Challenge 4 - EQ1

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 the total area of more than 1 km², it's possible to assume that around 50 forklifts will be deployed

Concept

In order to track forklifts position in the outdoor yard, a **GPS module** can be used. Inside the underground warehouse of the facility, where GPS is unavailable, a system based on **RFID semi-passive tags** can be implemented: placing **fixed RFID tags** throughout underground space at regular intervals and equipping each forklift with an **RFID reader** it's possible to determine the position of each forklift by reading the hard-coded position data in each tag when the forklift passes near it.

All position data is transmitted by the forklifts to a central server via LoRaWAN at regular intervals, monitoring the entire fleet nearly in real-time, including the status of each unit. Inside every sent message, information collected by their onboard sensors is also provided: battery level, traveled distance, speed and a warning on collisions and impacts.

Given the dimension of the facility, LoRaWAN is a good fit due to its capability of long range communication, even tough it limits the update rate due to the rising of conflicts.

Required Hardware

Forklifts

Every forklift must be equipped with an **Arduino MKR WAN 1310** board (~45€ per board), which is LoRaWAN-enabled and can support GPS boards and RFID readers.

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)
- **RFID reader**: to read the semi-passive tags placed around the facility (~15€ per piece)
- **GPS**: to locate the forklift in the outdoor yard (~10€ per piece)

RFID Tags

Semi-passive RFID tags will be placed throughout the underground warehouse at strategic positions. Each tag contains hard-coded coordinates that will be read by the forklift's RFID reader when in proximity.

Given the total indoor area, and considering an average read range of approximately 5-10 meters for semi-passive RFID tags, around 50 tags would be required.

The cost per RFID semi-passive tag is approximately 2-3€, making this a cost-effective solution.

Communication Protocol

LoRaWAN protocol has been chosen to implement this solution because it offers:

- Long-range communication
- Low power consumption
- Cost-effective infrastructure

Each forklift sends data to a **LoRaWAN gateway** that forwards messages to the central server. The transmission frequency will be set to one message every 30 seconds, which is enough to have an estimate on the area in which forklifts are operating without incurring in conflicts between messages.

System Architecture

The process of gathering, cleaning and structuring raw data is called **Data Wrangling** and follows this flow:

1. **Ingestion**:
   Before all, the forklifts send data periodically to the LoRaWAN gateway which routes it to the server
2. **Discovery**:
   A preliminary analysis is done to identify missing values or anomalies.
3. **Data Cleaning**:
   Correct or delete wrong, duplicated or missing data in order to improve quality. In this specific case we verify the RFID position data and then check for missing data in received payloads.
4. **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:

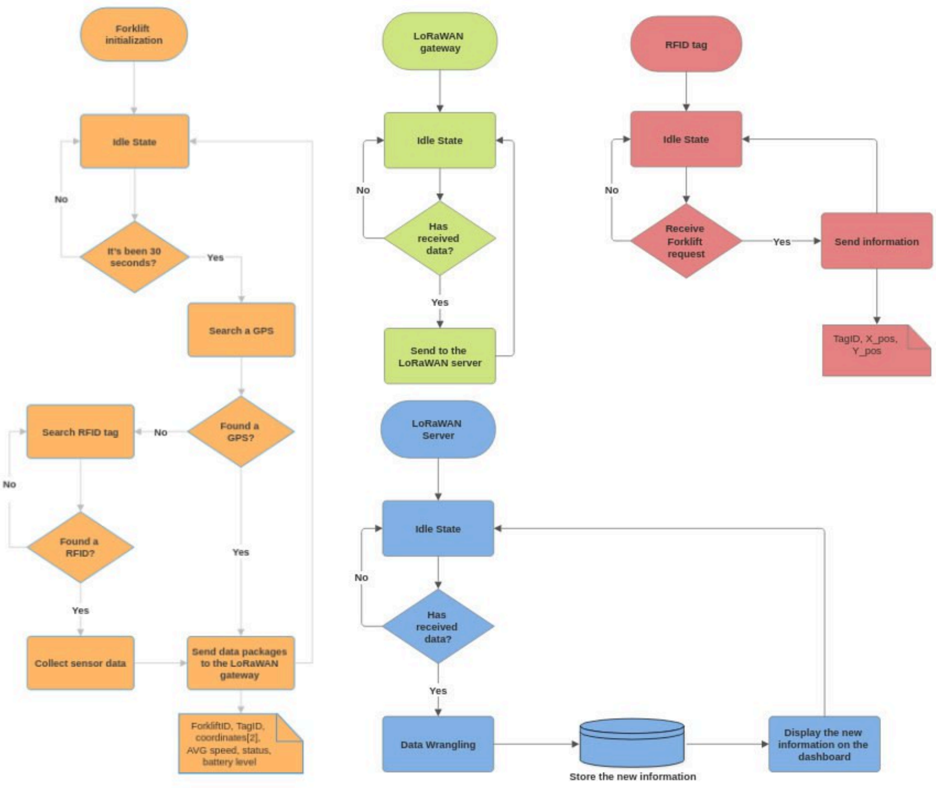
```
{
  "_id": ObjectId("..."),
  "timestamp": ISODate("2025-05-16T14:30:00Z"),
  "forklift_id": "FL-103",
  "status": "active",
  "avg_speed": 4.3,
  "position": {
    "type": "Point",
    "coordinates": [45.23, 12.87] // [x, y] coordinate
  },
  "battery_level": 78,
  "rfid_tag": "TAG-42",
  "tag_read_time": ISODate("2025-05-16T14:29:45Z")
}
```

Data Visualization

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

System-wide block diagram

Assuming all forklifts have been properly equipped and configured:



Forklift pseudo-code

```
#include <Wire.h>
#include <MKRWAN.h>  // LoRaWAN Library
#include <MPU6050.h> // Accelerometer Library
#include <MFRC522.h> // RFID Library
#include <MKRGPS.h>  // GPS Library for Arduino MKR GPS Shield

//// LoRaWAN communication ////
LoRaModem modem;
String appEui = "XXXXXXXXXXXXXXXX";
String appKey = "XXXXXXXXXXXXXXXX";
int lastTransmissionTime = 0;
const int transmissionInterval = 30000; // 30 seconds

//// Position tracking ////
float positionX = 0.0;
float positionY = 0.0;

String tagId = "";
int lastTagReadTime = 0;

int lastGPSReadTime = 0;
const int GPS_READ_INTERVAL = 5000; // Read GPS every 5 seconds
const int GPS_TIMEOUT = 10000; // Consider GPS unavailable after 10 seconds

//// Sensor data ////
long encoderTicks = 0;
float distance = 0.0;
float curSpeed = 0.0;
float maxSpeed = 0.0;
float avgSpeed = 0.0;
int impacts = 0;

MPU6050 accelerometer;
MFRC522 rfid(SS_PIN, RST_PIN);

// 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++;
}

// Read GPS position for outdoor tracking
void readGPSPosition() {
  // Check if it's time to read GPS
  if (millis() - lastGPSReadTime < GPS_READ_INTERVAL) {
    return;
  }

  if (GPS.available()) {
    float latitude = GPS.latitude();
    float longitude = GPS.longitude();

    // Check if GPS data is valid
    if (GPS.satellites() >= 4 && latitude != 0.0 && longitude != 0.0) {
      // Convert GPS coordinates to local warehouse coordinate system
      positionX = convertLatToLocalX(latitude);
      positionY = convertLonToLocalY(longitude);

      lastGPSReadTime = millis();
    }
  }
}

// Read RFID tags to determine position (when GPS unavailable)
void readRFIDTags() {
  // Only use RFID if GPS is not available
  if (GPS.available()) {
    return;
  }

  if (rfid.PICC_IsNewCardPresent() && rfid.PICC_ReadCardSerial()) {
    String newTagId = "";
    for (byte i = 0; i < rfid.uid.size; i++) {
```

```
      newTagId += (rfid.uid.uidByte[i] < 0x10 ? "0" : "") +
                 String(rfid.uid.uidByte[i], HEX);
    }

    // Read position data from tag (simplified)
    byte buffer[18];
    byte size = sizeof(buffer);

    // Read block where position is stored
    rfid.MIFARE_Read(POSITION_BLK, buffer, &size);

    // Extract position data (simplified)
    float x = (buffer[0] << 8 | buffer[1]) / 100.0;
    float y = (buffer[2] << 8 | buffer[3]) / 100.0;

    // Update position
    positionX = x;
    positionY = y;
    tagId = newTagId;

    lastTagReadTime = millis();

    rfid.PICC_HaltA();
    rfid.PCD_StopCrypto1();
  }
}

// Send position and sensor data to server via LoRaWAN
void reportData() {
  // Check if it's time to send data
  if (millis() - lastTransmissionTime < transmissionInterval) {
    return;
  }

  // Update position source status before reporting
  updatePositionStatus();

  // Prepare data packet with position source information
  char payload[60];
  sprintf(payload, "%s,%d,%.2f,%.2f,%.2f,%d,%.2f,%.2f,%s",
          "FL-103",                    // Forklift ID
          getBatteryLevel(),           // Battery level
          distance,                    // Total distance traveled
          maxSpeed,                    // Maximum speed
          avgSpeed,                    // Average speed
          impacts,                     // Number of impacts detected
          positionX,                   // X position
          positionY,                   // Y position
          positionSource.c_str());     // Position source (GPS/RFID/NONE)

  // Send data via LoRaWAN
  modem.beginPacket();
  modem.print(payload);
  modem.endPacket(true);  // true for confirmed transmission

  lastTransmissionTime = millis();
}

void setup() {
  // Initialize serial communication
  Serial.begin(9600);

  // Initialize GPS
  if (!GPS.begin()) {
    Serial.println("Failed to initialize GPS!");
  }

  // Initialize LoRaWAN
  while (!modem.begin(EU868)) {
    Serial.println("Failed to start LoRaWAN module");
  }

  // Connect to LoRaWAN network
  int connected = modem.joinOTAA(appEui, appKey);
  while (!connected) {
    Serial.println("Failed to connect to LoRaWAN network");
  }

  // Initialize RFID reader
  rfid.PCD_Init();
  Serial.println("RFID reader initialized");

  // Initialize accelerometer
  accelerometer.initialize();

  // Setup sensor pins and interrupts for encoder
  pinMode(ENCODER_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(ENCODER_PIN), encoderISR, RISING);

  Serial.println("Forklift tracking system initialized");
```

```
  Serial.println("GPS primary, RFID fallback positioning active");
}

void loop() {
  processOdometry();
  detectImpacts();
  readGPSPosition();
  readRFIDTags();
  reportData();
}
```