Dog Activity Tracker

Introduction

Using accelerometer and gyroscope data, the Dog Activity Tracker is a simulated Internet of Things project that simulates real-time physical activity tracking for dogs. The system, which was constructed with a virtual IMU sensor (QMI8658), categories motion into different activities as playing, walking, running, and resting.

Objective

The primary objective of this project is to:

- Simulate motion detection using IMU data
- Classify the type of activity a dog is performing
- Log and display this information in real-time on an LCD
- Serve as a foundation for integrating more complex features such as BLE and machine learning

Hardware

- Microcontroller: Arduino Uno
- LCD Display: 16x2 character LCD
- IMU Sensor: QMI8658 (Accelerometer + Gyroscope)
- Serial Monitor: Used for live debugging and activity logs

Software Components

- **Simulated Sensor Class:**
  The SimulatedQMI8658 class mimics accelerometer and gyroscope readings, with random value generation and motion event triggering (20% probability).

- **Activity Classifier:**
  Based on average acceleration magnitude, activities are classified as:
  < 0.1 -> Resting
  < 0.5 -> Walking
  < 1.0 -> Running

>= 1.0 -> Playing

- **LCD Display**
  The 16x2 LCD displays:
  Line 1: Current activity
  Line 2: Motion detection alerts

- **Logger**
  A simple String logData[10] array stores the last 10 classified activities.

Implementation Details

Project Flow:

- The sensor and LCD are initialised before the system starts.
- produces sensor data and simulated motion on a regular basis.
- uses sensor data to classify the type of motion.
- records and shows the activity that is classified.
- Debug data is printed using a serial monitor.

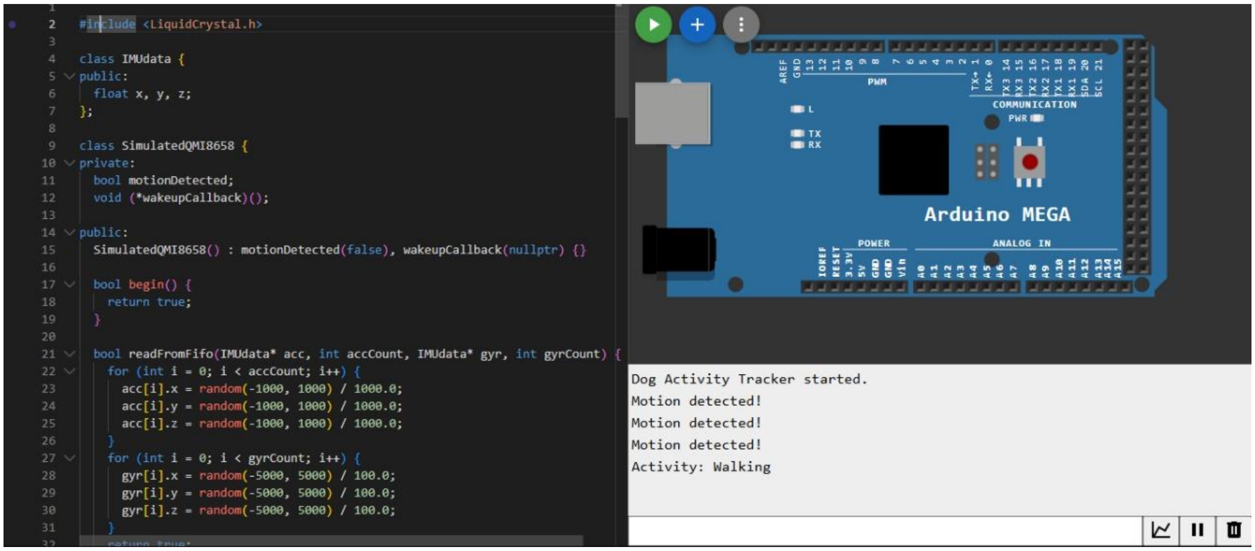
Timing

- Activity is logged every 10 seconds
- Motion detection is simulated every 0.5 seconds

Future Enhancements

- TensorFlow Lite model integration for smarter classification
- BLE support to send logs to a mobile app
- Store activity logs in SPIFFS or LittleFS on ESP boards
- Add button to manually print historical logs

Output Example



```
#include <LiquidCrystal.h>

class IMUdata {
public:
  float x, y, z;
};

class SimulatedQMI8658 {
private:
  bool motionDetected;
  void (*wakeupCallback)();

public:
  SimulatedQMI8658() : motionDetected(false), wakeupCallback(nullptr) {}

  bool begin() {
    return true;
  }

  bool readFromFifo(IMUdata* acc, int accCount, IMUdata* gyr, int gyrCount) {
    for (int i = 0; i < accCount; i++) {
      acc[i].x = random(-1000, 1000) / 1000.0;
      acc[i].y = random(-1000, 1000) / 1000.0;
      acc[i].z = random(-1000, 1000) / 1000.0;
    }
    for (int i = 0; i < gyrCount; i++) {
      gyr[i].x = random(-5000, 5000) / 100.0;
      gyr[i].y = random(-5000, 5000) / 100.0;
      gyr[i].z = random(-5000, 5000) / 100.0;
    }
    return true;
```

```
Dog Activity Tracker started.
Motion detected!
Motion detected!
Motion detected!
Activity: Walking
```

Serial Output:

Dog Activity Tracker started.
Motion detected!
Activity: Running
Activity: Resting
... (updates every 10s)

LCD Display

Activity: Playing

Motion Detected

#code sketch in C++ Implementation in Arduino Uno

```
#include <LiquidCrystal.h>

class IMUdata {
public:
  float x, y, z;
};

class SimulatedQMI8658 {
private:
  bool motionDetected;
  void (*wakeupCallback)();

public:
  SimulatedQMI8658() : motionDetected(false), wakeupCallback(nullptr) {}

  bool begin() {
    return true;
  }

  bool readFromFifo(IMUdata* acc, int accCount, IMUdata* gyr, int gyrCount) {
    for (int i = 0; i < accCount; i++) {
      acc[i].x = random(-1000, 1000) / 1000.0;
      acc[i].y = random(-1000, 1000) / 1000.0;
      acc[i].z = random(-1000, 1000) / 1000.0;
    }
    for (int i = 0; i < gyrCount; i++) {
      gyr[i].x = random(-5000, 5000) / 100.0;
      gyr[i].y = random(-5000, 5000) / 100.0;
      gyr[i].z = random(-5000, 5000) / 100.0;
    }
    return true;
  }

  void setWakeupMotionEventCallBack(void (*callback)()) {
    wakeupCallback = callback;
  }

  void simulateMotion() {
    if (random(100) < 20 && !motionDetected) {
      motionDetected = true;
      if (wakeupCallback) wakeupCallback();
    } else {
      motionDetected = false;
    }
  }
};

String classifyActivity(IMUdata* acc, int count) {
  float total = 0;
  for (int i = 0; i < count; i++) {
    total += abs(acc[i].x) + abs(acc[i].y) + abs(acc[i].z);
  }
  float avg = total / (count * 3);

  if (avg < 0.1) return "Resting";
  else if (avg < 0.5) return "Walking";
  else if (avg < 1.0) return "Running";
  else return "Playing";
}

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
String logData[10];
int logIndex = 0;

SimulatedQMI8658 sensor;
unsigned long lastLogTime = 0;

void motionCallback() {
  Serial.println("Motion detected!");
  lcd.setCursor(0, 1);
  lcd.print("Motion Detected ");
}

void setup() {
  Serial.begin(9600);
  sensor.begin();
  sensor.setWakeupMotionEventCallBack(motionCallback);

  lcd.begin(16, 2);
  lcd.print("Dog Tracker Ready");
  Serial.println("Dog Activity Tracker started.");
}

void loop() {
  static IMUdata acc[10], gyr[10];

  sensor.simulateMotion();

  sensor.readFromFifo(acc, 10, gyr, 10);

  if (millis() - lastLogTime > 10000) {
    String activity = classifyActivity(acc, 10);
    Serial.print("Activity: ");
    Serial.println(activity);

    lcd.setCursor(0, 0);
    lcd.print("Activity:       ");
    lcd.setCursor(10, 0);
    lcd.print(activity);

    if (logIndex < 10) {
      logData[logIndex++] = activity;
    } else {
      for (int i = 0; i < 9; i++) logData[i] = logData[i + 1];
      logData[9] = activity;
    }
    lastLogTime = millis();
  }

  delay(500);
}
```