POLITECNICO
MILANO 1863

Internet of Things Homework

Andrea Pesciotti

10715428

Teachers:
Redondi Alessandro Enrico Cesare
Fabio Palmese
Boiano Antonio

May 2025

1 Exercise 1: Low-Cost IoT System for Forklift Tracking and Monitoring

In this exercise, I was tasked with designing an IoT system to localize electric forklifts and monitor their status (distance, speed, impacts). This system needs to operate across a 500 m^2 underground indoor area and a 1 km^2 outdoor yard. My design priorities were simplicity, low cost, and scalability, while addressing the challenges posed by the mixed operational environment.

1.1 Hardware on Each Forklift

I propose that each forklift unit be powered from its electrical system via a DC-DC converter and comprise the following:

- **Processing Unit:** My choice is an **ESP32 microcontroller**. This is due to its low cost, sufficient processing power for handling sensor data and basic logic (like impact detection), and its built-in Wi-Fi and Bluetooth capabilities, which I plan to use as part of the indoor localization strategy.
- **Sensors:**
  - *Outdoor Localization & Motion:* I selected a standard **GPS module** (e.g., u-blox NEO-6M series) to provide reliable outdoor positioning, speed, and distance data. GPS signals, however, will be unavailable in the underground section.
  - *Indoor Localization Aid & Motion/Impact:* For impact detection and general motion assistance, I chose an **IMU (Inertial Measurement Unit, e.g., MPU-6050)**. This sensor's accelerometer is key for robust impact detection. To address indoor localization where GPS fails, I propose a hybrid approach: the ESP32 will perform Wi-Fi RSSI (Received Signal Strength Indicator) scanning and will also scan for signals from low-cost **BLE (Bluetooth Low Energy) beacons**. These beacons would be strategically deployed throughout the 500 m^2 underground area. My aim with this hybrid BLE/Wi-Fi approach is to achieve acceptable zone-level accuracy, acknowledging that precise coordinate-based indoor tracking with low-cost sensors in such an environment is challenging due to signal attenuation and multipath propagation. The IMU can also help smooth location estimates between less frequent indoor RF-based updates.
- **Communication Module:** My choice is a **LoRaWAN module** (e.g., RFM95W). This selection is driven by LoRaWAN's capability for long-range communication, which is essential for covering the expansive 1 km^2 outdoor yard and for penetrating the challenging underground environment. It also aligns with the low-cost and relatively low-power profile desirable for such units.

Comment: I believe this hardware suite offers a strong balance between cost-effectiveness and functionality. The ESP32 is capable of managing the sensors and communication, the sensor combination addresses all specified monitoring needs (outdoor/indoor location, motion, impact), and LoRaWAN provides a suitable communication backbone for a private network deployment, critical for managing coverage in the varied operational zones.

1.2 Connectivity Strategy

I designed the connectivity strategy with LoRaWAN at its core, primarily due to its suitability for this project's specific requirements concerning cost-effectiveness and the critical need for controlled coverage, especially in the underground area.

- **Communication Protocol:** My choice remains **LoRaWAN**. It is well-suited for transmitting the necessary small data packets (location data, status updates, and immediate impact alerts) over the required distances, including into the challenging underground sections where other signals might struggle.
- **Network Architecture:** I propose establishing a **private LoRaWAN network**. This involves:
  - *LoRaWAN Gateways:* I would strategically place several gateways. For the 1 km^2 outdoor area, one or two well-placed outdoor gateways might suffice. However, for the 500 m^2 underground zone, dedicated gateways *inside* or at its immediate entrances will likely be necessary to counteract signal attenuation from earth and building materials. The exact number and optimal positions must be determined by a thorough site survey, which is a critical step for ensuring reliable underground coverage.
  - *LoRaWAN Network Server (LNS):* To minimize ongoing operational costs and retain data control, I selected a self-hosted open-source LNS, such as **The Things Stack (TTS)**. This can be run on-premises.
- **Data Transmission Frequency:** My strategy aims to balance the need for timely updates with network capacity and LoRaWAN's inherent duty cycle regulations:
  - *Regular Updates (Location/Status):* I decided on a transmission interval of every 1-2 minutes when a forklift is detected as moving. This frequency can be reduced to every 10-15 minutes when the forklift is stationary for an extended period.
  - *Impact Alerts:* Should an impact be detected by the IMU, an alert packet will be transmitted immediately by the forklift unit.

Comment: My decision to use LoRaWAN is based on its advantages in cost, private network control, and range, which are crucial for this mixed indoor/outdoor underground environment. While LoRaWAN's update frequency of 1-2 minutes is not continuous sub-second tracking, it provides sufficient "real-time" information for operational oversight, asset location, and reviewing historical data in a logistics context. More importantly, event-triggered alerts like impacts can be sent with high priority.

The primary challenge is the underground portion. Unlike public cellular networks (e.g., NB-IoT) where coverage underground can be unreliable or non-existent (and not easily augmented by the company), a private LoRaWAN network allows me to directly address these dead zones by installing dedicated gateways. This control over infrastructure is paramount for ensuring consistent tracking. While signal attenuation and multipath propagation are concerns underground for any RF technology (including the BLE/Wi-Fi used for indoor localization), LoRaWAN's robustness and the ability to densify gateway/beacon infrastructure make it a more pragmatic choice than relying on external network providers for critical coverage in such a challenging zone. The cost of deploying additional LoRaWAN gateways or BLE beacons to ensure underground functionality is generally lower and more controllable than alternatives for achieving similar private network coverage.

Backend Architecture

For the backend of the forklift tracking system, my design focuses on a layered architecture that handles data ingestion, processing, storage, and visualization efficiently. I have selected a stack of primarily open-source tools to ensure low cost, flexibility, and inherent scalability.

- **Data Ingestion Layer:**
  - *Component:* The LoRaWAN Network Server (LNS), specifically a self-hosted **The Things Stack (TTS)**, is responsible for receiving encrypted data from the forklifts via the gateways. After decryption, TTS forwards this uplink data to the next stage using **MQTT** as the messaging protocol.
- **Data Processing & Logic Layer:**
  - *Component:* **Node-RED** serves as this layer. Its role is to subscribe to the MQTT data stream, parse the incoming payloads, implement the logic for indoor localization (e.g., by fusing Wi-Fi scan data and BLE beacon proximity information), aggregate necessary statistics (like daily distance or average speeds, if these are not fully computed on the edge devices), trigger alerts for critical events like impacts, and then route the processed and enriched data to the storage system.
- **Data Storage Layer:**
  - *Component:* A specialized time-series database (TSDB), specifically **InfluxDB**, is responsible for persistently storing all relevant time-series data. This includes forklift locations, speed, distance traveled, impact events, and other sensor readings over time.
- **Data Visualization & Alerting Layer:**
  - *Component:* **Grafana** provides the human interface to the system. Its role is to allow logistics managers and staff to view forklift locations in real-time, monitor their operational status, analyze historical data through dashboards and charts, and receive notifications for critical events. Grafana will also handle the alerting mechanism based on predefined rules.

Comment: I selected this integrated stack—The Things Stack (LNS) for network management, MQTT for data transport, Node-RED for processing logic, InfluxDB for data storage, and Grafana for visualization and alerting—because it forms a cohesive, robust, and highly cost-effective solution for this IoT application. Each component is open-source, which significantly reduces software licensing costs and aligns with the "low-cost" project priority.

The modular nature, with communication via MQTT, provides excellent flexibility; individual components can be updated or even replaced without overhauling the entire backend. Node-RED was chosen for the processing layer due to its visual flow-based programming model, which accelerates development and simplifies the creation of data workflows suitable for parsing, indoor localization rule application, and data routing. Time-series databases like InfluxDB are specifically designed for the high-volume, timestamped data typical of IoT systems, offering superior performance for relevant queries and data management compared to traditional relational databases for this use case. Finally, Grafana is a powerful and versatile open-source platform for creating comprehensive dashboards and managing alerts, integrating seamlessly with the chosen database technologies.

This entire stack is also inherently scalable; while self-hosting requires managing the infrastructure, these tools are capable of handling a growing fleet of forklifts and increasing data volumes with appropriate resource allocation, thus meeting the "scalability" requirement. The overall approach prioritizes "simplicity" in terms of leveraging well-established tools with clear roles for the defined tasks.

1.3 Pseudocode for On-Forklift Logic

The logic covers data collection from sensors, processing for status monitoring (including impact detection), and managing LoRaWAN communication.

```
// --- Constants ---
DEFINE IMPACT_G_THRESHOLD AS 2.5
DEFINE MOVING_UPDATE_INTERVAL AS 90 // seconds
DEFINE STATIONARY_UPDATE_INTERVAL AS 750 // seconds
DEFINE MOTION_TIMEOUT AS 300 // seconds

// --- Variables ---
VAR isMoving : BOOLEAN = FALSE
VAR timeOfLastMovement : INTEGER = 0
VAR dailyDistanceMeters : REAL = 0.0
VAR maxSpeedTodayKmh : REAL = 0.0
VAR totalSpeedSumKmh : REAL = 0.0     // For calculating average speed
VAR speedReadingsCount : INTEGER = 0     // For calculating average speed
VAR lastTxTime : INTEGER = 0
VAR currentLoc : LocationType = NULL

PROCEDURE Initialize_System():
    CALL Initialize_Sensors() // GPS, IMU
    CALL Initialize_LoRaWAN() // Join network

    dailyDistanceMeters = Load_Persistent("daily_distance", 0.0)
    maxSpeedTodayKmh = Load_Persistent("max_speed", 0.0)
    totalSpeedSumKmh = Load_Persistent("total_speed_sum", 0.0)
    speedReadingsCount = Load_Persistent("speed_readings_count", 0)
    // Consider daily reset logic for stats

PROCEDURE Main_Loop():
    VAR currentTime : INTEGER = Get_System_Time()
    VAR gps : GpsData // Contains fix, location, speed
    VAR imu : ImuData // Contains acceleration vector
    VAR calculatedSeverity : REAL
    VAR calculatedSpeedKmh : REAL

    gps = Read_GPS()
    imu = Read_IMU()

    // 1. Impact Detection
    calculatedSeverity = Get_GForce(imu.acceleration)
    IF calculatedSeverity > IMPACT_G_THRESHOLD THEN
        CALL Send_LoRaWAN_Alert("IMPACT", calculatedSeverity, gps.fix ? gps.location : currentLoc)
    ENDIF

    // 2. Location, Motion & Speed Update
    IF gps.fix THEN
        currentLoc = gps.location
        calculatedSpeedKmh = gps.speed
        IF Is_Significant_Movement(gps.location, previousGpsLoc) THEN
            dailyDistanceMeters += Get_Distance_Moved_GPS(previousGpsLoc, gps.location)
        ENDIF
        isMoving = (calculatedSpeedKmh > 0.5)
        previousGpsLoc = gps.location
    ELSE // Indoor or no GPS
        currentLoc = Estimate_Indoor_Zone_From_RF_Scans() // Uses BLE/Wi-Fi
        isMoving = Detect_Movement_IMU(imu.acceleration)
        IF isMoving THEN
            dailyDistanceMeters += Estimate_Distance_IMU(imu.acceleration)
        ENDIF
        calculatedSpeedKmh = isMoving ? Estimate_Speed_IMU(imu.acceleration) : 0
    ENDIF

    // 3. Update Statistics
    IF isMoving THEN
        timeOfLastMovement = currentTime
        IF calculatedSpeedKmh > maxSpeedTodayKmh THEN maxSpeedTodayKmh = calculatedSpeedKmh ENDIF
        totalSpeedSumKmh += calculatedSpeedKmh
        speedReadingsCount += 1
    ELSE
        IF (currentTime - timeOfLastMovement > MOTION_TIMEOUT) THEN isMoving = FALSE ENDIF
    ENDIF

    // 4. Periodic LoRaWAN Status Update
    VAR updateInterval : INTEGER = isMoving ? MOVING_UPDATE_INTERVAL : STATIONARY_UPDATE_INTERVAL
    IF (currentTime - lastTxTime > updateInterval) THEN
        VAR avgSpeedKmh : REAL = (speedReadingsCount > 0) ? (totalSpeedSumKmh / speedReadingsCount) : 0
        VAR payload : ByteArray = Create_Payload(currentLoc, dailyDistanceMeters, calculatedSpeedKmh,
                                               maxSpeedTodayKmh, avgSpeedKmh, isMoving)
        IF Send_LoRaWAN_Data(payload) = SUCCESS THEN
            lastTxTime = currentTime
            CALL Save_Persistent("daily_distance", dailyDistanceMeters)
            CALL Save_Persistent("max_speed", maxSpeedTodayKmh)
            CALL Save_Persistent("total_speed_sum", totalSpeedSumKmh)
            CALL Save_Persistent("speed_readings_count", speedReadingsCount)
        ENDIF
    ENDIF

    CALL Enter_Sleep_Mode_Briefly()
END PROCEDURE
```

Listing 1: *Core On-Forklift Logic*

1.4 System-Wide Building Block Diagram

I envision the system architecture as a flow from the forklifts to the backend, enabling data-driven insights for the logistics company.

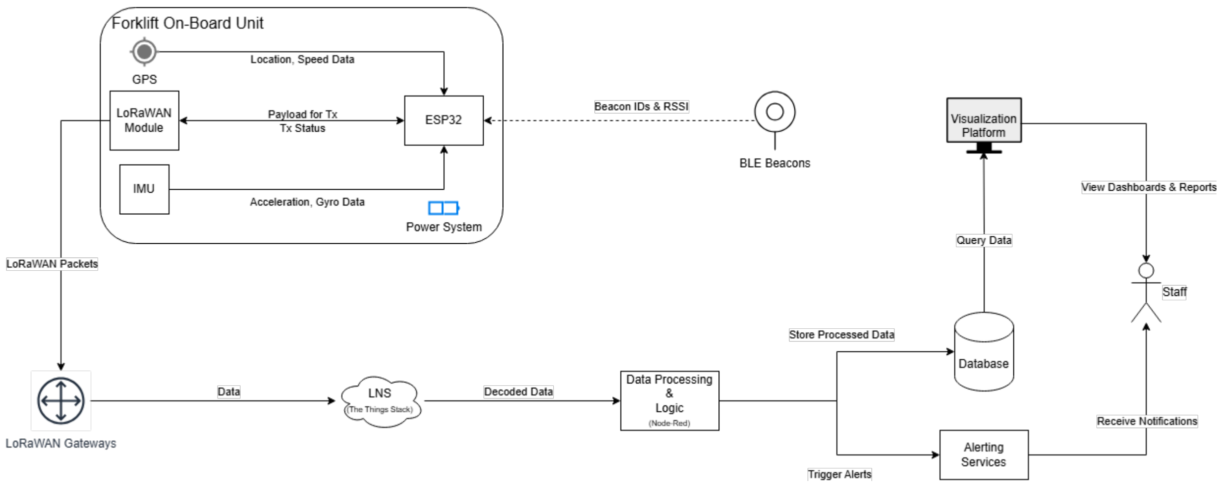

Figure 1: *High-Level System Architecture Diagram.*