PROJECT TITLE

SAFEWEAR: A SMART TRACKER FOR MONITORING LOVED ONES

A project report submitted in partial fulfilment of requirements to

CENTER FOR DEVELOPMENT OF ADVANCED COMPUTING HYDERABAD

For the award of the degree of

POST GRADUATE DIPLOMA

In

EMBEDDED SYSTEM AND DESIGN

Under the esteemed guidance of

SAI SHIVA REDDY GATLA

Submitted by

Vibhanshu Prajapati	(230950330042)
Shaik Shoiab	(230950330038)
Uma Bhati	(230950330041)
Surisetty Kusuma	(230950330040)
Shreeshail Patil	(230950330039)

INDEX

ABSTRACT

As for everyone their family is really important, so comes in picture their health. There are a lot of options to monitor health of family members such as Blood oximeter or Pulse-meter but they're helpful only if the person is present near them, so to avoid this and track health of family members wherever they're present Smart wearables were introduced but the catch in this is that they should always be connected either to mobile device or to the internet to share/transfer the health data. As long as the internet is present or the wearable is connected to mobile device the data is being given to the guardian of the family but as soon as both are not present the guardian will not be able to get the health data of the family members.

Keeping this in mind we've come up with a new smart wearable which is based on LoRa (Long-Range). LoRa sends data without the help of internet or WIFI. There'll be two modules of LoRa used one for the child/old person's Wearable and the other for the guardian so they could get the health data.
But…
For children and old people just health data is not enough therefore in addition to that we've also added a GPS module which would track the person within a range of 10-15 kms so just in case if the child or the old person suffering from Amnesia gets lost they could be tracked easily. We've also included a sensor which senses instant position change of the person which may be able to track if the person falls

All the data of health, the latitudes longitudes & alerts will be sent to the guardian via LoRa.

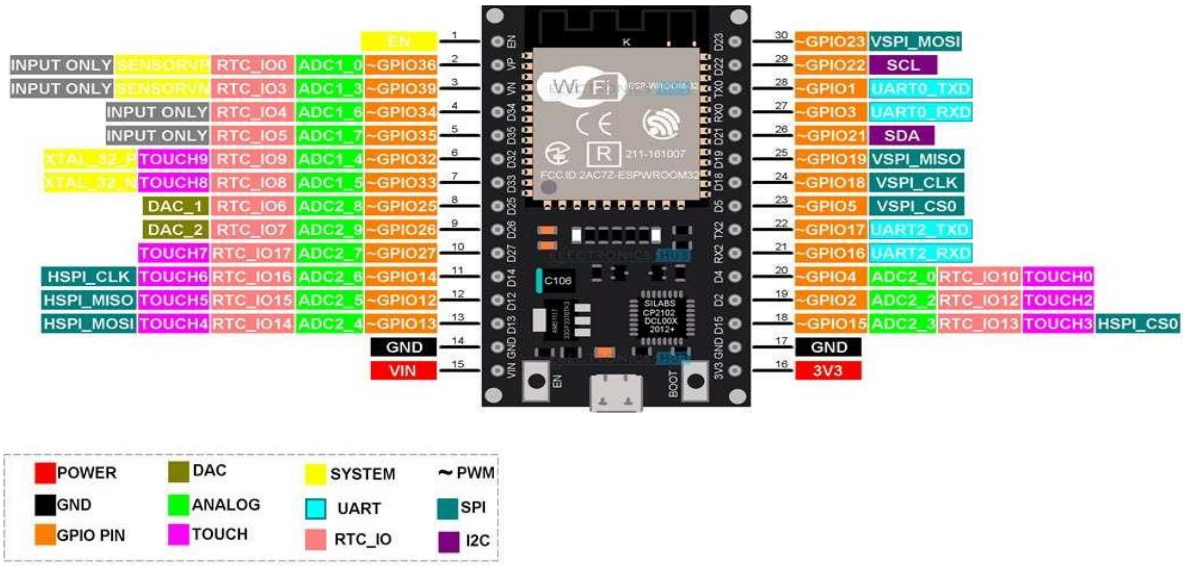
ESP32 PINOUT | ESP-WROOM-32 PINOUT

The 30-pin ESP32 Development Board will be used to demonstrate the ESP32 Pinout.

All ESP32 boards come with 4 MB of Flash Memory to store the programs. So, someof the GPIO Pins (6 to be specific) are connected to SPI Flash IC and those pins cannot be used as regular GPIO Pins

ESP32 PINOUT :

One popular ESP32 Development Board available today is the 30-pin version shownin the above image. It consists of ESP-WROOM-32 as the baseboard and additionally few pins and components to easily interact with ESP32.



As you can see from the image, each pin has more than one possible functionality and while using a pin for task, double check its alternative functions.

ESP32 PERIPHERALS :

Now that we have seen a little bit about ESP32 Pinout. Let us now focus on some of the important peripherals of ESP32 and their associated pins. ESP32 Microcontroller has:

- 34 Programmable GPIOs
- 15 12-bit ADC Channels
- 2 8-bit DAC Channels
- 16 PWM Channels
- 4 UART Interfaces
- 3 SPI Interfaces
- 2 I^2C Interfaces
- 2 I^2S Interfaces
- 10 Capacitive Touch Sensing GPIOs
- 16 RTC GPIOs

GPIO

The most used peripheral is the GPIO. ESP32 has 34 GPIO pins with each pin carrying out more than one function (only one will be active). You can configure a pin as either a GPIO or an ADC or an UART in the program.

ADC and DAC pins are predefined and you must use the manufacturer specified pins. But other functions like PWM, SPI, UART, I^2C etc. can be assigned to any GPIO pin through program.

RTC GPIO

ESP32 has 16 RTC GPIOs, which are part of the RTC Low-Power subsystem. These pins can be used to wake ESP32 from deep sleep as external wake-up source.

ADC

ESP32 has two 12-bit SAR Analog to Digital Converter Modules with 8-channels and 10-channels each. So, ADC1 and ADC2 blocks combined have 18 channels of 12-bit ADC. With 12-bit resolution, the output Digital values will be in the range of 0 – 4093.

DAC

ESP32 Microcontroller has two independent 8-bit Digital to Analog Converter channels to convert digital values to analog voltage signals. The DAC has internal resistor network and uses power supply as input reference voltage.

The following two GPIO Pins are associated with DAC functionalities.

- DAC1 — GPIO25
- DAC2 — GPIO26

CAPACITIVE TOUCH GPIOS

The ESP32 SoC has 10 capacitive-sensing GPIOs, which can detect variations incapacitance on a pin due to touching or approaching the GPIO Pin with a finger or stylus. These Touch GPIOs can be used in implementing capacitive touch pads, without any additional hardware.

SPI

The ESP32 Wi-Fi chip features three SPI blocks (SPI, HSPI and VSPI) in both master and slave modes. SPI is used to interface with Flash Memory. So, you have two SPIinterfaces.

I2C

There are two I^2C interfaces in ESP32 with complete flexibility on assigning pins i.e., SCL and SDA pins for both I^2C interfaces can be assigned in the program by the user.If you are using Arduino IDE, then the default I^2C pins are:

- SDA – GPIO21
- SCL – GPIO22

PWM

The PWM Controller in ESP32 have 16 independent PWM waveform channels with configurable frequency and duty cycle. The PWM waveform can be used to drive motors and LEDs. You can configure the PWM signal frequency, channel, GPIO pin and also the duty cycle.

GPIO PINS

Table which specifies the GPIO pins and their input / output capabilities.

GPIO Pin	Pin on ESP32	Information
0	–	Pulled HIGH. Connected to BOOT Button
1	TX0	Do not use while TXing
2	YES	Pulled LOW
3	RX0	Do not use while RXing

4		Pulled LOW
5	D5	Pulled HIGH
6	–	Connected to SPI Flash IC
7	–	Connected to SPI Flash IC
8	–	Connected to SPI Flash IC
9	–	Connected to SPI Flash IC
10	–	Connected to SPI Flash IC
11	–	Connected to SPI Flash IC
12	D12	Pulled LOW. Boot fails if pulled HIGH as it sets voltage of internal voltage regulator.
13	D13	
14	D14	
15	D15	Pulled HIGH
16	RX2	UART2 RX
17	TX2	UART2 TX

18	D18	
19	D19	
21	D21	I2C SDA
22	D22	I2C SCL
23	D23	
25	D25	
26	D26	
27	D27	
32	D32	
33	D33	
34	D34	Digital Input Only. No Digital Output.
35	D35	Digital Input Only. No Digital Output.
36	YES	Digital Input Only. No Digital Output.
39	YES	Digital Input Only. No Digital Output.

GPIOS CONNECTED TO SPI FLASH IC

If you look at the schematic of ESP-WROOM-32 Module, then you will see that GPIO6 to GPIO11 are connected to SPI Flash Memory IC. Even if these GPIO pins are accessible (which are not in 30-pin ESP32 Board), do not use them for any other purpose.

INPUT ONLY GPIO

There are 4 GPIO pins which can act as Digital Input only pins. They are GPIO34,GPIO35, GPIO36 and GPIO39.

INTERRUPTS

All GPIO pins are capable of interrupts.

BOOT STRAPPING PINS

ESP32 SoC has 5 boot strapping pins. They are:

- GPIO0 (HIGH during BOOT)
- GPIO2 (LOW during BOOT)
- GPIO5 (HIGH during BOOT)
- GPIO12 (LOW during BOOT)
- GPIO15 (HIGH during BOOT)

These pins are used to put the microcontroller in to flashing mode or bootloader mode.

SX1278 LoRa Module RA-02 433 MHz

The SX1278 LoRa module, specifically the RA-02 variant operating at 433MHz, is a wireless communication module designed for long-range, low-power communication. LoRa stands for "Long Range," and it's a modulation technique optimized for long-distance communication with low power consumption. The 433MHz frequency is commonly used for LoRa communication, especially in regions where this frequency is allocated for short-range communication.

Specifications:

Long Range Communication: LoRa modulation allows for long-range communication, making it suitable for applications that require communication over extended distances

Frequency: The RA-02 operates at 433MHz, and it's important to note that regulations regarding the use of this frequency band vary by country. Make sure you comply with local regulations when using the module.

Low Power Consumption: LoRa is designed for low-power, battery-operated devices, making it suitable for applications such as IoT (Internet of Things) devices that need to operate for extended periods on a single battery.

Data Rate: LoRa allows for different data rates. Higher data rates may result in shorter range but faster communication.

Pin Description:

Power Supply Pins: Typically, VCC and GND pins for providing power to the module.

SPI Interface Pins: Communication with microcontrollers is usually done using the Serial Peripheral Interface (SPI). Pins like SCK, MISO, MOSI, and NSS/SEL (chip select) are involved.

Antenna Connection: A pin or connector for connecting an external antenna to improve the communication range.

Reset Pin: A pin for resetting the module.

IO Pins: Some modules may have additional pins for general-purpose input/output.

Interrupt Pins: Pins that can be used to generate interrupts on specific events like packet reception.

Working Principle:

Frequency Modulation (FM): The SX1278 module uses frequency modulation to transmit data. It operates in the 433 MHz frequency band, which falls under the ISM (Industrial, Scientific, and Medical) band. This frequency band is license-free in many regions, making it suitable for various applications.

Spread Spectrum Modulation: LoRa uses spread spectrum modulation techniques to achieve long-range communication. Unlike traditional narrowband communication, where the entire signal is concentrated in a narrow frequency band, LoRa spreads the signal over a wider bandwidth. This spreading of the signal across a broad spectrum helps in achieving better signal robustness and increased resistance to interference.

Chirp Spread Spectrum (CSS): LoRa uses a form of spread spectrum modulation known as Chirp Spread Spectrum. In CSS, the frequency of the transmitted signal varies continuously over time. This results in a chirp-like waveform. The receiver, tuned to the same spreading factor, can demodulate the received signal even in the presence of noise and interference.

Spreading Factors: LoRa devices operate with different spreading factors (SF), which determine the rate at which data is transmitted and the range of the communication. A higher spreading factor provides better range but at the cost of lower data rate.

Low Power Operation: One of the key advantages of LoRa is its low power consumption. The module is designed to transmit data over long distances while consuming minimal power. This makes it suitable for battery-operated devices and applications where power efficiency is crucial.

Gateway and Node Architecture: In a typical LoRa network, there are two main components – a gateway and a node. The gateway is connected to the internet and acts as a bridge between the wireless LoRa network and the internet. Nodes, which can be sensors or other devices, communicate with the gateway over the LoRa network.

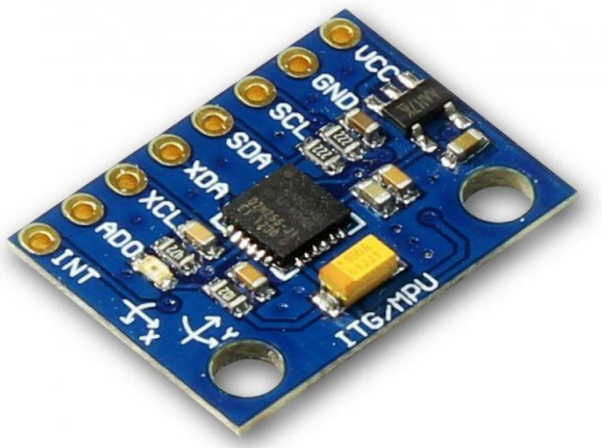
MPU6050 - Triple Axis Gyro Accelerometer Module

The MPU6050 is an IMU (Inertial Estimation Unit) sensor, a portion of the Micro-Electro-Mechanical Systems (MEMS) family. It highlights a 3-axis accelerometer and a 3-axis gyroscope. The MPU6050 module in addition called a magnetometer sensor.

It offers precise tracking for both quick and gradual movements. It includes a user-programmable gyroscope with a full-scale range of ±250, ±500, ±1000, and ±2000°/sec (DPS). Additionally, it features a user-programmable accelerometer with a full-scale range of ±2g, ±4g, ±8g, and ±16g.

Features of MPU 6050 Module:

- I2C Digital Output of 6/9-axis MotionFusion data in various formats such as Euler Angle, Quaternion, Rotation Matrix, or Raw Data.
- Chip built-in 16-bit AD converter, 16-bit data output.
- Voltage Range: 2.3V to 3.4V.
- Selectable Jumpers: CLK, FSYNC, AD0.
- Gyro: ±250 to ±2000dps sensitivity.
- Accelerometer: ±2g to ±16g programmable range.
- DMP™ Engine: Offloads MotionFusion, sync, and gesture detection.
- Embedded Algorithms: Run-time bias and compass calibration.
- Digital Temperature Sensor: Output for temperature monitoring.



MPU 6050 Specifications：

On-board	3.3V regulator
Interface	I2C Interface
Gyroscope operating current	3.6 mA

Accelerometer normal operating current	500 µA
Crystal frequency	32.768 KHz
Input power supply	5V
Dimensions	3 x 2 x 1cms

MPU6050 Pin Description:

The MPU-6050 module has 8 pins,

- **INT:** Interrupt digital output pin.
- **AD0:** I2C Slave Address LSB pin. This is 0th bit in 7-bit slave address of device. If connected to VCC then it is read as logic one and slave address changes.
- **XCL:** Auxiliary Serial Clock pin. This pin is used to connect other I2C interface enabled sensors SCL pin to MPU-6050.
- **XDA:** Auxiliary Serial Data pin. This pin is used to connect other I2C interface enabled sensors SDA pin to MPU-6050.
- **SCL:** Serial Clock pin. Connect this pin to microcontrollers SCL pin.
- **SDA:** Serial Data pin. Connect this pin to microcontrollers SDA pin.
- **GND:** Ground pin. Connect this pin to ground connection.
- **VCC:** Power supply pin. Connect this pin to +5V DC supply.
- MPU-6050 module has Slave address (When AD0 = 0, i.e. it is not connected to Vcc) as,
- **Slave Write address(SLA+W)**: 0xD0
- **Slave Read address(SLA+R)**: 0xD1

Working Principle Of MPU6050:

Gyroscope: The gyroscope measures angular velocity, indicating how fast the device is rotating around its axes. It works based on the principle of Coriolis force, where a moving object experiences a force perpendicular to its motion when it moves in a rotating frame of reference. The gyroscope senses this force to determine angular velocity.

Accelerometer: The accelerometer measures acceleration along the three axes. It typically uses the principle of capacitance change or piezoelectric effect to detect changes in acceleration. By measuring the force exerted on tiny masses inside the sensor due to acceleration, it can determine the device's orientation relative to the Earth's gravitational field.

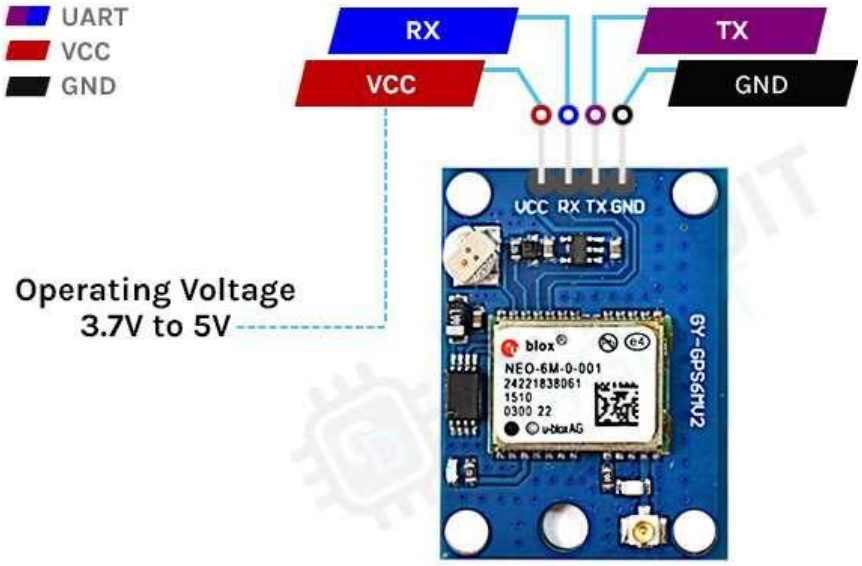
Sensor Fusion: To obtain accurate orientation data, the MPU6050 combines the outputs of the gyroscope and accelerometer using sensor fusion algorithms such as Kalman filtering or complementary filtering. By combining the strengths of both sensors (gyroscope for short-term accuracy and accelerometer for long-term stability), it provides precise orientation data, including pitch, roll, and yaw angles.

Output: The MPU6050 typically communicates with a microcontroller or other devices through an I2C (Inter-Integrated Circuit) or SPI (Serial Peripheral Interface) interface. Users can read raw sensor data or utilize built-in digital motion processing capabilities to obtain calibrated orientation data directly from the chip.

NEO-6M GPS MODULE

NEO-6M GPS MODULE PINOUT

The NEO-6M GPS module has four pins: GND, TxD, RxD, and VCC. The TxD and RxD pins are used to communicate with the microcontroller



GND is the ground pin of the GPS Module and it should be connected to the ground pin of the ESP32.

TXD is the transmit pin of the GPS module that needs to connect to the RX pin of theESP32.

RXD is the receive pin of the GPS module that needs to connect to the TX pin of the ESP32.

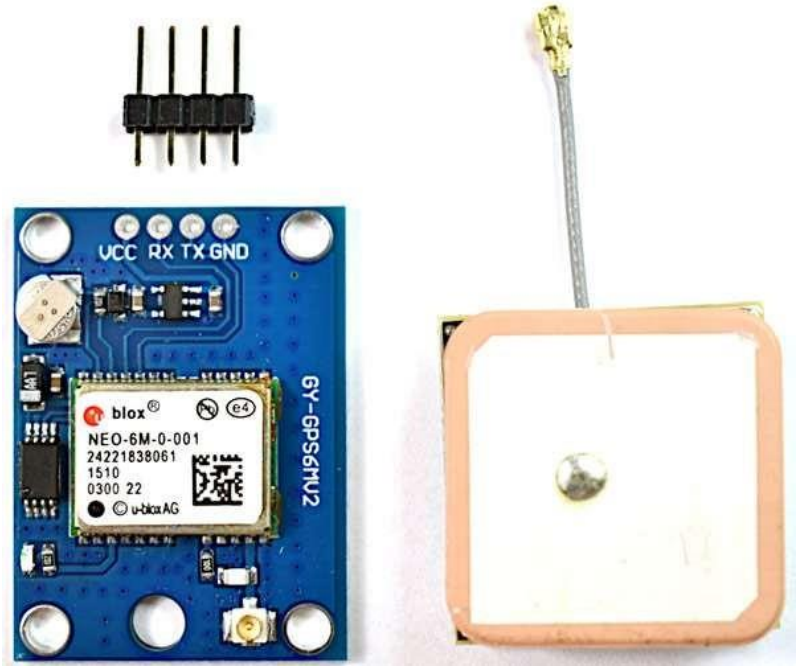
VCC is the power pin of the GPS module and needs to connect to the 3.3V pin of the ESP32.

NEO-6M GPS MODULE – PARTS

The NEO-6M module is a ready-to-use GSM module that can be used in many different applications. The parts on the NEO-6M GPS module are shown below-

The NEO-6M GPS module has five major parts on the board, the first major part is the NEO-6M GPS chip in the heart of the PCB. Next, we have a rechargeable battery and a serial EEPROM module. An EEPROM together with a battery helps retain the clock data, latest position data(GNSS orbit data), and module configuration but it's not meant for permanent data storage. Without the battery, the GPS always cold-starts so the initial GPS lock takes more time. The battery is automatically charged when power is applied and maintains data for up to two weeks without power. Next, we have our LDO, because of the onboard LDO, the module can be powered from a 5V supply. Finally, we have our UFL connector where we need to connect an external antenna for the GPS to properly work.

OVERVIEW OF THE NEO-6M GPS MODULE



The Global Positioning System (GPS) is a system consisting of 31 satellites orbiting earth. We can know their exact location because they are constantly transmitting position information with time through radio signals. At the heart of the breakout board, there is the NEO-6M GPS module that is designed and developed by u-blox. This is very small but it packs a lot of features. It can track up to 22 satellites over 50 channels while consuming only 45mA of current and has an operating voltage of 2.7V ~ 3.6V. One of the most interesting features of this module is its power-saving mode. This allows a reduction in system power consumption. With power-saving mode on, the current consumption of the module reduces to 11MA only.

NEO-6M Module datasheet.

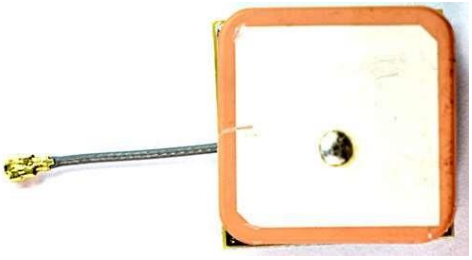
Position Fix LED Indicator: If you take a close look at the NEO-6M GPS module board, you can find a small LED that isused to indicate that the GPS module is able to communicate with the satellites.

- No blinking – it is searching for satellites.

- Blink every 1s – Position Fix is found (the module can see enough satellites).

ANTENNA:

The module comes with a -161 dBm sensitive patch antenna that can receive radio signals from GPS satellites. You can connect the antenna to a small UFL connector which we have mentioned in the parts marking section of this article.



For most outdoor applications, the patch antenna will work just fine but for more demanding or indoor applications it is advised to use a 3V active GPS antenna.

BUZZER

A **buzzer** is an electronic device that generates sound by converting electrical energy into sound energy. It typically consists of a piezoelectric crystal, which expands and contracts when an alternating current is applied to it, creating sound waves.

There are different types of **buzzers** available, including piezoelectric, magnetic, and electro-mechanical **buzzers**. Piezoelectric **buzzers** are the most common and widely used, as they are small, inexpensive, and easy to control.

Buzzers can be controlled using a variety of methods, such as using a microcontroller to generate a square wave with the correct frequency, or using a specialized IC (Integrated Circuit) such as a 555 timer. **Buzzers** are commonly used in a wide range of applications such as alarms, timers, and warning systems. They can also be used in electronic devices such as mobile phones, computers, and other electronic devices to generate different sounds and tones.

Piezoelectric **buzzers** are relatively simple to use and are widely used in DIY andhobby projects, including those that use microcontrollers such as the **ESP32**. They can be easily controlled by a microcontroller by applying a square wave with the desired frequencyto the control pin of the **buzzer**.

CONCLUSION

CODE

Transmit Code:

```
#include <SPI.h>
#include <LoRa.h>
#include <Wire.h>
#include <TinyGPS++.h>


TinyGPSPlus gps;
//defining pins of tx module
#define GPS_BAUDRATE 9600
#define ss 5
#define rst 14
#define dio0 2

const int buttonPin = 4;
int buttonState = 0;

const int MPU_addr = 0x68;  // I2C address of the MPU-6050
int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
float ax = 0, ay = 0, az = 0, gx = 0, gy = 0, gz = 0;
boolean fall = false;      //stores if a fall has occurred
boolean trigger1 = false;  //stores if first trigger (lower threshold) has occurred
boolean trigger2 = false;  //stores if second trigger (upper threshold) has occurred
boolean trigger3 = false;  //stores if third trigger (orientation change) has occurred
byte trigger1count = 0;    //stores the counts past since trigger 1 was set true
byte trigger2count = 0;    //stores the counts past since trigger 2 was set true
byte trigger3count = 0;    //stores the counts past since trigger 3 was set true
int angleChange = 0;
int counter = 0;


void setup() {
  // put your setup code here, to run once:
  //Initialising GPS Neo6M module
  Serial.begin(9600);
  Serial2.begin(GPS_BAUDRATE);
  Serial.println(F("ESP32 - GPS module"));


  /* This is the initialisation of MPU6050*/
  Serial.begin(9600);
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);  //// PWR_MGMT_1 register
  Wire.write(0);     // set to zero (wakes up the MPU-6050
  Wire.endTransmission(true);
```

```
 //initializing the serial monitor
 Serial.begin(9600);
 while (!Serial)
  ;
 Serial.println("Lora sender");
 LoRa.setPins(ss, rst, dio0);

 while (!LoRa.begin(433E6)) {
  Serial.println(".");
  delay(500);
 }

 LoRa.setSyncWord(0xF3);
 //Change sync word (0xF3) to match the receiver
 // The sync word assures you don't get LoRa messages from other LoRa transceivers
 // ranges from 0-0xFF
 Serial.println("LoRa initialised successfully!!");
 pinMode(buttonPin, INPUT);


}

void loop() {

buttonState = digitalRead(buttonPin);

 if(buttonState == LOW)
 {
  Serial.print("Sending packet : ");
 // Serial.println(counter);

  //send to lora packet to reciver

  for(int j=0;j<=5;j++)
  {
  LoRa.beginPacket();
  LoRa.print("panic need help !!!!!!!");
  LoRa.endPacket();
 // LoRa.print(counter);
  }


  counter++;
  delay(3000);
 }
```

```
mpu_read();
ax = (AcX - 2050) / 16384.00;
ay = (AcY - 77) / 16384.00;
az = (AcZ - 1947) / 16384.00;
gx = (GyX + 270) / 131.07;
gy = (GyY - 351) / 131.07;
gz = (GyZ + 136) / 131.07;

// calculating Amplitute vactor for 3 axis
float Raw_Amp = pow(pow(ax, 2) + pow(ay, 2) + pow(az, 2), 0.5);
int Amp = Raw_Amp * 10;  // Mulitiplied by 10 bcz values are between 0 to 1
Serial.println(Amp);
if (Amp <= 2 && trigger2 == false)  //if AM breaks lower threshold (0.4g)
{
  trigger1 = true;
  Serial.println("trigger 1 is activated");
}
if (trigger1 == true) {
  trigger1count++;
  if (Amp >= 12) {
    trigger2 = true;
    Serial.println("trigger 2 is activated");
    trigger1 = false;
    trigger1count = 0;
  }
}
if (trigger2 == true) {
  trigger2count++;
  angleChange = pow(pow(gx, 2) + pow(gy, 2) + pow(gz, 2), 0.5);
  Serial.println(angleChange);
  if (angleChange >= 30 && angleChange <= 400) {  //if orientation changes by between 80-100
degrees
    trigger3 = true;
    trigger2 = false;
    trigger2count = 0;
    Serial.println(angleChange);
    Serial.println("TRIGGER 3 ACTIVATED");
  }
}
if (trigger3 == true) {
  trigger3count++;
  if (trigger3count >= 10) {
    angleChange = pow(pow(gx, 2) + pow(gy, 2) + pow(gz, 2), 0.5);
    delay(10);
    Serial.println(angleChange);
    if ((angleChange >= 0) && (angleChange <= 10)) {  //if orientation changes remains between
0-10 degrees
      fall = true;
      trigger3 = false;
      trigger3count = 0;
```

```
        Serial.println(angleChange);
       } else {  //user regained normal orientation
        trigger3 = false;
        trigger3count = 0;
        Serial.println("TRIGGER 3 DEACTIVATED");
       }
     }
    }
    if (fall == true) {  //in event of a fall detection
     Serial.println("FALL DETECTED");
     for(int i=0;i<5;i++)
    {
     LoRa.beginPacket();
     LoRa.print("ALERT!!! The person has fallen");
     LoRa.endPacket();
     gps_loc();
     delay(1000);
    }


     fall = false;
    }
    if (trigger2count >= 6) {  //allow 0.5s for orientation change
     trigger2 = false;
     trigger2count = 0;
     Serial.println("TRIGGER 2 DEACTIVATED");
    }
    if (trigger1count >= 6) {  //allow 0.5s for AM to break upper threshold
     trigger1 = false;
     trigger1count = 0;
     Serial.println("TRIGGER 1 DEACTIVATED");
    }
    delay(100);

    gps_loc();
    // put your main code here, to run repeatedly:
    // Serial.print("Sending packet : ");
    // Serial.println(counter);

    // //send to lora packet to reciver
    // LoRa.beginPacket();
    // LoRa.print("hello world");
    // LoRa.print(counter);
    // LoRa.endPacket();

    // counter++;
    // delay(3000);
   }

   void mpu_read() {
    Wire.beginTransmission(MPU_addr);
```

```
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr, 14, true);  // request a total of 14 registers
  AcX = Wire.read() << 8 | Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C
(ACCEL_XOUT_L)
  AcY = Wire.read() << 8 | Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E
(ACCEL_YOUT_L)
  AcZ = Wire.read() << 8 | Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40
(ACCEL_ZOUT_L)
  Tmp = Wire.read() << 8 | Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  GyX = Wire.read() << 8 | Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY = Wire.read() << 8 | Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ = Wire.read() << 8 | Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
 }
 void gps_loc()
 {
  if (Serial2.available() > 0) {
    if (gps.encode(Serial2.read())) {
      if (gps.location.isValid())
      {
        LoRa.beginPacket();
        //LoRa.print("Everything is alright!");
        LoRa.print("- GPS Parameters -");
        LoRa.endPacket();
        //Serial.print(F("- latitude: "));
        LoRa.beginPacket();
        LoRa.print("- latitude:");
        //Serial.println(gps.location.lat());

        LoRa.print(gps.location.lat());
        LoRa.endPacket();
        //Serial.print(F("- longitude: "));
        LoRa.beginPacket();
        LoRa.print("- longitude: ");
        //Serial.println(gps.location.lng());
        LoRa.print(gps.location.lng());
        LoRa.endPacket();

        // Serial.print(F("- altitude: "));
        // if (gps.altitude.isValid())
        //   Serial.println(gps.altitude.meters());
        // else
        //   Serial.println(F("INVALID"));
      } else
      {
        LoRa.beginPacket();
        LoRa.print("- location: INVALID");
        LoRa.endPacket();
      }

      // Serial.print(F("- speed: "));
```

```
    // if (gps.speed.isValid()) {
    //   Serial.print(gps.speed.kmph());
    //   Serial.println(F(" km/h"));
    // } else {
    //   Serial.println(F("INVALID"));
    // }

    // Serial.print(F("- GPS date&time: "));
    // if (gps.date.isValid() && gps.time.isValid()) {

    //   Serial.print(gps.date.year());
    //   Serial.print(F("-"));
    //   Serial.print(gps.date.month());
    //   Serial.print(F("-"));
    //   Serial.print(gps.date.day());
    //   Serial.print(F(" "));
    //   Serial.print(gps.time.hour());
    //   Serial.print(F(":"));
    //   Serial.print(gps.time.minute());
    //   Serial.print(F(":"));
    //   Serial.println(gps.time.second());
    // } else {
    //   Serial.println(F("INVALID"));
    // }

     Serial.println();
     delay(1000);
   }
  }

  if (millis() > 5000 && gps.charsProcessed() < 10)
    Serial.println(F("No GPS data received: check wiring"));
}
```

Recivew Code:

```
#include <SPI.h>
#include <LoRa.h>

#define ss 5
#define rst 14
#define dio0 2
#define buzzerPin 4  // Define the pin connected to the buzzer

void setup() {
  Serial.begin(9600);
  while (!Serial);
  Serial.println("LoRa Receiver");
```

```
  // Setup LoRa transceiver module
  LoRa.setPins(ss, rst, dio0);
  while (!LoRa.begin(433E6)) {
    Serial.println(".");
    delay(500);
  }
  LoRa.setSyncWord(0xF3);
  Serial.println("LoRa Initializing OK!");

  pinMode(buzzerPin, OUTPUT);  // Initialize the buzzer pin as an output
}

void loop() {
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    Serial.print("Received packet");

    String LoRaData = "";
    while (LoRa.available()) {
      LoRaData += (char)LoRa.read();
    }
    Serial.print(LoRaData);
    Serial.print("' with RSSI ");
    Serial.println(LoRa.packetRssi());

    // Check if the received message indicates the button was pressed
    if (LoRaData == "panic need help !!!!!!!") {
      activateBuzzer();  // Activate the buzzer if the specific message is received
    }
  }
}

void activateBuzzer() {
  // Example of buzzer activation for 1 second
  digitalWrite(buzzerPin, HIGH);  // Turn on the buzzer
  delay(1000);                    // Leave it on for 1 second
  digitalWrite(buzzerPin, LOW);   // Turn off the buzzer
}
```