Blind School Smart Ecosystem

Contents:

Smart Ecosystem for Blind School

Problem statement:
Unlike regular schools, schools for specially abled posses a plethora of special challenges like ensuring physical safety of students, psychological well being and efficient teaching methods. These challenges can be solved using modern technologies like IoT, embedded systems and technology assisted teaching.

Objectives:
Our objective for this project is to effectively utilize the technology, resources and tools taught to us in the course of DPIoT.
Smart Ecosystem for Blind School aims at achieving the following objectives:

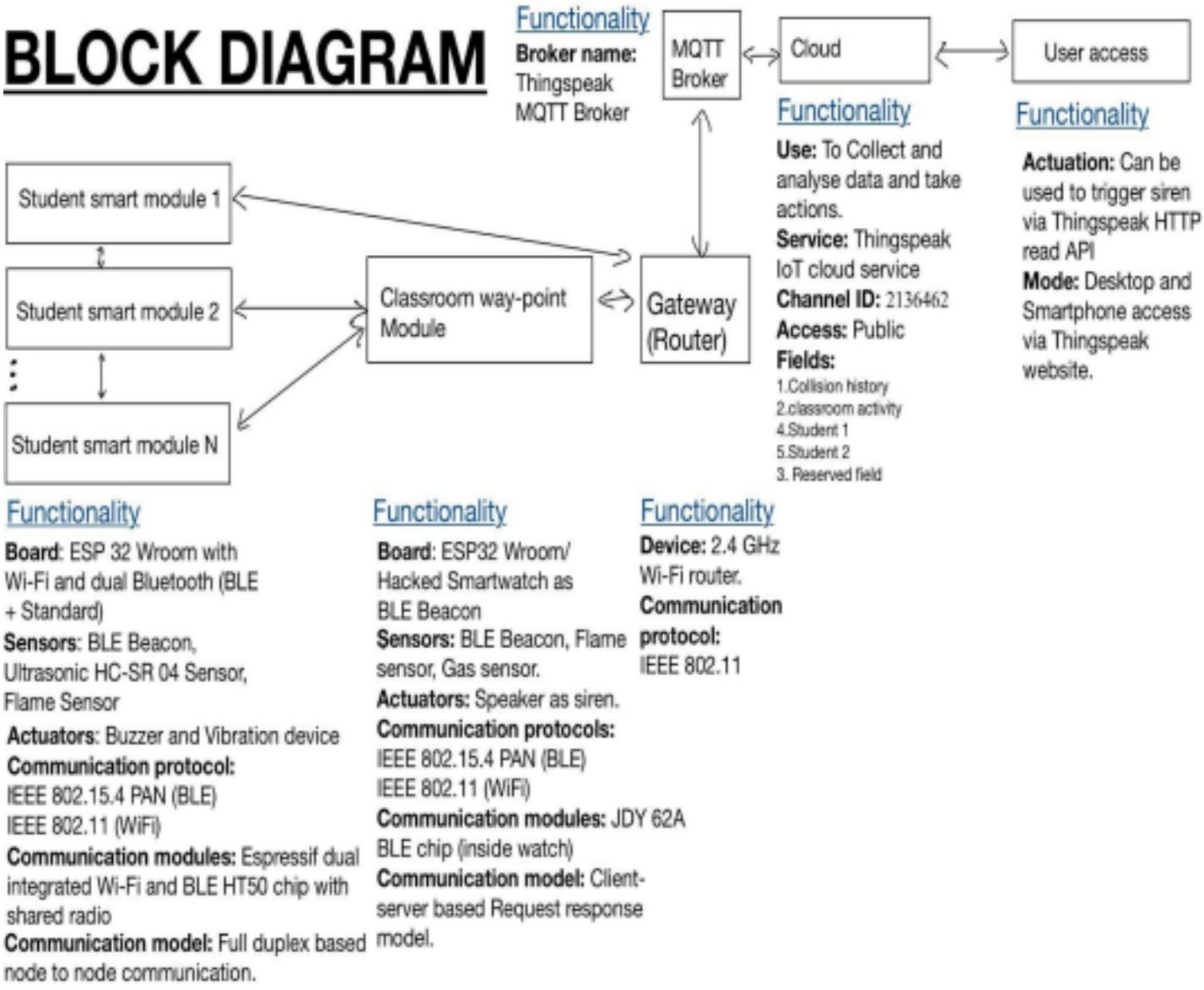
1. Collision prevention by intercommunication between smart glasses worn by each student using short range BLE mesh.

2. Position tracking of students using waypoints to monitor students and detect any accidents or missing students.

3. Navigation assistance by intercommunication between student nodes.

4. Assisting analysis of collected student data via MQTT broker on

Thingspeak IoT platform.



Dataflow

Data Collection Module:

Objective:

- Collection of each node's proximity data
- Collection of accurate distance from nearby physical objects
- Identification of nearby flame
- Identification of relative position with respect to classroom

Data:

- BLE Beacon for distance measurement
- Ultrasonic Sensor (HC-SR04) for proximity detection
- Flame Sensor for flame detection

Algorithm:

- Sensor information is collected via pins 12 and 13 to input data from ultrasonic sensor and BLE beacon RSSI (Relative Signal Strength Indicator) using the integrated WiFi and Bluetooth module on ESP32.

Communication Module:

Objective:

- Implementation of node-to-node communication on student level
- Fast response time to prevent collisions in time
- Establishing a reliable connection to the cloud

Data:

- IEEE 802.15.4 (BLE)
- IEEE 802.11 (WiFi)
- Mesh Topology
- Star Topology

Algorithm:

- BLE based server is established which advertises its existence under user defined services (characteristics).
- All clients scan for advertised servers.
- The ones in range are identified and connected to through a time slicing based scheduling algorithm. Each server/client acts as client/server in the next time slice to send data in a full duplex node-to-node manner.
- Using this sensor data is communicated between nodes.
- Each student node is connected to classroom node via client/server model utilising BLE to implement star topology

Data Processing Module:

Objective:

- The processing of data to be done in a timely manner in order to prevent any starvation
- Event scheduling to be done to share single radio antenna for WiFi and BLE. ● Memory management

Data:

- Tensilica Xtensa LX6 Microprocessor

Algorithm:

- Event handling is done using a time slicing based algorithm available in Espressif event handler for ArduinoIDE
- Expansion of internal OAT storage by changing partitioning of main memory to Huge APP (3MB)

Gateway Module:

Objective:

- Routing of data to the MQTT Broker of ThingSpeak

Data:
- 2.4 GHz WiFi Router

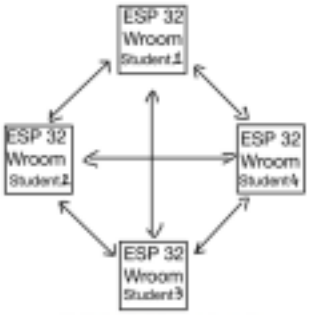
Cloud:
Objective:
- Fast response time of subscribed data

Data:
- ThingSpeak Cloud

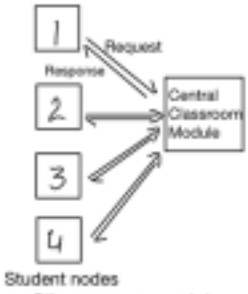
User Access Module:
Objective:
- Data should be presented in an user friendly and interactive manner

Data:
- Public URL: https://thingspeak.com/channels/2136462
- Channel ID: 2136462
- No. of fields = 4

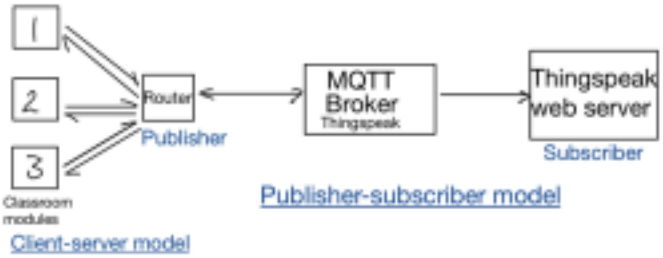
Communication model:



Full duplex based node to node communication using mesh topology

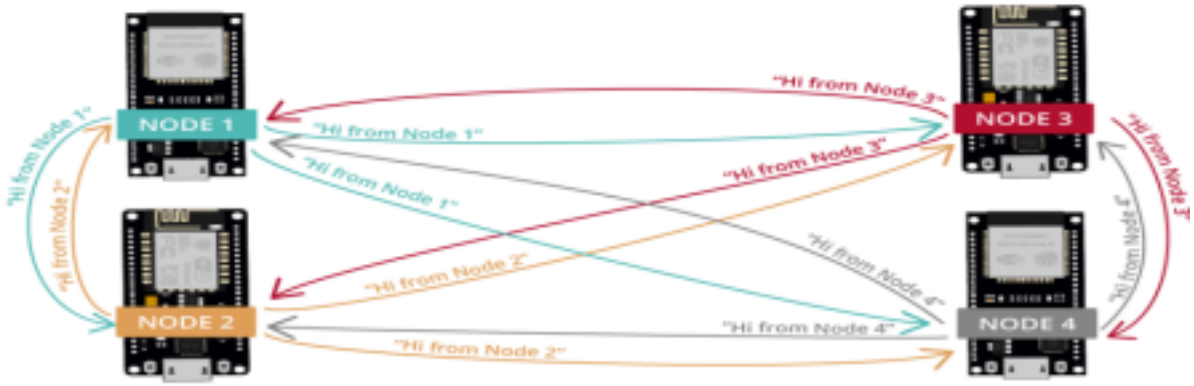


Client-server model

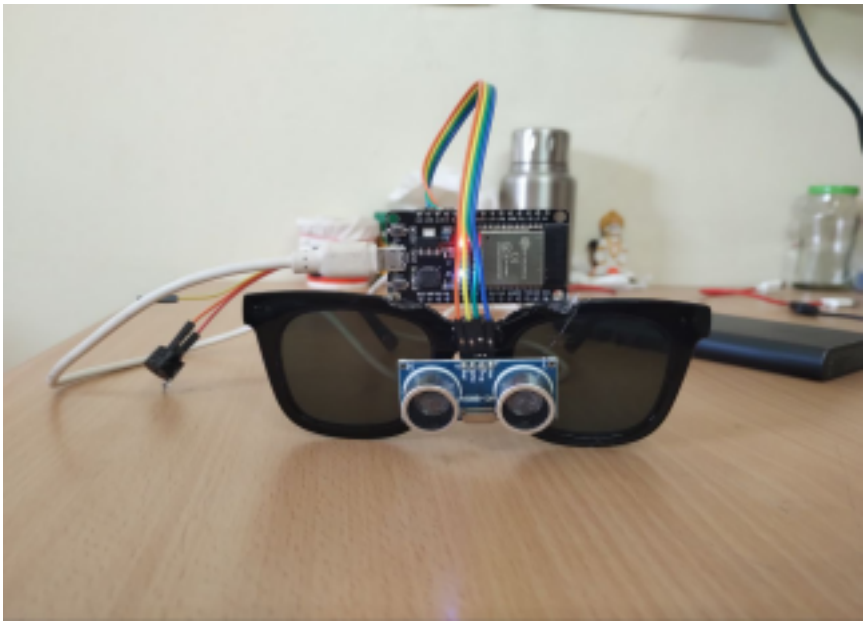


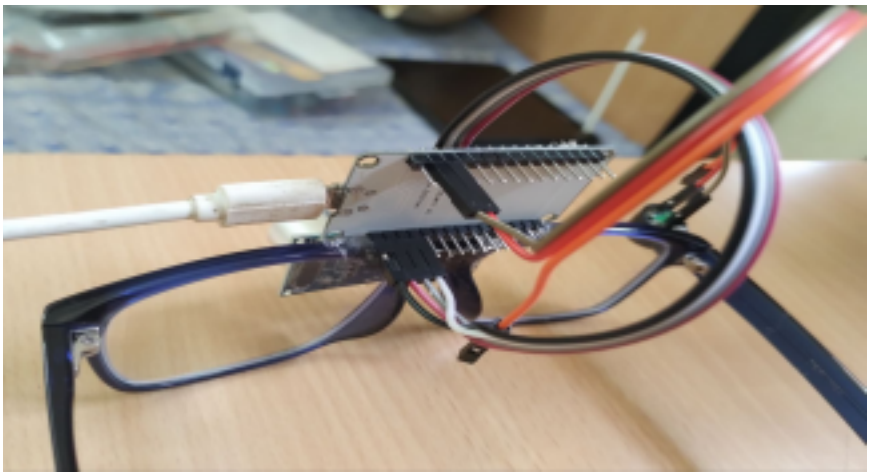
Client-server model

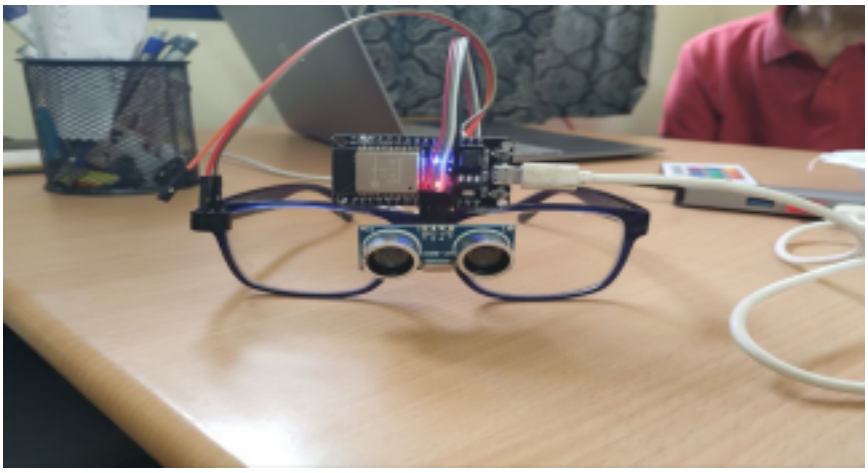
Publisher-subscriber model



Implementation:







Student module code:

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEScan.h>
#include <BLEAdvertisedDevice.h
#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b" #define
CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8" #define
buzzer 15
int beaconInRange = -1;
int scanTime = 2; // In seconds

BLEScan *pBLEScan;

class MyAdvertisedDeviceCallbacks : public BLEAdvertisedDeviceCallbacks
{
   void onResult(BLEAdvertisedDevice advertisedDevice)
   {
     if (advertisedDevice.getName() == "BSW004")
     {
```

```
        int ble_rssi = advertisedDevice.getRSSI();
        Serial.println("RSSI: " + String(ble_rssi));
        Serial.println(advertisedDevice.getAddress().toString().c_str());

        if (ble_rssi > -85 && ble_rssi < -60)
        {
          beaconInRange = 0;
        }
        else if (ble_rssi > -60 && ble_rssi < -40)
        {
          beaconInRange = 1;
        }
        else if (ble_rssi > -40)
        {
          beaconInRange = 2;
        }
      }
    }
};
void setup()
{
  Serial.begin(115200);

  Serial.println("Starting BLE work!");

  BLEDevice::init("student1");

  BLEServer *pServer = BLEDevice::createServer();
   BLEService *pService = pServer->createService(SERVICE_UUID);
  BLECharacteristic *pCharacteristic = pService->createCharacteristic(
                                         CHARACTERISTIC_UUID,
                                             BLECharacteristic::PROPERTY_READ |
                                             BLECharacteristic::PROPERTY_WRITE);

  pCharacteristic->setValue("student1");
  pService->start();

  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();

  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(true);
```

```
  pAdvertising->setMinPreferred(0x06); // functions that help with iPhone
connections issue
  pAdvertising->setMinPreferred(0x12);

  BLEDevice::startAdvertising();

  Serial.println("Characteristic defined!");
  pBLEScan = BLEDevice::getScan(); //create new scan
  pBLEScan->setAdvertisedDeviceCallbacks(new
MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true); //active scan uses more power, but
get results faster
  pBLEScan->setInterval(10);
  pBLEScan->setWindow(9); // less or equal setInterval value
  pinMode(buzzer, OUTPUT); // set pin to output mode

}
void loop()
{
  BLEScanResults foundDevices = pBLEScan->start(scanTime, false);

  Serial.println("Flag: " + String(beaconInRange));
  pBLEScan->clearResults(); // delete results fromBLEScan buffer to release
memory
  if (beaconInRange == 1)
  {
    for (int i = 0; i < 20; i++)
    {
      digitalWrite(buzzer, HIGH);
      delay(40);
      digitalWrite(buzzer, LOW);
      delay(40);
    }
  }
  beaconInRange = -1;

}
```

Classroom module code:

```
#include <Arduino.h>
#include <BLEDevice.h>
```

```
#include <BLEUtils.h>
#include <BLEScan.h>
#include "ThingSpeak.h"
#include <BLEAdvertisedDevice.h>
#include "WiFi.h"
#define trigPin 13
#define echoPin 12
#define buzzer 15
#define SOUND_SPEED 0.034
int deviceConnect = 0;
long duration;
int scanTime = 4; // In seconds
BLEScan *pBLEScan;
int c1; //COUNTER for collision
int c2; //Keeps track of movement in-out
//int cbuffer=0; //This helps debug classroom in-out errors
int prev = 1; //To store previous position of student
int classdetect = -1;
WiFiClient client;
unsigned long myChannelNumber = 1; //Thingspeak channel no. const char *
myWriteAPIKey = "5AXVK8IQ6OLNKMLQ"; //Thingspeak WRITE API KEY

class MyAdvertisedDeviceCallbacks : public BLEAdvertisedDeviceCallbacks
{
    void onResult(BLEAdvertisedDevice advertisedDevice)
    {
      classdetect = -1;
     // deviceConnect = 0; //This keeps check on no. of devices
      int ble_rssi;
      if (advertisedDevice.getName() == "student1")
      {
      // deviceConnect++;
        ble_rssi = advertisedDevice.getRSSI();
        Serial.println("RSSI: " + String(ble_rssi));
        Serial.println("Student1 detected beware of collision");

        if (ble_rssi >= -60)
        {
          digitalWrite(trigPin, LOW);
          delayMicroseconds(2);
          // Sets the trigPin on HIGH state for 10 microseconds
```

```
          digitalWrite(trigPin, HIGH);
          delayMicroseconds(10);
          digitalWrite(trigPin, LOW);
          duration = pulseIn(echoPin, HIGH);
          int dist = duration * SOUND_SPEED / 2;
          if (dist < 100)
          {
            c1++;
            Serial.println("COLLISION!!");
            for (int i = 0; i < 150; i++)
            {
              digitalWrite(buzzer, HIGH);
              delay(5);
              digitalWrite(buzzer, LOW);
              delay(5);
            }
            ThingSpeak.setField(1, c1);
            int x1 = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
            if (x1 == 200) {
              Serial.println("Channel update successful.");
            }
            else {
              Serial.println("Problem updating channel. HTTP error code " +
String(x1));
            }
            for (int i = 0; i < 120; i++)
            {
              digitalWrite(buzzer, HIGH);
              delay(5);
              digitalWrite(buzzer, LOW);
              delay(5);
            }
            delay(500);
          }
        }
        //Otherwise do nothing
      }
      if (advertisedDevice.getName() == "BSW004")
      {
       // deviceConnect++;
        ble_rssi = advertisedDevice.getRSSI();
```

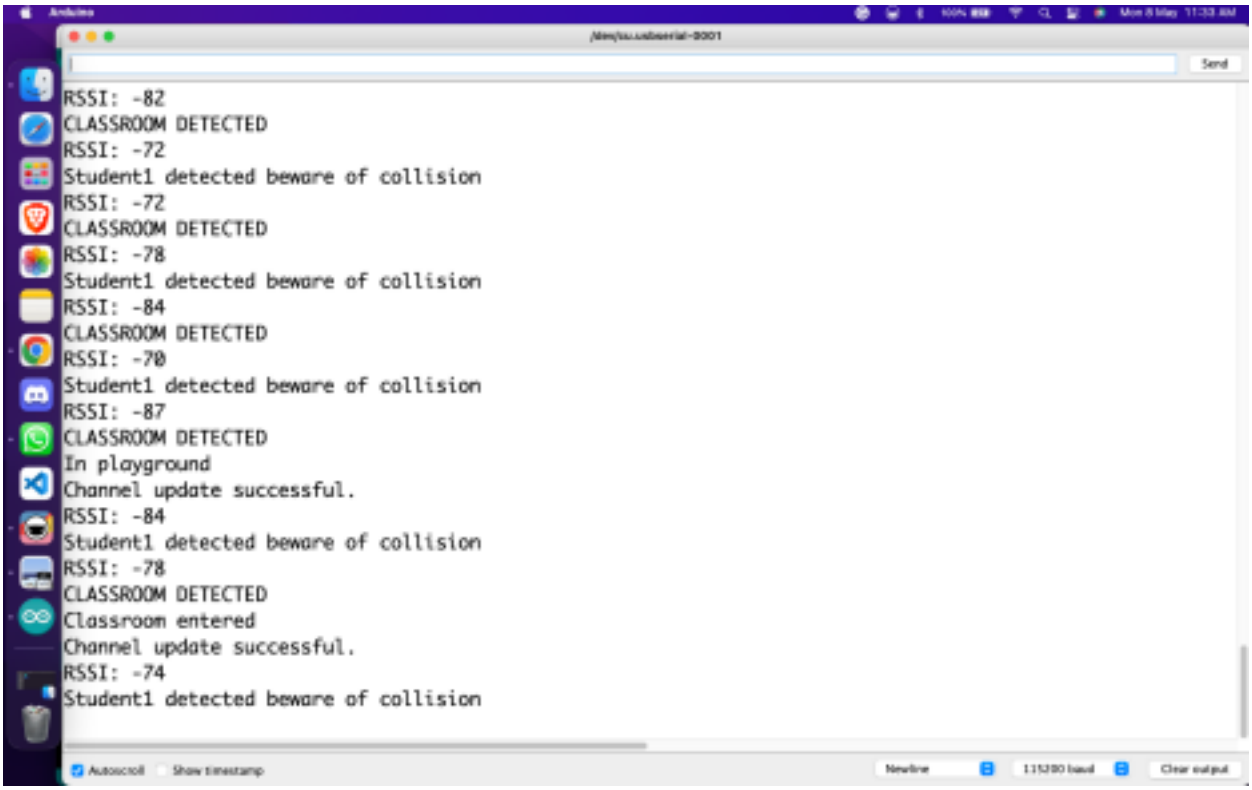
```
        Serial.println("RSSI: " + String(ble_rssi));
        Serial.println("CLASSROOM DETECTED");
        classdetect = 1;
        if (ble_rssi > -85 && prev == 1)
        {
          c2++;
          Serial.println("Classroom entered");
          ThingSpeak.setField(2, c2);
          ThingSpeak.setField(5, 1);
          int x2 = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
          if (x2 == 200) {
            Serial.println("Channel update successful.");
           // cbuffer=1;
          }
          else {
            Serial.println("Problem updating channel. HTTP error code " +
String(x2));
          }

          for (int i = 0; i < 20; i++)
          {
            digitalWrite(buzzer, HIGH);
            delay(40);
            digitalWrite(buzzer, LOW);
            delay(40);
          }
          prev = 0;
          delay(500);
        }
        if (ble_rssi < -85 && prev == 0)
        {
          c2++;
          Serial.println("In playground");
          ThingSpeak.setField(2, 0);
          ThingSpeak.setField(5, 0);
          int x2 = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
          if (x2 == 200) {
            Serial.println("Channel update successful.");
          }
          else {
            Serial.println("Problem updating channel. HTTP error code " +
```

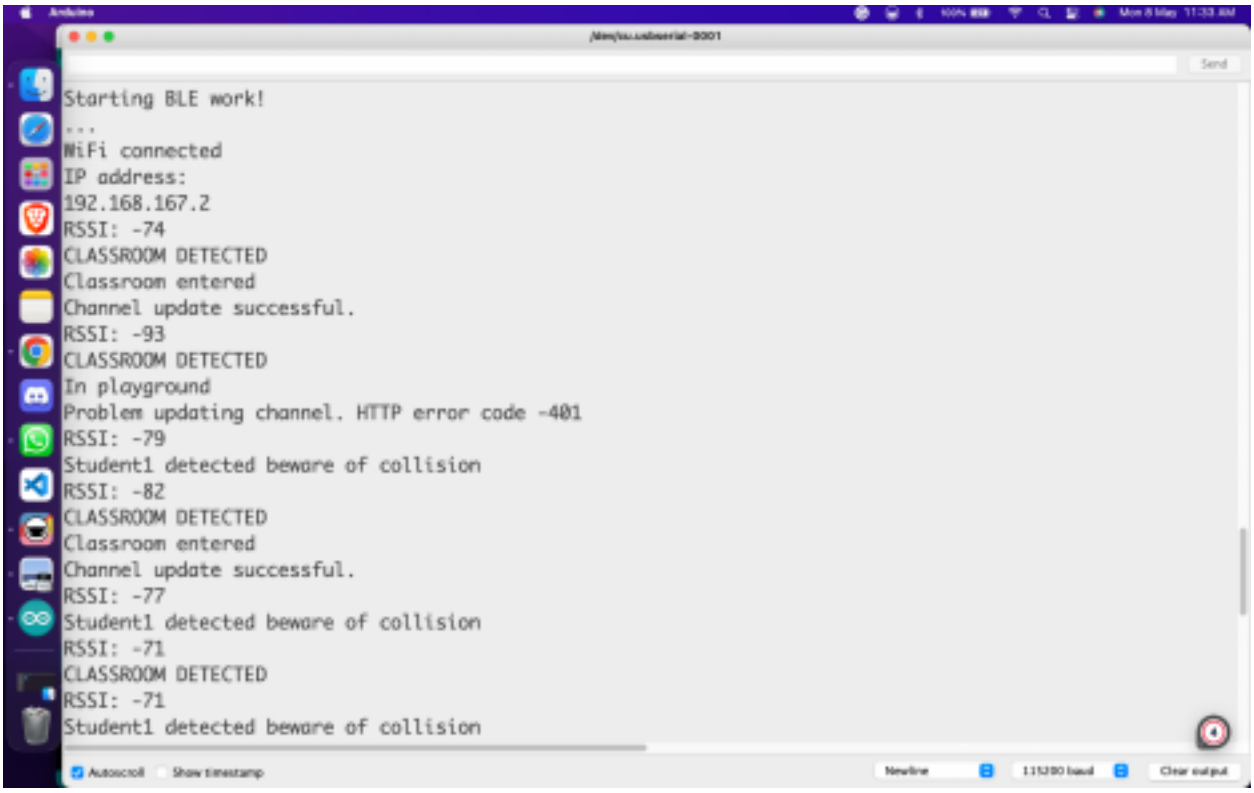
```
String(x2));
          }
          prev = 1;
          delay(2000);
        }
      }
      /*
      if (classdetect != 1 && prev!=1)
      {
        c2++;
        Serial.println("In playground");
        ThingSpeak.setField(2, 0);
        ThingSpeak.setField(5, 0);
        int x2 = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
        if (x2 == 200) {
          Serial.println("Channel update successful.");
        }
        else {
          Serial.println("Problem updating channel. HTTP error code " + String(x2));
        }
        prev = 1;
        delay(2000);
      }
      */
    }
};


 static void my_gap_event_handler(esp_gap_ble_cb_event_t event,
esp_ble_gap_cb_param_t* param) {
   ESP_LOGW(LOG_TAG, "custom gap event handler, event: %d",
(uint8_t)event);
 }

 static void my_gattc_event_handler(esp_gattc_cb_event_t event, esp_gatt_if_t
gattc_if, esp_ble_gattc_cb_param_t* param) {
   ESP_LOGW(LOG_TAG, "custom gattc event handler, event:
%d", (uint8_t)event);
 }
```

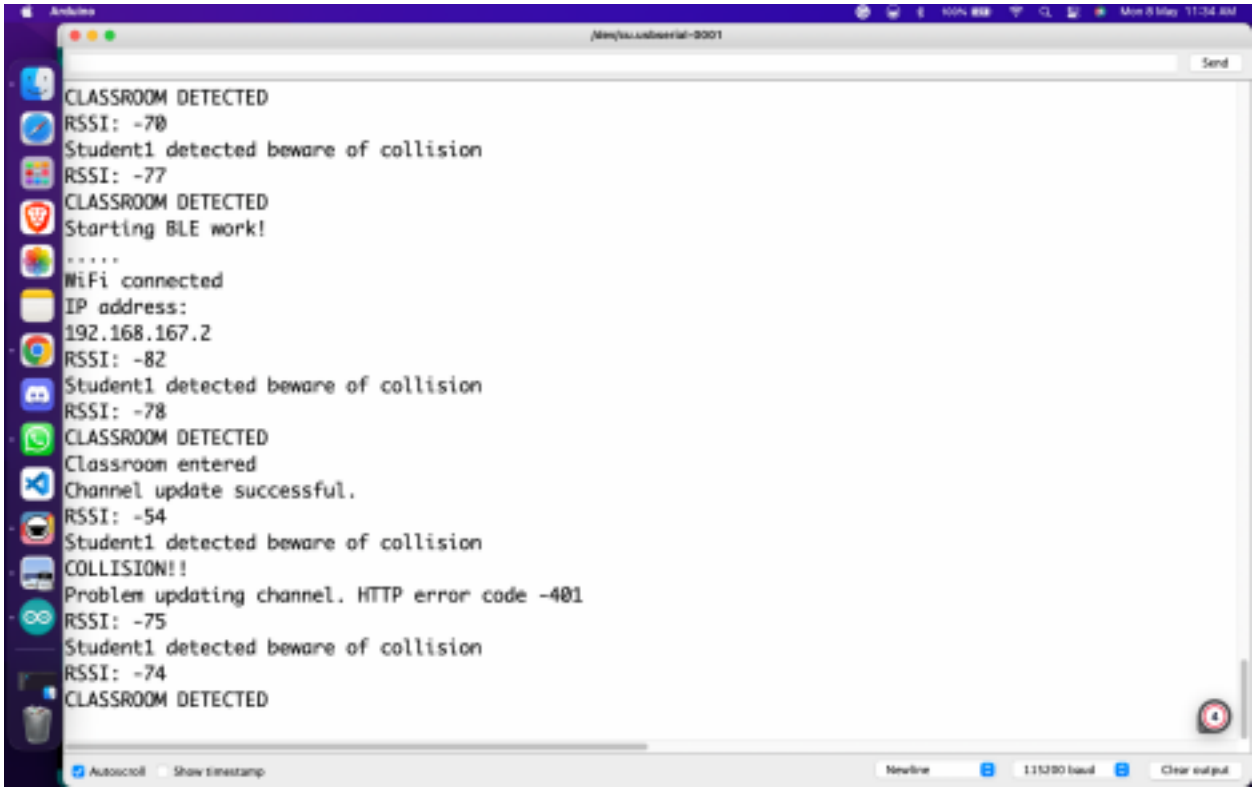
```
 static void my_gatts_event_handler(esp_gatts_cb_event_t event, esp_gatt_if_t
gattc_if, esp_ble_gatts_cb_param_t* param) {
   ESP_LOGW(LOG_TAG, "custom gatts event handler, event:
%d", (uint8_t)event);
 }

 void setup() {
   Serial.begin(115200);
   pinMode(trigPin, OUTPUT);
   pinMode(echoPin, INPUT);
   pinMode(buzzer, OUTPUT);
   c1 = 0;
   c2 = 0;
   WiFi.mode(WIFI_STA);
   ThingSpeak.begin(client);
   Serial.println("Starting BLE work!");
   WiFi.begin("NagendraC", "********************");// removed for security reasons
   while (WiFi.status() != WL_CONNECTED) {
     delay(500);
     Serial.print(".");
   }

   Serial.println("");
   Serial.println("WiFi connected");
   Serial.println("IP address: ");
   Serial.println(WiFi.localIP());

   BLEDevice::setCustomGapHandler(my_gap_event_handler);
   BLEDevice::setCustomGattsHandler(my_gatts_event_handler)
   ;
   BLEDevice::setCustomGattcHandler(my_gattc_event_handler)
   ;

   BLEDevice::init("");
   pBLEScan = BLEDevice::getScan(); //create new scan
   pBLEScan->setAdvertisedDeviceCallbacks(new
MyAdvertisedDeviceCallbacks());
   pBLEScan->setActiveScan(true); //active scan uses more power, but
get results faster
   pBLEScan->setInterval(10);
```

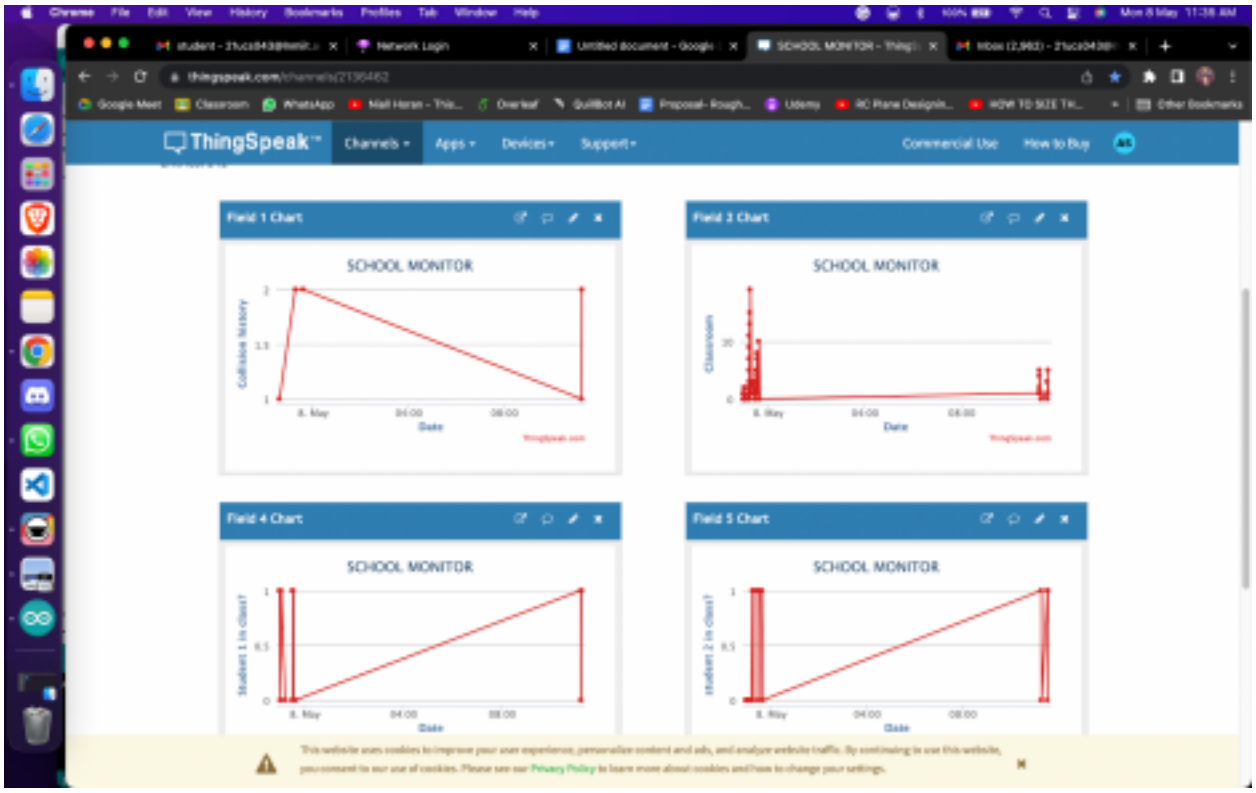
```
  pBLEScan->setWindow(9); // less or equal setInterval value

}

 void loop() {
  BLEScanResults foundDevices = pBLEScan->start(scanTime, false);

  //Serial.println("No. of connected devices: " + String(deviceConnect));
pBLEScan->clearResults(); // delete results fromBLEScan buffer to release
memory
  delay(1000);
 }
```

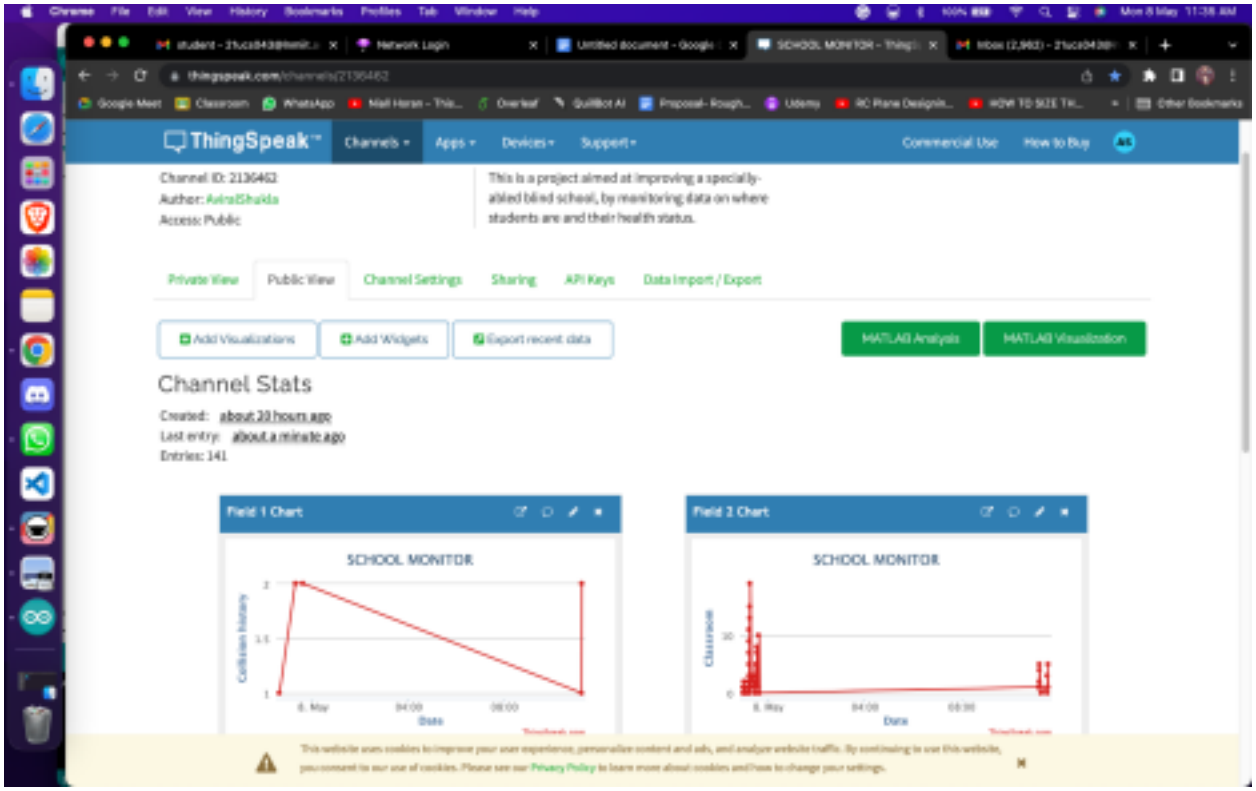
Results:




Starting BLE work!
...
WiFi connected
IP address:
192.168.167.2
RSSI: -74
CLASSROOM DETECTED
Classroom entered
Channel update successful.
RSSI: -93
CLASSROOM DETECTED
In playground
Problem updating channel. HTTP error code -401
RSSI: -79
Student1 detected beware of collision
RSSI: -82
CLASSROOM DETECTED
Classroom entered
Channel update successful.
RSSI: -77
Student1 detected beware of collision
RSSI: -71
CLASSROOM DETECTED
RSSI: -71
Student1 detected beware of collision


CLASSROOM DETECTED
RSSI: -70
Student1 detected beware of collision
RSSI: -77
CLASSROOM DETECTED
Starting BLE work!
.....
WiFi connected
IP address:
192.168.167.2
RSSI: -82
Student1 detected beware of collision
RSSI: -78
CLASSROOM DETECTED
Classroom entered
Channel update successful.
RSSI: -54
Student1 detected beware of collision
COLLISION!!
Problem updating channel. HTTP error code -401
RSSI: -75
Student1 detected beware of collision
RSSI: -74
CLASSROOM DETECTED





Future Plans:

1. Integration with AI and machine learning algorithms to provide personalized assistance to students based on their preferences and needs. This could include adapting to different environments, recognizing

objects and people, and providing real-time recommendations and alerts.

2. Expansion of the system to include other sensory devices, such as haptic feedback and auditory cues, to provide a more immersive and interactive experience for students.

3. Collaboration with other schools and organizations to share best practices and lessons learned, and to create a standardized framework for smart ecosystems in blind schools.

4. Incorporation of additional sensors and analytics tools to monitor environmental factors, such as temperature, humidity, and air quality, and to detect anomalies and potential hazards.

5. Development of a mobile application for teachers and caregivers to monitor and manage student data, receive alerts, and communicate with other stakeholders in the ecosystem.

6. Deployment of the system in other schools and educational settings to promote inclusivity and accessibility for students with visual impairments. This could involve adapting the system to different cultural contexts and regulatory requirements.

7. Integration with smart home technologies and wearable devices to provide a seamless and integrated experience for students both in and out of school.

Conclusion:

Challenges Faced:

1. Lower resources availability for BLE based node-to-node mesh. 2. ESP32 Partition size was too small: Solved by extending partition to Huge APP configuration which extended memory to 3MB
3. MQTT on ThingSpeak was unstable: Solved by using HTTP based on TCP/IP protocol
4. Security Issue with HTTP: Had to secure data using SSL connection 5. BLE RSSI Inaccuracy: Solved by extending scan time and active scan bit to 1.
    a. Downsides: Uses more power
6. Single radio antenna use by BLE and WiFi: Solved by event handling through time slicing between WiFi and BLE

7. ESP Flash memory defected: Solved by hacking BSW400 Smart Watch to use heart rate characteristic as a BLE beacon
    a. Downsides: Could not include flame sensor

In conclusion, the Smart Ecosystem for Blind School project successfully utilized the technology, resources, and tools taught in the course of DPIoT to achieve its objectives. The implementation of collision prevention, position tracking, navigation assistance, and data analysis using short-range BLE mesh, waypoints, intercommunication between student nodes, and an MQTT broker on the Thingspeak IoT platform, has the potential to significantly improve the safety and mobility of blind students within the school environment.

By preventing collisions, tracking student positions, and providing real-time navigation assistance, the smart glasses could help to reduce accidents and improve the independence and mobility of blind students. Additionally, by collecting and analyzing data, the system could help school administrators to identify areas of the school that require additional safety measures and implement them more proactively.

References:

https://in.mathworks.com/help/thingspeak/use-arduino-client-to-publish-to-a-channel.html

https://www.youtube.com/watch?v=JIh2YYwkzoE&list=PLQi2cySxF1TxDKTIiAx5iVUCH4RRMDgfp

https://thingspeak.com/channels/2136462

https://community.appinventor.mit.edu/t/esp32-mqtt-broker-publish-subscribe-thingspeak/10490/8