



SMART GARDEN ASSISTANT

Group 27

Abstract

An Embedded System that takes care of
plants and all its functional needs

Authors: Muhammad Choudhury and Muhammad Ahmed
Advisors: Dr. Bassem Alhalabi and David Wilson

Contents

| | |
|---|----|
| 1. Introduction | 2 |
| 1.1 Problem Description | 2 |
| 1.2 Significance of the Problem..... | 3 |
| 1.3 Overview of the Solution | 4 |
| 1.4 Goals and Objectives..... | 4 |
| 1.5 Related Research | 5 |
| 2. System Design | 7 |
| 2.1 System Requirements..... | 7 |
| 2.2 System Diagrams | 8 |
| 2.3 Project Diagram..... | 9 |
| 3. System Implementation | 10 |
| 3.1 Software..... | 10 |
| 3.2 Hardware | 11 |
| 3.3 User Interface..... | 13 |
| 3.4 Device Communication | 14 |
| 4. Testing and Performance Evaluation..... | 15 |
| 5. Budget | 16 |
| 6. Project Manage and Distribution Workload | 17 |
| 7. References..... | 18 |

1. Introduction

Over the past decade, we have seen major climate changes which have resulted in more natural disasters like forest fires, tsunamis and hurricanes which especially affects us since Florida is home to those. To counter the effect of those human induced climate changes, which are mainly due to excessive output of carbon dioxide into the atmosphere, we need to grow more plants and allow the current generation of plants to grow at best possible rate. Hence, our project was made with that issue in mind in accordance with that everyone regardless of where they are has to be on a fixed schedule due to COVID-19. Also with our Smart Garden Assistant, we will not have to use excessive amounts of water because our technology knows exactly how much water the plants need. Lastly, using our Smart Garden Assistant, the user can also control use of motors to give more/less water, oxygen and light to their plants and they do this through the app that will show them their plants current condition and let them set values they prefer their plants condition to be within or use app to manually operate the motors.

1.1 Problem Description

The problem we are solving with our system is to allow the users to take care of their plants regardless of them being near their plants. We allow this to the users

by providing them an app that gives access to the plant's data, which lets the user know the condition the plant is in, and the user asks the option to turn the actuators on and off to provide more or less air or water. The user can also set their preferred settings that the system will then follow to take care of the plant. Hence, we not only allow the user to take care of the plants but also are improving the environment by countering the effect that excessive carbon dioxide is having on the planet.

1.2 Significance of the Problem

The significance of this problem is to improve the health of our environment by allowing the user of the system easy access to data and actuators that would allow the user to ensure the plants are experiencing good growth and health. Also, due to our system being used, the reported loss of up to 50% of the water used for outdoor irrigation in this country may be going to waste, according to EPA reports. The wasted water will be gone and a lot of money will be saved in doing so. Hence, we are improving the health and life of plants while being responsible for water use. Therefore, using a Smart Garden Assistant can help you keep your water consumption to a reasonable level. It is also an excellent Garden maintenance tool that can help improve your Garden's health and appearance since it let's you control actuators which would give the user's plant more/less water, oxygen and light and also keep plants in the range of conditions preferred by the and entered by the user.

1.3 Overview of the Solution

Our system uses ESP-32 to gather all the data from the sensors which through serial communication is passed on to the Raspberry Pi which then uses the data control the actuators, via commands. The ESP-32 and Raspberry Pi have bluetooth which is how the app is able to use the internet to communicate with those processes to control the actuators and also to render the data about the plant on the app. Also, the app allows the user to explicitly control the actuators and adjust the range of conditions in which they want their plant to stay within.

1.4 Goals and Objectives

The main goal of this project is to create a modern Embedded System that will essentially take care of your plant for you in the same way a human being would. As society has progressed for the last several decades, we have seen that technology makes our lives easier by doing things that we don't need to do ourselves like with the invention of autonomous vehicles. And in the same way we made our system advanced enough so that it can take care of not only how hot or cold the plant is but also if it needs more water and also if the blinds should be opened or closed depending on light exposure. So this leads to our goal of using microcontrollers to

take in data from sensors and then adjust the actuators depending on the sensory values. But we didn't want to make something too simple so we decided to write smarter code that will not do anything that would harm the plant or do anything that is otherwise illogical. We made the project smart by creating dead zones in the code so that actuators wouldn't shift between on and off several times in a span of a few minutes which is efficient and logical for many reasons. Also since the user might want to add their own personality to it, we made the system so that it will change to user input so that, for instance, the fan will turn on if the user wants to make the plant's soil at 90% wet when it is currently lower than that.

1.5 Related Research

Engineers make it their life goal to solve problems that exist on this planet and the prospect of watering plants is no different. We did some research to find anything on the market that is close to what we are trying to create and what we found was the Rachio 3 smart sprinkler which waters your plants in a way that is efficient and cuts back on unnecessary watering. This means that it will not water the plants after it rains or if they are frozen. And this is all controlled by the smartphone meaning

that the user has control of everything that is going on and they can alter any of the settings so that they can make their lawn turn out to be however they want it to be. This is similar to what we plan on doing because we intend to create a smart system that cuts back on waste while also giving the user the full freedom and power to change anything to their plant.



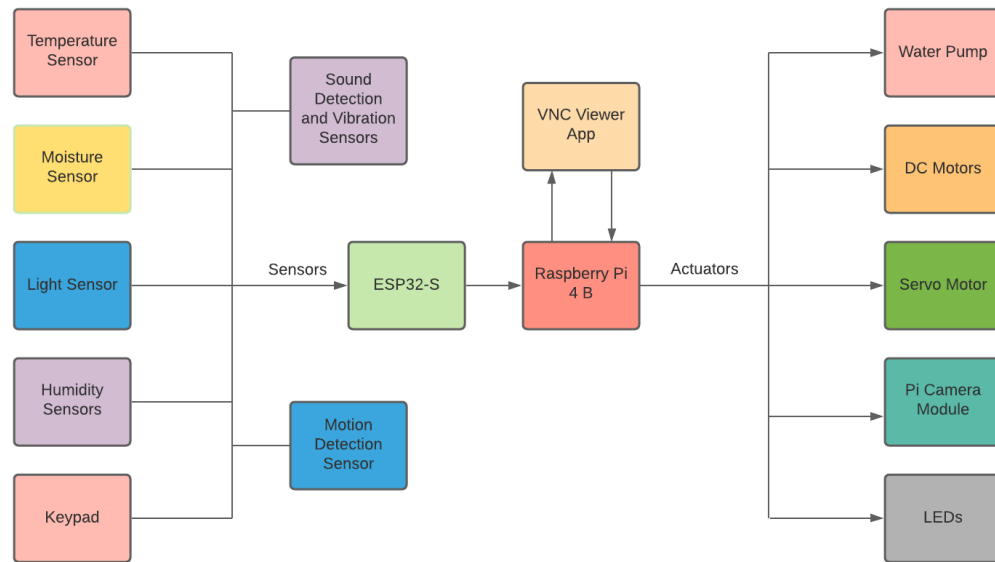
The Smart Garden Assistant

2. System Design

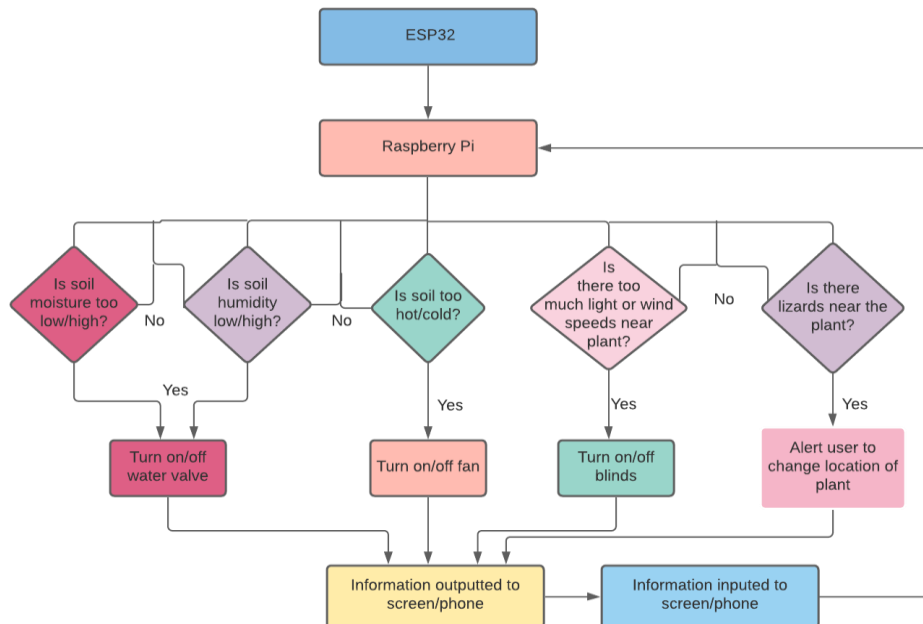
2.1 System Requirements

- Humidity Sensor: Senses the humidity in the soil
- Temperature Sensors: Senses the temperature in the air and soil
- Light Sensor: Senses the intensity of the outdoor light
- Gas Sensor: Senses the CO2 levels in the air near the plant
- Moisture Sensors: Senses the amount of moisture in the air and soil
- DC Motor: Acts as a fan to cool the air near the plant
- Servo Motor: Acts as an opening for water valve
- Keypad: Used to take in user input
- Microcontroller: Used to process data and control the actuators
- Bluetooth Module: Sends data from microcontroller to smartphone
- VNC Viewer App: User is able to control and monitor everything
- Mini Water Pump: Used to send in water when plant is dehydrate
- Vinyl WaterPipe: For the water to flow into the plant
- Relay Module: Will switch from controlling one device to another
- USB Power Supply: To power everything
- Breadboard: To connect everything together requirement

2.2 System Diagrams

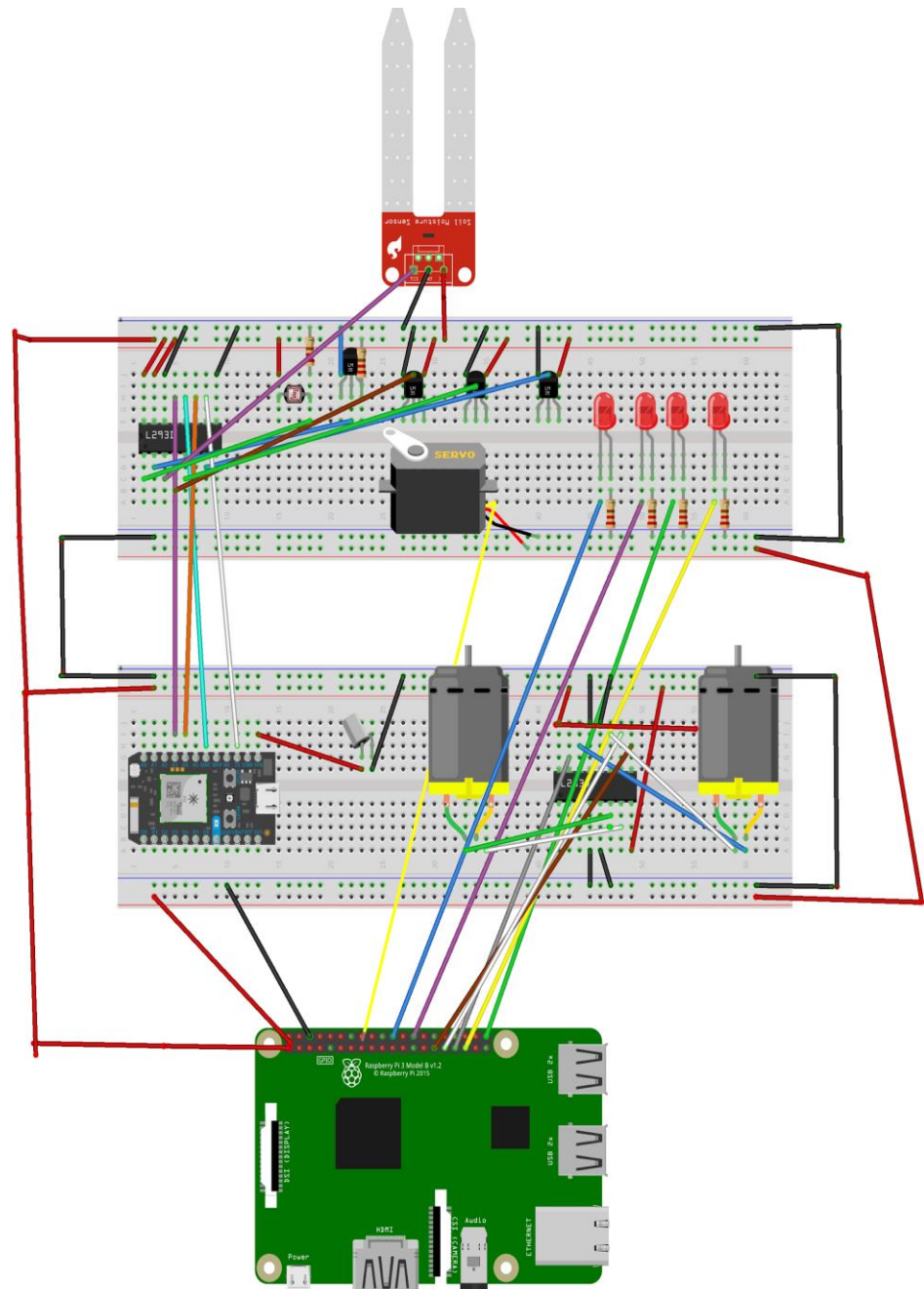


System Diagram



Logical Flowchart of System

2.3 Project Diagram



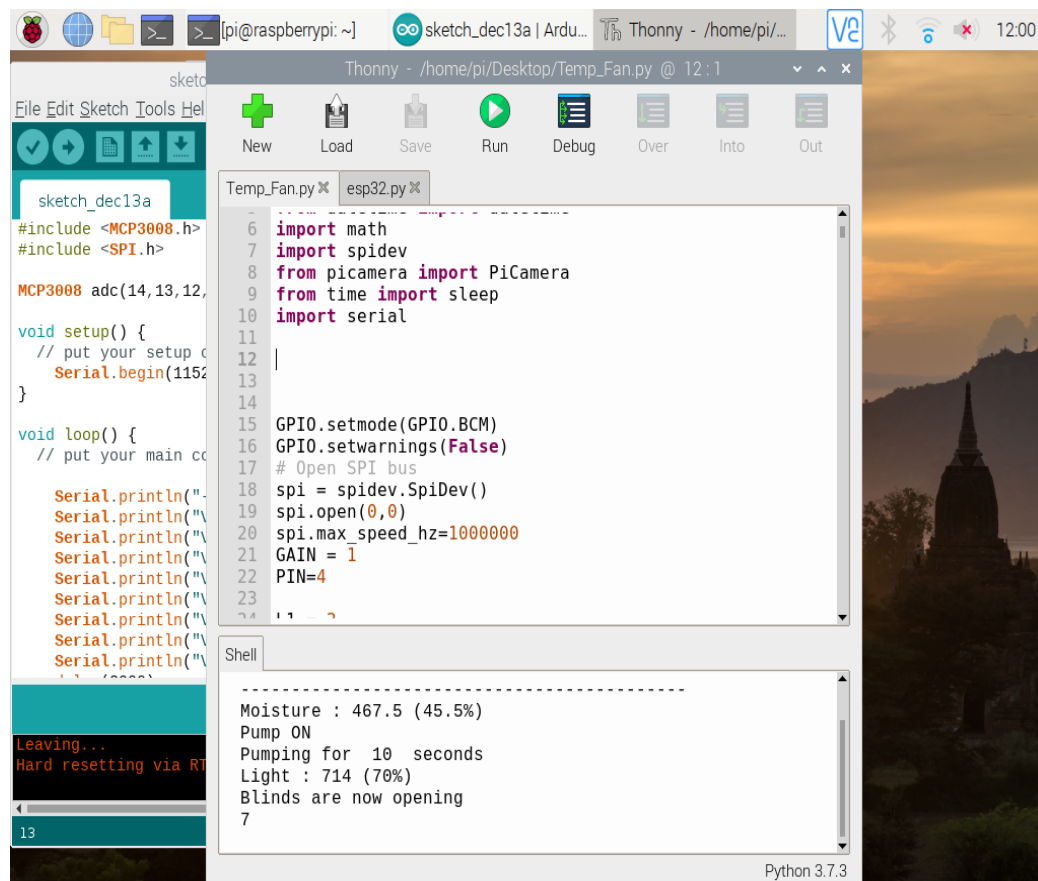
fritzing

Diagram of Project

3. System Implementation

3.1 Software

For this project we used the Thonny IDE in the Raspberry Pi to program everything in Python related to the Pi. We had to pip install several Python modules for the program to control several components of our project like the Pi camera modules and the keypad. We also programmed in C when we used Arduino IDE for controlling the ESP32 microcontroller. There was only one program per each microcontroller and it was set up so that data can be sent from the program in the Arduino IDE to the Thonny IDE.



```

Temp_Fan.py
6 import math
7 import spidev
8 from picamera import PiCamera
9 from time import sleep
10 import serial
11
12
13
14
15 GPIO.setmode(GPIO.BCM)
16 GPIO.setwarnings(False)
17 # Open SPI bus
18 spi = spidev.SpiDev()
19 spi.open(0,0)
20 spi.max_speed_hz=1000000
21 GAIN = 1
22 PIN=4
23
24

```

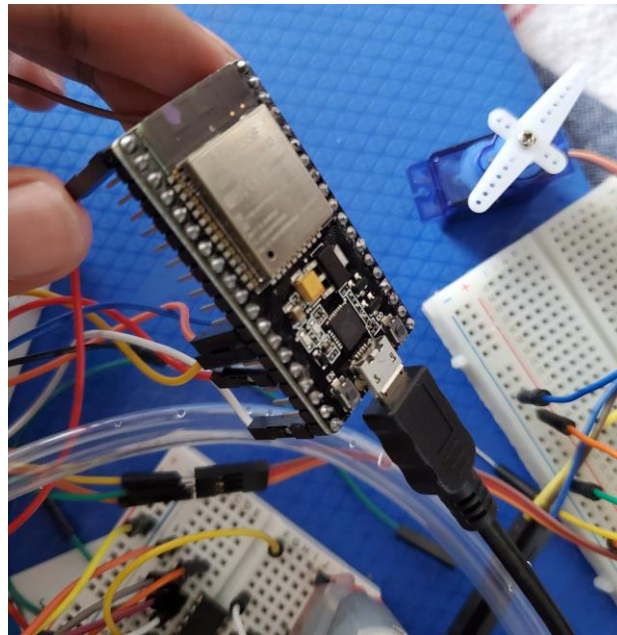
```

Moisture : 467.5 (45.5%)
Pump ON
Pumping for 10 seconds
Light : 714 (70%)
Blinds are now opening
7

```

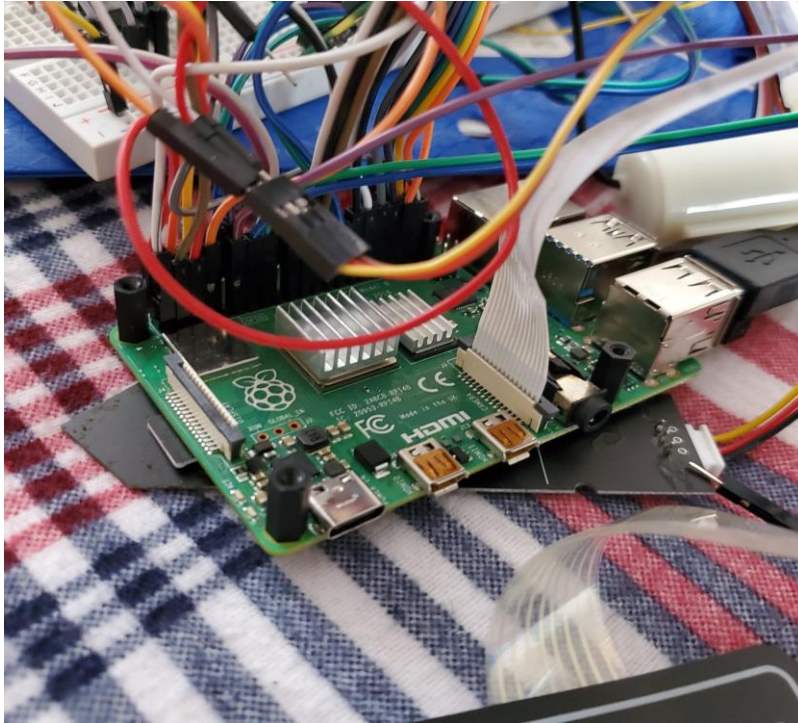
Data is being sent from the ESP32 to the Raspberry Pi

3.2 Hardware

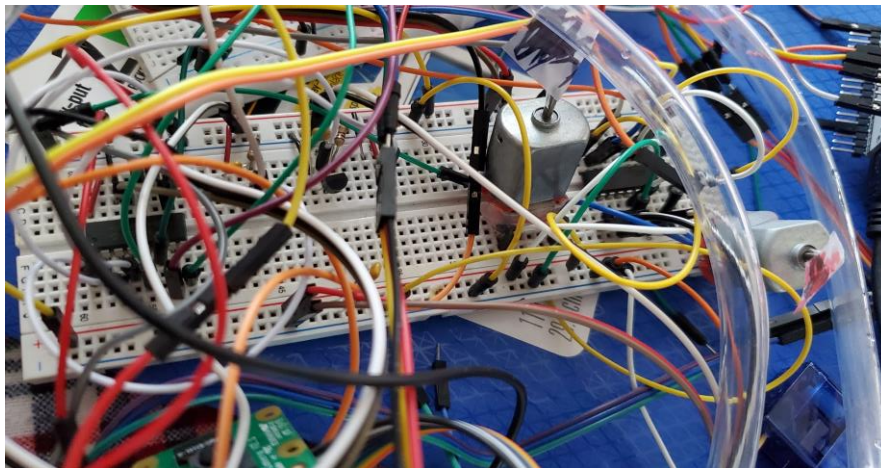


The ESP32 Microcontroller

We used the Raspberry to control all of the actuators like the water pump and DC motors. We also used an ESP32 microcontroller to get all of the data from the sensors like the light and temperature sensors which is a good idea as it creates a separation of concerns. The program that ran the ESP32 sent data to the program in Raspberry Pi using serial communication through a USB connection. We also used a Pi camera module to take pictures of the plant and a keypad was also used to input numbers into the Pi so values could be set by the user. LEDs were also used so that we could know which actuators were on.



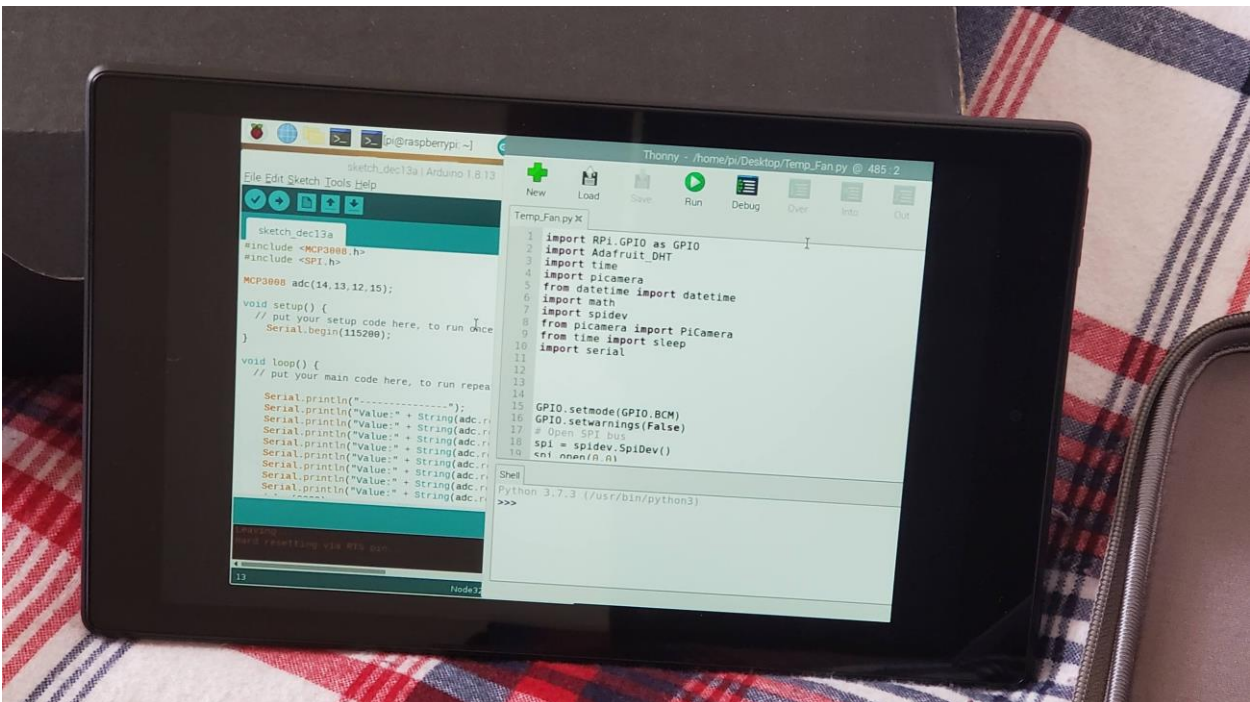
The Raspberry Pi Microcontroller



Most of the other hardware components on the breadboard

3.3 User Interface

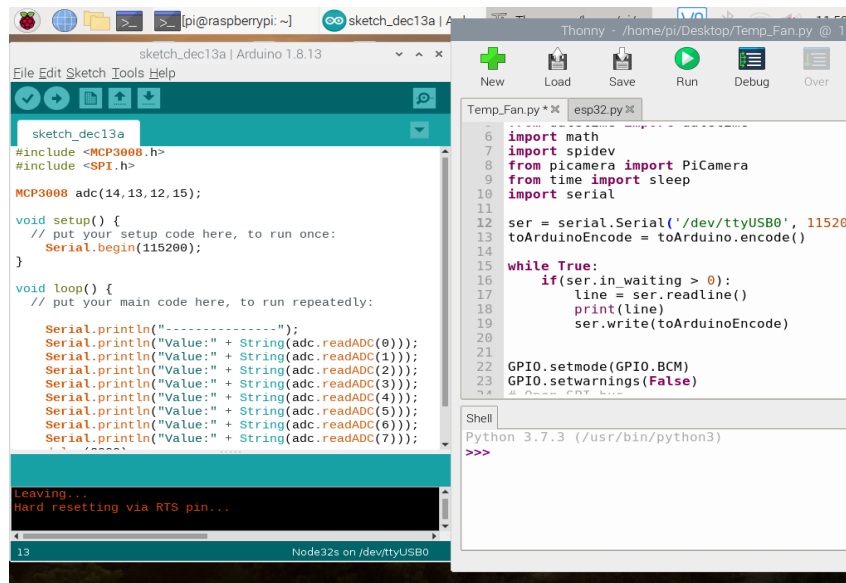
The user can interact with the system through various ways. They can input data through a keypad, keyboard, or through the VNC viewer app such as the values of what temperature, moisture, or light exposure that they want the plan to have. And all the information and prompts will be displayed through the VNC viewer app which can be downloaded through one's smartphone or tablet. The mobile device will have to share the same Wi-Fi as the Pi though for that to work.



We used the VNC Viewer application for the user to interact with

3.4 Device Communication

Information regarding sensory input is sent from the ESP32 to the Raspberry Pi through a USB connection so that the Pi can do most of its necessary tasks like controlling the actuators. The ESP32 is wired up to an MCP3008 ADC which makes it easy to collect all the data from all the various sensors then send it to the devices. The serial communication between both microcontrollers was initially hard to configure but it worked out fine at the end and we ended up with more GPIO pins to use after adding the ESP32 to the project.



The image shows two software interfaces side-by-side. On the left is the Arduino IDE, displaying a C++ sketch named 'sketch_dec13a'. The sketch includes the MCP3008 library and sets up a serial connection at 115200 baud. The loop function reads 8 ADC values from the MCP3008 and prints them to the serial monitor. The serial monitor shows the output: 'Leaving... Hard resetting via RTS pin...'. On the right is the Thonny Python IDE, displaying a Python script named 'esp32.py'. The script imports the serial module and sets up a serial connection to the ESP32 at '/dev/ttyUSB0' with a baud rate of 115200. It then enters a while loop that reads data from the ESP32 and prints it to the console. The shell at the bottom shows 'Python 3.7.3 (/usr/bin/python3)' and a prompt '>>>>'.

```

sketch_dec13a | Arduino 1.8.13
File Edit Sketch Tools Help

sketch_dec13a
#include <MCP3008.h>
#include <SPI.h>

MCP3008 adc(14,13,12,15);

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
}

void loop() {
  // put your main code here, to run repeatedly:

  Serial.println("-----");
  Serial.println("Value:" + String(adc.readADC(0)));
  Serial.println("Value:" + String(adc.readADC(1)));
  Serial.println("Value:" + String(adc.readADC(2)));
  Serial.println("Value:" + String(adc.readADC(3)));
  Serial.println("Value:" + String(adc.readADC(4)));
  Serial.println("Value:" + String(adc.readADC(5)));
  Serial.println("Value:" + String(adc.readADC(6)));
  Serial.println("Value:" + String(adc.readADC(7)));
  // ... (more code) ...
}

Leaving...
Hard resetting via RTS pin...

Node32s on /dev/ttyUSB0

Thonny - /home/pi/Desktop/Temp_Fan.py @ 12
New Load Save Run Debug Over

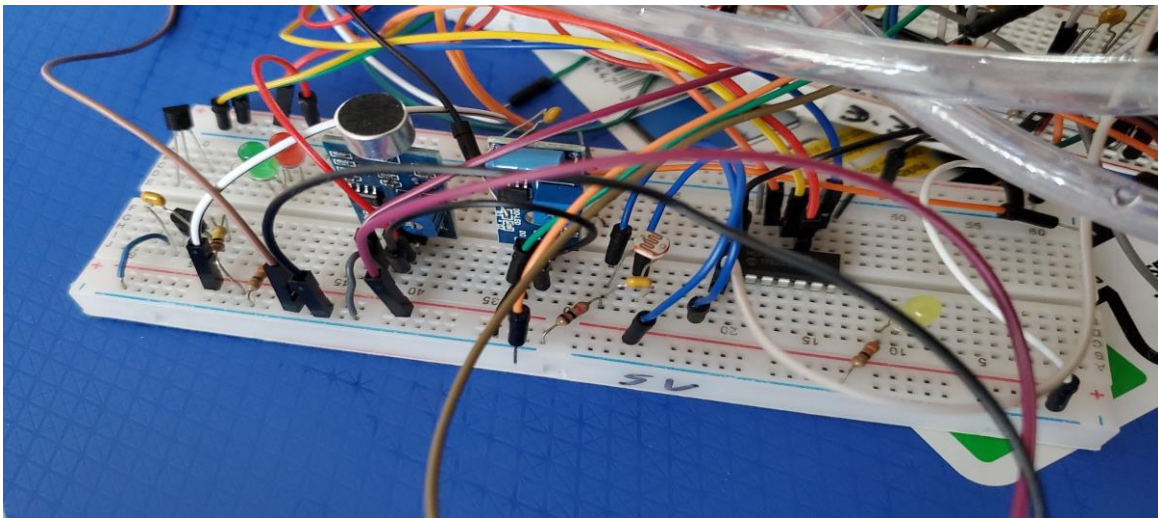
Temp_Fan.py * esp32.py
6 import math
7 import spidev
8 from picamera import PiCamera
9 from time import sleep
10 import serial
11
12 ser = serial.Serial('/dev/ttyUSB0', 115200)
13 toArduinoEncode = toArduino.encode()
14
15 while True:
16     if(ser.in_waiting > 0):
17         line = ser.readline()
18         print(line)
19         ser.write(toArduinoEncode)
20
21 GPIO.setmode(GPIO.BCM)
22 GPIO.setwarnings(False)
23
24
Shell
Python 3.7.3 (/usr/bin/python3)
>>>

```

Data is being sent from the ESP32 to the Raspberry Pi

4. Testing and Performance Evaluation

For most of the project it turned out that our project ran well. The MCP3008 ADC is used to turn the analog signals into digital values for the Pi to read with a 10-bit precision which is enough for this project as we only need to turn on or off actuators given what values we have. We altered the environment in the room to see how the system would react to the change to measure how well it performs as a garden assistant. Meaning we dimmed and increased the lights and changed the room temperature to see what would happen to the sensors and actuators and how the program would respond in outputting the data. Everything worked fine after several dozens of trials were held.



All of the sensors used in this project were tested repeatedly

5. Budget

| Component Name | Price |
|-----------------------|----------------|
| ESP-WROOM-32 | \$10.99 |
| Mini Water Pump | \$4.99 |
| Moisture Sensor | \$7.99 |
| MCP3008 (ADC) | \$11.99 |
| Jumper Wires | \$5.99 |
| Sensor Kit | \$18.99 |
| Pi Camera Module | \$23.99 |
| Total | \$84.93 |

6. Project Manage and Distribution Workload

Muhammad Choudhury: Specialized in wiring most of the hardware components together including sensors and actuators to the microcontrollers. The physical project was kept at Choudhury's so it was his duty to maintain those pieces and make sure there were no damages. He also wrote most of the program that runs the embedded system for both the Raspberry Pi and the ESP32.

Muhammad Ahmed: Planned out most of the subsystems of the project like the system that cools the plant and also the subsystem that waters the plant. He also wrote most of the final report of this project and the executive summary.

7. References

“Rachio 3 Smart Sprinkler Controller.” Rachio 3 Smart Sprinkler, Rachio, rachio.com/rachio-3/

Projects.raspberrypi.org, Raspberry Pi Foundation,
projects.raspberrypi.org/en/projects/getting-started-with-picamera/5

“MCP3008 - Analog to Digital Converter.” MCP3008 - Analog to Digital Converters, Microchip, www.microchip.com/wwwproducts/en/MCP3008

“ESP32-S, WiFi+Bluetooth Module Based on ESP32.” ESP32-S, WiFi+Bluetooth Module, Waveshare, www.waveshare.com/esp32-s.htm

Staff, Maker.io. “How to Communicate Between Arduino Boards and Raspberry Pi SBCs.” Communication between Raspberry Pi and Other Boards, Maker.io, 29 July 2020, www.digikey.com/en/maker/blogs/2020/how-to-communicate-between-arduino-boards-and-raspberry-pi-sbcs