SAN FRANCISCO STATE UNIVERSITY

Intelligent Room Lighting Control System Using IoT and Home Automation

By:

Saumeek Upadhyay

(supadhyay1@sfsu.edu)

SF State ID: 921438502

Under guidance:

Prof. Xiaorong Zhang

(xrzhang@sfsu.edu)

Table of Contents

Abstract

Electricity is a critical resource, and light, as a form of electricity, holds immense value. In today's world, the conservation of light has become paramount, particularly in regions like the US where lights often remain illuminated throughout the night. This project addresses the optimization of light sources within a room by intelligently controlling both external sunlight and internal LED lighting. Leveraging IoT and Home Automation technologies, the project utilizes Google Home Assistant as its server, running on Debian 11 OS and communicates with various systems MQTT protocol. Each system is connected to Wi-Fi using NodeMCU microcontroller programmed in Arduino IDE, while the Home Assistant is configured using a combination of user-friendly interface and YAML files. Further, I2C and SPI protocol are also implemented to interface with the peripherals of the microcontrollers. To achieve precision in controlling the position of the blinds, a stepper motor has been used. To throw light on the implementation in a real world, an insight is presented by developing a prototype to simulate the room in a box using light sensor, mini blinds, LEDs and an external light source. The objective is to establish a smart system that not only balances the room's brightness but also conserves energy through efficient control mechanisms.

Chapter 1: Introduction

1.1 Introduction

IoT, or the Internet of Things, refers to the interconnected network of everyday physical gadgets, devices, and systems that communicate and exchange data over the internet. These gadgets, often equipped with sensors, actuators, and computing capabilities, are embedded into our environment to collect, transmit, and receive data, enabling them to interact with each other and with humans in various ways.

Small and constrained embedded devices are used to remotely monitor the conditions within home and control the home appliances. In such case, power consumption and network bandwidth become a major concern [1]. In [2], the paper presents an implementation of Energy-aware Smart University focusing on Smart Lighting, Air-conditioning, and Ventilating system, whose scope can be expanded to any electrical appliances. This paper attempts to make a low-cost, energy-efficient system. The proposed solution uses MQTT Client protocol. This research in [3] aims to design the workflow overview of the automatic light system based on sensors, devices, components, and servers. The system achieves the light control manually and automatically. In [4], A setup is established with the connection of devices with complex functionally such as RaspberryPi and NodeMCU (Wi-Fi) to analyze the performance of MQTT when several devices with high request/response ratio are connected to it.

With home automation and smart systems being ubiquitous, network strength and connectivity are a concern. Energy Conservation and Utilization also comes into the picture. In 2019, researchers found that creating a generative AI model called BERT with 110 million parameters consumed the energy of a round-trip transcontinental flight for one person. Researchers estimated that creating GPT-3, for example, which has 175 billion parameters, consumed 1,287 megawatt hours of electricity and generated 552 tons of carbon dioxide - the equivalent of 123 gasoline-powered passenger vehicles driven for one year [5]. The comfort of automation should thus be reasonable with energy for sustainability.

1.2 Objective

The objective of this project is to automate and balance the control of light coming in the room and its sources: external and internal. This project implements these using devices including microcontrollers (NodeMCUs), a light sensor, the Google Home Assistant Server and Wi-Fi network protocol and MQTT communication protocol. The system thus designed consists of 4 subsystems: One to monitor the brightness of the room (light sensor and NodeMCU), one to manage the communication of this data between each of the systems (Home Assistant), another to control the external lighting coming in (motor system with NodeMCU), and one for controlling the internal lighting source (LED system with NodeMCU).

Chapter 2: Design And Implementation

This section explains the design, workflow and hardware module working of the project. A prototype of the proposed system is shown in Fig. 1.

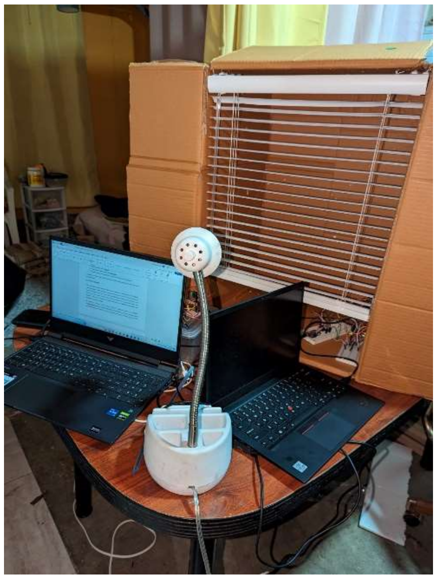


Fig. 1: Prototype of IoT based Smart Room Lighting Control system

2.1 System Architecture

The system further consists of 3 subsystems as evident in the Block Diagram in Fig. 2.

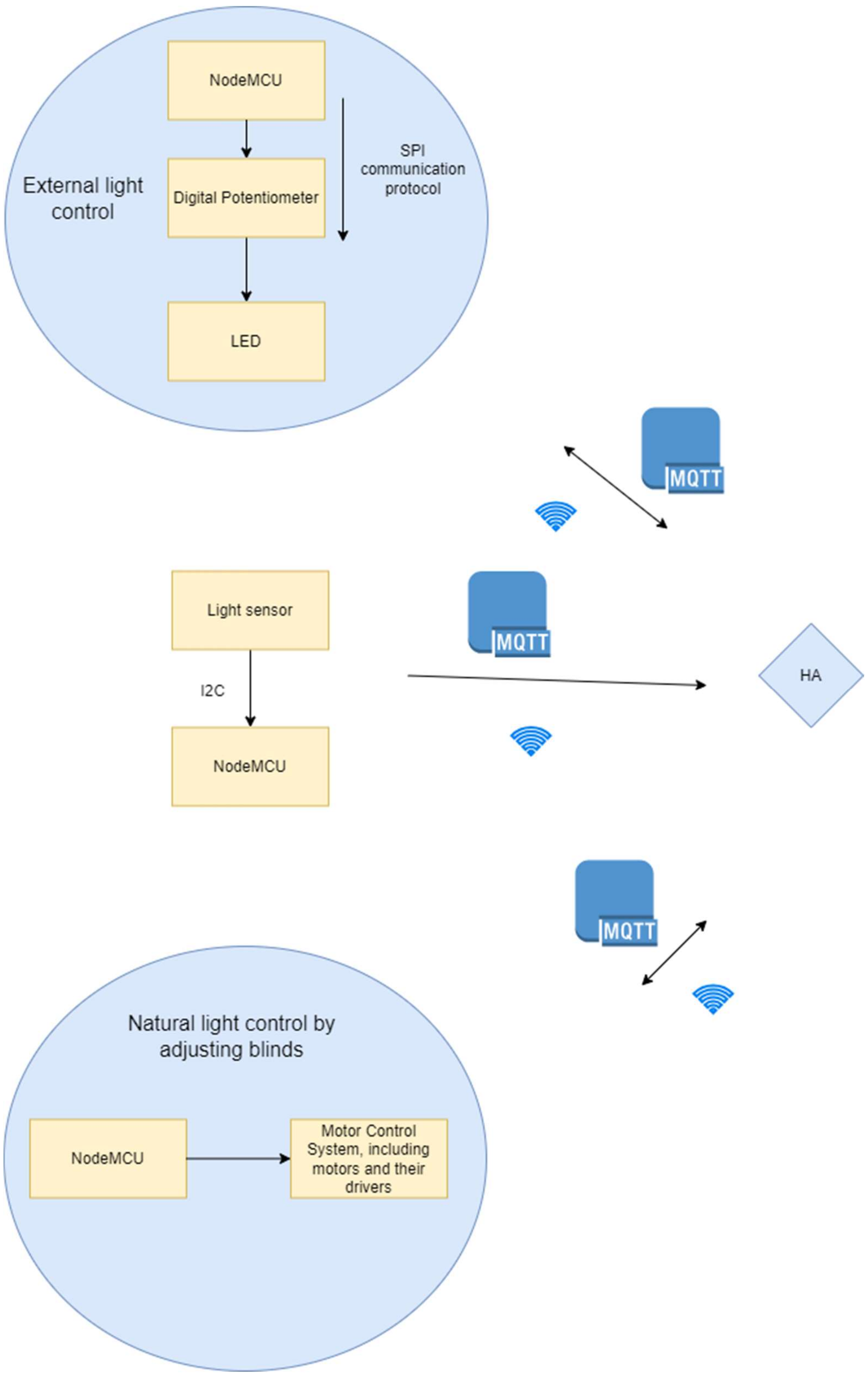

External light control
NodeMCU
Digital Potentiometer
LED
SPI communication protocol
MQTT
Light sensor
I2C
NodeMCU
MQTT
HA
MQTT
Natural light control by adjusting blinds
NodeMCU
Motor Control System, including motors and their drivers

Fig. 2: Block Diagram of the system

2.1.1 Light sensor and NodeMCU system:

Each of the subsystems and their components have been described here:

<u>**NodeMCU**</u>:

The NodeMCU microcontroller serves a source of Wi-Fi as well as a controller for actions needed in all systems it is used. This module receives the data transmitted from the light sensor and sends it to HA. As shown in Fig. 3, NodeMCU has pins that can function as general purpose input/outputs or to function for networking with the peripherals.

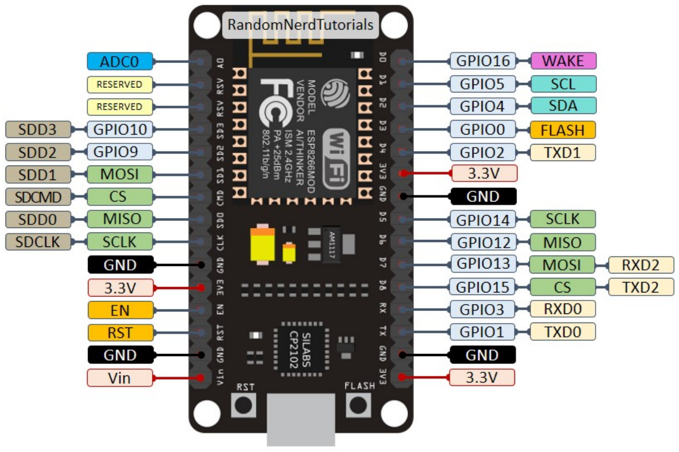


Fig. 3: NodeMCU [6]

A light sensor (TSL2591), as shown in Fig. 4. is interfaced with the NodeMCU using I2C communication protocol.

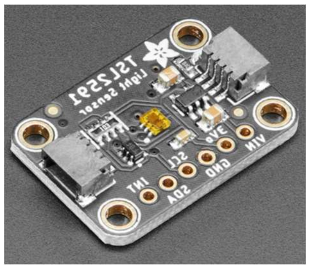


Fig. 4: TSL 2591 [7]

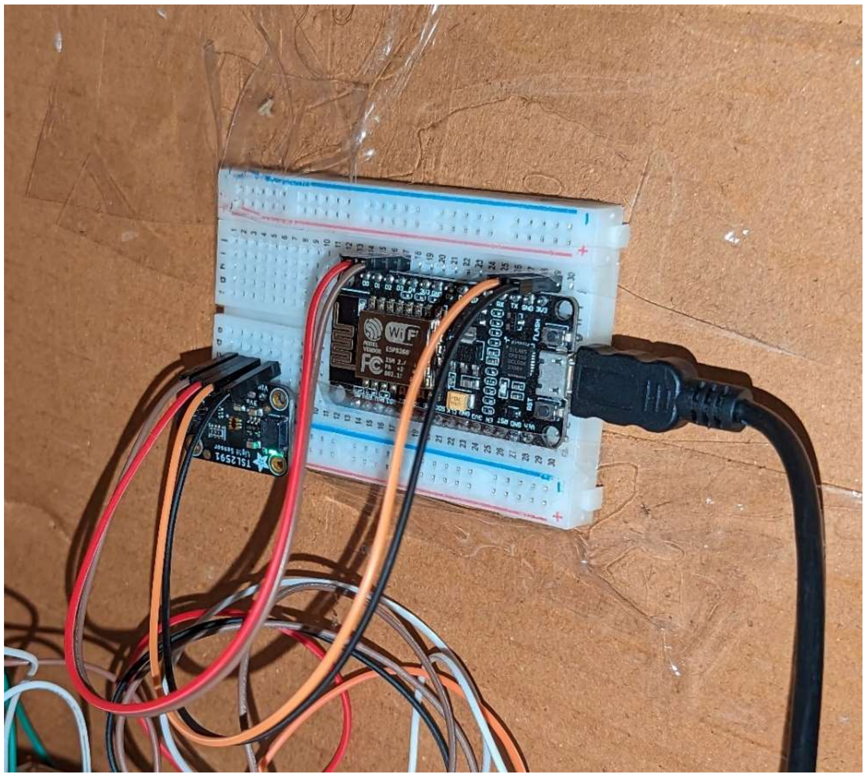


Fig. 5 Light sensor and NodeMCU system

I2C can support a multi-controller system, allowing more than one controller to communicate with all peripheral devices on the bus (although the controller devices can't talk to each other over the bus and must take turns using the bus lines). As per the protocol, a NodeMCU acts as the master and a light sensor acts as a slave. Each I2C bus consists of two signals: SDA and SCL. SDA (Serial Data) is the data signal and SCL (Serial Clock) is the clock signal. The hardware setup is as follows:

- Connect the VCC pin of the light sensor to a 3.3V pin on the NodeMCU.
- Connect the GND pin of the light sensor to the GND pin on the NodeMCU.
- Connect the SDA pin of the light sensor to the D2 (SDA) pin on the NodeMCU.
- Connect the SCL pin of the light sensor to the D1 (SCL) pin on the NodeMCU.

These connections are as shown in Fig. 5.

The software programming was done in Arduino ide. endix

2.1.2 LEDs, Digital Potentiometer and NodeMCU system:

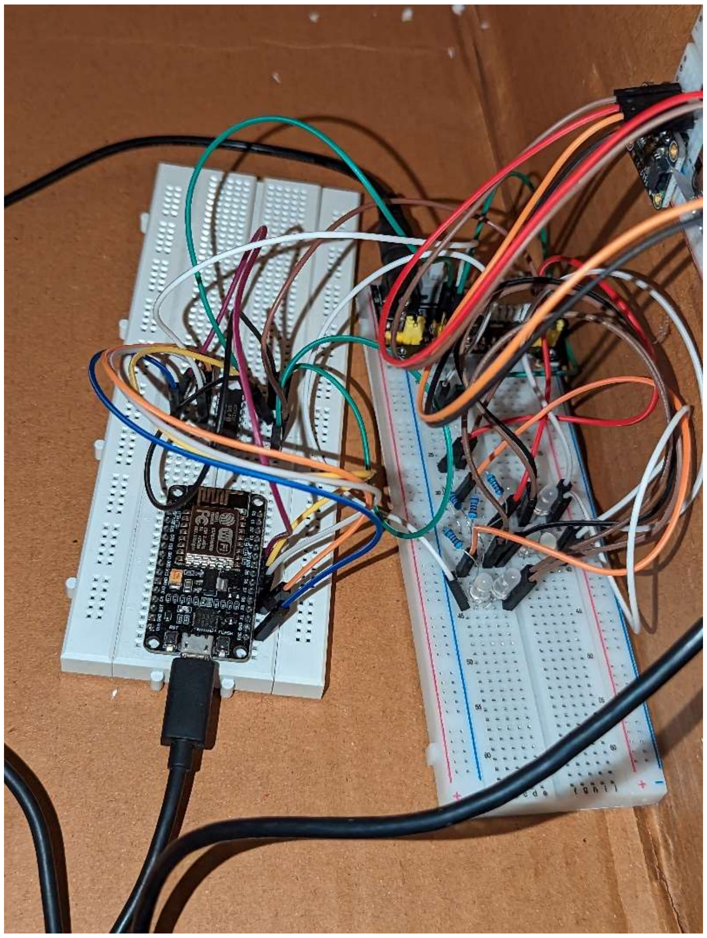


Fig. 6: LEDs, potentiometer and NodeMCU system

An array of 6 LEDs in parallel acts as an internal source of light. A digital potentiometer controls the brightness of the LEDs. The potentiometer is adjusted by programming NodeMCU. The NodeMCU receives data from the HA regarding the lux (room brightness) and takes appropriate action on the potentiometer. The potentiometer, LED and NodeMCU system is shown in Fig. 6.

The potentiometer works on the principle of controlling resistance: Increasing the resistance decreases the LED brightness and decreasing the resistance increases the LED brightness.

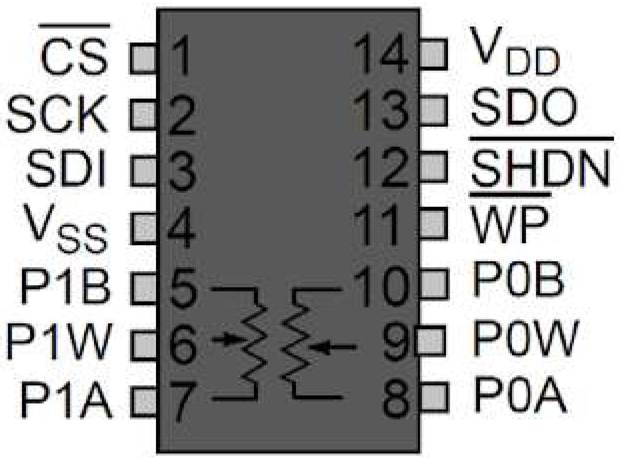


Fig. 7: MC 4231-103 pinout [8]

A multimeter reading full-scale reading of 6.9 kohm was obtained for the potentiometer MC 4231-103 (Fig. 7). The communication between potentiometer and NodeMCU happens using SPI communication protocol. SPI is a synchronous (same clock for Master and Slave), full-duplex (two-way communication on the same carrier at a time) protocol that uses multiple data lines to exchange data between a master and one or more slave devices.

Pin Description:

- Pins P0A, P0W, and P0B: These are the pins for the first digitally controlled potentiometer. The LED array is connected via a current limiting resistor of 110Kohm to the P0W wiper pin 9 of the digital potentiometer.
- Pins P1A, P1W, and P1B: These are the pins for the second digitally controlled potentiometer, here not connected.
- VDD: Connects to 5V supply.
- VSS: Connects to ground.
- CS: CS is the Chip Select pin for the SPI interface and is connected to D8 (CS) pin of NodeMCU

- SDI and SDO: These pins correspond to serial data in and out, respectively (a.k.a. MOSI and MISO). They are connected to the MISO (D6) and MOSI (D7) pins respectively, of NodeMCU.
- SCK: This is the SPI clock line that is connected to D5 (SCLK) clock pin of NodeMCU.
- SHDN: This stand for shut down respectively. When held low, the hardware "disconnects" the wiper from the internal resistor network. The SHDN pin is connected directly to 5V.
- WP: This is Write Protect pin which is internally not connected (NC) pin. One can ignore this pin.

2.1.3 Motor, Blinds and NodeMCU system

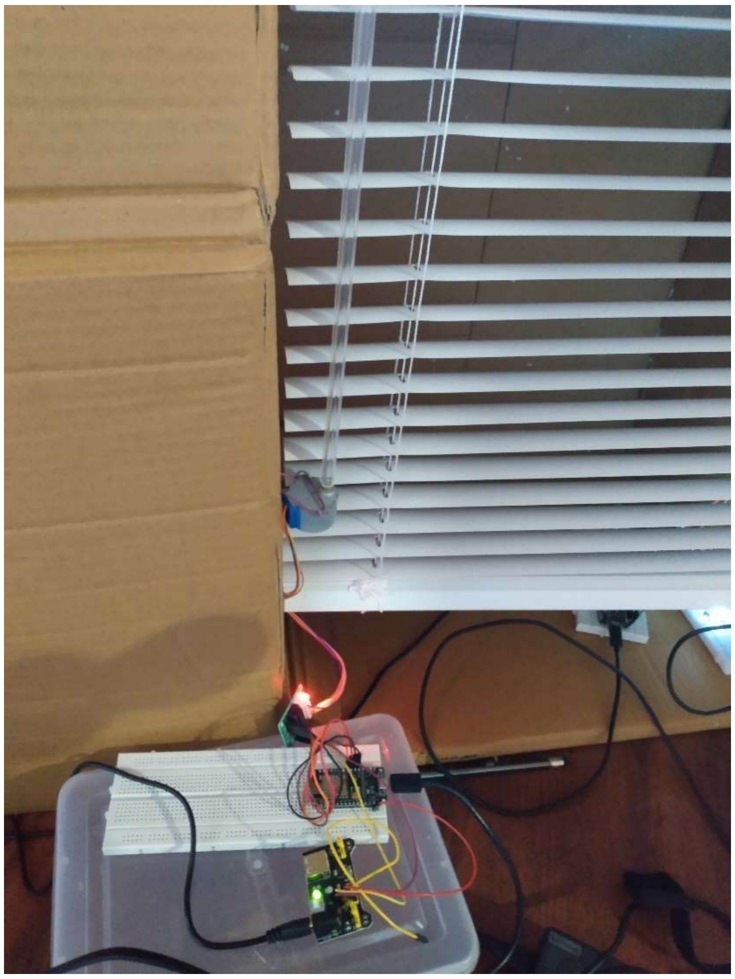


Fig. 8: Motor, Blinds and NodeMCU system

Fig. 8 shows the blinds' wand connected to the shaft of the motor. The motor rotates the wand which in turn opens and closes the blinds. To have precise control over the position of the blinds, a Unipolar Stepper Motor (Fig. 9) and its driver were chosen.

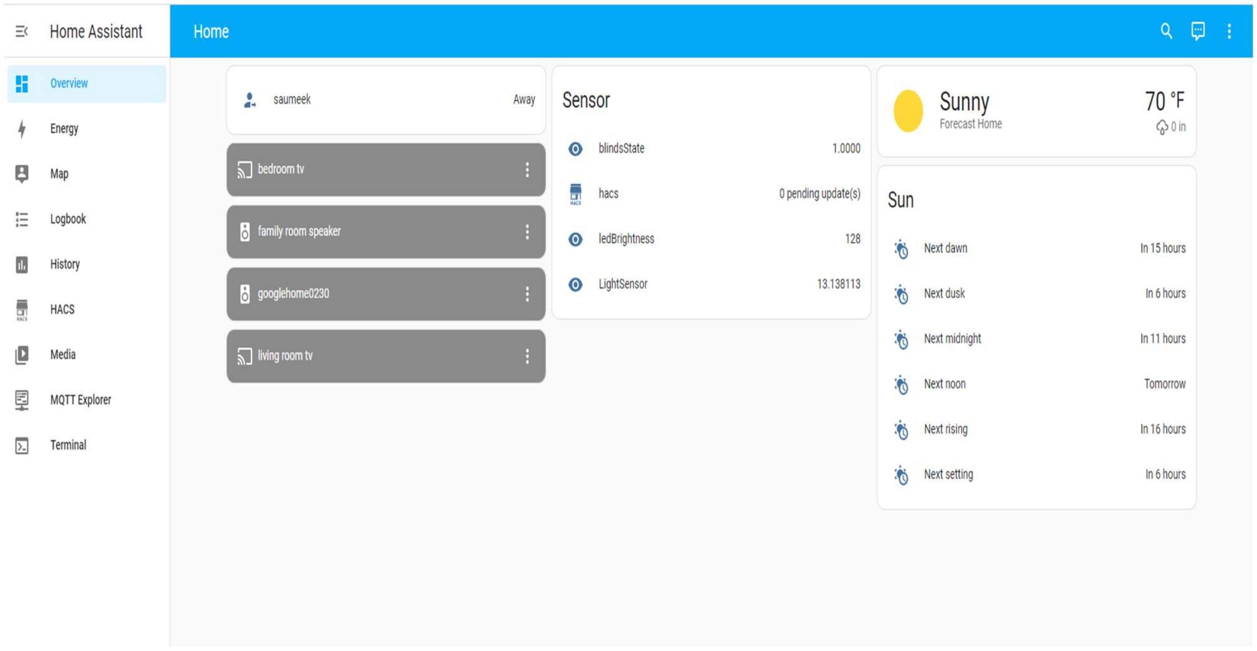
Fig. 9: Stepper Motor and its driver [9]

Pins D8, D7, D6 and D5 of the NodeMCU were connected to IN1, IN2, IN3 and IN4 of the motor.

The NodeMCU receives data regarding the lux values and sends commands to rotate the motor in an appropriate direction with the number of steps.

2.1.4 Google Home Assistant (HA) server:

The Google HA server is a centralized Mosquito MQTT broker for communication between all the NodeMCUs in the project. It sends and receives the required data to and from each NodeMCU. It also stores the state of various variables like the motor position, so that they remain in a known position every time the motor system changes its position (state) or is turned on, in their "sensors". (In HA terminology, a sensor refers to physical or virtual representation of sensor, for monitoring various entities which could, for instance, be, the position of a motor, the battery level of a device or the free space on a laptop). Fig. 10 shows the dashboard of the HA server.



saumeek Away

Sensor

blindsState	1.0000
hacs	0 pending update(s)
ledBrightness	128
LightSensor	13.138113

Sunny 70 °F
Forecast Home 0 in

Sun

Next dawn	In 15 hours
Next dusk	In 6 hours
Next midnight	In 11 hours
Next noon	Tomorrow
Next rising	In 16 hours
Next setting	In 6 hours

Fig. 10: Google Home Assistant Server

2.2 Firmware Design

2.3.1 Arduino IDE

The NodeMCU in each system is programmed in Arduino IDE. We need to specify the Wi-Fi username as password in the IDE for it to connect to the internet. Further, the IP address of the Home Assistant server, the I2C connections on NodeMCU board, the SPI connections and the motor driver connections also need to be specified, The Mosquito MQTT broker name and password are also to be specified. The IDE is very popular and thus has an advantage of numerous repositories and libraries for the microcontrollers it supports.

2.3.2 MQTT protocol

MQTT or message queueing telemetry transport is an extremely lightweight way of exchanging messages between devices making it extremely popular for smart home or other IoT products where bandwidth is more limited. MQTT uses a publish and subscribe model in order to transmit data. There are three main elements to MQTT: an MQTT broker, a client and a topic (Fig. 11).

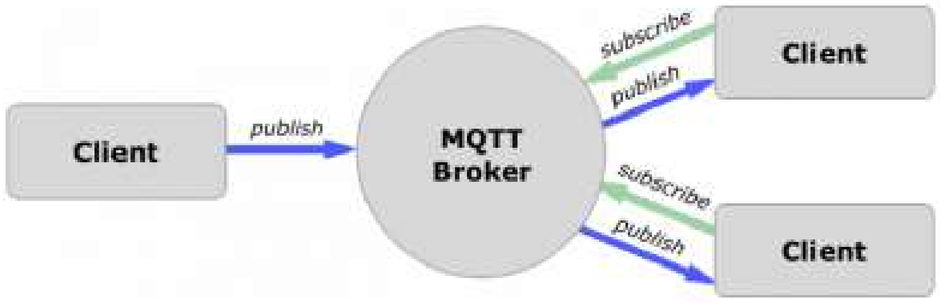


Fig 11. MQTT client and broker connection [10]

A. MQTT Client

A client is usually an endpoint device, and these publish or subscribe to topics that they are interested in to transmit but also receive data. Clients can be subscribed to but also published to multiple topics simultaneously.

B. MQTT Broker

A MQTT broker receives all transmitted messages and it's responsible for routing those messages to the correct clients.

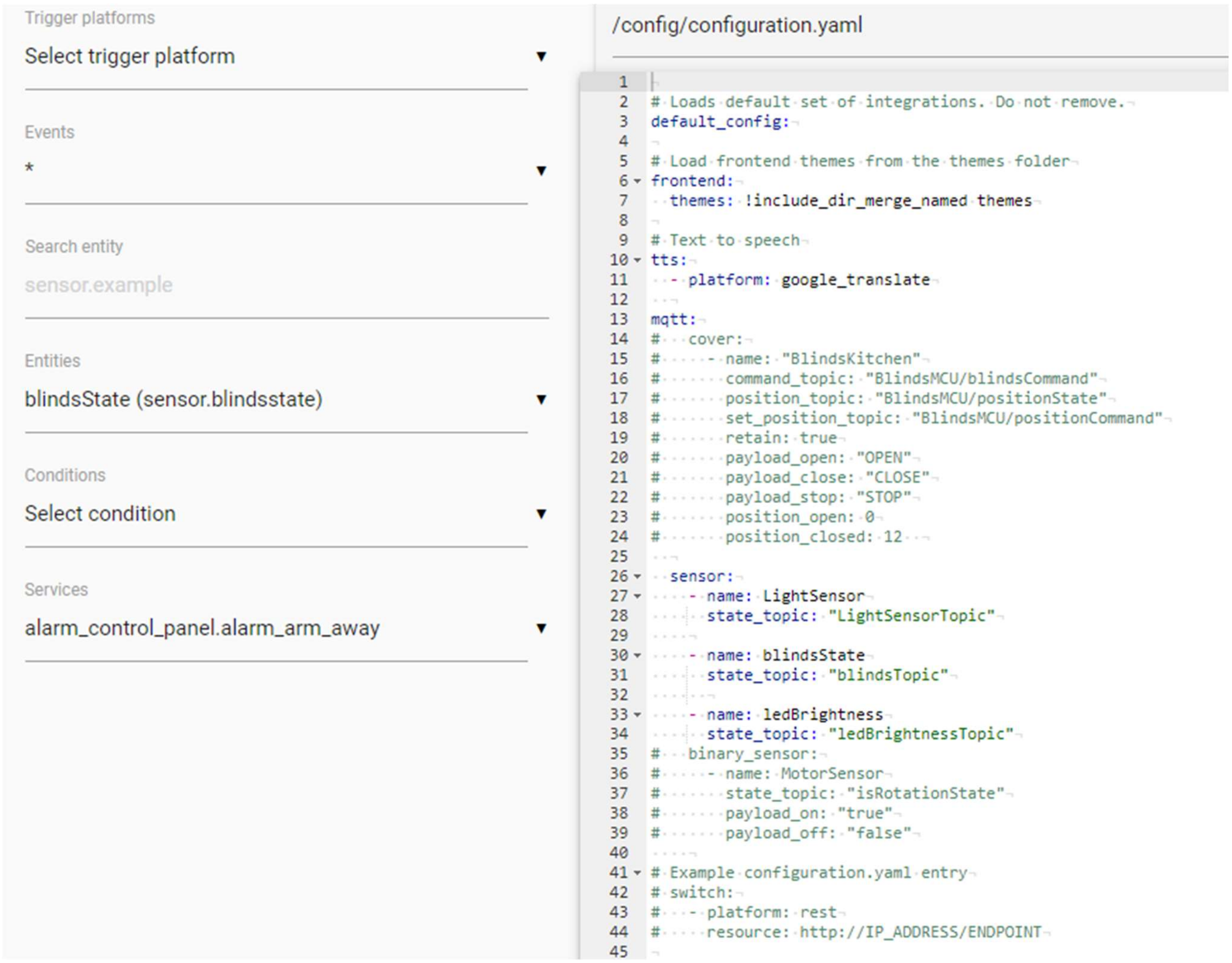
C. Topic

A topic is what the client publishes or subscribes. Any message published to a topic is received by all other subscribers of that topic. A topic can be split up into individual levels in order to better organize them. This is usually done by including a forward slash in the topic name. For example, we could have two devices one light sensor and one light bulb and our light bulb is subscribed to a topic called /bedroom/light. Our light sensor may be programmed such that when it falls below a certain light threshold it will publish an on message to the /bedroom/light topic where it is picked up by the light bulb system and the light bulb will turn on.

Quality of Service (QoS)

There are three levels of QoS: a QoS of level zero is where there is no guarantee that the message will actually be delivered. A QoS of level one is called at least once and this is where the message is guaranteed to be delivered at least once but also it could be received multiple times. A QoS of level two is where the message is guaranteed to be delivered once and only once. QoS of level 0 is generally the fastest one and a QoS of level 2 is the slowest method. However, this speed issue isn't generally a problem until there are much larger networks where there are hundreds or even thousands of devices all talking at once.

2.3.4 Google Home Assistant Web Server



Trigger platforms: Select trigger platform

Events: *

Search entity: sensor.example

Entities: blindsState (sensor.blindsstate)

Conditions: Select condition

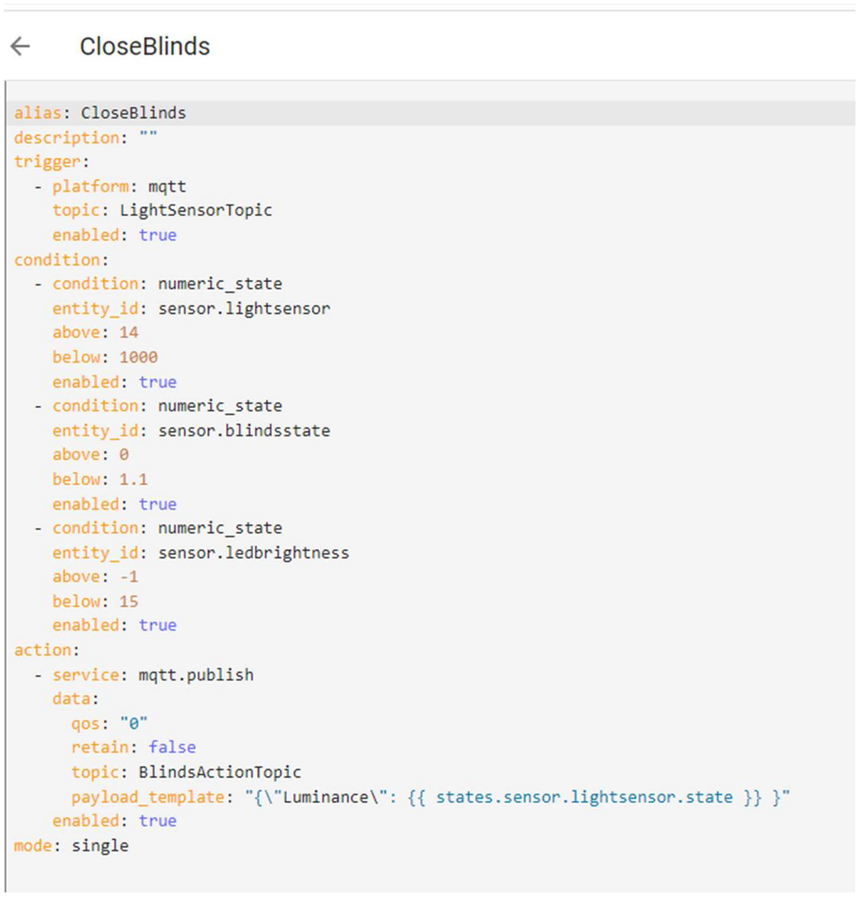
Services: alarm_control_panel.alarm_arm_away

/config/configuration.yaml

```yaml

# Loads default set of integrations. Do not remove.
default_config:

# Load frontend themes from the themes folder
frontend:
  themes: !include_dir_merge_named themes

# Text to speech
tts:
  - platform: google_translate
  
mqtt:
#   cover:
#     - name: "BlindsKitchen"
#       command_topic: "BlindsMCU/blindsCommand"
#       position_topic: "BlindsMCU/positionState"
#       set_position_topic: "BlindsMCU/positionCommand"
#       retain: true
#       payload_open: "OPEN"
#       payload_close: "CLOSE"
#       payload_stop: "STOP"
#       position_open: 0
#       position_closed: 12  
  
  sensor:
    - name: LightSensor
      state_topic: "LightSensorTopic"
    
    - name: blindsState
      state_topic: "blindsTopic"
      
    - name: ledBrightness
      state_topic: "ledBrightnessTopic"
#   binary_sensor:
#     - name: MotorSensor
#       state_topic: "isRotationState"
#       payload_on: "true"
#       payload_off: "false"
    
# Example configuration.yaml entry
# switch:
#   - platform: rest
#     resource: http://IP_ADDRESS/ENDPOINT

```

Fig. 12: configuration.yaml of HA

The Google Home Sever needs to be configured in its YAML file, shown in Fig. 12, before programming it. As seen, the under MQTT, three sensors and the topics to which they have subscribed are listed.



```
alias: CloseBlinds
description: ""
trigger:
  - platform: mqtt
    topic: LightSensorTopic
    enabled: true
condition:
  - condition: numeric_state
    entity_id: sensor.lightsensor
    above: 14
    below: 1000
    enabled: true
  - condition: numeric_state
    entity_id: sensor.blindsstate
    above: 0
    below: 1.1
    enabled: true
  - condition: numeric_state
    entity_id: sensor.ledbrightness
    above: -1
    below: 15
    enabled: true
action:
  - service: mqtt.publish
    data:
      qos: "0"
      retain: false
      topic: BlindsActionTopic
      payload_template: "{\"Luminance\": {{ states.sensor.lightsensor.state }} }"
    enabled: true
mode: single
```

Fig. 13: MQTT implementation of motor rotating backwards (in the direction of blinds closure) using Automation

The MQTT protocol implementation using Automation is done in the YAML files of the respective sub-automation as in CloseBlinds in Fig. 13.

Chapter 3: Workflow

3.1 System Operation Mechanism

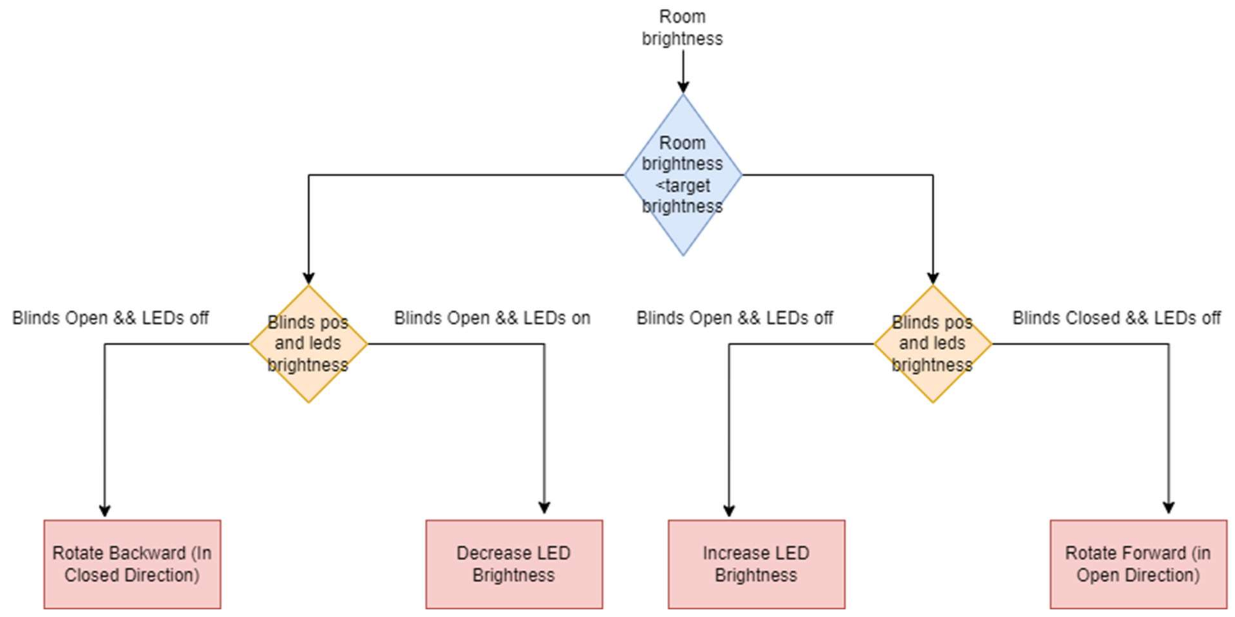
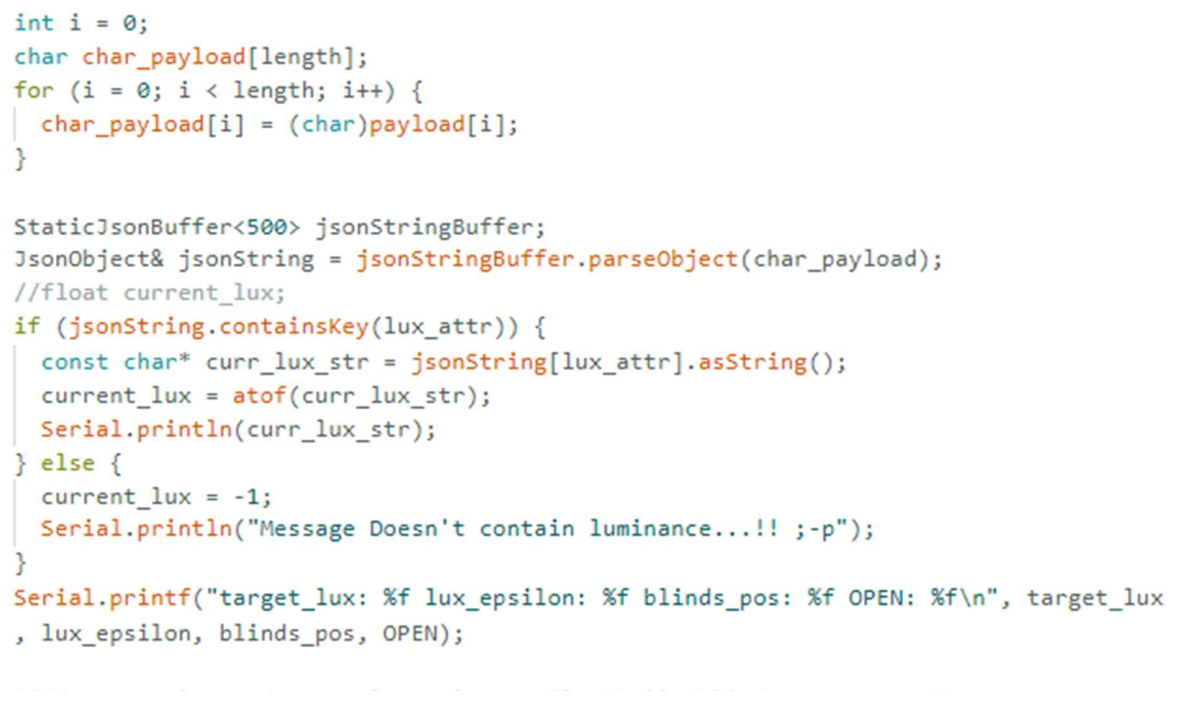

Room brightness
Room brightness <target brightness
Blinds Open && LEDs off
Blinds pos and leds brightness
Blinds Open && LEDs on
Blinds Open && LEDs off
Blinds pos and leds brightness
Blinds Closed && LEDs off
Rotate Backward (In Closed Direction)
Decrease LED Brightness
Increase LED Brightness
Rotate Forward (in Open Direction)

Fig. 14 Flowchart describing the workflow

The flowchart of high-level system operation is shown in Fig. 14. The Light Sensor publishes its readings to a topic to which the HA broker has also subscribed. If the room brightness is different than user's desired level by a margin so that the room brightness does not change abruptly, the room brightness is published to the topics subscribed by the respective systems (LED system and Motor system), only if the systems are supposed to take some action (change motor position or LED brightness). The actions occur in a specific order: If the room brightness is more than the targeted brightness, the LEDs brightness decreases first and if the LEDs turn off, but the room brightness is still more, the blinds start closing. If the room brightness is less than the targeted brightness, the blinds start opening and if the brightness is still less, then the LEDs increase their brightness. The systems publish their respective states (Motor position and LED brightness) to the topics subscribed by the HA.



```
int i = 0;
char char_payload[length];
for (i = 0; i < length; i++) {
  char_payload[i] = (char)payload[i];
}

StaticJsonBuffer<500> jsonStringBuffer;
JsonObject& jsonString = jsonStringBuffer.parseObject(char_payload);
//float current_lux;
if (jsonString.containsKey(lux_attr)) {
  const char* curr_lux_str = jsonString[lux_attr].asString();
  current_lux = atof(curr_lux_str);
  Serial.println(curr_lux_str);
} else {
  current_lux = -1;
  Serial.println("Message Doesn't contain luminance...!! ;-p");
}
Serial.printf("target_lux: %f lux_epsilon: %f blinds_pos: %f OPEN: %f\n", target_lux
, lux_epsilon, blinds_pos, OPEN);
```

Fig. 15: Code snippet to illustrate NodeMCU algorithm to parse JSON

The data sent by the HA to NodeMCU IDE is in JSON format and hence needs to be parsed to string (Fig. 15).

Chapter 4: Experimental Results And Analysis

4.1 ESP8622 NodeMCU

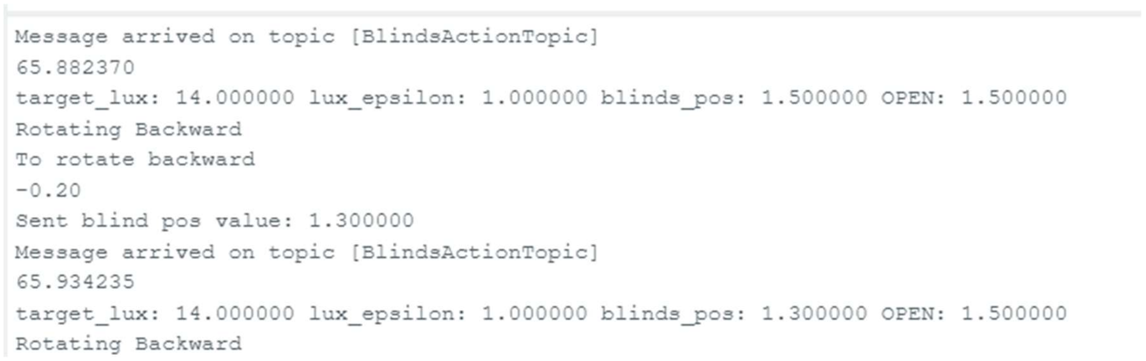
The NodeMCU effectively communicates over Wi-Fi and MQTT with the HA. By programming the NodeMCU, we could remove junk data and stop resetting the NodeMCU of the motor system which happened after every one or two readings.

4.2 Light Sensor

The light sensor readings are printed in the serial monitor of the COM port to which its NodeMCU is connected. The LED readings are published to “LedSensorTopic”, which is also subscribed to by the HA.

4.3 Stepper Motor and Driver System

The values of the motor position and the blinds position are also visible on the serial monitor of the program (Fig. 16) which is uploaded in its NodeMCU. 1.5 is the maximum position of the blinds, at which they are Open, while they are closed at 0. lux_epsilon is the small margin by which the light sensor readings should change for the system to take action to avoid abrupt changes in the room brightness. The motor rotates forward or backward in steps such that the blinds position is changed by 0.2. Target lux is the desired brightness.

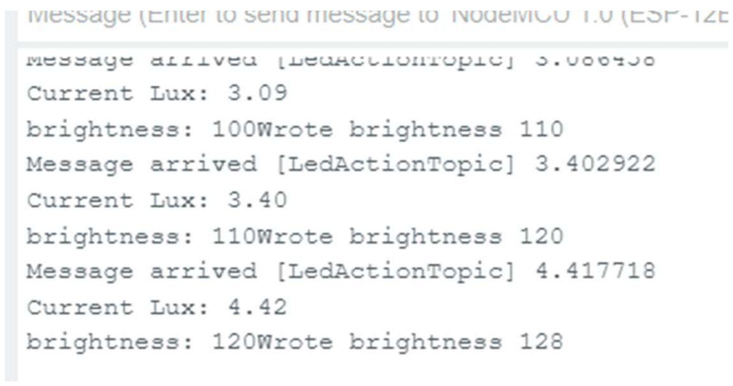


```
Message arrived on topic [BlindsActionTopic]
65.882370
target_lux: 14.000000 lux_epsilon: 1.000000 blinds_pos: 1.500000 OPEN: 1.500000
Rotating Backward
To rotate backward
-0.20
Sent blind pos value: 1.300000
Message arrived on topic [BlindsActionTopic]
65.934235
target_lux: 14.000000 lux_epsilon: 1.000000 blinds_pos: 1.300000 OPEN: 1.500000
Rotating Backward
```

Fig:16 Serial Monitor readings of Motor Blinds system

4.4 LED and Potentiometer System

The LEDs brightness is also reported to the Serial Monitor (Fig. 17) of the COM port to which its NodeMCU is connected. The brightness of the LEDs can be between 0(min) and 128(max). The room brightness readings are published to "LedActiontTopic", the topic to which its NodeMCU has subscribed, only if the system is supposed to take action.

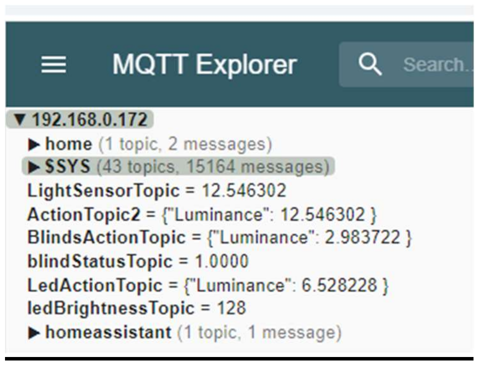


```
Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E
Message arrived [LedActionTopic] 3.086436
Current Lux: 3.09
brightness: 100Wrote brightness 110
Message arrived [LedActionTopic] 3.402922
Current Lux: 3.40
brightness: 110Wrote brightness 120
Message arrived [LedActionTopic] 4.417718
Current Lux: 4.42
brightness: 120Wrote brightness 128
```

Fig. 17: LEDs Brightness readings and current lux

4.5 Google Home Assistant:

The data regarding the status of the motor position, LED brightness and room brightness are published to the topics to which HA has subscribed. All topics to which the HA publishes and subscribes report their values in the MQTT Explorer also, as seen in Fig. 18.



```
▼ 192.168.0.172
  ▶ home (1 topic, 2 messages)
  ▶ $SYS (43 topics, 15164 messages)
  LightSensorTopic = 12.546302
  ActionTopic2 = {"Luminance": 12.546302 }
  BlindsActionTopic = {"Luminance": 2.983722 }
  blindStatusTopic = 1.0000
  LedActionTopic = {"Luminance": 6.528228 }
  ledBrightnessTopic = 128
  ▶ homeassistant (1 topic, 1 message)
```

Fig. 18: MQTT Explorer

Chapter 5: Analysis and Discussion

5.1: Numeric values

The LEDs brightness could vary from 0 (min bright) to 128 (max bright).

The maximum position of the blinds is 1.5 (Open) and 0 (closed at 0).

The target lux of the room was set to 14.

The margin by which the room brightness should change (i.e., lux_epsilon) for the blinds or motor system to take action was set to be 1.

The motor changed its steps such that the position of the blinds changed by 0.2.

Following were the automations:

Decrease LEDs Brightness:

When the room brightness is more than the target brightness (14), then only if the blinds are nearly open (Between 0.95 and 1.1), the room brightness is published to "LEDActionTopic" to which the LEDs system has subscribed. Based on this, the LEDs brightness decreases if they are not in their dim state.

CloseBlinds (Rotate Backwards):

When the room brightness is more than the target brightness (14), then only if the blinds are nearly open (Between 0.95 to 1.1) and the LEDs brightness is less(-1 to 15, on scale of 128), the room brightness is published to "BlindsActionTopic". Based on which the decision to rotate the motor backwards (towards blinds closure) is taken, until the target brightness is not met and the blinds are not fully closed.

OpenBlinds (Rotate Forwards):

When the room brightness is less than the target brightness (14), then only if the blinds are partially open or fully open (Between -1 (negative sign only indicates partially open in the other direction: motor takes steps of 0.2 for rotate forward and -0.2 for rotate backward) the room brightness is published to “LedActionTopic”. Based on which the decision to taken, until the target brightness is not met and the blinds are not fully open.

Increase LEDs Brightness:

When the room brightness is less than the target brightness (14), then only if the blinds are nearly open (Between 0.95 and 1.1), the room brightness is published to “LEDActionTopic” to which the LEDs system has subscribed. Based on this, the LEDs brightness increases until they are fully bright.

Interpretation:

The calibration in each of these cases was done to satisfy the prototype requirements and hence are not practical: A box was taken for a room, while 6 LEDs in parallel were the internal source of light which and the maximum brightness emitted was much less than the external source of light (a lamp). But the actual room and the light sources can be setup in a similar manner with the current logic.

Thus, it can be observed that the LEDs turn on only if the external brightness is not sufficient after the blinds are fully open and turn off first when the room brightness is more. Thus, energy is conserved.

The calibration parameters used in the project have proven to be effective in maintaining gradual and energy-efficient adjustments to room brightness. By setting the target lux at 14 and employing a small lux_epsilon margin, the system ensures that any changes in room brightness are addressed without abrupt shifts. This approach aligns with the project's goal of optimizing light sources to conserve energy while providing a comfortable environment. The system's ability to dynamically control LED brightness and blinds' position showcases its adaptability to varying lighting conditions.

5.2: Advantages of integrating Home Assistant (HA)

This system implements the Hub and Spoke model: This system essentially implements a server client model and can be extended to any number of clients.

The need of a centralized MQTT broker is also addressed. All NodeMCUs have different topics to publish and subscribe. In such a scenario, it would be very difficult to co-ordinate the systems using MQTT.

The successful integration of Google Home Assistant as a central MQTT broker offers several advantages to the project. HA acts as a mediator for data exchange between subsystems, promoting seamless communication and efficient coordination. This centralization simplifies the control and monitoring of the entire system, enhancing user experience and usability. The implications extend beyond this project, as the scalable nature of HA enables future expansions with diverse devices and integrations, supporting the growth of smart automation systems.

5.3 Relative Brightness Algorithm

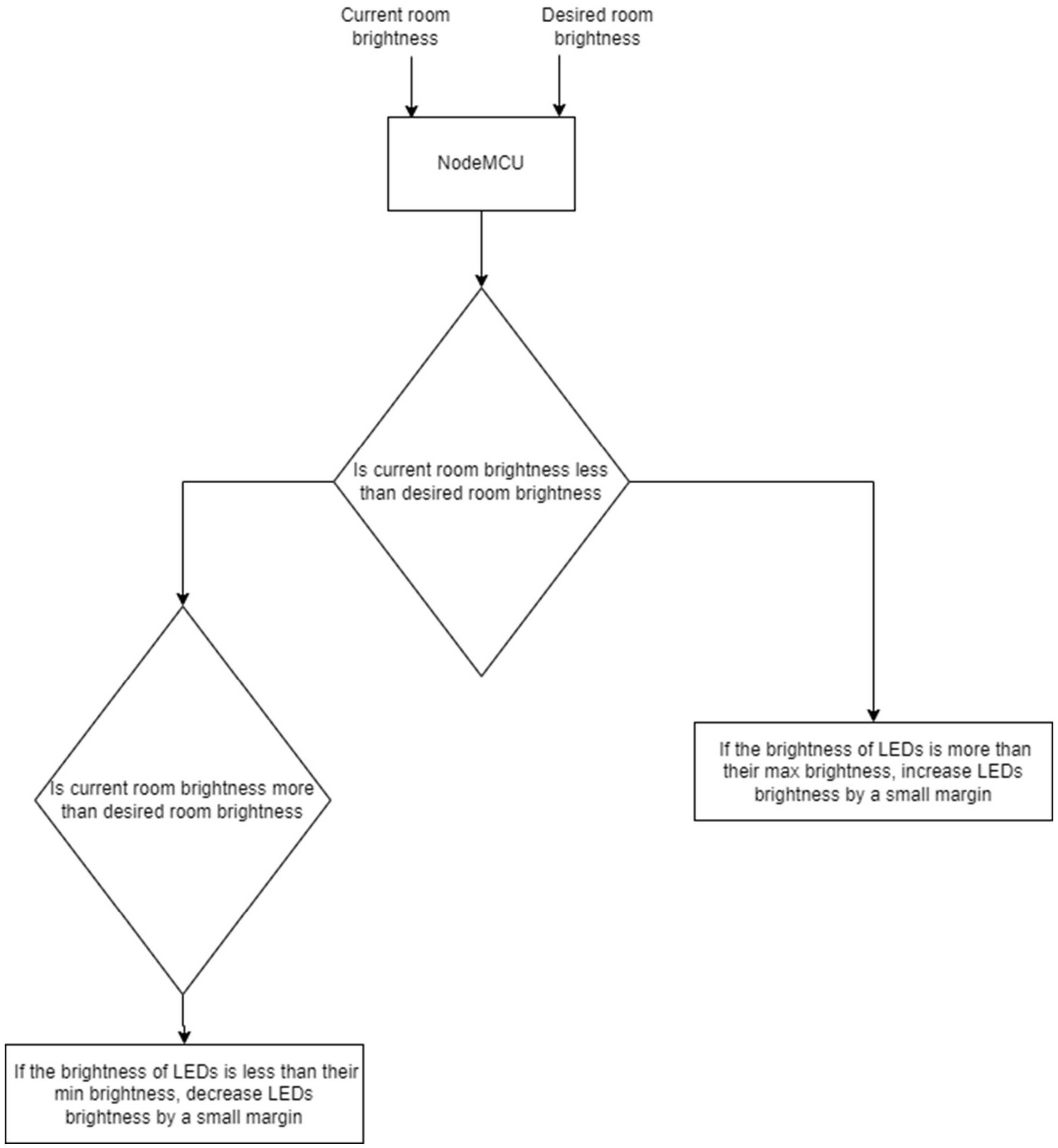

Current room brightness
Desired room brightness
NodeMCU
Is current room brightness less than desired room brightness
Is current room brightness more than desired room brightness
If the brightness of LEDs is more than their max brightness, increase LEDs brightness by a small margin
If the brightness of LEDs is less than their min brightness, decrease LEDs brightness by a small margin

Fig. 19: Relative Brightness Algorithm for the LEDs

The order of execution of the motor system and the led system is such that the room brightness is compensated first by the external light (for instance, sunlight) and then an internal light source. This leads to conservation of electricity.

The LEDs brightness changes or the motor position changes only when the room brightness is different by a margin. This ensures that the brightness of the room does not change abruptly.

The brightness of the room is changed relatively. This eliminates defining the need of an array of absolute values of lux (for e.g., 0 to 1000), as seen in Fig. 17.

The algorithm employed for maintaining relative room brightness has proven effective in achieving the desired balance between external and internal light sources. By adjusting LED brightness and blinds' position based on room brightness data, the system ensures a gradual and natural transition between different lighting conditions. This aligns with the project's energy conservation objective by minimizing sudden changes that could lead to unnecessary energy consumption. The algorithm's success underscores the feasibility of using smart automation for nuanced control of lighting environments.

5.4: Data logging

Every time the state (position) of the motor changes, it is saved by the HA in a 'Motor Sensor' (In HA terminology, a sensor refers to physical or virtual representation of sensor, for monitoring various entities which could, for instance, be, the position of a motor, the battery level of a device or the free space on a laptop). This ensures that the motor rotates just enough to let light in through the blinds, just enough in adjust the room brightness to the user's desired level. It also ensures that the motor does not rotate the blinds more than its (blinds') closed state (position) to avoid breaking the blinds. Thus, Home Automation is manifested.

The incorporation of data logging within the HA server contributes to the project's reliability and long-term functionality. By storing motor positions and other critical states, the system maintains a consistent behavior over time. This is especially important for the blinds' motor, where precise position control is crucial. The implications of data logging extend to future developments, as historical data can be analyzed to optimize control algorithms, identify patterns, and enhance overall system performance.

5.5 Limitations

The system needs to be calibrated when used in a different place, to meet the requirements of the place and the user.

In certain scenarios, the system may exhibit a tendency to overadjust, resulting in unnecessary motor rotations or LED brightness fluctuations. Implementing advanced control strategies, such as predictive modeling or proportional-integral-derivative (PID) controllers, could fine-tune the responsiveness of the system and reduce the risk of overadjustment.

The system's performance is contingent on a stable Wi-Fi network connection and reliable MQTT communication. Redundancy mechanisms, offline caching, or the integration of edge computing could be explored to enhance the system's robustness in the face of network challenges.

The current implementation primarily relies on automated control mechanisms based on predefined thresholds. While this approach aligns with energy conservation, it may limit user interaction and customization. Incorporating a user-friendly interface, such as a smartphone app or voice commands, could empower users to manually override or fine-tune the system's behavior according to their preferences.

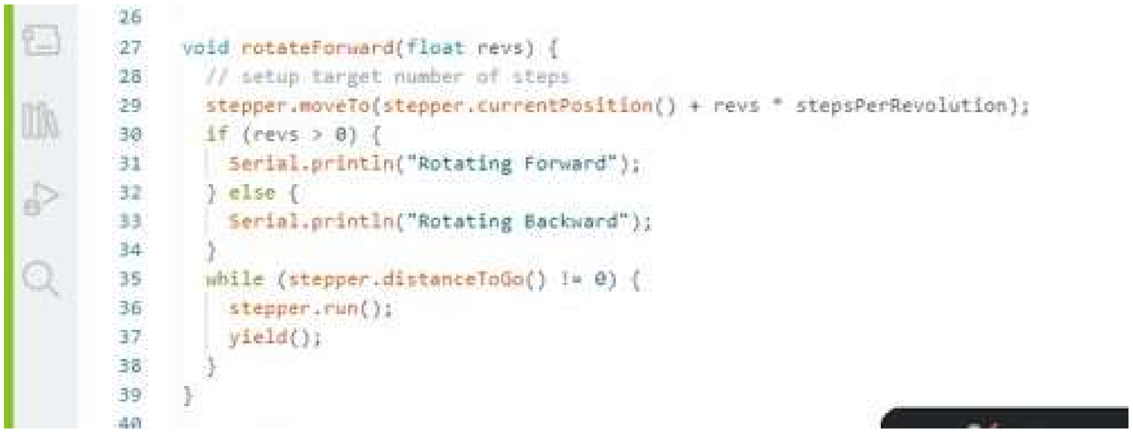
Chapter 6: Troubleshooting

6.1 Handling Cooperative Multitasking

6.1.1 Removing junk data

The serial monitor of the Blinds system printed a lot of junk data. The function yield() was used to remove this.

The **yield()** function is used in programming to temporarily yield control of the execution task to other tasks. It allows other tasks to run while the current task voluntarily stops its execution time. yield() can be called as shown in Fig. 20



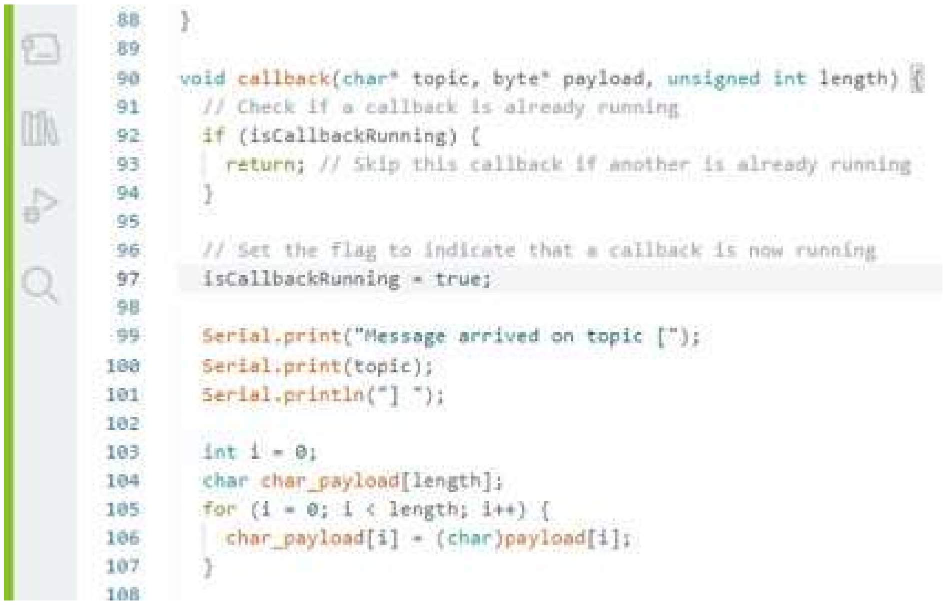
```

void rotateForward(float revs) {
  // setup target number of steps
  stepper.moveTo(stepper.currentPosition() + revs * stepsPerRevolution);
  if (revs > 0) {
    Serial.println("Rotating Forward");
  } else {
    Serial.println("Rotating Backward");
  }
  while (stepper.distanceToGo() != 0) {
    stepper.run();
    yield();
  }
}
```

Fig. 20: Use of yield() to remove junk data

6.1.2 Implementation of mutex

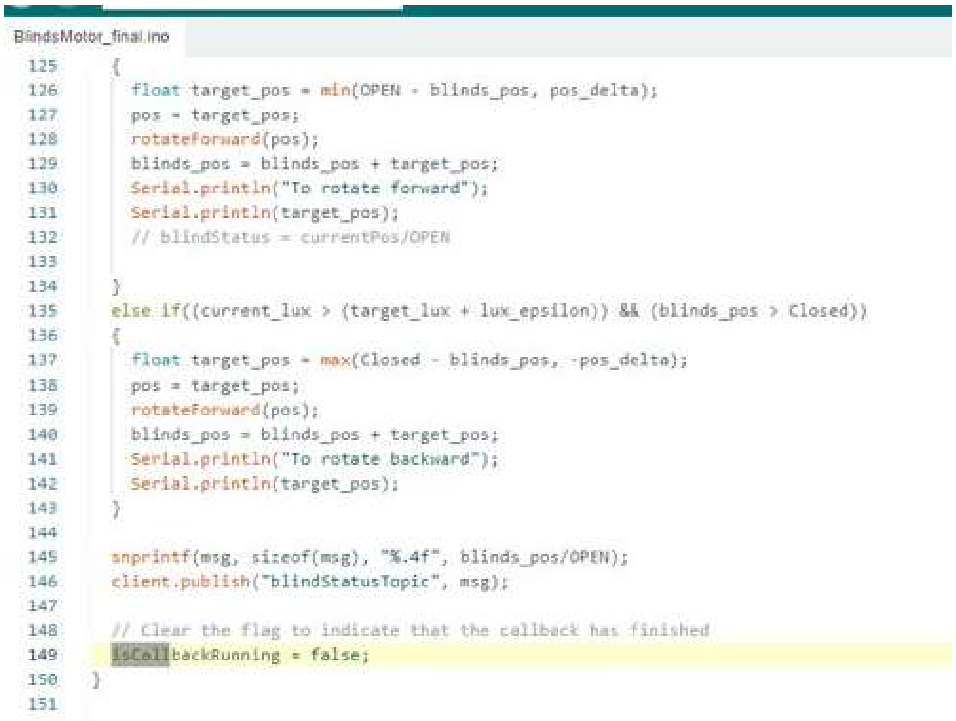
The NodeMCU of the motor system also got disconnected from the Wi-Fi after every one or two readings. Mutex was implemented without using its library because the latter gave conflicts. This is shown in Fig. 21.

Mutex stopped servicing callback requests that occurred in the middle of a callback being serviced.



```
}

void callback(char* topic, byte* payload, unsigned int length) {
  // Check if a callback is already running
  if (isCallbackRunning) {
    return; // Skip this callback if another is already running
  }

  // Set the flag to indicate that a callback is now running
  isCallbackRunning = true;

  Serial.print("Message arrived on topic [");
  Serial.print(topic);
  Serial.println("] ");

  int i = 0;
  char char_payload[length];
  for (i = 0; i < length; i++) {
    char_payload[i] = (char)payload[i];
  }

```

Fig. 21: Mutex implementation using Volatile Variables i.e. without using lib (Because ESP8266 doesn’t support the lib of ESP32 which contained mutex inherently). A flag is set at the start of a callback.



BlindsMotor_final.ino

```
    {
      float target_pos = min(OPEN - blinds_pos, pos_delta);
      pos = target_pos;
      rotateForward(pos);
      blinds_pos = blinds_pos + target_pos;
      Serial.println("To rotate forward");
      Serial.println(target_pos);
      // blindStatus = currentPos/OPEN

    }
    else if((current_lux > (target_lux + lux_epsilon)) && (blinds_pos > Closed))
    {
      float target_pos = max(Closed - blinds_pos, -pos_delta);
      pos = target_pos;
      rotateForward(pos);
      blinds_pos = blinds_pos + target_pos;
      Serial.println("To rotate backward");
      Serial.println(target_pos);
    }

    snprintf(msg, sizeof(msg), "%.4f", blinds_pos/OPEN);
    client.publish("blindStatusTopic", msg);

    // Clear the flag to indicate that the callback has finished
    isCallbackRunning = false;
  }

```

Fig. 22: The flag is reset after the task is complete

As seen in Fig. 22, the flag is reset after the task is complete, for servicing the next callback, if any.

6.2 Technical Challenges

Installing HA Supervisor required Debian 11 Operating System. Debian 11 required ethernet connection to the laptop. Lenovo ThinkPad has a mini ethernet cable. It had to be assigned an IP address, instead of already having one, to install HA Supervisor.

Installing Wi-Fi system on Debian 11 - Debian 11 can access the internet only using ethernet by default. Wi-Fi drivers were to be installed to have access to Wi-Fi.

Chapter 7: Conclusion

The system thus designed operates effectively and precisely to maintain the room brightness without bringing any abrupt changes in it, using MQTT protocol over Wi-Fi. The sub-systems also store their respective states in the HA, which remembers them for next time the system is used. Leveraging the capabilities of Google Home Assistant as a central communication hub enhances the system's efficiency and effectiveness. By programming the system to auto-calibrate, its versatility can be improved for various automations at the industrial and home use.

The project's outcomes hold significant implications for energy conservation and home automation. The ability to dynamically adjust room brightness based on real-time conditions contributes to energy efficiency, reducing unnecessary power consumption.

In the broader context of technological advancements, this project contributes to the growing field of smart home automation and energy management. By combining hardware integration, communication protocols, and centralized control, the project exemplifies the potential of interdisciplinary approaches to solving real-world challenges.

In conclusion, the developed IoT-based light control system showcases the synergy between technology, energy conservation, and user convenience. As we continue to explore innovative solutions for a sustainable future, this project serves as a testament to the power of automation and intelligent control in optimizing resource utilization and enhancing quality of life.

References

[1] P. Sheth, Soumya, A. Lad and Y. Solanki, "Energy Aware IoT based Green Smart University with Automated Lighting and CCTV System using MQTT and MySQL," 2021 2nd Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2021, pp. 1-5, doi: 10.1109/GCAT52182.2021.9587709.

[2] R. K. Kodali and S. Soratkal, "MQTT based home automation system using ESP8266," 2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC), Agra, India, 2016, pp. 1-5, doi: 10.1109/R10-HTC.2016.7906845.

[3] M. Kashyap, V. Sharma and G. Verma, "Implementation and Analysis of IoT Based Automation Using MQTT," 2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON), Kuala Lumpur, Malaysia, 2021, pp. 1-5, doi: 10.1109/GUCON50781.2021.9573857.

[4] Frederick C. Harris Jr, Rui Wu and Alexander Redei (editors). Proceedings of ISCA 30th International Conference on Software Engineering and Data Engineering, vol 77, pages 115--122

[5] "Is Generative AI bad for the environment?," euronews, https://www.euronews.com/next/2023/05/24/chatgpt-what-is-the-carbon-footprint-of-generative-ai-models (accessed Aug. 7, 2023).

[6] S. Santos and S. Santos, "ESP8266 Pinout Reference: Which GPIO pins should you use? | Random Nerd Tutorials," Random Nerd Tutorials, Oct. 2022, [Online]. Available: https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/

[7] Industries, A. Adafruit TSL2591 High Dynamic Range Digital Light Sensor, adafruit industries blog RSS. Available at: https://www.adafruit.com/product/1980 (Accessed: 11 August 2023).

[8] Ee-Diary (2023) How to use digital potentiometer with Arduino, ee. Available at: https://www.ee-diary.com/2023/02/how-to-use-digital-potentiometer-with.html (Accessed: 11 August 2023).

[9] GP (2011) Amazon. Available at: https://www.amazon.com/gp/product/B01IP7IOGQ/ref=as_li_ss_tl?ie=UTF8 (Accessed: 11 August 2023).

[10] Mqtt things and channels - bindings openHAB. Available at: https://www.openhab.org/addons/bindings/mqtt.generic/ (Accessed: 11 August 2023).

[11] "Automated Motorized Window Blinds (Horizontal Blinds) – The Hook Up." https://www.thesmarthomehookup.com/automated-motorized-window-blinds-horizontal-blinds/ (accessed Aug. 07, 2023).