A Project Report on

Smart Bike Helmet

Submitted by

Drishti Rao (ROLL NO. 23)

Rohan Suresh (ROLL NO. 29)

Stefen Rosario (ROLL NO. 30)

Swapnil Samant (ROLL NO. 33)

in fulfillment of

MINI-PROJECT I

in

Electronics & Telecommunication Engineering

Under the Guidance of

Mrs. Monica Cheema

Department of Electronics and Telecommunication Engineering

St. Francis Institute of Technology, Mumbai
University of Mumbai
(2016-2017)

ABSTRACT

Road safety has become an alarming issue in world, especially in India where it is not taken as a potential issue. The disrespect for safety rules is one of the major reasons. People are seen driving while talking on phone. They even ride a scooter or a bike while talking on a phone, which is comparatively more dangerous.

We propose a solution to ensure that the bikers hand will always be on the controllers even if he receives an urgent call. Our topic is the Smart Bike Helmet. From this the biker won't be distracted by an incoming call and he won't have to remove his phone from the pockets to answer it nor would he have to adjust his phone in the helmet and click on the acceptance button either. It's totally hands free.

The basic idea is that when there is an incoming call in your phone, the biker will get to know who is calling him. He just has to nod or shake his head to answer the phone or reject it, respectively. The call is transferred to an ear peace without him actually "accepting" the call via any button.

CERTIFICATE

This is to certify that the project entitled **Smart Bike Helmet** is a bonafide work of **Drishti Rao (ROLL NO. 23), Rohan Suresh (ROLL NO. 29), Stefen Rosario (ROLL NO. 30), Swapnil Samant (ROLL NO. 33)** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **Bachelor of Engineering** in **Electronics and Telecommunication Engineering**.

(Mrs. Monica Cheema)
Internal Guide

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

The satisfaction that accompanies the completion of our project would be incomplete without the mention of the people who made it possible, without whose constant guidance and encouragement our efforts would go in vain. We consider ourselves privileged to express gratitude and respect towards all those who guided us throughout the completion of this project.

We convey our immense gratitude to our project guides Mrs. Monica Cheema, Assistant Professor – Electronics and Telecommunication Engineering Department and Mr. Inderkumar Kochar, Assistant Professor - Electronics and Telecommunication Engineering Department, for providing encouragement and guidance which was of great help to complete this project successfully. We would like to thank Mr. Siddhesh Gotad, Research Assistant – Computer Engineering Department and Mr. Vikrant Fernandes, Research Assistant – Computer Engineering Department, Mr. Jerin Thankappan, Assistant Professor – Computer Engineering Department and Mr. Kiran Patil, Lab Assistant-Electronics and Telecommunication Engineering Department for their continuous guidance.

(Mr. Stefen Rosario) (Mr. Rohan Suresh)

(Mr. Swapnil Samant) (Ms. Drishti Rao)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Smart Bike Helmet is a solution for all the bike riders. It allows them to answer their phone calls without taking their hands off the clutches. They would not even have to take their eyes off the road and look down which is the general case, if they were to keep a phone case on attached to the bike. The display will be in the same eyesight level. This will very helpful for all the race drivers and test drivers. With an effective HUD (heads up display), you can seamlessly glance from road to display without blurring your vision. This removes the danger, for example, of taking your attention from the road as you look to a phone mounted on your bike for GPS.

1.1 Motivation

We hear a lot of news about accidents these days where the driver is either drunk, but in most cases is, driving while talking on the phone because of which his hands are not on the proper controls.

This led us to an idea of making a helmet where a person will not have to physically answer a phone by swiping the answering option, he would just have to nod or shake his head.

1.2 Scope of the Project

Our idea was to develop a device which will eliminate the need to physically answer a phone while someone is driving a two-wheeler vehicle.

1.3 Methodology

We used an android based app to link with the inbuilt activity starter of our phones. The call gets transferred from our mobile to our ear piece. We used a gyro sensor which indicates the position of our head. I.e. If we have nodded or shook our head. The system also uses a Bluetooth linking part where the phone Bluetooth pairs with the Bluetooth with Arduino and sends 1/0 indicating whether there is an incoming call or not.

1.4 Organization of Project Report

This project report is organized as follows:

Chapter 2 presents the literature survey on the existing techniques

Chapter 3 provides a brief explanation of the various components which are used and the android based app

Chapter 4 is dedicated to the experimental results.

Chapter 5 presents the conclusions and future scope for this project.

Chapter 2

Literature Review

2.1 Literature Review on Existing Techniques (Skully)

The Skully offers a complete 180° Blind spot Camera, which eliminates the need for riders to turn their heads to check for other motorists. Not only is this safer for the rider, but it is safer for everyone else on the road too.

It is also important that a rider be able to glance at the information on the HUD (heads-up display) without losing sight of the road. SKULLY accomplishes this trick with its patented Infinite Focus™ technology. The AR-1's HUD appears to float six meters away, allowing riders to glance from road to display without ever losing focus.

With an effective HUD, you can seamlessly glance from road to display without blurring your vision. This removes the danger, for example, of taking your attention from the road as you look to a phone mounted on your bike for GPS. The goal of any HUD should be to provide the user with superior information in the most digestible way possible. The AR-1's camera is 180°, removing blind spots for near 360° awareness. Side mirrors on motorcycles are notorious for their inadequacy and many riders forgo using them altogether, looking over their shoulders and taking their focus away from the road. With the rear-view camera, you have full situational awareness with a quick saccade of the eye.

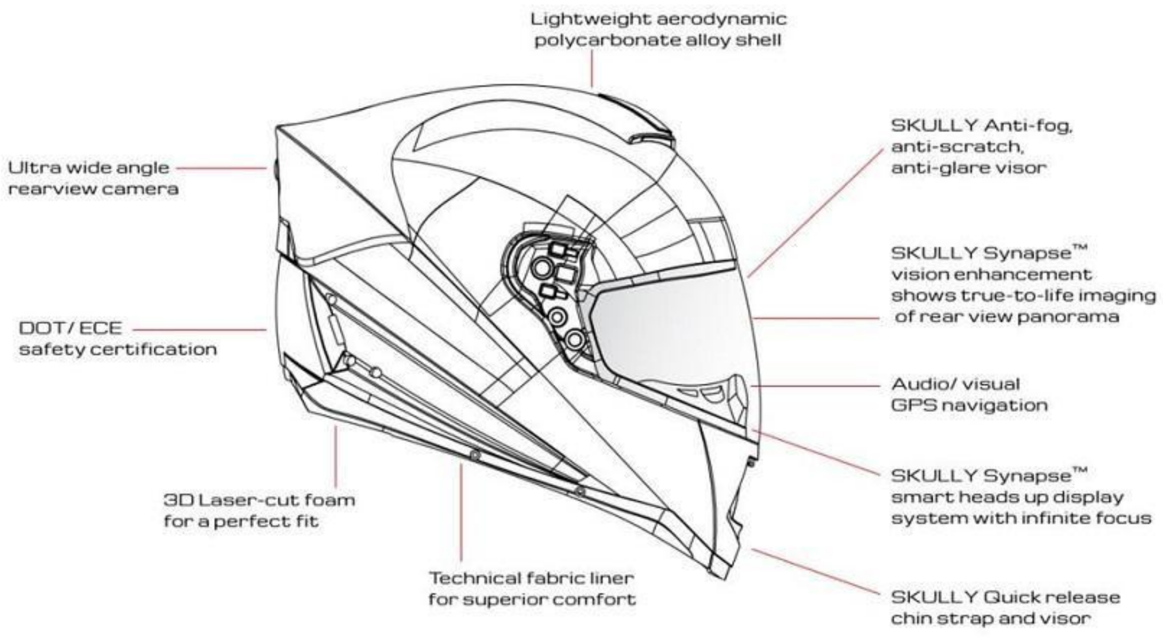

Lightweight aerodynamic polycarbonate alloy shell
Ultra wide angle rearview camera
SKULLY Anti-fog, anti-scratch, anti-glare visor
SKULLY Synapse™ vision enhancement shows true-to-life imaging of rear view panorama
DOT/ ECE safety certification
Audio/ visual GPS navigation
SKULLY Synapse™ smart heads up display system with infinite focus
3D Laser-cut foam for a perfect fit
Technical fabric liner for superior comfort
SKULLY Quick release chin strap and visor

Fig. 2.1: Features of Skully Helmet

Chapter 3

Software and Hardware Support

3.1 Software

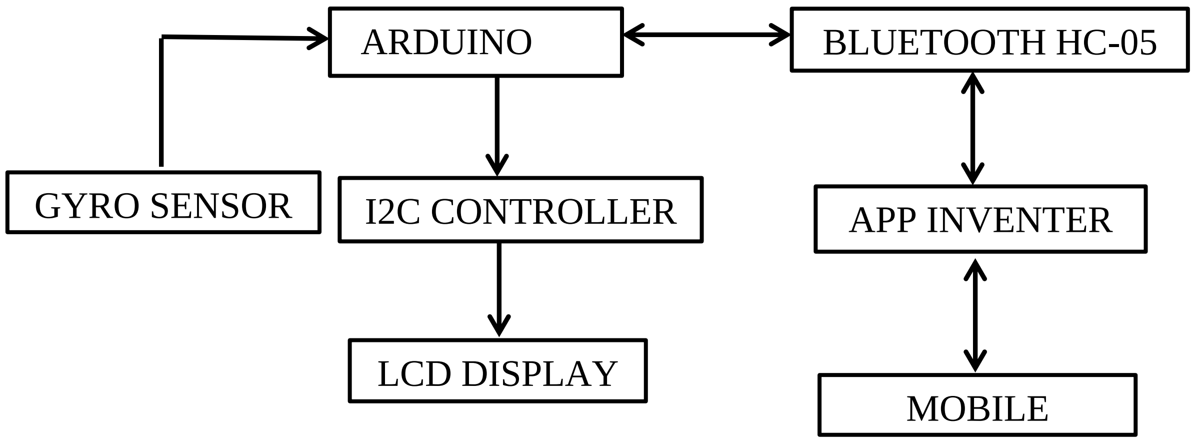


Fig. 3.1: Block Diagram of the circuit

In order to create an interface between hardware (Arduino) and mobile we used an App made using MIT App Inventor.

Process 1

The hardware part: Here the Arduino is interfaced with blue tooth module HC05 will be paired with our Mobile Bluetooth.

When someone calls on one's phone the application on mobile will send 1 to the Bluetooth of the Arduino by our mobile Bluetooth. This will be achieved by using background process concept.

During the process of receiving call the Arduino will accept 1 and initialize the process 2 based on it.

Process 2

There is an LCD interface connected to our Arduino which displays the text according to which condition which is met.

Arduino is so programed that as soon as Arduino receives 1 from Bluetooth, the person riding bike will be able to see who is calling, which will be displayed on the LCD screen. The screen is placed in a such a way that it comes in our vision, but does not hinder it.

A Gyro Sensor is connected to chin side of our helmet. Reference values for all the conditions are set in code. A nod from the rider indicates that he has to answer the call. This nod is reflected by a change in gyro reading, i.e. there will be certain changes in xyz coordinates reading of sensor and the Bluetooth will send 1 to the paired mobile device. Mobile app will receive 1 and then accept the call. Parallel/simultaneously LCD will show 'call accepted' according to Arduino code

If the rider nods in horizontal direction then the rider wants to reject the call. Then according to change in position in Gyro the code will run. In this case too, there will be certain changes in xyz coordinates reading of sensor and the Bluetooth will send 0 to the paired mobile device. Mobile app will receive 0 and then reject the call. Parallel/simultaneously LCD will show 'call rejected' according to Arduino code.

During call if person wants to reject then he can nod horizontally. Parallel/simultaneously LCD will show 'call cancelled' according to Arduino code.

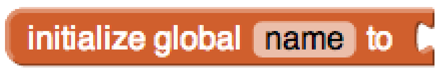
In screen 1, there is button to start the app. On clicking this button another screen is opened. Screen 2 is the main screen. The incoming call notification is kept off until a call is received. The options for accepting call, rejecting call and end call is also kept off until a call is received.

There is a note on Screen2 to turn on the Bluetooth of the phone, so that it retrieves the list of paired device which is Arduino Bluetooth. When a call button is pressed, the LCD will display the text Incoming Call. In mobile the Incoming Call notification and Call End button becomes visible. According to the response of the gyro the Arduino will either send 1 or 0, depending on which the notification changes.

Case 1: If returned value is 1 then it will make accepted notification visible. (Here you are allowed to use call end button). Here if you click Call End button, app goes to its original state.

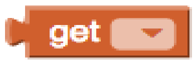
Case 2: If returned value is 0 then it will make rejected notification visible. (Here you are not allowed to use call end button). And hence the app goes to its original state.

Some of the functions used for our app are:

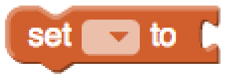


This block is used to create global variables. It takes in any type of value as an argument. Global variables are used in all procedures or events so this block will stand alone.

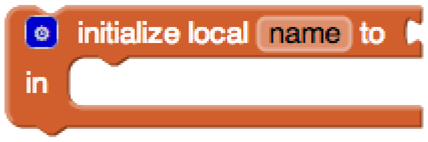
Global variables can be changed while an app is running and can be referred to and changed from any part of the app even within procedures and event handlers.



This block provides a way to get any variables which may have created.



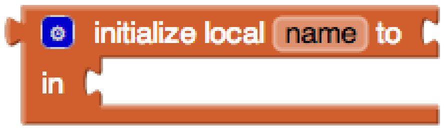
This block follows the same rule as get. Only variables in scope will be available in the dropdown. Once a variable *v* is selected, the user can attach a new block and give *v* a new value.



This block is a mutator that allows you to create new variables that are only used in the procedure you run in the DO part of the block. This way all variables in this procedure will all start with the same value each time the procedure is run.

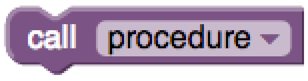
NOTE: This block differs from the block described below because it is a DO block. You can attach *statements* to it. Statements *do* things. That is why this block has space inside for statement blocks to be attached.

initialize Local name to - in (return)

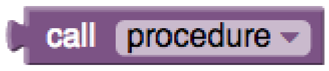


This block is a mutator that allows you to create new variables that are only used in the procedure you run in the RETURN part of the block. This way all variables in this procedure will all start with the same value each time the procedure is run. NOTE: This block differs from the block described above because it is a RETURN block. The *expressions* can be attached to it. Expressions *return* a value. That is why this block has a socket for plugging in expressions.

The variables can be renamed in this block at any time and any corresponding blocks elsewhere in your program that refer to the old name will be updated



When you create a procedure, App Inventor automatically generates a call block and places it in the My Definitions drawer. You use the call block to invoke the procedure.



After creating this procedure, a call block that needs to be plugged in will be created. This is because the result from executing this procedure will be returned in that call block and the value will be passed on to whatever block is connected to the plug.

3.2 Hardware Support

3.2.1 Arduino Uno

Arduino/Genuino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Arduino/Genuino Uno board can be powered via the USB connection or with an external power supply. The power source is selected automatically. The board can operate on an external supply from 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may become unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts. It has 2 KB of SRAM and 1 KB of EEPROM. Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive 20 mA as recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50k ohm. A maximum of 40mA is the value that must not be exceeded on any I/O pin to avoid permanent damage to the microcontroller. Arduino/Genuino Uno has a number of facilities for communicating with a computer, another Arduino/Genuino board, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The 16U2 firmware uses the standard USB COM drivers, and no external driver is needed.

TABLE 3.1 TECHNICAL SPECS OF ARDUINO UNO

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 Ma
DC Current for 3.3V Pin	50 Ma
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25

3.2.2 LCD

LCD (Liquid Crystal Display) screen is an electronic display module and find a wide range of applications. A 16x2 LCD display is very basic module and is very commonly used in various devices and circuits. These modules are preferred over seven segments and other multi segment LEDs. The reasons being: LCDs are economical; easily programmable; have no limitation of displaying special & even custom characters (unlike in seven segments), animations and so on.

A 16x2 LCD means it can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix. This LCD has two registers, namely, Command and Data.

The command register stores the command instructions given to the LCD. A command is an instruction given to LCD to do a predefined task like initializing it, clearing its screen, setting the cursor position, controlling display etc. The data register stores the data to be displayed on the LCD. The data is the ASCII value of the character to be displayed on the LCD.

FEATURES

- 5 x 8 dots with cursor
- Built-in controller (KS 0066 or Equivalent)
- + 5V power supply (Also available for + 3V)
- 1/16 duty cycle
- B/L to be driven by pin 1, pin 2 or pin 15, pin 16 or A.K (LED)
- N.V. optional for + 3V power supply

MECHANICAL DATA

ITEM	STANDARD VALUE	UNIT
Module Dimension	80.0 x 36.0	mm
Viewing Area	66.0 x 16.0	mm
Dot Size	0.56 x 0.66	mm
Character Size	2.96 x 5.56	mm

ABSOLUTE MAXIMUM RATING

ITEM	SYMBOL	STANDARD VALUE			UNIT
		MIN.	TYP.	MAX.	
Power Supply	VDD-VSS	- 0.3	–	7.0	V
Input Voltage	VI	- 0.3	–	VDD	V

NOTE: VSS = 0 Volt, VDD = 5.0 Volt

ELECTRICAL SPECIFICATIONS

ITEM	SYMBOL	CONDITION		STANDARD VALUE			UNIT
				MIN.	TYP.	MAX.	
Input Voltage	VDD	VDD = + 5V		4.7	5.0	5.3	V
		VDD = + 3V		2.7	3.0	5.3	V
Supply Current	IDD	VDD = 5V		–	1.2	3.0	mA
Recommended LC Driving Voltage for Normal Temp. Version Module	VDD - V0	- 20 °C		–	–	–	V
		0°C		4.2	4.8	5.1	
		25°C		3.8	4.2	4.6	
		50°C		3.6	4.0	4.4	
		70°C		–	–	–	
LED Forward Voltage	VF	25°C		–	4.2	4.6	V
LED Forward Current	IF	25°C	Array	–	130	260	mA
			Edge	–	20	40	
EL Power Supply Current	IEL	Vel = 110VAC:400Hz		–	–	5.0	mA

Table 2.2 Technical Specifications of LCD

3.2.3 Gyro

Analog Devices like MEMS accelerometers offer highly accurate motion detection when measuring acceleration, tilt, shock, and vibration in performance-driven applications. ADI's MEMS accelerometer portfolio leads the industry in power, temperature, noise, and bandwidth performance.

Typical Features

- Low power
- High resolution
- Activity/inactivity monitoring
- Free-fall detection
- Supply voltage range: 1.8V to 3.6V
- I/O voltage range: 1.6V to VS
- SPI (3- and 4-wire) and I2C digital interfaces
- Measurement ranges selectable via serial, SPI command
- Wide temperature range (−40°C to +125°C)
- High shock survival up to 10,000g

Typical Applications

- Handsets
- Medical instrumentation
- Gaming and pointing devices
- Industrial instrumentation
- Personal navigation devices
- Hard disk drive(HDD) protection
- Vehicle dynamic controls
- Electronic chassis controls
- Platform stabilization/leveling
- Alarms and motion detectors
- High accuracy, 2-axis tilt sensing
- Abuse event detection
- Vibration monitoring and control
- Vehicle collision sensing
- Shock detection

GENERAL DESCRIPTION: The ADXL335 is a small, thin, low power, complete 3-axis accelerometer with signal conditioned voltage outputs. The product measures acceleration with a minimum full-scale range of ±3 g. It can measure the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion, shock, or vibration. The user selects the bandwidth of the accelerometer using the CX, CY, and CZ capacitors at the XOUT, YOUT, and ZOUT pins. Bandwidths can be selected to suit the application, with a range of 0.5 Hz to 1600 Hz for the X and Y axes, and a range of 0.5 Hz to 550 Hz for the Z axis. The ADXL335 is available in a small, low profile, 4 mm × 4 mm × 1.45 mm, 16-lead, plastic lead frame chip scale package (LFCSP_LQ).

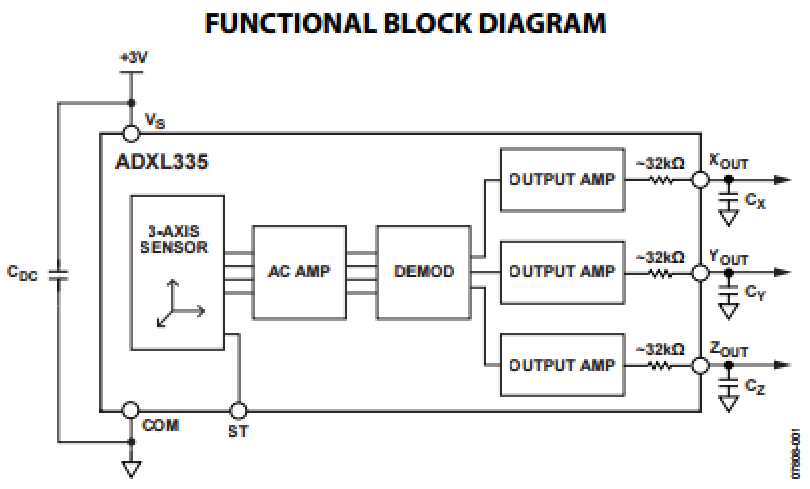


Fig. 3.2: Block Diagram of Gyro

It contains a polysilicon surface-micromachined sensor and signal conditioning circuitry to implement an open-loop acceleration measurement architecture. The output signals are analog voltages that are proportional to acceleration. The accelerometer can measure the static acceleration of gravity in tilt-sensing applications as well as dynamic acceleration resulting from motion, shock, or vibration. The sensor is a polysilicon surface-micromachined structure built on top of a silicon wafer. The ADXL335 uses a single structure for sensing the X, Y, and Z axes. As a result, the three axes' sense directions are highly orthogonal and have little cross-axis sensitivity.

3.2.4 Bluetooth module HC05

Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with

AFH(Adaptive Frequency Hopping Feature). It has the footprint as small as 12.7mmx27mm. Hope it will simplify your overall design/development cycle.

Specifications:

Hardware Features

- Typical -80dBm sensitivity
- Up to +4dBm RF transmit power
- Low Power 1.8V Operation ,1.8 to 3.6V I/O
- PIO control
- UART interface with programmable baud rate
- With integrated antenna
- With edge connector

Software Features

- Default Baud rate: 38400, Data bits:8, Stop bit:1, Parity: No parity, Data control: has.
- Supported baud rate: 9600, 19200, 38400, 57600, 115200, 230400, 460800.
- Given a rising pulse in PIO0, device will be disconnected.
- Status instruction port PIO1: low-disconnected, high-connected;
- Auto-connect to the last device on power as default.
- Permit pairing device to connect as default.
- Auto-pairing PINCODE:”0000” as default
- Auto-reconnect in 30 min when disconnected as a result of beyond the range of connection.

The module has two modes of operation, Command Mode where we can send AT commands to it and Data Mode where it transmits and receives data to another Bluetooth module.

3.2.5 I2C Connector

I²C (Inter-Integrated Circuit) is a multi-master, multi-slave, single-ended, serial computer bus. It is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.

I2C uses only two wires: SCL (serial clock) and SDA (serial data). Both need to be pulled up with a resistor to +Vdd. There are also I2C level shifters which can be used to connect to two

I2C buses with different voltages. The two wires can support up to 1008 slave devices. Also, unlike SPI, I2C can support a multi-master system, allowing more than one master to communicate with all devices on the bus (although the master devices can't talk to each other over the bus and must take turns using the bus lines).
Data rates fall between asynchronous serial and SPI; most I2C devices can communicate at 100 kHz or 400 kHz. There is some overhead with I2C; for every 8 bits of data to be sent, one extra bit of Meta data (the "ACK/NACK" bit, which we'll discuss later) must be transmitted.

The hardware required to implement I2C is more complex than SPI, but less than asynchronous serial. It can be easily implemented in software.

I2C is appropriate for peripherals where simplicity and low manufacturing cost are more important than speed. Common applications of the I2C bus are:

- Reading configuration data from SPD EEPROMs on SDRAM, DDR SDRAM, DDR2 SDRAM memory sticks (DIMM) and other stacked PC boards
- Changing contrast, hue, and color balance settings in monitors (Display Data Channel).
- Changing sound volume in intelligent speakers.
- Controlling OLED/LCD displays, like in a cellphone.
- Reading hardware monitors and diagnostic sensors, like a CPU thermistor or fan speed.
- Reading real-time clocks.
- Turning on and turning off the power supply of system components.

A particular strength of I2C is the capability of a microcontroller to control a network of device chips with just two general purpose I/O pins and software. Many other bus technologies used in similar applications, such as Serial Peripheral Interface Bus, require more pins and signals to connect devices.

I2C bus is used by many integrated circuits and is simple to implement. Any microcontroller can communicate with I2C devices even if it has no special I2C interface. I2C specifications are flexible - I2C bus can communicate with slow devices and can also use high speed modes to transfer large amounts of data.

Chapter 4

Implementation Results and Discussion

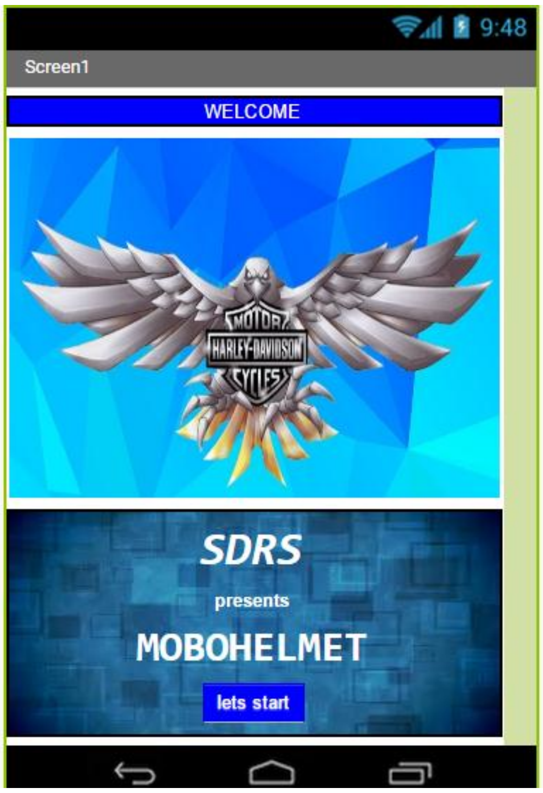


Fig. 4.1: Screen 1 of App

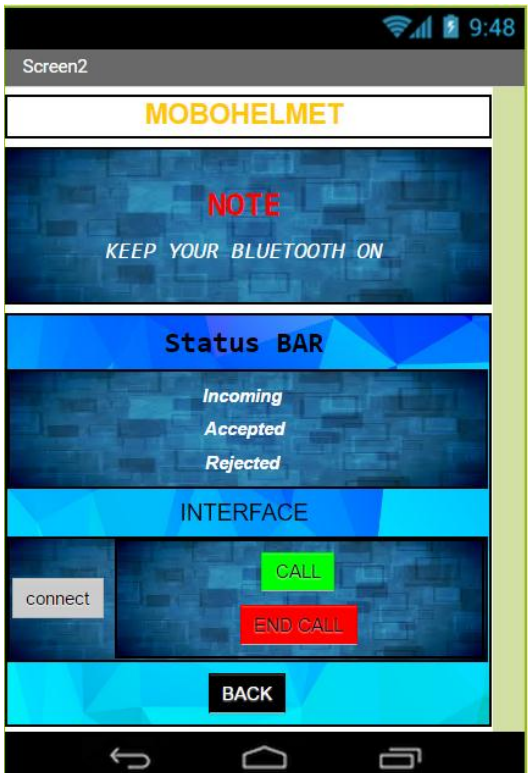


Fig. 4.2: Screen 2 of App

There are four possible cases for a rider. They are as follows:

Case 1: Normal state: It's when the person is riding the bike, without any interference.

Case 2: Incoming call: when a call is received on the biker's phone, the display screen will notify the rider, about the same, along with the identification of the caller.

Case 3: Accepting call: if the person riding the bike wants to accept the call and talk, he would have to nod. This will change the z axis coordinates of the gyro, and hence, the points will differ from the original ones, and hence the call gets accepted.

Case 4: Rejecting call: If the person riding the bike wants to reject the call, he would have to shake his head. This will change the x and y axes coordinates of the gyro, and hence, the points will differ from the original ones, and hence the call gets rejected.

Case 5: End call.

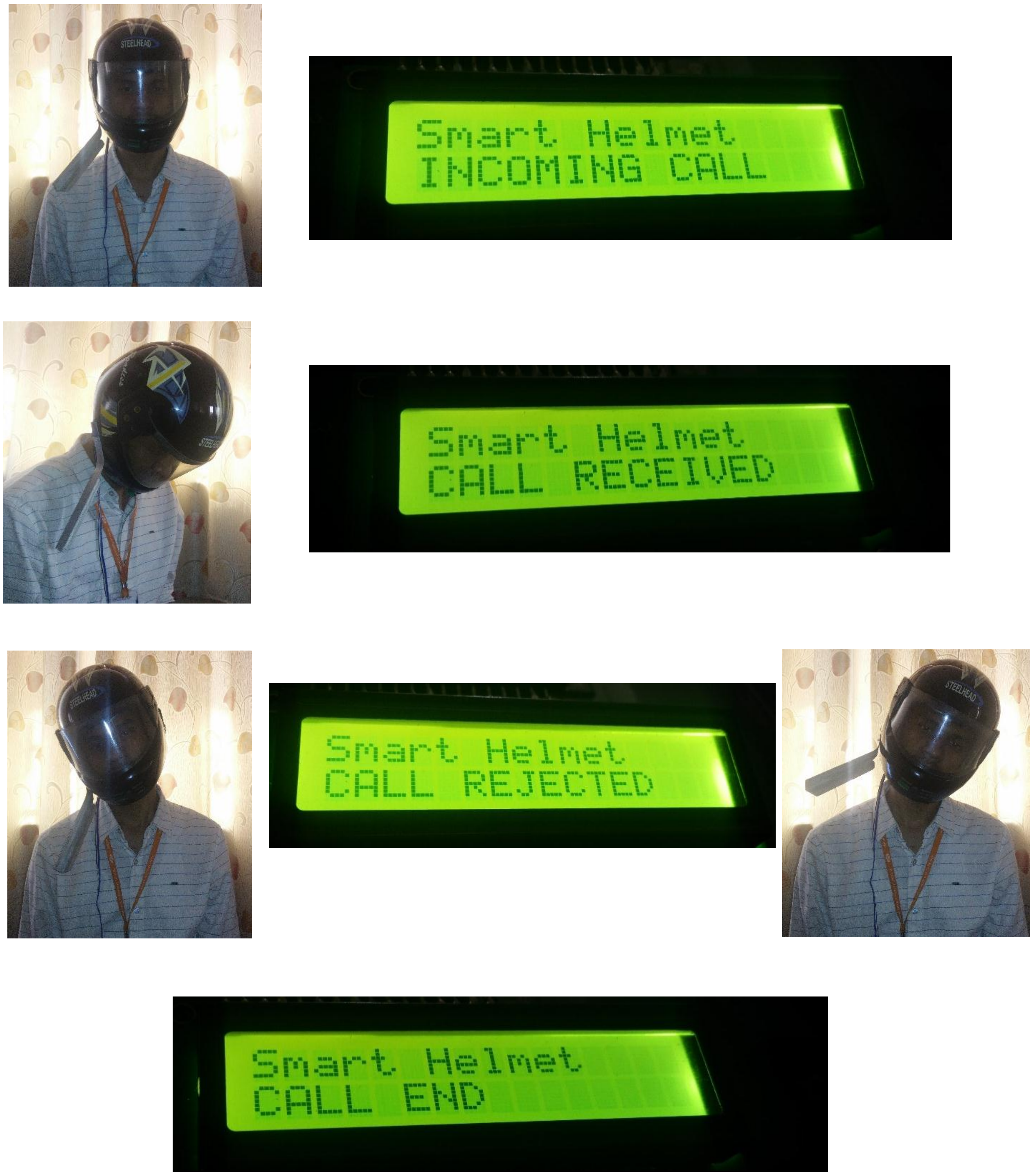
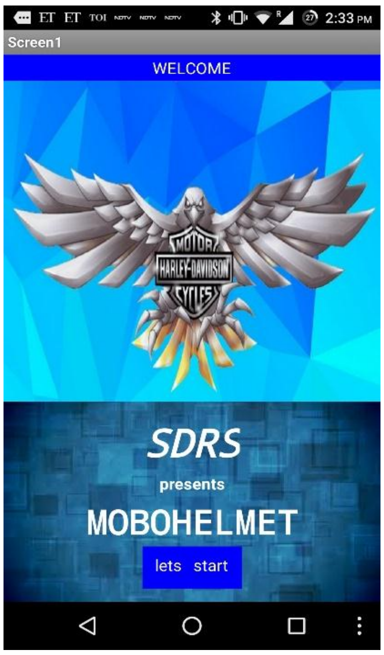
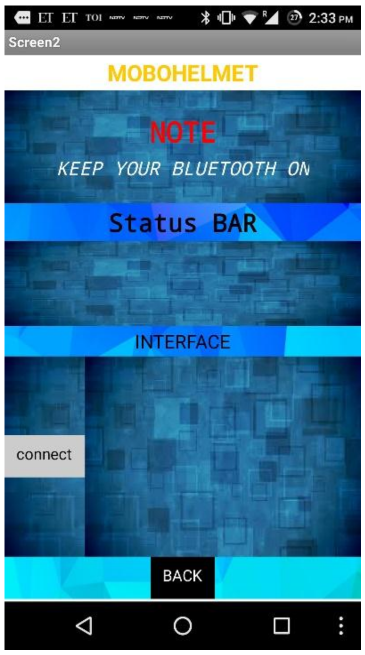


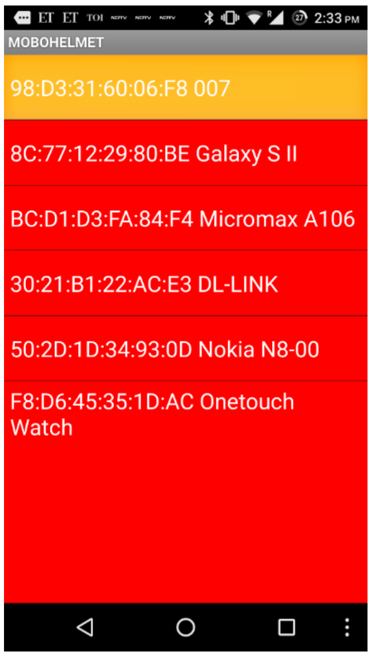
Fig. 4.3: Various Cases and its Results



Starting with Screen



After clicking 'let's start'



98:D3:31:60:06:F8 007

8C:77:12:29:80:BE Galaxy S II

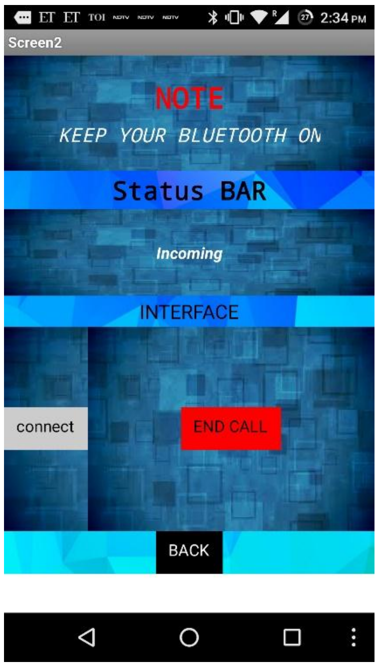
BC:D1:D3:FA:84:F4 Micromax A106

30:21:B1:22:AC:E3 DL-LINK

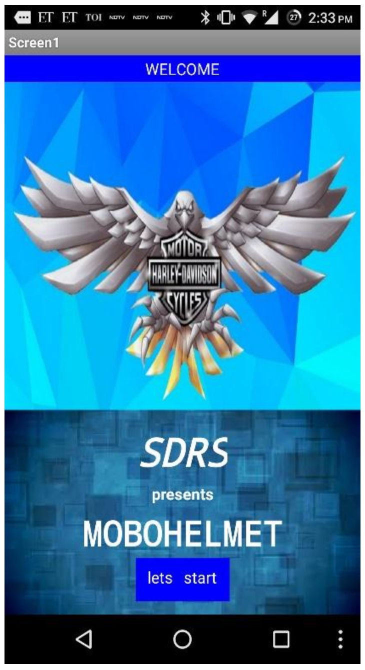
50:2D:1D:34:93:0D Nokia N8-00

F8:D6:45:35:1D:AC Onetouch Watch

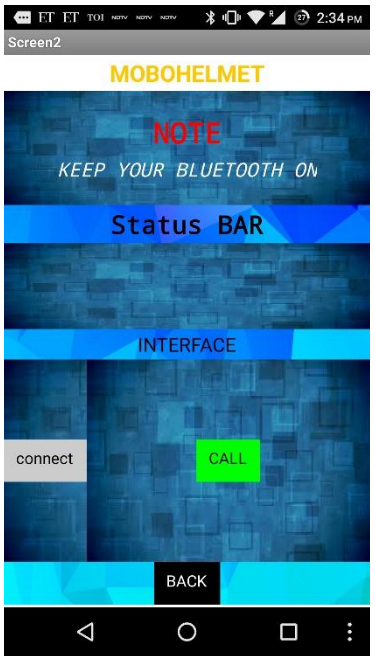
After clicking connect.



. After selecting MAC Address Arduino Bluetooth



When Bluetooth of mobile detects 'y'



After clicking 'END of CALL' or 'BACK'

Fig. 4.4: App Display for Various Cases

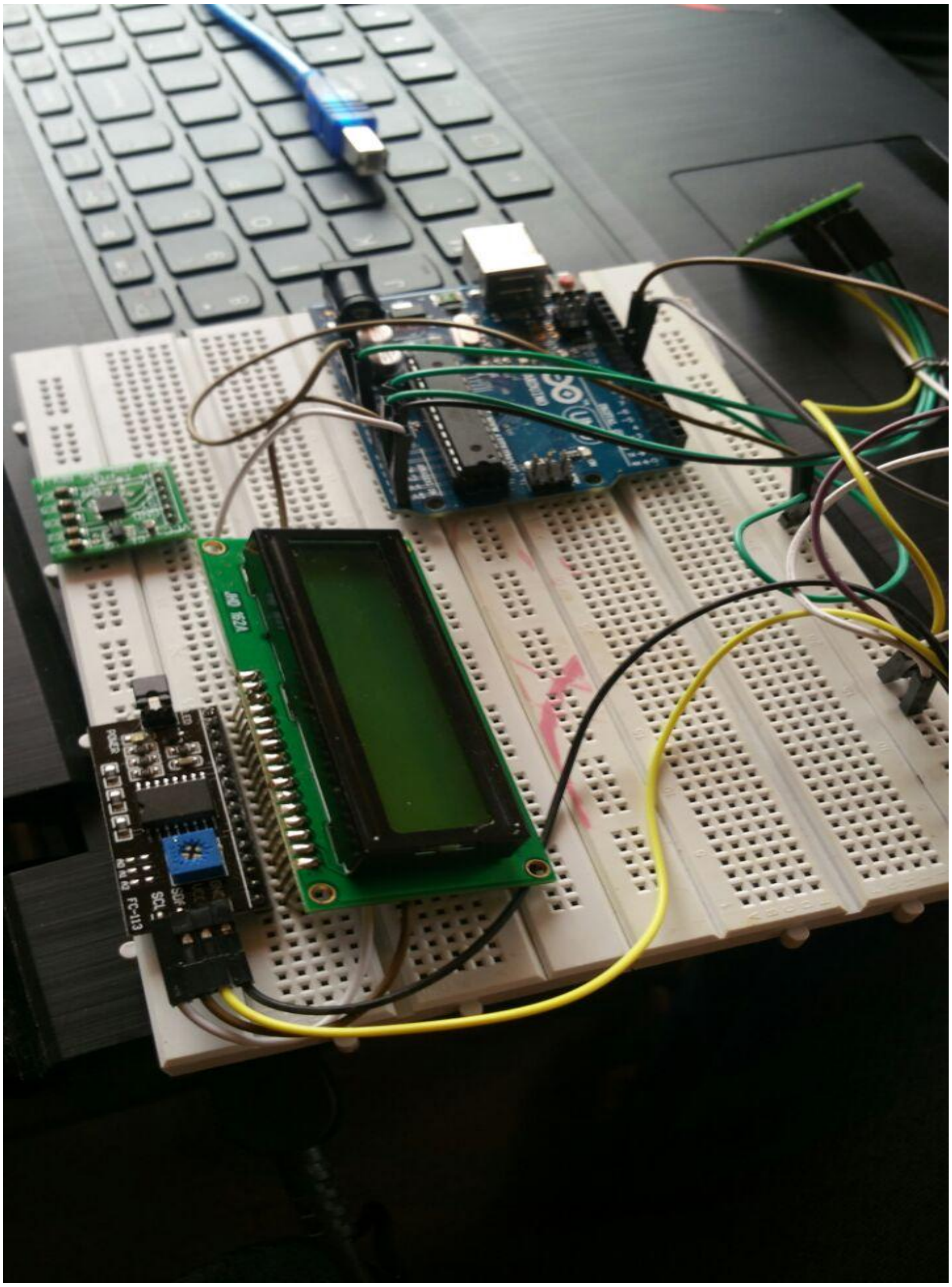


Fig. 4.5: Circuit Implemented on the breadboard

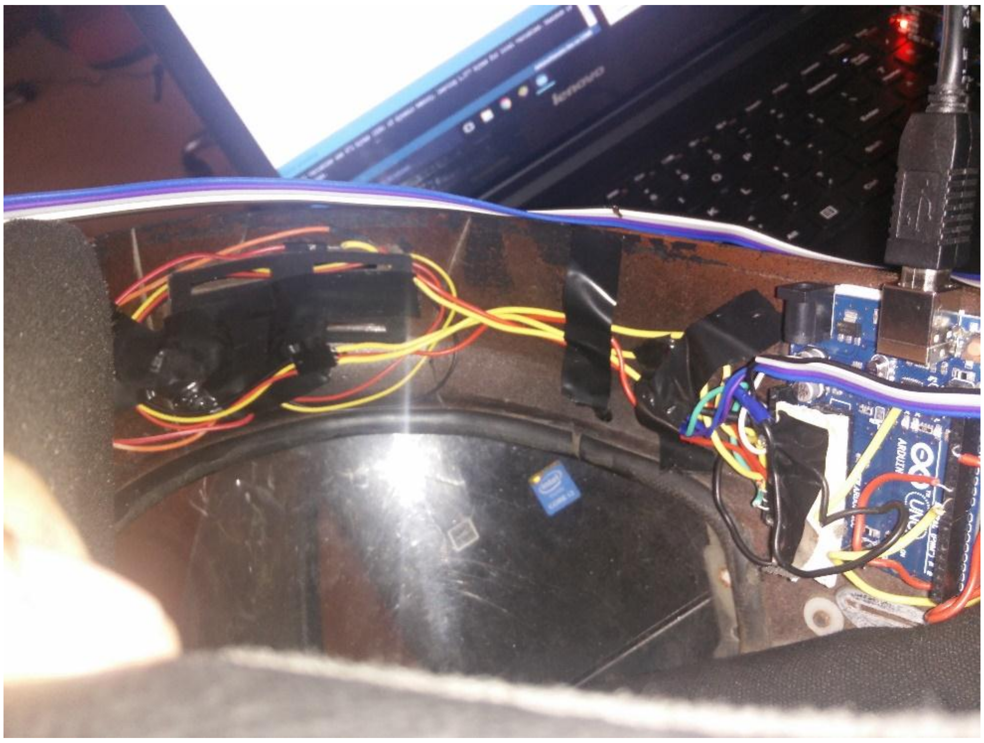


Fig. 4.6: Internal Circuitry for the Helmet

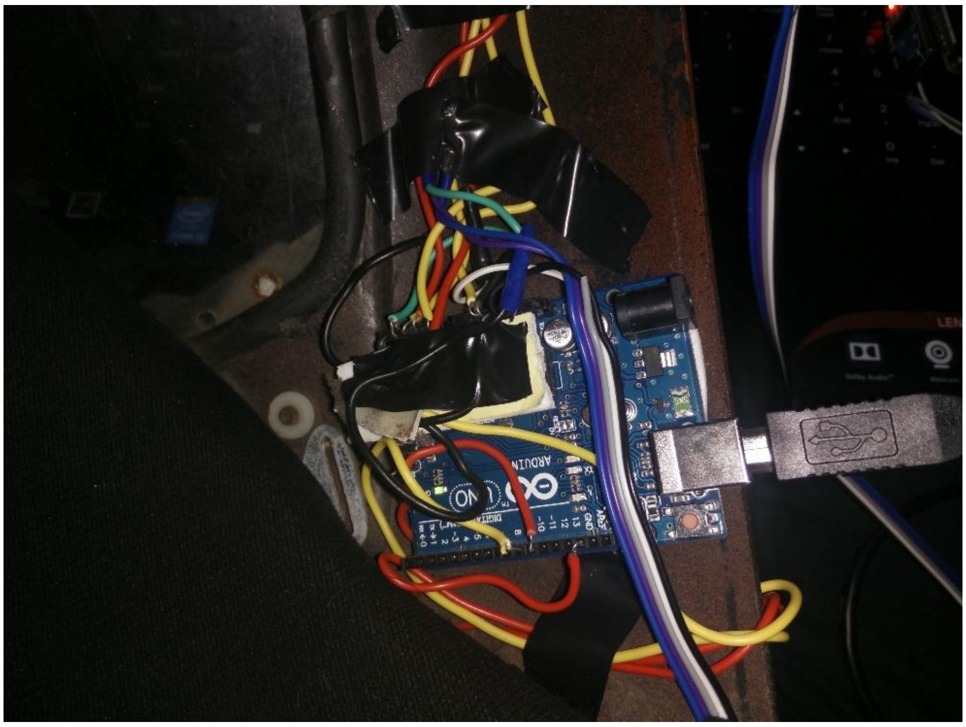


Fig. 4.7: Connections for the Arduino

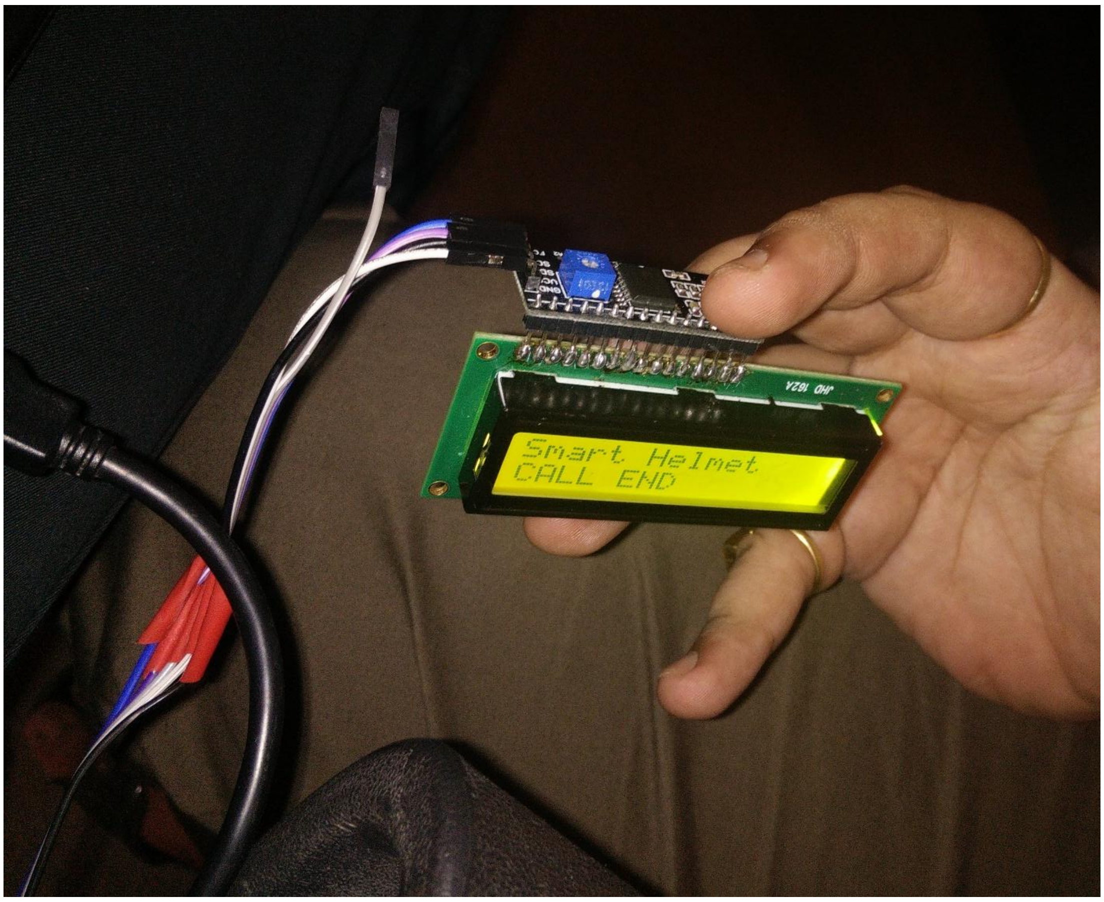


Fig. 4.8: LCD Display with I2C Controller

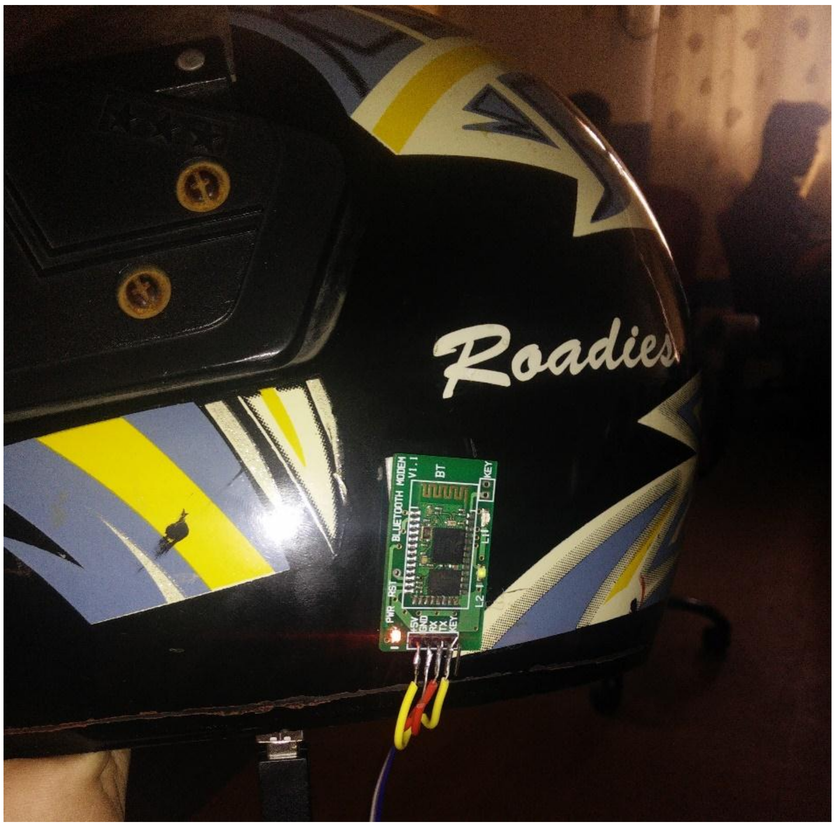


Fig. 4.9: Bluetooth device attached to the Helmet

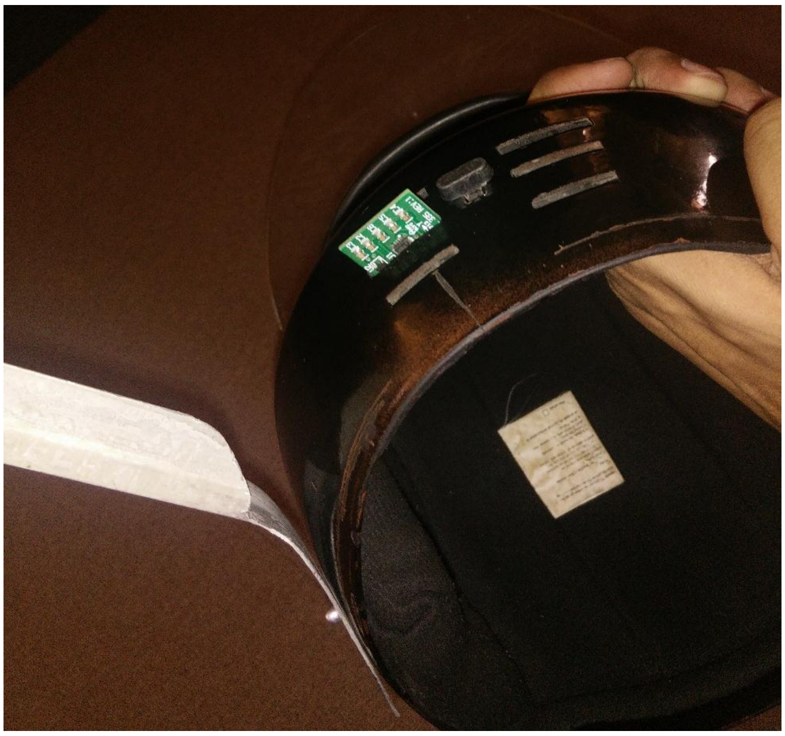


Fig. 4.10: Various Cases and its results

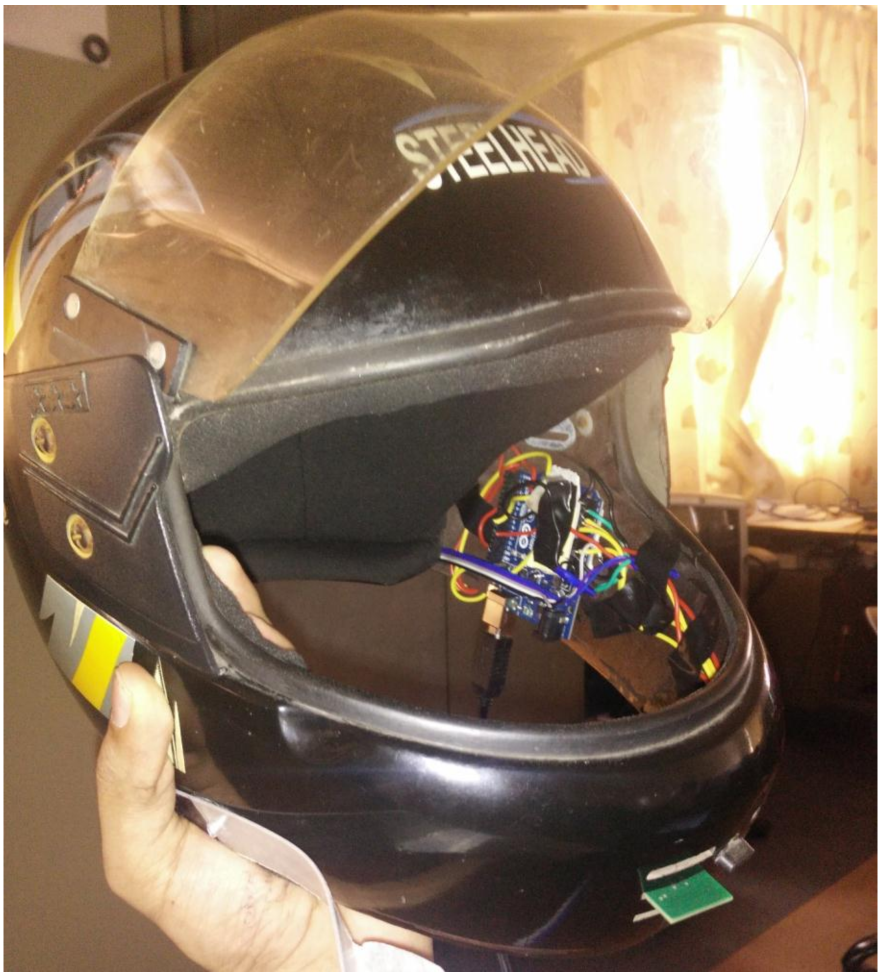


Fig. 4.11: Various Cases and its results

Chapter 5

Conclusion

5.1 Conclusion

Unlike Car-Drivers, Bikers don’t have a handsfree device to receive calls. Our project aims at solving this common issue among Bike-Riders

In this project, we have made a prototype of the helmet which has the feature of call acceptance and rejection. We used Arduino as the application level language for programming and MIT-App Inventor [Massachusetts Institute of Technology] for android programming.

5.2 Future Scope

The Helmet we made for this project is the base of our overall product. Further we will add many features to this helmet like GPS navigating system, a rear-view camera, accelerometer reading, headsets for sound, smart glass front (tinted during noon time- depending on glare) with inbuilt segmented screen

References

[1]

[2]

[3]

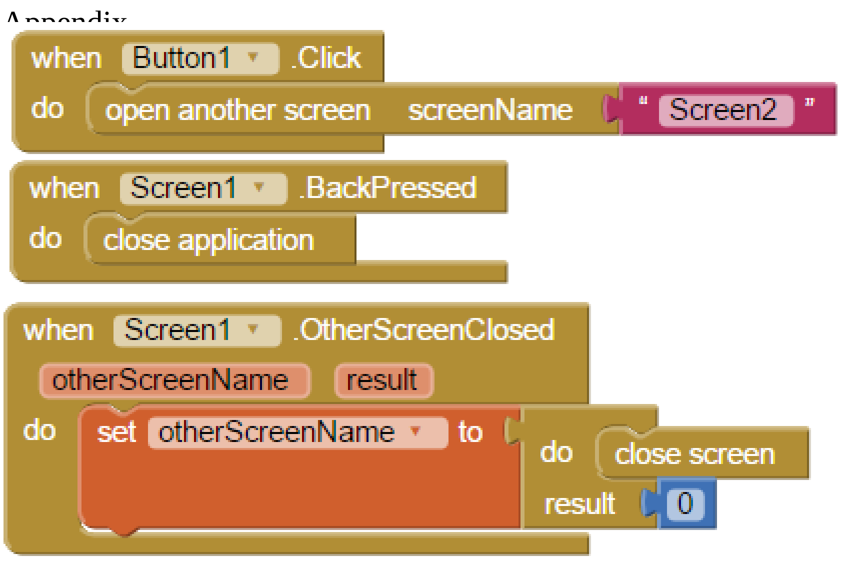
[4]

[5]

[6]

Appendix

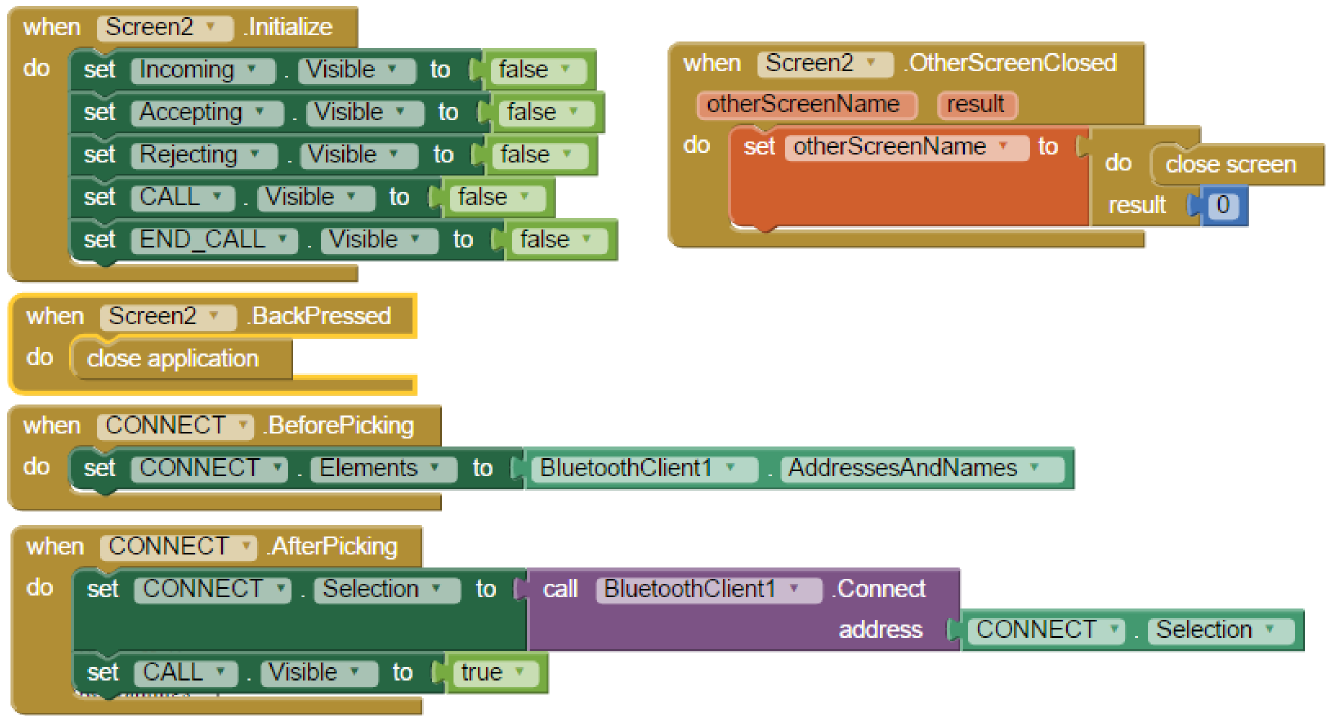
The codes of the app are:



```
when Button1 .Click
do  open another screen  screenName  " Screen2 "

when Screen1 .BackPressed
do  close application

when Screen1 .OtherScreenClosed
  otherScreenName  result
do  set otherScreenName to  do  close screen
                            result  0
```



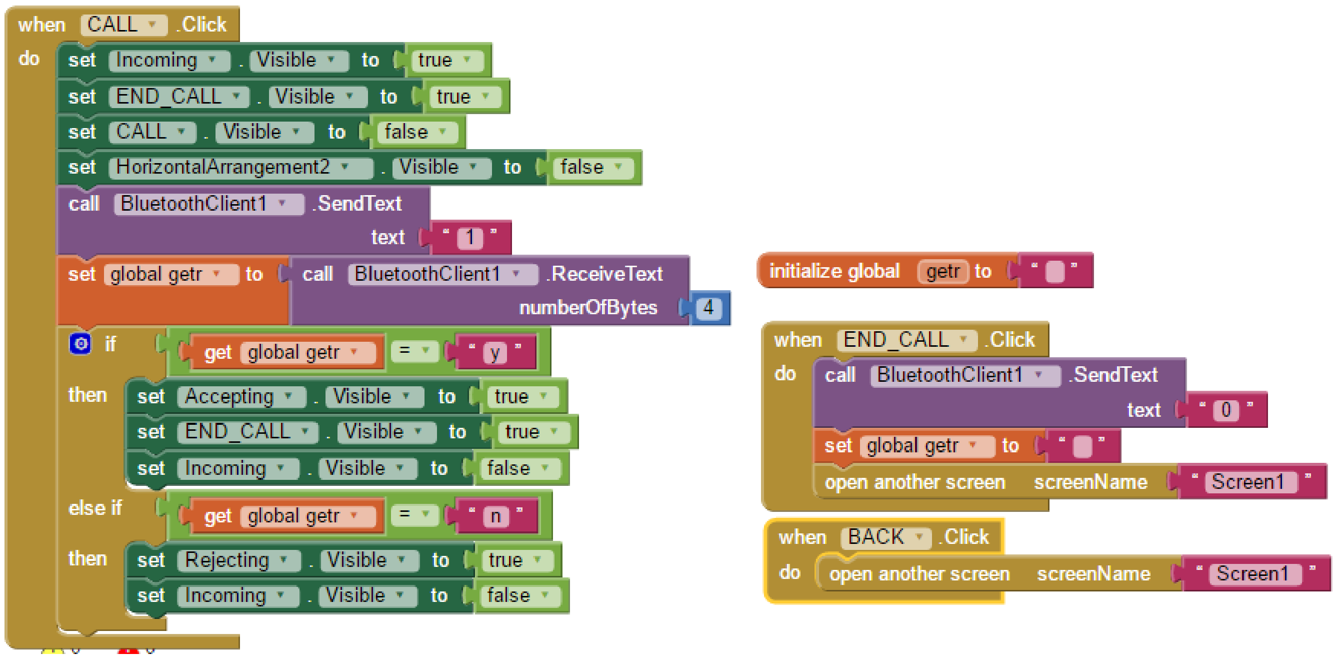
```
when Screen2 .Initialize
do  set Incoming . Visible to false
    set Accepting . Visible to false
    set Rejecting . Visible to false
    set CALL . Visible to false
    set END_CALL . Visible to false

when Screen2 .OtherScreenClosed
  otherScreenName  result
do  set otherScreenName to  do  close screen
                            result  0

when Screen2 .BackPressed
do  close application

when CONNECT .BeforePicking
do  set CONNECT . Elements to  BluetoothClient1 . AddressesAndNames

when CONNECT .AfterPicking
do  set CONNECT . Selection to  call BluetoothClient1 .Connect
                                                address  CONNECT . Selection
    set CALL . Visible to true
```



```
when CALL .Click
do  set Incoming . Visible to true
    set END_CALL . Visible to true
    set CALL . Visible to false
    set HorizontalArrangement2 . Visible to false
    call BluetoothClient1 .SendText
                         text "1"
    set global getr to call BluetoothClient1 .ReceiveText
                                        numberOfBytes 4
    if      get global getr = "y"
    then    set Accepting . Visible to true
            set END_CALL . Visible to true
            set Incoming . Visible to false
    else if get global getr = "n"
    then    set Rejecting . Visible to true
            set Incoming . Visible to false

initialize global getr to " "

when END_CALL .Click
do  call BluetoothClient1 .SendText
                         text "0"
    set global getr to " "
    open another screen screenName "Screen1"

when BACK .Click
do  open another screen screenName "Screen1"
```

Hardware Code:

```
//I2C_LCD
#include <Wire.h>  // Comes with Arduino IDE
#include <LiquidCrystal_I2C.h>
// set the LCD address to 0x27 for a 16 chars 2 line display
// Set the pins on the I2C chip used for LCD connections:
//  addr, en,rw,rs,d4,d5,d6,d7,bl,blpol
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);   // Set the LCD I2C
address//LCD
//For BT
#include <SoftwareSerial.h>
SoftwareSerial mySerial(9,10);
char inputString;
//GYRO >these constants describe the pins. They won't change:
const int groundpin = 18;             // analog input pin 4 -- ground
const int powerpin = 19;              // analog input pin 5 -- voltage
const int xpin = A3;                  // x-axis of the accelerometer
const int ypin = A2;                  // y-axis
const int zpin = A1;                  // z-axis (only on 3-axis models)
int a,b,c ;
```

```
void setup()
{
  // initialize the serial communications:
  Serial.begin(9600);
  /* Provide ground and power by using the analog inputs as normal digital pins.  This makes
it
      possible to directly connect the breakout board to the Arduino.  If you use the normal 5V
and
      GND pins on the Arduino, you can remove these lines. */
  pinMode(groundpin, OUTPUT);
  pinMode(powerpin, OUTPUT);
  digitalWrite(groundpin, LOW);
  digitalWrite(powerpin, HIGH);
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  lcd.print("Smart Helmet");
 mySerial.begin(9600);
 pinMode(13, OUTPUT);
}
void loop()
{
  if(mySerial.available()>0)
   {
      inputString= mySerial.read(); //read the input
      Serial.println(inputString);
   }
    a=analogRead(xpin);
    b=analogRead(ypin);
    c=analogRead(zpin);
    delay(100);
// TO PRINT GYRO VALUES IF NECESSARY
  Serial.print(a);
  // print a tab between values:
  Serial.print("\t");
  Serial.print(b);
  // print a tab between values
  Serial.print("\t");
  Serial.print(c);
  Serial.println();
  // delay before next reading:
  delay(100);
 if(inputString == '1')       //in case of '1' turn the LED on and pick display the incoming call
 { //----------------------------------------------------------------------------------------------------
    if((140<c<160)&&(500<a<530)&&(530<b<550))
      {
         lcd.setCursor(0, 1);
         lcd.write("INCOMING CALL");
         digitalWrite(12, HIGH);
         delay(1000);
       }
         if((140<c<160)&&(500<a<530)&&(b>535))        //input values according to gyro
                   {
                      lcd.setCursor(0, 1);
                      digitalWrite(13, HIGH);
                      lcd.write("CALL RECEIVED");               // Receive Call
                      delay(5000);
                      //mySerial.write('y');
                      mySerial.println("y");
                      Serial.print("yes");
                    }
              if((540<a)||((a<500))&&(500<b<534)&&(140<c<160))   //input values by gyro
                   {  lcd.setCursor(0, 1);
                      digitalWrite(13, LOW);
                      lcd.write("CALL REJECTED");               // Reject Call
                      delay(5000);
                      //mySerial.write("n");
                      mySerial.println("n");
                      Serial.print("no"); }}
```

```
else if(inputString == '0')      //incase of '0' turn the LED off and display normal state
{   lcd.setCursor(0, 1);
digitalWrite(13, LOW);
lcd.write("CALL END ");}}
```