EVD520

Internet of Things - Project 1

Smart Game Controller

Final Report

Group 01:

Dhruti Chandarana (201501015)

Hena Ghonia (201501032)

Shanaya Mehta (201501055)

Index

1. Project Initiation Details

a. Motivation

There are many video game controllers available today. As we are moving towards smart futures, controllers should be made smarter as well. With this idea in mind, we decided to make a smart controller that would enable a person to play a game through hand movements.

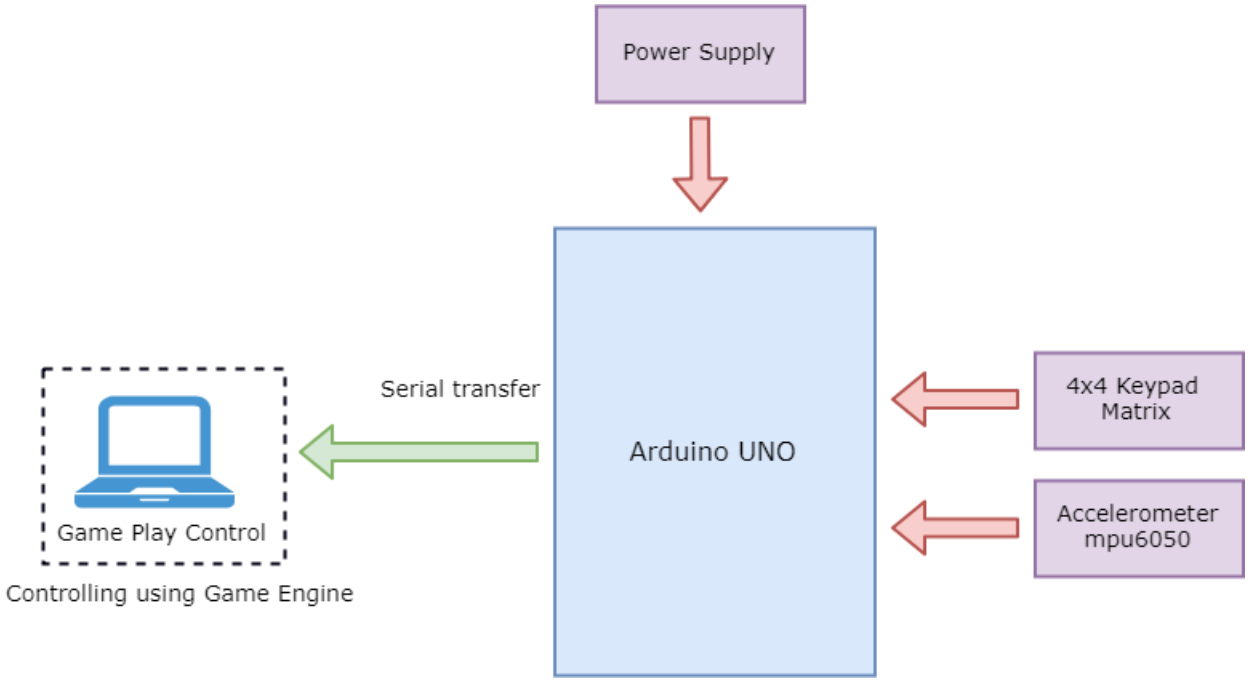
b. Description

The device consists of a simple controller that is made up of an embedded sensor, the accelerometer. The game we will make will be a car racing game made using the Unity web engine. The player would control the movement of the car in the game by simply turning the hand wrist, which would turn the car, accordingly. If the player wants to turn the car to the right side of the lane, the player will simply turn their wrist and the car will turn; likewise, to turn to the right side of the lane, the player will turn the wrist to the left. Furthermore, the gears can be manually changed by entering the gear number in a keypad. The default gear will be gear 1, which can be changed gradually to gear 5 as the car accelerates, just like gears change in a manual gear car. The game data will be shared through either the ethernet module or the bluetooth module. The speed of the car will be displayed on a LCD and the camera view can be changed from 4x4 keyboard.

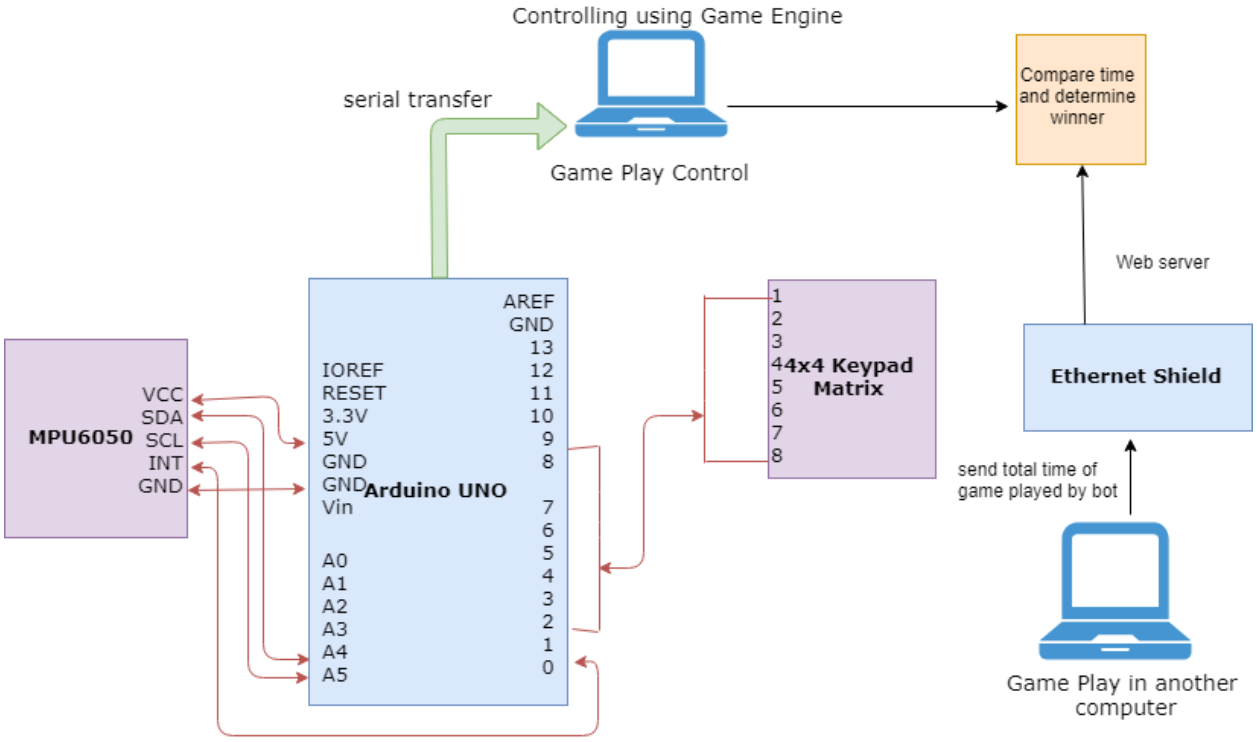
c. Final Outcome

The final outcome of the project will be a smart internet-enabled controller, which will empower the player to play a game controlled by the wireless controller in which the movement of the game object (for example, a car) is controlled by hand gestures, with ease.

2. Block Diagram



3. Circuit Diagram



Arduino Connection Circuit Diagram

4. Details of components:

a. Operating Principles

MPU6050

MPU6050 sensor module is an integrated 6-axis Motion tracking device.
It has a 3-axis Gyroscope, 3-axis Accelerometer, Digital Motion Processor and a Temperature sensor, all in a single IC. It can accept inputs from other sensors like 3-axis magnetometer, pressure sensor using its Auxiliary I2C bus. If external 3-axis magnetometer is connected, it can provide complete 9-axis Motion Fusion output.
A microcontroller can communicate with this module using I2C communication protocol.
Gyroscope and accelerometer reading along X, Y and Z axes are available in 2's complement form. Temperature reading is available in signed integer form. Gyroscope readings are in degrees per second (dps) unit; Accelerometer readings are in g unit; and Temperature reading is in degrees Celsius. Acceleration along the axes deflects the movable mass.

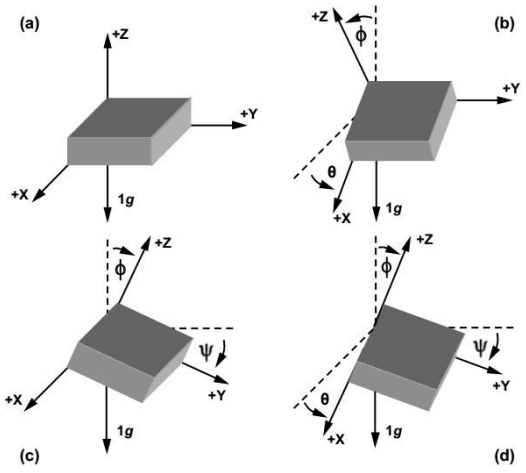
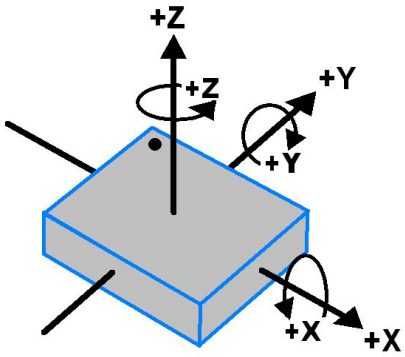


Figure: 3-axis accelerometer



MPU-6050
Orientation & Polarity of Rotation

Reference: [1]

Keypad

Keypad is used as an input device to read the key pressed by the user and to process it. 4x4 keypad consists of 4 rows and 4 columns. Switches are placed between the rows and columns. A key press establishes a connection between the corresponding row and column, between which the switch is placed.

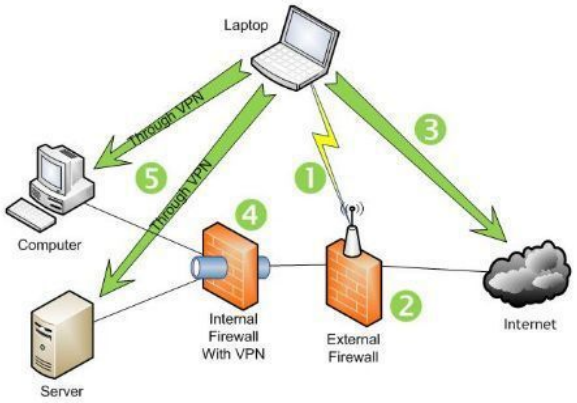
When we want to interface one key to the microcontroller then it needs one GPIO pin. To save some GPIO pins of microcontroller, we can use matrix keypad. Matrix keypad is nothing but keys arrange in row and column. To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, and then it reads the pressed column.

Wi-Fi Module

Radio signal is the base of operation of Wi-Fi. It is made up of three elements which are essential for its working:

1. Signal
2. Antenna
3. Router

The radio signals are transmitted by antenna and routers and they are received by Wi-Fi receiver such as computers.



Wi-Fi Connections

4. Details of components:
b. Arduino interfacing code

MPU6050

To stabilize the values obtained from the sensor, we need to set the offset.

Code for stabilizing sensor values:

```
// I2Cdev and MPU6050 must be installed as libraries
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

//////////////////////////////////   CONFIGURATION   //////////////////////////////
//Change this 3 variables if you want to fine tune the skecth to your needs.
int buffersize=1000;     //Amount of readings used to average, make it higher to get more precision but sketch will be slower  (default:1000)
int acel_deadzone=8;     //Acelerometer error allowed, make it lower to get more precision, but sketch may not converge  (default:8)
int giro_deadzone=1;     //Giro error allowed, make it lower to get more precision, but sketch may not converge  (default:1)

// default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for InvenSense evaluation board)
// AD0 high = 0x69
//MPU6050 accelgyro;
MPU6050 accelgyro(0x68); // <-- use for AD0 high

int16_t ax, ay, az,gx, gy, gz;

int mean_ax,mean_ay,mean_az,mean_gx,mean_gy,mean_gz,state=0;
int ax_offset,ay_offset,az_offset,gx_offset,gy_offset,gz_offset;

//////////////////////////////////   SETUP   ////////////////////////////////////
void setup() {
  // join I2C bus (I2Cdev library doesn't do this automatically)
  Wire.begin();
  // COMMENT NEXT LINE IF YOU ARE USING ARDUINO DUE
  TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz). Leonardo measured 250kHz.
```

```
void setup() {
  // join I2C bus (I2Cdev library doesn't do this automatically)
  Wire.begin();
  // COMMENT NEXT LINE IF YOU ARE USING ARDUINO DUE
  TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz). Leonardo measured 250kHz.

  // initialize serial communication
  Serial.begin(115200);

  // initialize device
  accelgyro.initialize();

  // wait for ready
  while (Serial.available() && Serial.read()); // empty buffer
  while (!Serial.available()){
    Serial.println(F("Send any character to start sketch.\n"));
    delay(1500);
  }
  while (Serial.available() && Serial.read()); // empty buffer again

  // start message
  Serial.println("\nMPU6050 Calibration Sketch");
  delay(2000);
  Serial.println("\nYour MPU6050 should be placed in horizontal position, with package letters facing up. \nDon't touch it until you see a finish message.\n");
  delay(3000);
  // verify connection
  Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050 connection failed");
  delay(1000);
  // reset offsets
  accelgyro.setXAccelOffset(0);
```

```
  Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050 connection failed");
  delay(1000);
  // reset offsets
  accelgyro.setXAccelOffset(0);
  accelgyro.setYAccelOffset(0);
  accelgyro.setZAccelOffset(0);
  accelgyro.setXGyroOffset(0);
  accelgyro.setYGyroOffset(0);
  accelgyro.setZGyroOffset(0);
}

////////////////////////////////   LOOP   ////////////////////////////////////
void loop() {
  if (state==0){
    Serial.println("\nReading sensors for first time...");
    meansensors();
    state++;
    delay(1000);
  }

  if (state==1) {
    Serial.println("\nCalculating offsets...");
    calibration();
    state++;
    delay(1000);
  }

  if (state==2) {
    meansensors();
    Serial.println("\nFINISHED!");
    Serial.print("\nSensor readings with offsets:\t");
    Serial.print(mean_ax);
    Serial.print("\t");
```

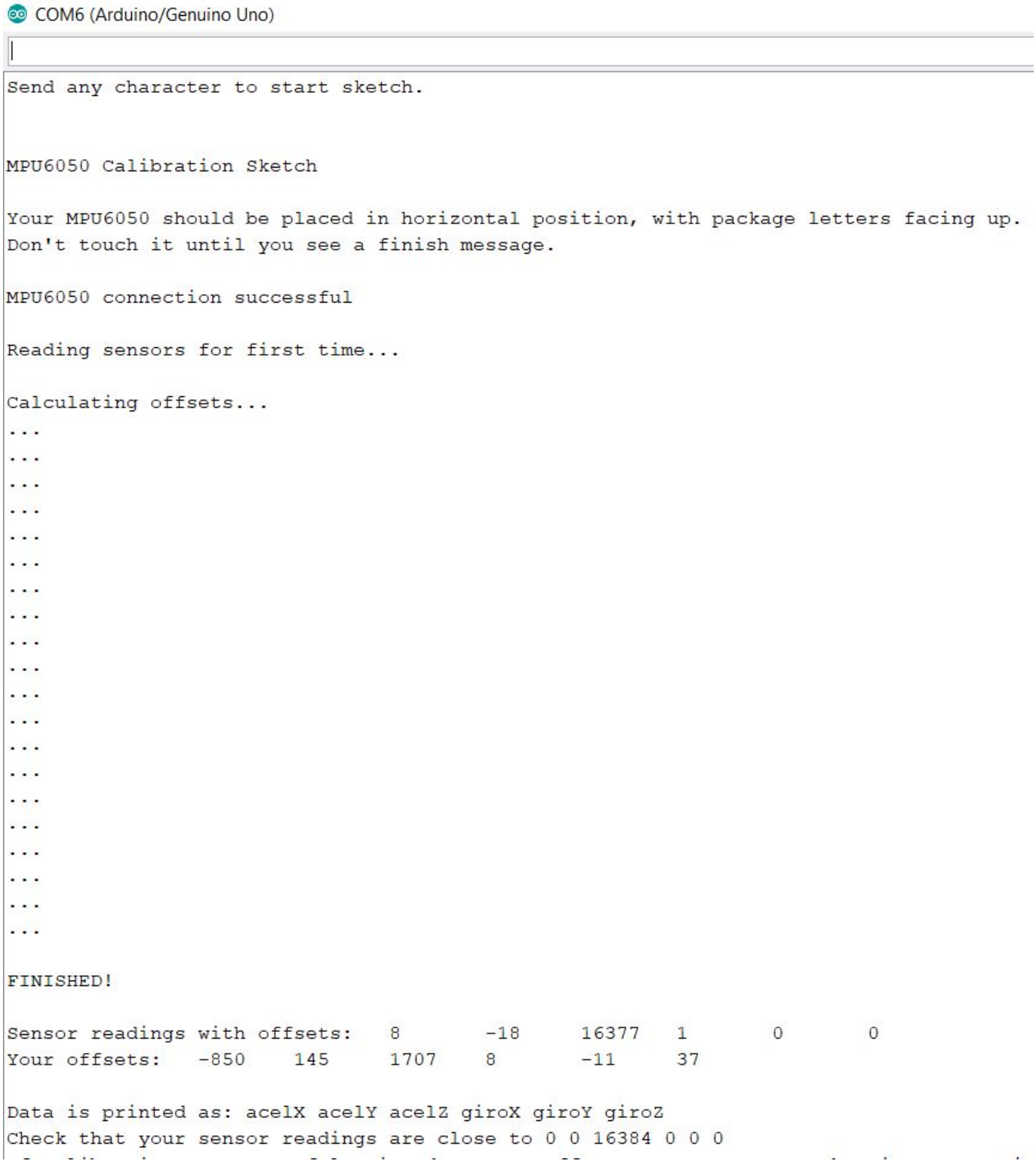
```
if (state==2) {
  meansensors();
  Serial.println("\nFINISHED!");
  Serial.print("\nSensor readings with offsets:\t");
  Serial.print(mean_ax);
  Serial.print("\t");
  Serial.print(mean_ay);
  Serial.print("\t");
  Serial.print(mean_az);
  Serial.print("\t");
  Serial.print(mean_gx);
  Serial.print("\t");
  Serial.print(mean_gy);
  Serial.print("\t");
  Serial.println(mean_gz);
  Serial.print("Your offsets:\t");
  Serial.print(ax_offset);
  Serial.print("\t");
  Serial.print(ay_offset);
  Serial.print("\t");
  Serial.print(az_offset);
  Serial.print("\t");
  Serial.print(gx_offset);
  Serial.print("\t");
  Serial.print(gy_offset);
  Serial.print("\t");
  Serial.println(gz_offset);
  Serial.println("\nData is printed as: acelX acelY acelZ giroX giroY giroZ");
  Serial.println("Sensor readings are close to 0 0 16384 0 0 0");

  while (1);
}
```

```
void meansensors(){
  long i=0,buff_ax=0,buff_ay=0,buff_az=0,buff_gx=0,buff_gy=0,buff_gz=0;

  while (i<(buffersize+101)){
    // read raw accel/gyro measurements from device
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    if (i>100 && i<=(buffersize+100)){ //First 100 measures are discarded
      buff_ax=buff_ax+ax;
      buff_ay=buff_ay+ay;
      buff_az=buff_az+az;
      buff_gx=buff_gx+gx;
      buff_gy=buff_gy+gy;
      buff_gz=buff_gz+gz;
    }
    if (i==(buffersize+100)){
      mean_ax=buff_ax/buffersize;
      mean_ay=buff_ay/buffersize;
      mean_az=buff_az/buffersize;
      mean_gx=buff_gx/buffersize;
      mean_gy=buff_gy/buffersize;
      mean_gz=buff_gz/buffersize;
    }
    i++;
    delay(2); //Needed so we don't get repeated measures
  }
}
```

```
void calibration(){
  ax_offset=-mean_ax/8;
  ay_offset=-mean_ay/8;
  az_offset=(16384-mean_az)/8;

  gx_offset=-mean_gx/4;
  gy_offset=-mean_gy/4;
  gz_offset=-mean_gz/4;
  while (1){
    int ready=0;
    accelgyro.setXAccelOffset(ax_offset);
    accelgyro.setYAccelOffset(ay_offset);
    accelgyro.setZAccelOffset(az_offset);

    accelgyro.setXGyroOffset(gx_offset);
    accelgyro.setYGyroOffset(gy_offset);
    accelgyro.setZGyroOffset(gz_offset);

    meansensors();
    Serial.println("...");

    if (abs(mean_ax)<=acel_deadzone) ready++;
    else ax_offset=ax_offset-mean_ax/acel_deadzone;

    if (abs(mean_ay)<=acel_deadzone) ready++;
    else ay_offset=ay_offset-mean_ay/acel_deadzone;

    if (abs(16384-mean_az)<=acel_deadzone) ready++;
    else az_offset=az_offset+(16384-mean_az)/acel_deadzone;

    if (abs(mean_gx)<=giro_deadzone) ready++;
    else gx_offset=gx_offset-mean_gx/(giro_deadzone+1);
```

- **Output for MPU6050**

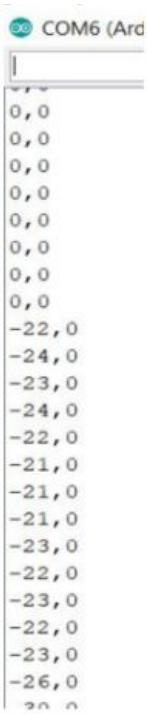
  Offset for accelerometer and gyrometer in X, Y, Z direction is shown as below:



- **Mapping output of MPU6050 in order to interface with Game Engine**

  Now we feed the sensor output values in the code below to obtain scaled values for the accelerometer in X and Y direction and keypad value as to change camera mode in game (1 for normal camera mode, 2 for far camera mode and 3 for first person camera mode)which are ultimately passed to the game code in Unity through the communication port, as shown below:

```
void setup() {
  lastReport = millis();
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif
    // initialize device
    DEBUG_PRINTLN("Initializing I2C devices...");
    accelgyro.initialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    accelgyro.setXGyroOffset(-850);
    accelgyro.setYGyroOffset(145);
    accelgyro.setZGyroOffset(1707);
    accelgyro.setXAccelOffset(8;
    accelgyro.setYAccelOffset(-11);
    accelgyro.setZAccelOffset(37);//1688 factory default for my test chip

    // verify connection
    DEBUG_PRINTLN("Testing device connections...");
    DEBUG_PRINTLN(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050 connection failed");

    // configure Arduino LED for
    pinMode(LED_PIN, OUTPUT);


void loop() {
    // read raw accel/gyro
if ( Serial.available() ){

     serial_read=Serial.read();
}

//   if(serial_read=='*'){
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

int ax_map=map(ax,-17000,17000,-100,100);
//scale it to use it with the car speed
  if(ax_map>-20&&ax_map<20)
  ax_map=0;
  last_ay=map(ay,-17000,17000,-100,100);

 if(last_ay>-30&&last_ay<30)
  last_ay=0;
Serial.print(ax_map);
Serial.print(",");
Serial.println(last_ay); //scale it for the car rotation
```


COM6 (Ard
0,0
0,0
0,0
0,0
0,0
0,0
0,0
0,0
-22,0
-24,0
-23,0
-24,0
-22,0
-21,0
-21,0
-21,0
-23,0
-22,0
-23,0
-22,0
-23,0
-26,0

Keypad code:

To change camera view.

Keypad interfacing code:

```
#include <Keypad.h>

const byte numRows= 4; //number of rows on the keypad
const byte numCols= 4; //number of columns on the keypad
char keymap[numRows][numCols]=
{
{'1', '2', '3', 'A'},
{'4', '5', '6', 'B'},
{'7', '8', '9', 'C'},
{'*', '0', '#', 'D'}
};

//Code that shows the the keypad connections to the arduino terminals
byte rowPins[numRows] = {9,8,7,6}; //Rows 0 to 3
byte colPins[numCols]= {5,4,3,2}; //Columns 0 to 3

//initializes an instance of the Keypad class
Keypad myKeypad= Keypad(makeKeymap(keymap), rowPins, colPins, numRows, numCols);

void setup()
{
  Serial.begin(9600);
}
void loop()
{
  char keypressed = myKeypad.getKey();
 if(keypressed)
      Serial.println(keypressed);

}
```

Wi-Fi module:

To send game data through internet.

Code for interfacing Wi-Fi:

```
#include<SoftwareSerial.h>
SoftwareSerial client(2,3); //RX, TX
String webpage="";
int i=0,k=0;
String readString;
int x=0;
boolean No_IP=false;
String IP="";
char temp1='0';
String name="<p>Circuit Digest</p>";   //22
String dat="<p>Data Received Successfully....</p>";     //21

void check4IP(int t1)
{
  int t2=millis();
  while(t2+t1>millis())
  {
    while(client.available()>0)
    {
      if(client.find("WIFI GOT IP"))
      {
        No_IP=true;
      }
    }
  }
}
```

```
void get_ip()
{
  IP="";
  char ch=0;
  while(1)
  {
    client.println("AT+CIFSR");
    while(client.available()>0)
    {
      if(client.find("STAIP,"))
      {
        delay(1000);
        Serial.print("IP Address:");
        while(client.available()>0)
        {
          ch=client.read();
          if(ch=='+')
          break;
          IP+=ch;
        }
      }
      if(ch=='+')
      break;
    }
    if(ch=='+')
    break;
    delay(1000);
```

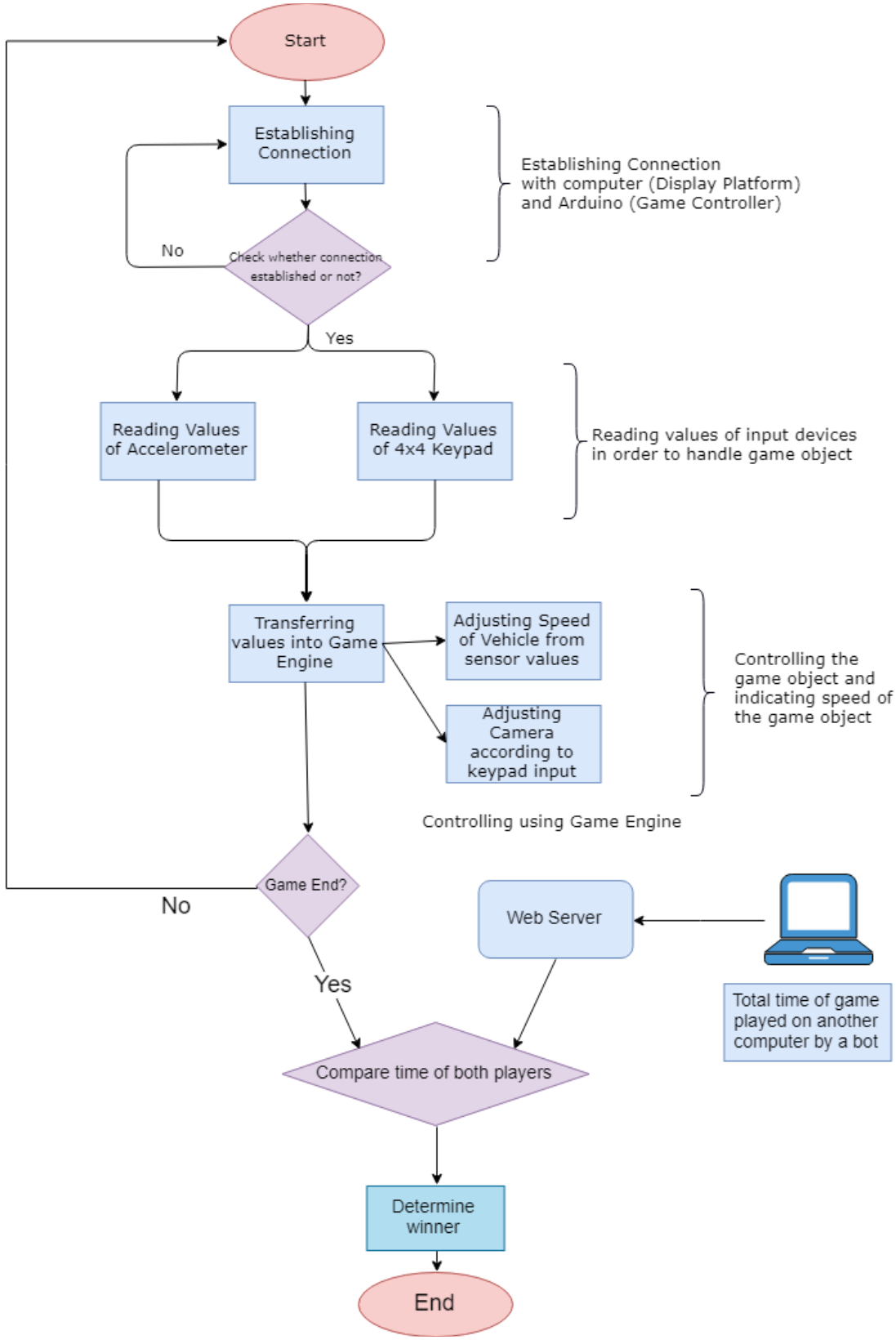
```
  Serial.print(IP);
  Serial.print("Port:");
  Serial.println(80);
}
void connect_wifi(String cmd, int t)
{
  int temp=0,i=0;
  while(1)
  {
    Serial.println(cmd);
    client.println(cmd);
    while(client.available())
    {
      if(client.find("OK"))
      i=8;
    }
    delay(t);
    if(i>5)
    break;
    i++;
  }
  if(i==8)
  Serial.println("OK");
  else
  Serial.println("Error");
}
```

```
void wifi_init()
{
       connect_wifi("AT",100);
       connect_wifi("AT+CWMODE=3",100);
       connect_wifi("AT+CWQAP",100);
       connect_wifi("AT+RST",5000);
       check4IP(5000);
       if(!No_IP)
       {
         Serial.println("Connecting Wifi....");
         connect_wifi("AT+CWJAP=\"Connectify-\",\"dhruti1997\"",7000);
      // connect_wifi("AT+CWJAP=\"vpn address\",\"wireless network\"",7000);
       }
       else
         {
         }
       Serial.println("Wifi Connected");
       //get_ip();
       connect_wifi("AT+CIPMUX=1",100);
       connect_wifi("AT+CIPSERVER=1,80",100);
}
void sendwebdata(String webPage)
{
    int ii=0;
     while(1)
     {
      unsigned int l=webPage.length();
```

```
      Serial.print("AT+CIPSEND=0,");
      client.print("AT+CIPSEND=0,");
      Serial.println(l+2);
      client.println(l+2);
      delay(100);
      Serial.println(webPage);
      client.println(webPage);
      while(client.available())
      {
        //Serial.print(Serial.read());
        if(client.find("OK"))
        {
          ii=11;
          break;
        }
      }
      if(ii==11)
      break;
      delay(100);
     }
}
void setup()
{
   Serial.begin(115200);
   client.begin(115200);
   wifi_init();
   Serial.println("System Ready..");
```

```
void loop()
{
  k=0;
  Serial.println("Please Refresh your Page");
  while(k<1000)
  {
    k++;
   while(client.available())
   {
    if(client.find("0,CONNECT"))
    {
      Serial.println("Start Printing");
      Send();
      Serial.println("Done Printing");
      delay(1000);
    }
  }
  delay(1);
 }
}
void Send()
{
      webpage = "<h1>Welcome</h1><body bgcolor=f0f0f0>";
      sendwebdata(webpage);

     client.println("AT+CIPCLOSE=0");
}
```

5. Flow Chart of the Program



6. Complete Code

```
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050.h"

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif
#include <Keypad.h>

const byte numRows= 4; //number of rows on the keypad
const byte numCols= 4; //number of columns on the keypad
char keymap[numRows][numCols]=
{
{'1', '2', '3', 'A'},
{'4', '5', '6', 'B'},
{'7', '8', '9', 'C'},
{'*', '0', '#', 'D'}
};

//Code that shows the the keypad connections to the arduino terminals
byte rowPins[numRows] = {9,8,7,6}; //Rows 0 to 3
byte colPins[numCols]= {5,4,3,2}; //Columns 0 to 3

//initializes an instance of the Keypad class
Keypad myKeypad= Keypad(makeKeymap(keymap), rowPins, colPins, numRows, numCols);

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 accelgyro;
```

```
int16_t ax, ay, az;
int16_t gx, gy, gz;

int last_ay=0;
// uncomment "OUTPUT_READABLE_ACCELGYRO" if you want to see a tab-separated
// list of the accel X/Y/Z and then gyro X/Y/Z values in decimal. Easy to read,
// not so easy to parse, and slow(er) over UART.
#define OUTPUT_READABLE_ACCELGYRO

//#define DEBUG

#ifdef DEBUG
 #define DEBUG_PRINT(x)  Serial.print (x)
 #define DEBUG_PRINTLN(x)  Serial.println (x)
#else
 #define DEBUG_PRINT(x)
 #define DEBUG_PRINTLN(x)
#endif

#define USE_ACCEL 3
#define USE_GYRO 3


#define LED_PIN 13
bool blinkState = false;

uint16_t lastReport;

const int numReadings = 25;

#if defined(USE_ACCEL) && defined(USE_GYRO)
const int numAxis = USE_ACCEL + USE_GYRO;
```

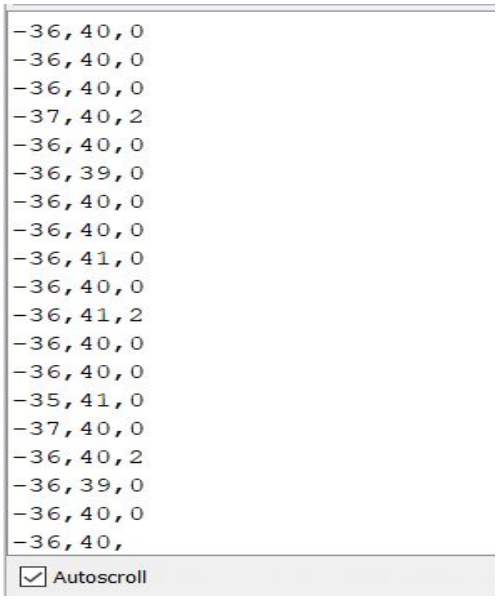
```
const int numAxis = USE_ACCEL + USE_GYRO;
const int AX = 0;
const int AY = 1;
const int AZ = 2;
const int GX = 3;
const int GY = 4;
const int GZ = 5;
#elif defined(USE_ACCEL)
const int numAxis = USE_ACCEL;
const int AX = 0;
const int AY = 1;
const int AZ = 2;
#elif defined(USE_GYRO)
const int numAxis = USE_GYRO;
const int GX = 0;
const int GY = 1;
const int GZ = 2;
#endif

int32_t readings[numAxis][numReadings];  // the reading history
int32_t readIndex[numAxis];              // the index of the current reading
int32_t total[numAxis];                  // the running total
int32_t average[numAxis];                // the average
char serial_read;

void setup() {
  lastReport = millis();
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
```

```
    Wire.begin();
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
#endif
Serial.begin(9600);
DEBUG_PRINTLN("Initializing I2C devices...");
accelgyro.initialize();

// supply your own gyro offsets here, scaled for min sensitivity
accelgyro.setXGyroOffset(26);
accelgyro.setYGyroOffset(-7);
accelgyro.setZGyroOffset(38);
accelgyro.setXAccelOffset(-814);
accelgyro.setYAccelOffset(-63);
accelgyro.setZAccelOffset(1665); // 1688 factory default for my test chip

// verify connection
DEBUG_PRINTLN("Testing device connections...");
DEBUG_PRINTLN(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050 connection failed");

// configure Arduino LED for
pinMode(LED_PIN, OUTPUT);


// zero-fill all the arrays:
for (int axis = 0; axis < numAxis; axis++) {
    readIndex[axis] = 0;
    total[axis] = 0;
    average[axis] = 0;
    for (int i = 0; i<numReadings; i++){
        readings[axis][i] = 0;
    }
```

```
void loop() {
    // read raw accel/gyro measurements from device
if ( Serial.available() ){
      serial_read=Serial.read();
}
accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
int ax_map=map(ax,-17000,17000,-100,100);
//scale it to use it with the car speed
  if(ax_map>-20&&ax_map<20)
  ax_map=0;

Serial.print(ax_map);
Serial.print(",");
last_ay=map(ay,-17000,17000,-100,100);
 if(last_ay>-30&&last_ay<30)
  last_ay=0;
  Serial.print(last_ay); //scale it for the car rotation
  char keypressed = myKeypad.getKey();
  Serial.print(",");
 if(keypressed)
      Serial.println(keypressed);
 else
      Serial.println("0");
delay(50);
//    #endif

    // blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
    serial_read=0;
    int speed_map=map(ax_map,-100,100,0,200);
      lcd.print(speed_map);
```

```
void smooth(int axis, int32_t val) {
    // pop and subtract the last reading:
    total[axis] -= readings[axis][readIndex[axis]];
    total[axis] += val;

    // add value to running total
    readings[axis][readIndex[axis]] = val;
    readIndex[axis]++;

    if(readIndex[axis] >= numReadings)
        readIndex[axis] = 0;

    // calculate the average:
    average[axis] = total[axis] / numReadings;
}
```

- **Output: Mapping values**
  Accelerometer X, Accelerometer Y
  Accelerometer X will be input as speed and Accelerometer Y will input as rotation for the gameobject car in Unity.
  Keypad 4x4: To change camera view through keypad 4x4: code to interface the keypad with arduino:



7. Timeline of the Project

Work	6/8/2018	10/8/2018	17/8/2018	24/8/2018	30/8/2018	7/9/2018
Project Formation	✔					
Test scene design and scaling values		✔				
Interfacing Wifi module with Arduino			✔			
Interfacing accelerometer with Game Engine			✔			
Implementing manual gear system				✔		
Designing the final game graphics					✔	
Checking for bugs and errors					✔	
Final report submission						✔

8. References

[1] For MPU6050 sensor: https://playground.arduino.cc/Main/MPU-6050

[2] For Keypad: https://playground.arduino.cc/Code/Keypad

[3] For Game development in UNITY game engine: https://unity3d.com/

[4] For the project idea: https://electronicsforu.com/electronics-projects/hardware-diy/game-controller-using-arduino-diy

[5] For sensor details: http://www.electronicwings.com/sensors-modules

9. Appendix [for datasheets of selected components]

- **Arduino UNO**

OVERVIEW | TECH SPECS | DOCUMENTATION

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 g

- **MPU6050**
  - **Gyroscope Specifications**

6.1 Gyroscope Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
GYROSCOPE SENSITIVITY						
Full-Scale Range	FS_SEL=0		±250		°/s	
	FS_SEL=1		±500		°/s	
	FS_SEL=2		±1000		°/s	
	FS_SEL=3		±2000		°/s	
Gyroscope ADC Word Length			16		bits	
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)	
	FS_SEL=1		65.5		LSB/(°/s)	
	FS_SEL=2		32.8		LSB/(°/s)	
	FS_SEL=3		16.4		LSB/(°/s)	
Sensitivity Scale Factor Tolerance	25°C	-3		+3	%	
Sensitivity Scale Factor Variation Over Temperature			±2		%	
Nonlinearity	Best fit straight line; 25°C		0.2		%	
Cross-Axis Sensitivity			±2		%	
GYROSCOPE ZERO-RATE OUTPUT (ZRO)						
Initial ZRO Tolerance	25°C		±20		°/s	
ZRO Variation Over Temperature	-40°C to +85°C		±20		°/s	
Power-Supply Sensitivity (1-10Hz)	Sine wave, 100mVpp; VDD=2.5V		0.2		°/s	
Power-Supply Sensitivity (10 - 250Hz)	Sine wave, 100mVpp; VDD=2.5V		0.2		°/s	
Power-Supply Sensitivity (250Hz - 100kHz)	Sine wave, 100mVpp; VDD=2.5V		4		°/s	
Linear Acceleration Sensitivity	Static		0.1		°/s/*g*	
SELF-TEST RESPONSE						
Relative	Change from factory trim	-14		14	%	1
GYROSCOPE NOISE PERFORMANCE	**FS_SEL=0**					
Total RMS Noise	DLPFCFG=2 (100Hz)		0.05		°/s-rms	
Low-frequency RMS noise	Bandwidth 1Hz to10Hz		0.033		°/s-rms	
Rate Noise Spectral Density	At 10Hz		0.005		°/s/ √Hz	
GYROSCOPE MECHANICAL FREQUENCIES						
X-Axis		30	33	36	kHz	
Y-Axis		27	30	33	kHz	
Z-Axis		24	27	30	kHz	
LOW PASS FILTER RESPONSE						
	Programmable Range	5		256	Hz	
OUTPUT DATA RATE						
	Programmable	4		8,000	Hz	
GYROSCOPE START-UP TIME	**DLPFCFG=0**					
ZRO Settling (from power-on)	to ±1°/s of Final		30		ms	

○ Accelerometer Specifications

6.2 Accelerometer Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY						
Full-Scale Range	AFS_SEL=0		±2		*g*	
	AFS_SEL=1		±4		*g*	
	AFS_SEL=2		±8		*g*	
	AFS_SEL=3		±16		*g*	
ADC Word Length	Output in two's complement format		16		bits	
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/*g*	
	AFS_SEL=1		8,192		LSB/*g*	
	AFS_SEL=2		4,096		LSB/*g*	
	AFS_SEL=3		2,048		LSB/*g*	
Initial Calibration Tolerance			±3		%	
Sensitivity Change vs. Temperature	AFS_SEL=0, -40°C to +85°C		±0.02		%/°C	
Nonlinearity	Best Fit Straight Line		0.5		%	
Cross-Axis Sensitivity			±2		%	
ZERO-G OUTPUT						
Initial Calibration Tolerance	X and Y axes		±50		*mg*	1
	Z axis		±80		*mg*	
Zero-G Level Change vs. Temperature	X and Y axes, 0°C to +70°C		±35			
	Z axis, 0°C to +70°C		±60		*mg*	
SELF TEST RESPONSE						
Relative	Change from factory trim	-14		14	%	2
NOISE PERFORMANCE						
Power Spectral Density	@10Hz, AFS_SEL=0 & ODR=1kHz		400		µg/√Hz	
LOW PASS FILTER RESPONSE						
	Programmable Range	5		260	Hz	
OUTPUT DATA RATE						
	Programmable Range	4		1,000	Hz	
INTELLIGENCE FUNCTION INCREMENT			32		mg/LSB	

- **LCD**
  - **Pin Specifications**

Pin no.	Symbol	External connection	Function
1	V_{SS}	Power supply	Signal ground for LCM
2	V_{DD}		Power supply for logic for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7~10	DB0~DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11~14	DB4~DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU
15	LED+	LED BKL power supply	Power supply for BKL
16	LED-		Power supply for BKL

  - **Electrical Specifications**

DC characteristics

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Supply voltage for LCD	V_{DD}-V_0	Ta =25℃	-	3.0	-	V
Input voltage	V_{DD}		3.1	3.3	3.5	
Supply current	I_{DD}	Ta=25℃, V_{DD}=3.3V	-	1.5	2.5	mA
Input leakage current	I_{LKG}		-	-	1.0	uA
"H" level input voltage	V_{IH}		2.2	-	V_{DD}	V
"L" level input voltage	V_{IL}	Twice initial value or less	0	-	0.6	
"H" level output voltage	V_{OH}	LOH=-0.25mA	2.4	-	-	
"L" level output voltage	V_{OL}	LOH=1.6mA	-	-	0.4	
Backlight supply voltage	V_F		-	3.0		
Backlight supply current	I_{LED}	V_{LED}=3.3 V R=25Ω			16	mA

- **Keypad 4x4**

Features

- Ultra-thin design
- Adhesive backing
- Excellent price/performance ratio
- Easy interface to any microcontroller
- Example programs provided for the BASIC Stamp 2 and Propeller P8X32A microcontrollers

Key Specifications

- Maximum Rating: 24 VDC, 30 mA
- Interface: 8-pin access to 4x4 matrix
- Operating temperature: 32 to 122 °F (0 to 50°C)
- Dimensions:
  Keypad, 2.7 x 3.0 in (6.9 x 7.6 cm)
  Cable: 0.78 x 3.5 in (2.0 x 8.8 cm)

Application Ideas

- Security systems
- Menu selection
- Data entry for embedded systems

- **Wi-Fi Module**

Categories	Items	Values
WiFi Paramters	Certificates	FCC/CE/TELEC/SRRC
	WiFi Protocles	802.11 b/g/n
	Frequency Range	2.4G-2.5G (2400M-2483.5M)
	Tx Power	802.11 b: +20 dBm
		802.11 g: +17 dBm
		802.11 n: +14 dBm
	Rx Sensitivity	802.11 b: -91 dbm (11 Mbps)
		802.11 g: -75 dbm (54 Mbps)
		802.11 n: -72 dbm (MCS7)
	Types of Antenna	PCB Trace, External, IPEX Connector, Ceramic Chip
Hardware Paramaters	Peripheral Bus	UART/SDIO/SPI/I2C/I2S/IR Remote Control
		GPIO/PWM
	Operating Voltage	3.0~3.6V
	Operating Current	Average value: 80mA
	Operating Temperature Range	-40°~125°
	Ambient Temperature Range	Normal temperature
	Package Size	5x5mm
	External Interface	N/A
Software Parameters	WiFi mode	station/softAP/SoftAP+station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network)
	Ssoftware Development	Supports Cloud Server Development / SDK for custom firmware development
	Network Protocols	IPv4, TCP/UDP/HTTP/FTP