UNIVERSITÀ DI PISA

Department of Information Engineering – MSc AIDE

Internet of Things – University of Pisa

GreenHouse

Aida Himmiche

Supervised by:
Prof. Giuseppe Anastasi
Prof. Carlo Vallati
Prof. Francesca Righetti

ABSTRACT

The objective of this project is to build an IoT telemetry and control system to simulate a smart environment for a greenhouse, using which it is possible to monitor the humidity of soil and automatically activate or deactivate irrigation protocols as needed.

This application was developed within the context of the course *Internet of Things* at University of Pisa, and as such it was implemented using the tools introduced and covered in the learning material.

This project demonstrates an understanding of the MQTT and CoAP networks, the flow of data between the different devices in the control system, and an unambiguous navigation of the Contiki-NG operating system, including the Cooja tool.

***Keywords**: Internet of Things, Smart Environment, Control System, Contiki-NG, Cooja, MQTT, CoAP.*

TABLE OF CONTENTS

ABSTRACT **2**

TABLE OF CONTENTS **3**

INTRODUCTION **4**

- Functionalities 4
- Technologies Used 4

NETWORKS **4**

- MQTT 4
- CoAP 7

SIMULATION **11**

MySQL **13**

DATA ENCODING **14**

CONCLUSION **14**

References **15**

INTRODUCTION

GreenHouse was developed for the purpose of monitoring humidity in plant soil without the need of human intervention, and its regulation by automatic activation or deactivation of available irrigation protocols. This is achieved by designing a smart system composed of sensors and actuators, as well as software to store and process the collected data.

Functionalities

There are two different networks of IoT devices in the system. First the **MQTT** network, it includes a sensor deployed at the level of the soil, which reports humidity values periodically to a Java collector through a border router. This collector stores the values into a database then sends the appropriate command to the irrigation actuator.

The actuator is deployed on a **CoAP** network, where a device receives single commands of activation and deactivation. For visual representation, the activation of irrigation turns on a green LED light.

If the CoAP resource's response is successful, the Java collector executes simple logic to simulate the regulation of humidity, and the value is again stored in the database for future uses.

Technologies Used

Contikier: The container on which the IoT application runs on top of the Ubuntu VM.
Cooja: A Contiki-NG tool for running a simulation of the system.
MQTT: used for implementing a Mosquitto broker and Mosquitto publisher, deployed on a real Launchpad sensor device.
CoAP: used for implementing the irrigation actuator, deployed on a real Launchpad sensor device.
Java: used for implementing the collector and the logic, runs locally on the Ubuntu VM.
MySQL: the database chosen for this project, runs locally on the Ubuntu VM.
Texas Instruments cc2650 Launchpad: 3 instances were used (border router, MQTT resource observer, CoAP actuator).

NETWORKS

MQTT

Humidity is monitored by the MQTT sensor and sent via a connection established by the MQTT broker, in a publisher/subscriber mechanism. The publisher sends the data periodically on the humidity topic, every 30 seconds, and the Java collector plays the role of the

subscriber to retrieve the message. Here is the code snippet responsible for this at the level of the MQTT node.

```
                if(state == STATE_SUBSCRIBED){

                        //publish humidity percentage every 30 seconds
                        etimer_set(&publish_timer, DEFAULT_PUBLISH_INTERVAL);
                        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&publish_timer));

                        //sprintf(pub_temperature, "%s", "temperature");
                        sprintf(pub_topic, "%s", "humidity");

                        //Random humidity values
                        humidity_perc = (rand() % 40);

                        sprintf(app_buffer, "{\"HUM\":%d}",humidity_perc);

                        printf("%s \n",app_buffer);

                        mqtt_publish(&conn, NULL, pub_topic, (uint8_t *)app_buffer,
strlen(app_buffer), MQTT_QOS_LEVEL_0, MQTT_RETAIN_OFF);
```

There are two timers in the code, a periodical timer which is the duration after which the MQTT client connection is checked, and a publishing timer which specifies the frequency of the messages. The publishing timer is reset at the expiration of each 30s period with the use of the *PROCESS_WAIT_EVENT_UNTIL*.

To simulate a humidity resource observation, a random value in the range of 0 to 40 is generated and the message is composed in the following format: “{Humidity: 31}”. *mqtt_publish()* is the method responsible for sending it via the topic “humidity”.

```
public class ClientMqttHumidity implements MqttCallback{

    String subscriber_topic = "humidity";
    String publisher_topic = "irrigator";
    String broker = "tcp://127.0.0.1:1883";
    String clientId = "Application";
    int value = 0;
    Interface i = new Interface();

    MqttClient mqttClient;

    public ClientMqttHumidity() throws MqttException{
        try {
        mqttClient = new MqttClient(broker,clientId);
        }catch (Error e) {
        }
        mqttClient.setCallback(this);
        mqttClient.connect();
        mqttClient.subscribe(subscriber_topic);

        System.out.println("Subscribing to the " +subscriber_topic+ "topic..\n\n");

    }
```

The message is then received by the Java collector in the *MQTTClientHumidity* class, which contains an MQTT client subscribed to the topic “humidity” and the method MessageArrived(); this extracts the payload, parses the JSON content and stores the humidity value in the dedicated database.

```
public void messageArrived(String topic, MqttMessage message) throws Exception {

    String json_message = new String(message.getPayload());

    //parsing
    JSONParser parser = new JSONParser();
    JSONObject jsonObject = null;
    try {
        jsonObject = (JSONObject) parser.parse(json_message);
        //System.out.println(jsonObject);
    } catch (ParseException e) {
        e.printStackTrace();
    }
    if(jsonObject != null) {
        Long v = (Long) jsonObject.get("HUM");
        this.value = v.intValue();
        System.out.println("Humidity observed is\": "+ this.value);

        if (this.value < 20) {
            i.irrigationReq = true;
        } else i.irrigationReq = false;

          SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
          Date d = new Date();
          String[] tokens = dateFormat.format(d).split(" ");
          String date = tokens[0];
          String time = tokens[1];

        i.storeMqttData(time, date, this.value, i.irrigationReq);
        i.MonitorHumidity();
    }
}
```

After collecting comes processing the data, and that is done using simple logic implemented at the level of the Java application. After the MySQL insertion code, *MonitorHumidity()* is called to check the value of the humidity observed and decide accordingly whether to turn irrigation on or not.

```java
public void MonitorHumidity() {
    int hum = 0;

    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
      String connectionUrl = "jdbc:mysql://localhost:3306/greenhousesql";
      String query = "SELECT * FROM mqtt ORDER BY time DESC LIMIT 1;";
      try {
          Connection conn = DriverManager.getConnection(connectionUrl,"root","admin");
          Statement st = conn.createStatement();
          ResultSet rs = st.executeQuery(query);

          if (rs.next()) {//get first result
              hum = rs.getInt(3);
            }

          System.out.println(hum);

          if(hum < 20) {
              irrigationReq = true;
              String code = actuatorActivation("irrigation-actuator");
              regulateHumidity(code);
          }
          irrigationReq = false;
          set = false;
          conn.close();
      }catch(SQLException e){
          e.printStackTrace();
      }
```

This first method is used to read the latest stored record in the database. If the humidity is lower than a threshold of 20, the irrigation system is activated via the *actuatorActivation()* method.

CoAP

This network is reserved for the actuator part of the architecture. Before using it, it is necessary to register its resources into the CoAP application using the *CoAPResource* class.

```
public static void addResources(String source, String response) {

    String[] resources = response.split(",");

    for(int i = 1; i < resources.length; i++) {
        try{

            String[] parameters = resources[i].split(";");
            String path = parameters[0].split("<")[1].split(">")[0];
            String name = path.split("/")[1];


            Resource newResource = new Resource(name, path, source);

            Interface.registeredResources.put(name,newResource);

            System.out.println("\n" + name + " is registered.");
```

The CoAP resource It is responsible for switching on and off the irrigation system based on a command sent by the Java collector through the *actuatorActivation()* and *actuatorDeactivation()* methods.

```
public String actuatorActivation(String name) {

    /* Resource discovery */

    CoapClient client = new CoapClient(registeredResources.get(name).getCoapURI());

    CoapResponse res = client.post("mode="+ "on", MediaTypeRegistry.TEXT_PLAIN);

    String code = res.getCode().toString();

    registeredResources.get(name).setActuatorState(true);

    return code;

}
```

The first one is called after comparing the value of humidity to the minimum threshold, it sends a plain text command “on” to the CoAP device connected on port 5683. Irrigation is considered active by turning on a GREEN_LED, at the level of the CoAP code.

```
if((len = coap_get_post_variable(request, "mode", &irrigation_status_x))){

        printf("Getting the mode");

        if(strncmp(irrigation_status_x, "on", len)== 0){
                irrigation_status = true;
                LOG_DBG("Beginning Irrigation...\n");
                leds_on(LEDS_NUM_TO_MASK(LEDS_GREEN));
                success = 1;

        } else if(strncmp(irrigation_status_x, "off", len)== 0){
                irrigation_status = false;
                LOG_DBG("Stopping Irrigation...\n");
                leds_off(LEDS_NUM_TO_MASK(LEDS_GREEN));
                success = 1;
        }
```

If irrigation was indeed required and the irrigation was successfully activated, a simple consistency management portion of code is executed, aiming to assign a new “regulated” humidity value. It is also a random number within the acceptable threshold (20 to 40) and it is again stored into the database.

```
public void regulateHumidity(String code) {

    if(!code.startsWith("2")) {
        System.err.println("error: " + code);
        return;
    }

    int min = 20;
    int max = 40;
    int newhum = (int)Math.floor(Math.random()*(max-min+1)+min);
    set = true;

      SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
      Date d = new Date();
      String[] tokens = dateFormat.format(d).split(" ");
      String date = tokens[0];
      String time = tokens[1];


    System.out.println("Irrigation has been activated... \nHumidity after regulation:
    storeMqttData(time, date, newhum, irrigationReq);
    try {
        TimeUnit.SECONDS.sleep(5);
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    actuatorDeactivation("irrigation-actuator");

}
```

To make the mechanism slightly more realistic, the LED light is turned off 5 seconds after activation, using deactivateIrrigation() to show that water has been administered to the plant.

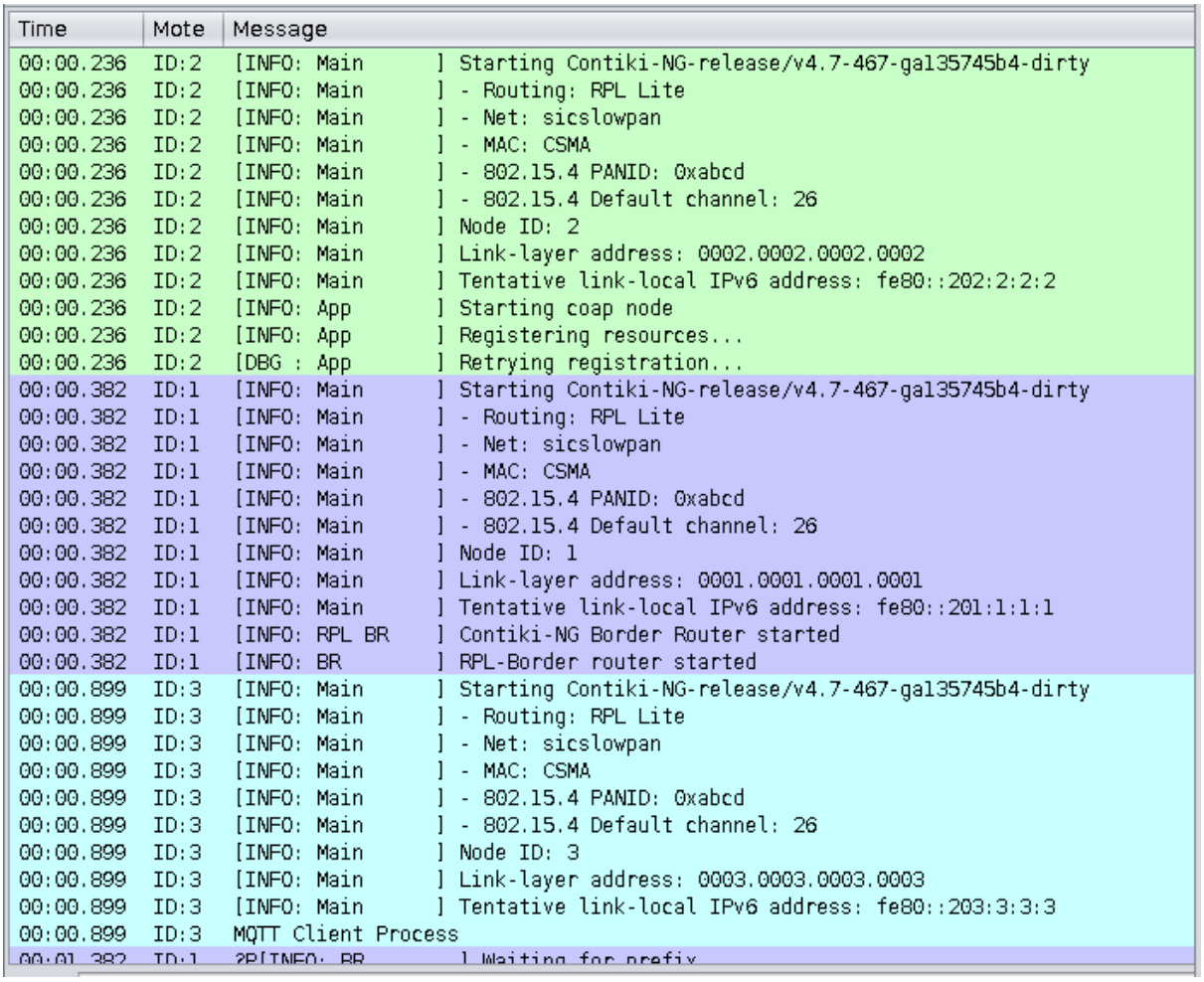
```
public void actuatorDeactivation(String name) {

    /* Resource discovery */

    CoapClient client = new CoapClient(registeredResources.get(name).getCoapURI());

    CoapResponse res = client.post("mode="+ "off", MediaTypeRegistry.TEXT_PLAIN);

    String code = res.getCode().toString();

    registeredResources.get(name).setActuatorState(false);

    if(!code.startsWith("2")) {
        System.err.println("error: " + code);
        return;
    }
}
```

SIMULATION

The simulation was done using Cooja, which allows users to create and load nodes with their respective C code to demonstrate the communication between them.

There are 3 nodes in the GreenHouse simulation, the first one deployed is the border router, connected to a Server port at address 600001. The second is the CoAP node, for which the LED panel is open, and the third one is the MQTT node.

The log shows the initialization of the nodes, then the connection of the router is established at the specified port. The MQTT node begins its connection to the broker (which should be started ulteriorly using the appropriate command), and finally the CoAP resource (Irrigation) is registered to finish the setup.



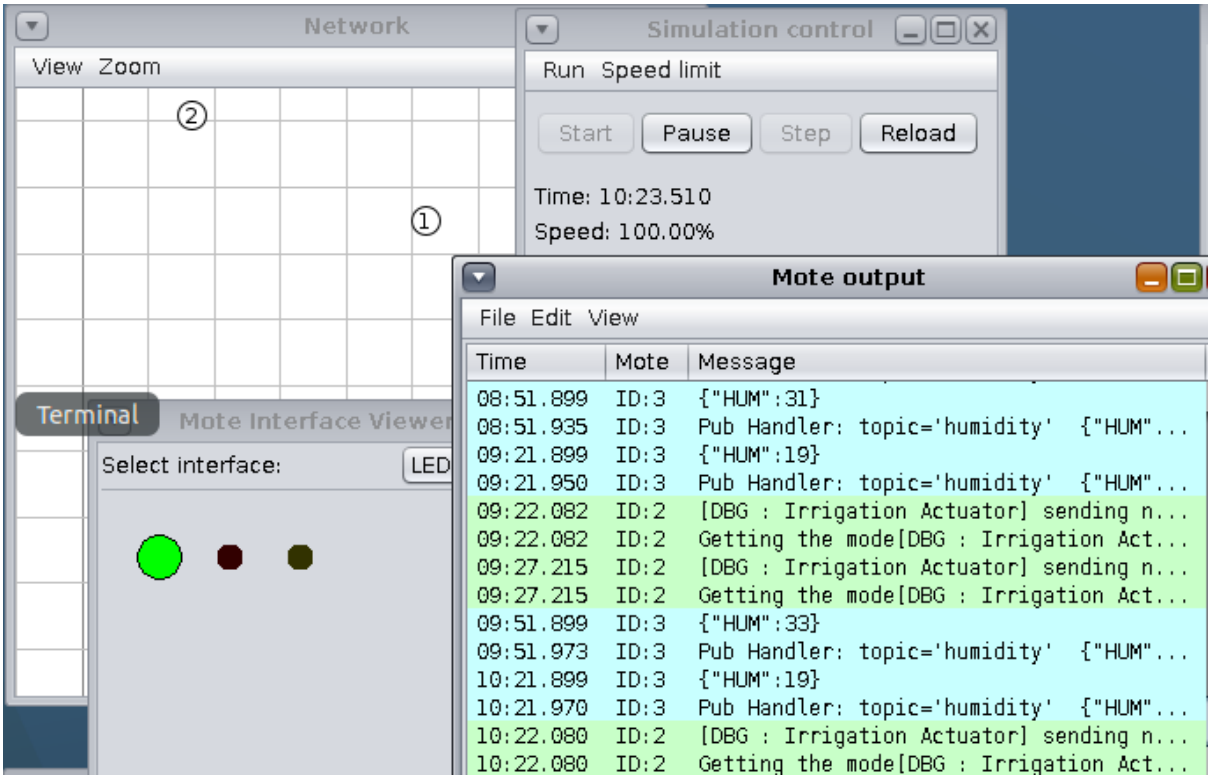
Time	Mote	Message
00:00.236	ID:2	[INFO: Main] Starting Contiki-NG-release/v4.7-467-ga135745b4-dirty
00:00.236	ID:2	[INFO: Main] - Routing: RPL Lite
00:00.236	ID:2	[INFO: Main] - Net: sicslowpan
00:00.236	ID:2	[INFO: Main] - MAC: CSMA
00:00.236	ID:2	[INFO: Main] - 802.15.4 PANID: 0xabcd
00:00.236	ID:2	[INFO: Main] - 802.15.4 Default channel: 26
00:00.236	ID:2	[INFO: Main] Node ID: 2
00:00.236	ID:2	[INFO: Main] Link-layer address: 0002.0002.0002.0002
00:00.236	ID:2	[INFO: Main] Tentative link-local IPv6 address: fe80::202:2:2:2
00:00.236	ID:2	[INFO: App] Starting coap node
00:00.236	ID:2	[INFO: App] Registering resources...
00:00.236	ID:2	[DBG : App] Retrying registration...
00:00.382	ID:1	[INFO: Main] Starting Contiki-NG-release/v4.7-467-ga135745b4-dirty
00:00.382	ID:1	[INFO: Main] - Routing: RPL Lite
00:00.382	ID:1	[INFO: Main] - Net: sicslowpan
00:00.382	ID:1	[INFO: Main] - MAC: CSMA
00:00.382	ID:1	[INFO: Main] - 802.15.4 PANID: 0xabcd
00:00.382	ID:1	[INFO: Main] - 802.15.4 Default channel: 26
00:00.382	ID:1	[INFO: Main] Node ID: 1
00:00.382	ID:1	[INFO: Main] Link-layer address: 0001.0001.0001.0001
00:00.382	ID:1	[INFO: Main] Tentative link-local IPv6 address: fe80::201:1:1:1
00:00.382	ID:1	[INFO: RPL BR] Contiki-NG Border Router started
00:00.382	ID:1	[INFO: BR] RPL-Border router started
00:00.899	ID:3	[INFO: Main] Starting Contiki-NG-release/v4.7-467-ga135745b4-dirty
00:00.899	ID:3	[INFO: Main] - Routing: RPL Lite
00:00.899	ID:3	[INFO: Main] - Net: sicslowpan
00:00.899	ID:3	[INFO: Main] - MAC: CSMA
00:00.899	ID:3	[INFO: Main] - 802.15.4 PANID: 0xabcd
00:00.899	ID:3	[INFO: Main] - 802.15.4 Default channel: 26
00:00.899	ID:3	[INFO: Main] Node ID: 3
00:00.899	ID:3	[INFO: Main] Link-layer address: 0003.0003.0003.0003
00:00.899	ID:3	[INFO: Main] Tentative link-local IPv6 address: fe80::203:3:3:3
00:00.899	ID:3	MQTT Client Process
00:01.382	ID:1	2P[INFO: BR] Waiting for prefix

The MQTT node then makes a connection through the started MQTT broker, and the CoAP node begins registering its resources.

```
00:04.382  ID:1   [INFO: BR        ] Server IPv6 addresses:
00:04.382  ID:1   [INFO: BR        ]   fd00::201:1:1:1
00:04.382  ID:1   [INFO: BR        ]   fe80::201:1:1:1
00:14.899  ID:3   Connecting!
00:21.592  ID:3   Application has a MQTT connection
00:21.899  ID:3   Subscribing!
00:21.924  ID:3   Application is subscribed to topic successfully
00:26.592  ID:2   |[DBG : App       ] Registered!
```

After running the executable Java Jar file of the Application, the automatic humidity monitoring system begins receiving messages from the MQTT node, and subsequently running the necessary methods.

```
xxxxxxxxxxxxxxxxxxx GREEN HOUSE AUTO SYSTEM xxxxxxxxxxxxxxxxxxx

This application serves to monitor the humidity and temperature
of soil in a green house and regulate those values accordingly.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


irrigation-actuator is registered.
Humidity observed is": 11
Irrigation is required.

Irrigation has been activated...
Humidity after regulation: 20

Humidity observed is": 5
Irrigation is required.

Irrigation has been activated...
Humidity after regulation: 38
```



MySQL

For the database part of the application, one single table is used to store all the information, consisting of four columns

Time: the time of observation.
Date: the date of observation.
Humidity: the value observed.
Irrigation_status: includes 3 possible values:

- Non-Required: The humidity value is above the threshold.
- Required: The humidity value is below the threshold.
- Regulated: This should always appear in the record after “Required” as it shows that the irrigation has been activated and a new humidity value was assigned.

time	date	humidity	irrigation
06:27:59	05-09-2022	14	Required
06:28:01	05-09-2022	24	Regulated
06:28:29	05-09-2022	26	Non-Required
06:28:59	05-09-2022	11	Required
06:28:59	05-09-2022	24	Regulated
06:29:29	05-09-2022	2	Required
06:29:30	05-09-2022	26	Regulated
06:29:59	05-09-2022	12	Required
06:30:00	05-09-2022	24	Regulated
06:30:29	05-09-2022	6	Required
06:30:30	05-09-2022	30	Regulated
06:30:59	05-09-2022	35	Non-Required
06:31:30	05-09-2022	25	Non-Required
06:32:00	05-09-2022	16	Required
06:32:00	05-09-2022	32	Regulated
06:32:30	05-09-2022	30	Non-Required
06:33:00	05-09-2022	9	Required
06:33:00	05-09-2022	36	Regulated
06:33:30	05-09-2022	8	Required
06:33:30	05-09-2022	23	Regulated
06:34:00	05-09-2022	38	Non-Required

DATA ENCODING

For the MQTT publisher, a **JSON** format was chosen because of its many qualities with respect to memory, time and efficiency. It is less wordy and more compact than XML, therefore it takes less space and consequently less time to be processed and transmitted. It is readable as it is allowed to put a label on the value observed, this can be particularly useful when data collected from multiple sensors is hubbed in a transit node, as done in a hierarchical architecture for example. The data will be properly labeled and no chance for confusion will occur.

From the Java application to the CoAP application, a **Plain Text** format is used as the commands are simple“on/off”. The message is never processed or used elsewhere than by the node itself, it is concise and self explanatory, so it doesn’t need more structure or space than that.

CONCLUSION

Developing **GreenHouse** was a good way to get used to the IoT technologies, it takes the learning material beyond theory and sheds light on the different practical aspects and implementation issues one may run into. This project is also a good basis for a grander application, gardening and agriculture are without a doubt domains of precision and diligence, and process automation is not only prefered but needed. GreenHouse can surely be scaled into a fully fledged smart system.

References

Project implementation https://github.com/aidahimm/GreenHouse