EMBEDDED SYSTEMS PROJECT REPORT

BLIND AID

GROUP MEMBERS (GROUP 39)

NAME	STUDENT NUMBER	REGISTRATION NUMBER
KAWUMA RODNEY	2400705564	24/U/05564/PS
TWEBAZE RODLIN	2400726022	24/U/26022/PS
MUGANZI BAKER KASIRYE	2400624227	24/U/24227/PS
MUKISA MARK	2400724248	24/U/24248/PSA
KYOMUHENDO SUMAYAH	2300710580	23/U/10580/PS

GITHUB LINK

https://github.com/CS-GROUP39/blind_aid.git

WEBSITE LINK

https://call-me-rodney.github.io/project-website-template/index.html

2. Abstract

- This project presents a wearable system designed to assist visually impaired individuals by detecting obstacles and fall events. It continuously measures surrounding object distance and relative movement using ultrasonic (HC-SR04) and time of flight (VL53L0X) sensors, refines speed estimates via the MPU6050's temperature-compensated motion data, and provides haptic feedback through a vibration motor. In case of a fall, a SIM800L cellular module sends an SMS alert with context to a caregiver. The system achieves real-time obstacle awareness and emergency notification, potentially enhancing user safety and independence.

3. Introduction

- **Background**
  Navigational aids for visually impaired individuals have advanced to include sensor-based feedback systems. However, many existing models lack integrated motion context and fall-alert capabilities.
- **Problem Statement**
  Current assistive wearables may fail to inform users about moving obstacles or fall risks in real time, limiting user safety during navigation.
- **Objectives**

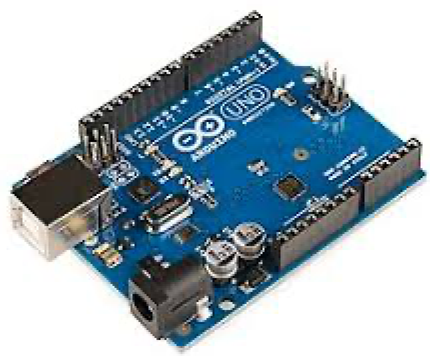
1. Detect objects at various ranges and compute their relative speed.
2. Provide intuitive vibration-based feedback correlating speed and proximity.
3. Automatically detect falls and dispatch emergency alerts.
4. Maintain system reliability across distance, motion, and temperature variations.
5. Make the system portable and power efficient.

Scope

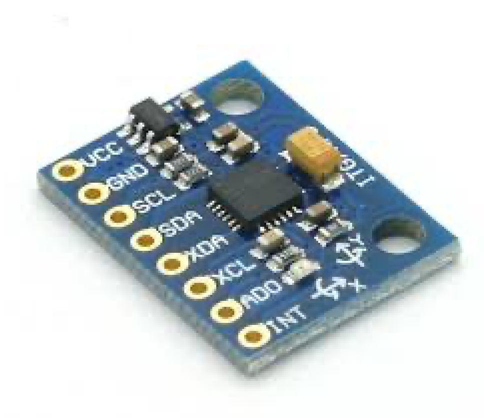
- Focus on short- and long-range object detection outdoors or in indoor public environments.
- Real-time performance with simple embedded hardware and accessible sensors.
- Use within a 30° field of view.

4. Materials and Tools

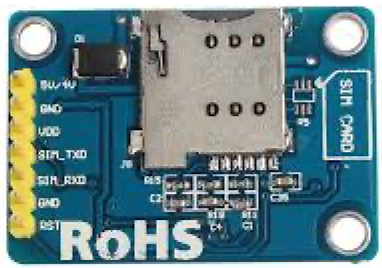
- List of hardware components:
  - Arduino Uno



  -
  - MPU6050(Gyroscope and Accelerometre)



  -
  - SIM800L EVB



-
- Power Source: We chose a rechargeable 2000mAh Li-ion battery for portability and efficiency.

-
- Ultrasonic Distance Sensor (HC-SR04): Used for initial object detection and proximity calculations. Provides a broader view of the immediate environment.

-

- ○ Time of flight sensor (VL53L0X): Paired with the ultrasonic distance sensor for more accurate close range proximity and distance calculations. Excels in scenarios where the HC-SR04 struggles such as detection of soft surfaces and narrow objects.

- ○
- ○ PWM Vibration Motor Module: To provide haptic feedback to the user. The strength and frequency will vary with object proximity, providing a strong but subtle alert when an obstacle lies ahead.

- ○
- ○ Jumper wires
- ○ 18650 battery shield
- ○ Breadboard

● Software tools:

- ○ C++

- Arduino_IDE

5. Design and Implementation

Overview

The HC-SR04 is used to detect objects greater than 2 meters away. The distance data it returns is also used to calculate the incoming object's speed by measuring change in distance over time. The MPU6050's temperature sensor provides temperature data to refine the speed calculations made using the HC-SR04's data. At distances of 2 meters or less, the VL53L0X is used to perform distance calculations and utilize its data to calculate object speed. If the VL53L0X encounters an error or the object is out of range, the system falls back to the HC-SR04. In both cases, the relative speed of the object in the person's path (30° field of view) is calculated, using the MPU6050 to determine the wearer's movement and adjust accordingly. The vibration motor has two types of vibration queues: pulsed patterns to signal incoming object speed and continuously increasing strength to indicate proximity. If the object is stationary, this is conveyed through a continuous vibration that strengthens as it gets closer. Faster-moving objects yield faster pulse rates; closer objects increase vibration intensity. When a drastic change in angular acceleration, tilt, and speed is detected within a small time window, the system identifies this as a potential fall. The SIM800L module then sends an SMS alert to the wearer's caretaker, including any available object proximity data to clarify whether the fall was self-caused or due to collision.

Hardware Components

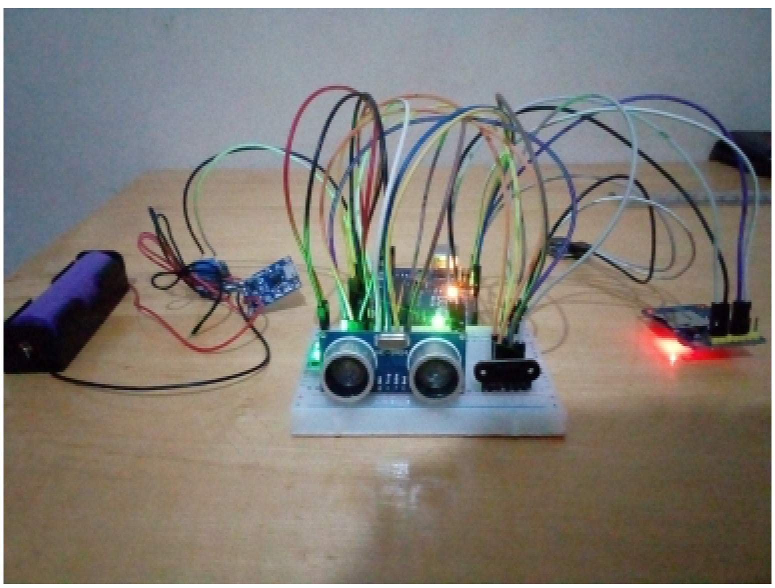
- **HC-SR04 Ultrasonic Sensor** – Long-range detection beyond 2 meters
- **VL53L0X Time of Flight Module** – Accurate short-range (< 2 m) sensing
- **MPU6050 IMU** – Measures acceleration, angular rate, and temperature for speed calibration and fall detection
- **PWM Vibration Motor** – Provides haptic feedback encoding proximity and movement

- **SIM800L GSM Module** – Mobile connectivity to send fall alerts via SMS

Software Design

- **Pseudocode**: Sequence sensor reads, calculate distance & relative motion, generate feedback patterns, monitor IMU state changes, trigger alert logic.
- **Sensor Logic**: Use VL53L0X when available; otherwise fallback to HC-SR04. Adjust speed calculation using MPU6050 temperature data.
- **Fall-Detection Algorithm**: Track short-term spikes in tilt or acceleration beyond threshold; discriminate between normal motion and potential fall.

- **Hardware Design**:



    - The HC-SR04 will be used to detect objects greater than 2 meters away, the distance data it returns will be used also to calculate the incoming object's speed by measuring change in distance over time.
    - The MPU6050's temperature sensor will provide temperature data to refine the speed calculations made using the HC-SR04's data. At distances of 2

meters or less, use the VL53L0X to perform distance calculations and utilize its data to calculate object speed. If the VL53L0X encounters an error or the object is out of range, fall back to the HC-SR04. In both cases, the relative speed of the object in the person's path (30 FOV) will be calculated, using the MPU6050 to get the wearer's speed to make the calculation.

  - The vibration motor will have two types of vibration queues: pulses to indicate the incoming object's speed using the relative speed calculations, and vibration strength to convey proximity. If the object is stationary, convey this through a continuous vibration that grows stronger with proximity. The faster the object, the faster the pulses, the closer the object, the stronger the vibrations. Once a drastic change in angular acceleration, tilt and speed is detected over a small time window, identify this as a potential fall. Trigger the SIM800L to send an SMS alert to the wearer's caretaker to alert them of the fall. Send any object proximity data if available to give context to whether the fall was on their own accord, or as a result of a collision with an object
  -

- **Software Design**:
- **<u>BLIND AID PRODUCT'S PSEUDOCODE.</u>**

// Constants

Define TRIG_PIN as 6

Define ECHO_PIN as 7

Define MAX_DISTANCE as 400 cm

Define VIB_PIN as 9

Define SIM_RX as 11

Define SIM_TX as 10

Define DISTANCE_THRESHOLD as 200.0 cm

Define MIN_DISTANCE as 2.0 cm

Define MAX_VIB_STRENGTH as 255.0

Define MIN_VIB_STRENGTH as 50.0

```
Define SAMPLE_INTERVAL as 60 ms
Define FREE_FALL_THRESHOLD as 2.0 m/s²
Define IMPACT_THRESHOLD as 29.4 m/s²
Define TILT_THRESHOLD as 45.0 degrees
Define FALL_WINDOW as 2000 ms
Define SMS_COOLDOWN as 60000 ms
Define CAREGIVER_NUMBER as "+256123456789"

// Global Variables
ax_offset = 0.0
gx_offset = 0.0
lastDistance = 0.0
lastTime = 0
wearerSpeed = 0.0
fallDetected = false
lastSMSTime = 0

// Setup Function
Function setup:
  Initialize serial communication at 9600 baud
  Set VIB_PIN as output

  Initialize VL53L0X sensor
  If VL53L0X initialization fails:
    Print "VL53L0X init failed"
  Set VL53L0X measurement timing budget to 60000 µs
```

```
  Initialize MPU6050 sensor
  If MPU6050 initialization fails:
    Print "MPU6050 init failed"
    Halt program
  Set MPU6050 accelerometer range to ±2g
  Set MPU6050 gyroscope range to ±250°/s
  ax_offset = Call ax_calibrate
  gx_offset = Call gx_calibrate

  Initialize SIM800L at 9600 baud
  Wait 1000 ms
  If initializeSIM800L fails:
    Print "SIM800L init failed"

// Calibrate Accelerometer X-axis
Function ax_calibrate:
  ax_sum = 0.0
  For i from 0 to 999:
    Read MPU6050 accelerometer, gyroscope, temperature
    ax_sum += accelerometer x-axis value
    Wait 5 ms
  Return ax_sum / 1000.0

// Calibrate Gyroscope X-axis
Function gx_calibrate:
```

```
  gx_sum = 0.0
  For i from 0 to 999:
    Read MPU6050 accelerometer, gyroscope, temperature
    gx_sum += gyroscope x-axis value
    Wait 5 ms
  Return gx_sum / 1000.0

// Initialize SIM800L
Function initializeSIM800L:
  Send "AT" command to SIM800L
  Wait 100 ms
  If response contains "OK":
    Send "AT+CMGF=1" to set SMS text mode
    Wait 100 ms
    If response contains "OK":
      Return true
  Return false

// Send SMS Alert
Function sendSMS(distance):
  If current time - lastSMSTime < SMS_COOLDOWN:
    Return
  Create message buffer (50 bytes)
  Format message: "Fall! Dist: <distance> cm"
  Send "AT+CMGS=\"<CAREGIVER_NUMBER>\""
  Wait 100 ms
```

```
  Send message
  Wait 100 ms
  Send Ctrl+Z (ASCII 26)
  Wait 1000 ms
  If response contains "OK":
    Print "SMS sent"
  lastSMSTime = current time

// Get Temperature from MPU6050
Function getTemperature:
  Read MPU6050 accelerometer, gyroscope, temperature
  Return temperature in °C

// Calculate Wearer Speed
Function getWearerSpeed:
  Read MPU6050 accelerometer, gyroscope, temperature
  ax = accelerometer x-axis value - ax_offset
  filtered_ax = 0.1 * ax + 0.9 * filtered_ax
  velocity = 0.0 (static)
  lastUpdate = 0 (static)
  now = current time
  If absolute(filtered_ax) < 0.1:
    velocity = 0.0
  Else if lastUpdate != 0:
    dt = (now - lastUpdate) / 1000.0
    velocity += filtered_ax * dt
```

```
    Constrain velocity to [-2.0, 2.0]
  lastUpdate = now
  Return velocity


// Detect Fall
Function detectFall:
  fallStartTime = 0 (static)
  freeFallDetected = false (static)
  impactDetected = false (static)
  total_tilt = 0.0 (static)
  Read MPU6050 accelerometer, gyroscope, temperature
  acc_magnitude = sqrt((accelerometer.x - ax_offset)² + accelerometer.y² + accelerometer.z²)
  tilt = absolute(gyroscope.x - gx_offset) * (SAMPLE_INTERVAL / 1000.0) * 180.0 / PI
  total_tilt += tilt
  If not freeFallDetected and acc_magnitude < FREE_FALL_THRESHOLD:
    freeFallDetected = true
    fallStartTime = current time
  Else if freeFallDetected and not impactDetected and acc_magnitude >
IMPACT_THRESHOLD:
    impactDetected = true
  Else if freeFallDetected and impactDetected and total_tilt > TILT_THRESHOLD:
    If current time - fallStartTime ≤ FALL_WINDOW:
      freeFallDetected = false
      impactDetected = false
      total_tilt = 0.0
      Return true
```

```
  If freeFallDetected and current time - fallStartTime > FALL_WINDOW:
    freeFallDetected = false
    impactDetected = false
    total_tilt = 0.0
  Return false

// Get Distance
Function getDistance:
  distance = -1.0
  If lastDistance ≤ DISTANCE_THRESHOLD:
    Read VL53L0X measurement
    If measurement is valid:
      distance_cm = measurement in mm / 10.0
      If distance_cm ≥ MIN_DISTANCE and distance_cm ≤ DISTANCE_THRESHOLD:
        distance = distance_cm
  If distance < 0:
    numReadings = 5
    total Distance = 0.0
    validReadings = 0
    For i from 0 to numReadings-1:
      duration = HC-SR04 ping
      temperature = getTemperature
      speedOfSound = 331.4 + (0.606 * temperature)
      distance_cm = (duration * (speedOfSound / 10000)) / 2
      If distance_cm ≥ MIN_DISTANCE and distance_cm ≤ MAX_DISTANCE:
        totalDistance += distance_cm
```

```
      validReadings += 1
    Wait 5 ms
  If validReadings > 0:
    distance = totalDistance / validReadings
  Return distance


// Calculate Object Speed
Function calculateObjectSpeed(currentDistance, deltaTime):
  If lastDistance = 0.0 or deltaTime = 0:
    Return 0.0
  distanceChange = lastDistance - currentDistance
  Return (distanceChange / (deltaTime / 1000.0)) / 100.0


// Vibration Feedback
Function vibrateFeedback(distance, relativeSpeed):
  If distance < 0:
    Set vibration motor to 0
    Return
  strength = map distance from [MIN_DISTANCE, DISTANCE_THRESHOLD] to
[MAX_VIB_STRENGTH, MIN_VIB_STRENGTH]
  Constrain strength to [MIN_VIB_STRENGTH, MAX_VIB_STRENGTH]
  If absolute(relativeSpeed) < 0.1:
    Set vibration motor to strength
  Else:
    absSpeed = absolute(relativeSpeed)
    pulseDelay = map absSpeed*100 from [0, 200] to [500, 50]
```

```
    Constrain pulseDelay to [50, 500]
    Set vibration motor to strength
    Wait pulseDelay / 2
    Set vibration motor to 0
    Wait pulseDelay / 2

// Main Loop
Function loop:
  currentTime = current time
  If currentTime - lastTime ≥ SAMPLE_INTERVAL:
    distance = getDistance
    wearerSpeed = getWearerSpeed
    deltaTime = currentTime - lastTime
    objectSpeed = calculateObjectSpeed(distance, deltaTime)
    relativeSpeed = objectSpeed - wearerSpeed
    If detectFall:
      fallDetected = true
      sendSMS(distance)
    If not fallDetected:
      vibrateFeedback(distance, relativeSpeed)
    lastDistance = distance
    lastTime = currentTime
```

Implementation of Required Functionalities

- **Circuit Diagrams**: Include wiring to microcontroller (Arduino Uno).
- **Sensor Integration**: Calibration method for ultrasonic and time of flight, temperature compensation steps.
- **Microcontroller Setup**: Pin configuration, power management.
- **Code Snippets**:
  - Distance read + filter
  - Speed estimation: `speed = (d1 − d0)/Δt`
  - Vibration pattern generator
  - Fall-detection and SMS send routine

Challenges faced

- Sensor latency and ambient noise interference with HC-SR04
- Temperature dependency causing speed estimate drift
- Syncing TOF and ultrasonic sensor readings for accurate vibration queues
- Power efficiency and battery life optimization.
- Managing GSM connectivity and supplying sufficient power and current.
- Arduino Uno limited computing resources.
- Narrow field of view for both sensors.

Conclusion

This system successfully integrates multiple sensors to deliver real-time obstacle and fall detection while conveying intuitive haptic feedback. Initial testing demonstrates responsiveness and usability.

Future Recommendations.

- Integrate GPS for geo-tagging fall alerts and 3D route tracing.
- Using a more compact and efficient LI-Po battery.
- Explore waterproof enclosures for outdoor use
- Upgrade vibration motor for adjustable tactile patterns

- Expand field-of-view with sensor arrays (up to 90 degrees)

7. References

- https://lastminuteengineers.com/28byj48-stepper-motor-arduino-tutorial/
- David Russell. *Introduction to Embedded Systems.* 2nd edition. Morgan and Claypool Publishers, 2010.
- Wayne Wolf. *Computers as components.* 2nd edition. Morgan Kaufmann Publishers, 2008.