Motion Game Controller

IoT project report

Jintao Lai(jl3837@cornell.edu)
Shwetha Anand(sa2249@cornell.edu)

12.12.2018

INTRODUCTION:

In the recent days, it is observed that electronic devices such as tablets and smartphones have become integral part of everyone's life, especially that of children. These devices are making children sedentary and are decreasing engagement in physical activities. In order to stimulate physical activities in children, in this project we have developed (multiplayer) motion game. An ESP32 microcontroller along with a MPU6050 motion sensor is attached to each players' hands to sense the motion and connect to the game running on a personal computer.

INVENTORY:

Components	Quantity	Price
ESP32 Microcontroller	2	NIL
MPU6050	2	$5*2=$10
Lithium battery	2	NIL
LEDs	2	NIL
Buzzer	2	NIL

PIN ASSIGNMENT

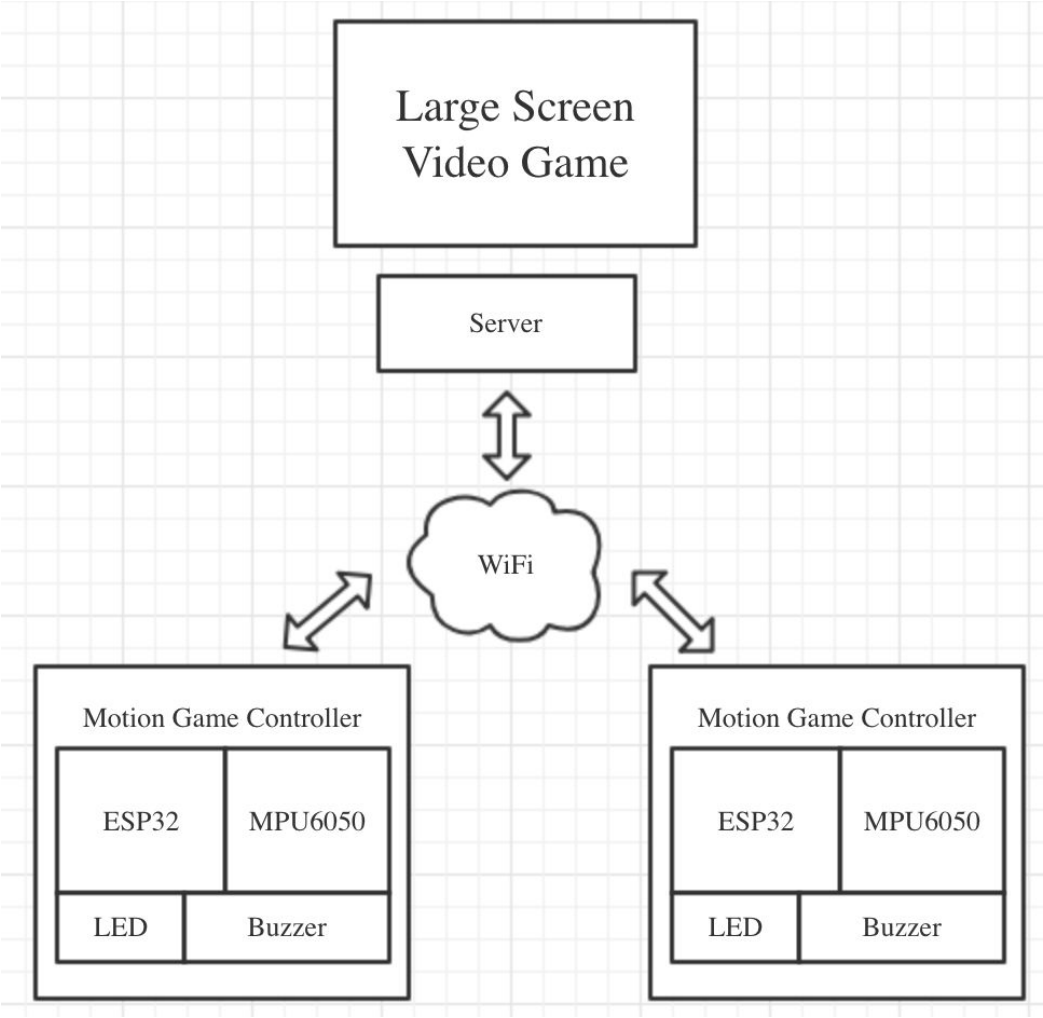
ESP32 Pin number	**Connection**
Pin 2	SDA of MPU6050
Pin 4	SCL of MPU6050
Pin 12	LED
Pin 13	Buzzer

HIGH LEVEL DESCRIPTION:

MPU6050 is a motion sensor which detects change in angle along 3 axes, roll, yaw and pitch. This sensor when held in person's hand and moved, the change in angle from the

initial position can be measured. The direction of the movement can be sent to a microcontroller, here ESP32, and can be translated to keyboard arrows up, down, left and right which are the most often used keys for several games, using appropriate thresholds. In this project, we are developing motion game controller with the above mentioned sensor and microcontroller which requires movement of the players' hands to input controls to a game.

MODEL ARCHITECTURE:



There are two main components:

1. TCP server:

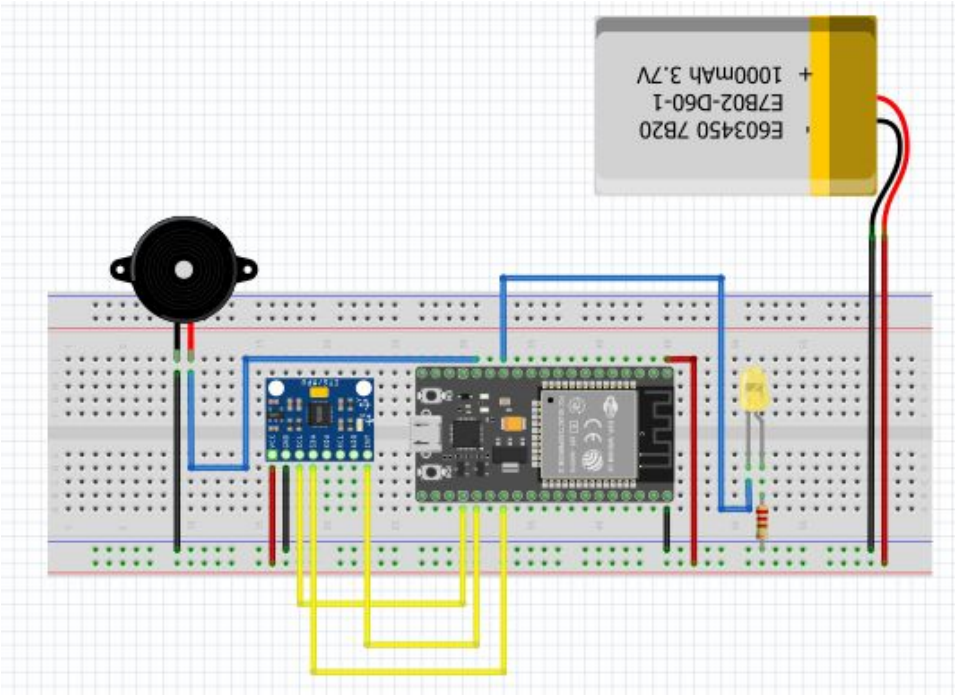
   The server creates a connection with the nodes upon request. a control panel using WXpython GUI was created. This control panel is used for 2 purposes. One, for

opening a port to start listening to the devices and second, to choose from different games to play.

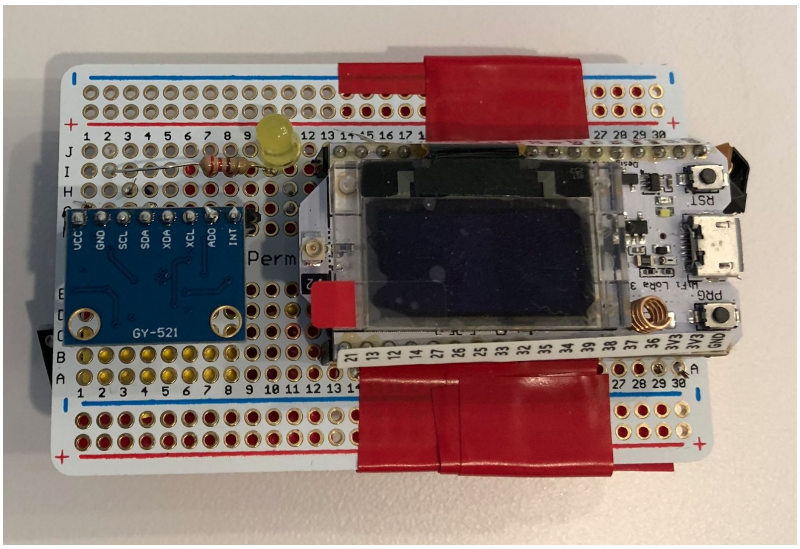
2. The hand-held device:

   Each player has his own hand held device (node). Each node consists of a ESP32 microcontroller, MPU 6050 motion sensor, a battery,a buzzer and a LED. These nodes communicate with the server over WiFi.

SCHEMATIC DIAGRAM:



The controller:



Control panel:

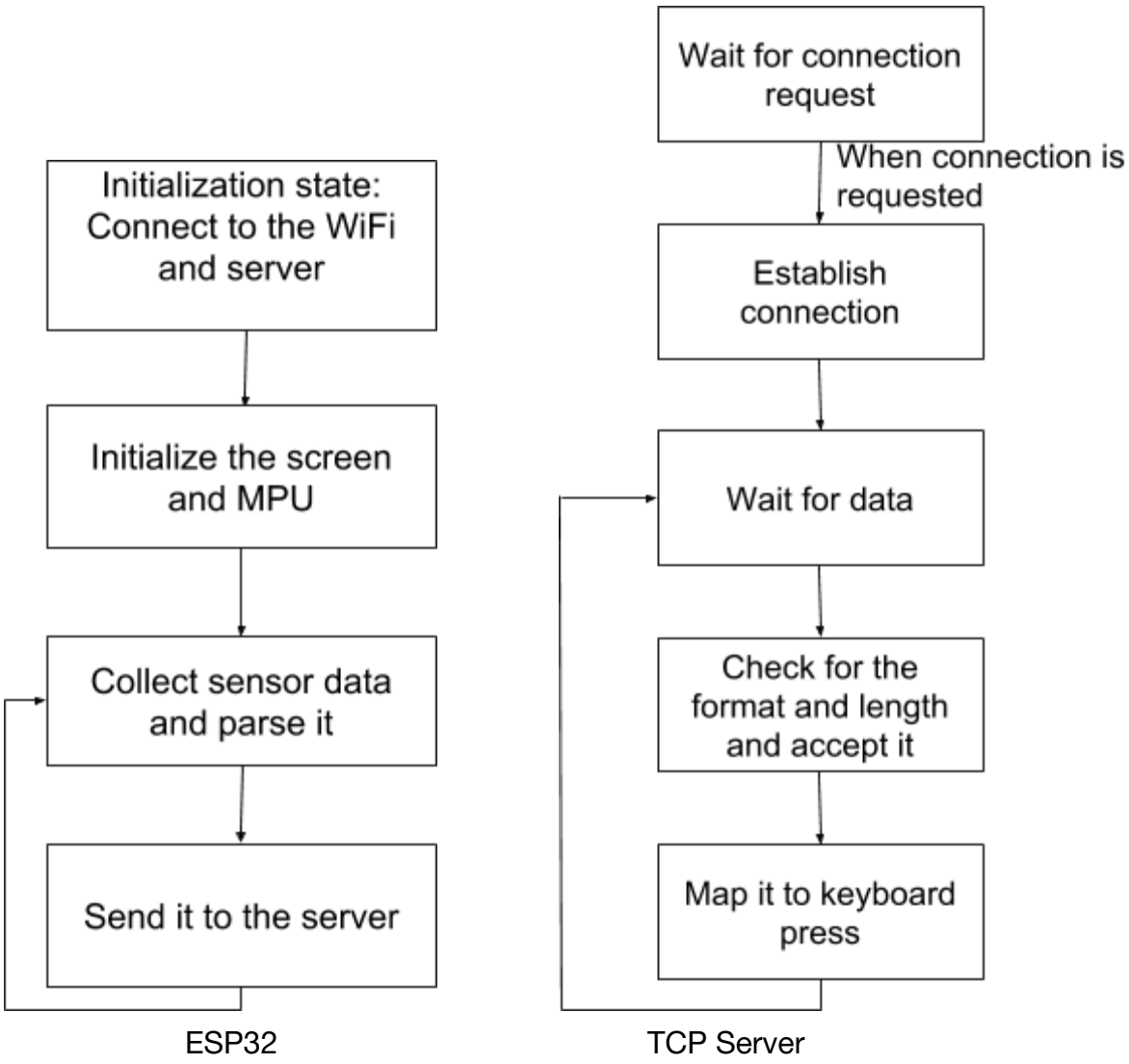
DETAILED DESCRIPTION:

The ESP32 first initializes the screen and the motion sensor. Then, it tries to connect to the Wifi and the server. Once the connection with the server is established, LED gets illuminated and stays illuminated indicating a stable connection. The players can choose from one of the two games from the control panel: 1. Minion rush 2. Bumper car mania. Minion rush is a single player game and Bumper car mania is a multi player game. When a game is chosen and started, buzzer buzzes indicating the game start.

The ESP32 starts sending the player's movement information to the server which is captured by the sensor. The sensor detects the change in the angle along the roll and pitch axes after sensor fusion (complementary filter for sensor). This gets passed to the microcontroller. The microcontroller parses this value using set thresholds for each direction into keyboard key press and sends it to the server. The server creates a separate thread for this communication.

For security purposes, the server receives data from the ESP32 and accepts only if it is in a specific format and a fixed length. Then it maps the inputs to keyboard press for the game.

STATE DIAGRAM:



NETWORK SECURITY:

In order to prevent the TCP server from reading unrelated data, the information sent by the ESP32 is made of a specific length and of a fixed format. The server accepts the data and further processes it only if the data is of this format and length.

CHALLENGES:

Rate of sending packets:
The rate at which packets were sent had to be controlled in order to prevent heavy network traffic, which would incur delays in the communication. Thus the packets were sent at the rate of 20 Hz using a timer, which is enough to have the game running smoothly and also not to increase the network traffic.

Debounce:
Since the packets were received at 20 Hz, debouncing of the inputs had to be done to avoid unintended and too many moves in the game.

Sensor inputs:
The sensor inputs had to be taken continuously because sensor fusion samples at the rate of 200Hz.

CODE EXPLANATION:

1. Connecting to the WiFi and TCP server: wifi() function establishes the wifi connection and displays " Connecting to the SSID" on the ESP32's screen while it is trying to make the connection and once it is connected, it displays "Connected to Wifi". Then it tries to connect to the TCP server and once the connection is stable, "Connected" is printed on the screen. The LED gets illuminated.
2. Buzzer: playTone() plays the buzzer when the server indicates the connection of the node to the game and has the required tone represented in hexadecimal.
3. Stable TCP server connection: check_disconnect() constantly checks for a stable connection.
4. Game data: send_location() initializes the data that has to be sent to the server and sets a constant sized memory block for this purpose. It sends the keyboard press data i.e., if the arrows are being pressed or not in the format that the TCP server accepts.
5. setup(): setup() sets the buzzer pin and LED to output, initializes the screen, initializes the sensor, the interrupt to input sensor data and the thresholds for detecting angle change in each direction.

6. Main loop: The main loop constantly awaits for the sensor data and checks for buffer overflow. Reads the data when interrupt is set by the sensor and compares with the various thresholds set to determine the angle and direction and then sends it to the server.

CONCLUSION:

A motion game controller that takes in player's hand movements as input was built using ESP32 and motion sensor MPU 6050. The player has to engage in physical activity to play the game. Thus playing the same games in the smart devices has been made more physically engaging.

APPENDIX

ESP32:

```
#include <Ticker.h>

#include <esp_wifi.h>
#include "SSD1306.h"
#include "WiFi.h"
#include <string.h>


#define host_ip // your server ip add
#define port 20000
#define Node 1

#define  c3    7634
#define  d3    6803
#define  e3    6061
#define  f3    5714
#define  g3    5102
#define  a3    4545
#define  b3    4049
#define  c4    3816    // 261 Hz
#define  d4    3401    // 294 Hz
#define  e4    3030    // 329 Hz
#define  f4    2865    // 349 Hz
#define  g4    2551    // 392 Hz
#define  a4    2272    // 440 Hz
#define  a4s   2146
#define  b4    2028    // 493 Hz
#define  c5    1912    // 523 Hz
#define  d5    1706
#define  d5s   1608
#define  e5    1517
#define  f5    1433
#define  g5    1276
#define  a5    1136
#define  a5s   1073
#define  b5    1012
```

```
#define  c6    955

#define  R     0
int speakerOut = 13;



// star wars theme
int melody[] = {  f4,  f4, f4,  f5};//
int beats[]  = {  21,  21, 21,  30 };


int MAX_COUNT = sizeof(melody) / 2; // Melody length, for looping.

// Set overall tempo
long tempo = 10000;
// Set length of my_pp between notes
long my_pp = 1000;
// Loop variable to increase Rest length
int rest_count = 50; //
// Initialize core variables
int toneM = 0;
int beat = 0;
long duration  = 0;

// PLAY TONE  ==============================================
// Pulse the speaker to play a tone for a particular duration
void playTone() {
  long elapsed_time = 0;
  if (toneM > 0) { // if this isn't a Rest beat, while the tone has
    //  played less long than 'duration', pulse speaker HIGH and LOW
    while (elapsed_time < duration) {

      digitalWrite(speakerOut,HIGH);
      delayMicroseconds(toneM / 2);

      // DOWN
      digitalWrite(speakerOut, LOW);
      delayMicroseconds(toneM / 2);

      // Keep track of how long we pulsed
      elapsed_time += (toneM);
    }
```

```
  }
  else { // Rest beat; loop times delay
    for (int j = 0; j < rest_count; j++) { // See NOTE on rest_count
      delayMicroseconds(duration);
    }
  }
}


#define WIFI // your wifi
#define PWD // your wifi password
const char* ssid     = WIFI;
const char* password = PWD;

// This is the IP address of the system we will connect to for testing purposes.
const char* host = host_ip;
uint16_t    host_port = port;
WiFiClient client;

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

Ticker flipper;

#define SDA   4
#define SCL   15
#define RST   16
#define DISPLAY_HEIGHT 128
#define DISPLAY_WIDTH  64
#define V2    1
#define Vext  21

#define len  127
#define wid  63

SSD1306  display(0x3c, SDA, SCL, RST);

#include "MPU6050_6Axis_MotionApps20.h"

MPU6050 mpu;

#define strength 1000
```

```
#define strength_lr 500



#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 12 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
bool wifiReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}

void oled_println(const char* str) {
  display.clear();
  display.println(str);
  display.drawLogBuffer(0, 0);
  display.display();
}


void wifi(){
```

```
  // Setup Wifi.
  byte mac_address[7];
  WiFi.begin(ssid, password);

// Setup OLED display.
  pinMode(Vext, OUTPUT);
  digitalWrite(Vext, LOW);
  delay(50);
  display.init();
  //display.flipScreenVertically();  // Uncomment as needed.
  display.setContrast(255);
  display.setLogBuffer(5, 30);
  display.clear();


  oled_println("Connecting to Wifi SSID: ");
  oled_println(ssid);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    oled_println(".");
  }

  oled_println("\nConnected. IP address: ");
  Serial.println(WiFi.localIP());
  oled_println("Connected to Wifi.");

  delay(1000);

   oled_println("Connecting to ");
  oled_println(host);

  // Use WiFiClient class to create a TCP connection.
  while(!client.connect(host, host_port)) {
    Serial.println("Connect failed. Retrying...");
      oled_println("Connect failed. Retrying...");

    delay(3000);
  }
  Serial.println("Connected.");
  oled_println("Connected.");

  while(!wifiReady){
```

```
    int count = 0;
    char reply_array[3];
    while(count<3)
    {
      while (client.connected() && client.available()) {
      uint8_t val = (uint8_t) client.read();
      reply_array[count] = val;
      count += 1;
      Serial.println(val);

      }
    }
    if( reply_array[0] == 0x11 && reply_array[1] == 0x22 && reply_array[2] == 0x33){
      wifiReady = true;
    }

  }

    // Set up a counter to pull from melody[] and beats[]
  for (int i=0; i<MAX_COUNT; i++) {
    toneM = melody[i];
    beat = beats[i];

    duration = beat * tempo; // Set up timing

    playTone();
    // A my_pp between notes...
    delayMicroseconds(my_pp);
    if (i==3){
      break;
    }
  }
  Serial.println("out");

  pinMode(speakerOut, INPUT);


  digitalWrite(Vext, HIGH);
}

bool check_disconnect(){
      // Make sure the TCP connection is still alive.
```

```
    if (client.connected() == 0) {
      Serial.println("\n\nDisconnected. Retrying in 3 seconds...");
      oled_println("Disconnected. Reconnecting...");
      delay(3000);
      return true;
    }
    return false;
}


bool send_flag = false;

#define SEND_SIZE 10
void send_location(bool up,bool down,bool left,bool right){
  if (send_flag){
      uint8_t sned_array[SEND_SIZE];
      memset( sned_array, 0, SEND_SIZE );

      char up_b = up ? 1 : 0;
      char down_b = down ? 1 : 0;
      char left_b = left ? 1 : 0;
      char right_b = right ? 1 : 0;

      sned_array[0] = 0xa0;
      sned_array[1] = 0xa1;
      sned_array[2] = up_b;
      sned_array[3] = down_b;
      sned_array[4] = left_b;
      sned_array[5] = right_b;
      sned_array[6] = Node;

        for (int i = 0; i < SEND_SIZE; i++) {
          client.write(sned_array[i]);
        }
      send_flag = false;
  }

}

void flip()
{
  send_flag = true;
}
```

```
// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================

void setup() {

  pinMode(speakerOut, OUTPUT);


    flipper.attach(0.05, flip);

    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin(SDA, SCL);
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(9600);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection
failed"));

    wifi();

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
```

```
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
        Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
        Serial.println(F(")..."));
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}



// ================================================================
// ===                    MAIN PROGRAM LOOP                     ===
// ================================================================


void loop() {
```

```
//Serial.println("loop");
// if programming failed, don't try to do anything
if (!dmpReady or !wifiReady) return;

// wait for MPU interrupt or extra packet(s) available
while (!mpuInterrupt && fifoCount < packetSize) {
    if (mpuInterrupt && fifoCount < packetSize) {
      // try to get out of the infinite loop
      fifoCount = mpu.getFIFOCount();
    }

}

// reset interrupt flag and get INT_STATUS byte
mpuInterrupt = false;
mpuIntStatus = mpu.getIntStatus();

// get current FIFO count
fifoCount = mpu.getFIFOCount();

// check for overflow (this should never happen unless our code is too inefficient)
if ((mpuIntStatus & _BV(MPU6050_INTERRUPT_FIFO_OFLOW_BIT)) || fifoCount >= 1024) {
    // reset so we can continue cleanly
    mpu.resetFIFO();
    fifoCount = mpu.getFIFOCount();
    Serial.println(F("FIFO overflow!"));

// otherwise, check for DMP data ready interrupt (this should happen frequently)
} else if (mpuIntStatus & _BV(MPU6050_INTERRUPT_DMP_INT_BIT)) {
    // wait for correct available data length, should be a VERY short wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);

    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an interrupt)
    fifoCount -= packetSize;

    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
```

```
mpu.dmpGetGravity(&gravity, &q);
mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

mpu.dmpGetAccel(&aa, fifoBuffer);
mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);

float roll = ypr[2]* 180/M_PI;
float pitch = ypr[1]* 180/M_PI;

float up_greater =  0;
float up_smaller =  125;

float down_greater =  -120 ;
float down_smaller =  -40;

float left_greater =  -120;
float left_smaller =   -30;

float right_greater =  60;
float right_smaller =  120;

bool up = roll > up_greater && roll < up_smaller ;
bool down = roll > down_greater && roll < down_smaller ;
bool left = pitch > left_greater && pitch < left_smaller ;
bool right = pitch > right_greater && pitch < right_smaller;


if(up)
{
  Serial.print("up") ;
}
if(down)
{
  Serial.print("down") ;
}

if(left)
{
  Serial.print("left") ;
}

if(right)
```

```
      {
        Serial.print("right") ;
      }

      send_location(up,down,left,right);

      // blink LED to indicate activity
      blinkState = !blinkState;
      digitalWrite(LED_PIN, blinkState);
   }
}
```

TCP Server:

```
from pymouse import PyMouse
from pykeyboard import PyKeyboard
from socket import socket, AF_INET, SOCK_STREAM

port = 20000
# -*- coding: utf-8 -*-

client_addr = []
client_socket = {}

import wx
from socketserver import ThreadingTCPServer

###########################################################################
## Class MotionGame
###########################################################################

class MotionGame(wx.Frame):

   def __init__(self, parent):
      wx.Frame.__init__(self, parent, id=wx.ID_ANY, title=wx.EmptyString, pos=wx.DefaultPosition,
                  size=wx.Size(500, 300), style=wx.DEFAULT_FRAME_STYLE | wx.TAB_TRAVERSAL)

      self.SetSizeHintsSz(wx.DefaultSize, wx.DefaultSize)

      bSizer11 = wx.BoxSizer(wx.VERTICAL)

      self.m_staticText1 = wx.StaticText(self, wx.ID_ANY, u"ECE 5413 Motion Game", wx.DefaultPosition,
wx.DefaultSize,
```

```
                                0)
        self.m_staticText1.Wrap(-1)
        bSizer11.Add(self.m_staticText1, 0, wx.ALL | wx.ALIGN_CENTER_HORIZONTAL, 5)

        self.m_button1 = wx.Button(self, wx.ID_ANY, u"Start Server", wx.DefaultPosition, wx.DefaultSize, 0)
        bSizer11.Add(self.m_button1, 0, wx.ALL | wx.ALIGN_CENTER_HORIZONTAL, 5)

        self.m_staticText2 = wx.StaticText(self, wx.ID_ANY, u"server is down", wx.DefaultPosition,
wx.DefaultSize, 0)
        self.m_staticText2.Wrap(-1)
        bSizer11.Add(self.m_staticText2, 0, wx.ALL | wx.ALIGN_CENTER_HORIZONTAL, 5)

        gbSizer1 = wx.GridBagSizer(0, 0)
        gbSizer1.SetFlexibleDirection(wx.BOTH)
        gbSizer1.SetNonFlexibleGrowMode(wx.FLEX_GROWMODE_SPECIFIED)

        self.m_staticText12 = wx.StaticText(self, wx.ID_ANY, u"Game 1", wx.Point(20, 20), wx.DefaultSize,
                                    wx.ALIGN_CENTRE)
        self.m_staticText12.Wrap(-1)
        gbSizer1.Add(self.m_staticText12, wx.GBPosition(0, 0), wx.GBSpan(1, 1),
                    wx.ALL | wx.ALIGN_CENTER_VERTICAL | wx.ALIGN_RIGHT, 5)

        self.m_button2 = wx.Button(self, wx.ID_ANY, u"Set Game 1", wx.DefaultPosition, wx.DefaultSize, 0)
        gbSizer1.Add(self.m_button2, wx.GBPosition(0, 1), wx.GBSpan(1, 1), wx.ALL, 5)

        self.m_staticText14 = wx.StaticText(self, wx.ID_ANY, u"Player 1", wx.DefaultPosition, wx.DefaultSize,
0)
        self.m_staticText14.Wrap(-1)
        gbSizer1.Add(self.m_staticText14, wx.GBPosition(0, 2), wx.GBSpan(1, 1), wx.ALL, 5)

        self.m_staticText4 = wx.StaticText(self, wx.ID_ANY, u"disconnected", wx.DefaultPosition,
wx.DefaultSize, 0)
        self.m_staticText4.Wrap(-1)
        gbSizer1.Add(self.m_staticText4, wx.GBPosition(0, 3), wx.GBSpan(1, 1), wx.ALL, 5)

        bSizer11.Add(gbSizer1, 1, wx.EXPAND, 5)

        gbSizer11 = wx.GridBagSizer(0, 0)
        gbSizer11.SetFlexibleDirection(wx.BOTH)
        gbSizer11.SetNonFlexibleGrowMode(wx.FLEX_GROWMODE_SPECIFIED)

        self.m_staticText121 = wx.StaticText(self, wx.ID_ANY, u"Game 2", wx.Point(20, 20), wx.DefaultSize,
                                    wx.ALIGN_CENTRE)
```

```
        self.m_staticText121.Wrap(-1)
        gbSizer11.Add(self.m_staticText121, wx.GBPosition(0, 0), wx.GBSpan(1, 1),
                      wx.ALL | wx.ALIGN_CENTER_VERTICAL | wx.ALIGN_RIGHT, 5)

        self.m_button3 = wx.Button(self, wx.ID_ANY, u"Set Game 2", wx.DefaultPosition, wx.DefaultSize, 0)
        gbSizer11.Add(self.m_button3, wx.GBPosition(0, 1), wx.GBSpan(1, 1), wx.ALL, 5)

        self.m_staticText141 = wx.StaticText(self, wx.ID_ANY, u"Player 1", wx.DefaultPosition,
wx.DefaultSize, 0)
        self.m_staticText141.Wrap(-1)
        gbSizer11.Add(self.m_staticText141, wx.GBPosition(0, 2), wx.GBSpan(1, 1), wx.ALL, 5)

        self.m_staticText5 = wx.StaticText(self, wx.ID_ANY, u"disconnected", wx.DefaultPosition,
wx.DefaultSize, 0)
        self.m_staticText5.Wrap(-1)
        gbSizer11.Add(self.m_staticText5, wx.GBPosition(0, 3), wx.GBSpan(1, 1), wx.ALL, 5)

        self.m_staticText40 = wx.StaticText(self, wx.ID_ANY, u"Player 2", wx.DefaultPosition, wx.DefaultSize,
0)
        self.m_staticText40.Wrap(-1)
        gbSizer11.Add(self.m_staticText40, wx.GBPosition(0, 4), wx.GBSpan(1, 1), wx.ALL, 5)

        self.m_staticText6 = wx.StaticText(self, wx.ID_ANY, u"disconnected", wx.DefaultPosition,
wx.DefaultSize, 0)
        self.m_staticText6.Wrap(-1)
        gbSizer11.Add(self.m_staticText6, wx.GBPosition(0, 5), wx.GBSpan(1, 1), wx.ALL, 5)



        bSizer11.Add(gbSizer11, 1, wx.EXPAND, 5)

        bSizer12 = wx.BoxSizer(wx.VERTICAL)

        self.m_staticText57 = wx.StaticText(self, wx.ID_ANY, u"Game 2 Link: ", wx.DefaultPosition,
wx.Size(50, -1), 0)
        self.m_staticText57.Wrap(-1)
        self.m_staticText57.SetMaxSize(wx.Size(100, -1))

        bSizer12.Add(self.m_staticText57, 1, wx.ALL | wx.EXPAND, 5)

        self.m_textCtrl12 = wx.TextCtrl(self, wx.ID_ANY, u"http://www.4399.com/flash/187228_1.htm",
wx.DefaultPosition,
                                        wx.DefaultSize, 0)
```

```
    bSizer12.Add(self.m_textCtrl12, 0, wx.ALL | wx.EXPAND, 5)

    bSizer11.Add(bSizer12, 1, wx.EXPAND, 5)

    self.SetSizer(bSizer11)
    self.Layout()

    self.Centre(wx.BOTH)

    # Connect Events
    self.m_button1.Bind(wx.EVT_BUTTON, self.start_server)
    self.m_button2.Bind(wx.EVT_BUTTON, self.set_game1)
    self.m_button3.Bind(wx.EVT_BUTTON, self.set_game2)

def __del__(self):
    pass

# Virtual event handlers, overide them in your derived class
def start_server(self, event):
    frame.m_staticText2.SetLabel("Server is Running !!! ")

    print("start server")
    timer = threading.Timer(timer_period, fun_timer)
    timer.start()
    server = ThreadingTCPServer(('', port), EchoHandler)

    server_thread = threading.Thread(target=server.serve_forever)
    # Exit the server thread when the main thread terminates
    server_thread.daemon = True
    server_thread.start()
    #sudo netstat -lntup|grep 20000
    #ps -ef|grep python
    #kill -9 51976


def set_game1(self, event):
    global mode
    global mode_1_flag
    global mode_2_flag
    mode_1_flag = True
    mode = 1
    print("Mode 1")
    for key,value in client_socket.items():
```

```
            value.sendall(bytes([0x11,0x22,0x33]))


    def set_game2(self, event):
        global mode
        global mode_1_flag
        global mode_2_flag
        mode_2_flag = True
        mode = 2
        print("Mode 2")
        for key,value in client_socket.items():
            try:
                value.sendall(bytes([0x11, 0x22, 0x33]))
            except IOError:
                pass
            else:
                pass


m = PyMouse()
k = PyKeyboard()


from socketserver import BaseRequestHandler, TCPServer

buffer_size = 10

key_flag = False
import threading

timer_period = 0.1
def fun_timer():
    global key_flag
    #print('Hello Timer!')
    key_flag = True
    global timer
    timer = threading.Timer(timer_period, fun_timer)
    timer.start()

previous_key = 0

mode = 1
frame =None
```

```
mode_1_flag= False
mode_2_flag= False

d = {}

# BaseRequestHandlerbase class，handle()
class EchoHandler(BaseRequestHandler):

    def setup(self):
        ip = self.client_address[0].strip()     # ip
        port = self.client_address[1]           # port
        print(ip+":"+str(port)+" is connect!")
        client_addr.append(self.client_address) #
        client_socket[self.client_address] = self.request # socket

    def finish(self):
        print("client is disconnect!")
        client_addr.remove(self.client_address)
        del client_socket[self.client_addr]

    def handle(self):
        global key_flag
        global previous_key
        global mode_1_flag
        global mode_2_flag
        print('Got connection from', self.client_address)

        print(type(self.request))

        # self.request is the TCP socket connected to the client
        count = 0
        msg = []



        while True:
            # 8192代表每次读取8192字节
            temp = self.request.recv(buffer_size)
            msg.extend(temp)

            while len(msg) >= 2 and (msg[0]!=0xa0 or msg[1]!=0xa1):
                msg.pop(0)
```

```
if len(msg)<buffer_size:
    continue

if not key_flag:
    continue

up  = msg[2]
down  = msg[3]
left  = msg[4]
right  = msg[5]
node = msg[6]

if node == 1:
    frame.m_staticText4.SetLabel("Connected !!! ")
    frame.m_staticText5.SetLabel("Connected !!! ")

if node == 2:
    frame.m_staticText6.SetLabel("Connected !!! ")

if mode == 1:

    key = 0
    if up and not left and not right:
        key =1
    if down and not left and not right:
        key =2
    if left:
        key =3
    if right:
        key =4


    if key != 0 and previous_key != key:
        print(key)
        if key == 1:
            k.press_key("up")
            print(" node 1 up")

        # else:
        #     k.release_key("up")

        if key == 2:
```

```
            k.press_key("down")
            print(" node 1 down")

        # else:
        #     k.release_key("down")

        if key == 3:
            k.press_key("left")
            print(" node 1 left")

        # else:
        #     k.release_key("left")

        if key == 4:
            k.press_key("right")
            print(" node 1 right")

        # else:
        #     k.release_key("right")

    previous_key = key


if mode == 2:

    if node == 1:

        if up == 1:
            k.press_key("up")
            print(" node 1 up")
        else:
            k.release_key("up")

        if down == 1:
            k.press_key("down")
            print(" node 1 down")
        else:
            k.release_key("down")

        if left == 1:
            k.press_key("left")
            print(" node 1 left")
        else:
```

```
                k.release_key("left")

            if right == 1:
                k.press_key("right")
                print(" node 1 right")
            else:
                k.release_key("right")

        if node == 2:

            if up == 1:
                k.press_key("w")
                print(" node 2 up")
            else:
                k.release_key("w")

            if down == 1:
                k.press_key("s")
                print(" node 2 down")
            else:
                k.release_key("s")

            if left == 1:
                k.press_key("a")
                print(" node 2 left")
            else:
                k.release_key("a")

            if right == 1:
                k.press_key("d")
                print(" node 2 right")
            else:
                k.release_key("d")



        msg = []
        #key_flag = False



if __name__ == '__main__':
```

```
app = wx.App()
frame = MotionGame(None)
frame.Show()

app.MainLoop()
```