A. Batson (28067), A. Akuri (29630)

Building Appliance Optimization

Time & Energy Saving through better heating and lighting management.

1.0 Introduction

Batson

As the urgency to mitigate and combat climate change grows, more and more solutions are required to foster a more sustainable world. On the radar for these solutions energy conservation and consumption reduction techniques comprise a significant measure in realizing greater sustainability in home & industry (Mahiri et al. 2022). However, one of the greatest limitations in realizing these developments is the human contingent, for example forgetting to turn/off lights and heating. This project targets energy-use optimization by smart control of lights and heating use in a prototype designed to simplify lighting and heating appliances in a building. The applicability of this project is not only limited to building energy use optimization but also vertical farming installations, in which light and temperature control are imperative to control parameters for crop yield. In the end effect, we aim to construct a prototype with a functioning loop, where light, heating, and cooling measures are triggered by appropriate thresholds that 'justify' the use phase. Data visualization and monitoring are also important to us.

The 'brains' of this project will be the ESP32 microcontroller. Its WiFi capabilities are of top interest for our project design, coupled with its I2C capabilities and hardware compatibility, it would be a great project fit.

A Light Dependent Resistor (LDR) photoresistor takes light value data. Light Emitting Diodes (LEDs) will be operated depending on ambient light inputs to automate electricity utilization. A DHT11 sensor works similarly, in taking room temperature data, which determines the switching on/off of a ventilator, used to represent an AC unit.

Our aim is to develop a smart system to fit exactly this niche described above, we expect challenges and hope to account for possible limitations and improvements along the way.

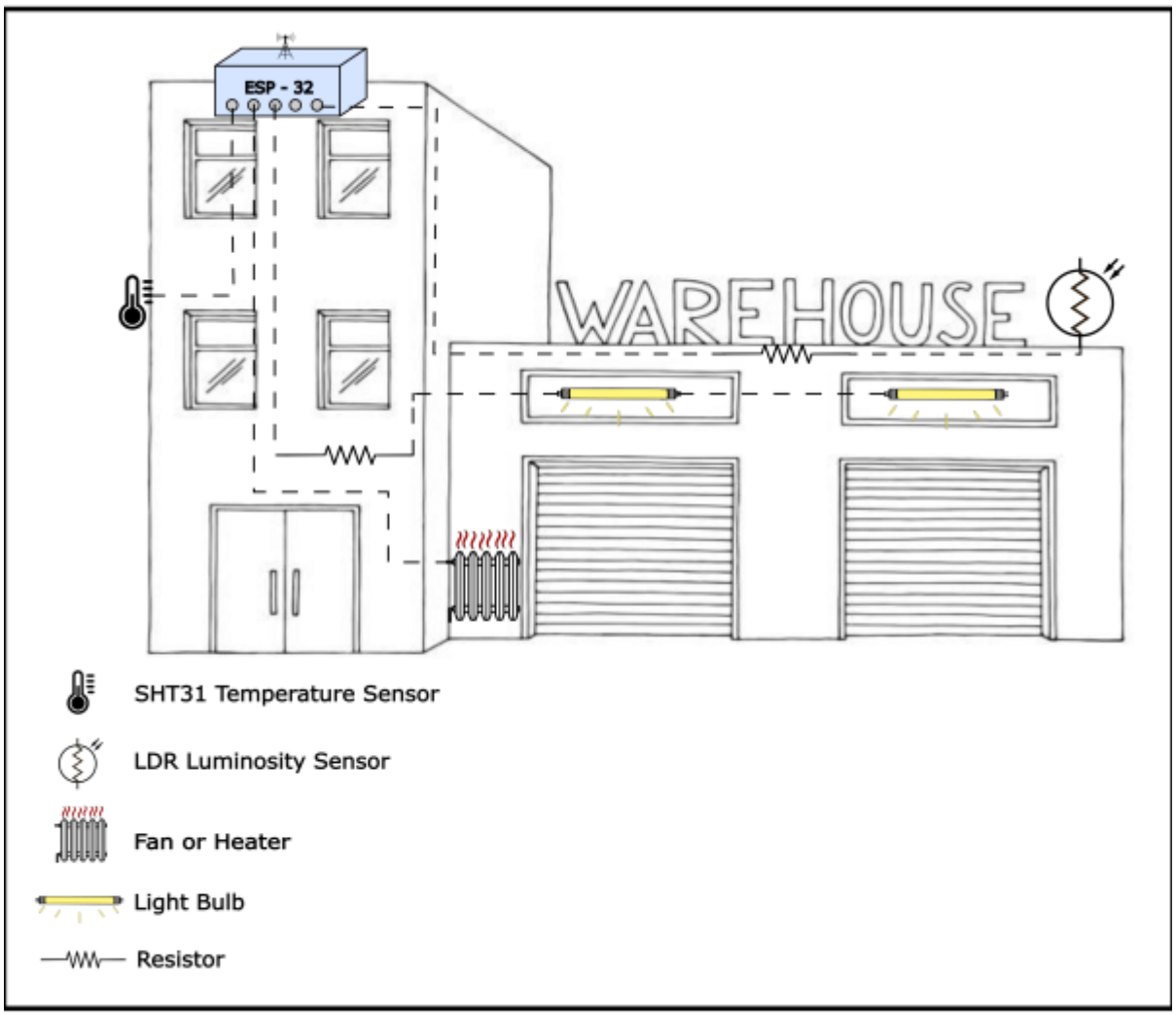


Figure 1	A simplified Overview of the Project (Akuri).

This diagram shows the connection between the microcontroller(esp32), sensors (SHT31 and LDR), and the pathways to their target devices. For conceptualization purposes only.

2.0 Materials & Software

Batson

```
*  Arduino UNO.
*  ESP32 : WROOM-32 & ESPRESSIF WROVER-B.
*  LDR Photoresistor & 10k Ohm resistor.
*  DHT11 Temperature Sensor & SHT Temperature Sensor.
*  Groove Relay & SPDT Relay.
*  LEDs and 220 Ohm resistors.
```

```
*  Connecting cables.

*  MQTT.
*  Node-Red.
*  InfluxDB.
*  Grafana.
```

2.1 ESP32 & Wi-Fi Connection

Batson

|Purpose| The ESP32 is a microcontroller designed for low power IoT applications. It's built-in Wi-Fi capabilities are important for the data transmitting sections of the project. Outside of that it boastm Bluetooth and Deep Sleep Operating abilities.

|Functioning|

The ESP32 packs an outstanding 2.4GHz Wi-Fi and Bluetooth combined system chip which is also designed with ultra low power consumption technology. It has quite a few other interesting benefits for optimum functionability and RF performance. It is robust, versatile and reliable within a wide array of scenarios and uses. Some of the interesting things you can use ESP32 for are building web servers, it comes with classic and also low energy Bluetooth, the LoRa technology, it can work with MQTT and even remote control a car robot among many other things (Espressif Systems, ESP32 Series Datasheet).

Specifications:

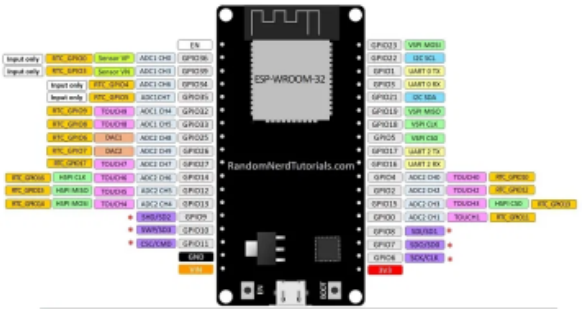


Figure 2 Schematic of the ESP32 DevKit V1 Micro-controller.

For more information on the esp32 visit Here.

2.2 Arduino UNO

Akuri

|Purpose| Arduino is an open-source electronics platform based on easy-to-use software and hardware. Arduino boards are able to read

inputs, i.e. light on a sensor, a finger on a button, or a simple message - and turn it into an output: activating a motor, turning on an LED or publishing something online.

|Functioning| The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet).[14] It contains 6 analog inputs, a 16 MHz ceramic resonator, 14 digital input/output pins (six of which can be used as PWM outputs), a USB port, an ICSP header, a power jack and a reset button. It comes with everything needed to support the microcontroller; to get started you simply need to connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery. **|Applicability|** Even if the Arduino Uno board is accessible to start with and easy to use, the limitation it faces is the 32KB memory which is not a lot of space, therefore it has no memory safety checks [16] but it depends on your application. Nevertheless, we consider it suitable for our project, together with the Arduino board, since it can bear to 20mA. (I/O pins are limited to 40mA before they are damaged, that's why 20mA is the suggested limit to prevent any damage.)

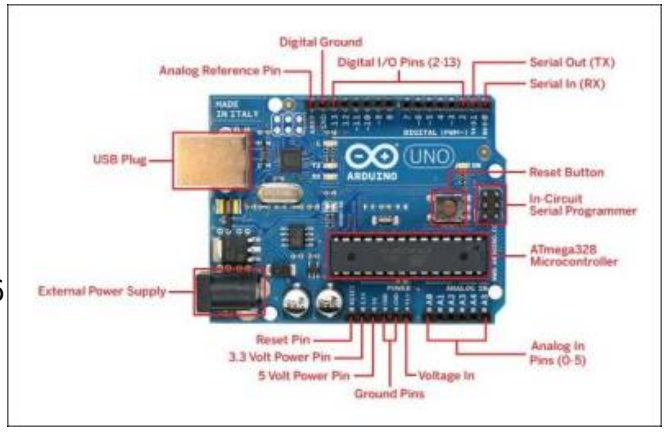


Figure 3 Arduino Uno Pin Diagram.

Some Specifications:

1. The Operating Voltage of the Arduino is 5V
2. The recommended input voltage ranges from 7V to 12V
3. The I/P voltage (limit) is 6V to 20V
4. Flash Memory - 32 KB, and 0.5 KB memory is used by the boot loader

---

2.3 LDR Photoresistor

Batson

|Purpose| A Photoresistor Light dependent resistor (LDR) is an electronic device which is used in an electronic circuit to detect and measure the presence and level of light intensity. These LDRs are specifically designed for the purpose of light sensitivity detection and the change in the resistance this creates, which differentiates them from other resistors like the carbon film resistor or the metal oxide film resistor, hence increasing their importance in an electrical circuit.

|Applicability| The LDR comes with a lot of advantages; it is cheap, readily available, easy to use and also easy to manufacture. They are made from semiconductor materials allowing

|Functioning| The way the LDR works is simple. When light makes contact with the upper surface it causes a change in resistance. This resistance can be measured in Ohms. In the serial monitor however, a digital to analog range would be produced (depending on the resolution of the board). On pin 34 of the esp32 board we expect a 0-4096 (12 bit resolution) range of values and with Arduino uno 0-1024 (10 bit resolution).

them to have light sensitive properties. The simple structure and labeled parts of an LDR can be seen in figure 3.

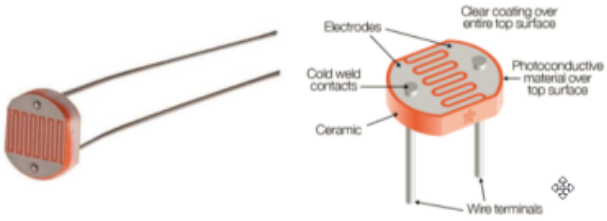


Figure 4 LDR Photoresistor.

2.4 DHT11 Temperature Sensor

Batson

|Purpose|

The DHT11 it is a simple effective but inexpensive temperature and humidity sensor. It is made up of features that measure analog signals and converts them into temperature and humidity values. This means they are quite intuitive and easy to connect to a microcontroller.

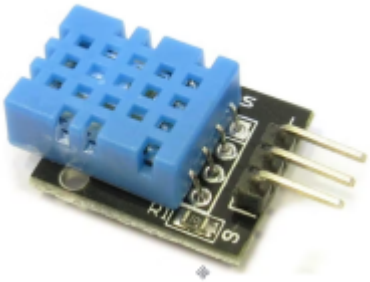


Figure 5 DHT11 .

|Applicability| The DHT is a lower power consumption device and has reliable long term stability. It's ease of accessibility and familiarity were also factors placing into our sensor selection at the project's beginning(D. Srivastava et. al. 2018)

|Functioning| As shown in the image above the data pin is connected to an I/O pin of the MCU and a 5K pull-up resistor is used. This data pin is sending the value of both temperature and humidity as serial data. There are ready-made libraries for interfacing DHT11 with Arduino.

Specifications

- Temperature range: 0 to 50ºC +/- 2ºC
- Relative humidity range: 20 to 90% +/-5%
- Temperature resolution: 1ºC
- Humidity resolution: 1%
- Operating voltage: 3 to 5.5 V DC
- Current supply: 0.5 to 2.5 mA
- Sampling period: 1 second

2.7 - SHT

Akuri

|Purpose| The SHT31 looks modest but is a very

powerful and accurate sensor when it comes to detecting temperature and humidity (Seeedstudio, 2021). It has an accuracy of ±2%RH (for relative humidity) and ±0.3°C (for temperature).

|Applicability| It is an analog sensor, it is compatible with both 3v and 5v which makes it really useful because it doesn't require a voltage shifter. This makes it one of the major parts of this project adding to the fact that it's easy to integrate.

|Functioning| This module communicates using with I2C serial bus and can work up to 1 MHz speed. This means it offers high reliability and long-term stability with low power consumption, fast response, and strong anti-interference ability.

Figure 6 Electromechanical Relay.

2.8 Software Packages

Batson

The first software required is the Arduino IDE, which was necessary for the writing, testing, and implementation of the code. This software also provided the serial monitor which was used as a first-hand view for the testing of the sensors. It was also used as a demonstration screen. The programming tool Node-Red uses a browser-based flow editor to wire together hardware devices, APIs, and online services. InfluxDB is a time series database built specifically for storing time series data. It was used along with Grafana. Grafana is a time series data visualization tool. It provides charts, graphs, and alerts for the web when connected to supported data sources.

3.0 Execution of Project

Batson

At the beginning of the project, many of the sensors and tools were clearly used, and parts were missing from the kit. In order to confirm the functionality of the sensors and the microcontrollers, as well as wires and resistors, we set out to test individual components with the Arduino UNO board and esp32 board. The idea behind was to be able to streamline the debugging by narrowing the troubleshooting efforts required later when using the esp32. This was simultaneously useful in getting to know the components and connection methods more intricately.

We start with a testing phase of the materials, debugging and selecting fitting components followed by the execution of the project goal; a prototype for building optimization of heat and lighting use.

Later on in the project development, the group became split geographically, coordinating efforts between Barbados and Germany. As a result, some variance of components are used (eg. an SHT module in place of the DHT sensor, relay modules), as a means of meeting the project execution. In end effect the project design was improved. The duplicity of components allowed us to work remotely and cooperatively, through core stages of the project.

4.0 Testing Phase

Batson

Testing was conducted with Arduino UNO at first as it was a more familiar controller, with familiar ports and specifications. Later on, testing was conducted on the DHT11 using the ESP32.

4.1 LDR Photoresistor x Arduino UNO

Batson
For the Setup, a 10k Ohm resistor was paired to the resistor on a breadboard fed by arduino UNO. It was found that the components were working responsively. To demonstrate the on and off cycle of a lightulb or lighting system 2 LEDS were used with 220 Ohm resitors each.

4.1.1 Schematic *Batson*

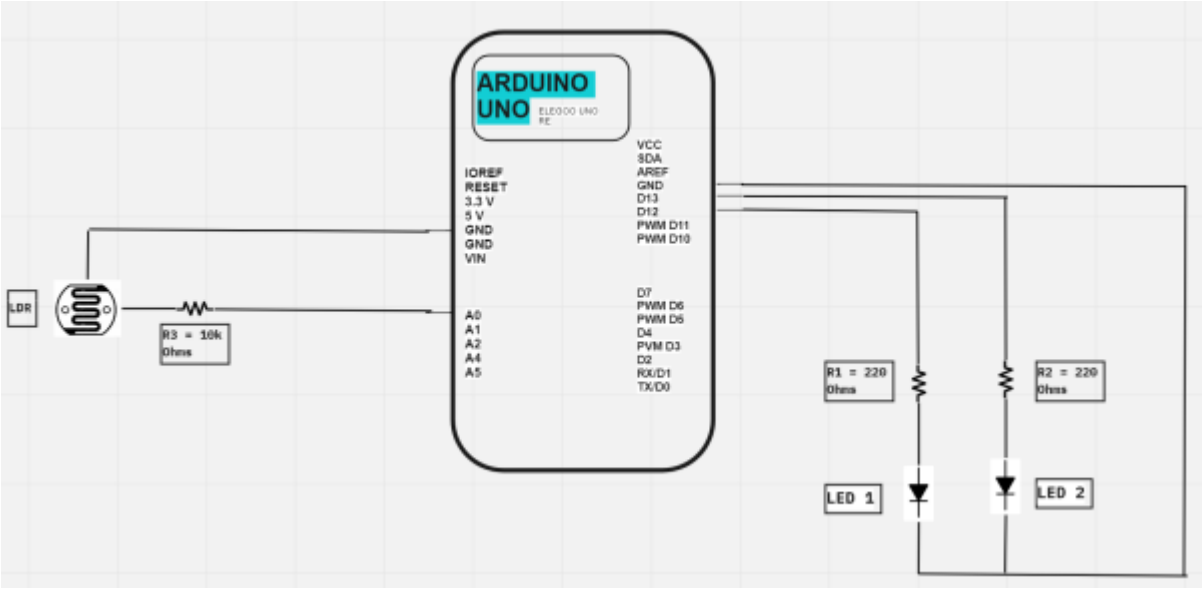


Figure 7	LDR Setup with LEDs on Arduino UNO

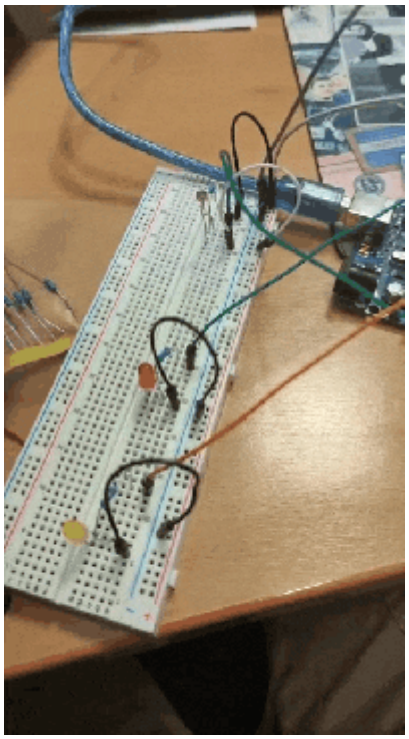
4.1.2 Code

Batson

Photoresistor Testing With Arduino Uno

```
/* Here the LDR photoresistor is tested with Arduino.
   2 LEDs are paired to the different light values to simulate an
indoor response to sunny,
   overcast, and dark natural lighting conditions
   The application of further development of this test could be useful
in vertical farming systems
   and smart lighting services for industrial buildings/homes with many
windows and architectural utilization/
   reliance on natural light.
*/
int light = 0; // store current light value

void setup() {
  Serial.begin(9600);
  pinMode(13, OUTPUT); // setting LEDs to visualize changes in lighting
in the given pin IDs.
  pinMode(12, OUTPUT);

}

void loop() {

  light = analogRead(A0); // read the light value from the LDR
photoresistor.
  Serial.println(light);

  if (light > 450) {  // when its bright (i.e greater than 450Ohms),
both LEDs will be off to save eergy)
    Serial.println("It very bright, lots of natural light");
    digitalWrite(13, LOW);
    digitalWrite(12, LOW);
  }

  else if (light > 229 && light < 451) { //medium amount of light to
show one LED at a time).
    Serial.println("moderate light quantity, quite cloudy");
    digitalWrite(13, HIGH);
    digitalWrite(12,LOW);
  }
  else { // both LEDs turn on when the surroundings are dark.
```

```
    Serial.println("It very dark, extremely overcast or night time");
    digitalWrite(13, HIGH);
    digitalWrite(12, HIGH);
  }
 delay(750); // 750ms time delay for sensor readings.
}
```



GIF 1	LDR Photoresistor & LEDs Loop in Action.

```
//                    LEGEND
//



No fingers on -> its bright -> LEDs
off.



One finger on -> it moderately
bright -> one LED on.



Two fingers on -> it's dark -> both
LEDs on.
```

4.1.3 Calibration of the LDR

Batson

LDR Calibration

```
/* Calibration of LDR values of Ohms to LUX values using a LUXmeter.
In this sketch, the calibration of LDR values to
*/


#include <Adafruit_Sensor.h>
```

```
int light = 0; //LDR readings taken and stored at pin A0.
int lightCal = 0; //creating variable to store calibrated LUX values
values and be called later in the loop.

int LEDPIN_1 = 27;  //LED pins used to react to light value
differences.
int LEDPIN_2 = 14;

//LDR Calibration Parameters
#define SNS_LIGHT_PIN 0
int sns_light_raw = 0; // the raw light value is taken from the input
pin of the LDR
float SNS_LIGHT_SLOPE = 0; //variable to store value relating to
(assumed) linear relationship between LUX and Ohmsreadings
float SNS_LIGHT_OFFSET = 0;

int light_sensor_mean(int sensor_pin, int iterations=10) { //Creating
an initial reading to measure against iterations.
  int reading = 0;
  Serial.println("Calculationg mean of readings... ");
  Serial.println("\nReading raw value...");
  Serial.print("Iterations: ");
  Serial.println(iterations);

  int temporary_value = 0; //temporary value collected of raw light
reading.
  for (int i=0; i<iterations; i++){
    reading = analogRead(sensor_pin);
    Serial.println(reading);
    temporary_value = temporary_value + reading; //saving the raw LDR
sensor readings into a temporay value folder for further manipulation.
    delay (10);

  }

  Serial.print("/nMean of raw values: ");
  int mean = temporary_value/iterations; //calculating the mean value
of raw readings using the number of trials(iterations) it passed
through.
  Serial.println(mean);
  return mean;
}

void light_sensor_calibration( int sensor_pin, int iterations=10){
  Serial.println("==========================");
  Serial.println("Light Sensor Calibration");
  Serial.println("==========================");

  int x1 = light_sensor_mean(sensor_pin, iterations);  //calibration at
this stage takes place in teh serial monitor. Corresponding values on
```

```
the LUX meter and given in.
  Serial.print("x1 from reading: ");
  Serial.println(x1);

  Serial.println("Enter LUX value from luxmeter (y1): ");
  delay(100);
  while(Serial.available() == 0){} //Wait for user input
  int y1 = Serial.parseInt(); // Lux value
  Serial.read(); // Drop "/n" character from partInt()
  Serial.print("Lux value registed: ");
  Serial.println(y1);

  int x2 = light_sensor_mean(SNS_LIGHT_PIN, 10);  // several x values
are taking to better map the relationship between LUX from the meter
and digital to analog range (0-4096) from the LDR.
  Serial.print("x2 from reading: ");
  Serial.println(x2);

  Serial.println("Enter LUX value from luxmeter (y2): ");
  delay(100);
  while(Serial.available() == 0){} //Wait for user input
  int y2 = Serial.parseInt(); // Lux value
  Serial.read(); // Drop "/n" character from partInt()
  Serial.print("Lux value registed: ");
  Serial.println(y2);

  SNS_LIGHT_SLOPE = get_slope(x1,x2,y1,y2);  //
  Serial.print("Calculated Slope: ");
  Serial.println(SNS_LIGHT_SLOPE);
  Serial.println("Save this value in variable SNS_LIGHT_SLOPE");

  SNS_LIGHT_OFFSET = get_offset(x1,x2,y1,y2);
  Serial.print("Calculated Offset: ");
  Serial.println(SNS_LIGHT_OFFSET);
  Serial.println("Save this value in variable SNS_LIGHT_OFFSET");

  Serial.println("======================");
  Serial.println("Calibration complete!");
  Serial.println("======================");

}

void light_sensor_test(int sensor_pin){
  Serial.print("Light sensor reading raw: ");
  Serial.println(light_sensor_value_raw(sensor_pin));
  Serial.print("Light sensor reading lux: ");
  Serial.println(light_sensor_value_lux(sensor_pin));
}

int light_sensor_value_raw(int sensor_pin){
  /*
```

```
   * Return Light sensor raw value
   */
  return analogRead(sensor_pin);
}

int light_sensor_value_lux(int sensor_pin){
  /*
   * Return Light sensor in lux
   */
  int raw = light_sensor_value_raw(sensor_pin);
  return (SNS_LIGHT_SLOPE*raw) + SNS_LIGHT_OFFSET;
}

int get_slope(int x1, int x2, int y1, int y2){
  /*
   * Return slope (m) from linear function ADC
   *
   * Linear function y=mx+b
   * m = (y2-y1)/(x2-x1)
   * reference: https://keisan.casio.com/exec/system/1223508685
   */

  return (y2-y1)/(x2-x1);
}

int get_offset(int x1, int x2, int y1, int y2){
  /*
   * Linear function y=mx+b
   * b = (x2y1-x1y2)/(x2-x1)
   * reference: https://keisan.casio.com/exec/system/1223508685
   */

  return (x2*y1-x1*y2)/(x2-x1);
}


void setup() {
  pinMode(LEDPIN_1, OUTPUT);
   pinMode(LEDPIN_2, OUTPUT);
  delay(100);  //100ms delay
  lightCal = analogRead(SNS_LIGHT_PIN); //take a single reading and
store it in lightCal variable.
  // This gives us preleminary value to compare and set the standard in
the loop.



  light_sensor_calibration(SNS_LIGHT_PIN);
}

void loop() {
```

```
  // || PHOTORESISTOR ||
  // Take reading using analogRead() on sensor pin and store it in
lightcal
  lightCal = analogRead(SNS_LIGHT_PIN);

  //set threshold reactions based on lightcalibrated value
  if (lightCal < 2000) {  // when its bright (i.e greater than
450Ohms), both LEDs will be off to save eergy)
    Serial.println("It very bright, lots of natural light");
    digitalWrite(LEDPIN_1, LOW);
    digitalWrite(LEDPIN_2, LOW);
  }

  else if (lightCal > 1000 && light < 2000) { //medium amount of light
to show one LED at a time).
    Serial.println("moderate light quantity, quite cloudy");
    digitalWrite(LEDPIN_1, HIGH);
    digitalWrite(LEDPIN_2,LOW);
  }
  else { // both LEDs turn on when the surroundings are dark.
    Serial.println("It very dark, extremely overcast or night time");
    digitalWrite(LEDPIN_1, HIGH);
    digitalWrite(LEDPIN_2, HIGH);
  }
 delay(750); //750ms delay between sensor readings.
}
```

While this calibration technique was very interesting to learn and execute, it was later on decided in the project that the LDR reading could be more simply mapped, and more importantly with less error. The problem with this calibration is that a linear relationship is assumed between LUX readings and 12 bit ADC range recorded from the LDR. This is in reality not the case, and with the nuisance of first needing a LUX meter to first calibrate the sensor when using it (which I didn't have access to in Barbados) a simpler function was used to interpret light readings from the LDR for the later parts of the project.

Table 1 : Qualitative Calibration for the LDR Setup in a Closed Room in Barbados. *Batson*

Inside measurements.	**light conditions**	**corresponding value Arduino UNO**	Corresponding Value ESP32
		ADC Range = 0 - 1024	**ADC Range = 0 - 4096**
	Lots of Natural and Ambient	light value < 230	light value < 1300
	Moderate amounts of Light	230 < light value < 500	1300 < light value < 2500
	Meager illumination of Light	light value > 500	light value > 2500

4.2 DHT 11 x ESP32

Batson

Having gained some traction with Arduino UNO as the microcontroller, we decided it would be a good idea to try out the ESP32 to test the DHT11 sensor. Lessons learned so far from the LDR setup described above were concatenated in this first try.

4.2.1 Code

Batson

DHT11xESP32 Test: Heating Management Optimization

```
/* Use ESP32 to read temperature values of DHT11.
   Control the on/off cycles of a fan (actuator) by threshold values.
*/

/* Slowly combining parts of the code.
   Heating & lighting components, no MQTT.
*/
#include "DHT.h"
#include <Adafruit_Sensor.h>


int light = 15; //LDR readings taken and stored at pin A0.
int DHTPIN = 4; //DHT pin to store and take readings at A1.

int LEDPIN_1 = 27;  //LED pins used to react to light value
differences.
int LEDPIN_2 = 14;
int FAN = 26; //Fan to simulate actuator heat pump, reacts to
temperature input.

#define DHTTYPE DHT11 // defining DHT type for ESP32 and scope in
general.
DHT dht(DHTPIN, DHTTYPE);


void setup() {
  Serial.begin(9600); //initiate serial.
  Serial.print(F("DHT11 Test"));
  dht.begin();
```

```
  pinMode(15, INPUT);  //setting LDR pin as input.
  pinMode(LEDPIN_1, OUTPUT); //Setting LEDs as outputs, since they are
reacting to incoming light data
  pinMode(LEDPIN_2, OUTPUT);
}

void loop() {
  //DHT
  int temperature = dht.readTemperature(); // read Temperature in °C

  Serial.print(temperature);
  String temperatureString = String(temperature);

  //Serialprintln(hic);
  Serial.println(F("°C"));
  delay (2000);

  Serial.println("The string equivalent to integer temperature is " +
temperatureString);
  char* TempArr = &temperatureString[0];
  //Serial.println(TempArr);

  //Fan off-on cycle
  if (temperature > 30) {
    Serial.println("Its becoming hot in here - The fan is on");
    digitalWrite(FAN, HIGH);
  }
  else {
    Serial.println("ITs cold or tolerable; lets save money!");
    digitalWrite(FAN, LOW);
  }



  //LDR
  light = analogRead(15); //read light value from LDR in Ohms EVALUATE
SHOUÖD LIGHT NOT 15
  Serial.println(light);

  if (light < 1300) {
    Serial.println("It very bright, lots of natural light");
    Serial.println("Both LEDs are off");
    digitalWrite(LEDPIN_1, LOW);
    digitalWrite(LEDPIN_2, LOW);
  }
  else if (light > 1300 && light  < 2500) { //medium amount of light to
show one LED at a time).
    Serial.println("moderate light quantity, quite cloudy");
    digitalWrite(LEDPIN_1, HIGH);
    digitalWrite(LEDPIN_2, LOW);
  }
```

```
  else { // both LEDs turn on when the surroundings are dark.
    Serial.println("Its very dark, extremely overcast or night time");
    digitalWrite(LEDPIN_1, HIGH);
    digitalWrite(LEDPIN_2, HIGH);
  }
  delay(750); //750ms delay between sensor readings.
}
```

4.2.2 Schematics

Batson

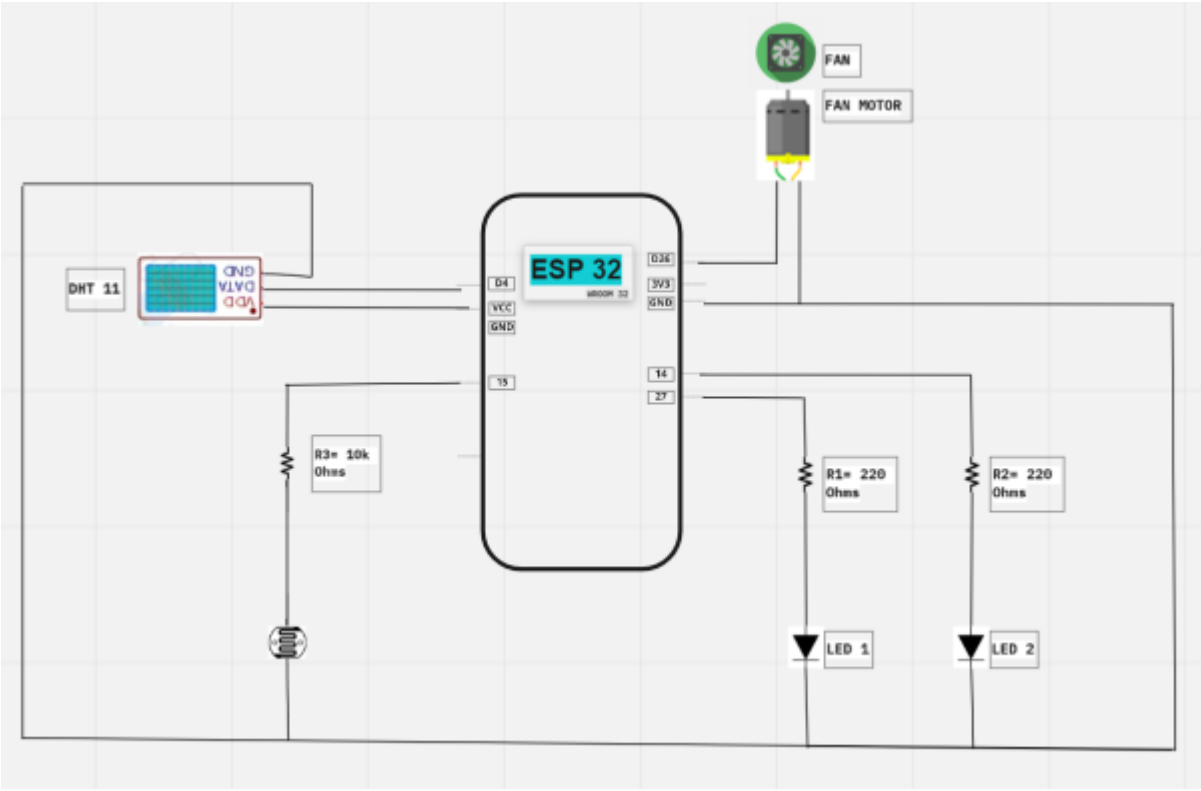


Figure 8 Schematic Representation of the First Trial of Fan Connection to esp32.

4.2.3.1 - Troubleshooting Components and Code

Akuri & Batson

Upon execution of this set-up, some substantial dilemmas came to light. For one, it was evident the microcontroller was not adequately powering the fan optimally. Primarily for this reason we decided to configure a relay and battery to adjacently control the functioning of the fan as an output to

temperature values which, for long periods of time could be recorded over 30°C in Barbados where the trial was conducted. The relay was our solution to an independent power supply for the operation of the motor.

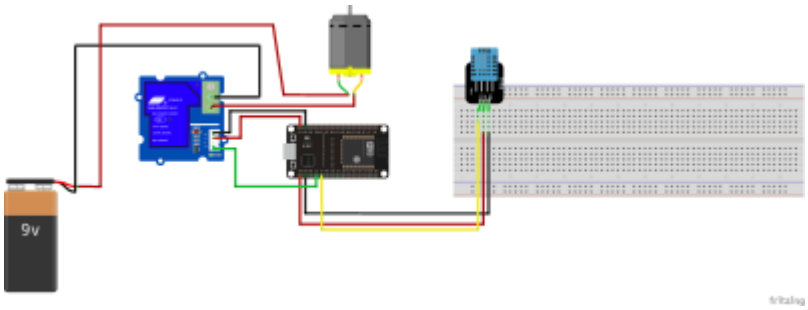


Figure 9	Change in Configuration of the fan, now moderated by a relay and 9V battery pack.

Although the code for this setup worked, there were two obstacles that arose when we tried testing the setup. The first issue was with the sensor. During the testing phase, the DHT11 proved to be quite unreliable. Sometimes it would work as required but most of the time, it became quite inconsistent with the data it provided slowing down the entire process. The second issue was with the first type of relay used. The first type of relay used with the fan connection was only compatible with a 5v supply. However, the board used only had a 3v pin. Because of this, the relay will receive the signal from the pin but would not be able to turn on the fan because of the low voltage supply. Notwithstanding, we were able to find solutions to both of these issues. The first one was solved by completely changing the DHT11 sensor to a more responsive and reliable option which was the SHT31 and a less powerful but equally efficient relay was used. The new relay could support both 3v and 5v. Parts of the code were modified and rewritten in accordance with the new components. The new changes proved to be the right move, as the setup worked without any issues.

4.2.3.2 - SHT31 with Fan

Akuri

As the project moved further, there was a lot of trial and error. This also meant our understanding and knowledge of the different components started expanding. This path led us to change from the DHT11 sensor to the SHT31 Sensor. Although both measure temperature, we discovered that the DHT11 was not as reliable and as responsive as the SHT11. The reason was not explicitly clear, the hardware simply read values inconsistently. After switching from the DHT to the SHT temperature sensor, value readings were clearer. The coding for the simultaneous detection and action of both sensors is shown below:

Modified Relay and SHT

```
/* Combined parts of the code for both sensors and their target
devices.
   Cooling(Fan) & lighting components (No MQTT yet).
*/
#include <Arduino.h>
#include <Wire.h>
#include "Adafruit_SHT31.h"
#include <Adafruit_Sensor.h>

bool enableHeater = false;
```

```
uint8_t loopCnt = 0;

Adafruit_SHT31 sht31 = Adafruit_SHT31();

int light = 15; //LDR readings taken and stored at pin A0.
int LEDPIN_1 = 27;  //LED pins used to react to light value
differences.
int LEDPIN_2 = 14;
int FAN = 26; //Fan to simulate actuator heat pump, reacts to
temperature input.
#define relay 4

void setup() {

  Serial.begin(9600); //does this start both sensors or just one?
  Serial.print(F("DHT11 Test"));
  pinMode(relay,OUTPUT);
  if (! sht31.begin(0x44)) {   // Set to 0x45 for alternate i2c addr
    Serial.println("Couldn't find SHT31");
    while (1) delay(1);
  }

  Serial.print("Heater Enabled State: ");
  if (sht31.isHeaterEnabled())
    Serial.println("ENABLED");
  else
    Serial.println("DISABLED");

  pinMode(15, INPUT);
  pinMode(LEDPIN_1, OUTPUT); //Setting LEDs as outputs, since they are
reacting to incoming light data
  pinMode(LEDPIN_2, OUTPUT);
}

void loop() {
  //DHT
  float temp = sht31.readTemperature();
  float h = sht31.readHumidity();

  if (! isnan(temp)) {  // check if 'is not a number'
    Serial.print("Temp *C = "); Serial.print(temp);
Serial.print("\t\t");
  } else {
    Serial.println("Failed to read temperature");

  }
  if(temp>=30)                    // Turn the fan on: We are a using a
Normally closed connection here for the relay
  {
    // Normally Closed (NC) configuration, sends HIGH current signals
to stop the current flow
```

```
    digitalWrite(relay, HIGH);
  Serial.println("Fan is ON");
  delay(5000);
  }
  else                          // Turn the fan off
  {

  // Normally Closed (NC) configuration, sends LOW signals to stop the
current flow
  digitalWrite(relay, LOW);
  Serial.println("Fan is OFF");
  delay(5000);
   }
  }



  //LDR
  int light = analogRead(15); //read light value from LDR in Ohms
EVALUATE SHOULD LIGHT NOT 15
  Serial.println(light);

  String lightString = String(light);
  Serial.println("The string equivalent to integer light is " +
lightString + "°C");

  if (light < 1300) {
    Serial.println("It very bright, lots of natural light");
    Serial.println("Both LEDs are off");
    digitalWrite(LEDPIN_1, LOW);
    digitalWrite(LEDPIN_2, LOW);
  }
  else if (light > 1300 && light  < 2500) { //medium amount of light to
show one LED at a time).
    Serial.println("moderate light quantity, quite cloudy");
    digitalWrite(LEDPIN_1, HIGH);
    digitalWrite(LEDPIN_2, LOW);
  }
  else { // both LEDs turn on when the surroundings are dark.
    Serial.println("Its very dark, extremely overcast or nighttime");
    digitalWrite(LEDPIN_1, HIGH);
    digitalWrite(LEDPIN_2, HIGH);
  }
  delay(750); //750ms delay between sensor readings.

  char* lightArr = &lightString[0]; //client.publish needs matching
format of topic folder and light value
  //Serial.println(lightArr);
 // Serial.print (" this is the array");
```

```
}
```

This setup is not just relatively simple to put together but also very efficient and responsive. The different components have also been connected in such a way that there is enough energy available to power up each component in the circuit but the circuit is also kept in an open state (meaning there's no flow of current) while the readings of the sensor remain within a certain limit as stated in the code. A simplified colored view of the setup and its schematic is displayed in the figures below. The 3v relay plays an important role because it helps control the flow of current to the fan. It closes the circuit (allowing the flow of current) based on the commands it receives from the esp32 board and the batteries are there to make sure that there is always enough power being provided to the circuit.

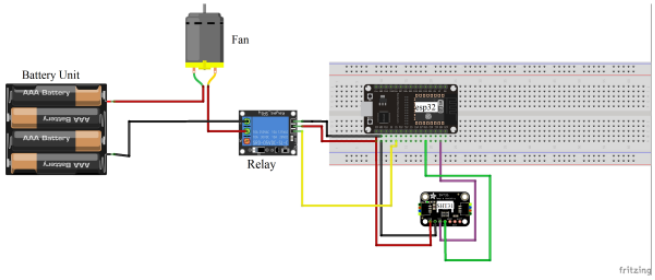


Figure 10 - A Simple view of the circuit setup (Akuri)

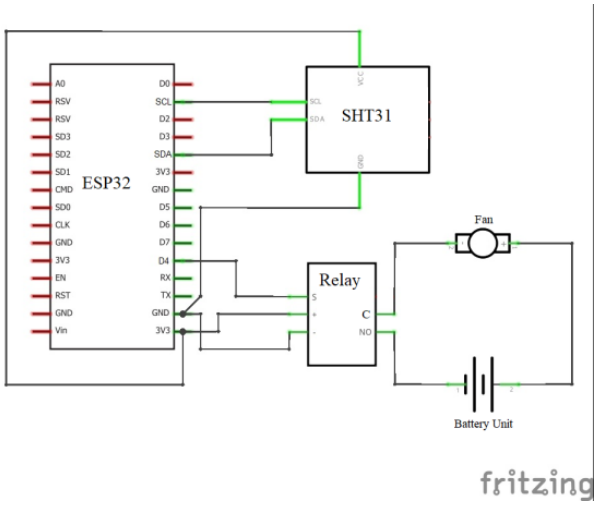


Figure 11 - Schematics of the setup (Akuri)

5.0 Results of Combining the Codes, MQTT & NIG

Batson, Akuri

After receiving successful testing and debugging for the used components of the kit, the project continued to be further developed. At this point the project takes sensory data and turns on components representing appliances, but, we want to be able to publish these results online to visualize and monitor them. For this reason we used MQTT and NIG. Important as well to note; we we're working between Germany and Barbados, so we varied the setup to enable project execution exercise on both sides.

5.1 Schematics

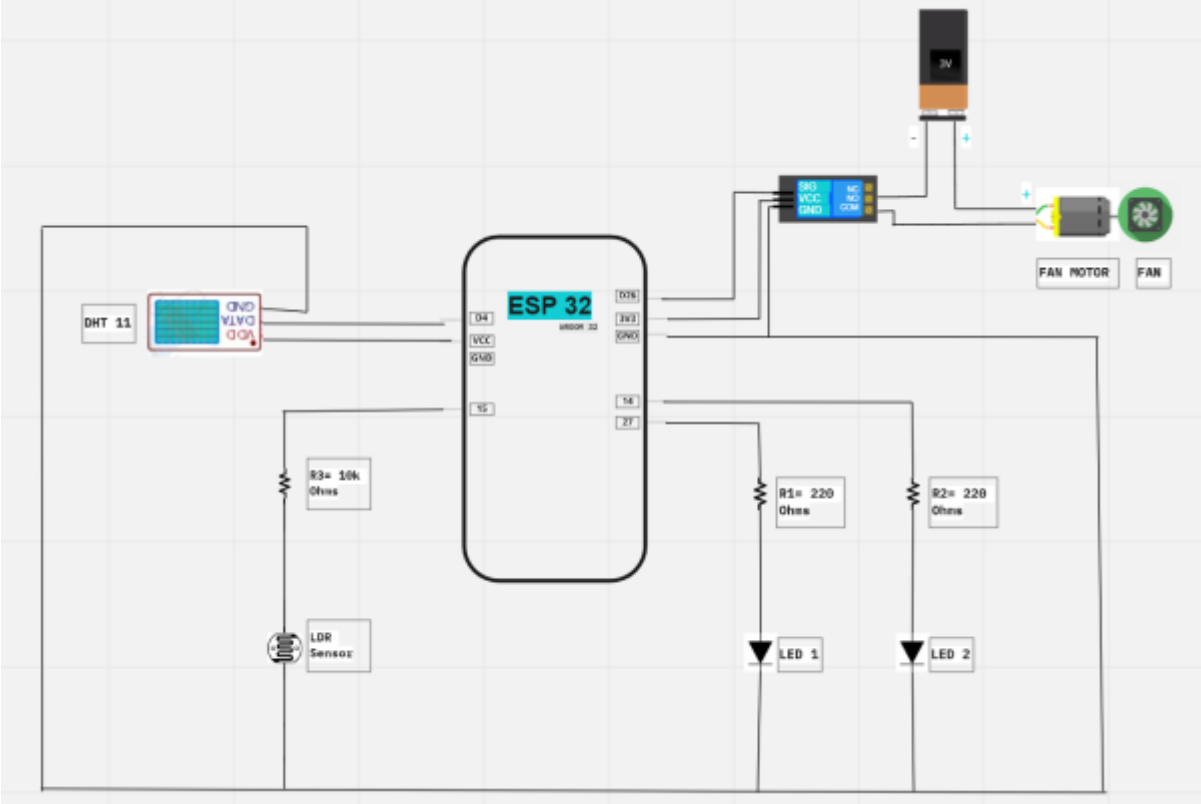


Figure 12	Complete Configuration of the Project (with DHT11 & Groove Relay). (Batson)

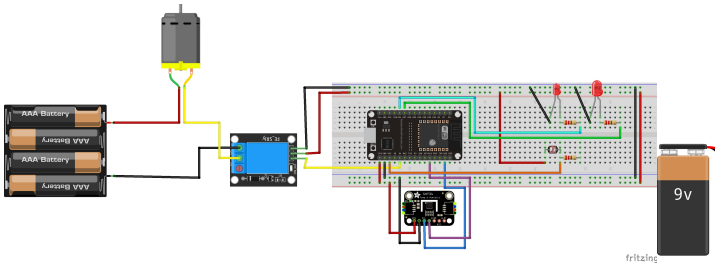


Figure 13	Complete Configuration of the Project (SHT31). 3v Relay + 4v battery (Akuri)

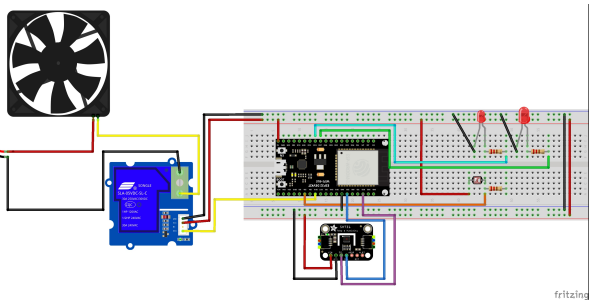


Figure 14	Complete Configuration of the Project (SHT31). 5v Relay + 9v battery (Akuri)

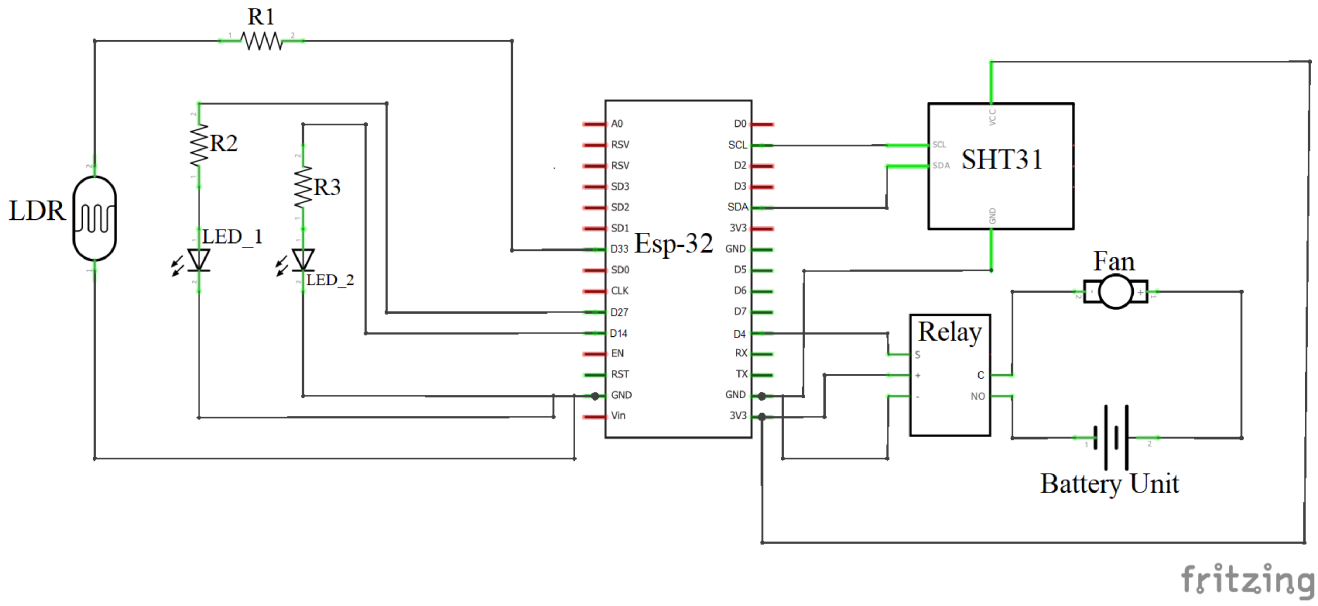


Figure 15 Complete Schematic, R1= 10K Ohms, R2 & R3 = 220Ohms

5.2 Code

Batson, Akuri

Complete Code: Warehouse Management and Optimization

```
//#include "DHT.h"  //library relevant for dht sensor operation commands.
#include <Adafruit_Sensor.h>  //command repository for basic sensor communication functions.
#include <SPI.h> //librarz facilitating communication with SPI devices (eg. RTC).
#include <Ethernet.h>
#include <PubSubClient.h> //library allows sending and receiving MQTT messages.
#include <WiFi.h> //internet connection enabling library.
#include <Wire.h>  //communication with I2C devices.
#include "Adafruit_SHT31.h"


/* int DHTPIN = 33; //DHT pin to store and take readings at A1.*/
int LEDPIN_1 = 27;  //LED pins used to react to light value differences.
int LEDPIN_2 = 14;
int FAN = 32; //Fan to simulate actuator of fan, reacts to temperature input.

char lightarray[16]; // store the light value.
char temperaturearray[16]; //  store the temperature value.

Adafruit_SHT31 sht31 = Adafruit_SHT31();
```

```
// WiFi
const char* ssid = "iotlab"; // The network's name
// CNXK004A2418
const char* password = "iotlab18"; // password for the network
// ****************

// MQTT Broker
const char* mqtt_broker = "broker.hivemq.com"; //
//const char* mqtt_server   = "hsrw.space"; // MQTT Broker IP address
const char *mqtt_username = "emqx";
//const char* mqtt_username = "user";
const char *mqtt_password = "public";
//const char* mqtt_password = "mqtt";
const int mqtt_port = 1883;

const char* myname = "Adiel";
const char* topic_up_temperature = "amc2022/groupD/up/Temperature";
const char* topic_up_Light = "amc2022/groupD/up/Light";
const char* topic_up_Fan_Switch = "amc2022/groupD/up/FanSwitch";
const char* topic_up_LEDPIN_1 = "amc2022/groupD/up/LEDPIN_1";
const char* topic_up_LEDPIN_2 = "amc2022/groupD/up/LEDPIN_2";



void initWiFi() {  //initiate wifi connections void
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
  }
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
//callback setup to ensure functtion doesnt run before task is
completed
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}

WiFiClient ethClient;
PubSubClient client(ethClient);
```

```
void reconnect() { //It will reconnect to the server if the connection
is lost using a blocking reconnect function

  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("arduinoClient", mqtt_username, mqtt_password))
{
      Serial.println("connected");

    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup() {  //basic setting for iniating serial, sensors, setting
baud rate.
  Serial.begin(115200);
  initWiFi();
  Serial.print("RRSI: ");
  Serial.println(WiFi.RSSI()); //Ensure & check if wifi connection is
quality. ESP32 has low range, comparative to other devices.
  client.setServer(mqtt_broker, 1883); //MQTT data transmission using
port 1883
  client.setCallback(callback);


  delay(1500); // Allow the hardware to sort itself out, avoid clashes
of commands

    Serial.println("SHT31 test");
  if (! sht31.begin(0x44)) {   // 0x44 is the i2c address
    Serial.println("Couldn't find SHT31");
    while (1) delay(1);
  }


  /* pinMode (DHTPIN, INPUT); // sets digital pin 33(DHT) as an input.
*/
  pinMode(33, INPUT);       //Sets LDR as an input on pin .
  pinMode(LEDPIN_1, OUTPUT); //Setting LEDs as outputs, since they are
reacting to incoming light data.
  pinMode(LEDPIN_2, OUTPUT);
  pinMode(FAN, OUTPUT); // sets digital pin 32 (the relay) as an
output, to temperature data.
```

```
  client.subscribe (topic_up_temperature); // subscribe to temperature
topic
  client.subscribe (topic_up_Light); // subscribe to temperature topic
  client.subscribe (topic_up_Fan_Switch); // subscribe to temperature
topic
  client.subscribe (topic_up_LEDPIN_1); // subscribe to temperature
topic
  client.subscribe (topic_up_LEDPIN_2); // subscribe to temperature
topic

}

void reading_data(){

  //LDR
  int light = analogRead(33); //read light value from LDR in
Ohms(Physical pin D33)
  Serial.println(light);

  if (light > 3500) {

    Serial.println("Its very dark, extremely overcast or night time");
    Serial.println("Both LEDs are ON");
    digitalWrite(LEDPIN_1, HIGH);
    digitalWrite(LEDPIN_2, HIGH);

  }
  else if (light > 1100 && light  < 3500) { //medium amount of light to
show one LED at a time).
    Serial.println("moderate light quantity, quite cloudy");
    digitalWrite(LEDPIN_1, HIGH);
    digitalWrite(LEDPIN_2, LOW);
  }
  else { // both LEDs turn on when the surroundings are dark.
    Serial.println("It very bright, lots of natural light");
    Serial.println("Both LEDs are OFF");
    digitalWrite(LEDPIN_1, LOW);
    digitalWrite(LEDPIN_2, LOW);
  }
  delay(10); //750ms delay between sensor readings.


//SHT

  int temperature = sht31.readTemperature(); // read Temperature in °C
  Serial.print(temperature);
  Serial.println(F("°C"));
  delay (50);

  if (temperature > 30) {
    Serial.println("Its becoming hot in here - The fan is on");
```

```
    digitalWrite(FAN, HIGH);
  }
  else {
    Serial.println("Its cold or tolerable; lets save money!");
    digitalWrite(FAN, LOW);
  }


  if (! isnan(temperature)) {  // check if 'is not a number'
    Serial.print("Temp *C = "); Serial.print(temperature);
Serial.print("\t\t");
  } else {
    Serial.println("Failed to read temperature");
  delay (50);
  }
  /*
  if(temperature>=30) {
    // Turn the fan on: We are a using a Normally closed connection
here for the relay (normally open grove-relay).
    // Normally Closed (NC) configuration, sends HIGH current signals
to stop the current flow
    digitalWrite(FAN, HIGH);
  Serial.println("Fan is ON");
  delay(50);
  }
  else   {                      // Turn the fan off
  // Normally Closed (NC) configuration, sends LOW signals to stop the
current flow
  digitalWrite(FAN, LOW);
  Serial.println("Fan is OFF");
  delay(50);
   }*/
     String temperaturestr = String(temperature);// for MQTT
transmission
    temperaturestr.toCharArray(temperaturearray,
temperaturestr.length() + 1);
    client.publish(topic_up_temperature, temperaturearray);// To
publish the topic under
    Serial.println("Publish temperature");


    String lightstr = String(light); // for MQTT transmission
    lightstr.toCharArray(lightarray, lightstr.length() + 1);

     client.publish(topic_up_Light, lightarray); // light values are
published under the topic light
     //client.publish(topic_up_LEDPIN_1, lightarray);
    delay(2000);
}

void loop() {
```

```
  //MQTT
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  reading_data(); // reading data from the different sensors from
MQTT_publishing

  /* delay(2000); // delay by 2 sec */

}
```

5.3 Serial Monitor, Graphana and Node-Red

Akuri

After the setup was installed in a specific location inside a building, the SHT31 and LDR were programmed accordingly. The code was uploaded to the ESP32, the results could be seen instantly on the serial monitor, Node-Red, and the dashboard in Grafana. For outdoor testing, the sensors need to be isolated safely in order not to get into contact with the water from the pond, blown away by the wind, or even taken away by a bird. Therefore, for testing, the measurements were taken indoors over a period of about 5 days during the summer but in an indoor location with air conditioning. The results of those measurements can be seen in figure 16. The temperature spikes from 28 to 30 to activate the fan. It can be seen, that the sensors immediately detect their respective values and report changes almost instantly.

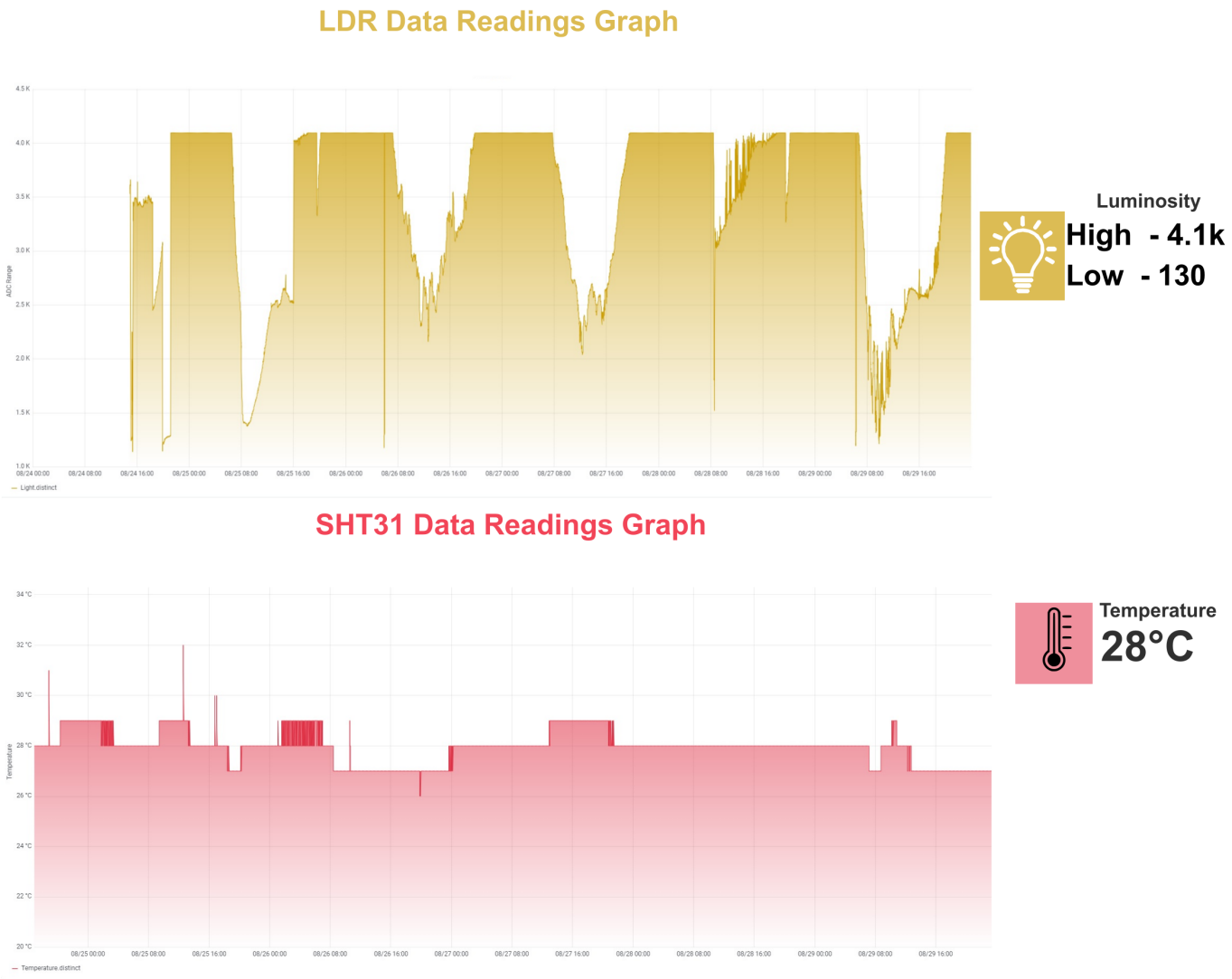


Fig. 16	Measurement results of the SHT-31 and LDR sensors visualized in Grafana

6.0 Discussion & Reflection

Batson, Akuri

A simple optimization of heating and lighting was successfully conducted. The project has many possibilities for further refining and development. The flow chart below shows a simplified view of what is being executed

Fig. 17 - A flowchart of the processes taking place (Akuri).

The entire design and setup of a prototype that interprets environmental parameters (light intensity & temperature) and takes action autonomously with respect to given measurement intervals and parameters, was conceived to optimize building functionality and save resources. Data transaction and monitoring were also successfully conducted using MQTT, influx DB(for connecting data points and types) and Grafana (visualization). Although this goal was achieved, the setup can still be further upgraded to better optimize output switches, include more appliances, use better sensors, better data monitoring and transferring software and more.

Obstacles Some obstacles which were met included; integrating both sensors individually, and then together with MQTT and internet protocol. In this regard, many of the ADC2 pins of the board are not usable while trying to establish an internet connection. This produced the most unspecified error message of the project which consumed the most time first recognize the exact problem and then fix it.

Limitations & Possible Improvements Despite the fact that the project is functional, it should only be run under controlled conditions. This is because, if the LDRs are to be installed on the roof of the warehouse, they need to be placed in special protective frames to protect them from rain, wind, or even birds. Important to note, is the range as well for which the LDR is or can be applicable; different measurements of light intensity inside and outside for example speak more about differences in light quality differences in than light intensity. Therefore measurements made with sunlight would have an alternate value to measurements from artificial light sources. In reality, this ambient light intensity difference should be accounted for with a standardized sensor and method of placement. The same

holds for the temperature sensor. This is especially necessary in the case where it rains, it needs to be protected from getting wet if not this would spoil the sensor. It would be advantageous to either install signal boosters or position the esp32 in a strong WiFi signal strenght because the I2C at 100 kHz bus frequency runs slowly (Adafruit,2022). At this rate, the ESP32 leaves 10ms gaps between I2C transactions. This can slow down your I2C interactions considerably, especially if there are a lot of output device to turn on and off simultaneously and data to be transferred successfully in the first place.

As a prototype, this project has many areas for improvement. The LDR calibration in Table 1 can be better designed to reflect legal workplace light quantity levels requirements. This would ensure the amount of light delivered by the LEDs is sufficient and legally allowable. On this thread, the type, and size of the LEDs and therefore the connecting apparatus and power supply would need to be recalculated and implemented using certified replacement products from the market, and conducted by certified personnel.

With a working system another consideration would be the trigger for the lights to turn off when the warehouse is not in use and it is simultaneously dark outside. This can be amended for in code, in conjugation with an RTC. This way the optimization system will work for duration of shift times and turn off all necessary appliances for the duration of closing time each day.

Another advantage of the RTC would be in optimizing the energy saving potential of the system itself. Measurements can be taken twice every hour, which would take a matter of seconds for the esp32 to connect to the Wi-Fi, and transmit data and control outputs. A deep sleep protocol with a Real Time Clock (RTC) would be extremely advantageous in bringing about these energy savings; the esp32 could be idle (in deep sleep) the vast majority of the time.

Video Tutorial In this vdieo we introduce the project, project prototype and outline the execution of the project in detail.

amc_presentation.mp4

In conclusion, the prototype enabled us to gain a deep insight into the planning and implementation of smart systems. While we've learned quite a lot, we've just scratched the surface. These systems are complex to develop and integrate. It was fun giving it a try.

7.0 Literature References and Resources


```
- Mahiri, F., Najoua, A., & Ben Souda, S. (2022). 5G-Enabled IIoT Framework
Architecture Towards Sustainable Smart
Manufacturing. In International Journal of Online & Biomedical Engineering;
2022, Vol (Bd. 16, Nummer 4, S. 4–20).
https://doi.org/10.3991/ijoe.v18i04.27753

- D. Srivastava, A. Kesarwani, S. Dubey(2018) Measurement of Temperature and
Humidity by using Arduino Tool and
DHT11 Department of Computer Applications, JSS Academy of Technical
Education, Noida, India
```

- Adafruit, (2022),//ESP32-S2 Bugs & Limitations// https://learn.adafruit.com/adafruit-magtag/esp32-s2-bugs-limitations

- Seeedstudio. (2021). //Grove - Temp and Humi Sensor(SHT31).// https://wiki.seeedstudio.com/Grove-TempAndHumi_Sensor-SHT31/.

- Seeedstudio. (2021). //Grove - Relay.// https://wiki.seeedstudio.com/Grove-Relay/ .

- Amperite. (2019). //WHAT IS A RELAY AND WHY ARE THEY SO IMPORTANT?// https://amperite.com/blog/relays/.

Source of components:
- https://www.amazon.de/gp/product/B01IHCCKKK/
- iotlab of Hochschule Rehin Waal

From:
https://student-wiki.eolab.de/ - **HSRW EOLab Students Wiki**

Permanent link:
https://student-wiki.eolab.de/doku.php?id=amc2022:groupd:start&rev=1662484580

Last update: **2023/01/05 14:38**