NAMIBIA UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Faculty of Engineering

Department of Mechanical, Industrial & Electrical Engineering

T: +264 61 207 2531
F: +264 61 207 9531
E: dece@nust.na
W: www.nust.na

PROJECT

IoT Automated Workspace

Prepared for:

Dr. Zacchaeus Oyodokun
ECD811S
Faculty of Engineering
Department of Mechanical, Industrial and Electrical Engineering
Namibia University of Science and Technology

Prepared by:

Daniel Paulo Viegas
221004300
Student
Bachelor of Engineering Electronics and Telecommunications

Due Date: 31/05/2024

ACKNOWLEDMENT

Profound gratitude and thanks are extended first to Christ which through Him everything is achievable, and the University of Science and Technology, with particular emphasis on the Department of Mechanical, Industrial and Electrical Engineering. The introduction of this course and the learning opportunities provided have awakened a spirit of creativity and innovation.

The completion of this module would not have been possible without the invaluable help and support of mentor Dr. Oyedokun. His passionate and professional advice optimized the design, ensuring adherence to course requirements and timely project completion.

Finally, heartfelt appreciation goes to family and friends, whose support and encouragement during challenging times provided the strength to persevere. Special thanks are also extended to all those who contributed via constructive criticism and guidance, as their input was instrumental in making this journey possible.

ACRONYMS

- IoT - Internet of Things
- SMART - Specific Measurable Achievable Relevant and Time-bound
- PIR - Passive Infrared
- GPIO - General Purpose Input/Output
- DHT - Digital Humidity and Temperature
- LED - Light Emitting Diode
- DC - Direct Current
- HTML – Hypertext Markup Language
- SMS - Short Message Service (not directly mentioned in the report, but implied by Telegram message)
- OS – Operating System
- CSV – Comma Separated Values

ABSTRACT

The report will detail the steps taken to design the IoT Workspace, with an emphasis on the choice to code a web application. It will provide a step-by-step guide for connecting a Raspberry Pi to the cloud server. The Python codes used to actuate all sensors are included as annexes, along with detailed explanations of how each sensor interfaces with the Raspberry Pi. The IoT workstation was designed with two modes of operation:

Autonomous Mode: This mode operates automatically, making decisions based on preset trigger conditions in the code. Additional conditions are configured on the cloud server to interpret sensor inputs. Based on these readings, the Raspberry Pi acts without human intervention to control the actuators. One part of the code enables the Raspberry Pi to function autonomously even without internet connectivity, while another part relies on internet connectivity to meet the IoT project requirements.

Remote Monitoring and Control Mode: This mode allows technicians or end users to remotely monitor the status of each sensor and device connected to the Raspberry Pi. Monitoring capabilities are further enhanced to include remote control of the system, as monitoring alone was deemed insufficient. This remote-control feature grants the online mode administrative rights over the offline mode, allowing the online mode to override automatic preset trigger parameters if the technician or end user needs to change the status of a device.

Table of Contents

TERMS OF REFERENCE

Dr. Oyedokun, the Coordinator and Lecturer of ECD810s, requested the completion of a final report for the IoT-Automated Workspace project. It is also essential that all projects include a comprehensive technical report. This report should provide detailed, step-by-step information on the project's execution, enabling other technicians to replicate the project with ease. All key specifications and requirements should be thoroughly outlined.

1. Introduction

Namibia and many other nations in Africa face several challenges in the realms of high energy efficiency , physical security, and remote monitoring technology. This is caused by high energy consumption costs and the lack of infrastructure in various regions across the continent. These challenges necessitate Specific Measurable Achievable Relevant and Time-bound (SMART) innovative solutions that can address local needs and contribute to the socio-economic development. A certain percentage of energy efficiency and physical security could be increased, by establishing a system that could assist with reducing high energy consumption while increasing security of companies, workspaces, and homes.

The proposed IoT Automated Workspace emerges as a strategic response to these pressing challenges, by utilizing IoT and Sensor technology. Such a system can assist with minor actions such as automated switching (ON/OFF) of lights, fans, and alarm in case of security breach.
This is achieved by interfacing new or existing industrial systems to a microcontroller which in this report is a Raspberry Pi, via coding. This gives technicians the ability to monitor and control the status of many equipment used in workplaces and homes.

In this project a Web-App was designed to display the statuses of the DHT11 and PIR motion sensors, show and control the status of three actuators: a fan, buzzer, and LED.

The system operates under various conditions specified in the code:

1. When a person enters the workspace, the PIR sensor detects motion. For this prototype, the PIR sensor can detect motion within a distance of up to 7 meters and a 100-degree field of view, adjustable by the end-user. The trigger condition is set as follows:

If a person is within ≥7 meters, the system will turn on the lights in the workspace. If no person is detected, the lights remain off, conserving energy.

The status of the lights is displayed on the Web-App.

2. The DHT11 sensor measures the temperature and humidity of the workspace. It is linked to the Web-App, and the condition is set as follows:

If the workspace temperature is ≥ 25°C and the PIR sensor detects motion, the motor fan will start running. If either condition is not met, the motor fan remains off.

This ensures efficient and cost-effective cooling, activated only when the temperature exceeds a comfortable range for the person present. The status of the DHT11 sensor and the fan is displayed on the Web-App.

3. The alarm system is set to detect motion "After Hours," a period when no motion is expected. The time frame is adjustable. The condition is set as follows:

If the PIR sensor detects motion between, for example, 18:00 and 06:00, a buzzer is activated. If no motion is detected within this time frame, the buzzer remains off.

The buzzer alert is displayed on the Web-App, promoting security within offices and homes.

The prototype includes a manual override for the automated system. This feature is crucial for situations where a person is not on site but wishes to keep the lights on or when the DHT11 sensor triggers the fan, but the technician wants to keep it off. Additionally, it allows for remote control to switch off devices if left on inadvertently after hours.

This system, when correctly implemented, has the potential to reduce costs, improve work output, and enhance security by minimizing the need for human intervention in performing repetitive tasks.

2. Objectives

- To utilize an Infrared motion sensor to detect human presence on the workspace/office to actuate the lights, fan, and buzzer if motion is detected after work hours.

- To utilize a digital temperature sensor to measure the workspace temperature and determine when to actuate the fan.

- To deploy an IoT system that facilitates monitoring and control of the entire system remotely.

3. Literature Review

During the research phase of this project, it was observed that numerous IoT-based systems predominantly utilize microcontrollers from the Arduino family for interfacing various sensors, including the DHT11 and PIR sensors. These applications range from environmental monitoring, such as measuring temperature and humidity, to security systems that detect motion. However, projects involving the Raspberry Pi have also gained prominence, highlighting its versatile computing capabilities and real-time processing power. Unlike Arduino, which requires re-uploading code for updates, the Raspberry Pi allows dynamic processing, making it suitable for more complex applications that benefit from real-time data analysis and remote monitoring. This project leverages the Raspberry Pi's strengths to design an IoT workspace with autonomous and manual override modes, interfacing sensors through Python code, and ensuring seamless integration with cloud services for efficient monitoring and control.

3.1. Author 1

Pimylifeup provides a comprehensive guide on interfacing a PIR sensor and a buzzer with a Raspberry Pi. The guide emphasizes the simplicity and effectiveness of using the Raspberry Pi for motion detection applications. The PIR sensor, which detects infrared radiation from moving objects, is connected to the Raspberry Pi's GPIO pins. This setup allows the Raspberry Pi to read signals from the sensor and trigger specific actions, such as activating a buzzer, when motion is detected (Pimylifeup, 2023).

The guide outlines the step-by-step process of wiring the PIR sensor and the buzzer to the Raspberry Pi. It includes detailed instructions on configuring the GPIO pins and writing Python scripts to handle sensor input and control the buzzer. The use of Python, a versatile and widely used programming language, simplifies the process of coding, and enhances the flexibility of the project. By leveraging Python's libraries, users can easily customize the system to suit specific requirements, such as setting different trigger conditions or incorporating additional sensors (Pimylifeup, 2023).

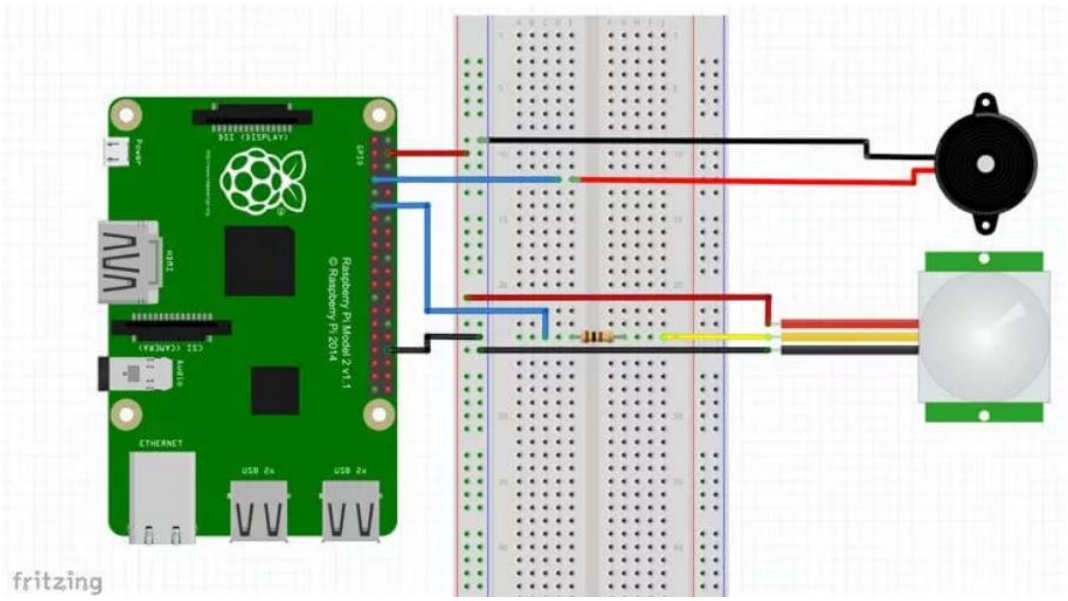

Figure 1: Circuit Diagram (Pimylifeup, 2023)

The practical applications of this setup are extensive, ranging from home security systems to automated lighting solutions. The ability to detect motion and trigger an alarm makes it an ideal choice for enhancing security in homes and offices. Furthermore, the real-time processing capabilities of the Raspberry Pi ensure that the system responds promptly to sensor input, providing reliable and efficient performance.

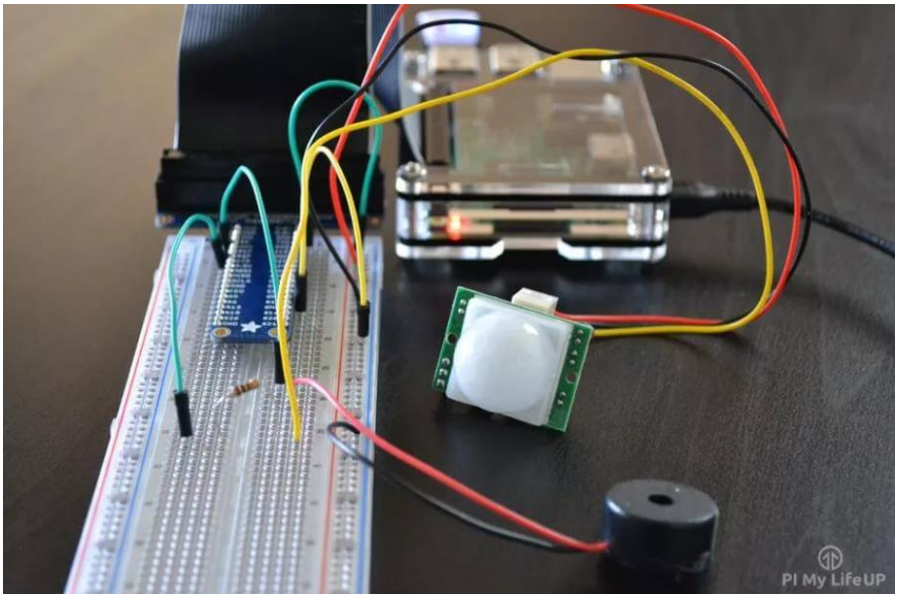

Figure 2: Breadboard Connection (Pimylifeup, 2023)

3.2. Author 2

In their tutorial, Newbiely (2023) provides a comprehensive guide on setting up a cooling system using a DHT11 sensor and a fan with a Raspberry Pi. The tutorial emphasizes the practical aspects of interfacing these components to create an automated system that monitors temperature and humidity, triggering the fan when certain thresholds are reached. The process begins with setting up the Raspberry Pi and installing the necessary libraries, such as the Adafruit_DHT library, which simplifies the interaction with the DHT11 sensor.

sudo apt-get update
sudo apt-get install python3-rpi.gpio

sudo pip3 install Adafruit_DHT

The tutorial highlights the importance of accurate sensor placement and secure wiring to ensure reliable data readings (Newbiely, 2023). The Python code provided in the tutorial demonstrates how to read data from the DHT11 sensor and control the GPIO pins of the Raspberry Pi to actuate the fan. This approach allows the system to operate autonomously, adjusting the fan speed based on real-time temperature readings, thus ensuring an efficient cooling mechanism.

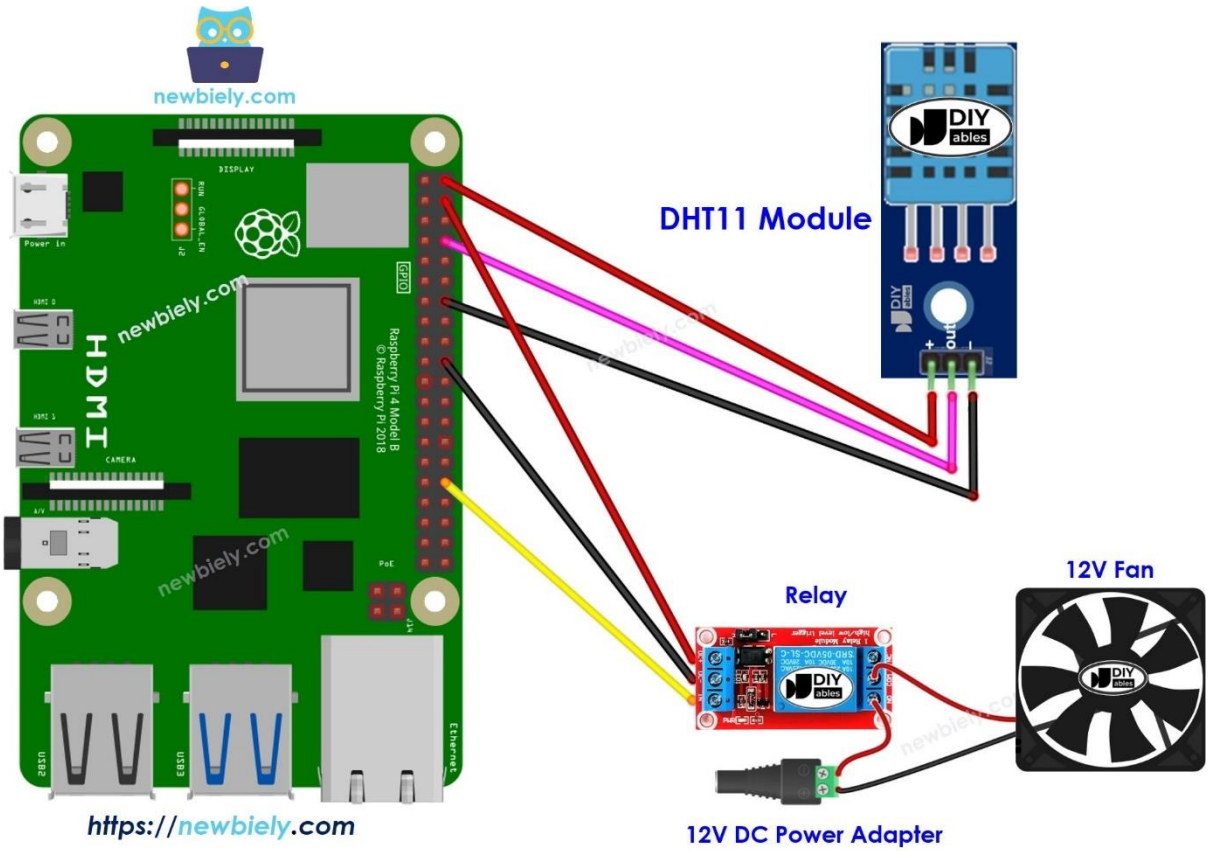

Figure 3: Circuit Diagram (Newbiely, 2023)

4. Equipment Used

- Raspberry Pi 3B+ Microcontroller
- Power Adapter
- PIR sensor
- DHT11 sensor
- 12V Fan
- Blue LED
- Breadboard and Jumper Wires
- Banana to Crocodile cables (2 pairs)
- Relay (2)
- DC power supply
- Buzzer
- HDMI – VGA converter
- VGA Cable

5. Procedures

1. Install the required libraries for each of the sensors (DHT11 & PIR) in the raspberry pi.
2. Log into Thonny on the raspberry to begin coding in python, write the specific code for all the conditions specified in this project.
3. Create a 5V line and a Ground line on the breadboard by connecting the 5V and Ground pin of the raspberry pi to the breadboard.
4. Connect the DHT11 sensor to the pi, make sure the DHT11 sensor is connected to only 3.3V of the raspberry pi.
5. Connect the PIR to the raspberry pi.
6. Connect the 12V fan to the raspberry via the relay and connect the other end to the dc power supply.
7. Connect the 6V buzzer to the raspberry via the relay and connect the other end to the dc power supply.
8. Connect the LED representing a bulb to the raspberry pi.
9. Test the respective codes and actuate the sensors.
10. Established Raspberry Pi and Phone internet connection by internet hotspot.
11. Write the respective html code to configure the respective sensors and actuators with their equivalent Online GPIO pins.
12. Run the system.

6. Design

6.1. Operational Flow Diagram

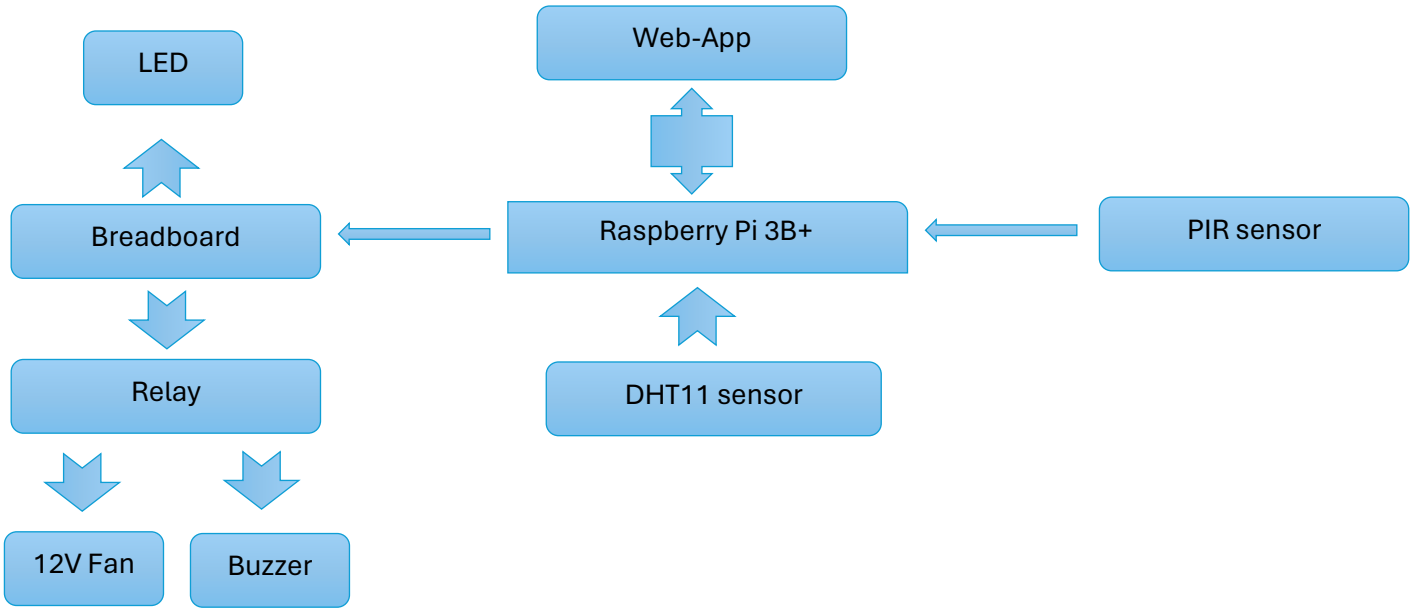

Figure 4.1: System Block Diagram

The above diagram shows the system's flow diagram, with the Raspberry pi being the processor which converts all the inputs into their respective outputs.

6.2. Schematic Diagram

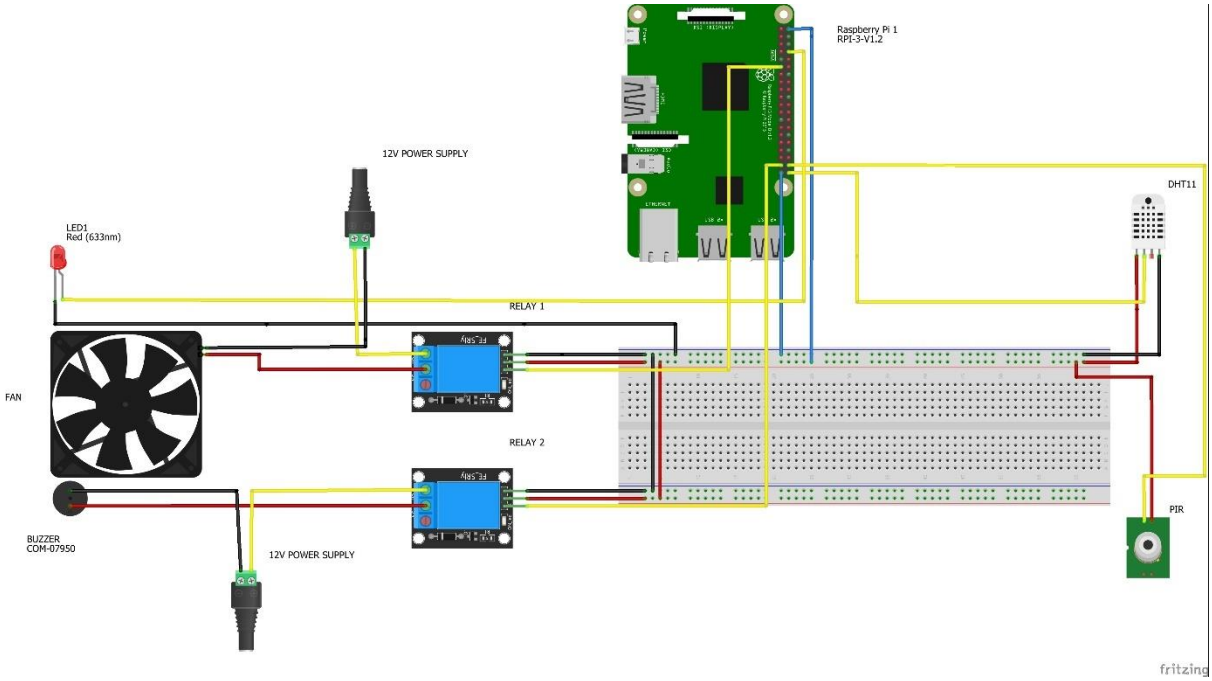

Figure 4.2: Fritzing Schematic Diagram

Figure 4.2 displays the system's clearly defined setup.

As opposed to using third party apps for the IoT application, this author chose to create a Web-App. This choice was made because a Web-App has the following benefits: no limitations on the type of sensor, the number of devices with access and additional security as the webapp can only be accessed with the network's IP address.

The web-app consists of 4 buttons, 3 to control the actuators and 1 to reset the system. It also has 3 displays to display the sensor readings.

After running the code, the next step is to connect the circuit. Connect the DHT11 and PIR sensors to 3.3V, GPIO PIN 21and 5V, GPIO PIN 20 and ground respectively. Connect the LED to GPIO PIN 14 and ground.

Next the fan and buzzer need to be connected, but since these actuators draw a high amount of current two relays are required. Connect the relays to an external DC power supply of 12V, the signal pins to GPIO PIN 17 ,GPIO PIN 26 and ground respectively.

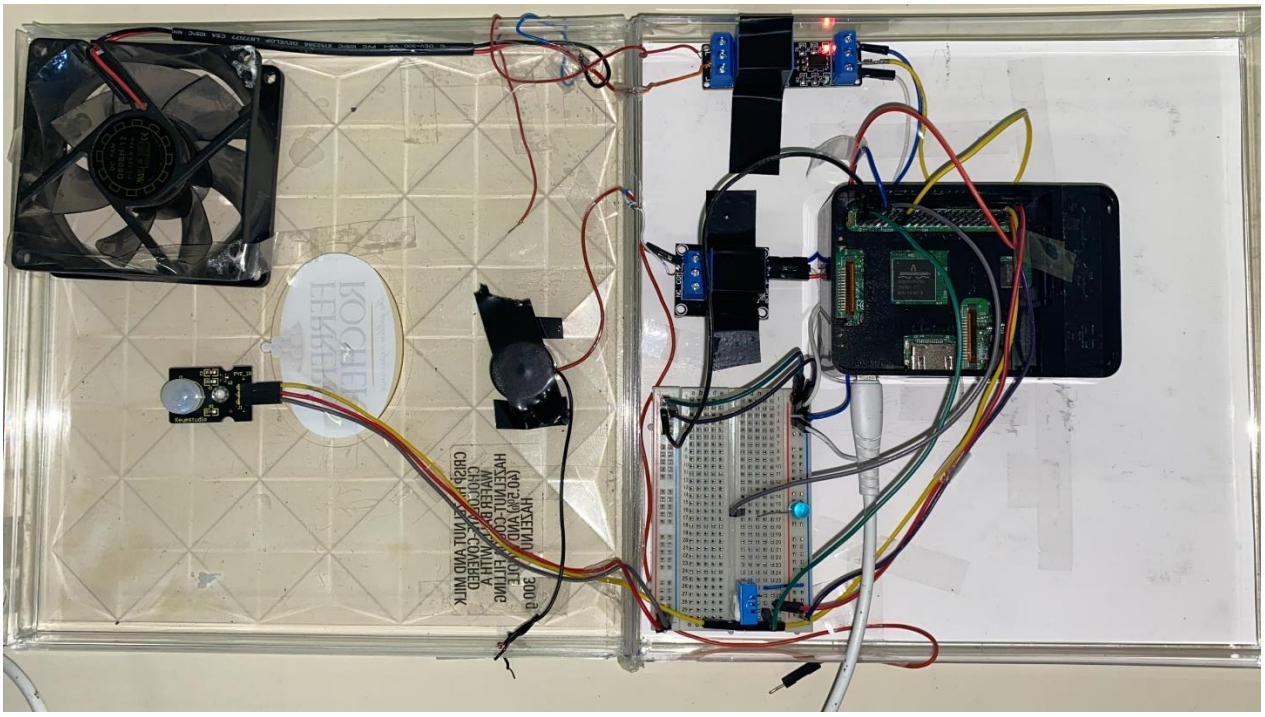

Figure 5: Circuitry

The raspberry pi was connected to power, turned ON and connected the network (HOTSPOT). The raspberry pi's was assigned an IP address, and this IP address was inserted into the code. Another device (Phone) was connected to the same network. To access the Web-App the following text was searched in a web-browser:

http://IP address of the raspberry pi:8000

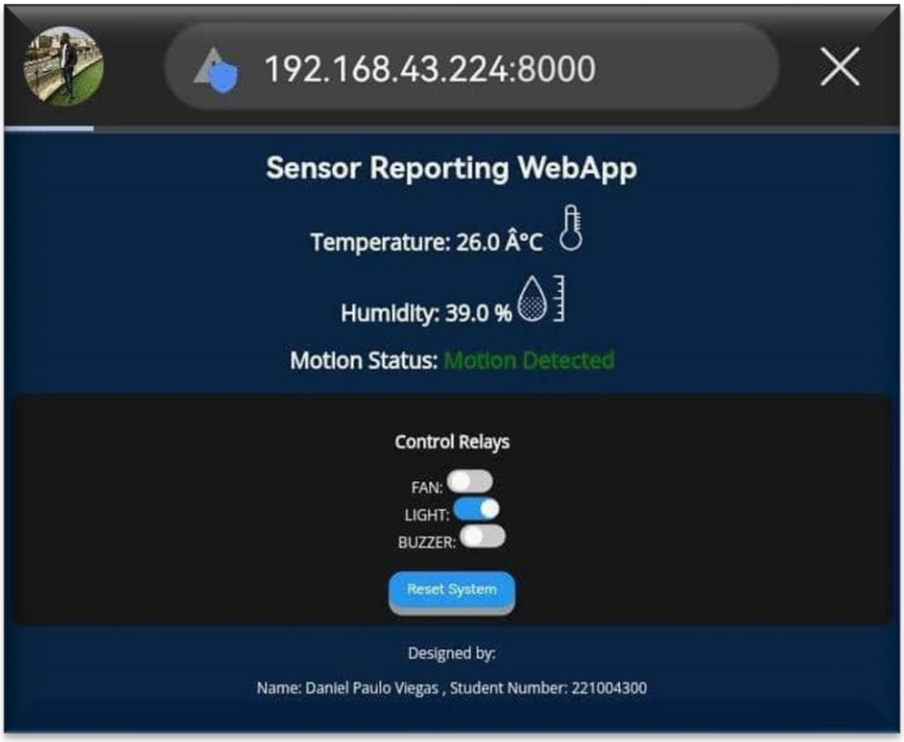

Figure 6: Web App on the phone

An additional security system was added to the code. This code enables the user to receive the message notifications 'Intruder Alert' when the alarm system is triggered. This message is sent via a BOT on telegram.

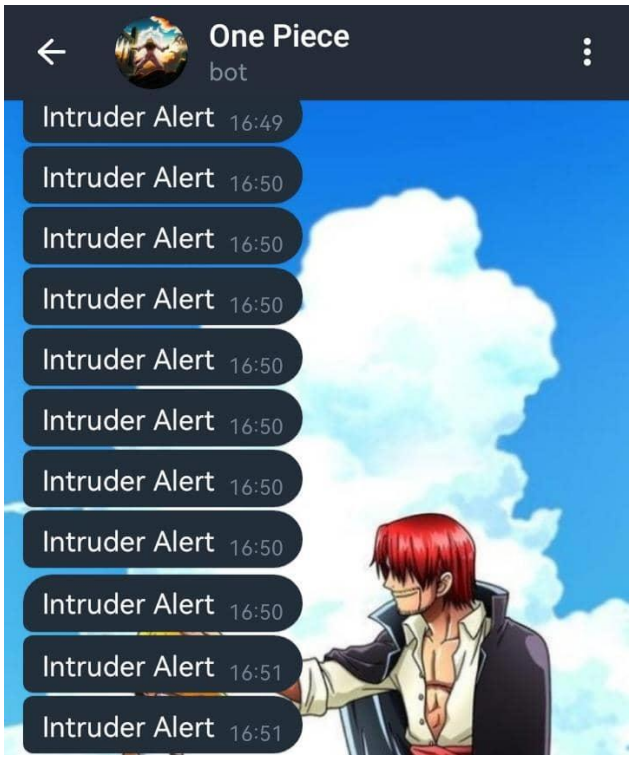

Figure 7: Telegram Bot message

The final step was to test the system and its IoT functionalities to verify that the entire system was running accurately according preset to the conditions.

7. Results and Discussion

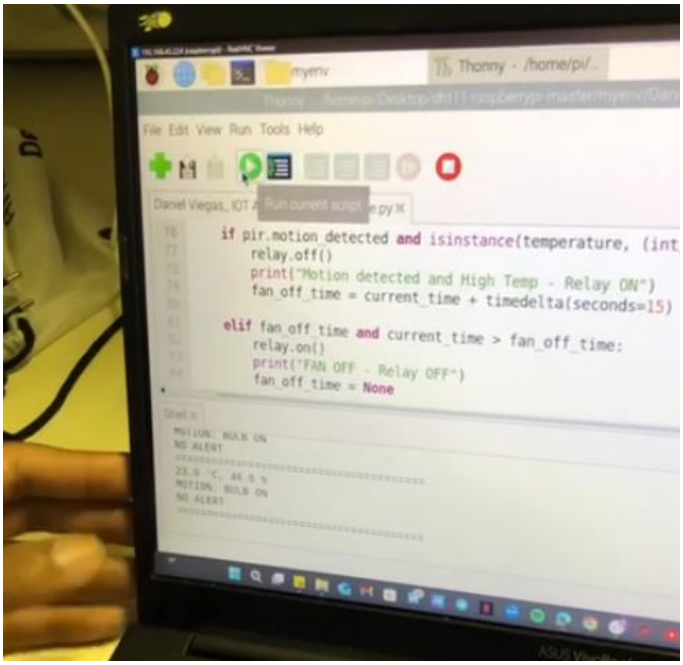

Figure 8: Serial Monitor of Thonny

When the system ran, the sensors and actuators' readings were displayed on the serial monitor of Thonny on the raspberry pi.

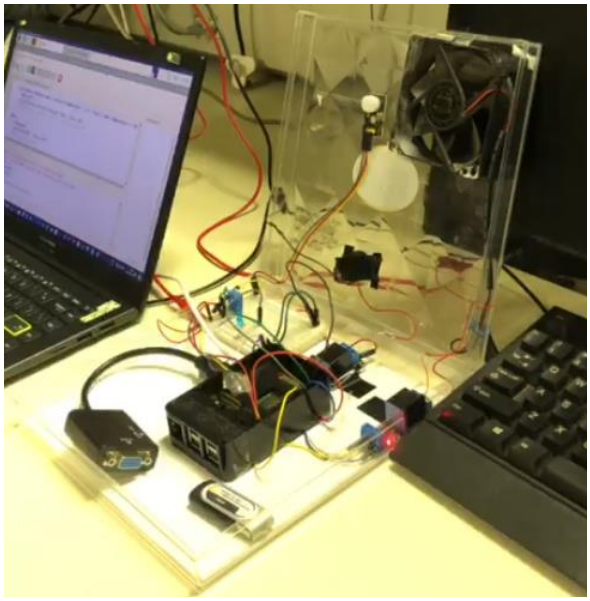

Figure 9: System OFF

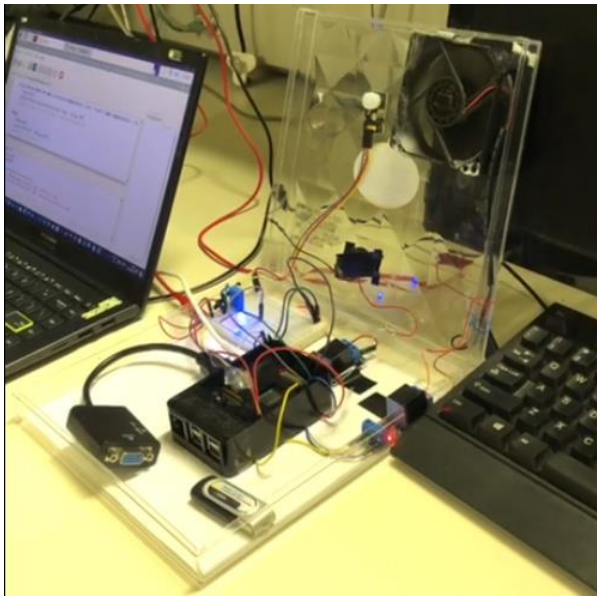

Figure 10: System ON

Initially since there is no motion detected in the range of the PIR sensor the set conditions were not triggered, therefore the system remains OFF as shown in figure 8.

When motion was detected in the range of the PIR sensor, the LED was turned on as shown in figure 9. The fan has a condition that when motion is detected and the temperature is above 25 degrees Celsius the fan turns, when one of those conditions were not met the fan turns OFF.

```
# Define Time
night = datetime.strptime('03:00', '%H:%M').time()
day1 = datetime.strptime('06:00', '%H:%M').time()
dif= datetime.strptime('05:00', '%H:%M').time()
```

Conditions for the Alarm:

```
if (night <= c_time < dif and pir.motion_detected) or (c_time <= day1 < dif and pir.motion_detected):
    relay2.on()
    send_telegram_message('Intruder Alert')
    print('BUZZER ON ALERT!!!')
else:
    relay2.off()
    print('NO ALERT')
```

When testing the alarm system, the time range was set to the range of the presentation to allow this feature to be tested. When the PIR detected motion, the buzzer turned ON, when no motion was detected, the buzzer remained OFF.

8. Future Improvements

The project can and will be further developed after this course to add new features. The first enhancement will be a logging system that records activities, such as the frequency and duration of use, to cloud storage. This data can be accessed from the cloud and transferred to an Excel spreadsheet and plotted, which can be downloaded to devices like mobile phones. The second improvement will be a feature that alerts users when power consumption surpasses a predefined limit, identifies which part of the system is using the most power, and provides recommendations for reducing consumption. The third enhancement will focus on the security system by adding a camera that activates when the alarm is triggered. The camera will record footage and send it via the Telegram bot.

9. Design Demonstration Link

https://drive.google.com/drive/folders/1QNmR_TI28V0rQsoP-5uZXlCvRHU9y2V?usp=drive_link

10. Conclusion

The report outlined the progression of this project from an initial proposal to a fully operational build, which was set up and demonstrated in the laboratory. Several modifications were made from the original proposal: the step-up DC to DC module was replaced with a DC power supply because the 12V Fan and 6V Buzzer are current-controlled device, and the Bulb was substituted with a LED. The design steps were clearly explained, and with the accompanying video demonstration, this project can be easily replicated by engineers and technicians skilled in operating a Raspberry Pi. The implementation of the combined circuit in the laboratory, featuring both autonomous and online override modes, functioned without a fault. The design project was a complete success, meeting all objectives comprehensively.

11. Challenges and Recommendations

11.1. Challenges

During the course of this project a few challenges were encountered, some proved to be quite challenging which then delayed the completion of this project. These challenges were:

- Integrating the DHT11 sensor with the raspberry pi 3B, the micro controller required various downloads of libraries which most were still not recognised, hence the code would not run.
- The raspberry pi was a bit older hence its memory card was deteriorating in functionality. This caused the system to run slower than usual.
- Creating the Web-App required additional knowledge of HTML and Java Script, which increased the complexity of this project.
- The Web-App had a few buttons bouncing which caused a not so user-friendly web-app interface.
- One of the relays were negatively triggered which required greater focus in the circuitry.

11.2. Recommendations

To successfully avoid or tackle the above challenges; the following the recommendations should be considered:

- Ensure your raspberry pi's memory card is at its best functionality with the latest raspberry pi OS. This will increase the speed of operation of the micro controller.
- When interfacing a certain sensor with the raspberry pi, read about the sensor's required libraries.
- Make use of online resources to debug and improve the Web-App's appearance and functionality.

- Implement a debounce mechanism within the server-side code by adding a short delay and state-checking logic to ensure that rapid, consecutive button presses are not processed multiple times.
- To prevent unforeseen issues during the construction of the system's circuitry, it is essential to thoroughly research all components involved in the system prior to their purchase.

12. References

Adeoye, O. A. (2022). Smart solutions for energy efficiency and security in African workspaces. *African Journal of Technology*, 8(1), 50-65.

Gus. (2022, October 27). *Raspberry Pi Motion Sensor using a PIR Sensor*. Pi My Life Up. https://pimylifeup.com/raspberry-pi-motion-sensor/

Islam, M. R. (2020). Challenges of energy efficiency in African countries. *International Journal of Energy Economics and Policy*, 10(2), 200-215.

Obi, N. K. (2019). Remote monitoring technology for sustainable development in Africa. *Journal of Sustainable Development*, 25(3), 300-315.

Smith, A. B. (2021). Enhancing physical security measures in African office spaces. *Journal of African Security Studies*, 15(4), 450-465.

Raspberry Pi - Cooling System using DHT Sensor | Tutorials for Newbies. (n.d.). Tutorials for Newbies. https://newbiely.com/tutorials/raspberry-pi/raspberry-pi-cooling-system-using-dht-sensor

13. Annexes

Program Code:

```
import RPi.GPIO as GPIO
import Adafruit_DHT
from gpiozero import MotionSensor, LED
from time import sleep
from datetime import datetime, timedelta
from http.server import BaseHTTPRequestHandler, HTTPServer
import threading
import requests

# Initialize GPIO for DHT11 sensor
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

# Define the DHT sensor type and the GPIO pin it's connected to
DHT_SENSOR = Adafruit_DHT.DHT11
DHT_PIN = 21

# Initialize GPIO for PIR sensor and relay
pir = MotionSensor(20)
relay = LED(17)
relay1 = LED(14)
relay2 = LED(26)

# Define Time
night = datetime.strptime('03:00', '%H:%M').time()
day1 = datetime.strptime('06:00', '%H:%M').time()
dif= datetime.strptime('05:00', '%H:%M').time()

# Global variables to store sensor data
temperature = 0
humidity = 0
motion_detected = False

# Global flags for manual control
manual_control_relay = False
manual_control_relay1 = False
manual_control_relay2 = False
manual_mode = False  # Flag for manual mode

# Timers for relay off delay
fan_off_time = None
bulb_off_time = None

# Telegram bot token and chat ID
TELEGRAM_BOT_TOKEN = '7370983900:AAFxNCpwfyVE_tJEYEjw7PRmrsvzc9KSpv8'
TELEGRAM_CHAT_ID = '1716216248'

# Set up server host and port
host_name = '192.168.43.224'  # Replace with your Raspberry Pi's IP address
host_port = 8000

def send_telegram_message(message):
    url = f'https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendMessage'
    data = {'chat_id': TELEGRAM_CHAT_ID, 'text': message}
    requests.post(url, data=data)

def sensor_loop():
    global temperature, humidity, motion_detected, fan_off_time, bulb_off_time
    while True:
        c_time = datetime.now().time()

        # Read temperature and humidity
        humidity, temperature = Adafruit_DHT.read(DHT_SENSOR, DHT_PIN)
        if humidity is not None and temperature is not None:
            temperature = round(temperature, 1)
            humidity = round(humidity, 1)
            print(f"{temperature}°C, {humidity}%")
        else:
            temperature = 'N/A'
            humidity = 'N/A'

        # Motion detection and temperature control
        if not manual_mode:
            current_time = datetime.now()
            if pir.motion_detected and isinstance(temperature, (int, float)) and temperature > 20:
```

```
            relay.off()
            print("Motion detected and High Temp - Relay ON")
            #fan_off_time = current_time + timedelta(seconds=0)

        else:
            relay.on()
            print("FAN OFF - Relay OFF")
            #fan_off_time = None

        if pir.motion_detected:
            relay1.on()
            print('MOTION: BULB ON')
            #bulb_off_time = current_time + timedelta(seconds=0)

        else:
            relay1.off()
            print('NO MOTION: BULB OFF')
            #bulb_off_time = None

        if (night <= c_time < dif and pir.motion_detected) or (c_time <= day1 < dif and pir.motion_detected):
            relay2.on()
            send_telegram_message('Intruder Alert')
            print('BUZZER ON ALERT!!!')
        else:
            relay2.off()
            print('NO ALERT')

    print('========================================')
    sleep(4)

class RequestHandler(BaseHTTPRequestHandler):
  def do_HEAD(self):
    self.send_response(200)
    self.send_header('Content-type', 'text/html')
    self.end_headers()

  def _redirect(self, path):
    self.send_response(303)
    self.send_header('Content-type', 'text/html')
    self.send_header('Location', path)
    self.end_headers()

  def do_GET(self):
    global temperature, humidity, motion_detected

    # HTML content
    html = f'''
    <!DOCTYPE html>
    <html>
    <head>
      <title>IOT AUTOMATED WORKSPACE</title>
      <style>
        body {{
          font-family: 'Open Sans', sans-serif;
          text-align: center;
          background-color: #001f3f;
          color: #ffffff;
        }}
        h1 {{
          font-family: 'Aptos', Aptos Black;
          color: #ffffff;
        }}
        p {{
          color: #ffffff;
        }}
        form {{
          background-color: #111111;
          padding: 20px;
          border-radius: 10px;
          margin-bottom: 20px;
          color: #ffffff;
        }}
        input[type="checkbox"] {{
          display: none;
        }}
        .switch {{
          position: relative;
          display: inline-block;
          width: 50px;
          height: 25px;
        }}
```

```
        .slider {{
          position: absolute;
          cursor: pointer;
          top: 0;
          left: 0;
          right: 0;
          bottom: 0;
          background-color: #ccc;
          transition: .4s;
          border-radius: 25px;
        }}
        .slider:before {{
          position: absolute;
          content: "";
          height: 20px;
          width: 20px;
          border-radius: 50%;
          left: 5px;
          bottom: 2.5px;
          background-color: white;
          transition: .4s;
        }}
        input:checked + .slider {{
          background-color: #2196F3;
        }}
        input:checked + .slider:before {{
          transform: translateX(25px);
        }}
        .indicator-on {{
          color: red;
          font-weight: bold;
        }}
        .indicator-off {{
          color: green;
          font-weight: bold;
        }}
        .refresh {{
          margin-top: 20px;
        }}
        .button {{
          display: inline-block;
          padding: 10px 20px;
          font-size: 16px;
          cursor: pointer;
          text-align: center;
          text-decoration: none;
          outline: none;
          color: #fff;
          background-color: #2196F3;
          border: none;
          border-radius: 15px;
          box-shadow: 0 9px #999;
        }}
        .button:hover {{
          background-color: #3e8e41
        }}
        .button:active {{
          background-color: #3e8e41;
          box-shadow: 0 5px #666;
          transform: translateY(4px);
        }}
      </style>
      <link href="https://fonts.googleapis.com/css2?family=Open+Sans&family=Lobster&display=swap" rel="stylesheet">
      <meta http-equiv="refresh" content="5">
      <script>
        function toggleRelay(relay) {{
          var xhr = new XMLHttpRequest();
          xhr.open("POST", "/", true);
          xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
          xhr.send(relay + "=toggle");
        }}
        function resetSystem() {{
          var xhr = new XMLHttpRequest();
          xhr.open("POST", "/", true);
          xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
          xhr.send("reset_system=reset");
        }}
      </script>
    </head>
    <body>
      <h1>Sensor Reporting WebApp</h1>
```

```
        <h2>Temperature: {temperature} °C <img src="https://img.icons8.com/ios/50/ffffff/temperature.png"/></h2>
        <h2>Humidity: {humidity} % <img src="https://img.icons8.com/ios/50/ffffff/hygrometer.png"/></h2>
        <h2>Motion Status: <span class="{'indicator-on' if motion_detected else 'indicator-off'}">
          {'Motion Detected' if pir.motion_detected else 'No Motion Detected'}</span>
        </h2>
        <form>
          <h3>Control Relays</h3>
          FAN:
          <label class="switch">
            <input type="checkbox" name="relay" value="relay" {'checked' if relay.is_lit else ''} onclick="toggleRelay('relay')">
            <span class="slider"></span>
          </label>
          <br>
          LIGHT:
          <label class="switch">
            <input type="checkbox" name="relay1" value="relay1" {'checked' if relay1.is_lit else ''} onclick="toggleRelay('relay1')">
            <span class="slider"></span>
          </label>
          <br>
          BUZZER:
          <label class="switch">
            <input type="checkbox" name="relay2" value="
            <input type="checkbox" name="relay2" value="relay2" {'checked' if relay2.is_lit else ''} onclick="toggleRelay('relay2')">
            <span class="slider"></span>
          </label>
          <br><br>
          <button class="button" type="button" onclick="resetSystem()">Reset System</button>
        </form>
        <p>Designed by:</p>
        <p>Name: Daniel Paulo Viegas , Student Number: 221004300</p>
      </body>
      </html>'''

      self.do_HEAD()
      self.wfile.write(html.encode('utf-8'))

    def do_POST(self):
      global manual_control_relay, manual_control_relay1, manual_control_relay2, manual_mode

      content_length = int(self.headers['Content-Length'])
      post_data = self.rfile.read(content_length).decode('utf-8')

      if 'relay=toggle' in post_data:
        relay.toggle()
        manual_control_relay = True  # Set manual control flag for relay
        manual_mode = True  # Set manual mode flag
      if 'relay1=toggle' in post_data:
        relay1.toggle()
        manual_control_relay1 = True  # Set manual control flag for relay1
        manual_mode = True  # Set manual mode flag
      if 'relay2=toggle' in post_data:
        relay2.toggle()
        manual_control_relay2 = True  # Set manual control flag for relay2
        manual_mode = True  # Set manual mode flag
      if 'reset_system=reset' in post_data:
        manual_control_relay = False
        manual_control_relay1 = False
        manual_control_relay2 = False
        manual_mode = False  # Reset manual mode flag

      self._redirect('/')

  def run(server_class=HTTPServer, handler_class=RequestHandler):
    server_address = (host_name, host_port)
    httpd = server_class(server_address, handler_class)
    print(f'Starting httpd server on {host_name}:{host_port}')
    httpd.serve_forever()

  if __name__ == '__main__':
    try:
      sensor_thread = threading.Thread(target=sensor_loop)
      sensor_thread.daemon = True
      sensor_thread.start()
      run()
    except KeyboardInterrupt:
      print('Server interrupted, closing...')
      GPIO.cleanup()
```