Gesture Controlled Robot

Instructor: Prof. Anurag Lakhlani

Pratham Solanki (1401049)

Deval Shah (1401060)

Maharshi Bhavsar (1401061)

Index Page

Introduction:

Nowadays, robotics is becoming one of the most advanced in the field of technology. The applications of robotics mainly involve in automobiles, medical, construction, defense and also used as a fire fighting robot to help the people from the fire accident. In many application of controlling robotic gadget, it becomes quite hard and complicated when there comes the part of controlling it with remote or many different switches.

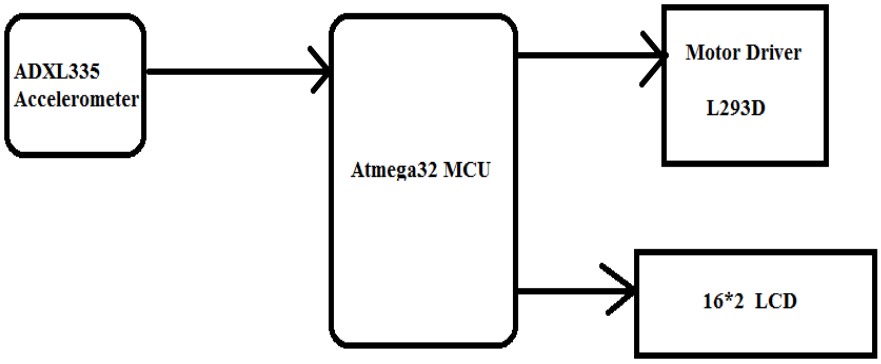
Mostly in military application, industrial robotics, construction vehicles in civil side, medical application for surgery. In this field it is quite complicated to control the robot or particular machine with remote or switches, sometime the operator may get confused in the switches and button itself, so a new concept is introduced to control the machine with the movement of hand which will simultaneously control the movement of robot. So, a new project is developed that is, an accelerometer based gesture control robot. The main goal of this project is to control the movement of the robot with hand gesture using accelerometer.

Project Background:

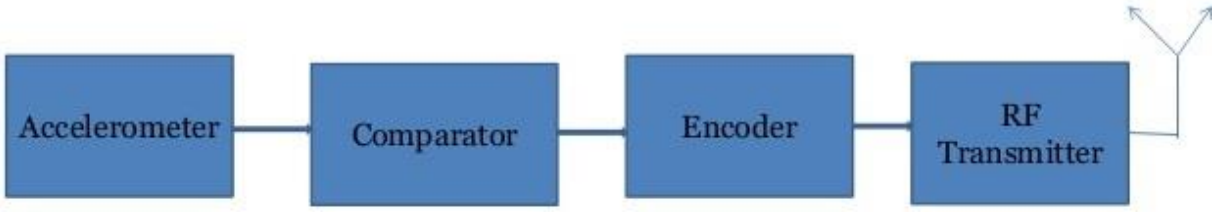
A Gesture Controlled robot is a kind of robot which can be controlled by your hand gestures not by buttons. You just need to wear a small transmitting device in your hand which includes an acceleration meter. This will transmit an appropriate command to the robot so that it can do whatever we want. The transmitting device includes an encoder IC (HT12E) which is used to encode the four-bit data and then it will transmit by an RF Transmitter module. At the receiving end an RF Receiver module receives the encoded data and decoder IC (HT12D) decodes it. This data is then processed by microcontroller and finally our motor driver to control the motors. In our project we have implemented wired gesture controlled robot and tried implementing wireless gesture controlled robot using RX TX module. Following pages in report would include both wired and wireless implementation. The stages explained in following page describe working of gesture controlled robot. The stages in which it is mentioned wireless explains what additional stages it would require for wireless implementation of (wired)gesture controlled robot.

In our project we have shown wired gesture controlled robot as our working demo

Block Diagram



Wired Gesture Controlled Robot



Block Diagram of Transmitter circuit

Block Diagram of Receiver circuit

Wireless Gesture Controlled Robot

2.1 How the gesture controlled robot works?

Stage 1:

The accelerometer sensor is initialized. It reads the analog values of the coordinates when a hand gesture is performed. This output from the accelerometer is connected to one of the ADC channels of the microcontroller. It converts the analog readings to digital.

Stage 2:

Once the digital readings of the coordinates are obtained, they are compared with their respective threshold values. Digital readings are displayed on LCD Based on the differences (between the threshold value and the recorded value) obtained the microcontroller decides what task should be performed in response to that particular hand gesture. The data for the decided task is generated. It means that as per decision that comes from threshold we can decide what values is to be passed to input of motor driver for necessary direction.

Stage 3: (Wireless)

The generated data is encoded using an encoder and then the encoded data is transmitted with help of an RF transmitter.

Stage 4: (Wireless)

The RF receiver receives the transmitted data and passes it on to the decoder. The encoded data that is received is decoded by the decoder.

Stage 5: (Wireless)

The decoded data contains the exact task that must be performed in response to the hand gesture.

Stage 6:

The data is passed on the motor drivers. They directly put the assigned task into action by moving the robot in the appropriate direction. LCD displays the direction of robot at runtime.

Components used:

(1) ATMega 32
(2) 16*2 LCD
(3) RF receiver
(4) RF transmitter
(5) HT12D Decoder
(6) HT12E Encoder
(7) SW-DIP8 switch
(8) ADXL335 accelerometer
(9) 10k potentiometer
(10) Resistors
(11) LED
(12) L293D motor driver
(13) Antennas
(14) General Metal Chassis
(15) DC motor of 150 RPM
(16) Batteries
(17) General purpose PCB

Selection Criteria for Major Components

1)Atmega32 MCU

Features

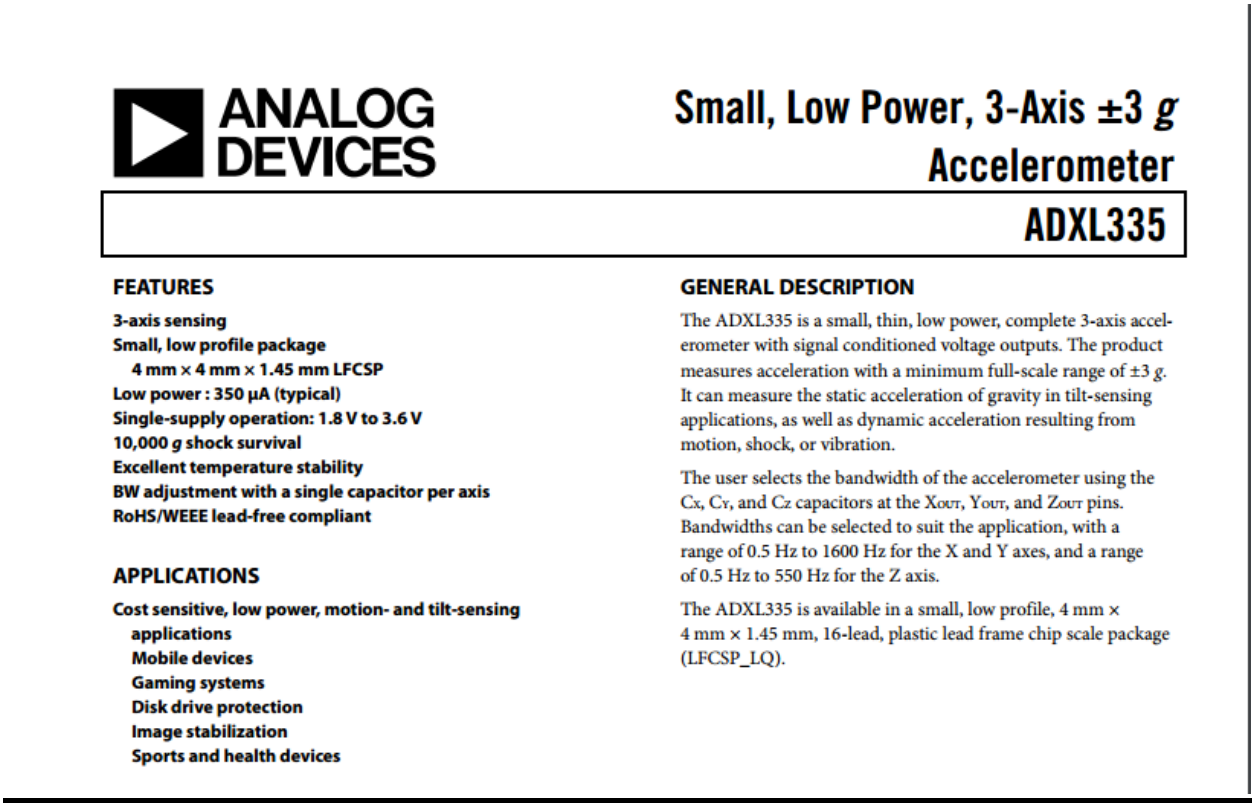
- High-performance, Low-power Atmel®AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 × 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
  - 32Kbytes of In-System Self-programmable Flash program memory
  - 1024Bytes EEPROM
  - 2Kbytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    8 Single-ended Channels
    7 Differential Channels in TQFP Package Only
    2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF

ATMEL®

8-bit AVR® Microcontroller with 32KBytes In-System Programmable Flash

ATmega32
ATmega32L

2)ADXL335 Accelerometer



ANALOG DEVICES

Small, Low Power, 3-Axis ±3 *g* Accelerometer

ADXL335

FEATURES

3-axis sensing
Small, low profile package
4 mm × 4 mm × 1.45 mm LFCSP
Low power : 350 µA (typical)
Single-supply operation: 1.8 V to 3.6 V
10,000 *g* shock survival
Excellent temperature stability
BW adjustment with a single capacitor per axis
RoHS/WEEE lead-free compliant

APPLICATIONS

Cost sensitive, low power, motion- and tilt-sensing applications
Mobile devices
Gaming systems
Disk drive protection
Image stabilization
Sports and health devices

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 C_X, C_Y, and C_Z capacitors at the X_{OUT}, Y_{OUT}, and Z_{OUT} 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).

- ADXL335 accelerometer is selected because it serves the purpose of measuring acceleration values at nice accuracy and cheap compared to other accelerometers in market.

3)RX(Receiver) and TX(Transmitter) module

Transmitter :

Working voltage: 3V - 12V fo max. power use 12V
Working current: max Less than 40mA max , and min 9mA
Resonance mode: (SAW)
Modulation mode: ASK
Working frequency: Eve 315MHz Or 433MHz
Transmission power: 25mW (315MHz at 12V)
Frequency error: +150kHz (max)
Velocity : less than 10Kbps

So this module will transmit up to 90m in open area .

Receiver :

Working voltage: 5.0VDC +0.5V
Working current:≤5.5mA max
Working method: OOK/ASK
Working frequency: 315MHz-433.92MHz
Bandwidth: 2MHz
Sensitivity: excel –100dBm (50Ω)
Transmitting velocity: <9.6Kbps (at 315MHz and -95dBm)

- RF Transmitter and Receiver are most cost effective in low cost wireless implementation.

4)L293D Motor Driver

L293x Quadruple Half-H Drivers

1 Features

- Wide Supply-Voltage Range: 4.5 V to 36 V
- Separate Input-Logic Supply
- Internal ESD Protection
- High-Noise-Immunity Inputs
- Output Current 1 A Per Channel (600 mA for L293D)
- Peak Output Current 2 A Per Channel (1.2 A for L293D)
- Output Clamp Diodes for Inductive Transient Suppression (L293D)

2 Applications

- Stepper Motor Drivers
- DC Motor Drivers
- Latching Relay Drivers

3 Description

The L293 and L293D devices are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, DC and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN.

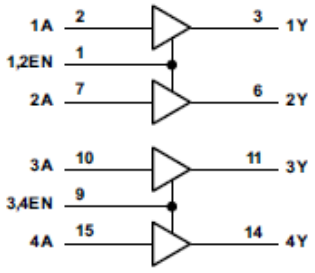
The L293 and L293D are characterized for operation from 0°C to 70°C.

Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
L293NE	PDIP (16)	19.80 mm × 6.35 mm
L293DNE	PDIP (16)	19.80 mm × 6.35 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Logic Diagram



Costing Table:

	Components	Price
1	Atmega32 Development Board	450
2	16*2 LCD	130
3	RF Receiver	170
4	RF Transmitter	70
5	HT12E Encoder	20
6	HT12D Decoder	20
7	SW-DIP8 Switch x 2	20
8	ADXL335 accelerometer	220
9	10k potentiometer	10
10	Resistors	10
11	LED	10
12	L293D Motor Driver	150
13	Antenna x 2	30
14	Metal Chassis	170
15	DC motor of 150 RPM x 2	100
16	Batteries 9 V x 4	100
17	General Purpose PCB x 3	150
18	Wires	100
	Total	1930

Circuit schematic for Wireless Implementation:

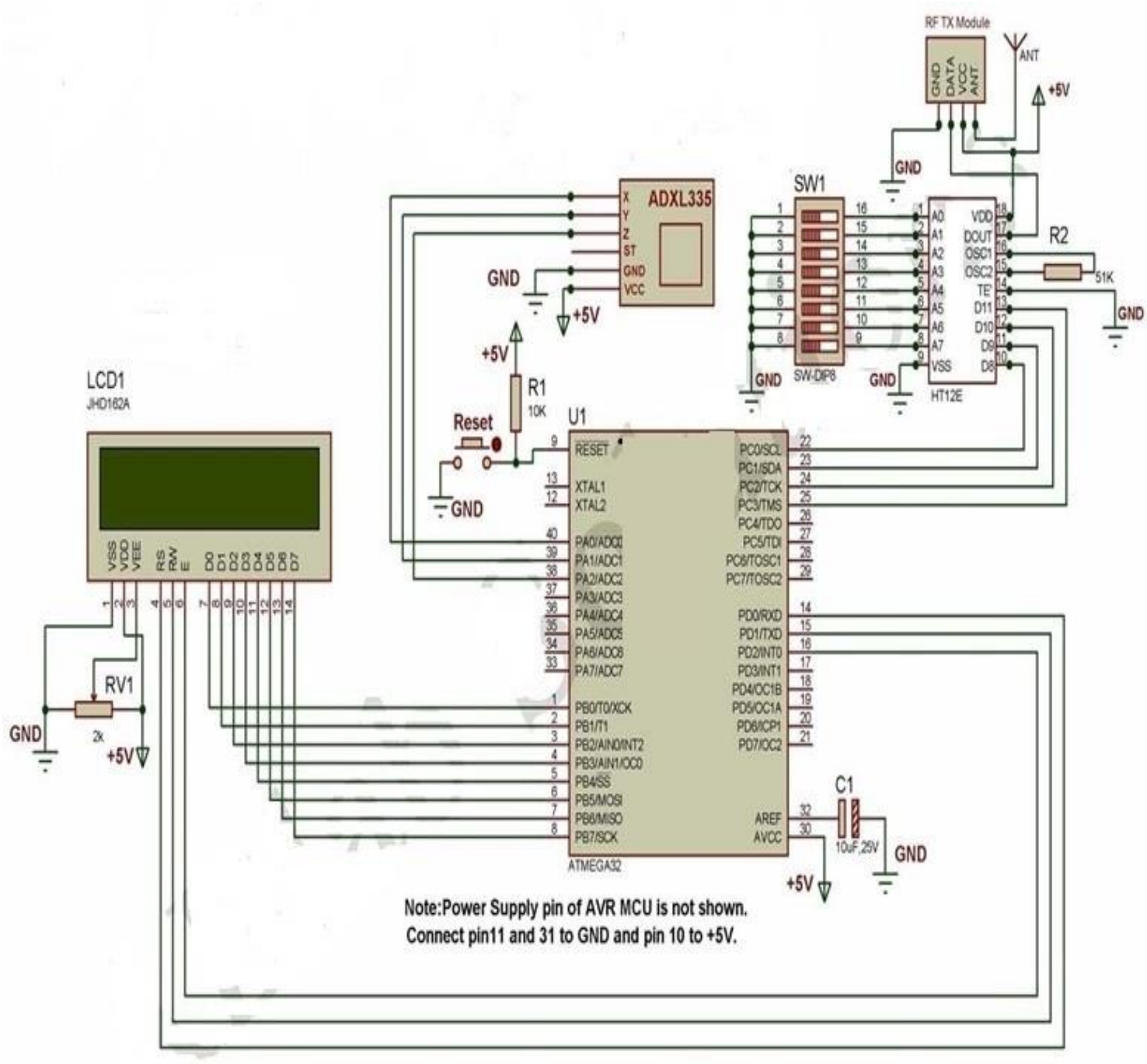


Fig 2.1 Transmitter Circuit

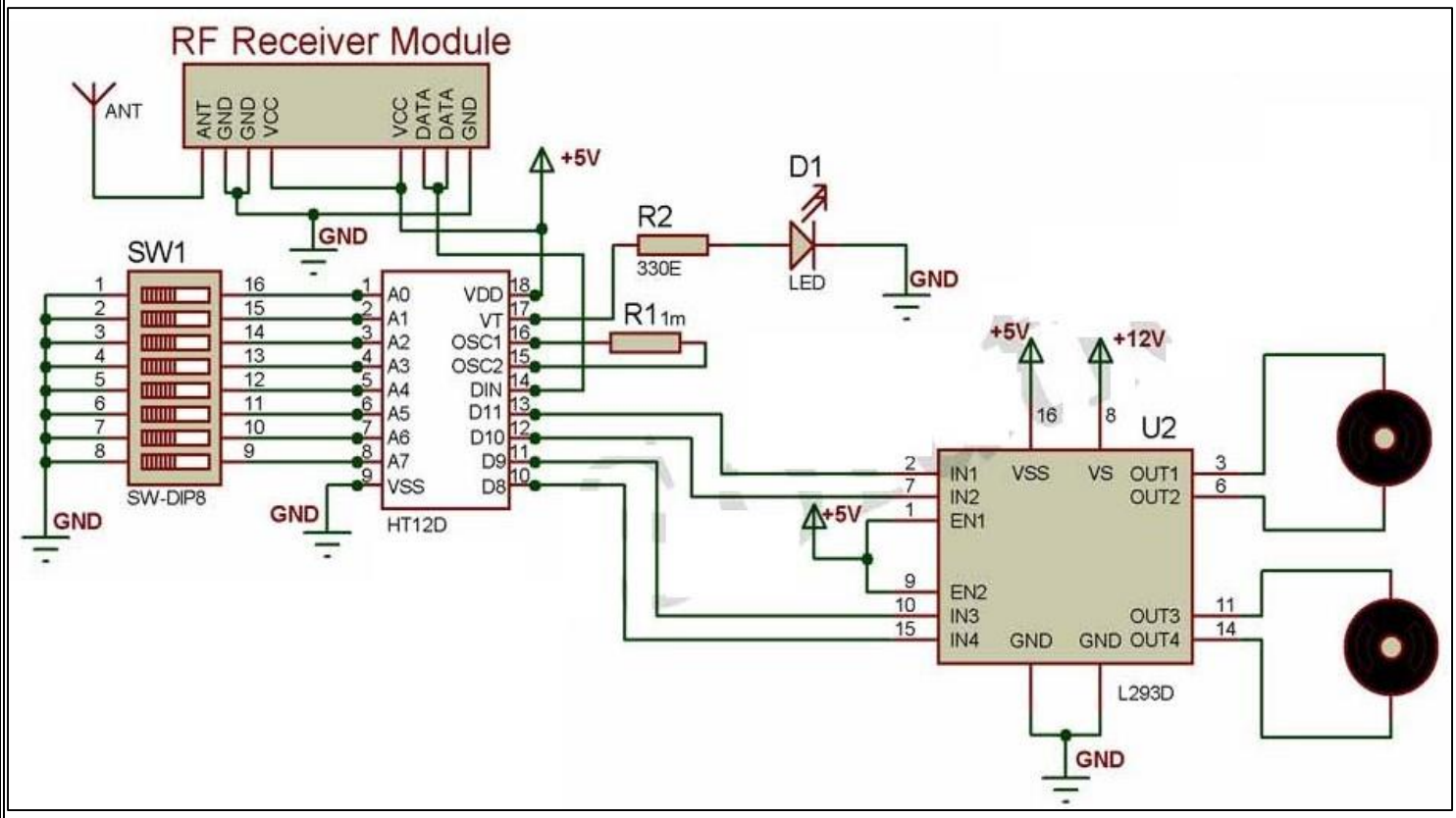


Fig 2.2 Receiver Circuit

- For **wired gesture controlled robot** the circuit schematic remains same except there would be one circuitry where the output from PC0-PC3 going to TX module would directly be given to motor driver because TX module is used to transmit the data wirelessly.

Explain examples of debugging and troubleshooting

Problem: During initial phase of project we were stuck in fetching values from accelerometer because as accelerometer was analog so continuously fetching values for every movement was a challenging problem.

Solution: We solved this problem by analyzing the datasheet of atmega32 and adxl335 rigorously and founded that for fetching values we need to do ADC conversion as well as set system freq. to 1MHz for the same.

Problem: We were then stuck in displaying values of adxl335 on LCD because after ADC conversion, when we displayed the values on LCD it would give us random characters.

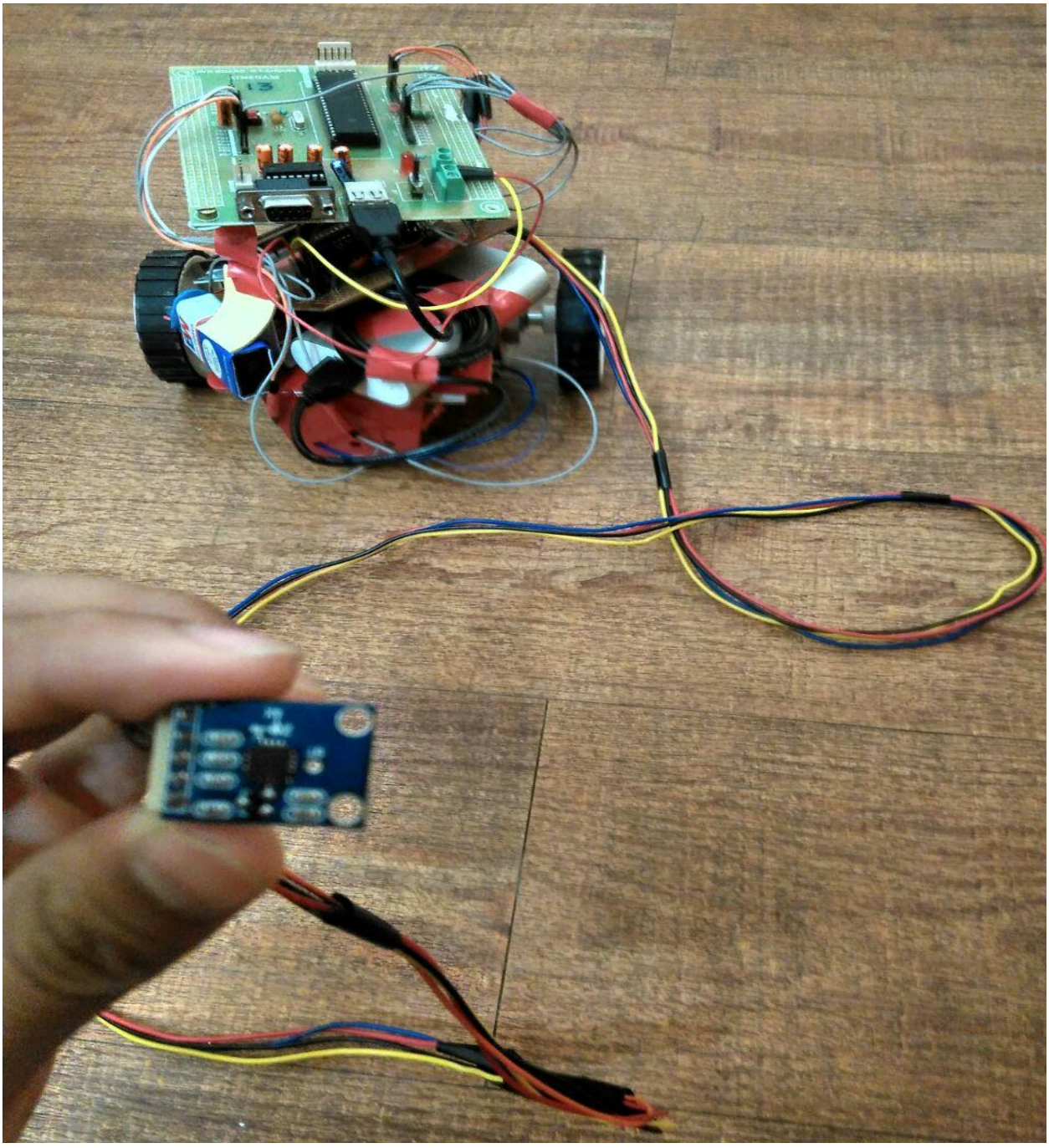
Solution: We solved this problem by analyzing one of the lab assignments where for displaying a number on LCD we have to convert it into ASCII and then pass it onto LCD_Data function for displaying each digit of number. Also by analyzing the datasheet of adxl335 we came to know that the range of it is [100-500,100-500] for x and y coordinates, so we would need to break a 3-digit number into three individual digits and convert it to ascii and then print it accordingly on LCD.

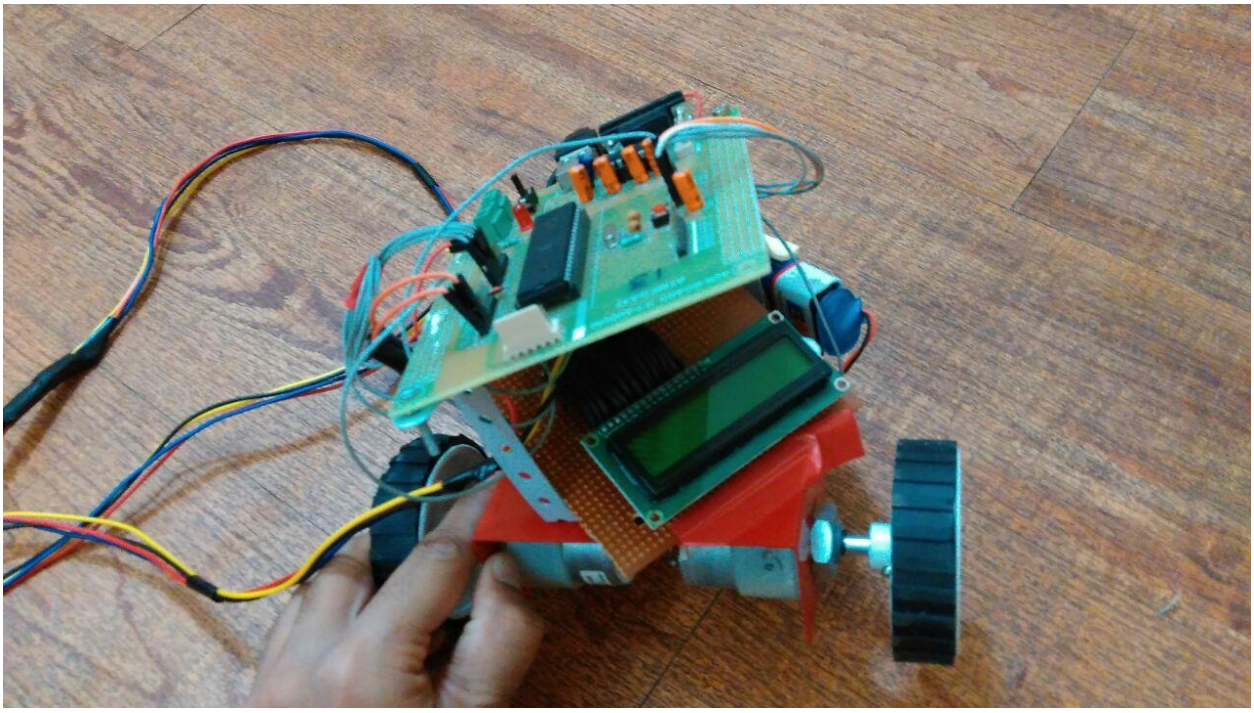
Problem: After displaying accelerometer values on LCD our next challenge was to define an accurate threshold (X and Y values) for hand gesture movement that would react according to its directions.

Solution: Firstly, for different hand movements (forward, left etc.) we analyzed accelerometer values and we decided that whenever the hand holding accelerometer turns at 90 degrees in any direction (left, right, forward or backward) the respective motor driver inputs in that direction would be passed.

Also along with these challenges, we learned soldering in process and soldered three PCB's for receiver, transmitter and LCD display.

Snapshots of working model



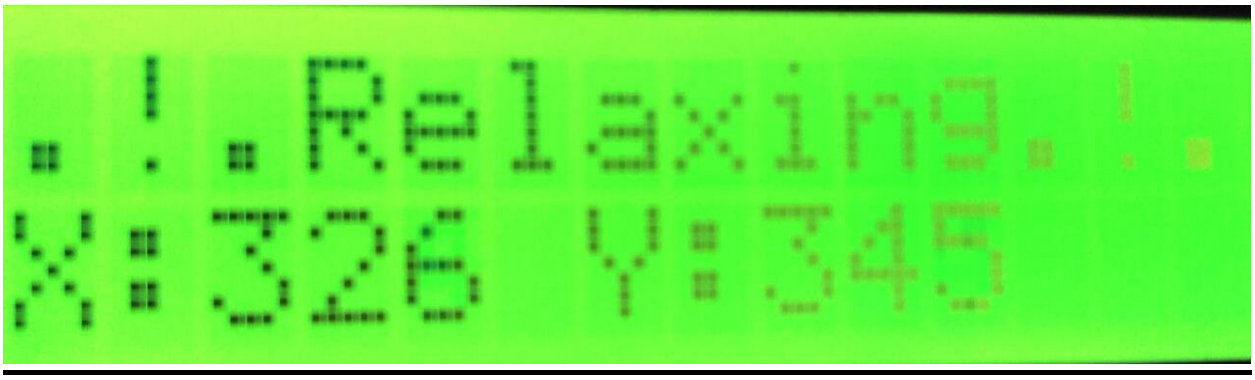



Forward
X:313 Y:394


Backward
X:418 Y:352


Left
X:368 Y:408


Right
X:358 Y:275


Relaxing
X:326 Y:345

Flowchart for Wireless Implementation:

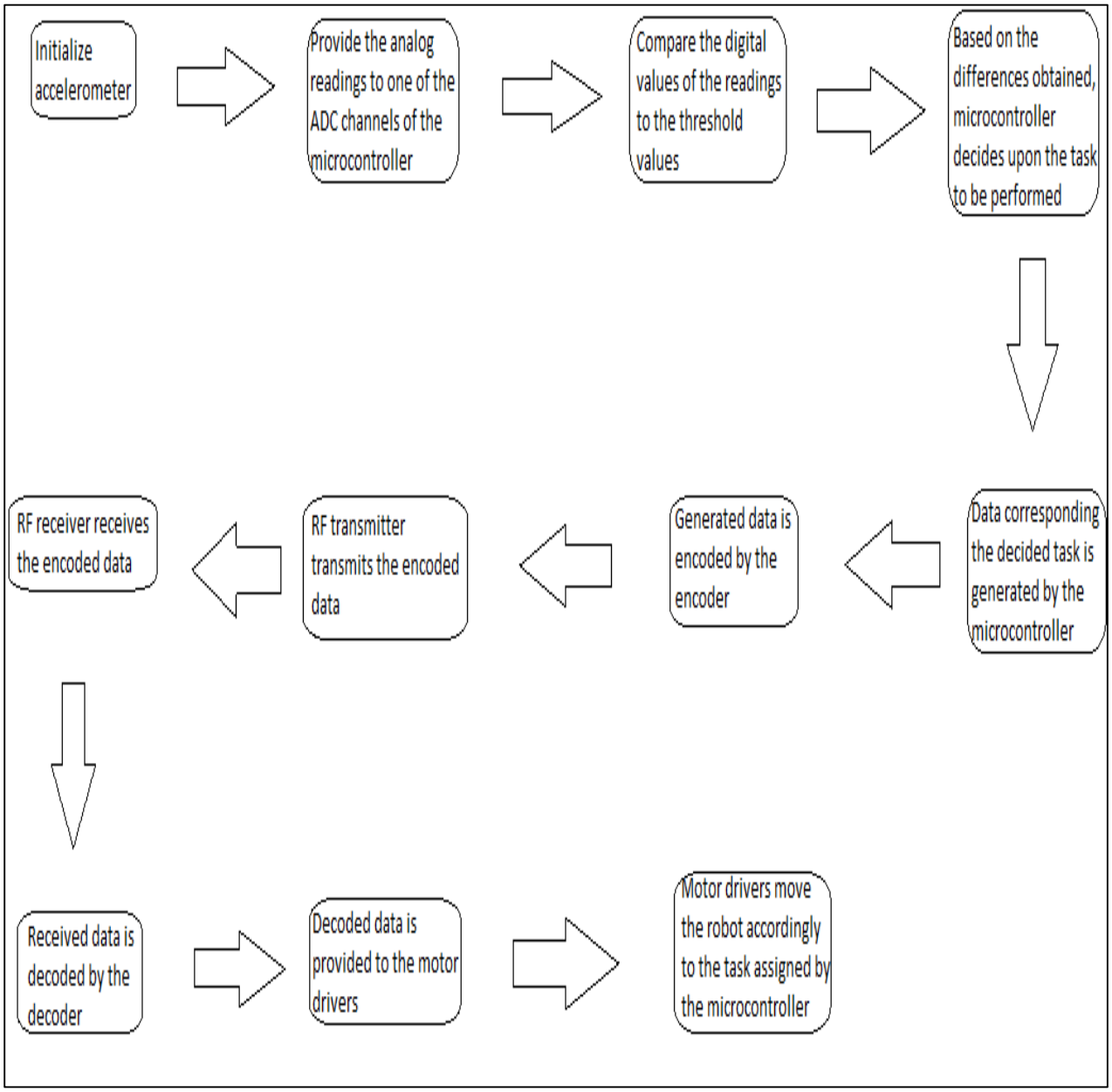


Fig 2.1 Flowchart for Wireless Implementation

Flowchart for Wired Implementation:

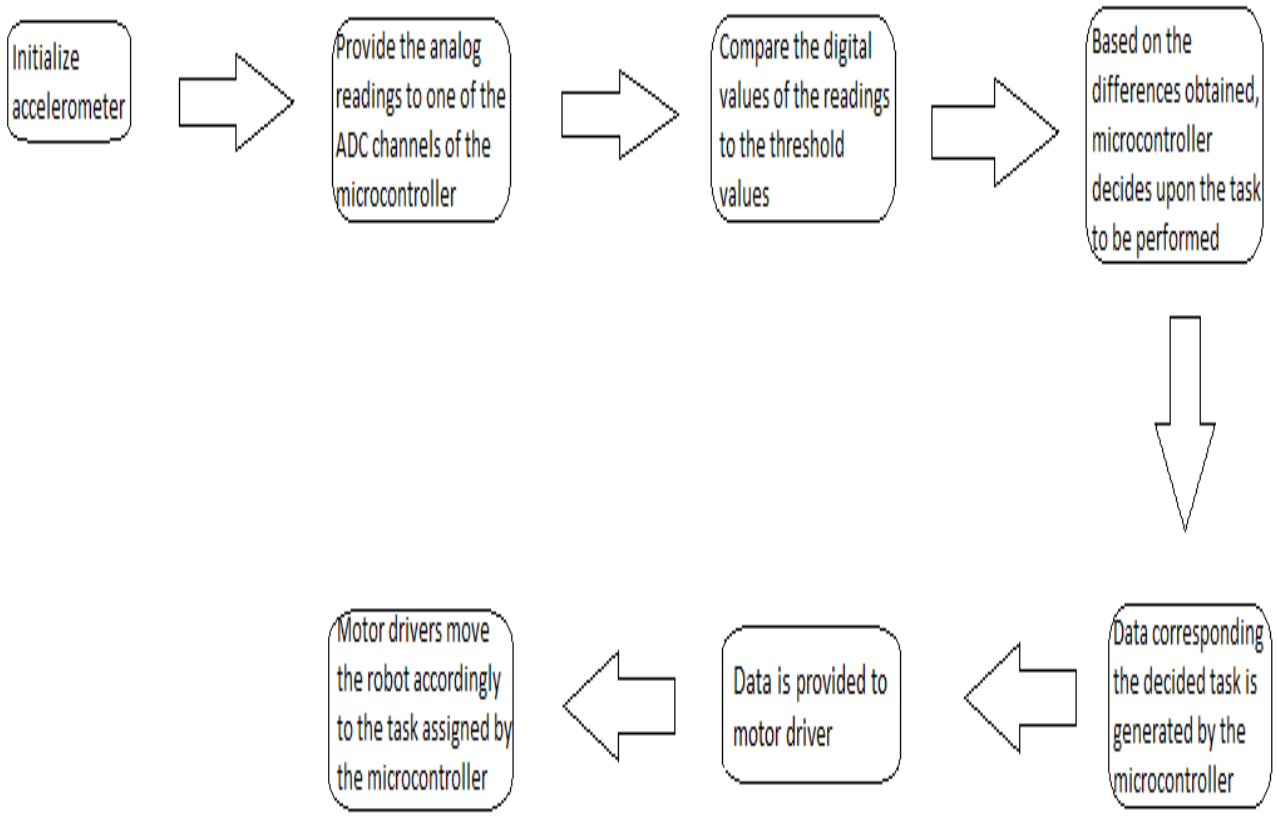


Fig 2.2 Flowchart for Wired Implementation

Code:

```
/*
  Code Abstract: The code written below does the following:
     1)First fetch accelerometer values by reading actual X and Y co-ordinates from analog channel 0 and 1.
     2)ADC conversion method converts that values into digital values
     3)The digital value of X and Y is then compared to threshold value which is determined by X and Y co-ordinates to set left, right, forward and backward directions.
     4)After comparing to threshold value, the atmega32 gives appropriate values to input of motor driver of robot to move in respective direction.
     5)The values of X and Y coordinates and direction in which robot is moving is displayed on runtime in LCD kept on robot.
*/

//LCD Connections
#define LCD_RS PORTA.B2   // RS
#define LCD_EN PORTD.B6   //Enable
#define LCD_D4 PORTC.B4   //Data Bit 4
#define LCD_D5 PORTC.B5   //Data Bit 5
#define LCD_D6 PORTC.B6   //Data Bit 6
#define LCD_D7 PORTC.B7   //Data Bit 7
```

```
#define         X_MIN         320
/*Defines the lower threshold for the x-axis value of adxl335*/

#define         X_MAX         400
/*Defines the upper threshold for the x-axis value of adxl335*/

#define         Y_MIN         320
/*Defines the lower threshold for the y-axis value of adxl335*/

#define         Y_MAX         400
/*Defines the upper threshold for the y-axis value of adxl335*/

int x_axis=0,y_axis=0;
/*Declaring temporary variables for storing digital values of X and Y*/

/*This function is declared to initialize the Analog to Digital Converter of
AVR microcontrollers. The Analog to Digital Converter is initialized with the
following features: -
        Auto Triggering mode is enabled,
        Free running mode is the source of trigger,
        ADC conversion is started,
        ADC frequency is 62.5 KHz at 1 MHz system frequency. */

void adc1_init ();
```

```
/*This function is declared to read the digital value of the ADC conversion of the selected channel with following conditions: - AVCC is selected as the reference voltage and ADC output is right adjusted. */
int read_adc_channel (unsigned char channel);

/*Function definitions*/
void adc1_init ()
{
ADCSRA=(1<<ADEN)|(1<<ADSC)|(1<<ADATE)|(1<<ADPS2);
SFIOR=0x00;
}

int read_adc_channel(unsigned char channel)
{
    int adc_value;
    unsigned char temp;
    ADMUX=(1<<REFS0)|channel;
    delay_ms(1);
    temp=ADCL;
    adc_value=ADCH;
    adc_value=(adc_value<<8)|temp;
    return adc_value;
}
```

```
/*The function is declared to convert the analog x-axis value of ADXL335 to digital value.*/
int read_adxl335_x_value(unsigned char channel);
/*The function is declared to convert the analog y-axis value of ADXL335 to digital value.*/
int read_adxl335_y_value(unsigned char channel);

/*Function definitions*/
int read_adxl335_x_value(unsigned char channel)
{
int x_value;
x_value=read_adc_channel(channel);
return x_value;
}

int read_adxl335_y_value(unsigned char channel)
{
int y_value;
y_value=read_adc_channel(channel);
return y_value;
}
```

```
void LCD_data(unsigned char Data)
{
PORTC=Data&0xF0;    // Send Higher nibble (D7-D4)
LCD_RS=1;           // Register Select =1 (for data select register)
LCD_EN=1;           //Enable=1 for H to L pulse
delay_us(5);
LCD_EN=0;


PORTC=((Data<<4)&0xF0); // Send Lower nibble (D3-D0)
LCD_EN=1;                   //Enable=1 for H to L pulse
delay_us(5);
LCD_EN=0;

delay_us(100);}
//LCD Print
void LCD_Print(char * str)
{
    unsigned char i=0;
    while(*(str+i)!=0) {// Till NULL character is reached, take each character
    LCD_data(*(str+i)); // Data sent to LCD data register
    i++;
    delay_ms(10);
    }
}
```

```
//LCD Command
void lcdcommand(unsigned char command)
{
        PORTC=command&0xF0;     // Send Higher nibble (D7-D4)
        LCD_RS=0;               // Register Select =0 (for Command register)
        LCD_EN=1;               //Enable=1 for H to L pulse
        delay_us(5);
        LCD_EN=0;
        delay_us(100);

        PORTC=((command<<4)&0xF0);  // Send Lower nibble (D3-D0)
        LCD_EN=1;                   //Enable=1 for H to L pulse
        delay_us(5);
        LCD_EN=0;
        delay_us(40);
}

// Cursor Posotion
void Cursor_Position(unsigned short int x,unsigned short int y)
{
        unsigned char firstcharadd[] ={0x80,0xC0}; // First line address 0X80
                                                   //Second line address 0XC0
        lcdcommand((firstcharadd[x-1]+y-1));
}
```

```
 //Function for clearing LCD Screen
void clear()
{
        lcdcommand(0x01);
         delay_ms(2);
}

 //LCD Iniatialize
void LCD_Initialize()
{
        LCD_EN=0;
        lcdCommand(0x33); // Initialize LCD for 4 bit mode
        lcdCommand(0x32); // Initialize LCD for 4 bit mode
        lcdCommand(0x28); // Initialize LCD for 5X7 matrix mode
        lcdCommand(0x0E); //Display on,cursor blinking
        clear();
        lcdCommand(0x06); //Shift cursor to right
        lcdCommand(0x82);
}
```

```
//Main Function
void main() {

/*Storing different directions in array to be displayed on LCD*/
  char disp_left[]=".!.Left.!.";
  char disp_right[]=".!.Right.!.";
  char disp_forward[]="..!!Forward.!.";
  char disp_backward[]=".!.Backward.!.";
  char disp_stationary[]=".!.Relaxing.!.";

  /*Storing different coordinates in array to be displayed on LCD*/
  char disp2[]="X:";
  char disp3[]="Y:";
 //Set-up PORTS for LCD
  DDRD = 0x0F;
  PORTD = 0x00;
  DDRC=0xF0;      // For D3-D0
  DDRA.B2=1;      //For RS
  DDRD.B6=1;      //For Enable

 /*Initializing the ADC conversion*/
 adc1_init();

/*Initializing LCD*/
 LCD_Initialize();
```

```
while(1){
  x_axis = read_adxl335_x_value(0);
  /*Reading x-axis value of Accelerometer Sensor*/
  y_axis = read_adxl335_x_value(1);
  /*Reading y-axis value of Accelerometer Sensor*/

  Cursor_Position(2,1);  //Using Cursor Position fn to display on row 2 //col 1
  LCD_Print(disp2);      //Prints X:

  /*At cursor position 3,4,5 it prints X coordinate value at runtime after acsii conversion*/
  Cursor_Position(2,3);//Using Cursor Position fn to display on row 2 //col 3
  LCD_Data((x_axis/100)|0x30);

  Cursor_Position(2,4);//Using Cursor Position fn to display on row 2 //col 4
  LCD_Data(((x_axis/10)%10)|0x30);

  Cursor_Position(2,5);//Using Cursor Position fn to display on row 2 //col 5
  LCD_Data(x_axis%10|0x30);

  /*Prints blank space*/
  Cursor_Position(2,6);//Using Cursor Position fn to display on row 2 //col 6
  LCD_Data(20);

  Cursor_Position(2,7);    //Using Cursor Position fn to display on row 2 //col 7
```

```
  LCD_Print(disp3);       //Prints Y:

  /*At cursor position 9,10,11 it prints Y coordinate value at runtime after
acsii conversion*/
  Cursor_Position(2,9);    //Using Cursor Position fn to display on row 2 //col 9
  LCD_Data((y_axis/100)|0x30);

  Cursor_Position(2,10);  //Using Cursor Position fn to display on row 2 //col 5
  LCD_Data(((y_axis/10)%10)|0x30);

  Cursor_Position(2,11);  //Using Cursor Position fn to display on row 2 //col 5
  LCD_Data(y_axis%10|0x30);



  if(x_axis>X_MAX)
   {
     PORTD=0x0A;
     /*Transmitter will transmit 0x0A to drive Robot in forward direction*/
     Cursor_Position(1,1);
     /*Using Cursor Position fn to display on row 1 //col 1*/
     LCD_Print(disp_forward);
     /*Prints .!.Forward.!. on LCD*/
   }
   else if(x_axis<X_MIN)
   {
     PORTD=0x05;
```

```
  /*Transmitter will transmit 0x05 to drive Robot in reverse direction*/
  Cursor_Position(1,1);
  /*Using Cursor Position fn to display on row 1 //col 1*/
  LCD_Print(disp_backward);
  /*Prints .!.Backward.!. on LCD*/

}
else if(y_axis>Y_MAX)
{
  PORTD=0x08;
  /*Transmitter will transmit 0x08 to drive Robot in right direction*/
  Cursor_Position(1,1);
  /*Using Cursor Position fn to display on row 1 //col 1*/
  LCD_Print(disp_right);
  /*Prints .!.Right.!. on LCD*/
}
else if(y_axis<Y_MIN)
{
  PORTD=0x02;
  /*Transmitter will transmit 0x02 to drive Robot in left direction*/
  Cursor_Position(1,1);
  /*Using Cursor Position fn to display on row 1 //col 1*/
  LCD_Print(disp_left);
  /*Prints .!.Left.!. on LCD*/
}
```

```
    else
    {
       PORTD=0x0F;
       /*Transmitter will transmit 0x0F to keep in stationary position*/
       Cursor_Position(1,1);
       /*Using Cursor Position fn to display on row 1 //col 1*/
       LCD_Print(disp_stationary);
       /*Prints .!.Relaxing.!. on LCD*/
    }
  }
}
```

Conclusions

This project has been a huge learning curve for each one of us in terms of understanding of real world applications and how embedded systems are designed and what are the different phases in building embedded systems.

We learned how to build a robot from scratch and interfacing of atmega32 with different types of sensors like adxl335 (accelerometer + gyro), digital accelerometers like mpu6050 etc. ADC conversions techniques, wireless signal transmission and reception techniques etc. were cleared in our mind through this project.

We used concepts that were clearly taught by our Prof. Anurag Sir in class and TA's in lab about ADC conversion, interfacing of mcu with different sensors and other devices.

Professor and TA's have helped us greatly in completing this project with proper guidance.

Project Timeline

	23/03/16	04/04/16	11/04/16	18/04/16	25/04/16	09/05/16
Finalizing Project Idea and submission of assignment 1	✓					
Components and basic hardware connection		✓				
Code testing on individual components		✓				
Code testing and debugging			✓			
Assembly of hardware and code-1				✓		
Assembly of hardware and code-2					✓	
Design improvements and final testing						✓

References

[1] Muhammad Ali Mazidi and Sarmad Naimi and Sepehr Naimi," ADC and Sensor interfacing" in the avr microcontroller and embedded system, pp 464-483.

[2]http://www.ablab.in/ll293d-driver-interfacing-with-avr-atmega32-microcontroller/

[3] LCD 16x2 datasheet: www.engineersgarage.com/electronic-components/16x2-lcd-module-datasheet/

[4] ATMega32 datasheet, which was part of Embedded Systems Design lab documents