Smart Stadium

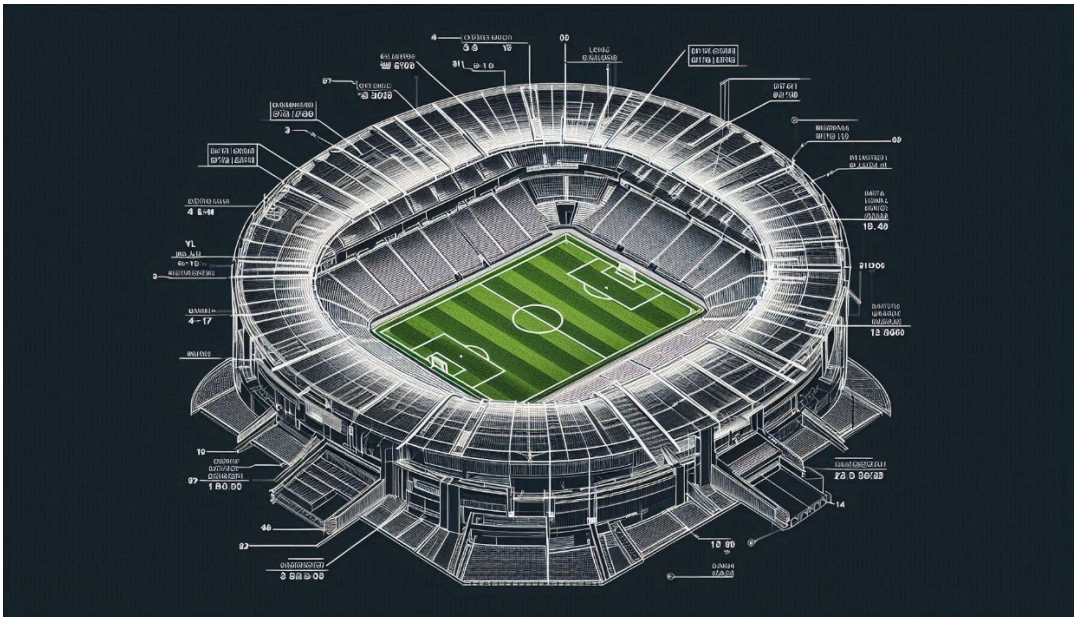
Haqiqat Arsham, Bayan Mohammad Mahdi, Shirvani Fateme, Golshahi Amir Mohammad, Nosrati Mohammad Amin

Abstract

The smart stadium project aims to enhance the overall fan experience by leveraging Internet of Things (IoT) technology to monitor and actuate various aspects of the stadium. Sensors are strategically placed throughout the stadium to monitor crowd density, temperature, air quality, and other environmental factors. These sensors allow for real-time data collection and analysis to optimize crowd flow, improve comfort levels, and ensure safety.

Additionally, the smart stadium project includes actuating devices that can adjust ceiling, temperature, player kits, … based on the data collected. This not only enhances the fan experience but also improves operational efficiency and decreases energy consumption.

By implementing IoT technology in the stadium, fans can enjoy a more comfortable and engaging experience, while stadium operators can streamline operations and better manage resources. This project showcases the potential of IoT in revolutionizing the sports and entertainment industry.



Components and Methodology

1.DHT11 sensor:

The DHT11 sensor is utilized in the context of a smart stadium project to determine whether players should wear long or short sleeves before a match (fig 1). The DHT11 sensor measures temperature and humidity levels in the stadium, providing valuable data for determining the optimal apparel for players based on weather conditions. If the temperature is below a certain threshold, indicating cold weather, the system will recommend players to wear long sleeves. Conversely, if the temperature is above a certain threshold, indicating warmer weather, the system will recommend players to wear short sleeves.

In addition to the DHT11 sensor, the project incorporates the MQ135 gas sensor to determine the opening and closing of the stadium ceiling. The MQ135 gas sensor detects the presence of harmful gases in the stadium, such as CO2 or smoke, which may necessitate the opening of the ceiling for ventilation. By monitoring the air quality in real-time, the system can automatically trigger the opening or closing of the stadium ceiling to ensure a safe and comfortable environment for spectators and players.

To visualize the status of the stadium ceiling, an HTML code has been written to display an image of the stadium with either an open or closed ceiling based on the data collected from the sensors (fig 2). The HTML code is designed to update in real-time, reflecting any changes in the stadium environment and providing a visual representation of the current conditions. This user-friendly interface enables stakeholders to easily monitor and manage the stadium's operations, ensuring a seamless and efficient experience for all involved.

Overall, the integration of the DHT11 and MQ gas sensors in the smart stadium project enhances the safety, comfort, and performance of players and spectators alike. By leveraging IoT technology and data-driven insights, the project demonstrates the potential for innovative solutions to optimize stadium operations and enhance the overall experience for all stakeholders.

Figure 1

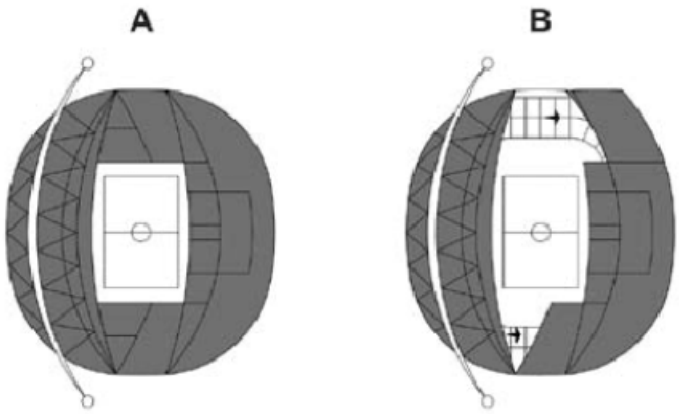


Figure 2

The code used:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
#define DHT_SENSOR_PIN  D7 // The ESP8266 pin D7 connected to DHT11 sensor
#define DHT_SENSOR_TYPE DHT11



const char* ssid = "*****";
const char* password = "*********";
const char* mqttServer = "mqtt.thingsboard.cloud";
const int mqttPort = 1883;
```

```
const char* accessToken = "vv7OPIVTJoGAuPVr5ngj"; // Replace with the Access Token generated for your device

WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {
    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

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

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("ESP8266Client", accessToken, NULL)) {
            Serial.println("connected");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

DHT dht_sensor(DHT_SENSOR_PIN, DHT_SENSOR_TYPE);
```

```
void setup() {
  dht_sensor.begin();
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqttServer, mqttPort);
  // initialize the DHT sensor
}

void loop() {

  if (!client.connected()) {
        reconnect();
    }
    client.loop();

  // read humidity
  float humi  = dht_sensor.readHumidity();
  // read temperature in Celsius
  float temperature_C = dht_sensor.readTemperature();
  // read temperature in Fahrenheit
  float temperature_F = dht_sensor.readTemperature(true);
  int is_long_sleeve = 1;
  int is_ceiling_closed = 1;
  if (temperature_C > 28.0) {
      is_long_sleeve = 0;
      is_ceiling_closed = 0;
  }

  char payload[100];
  sprintf(payload,
"{\"temperature\":%f,\"humidity\":%f,\"is_long_sleeve\":%d,\"is_ceiling_closed\":%d}",temperature_C, humi,
is_long_sleeve,is_ceiling_closed); // Construct the payload
  client.publish("v1/devices/me/telemetry", payload); // Publish the payload to the telemetry topic
  delay(2000); // Adjust delay as needed

  // check whether the reading is successful or not
  if ( isnan(temperature_C) || isnan(temperature_F) || isnan(humi)) {
    Serial.println("Failed to read from DHT sensor!");
  } else {
    Serial.print("Humidity: ");
    Serial.print(humi);
    Serial.print("%");

    Serial.print("  |  ");
```

```
    Serial.print("Temperature: ");
    Serial.print(temperature_C);
    Serial.print("°C  ~  ");
    Serial.print(temperature_F);
    Serial.println("°F");
  }

  // wait a 2 seconds between readings
  delay(2000);
}
```

2.MQ135 sensor:

The MQ135 sensor is known for its ability to detect various harmful gases, making it an essential component in ensuring the safety and security of individuals within a smart stadium environment. When used in combination with the DHT11 sensor, which measures temperature and humidity levels, the two sensors can provide valuable data on the air quality within the stadium.

One of the key applications of this sensor combination in a smart stadium setting is the monitoring of the stadium's ceiling. By analyzing the slope of the data collected from the sensors, particularly the change in gas levels detected by the MQ135 sensor, the system can determine if there is a positive change indicating the presence of dangerous gases. This can serve as an early warning system to alert stadium authorities of potential risks, such as fanatic fans using hazardous substances.

Additionally, the data collected from the sensors can be used to categorize the quality of the stadium air into three colors: green, yellow, and red. The color coding system provides a quick visual indicator of the air quality for stadium operators and attendees. Green signifies clean and safe air, yellow indicates a moderate level of pollutants or gases present, while red indicates a high level of pollutants that may be harmful to health.

Overall, the utilization of the MQ135 and DHT11 sensors in combination offers a comprehensive approach to monitoring and controlling the air quality within a smart stadium. This technology not only enhances the safety and comfort of individuals in the stadium but also enables efficient management of potentially hazardous situations.

The code used:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <string.h>

const char* ssid = "*****";
const char* password = "*****";
const char* mqttServer = "mqtt.thingsboard.cloud";
const int mqttPort = 1883;
const char* accessToken = "xgtGUWwiw5fKm6s5PraB";// Replace with the Access Token generated for your
device

WiFiClient espClient;
PubSubClient client(espClient);

int lastSensorValue = 0;
int lastTime = 0;

void setup_wifi() {
    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

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

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("ESP8266Client", accessToken, NULL)) {
```

```
            Serial.println("connected");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

void setup() {
    Serial.begin(115200);
    setup_wifi();
    client.setServer(mqttServer, mqttPort);
    lastTime = millis();
    lastSensorValue = analogRead(A0); // Initial sensor value
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    int currentTime = millis();
    if (currentTime - lastTime >= 2000) { // Adjust delay as needed
        int sensorValue = analogRead(A0); // Assuming YL-69 sensor is connected to analog pin A0
        int deltaValue = sensorValue - lastSensorValue;
        float deltaTime = (currentTime - lastTime) / 1000.0; // Time in seconds
        float slope = deltaValue / deltaTime;

        // Determine color based on sensor value
        int color;
        if (sensorValue < 600) {
            color = 1;
        } else if (sensorValue >= 600 && sensorValue <= 700) {
            color = 2;
        } else {
            color = 3;
        }

        // Publish air quality data
        char payload[100];
        sprintf(payload, "{\"airquality\":%d,\"color\":%d}", sensorValue, color); // Construct the payload
```

```
        client.publish("v1/devices/me/telemetry", payload); // Publish the payload to the telemetry topic

        // Publish slope data
        if (abs(slope) > 6) {
            client.publish("v1/devices/me/telemetry", "{\"slope\":1}"); // Slope is greater than 6
        } else {
            client.publish("v1/devices/me/telemetry", "{\"slope\":0}"); // Slope is less than or equal to
6
        }

        // Update last sensor value and time
        lastSensorValue = sensorValue;
        lastTime = currentTime;
    }
}
```

3. YL69 sensor:

The YL69 sensor is a type of soil moisture sensor that is commonly used in agricultural settings to determine the dryness of the soil. In a smart stadium context, the YL69 sensor can be strategically placed in the ground to monitor the moisture levels of the grass on the field.

The sensor works by measuring the electrical conductivity of the soil, which changes as the moisture content of the soil changes. When the grass becomes too dry, the sensor will detect a decrease in conductivity, indicating that the grass needs to be watered.

In order to automate the watering process, an actuator can be connected to the YL69 sensor. The actuator is a device that is capable of physically moving or adjusting something based on input from a sensor. In this case, when the YL69 sensor detects that the grass is too dry, it will send a signal to the actuator to start watering the grass.

This process can be automated by integrating the YL69 sensor and actuator with a central control system, such as a microcontroller or a smart management platform. The control system can be programmed to monitor the moisture levels of the grass and activate the actuator when necessary, ensuring that the grass is always kept at the optimal level of hydration.

By using the YL69 sensor and actuator in this way, the smart stadium can ensure that the grass on the field is always healthy and well-maintained, providing a better experience for both players

and spectators. Additionally, this automated system can help save time and resources by only watering the grass when needed, rather than on a predefined schedule.

The code used:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// Define the GPIO pin where the actuator is connected
const int actuatorPin = 5;  // GPIO5 (D1)

// Threshold value for the parameter
const int threshold = 25;

// WiFi credentials
const char* ssid = "***";
const char* password = "******";

// MQTT server settings
const char* mqttServer = "mqtt.thingsboard.cloud";
const int mqttPort = 1883;
const char* accessToken = "qtwiypl9yaiq7o027fo2"; // Replace with the Access Token generated for your
device

WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {
    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

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

```
    Serial.println(WiFi.localIP());
}

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("ESP8266Client", accessToken, NULL)) {
            Serial.println("connected");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

void setup() {
    Serial.begin(115200);
    setup_wifi();
    client.setServer(mqttServer, mqttPort);

    // Initialize the actuator pin as an output
    pinMode(actuatorPin, OUTPUT);

    // Start with the actuator turned off
    digitalWrite(actuatorPin, LOW);
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    // Read data from the analog sensor (e.g., YL-69 moisture sensor)
    int sensorValue = analogRead(A0); // Assuming the sensor is connected to analog pin A0


    // Calibrate sensor value from 0 (wet) to 100 (dry)
    int calibratedValue = map(sensorValue, 1024, 0, 0, 200);
    // Constrain the value to be within 0-100 range
```

```
    calibratedValue = constrain(calibratedValue, 0, 100);
    // Publish data to ThingsBoard
    char payload[50];
    sprintf(payload, "{\"moisture\":%d}", calibratedValue); // Construct the payload
    client.publish("v1/devices/me/telemetry", payload);

    // Check if the sensor value exceeds the threshold
    if (calibratedValue < threshold) {
        // Turn on the actuator
        digitalWrite(actuatorPin, HIGH);
        Serial.println("Actuator ON");
    } else {
        // Turn off the actuator
        digitalWrite(actuatorPin, LOW);
        Serial.println("Actuator OFF");
    }

    delay(2000); // Adjust delay as needed
}
```

4.MPU6050 sensor:

The MPU6050 sensor is a highly sensitive and accurate motion tracking sensor commonly used in sports applications to determine the speed of a player on the field. This sensor works by measuring the acceleration and rotational movements of the player, allowing for the calculation of their speed and movement patterns.

To determine the speed of a player using the MPU6050 sensor, the sensor is typically placed on the player's body or equipment, such as a jersey or a shoe. As the player moves around the field, the sensor collects data on their acceleration and rotation in three dimensions – X, Y, and Z axes. This data is then processed using algorithms to calculate the player's velocity in real-time.

By continuously tracking the player's movements, the MPU6050 sensor can provide valuable insights into the player's speed, acceleration, deceleration, change of direction, and other performance metrics. This information can be used by coaches, trainers, and sports analysts to assess the player's performance, monitor their progress, and make data-driven decisions to improve their training and gameplay.

In addition to determining the speed of a player, the MPU6050 sensor can also be used to create a heatmap of the player's movements on the field (figure 3). By overlaying the player's movement data onto a visual representation of the field, coaches and analysts can identify hotspots where the player spends the most time, areas of high activity, patterns in their movement, and potential areas for improvement.

Overall, the MPU6050 sensor is an essential tool in modern sports technology for tracking and analyzing the performance of players on the field. Its high accuracy and real-time monitoring capabilities make it a valuable asset for coaches, trainers, and players looking to optimize their training and gameplay strategies in a smart stadium environment.

the code:

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "Gili";
const char* password = "@amir127";
const char* mqttServer = "mqtt.thingsboard.cloud";
const int mqttPort = 1883;
const char* accessToken = "mq1GzuYZmceM5Cufd2j1"; // Replace with the Access Token generated for your device

WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {
    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

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

void reconnect() {
    // Loop until we're reconnected
```

```
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("ESP8266Client", accessToken, NULL)) {
            Serial.println("connected");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

Adafruit_MPU6050 mpu;

// Variables for velocity calculation
float velocityZ = 0;
float velocityX = 0;
float velocityY = 0;
unsigned long lastTime = 0;
float baselineZ = 0;
float baselineX = 0;
float baselineY = 0;
float first=0;
float firstx=0;
float firsty=0;
float speedSum=0;
int speedCount=0;
float meanSpeed;
float displacementX = 0, displacementY = 0, displacementZ = 0;




void calibrateSensor() {


  Serial.println("Calibrating sensor...");

   const int calibrationSamples = 1000;
  float sumX = 0, sumY = 0, sumZ = 0;
```

```
  Serial.println("Calibrating sensor...");

  for (int i = 0; i < calibrationSamples; i++) {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);

    sumX += a.acceleration.x;
    sumY += a.acceleration.y;
    sumZ += a.acceleration.z;

    delay(2); // Delay to allow sensor to stabilize
  }

  baselineX = sumX / calibrationSamples;
  baselineY = sumY / calibrationSamples;
  baselineZ = sumZ / calibrationSamples;

  Serial.print("Baseline X acceleration: ");
  Serial.println(baselineX);
  Serial.print("Baseline Y acceleration: ");
  Serial.println(baselineY);
  Serial.print("Baseline Z acceleration: ");
  Serial.println(baselineZ);
}

void setup(void) {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqttServer, mqttPort);
  while (!Serial)
    delay(10); // will pause Zero, Leonardo, etc until serial console opens

  Serial.println("Adafruit MPU6050 test!");

  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  Serial.print("Accelerometer range set to: ");
```

```
switch (mpu.getAccelerometerRange()) {
case MPU6050_RANGE_2_G:
  Serial.println("+-2G");
  break;
case MPU6050_RANGE_4_G:
  Serial.println("+-4G");
  break;
case MPU6050_RANGE_8_G:
  Serial.println("+-8G");
  break;
case MPU6050_RANGE_16_G:
  Serial.println("+-16G");
  break;
}
mpu.setGyroRange(MPU6050_RANGE_500_DEG);
Serial.print("Gyro range set to: ");
switch (mpu.getGyroRange()) {
case MPU6050_RANGE_250_DEG:
  Serial.println("+- 250 deg/s");
  break;
case MPU6050_RANGE_500_DEG:
  Serial.println("+- 500 deg/s");
  break;
case MPU6050_RANGE_1000_DEG:
  Serial.println("+- 1000 deg/s");
  break;
case MPU6050_RANGE_2000_DEG:
  Serial.println("+- 2000 deg/s");
  break;
}

mpu.setFilterBandwidth(MPU6050_BAND_5_HZ);
Serial.print("Filter bandwidth set to: ");
switch (mpu.getFilterBandwidth()) {
case MPU6050_BAND_260_HZ:
  Serial.println("260 Hz");
  break;
case MPU6050_BAND_184_HZ:
  Serial.println("184 Hz");
  break;
case MPU6050_BAND_94_HZ:
  Serial.println("94 Hz");
  break;
case MPU6050_BAND_44_HZ:
  Serial.println("44 Hz");
```

```
    break;
  case MPU6050_BAND_21_HZ:
    Serial.println("21 Hz");
    break;
  case MPU6050_BAND_10_HZ:
    Serial.println("10 Hz");
    break;
  case MPU6050_BAND_5_HZ:
    Serial.println("5 Hz");
    break;
  }

  Serial.println("");
  delay(100);

  // Calibrate the sensor to determine the baseline offset
  calibrateSensor();

  // Initialize the lastTime variable
  lastTime = millis();
}

void loop() {
  if (!client.connected()) {
        reconnect();
     }

      client.loop();
  /* Get new sensor events with the readings */
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  // Calculate the time elapsed
  unsigned long currentTime = millis();
  float deltaTime = (currentTime - lastTime) / 1000.0; // Convert milliseconds to seconds
  lastTime = currentTime;

  // Remove baseline from the acceleration readings (assuming Z-axis is aligned with gravity)
  float accelerationZ = a.acceleration.z - baselineZ;
    // Remove baseline from the acceleration readings (assuming Z-axis is aligned with gravity)
  float accelerationX = a.acceleration.x - baselineX;  // assuming you also calibrate baselineX like
baselineZ
  float accelerationY = a.acceleration.y - baselineY;  // assuming you also calibrate baselineY like
baselineZ
```

```
  // Integrate acceleration to get velocity
  velocityZ=0;
  velocityX=0;
  velocityY=0;



// Integrate acceleration to get velocity for X and Y
float deltay,deltax;
deltay=0;
deltax=0;
deltax=accelerationX-firstx;
deltay=accelerationY-firsty;


  velocityX = deltax * deltaTime;  // Assuming velocityX is declared and initialized
  velocityY = deltay * deltaTime;  // Assuming velocityY is declared and initialized
  firstx= accelerationX;
  firsty= accelerationY;
// Calculate the resultant speed using Pythagorean theorem
  float speed = sqrt(velocityX * velocityX + velocityY * velocityY);

float delta;
delta=0;
delta=accelerationZ-first;
  velocityZ = delta * deltaTime;
  first= accelerationZ;
  // finde place
  // Integrate velocity to update displacement
displacementX += velocityX * deltaTime;
displacementY += velocityY * deltaTime;
displacementZ += velocityZ * deltaTime;


// this part for  get mean of speed

 speedSum += speed;
  speedCount++;

  // Calculate mean speed every 10 samples
  if (speedCount >= 10) {

    meanSpeed = speedSum / speedCount;
    Serial.print("Mean Speed: ");
    Serial.println(meanSpeed);
```

```
    // Reset for next batch
    speedSum = 0;
    speedCount = 0;
  }


  /* Print out the values */
  Serial.print("Acceleration X: ");
  Serial.print(a.acceleration.x);
  Serial.print(", Y: ");
  Serial.print(a.acceleration.y);
  Serial.print(", Z: ");
  Serial.print(a.acceleration.z);
  Serial.println(" m/s^2");

  Serial.print("Rotation X: ");
  Serial.print(g.gyro.x);
  Serial.print(", Y: ");
  Serial.print(g.gyro.y);
  Serial.print(", Z: ");
  Serial.print(g.gyro.z);
  Serial.println(" rad/s");

  Serial.print("Temperature: ");
  Serial.print(temp.temperature);
  Serial.println(" degC");

  Serial.print("Instantaneous Velocity Z: ");
  Serial.print(velocityZ);
  Serial.println(" m/s");

  Serial.println("");

Serial.print("Mean Speed: ");
    Serial.println(meanSpeed);

//show result place
Serial.print("Displacement X: ");
Serial.print(displacementX);
Serial.print(", Y: ");
Serial.print(displacementY);
Serial.print(", Z: ");
Serial.print(displacementZ);
Serial.println(" meters");
```

```
   Serial.print("Resultant Speed: ");
  Serial.print(speed);
  Serial.println(" m/s");
// Publish data to ThingsBoard
     // Publish data to ThingsBoard
  char payload[50];
  sprintf(payload, "{\"speed\":%.1f,\"meanspeed\":%.1f, \"x\":%.1f,\"y\":%.1f}",
velocityZ,meanSpeed,displacementX,displacementY); // Construct the payload with float formatting
  Serial.print("Publishing payload: ");
  Serial.println(payload);
  client.publish("v1/devices/me/telemetry", payload); // Publish the payload to the telemetry topic

  Serial.println("");

  delay(1000);
}
```

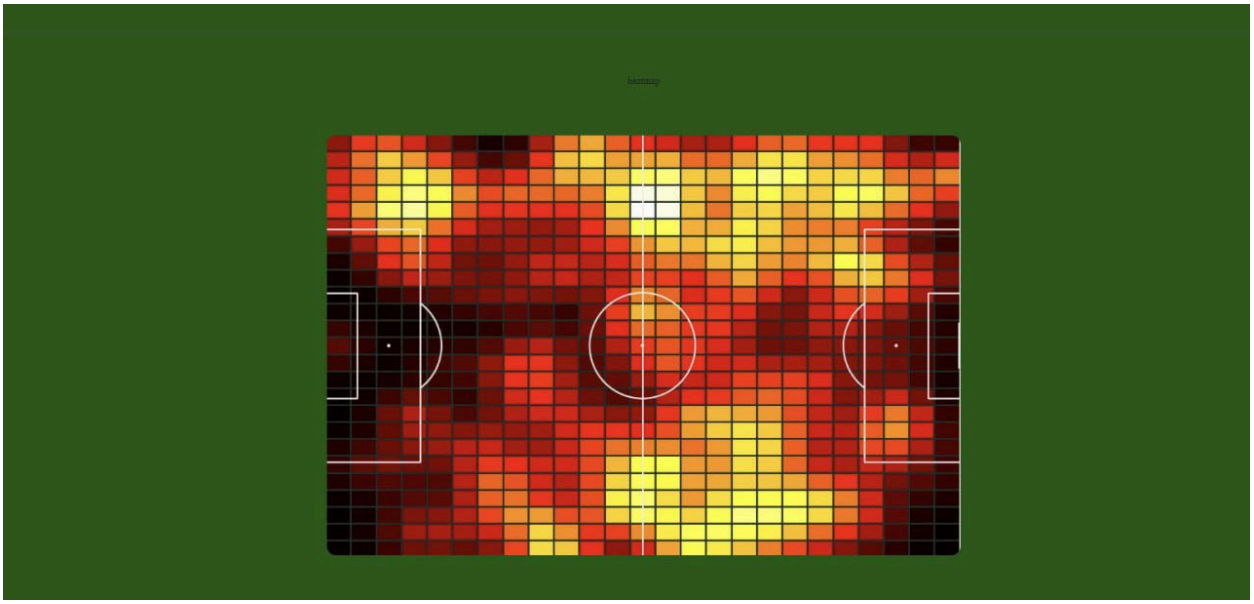


Figure 3

Conclusions

A combination of sensors and virtual sensors together can be the key to the new era of sports. Internet of things can be the thing, managers and other sports related entities have been looking for to revolutionize the game. By monitoring and automating the stadium related chores and challenges, we can implement the Ashton principle. The more data we collect automatically the more we can use the power of Internet Of Things and save human time and labor. This is

what we tried to do in this project and he hope we would be able to help make sports more enjoyable and with more quality.