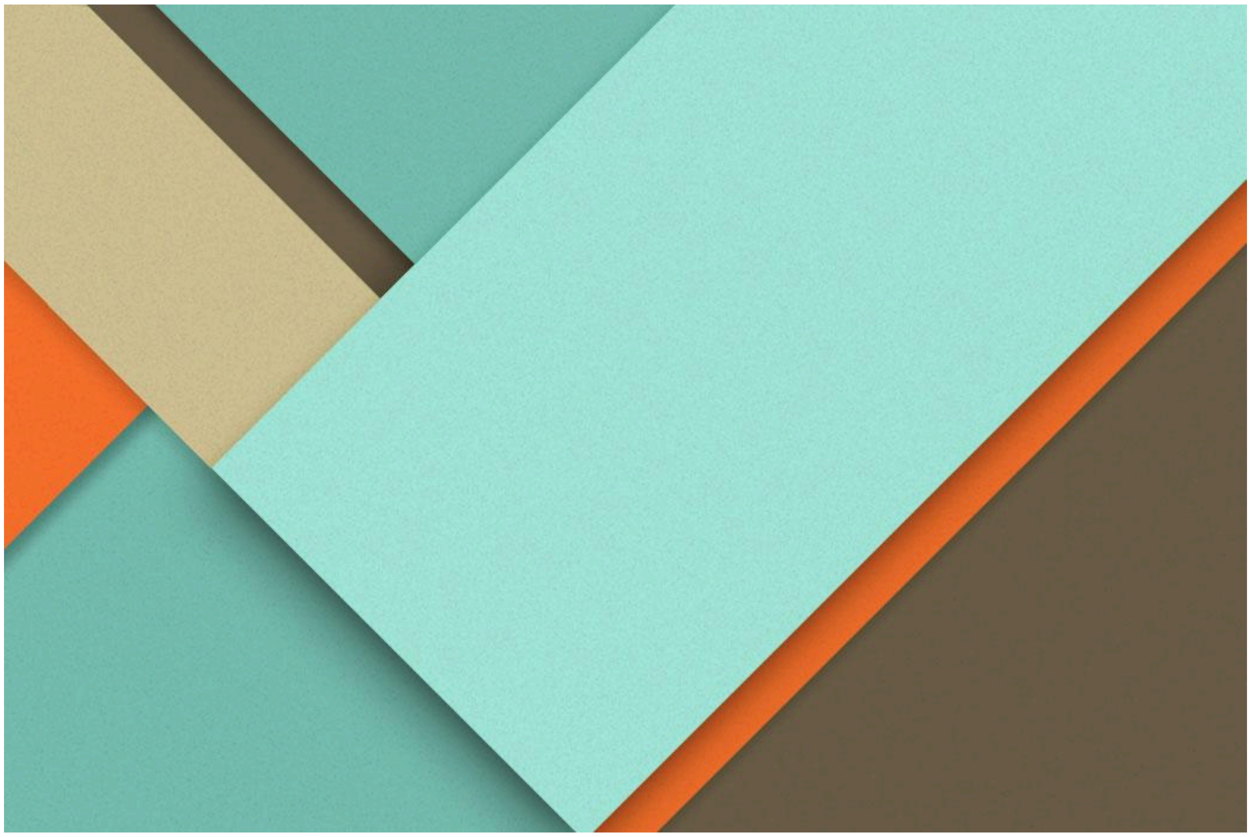


FSM and Closed-loop integration

Project 4 - Player Tracking and Crowd Monitoring

Ayush Kumar Som
222198016

Introduction

Player tracking in sports using sensor integration is an evolving area that benefits from precise control and monitoring systems. Employing closed-loop control and finite state machines (FSMs) can significantly enhance the performance of such systems, particularly in wearable tracksuits for players. This report provides detailed steps and specific methods to implement a robust player-tracking system with multiple sensors, leveraging closed-loop control and FSMs.

Project Overview

The player tracking system involves an Arduino Nano controlling and processing data from the following sensors:

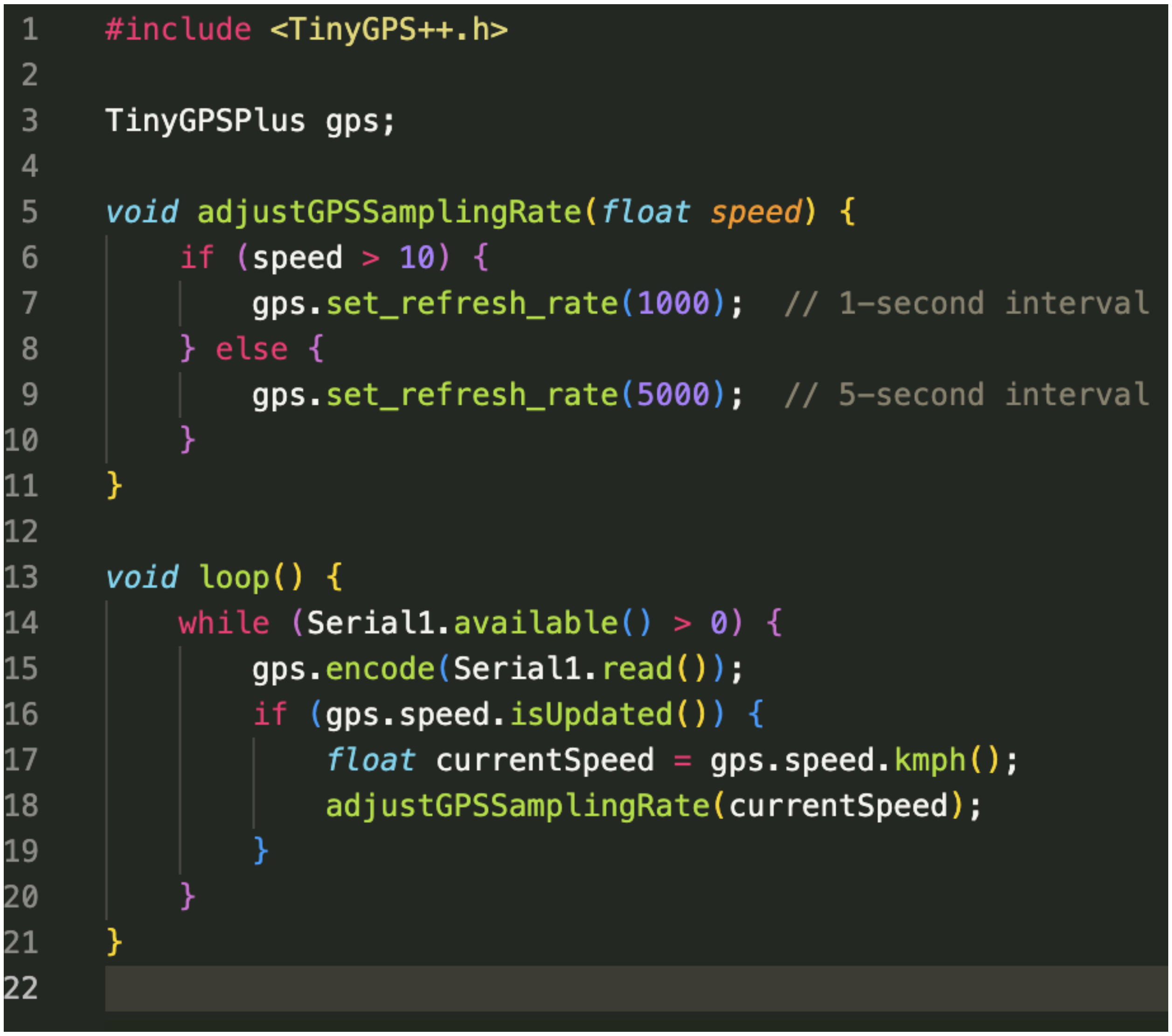
1. **NEO-6M GPS Module**: Provides positional and velocity information.
2. **QMC5883L Compass**: Provides directional data.
3. **Wahoo Sensor**: An accelerometer or heart rate monitor providing physiological or motion data.

Closed-Loop Control

Closed-loop control uses feedback to adjust system behaviour. In the context of player tracking, this approach maintains steady signals, optimises sensor orientation, or regulates data acquisition rates.

Method: Dynamic GPS Sampling Rate Adjustment

1. **Initialisation:**
    - Set up the GPS module using the TinyGPS++ library.
    - Initialize variables to store position, speed, and time.
2. **Feedback:**
    - Read the player’s speed using gps.speed.kmph().
3. **Control:**
    - Adjust the GPS sampling rate based on the player’s speed.
    - Increasing the sampling rate if the speed is above a threshold (e.g., 10 km/h).
    - If the speed is below a threshold, decrease the sampling rate to save power.
4. **Implementation:**
    - Use the TinyGPSCustom object to adjust the GPS refresh rate.
    - Update the rate dynamically based on the feedback.



```cpp
#include <TinyGPS++.h>

TinyGPSPlus gps;

void adjustGPSSamplingRate(float speed) {
    if (speed > 10) {
        gps.set_refresh_rate(1000);  // 1-second interval
    } else {
        gps.set_refresh_rate(5000);  // 5-second interval
    }
}

void loop() {
    while (Serial1.available() > 0) {
        gps.encode(Serial1.read());
        if (gps.speed.isUpdated()) {
            float currentSpeed = gps.speed.kmph();
            adjustGPSSamplingRate(currentSpeed);
        }
    }
}
```

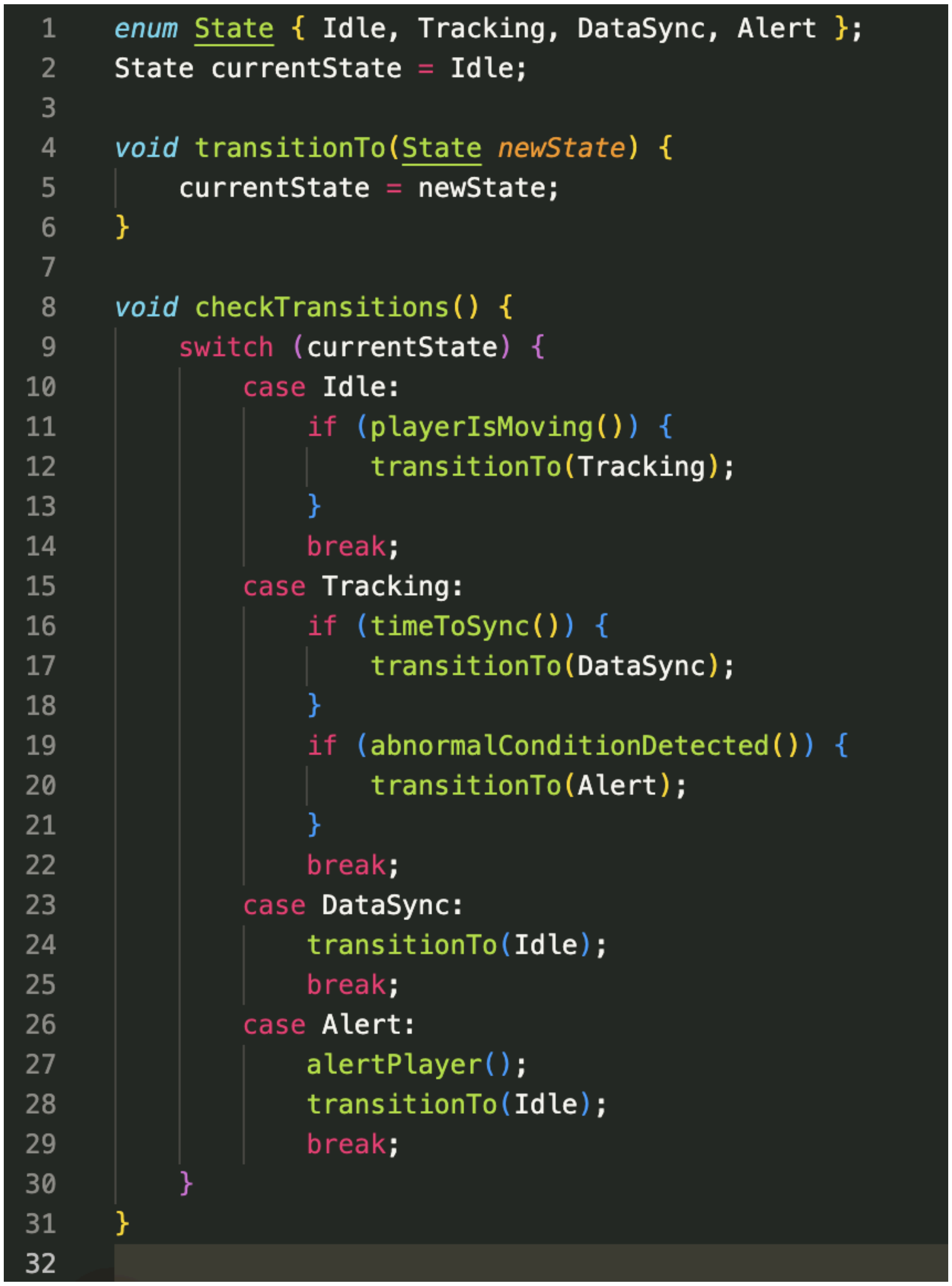
Finite State Machine

Finite state machines (FSMs) manage system behavior through discrete states and transitions influenced by sensor inputs.

Method: FSM for Player Tracking

1. **Define States:**
    - Idle: Waiting for player movement.
    - Tracking: Collecting data from sensors.
    - Data Sync: Syncing data with a central server or storage.
    - Alert: Generating an alert based on abnormal conditions.

2. **Set Initial State:**
   - Set the initial state to Idle.
3. **Implement Transitions:**



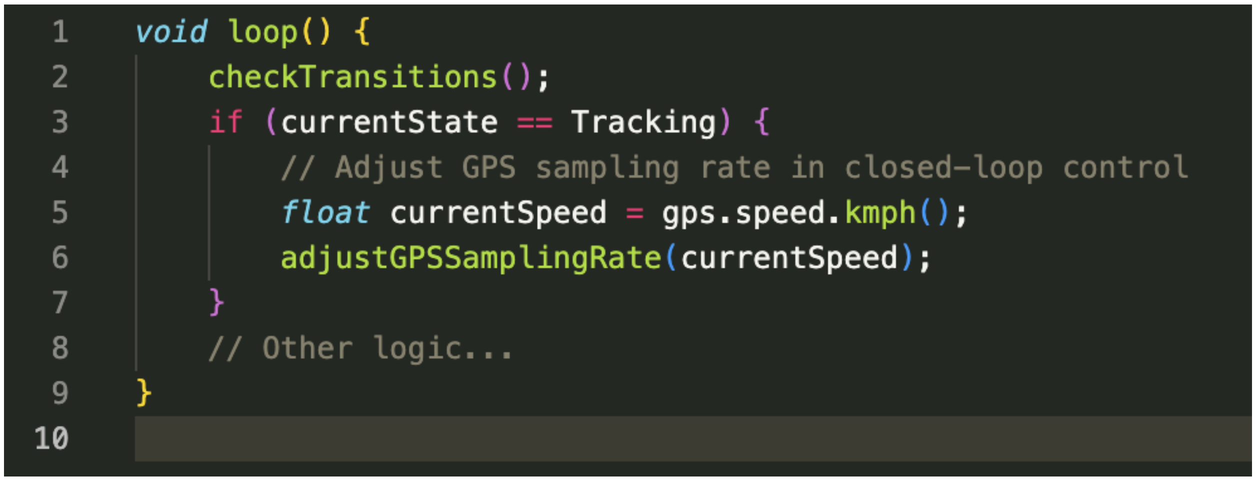
```
enum State { Idle, Tracking, DataSync, Alert };
State currentState = Idle;

void transitionTo(State newState) {
    currentState = newState;
}

void checkTransitions() {
    switch (currentState) {
        case Idle:
            if (playerIsMoving()) {
                transitionTo(Tracking);
            }
            break;
        case Tracking:
            if (timeToSync()) {
                transitionTo(DataSync);
            }
            if (abnormalConditionDetected()) {
                transitionTo(Alert);
            }
            break;
        case DataSync:
            transitionTo(Idle);
            break;
        case Alert:
            alertPlayer();
            transitionTo(Idle);
            break;
    }
}

```

4. **Execute FSM Logic:**
   - In the main loop, call checkTransitions() to evaluate and execute the state transitions based on sensor feedback and predefined rules.

Integration of Closed-Loop Control and FSM

Combining closed-loop control and FSMs enhances the system's adaptability and robustness:

1. **Closed-loop Control:**
   - During the "Tracking" state, the system adjusts the GPS sampling rate based on the player's speed.
2. **FSM:**
   - The FSM manages transitions between "Idle," "Tracking," "Data Sync," and "Alert" states based on sensor feedback and predefined rules.



```
void loop() {
    checkTransitions();
    if (currentState == Tracking) {
        // Adjust GPS sampling rate in closed-loop control
        float currentSpeed = gps.speed.kmph();
        adjustGPSSamplingRate(currentSpeed);
    }
    // Other logic...
}

```

Conclusion

Integrating multiple sensors for player tracking using closed-loop control and FSM provides robust performance in dynamic environments. By combining these methodologies with appropriate hardware and software, systems can achieve adaptive, efficient, and responsive player monitoring in wearable tracksuits, enhancing sports performance and safety.

Resources

Hardware:

1. **Arduino Nano:** Central controller, available from Arduino resellers or online marketplaces.
2. **NEO-6M GPS Module:** Positional sensor, obtainable from electronics retailers.

3. **QMC5883L Compass Module:** Directional sensor, commonly available with Arduino-compatible sensors.
4. **Wahoo Sensor:** Physiological or motion sensor, check sports or electronics stores for compatible sensors.

Software:

1. **Arduino IDE:** For developing control and FSM logic, available from the Arduino website.
2. **TinyGPS++ Library:** For parsing GPS data, installable from the Arduino Library Manager.
3. **QMC5883L Library:** For interfacing with the compass, available through Arduino Library Manager or other repositories.

References:

1. **Arduino Documentation:** Comprehensive guides on the Arduino website.
2. **Sensor Documentation:** Refer to the datasheets and manuals for detailed specifications.
3. **Online Tutorials:** Numerous online tutorials cover Arduino-based sensor projects.