Challenge 4 - Homework

Abate Kevin Pio: 10812892
Pigato Lorenzo: 10766953 [Team Leader]

Exercise 1 - Low Cost Forklift Tracking

Context

A logistics company operates a warehouse composed of a 500 m² underground indoor area and a 1 km² outdoor yard.
Electric forklifts are used across both zones and return to specific docking stations to recharge.

Objective

Design a low-cost IoT system to:

- localize forklifts in real time
- monitor their status, including daily distance traveled, maximum and average speed, and impact detection.

Assumptions

- Forklifts will be driven by an operator, they are not autonomous vehicles
- Given a total area of 1500 squared meters, it's possible to assume that no more than 10 forklifts will be deployed

Concept

In order to track forklifts position in a warehouse with an underground facility where GPS is unavailable, a Zigbee mesh network can be implemented.
Placing fixed Zigbee **anchor nodes** throughout the facility and equipping each forklift with an ESP32-C6 Zigbee transmitter, it's possible to determine the position of each forklift using triangulation based on the *Received Signal Strength Indicator* of each beacon received by the anchors.
All position data is transmitted by the anchors to a central server that monitors the entire fleet in real-time, including the status of each unit - inside each sent message, information collected by their onboard sensors is also provided.

Required Hardware

Forklifts

Every forklift must be equipped with a **ESP32-C6 board** (~5€ per board), which is ZigBee-enabled.
To monitor the status of the units, including information about actual battery level, traveled distance, collision detection, maximum and average speed, dedicated sensors must be installed on board:

- **Rotary encoders**: measure forklift odometer (traveled distance) and calculate maximum and average speed based on wheels spinning, (~5-10€ per piece)
- **Accelerometer**: can be used to detect collisions (sudden accelerations and decelerations) and also speed through direct integration (~1€ per piece)

Anchors

ESP32-C6 boards will be necessary also for anchor nodes displaced throughout the warehouse.

Communication Protocol

ZigBee protocol with **beacons** has been chosen to implement this solution because it offers **low-power communication** with a range of nearly 10 meters, paired with the possibility to build a **mesh network** it can easily cover all the warehouse area, both the outdoor yard and the indoor space, while keeping the topology updated.

Anchors will be set as ***Full Function Devices*** and will send beacons to the forklifts to capture variations in network topology due to them moving around the facility.
These beacons are essential also to implement the position tracking algorithm: every time a beacon is received by a forklift, it stores the ID of the sender and the *Received Signal Strength Indicator* associated to the message, which can be used to determine the distance from the sender.

Forklifts are configured as ***Reduced Function Devices***.
When beacons from at least 3 anchors are acquired, the forklift can sand to the PAN coordinator a standard message containing the list of beacons ID and *RSSI* collected, alongside data collected by on-board sensors.
All the data will be processed by the coordinator, which will be linked to a local server.
Forklifts will dynamically change their parent node according to ZigBee *transparent roaming*.

In order to keep the localization precise enough, it is necessary to send beacons at least every 10 seconds.

System Architecture

Data Wrangling

The process of cleaning and structuring raw data follows this flow:

- **Ingestion**:
  Before all, the forklifts send data to the anchors which route it to the PAN coordinator
- **Discovery**:
  A preliminary analysis is done to identify missing values or anomalies.
- **Data Cleaning**:
  Correct or delete wrong, duplicated or missing data in order to improve quality. In this specific case we delete all the outliers in RSSI field and then check for missing data in received payloads.
- **Data Publishing**:
  Now data can be collected. A good fit for our problem is a *Document Database*, as MongoDB. It's simple to use, flexible, with a good performance and scalable.

Data will be stored as described below:

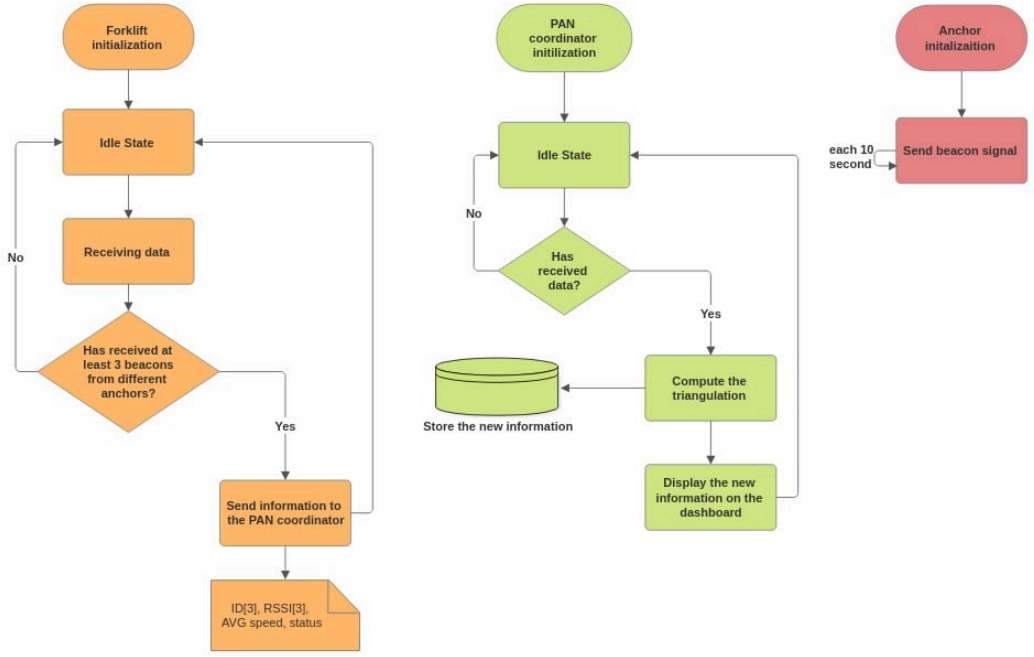
```
{
  "_id": ObjectId("..."),
  "timestamp": ISODate("2025-05-16T14:30:00Z"),
  "forklift_id": "FL-103",
  "status": "active",
  "avg_speed": 4.3,
  "position": {
    "type": "Point",
    "coordinates": [45.23, 12.87]  // [x, y] coordinate
  },
  "battery_level": 78,
  "rssi_readings": [
    { "anchor_id": "AN-01", "rssi": -67, "lqi": 203 },
    { "anchor_id": "AN-05", "rssi": -72, "lqi": 187 },
    { "anchor_id": "AN-12", "rssi": -85, "lqi": 132 }
  ]
}
```

Data Visualization

Finally data can be visualized in real-time using tools as Grafana, which are compatible with the implemented database solution.

System-wide block diagram

> Assuming all *Anchors* and *Forklifts* had joined the Zigbee network



Forklift pseudo-code

```
#include <Wire.h>
#include <XBee.h>      // ZigBee Library
#include <MPU6050.h>   // Accelerometer Library

//// Zigbee communication ////
XBee xbee;

struct BeaconData {
    int anchorId;
    int rssi;
    int lastSeen;
};

BeaconData beacons[10];
int beaconCount = 0;

//// Sensor data ////
volatile long encoderTicks = 0;

float distance = 0.0;
float curSpeed = 0.0;
float maxSpeed = 0.0;
float avgSpeed = 0.0;

int impacts = 0;
MPU6050 accelerometer;

// Process encoder data to calculate odometry
void processOdometry() {

    // Encoder ticks represent amount of rotation done by the wheels
    distance = calculateDisance(encoderTicks);
    // Calulate speed as space/time
    curSpeed = calulateSpeed(deltaDistance, deltaTime);

    if (curSpeed > maxSpeed) maxSpeed = curSpeed;
    avgSpeed = calculateAvgSpeed(distance, totalTime)
}

// Detect impacts using accelerometer
void detectImpacts() {
    int ax, ay, az;
    // Get raw acceleration raw data
    mpu.getAcceleration(&ax, &ay, &az);
    // Evaluate if an impact occurred
    if(newImpact(&ax, &ay, &az))
        impacts++;
}

// Process incoming beacons
void processZigbeeBeacons() {
    if (xbee.available()) {

        ZBRxResponse rx;
        xbee.readPacket();

        if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE) {
            xbee.getResponse().getZBRxResponse(rx);

        // Parse beacon message (simplified)
        if (rx.getType(0) == 'B') { // 'B' for Beacon
            uint16_t anchorId = rx.getData();
            int8_t rssi = rx.getRssi();

            // Update beacon list
            bool found = false;
            for (int i = 0; i < beaconCount; i++) {
                if (beacons[i].anchorId == anchorId) {
                    beacons[i].rssi = rssi;
                    beacons[i].lastSeen = millis();
                    found = true;
                    break;
                }
            }

            // Add new beacon if not found
            if (!found && beaconCount < 10) {
                beacons[beaconCount].anchorId = anchorId;
                beacons[beaconCount].rssi = rssi;
                beacons[beaconCount].lastSeen = millis();
                beaconCount++;
            }
        }
    }
    // Clean up old beacons (not seen in 10 seconds)

    unsigned long now = millis();
```

```
    for (int i = 0; i < beaconCount; i++) {
        if (now - beacons[i].lastSeen > 10000) {

        // Remove by shifting array
        for (int j = i; j < beaconCount - 1; j++) {
            beacons[j] = beacons[j + 1];
        }
        beaconCount--;
        i--; // Recheck this position
    }

}

// Send position and sensor data to coordinator
void reportData() {
    // Check if
    if (beaconCount < 3 || millis() - lastReportTime < 5000) {
        return;
    }

    // Prepare data packet
    int payload[32];
    payload = preparePayload('D', 'FL-103', impacts, disance, maxSpeed,
                             avgSpeed, beacons)

    // Send to coordinator (address 0x0000)
    ZBTxRequest tx = ZBTxRequest(XBeeAddress64(0x00, 0x00), payload, idx);
    xbee.send(tx);

    lastReportTime = millis();
}

void setup() {
    // -- Setup sensor pins -- //
    // -- Setup interrupts for encoder -- //
    // -- Setup ZigBee -- //
}

void loop() {
    processOdometry();
    detectImpacts();
    processZigbeeBeacons();

    // Report data when appropriate
    reportData();
}
```