UNIVERSITY OF PISA

Artificial Intelligence and Data Engineering

Internet of Things

SeedBot: Automated Sowing Device

Francesco Nocella, Noemi Cherchi

Academic Year 2023/2024

Contents

Introduction

SeedBot is an automated system designed for agricultural sowing. It uses a network of wireless sensors to monitor soil conditions and control the seed distribution mechanism, improving efficiency and reducing manual labor. Using a pre-trained machine learning model, SeedBot can determine the most suitable crop for a specific plot of land.

Use Case

The agricultural sector plays a crucial role in the industry, specifically when considering large-scale food production, which requires high efficiency, a strict quality check and optimized comunication among the different phases of the production process. However, traditional farming methods are labor-intensive and often inefficient. In the contest of smart industries, the use of IoT devices and machine learning algorithms can significantly improve agricultural processes.

SeedBot aims to address these challenges by automating the sowing process, optimizing seed distribution based on real-time soil data, and reducing the need for manual intervention. This allows the sowing to happen in in optimal conditions, minimazing errors and the waste of resources.

SeedBot receives real-time data from various sensors, including nitrogen, phosphorus, and potassium (NPK) levels, soil moisture, pH, and air temperature. This data is used to determine the best seed type for the soil conditions and control the seed distribution mechanism accordingly. The position and the results of the sowing are then sent to a central server for monitoring and analysis, which allows farmers to correct anomalies and optimize the sowing process.

This means that SeedBot allows to reduce operational costs and production time, helping to make the farms more competitive on the global market.

System Architecture and Load Distribution

In this project, six nRF52840 devices are employed to implement the testing of a fully automated sowing system. These devices are specifically organized into roles for sensors and actuators to ensure efficient communication, data collection, and actuation. The system is designed to optimize task distribution across devices while maintaining scalability and performance.

The system architecture is composed of the following elements

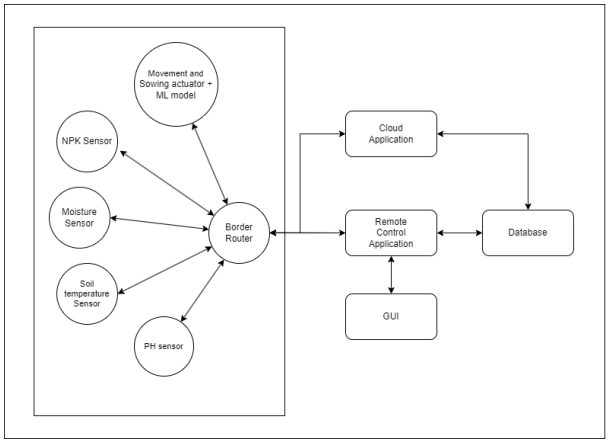

Figure 2.1: SeedBot Architecture

- **Border Router:**
  - Acts as the central communication hub, managing all data exchanges between sensors, actuators, and external networks. It handles routing, ensuring efficient data transmission, and maintains the network's topology.
- **Sensors:**
  - **NPK Sensor:** Measures the soil's nitrogen, phosphorus, and potassium levels.
  - **Moisture Sensor:** Monitors soil moisture levels to ensure optimal growing conditions.
  - **PH Sensor:** Measures the soil pH levels, helping optimize the sowing conditions based on acidity or alkalinity.
  - **Temperature Sensor:** Collects soil temperature data to help in seed germination analysis.
- **Actuator:**
  - **Movement and Sowing Actuator:** Controls the robot's movement and seed distribution. Decisions are made based on input from the machine learning model to ensure efficient sowing.
- **Applications:**
  - **Cloud Application:** Handles the storage and processing of sensor data in a central database.
  - **Remote Control Application:** Provides a graphical user interface that allows users to remotely control the system and monitor real-time data.

This balanced division of labor ensures that the workload is effectively shared among the devices, allowing the system to operate smoothly and in real-time while minimizing latency and maximizing resource utilization.

Machine Learning Model

The initial dataset was taken from kaggle (link). The dataset contains information about different crops and their requirements in terms of soil conditions. We selected the most relevant ones for our model, which included soil pH, nitrogen, phosphorus, potassium, temperature, and moisture.
The types of crop include: rice, chickpea, kidney beans, lentil, pomegranate, banana, mango, papaya, etc.
The dataset was preprocessed to remove missing values and was then split into training and testing sets. To create the model, multiple machine learning methods were used and evaluated for their accuracy in predicting crop suitability. In particular, classifiers like Logistic Regression, Random Forest, Gradient Boosting, Support Vector Machines (SVM), Naive Bayes, Decision Tree, and K-Nearest Neighbors (KNN) were used.
The Decision Tree algorithm was chosen for its high accuracy and interpretability and its simplicity, and using the library `emlearn` it was exported as header file and integrated into the SeedBot system to make real-time crop recommendations, providing a practical tool to optimize agricultural production.

Code and Software

The code for SeedBot is written in C and Python. We decided to use json format to exchange data between the devices for its simplicity and readability. The nRF52840 devices run the C code, while the cloud and remote applications are written in Python. The code is available on GitHub at the following link.

C Code

The actuator code is responsible for controlling the movement and seed distribution. It retrieves data from sensors and applies the machine learning model to determine autonomously the type of seed to sow. The sensors code simulates data readings and sends them to the border router for processing.

Actuator Workflow

1. *Initialization and Registration:*
   - The first step involves registering with the CoAP server. This process includes sending a registration request to the server with the actuator's name and its IPv6 address. The registration can be attempted 5 times.
   - To notify the active status of the bot, a yellow LED is turned on.
   - The actuator activates its corresponding resources.

```
1        while (max_registration_retry > 0)
2        {
3            coap_endpoint_parse(SERVER_EP, strlen(SERVER_EP), &server_ep);
4            coap_init_message(reg_request, COAP_TYPE_CON, COAP_POST, 0);
5            coap_set_header_uri_path(reg_request, REGISTER_URL);
6            const char msg[] = "sowing_actuator";
7            coap_set_payload(reg_request, (uint8_t *)msg, sizeof(msg) - 1);
8
9            // Send the registration request and handle the response
10           COAP_BLOCKING_REQUEST(&server_ep, reg_request, client_chunk_handler);
11
12           if (max_registration_retry > 0)
13           {
14               // Wait and retry if registration failed
15               etimer_set(&timer, 15 * CLOCK_SECOND);
16               PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
17
18               // Decrement retry count if registration failed
19               max_registration_retry--;
20           }
21       }
```

2. *Discovery of sensor IP addresses:*
   - After the registration, a CoAP discovery request to the coap server is performed to find the IP addresses of the sensors: `npk`, `ph`, `moisture`, and `temperature`. Each answer is analyzed to extract and store their IP addresses.
3. *Retrieving sensor data and determining the seeding type:*
   - Once the command is received, a CoAP GET request is performed to collect data from the sensors.
   - The decision tree model is applied to the sensor data to determine the type of seed to sow.
4. *Simulating the seeding operation:*

- The seeding operation is simulated by introducing a delay that represents the time taken for seeding.
- A green LED is turned on during the seeding operation.
- The actuator can start or pause the movement of the robot based on the pressing of the button that modifies the `active` variable. The modification of the status is notified by an observable event.

```
PROCESS_THREAD(button_process, ev, data)
{
    PROCESS_BEGIN();

    while (!exit_flag)
    {
        // Wait for the button release event
        PROCESS_YIELD();

        if (ev == button_hal_press_event)
        {
            // Check if the event is indeed a button release event
            if (is_movement_active(&mov_data))
            {
                stop_movement(&mov_data);
                printf("Pause request initiated via button\n");
            }
            else
            {
                start_movement(&mov_data);
                printf("Movement request initiated via button\n");
            }
        }
    }

    PROCESS_END();
}
```

5. *Updating the Central CoAP Server:*
   - A CoAP message is sent to the central server with details of the seeding operation, including the type of seed used, the current row and column, and the sensor values.
6. *Restarting Until the Field is Completely Sowed:*
   - The process loops back until the entire field is completely sowed.

Sensors Workflow

1. *Initialization and Registration:*
   - Each sensor is initialized by registering with the central CoAP server. The sensor's name and its IPv6 address is sent to the server.
2. *Activation of Sensor Resources:*
   - Once registration is successful, the sensor activates its corresponding resources (e.g., moisture, pH, NPK, temperature) to be available for CoAP requests.
3. *Data Simulation:*
   - Each sensor simulates data readings based on predefined statistical distributions. The values are constrained to realistic ranges for each parameter using as reference the data from the dataset.
4. *Handling CoAP GET Requests:*
   - Each sensor responds to CoAP GET requests. It the simulated data and returns it in a JSON format within the CoAP response.

```
1  static void res_get_handler(coap_message_t *request, coap_message_t *response,
       uint8_t *buffer, uint16_t preferred_size, int32_t *offset)
2  {
3      int moisture = simulate_soil_moisture();
4      coap_set_header_content_format(response, APPLICATION_JSON);
5      int payload_len = snprintf((char *)buffer, preferred_size, "{\"moisture\":%d}",
           moisture);
6      coap_set_payload(response, buffer, payload_len);
7  }
```

5. *Retry Mechanism for Registration:*
   - If the initial registration attempt fails, the sensor will retry registration for a predefined number of times, introducing a delay between each attempt.
6. *Operational Loop:*
   - Once registered, the sensor enters a loop where it continues to respond to incoming CoAP requests while maintaining its simulated data. The process continues indefinitely as long as the sensor is active.

Python Code

The Python code is responsible for managing the cloud and remote applications. The cloud application stores data collected from the sensors in a database, while the remote control application allows users to interact with the system via a graphical interface.

There are two comunication protocols: HTTP (through the Flask library) and CoAP (Constrained Application Protocol). The HTTP protocol is used for the cloud application, while the CoAP protocol is used by the remote control application.
Flask is used to create a RESTful API that allows users to interact with the system. Through the graphical interface, users can initiate, pause and completely stop the sowing process. The API provides endpoints for registering sensors, receiving sensor data, and storing it in the database. Additionally, it handles communication with the actuators via CoAP to manage the different stages of the sowing process, including start, idle, and stop.

The server CoAP is used to manage the communication between the sensors and the actuators. It is responsible for registering sensors, receiving sensor data, and sending commands to the actuators. It provides endpoints for sensor registration, data retrieval, and actuator control allowing the system to respond in real-time to the commands sent from the Flask-based user interface.

Workflow

1. *Sensor Registration:*
   - The CoAP server receives registration requests from the sensors and stores their information in the database using the `db_manager_mysql` module. Each sensor is assigned a unique identifier and its status is tracked in the database. The server confirms the registration and handles any registration errors.
2. *Data Retrieval:*
   - The CoAP server responds to CoAP GET requests from sensors or actuators. The data is read from the database.

```
    def render_POST_advanced(self, request, response):

        print("Received advanced POST request.")

        try:
            payload_str = request.payload
            print(f"Raw payload: {payload_str}")

            if not payload_str:
                response.code = defines.Codes.BAD_REQUEST.number
                response.payload = "No payload received"
                return self, response

            # Parse the JSON payload
            payload = json.loads(payload_str)
            print(f"Parsed JSON payload: {payload}")


            global received_data
            received_data.update(payload)
```

3. *Sowing Process Management:*
   - The cloud application monitors and manages the sowing process by sending start, idle, or stop commands to the actuators via the CoAP server.
4. *Database Interaction:*
   - Both `app.py` and `coap_server.py` interact with the MySQL database through the `db_manager_mysql.py` module. This module handles all CRUD operations, ensuring that sensor data, actuator statuses, and other critical information are properly stored and retrieved as needed.

```
1         def create_tables():
2             try:
3                 engine = create_engine(DATABASE_URL)
4
5                 Base.metadata.create_all(engine)
6                 logger.info(f"Tables successfully created in the database '{engine.url.
                      database}'.")
7
8             except SQLAlchemyError as e:
9                 logger.error(f"Error during the creation of tables: {str(e)}")
```

Grafana

Grafana is an open-source platform for monitoring and observability. It allows you to query, visualize, alert on, and understand your metrics no matter where they are stored. In this project, Grafana is used to visualize the database data and analytics.

Figure 5.1: Grafana Dashboard

Figure 5.2: Graphic showing maximum and minimum statistics for various plant growth parameters such as temperature, pH, humidity, and nutrients (N, P, K), grouped by sowing state.

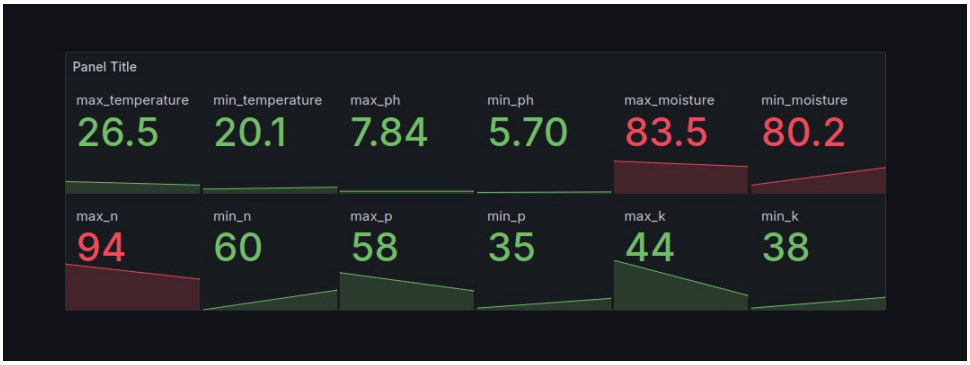

Figure 5.3: Graphic showing the average of soil parameters such as temperature, pH, humidity, and nutrients (N, P, K) for each field.

Conclusion

Future Implementation

In the future, SeedBot could be improved by including:

- **Advanced Machine Learning Algorithms:** To further optimize seed distribution and adapt to various weather and soil conditions, enabling predictive analytics for sowing times and crop yields.
- **Advanced Remote User Interface:** Using platforms like Grafana to monitor sensor data in real-time and provide a more interactive user interface, allowing users to set thresholds for various parameters (e.g., soil moisture, pH levels) and receive automated alerts.
- **Integration with Irrigation and Nutrition Systems:** To provide a complete crop management system that includes not only sowing but also irrigation and plant nutrition, with intelligent scheduling based on environmental data.
- **Energy Efficiency and Sustainability Improvements:** Introducing low-power modes and energy-efficient protocols to extend the battery life of the devices and explore the use of renewable energy sources like solar panels to power the system in the field.
- **Blockchain for Data Integrity:** Utilizing blockchain technology to ensure the integrity of data collected from the sensors, providing transparent and tamper-proof records of farming activities, useful for certifications and traceability.

Conclusion

This project has demonstrated that handling advanced technologies, such as IoT devices and machine learning, can greatly improve the agricultural sector. The power of nRF52840 devices and the CoAP protocol helps to utilize the resources in an efficient and precise way. Automating key processes like soil monitoring and seed sowing, means that the need for manual intervention reduces significantly. This approach has the potential to revolutionize farming practices, making them more sustainable and productive. The successful implementation of this system paves the way for further innovations in smart agriculture, ultimately contributing to a more efficient and data-driven approach to farming.