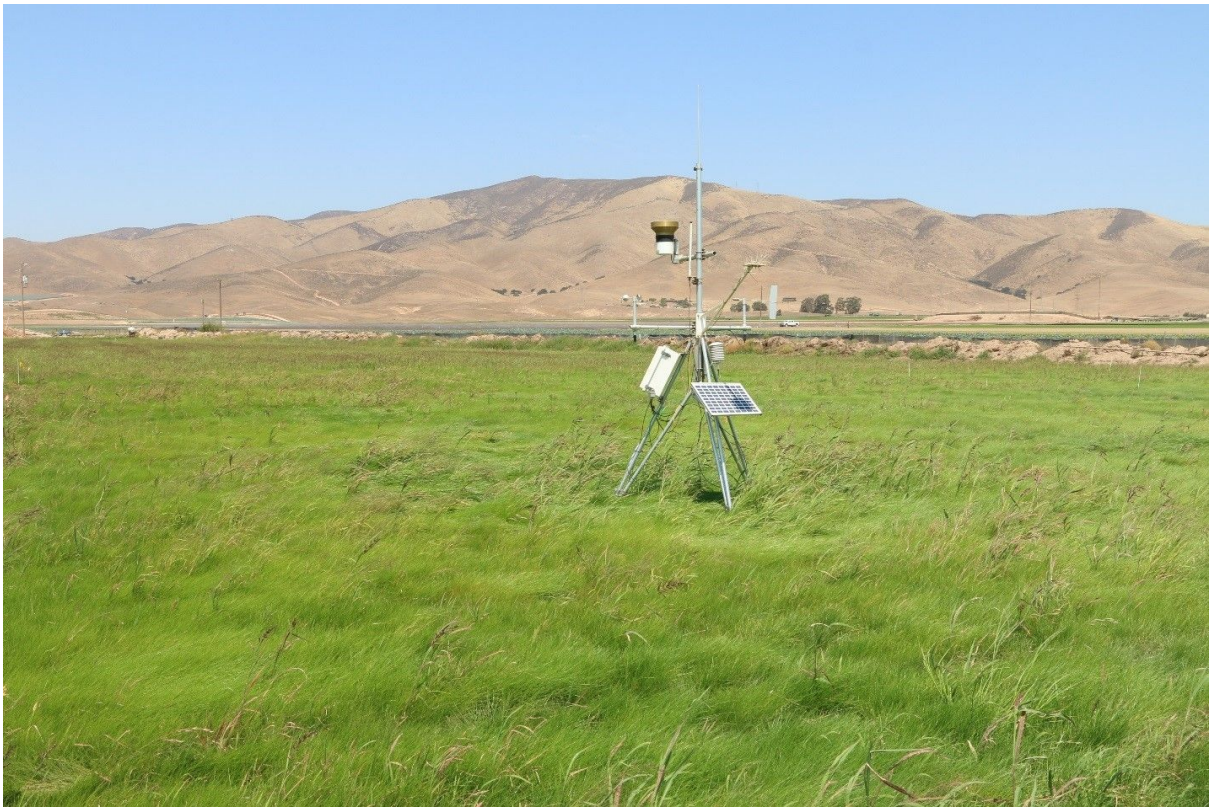


Final Project
Atmosphere Monitoring System

Group members	Student ID
Mehdi Lakhoua	34592546
Rijul Arora	80972263
Ye Myat Kyaw	42712819
Yuki Hayashi	28763963

Introduction

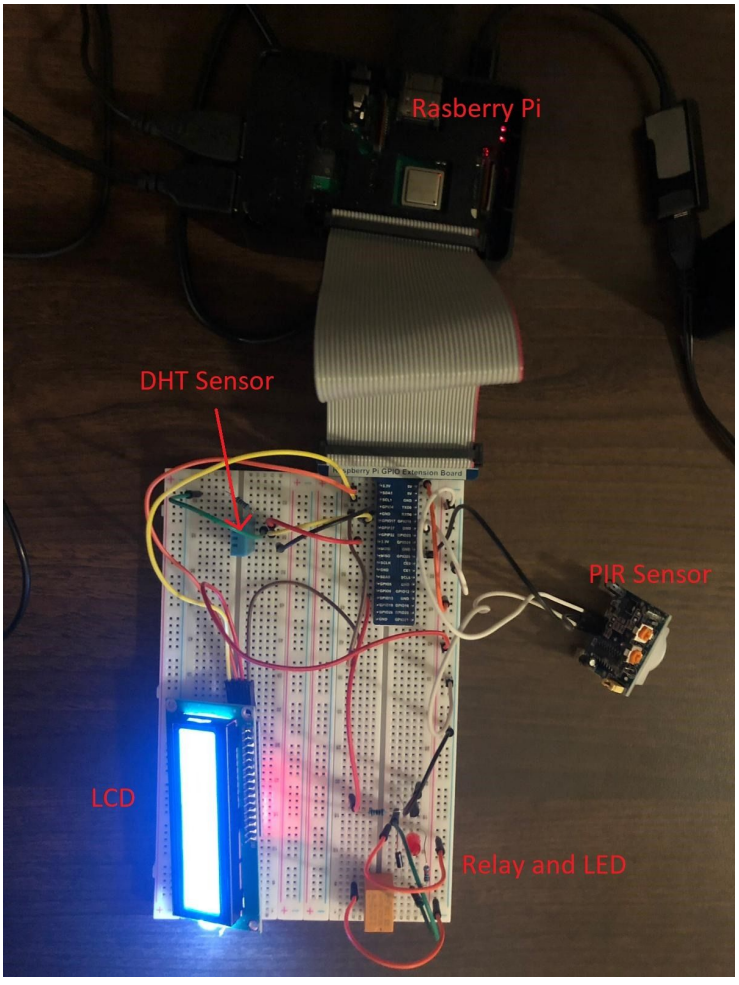
Water is a scarce resource. Thus, we would like to avoid wasting as much water as possible. A lot of water is wasted during irrigation, so this project aims to solve this by creating a system that can efficiently irrigate soil. We will use humidity and temperature values from both the local area and the CIMIS station in Irvine to calculate how long we should be irrigating our soil.

Hardware Setup

1. Equipments

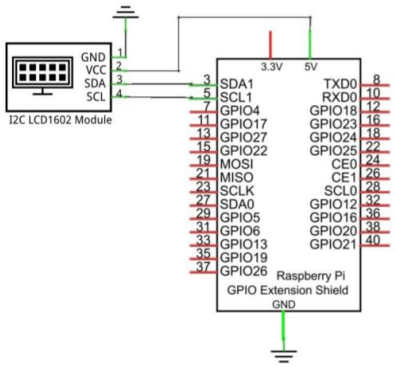
- Raspberry Pi
- Temperature and humidity sensor (DHT-11)
- PIR Sensor
- I2C LCD1602
- Relay
- 2 Breadboards
- Jump Wires (M-M, F-M)
- 40 Pin GPIO Cable
- GPIO extension Board
- Red LED
- 1N4001 Diode
- NPN transistor 8050
- 10kΩ and 220 kΩ resistors
-

2. Circuit

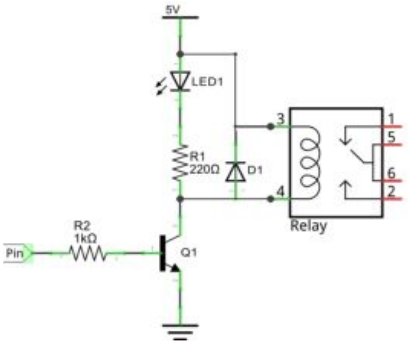


Pin connections:

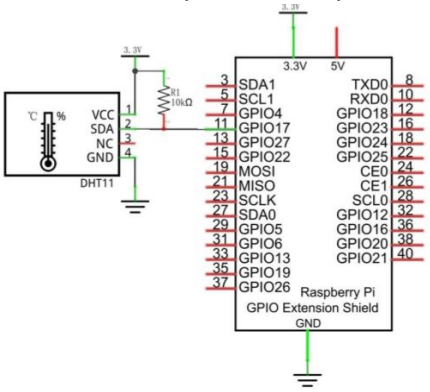
LCD



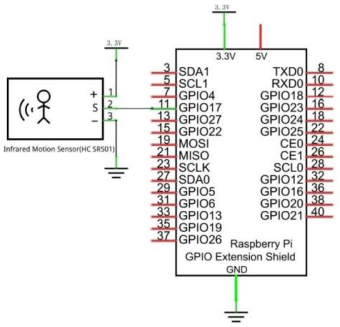
Relay (GPIO 13)



DHT (GPIO 17)



PIR (GPIO 25)



Modules and methods

CIMIS

cmisAPI.py

This module allows us to acquire Et0 (reference evapotranspiration), temperature, and humidity from the CIMIS database. We access to CIMIS website through this API.
We created a class called cimis_data to hold the information we retrieve from the database
This class has the following variables: Data, Hour, Humidity, Temperature and ET0.
We obtained an app_key through creating an account on the CIMIS website which allows us to interface with their API.

Methods:

- **get_cimis_data_for (current_hour)**
  This function returns a class cimis_data object that contains the information retrieved from the CIMIS database.
  -current_hour: The hour of the day for which to retrieve data. The hour ranges from 0 to 23. 0 refers to midnight on the previous day.

- **run_cimis(appKey, station, start, end):**
  This function is called to send a request to the CIMIS website. It generates the URL to get all information saved by CIMIS for the given day range and pass this URL and station number (75 for Irvine) to retrieve_cimis_data function. It then extracts the data portion of the response
  -appKey: the key needed to access the database
  -station: CIMIS station number (Irvine is 75)
  -start: The date to start collecting data
  -end: The date to end collecting data

- **retrieve_cimis_data(url, target):**
  This code gets the date today and yesterday and calls run_cimis to create the URL to access all information at Irvine station at the date. T
  -url: The URL to access information stored in the database
  -target: CIMS station number (75 for Irvine)

LCD display

lcdAPI.py Adafruit_LCD1602.py PCF8574.py

Adafruit_LCD1602.py and PCF8574.py contain API found on GitHub. Adafruit contains a class Adafruit_CharLCD and provides methods to initialize the LCD, clear LCD, display messages on the LCD, changing the cursor etc. PCF8574.py is used by Adafruit to interface with I2C. It contains a class PCF8574_I2C that Adafruit uses internally to implement the functions mentioned above. We wrote our own API that uses this library. The following are methods we created for the LCD.

The LCD displays

1. The current time and date

2. Local data whenever local data is acquired (every minute)

3. Average data for the past hour whenever it is calculated (every hour)

4. Cimis data whenever it is acquired (Humidity and temperature)

Methods:

- **lcd_setup():**
  This function creates the Adafruit_CharLCD object and sets it up by turn-on LCD backlight and set number of LCD rows and columns using Adafruit functions. We then start a thread lcd_t displays messages on the LCD.

- **lcd_thread():**
  This is the main thread that updates the LCD. It will display the current time and date every second on the LCD and monitor a variable called display-request. If the display request is not None, it will display the requested message instead of the time and wait for 5 seconds before continuing to display the current time. It will terminate when a variable terminate is set to True. It is the only function that modifies the display on the LCD. The messages are displayed on the LCD using the message function provided by Adafruit

- **display_cimis_data(hour,temperature, humidity):**
  This function is called whenever CIMIS data is obtained to display the data obtained from CIMIS on the LCD. This function creates the message containing the hour, temperature and humidity passed in the parameters, then calls display_message passing in the created string.
  -hour: the starting hour
  -temperature: temperature from CIMIS
  -humidity: humidity from CIMIS

- **display_average_data(hour,temperature, humidity):**
  This function is called whenever average local data is calculated (every hour) to display the average local temperature and humidity on the LCD. This function will create the message containing the hour, average temperature and average humidity passed from the parameter, then calls display_message passing in the created string.
  -hour: the starting hour
  -temperature: average local temperature for one hour
  -humidity: average local humidity for one hour

- **display_local_data(counter,temperature, humidity):**
  This function is called whenever DHT data is obtained from temperature and humidity sensor (every minute). This function will create the message containing the counter (minute number), temperature and humidity passed from the parameter, then calls display_message passing in the created string.
  -counter: from 1 to 60 in order to display the values for each minute
  -temperature: local temperature read from DHT

-humidity: local humidity read from DHT

- **display_message(message):**
  This function is called to display a message on the LCD. If the variable display_request is not None, it will wait for it to be set to None then place the message in the display_request. The request will later be picked up by the lcd_thread who will display it
  -message: message to be displayed on LCD

- **lcd_cleanup():**
  This function sets terminates the LCD thread, turns off the LCD backlight and clears the LCD.

DHT sensor

Freenove_DHT.py

The file Freenove_DHT contains API found on GitHub that allows us to interface with the DHT sensor. This file defines a DHT class object and we interface with it using the constructor and the readDHT11 method which updates the temperature/ humidity variables of the class that we can then read.
Since the temperature and humidity may be off, we reattempt to get the data until we have valid values from the DHT.
We defined two methods in main.py get_local_teperature() and get_local_humidity() that reattempt to acquire the local data and humidity data by calling readDHT11 until valid data is acquired.

Important Methods:

readDHT11: (provided)

Requests data from the sensor reads the values and updates the temperature, humidity of the class. This method uses another method readSensor(). It returns DHTLIB_OK if the data is valid and other values if the data was not.

Main module

main.py

In our program, we first set up all the components such as relay pins, LCD and the DHT sensor. Then we start a thread called data_acquisition thread. This thread interfaces with the DHT sensor and obtains local data every minute. After an hour elapses, it calculates the average temperature and humidity and places it in a specific position in an array.
We use two arrays of size 24 to hold this local data. The data acquisition thread will place the average data for hour x at position x in the array. When the main thread is ready to read data for hour x, he will check that position in the array. If it is not yet ready, he will wait until it is. Once local data is consumed, it is set to None. The threads operate in a round robin fashion.
The main thread starts a loop that attempts to acquire CIMIS data every hour by using our API. If the data is not obtained it will wait until it is available. Once data is available for hour x, it will read the local data for that hour from the arrays and use it to calculate the modified ET0 and the time to irrigate.
It then starts the relay and checks for motion through the PIR sensor while irrigation is in progress. Once irrigation for hour x is finished, it will either move on to the next hour immediately if it has been delayed, or wait until it is time to process the data for the next hour.
We define a variable hours_to_run that indicates how many hours the threads need to irrigate for. The actual program may run longer than this since it may be delayed if CIMIS data is not available for some of the hours. For our 24 hours test, the variable is set to 24.

Methods:

- **time_now():**
  This function returns the current time based on the format (hour:minutes: seconds)

- **console_msg1(message):**
  This function prints the current time and a message from the data acquisition thread on the console
  -message: The message to print on the console

- **console_msg2(message):**
  This function prints the current time and message from the main thread with the format
  -message: The message to print on the console

- **setup():**
  This function sets up the GPIO pins for Relay, and PIR, the DHT, LCD, and other variables.

- **cleanup():**
  This function is called in main to clean up all GPIO pins on the board. It will call lcd_cleanup() and GPIO.cleanup()

- **get_local_temperature():**
  This function is called to read the local temperature from temperature and humidity sensor and returns the local temperature value. The variable chk will be assigned with the return value from readDHT11() function as long as the return value is valid.

- **get_local_humidity():**
  This function is called to read the local humidity from temperature and humidity sensor and returns the local sensor value. The variable chk will be assigned with the return value from readDHT11() function as long as the return value is valid.

- **data_acquisition_thread():**
  This function gets the local data every hour and stores it in the arrays.

- **mainloop():**
  This main loop function acquires CIMIS data every hour and waits for local data every 60 seconds. Then, using the acquired data, it computes time to irrigate and turns on the relay.

- **get_time_to_irrigate(data, local_temperature, local_humidity):**
  This function calculates the time to irrigate in seconds based on the CIMIS data and local average temperature and humidity.

$$Modified\ ET0\ =\ Cimis\ ET0\ \times \frac{Cimis\ Humidity}{Local\ Humidity}\ \times \frac{Local\ Temperature}{Cimis\ Tempearture}$$
$$Gallons\ needed\ per\ hour\ =\ \frac{Modified\ ET0\ \times PF\ \times SF\ \times 0.62}{IE \times 24}$$
$$Time\ needed\ =\ \frac{Gallons\ needed\ per\ hour}{WD}$$
$$PF\ :\ Plant\ Factor$$
$$SF\ :\ Area\ to\ be\ irrigated\ in\ square\ feet$$
$$IE\ :\ Irrigation\ efficiency$$

  -data: the cimis_data class that stores CIMIS data information
  -local_temperature: the calculated local temperature
  -local_humidity: the calculated local humidity

- **Main:**
  This function calls setup() function to do setup GPIO library. Then it creates and starts data acquisition thread. When the KeyboardInterrupt happens, it stops the program and calls cleanup() to clean up all GPIO pins on the board.

Contribution

Mehdi Lakhoua: Relay Driver, Main, Combine all components both in software & hardware
Rijul Arora: PIR Sensor, LCD
Ye Myat Kyaw: Temperature and humidity sensor, LCD
Yuki Hayashi: CIMIS Data Retrieval

Conclusion

With the completion of this project, we were able to create an efficient soil irrigator that will allow a large amount of water to be saved. This was accomplished by combining various hardware peripherals, such as the PIR motion sensor, the LCD, etc. We used temperature and humidity values from the local area and the CIMIS station in Irvine to be able to calculate how long we should irrigate our soil.