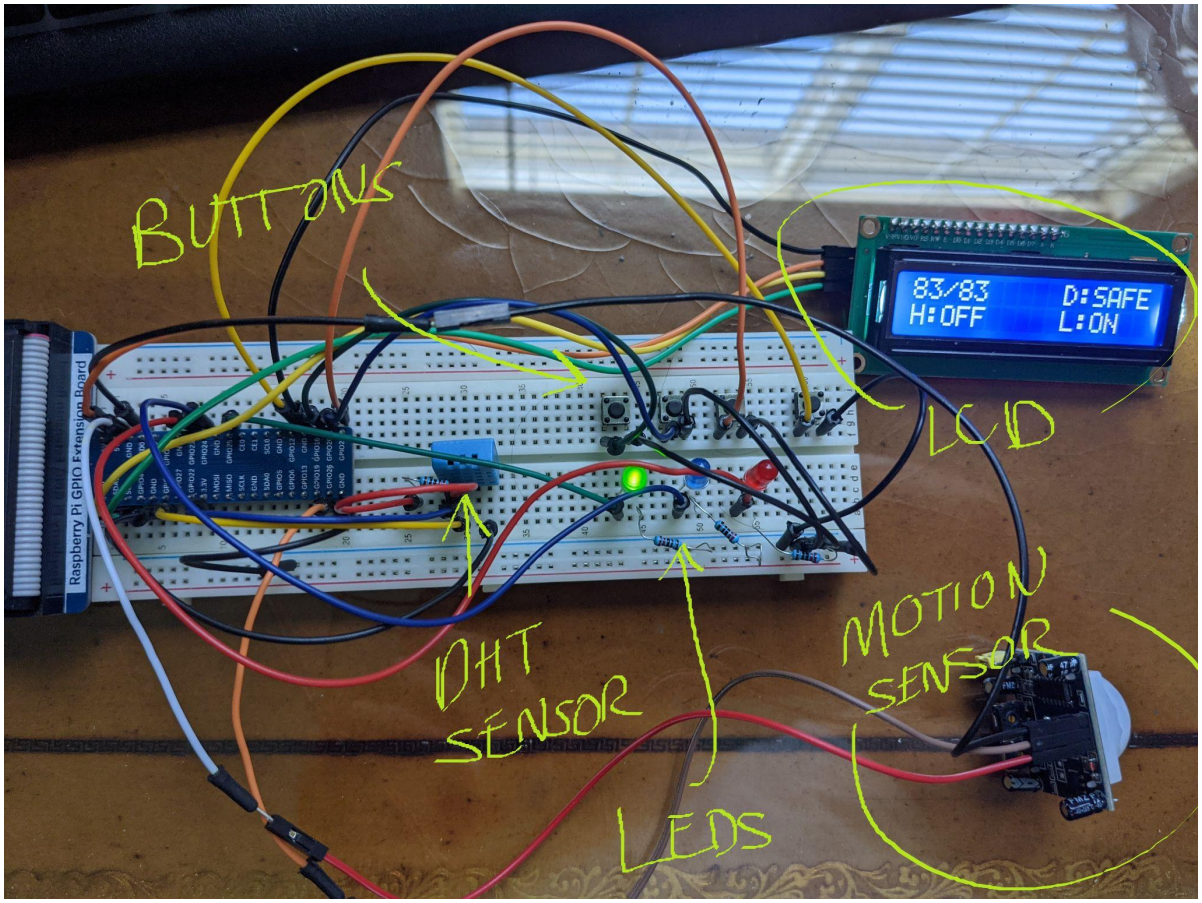
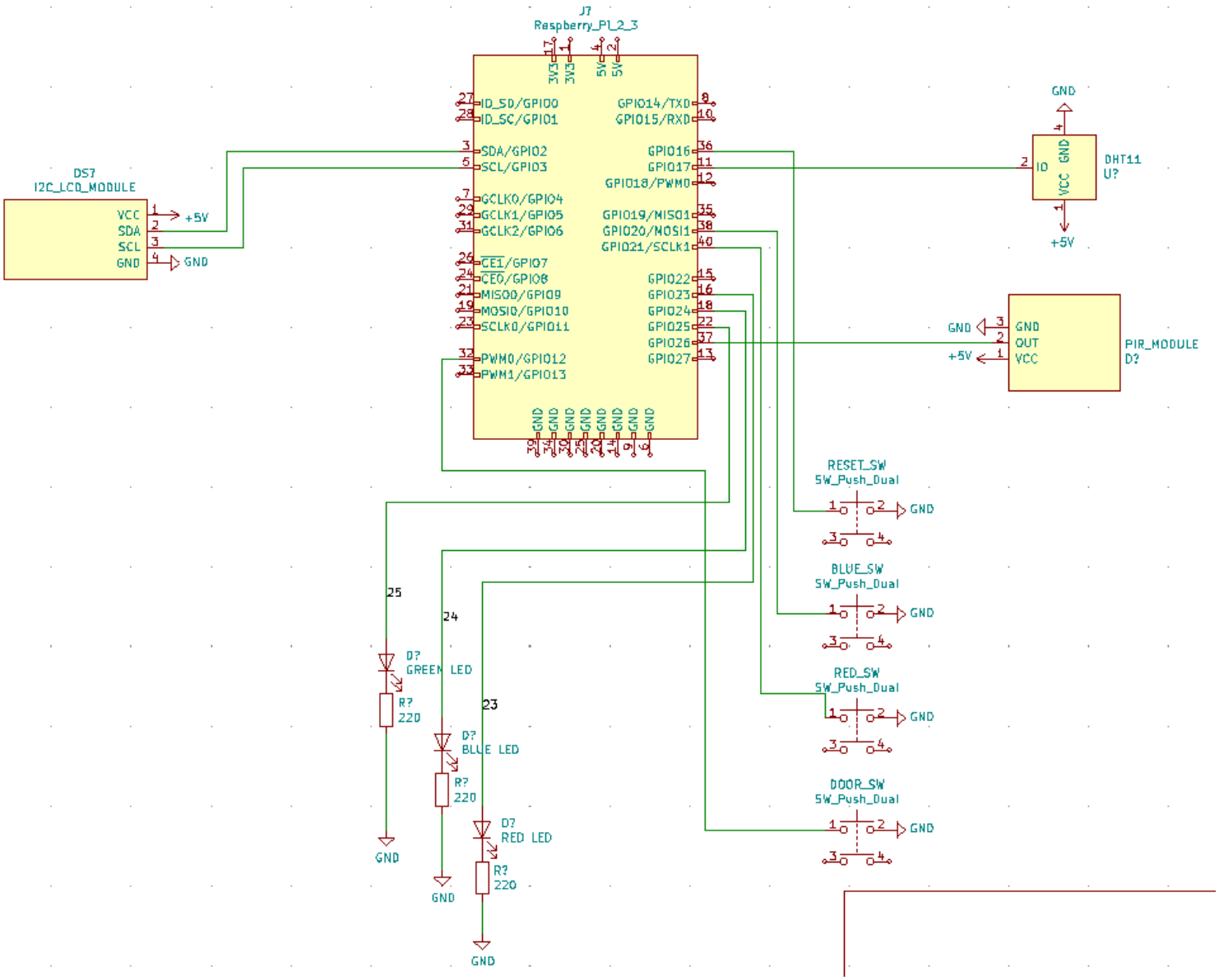
Chum, Ryan
57929221
EECS 113
Final Project

Schematic





The pin configurations are:

DHT_PIN = GPIO 17	#DHT input
LIGHT_PIN = GPIO 26	#PIR input
BTN_D = GPIO 12	#door input button
BTN_R = GPIO 21	#red input button
BTN_B = GPIO 20	#blue input button
BTN_S = GPIO 16	#reset input button
LED_R = GPIO 23	#red output led
LED_G = GPIO 25	#green output led
LED_B = GPIO 24	#blue output led

Procedure

Approach: After considering the separate modules involved in the building management system, I determined that a multithreaded program would be most appropriate. I used 3 different threads running concurrently to monitor the buttons, lights, and temperature. I used global variables to share common resources between these operations.

```
#global variables
dht = None #DHT object
temp_lock = threading.Lock() #temp mutex
lcd_lock = threading.Lock() #lcd mutex
door = 1    #0=open, 1=closed
temp = 0
set_temp = 0
temp_set_flag = 0 #set if temp is changed by user
hvac = 0 #0 = off, 1=AC, 2=Heater
lights = 0 #lcd lights indicator, 0=off, 1=on
motion = 0 #lights signal to turn on, 0=off, 1=on
```

The button_thread handles user input by instantiating the listener function and waiting on events to trigger the callbacks. There is an event for the 4 buttons: one increasing the set temperature (red), one decreasing the set temperature (blue), one opening and closing the door (btn_d), and one resetting the UI (my addition). Lastly, there is an event for the PIR sensor which calls the green function.

The dht_thread retrieves the temperature through the DHT sensor and updates the LCD every second.

The light_thread polls a global variable for motion and controls the green LED.

The main thread initializes the temperature for the first lcd update, starts the three threads and waits on them.

```
if __name__ == '__main__':     # Program entrance
    print('Program is starting...')
    #initialize temps
    dht = DHT.DHT(DHT_PIN) #create a DHT class object
    dht.readDHT11()
    temp = int(dht.temperature*9/5+32)
    set_temp = temp
    print('tempflag ='+str(temp_set_flag)+', door='+str(door)+'\n')
```

```
time.sleep(.5)
lcd_refresh()
button_thread = threading.Thread(target=button_loop)
dht_thread = threading.Thread(target=dht_loop)
light_thread = threading.Thread(target=light_loop)
button_thread.daemon = True
dht_thread.daemon = True
light_thread.daemon = True
try:
    button_thread.start()
    dht_thread.start()
    light_thread.start()
except KeyboardInterrupt:
    destroy()
button_thread.join()
dht_thread.join()
light_thread.join()
```

1. Display Status on LCD:
    a. The function lcd_refresh() displays the UI and is called whenever there are changes to the lcd. First it grabs the lcd_lock so that no other thread can make changes to the lcd while it is accessing it. It displays the temperature and the set temperature. Then, depending on whether the global variables are set or not, it displays if the door is open/closed, if the ac/heater is on, or if the lights are on. Lcd_refresh() is called every second by the dht_thread as it updates the temperature.

```
#update LCD hud
def lcd_refresh():
    lcd_lock.acquire()
    lcd.clear()
    lcd.setCursor(0,0)
    lcd.message(str(temp)+'/'+str(set_temp))
    lcd.setCursor(10,0)
    if (door == 1):
        lcd.message('D:SAFE')
    else:
        lcd.message('D:OPEN')
    lcd.setCursor(0,1)
    if (hvac == 0):
```

```
        lcd.message("H:OFF")
    elif (hvac == 1):
        lcd.message("H:AC")
    else:
        lcd.message("H:HEAT")
    lcd.setCursor(10,1)
    if (lights == 0):
        lcd.message('L:OFF')
    else:
        lcd.message('L:ON')
    lcd_lock.release()
```

2. Control Room Temperature (HVAC)
    a. In main, dht_thread starts, calling dht_loop() to update the temperature every second. First, it gets the humidity using get_hum(), a function that utilizes the python request library to fetch data from the CIMIS website and parses it to find the humidity value. The get_hum() function contains a while loop and handles exceptions on the request and parsing functions in case they create errors. If the data from today cannot be obtained, then it obtains the value from the day prior until it succeeds.

```
#CIMIS app key
app_key = 'c0a7eebb-598f-4558-a7fa-1d9d35668784'
station = 75 #irvine CIMIS station
#get humidity from CIMIS website
def get_hum():
    today = date.today()
    value = None
    while (value==None): #if current humidity not available, finds last
registered humidity
        try:
            response =
requests.get('http://et.water.ca.gov/api/data?appKey='+app_key+'&targets='
+str(station)+'&startDate='+str(today)+'&endDate='+str(today)+'&dataItems=
day-rel-hum-avg&unitOfMeasure=M')
        except requests.exceptions.RequestException as e:
            continue
        yesterday = today-datetime.timedelta(days=1)
        today = yesterday
        data = json.loads(response.text)
```

```
        try:
            value =
data["Data"]["Providers"][0]["Records"][0]["DayRelHumAvg"]["Value"]
        except:
            continue
    print("Humidity = "+value+"\n")
    return value
```

b. Next in dht_loop(), a list of three values is initialized to store the last 3 temperatures. A while loop begins, retrieving temperature values from the DHT sensor, converting them to Fahrenheit, and adding them to the list. Once the number of total temperatures retrieved reaches 3, the global variable, temperature, is computed as the weather index. Then, if the temp_set_flag is not set, set_temp becomes the temperature. This is so that the set_temp matches the temp until the user makes a change to set_temp. A mutex is needed whenever set_temp is accessed to prevent concurrent access from multiple threads.

```
def dht_loop():
    global temp, dht, set_temp, temp_set_flag
    print('DHT thread started')
    hum = get_hum()
    count = 0
    vals = [0,0,0]
    while(True):
        chk = dht.readDHT11()
        if (chk is dht.DHTLIB_OK): #read DHT11 and get a return
value. Then determine whether data read is normal according to the
return value.
            #convert to fahrenheit
            t = dht.temperature*9/5+32
            vals[count%3] = t
            count+=1
            #print("Temperature received: %.2f \n"%(t))
        if (count >= 3):
            temp = int(sum(vals)/3+.05*float(hum))
            if (temp_set_flag == 0):
                temp_lock.acquire()
                set_temp = temp
                temp_lock.release()
            print("Temperature : %d \n"%(temp))
```

```
        lcd_refresh() #update lcd with new temp every second
        time.sleep(1)   #repeat every second
```

c. The lcd refreshes and the loop waits one second before repeating again. This is the main driver for changes to the lcd.
d. The button_thread() updates the set_temp variable by waiting on the red and blue buttons to be pressed and triggering their callbacks.

```
#button handling thread
def button_loop():
    print('Button thread started')
    button_listener()
    while True:
        time.sleep(1e6)
def button_listener():
    GPIO.add_event_detect(BTN_R, GPIO.FALLING, callback=red,
bouncetime=1000)
    GPIO.add_event_detect(BTN_B, GPIO.FALLING, callback=blue,
bouncetime=1000)
    GPIO.add_event_detect(BTN_S, GPIO.FALLING, callback=reset,
bouncetime=1000)
    GPIO.add_event_detect(BTN_D, GPIO.FALLING, callback=set_door,
bouncetime=3000)
    GPIO.add_event_detect(LIGHT_PIN, GPIO.FALLING, callback =green,
bouncetime = 1000)
```

e. When red or blue is pressed, if the door is open (set to 0) before pressing the button, the lcd displays ‘door open, hvac off’ and no changes are made.
f. When either the red or blue button is pressed for the first time, the temp_set_flag is set so that set_temp no longer copies the real temperature.
g. When the red button is pressed and the set_temp (user set temperature) is less than 85, set_temp is increased by 1.
h. If the set_temp is less than the real temp-3, the heater is turned on, the screen is cleared and the lcd displays ‘heater on’. The global variable hvac is set to 2 to indicate that the heater is on.

```
def red(pin):   #heater
    global lcd, temp, set_temp, temp_set_flag, hvac
    #block changes if door is open
    if (door == 0):
```

```
        lcd_lock.acquire()
        lcd.clear()
        lcd.setCursor(0,0)
        lcd.message('Door Open')
        lcd.setCursor(0,1)
        lcd.message('HVAC Off')
        time.sleep(1)
        lcd_lock.release()
        lcd_refresh()
        return
    #once flag is set, set_temp no longer updates to temp
    if (temp_set_flag==0):
        temp_set_flag = 1
    GPIO.output(LED_R, True)
    #update set temp
    if (set_temp<85):
        temp_lock.acquire()
        set_temp+=1
        time.sleep(.5)
        temp_lock.release()
        print('set temp:'+str(set_temp)+'\n')
    #if temp<set_temp+3, turn on heater
    if (temp<=set_temp-3 and hvac!=2):
        hvac = 2
        lcd_lock.acquire()
        lcd.clear()
        lcd.setCursor(0,0)
        lcd.message('Heater On')
        time.sleep(3)
        lcd_lock.release()
    GPIO.output(LED_R, False)
    set_hvac()
    lcd_refresh()
```

i. Steps g and h are the same for the blue button except the set_temp is decreased by 1, it must be greater than 65, and the AC is turned on if it is greater than temp+3. Hvac is set to 1 when the AC is on.

j. After each button press, set_hvac() is called. Depending on the value of hvac, this function turns the blue and red leds on or off.

```
#turn on leds for ac and heater
def set_hvac():
    #if hvac off, turn off leds
    if (hvac == 0):
        GPIO.output(LED_B, False)
        GPIO.output(LED_R, False)
    #if ac set, turn on ac, turn off heater
    elif (hvac == 1):
        GPIO.output(LED_B, True)
        GPIO.output(LED_R, False)
    #if heater set, turn on heater, turn off ac
    elif (hvac ==2):
        GPIO.output(LED_B, False)
        GPIO.output(LED_R, True)
```

3. Monitor perimeter's entrances

```
def set_door(pin):
    global lcd, door, hvac, temp_set_flag, set_temp
    if (door == 1):
        door = 0    #open door
        lcd_lock.acquire()
        lcd.clear()
        lcd.setCursor(0,0)
        lcd.message('Door/Window Open')
        if (hvac != 0):
            lcd.setCursor(0,1)
            lcd.message('HVAC Halted')
            hvac = 0
            set_hvac()
        #reset set temp and flag
        temp_lock.acquire()
        set_temp = temp
        temp_set_flag = 0
        temp_lock.release()
        time.sleep(3.0)
        lcd.clear()
        lcd_lock.release()
        lcd_refresh()
        return
```

```
    else:
        door = 1 #close door
        lcd_lock.acquire()
        lcd.clear()
        lcd.setCursor(0,0)
        lcd.message('Door/Window')
        lcd.setCursor(0,1)
        lcd.message('Closed')
        #reset set temp and flag
        temp_lock.acquire()
        set_temp = temp
        temp_set_flag = 0
        temp_lock.release()
        time.sleep(3.0)
        lcd.clear()
        lcd_lock.release()
        set_hvac()
        lcd_refresh()
        return
```

a. set_door() monitors the door/windows and is also activated in the button_thread().
b. If the door is closed, the door is opened. The lcd displays ‘door/window open’ for 3 seconds, along with ‘HVAC halted’ if hvac is not 0. It also resets hvac as well as the set temperature and temp_set_flag.
c. If the door is open, the door is closed and the lcd displays ‘door/window closed’ for 3 seconds.
d. The leds are set using set_hvac() and the display is refreshed with lcd_refresh() in each case.

4. Control the Ambient Lighting
    a. Lighting is handled in two threads. First, the button_thread() detects motion from the sensor, turns on the green LED, and sets the light and motion variables.

```
#motion detected light
def green(pin):
    global lights, motion
    GPIO.output(LED_G,True)
    print ('led turned on <<<')
    motion = 1
    lights = 1
```

b. Meanwhile, the light_thread polls the motion variable. If it is set, it unsets it and waits 10 seconds. Because motion is unset during this wait period, the lights variable needs to still be set so that the LCD can display it as on. If the button_thread() detects more motion during this wait, motion is set once again and the wait is extended another 10 seconds after completion. Otherwise, the light is turned off and the lights variable is unset.

```
#turn on light when movement detected
def light_loop():
    global lights, motion
    #polling loop checks if motion signal is set
    while True:
        if (motion == 1):
            motion = 0
            time.sleep(10)  #light turns off if no motion detected during
this period, else renews
            if (motion == 0):
                GPIO.output(LED_G, False)
                lights = 0
```

5. Reset Button (Extra)

```
#reset leds and default values
def reset(pin):
    global door, temp, set_temp, temp_set_flag, hvac, lights, motion
    lcd_lock.acquire()
    temp_lock.acquire()
    GPIO.output(LED_R, False)
    GPIO.output(LED_G, False)
    GPIO.output(LED_B, False)
    door = 1
    temp = 0
    set_temp = 0
    temp_set_flag = 0
    hvac = 0
    lights = 0
    motion = 0
```

```
    lcd.clear()
    lcd.setCursor(0,0)
    lcd.message('HVAC RESET')
    time.sleep(1)
    lcd.clear()
    temp_lock.release()
    lcd_lock.release()
```

a. The button thread also handles the reset button when it is clicked, calling the reset() function. The function refreshes all global variables, clears the LCD, and turns off all LEDs. Essentially, it would be the same as if the device were restarted. The lcd will refresh after the brief message, ‘HVAC RESET’, due to the dht thread in the background.

Code:

```
#!/usr/bin/python
#Assignment 4

import threading
import RPi.GPIO as GPIO
import time
import requests
import json
from datetime import date
import datetime
import Freenove_DHT as DHT
from PCF8574 import PCF8574_GPIO
from Adafruit_LCD1602 import Adafruit_CharLCD

##Pin numbering declarations (setup channel mode of the Pi to Board values)
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
##Set GPIO pins (for inputs and outputs) and all setupts needed based on assignment description
DHT_PIN = 17
LIGHT_PIN = 26
BTN_D = 12
BTN_R = 21
BTN_B = 20
BTN_S = 16
LED_R = 23
LED_G = 25
LED_B = 24

GPIO.setup(21,GPIO.IN, pull_up_down=GPIO.PUD_UP)    #red button
GPIO.setup(20,GPIO.IN, pull_up_down=GPIO.PUD_UP)    #blue button
GPIO.setup(16,GPIO.IN, pull_up_down=GPIO.PUD_UP)    #green button
GPIO.setup(12,GPIO.IN, pull_up_down=GPIO.PUD_UP)    #door button
GPIO.setup(LIGHT_PIN, GPIO.IN)  # set sensorPin to INPUT mode

GPIO.setup(23,GPIO.OUT)                     #red LED
GPIO.setup(24,GPIO.OUT)                    #blue LED
GPIO.setup(25,GPIO.OUT)                     #green LED
```

```
#global variables
dht = None #DHT object
temp_lock = threading.Lock() #temp mutex
lcd_lock = threading.Lock() #lcd mutex
door = 1    #0=open, 1=closed
temp = 0
set_temp = 0
temp_set_flag = 0 #set if temp is changed by user
hvac = 0 #0 = off, 1=AC, 2=Heater
lights = 0 #lcd lights indicator, 0=off, 1=on
motion = 0 #lights signal to turn on, 0=off, 1=on

#Configure LCD
PCF8574_address = 0x27 # I2C address of the PCF8574 chip.
PCF8574A_address = 0x3F # I2C address of the PCF8574A chip.
try:
        mcp = PCF8574_GPIO(PCF8574_address)
except:
        try:
                mcp = PCF8574_GPIO(PCF8574A_address)
        except:
                print ('I2C Address Error !')
                exit(1)
# Create LCD, passing in MCP GPIO adapter.
lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4,5,6,7], GPIO=mcp)
mcp.output(3,1)
lcd.begin(16,2)          #set number of LCD lines

#CIMIS app key
app_key = 'c0a7eebb-598f-4558-a7fa-1d9d35668784'
station = 75 #irvine CIMIS station
#get humidity from CIMIS website
def get_hum():
   today = date.today()
   value = None
   while (value==None): #if current humidity not available, finds last registered humidity
      try:
```

```
            response =
requests.get('http://et.water.ca.gov/api/data?appKey='+app_key+'&targets='+str(station)+'&start
Date='+str(today)+'&endDate='+str(today)+'&dataItems=day-rel-hum-avg&unitOfMeasure=M')
        except requests.exceptions.RequestException as e:
            continue
        yesterday = today-datetime.timedelta(days=1)
        today = yesterday
        data = json.loads(response.text)
        try:
            value = data["Data"]["Providers"][0]["Records"][0]["DayRelHumAvg"]["Value"]
        except:
            continue
    print("Humidity = "+value+"\n")
    return value

## Event listener (Tell GPIO library to look out for an event on each pushbutton and pass handle
function)
## fucntion to be run for each pushbutton detection ##
def button_listener():
            GPIO.add_event_detect(BTN_R, GPIO.FALLING, callback=red, bouncetime=1000)
            GPIO.add_event_detect(BTN_B, GPIO.FALLING, callback=blue, bouncetime=1000)
            GPIO.add_event_detect(BTN_S, GPIO.FALLING, callback=reset, bouncetime=1000)
            GPIO.add_event_detect(BTN_D, GPIO.FALLING, callback=set_door,
bouncetime=3000)
            GPIO.add_event_detect(LIGHT_PIN, GPIO.FALLING, callback =green, bouncetime =
1000)

#button and pir handler functions
#change door open/closed
def set_door(pin):
    global lcd, door, hvac, temp_set_flag, set_temp
    if (door == 1):
        door = 0    #open door
        lcd_lock.acquire()
        lcd.clear()
        lcd.setCursor(0,0)
        lcd.message('Door/Window Open')
        if (hvac != 0):
            lcd.setCursor(0,1)
            lcd.message('HVAC Halted')
```

```
            hvac = 0
            set_hvac()
        #reset set temp and flag
        temp_lock.acquire()
        set_temp = temp
        temp_set_flag = 0
        temp_lock.release()
        time.sleep(3.0)
        lcd.clear()
        lcd_lock.release()
        lcd_refresh()
        return
    else:
        door = 1 #close door
        lcd_lock.acquire()
        lcd.clear()
        lcd.setCursor(0,0)
        lcd.message('Door/Window')
        lcd.setCursor(0,1)
        lcd.message('Closed')
        #reset set temp and flag
        temp_lock.acquire()
        set_temp = temp
        temp_set_flag = 0
        temp_lock.release()
        time.sleep(3.0)
        lcd.clear()
        lcd_lock.release()
        set_hvac()
        lcd_refresh()
        return
#update LCD hud
def lcd_refresh():
    lcd_lock.acquire()
    lcd.clear()
    lcd.setCursor(0,0)
    lcd.message(str(temp)+'/'+str(set_temp))
    lcd.setCursor(10,0)
    if (door == 1):
        lcd.message('D:SAFE')
```

```
    else:
        lcd.message('D:OPEN')
    lcd.setCursor(0,1)
    if (hvac == 0):
        lcd.message("H:OFF")
    elif (hvac == 1):
        lcd.message("H:AC")
    else:
        lcd.message("H:HEAT")
    lcd.setCursor(10,1)
    if (lights == 0):
        lcd.message('L:OFF')
    else:
        lcd.message('L:ON')
    lcd_lock.release()

#turn on leds for ac and heater
def set_hvac():
    #if hvac off, turn off leds
    if (hvac == 0):
        GPIO.output(LED_B, False)
        GPIO.output(LED_R, False)
    #if ac set, turn on ac, turn off heater
    elif (hvac == 1):
        GPIO.output(LED_B, True)
        GPIO.output(LED_R, False)
    #if heater set, turn on heater, turn off ac
    elif (hvac ==2):
        GPIO.output(LED_B, False)
        GPIO.output(LED_R, True)

#reset leds and default values
def reset(pin):
    global door, temp, set_temp, temp_set_flag, hvac, lights, motion
    lcd_lock.acquire()
    temp_lock.acquire()
    GPIO.output(LED_R, False)
    GPIO.output(LED_G, False)
    GPIO.output(LED_B, False)
    door = 1
```

```
    temp = 0
    set_temp = 0
    temp_set_flag = 0
    hvac = 0
    lights = 0
    motion = 0
    lcd.clear()
    lcd.setCursor(0,0)
    lcd.message('HVAC RESET')
    time.sleep(1)
    lcd.clear()
    temp_lock.release()
    lcd_lock.release()

#increase set temperature, turn on heater if 3 degrees above temp
def red(pin):    #heater
    global lcd, temp, set_temp, temp_set_flag, hvac
    #block changes if door is open
    if (door == 0):
        lcd_lock.acquire()
        lcd.clear()
        lcd.setCursor(0,0)
        lcd.message('Door Open')
        lcd.setCursor(0,1)
        lcd.message('HVAC Off')
        time.sleep(1)
        lcd_lock.release()
        lcd_refresh()
        return
    #once flag is set, set_temp no longer updates to temp
    if (temp_set_flag==0):
        temp_set_flag = 1
    GPIO.output(LED_R, True)
    #update set temp
    if (set_temp<85):
        temp_lock.acquire()
        set_temp+=1
        time.sleep(.5)
        temp_lock.release()
        print('set temp:'+str(set_temp)+'\n')
```

```
    #if temp<set_temp+3, turn on heater
    if (temp<=set_temp-3 and hvac!=2):
        hvac = 2
        lcd_lock.acquire()
        lcd.clear()
        lcd.setCursor(0,0)
        lcd.message('Heater On')
        time.sleep(3)
        lcd_lock.release()
    GPIO.output(LED_R, False)
    set_hvac()
    lcd_refresh()

#decrease set temperature, turn on AC if 3 deg below temp
def blue(pin):  #AC
    global lcd, temp, set_temp, temp_set_flag, hvac
    #block changes if door is open
    if (door == 0):
        lcd_lock.acquire()
        lcd.clear()
        lcd.setCursor(0,0)
        lcd.message('Door Open')
        lcd.setCursor(0,1)
        lcd.message('HVAC Off')
        time.sleep(1)
        lcd_lock.release()
        lcd_refresh()
        return
    #once flag is set, set_temp no longer updates to temp
    if (temp_set_flag==0):
        temp_set_flag = 1
    GPIO.output(LED_B, True)
    #update set temp
    if (set_temp>65):
        temp_lock.acquire()
        set_temp-=1
        time.sleep(.5) #led time
        temp_lock.release()
        print('set temp:'+str(set_temp)+'\n')
    #if temp<set_temp+3, turn on heater
```

```
    if (temp>=set_temp+3 and hvac!=1):
        hvac = 1
        lcd_lock.acquire()
        lcd.clear()
        lcd.setCursor(0,0)
        lcd.message('AC ON')
        time.sleep(3)
        lcd_lock.release()
    GPIO.output(LED_B, False)
    set_hvac()
    lcd_refresh()

#motion detected light
def green(pin):
    global lights, motion
    GPIO.output(LED_G,True)
    print ('led turned on <<<')
    motion = 1
    lights = 1

#turn on light when movement detected
def light_loop():
    global lights, motion
    #polling loop checks if motion signal is set
    while True:
        if (motion == 1):
            motion = 0
            time.sleep(10)  #light turns off if no motion detected during this period, else renews
            if (motion == 0):
                GPIO.output(LED_G, False)
                lights = 0

#button handling thread
def button_loop():
    print('Button thread started')
    button_listener()
    while True:
        time.sleep(1e6)

#temperature updating thread
```

```
def dht_loop():
    global temp, dht, set_temp, temp_set_flag
    print('DHT thread started')
    hum = get_hum()
    count = 0
    vals = [0,0,0]
    while(True):
        chk = dht.readDHT11()
        if (chk is dht.DHTLIB_OK): #read DHT11 and get a return value. Then determine whether
data read is normal according to the return value.
            #convert to fahrenheit
            t = dht.temperature*9/5+32
            vals[count%3] = t
            count+=1
            #print("Temperature received: %.2f \n"%(t))
        if (count >= 3):
            temp = int(sum(vals)/3+.05*float(hum))
            if (temp_set_flag == 0):
                temp_lock.acquire()
                set_temp = temp
                temp_lock.release()
            print("Temperature : %d \n"%(temp))
        lcd_refresh() #update lcd with new temp every second
        time.sleep(1)   #repeat every second

def destroy():
    GPIO.cleanup()      # Release GPIO resource
    sys.exit()

if __name__ == '__main__':     # Program entrance
    print('Program is starting...')
    #initialize temps
    dht = DHT.DHT(DHT_PIN) #create a DHT class object
    dht.readDHT11()
    temp = int(dht.temperature*9/5+32)
    set_temp = temp
    print('tempflag ='+str(temp_set_flag)+', door='+str(door)+'\n')
    time.sleep(.5)
    lcd_refresh()
    button_thread = threading.Thread(target=button_loop)
```

```
dht_thread = threading.Thread(target=dht_loop)
light_thread = threading.Thread(target=light_loop)
button_thread.daemon = True
dht_thread.daemon = True
light_thread.daemon = True
try:
    button_thread.start()
    dht_thread.start()
    light_thread.start()
except KeyboardInterrupt:
    destroy()
button_thread.join()
dht_thread.join()
light_thread.join()
```