Abstract

To simplify the necessary steps required to get into urban farming using AI LLM models such as ChatGPT for AIOT, and in doing so, creating a larger community of urban farmers for sustainability.

30.201 Wireless Communications and Internet-of-Things

Espress(if) Farming

Sean Ang 1005019

**Introduction**

In the current century, where global warming is taking the world by storm (quite literally), it is now, more than ever, where sustainability should be taken more seriously by the average person. In Singapore alone, the greenhouse gas emissions in 2021 totaled to 50,089.9 gigagrams of CO2[[1]](#footnote-2). With a lack of land and natural forests, Singapore does not have a suitable outlet for the reduction of these greenhouse gas emissions. Thus, we see a move to vertical farming in space-constricted areas like Singapore, which uses many different smart systems and even Internet-of-Things (IOT) technologies to care for these vertical farms. There is one key area in which Singapore has yet to utilize, however, and I believe that solution lies with the average Joe. Households are yet an untapped source of urban farming, and I believe one of the key reasons is the accessibility to information. How does one get started? What are the easiest plants to take care of? How do I germinate a plant? These are all, I believe, reasonable questions that are enough to turn people off from gardening altogether. Thus, I aim to provide a solution to this problem, by incorporating the current use of smart, automated farming, coupled with the use of Artificial Intelligence (AI) Large Language Models (LLM) such as chat GPT to alleviate this barrier to entry.

**Methodology**

To create said solution, let us first break this down into multiple parts:

* AI assistant (Think ChatGPT, Copilot)
* Sensor reading
* Data processing
* Notification / Action

**AI assistant:**

We will be incorporating an AI assistant chatbot into our front-end to streamline the entire process of learning about the processes of farming.

**Sensor reading:**

A few sensors would ideally be used for the monitoring of the soil and plant, but for the scope of this project, we will focus on the monitoring of the moisture level of the soil. Additionally, we will be using a second ESP to track the health of the plants using CV and object detection.

**Data processing:**

From the sensor readings, we will determine whether the moisture level is within the acceptable range and will automatically trigger our action to water the plant while sending this information to our backend.

**Notification / Action:**

From the processed sensor data, there would be two outcomes:

1. Healthy
2. Watered

These two pieces of information would be sent to the database through:

1. Establishment with AWS IOT core from ESP32
2. Sending the information in JSON via MQTT to AWS IOT core
3. Routing the message from AWS IOT core to AWS DynamoDB

A screenshot of a computer program

Description automatically generatedCode in ESP IDF, gets reading at fixed intervals and sends the information via MQTT to AWS IOT.A screenshot of a computer

Description automatically generatedAWS IOT core receives this data in JSON.

A screenshot of a computer

Description automatically generatedAWS IOT core rule created to route the message to DynamoDB created.A screenshot of a computer

Description automatically generatedAWS DynamoDB receives and stores the information.

**Hardware / Software**

A diagram of a circuit board

Description automatically generatedSchematic for hygrometer sensor and water pump

The Hygrometer sensor reads the soil moisture using ADC\_ONESHOT\_READ, and when it is past the threshold, the GPIO output is set to high to activate the L298N motor driver, running the pump for a fixed period.

For the MQTT and connection to AWS IOT core, we use the example provided by the AWS IOT SDK, tls\_mutual\_auth, adapting the code to include our application of sensors. Since the full code is thousands of lines, a snapshot is provided:

static int publishToTopic( MQTTContext\_t \* pMqttContext )

{

    int returnStatus = EXIT\_SUCCESS;

    MQTTStatus\_t mqttStatus = MQTTSuccess;

    uint8\_t publishIndex = MAX\_OUTGOING\_PUBLISHES;

    int reading = get\_sensor\_reading();

    assert( pMqttContext != NULL );

    /\* Get the next free index for the outgoing publish. All QoS1 outgoing

     \* publishes are stored until a PUBACK is received. These messages are

     \* stored for supporting a resend if a network connection is broken before

     \* receiving a PUBACK. \*/

    returnStatus = getNextFreeIndexForOutgoingPublishes( &publishIndex );

    if( returnStatus == EXIT\_FAILURE )

    {

        LogError( ( "Unable to find a free spot for outgoing PUBLISH message.\n\n" ) );

    }

    else

    {

        cJSON \*json = cJSON\_CreateObject();

        cJSON\_AddNumberToObject(json, "humidity", reading);

        char \*str = cJSON\_Print(json); // or cJSON\_PrintUnformatted() or cJSON\_PrintBuffered() ...

        size\_t len = strlen(str);

        /\* This example publishes to only one topic and uses QOS1. \*/

        outgoingPublishPackets[ publishIndex ].pubInfo.qos = MQTTQoS1;

        outgoingPublishPackets[ publishIndex ].pubInfo.pTopicName = MQTT\_EXAMPLE\_TOPIC;

        outgoingPublishPackets[ publishIndex ].pubInfo.topicNameLength = MQTT\_EXAMPLE\_TOPIC\_LENGTH;

        outgoingPublishPackets[ publishIndex ].pubInfo.pPayload = str;

        outgoingPublishPackets[ publishIndex ].pubInfo.payloadLength = len;

Function publishToTopic() is edited to include getting the sensor readings.

1. [(https://www.nea.gov.sg/our-services/climate-change-energy-efficiency/climate-change/greenhouse-gas-inventory)](https://www.nea.gov.sg/our-services/climate-change-energy-efficiency/climate-change/greenhouse-gas-inventory) [↑](#footnote-ref-2)