EECS 113 Final Project Report

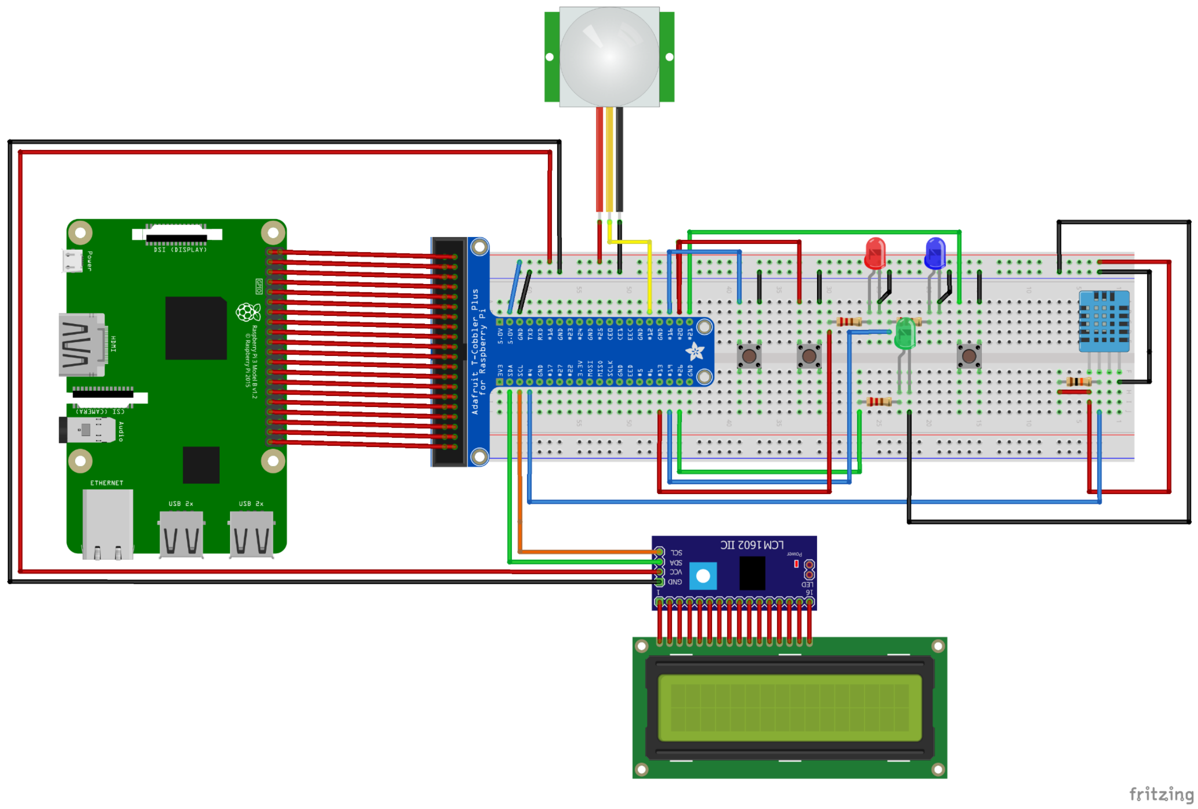
Jose Juan Jr. Salgado
Student ID#: 64725215

Raspberry Pi Building Management System (BMS)

DHT-11 Sensor
PIR Sensor
LCD
Buttons
LEDs

June 9th, 2021

1. Schematic



This schematic is very close to the exact connections I used on my actual breadboard.

2. Steps

Before starting the project, I first wired and tested each component and sensor that I was going to use in this project. To test, I wrote small snippets of code that grew into the final BMS.py file.

2.1. Ambient light control

2.1.1. Design

Ambient light control is done inside my code's pir_function(). It is continuously run with a while loop on a thread when the BMS file is run. Taking the PIR sensor's input, if the input is 1, the green LED is lit up and the lights flag is set to True so that the LCD displays "L:ON". If the PIR sensor input is 0, the code goes into a while loop that starts a clock. The code in this second while loop sleeps for 1 second (to simulate a timer) and reads the sensor input. If the input is 0 for 10 seconds, the green LED is shut off and the lights flag is set to False so that the LCD displays "L:OFF". If the input is 1, we leave the clock loop and the green LED is turned on again.

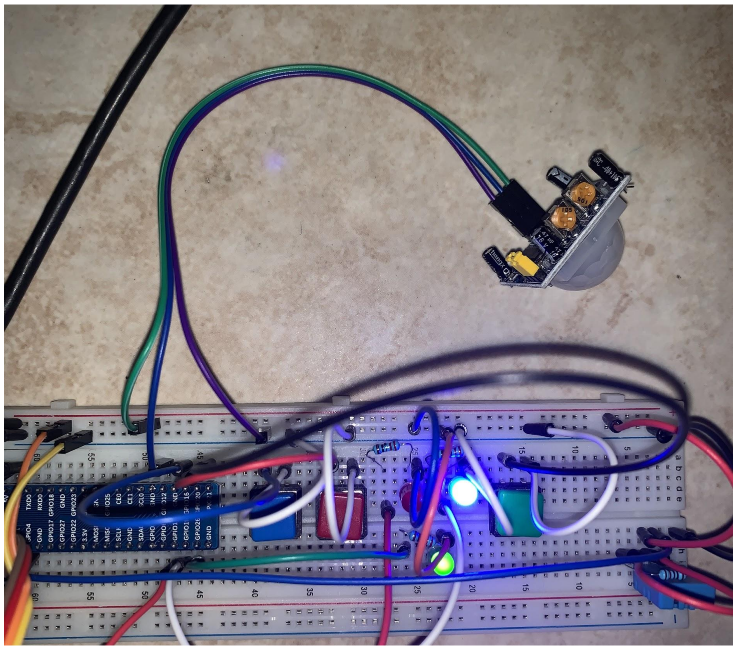
2.1.2. Code

```
def pir_function():
    global pirPin, green_led, lights
    while True:
        motion = GPIO.input(pirPin)
        if motion == 1:
            lights = True
            GPIO.output(green_led, 1)
        if motion == 0:
            clk = 0
            while motion == 0:
                time.sleep(1)
                clk += 1
                motion = GPIO.input(pirPin)
                if clk == 10:
                    print("No motion for 10 sec, LED off")
                    lights = False
                    GPIO.output(green_led, 0)
```

2.1.3. Notes

I had to turn down the delay on the PIR sensor. This helped because my sensor was reading 1 for too long and the clock was taking forever to start. The thread for the pir_function() is started in the in_loop() function when the BMS is started. The LEDs are wired the same as in assignment 4, with the 220 ohm resistor.

2.1.4. Pictures



2.2. ***Room Temperature - HVAC***

2.2.1. *Design*

The HVAC works by using many different functions at the same time. These functions are the following: dht_update(), hvac_function(), check_diff(), calculate_weather_index(), and dht_function(). Two of these functions (dht_update() and dht_function()) run continuously on threads.

The HVAC obtains room temperature from the DHT-11 sensor, and takes two buttons as inputs to set a desired temperature. On boot, the desired temperature is set at 75. The LCD displays the weather index and desired temperature together separated by a “/”, and shows status of the HVAC (“H”) as AC, HEAT, or OFF. The weather index is calculated using data from CIMIS and the average temperature given by the DHT-11 sensor.

When the BMS starts, the dht_function() is continuously run on a thread. Inside this function the dht_update() function is also run on a thread and grabs temperature in celsius from the DHT-11 sensor. This value is converted to Fahrenheit and sent to a list (temperature_list) that keeps that last 3 seconds of input. The temperature_list starts as empty, but eventually grows to 3 items. If the temperature_list has 3 items, the first item in the list is removed so that the most recent temperature input can be added.

The dht_function() then continues to calculate the average temperature by computing the sum of temperature_list and dividing it by 3. Before the weather index is calculated, our CIMIS data is updated to have the most recent values. After the most recent values are pulled, the weather index is calculated by taking the average temperature, adding 0.05, and multiplying by the CIMIS humidity value.

After this value is computed, the dht_function() calls hvac_function(). This function first checks if a window is open, and if it is, it shuts down the LEDs and sets the flags to their appropriate values, and shows the appropriate message on the LCD (LCD event happens in the door/window function). If a window is not open, it checks the result of weather index minus desired temperature. If this value is more than 3, the AC is turned on, along with the blue LED, and if it is less than -3, the heater is turned on, along with the red LED. Everytime an HVAC event is triggered, the LCD shows the appropriate message and the main LCD screen is halted with a pop_up flag.

When a user adjusts the temperature with the buttons, the hysteresis like function is done again in case the HVAC needs to switch modes. The blue button brings the desired temperature down, and the red button brings it up. A min and max of 65 and 85 are set for the desired temperature.

2.2.2. *Code*

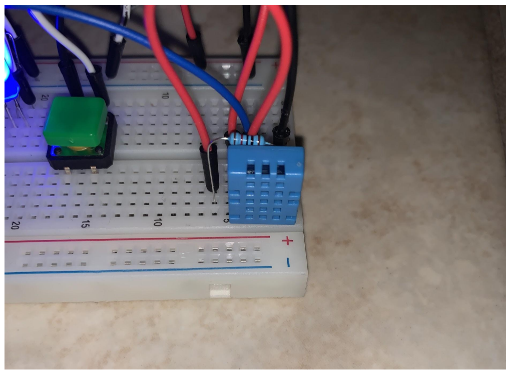
```
def dht_update():
    while True:
        global humidity, temperature, temperature_list
        humidity, temperature = Adafruit_DHT.read_retry(dhtType, dhtPin)
        temperature = (temperature * 1.8) + 32
        if len(temperature_list) == 3:
            temperature_list.pop(0)
        if len(temperature_list) < 3:
            temperature_list.append(temperature)
        print("Local Hum.:", humidity)
        print("Local Temp.:", temperature)
        time.sleep(1)

def hvac_function():
    global door_window_open, hvac_on, ac_on, heat_on, pop_up, ac_boot, heat_boot
    # check if door/window open
    if door_window_open:
        hvac_on = False
        ac_on = False
        heat_on = False
        ac_boot = True
        heat_boot = True
        GPIO.output(blue_led, 0)
        GPIO.output(red_led, 0)
    else:
        hvac_on = True
        if (weather_index - desired_temp) > 3:
            ac_on = True
            # turn on blue LED
            GPIO.output(red_led, 0)
            GPIO.output(blue_led, 1)
            if ac_boot:
                lcd_ac_on()
                ac_boot = False
        elif (weather_index - desired_temp) < -3:
            heat_on = True
            # turn on red LED
            GPIO.output(blue_led, 0)
            GPIO.output(red_led, 1)
            if heat_boot:
                lcd_heat_on()
                heat_boot = False

def check_diff():
    global weather_index, desired_temp, ac_on, heat_on
    if (weather_index - desired_temp) > 3 and ac_on == False:
        lcd_ac_on()
    if (weather_index - desired_temp) < -3 and heat_on == False:
        lcd_heat_on()
```

```
# calculation functions

def calculate_weather_index():  # calculating data from calling function
    global cimis, average_temperature
    cimis.update_values()
    return average_temperature + 0.05 * cimis.humidity

def blue_press(button_press):
    global desired_temp, weather_index
    if desired_temp == 65:
        print("Min AC reached")
    if desired_temp > 65:
        desired_temp -= 1
    check_diff()
    print("Set temp. to:", desired_temp)

def red_press(button_press):
    global desired_temp
    if desired_temp == 85:
        print("Max HEAT reached")
    if desired_temp < 85:
        desired_temp += 1
    check_diff()
    print("Set temp. to:", desired_temp)
```

2.2.3. *Notes*

The libraries used for this section of the project were Adafruit_DHT and CIMIS_Extract. The thread for the dht_function() is started in the in_loop() function when the BMS is started. The buttons are wired and handled the same as in assignment 4. The button_interrupt() function is called after the threads are set.

2.2.4. *Pictures*



2.3. *Security System*

2.3.1. *Design*

The security system is simply a button event function. It displays on the LCD as D:SAFE or D:OPEN. If pressed, the open flag is set to True and the screen is prompted, along with the pop_up flag so it can display for 3 seconds. The HVAC is stopped as handled in hvac_function() and the LCD displays. If pressed again, the open flag is False and the opposite occurs. HVAC resumes.

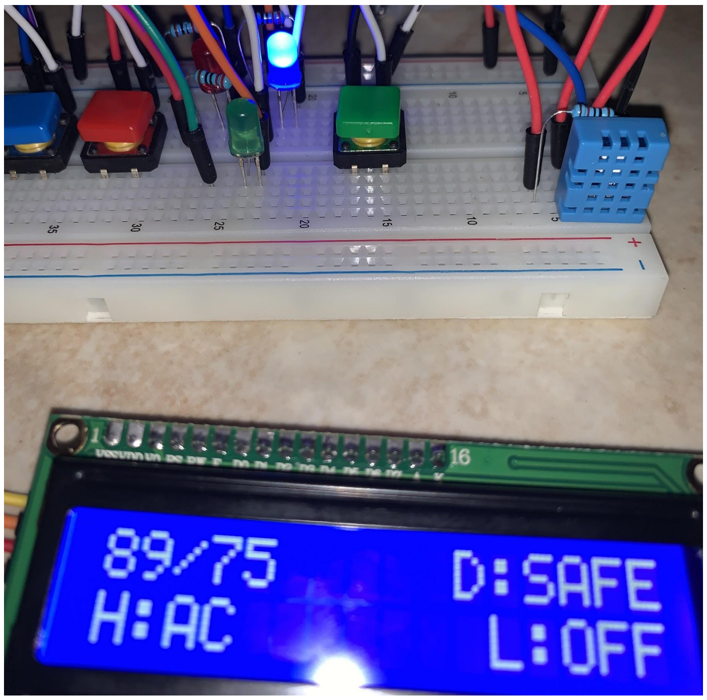
2.3.2. *Code*

```
def green_press(button_press):
    global door_window_open, green_button
    if door_window_open:
        print("Door/window closed")
        door_window_open = False
        lcd_door_window_closed()
    elif not door_window_open:
        print("Door/window opened")
        door_window_open = True
        lcd_door_window_open()
```

2.3.3. *Notes*

The button is wired and handled the same as in assignment 4. The button_interrupt() function is called after the threads are set.

2.3.4. *Pictures*



3. LCD

3.1. Design

My LCD uses the Adafruit_LCD1602 & PCF8574 libraries to function. All I have to do is simply position my cursor and write a message.

3.2. Code

```
def lcd_setup():
    global Line1
    mcp.output(3, 1)
    lcd.begin(16, 2)
    Line1 = "EECS113 \nFinal Project"
    time.sleep(3)
    update_lcd()

def update_lcd():
    global Line1
    lcd.clear()
    message = str(Line1)
    lcd.message(message)
    time.sleep(3)

def lcd_door_window_open():
    global Line1, pop_up
    Line1 = "DOOR/WINDOW OPEN!\nHVAC HALTED"  # update line 1
    pop_up = True
    update_lcd()
    pop_up = False

def lcd_door_window_closed():
    global Line1, pop_up
    Line1 = "DOOR/WINDOW CLOSED!\nHVAC RESUMES"  # update line 1
    pop_up = True
    update_lcd()
    pop_up = False

def lcd_ac_on():
    global Line1, pop_up
    Line1 = "HVAC AC"  # update line 1
    pop_up = True
    update_lcd()
    pop_up = False

def lcd_heat_on():
    global Line1, pop_up
    Line1 = "HVAC HEAT"  # update line 1
    pop_up = True
    update_lcd()
    pop_up = False

def show_stats():
    global Line1
    Line1 = get_string()
    update_lcd()
```

```
def get_string():
    global weather_index, desired_temp, door_window_open, hvac_on, ac_on, heat_on, lights

    w_index = str(round(weather_index))

    d_temp = str(round(desired_temp))

    dw_open = "OPEN" if (door_window_open is True) else "SAFE"

    hvac_status = "OFF"
    if ac_on:
        hvac_status = "AC       "
    elif heat_on:
        hvac_status = "HEAT     "
    else:
        hvac_status = "OFF      "

    l_status = "OFF" if (lights is False) else "ON"

    return w_index + "/" + d_temp + "     D:" + dw_open + "\n"+"H:" + hvac_status + "L:" + l_status
```

3.3. Notes

My LCD updates slow. Could speed it up with more threads but had time constraints due to other finals.

4. Libraries

The packages I used for this project are Adafruit_DHT, Adafruit_LCD1602, PCF8574, CIMIS_Extract, RPi.GPIO, time, threading, and sys. I will provide the files for Adafruit_LCD1602.py, CIMIS_Extract.py, and PCF8574.py.

4.1. Adafruit_DHT

I used the read_entry function from this library to get DHT sensor input.

4.2. Adafruit_LCD1602 & PCF8574

I used this library to setup my LCD as follows:

```
PCF8574_address = 0x27
mcp = PCF8574_GPIO(PCF8574_address)
lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4, 5, 6, 7], GPIO=mcp)
```

I mainly used the clear and message functions to display text on the LCD.

4.3. CIMIS_Extract

I used a CIMIS object from this library to update my CIMIS values repeatedly using cimis.update_value().

4.4. RPi.GPIO

Used for setting up inputs/outputs and the RaspberryPi circuit.

4.5. time

I used time.sleep() for delays in my script.

4.6. threading

Used to set up the different threads in m program