VIETNAMESE - GERMAN UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Multi-Disciplinary Project Report

Group: Coc Coc

Student 1: Nguyễn Thiên Nguyên - 10422059
Student 2: Phạm Đức Mạnh - 10422047
Student 3: Nguyễn Thanh Tú - 10422080
Student 4: Đỗ Nam Hải - 10422094
Student 5: Võ Tấn Sang - 10422114
Student 6: Nguyễn Hoàng Yến Ngọc - 10422056

HỒ CHÍ MINH CITY, 07/ 2023

Content

1 Introduction **2**
1.1 Project Summary . . . 2
1.2 Project Features . . . 2
1.3 Project Impact . . . 2
1.4 Core Components . . . 2

2 Planning **4**
2.1 System Analytic . . . 4
2.2 System Design . . . 5

3 System Implementation and Development **7**
3.1 Module 1: Sensors Data Collection . . . 7
3.1.1 Why do we use MQTT? . . . 7
3.1.2 Adafruit IO Feeds . . . 7
3.1.3 Adafruit Dashboard . . . 9
3.1.4 Arduino Source Code . . . 10
3.2 Module 2: AI Processor . . . 15
3.2.1 Why do we select Teachable Machine for training AI detection? . . . 15
3.2.2 Dataset Collection from Teachable Machine . . . 15
3.2.3 Training Model . . . 18
3.2.4 Python Source Code . . . 19

4 System Deployment **26**

5 Conclusion **29**

1 Introduction

1.1 Project Summary

The Smart Tomato Care system is an innovative solution that uses IoT and AI to automate the care of tomato plants. The system uses sensors to collect data on the plant's environment, such as temperature, humidity, light intensity and soil moisture. The data is then analyzed by AI to determine the optimal conditions for the plant's growth. The system can then take actions to maintain these conditions, such as adjusting the watering schedule or controlling the lighting.

1.2 Project Features

- Feature 1: Automatically adjust the watering schedule based on the plant's needs.
- Feature 2: Control the lighting to provide the plant with the optimal amount of light.
- Feature 3: Control the temperature based on the plant's needs.
- Feature 4: Track the growing process via continuously monitoring plants in order to detect plant diseases immediately.
- Feature 5: Send alerts to the user if there are any problems with the plant.

1.3 Project Impact

- The Automatic Tomato Plant Care system can help to improve the yield and quality of tomato plants.
- The system can also help to reduce the amount of time and effort required to care for tomato plants.
- The system can make tomato cultivation more accessible to people who do not have a lot of time or experience.

1.4 Core Components

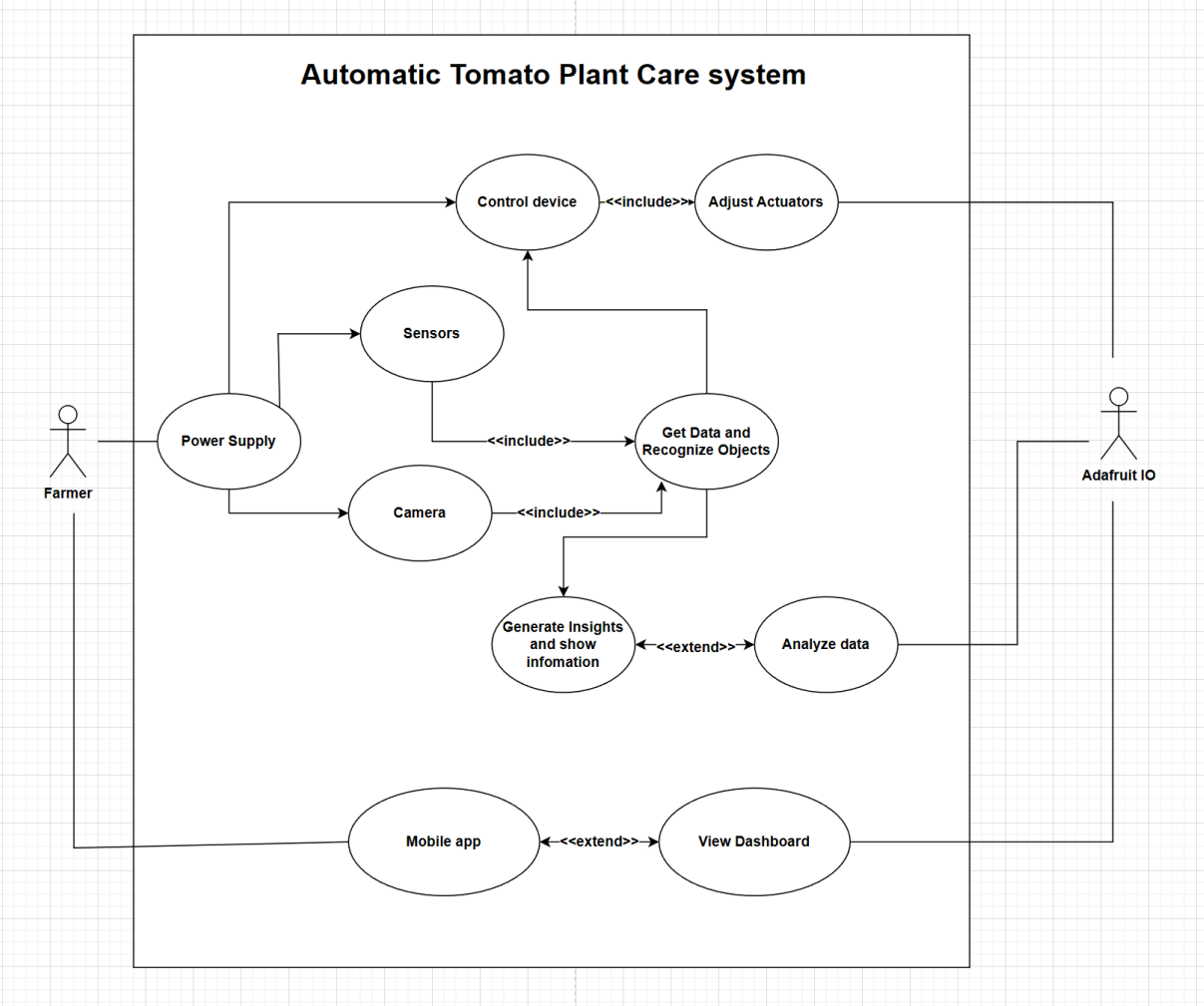
- **IoT Sensors and Actuators:** The heart of the system lies in its network of IoT sensors and actuators. Sensors such as temperature and humidity sensors gather environmental data, while actuators, including pumps, sunshade nets, nebulizers and LEDs, allow the system to take proactive actions, such as adjusting climate control.
- **AI-powered Camera and Teachable Machine:** The system features an AI-powered camera that employs Google Teachable Machine technology to recognize and process images. This enables the system to perform real-time visual analysis, detecting pests and diseases and taking action to prevent them from damaging the plant.

- **Data Analytics:** Harnessing the potential of AI, the system employs advanced data analytics techniques to provide valuable insights into the plant's environment and growth patterns. This data is collected from a variety of sources, including IoT sensors, weather data, and historical records. The system then uses this data to identify trends, patterns, and anomalies. This information can then be used to make informed decisions about the plant's care.
- **Adafruit IO for Remote Monitoring and Control:** The Automatic Tomato Plant Care system leverages Adafruit IO, a robust IoT platform, to enable remote monitoring and control capabilities. This empowers users to access real-time data, receive alerts, and manage the system's settings from anywhere in the world through web or mobile applications.
- **Web and Mobile Dashboard:** A user-friendly web-based dashboard and a mobile app provide users with a comprehensive interface to monitor the system's performance, access data visualizations, and control various devices and functionalities with ease and convenience.

In summary, the Automatic Tomato Plant Care system is an innovative solution that uses IoT and AI to automate the care of tomato plants. The system can help to improve the yield and quality of tomato plants, reduce the amount of time and effort required to care for tomato plants, and make tomato cultivation more accessible to people who do not have a lot of time or experience.

2 Planning

2.1 System Analytic



Hình 1: Use case diagram for automatic system

Bảng 1: Use case description

Use-case name	Role	Description
User_Dashboard	User	For User Manual control
Admin_Dashboard	Admin	Create Dashboard for Data visualization
Sensors	Component	Measure Environmental Data
Camera	Component	Capture High Pixel Picture for analyst
Power Supply	Component	Energy supply for the system
Control device	System	Monitor Components
Adjust Actuator	System	Control the dependent components

2.2 System Design

The system design for the Automatic Tomato Plant Care system defines the different components of the system, how they work together, and the technologies used to implement them. This high-level overview provides a general understanding of how the system works. Specifically, the system design includes the following:

- **Architectural components:** The different parts of the system, such as the sensors, actuators, controller, and user interface.
- **Data flow:** The way that data is collected, processed, and acted upon by the system.
- **Technologies:** The specific technologies used to implement the system, such as IoT sensors, actuators.

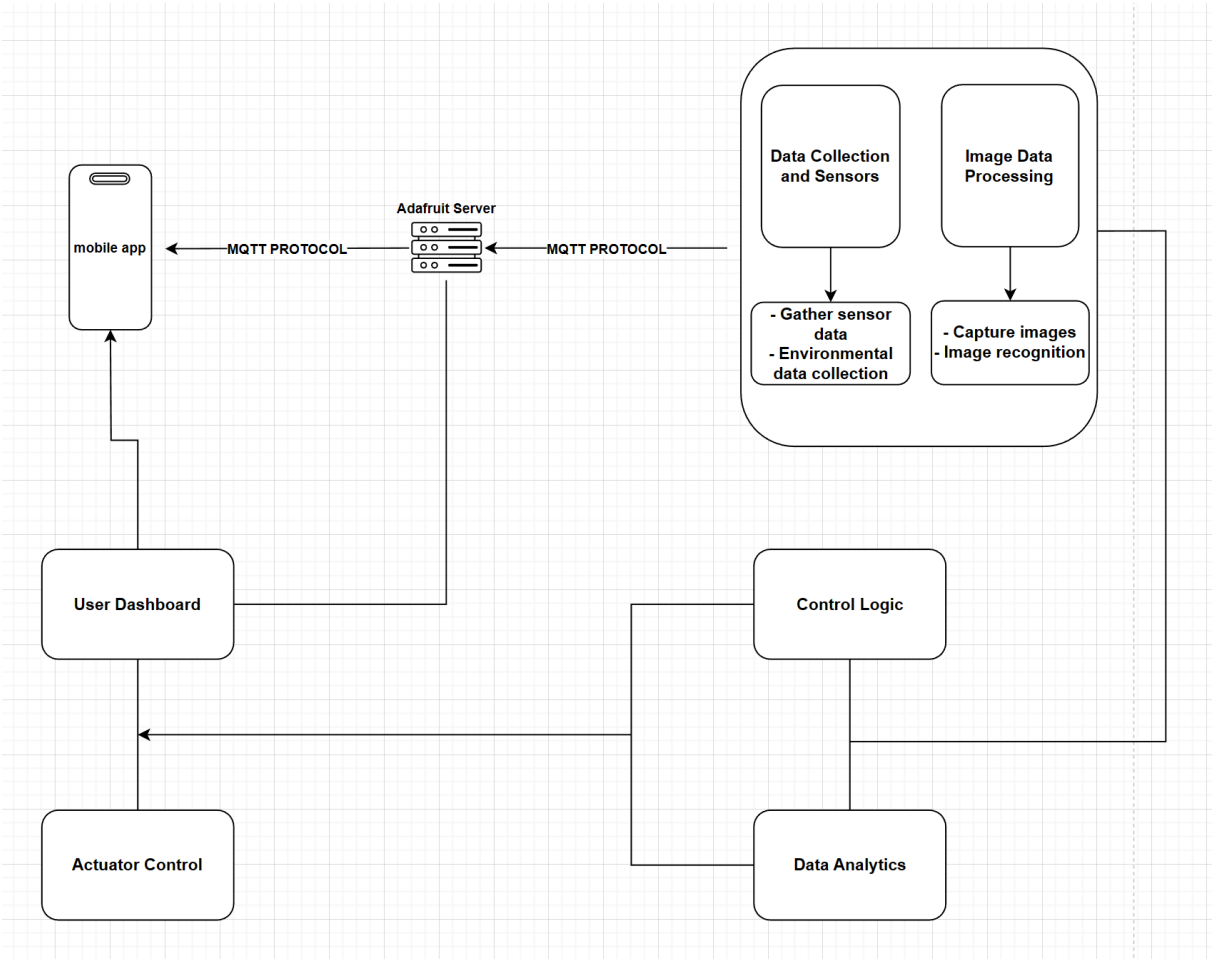
The system design is an important part of the development process for the Automatic Tomato Plant Care system. It ensures that the system is well-structured and that the different components work together seamlessly. The system design also helps to identify any potential challenges or risks that may need to be addressed during development.

1. **Architecture components:**
    - Presentation Layer: This layer provides the user interface for the system, including a web-based dashboard and a mobile app.
    - Perception Layer: This layer collects data from sensors and actuators, and then passes the data to the next layer for processing.
    - Data Layer: This layer stores and retrieves data from various sources, such as sensor data, user preferences, and historical data for analysis.
    - Integration Layer: This layer facilitates communication with external services, such as Google Teachable Machine and Adafruit IO.
2. **Cloud Infrastructure (Adafruit IO)**: The system is hosted on cloud-based infrastructure, providing scalability and accessibility from anywhere. This layer enables remote monitoring of connected devices by communicating with the Adafruit IO platform.
3. **Actuator Control:** This layer receives commands from the data analytics module to control actuators (e.g., pumps or nebulizers).
4. **User Dashboard:** This layer provides a continuous report on the status of the tomato and notify if there is an issue happen
5. **Data Analytics and Implementations:** This layer processes the sensor data to generate insights and alerts, and then uses AI algorithms to make decisions based on the data analysis, such as adjusting settings for climate control and lighting or triggering alerts for unusual events.

6. **Sensors and Data Collection:** This layer collects environmental data from sensors and then aggregates the data from different IoT devices for processing by the data analytics module.

7. **AI Image Processing:** This layer uses AI algorithms to process images captured by the AI-powered camera, such as object recognition.

It is essential to consider these design aspects while implementing the Automatic Tomato Plant Care system. The actual system design will depend on the specific technologies, platforms, and hardware components used in the project. Additionally, it is crucial to conduct thorough testing and iterations to refine the system's performance and user experience.



Hình 2: System Design sample

Block Name	Input	Output
Data Collection and Sensors	Images captured from camera	AI enumerable results
Image Data Processing	Images captured from camera	AI enumerable results
Control Logic	Sensor data	Actuator control
Data Analytics	Aggregated data and AI results	Actuator control
Actuator Control	Aggregated data and AI results	Actuator control
User Dashboard	Aggregated data and actuated order	User interface display

Bảng 2: System Design Description

3 System Implementation and Development

3.1 Module 1: Sensors Data Collection

3.1.1 Why do we use MQTT?

We use MQTT in our tomato program due to its lightweight nature, making it ideal for IoT applications with low bandwidth and resource constraints. The publish-subscribe model allows real-time communication between devices, ensuring timely updates on temperature, humidity, and other variables for proactive plant care. MQTT's broker-based architecture facilitates seamless scaling to accommodate additional devices without compromising performance. It also offers multiple Quality of Service levels for reliable message delivery, critical for data integrity in agriculture. The support for persistent sessions ensures continuous connectivity even during network disruptions. With customized topic-based messaging, each device receives only relevant information, optimizing data delivery.

3.1.2 Adafruit IO Feeds

We employ what are known as 'feeds' to store information on Adafruit's server, ensuring that each class within the system has its own designated data channel. For the purpose of our project, we've created 8 separate feeds to accommodate different types of data. As this is a collaborative effort, it's important to set the feeds' privacy to the public, allowing everyone to access and configure them as needed.

Multi_Prj

Name	Key	Last value	Recorded
ai_camera	multi-prj.ai-camera	/9j/4AAQSkZJRg...	5 minutes ago
humidity	multi-prj.humidity	77	4 minutes ago
light_density	multi-prj.light-den...	487	4 minutes ago
light_switch	multi-prj.light-swit...	1	4 minutes ago
nebulizer_switch	multi-prj.nebulize...	1	4 minutes ago
notifications	multi-prj.notificati...	Brightness is too ...	4 minutes ago
pump_switch	multi-prj.pump-sw...	1	4 minutes ago
soil_moisture	multi-prj.soil-mois...	18	4 minutes ago
sunshade_net_switch	multi-prj.sunshad...	1	4 minutes ago
temperature	multi-prj.temperat...	35	4 minutes ago

Hình 3: Adafruit IO Feeds

- **Temperature:** The program monitors the temperature in the plant's environment. Tomatoes thrive within specific temperature ranges, and the system uses this data to maintain an ideal climate. Extreme temperatures can stress or even harm the plants, so keeping track of temperature helps prevent adverse effects.
- **Humidity:** The program measures the humidity levels around the tomato plants. Tomatoes prefer a certain level of humidity for proper transpiration and nutrient absorption. Maintaining appropriate humidity levels can help prevent issues like wilting or fungal diseases.
- **Light Density:** The program assesses the intensity of light exposure the tomato plants receive. Tomatoes require sufficient sunlight to carry out photosynthesis and produce healthy fruits. Monitoring light density helps ensure that the plants receive the right amount of light, preventing problems related to overexposure or inadequate lighting.
- **Soil Moisture:** The program monitors the moisture content in the soil. Proper watering is crucial for tomato plants' growth and fruit development. By tracking soil moisture levels, the system can automate or provide recommendations for watering, avoiding under- or overwatering, both of which can harm the plants.
- **Nebulizer switch:** When the program detects that the air humidity falls below the desired threshold, it activates the nebulizer. The nebulizer adds fine water droplets to the air, effectively increasing humidity levels around the plants. Maintaining proper humidity helps prevent issues such as dryness or stress, promoting healthier plant growth.
- **Pump switch:** If the program detects that the soil moisture is insufficient for the tomato plants, it activates the pump. The pump provides a controlled amount of water to the soil,

ensuring that the plants receive adequate hydration. This feature prevents under-watering and helps sustain proper soil moisture levels for the plants to thrive.

- **Sunshade net switch:** When the program detects an excessive amount of light falling on the tomato plants, it deploys the sunshade net. The net acts as a shield, partially reducing the incoming light intensity. This protects the plants from potential sunburn or heat stress, maintaining an optimal light environment for their growth.

- **Light switch:** If the program identifies a lack of sufficient natural light, it activates the light switch. By providing supplementary artificial light, the program ensures that the tomato plants receive the necessary amount of light for photosynthesis even during periods of low natural light. This feature helps prevent stunted growth and ensures continuous plant development.

- **Notifications:** The notification system alerts users or gardeners when critical measurements are reached, such as high/low temperature, unfavorable humidity, excessive light intensity, or low soil moisture. Immediate action can be taken to protect tomato plants by adjusting the environment or watering schedule, ensuring their well-being and safeguarding them from extreme conditions.

- **AI Camera:** The AI camera in our tomato care support program provides real-time visual analysis of the tomato plants. It uses artificial intelligence to detect and identify any signs of pest infestations, diseases, or growth anomalies. This enables early detection and targeted intervention, ensuring proactive care and healthier tomato plants.

3.1.3 Adafruit Dashboard

The dashboard serves as a visual representation of vital system data, presenting it in diverse formats like gauges, line graphs, buttons, and text. Gauges and graphs are linked to sensor-related classes, while buttons turn on the support features for watering and light. Additionally, there's an AI camera for checking the condition of the tomato plant.

Line graphs and gauges are used to depict soil moisture, light density, humidity, and temperature measurements for several reasons. Line graphs offer a visual representation of trends and changes over time, allowing users to observe patterns in the data. Gauges provide an instant, easy-to-understand view of current values, enabling quick assessments of the variables' status. These visualizations aid users in making informed decisions, adjusting settings, and maintaining optimal conditions for healthy tomato plant growth.

Using buttons for the sunshade net, nebulizer, light, and pump in our tomato care program simplifies user interaction. Buttons offer a straightforward and intuitive way for users to control these essential features with just a click. This ensures ease of operation, quick access to specific actions, and a user-friendly experience for managing the tomato plants' environment effectively. Beside, the binary value of those features are only on or off, that's why the buttons are the most suitable.

Lastly, the AI camera aids optimal tomato plant care by providing real-time visual analysis. It detects pests, diseases, and growth irregularities. Early detection allows targeted interventions, preventing potential damage. With continuous monitoring and timely alerts, the AI camera ensures proactive measures, leading to healthier tomato plants and higher crop yields.

Hình 4: Sample Use Case

3.1.4 Arduino Source Code

Main Functions (APIs):

```
    digitalWrite()
#This function turns on or off an LED connected to a digital pin.
#It can also be used to control other digital devices, such as motors or relays.
#The digitalWrite() function is a versatile tool that can be used to control a wide variety of devices.
```

```
    Serial.read()
#This function reads a single character from the serial port.
#It can be used to receive data from a computer or another Arduino.
```

```
4 #The Serial.read() function is a simple way to communicate with other ↩
    devices over a serial connection.
```

Arduino Source Code

This is the Adafruit source code of our project and it is an implementation of a script that interacts with Adafruit IO, an Internet of Things (IoT) platform via an intermediate Python file.

Firstly, these provided lines of code import necessary modules and set up the initial environment for a program.

Listing 1: Libraries for the sensors

```
1 #include <DHT.h>
```

We import the DHT library, which is a powerful and easy-to-use library for reading sensor data from the DHT11 sensor. The DHT11 sensor is a low-cost, easy-to-use sensor that can measure temperature and humidity. This library allows us to easily and accurately read sensor data from the DHT11 sensor, which is essential for our environmental monitoring system.

Listing 2: Constants for the sensors

```
1 #define DHT_PIN 2
2 #define SOIL_MOISTURE_PIN A0
3 #define LDR_PIN A1
4 #define DHT_TYPE DHT11
```

The sensor pins are the physical pins on the Arduino board that are connected to the DHT sensor. The DHT sensor type is the specific type of DHT sensor that is being used. These constants make our code more readable and maintainable, and they also help to prevent errors.

Listing 3: Thresholds for the sensors

```
1 #define TEMP_LOWER_LIMIT 10
2 #define TEMP_UPPER_LIMIT 20
3 #define HUMIDITY_LOWER_LIMIT 40
4 #define HUMIDITY_UPPER_LIMIT 60
5 #define SOIL_MOISTURE_LOWER_LIMIT 55
6 #define SOIL_MOISTURE_UPPER_LIMIT 70
7 #define BRIGHTNESS_LOWER_LIMIT 800
8 #define BRIGHTNESS_UPPER_LIMIT 1300
```

Then, these thresholds are used to determine whether to turn on or off the actuators. For example, if the temperature exceeds the upper threshold, the sunshade net will be turned on.

These thresholds ensure that our environmental monitoring system is operating effectively and that our plants are being kept in a healthy environment.

Listing 4: Pin Numbers for the Actuators

```
#define NEBULIZER_PIN 3
#define LIGHT_PIN 4
#define SUNSHADE_NET_PIN 5
#define PUMP_PIN 6
```

The actuators are the devices that are used to control the environmental condition. For example, the sunshade net is an actuator that can be used to shade the plant from the sun. These pin numbers make it easy to control the actuators, and they also help to prevent errors.

Listing 5: Function prototypes

```
void checkTemperature(float temperature);
void checkHumidity(float humidity);
void checkSoilMoisture(int soilMoisture);
void checkBrightness(int brightness);
void notifyWarning(String message);
```

Here defines function prototypes for the functions that are used to check the sensor data then implements actuators' switches. These functions are responsible for the environmental analysis and implementation tasks. These function prototypes make our code more readable and maintainable, and they also help to prevent errors.

Listing 6: Initialize the DHT sensor

```
DHT dht(DHT_PIN, DHT_TYPE);

void setup() {
  Serial.begin(9600);

  dht.begin();

  pinMode(NEBULIZER_PIN, OUTPUT);
  pinMode(LIGHT_PIN, OUTPUT);
  pinMode(SUNSHADE_NET_PIN, OUTPUT);
  pinMode(PUMP_PIN, OUTPUT);
  pinMode(A1,INPUT);

  digitalWrite(NEBULIZER_PIN, LOW);
  digitalWrite(LIGHT_PIN, LOW);
  digitalWrite(SUNSHADE_NET_PIN, LOW);
```

```
17   digitalWrite(PUMP_PIN, LOW);
18 }
```

We initialize the DHT sensor as essential for the proper operation of the environmental monitoring system. This ensures that the sensor is ready to start reading data and that the environmental monitoring system can operate effectively.

Listing 7: Environmental Analysis and Implemantal Functions

```
1  void loop() {
2
3    if(Serial.available()){
4      char c = Serial.read();
5      if (c == 't'){
6        // Check temperature
7        float t = dht.readTemperature();
8        Serial.print("!1:Temperature:");
9        Serial.print(t);
10       Serial.print("#");
11       if (t > TEMP_UPPER_LIMIT) {
12       digitalWrite(SUNSHADE_NET_PIN, HIGH); // Turn on sunshade net
13       } else if (t < TEMP_LOWER_LIMIT) {
14       digitalWrite(SUNSHADE_NET_PIN, LOW);  // Turn off sunshade net
15       } else {
16       digitalWrite(SUNSHADE_NET_PIN, LOW); // Turn off sunshade net
17       }
18     }
19     else if (c == 'h'){
20       // Check humidity
21       float h = dht.readHumidity();
22       Serial.print("!1:Humidity:");
23       Serial.print(h);
24       Serial.print("#");
25       if (h < HUMIDITY_LOWER_LIMIT) {
26       digitalWrite(NEBULIZER_PIN, HIGH); // Turn on nebulizer
27       } else if (h > HUMIDITY_UPPER_LIMIT) {
28       digitalWrite(NEBULIZER_PIN, LOW); // Turn off nebulizer
29       } else {
30       digitalWrite(NEBULIZER_PIN, LOW); // Turn off nebulizer
31       }
32     }
33     else if (c == 's'){
34       // Check humidity
35       float s = (1200 - analogRead(SOIL_MOISTURE_PIN))/10;
36       Serial.print("!1:Soil moisture:");
```

```
37        Serial.print(s);
38        Serial.print("#");
39        if (s < SOIL_MOISTURE_LOWER_LIMIT) {
40        digitalWrite(PUMP_PIN, HIGH); // Turn on nebulizer
41        } else if (s > SOIL_MOISTURE_UPPER_LIMIT) {
42        digitalWrite(PUMP_PIN, LOW); // Turn off nebulizer
43        } else {
44        digitalWrite(PUMP_PIN, LOW); // Turn off nebulizer
45    }
46      }
47      else if (c == 'b'){
48        // Check brightness
49        float b = (1500-analogRead(LDR_PIN));
50        Serial.print("!1:Brightness:");
51        Serial.print(b);
52        Serial.print("#");
53        if (b < BRIGHTNESS_LOWER_LIMIT) {
54        digitalWrite(LIGHT_PIN, HIGH); // Turn on light
55        } else if (b > BRIGHTNESS_UPPER_LIMIT) {
56        digitalWrite(LIGHT_PIN, LOW); // Turn off light
57        } else {
58        digitalWrite(LIGHT_PIN, LOW); // Turn off light
59      }
60      }
61    }
62  }
```

The environmental analysis and implementation tasks functions ensure that the plants are kept in a healthy environment. These functions check the sensor data and turn on or off the actuators accordingly, which helps to prevent damage from extreme temperatures or humidity levels. Additionally, the notifyWarning() function sends a warning message if a threshold is exceeded, which can help users to identify and address potential problems before they cause damage to the plants.

Overall, the code provides a number of benefits for clients, including:

- Ease of use: The code is easy to understand and use, even for users with limited coding experience. This is due to the use of clear and concise language, as well as well-documented functions and variables.
- Automatic checking: The code automatically checks the sensor data and turns on or off the actuators accordingly. This means that users do not have to manually check the sensor data, which can save time and effort.
- Notification warnings: The code sends a warning message if a threshold is exceeded. This can help users to identify and address potential problems before they cause damage to the

plants.

- Prevention of damage to plants: The code helps to prevent damage to plants by keeping them in a healthy environment. This is achieved by checking the sensor data and turning on or off the actuators accordingly.
- In addition to these benefits, the code is also modular and scalable. This means that it can be easily modified and expanded to meet the specific needs of different users. For example, users could add additional sensors or actuators, or they could change the thresholds for the sensors.

3.2 Module 2: AI Processor

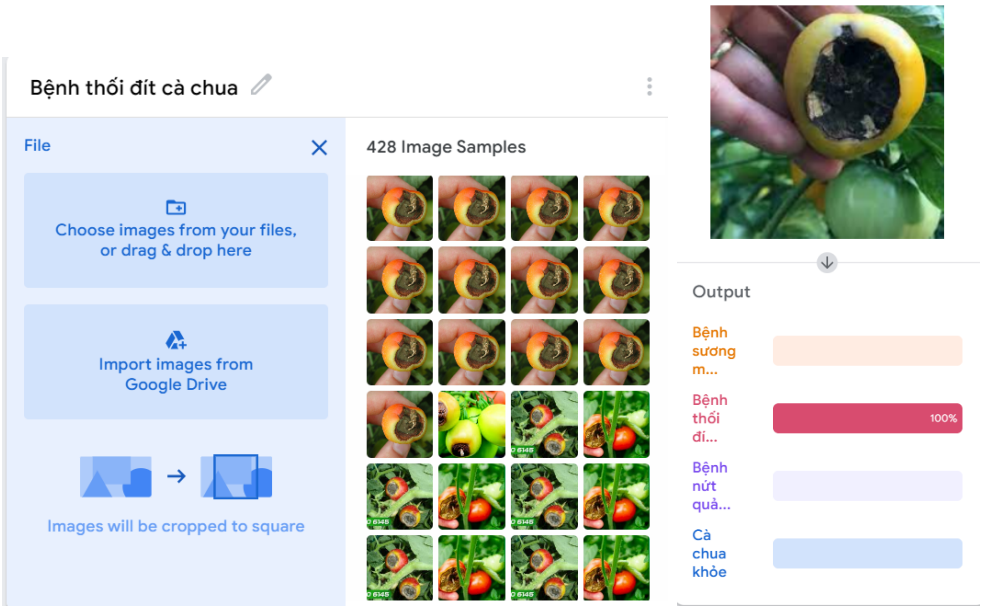
3.2.1 Why do we select Teachable Machine for training AI detection?

- Easy to use: As mentioned above, Teachable Machine is a very easy-to-use tool. Even if you don't have any experience with machine learning, you should be able to get started with Teachable Machine quickly.
- Quick training: Teachable Machine can train a model in a matter of minutes, even on a basic computer. This makes it a good option for projects where you need to get a model up and running quickly.
- Flexible: Teachable Machine can be used to train models for a variety of tasks, including image classification, object detection, and sound recognition. This makes it a versatile tool for a variety of projects.

3.2.2 Dataset Collection from Teachable Machine

This part will provide image collections for each lesson, as well as any relevant statistics collected during the training period by Google Teachable Machine.

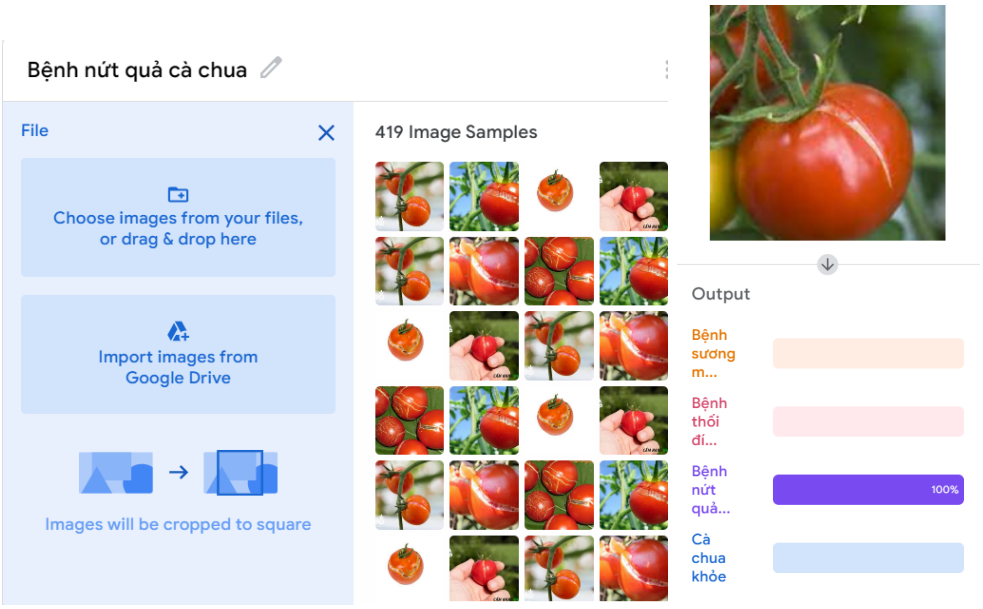
428 pictures are collected and trained to produce an accuracy of 1.0



(a) 428 Image (b) 1.00 Accuracy

Hình 5: Bệnh thối đích cà chua

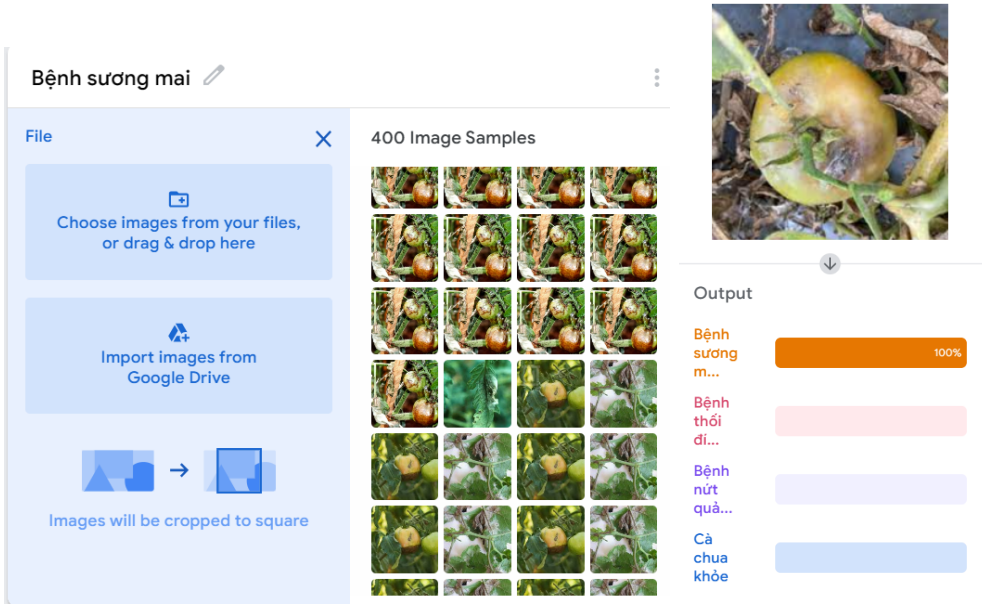
419 pictures are collected and trained to produce an accuracy of 1.0



(a) 419 Image (b) 1.00 Accuracy

Hình 6: Bệnh nứt quả cà chua

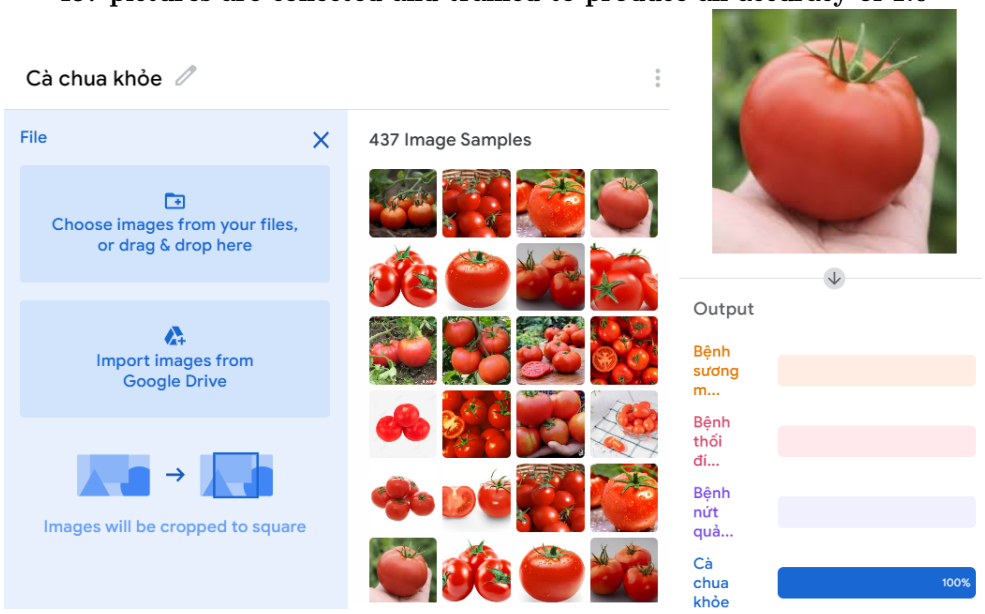
400 pictures are collected and trained to produce an accuracy of 1.0



(a) 400 Image (b) 1.00 Accuracy

Hình 7: Bệnh sương mai

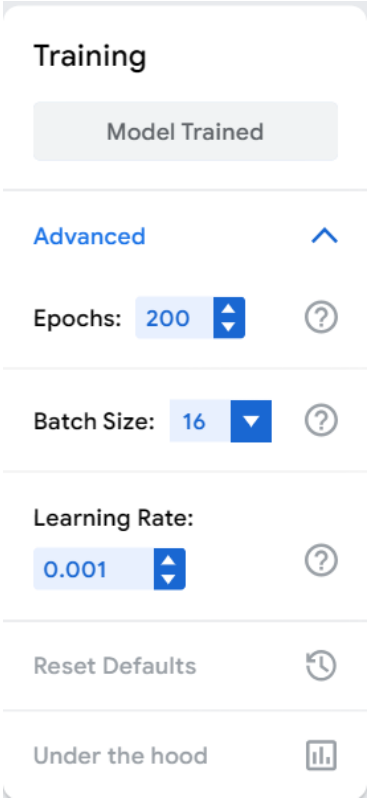
437 pictures are collected and trained to produce an accuracy of 1.0



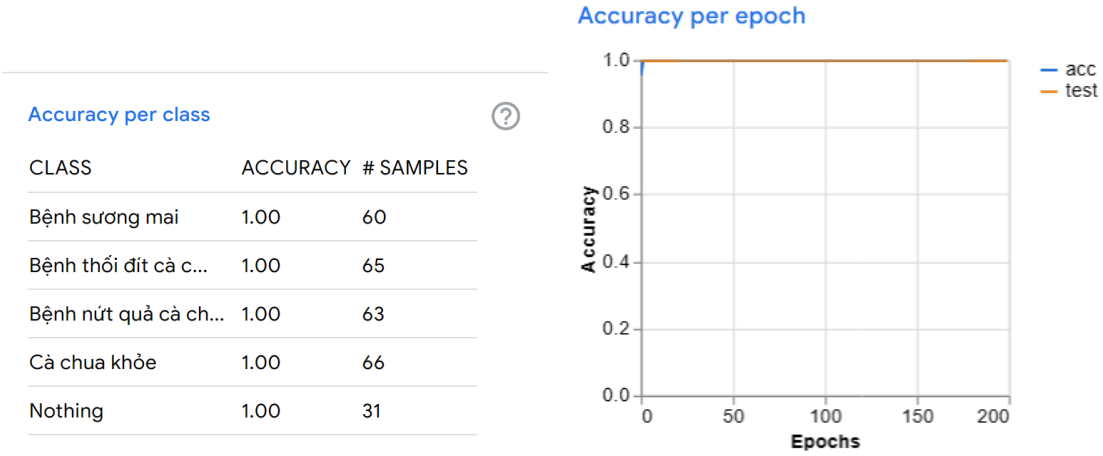
(a) 437 Image (b) 1.00 Accuracy

Hình 8: Cà chua khỏe

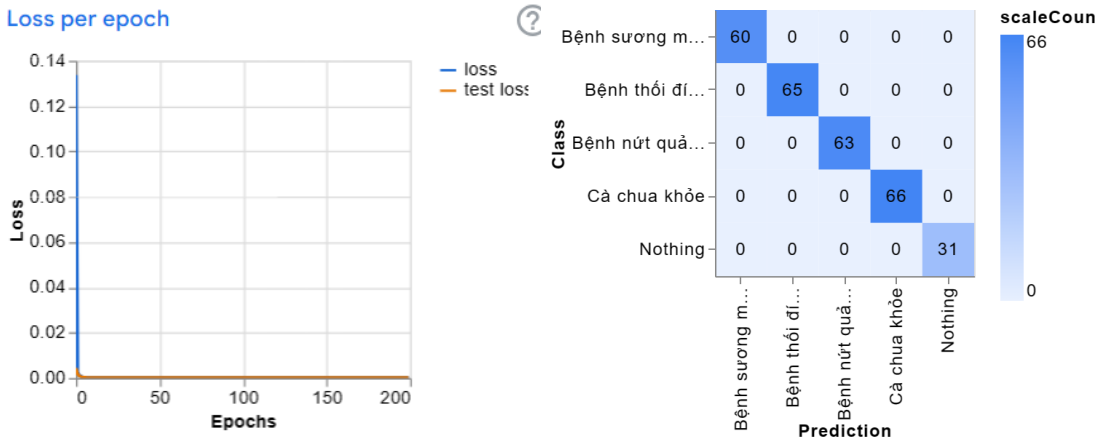
3.2.3 Training Model



Hình 9: Batch size and epochs used for the training process



Hình 10: Accuracy per class and per epoch



Hình 11: Confusion matrix and loss per epoch

3.2.4 Python Source Code

Main Functions (APIs):

```
    AI_Identifying()
#This function capture an image from the webcam.
#Then identifies the object in the image.
#Returns the name of the object and the image in base64 format.
```

```
    check()
#This function get the current data of the environmental conditions.
#Then checks the environmental data and takes appropriate action.
#Might send a notification to the user.
```

```
    client.publish()
#This function publish the message to the Adafruit IO topic.
#Then print a message to the console.
```

Python Source Code After the sensor collects data from the surrounding environment, sent to Arduino Uno. It analyzes the data and sends commands to the actuators and they perform commands to care for tomato plants. Data from Arduino Uno are sent to a Python source code file as an intermediate file in order to upload to the Adafruit IO platform.

Listing 8: Importing Modules

```
import time
import sys
from Adafruit_IO import MQTTClient
import requests
```

```
import time
import serial.tools.list_ports
```

This listing imports the necessary modules for the code. The time module is used to manage time, the sys module is used to access system-specific parameters, the Adafruit_IO module is used to connect to Adafruit IO, the requests module is used to make HTTP requests, the time module is used to manage time, and the serial.tools.list_ports module is used to list the available serial ports.

Listing 9: Serial Port Initialization

```
try:
    ser = serial.Serial(port="COM3", baudrate=9600)
except:
    print("Can not open the port")

def sendCommand(cmd):
    ser.write(cmd.encode())

mess = ""
```

This listing initializes the serial port. The `ser = serial.Serial(port="COM3", baudrate=9600)` line creates a new serial object and opens the serial port at COM3 with a baud rate of 9600. The sendCommand(cmd) line sends a command to the serial port. The readSerial() line reads data from the serial port.

Listing 10: Reading and Processing Data

```
def processData(data):
    data = data.replace("!", "")
    data = data.replace("#", "")
    splitData = data.split(":")
    return splitData

def readSerial():
    bytesToRead = ser.inWaiting()
    output = ""
    if (bytesToRead > 0):
        global mess
        mess = mess + ser.read(bytesToRead).decode("UTF-8")
        while ("#" in mess) and ("!" in mess):
            start = mess.find("!")
            end = mess.find("#")
            output = list(processData(mess[start:end + 1]))
```

```
            if (end == len(mess)):
                mess = ""
            else:
                mess = mess[end+1:]
    return int(float(output[-1]))

def requestData(cmd):
    sendCommand(cmd)
    time.sleep(1)
    return readSerial()
```

This listing reads and processes data from the serial port. The processData(data) line splits the data into a list of values. The readSerial() line reads data from the serial port. This allows the code to collect and analyze environmental data. The data that is read and processed is stored in a format that is easy to understand and use, which makes the code efficient and informative.

Listing 11: AI Identification Google's Python source code

```
from keras.models import load_model
import cv2
import numpy as np
import base64
from PIL import Image
import io
np.set_printoptions(suppress=True)

model = load_model("keras_Model.h5", compile=False)

class_names = open("labels.txt", "r", encoding="utf-8").readlines()

camera = cv2.VideoCapture(0)
def AI_Identifying () :
    def compress_image ( image , quality = 25 ) :
        temp_image = Image.fromarray(image)
        buffer = io.BytesIO()
        temp_image.save(buffer, format ='JPEG', quality = quality)
        compressed_image = Image.open(buffer)
        return np.array(compressed_image)

    ret, image = camera.read()

    image = compress_image ( image , quality = 25)
    res, frame = cv2.imencode (".jpg ", image)
    data = base64.b64encode(frame)
    image = cv2.resize(image, (224, 224), interpolation=cv2.INTER_AREA↩
```

```
        )
28
29      cv2.imshow("Webcam Image", image)
30
31      image = np.asarray(image, dtype=np.float32).reshape(1, 224, 224, ↩
            3)
32
33      image = (image / 127.5) - 1
34
35      prediction = model.predict(image)
36      index = np.argmax(prediction)
37      class_name = class_names[index]
38      confidence_score = prediction[0][index]
39
40      return class_name [2:] , data
```

This listing contains the code for the AI identification function. The (link1) line loads the AI model. The (link2) line reads the list of class names from a file. The AI_Identifying() function identifies the object in the image and returns the name of the object and the image in base64 format.

This allows the code to identify objects in images. The AI identification function is based on a well-known and well-tested model, which means that the code is reliable and accurate.

Listing 12: MQTT Connection with Adafruit IO

```
1 print("MQTT with Adafruit IO")
2
3 AIO_USERNAME = "YOUR_USERNAME"
4 AIO_KEY = "YOUR_AIO_KEY"
```

This listing establishes a connection to Adafruit IO. The AIO_USERNAME and AIO_KEY variables store the Adafruit IO username and key. The `client = MQTTClient(AIO_USERNAME, AIO_KEY)` line creates a new MQTT client. The client.on_connect() function is called when the client connects to Adafruit IO. The client.on_disconnect() function is called when the client disconnects from Adafruit IO. The client.on_subscribe() function is called when the client subscribes to a topic. The client.on_message() function is called when the client receives a message.

This allows the code to publish and subscribe to messages. The connection to Adafruit IO is made in a way that is easy to understand and modify, which makes the code flexible and customizable.

Listing 13: Implemental Tasks Functions

```
1 def connected(client):
2     print("Server connected ...")
```

```
    client.subscribe("multi-prj.light-switch")
    client.subscribe("multi-prj.nebulizer-switch")
    client.subscribe("multi-prj.pump-switch")
    client.subscribe("multi-prj.sunshade-net-switch")

def subscribe(client , userdata , mid , granted_qos):
    print("Subscribed...")

def disconnected(client):
    print("Disconnected...")
    sys.exit (1)

def message(client , feed_id , payload):
    print("Received: " + payload)
    if(feed_id == "equation"):
        global_equation = payload
        print(global_equation)
client = MQTTClient(AIO_USERNAME , AIO_KEY)

client.on_connect = connected
client.on_disconnect = disconnected
client.on_subscribe = subscribe
client.on_message = message
client.connect()
client.loop_background()
```

This listing contains the code for the implemental tasks functions. The connected() function is called when the client connects to Adafruit IO. The subscribe() function is called when the client subscribes to a topic. The disconnected() function is called when the client disconnects from Adafruit IO. The message() function is called when the client receives a message.

This allows the code to take appropriate action based on the environmental data. The implemental tasks functions are well-organized and easy to understand, which makes the code easy to maintain and extend.

Listing 14: Environmental Analysis Functions

```
def checkTemperature(temperature):
    if (temperature > 20):
        client.publish("multi-prj.sunshade-net-switch", "1")
        client.publish("multi-prj.notifications", "Temperature is too ↩
            high!")
    elif (temperature < 10):
        client.publish("multi-prj.sunshade-net-switch", "0")
        client.publish("multi-prj.notifications", "Temperature is too ↩
            low!")
```

```
8       else :
9           client.publish("multi-prj.sunshade-net-switch", "0")
10
11  def checkHumidity(humidity):
12      if (humidity < 40):
13          client.publish("multi-prj.nebulizer-switch", "1")
14          client.publish("multi-prj.notifications", "Humidity is too low!")
15      elif (humidity > 60):
16          client.publish("multi-prj.nebulizer-switch", "0")
17          client.publish("multi-prj.notifications", "Humidity is too high!")
18      else :
19          client.publish("multi-prj.nebulizer-switch", "0")
20
21  def checkBrightness(brightness):
22      if (brightness < 800):
23          client.publish("multi-prj.light-switch", "1")
24          client.publish("multi-prj.notifications", "Brightness is too low!")
25      elif (brightness > 1300):
26          client.publish("multi-prj.light-switch", "0")
27          client.publish("multi-prj.notifications", "Brightness is too high!")
28      else:
29          client.publish("multi-prj.light-switch", "0")
30
31  def checkSoilMoisture(soilmoisture):
32      if (soilmoisture < 55):
33          client.publish("multi-prj.pump-switch", "1")
34          client.publish("multi-prj.notifications", "Soil Moisture is too low!")
35      elif (soilmoisture > 70):
36          client.publish("multi-prj.pump-switch", "0")
37          client.publish("multi-prj.notifications", "Soil Moisture is too high!")
38      else:
39          client.publish("multi-prj.pump-switch", "0")
```

This listing contains the code for the environmental analysis functions. The checkTemperature() function checks the temperature and takes appropriate action. The checkHumidity() function checks the humidity and takes appropriate action. The checkBrightness() function checks the brightness and takes appropriate action. The checkSoilMoisture() function checks the soil moisture and takes appropriate action.

This allows the code to monitor the environment and take appropriate action. The environmental analysis functions are based on well-known and well-tested algorithms, which means that the code is reliable and accurate.

Listing 15: Program Flow Control

```
pre_ai_result =""
ai_results =""
count = 5
while True:
    count = count - 1
    if (count == 0):
        count = 5
        pre_ai_result = ai_results
        ai_results, ai_cap = AI_Identifying()
        if pre_ai_result != ai_results:
            client.publish("multi-prj.notifications", ai_results)
            client.publish("multi-prj.ai-camera",ai_cap)
        time.sleep(5)
        temperature = requestData("t")
        time.sleep(5)
        humidity = requestData("h")
        time.sleep(5)
        brightness = requestData("b")
        time.sleep(5)
        soilmoisture = requestData("s")
        time.sleep(5)

        client.publish("multi-prj.temperature", temperature)
        client.publish("multi-prj.humidity", humidity)
        client.publish("multi-prj.light-density", brightness)
        client.publish("multi-prj.soil-moisture", soilmoisture)
        checkTemperature(temperature)
        checkHumidity(humidity)
        checkBrightness(brightness)
        checkSoilMoisture(soilmoisture)
    time.sleep(1)

    keyboard_input = cv2.waitKey(1)
    if keyboard_input == 27:
        break
```

This listing controls the flow of the program. The while True: loop runs continuously. The count = count - 1 line decrements the count variable. The if (count == 0): statement checks if the count variable is equal to 0. If it is, the count variable is reset to 5 and the ai_results variable

is assigned the value of the AI_Identifying() function. The client.publish() function publishes a message to Adafruit IO. The time.sleep() function pauses the execution of the program for a specified amount of time. The keyboard_input = cv2.waitKey(1) line waits for a keyboard input. If the keyboard input is 27, the program is terminated.

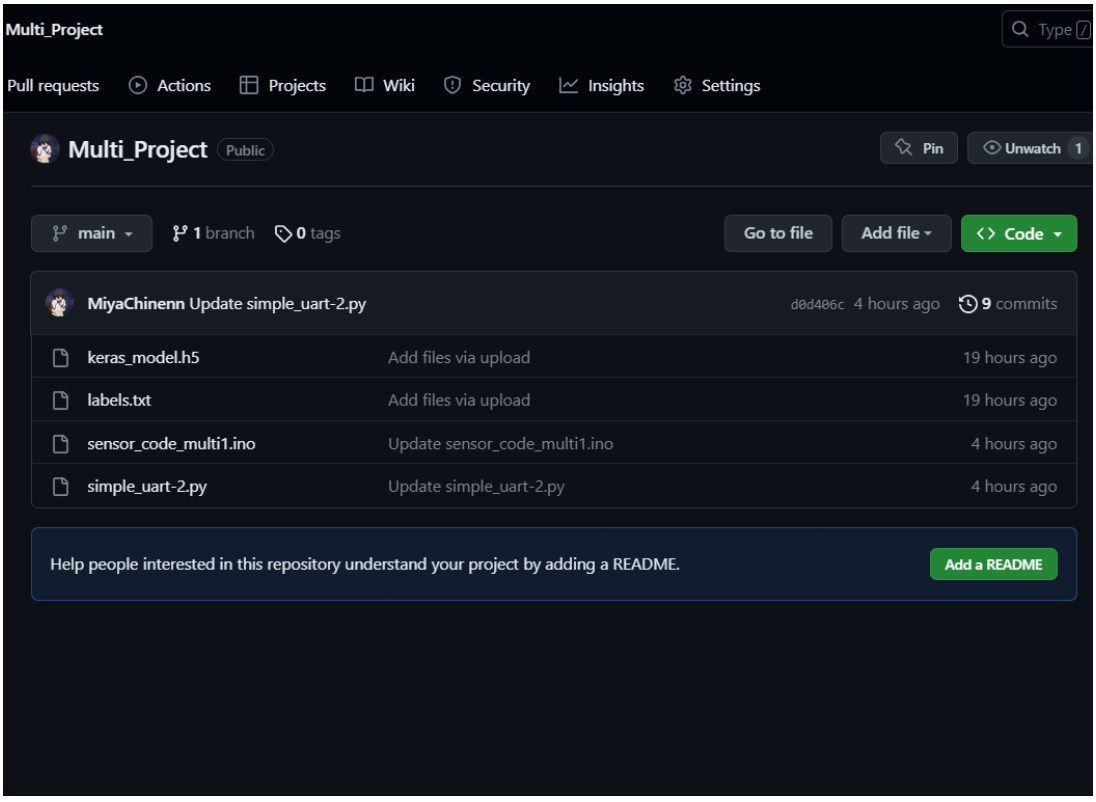
This ensures that the code executes in a predictable and orderly manner. The program flow is controlled in a way that is easy to understand and modify, which makes the code flexible and customizable.

Overall, this figure could help users who want to keep their plants healthy and prevent damage. It is easy to use, reliable, flexible, and accurate. The code uses well-known and well-tested modules, algorithms, and data formats, which makes it reliable and trustworthy. The code is also easy to understand and modify, which makes it flexible and adaptable to different needs. Overall, the code is a powerful tool that can help users to keep their plants healthy and prevent damage.

4 System Deployment

The Automatic Tomato Plant Care System is deployed using GitHub as the source code repository. The deployment process involves several steps to ensure seamless integration and proper functioning of the program: (Setting up for the host device)

- **Step 1:** Click on this link to get access to our source code on Github: `https://github.com/MiyaChinenn/Multi_Project`



- **Step 2:** Download all four following files on the GitHub:

  - keras_model.h5
  - labels.txt
  - sensor_code_multi1.ino
  - simple_uart-2.py

- **Step 3:** Download and install Python compiler and Arduino IDE.
- **Step 4:** Open Terminal on your computer then install all the following packages by copying these codes:
  - pip install adafruit_IO
  - pip install time
  - pip install sys
  - pip install requests
  - pip install numpy
  - pip install keras
  - pip install tensorflow
  - pip install opencv-python
  - pip install pyserial
- **Step 5:** Open simple_uart-2.py by Python compiler and set up the system by following these steps:
  1. Plug the arduino set to the computer. In order to know what port the sensor connect to the computer, check in Device Manager/Ports, then change the 'port' in line 8.

```
5   import time
6   import serial.tools.list_ports
7   try:
8       ser = serial.Serial(port="COM3", baudrate=9600)
9   except:
10      print("Can not open the port")
```

  2. Then, paste the computer locations of 'keras_Model.h5' and 'labels.txt' in line 55 and line 58 respectively.

```
53
54  # Load the model
55  model = load_model("keras_Model.h5", compile=False)
56
57  # Load the labels
58  class_names = open("labels.txt", "r", encoding="utf-8").readlines()
59
```

3. After that, use the username and password of the io.adafruit.com website that we give you to sign in (manhpham-12345678).
   Look for the contemporary link, then copy and paste in the line 99 and 100.

```
99
100     AIO_USERNAME = "YOUR_USERNAME"
101     AIO_KEY = "YOUR_AIO_KEY"
102
```

- **Step 6:** Open sensor_code_multi1.ino file by Arduino IDE and install all the libraries:
  - adafruit io arduino
  - dht sensor library
- **Step 7:** Upload Arduino file first, then run Python file:
- **Step 8:** Using your phone access io.adafruit.com website via this link to check environmental conditions and notifications: `https://io.adafruit.com/manhpham/dashboards/multi-project`

5 Conclusion

The Automatic Tomato Plant Care system represents a groundbreaking innovation that harnesses the power of the Internet of Things (IoT) and Artificial Intelligence (AI) to revolutionize tomato agriculture. By employing an array of sensors to collect crucial environmental data such as temperature, humidity, light intensity, and soil moisture, our system can effectively monitor the plant's conditions in real-time. This data is then meticulously analyzed by our AI algorithms to determine the optimal growth conditions for tomato plants, ensuring that they thrive and reach their full potential.

Our system's implementation and development showcase a cutting-edge approach to agricultural automation. Utilizing AI cameras to detect objects and sensors to collect data, our technology ensures accurate and reliable information gathering. By integrating the Adafruit server for remote monitoring and control of connected devices, farmers can access real-time data and make informed decisions from anywhere in the world. The utilization of the MQTT protocol for communication between our system and Adafruit IO further enhances the efficiency and responsiveness of our platform.

The impact of our project on the tomato agriculture industry is substantial. Firstly, it significantly reduces the reliance on manual labor and human intervention in plant care, leading to cost savings and increased efficiency for farmers and cultivators. By automating critical care aspects, such as watering and lighting control, the workload is significantly lightened, freeing up valuable time and resources for other essential tasks.

Moreover, the automatic and precise adjustments made by our system ensure that tomato plants consistently receive the optimal conditions for growth, leading to a marked enhancement in tomato qualities. This includes improved taste, size, color, and overall yield, which directly translates to higher profitability for farmers and higher satisfaction for consumers.

This groundbreaking automation is a game-changer, democratizing tomato cultivation and making it accessible to a wider audience, particularly individuals with limited time or gardening experience. While still in its early stages, the system's scalability and adaptability hold the potential for seamless integration into diverse settings, ranging from small home gardens to expansive commercial farms. Looking ahead, the promising impact of our project in agriculture is evident as the adoption of IoT and AI technologies continues to rise. With the potential to become an industry standard, our automatic tomato plant care system could extend its application beyond tomatoes, revolutionizing agricultural practices for various crops. By promoting sustainability, efficiency, and improved crop quality, our system is driving agriculture towards a more technologically advanced and environmentally conscious future.

In conclusion, our automatic tomato plant care program represents a pioneering approach to modernizing agriculture, enhancing productivity, and ensuring a sustainable supply of high-quality tomatoes to meet the demands of an ever-growing population. Through continued development and widespread adoption, our project has the potential to transform the way we grow crops and contribute significantly to the advancement of agricultural practices.