INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Indoor climate control based on blinds

Oleksandr Zakordonets

Supervisors

Paulo Jorge Teixeira Matos

Luís Frölén Ribeiro

Master in Industrial Engineering

2020-2021

Indoor climate control based on blinds

Oleksandr Zakordonets

In modern society, people spend around 90% of their time in buildings, thus indoor lighting condition plays an important role in determining inhabitants' mood and health, as well as visual comfort. Window blinds and shading are ideal approaches to eliminate uncomfortable glare and save energy by utilizing natural daylight.

Abstract

In modern society, people spend around 90% of their time in buildings, thus indoor lighting condition plays an important role in determining inhabitants' mood and health, as well as visual comfort. Window blinds and shading are ideal approaches to eliminate uncomfortable glare and save energy by utilizing natural daylight.

In the work the following goals are defined

1) Getting the energy balance in closed rooms;

2) Given the energy balance from the 1, implement the decision procedure to close or open the blinds without human involvement;

3) Implement solution to identify human presence;

4) Identify humans to get the preferences;

5) Given the energy balance from the 1 and the human presence from 3, make a decision to close or open the blinds with human presence

Keywords: blinds control, indoor climate control, indoor energy balance, energy saving.

Content

List of Figures

Chapter 1

Introduction

In modern society, people spend around 90% of their time in buildings, thus indoor lighting condition plays an important role in determining inhabitants' mood and health, as well as visual comfort. Window blinds and shading are ideal approaches to eliminate uncomfortable glare and save energy by utilizing natural daylight [1].

However, manually operating shading devices rarely obtains the optimal balance between the daylight utilization, glare control and energy savings. Automated shading control can lead to better overall energy performance, nevertheless visual comfort is always a priority.

The typical blind control strategy to avoid direct illuminance is open or close the blinds to get optimal energy performance, but satisfying the user's preferences at the same time. Tsempelikos and Chan evaluated different strategies - a cut-off angle control, a daylight redirecting control, and two glare protection control modes. Zhang and Birru (2012) developed an open-loop control algorithm to calculate cut-off angle for arbitrary blind opening/slat height (including curved slats) to avoid direct daylight on work plane beyond an assigned distance. Koo et al. (2010) demonstrated a strategy controlling both tilt angle and blind position for multiple blind products. Hu and Olbina (2011) developed an illuminance-based slat angle selection model using the artificial neural network (ANN) method and they trained their ANN by the data generated from EnergyPlus. The ANN model had higher simulation speed than a traditional analytical model. [2]

1.1. Goals

In the work the following goals are defined

1. Getting the energy balance in closed rooms;

2. Given the energy balance from the 1, implement the decision procedure to close or open the blinds without human involvement;

3. Implement solution to identify human presence;

4. Identify humans to get the preferences;

5. Given the energy balance from the 1 and the human presence from 3, make a decision to close or open the blinds with human presence

1.2. Organization

This work is organized as following:

Chapter 2 shows the state-of-art methods to get energy balance and human presence and gives the analysis of the shown methods, presenting their advantages and disadvantages.

The Chapter 3 shows the used hardware and software for the project.

The Chapter 4 shows the implementation of the picked methods and algorithms that define the decision for the blinds.

The Chapter 5 shows the results of implementation and possible ways to improve the work.

The Conclusion chapter shows the final achievements and the main results of the work.

Chapter 2

Context and Technologies/Tools

To effectively control the blinds and make it energy effective, there is a need to make an energy balance in building, that creates a need to understand what is indoor energy balance and what are the approaches to make it.

Indoor energy balance is the balance of energy between the energy that is received from the environment and the energy released to the environment. For example, if the weather is sunny, the building receives the energy from the sun. If the weather is cloudy, the building gives the energy to the environment.

In the work the effect of blinds in energy saving will be consider.

2.1. Blinds and energy savings

In [6] different ways to control the blinds are considered. The metrics used in this paper are hybrid ray-tracing and radiosity method. A detailed daylighting analysis was performed in order to evaluate set points for blind control actions, annual daylighting metrics, lighting energy use and daylight glare probability and duration. Different room sizes, glazing properties, blind material characteristics and orientations were studied. The results showed that diffuse blinds do not reduce the risk of glare when light redirection is desired; double-sided slats with a diffuse top and a specular bottom surface are better in this regard.

There are several approaches to automatically control the blinds.

In [7] highly integrated automated shading is considered. It is designed to regulate daylighting according to the varying sky conditions in decentralized applications, it optimizes visual comfort for occupants and maximizes their view outwards.

In [11] neural network-based approach is proposed. For the indoor shading and daylighting requirement, the smart control system, which has both function of adjusting the venetian blind position by machine-learning algorithm and indoor lighting control automatically, is built depending on neural network in this paper.

In [8] there is a brief overview for different types of temperature sensors.

In [9] the working principle of solar panels are presented. In [6] can see the usage of solar panels as radiation sensors is considered.

Chapter 3

Modelling

Here the implementation of the blinds control is described. Here an algorithm to open or close the blinds to optimize user preferences and climate efficiency needs to be provided. Given the goal, the mixed approach is used:

1. Without human presence
2. With human presence

3.1. Without human presence

Here we have following crucial parameters for blinds

1. Temperature
2. Solar radiation

With these parameters, there is a need to get the data from the sensors.

1. Temperature sensor

2. Solar panel as solar radiation sensor

1. Temperature Sensors measure the amount of heat energy or even coldness that is generated by an object or system, allowing us to “sense” or detect any physical change to that temperature producing either an analogue or digital output. There are many different types of Temperature

Sensor available and all have different characteristics depending upon their actual application. A temperature sensor consists of two basic physical types:

1. Contact Temperature Sensor Types – These types of temperature sensor are required to be in physical contact with the object being sensed and use conduction to monitor changes in temperature. They can be used to detect solids, liquids or gases over a wide range of temperatures.

2. Non-contact Temperature Sensor Types – These types of temperature sensor use convection and radiation to monitor changes in temperature. They can be used to detect liquids and gases that emit radiant energy as heat rises and cold settles to the bottom in convection currents or detect the radiant energy being transmitted from an object in the form of infra-red radiation (the sun).

The two basic types of contact or even non-contact temperature sensors can also be sub-divided into the following three groups of sensors, *Electro-mechanical*, *Resistive* and *Electronic* and all three types are discussed below.

1. The Thermostat

The Thermostat is a contact type electro-mechanical temperature sensor or switch, that basically consists of two different metals such as nickel, copper, tungsten or aluminium etc, that are bonded together to form a Bi-metallic strip. The different linear expansion rates of the two dissimilar metals produces a mechanical bending movement when the strip is subjected to heat.

2. The Thermistor

The Thermistor is another type of temperature sensor, whose name is a combination of the words THERM-ally sensitive res-ISTOR. A thermistor is a special type of resistor which changes its physical resistance when exposed to changes in temperature. Thermistors are generally made from ceramic materials such as oxides of nickel, manganese or cobalt coated in glass which makes them easily damaged. Their main advantage over snap-action types is their speed of response to any changes in temperature, accuracy and repeatability.

Most types of thermistor's have a *Negative Temperature Coefficient* of resistance or *(NTC)*, that is their resistance value goes DOWN with an increase in the temperature, and of course there are some which have a *Positive Temperature Coefficient, (PTC)*, in that their resistance value goes UP with an increase in temperature.

3. Resistive Temperature Detectors (RTD).

Another type of electrical resistance temperature sensor is the Resistance Temperature Detector or RTD. RTD's are precision temperature sensors made from high-purity conducting metals such as platinum, copper or nickel wound into a coil and whose electrical resistance changes as a function of temperature, similar to that of the thermistor. Also available are thin-film RTD's. These devices have a thin film of platinum paste is deposited onto a white ceramic substrate.

4. The Thermocouple

The Thermocouple is by far the most commonly used type of all the temperature sensor types. Thermocouples are popular due to its simplicity, ease of use and their speed of response to changes in temperature, due mainly to their small size. Thermocouples also have the widest temperature range of all the temperature sensors from below -200ºC to well over 2000ºC.

Thermocouples are thermoelectric sensors that basically consists of two junctions of dissimilar metals, such as copper and constantan that are welded or crimped together. One junction is kept at a constant temperature called the reference (Cold) junction, while the other the measuring (Hot) junction. When the two junctions are at different temperatures, a voltage is developed across the junction which is used to measure the temperature sensor as shown below.

2. A standard solar panel (also known as a solar module) consists of a layer of silicon cells, a metal frame, a glass casing, and various wiring to allow current to flow from the silicon cells. Silicon (atomic #14 on the periodic table) is a nonmetal with conductive properties that allow it to absorb and convert sunlight into electricity. When light interacts with a silicon cell, it causes electrons to be set into motion, which initiates a flow of electric current. This is known as the "photovoltaic effect," and it describes the general functionality of solar panel technology.

We just use the radiation as the power, divided by area of solar panel with the formula used in [10].

3.2. With human presence

Here we have two basic problems

1. Human identification

2. Human behaviour prediction

3.2.1. Human identification

Here we have several basic approaches to identify indoor human presence. They are the following:

1. Wi-Fi signals based

2. Physiological feature based

3. Behavioural feature

4. Camera-based solutions

1. Tong et al. [11] in their work proposed a method of human identification based on how a body changes the Wi-Fi signal. Due to the difference of body shapes and motion patterns, each person can have specific influence patterns on surrounding Wi-Fi signals while an individual moves indoors, generating a unique pattern on the CSV time series of the Wi-Fi device. Experimental results indicate that the identification accuracy is about 88.9% to 94.5% when the candidate user set changes from 6 to 2, showing that the proposed human identification method is effective in domestic environments.

2. Fingerprint [12], iris [13], and vein authentications [14] have been successfully employed in automatic human identification systems. Furthermore, Duta [15] used the hand-shape to distinguish human. Chellappa et al. [16] proposed a method to recognize human based on face recognition. However, these systems require the subject to be close to the sensor for accurate identification.

3. There are also some studies about behavioral biometrics, especially in gait analysis. Since human walking motion results in a self-sustaining and dynamic rhythm owing to the integrated signals generated from the spinal cord and sensory feedback, it contains unique characteristics of each individual. Little and Boyd [17] developed a model-free description of instantaneous motion and used it to recognize individuals by their gait. The work is based on camera sensing, which requires line of sight and enough lighting, and it also causes privacy issues. Nickel et al. [18] used HMM to recognize human gait based on accelerometer data, and it requires users to wear relevant sensors. Wi-Fi signal is used to recognize human, which can provide better coverage and are device-free

4. A novel method for automatic person identification is proposed by Dingbo D. et al. This method innovatively leverages correlation of body motion features from two different sensing sources, that is, accelerometer and camera. Experiment results demonstrate the performance and accuracy of the proposed method. However, the proposed method is limited in the following aspects:

1. First, users have to register and carry their smart phones in order to be discernible in camera.
2. Second, that phones had to relatively still with human body during the experiments, but in practice, people tend to take out and check their phones from time to time. Acceleration data collected during these occasions would damage the identification accuracy. Besides, the method relies heavily on background subtraction in the process of patch tracking. Thus, a more practical and reliable strategy for motion data collection is needed.
3. Subjects in archived video clips without available contextual motion information cannot be identified using the proposed method. Therefore, this method only works at the time of video capture [19].

3.2.2. Human behaviour problem

Here the data that can be used for the behaviour prediction is the decision that was made to achieve the maximal comfort. As the example, the work [20] can be used, where the historical data was used to predict user behaviour. The algorithm is based on real-time data of smartphone user behaviour, through Prophet algorithm for feature decomposition and time series prediction, and to find the inherent cycle and other characteristics, so as to perform user behavior recognition. This data-driven auxiliary authentication method can effectively solve the problem of easy forgery of static feature recognition such as password, fingerprint and face recognition.

As it was mentioned earlier, there is a problem of human identification in closed rooms and four types of solutions proposed.

The results of the given analysis for sensors are given below

1. Wi-Fi signals based.

This approach uses the fact that Wi-Fi signal is changed when it comes through the human body

a) Advantages

- High accuracy

b) Disadvantages

- Limitations in complex environments

2. Physiological feature based

This approach uses the human fingerprints or vein. This helps with human identification, but requires close contact.

a) Advantages

- High accuracy

b) Disadvantages

- Human action is required
- Close contact is needed
- Sometimes we need a person with device

3. Behavioural feature;

a) Advantages

- High accuracy

b) Disadvantages

- The question of defining the human motion can be quite hard

4. Camera-based solutions;

a) Advantages

- Set of well-known libraries to identify

- Relatively easy to add new people

b) Disadvantages

- Limitations to for the completely new people (for example, in rooms where there is a lot of people)
- High dependency of the position of the camera

Chapter 4

Implementation

To achieve the goals of the given work, the following stack of technologies was picked

I. Hardware

1. ESP32 board

2. Temperature sensors. Here DHT22 sensor is used

3. Solar panel as sensor to measure the solar radiation

4. 0v7670 camera to detect human presence

1. First of all, the microcontroller unit needs to be chosen. Here SparkFun ESP32 Thing is used. The SparkFun ESP32 Thing is a comprehensive development platform for Espressif's ESP32, their super-charged version of the popular ESP8266. Like the 8266, the ESP32 is a WiFi-compatible microcontroller but adds nearly 30 I/O pins. The ESP32's power and versatility will make it the foundation of IoT and connected projects for many years to come.

Espressif's ESP32 is one of the most popular microcontrollers on the market. Its laundry list of features include:

1) Dual-core Tensilica LX6 microprocessor

2) Up to 240MHz clock frequency

3) 520kB internal SRAM

4) Integrated 802.11 BGN WiFi transceiver

5) 2.2 to 3.6V operating range

6) 2.5 μA sleep current under hibernation

7) 32 GPIO

8) 10-electrode capacitive touch support

9) Hardware accelerated encryption (AES, SHA2, ECC, RSA-4096)

The ESP32 Thing is designed to surround the ESP32 with everything necessary to run and program the microcontroller, plus a few extra goodies to take advantage of the chip's unique features.

The Fig.1 here represents the structure of Sparkfun ESP32 Thing. The ESP32 features your standard fare of hardware peripherals, including:

1) 18 analog-to-digital converter (ADC) channels

2) 3 SPI interfaces

3) 3 UART interfaces

4) Two I^2C interfaces

5) 16 PWM outputs

6) 2 digital-to-analog converters (DAC)

7) Two I2S interfaces

8) And, thanks to the chip's pin multiplexing feature, those peripherals can be connected to just about any of the 28 broken out I/O pins. That means *you* decide which pins are RX, TX, MISO, MOSI, SCLK, SDA, SCL, etc.

There are, however, a few hardware features -- namely the ADC and DAC -- which are assigned static pins. The graphical reference below helps demonstrate where you can find those peripherals.

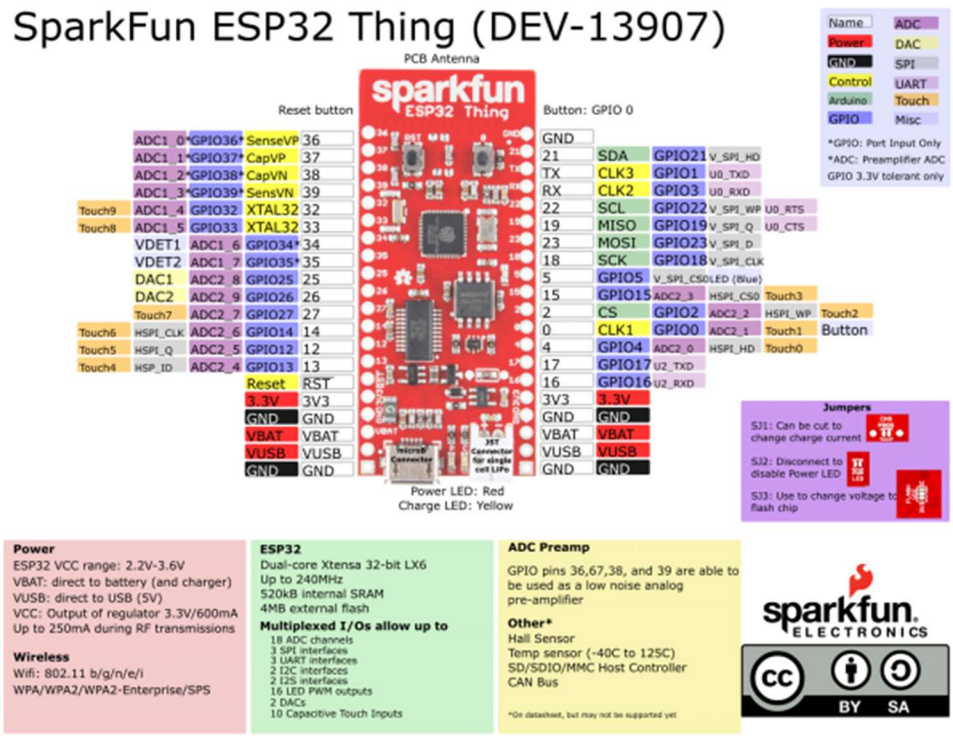

Fig. 1. Structure of SparkFun ESP32 Thing

2. DHT22 temperature sensor

The DHT22 is a basic, low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and spits out a digital signal on the data pin (no analog input pins needed). It's fairly simple to use but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds, so when using our library, sensor readings can be up to 2 seconds old. The Fig. 2 shows how the DHT22 sensor looks like.

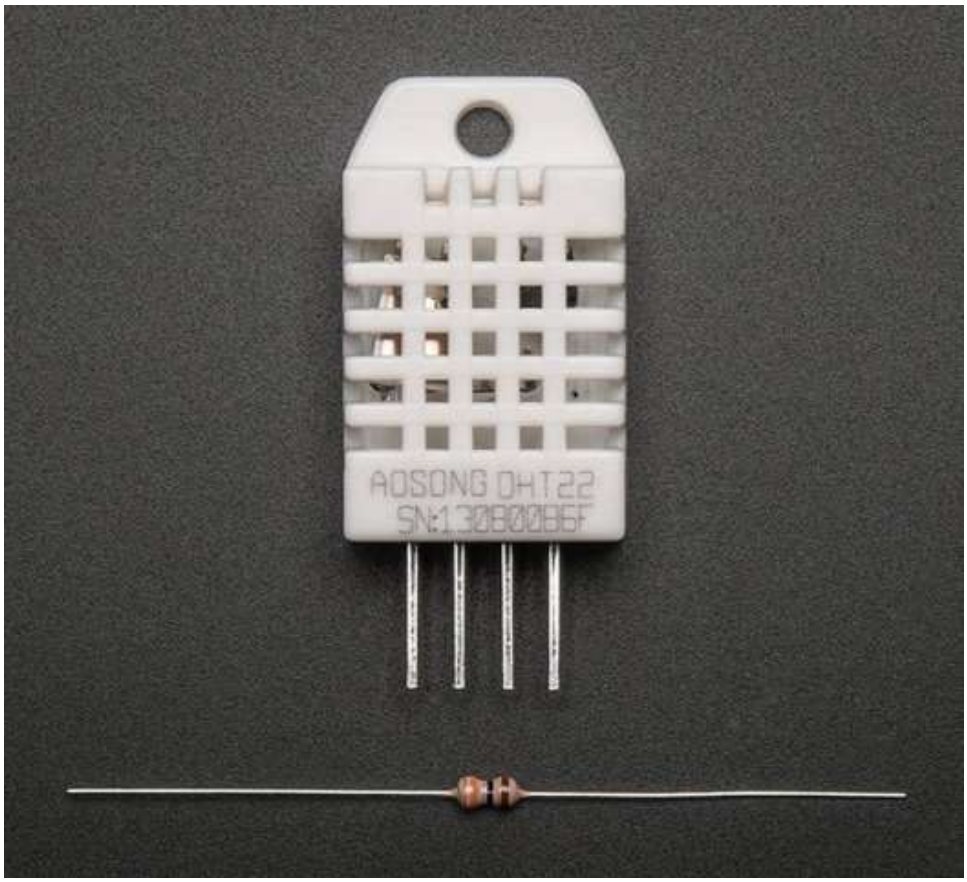

Fig. 2. DHT22 sensor

Technical characteristics

1) 3 to 5V power and I/O

2) 2.5mA max current use during conversion (while requesting data)

3) Good for 0-100% humidity readings with 2-5% accuracy

4) Good for -40 to 80°C temperature readings ±0.5°C accuracy

5) No more than 0.5 Hz sampling rate (once every 2 seconds)

6)Body size 27mm x 59mm x 13.5mm (1.05" x 2.32" x 0.53")

7) 4 pins, 0.1" spacing

8) Weight (just the DHT22): 2.4g

3. Here the solar panel ZW85X115 is used, The Fig. 3 shows how the ZW85X115 solar panel looks like

Fig. 3. ZW85X115 solar panel

Technical characteristics

1) 6V 2W Solar Panel

2) High conversion rate, high efficiency output

3) Voltage: 6V

4) Power: 2W

5) Current: 200 mA

6) Material: Monocrystalline

7) Size: 85 x 115mm

4. OV7670 image sensor, small volume, low operating voltage, providing all functions of a single chip of VGA camera and image processor. Through SCCB bus control, the sensor can output the whole frame, sampling, and various resolution 8 bits of data. The product VGA image can reach up to a maximum of 30 frames per second. Users can completely control the image quality, data format and transmission mode. All the process of image processing functions can be through the SCCB programming interface, including gamma curve, white balance and saturation.

OmmiVision image sensor has been in application of unique sensor technology, by reducing or eliminating the optical or electronic defect such as fixed pattern noise, tail, floating away, etc., to improve the quality of the image, and get the clear and stable color images.

Some technical characteristics
1) Photosensitive array: 640X480

2) IO Voltage: 2.5V to 3.0V (internal LDO for nuclear power 1.8V)

3) Power operation: 60mW/15fpsVGAYUV

4) Sleep: <20μA

5) Temperature Operating: -30 ℃ to 70 ℃

6) Stable: 0 ℃ to 50 ℃

7) Output Formats (8): YUV/YCbCr4: 2:2 RGB565/555/444 GRB4: 2:2 Raw RGB Data

8) Optical size: 1/6 "

9) FOV: 25 °

10) Maximum Zhen rate: 30fps VGA

11) Sensitivity: 1.3V / (Lux-sec)

12) SNR: 46 dB

13) Dynamic range: 52 dB

14) View Mode: Progressive

15) Pixel Size: 3.6μm x 3.6μm

16) Dark current: 12 mV / s at 60 ℃

II. Software

1. ESP32 Arduino stack

2. Custom Arduino libraries to work with different sensors and ESP32 libraries.

3. To send data on server, WiFi submodules were used

4. ESP32 software to get data from camera

5. Algorithm to detect human presence

1. As for the first part of software, Arduino core for ESP32 is located at

https://github.com/espressif/arduino-esp32

Like other microcontrollers, the ESP32 installed on the Arduino Uno should be programmed to perform the specified task. The most common programming languages for Arduino are the visually-block Scratch language and the C/C++.

The Arduino IDE is not available from the box for ESP32, so the board and libraries for it need to be installed manually. The guide to install the ESP32 board and all necessary libraries for Arduino is located at https://learn.sparkfun.com/tutorials/esp32-thing-hookup-guide

Arduino development environment consists of text editor for program code, message area (output of errors), text output window, toolbars and menu bar (Figure 4). To run programs and communication, the development environment connects to the Arduino debug card. The Arduino IDE allows to select a port for board connecting, compiles and loads the sketch (program) into the microcontroller installed on the board. In free access there is many educational materials, examples of sketches with step-by-step comments. There are also many 9 different libraries that allow to work with different sensors, modules and expansion cards. This greatly simplifies the process of creating a sketch and reduces its size, accordingly increases the speed. The drivers for the boards were created by one company and work not only on different versions of Windows operating system, but also on Linux and Mac OS. Connection with desktop computer or laptop for debugging is via the USB cable that comes with board without additional cables or buses.

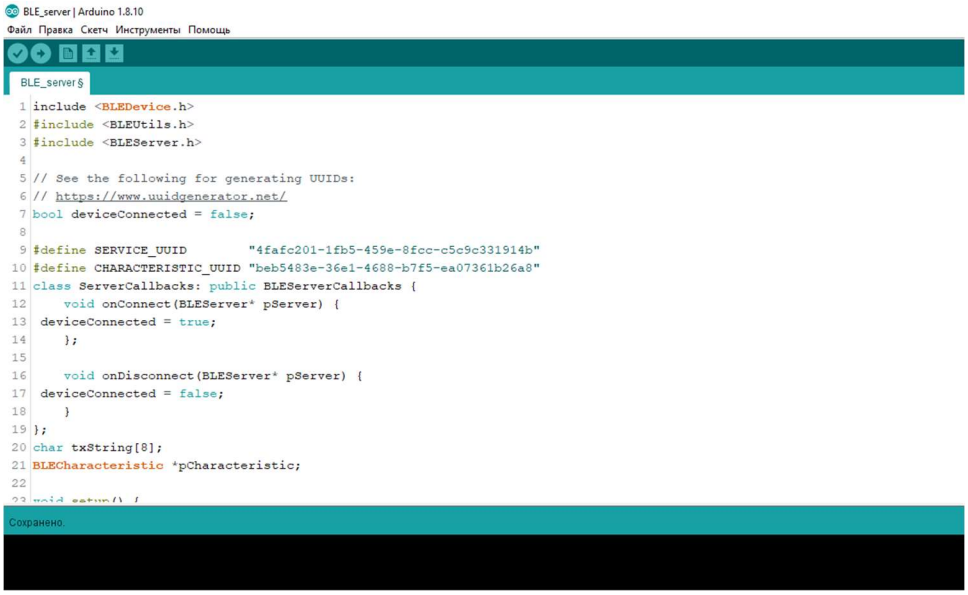

Fig. 4. Arduino IDE Development Environment workspace

As for the alternatives for the Arduino IDE MicroPython can be considered

MicroPython aims to implement the Python 3.4 standard (with selected features from later versions) with respect to language syntax, and most of the features of MicroPython are identical to those described by the "Language Reference" documentation at docs.python.org The MicroPython has some important differences from default implementation for Python – CPython.

The install guide for Micropython at ESP32 is available at https://docs.micropython.org/en/latest/esp32/tutorial/intro.html

As for the IDE, the standard IDEs can be used to edit the code written here.

The main advantage of Arduino is better documentation and larger community that is able to help with different problems. Also, the advantage of Arduino is that it was relatively harder to implement

The main advantage of MicroPython is the language that is relatively simpler than C/C++.

The code for Arduino ESP32 is the following. Here is a code for Wi-Fi connection.

```
#include <WiFi.h>
```

```
const char* ssid     = "SSID";

const char* password = "Password";


const char* host = "docs.micropython.org";

const char* streamId   = ".....................";

const char* privateKey = ".....................";

int period = 5000;


void setup()

{

   Serial.begin(115200);

   delay(10);


   // We start by connecting to a WiFi network


   Serial.println();

   Serial.println();

   Serial.print("Connecting to ");

   Serial.println(ssid);


   WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

int value = 0;

void loop()
{
  time_now = millis();
  while(millis() < time_now + period){
  ++value;

  Serial.print("connecting to ");
  Serial.println(host);

  // Use WiFiClient class to create TCP connections
```

```
WiFiClient client;
const int httpPort = 80;
if (!client.connect(host, httpPort)) {
   Serial.println("connection failed");
   return;
}

// We now create a URI for the request
String url = "/en/latest/esp32/quickref.html";

Serial.print("Requesting URL: ");
Serial.println(url);

// This will send the request to the server
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
            "Host: " + host + "\r\n" +
            "Connection: close\r\n\r\n");
unsigned long timeout = millis();
while (client.available() == 0) {
   if (millis() - timeout > 5000) {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
```

```
    }
  }

  // Read all the lines of the reply from server and print them to Serial
  while(client.available()) {
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }

  Serial.println();
  Serial.println("closing connection");
}
}
```

The code for the MicroPython is the following. As you can see, it has a lot of differences with the code in Arduino language. But it also uses the standard libraries of MicroPython that cannot be used for default CPython

```
import network
import socket
import ure
import time

ap_ssid = "SSID"
ap_password = "Password"
ap_authmode = 3  # WPA2

NETWORK_PROFILES = 'wifi.dat'
```

```
wlan_ap = network.WLAN(network.AP_IF)
wlan_sta = network.WLAN(network.STA_IF)

server_socket = None


def get_connection():
    """return a working WLAN(STA_IF) instance or None"""

    # First check if there already is any connection:
    if wlan_sta.isconnected():
        return wlan_sta

    connected = False
    try:
        # ESP connecting to WiFi takes time, wait a bit and try again:
        time.sleep(3)
        if wlan_sta.isconnected():
            return wlan_sta

        # Read known network profiles from file
        profiles = read_profiles()

        # Search WiFis in range
        wlan_sta.active(True)
        networks = wlan_sta.scan()

        AUTHMODE = {0: "open", 1: "WEP", 2: "WPA-PSK", 3: "WPA2-PSK", 4:
"WPA/WPA2-PSK"}
        for ssid, bssid, channel, rssi, authmode, hidden in sorted(networks,
key=lambda x: x[3], reverse=True):
            ssid = ssid.decode('utf-8')
            encrypted = authmode > 0
```

```
            print("ssid: %s chan: %d rssi: %d authmode: %s" % (ssid, channel,
rssi, AUTHMODE.get(authmode, '?')))
            if encrypted:
                if ssid in profiles:
                    password = profiles[ssid]
                    connected = do_connect(ssid, password)
                else:
                    print("skipping unknown encrypted network")
            else:  # open
                connected = do_connect(ssid, None)
            if connected:
                break

    except OSError as e:
        print("exception", str(e))

    # start web server for connection manager:
    if not connected:
        connected = start()

    return wlan_sta if connected else None


def read_profiles():
    with open(NETWORK_PROFILES) as f:
        lines = f.readlines()
    profiles = {}
    for line in lines:
        ssid, password = line.strip("\n").split(";")
        profiles[ssid] = password
    return profiles
```

```
def write_profiles(profiles):
    lines = []
    for ssid, password in profiles.items():
        lines.append("%s;%s\n" % (ssid, password))
    with open(NETWORK_PROFILES, "w") as f:
        f.write(''.join(lines))


def do_connect(ssid, password):
    wlan_sta.active(True)
    if wlan_sta.isconnected():
        return None
    print('Trying to connect to %s...' % ssid)
    wlan_sta.connect(ssid, password)
    for retry in range(100):
        connected = wlan_sta.isconnected()
        if connected:
            break
        time.sleep(0.1)
        print('.', end='')
    if connected:
        print('\nConnected. Network config: ', wlan_sta.ifconfig())
    else:
        print('\nFailed. Not Connected to: ' + ssid)
    return connected


def send_header(client, status_code=200, content_length=None ):
    client.sendall("HTTP/1.0 {} OK\r\n".format(status_code))
    client.sendall("Content-Type: text/html\r\n")
    if content_length is not None:
      client.sendall("Content-Length: {}\r\n".format(content_length))
    client.sendall("\r\n")
```

```
def send_response(client, payload, status_code=200):
    content_length = len(payload)
    send_header(client, status_code, content_length)
    if content_length > 0:
        client.sendall(payload)
    client.close()


def handle_root(client):
    wlan_sta.active(True)
    ssids = sorted(ssid.decode('utf-8') for ssid, *_ in wlan_sta.scan())
    send_header(client)
    client.sendall("""\
        <html>
            <h1 style="color: #5e9ca0; text-align: center;">
                <span style="color: #ff0000;">
                    Wi-Fi Client Setup
                </span>
            </h1>
            <form action="configure" method="post">
                <table style="margin-left: auto; margin-right: auto;">
                    <tbody>
    """)
    while len(ssids):
        ssid = ssids.pop(0)
        client.sendall("""\
                        <tr>
                            <td colspan="2">
                                <input type="radio" name="ssid" value="{0}"
/>{0}
                            </td>
```

```
                        </tr>
        """.format(ssid))
    client.sendall("""\
                        <tr>
                            <td>Password:</td>
                            <td><input name="password" type="password"
/></td>
                        </tr>
                    </tbody>
                </table>
                <p style="text-align: center;">
                    <input type="submit" value="Submit" />
                </p>
            </form>
            <p> </p>
            <hr />
            <h5>
                <span style="color: #ff0000;">
                    Your ssid and password information will be saved into the
                    "%(filename)s" file in your ESP module for future usage.
                    Be careful about security!
                </span>
            </h5>
            <hr />
            <h2 style="color: #2e6c80;">
                Some useful infos:
            </h2>
            <ul>
                <li>
                    Original code from <a
href="https://github.com/cpopp/MicroPythonSamples"
                        target="_blank"
rel="noopener">cpopp/MicroPythonSamples</a>.
```

```
                </li>
                <li>
                    This code available at <a
href="https://github.com/tayfunulu/WiFiManager"
                        target="_blank"
rel="noopener">tayfunulu/WiFiManager</a>.
                </li>
            </ul>
        </html>
    """ % dict(filename=NETWORK_PROFILES))
    client.close()


def handle_configure(client, request):
    match = ure.search("ssid=([^&]*)&password=(.*)", request)

    if match is None:
        send_response(client, "Parameters not found", status_code=400)
        return False
    # version 1.9 compatibility
    try:
        ssid = match.group(1).decode("utf-8").replace("%3F",
"?").replace("%21", "!")
        password = match.group(2).decode("utf-8").replace("%3F",
"?").replace("%21", "!")
    except Exception:
        ssid = match.group(1).replace("%3F", "?").replace("%21", "!")
        password = match.group(2).replace("%3F", "?").replace("%21", "!")

    if len(ssid) == 0:
        send_response(client, "SSID must be provided", status_code=400)
        return False
```

```
    if do_connect(ssid, password):
        response = """\
            <html>
                <center>
                    <br><br>
                    <h1 style="color: #5e9ca0; text-align: center;">
                        <span style="color: #ff0000;">
                            ESP successfully connected to WiFi network
%(ssid)s.
                        </span>
                    </h1>
                    <br><br>
                </center>
            </html>
        """ % dict(ssid=ssid)
        send_response(client, response)
        try:
            profiles = read_profiles()
        except OSError:
            profiles = {}
        profiles[ssid] = password
        write_profiles(profiles)

        time.sleep(5)

        return True
    else:
        response = """\
            <html>
                <center>
                    <h1 style="color: #5e9ca0; text-align: center;">
                        <span style="color: #ff0000;">
                            ESP could not connect to WiFi network %(ssid)s.
```

```
                        </span>
                    </h1>
                    <br><br>
                    <form>
                        <input type="button" value="Go back!"
onclick="history.back()"></input>
                    </form>
                </center>
            </html>
        """ % dict(ssid=ssid)
        send_response(client, response)
        return False


def handle_not_found(client, url):
    send_response(client, "Path not found: {}".format(url), status_code=404)


def stop():
    global server_socket

    if server_socket:
        server_socket.close()
        server_socket = None


def start(port=80):
    global server_socket

    addr = socket.getaddrinfo('0.0.0.0', port)[0][-1]

    stop()
```

```
    wlan_sta.active(True)
    wlan_ap.active(True)

    wlan_ap.config(essid=ap_ssid, password=ap_password, authmode=ap_authmode)

    server_socket = socket.socket()
    server_socket.bind(addr)
    server_socket.listen(1)

    print('Connect to WiFi ssid ' + ap_ssid + ', default password: ' +
ap_password)
    print('and access the ESP via your favorite web browser at 192.168.4.1.')
    print('Listening on:', addr)

    while True:
        if wlan_sta.isconnected():
            return True

        client, addr = server_socket.accept()
        print('client connected from', addr)
        try:
            client.settimeout(5.0)

            request = b""
            try:
                while "\r\n\r\n" not in request:
                    request += client.recv(512)
            except OSError:
                pass

            print("Request is: {}".format(request))
            if "HTTP" not in request:  # skip invalid requests
                continue
```

```
            # version 1.9 compatibility
            try:
                url = ure.search("(?:GET|POST) /(.*?)(?:\\?.*?)? HTTP",
request).group(1).decode("utf-8").rstrip("/")
            except Exception:
                url = ure.search("(?:GET|POST) /(.*?)(?:\\?.*?)? HTTP",
request).group(1).rstrip("/")
            print("URL is {}".format(url))

            if url == "":
                handle_root(client)
            elif url == "configure":
                handle_configure(client, request)
            else:
                handle_not_found(client, url)

        finally:
            client.close()
```

2. Quite good basic tutorial for the temperature sensors was used. It is located at https://create.arduino.cc/projecthub/mafzal/temperature-monitoring-with-dht22-arduino-15b013
For the solar radiation sensors, solar panel is used. The basic guide is located at

https://create.arduino.cc/projecthub/jeffrey2/measuring-solar-radiation-with-arduino-f741ac

3. WiFi module is located at

https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi

III. Device Connection

The scheme of device connection is shown at Fig. 1. Here BLE (Bluetooth Low Energy) is used

to create connection between two devices. The BLE server was used as the device to get all the necessary information from the outside (such as solar radiation and outdoor temperature) and BLE client as the device to receive all the information. BLE client can be considered here as hub, where we serve all the information, that can be later processed by other algorithms.

BLE

Bluetooth Low Energy (BLE) [23] is a low power wireless technology used for connecting devices with each other. BLE operates in the 2.4 GHz ISM (Industrial, Scientific, and Medical) band, and is targeted towards applications that need to consume less power and may need to run on batteries for longer periods of time - months, and even years.

Bluetooth started as a short-distance cable replacement technology. For example, to replace wires in devices such as a mouse, keyboard, or a PC communicating with a Personal Digital Assistant (PDA) which were popular in the late 1990s and early 2000s. The first official version of Bluetooth was released by Ericsson in 1994, named after King Harald “Bluetooth” Gormsson of Denmark who helped unify warring factions in the 10th century CE.

Bluetooth Low Energy (BLE), however, was introduced in the 4.0 version of the Bluetooth specification in 2010. The original Bluetooth defined in the previous versions is referred to as Bluetooth Classic. BLE was not an upgrade to the original Bluetooth: Bluetooth Classic, but rather it’s a new technology that utilizes the Bluetooth brand but focuses on the Internet of Things (IoT) applications where small amounts of data are transferred at lower speeds. It’s important to note that there’s a big difference between Bluetooth Classic and Bluetooth Low Energy in terms of technical specification, implementation and the types of applications they’re each suitable for.

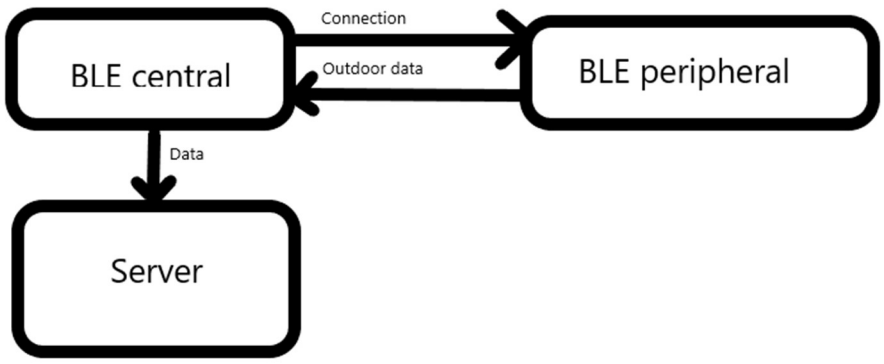

Fig. 5. Schematic for the device connections

Here custom Arduino-ESP32 libraries are used for their simplicity and flexibility for ESP32 board. The one more reason for using this stack of technologies is its low cost and open-source code that with well-developed community can help with overcoming lots of difficulties. There are the following things to perform the BLE device connection First, there is the concept of Central/Peripheral, which has to do with establishing a link. This is also known as the GAP role. A Peripheral can advertise, to let other devices know that it's there, but it is only a Central that can actually send a connection request to estalish a connection. When a link has been established, the Central is sometimes called a Master, while the Peripheral could be called a Slave.

In addition to the above roles, the Core Specification also defines the roles of an Observer and a Broadcaster. These are basically just non-connecting variants of the Central and Peripheral, in other words devices that just listens for advertisement packages (and possibly send scan responses) or just sends such packages, without ever entering a connection.

Then, there are the roles of a GATT Server and a GATT Client. Normally, the Server is the device that contains data, that the Client can read.

However, there is no connection between these roles. Even though it is most common for a Peripheral to be a Server and a Central to be a Client, it is perfectly possible to have a Peripheral that is only a Client, or a Central that is both a Server and a Client. The S110 can be a Peripheral and not a Central, but can be both a GATT Client and a GATT Server. That is why here we have

one device to get the data from the sensors and other to receive the data from the first device. In the example server is used to get data from the sensors and client receives data from the device. The BLE server code is the following. Here we have an example for BLE connection

```
#include <BLEDevice.h>

#include <BLEUtils.h>

#include <BLEServer.h>

#include <BLE2902.h>

#include "DHT.h"

#define DHTTYPE DHT22

#define DHTPIN 4

DHT dht(DHTPIN, DHTTYPE);


// See the following for generating UUIDs:

// https://www.uuidgenerator.net/

bool deviceConnected = false;


#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"

#define SERVICE_TUUID       "fd76c5e4-9dc5-4158-8c09-1463df2f9ff4"

#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"

#define CHARACTERISTIC_TUUID "81a19ab8-7c99-46c3-b3ea-c3d4181b957f"



#define DHTPIN 4
```

```
char txString[8];
class ServerCallbacks: public BLEServerCallbacks {
   void onConnect(BLEServer* pServer) {
deviceConnected = true;
   };

   void onDisconnect(BLEServer* pServer) {
deviceConnected = false;
   }
};
class MyCallbacks: public BLECharacteristicCallbacks {

   void onWrite(BLECharacteristic *pCharacteristic) {
     Serial.println(2);
   float t1 = dht.readTemperature();
   dtostrf(t1, 2, 2, txString);
   pCharacteristic->setValue(txString);
   Serial.println(txString);
   pCharacteristic->notify();

   }
};
```

```
BLECharacteristic *pCharacteristic;
BLECharacteristic *tCharacteristic;

void setup() {
  Serial.begin(115200);
  Serial.println("Starting BLE work!");

  BLEDevice::init("MyESP32");
  BLEServer *pServer = BLEDevice::createServer();
  BLEService *pService = pServer->createService(SERVICE_UUID);
  pServer->setCallbacks(new ServerCallbacks());
  pCharacteristic = pService->createCharacteristic(
                                CHARACTERISTIC_UUID,
                                BLECharacteristic::PROPERTY_READ   |
                                BLECharacteristic::PROPERTY_WRITE  |
                                BLECharacteristic::PROPERTY_NOTIFY
                               );
  pCharacteristic->addDescriptor(new BLE2902());
  tCharacteristic = pService->createCharacteristic(
                                CHARACTERISTIC_TUUID,
                                BLECharacteristic::PROPERTY_NOTIFY|
                                BLECharacteristic::PROPERTY_READ|
                                BLECharacteristic::PROPERTY_WRITE
```

```
                                );
  pCharacteristic->setCallbacks(new MyCallbacks());
  pCharacteristic->setValue("Hello World says Neil");
  pServer->setCallbacks(new ServerCallbacks());
  pService->start();
  Serial.println("Characteristic defined! Now you can read it in your phone!");

  // Start advertising
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(false);
  pAdvertising->setMinPreferred(0x0);  // set value to 0x00 to not advertise this parameter
  BLEDevice::startAdvertising();
  dht.begin();
}

void loop() {
  // put your main code here, to run repeatedly:
  if (deviceConnected) {
    int t;
    float v;
    float area;
    float power;
```

```
  float radiation;
  int LENGTH = 115;
  int WIDTH = 85;
  int RESISTANCE = 20;
  for (int i=1;i<=5;i++) {int a=analogRead(39);t=t+a;delay(200);}



  float k = t; t=0;
  Serial.println(t);
  k = k/5;
  area = (float)LENGTH*WIDTH/(100*100);
  power=pow(k,2)/RESISTANCE; // 0.33 is the ISC of my pannel in Amps. Use your ISC.
  radiation = power/area;
  dtostrf(radiation, 2, 2, txString);
  pCharacteristic->setValue(txString);
  pCharacteristic->notify();
  }
  delay(5000);
}
```

And this is the BLE client code

```
/**
* A BLE client example that is rich in capabilities.
*/
```

```
#include "BLEDevice.h"
//#include "BLEScan.h"

// The remote service we wish to connect to.
static BLEUUID serviceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
// The characteristic of the remote service we are interested in.
static BLEUUID    charUUID("beb5483e-36e1-4688-b7f5-ea07361b26a8");

static BLEAddress *pServerAddress;
static boolean doConnect = false;
static boolean connected = false;
static BLERemoteCharacteristic* pRemoteCharacteristic;

static void notifyCallback(
  BLERemoteCharacteristic* pBLERemoteCharacteristic,
  uint8_t* pData,
  size_t length,
  bool isNotify) {
    Serial.print("Notify callback for characteristic ");
    Serial.print(pBLERemoteCharacteristic->getUUID().toString().c_str());
    Serial.print(" of data length ");
    Serial.println(length);
}
```

```
bool connectToServer(BLEAddress pAddress) {

    Serial.print("Forming a connection to ");
    Serial.println(pAddress.toString().c_str());

    BLEClient*  pClient  = BLEDevice::createClient();
    Serial.println(" - Created client");

    // Connect to the remove BLE Server.
    pClient->connect(pAddress);
    Serial.println(" - Connected to server");

    // Obtain a reference to the service we are after in the remote BLE server.
    BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
    if (pRemoteService == nullptr) {
      Serial.print("Failed to find our service UUID: ");
      Serial.println(serviceUUID.toString().c_str());
      return false;
    }
    Serial.println(" - Found our service");
```

```
  // Obtain a reference to the characteristic in the service of the remote BLE server.
  pRemoteCharacteristic = pRemoteService->getCharacteristic(charUUID);
  if (pRemoteCharacteristic == nullptr) {
    Serial.print("Failed to find our characteristic UUID: ");
    Serial.println(charUUID.toString().c_str());
    return false;
  }
  Serial.println(" - Found our characteristic");


  // Read the value of the characteristic.
  std::string value = pRemoteCharacteristic->readValue();
  Serial.print("The characteristic value was: ");
  Serial.println(value.c_str());


  pRemoteCharacteristic->registerForNotify(notifyCallback);
}
/**
* Scan for BLE servers and find the first one that advertises the service we are looking for.
*/
class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
/**
  * Called for each advertising BLE server.
  */
```

```
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    Serial.print("BLE Advertised Device found: ");
    Serial.println(advertisedDevice.toString().c_str());

    // We have found a device, let us now see if it contains the service we are looking for.
    if (advertisedDevice.haveServiceUUID() && advertisedDevice.getServiceUUID().equals(serviceUUID)) {

      //
      Serial.print("Found our device!  address: ");
      advertisedDevice.getScan()->stop();

      pServerAddress = new BLEAddress(advertisedDevice.getAddress());
      doConnect = true;

    } // Found our server
  } // onResult
}; // MyAdvertisedDeviceCallbacks


void setup() {
  Serial.begin(115200);
  Serial.println("Starting Arduino BLE Client application...");
  BLEDevice::init("");
```

```
  // Retrieve a Scanner and set the callback we want to use to be informed when we
  // have detected a new device.  Specify that we want active scanning and start the
  // scan to run for 30 seconds.
  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true);
  pBLEScan->start(30);
} // End of setup.


// This is the Arduino main loop function.
void loop() {


  // If the flag "doConnect" is true then we have scanned for and found the desired
  // BLE Server with which we wish to connect.  Now we connect to it.  Once we are
  // connected we set the connected flag to be true.
  if (doConnect == true) {
    if (connectToServer(*pServerAddress)) {
      Serial.println("We are now connected to the BLE Server.");
      connected = true;
    } else {
      Serial.println("We have failed to connect to the server; there is nothin more we will do.");
```

```
  }

  doConnect = false;

 }


 // If we are connected to a peer BLE Server, update the characteristic each time we are reached

 // with the current time since boot.

 if (connected) {

  String newValue = "Time since boot: " + String(millis()/1000);

  Serial.println("Setting new characteristic value to \"" + newValue + "\"");


  // Set the characteristic's value to be the array of bytes that is actually a string.

  pRemoteCharacteristic->writeValue(newValue.c_str(), newValue.length());

 }


 delay(1000); // Delay a second between loops.

}
```

Chapter 5

Discussion

Based on previous work, we have the following methodology for indoor blinds control 1. Check for human indoor presence,

2, If there is no human inside, the check for energy balance is made. After this, a decision for closing or opening the blinds is made.

3. If there is a human inside, here check for history of previous decisions is made. If the history is empty, the default decision for human is made.

The designed framework allows to add more sensors and other equipment's, empowering the solution and allow it's use on many other areas and problems. Disadvantage of system is necessity to have BLE on both of devices. As the both of the devices can also be connected to the Wi-Fi network, it is possible to evolve for a solution where the control and monitoring can be done through the internet in a future. Also, this system can be adopted for mobile applications based on Android or iOS platforms.

Chapter 6

Conclusions

Based on the goals defined in Chapter 1, the following conclusions can be made

For indoor blinds control, the following decisions were made

1. Check for human indoor presence,

2, If there is no human inside, the check for energy balance is made. After this, a decision for closing or opening the blinds is made.

3. If there is a human inside, here check for history of previous decisions is made. If the history is empty, the default decision for human is made.

In the process of performing the work, the theoretical research was made for indoor blind control and different algorithms were considered.

References

[1] A. Tzempelikos, A., Athienitis, A., 2007. The impact of shading design and control on building cooling and lighting demand. Solar Energy 81, 369–382.

[2] B. Seong, Y.-B., Yeo, M.-S., Kim, K.-W., 2013. Optimized control algorithm for automated venetian blind system considering solar profile variation in buildings. Indoor and Built Environment, 1–25 (online) at https://www.scopus.com/record/display.uri?eid=2-s2.0-84887355963&origin=inward

[3] C. Hu, J., Olbina, S., 2011. Illuminance-based slat angle selection model for automated control of split blinds. Building and Environment 46, 786– 796 at https://www.sciencedirect.com/science/article/pii/S0360132310003057

[4] E. Chan Y., Tzempelikos, A., 2013. Efficient venetian blind control strategies considering daylight utilization and glare protection. Solar Energy 98, 241-254

[5] Chebotarova Y., Perekrest A., Ogar V. Comparative Analysis of Efficiency Energy Saving Solutions Implemented in the Buildings

[6] Tzempelikos A., Ying-Chieh Chan. Efficient venetian blind control strategies considering daylight utilization and glare protection

[7] Wua Y., Kämpf J., Scartezzini J.-L. Automated 'Eye-sight' Venetian blinds based on an embedded photometric device with real-time daylighting computing: user behaviour and dual-channel communication prediction

[8] https://www.electronics-tutorials.ws/io/io_3.html

[9] https://news.energysage.com/solar-panels-work/

[10] .https://create.arduino.cc/projecthub/jeffrey2/measuring-solar-radiation-with-arduino-f741ac

[11] Chen Y., Li H., Chen X. Venetian Blind Control System Based on Fuzzy Neural Network for Indoor Daylighting

[12] Tong X., Bin G., Zhu W., Mingyang L., Zhiwen Y. FreeSense: Indoor Human Identification with WiFi Signals

[13] K. Karu, and A.K. Jain. 1996. Fingerprint classification. Pattern Recognition 29 (3), pp. 389-404.

[14] H.A. Park, and K.R. Park. Iris recognition based on score level fusion by using SVM. Pattern Recognition Letters 28 (15), 2007, pp. 2019-2028

[15] D. Mulyono, and H.S. Jinn. A study of finger vein biometric for personal identification. International Symposium on Biometrics and Security Technologies, 2008, pp. 1-8.

[16] N. Duta. A survey of biometric technology based on hand shape. Pattern Recognition 42 (11), 2009, pp. 2797-2806

[17] Yan Wang, Jian Liu, Yingying Chen, Marco Gruteser,Jie Yang, and Hongbo Liu. E-eyes: device-free location-oriented activity identification using fine-grained WIFI signatures. In Proceedings of the 20th annual international conference on Mobile computing and networking, 2014, pp, 617–628.

[18] Rajalakshmi Nandakumar, Bryce Kellogg, and Shyamnath Gollakota. WIFI gesture recognition on existing devices. Eprint Arxiv, 2014.

[19] G Wang, Y Zou, Z Zhou, and K Wu. We Can Hear You with WIFI! In Proceedings of the 20st Annual International Conference on Mobile Computing and Networking(MobiCom '14), 2014, pp. 593-604

[20] Dingbo Duan, Guangyu Gao, Chi Harold Liu, and Jian Ma. Automatic Person Identification in Camera Video by Motion Correlation

[21] Mi Chunmin , Xu Runjie , Ching-Torng Lin. Real-time Recognition of Smartphone User Behavior Based on Prophet Algorithms

[22] Shamaila Hayat, Aimal Rextin, Adnan Idris, Mehwish Nasim. Text and phone calls

[23] https://www.novelbits.io/basics-bluetooth-low-energy/