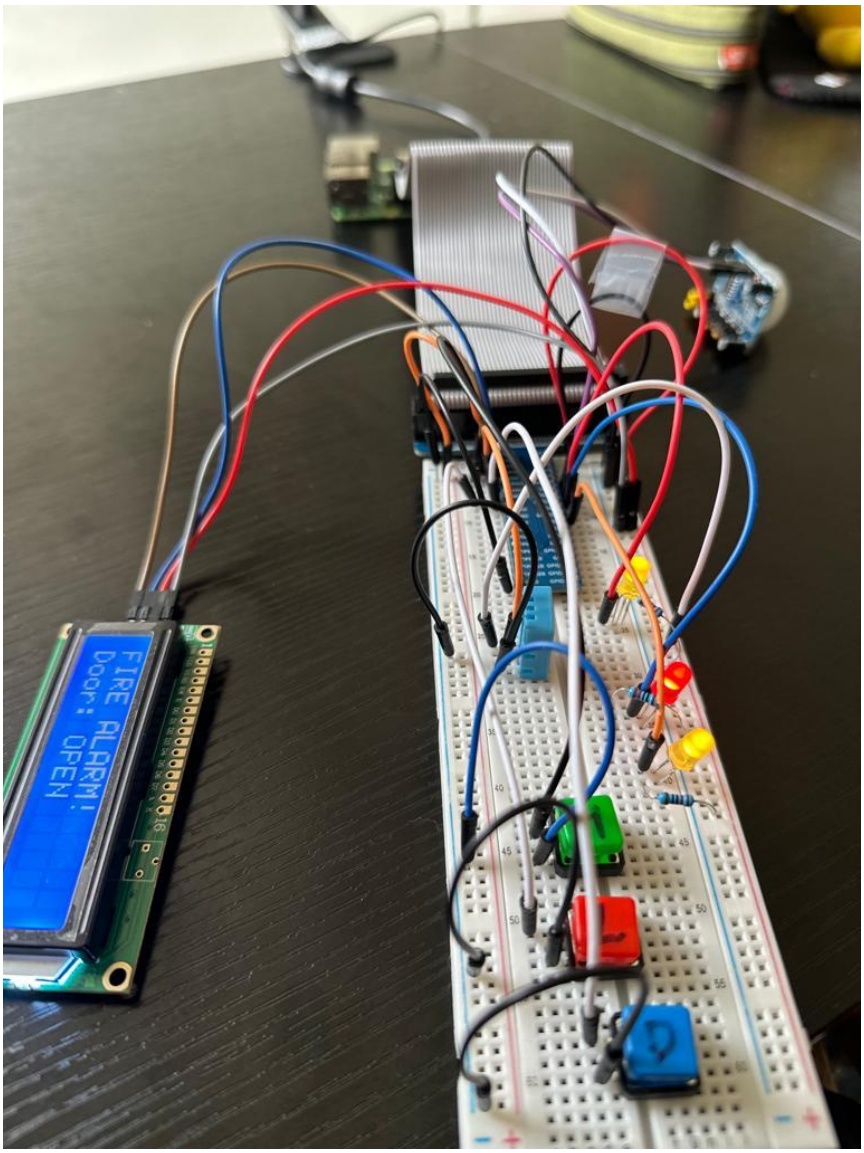
EECS-113: Final Project

Building Management System



Vishwanath Singh

1097396

EECS 113 Spring 2023

Final Project Video Link

INTRODUCTION

The project is a Building Management System implementation using a Raspberry Pi 3B+ and sensors like PIR Infrared Motion Detector with LED Indicator, the Infrared Motion Sensor can detect the heat signatures emitted by living humans within the infrared spectrum, the DHT -11, to read Temperature and Humidity data. It also utilizes the CIMIS API to retrieve weather data, specifically humidity, for more accurate monitoring. The goal of the project is to monitor and control the environment within a building to optimize energy efficiency, comfort, and safety.

Control of the HVAC (Heating, Ventilation, and Air Conditioning) system is the focus of the BMS. Based on the target temperature and the current weather, it modifies the HVAC system's functioning. The CIMIS API offers extra meteorological data for enhanced climate control, and the system uses temperature and humidity sensors to monitor the indoor environment. By calculating the energy consumption and related costs of the HVAC system, the BMS combines energy consumption monitoring. Users can use buttons to change the desired temperature, and an LCD screen shows real-time feedback.

Additionally, the system manages door/window sensor events. It recognizes changes in the open or closed condition of doors and windows, which may have an impact on how well the HVAC system functions. To reduce energy waste, the BMS cuts off the HVAC system whenever the sensors detect an open door or window. A fire alarm feature is also included in the BMS. The BMS activates a fire alarm if the weather indicates a potential fire threat (for example, a high weather index). It activates the door/window status to "OPEN" and issues warnings visually and audibly via the LCD screen and LEDs.

The project prioritizes user comfort and safety while also maximizing energy economy by modifying the HVAC system in response to current conditions. The BMS contributes to the maintenance of a comfortable and secure indoor environment while minimizing energy loss by integrating a variety of sensors and clever control algorithms.

IMPORTANT

- The max temperature value was changed to test the fire system, since did not have a safe heat source to test it with.
- Green and blue LED were damaged since it was a old kit, 2 yellow LED were used instead.

MATERIALS

1. Raspberry Pi 3 B+
2. DHT-11 Temperature and Humidity Sensor
3. Breadboard
4. PIR Sensor
5. I2C LCD 1602
6. Jump Wires
7. 10k**Ω** and 220k**Ω** Resistor
8. GPIO Extension Board and Cable
9. 3 – LED 2 yellow and 1 red (blue and green LED were not available)
10. 3 Pushbuttons

CONNECTIONS

We are using a Raspberry pi 3B+ which has one GB RAM, a ribbon cable and a breadboard. We have 5 main parts to the setup, the first being the DHT 11 module that helps us capture the local temperature and humidity, which is connected to GPIO pin 4. We also have a PIR sensor to detect the motion off any person in the vicinity it is connected to the GPIO pin 18. Next, we have the LCD module which is connected to SDA one and s SCL one. The next part is the LEDs, we have 3 leads. The first led is green colour and it is connected to GPIO pin 23 it turns on when the pir sensor detects any presence nearby. The second led is the red led connected to GPIO pin 24 this is turned on when the HVAC system goes into heating mode. The 3rd led is the blue led connected to GPIO pin 25 this is turned down when the HVAC system goes into AC mode. The last part of the setup is the 3 push buttons. The first push button is connected to GPIO pin 17 it is called the button up pin, it basically increases the desired temperature value by one. The second push button is deep button-down pin connected to the GPIO pin 27, 8 reduces the value of desired temperature by one. The last push button is the button sensor pin connected to GPIO pin 22, this push button controls the security mechanism which is opening and closing of the door and the window of the building.

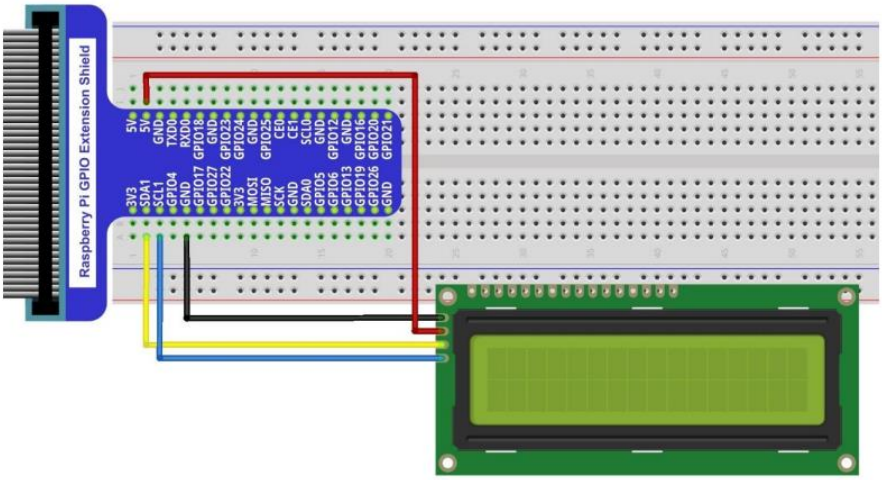


Figure 1: I2C LCD [1]

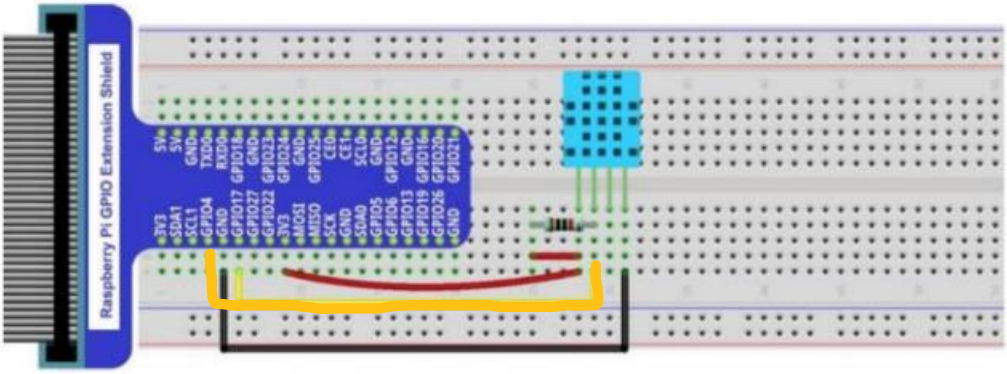


Figure 2: DHT-11 with 10kΩ resistor. [GPIO 4] [1]

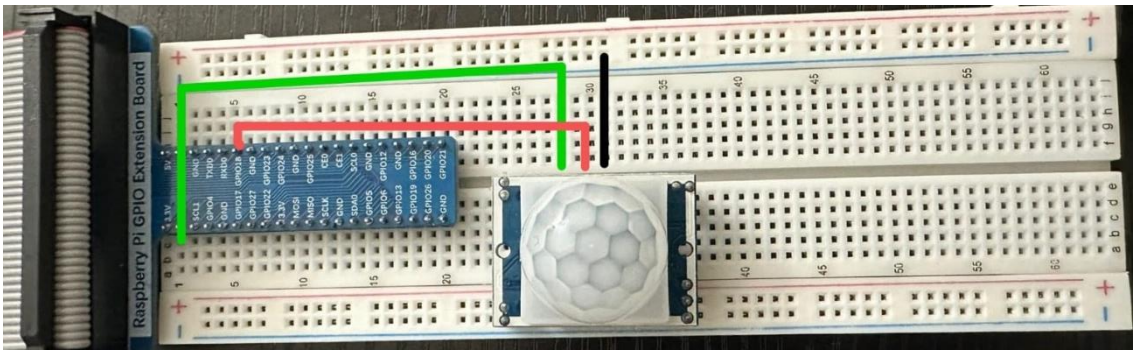


Figure 3: PIR Sensor [GPIO 18]

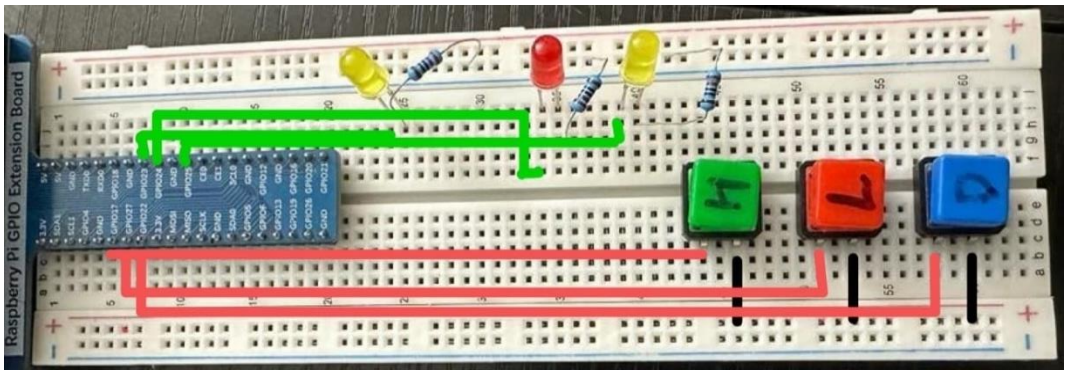


Figure 4: Green LED [GPIO 23], Red LED [GPIO 24], Blue LED [GPIO 25]

Green (button up) [GPIO 17], Red (button down) [GPIO 27], Blue (button sensor) [GPIO 22]

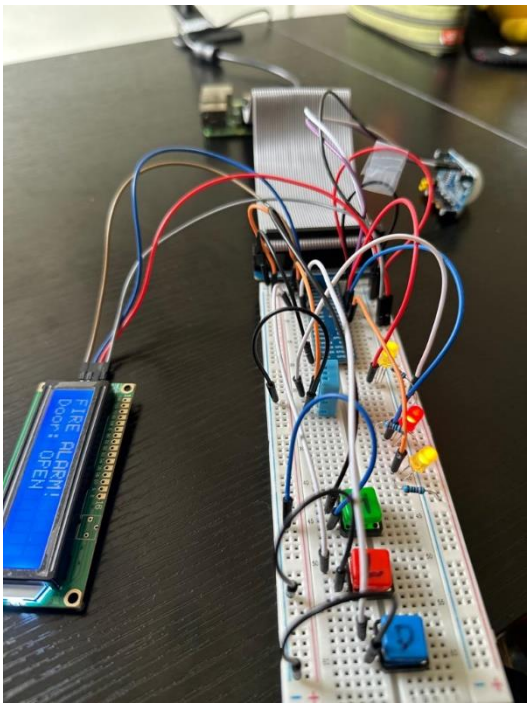


Figure 5: Breadboard configuration

SOURCE CODE

These are the main functions in the code that perform the 5 operations required according to the specification. We use libraries for the sensors and LCD along with time, threading and the main Rpi.GPIO. The Adafruit_DHT library enables interaction with DHT (Digital Humidity and Temperature) sensors. It provides functions for reading temperature and humidity values from DHT sensors connected to the Raspberry Pi. RPLCD library provides classes for interfacing with character LCD displays. The i2c module from RPLCD is specifically used for I2C communication with the LCD display. It allows displaying information on the LCD screen connected to the Raspberry Pi.

GPIO.setup lines configure the GPIO pins for specific purposes. The GPIO pins specified in these lines will be used for various functionalities such as PIR sensor, LEDs, buttons, and door/window sensor.

1. Ambient Light Control:

```
def bms_control():
    global desired_temperature, is_motion_detected, is_fire_alarm_active, energy_consumption, energy_cost, max_temp_for_fire
    global is_button_up_pressed, is_button_down_pressed, is_button_sensor_pressed  # Add global declaration
    while True:
        if GPIO.input(PIR_PIN) == GPIO.HIGH:
            is_motion_detected = True
            turn_on_green_led()
            lcd.clear()
            lcd.write_string("LED:ON")
            time.sleep(10)
        else:
            is_motion_detected = False
            turn_off_green_led()
            lcd.clear()
            lcd.write_string("LED:OFF")
            time.sleep(5)
```

The depicted function plays a crucial role in the system as it evaluates specific conditions before executing the desired action. The condition being assessed is GPIO.input(PIR_PIN) == GPIO.HIGH, which examines whether the PIR sensor has detected motion. When motion is detected, the program enters a loop to activate the green LED. Additionally, an indication of LED activation is displayed on the LCD screen for a duration of 10 seconds.

Conversely, if the PIR sensor does not detect any motion, the green LED remains inactive, resulting in the LCD displaying a message indicating LED deactivation for a period of 5 seconds. This functionality ensures that the system responds appropriately based on the presence or absence of detected motion, enhancing the overall monitoring capabilities and providing visual feedback through the LED and LCD components.

In figure 6, we can see that the LED ON message is displayed once the motion is detected.

Figure 6: LED ON Displaying

2. Room Temperature (HVAC):

```python
# Function for reading temperature and humidity from DHT sensor
def read_temperature_humidity():
    global temperature, humidity
    humidity, temperature = Adafruit_DHT.read_retry(Adafruit_DHT.DHT11, 4)
    if humidity is not None and temperature is not None:
        temperature = round(temperature, 1)
        humidity = round(humidity, 1)

# Function for retrieving humidity from CIMIS system
def get_cimis_humidity():
    params = {
        'appKey': '17ab4aac-00b7-4ffc-a2d8-06086ca5780e',
        'targets': '71',
        'startDate':'2023-05-01',
        'endDate': '2023-05-29',
        'dataItems': 'humidity'
    }
    response = requests.get(CIMIS_API_URL, params=params)
    if response.status_code == 200:
        data = json.loads(response.text)
        if 'Data' in data and len(data['Data']) > 0:
            humidity = round(data['Data'][0]['humidity'], 1)
            return humidity
    return None

# Function for calculating weather index
def calculate_weather_index():
    global weather_index
    weather_index = temperature + 0.05 * humidity
    weather_index = round(weather_index, 1)
```

Using a DHT sensor attached to a Raspberry Pi, the function read_temperature_humidity() reads the temperature and humidity readings. The Adafruit_DHT library is used to read the sensor's values. The global variables temperature and humidity include the values of the temperature and humidity. Using the round() function, the temperature result is rounded to one decimal point.

Using an API, the function get_cimis_humidity() receives the humidity reading from a CIMIS (California Irrigation Management Information System) system. It makes a GET request with the necessary arguments to the supplied CIMIS API URL. The response's humidity value is taken from it and rounded to one decimal place if the request is successful and the response contains correct data. Then the humidity value is given back. None is returned if the request encounters any errors, or the answer doesn't have any accurate data.

Using the data for temperature and humidity, the function calculate_weather_index() determines the weather index. The weather index value is calculated using the global variables temperature and humidity using the formula **weather_index = temperature + 0.05 * humidity**. The determined value is saved in the global variable weather_index after being rounded to the nearest decimal point.

The values calculated in these functions are displayed on to the LCD as shown in figure 7, once the display_information() function is called.

Figure 7: Displaying Temperature, Humidity, Weather index.

```
# Function for updating HVAC status
def update_hvac_status():
    global hvac_status, old_status
    old_status = hvac_status
    #print(old_status)
    if weather_index > desired_temperature + 3:
        hvac_status = "AC"
        turn_on_blue_led()
        turn_off_red_led()
    elif weather_index < desired_temperature - 3:
        hvac_status = "HEAT"
        turn_off_blue_led()
        turn_on_red_led()
    else:
        hvac_status = "OFF"
        turn_off_blue_led()
        turn_off_red_led()
```

The update_hvac_status() function modifies the HVAC status according to the desired temperature and the weather index. The function calculates the HVAC status by comparing the weather index with the intended temperature:

The HVAC status is changed to "AC" (air conditioning) if the weather index is higher than the intended temperature plus 3. The red LED is off and the blue LED is now on.

The HVAC status is set to "HEAT" (heating) if the weather index falls below the intended temperature minus 3 points. The red LED is now on while the blue LED is off.

The figure 8 shows the HCAC status is off and figure 9 shows the change in the HVAC status when the desired temperature was increased above the weather index+3 limit.

Figure 8: Desired temperature, HVAC status, Door and Window status

Figure 9: HVAC status changed.

The HVAC status is set to "OFF" if the prerequisites are not satisfied. The red and blue LEDs are both off.

```
# Function for displaying HVAC state on LCD
def display_hvac_state():
    lcd.clear()
    lcd.write_string(f"{hvac_status} is on")
    time.sleep(3)
    display_information()
```

This function displays the HVAC status only when it is in AC or HEAT mode, indicating that either the AC or HEAT is on.

3. Fire Alarm System

In the BMS_Control function, we check if the weather index is higher than the max temperature, which is 90F or 35C. *Due to lack of time in video and no safe heat source, I changed the value of maximum fire temperature to 20C**,* since weather index was at a value of 26C. Due to which we were able to see how the fire alarm system worked in the video. Figure 10 shows how the message was displayed on to the screen.

Figure 10: Fire Alarm and Evacuate message.

Once the weather index is greater than the max fire temperature, the is_fire_alarm_active is set to True and the check fire alarm() function is called.

If the fire alarm is on (is_fire_alarm_active is True), the function verifies that, the maximum temperature threshold for fire is set to 35 degrees Celsius when the fire alarm is turned on. *(This is only so that the code fire alarm does not keep running while the video is being recorded)*. The temperature at which the fire alarm is triggered is determined by this variable.

Two lines of text, "FIRE ALARM!" and "EVACUATE!" are written on the LCD panel once it has been cleared. Additionally, the console is printed with these messages. Using the time, the software pauses execution for 3 seconds. Use the sleep() function to give the messages enough time to load. The status of the doors and windows is "OPEN". The updated statuses for the doors and windows are printed to the console and written to the LCD panel. The HVAC is set to "OFF" mode.

The LCD displays the HVAC status. Using a loop and the turn_on_green_led(), turn_on_red_led(), turn_on_blue_led(), turn_off_green_led(), turn_off_red_led(), and turn_off_blue_led() functions, the green, red, and blue LEDs blink for 10 seconds. This blinking effect gives forth a visual fire alarm signal. The green, red, and blue LEDs are switched off after the loop is over. Since the fire alarm condition has been addressed, the is_fire_alarm_active variable is set to False. If there is no fire alarm (the else block), to maintain consistency, the is_fire_alarm_active variable is once more set to False. The green LED has been disabled.

The required information is displayed on the LCD panel by using the display_information() method, which is called.

```python
# Function for checking fire alarm condition
def check_fire_alarm():
    global is_fire_alarm_active, weather_index, door_status, window_status, max_temp_for_fire,hvac_status
    if is_fire_alarm_active:
        max_temp_for_fire =35
        lcd.clear()
        lcd.cursor_pos = (0, 0)
        lcd.write_string("FIRE ALARM!")
        print("FIRE ALARM!")
        lcd.cursor_pos = (1, 0)
        lcd.write_string("EVACUATE!")
        print('EVACUATE')
        time.sleep(3)
        lcd.clear()
        door_status = "OPEN"
        window_status = "OPEN"
        lcd.cursor_pos = (0, 0)
        lcd.write_string(f"DoorWin:{door_status},{window_status}")
        print(f"DoorWin:{door_status},{window_status}")
        hvac_status = "OFF"
        lcd.cursor_pos = (1, 0)
        lcd.write_string(f"HVAC:{hvac_status}")
        for _ in range(10):  # Blink for 10 seconds
            turn_on_green_led()
            turn_on_red_led()
            turn_on_blue_led()
            time.sleep(1)
            turn_off_green_led()
            turn_off_red_led()
            turn_off_blue_led()
            time.sleep(1)
        is_fire_alarm_active = False
        turn_off_green_led()
        turn_off_red_led()
        turn_off_blue_led()
    else:
        is_fire_alarm_active = False
        turn_off_green_led()
        display_information()
```

4. Energy Bill Generator

```python
# Function for updating energy consumption and cost
def update_energy_consumption():
    global energy_consumption, energy_cost
    if hvac_status == "AC":
        energy_consumption += 0.001 * 18000  # Assuming 1 second interval
    elif hvac_status == "HEAT":
        energy_consumption += 0.001 * 36000  # Assuming 1 second interval
    energy_cost = round(energy_consumption * 0.5, 2)
```

The code checks the current value of the HVAC status variable. If the HVAC status is "AC", it means that the air conditioning is active. In this case, the energy consumption is increased by 0.001 kilowatt-hours (kWh) every second, assuming a 1-second interval. The value 18000 represents the number of

Watts used to calculate the energy for cooling. If the HVAC status is "HEAT", it means that the heating system is active. In this case, the energy consumption is increased by 0.001 kWh every second, assuming a 1-second interval. The value 36000 represents the number of watts, which is used to calculate the energy consumption for heating.

After updating the energy_consumption variable, the energy cost is calculated by multiplying the energy_consumption value by 0.5. The value 0.5 represents the cost per kilowatt-hour (kWh) of energy consumed. The result is rounded to 2 decimal places and stored in the energy_cost variable.

```
# Function for displaying energy bill report on LCD
def display_energy_bill_report():
    lcd.clear()
    lcd.cursor_pos = (0, 0)
    lcd.write_string(f"Energy: {energy_consumption} KWh")
    lcd.cursor_pos = (1, 0)
    lcd.write_string(f"Cost: ${energy_cost}")
    time.sleep(3)
    display_information()
```

An energy bill report is shown on an LCD screen by the display_energy_bill_report() function. The screen is first cleared, after which the cursor is placed in the top-left corner. Kilowatt-hours (KWh) are displayed on the screen by the function as the energy consumption value taken from the global variable energy_consumption.

The energy cost, which was derived from the energy_cost global variable, is then displayed on the screen in dollars ($) with the cursor position set to the second row. The function displays the data and then waits for three seconds so the user can study the report. In order to replace the energy bill report with the primary information display, which includes temperature, humidity, HVAC status, and door/window status, the function calls display_information() at the end. This feature gives users a method to keep an eye on their energy expenditures and usage on the LCD panel.

The figure 11 shows the energy expenditure and the cost of for the AC and HEATER.

Figure 11: Energy expenditure and cost in dollars

5. Security System

```
# Function for handling door/window sensor events
def door_window_sensor_event_handler(channel):
    global door_status, window_status, weather_index, is_fire_alarm_active, hvac_status
    if GPIO.input(BUTTON_SENSOR_PIN) == GPIO.HIGH:
        if door_status == 'OPEN' and window_status == 'OPEN':
            door_status = "CLOSED"
            window_status = "CLOSED"
        else:
            door_status = "OPEN"
            window_status = "OPEN"
            hvac_status = "OFF"
            display_information()
            turn_off_blue_led()
            turn_off_red_led()
```

The security system function is called door window sensor event handler. Using the GPIO.input() method, the code determines the status of the button sensor pin (BUTTON_SENSOR_PIN). The code performs the following operations if the input is GPIO.HIGH, meaning that the sensor has been activated or the button has been pressed.

The function examines the door and window's current status inside the conditional block. When the window and door are both open (door_status == 'OPEN' and window_status == 'OPEN'), the code changes the values of the door_status and window_status variables to "CLOSED," indicating that the window and door have been closed.

The code updates the door_status and window_status variables to "OPEN" if the door or window is not in the open state, which can be seen in figure 12. Otherwise, it assumes that they are closed. The hvac_status variable is also set to "OFF," suggesting that the HVAC system ought to be turned off. In order to update the LCD panel with the new door and window status as well as other information, the code then calls display_information(). By invoking the turn_off_blue_led() and turn_off_red_led() methods, it also turns off the blue and red LEDs.

Figure 12: Door and Window status changed to open.

CONCLUSION

This report contains all the project deliverables according to specification provided. The implementation of the ambient light system helps us navigate when a motion, human or animal is detected. The fire alarm systems, one of the more important systems in a building management system, is implemented using the DHT-11 sensor and using the CIMIS data that we had retrieved. We were able to change the status of the HVAC based on the weather index and the desired temperature required by the user. The implementation also included the synchronization of multiple systems, such as when fire is detected the HVAC is turned off along with the doors and windows being open, which is a key aspect of the cyber physical systems, BMS being an example of that. The implementations also included the expenditure and cost report that was effectively displayed on the screen and gave us an idea as to how much money and Kwh were spent on one AC or HEAT cycle. The project was completed as per the requirement.

**: As suggested by the teaching assistant.

References:

[1] F. U. S. K. f. R. Pi, "Freenova.com," [Online]. Available: https://freenove.com/tutorial.html. [Accessed 2023].

[2] J. Tang, "Atmosphere Monitoring System," 2020.

[3] F. Kurdahi, "Lecture slides," EECS 113, Irvine, 2023.