Republic of the Philippines
BATANGAS STATE UNIVERSITY
The National Engineering University
Alangilan Campus
Golden Country Homes, Alangilan Batangas City, Batangas, Philippines 4200
Tel Nos.: (+63 43) 425-0139 local 2121 / 2221
E-mail Address: ceafa@g.batstate-u.edu.ph | Website Address: http://www.batstate-u.edu.ph

College of Engineering, Architecture and Fine Arts

Group 4:

Dimaculangan, Raven A.
Gonda, John Cedric T.
Lopez, Mark Erick F**.**

Topic Proposal: HELMOTECH (HELMET AND TECHNOLOGY)

Introduction

HelmoTech is a groundbreaking smart helmet designed to significantly enhance motorcycle safety by ensuring helmet compliance and providing advanced accident detection features. This innovative device integrates cutting-edge technology to protect riders and improve their overall riding experience. The helmet is equipped with infrared sensors that detect whether the rider is wearing it, preventing the motorcycle from starting unless the helmet is securely fastened. This promotes safe riding habits by ensuring that riders always wear their helmets. If the rider removes the helmet after starting the motorcycle, a 15-second timer is activated. If the helmet is not worn again within this timeframe, the motorcycle will automatically shut down to prevent accidents caused by reckless behavior. In the event of an accident, HelmoTech's impact sensors detect strong collisions and trigger an emergency response system. If the helmet is being worn during the accident, the system sends an SMSalert with the precise location of the incident to a designated emergency contact or contact person. This ensures timely assistance and reduces response times, which can be crucial in emergency situations. The helmet and motorcycle communicate seamlessly using RF modules, allowing for real-time data exchange and enhancing the overall safety and functionality of the system.

General Objectives:

- To develop a functional and reliable smart helmet system that improves motorcycle rider’s safety through enforced helmet compliance and rapid, location-based accident notification.

Specific Objectives:

- To ensure rider safety by enforcing helmet use before starting the motorcycle.
- To detect and respond to accidents in real-time.
- To provide an emergency alert system with real-time location tracking.
- To enhance road navigation and safety through voice-assisted Google Maps integration.

Project Components:

- Esp 32
- MPU6050 (Accelerometer/Gyroscope)
- IR sensor
- RF 433MHz Transmitter and Receiver
- Magnetic Reed Switch
- Relay Module
- GPS Module
- GSM Module (SIM800L)
- Lithium Battery
- Helmet for prototype
- DC motor as motor (for presentation)

4C's of Design:

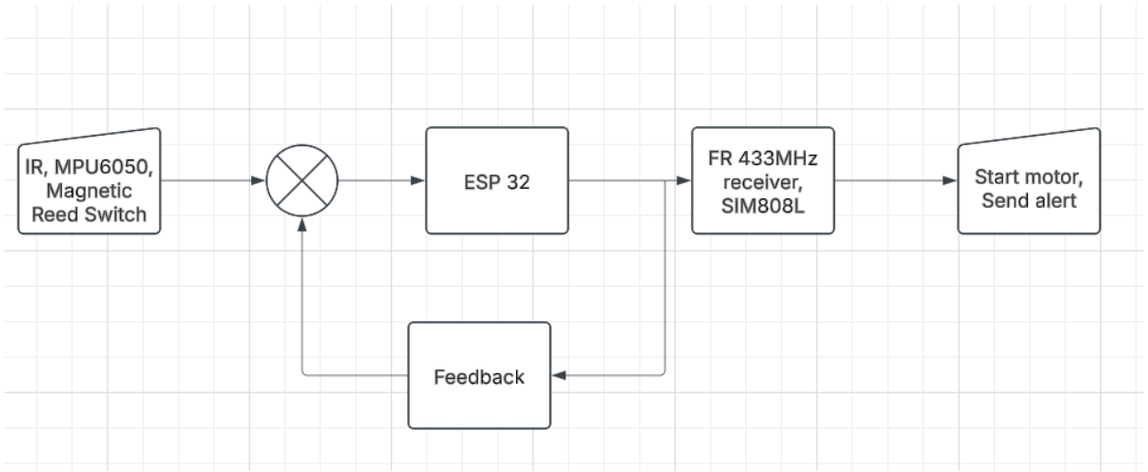
- **Cost:**
  - The cost of HelmoTech will depend on the integration of advanced technologies such as infrared sensors, RF modules, impact sensors, and Google Maps integration. These components are likely to increase production costs compared to standard helmets.
  - Additional expenses may include software development for voice commands and emergency response systems, as well as materials for durability and comfort.
  - Pricing must strike a balance between affordability for mass adoption and covering the cost of high-tech features.
- **Constraints:**
  - **Technological Constraints:** Ensuring seamless communication between the helmet and motorcycle via RF modules without interference or latency.
  - **Power Supply:** The helmet requires a reliable power source for sensors, GPS, and communication modules, which must be lightweight and long-lasting.
  - **Durability:** The helmet must remain functional under extreme weather conditions, impacts, and prolonged use.
  - **Regulatory Compliance:** It must meet safety standards and regulations in various markets.
  - **User Comfort:** Maintaining a lightweight design while accommodating all the embedded technology is critical to avoid rider fatigue.
- **Considerations:**
  - **Safety First:** The primary focus is on accident prevention (helmet compliance) and emergency response (impact detection).
  - **User Experience:** Features like voice commands for navigation and real-time road updates should be intuitive and distraction-free.
  - **Market Demand:** Targeting riders who prioritize safety and technology while ensuring accessibility for broader demographics.
  - **Scalability:** Designing the helmet to allow future upgrades, such as adding new sensors or integrating with emerging technologies like AI.
  -

- **Concept:**

  HelmoTech integrates cutting-edge safety features with advanced connectivity to enhance the riding experience:

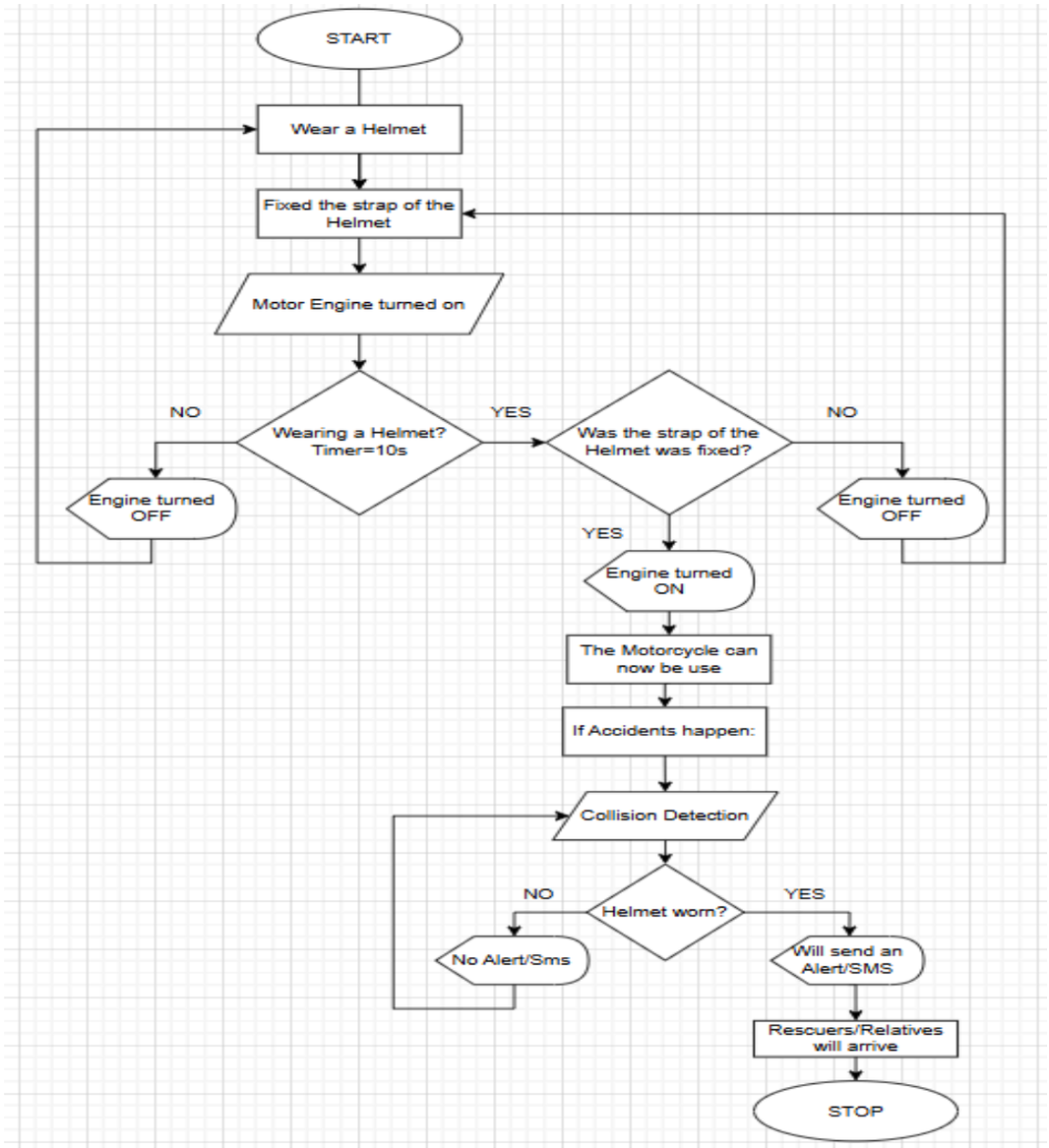
  - **Helmet Compliance System:** Infrared sensors ensure the helmet is worn before starting the motorcycle and during the ride.
  - **Accident Detection & Emergency Response:** Impact sensors trigger SMS alerts with location details in case of a crash.
  - **RF Communication Modules:** Enable real-time data exchange between the helmet and motorcycle for seamless functionality.

Block Diagram:



This block diagram represents the control system of the HelmoTech using IR sensor, MPU6050 Accelerometer/Gyroscope, Magnetic Reed switch, FR 433MHz transmitter and receiver, SIM808L, and ESP 32. System starts by detecting whether the helmet is worn by the user and if the strap is fastened using the IR sensors which uses infrared light and Magnetic Reed switch uses magnetic field to close the circuit. ESP 32 processes all these inputs including the reading from MPU6050 and then sends signal to the FR 433MHz to make the relay close the circuit and allow motor starting if helmet and strap condition satisfies and to SIM808L that will send alert to the designated contact person. Lastly, there will be feedback for continuous detection of error on the system, like for example the helmet is worn and the strap is fastened FR 433MHz will send signal to allow the motor to start and if the helmet is removed there will be feedback and the ESP 32 will process it again and FR 433MHz will not send signal so that the motor will not start..

Flowchart of the system:



This flowchart represents a motorcycle safety system that aims to ensure a user wears a helmet correctly and provide assistance incase of an accident. The process begins with a pre-ride safety check, reminding the user to wear a helmet and fasten the strap. The system then attempts to start the engine, and if the helmet is not detected within 15 seconds, the engine automatically turns off and the user is reminded again to wear a helmet. If the helmet is detected but the strap isn't fastened, the engine also turns off, and the user is reminded again to fix the strap of their helmet. Once the system confirms that the helmet is worn and the straps fastened, the engine is allowed to start, and the motorcycle can be used by the user. The second part of the flowchart deals with post-accidents response. If a collision is detected, the system checks if the user was wearing a helmet at the time of the accident. If not, no alert or sms is sent. However, if the user was wearing a helmet, an alert and SMS message are sent, prompting rescuers or relatives to arrive at the accident scene. This ensures that users who take the precaution of wearing a helmet receive timely assistance in case of an accident. This system, therefore, enforces helmet use through pre-ride checks and prioritizes assistance to users who have taken the necessary safety precautions in the event of a collision.

Pseudocode representation for the code of the system:

```
START

// Initialize variables, pins, and communication protocols
SET WiFi credentials
DEFINE pins for GPS, SIM800L, sensors
INITIALIZE libraries: WiFi, GPS, MPU6050, GSM, RF Transmitter
SETUP UART communication for GPS and SIM800L
CONNECT to WiFi and print IP
INITIALIZE GPS, MPU6050, GSM module, and RF Transmitter
SET pin modes for helmet and strap sensors
DISPLAY "System Initialized"

LOOP forever:

  // Read GPS data
  IF GPS data available:
    READ character from GPS
    IF GPS data is valid:
      STORE latitude, longitude, and speed

  // Print GPS data periodically
  IF 1 second has passed:
    IF GPS data valid:
      DISPLAY latitude, longitude, speed, satellite count
    ELSE:
      DISPLAY "Waiting for GPS fix..."

  // Check helmet and strap sensors
  CALL checkHelmetAndStrap()
    READ reed switch and IR receiver
    IF both indicate worn and fastened:
      SET helmetWorn and strapFastened to TRUE
      SEND "ON" via RF transmitter
    ELSE IF strap not fastened:
      DISPLAY "Fasten your strap!"
    ELSE IF helmet not worn:
      DISPLAY "Wear helmet first!"

  // Read MPU6050 acceleration periodically
  IF 1 second has passed:
    CALL getMpuData()
      READ x and y acceleration
      CALCULATE horizontal acceleration in m/s²
      DISPLAY acceleration
      IF speed is high AND sudden deceleration:
        SET accidentDetected to TRUE
        DISPLAY "Accident detected"
      ELSE:
        SET accidentDetected to FALSE
```

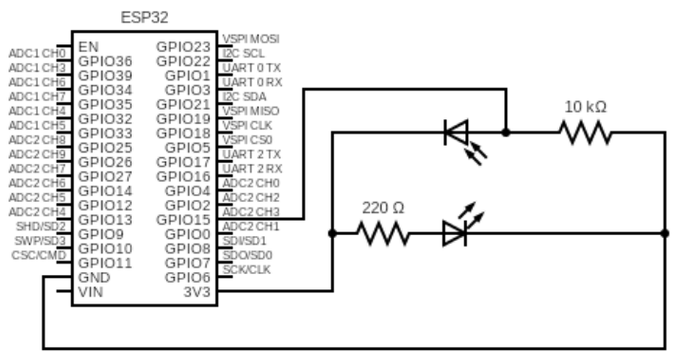
// Check if conditions are met to send SMS
IF 1 second has passed:
IF helmet worn AND strap fastened AND not already sent AND accident detected AND speed high:
CALL sendSMS()
COMPOSE message with coordinates
SEND SMS to admin number
DISPLAY success/failure message
SET alreadySentSignal to TRUE

END LOOP

This pseudocode represents a smart safety system for motorcyclists using the ESP32 microcontroller. In the setup phase, the system initializes various components: WiFi for connectivity, GPS for real-time location and speed, MPU6050 for detecting sudden decelerations (accidents), a SIM800L module for sending SMS alerts, and a 433 MHz RF transmitter for remote communication. It also configures input sensors for helmet and strap detection. Once the system is running in the loop, it continuously reads GPS data to determine position and speed, monitors the helmet and strap to ensure they are properly worn, and checks for sudden decelerations using the accelerometer. If all safety conditions are met (helmet worn, strap fastened, high speed, and sudden deceleration), the system identifies a potential accident and automatically sends an SMS alert with the rider's location to a predefined contact. The RF transmitter is also used to send an "ON" signal when the helmet is correctly worn. This pseudocode provides a structured overview of the safety logic and communication flow in the original Arduino code.

Design and Schematic diagram

IR Transmitter & Receiver:



Our group used IR Transmitter & Receiver in our project mainly just to detect whether the helmet is worn. These IRs are placed inside the helmet and facing each other by pair one pair in front and back and other pair on sides so when the helmet is worn, IR light is blocked meaning signal is low and motorcycle will be ready to start once other conditions are also met.

Other reasons why we chose IR Transmitter & Receiver is because it is really simple and

low-cost, it allows non-contact detection, it is reliable in short ranges just like inside the helmet that is short range and enclosed space which is really ideal for IR beam-break, and lastly, it is low power, it draws only very little current.

The typical forward voltage of IR Led is around 1.2V, our desired value of current is around 10 mA, it is chosen based on typical IR Led specifications that gives a safe and efficient operation. So by using a 220 Ω resistor we can get 9.5 mA which is a good and safe level and also strong enough for short-range detection. We don't want using a much lower value of resistor cause it will draw a really high current that can lead to damaging of the component since it will be on for a long time, and also a higher value resistor because IR light will go dim and receiver might not get the signal it needed, so 220 Ω is the best choice that we have.

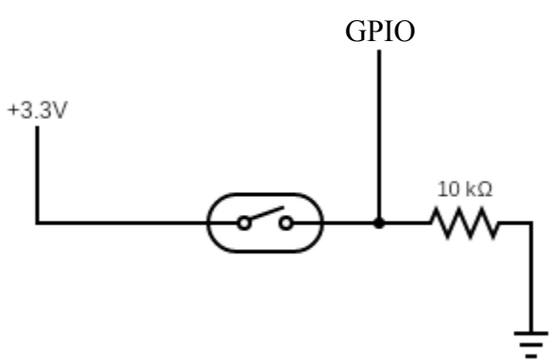
$$I = 3.3V\text{-}2.1V / 220\ \Omega = 9.5\ mA$$

The IR receiver photodiode is connected in reverse bias, with the cathode connected to 3.3V and the anode connected to a 10kΩ pull-down resistor and the ESP32 GPIO pin. The junction between the photodiode's anode, the pull-down resistor, and the ESP32 pin is normally at almost 0V when no IR light is present. When IR light is detected, the photodiode conducts, generating a small current from 3.3V that creates a voltage at the junction, which the ESP32 reads as a HIGH signal (1). There is a pull-down resistor so that when the photodiode is not conducting, meaning no IR light is detected , the pull-down resistor ensures that the ESP32 input pin is held at a defined LOW level. This prevents the pin from floating, which could otherwise result in unstable or unpredictable signal readings. The 10kΩ resistor was chosen because it provides a balance between generating a readable voltage drop assuming we have 200 µA photocurrent and keeping power consumption low.

$$V = 200\ \mu A \times 10{,}000\ \Omega = 2\ V$$
$$I = 3.3\ V / 10\ k\Omega = 0.33\ mA$$

The photodiode is connected in reverse bias to make it more sensitive to IR light. In this configuration, it normally blocks current flow. However, when IR light is present and hits the photodiode, it generates a small photocurrent. This behavior allows for better control and detection of light levels, as the current flow is directly related to the presence of IR light. For this project, reverse bias is ideal because we only want current to flow when IR light is detected, making it suitable for reliably determining whether a helmet is worn or not.

Magnetic Reed Switch:

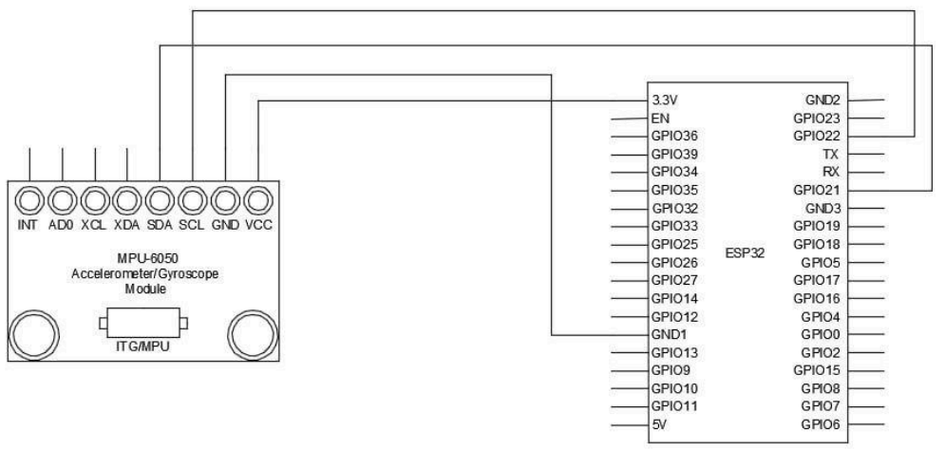


We used a normally open Magnetic Reed Switch for detecting whether the strap is fastened or not. The Magnetic Reed Switch is placed on the end of the other strap while the magnet that will be used to activate the Reed Switch is on the other strap, so that Reed Switch will only activate when the strap is fastened because the magnet and the Reed Switch will be close to each other. The reason why we need to detect if the strap is fastened is because what is the point of wearing a helmet if it is just going to fly away in an accident because the strap is not fastened?

Our group chose to use a magnetic reed switch not because it was our only option, but because it is the simplest and most practical component for our application. It is low-cost, easy to understand, and operates without consuming power, as it functions merely as a conductive medium, similar to a wire that is either open or closed depending on the magnetic field.

Same on the IR receiver, we used a 10kΩ pull-down resistor so that the esp32 pin is not floating that could possibly lead to unpredictable signal reading, and also the 10kΩ resistor is balanced between several factors like noise immunity, current consumption, and switching speed.

MPU-6050 Accelerometer/Gyroscope Module:



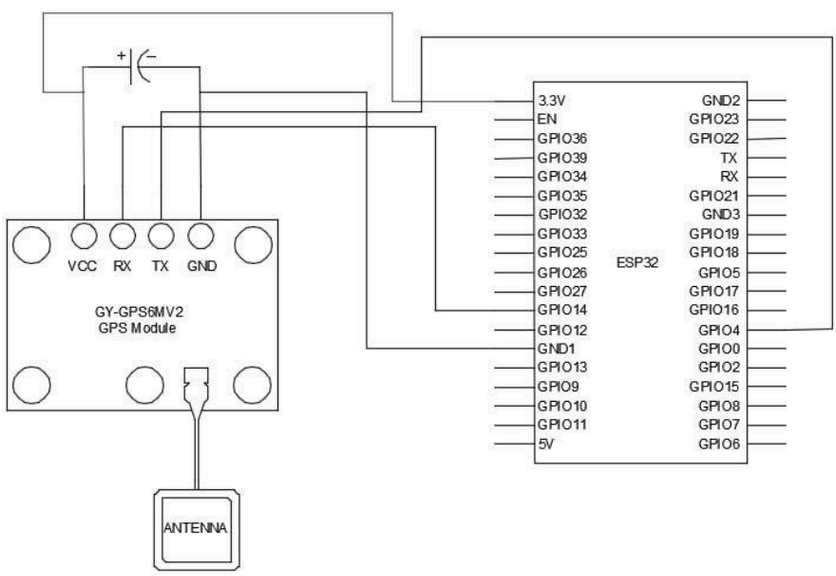
We used the MPU6050 accelerometer/gyroscope module for accident or impact detection, as it is capable of sensing sudden changes in motion, particularly deceleration. When a vehicle experiences a sudden stop or impact, the accelerometer detects a rapid decrease in velocity, indicating a potential collision or accident. The ESP32 continuously monitors this data,

and if an impact is detected, it sends a signal to the SIM800L GSM module to transmit an emergency SMS and flash alert to the user's designated contact.

However, this action is only triggered if the system also confirms that the helmet is being worn and the strap is properly fastened. If the helmet is not detected, or the strap is not secured, the impact signal is ignored to prevent false alerts.

The MPU6050 module we used is a small breakout board (such as the GY-521) that includes onboard pull-up resistors for the I2C communication lines. Therefore, it can be connected directly to the ESP32 using the I2C interface: VCC to 3.3V, GND to GND, SCL to GPIO22, and SDA to GPIO21. No additional components are required for communication, and care was taken to ensure the module was powered using 3.3V to match the logic level of the ESP32.

GY-NEO6MV2 GPS Module:

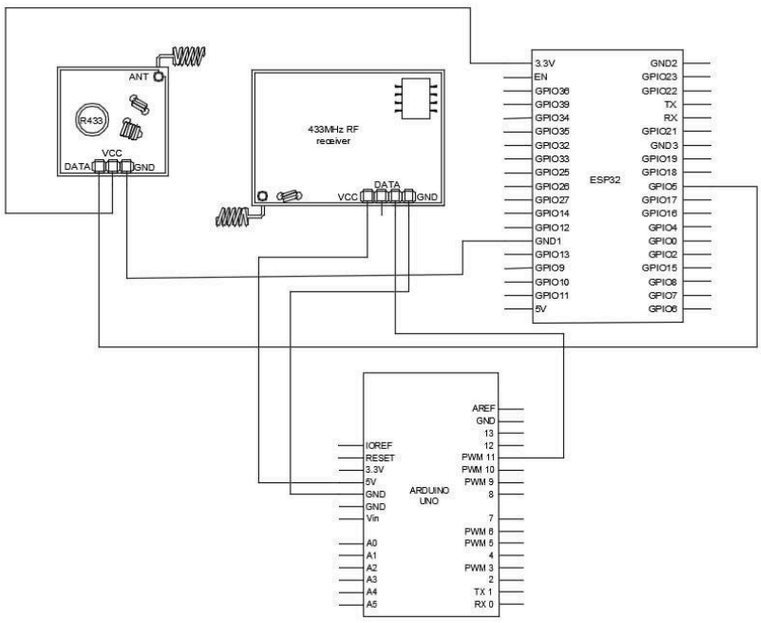


We decided to use a GPS module in conjunction with the MPU6050 because the MPU6050 alone cannot provide actual velocity or speed data. While the MPU6050 is effective

in detecting sudden changes in acceleration and deceleration, it cannot measure the true ground speed of the moving vehicle. By integrating a GPS module, we are able to obtain accurate real-time speed data, which enhances the reliability of our accident detection system. This combination allows us to determine if the user was moving at a significant speed like for example 35 km/h (9.72 m/s)before a sudden stop or impact occurred. With this data, we can reduce false alarms by ensuring that only high-speed accidents trigger alerts or emergency messages.

Practically speaking, the combination of these two prevents false alarms in scenarios like the helmet being accidentally dropped or the user slipped while walking wearing the helmet, where the MPU6050 might detect a sudden jolt, but the GPS confirms no actual travel speed. The combination of both sensors serves as a fail-safe mechanism, enhancing the overall accuracy, reliability, and real-world applicability of our accident detection system.

433MHz RF Transmitter Receiver:



Since we want to control the powering on and off of the motor wirelessly when the helmet is worn and the straps are fastened, we decided to use a 433MHz RF transmitter and receiver, where the transmitter is connected to the ESP32 microcontroller and the receiver is connected to Arduino with the motor alongside a relay module that we will control.

The transmitter will send a wireless signal only when the correct helmet conditions are met, and the receiver will then trigger the relay accordingly. This setup provides a simple and cost-effective wireless switching system, minimizing the need for long physical wiring between the helmet and the motor, and allowing greater flexibility and reliability in the operation of the system.

To further enhance the reliability and range of the wireless communication, we also decided to attach a 17.3 cm wire antenna to both the transmitter and the receiver. This antenna length is based on the 1/4 wavelength principle, which optimizes the signal transmission and reception at

433 MHz frequency. By adding antennas, we improve the signal strength, increase the effective communication distance, and ensure stable operation even in environments with minor obstructions.

$$\lambda = c/f$$

Where in: c is the constant speed of light 3×10^8 m/s, and f is the frequency of our device which is 433 MHz.

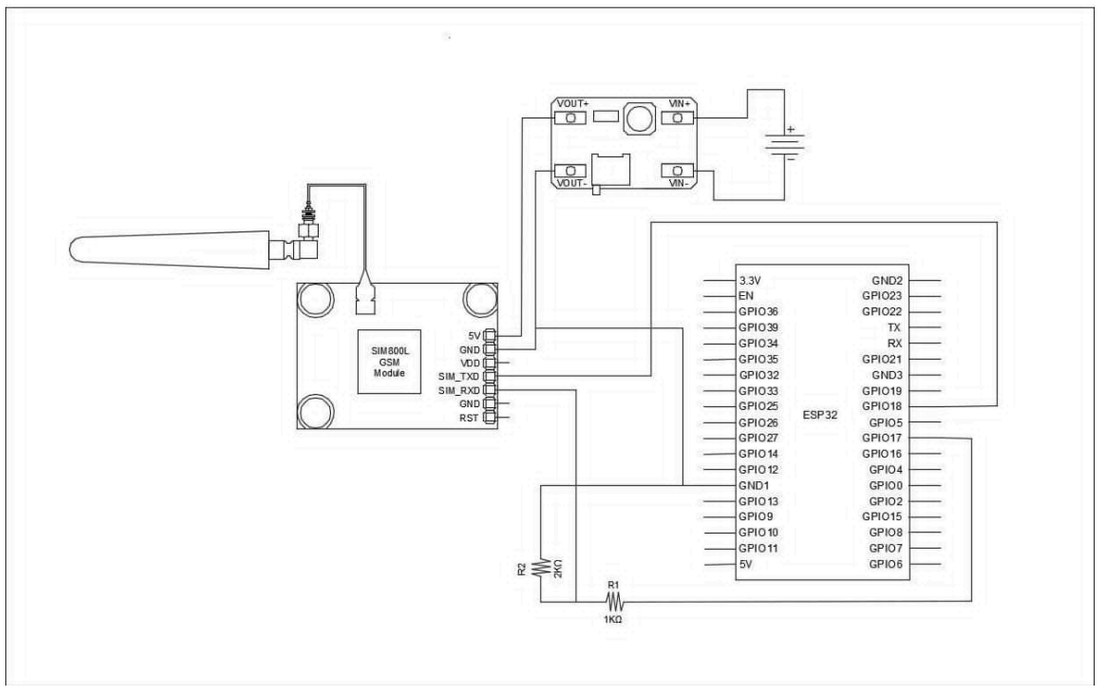
Therefore:

$$\lambda = (3x10^8 \text{ m/s}) / 433 \text{ MHz}$$
$$\lambda = 0.6928 \text{ m}$$

And then we will apply the 1/4 wavelength principle:

$$0.6928 \text{ m} / 4 = 0.1732 \text{ m or } 17.3 \text{ cm}$$

SIM800L GSM Module:



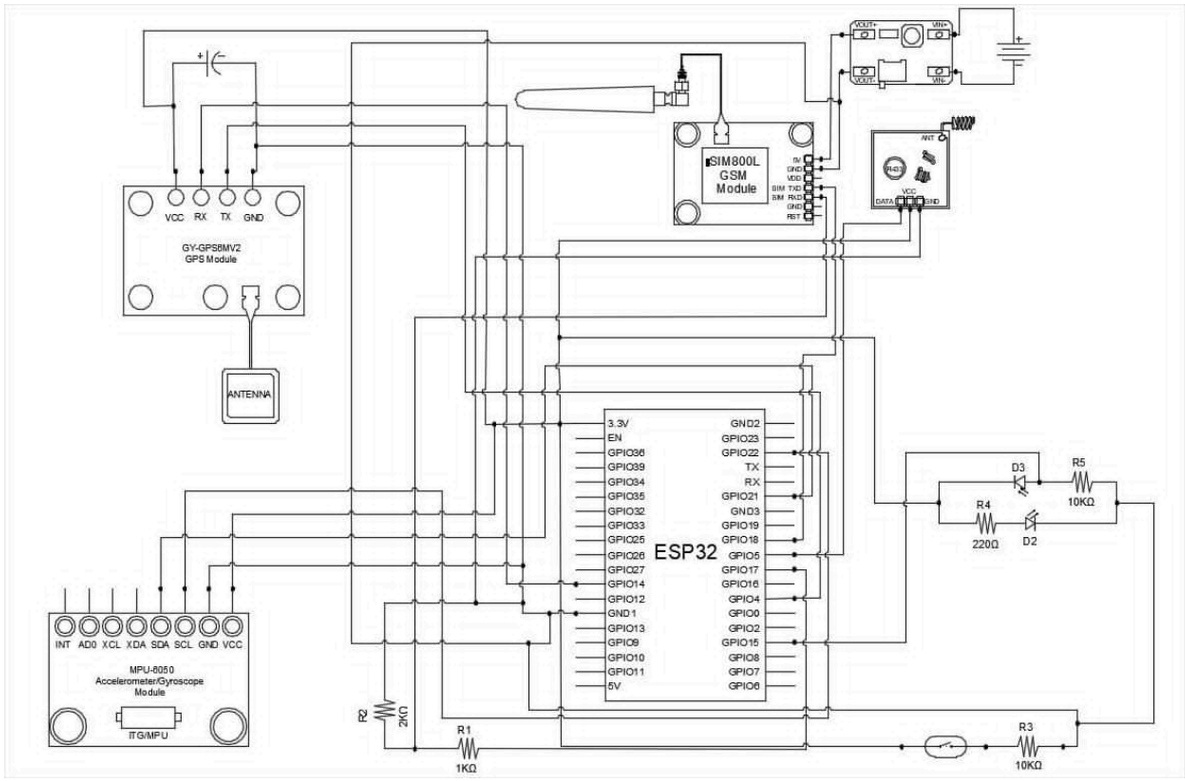
Since we are detecting accidents and wanted to send an alert or SMS to a designated contact person, we added a SIM800L GSM Module to our project, since it is capable of sending flash alerts and SMS. Additionally, the SIM800L module is ideal because it operates using standard mobile networks (2G GSM), making it reliable even in areas without Wi-Fi connectivity. It supports essential AT commands that allow flexible control over messaging and call features. Its compact size, low power consumption, and compatibility with microcontrollers like the ESP32 also make it an excellent choice for portable, battery-powered systems such as a smart helmet. By using the SIM800L, the project can immediately deliver emergency notifications in real-time without depending on external Wi-Fi or Bluetooth infrastructure, ensuring a faster response during critical situations.

Notice that we used a voltage divider on the Rx side of the SIM800L to limit the voltage that it is receiving from the ESP because ESP is 3.3V logic and the SIM800L only expects 2.8V, so, having a voltage divider if 1k ohm and 2k ohm will step down the voltage that it is receiving from 3.3V to 2.2V.

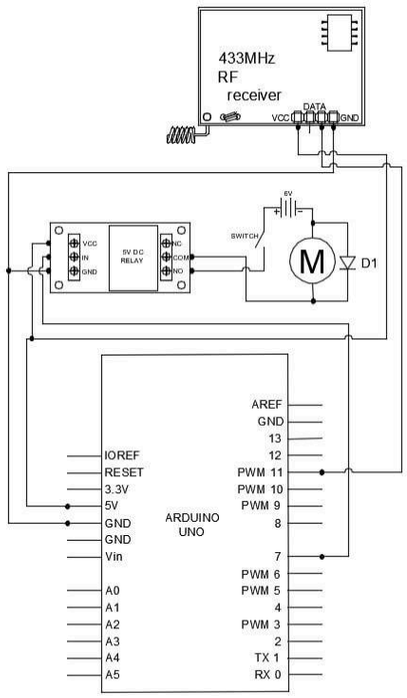
$$Vrx = 2k\Omega / (2k\Omega+1k\Omega) * 3.3V$$
$$Vrx = 2.2V$$

This 3.3 mA is very small and safe, and well below the damaging levels for the RX pin of SIM800L. Without the resistor, an accidental overvoltage could inject higher current instantly, which might damage the module. So, inserting a 1kΩ resistor just acts as a protection for the SIM800L, allowing safe communication even if there are small differences between the ESP32 output and SIM800L input.

Full System Schematic:



Motor side connection:



On the side of the motor, we used a 5VDC relay module to control the starting of the motor, but notice that we used an Arduino uno on the motor side because the receiver is prone to noise if its used without a microcontroller, using an Arduino uno makes the communication of the RF transmitter and receiver more efficient.

In addition to detail, we added an additional external power source just for the motor that is rated to 6V and also a switch, the switch is just to show that even though we close it the motor will not start as long as the IN of the relay is not getting a signal from our 433MHz receiver, and we also added a flyback diode on the motor just for protection because when motor is receiving power it is acting as motor but when the power stops the motor does not stop right away there will be a bit more rotation left and that is when the motor becomes a generator and throws power back, and that is the main reason of adding a flyback diode.

Actual project update

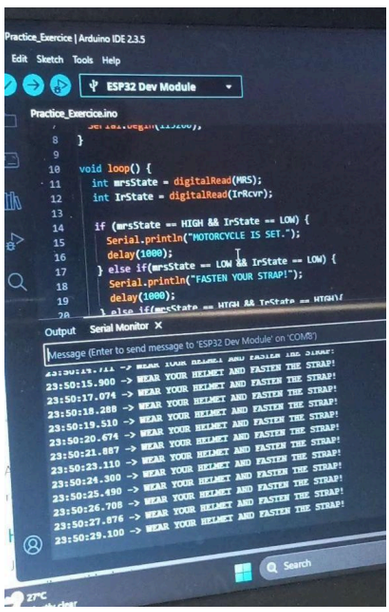
We are currently done with the helmet detection and motor starting where in the motor will only start when the microcontroller ESP32 detects that the helmet is being worn and that the strap is fastened, with the use of IR sensor and Reed switch, on testing the IR, we just blocked the IR receiver from receiving or detecting IR light using a small piece of cardboard indicating that the helmet is worn, and for the reed switch, we just used a magnet and puts it closed to the reed switch to indicate strap is fastened.

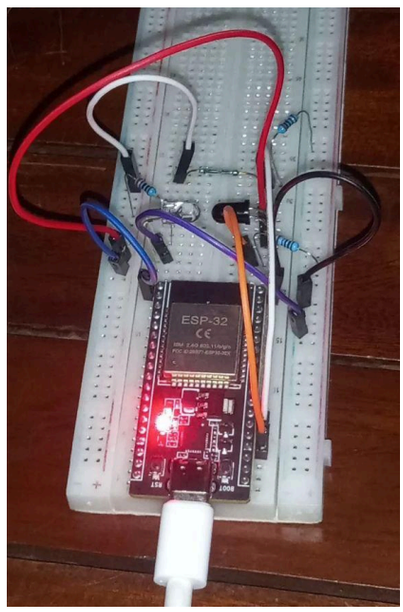
While on the motor side, we have decided to use another microcontroller which is the Arduino Uno since we already have it available. We used an Arduino on the motor side mainly because of the 433MHz RF receiver, upon troubleshooting and debugging, we noticed that if we directly connect the RF receiver to our relay, the relay is clicking rapidly indicating that it is sensing highs and lows or it is floating but before we decided to use arduino we first tried using

pull-down and pull-up resistors thinking it would solve the problem but it did not, that is when we used the arduino so that the receiver can communicate with the transmitter effectively and also for the relay to only trigger when signal from the transmitter is received, with that approach we did solved the problem on the relay that is rapidly turning on and off.

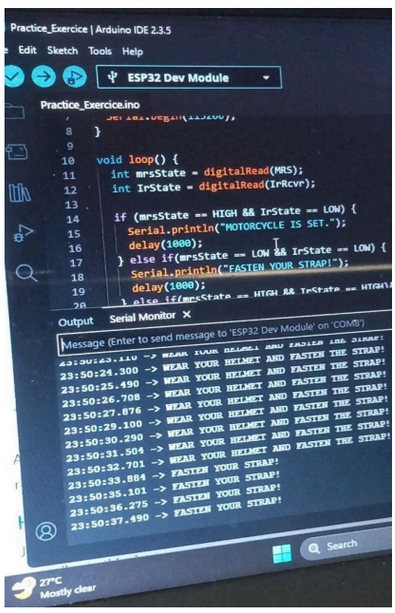
Take note, we only utilized Arduino on the motor side for the effective communication of our 433MHz RF transmitter receiver but the ESP32 is still the one that will have control of everything. As for the SIM800L, MPU6050, and GPS module, we are just starting to work on it because we ordered these components and it just arrived.

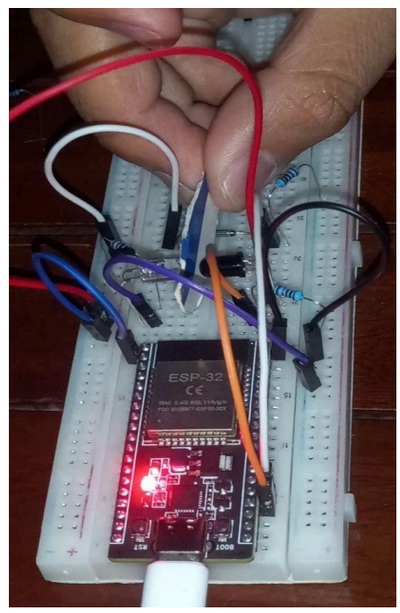
Helmet and Strap detection testing



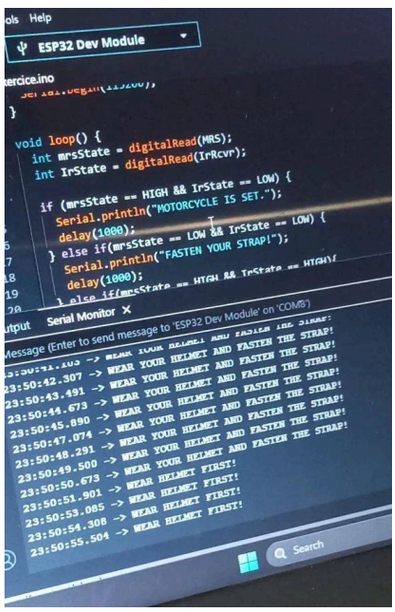


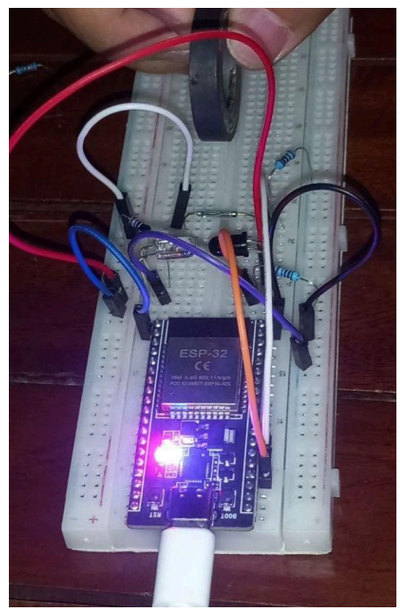
Helmet not worn; Strap not fastened



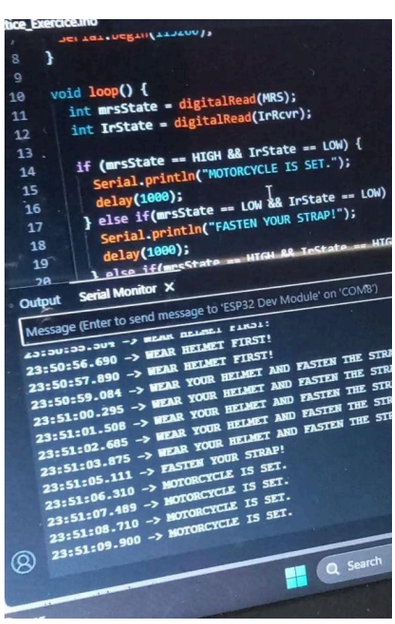


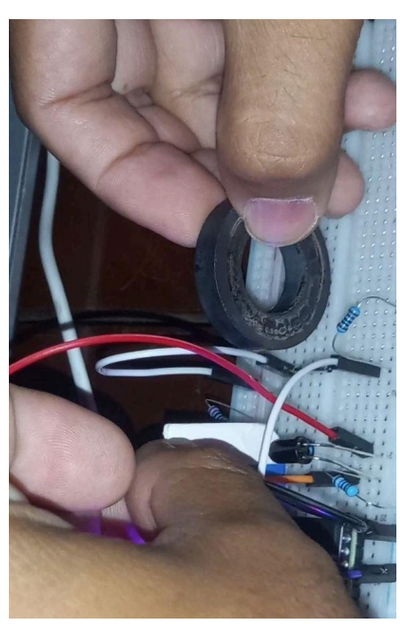
Helmet worn ; Strap not fastened





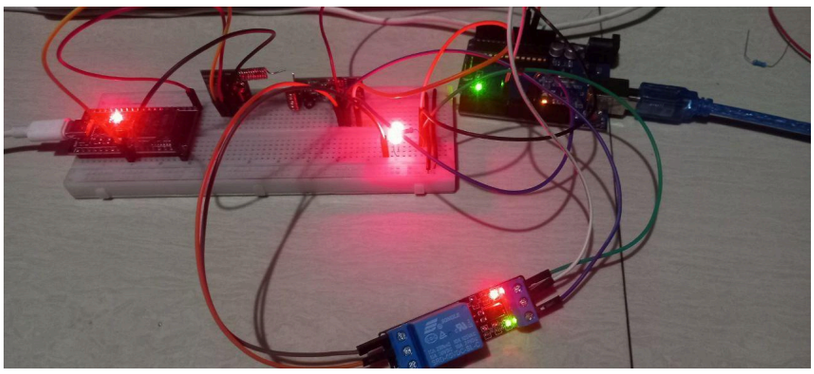
Helmet not worn; Strap fastened

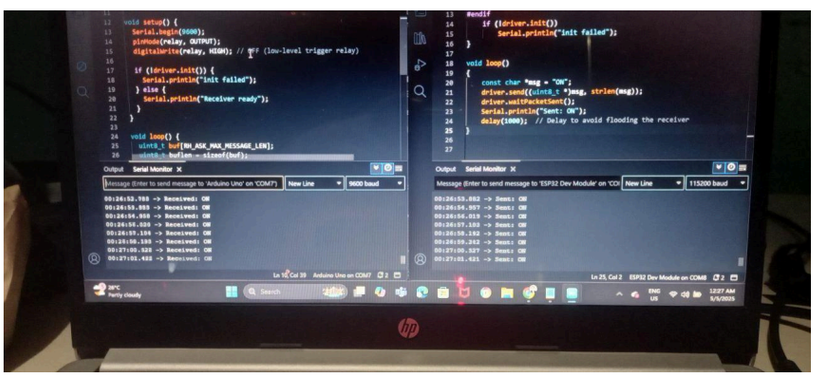




Helmet worn; Strap fastened

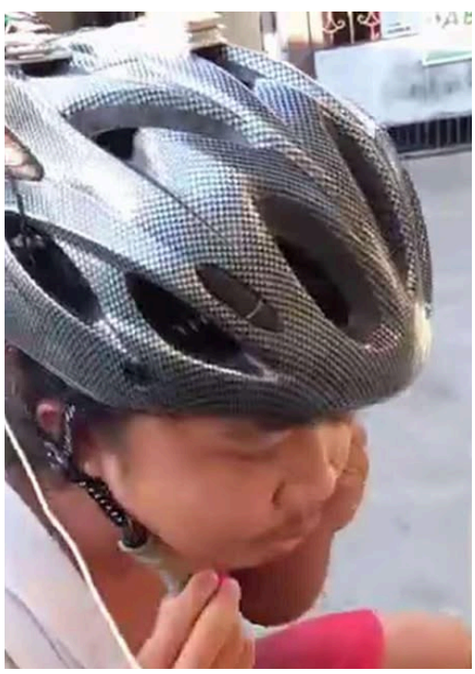
433MHz RF Transmitter Receiver and Relay testing



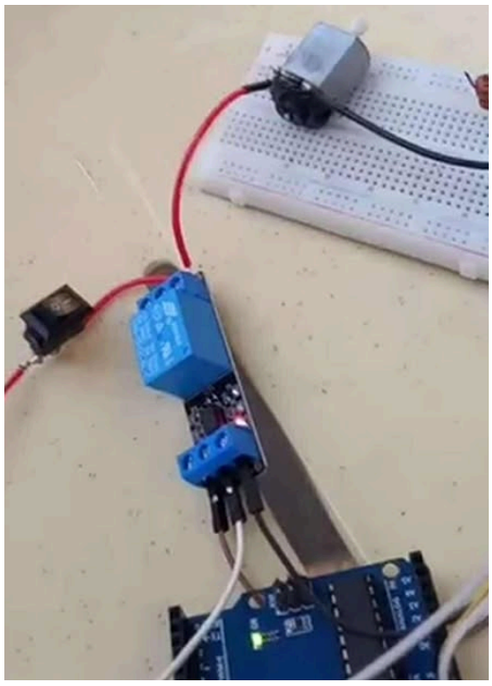


Actual Testing Of HELMOTECH:

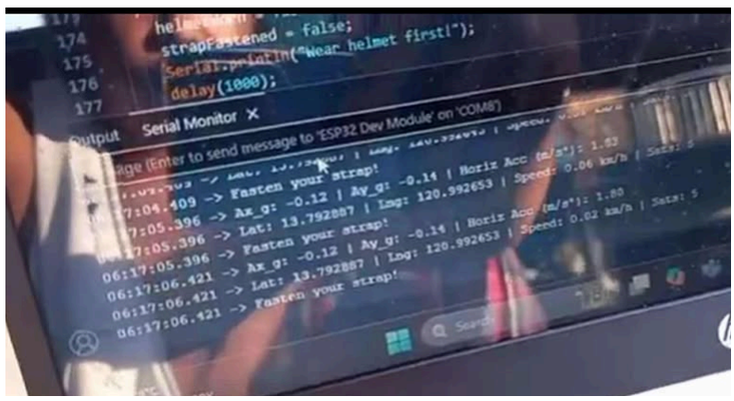
Helmet Worn



Strap Fastened



Motor Representation; Setup Ready



GPS Detection

References:

https://resources.pcb.cadence.com/blog/2019-how-ir-led-voltage-drop-effects-led-performance-and-life-cycle

https://www.teamwavelength.com/photodiode-basics/?srsltid=AfmBOoreqdZbWzLwnjszbvKKf7DTtM_Qnm-E5xU0KweELPQ9MYVVMdoO

https://robu.in/what-are-pull-up-and-pull-down-resistors/#:~:text=It%20ensures%20that%20the%20wire%20is%20at,prevent%20an%20undefined%20state%20at%20the%20input.

https://www.irjmets.com/uploadedfiles/paper/volume3/issue_3_march_2021/6597/1628083279.pdf

https://ctrfantennasinc.com/what-is-a-1-4-wave-antenna/

https://www.bitfoic.com/components/sim800l-gsm-module-description-pinout-features-and-how-to-use-sim800l-with-arduino?id=76

https://lastminuteengineers.com/sim800l-gsm-module-arduino-tutorial/

https://www.instructables.com/DC-DC-Boost-Converter-MT3608/

https://resources.altium.com/p/using-flyback-diodes-relays-prevents-electrical-noise-your-circuits

Whole system code (just need polishing)

```
// Include Libraries
#include <Wire.h>     // Enables I2C communication
#include <WiFi.h>       // ESP32 WiFi library to connect to hotspots
#include <I2Cdev.h>      // Simplifies I2C communication for MPU6050
#include <MPU6050.h>       // Library to interface with the MPU6050
#include <TinyGPSPlus.h>    // Parses/analyzes GPS data (NMEA sentences) from NEO-6M
#include <HardwareSerial.h>   // Enables use of extra UART ports on ESP32
#include <RH_ASK.h>     // RadioHead library for ASK RF communication(433 MHz)
#include <SPI.h>       // Required for RadioHead even if not using SPI directly

//Stores WiFi name and password for ESP32 to connect.
const char* ssid = "GoSurf 50k"; // WiFi (hotspot) name
const char* password = "04rae2003";

// Pin Definitions
#define GPS_RX_PIN 4  // GPS TX connects to GPIO4 (ESP32 receives here)
#define GPS_TX_PIN 14  // GPS RX conencts to GPIO14 (ESP32 transmits here)
#define MRS 2         // Reed switch pin for strap
#define IrRcvr 15     // IR receiver pin for helmet detection
#define MODEM_TX 18   // SIM800L TX pin
#define MODEM_RX 17  // SIM800L RX pin

// Global Variables
TinyGPSPlus gps; // GPS parser object
MPU6050 mpu; // MPU6050 object

// To avoid confusion on controller (due to combinations of components/sensors)
```

```
HardwareSerial SerialGPS(1);  // UART1 for GPS

HardwareSerial SerialAT(2);   // UART2 for SIM800L


float Lat = 0; // Holds the Lat value from GPS

float Lng = 0; // Holds the Lng value from GPS

float horizontal_speed = 0; // Holds the speed (mps) from GPS

int16_t ax, ay, az; // Raw values from MPU ax=acc along x-axis, ay=acc along y-axis, az=acc along z-axis

float ACC_THRESHOLD = -5.0;  // Threshold for deceleration in m/s²

float SPEED_THRESHOLD = 35.0 / 3.6; // speed in m/s


// To track the state of the helmet and system

bool helmetWorn = false;

bool strapFastened = false;

bool alreadySentSignal = false;

bool accidentDetected = false;


// Timing variables

unsigned long lastMpuCheck = 0;

unsigned long lastGsmCheck = 0;

unsigned long lastGpsPrint = 0;


const unsigned long MPU_INTERVAL = 1000; // Delay of 1sec for printing in serial mon

const unsigned long GSM_INTERVAL = 1000;

const unsigned long GPS_PRINT_INTERVAL = 1000;


RH_ASK driver(2000, 4, 5, 0); // 433 MHz Transmitter setup (2000bps, TX: 5, RX: 4(not really needed))
```

```
// GSM Setup
#define TINY_GSM_MODEM_SIM800 // Tells the TinyGSM library to use SIM800L
#define SerialMon Serial
#define TINY_GSM_DEBUG SerialMon // For debugging
#define ADMIN_NUMBER "+639476444391"

#include <TinyGsmClient.h>

#ifdef DUMP_AT_COMMANDS
#include <StreamDebugger.h> // Debugging tool, suitable for GSM mdoule
StreamDebugger debugger(SerialAT, SerialMon);
TinyGsm modem(debugger);
#else
TinyGsm modem(SerialAT);
#endif



void setup() {
  Serial.begin(115200);
  delay(1000);

  // Connect to wifi (hotspot)
  WiFi.begin(ssid, password); // Starts the process of connecting the ESP32 to the WiFi
  Serial.print("Connecting to WiFi");
  // Checks the WiFi connection status repeatedly.
  while (WiFi.status() != WL_CONNECTED) {
 delay(500);        // Check connection status again after 0.5sec
    Serial.print(".");
```

```
}
Serial.println("\nConnected to WiFi!"); // Printed once connected
Serial.print("IP Address: "); Serial.println(WiFi.localIP()); // Displays the local IP address

// Initialize GPS on UART1
SerialGPS.begin(9600, SERIAL_8N1, GPS_RX_PIN, GPS_TX_PIN);
delay(3000);
Serial.println("Initializing GPS...");

// Initialize MPU6050 (I2C)
Wire.begin(21, 22);  // SDA, SCL
mpu.initialize();
if (!mpu.testConnection()) {
  Serial.println("MPU6050 connection failed!");
} else {
  Serial.println("MPU6050 connected.");
}

// Initialize SIM800L on UART2
SerialAT.begin(9600, SERIAL_8N1, MODEM_TX, MODEM_RX);
delay(3000);
modem.restart();
if (modem.getSimStatus() != 1) { // For when SIM cant be detected
  Serial.println("SIM not ready.");
} else {
  Serial.println("GSM modem ready.");
}
```

```
  // Initialize 433 MHz Transmitter
  if (!driver.init()) {    // Checks if RF module is working and ready to send data
    Serial.println("433 MHz Transmitter init failed.");
  } else {
    Serial.println("RF Transmitter ready.");
  }

  // Set reed switch and IR as input
  pinMode(MRS, INPUT);
  pinMode(IrRcvr, INPUT);
  Serial.println("System Initialized");
}
void loop() {
  unsigned long currentMillis = millis();

  // To process GPS Data Continuously
  while (SerialGPS.available()) {
    char c = SerialGPS.read(); // Get NMEA sentences from GPS
    // Extract valid location/speed data (NMEA sentences).
    if (gps.encode(c)) {
      //Serial.println("GPS sentence parsed.");  // Debug line
      if (gps.location.isValid()) {
        Lat = gps.location.lat(); // Latitude
        Lng = gps.location.lng(); // Longitude

        if (gps.speed.isValid()) {
          horizontal_speed = gps.speed.mps(); // Get speed on GPS in m/s
        }
```

```
    }
  }
}



  // Debug info for location tracking
  if (currentMillis - lastGpsPrint >= GPS_PRINT_INTERVAL) {
    if (gps.location.isValid()) {
      Serial.print("Lat: "); Serial.print(Lat, 6); // Get and print latitude with 6 decimal value
      Serial.print(" | Lng: "); Serial.print(Lng, 6); // Get and print longitude with 6 decimal value
        Serial.print(" | Speed: "); Serial.print(horizontal_speed*3.6); // Get speed value, convert to km/h, then print
         Serial.print(" km/h | Sats: "); Serial.println(gps.satellites.value()); // To know how many satellites are present
    } else {
      Serial.println("Waiting for GPS fix...");
    }
    lastGpsPrint = currentMillis;
  }


  // Helmet and Strap Check
  checkHelmetAndStrap();


  // Check MPU6050
  if (currentMillis - lastMpuCheck >= MPU_INTERVAL) {
 getMpuData(); // Call MPU helper function
    lastMpuCheck = currentMillis;
  }
```

```
  // Accident Detection and SMS
  if (currentMillis - lastGsmCheck >= GSM_INTERVAL) {
    // Condition to satify before sending sms
    if (helmetWorn && strapFastened && !alreadySentSignal &&
        horizontal_speed >= SPEED_THRESHOLD && accidentDetected) {
      sendSMS();                    // Call sms function
      alreadySentSignal = true;
    }
    lastGsmCheck = currentMillis;
  }
}

// Helper function for the helmet and strap detection
void checkHelmetAndStrap() {
  // Read states for reed switch and IR
  int mrsState = digitalRead(MRS);
  int IrState = digitalRead(IrRcvr);

  if (mrsState == HIGH && IrState == LOW) {
    helmetWorn = true;
    strapFastened = true;
    Serial.println("Helmet worn, strap fastened.");
    driver.send((uint8_t *)"ON", 2); // Send signal to the RF receiver
    driver.waitPacketSent();
  } else if (mrsState == LOW && IrState == LOW) {
    helmetWorn = false;
    strapFastened = false;
    Serial.println("Fasten your strap!");
```

```
  } else if (mrsState == HIGH && IrState == HIGH) {
    helmetWorn = false;
    strapFastened = false;
    Serial.println("Wear helmet first!");
  }
}

// Helper function for MPU reading
void getMpuData() {
  mpu.getAcceleration(&ax, &ay, &az);

  // Convert Raw MPU data to gravitational force
   float ax_g = ax / 16384.0;  // Sensitivityn is 16384 LSB/g (Least Significant Bits per g) at ±2g range
  float ay_g = ay / 16384.0;
float horiz_g = sqrt(ax_g * ax_g + ay_g * ay_g); // Get horizontal acceleration
  float horiz_acc = horiz_g * 9.80665; // Convert to m/s^2

  Serial.print("Ax_g: "); Serial.print(ax_g);
  Serial.print(" | Ay_g: "); Serial.print(ay_g);
  Serial.print(" | Horiz Acc (m/s²): "); Serial.println(horiz_acc);

  if (horizontal_speed >= SPEED_THRESHOLD && horiz_acc < fabs(ACC_THRESHOLD)) {
    accidentDetected = true;
    Serial.println("Sudden deceleration detected, possible accident.");
  } else {
    accidentDetected = false;
  }
}
```

```
// Helper function for sending SMS
void sendSMS() {
  String message = "Alert: Accident detected! Location: " + String(Lat, 6) + ", " + String(Lng, 6);
  bool smsSent = modem.sendSMS(ADMIN_NUMBER, message.c_str());

  if (smsSent) {
    Serial.println("SMS sent successfully!");
  } else {
    Serial.println("Failed to send SMS.");
  }
}
```

Wokwi Project Link:

https://l.messenger.com/l.php?u=https%3A%2F%2Fwokwi.com%2Fprojects%2F428035554482440193&h=AT3tSajx6ENdGWyXa9WRW3zdEdDE1p2I-sPSiSNl6ynzY2jZCN8EYNOKXetNNVcsvIsCiSwvqWaQZVJ1vrZNSiPKwOKj36Hht_tv-H-QLyG27HtHeh4v_I07cT8TvsM