from its straight line path. Therefore, one side of the conductor becomes negative portion and the other side becomes positive one. The transverse voltage is measured and is known as Hall Voltage.

The charge separation continues until the force on the charged particles from the electric field balances the force produced by magnetic field. If the current is constant, then the Hall voltage is a measure of the magnetic flux density. There are two forms of Hall Effect Sensors. One is linear where the output voltage linearly varies with the magnetic flux density. The other is known as threshold where there is a sharp drop of output voltage at a particular magnetic flux density.

In our project we are using Hall Effect sensor to check the lift of our hand from the surface. We put a magnet on the surface and move our hand above it. When we move our hands up and down on top of that magnet, there is change in the flux through the Hall Effect sensor. This change is converted into voltage which is then transferred to the microprocessor which processes the signal and send it to MATLAB.

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.

I also made a surface for table, two cylinders to depict our fingers. I have also interfaced our microcontroller with MATLAB

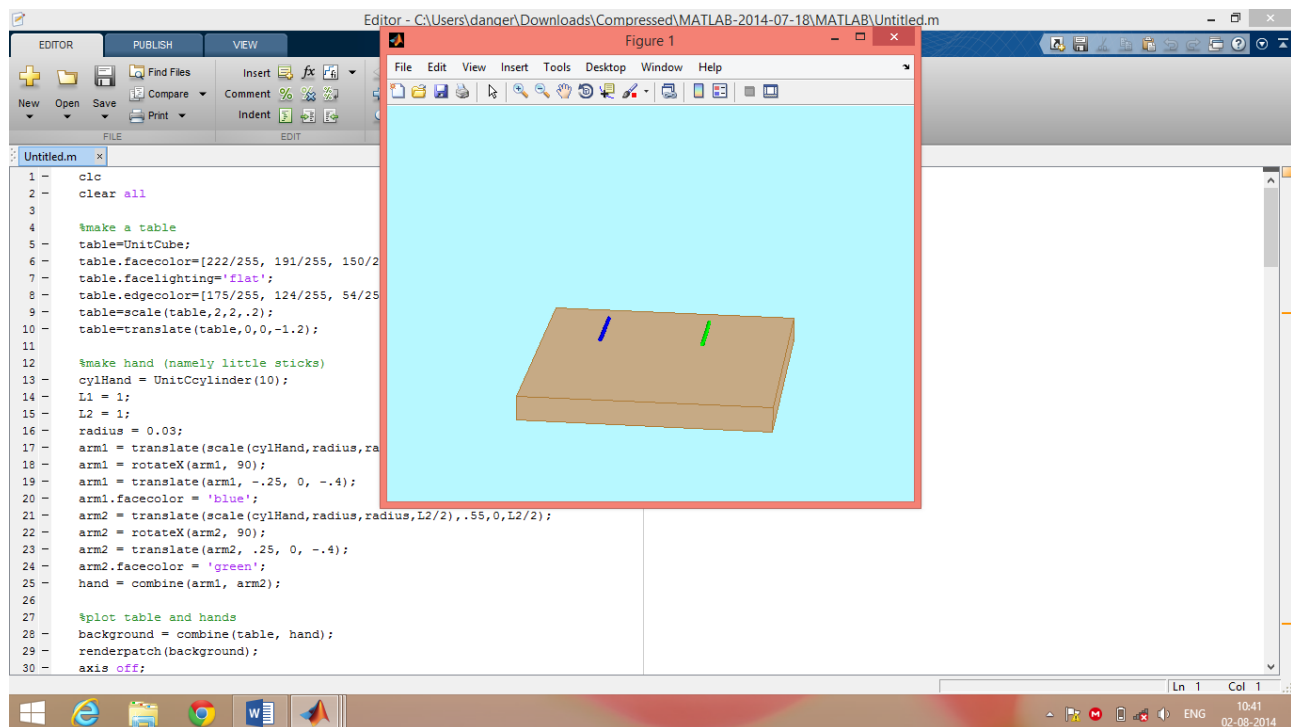


Fig 4.1 Working in MATLAB

Now the input is taken from Arduino. And we can select the virtual object which we wanted. Then we wait for signal from Arduino. After signal is received we can see the changes in MATLAB.

4.2 Basic Arduino Programs

We wrote a few basic programs like blinking of LED's and working of motors using 'Arduino ide' programming language, just to get familiar with the development board and its working. We wrote code so that our microcontroller can work with MATLAB and can be serially interfaced with it. It connects various sensors to our pc so that we can know what is happening. It defines how our microcontroller should work. Arduino has various pins on its board. We program it according to the pins in which we connect our sensors. We directly assign value to various pins so that it can work as needed. We program it in a way that it is serially interfaced with our pc i.e. MATLAB.

4.3 Making Flex Sensors and usage

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.

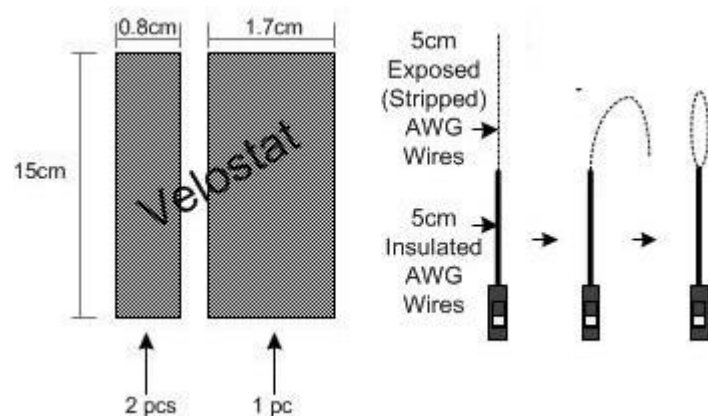


Fig 4.2
Preparation for making Flex sensors

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-center 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.

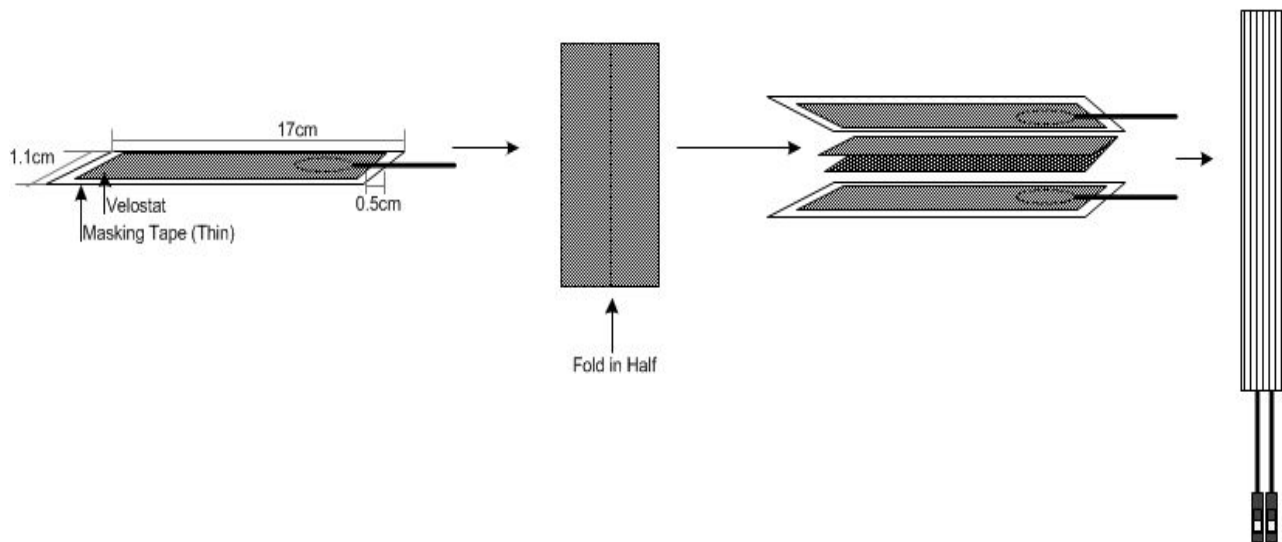


Fig 4.3
Procedure of making Flex sensors

Shown below (**Fig 4.4**) are the flex sensor and intermediate steps in making of flex sensors by us using Method 1.



Fig 4.4
Flex sensors 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-center 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 5**.



Fig 4.5
Flex sensors and readings using Method 2

Now we got many problems with our homemade Flex sensors therefore we had to purchase Flex sensors from market

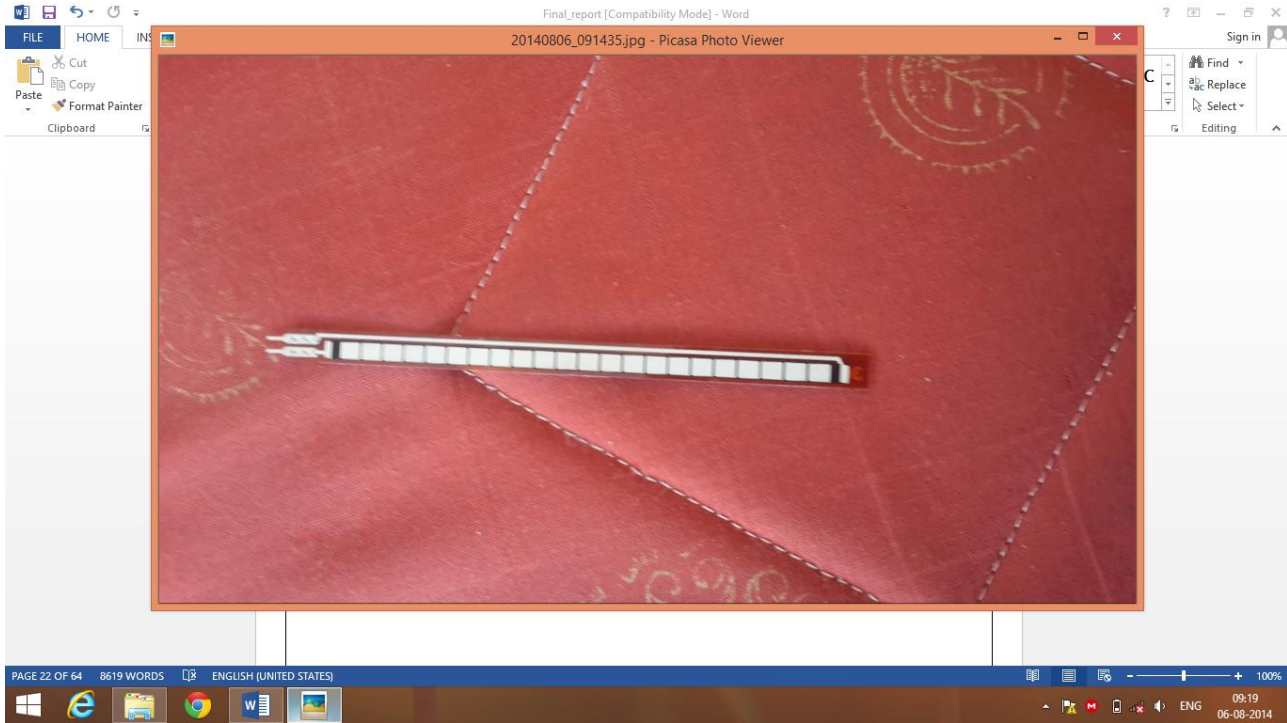


Fig 4.6
Flex sensors

After that we Calculated the value of voltage across our flex sensor for different angles it was bent by attaching a resistor of 22 kilo ohms in series with the sensor. We measure the voltage across the terminals of Flex sensor.

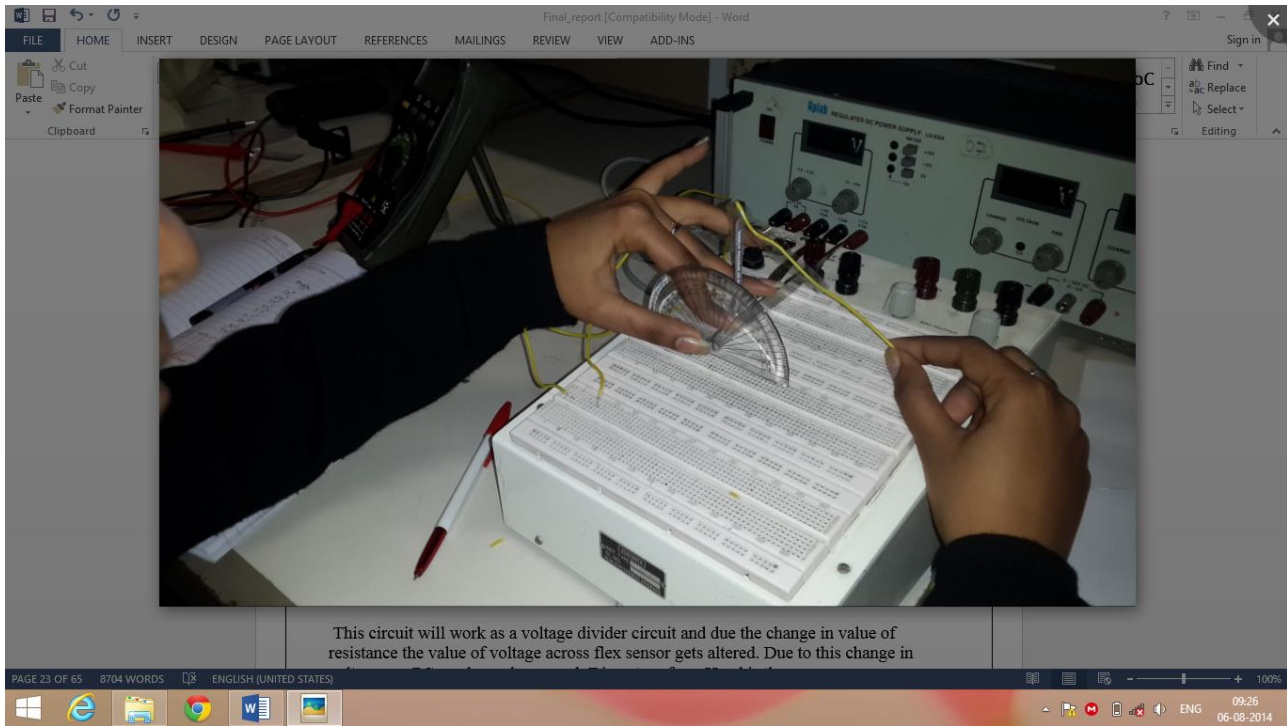


Fig 4.7
taking readings from Flex sensor

This circuit will work as a voltage divider circuit and due the change in value of resistance the value of voltage across flex sensor gets altered. Due to this change in voltage our PC can know how much Diameter of our Hand is there. Voltages for various angles have been depicted in the diagram below.

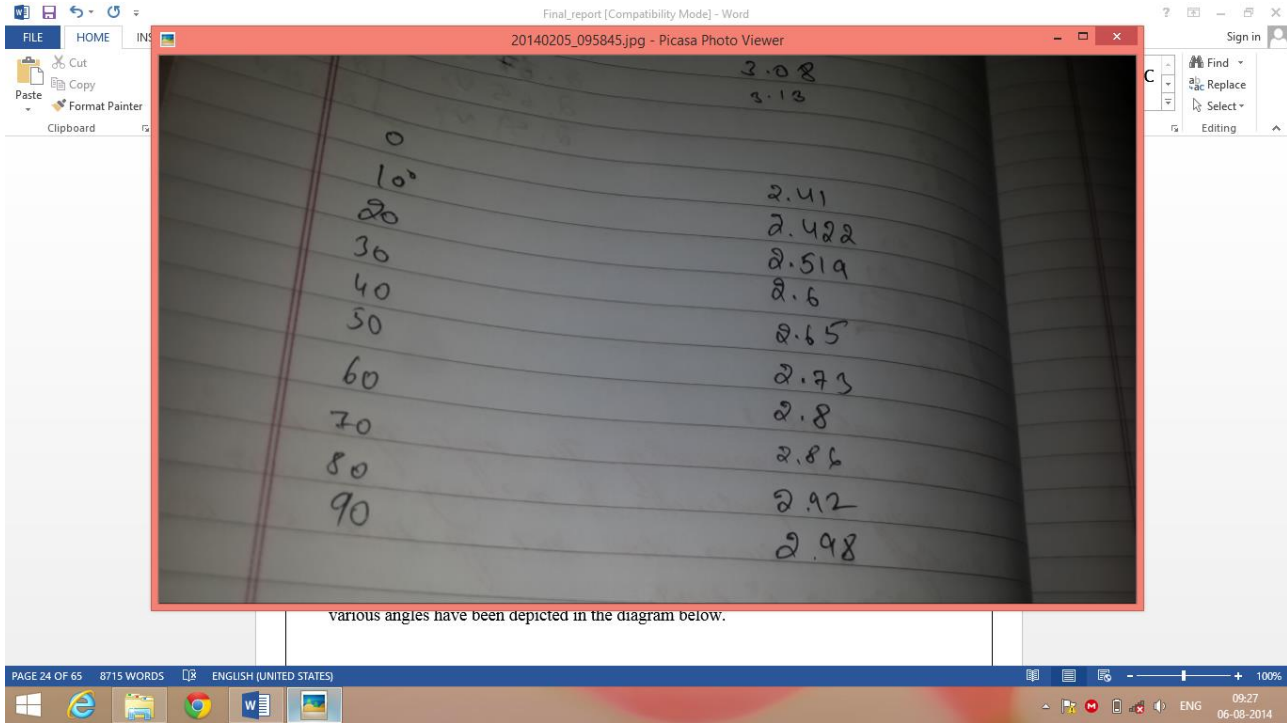


Fig 4.8
readings

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 mode 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 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 COM6.

4.4.2 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 COM6
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.

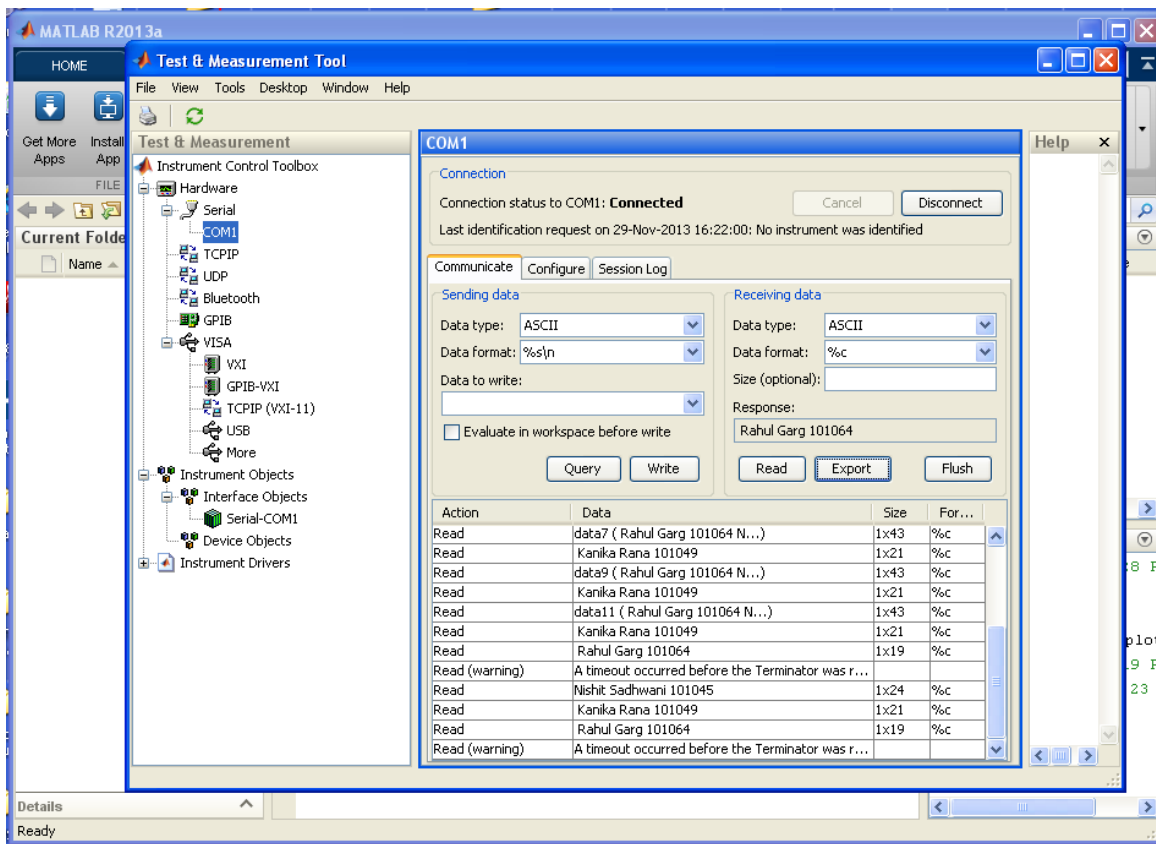


Fig 4.9
Data Read in MATLAB

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.

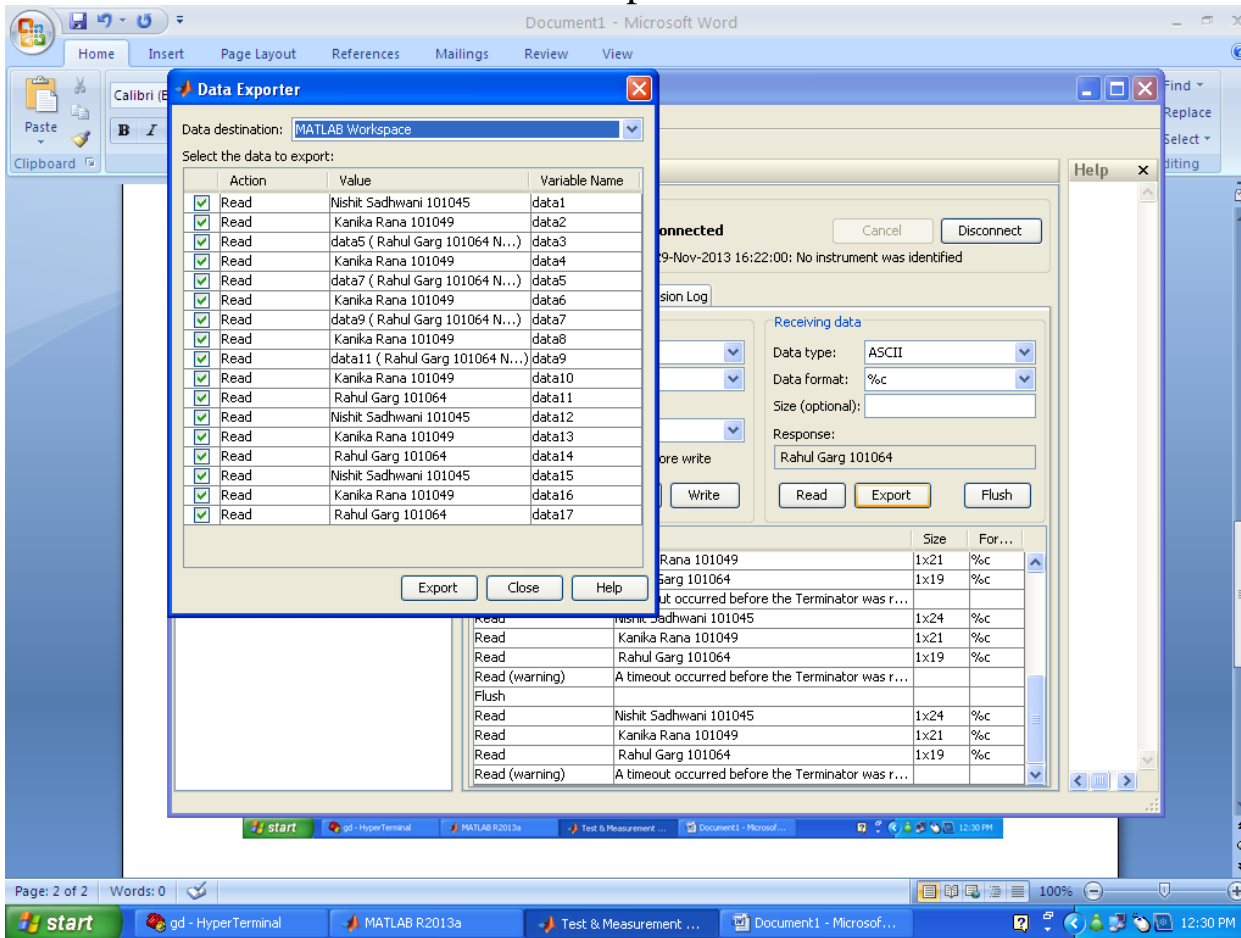


Fig 4.10

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

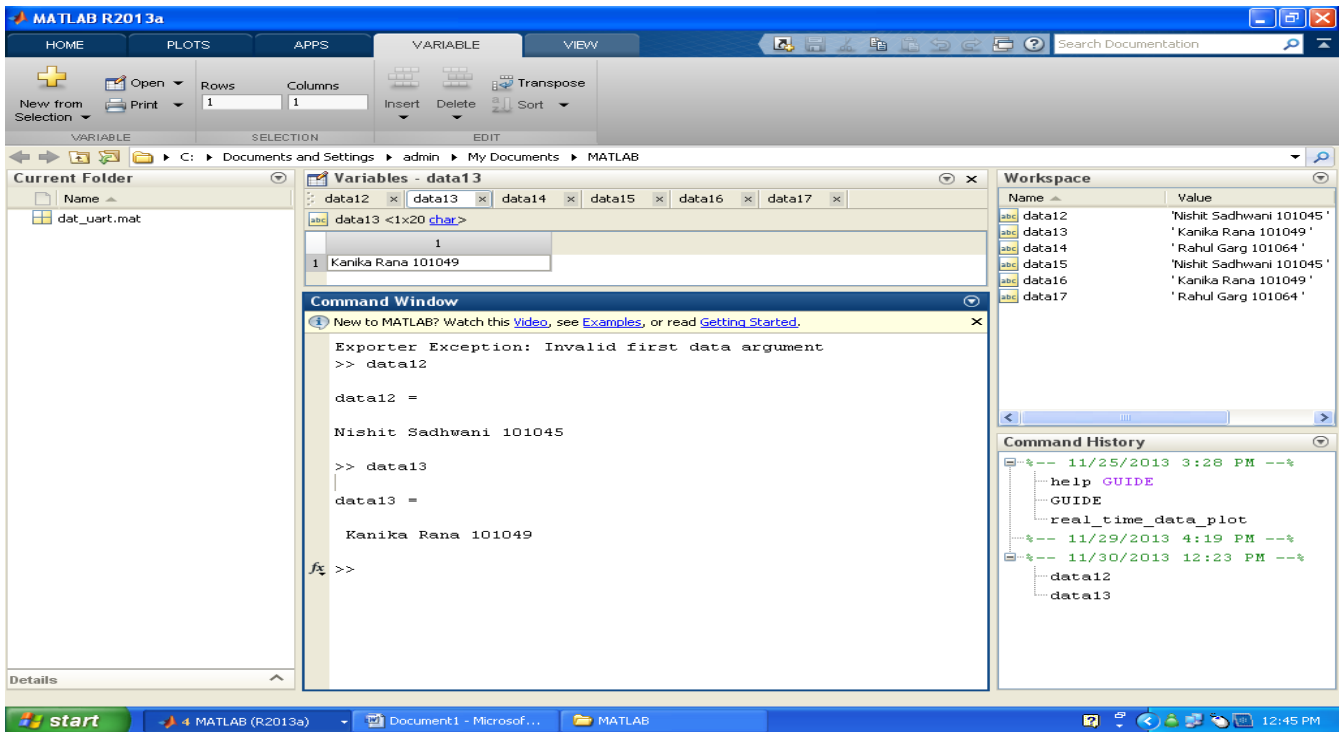
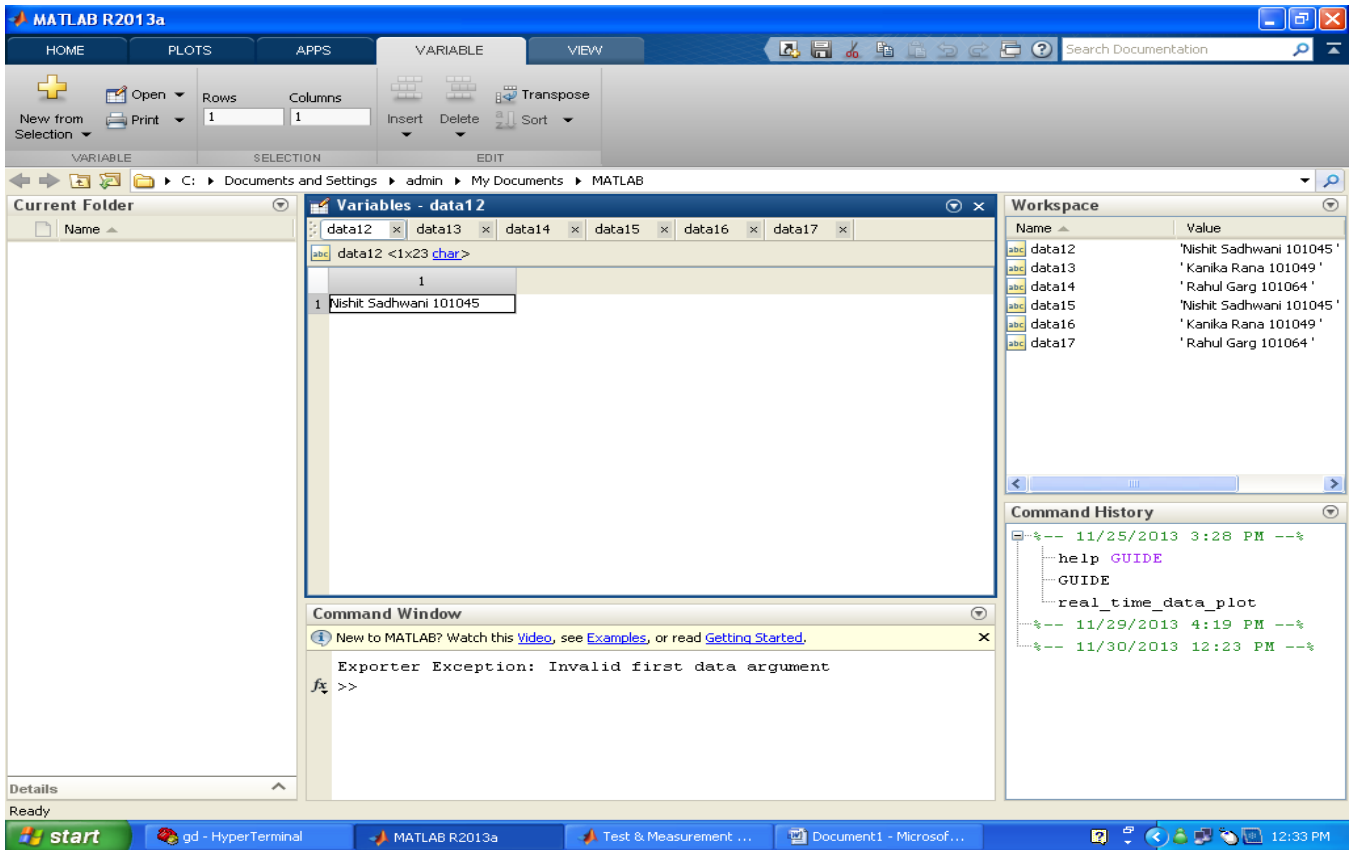


Fig 4.10 (a) and (b)

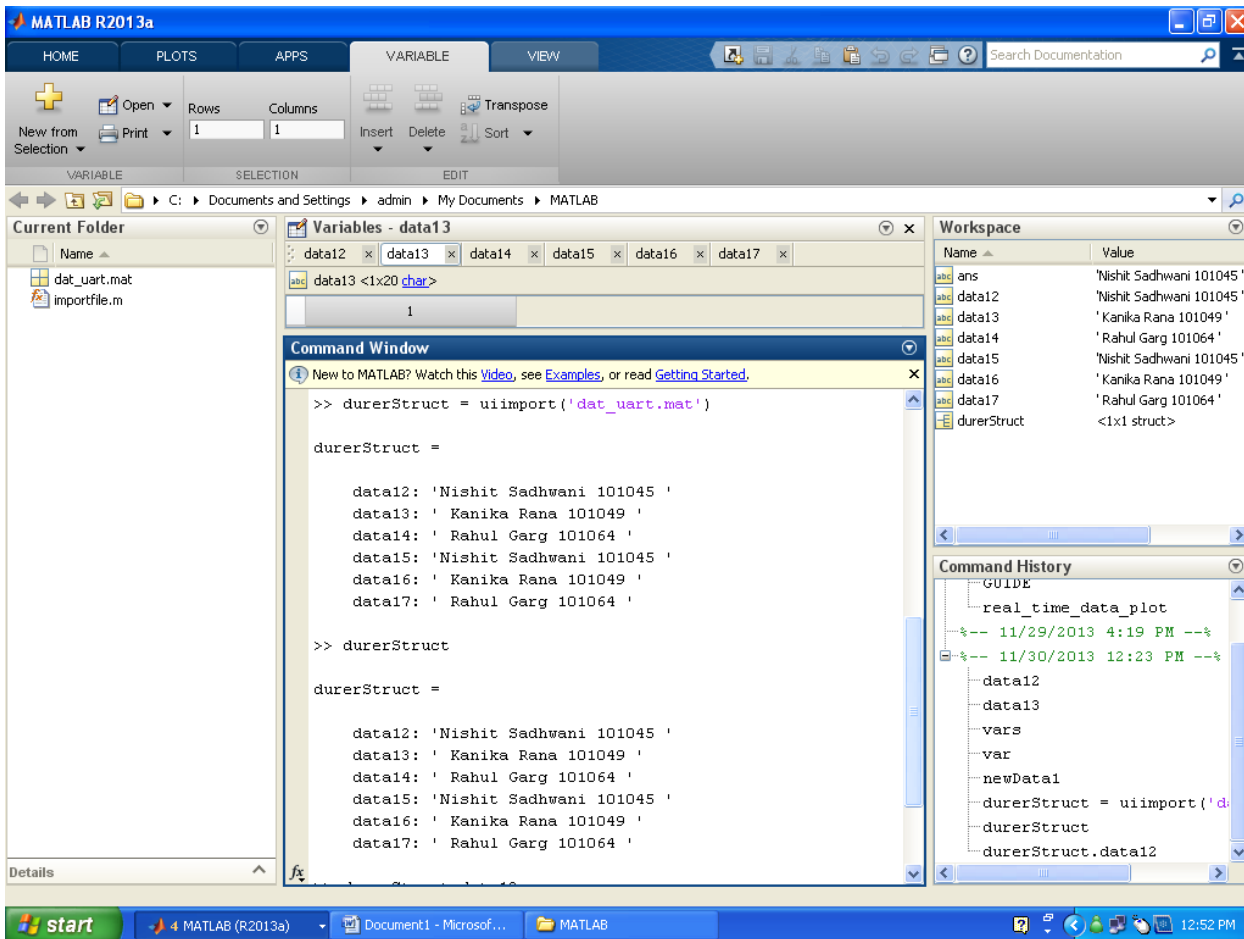


Fig 4.9 (c)
Structure of the imported data using the “uiimport” command from .mat file

4.4.3 Serial interfacing using MATLAB alternate way

There is an alternate way of connecting Arduino with MATLAB. For this we need to download Arduino support package from mathworks.in. In this file there are basic codes for Arduino board which we are going to upload on our microprocessor. Now add that folder in MATLAB path.

Now we can directly access our microprocessor from MATLAB using specific com port.

CHAPTER-5: Conclusions And Problems Faced

5.1 Conclusion

Finally I conclude that my project was not a big success because in the finally days of our working our codes were burnt. But rest was completed. Working on this project helped me to gain knowledge on MATLAB, Arduino IDE and many prospects of our microcontroller. The project was divided into two parts, one was software and the other was Hardware. The Software Part we were able to complete but we couldn't complete the Hardware Part because our components were burnt. Our project give us deep understanding of MATLAB.

5.2 Problems Faced

It is almost impossible that take up a 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 can we 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 software, 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. Another challenge we faced is to calibrate different sensors properly so that our glove should work as required especially the part where we have to stop the glove from moving we were unable to implement it.
4. Getting equation of different functions is quite tedious and take lots of time.
5. Another problem that I faced was my microcontroller got burnt damaging the glove and some sensors.

5.3: Future Prospects of Haptic Glove

Haptic Technology is a new age technology which is developing at a very fast pace. This technology provide us with lifelike interactions with virtual objects and environment. As we have researched for this project we are pretty much sure that this technology has a bright future. According to us Haptic Glove Technology can be used for various purposes live Remote surgery, controlling Robotic hand, Virtual Reality Gaming Experience, Teleoperators, interventional radiology, Holograph interaction. It can also be used to control hands of robots like mass rover making us feel that we are actually there. Research is being done in many fields using this technology so that we can understand better how things actually work by adding another sense to virtual reality i.e. a sense of touch.

Appendices

Appendix A1: MATLAB Codes

Various Functions

UnitCylinder

```
function cylinder=UnitCylinder(res)
%unit sphere in a format consistent with hierarchical
%modeler
%the input parameter is related to the sphere resolution.
%Range 1-10. Higher number is better approximation
%1=> 4-sided tube
%1.5=> 8-sided tube
%2=> 48 faces
%3=> 80 faces
%5=>136 faces
%10=>272 faces

%range check
if (res>10)
    res=10;
elseif (res<1)
    res=1;
end

res=1/res;
[x,y,z]=meshgrid(-1-res:res:1+res, ...
    -1-res:res:1+res, -1:1:1);
w=sqrt(x.^2+y.^2);
cylinder=isosurface(x,y,z,w,1);
```

UnitCube

```
function cube=UnitCube
%unit cube in a format consistent with hierarchical
```

```
%modeler
```

```
%Define a cube
```

```
cube.vertices=[ 0 0 0; 1 0 0; 1 1 0; 0 1 0; ...  
    0 0 1; 1 0 1; 1 1 1; 0 1 1; ] ;  
cube.faces=[ 1 2 6 5; 2 3 7 6; 3 4 8 7; 4 1 5 8; ...  
    1 2 3 4; 5 6 7 8; ] ;
```

```
cube.vertices = cube.vertices * 2 - 1;
```

UnitCylinder

```
function cylinder=UnitCylinder(res)
```

```
%unit sphere in a format consistent with hierarchical
```

```
%modeler
```

```
%The input parameter is related to the sphere resolution.
```

```
%Range 1-10. Higher number is better approximation
```

```
%1=> 4-sided tube
```

```
%1.5=> 8-sided tube
```

```
%2=> 48 faces
```

```
%3=> 80 faces
```

```
%5=>136 faces
```

```
%10=>272 faces
```

```
%range check
```

```
if (res>10)
```

```
    res=10;
```

```
elseif (res<1)
```

```
    res=1;
```

```
end
```

```
res=1/res;
```

```
[x,y,z]=meshgrid(-1-res:res:1+res, ...  
    -1-res:res:1+res, -1:1:1);
```



```
w=sqrt(x.^2+y.^2);  
cylinder=isosurface(x,y,z,w,1);
```

UnitSphere

```
function sphere=UnitSphere(res)  
%unit sphere in a format consistent with hieracrhical  
%modeler  
%The input paramenter is related to the sphere resolution.  
%Range 1-10. Higher number is better approximation  
%1=>octahedron  
%1.5=> 44 faces  
%2=> 100 faces  
%2.5 => 188 faces  
%3=> 296 faces  
%5=> 900 faces  
%10=>3600 faces  
  
%range check  
if (res>10)  
    res=10;  
elseif (res<1)  
    res=1;  
end  
  
res=1/res;  
[x,y,z]=meshgrid(-1-res:res:1+res, ...  
    -1-res:res:1+res, -1-res:res:1+res);  
w=sqrt(x.^2+y.^2+z.^2);  
sphere=isosurface(x,y,z,w,1);
```

UnitSurface

```

function surface=UnitSurface(res)
%unit flat surface in a format consistent with hieracrhical
%modeler
%The input paramenter is related to the sphere resolution.
%Range 1-10. Higher number is better approximation
%1=> 8 triangular faces
%2=> 32 faces
%5=>200 faces
%10=>800 faces
%20=>3200 faces
%50=>20000 faces

%range check
if (res>100)
    res=100;
elseif (res<1)
    res=1;
end

res=1/res;
[x,y,z]=meshgrid(-1:res:1, ...
    -1:res:1, -1:1:1);
w=z;
surface=isosurface(x,y,z,w,0);

```

rotateZ

```

function objOut = rotateZ(objIn,a)
%hierarchical rotate function for structs and cell arrays
a=a/57.29; %degrees to radians
if (iscell(objIn)) %a list of structs

```

```

for i=1:length(objIn)
    objOut{i}=objIn{i};
    V=objOut{i}.vertices;

    V=[cos(a)*V(:,1)-sin(a)*V(:,2), ...
        sin(a)*V(:,1)+cos(a)*V(:,2), ...
        V(:,3)];

    objOut{i}.vertices=V;
end
elseif (isstruct(objIn)) % must be a single struct
    V=objIn.vertices;

    V=[cos(a)*V(:,1)-sin(a)*V(:,2), ...
        sin(a)*V(:,1)+cos(a)*V(:,2), ...
        V(:,3)];

    objOut=objIn;
    objOut.vertices=V;
else
    error('input must be s struct or cell array')
end

```

rotateY

```

function objOut = rotateY(objIn,a)
%hierarchical rotate function for structs and cell arrays
a=a/57.29; %degrees to radians
if (iscell(objIn)) %a list of structs
    for i=1:length(objIn)
        objOut{i}=objIn{i};
        V=objOut{i}.vertices;

        V=[cos(a)*V(:,1)+sin(a)*V(:,3), ...
            V(:,2), ...

```

```

        -sin(a)*V(:,1)+cos(a)*V(:,3)];

    objOut{i}.vertices=V;
end
elseif (isstruct(objIn)) % must be a single struct
    V=objIn.vertices;

    V=[cos(a)*V(:,1)+sin(a)*V(:,3), ...
        V(:,2), ...
        -sin(a)*V(:,1)+cos(a)*V(:,3)];

    objOut=objIn;
    objOut.vertices=V;
else
    error('input must be s struct or cell array')
end

```

combine

```

function objOut = combine(varargin)

% Takes a list of objects (structs and cell arrays) and
% returns a cell array

num=length(varargin);

if (num==0)
    error('must have at least one input object');
end

objOut={ };

for i=1:num

```

```

if (iscell(varargin{i})) %a list of structs
    objOut=[objOut, varargin{i}];
elseif (isstruct(varargin{i})) %must be a single struct
    objOut=[objOut, {varargin{i}}];
else
    error('input must be s struct or cell array')
end %if (iscell(varargin{i}))
end

```

matcode

```

clc
clear all

```

```

%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 perspective 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('COM6',...
    'Baudrate',9600,...
    'Stopbits',1,...
    'Parity','none',...
    'FlowControl','none');
fopen(mcu)
try
    while (true)
        disp('Getting serial input...')
        tline = fgetl(mcu);
        result = sscanf(tline, '%d');

```

```

if numel(result)==9
    objectSelected = result(1);
    dIn = result(2);
    dOut = result(3);
    isSolid = result(4);
    temperature = result(5);
    isFull = result(6);
    objectGripped = result(7);
    objectLifted = result(8);
    height = result(9);

    % 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(temperature >= Tthresh)
    Tgradient = [1, 1-(temperature+30)/100, 0]; % color of the
object scaled to temperature
else
    Tgradient = [0, (temperature+20)/100, 1];
end
objectSurface.facecolor = Tgradient;

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 numel(result) == 6;
end % while(true)
```

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

References

1. Haptics:

http://en.wikipedia.org/wiki/Haptic_technology

<http://www.immersion.com/haptics-technology/what-is-haptics/>

<http://electronics.howstuffworks.com/everyday-tech/haptic-technology.htm>

<https://software.intel.com/en-us/blogs/2013/05/08/making-touch-more-realistic-advances-in-haptic-technology>

2. 3-D objects and modeling:

<http://www.nbb.cornell.edu/neurobio/land/PROJECTS/Hierarchy/>

3. ATmega 16:

<http://www.atmel.in/Images/doc8154.pdf>

4. 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/>

5. RS 232:

<http://www.engineersgarage.com/articles/what-is-rs232>

6. **USART:**

<http://www.engineersgarage.com/embedded/avr-microcontroller-projects/serial-communication-atmega16-usart>

7. **Computer Graphics:**[2] Shih-Liang (Sid) Wang, ‘Introducing Fundamentals of Computer Graphics Using MATLAB’

8. **Arduino:** www.arduino.cc, www.mathworks.in.